



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Gemelo Digital Oceanográfico: Discretización del entorno marino mediante un modelo multinivel de mallas H3.

Grado en Ingeniería Mecánica

ALUMNO: Pablo García Valera

DIRECTORES: Milagros Fernández Gavilanes
Carlos Pérez Collazo

CURSO ACADÉMICO: 2022-2023

Universida_{de}Vigo



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

*Gemelo Digital Oceanográfico: Discretización del entorno
marino mediante un modelo multinivel de mallas H3*

Grado en Ingeniería Mecánica
Intensificación en Tecnología Naval
Cuerpo General

UniversidadeVigo

RESUMEN

El TFG aborda temas de importancia y de actualidad en la Armada como es el concepto de Gemelo Digital, centrándose especialmente en el ámbito marítimo, haciendo uso de las nuevas tecnologías para mejorar el conocimiento de los fenómenos meteorológicos, en tiempo real y con previsión de futuro. Llevando a cabo un estudio previo de las posibles fuentes de datos con la posterior creación de una base de dato capaz de almacenar la información necesaria, y a la que se puede acceder desde cualquier lugar realizando una conexión al servidor. Para conseguir solucionar el problema de la representación gráfica de los datos, se ha desarrollado una página web que permite el intercambio de información entre un usuario y el servidor que almacena la información. Empleando mallados iniciales para llevar a cabo la descarga de información relevante en sus nodos y que serán empleados para construir el mallado multinivel final (H3). Analizando cuales son las resoluciones que mejor se adaptan a las unidades de la Armada, se asegurará una precisión de calidad. Los resultados finales serán analizados remarcando aquellos aspectos de importancia para prevenir errores de funcionamiento. El TFG finaliza proponiendo el modelo final diseñado, consiguiendo una calidad aceptable para la simulación.

PALABRAS CLAVE

Multinivel, Fenómeno meteorológico, Simulación, Mallado, Resolución.

AGRADECIMIENTOS

En primer lugar, me gustaría agradecerle todo el apoyo recibido a mi padre, pilar fundamental y que siempre ha estado dispuesto a ayudar a pesar de la distancia.

En segundo lugar, a mi madre y mis hermanas, por todos los momentos en los que los ánimos no estaban en lo más alto y eran ellas las que han sabido cómo volver a levantarlos y seguir adelante.

En tercer lugar, a todos aquellos familiares y amigos, que durante este tiempo fuera de casa han hecho que volver al hogar fuese el momento más deseado del año y diese la sensación de no haberme ido nunca.

Agradecer a mis tutores, Milagros y Carlos, por su gran paciencia, trabajo duro y la ayuda brindada durante todo el proceso de desarrollo del trabajo. Sin su apoyo el resultado final habría sido muy diferente.

También a Erika, que con una sonrisa y un buenos días nos recibe en el comedor llenando un poco más cada plato y hace del momento más importante del día algo agradable y feliz.

Per ultimo, a i miei nuovi amici Frances Casertano e Valerio Cascapera, i quali sono stati un supporto rilevante nella elaborazione di questo lavoro e hanno brindato sempre il suo aiuto con un sorriso, e sono stati attenti di non lasciarmi cadere, anche se la lingua poteva rappresentare una minuscola difficoltà.

¡A todos muchas gracias! A voi Grazie mille!

CONTENIDO

Contenido	1
Índice de Figuras	4
Índice de Tablas.....	8
1 Introducción y objetivos	9
1.1 Contexto y motivación	9
1.2 Objetivos	10
1.3 Estructura de la memoria	10
2 Estado del arte	12
2.1 Simulación en las Fuerzas Armadas	12
2.2 Gemelo digital.....	14
2.2.1 Antecedentes.....	14
2.2.2 Tipos de gemelo digital	14
2.2.3 Gemelo digital en la Armada	17
2.2.4 Gemelo digital oceanográfico.....	18
2.3 Discretización del entorno	20
2.3.1 Aproximaciones para realizar el mallado	20
2.3.2 Mallado MeteoGalicia	22
2.3.3 Mallado H3 Uber	22
2.4 Fuentes de datos	25
2.4.1 Fuentes nacionales	25
2.4.1 Fuentes abiertas	26
2.5 Bases de datos	29
2.5.1 Formato JSON y bases de datos	29
2.5.2 Tipos de bases de datos NoSQL	31
3 Materiales y métodos.....	34
3.1 Arquitectura	34
3.2 Software empleado.....	36
3.2.1 Python	36
3.2.2 MongoDB	37
3.2.3 DJANGO	38
3.2.4 Bootstrap.....	39
3.3 Metodología	39
3.3.1 Base de datos	39

3.3.2 Descarga de datos batimétricos	41
3.3.3 Descarga de datos MeteoGalicia	42
3.4 Implementación.....	42
3.4.1 Captura de datos de fuentes externas y almacenamiento en MongoDB.....	42
3.4.2 Extracción de datos y representación en mapa según el mallado H3	46
3.4.3 Interfaz: la página web.....	53
3.4.3.1 Gestión de usuarios.....	55
3.4.3.2 Consulta de la información del Gemelo Digital Oceanográfico.....	59
4 Prueba y validación de resultados	66
4.1 Descripción funcionalidades de la página web	66
4.1.1 Gestión de usuarios: registro y acceso.....	66
4.1.2 Acceso a datos y su representación	69
4.2 Simulaciones	73
4.3 Mensajes de error	77
4.3.1 Mensajes de error durante la administración de la cuenta de usuario	77
4.3.2 Mensajes de error durante la realización de simulaciones.....	80
5 Conclusiones y líneas futuras	86
5.1 Conclusiones	86
5.2 Líneas futuras	87
6 Bibliografía.....	90
Anexo I: Implicaciones Económicas y Ambientales.....	94
A.1.1 Implicaciones económicas	94
A 1.2 Implicaciones medioambientales	97
Anexo II: Reflexiones Éticas y Sociales	99
Anexo III: Instalaciones	100
A.3.1 Instalaciones	100
A.3.1.1 VirtualBox	100
A.3.1.2 Visual Studio Code y Python.....	101
A.3.1.2 MongoDB	102
A.3.1.2 DJANGO & BOOTSTRAP	103
Anexo IV: Códigos.....	104
A.4.1 Descarga datos	104
A.4.1.1 Almacenamiento batimetría en MongoDB	104
A.4.1.2 Definición clases.....	104
A.4.1.3 Descarga y almacenamiento de datos en MongoDB	105

A.4.1.4 Ejecución código para obtención de datos.....	110
A.4.2 Diseño web “Gemelo Digital Oceanográfico”	110
A.4.2.1 MYSITE.....	110
A.4.2.1.1 Settings.	110
A.4.2.2 URLS.	113
A.4.2.2 Templates.....	114
A.4.2.2.1 Base.	114
A.4.2.2 Snippets.	115
A.4.2.2.4.1 Header.....	115
A.4.2.2.4.2 Footer.....	116
A.4.2.3 Account.	117
A.4.2.2.1 Views.....	117
A.4.2.2.2 Models.	119
A.4.2.2.3 Forms.	120
A.4.2.2.4 Templates.	121
A.4.2.2.4.1 Account.....	121
A.4.2.2.4.2 Login.....	122
A.4.2.2.4.3 Register.....	123
A.4.2.4 Personal.....	124
A.4.2.4.1 Views.....	124
A.4.2.4.2 Templates.	153
A.4.2.4.2.1 Home.	153
A.4.2.4.2.2 Homes.....	155
A.4.2.5 Manage.....	156

ÍNDICE DE FIGURAS

Figura 1-1 Modelo predictivo viento del Estrecho de Gibraltar. Fuente: [48].	9
Figura 2-1 Representación gemelo digital de una turbina. Fuente: [47].	15
Figura 2-2 Gemelo Digital Fragatas F-110 Fuente: Revista General de Marina.	17
Figura 2-3 Sistema Integrado de control de plataforma (SICP). Fuente: [52].	18
Figura 2-4 Arquitectura de un gemelo digital oceanográfico. Fuente: [13].	19
Figura 2-5 Ejemplo de mallado no estructurado. Fuente: [16].	21
Figura 2-6 Ejemplo de conformidad y no conformidad. Fuente: [16].	22
Figura 2-7 Comparación entre los mallados de MeteoGalicia. Fuente: [18].	22
Figura 2-8 División de una superficie en hexágonos. Fuente: [18].	23
Figura 2-9 División de una superficie en hexágonos. Fuente: [18].	24
Figura 2-10 Buque Hidrográfico Hespérides atravesando mala mar. Fuente: [22].	26
Figura 2-11 Modelo Numérico MeteoGalicia. Fuente: [51].	28
Figura 2-12 Ejemplo datos desordenados Puerto de Marín, en formato JSON. Fuente: [6].	30
Figura 2-13 Ejemplo datos ordenados formato JSON. Fuente: [6].	30
Figura 2-14 Modelo "Clave-Valor" Fuente: [29].	31
Figura 2-15 Modelo base de datos columnar. Fuente: [33].	33
Figura 3-1 Arquitectura Global GDO. Fuente: Propia.	35
Figura 3-2 Ejemplo aplicación Python. Fuente: Propia.	36
Figura 3-3 Logo Django. Fuente: [41].	38
Figura 3-4 Logo Bootstrap. Fuente: [43].	39
Figura 3-5 Estructura Data Base en MongoDB. Fuente: Propia.	40
Figura 3-6 Ajustes acceso IP. Fuente: Propia.	40
Figura 3-7 Acceso a la descarga de batimetría. Fuente: [41].	41
Figura 3-8 Código para almacenar la batimetría en la base de datos. Fuente: Propia.	41
Figura 3-9 Ejemplo Mallado Rías Baixas GDO. Fuente: Propia.	43
Figura 3-10 Ejemplo definición diferentes clases derivadas. Fuente: Propia.	44
Figura 3-11 Llamada a la función "Temperatura". Fuente: Propia.	44
Figura 3-12 Código para obtención de datos. Fuente: Propia.	45
Figura 3-13 Diagrama de flujo para el filtrado de puntos por elevación. Fuente: Propia.	45
Figura 3-14 Texto indicando final de la descarga. Fuente: Propia.	46
Figura 3-15 Base de datos con los documentos almacenados. Fuente: Propia.	46
Figura 3-16 Inicio GDO. Fuente: Propia.	47
Figura 3-17 Búsqueda de datos en MongoDB. Fuente: Propia.	48
Figura 3-18 Diagrama de flujo filtrar datos mensaje. Fuente: Propia.	48

Figura 3-19 Iteración parámetros oleaje. Fuente: Propia.	49
Figura 3-20 Extracción datos batimetría. Fuente: Propia.	49
Figura 3-21 Creación hexágonos. Fuente: Propia.	49
Figura 3-22 Asignación puntos a la malla. Fuente: Propia.	50
Figura 3-23 Cálculo de la media. Fuente: Propia.	51
Figura 3-24 Asignación colores. Fuente: Propia.	51
Figura 3-25 Asignación polígonos. Fuente: Propia.	52
Figura 3-26 Finalización código y guardado. Fuente: Propia.	52
Figura 3-27 Carpetas creadas automáticamente en el proyecto. Fuente: Propia.	53
Figura 3-28 Código html base de la página web. Fuente: Propia.	54
Figura 3-29 Aplicación "account": Fuente: Propia.	55
Figura 3-30 Account.html. Fuente: Propia.	56
Figura 3-31 Modelo.py. Fuente: Propia.	56
Figura 3-32 Controlador. Fuente: Propia.	57
Figura 3-33 Página principal del administrador. Fuente: Propia.	57
Figura 3-34 Administrar usuarios. Fuente: Propia.	58
Figura 3-35 Información del usuario. Fuente: Propia.	58
Figura 3-36 Aplicación "Personal". Fuente: Propia.	59
Figura 3-37 Página principal de la web. Fuente: Propia.	60
Figura 3-38 Pagina web, recopilación datos. Fuente: Propia.	60
Figura 3-39 Código inicial del formulario. Fuente: Propia.	61
Figura 3-40 Formulario unidad. Fuente: Propia.	61
Figura 3-41 Formulario latitud y longitud. Fuente: Propia.	61
Figura 3-42 Filtros numéricos. Fuente: Propia.	62
Figura 3-43 Formulario radio de acción. Fuente: Propia.	62
Figura 3-44 Html Botones. Fuente: Propia.	63
Figura 3-45 Diseño botones. Fuente: Propia.	63
Figura 3-46 Elif. Fuente: Propia.	64
Figura 3-47 Asignación de variables. Fuente: Propia.	64
Figura 3-48 Añadir marcas. Fuente: Propia.	65
Figura 3-49 Establecimiento "iframe". Fuente: Propia.	65
Figura 4-1 Interface de la página principal. Fuente: Propia.	66
Figura 4-2 Formulario de registro. Fuente: Propia.	67
Figura 4-3 Inicio de sesión. Fuente: Propia.	67
Figura 4-4 Página principal. Fuente: Propia.	68

Figura 4-5 Cambios al usuario. Fuente: Propia.	68
Figura 4-6 Página principal. Fuente: Propia.	69
Figura 4-7 Indicar datos necesarios. Fuente: Propia.	69
Figura 4-8 Resultado batimetría. Fuente: Propia.	70
Figura 4-9 Comparación entre diferentes resoluciones (9/10). Fuente: Propia.	71
Figura 4-10 Mallado de resolución 7. Fuente: Propia.	72
Figura 4-11 Oleaje y viento. Fuente: Propia.	72
Figura 4-12 Batimetría. Fuente: Propia.	73
Figura 4-13 Humedad (2023-02-27T08:00:00+01). Fuente: Propia.	73
Figura 4-14 Nubosidad (2023-02-27T08:00:00+01). Fuente: Propia.	74
Figura 4-15 Oleaje (2023-03-01T17:00:00+01). Fuente: Propia.	74
Figura 4-16 Precipitaciones (2023-03-01T23:00:00+01). Fuente: Propia.	75
Figura 4-17 Presión (2023-03-02T02:00:00+01). Fuente: Propia.	75
Figura 4-18 Salinidad (2023-02-27T12:00:00+01). Fuente: Propia.	76
Figura 4-19 Temperatura (2023-02-08T03:00:00+01). Fuente: Propia.	76
Figura 4-20 Viento (2023-02-26T23:00:00+01). Fuente: Propia.	77
Figura 4-21 Email ya existente en la base de datos. Fuente: Propia.	77
Figura 4-22 Nombre de usuario ya existente. Fuente: Propia.	78
Figura 4-23 Error en la configuración de la contraseña. Fuente: Propia.	78
Figura 4-24 Error coincidencia de contraseñas. Fuente: Propia.	78
Figura 4-25 Formulario en blanco. Fuente: Propia.	79
Figura 4-26 Nombre de usuario ya utilizado. Fuente: Propia.	79
Figura 4-27 Email ya en uso. Fuente: Propia.	79
Figura 4-28 Insertar datos. Fuente: Propia.	80
Figura 4-29 Aviso error en la latitud. Fuente: Propia.	81
Figura 4-30 Aviso error en la longitud. Fuente: Propia.	81
Figura 4-31 Error en la consulta a los datos. Fuente: Propia.	81
Figura 4-32 Error radio acción. Fuente: Propia.	82
Figura 4-33 Error en el establecimiento del radio de acción. Fuente: Propia.	82
Figura 4-34 Error de formato en la resolución del mallado. Fuente: Propia.	83
Figura 4-35 Diferentes almacenamientos. Fuente: Propia.	83
Figura 4-36 Error mallado. Fuente: Propia.	84
Figura 4-37 Oleaje. Fuente: Propia.	85
Figura 5-1 Ejemplo batimetría S-100. Fuente: [45].	88

Figura A1 1 Distribución horas. Fuente: Propia.....	94
Figura A1 2 Horas totales. Fuente: Propia.	95
Figura A1 3 Ramas accion contra el cambio climático. Fuente: [49].	97
Figura A1 4 Buque vertiendo líquidos contaminantes al mar. Fuente: [50].	98
Figura A3 1 Interfaz VirtualBox Fuente: Propia.....	100
Figura A3 2 Logo Visual Studio Code. Fuente: Propia.	101
Figura A3 3 Entorno Visual Studio Code. Fuente: Propia.	102
Figura A3 4 Instalación Django. Fuente: Propia.....	103
Figura A3 5 Aplicación Bootstrap. Fuente: [46].....	103

ÍNDICE DE TABLAS

Tabla 2-1 Objetivos ETID 2020. Fuente: [2].	13
Tabla 2-2 Resolución celdas mallado H3. Fuente: [50].	24
Tabla 2-3 Tiempo de procesamiento de diferentes bases de datos. Fuente: [33].	33
Tabla 5-1 Proyectos del S-10X. Fuente: [44].	88
Anexo I	
Tabla A1 1 Costes elaboración GDO. Fuente: Propia.	95
Tabla A1 2 Costes mantenimiento servidores: Fuente: [48].	96

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Contexto y motivación

La cuarta revolución industrial ha comenzado, hasta tal punto que la tecnología es cada vez más relevante en cualquier ámbito de la sociedad. Es por esto por lo que, la sociedad busca aprovechar estas tecnologías a su favor, explotándolas de tal manera que sean capaces de facilitar todo tipo de trabajos. Hoy en día la Armada se encuentra inmersa en un proceso de modernización (Transformación Digital Armada 4.0), mediante el cual se busca aprovechar la infinidad de ventajas que la tecnología moderna es capaz de aportar a tareas rutinarias, así como su mejoría.

Uno de los objetivos planteados en las “Líneas Generales de la Armada” [1] es el del desarrollo de los conocidos como “Gemelos Digitales”. Estos modelos consisten en simulaciones en tiempo real que permitan al usuario conocer el estado de diferentes elementos presentes en un buque. De esta forma es posible conocer los puntos débiles del sistema para prevenir averías evitando esfuerzos elevados o sobrecalentamientos en zonas específicas. Esta herramienta está orientada a conseguir una mejoría en la prevención de riesgos laborales (PRL), como, por ejemplo, detectar cuando una tubería esta agrietada y vertiendo fluidos tóxicos en las zonas de habitabilidad de un buque. Así como conseguir una mejor gestión de los recursos durante el transcurso de una operación. También busca conseguir una vida operativa mayor de los diferentes componentes de una unidad.

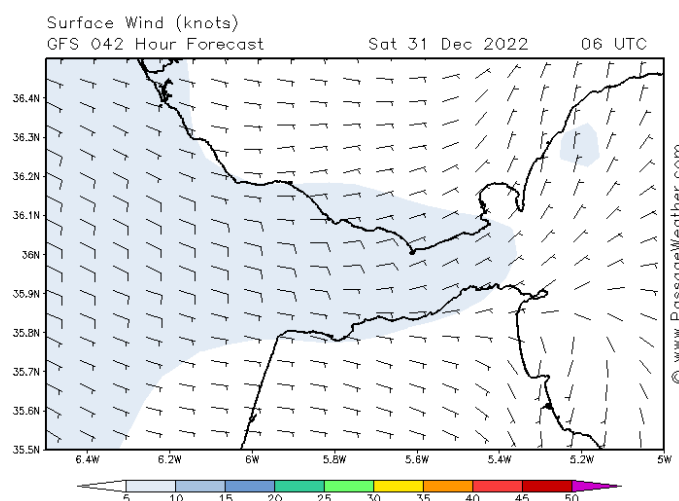


Figura 1-1 Modelo predictivo viento del Estrecho de Gibraltar. Fuente: [48].

El ámbito de la seguridad en las operaciones es uno de los más importantes en la Armada actual, y más concretamente el concepto de seguridad en la navegación. Existen múltiples factores que hay que tener en cuenta a la hora de planear una derrota, e infinidad de herramientas que permiten conocer diferentes características del entorno en el que se va a navegar. Sin embargo, en lo que a la meteorología respecta, la única forma de trabajo existente es mediante predicciones, y análisis de históricos.

En este caso, una de las herramientas más sonadas y populares de la actualidad para resolver este problema, es el conocido “Gemelo Digital Oceanográfico” (GDO), esta herramienta de simulación en tiempo real permitirá conocer la información procedente de diversas fuentes de datos para determinar las condiciones meteorológicas, y la batimetría entre otras, existentes en una zona determinada. Esto permitirá a cada tipo de buque estudiar, en función de sus capacidades de navegación, la derrota a establecer, una posible zona de ejercicios o incluso conocer el entorno donde se llevará a cabo una operación futura.

1.2 Objetivos

Este trabajo pretende brindar una serie de herramientas que permitan interaccionar de forma ágil y dinámica con bases de datos que recojan información relevante en el aspecto de la meteorología y batimetría. Al mismo tiempo que se permite una discretización del espacio para una correcta implementación y análisis de datos.

- En primer lugar, se procede a la implementación de un modelo capaz de descargar información de diferentes parámetros físicos y almacenarlos en una base de datos, de tal manera que sea posible disponer de un servidor al que se acceda únicamente para hacer consultas meteorológicas sin la necesidad de establecer conexión con las fuentes de datos. De esta forma es posible tener un sistema centralizado de gestión de la información.
- En segundo lugar, realizar un análisis de las diferentes posibilidades de integración de los diferentes elementos, ya sean datos meteorológicos específicos y batimetría con diferentes tipos de mapa. Es decir, todo tipo de elementos de interés para la navegación aprovechando dicho sistema. De la misma forma se lleva a cabo el diseño de un modelo multinivel que se nutre del mallado H3 de Uber para representar la información, permitiendo elegir entre otros factores la resolución del mallado empleado.
- En tercer lugar, realizar una prueba de concepto final para las Rías Baixas en la que sea posible observar todas las ventajas que este modelo es capaz de ofrecer.

1.3 Estructura de la memoria

Tras haber procedido a la contextualización y haciendo referencia al marco en el que se encuentra el proyecto, así como una breve descripción de los objetivos que se persiguen, en este apartado se procede a la breve explicación de los diferentes apartados que componen el trabajo de forma secuencial. El presente documento se divide en siete apartados, y finaliza con una bibliografía y una serie de anexos necesarios para el entendimiento del proyecto, así como su realización.

- En primer lugar: “Introducción y Objetivos. En este apartado se presentan una serie de factores por los que el desarrollo de un GDO sería capaz de facilitar la toma de decisiones, así como mejorar la seguridad a la navegación. En este mismo apartado se ha procedido a realizar una breve descripción de los principales objetivos a conseguir.
- En segundo lugar: “Estado del arte”. Con el presente apartado se busca realizar un análisis y una revisión crítica de donde se encuentra la tecnología en el ámbito de GDO actualmente, así como en el ámbito de la simulación, bases de datos y obtención de estos a través de diferentes fuentes meteorológicas.

- En tercer lugar: “Desarrollo del TFG”, donde se implementará el software empleados y todos los procedimientos seguidos para la realización del trabajo. Para ello se proporcionará una explicación de los diferentes aspectos que se consideren de interés para el entendimiento de este, explicando con detalle algunos aspectos del código implementado, tanto en el manejo de los datos como en la representación de estos.
- En cuarto lugar: “Prueba y validación de resultados”. En este apartado se procederá a demostrar el funcionamiento de cada modelo presentado, así como a realizar diversas pruebas.
- En quinto lugar: “Conclusiones y líneas futuras”. En el presente apartado se procederá a analizar cuáles son las posibilidades que este proyecto ofrece para el futuro, así como los proyectos en desarrollo de características similares.
- En sexto lugar, se plasmará la bibliografía de las fuentes consultadas durante la realización del trabajo.
- Finalmente se expondrán una serie de anexos en los que se encuentra todo lo necesario para el entendimiento y réplica del trabajo en caso de necesidad, como podrían ser, por ejemplo, los códigos de programación.

2 ESTADO DEL ARTE

En el presente apartado se presenta una definición del gemelo digital, así como las clasificaciones de estos según los diferentes usos que pueda tener o el momento en el que este gemelo digital sea concebido. Así mismo se estudiarán los diferentes tipos de mallado existentes que nos permitirán discretizar el espacio marino y operar con los flujos de datos necesarios. Para ello finalmente se expondrán las diferentes bases de datos existentes que permiten trabajar con un flujo de datos de entidad, como es el caso necesario para la realización del gemelo digital y su implementación en el simulacro del entorno de las Rías Baixas.

2.1 Simulación en las Fuerzas Armadas

La evolución de la tecnología está demostrando ser un factor clave en lo referente al adiestramiento de todos los miembros de las Fuerzas Armadas, no solo en ámbito de actuación sino también en lo referente al planeamiento. Es por esto por lo que, el Ministerio de Defensa, elabora un documento en el que recoge todos los objetivos a alcanzar en ámbito de I+D+I, el conocido como “Estrategia de Tecnología e Innovación para la Defensa ETID 2020”. Este documento persigue dos objetivos fundamentales [2].

- En primer lugar, se busca alcanzar un desarrollo óptimo de las capacidades militares, aportando soluciones que se encuentren en la vanguardia tecnológica y que sean capaces de proporcionar una ventaja operacional en los diferentes campos de aplicación.
- En segundo lugar, servir de base tecnológica de tal manera que las Fuerzas Armadas sirvan cuentan con el principio de libertad de acción en el empleo de las capacidades militares.

En concreto, este documento hace referencia a una serie de avances tecnológicos que busca conseguir en los próximos años, haciendo uso de tecnologías que supongan un cambio radical a la hora de hacer las cosas. Entre estos objetivos se encuentran los diferentes desarrollos de tecnologías para aplicaciones militares, de esta forma se destaca la transformación digital, que lleven a las Fuerzas Armadas a obtener una ventaja táctica en el campo de batalla [2].

Tal y como se puede comprobar en la Tabla 2-1, uno de los objetivos dentro de estas estrategias es la seguridad a bordo en las diferentes plataformas. Gracias a la aplicación de las tecnologías 4.0 es posible llevar a cabo simulaciones, gracias a la recogida de una serie de datos, que permite analizar el funcionamiento de los diferentes sistemas. Actualmente, existe un programa encargado de recoger diferentes datos de sensores a bordo de las unidades y analizarlo mediante operadores para detectar errores.

6. PLATAFORMAS NAVALES
6.1. DISEÑO DE PLATAFORMAS NAVALES
6.1.1. Diseño de plataformas navales de superficie
6.1.2. Diseño de plataformas navales submarinas
6.1.3. Invisibilidad de las plataformas navales de superficie y submarinas
6.1.4. Seguridad a bordo de las plataformas
6.2. CONTROL, ENERGÍA Y PROPULSIÓN NAVAL
6.2.1. Sistemas de propulsión de plataformas navales
6.2.2. Maniobrabilidad de plataformas navales
6.2.3. Sistemas de generación eléctrica en plataformas navales

Tabla 2-1 Objetivos ETID 2020. Fuente: [2].

De la misma forma que la ETID busca el desarrollo para reducir el coste, no solo en material, sino también en recursos de instalaciones a la hora de adiestrar a sus miembros y el empleo de simuladores que es algo cada vez más presente en las diferentes unidades y en los diferentes ejércitos. La simulación es una herramienta que permite adiestrar al personal en situaciones que es difícil replicar fuera de un entorno virtual, de tal manera que todos los miembros de las Fuerzas Armadas adquieran una serie de habilidades específicas [3].

En cuanto a las tecnologías existentes en el ámbito de la defensa, es posible encontrar diferentes tecnologías como la realidad virtual. Este tipo de tecnología permite romper una barrera espacio-tiempo y de esta forma reproducir una realidad ficticia que permita ejercitar al operador en el ámbito del empleo de los sentidos. Para llevar a cabo este adiestramiento es importante diferenciar entre las tecnologías inmersivas y las que no lo son [3]. Las primeras permiten modelar a través de un operador externo aquello que sucede en el entorno virtual y por lo tanto modifican aquello que el operador percibe. Mientras que la segunda, permite interactuar con diferentes personas en una serie de espacios inexistentes sin más allá del uso del mismo ordenador.

En este mismo ámbito militar, los usos de la simulación abarcan todo tipo de campos, desde la gestión del personal hasta servir como herramienta en la toma de decisiones. Así mismo estos simuladores que comúnmente son utilizados en la enseñanza, instrucción y adiestramiento pueden ser extrapolados y tener implicaciones en el mundo real [3].

Actualmente la Armada Española cuenta con la necesidad de llevar a cabo una renovación en cuanto a simuladores en servicio empleados en la enseñanza, instrucción y adiestramiento debido a la dificultad de recrear entornos bélicos reales sin que esto conlleve un coste de recursos elevado, al mismo tiempo que es capaz de resolver el gran dilema sobre la complejidad de estos entornos. De esta forma es posible diferenciar una gran cantidad de simuladores que actualmente prestan servicio en los centros de enseñanza. Estos simuladores se clasifican según el entorno que son capaces de generar, entre los más destacados se encuentran [3]:

- Navegación y táctica: estos simuladores ayudan en todo lo relacionado con la navegación, control de la propia plataforma y el empleo táctico y técnico de los diferentes equipos que se encuentran presentes en el puente del buque. Así mismo existen simuladores dedicados al empleo de los diferentes sistemas de combate.
- Seguridad interior: estos simuladores permiten recrear escenarios de emergencia que permitan al personal adiestrarse en situaciones de estrés en los que el flujo de información tiene que ser claro y continuo.

- Apoyo a la decisión y experimentación: este grupo de simuladores destaca debido a que es principalmente utilizado para analizar los resultados de someter diferentes unidades o incluso a sus miembros a situaciones complejas, estudiando al mismo tiempo como es el comportamiento de ciertos objetos o sistemas. En este ámbito de trabajo existen diversas metodologías de simulación, entre las que se encuentran las conocidas como simulaciones en vivo, que son aquellas simulaciones que cuentan con implicación en primera persona del usuario y dan como resultado un comportamiento muy similar o incluso en algunos casos idéntico, del proceso o equipo que se está estudiando o simulando [3]. De esta idea principal nace el proyecto actual del conocido “Gemelo Digital”.

2.2 Gemelo digital

2.2.1 Antecedentes

En la década de 1960, la NASA desarrollo una tecnología con la que simulaban los procesos que tenían lugar en el espacio en la tierra mediante la implementación de ordenadores, lo cual se tornó de vital importancia en misiones de rescate a más de 300 000 km de distancia. Posteriormente en la década de los 2000, el Dr. Grieves [4] propuso un modelo de gestión del ciclo de vida de los productos en los que un modelo digital y un modelo virtual intercambiaban información de manera continua para su monitorización, algo que hasta la fecha se ha visto como inalcanzable. Gracias a la aparición de la industrial 4.0 este tipo de proyectos es cada vez más posible.

El concepto de “Gemelo Digital” o “Digital Twin” (DT) tiene sus cimientos en las ya conocidas como técnicas de diseño, ingeniería y fabricación asistidos por ordenador (CAD, CAE, CAM) [5]. Mediante la integración de forma simultánea de estas técnicas se busca lograr un producto que simule el funcionamiento de cualquier objeto o proceso físico [6]. El objetivo fundamental que busca este tipo de gemelo digital no es más que el de proporcionar una herramienta que permita al usuario final tener la capacidad de tomar decisiones en diversas circunstancias. Para ello, estos modelos deberán contar con una arquitectura adecuada que les permita incorporar todos aquellos datos relevantes que satisfagan las necesidades del usuario. En otras palabras, se podría considerar al gemelo digital como un bloque compuesto por diversos módulos con capacidad de ser intercambiables, formado por una base de datos ordenada, actualizada en tiempo real y coherente [7].

Gracias a tecnologías, como el internet de las cosas (IoT), “*Machine Learning*”, la “Inteligencia Artificial” (IA) y el análisis de datos masivos, las cuales permiten la correcta gestión de los datos en tiempo real, combinándolos con los registros históricos, los gemelos digitales son capaces de: proporcionar un producto único de representación de la información, analizar exhaustivamente la información, y al mismo tiempo proporcionar una retroalimentación instantánea, dotando al modelo de un funcionamiento autónomo y capaz de agilizar la detección de errores [8].

A modo resumen, un “Gemelo Digital” no es más que un conjunto de información y diferentes algoritmos que correctamente integrados son capaces de representar el comportamiento de una operación, objeto o sistema. Y mediante una monitorización continua del gemelo real, es posible llevar a cabo un análisis de la respuesta ante determinadas situaciones.

2.2.2 Tipos de gemelo digital

En la actualidad existen diferentes clasificaciones de gemelos digitales, los cuales se pueden dividir atendiendo a diversos factores como el momento de la creación del gemelo digital, su nivel de integración, aplicaciones, el nivel de jerarquía y su nivel de madurez.



Figura 2-1 Representación gemelo digital de una turbina. Fuente: [47].

En primer lugar, se definen los gemelos digitales según el momento en el que han sido concebidos, dando lugar a la siguiente clasificación [9]:

- Gemelos digitales prototipo: estos modelos han sido diseñados previamente a la fabricación física del producto y permiten la realización de diferentes pruebas gracias a que son capaces de procesar información relativa a materiales, CAD y esfuerzos. De esta manera es posible corregir errores y comportamientos no deseados previos a la fabricación del producto.
- Gemelo digital instantáneo: este tipo de modelo está directamente conectado a su gemelo físico y gracias a la obtención de información de este es posible prevenir funcionamientos no deseados.

En segundo lugar, se encuentran los modelos digitales diferenciados por su nivel de integración. De esta manera diferenciamos entre [9]:

- Modelo digital: en este tipo de modelo la información que usa el gemelo digital es introducida manualmente por el operario.
- Sombra digital: la información utilizada por el gemelo digital es introducida automáticamente, mientras que aquella que se extrae para análisis se deberá extraer de forma digital.
- Gemelo digital: en este tipo de modelo el flujo de información es automático en ambos sentidos.

En tercer lugar, diferenciando por el servicio que ofrecerán estos modelos diferenciamos entre [9]:

- Gemelos digitales de producto: en estos modelos se plantea una simulación de un producto en concreto. Estos modelos se utilizan principalmente en la fase de prototipado del producto, previamente a la producción de este, de manera que es posible llevar a cabo una breve simulación del comportamiento del producto bajo unas condiciones determinadas.
- Gemelos digitales de proceso: estos gemelos se encargan de simular todas y cada una de las fases que componen un proceso de producción, analizando todos aquellos factores que puedan afectar negativamente.
- Gemelos de actuación: estos gemelos permiten analizar el comportamiento de cualquier objeto o proceso físico facilitando así la toma de decisiones y optimizando su uso y proporcionando capacidad de mejora gracias a la implementación de bucles de retroalimentación.

En cuarto lugar, obedeciendo a los diferentes niveles de jerarquía es posible diferenciar entre [9]:

- Nivel unidad: en el que la simulación en tiempo real se centra únicamente en un componente del sistema global.
- Nivel sistema: este tipo de gemelo digital permite simular un conjunto de elementos y estudiar y analizar su comportamiento. Este tipo de modelo está directamente relacionado con el nivel unidad y existe un continuo flujo de información entre ambos.
- Sistema de sistemas: este modelo es capaz de trabajar con un conjunto de niveles sistema de tal manera que proporciona una visión global del comportamiento permitiendo la correcta optimización del sistema en su conjunto.

En último lugar se encuentra la clasificación según el nivel de madurez de los diferentes modelos [9]:

- Gemelo digital parcial: el cual cuenta con un número limitado de datos que es capaz de procesar.
- Clon del gemelo digital: el cual contiene todos los datos que se consideran de relevancia a la hora de representar y analizar con posterioridad los resultados.
- Gemelo digital aumentado: este tipo de modelo, mediante el uso de análisis y algoritmo es capaz de relacionar la información que recibe en tiempo real con los registros históricos. Este tipo de modelos está relacionado con los conocidos también como Gemelos digitales inteligentes, los cuales son capaces de reconocer patrones en el entorno en el que operan al mismo tiempo que analizan esa información, aprovechando su capacidad de aprendizaje para optimizar el análisis de los resultados.

Es importante destacar, que al igual que existen diferentes modelos de “Gemelos Digitales” también es posible clasificarlos en niveles, según el nivel de tecnología que los conforma. De esta manera es posible diferenciar hasta cinco niveles diferentes [4]:

- El nivel uno se corresponde con el nivel más básico, el denominado “Gemelo Descriptivo”, este modelo es únicamente una versión digital en la que el usuario es el encargado de proporcionar los datos que desee y de esta manera se obtienen unos resultados específicos.
- El segundo nivel es el conocido “Gemelo Informativo”, en el que el propio gemelo integra los datos provenientes de los diferentes sensores y los verifica. De esta manera este nivel añade una capa que como resultado final asegura la correcta interoperabilidad entre los sistemas.
- El nivel tres se corresponde con los gemelos predictivos, estos modelos son capaces de interpretar los datos que están recibiendo y por lo tanto son capaces de llegar a ciertas conclusiones, activando diferentes alarmas en caso de necesidad.
- En cuarto lugar, se encuentran los gemelos integrados, estos gemelos son capaces de llevar a cabo análisis de datos y cuáles serían los resultados hipotéticos de situaciones que se puedan considerar de crisis. En otras palabras, este modelo de gemelo digital permite llevar a cabo simulaciones de escenarios de catástrofes y evaluar cuales han sido las causas, de tal forma que sea posible su prevención, y en caso de que esto no sea posible, evaluar las consecuencias.
- Por último, se encuentra el quinto nivel, el gemelo autónomo, el cual es capaz de llevar a cabo un aprendizaje basándose en los datos históricos y llegado el momento tomar decisiones en nombre del usuario. [4]



Figura 2-2 Gemelo Digital Fragatas F-110 Fuente: Revista General de Marina.

Estos últimos tres niveles requieren de tecnologías asociadas a la industria 4.0, ya que, como lleva a cabo el nivel número cinco, es necesario un aprendizaje por parte del modelo para poder llevar a cabo una toma de decisiones básica en nombre del usuario, posible gracias a la inteligencia artificial [5].

2.2.3 Gemelo digital en la Armada

Actualmente en la Armada Española se está desarrollando el proyecto conocido como Astillero 4.0, un proyecto directamente ligado a la Industria 4.0. En este nuevo proyecto, el concepto de “Gemelo Digital” es algo que está directamente relacionado. Este modelo de simulación permite dotar a las unidades de una continua monitorización de los diferentes sistemas de a bordo, así como el soporte en la toma de decisiones [10].

En los diferentes buques de la Armada Española existe un elemento conocido como el “Sistema Integrado de Control de Plataforma” (SICP), el cual permite la continua monitorización de todos los equipos y sistemas que componen un buque, con la excepción de que este sistema no tiene en cuenta el sistema de combate y nada que esté relacionado con el mismo, este tipo de sistema actualmente podría asociarse con un gemelo digital de nivel dos. Este sistema cuenta con una serie de algoritmos y secuencias de control que conforman el software, los cuales pueden ser controlados por operadores o simplemente estar integrados desde un inicio. Todo este conjunto permite al sistema organizar los diferentes datos y representarlos en la pantalla del ordenador, para que posteriormente sean analizados por el usuario, ya que este sistema es capaz de representar información en tiempo real, mostrar líneas de tendencia o incluso generar informes [11].



Figura 2-3 Sistema Integrado de control de plataforma (SICP). Fuente: [11].

Sin embargo, las funciones que asemejan el SICP al “Gemelo Digital” son aquellas relacionadas de manera directa con el adiestramiento a bordo, mantenimiento preventivo y por síntomas o incluso la gestión de repuestos. De la misma forma, es posible incorporar diferentes módulos en el SICP [11].

Este gemelo digital hace uso de dos clases de datos principalmente, en un primer lugar, los metadatos, es decir, aquellos datos que no cambian con frecuencia y que por lo general describen la estructura inicial del modelo. Y, en segundo lugar, se encuentran los datos que describen el funcionamiento dinámico de la unidad y que por lo tanto son únicos para el equipo [10].

Se necesitan una serie de herramientas para permitir a la Armada la explotación del “Gemelo Digital” de sus unidades las Fragatas F-110 clase “Bonifaz”. Estas herramientas se clasifican en cuatro grandes grupos [10]:

- Primero, los gestores del ciclo de vida de los productos, estos son los encargados de asegurar un acceso seguro a la información, permitiendo configuraciones diferentes a lo largo de todo el ciclo de vida del producto.
- Segundo, programas de diseño correcto que permitan a los diseñadores el poder llevar a cabo todas las tareas relacionadas con el aspecto de forma digital final del producto, permitiendo no solo la visualización del diseño, sino también su edición. Es importante que estos métodos vengán combinados con otras aplicaciones de ingeniería y fabricación.
- Tercero, un software capaz de soportar ciertos modelos matemáticos y algoritmos que permitan presentar los datos de comportamiento al usuario.
- Por último, la plataforma digital, que es la que proporciona una capacidad o servicio final.

2.2.4 Gemelo digital oceanográfico

Una parte fundamental de estos gemelos digitales es la obtención y procesamiento de la información, para lo cual existen diferentes métodos para recabar datos como pueden ser satélites, plataformas fijas o incluso mediciones realizadas desde los propios buques. Sin embargo, el gran interés que despiertan los gemelos digitales ha hecho que la Unión Europea (UE), junto con sus Estados Miembros, haya comenzado a desarrollar una infraestructura para poder nutrir de información relevante los modelos de los diferentes gobiernos, ciudadanos e industrias, de tal manera que sean capaces de llevar a cabo una toma de decisiones correcta. Para todo esto la UE contribuye en el desarrollo de modelos digitales interactivos, aportando capacidad de observación oceanográfica [6].

Así mismo, la UE aporta infraestructuras ya existentes como son Copernicus Marine Environment Monitoring System¹ o el Servicio de Vigilancia del Medio Marino² e infraestructuras de investigación de la Unión Europea [12]. Gracias al acceso libre y de forma gratuita a esta información todos los usuarios del mar podrán colaborar en la elaboración de un flujo de información continuo aportando todo tipo de datos de interés. Así mismo se pretende crear un modelo realista que permita gestionar diferentes aspectos del mar y permitan su exploración y simulación de todo tipo de escenarios. Este tipo de gemelos digitales busca la integración de IA y del “*machine learning*” de tal manera que sea posible una mejor gestión de la información y facilitar así la toma de decisiones [13]. Esto generará un modelo de simulación que permitirá a expertos evaluar, los daños producidos por las catástrofes naturales, la acción del ser humano sobre el ecosistema submarino, los movimientos de la fauna que se deriven de estos. Por último, los expertos planean utilizar este tipo de simulaciones para enseñar a su público menos experto las consecuencias que ciertos aspectos del océano podrían tener en un futuro cercano [12].

Finalmente, los gemelos digitales oceanográficos basan su arquitectura en tres bloques diferenciados, tal y como se observa en la Figura 2-4 [13]. El primero de ellos es el encargado de facilitar acceso a la información, al mismo tiempo que permite manejo y modificación. El segundo bloque tiene por objetivo el de servir de base para la simulación, en este bloque se combina la información recibida, se procesa y ordena de tal manera que pueda ser utilizada por los diferentes bloques y pueda integrar diferentes funcionalidades. En último lugar, el bloque tres es el encargado de llevar a cabo la simulación, obteniendo los resultados y presentándoselos al usuario. Todos estos datos habrán sido recogidos de los bloques anteriores y podrán variar los resultados en función de los datos de entrada [13].



Figura 2-4 Arquitectura de un gemelo digital oceanográfico. Fuente: [13].

¹ Copernicus Marine Environment Monitoring System es un servicio que proporciona información sobre diferentes aspectos relacionados con el ambiente marino como pueden ser parámetro físicos o biológicos.

² Servicio derivado de Copernicus Marine Environment Monitoring System especializado en recabar información, especialmente dinámica, sobre los diferentes ecosistemas.

2.3 Discretización del entorno

El océano es un sistema complejo y difícil de analizar en su conjunto. Es por esto por lo que los investigadores han creado un método de análisis capaz de solucionar este problema [14]. Para ello se procede a dividir el espacio en elementos más sencillos y simples de analizar por separado, estudiando posteriormente el resultado como conjunto. Estos elementos sencillos se conocen con el nombre de “discretos”, y su cantidad dentro de un sistema puede variar dependiendo de la precisión que se desee en la solución del problema. Esto se debe a que, al hacer esta división, también conocida como discretización, es posible que ciertos datos del sistema global se pierdan o sean eliminados por otros. Es por esto por lo que a menor tamaño de la malla siempre habrá una mayor precisión en el resultado aproximado.

A este método de división del espacio en diferentes elementos, su evaluación por separado para, posteriormente, juntar los resultados y tener una solución final del sistema total se le conoce como “Método de los elementos finitos”. Este método persigue principalmente dos objetivos [14]:

- Dividir el sistema objeto de estudio en un número finito de elementos, en los cuales se establecerán a su vez un número finito de parámetros que se podrán analizar con posterioridad.
- La solución final sigue el mismo método de análisis y resultados que se ha seguido con el análisis de cada elemento discreto.

Para poder llevar a cabo esta discretización del entorno uno de los métodos más utilizados es el conocido como mallado [15]. El objetivo del mallado es buscar el esquema adaptativo el cual sea capaz de resolver el problema proporcionando la solución más exacta con el menor número de incógnitas posibles. Para ello es posible asignar una malla inicial y aumentar la discretización allí donde sea necesario y el error sea necesario, repitiendo este proceso de manera iterativa hasta llegar a la solución deseada con una exactitud aceptable. Para alcanzar esta solución existen principalmente tres tipos de mallado:

- Tipo h: el tamaño del mallado es modificado en aquellas áreas donde es necesario debido a que se requiere una mayor precisión.
- Tipo p: en este tipo de mallado el grado del polinomio utilizado en la interpolación de la solución varía en función de los elementos.
- Tipo hp: este tipo de mallado es un enfoque híbrido que utiliza los mallados h y p aprovechando de las ventajas aportadas por cada uno de ellos.

2.3.1 Aproximaciones para realizar el mallado

Existen diferentes maneras de llevar a cabo la realización de los diferentes mallados en función del resultado que se desee obtener. Estos se dividen en función del tipo de forma que se obtenga en su elaboración, derivado principalmente de como haya sido el proceso en su generación. De esta forma se destacan [16]:

- Los mallados estructurados: estos mallados siguen un patrón reticular, usando generalmente elementos cuadrangulares y hexaédricos para 2D y 3D respectivamente. La característica principal de este mallado es que la relación entre las diferentes celdas es conocida de antemano. Esto supone al mismo tiempo una serie de limitaciones a la hora de su implementación en los diferentes objetos. Es importante que los sistemas u objetos que se deseen analizar cumplan con la condición de que puedan ser transformados en figuras cuadrangulares o hexagonales según corresponda. Así mismo este tipo de mallado requiere de la presencia de un operador encargado de controlar todo el proceso, debido a la complejidad que este tipo de mallado presenta [17].

Existen diferentes métodos para la creación de estas mallas, entre los que destacan los métodos algebraicos, en donde se transforma el entorno original en un entorno canónico. El método de las Ecuaciones Derivadas Parciales (EDPs), el cual se utiliza en ocasiones en las que se requiere unas características específicas de la malla. Métodos de superposición de retículas y por ultimo los métodos de mallado estructurado.

- El mallado no estructurado, que, a diferencia del mallado anterior este no presenta ningún tipo de ordenación preestablecida. De esta manera es posible encontrar patrones que se adapten mejor que otros según la precisión final con la que se desea trabajar en el proyecto [17].

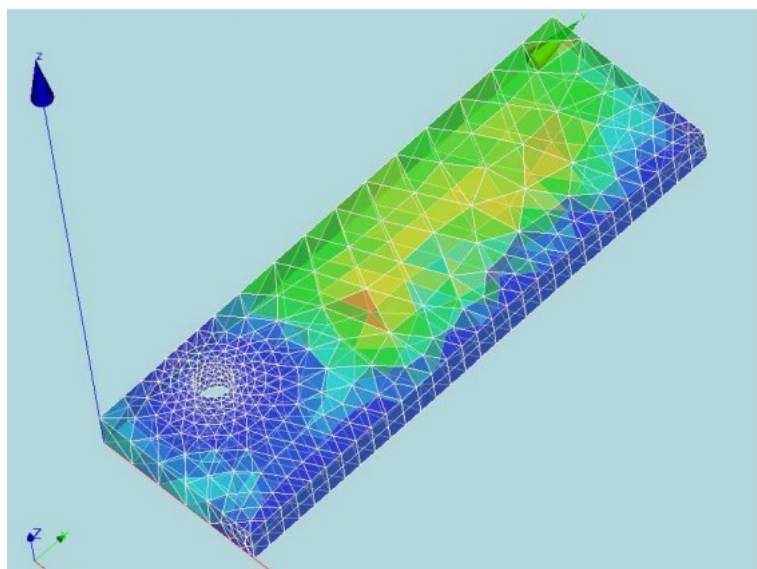


Figura 2-5 Ejemplo de mallado no estructurado. Fuente: [16].

Con el mallado no estructurado es necesario calcular la conectividad entre las diferentes celdas y establecer una relación, sin embargo, el proceso de mallado en estos casos suele estar mucho más automatizado, haciendo que sea posible su aplicación a objetos o sistemas de características muy irregulares [16, 17]. Los métodos que existen para el cálculo de estas mallas se dividen según el nivel de complejidad del mallado. En primer lugar, se encuentra el método de Delaunay-Voronoi, el cual hace diferencia entre mallados 2D y 3D. otro método conocido es el método “Por frente de avance”, en el que inicialmente la malla se empieza a conformar en el entorno del problema y continua su formación a medida que el proceso va teniendo lugar, desde el contorno inicial hacia el interior del dominio.

- Los mallados de bloques, este tipo de mallado es más avanzado y se utiliza en los casos en los que los mallados estructurados y no estructurados no son capaces de cumplir con las expectativas. El procedimiento que se debe seguir es crear una división del objeto o sistema por bloques más sencillos y proceder a su análisis individualmente, de tal forma que el resultado final sea la combinación de todos los bloques. Este tipo de mallado presenta la dificultad de realizar una correcta subdivisión en bloques y establecer las diferentes relaciones entre estos dependiendo de las características que se quieran analizar [16]. Estos mallados a su vez se pueden clasificar en conformes y no conformes, si las aristas y los nodos de dos bloques diferentes coinciden o no.

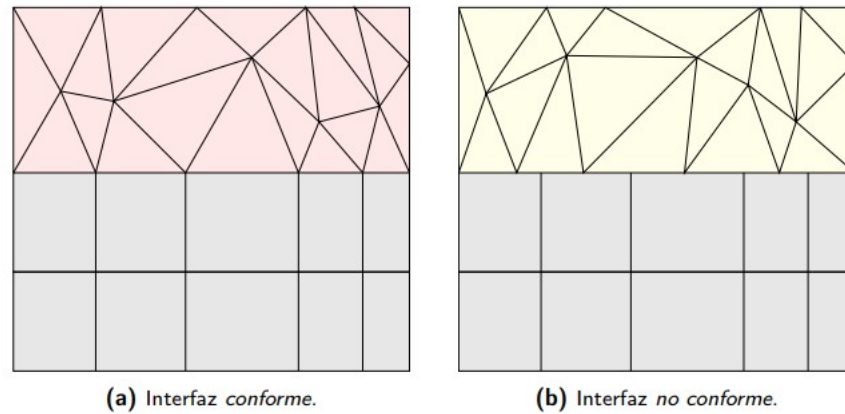


Figura 2-6 Ejemplo de conformidad y no conformidad. Fuente: [16].

- Por último, encontramos el modelo de mallado adaptativo, este tipo de mallado se adecúa a cada situación en busca de la solución óptima del problema. Este tipo de mallado se adapta a la forma de aquello que se desea estudiar de forma que es posible obtener con mayor detalle resultados en las zonas deseadas [16].

2.3.2 Mallado MeteoGalicia

En el caso de MeteoGalicia se siguen dos modelos de mallados bien diferenciados [18]:

- El primero de ellos es un mallado rectangular, en el que la separación entre nodos se mide en km. El modelo de mallado más actual que poseen es el WRF_1km_Novo, el cual posee una resolución de 1km entre nodos. Este modelo cuenta con la ventaja de ser el modelo de referencia utilizado por la agencia en el presente. Además, MeteoGalicia cuenta con mantener este mallado en el futuro.
- El otro modelo de mallado empleado por MeteoGalicia es el que mide la distancia entre nodos mediante ángulos, es decir, expresa la distancia en grados. De esta manera son capaces de proporcionar un mallado uniforme en cuanto a la distancia entre nodos.

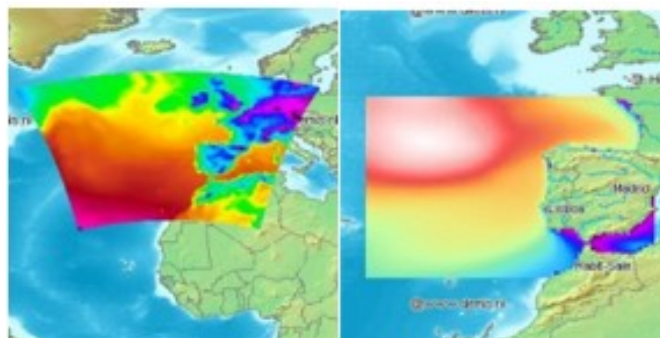


Figura 2-7 Comparación entre los mallados de MeteoGalicia. Fuente: [18].

2.3.3 Mallado H3 Uber

El sistema de mallado que utiliza la empresa Uber es crítico a la hora de manejar grandes cantidades de datos y al mismo tiempo realizar una división de la superficie terrestre eficiente para la gestión de sus servicios, entre los que se encuentran los servicios de VTC o Uber Eats. Este mallado, el conocido como H3, es un mallado específico diseñado para facilitar la toma de decisiones al mismo tiempo que optimizar los diferentes recursos que requieren los servicios de dicha empresa para optimizar su modelo de negocio [19].

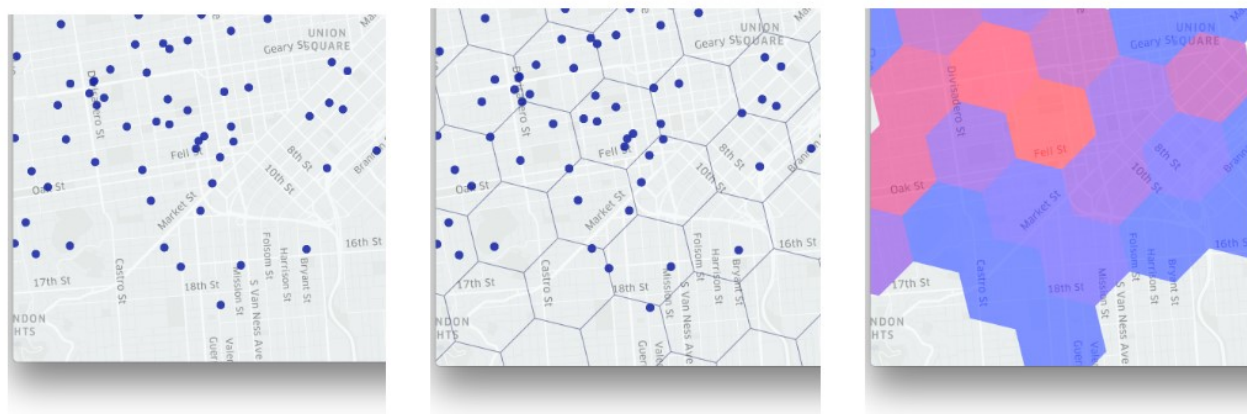


Figura 2-8 División de una superficie en hexágonos. Fuente: [18].

La empresa Uber ofrece una gran variedad de servicios que son requeridos en todo momento por sus usuarios. Para poder satisfacer las necesidades de la mejor manera posible Uber necesita que su mallado le permita ordenar la información y proporcionarles a sus conductores posiciones exactas [19]. Debido a la gran complejidad de este proceso, Uber utiliza su mallado para agrupar la información y conseguir una mejor distribución de sus empleados, realizando una medición de la oferta y la demanda en las diferentes áreas hexagonales conocidas como celdas.

Existen diferentes criterios para realizar la división del espacio en función de cómo se quiera estructurar la malla, por ejemplo, la división de una ciudad por códigos postales. Sin embargo, estos métodos no ofrecen las cualidades que se buscan en un mallado de estas características [19]. La razón principal por la que se elige un mallado hexagonal es la gran ventaja que esta forma ofrece en cuanto a medición de distancias, ya que permite aproximar radios. Y al mismo tiempo la facilidad con la que es capaz de recoger información de usuarios en movimiento, en otras palabras, datos cambiantes con el tiempo, con un índice de error muy reducido.

El sistema principalmente se compone de una representación en un mapa 2D y un mallado situado encima del mismo [19]. Es importante tener en cuenta el tipo de mapa que se está utilizando, es por esto por lo que ciertas proyecciones, como por ejemplo la mercatoriana³, no son de utilidad al no llevar a cabo una representación real en cuanto a dimensiones de la superficie terrestre, produciendo una serie de distorsiones que desembocan en que una misma celda pueda llegar a tener significativamente un mayor número de datos que sus celdas vecinas.

Es por esto por lo que Uber ha selecciona el tipo de proyección gnomónica⁴, dándole forma de icosaedro al globo terráqueo y situando los vértices de este en las zonas de menor interés para la empresa, el cual puede dividirse a su vez en un numero definido de áreas hasta conseguir la precisión deseada en la aproximación. Para dividir estas caras en áreas más pequeñas, la figura del hexaedro es importante ya que la distancia entre el centro entre una celda y sus vecinas es igual para todas ellas.

El mallado H3 permite una serie de resoluciones específicas según el área que abarquen. Cada resolución divide una celda en siete partes iguales, creando así una subdivisión de diferentes niveles [19].

³ La proyección mercatoriana es un tipo de proyección que fue utilizada en la creación de cartas náuticas ya que respeta las proporciones.

⁴ Es un Sistema de proyección geográfica, el cual se caracteriza por la representación en un plano tangente a la esfera de cualquier punto de una esfera, trazando una línea entre el centro de la esfera y el punto que se desea representar.

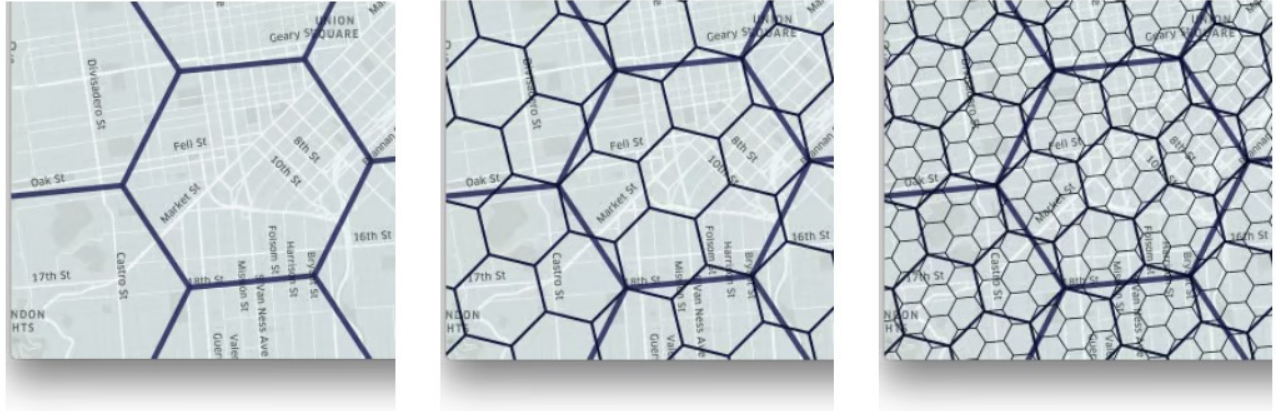


Figura 2-9 División de una superficie en hexágonos. Fuente: [18].

Como se puede observar, existen hasta 16 resoluciones diferentes, las cuales van reduciendo su valor de área dando lugar a un valor más exacto de la solución final requerida. Para conseguir las diferentes resoluciones, se deberá dividir el área de cada celda por un factor de escala equivalente a $\frac{1}{\sqrt{7}}$ [19, 20].

H3 Resolution	Average Hexagon Area (km ²)	Average Hexagon Edge Length (km)	Number of unique indexes
0	4,250,546.8477000	1,107.712591000	122
1	607,220.9782429	418.676005500	842
2	86,745.8540347	158.244655800	5,882
3	12,392.2648621	59.810857940	41,162
4	1,770.3235517	22.606379400	288,122
5	252.9033645	8.544408276	2,016,842
6	36.1290521	3.229482772	14,117,882
7	5.1612932	1.220629759	98,825,162
8	0.7373276	0.461354684	691,776,122
9	0.1053325	0.174375668	4,842,432,842
10	0.0150475	0.065907807	33,897,029,882
11	0.0021496	0.024910561	237,279,209,162
12	0.0003071	0.009415526	1,660,954,464,122
13	0.0000439	0.003559893	11,626,681,248,842
14	0.0000063	0.001348575	81,386,768,741,882
15	0.0000009	0.000509713	569,707,381,193,162

Tabla 2-2 Resolución celdas mallado H3. Fuente: [50].

Actualmente el código utilizado para el desarrollo de este tipo de mallado se encuentra disponible en internet en diferentes lenguajes de programación. Este código precisa de una librería específica que transforma la latitud y longitud de un punto en concreto en una entrada de 64-bit, asignado así una celda en concreto.

2.4 Fuentes de datos

Uno de los pilares fundamentales en la elaboración de gemelos digitales, es la continua obtención de datos del objeto o sistema que se quiere controlar. De esta forma en la Armada Española, para el desarrollo de un “Gemelo Digital Oceanográfico” es posible contar con diferentes servicios y fuentes que suministren de los datos necesarios a las futuras bases de datos.

2.4.1 Fuentes nacionales

Actualmente el Ministerio de Defensa cuenta con un servicio de apoyo a sus unidades en ámbitos de meteorología. Desde sus inicios, con la creación del Real Observatorio de la Isla de León, y haciendo énfasis en sus predicciones de primera clase (sobre el tiempo), hasta lo que se conoce hoy como el Real Observatorio de San Fernando [21].

Desde entonces, el Instituto Hidrográfico de la Marina (IHM), ha tomado el relevo y continua con todas las misiones relativas a la seguridad en la navegación, así como difundir toda la información relevante en cuando al mar y al litoral. De esta forma, aquellas misiones de mayor importancia para apoyar el desarrollo de un “Gemelo Digital Oceanográfico” son la continua observación y monitorización de mareas y corrientes, la temperatura y todos aquellos fenómenos físicos que afecten a la meteorología de diferentes regiones. De la misma forma, es importante destacar que el IHM participa en diferentes misiones internacionales relacionadas con este ámbito [22].

De la misma forma que el IHM apoya en cuestiones meteorológicas a la Armada Española, la Agencia Estatal de Meteorología (AEMET), es la encargada del asesoramiento meteorológico en todo lo que necesiten las Fuerzas Armadas y los Cuerpos y Fuerzas de Seguridad del Estado para el cumplimiento de su misión de asegurar la defensa nacional. AEMET apoya activamente tanto a las misiones en el extranjero, como las operaciones locales. Para ello cuenta con múltiples oficinas destacadas en los diferentes ejércitos, y con oficinas especializadas en la Unidad Militar de Emergencias (UME) [21].

En el ámbito de asistencia a emergencias, la meteorología juega un papel protagonista a la hora de llevar a cabo extinciones de incendios o el planeamiento de evacuaciones en caso de fuertes lluvias o inundaciones. [23]. Para apoyar a esta unidad, esta recibe un informe diario de las condiciones meteorológicas de las posibles zonas de actuación, y en caso de que se declare emergencia de nivel tres, contarán con personal experto en predicciones meteorológicas.

Hoy en día, los retos a los que se enfrenta la meteorología en las Fuerzas Armadas es el desarrollo de las “Ayudas de Decisión Tácticas” (TDAs de su nombre en inglés), de tal manera que sean capaces de representar de forma clara y concisa el impacto de la meteorología en las operaciones militares, tornándose de vital importancia el establecimiento de una serie de valores límite que afecten de forma crítica a estas operaciones [21].

Para conseguir estos objetivos, AEMET trabaja en el desarrollo de “*Recognized Enviromental Pictures*” (REP), las cuales son capaces de integrar en un conjunto de información de distinto carácter, de tal forma que sea posible disponer de toda la información relevante en un teatro de operaciones [21]. Esto requiere que toda la información sea dividida y proporcionada en diferentes capas, de tal manera que sea más sencillo gestionar los modelos matemáticos, los datos convencionales y las imágenes obtenidas por telemetría.

Junto con la colaboración de la AEMET para el Ministerio de Defensa, este cuenta con el “Centro Nacional de Predicción para Defensa” (CNPDP), el centro neurálgico del apoyo prestado por la AEMET [23]. Este centro estudia y monitoriza de manera continua las condiciones meteorológicas donde España tiene en curso una operación militar, o es de interés conocer dichas condiciones. De la misma forma que AEMET, el CNPDP cuenta con diversas oficinas distribuidas en puntos clave distribuidos por la geografía nacional, estando estas situadas en bases militares debido a su naturaleza estratégica.

Este centro se especializa en llevar a cabo predicciones en la fase de preparación de las misiones, y nutrir de manera continua y bajo demanda del mando desplegado de información meteorológica, así como avaluar cuales pueden ser los límites que afecten al desarrollo de las misiones en el extranjero.

El análisis meteorológico es un factor determinante en el éxito de las misiones. Tanto es así que, en la Misión Atalanta, que lucha contra la piratería en el Océano Indico, el conocimiento de los diferentes fenómenos que afectan a esta zona, como los vientos monzónicos, los cuales afectan directamente a la actividad de las pequeñas embarcaciones destinadas a realizar actos de piratería, afecta directamente a la planificación en ámbito de unidades destinadas a la prevención de dichos actos. De la misma forma un análisis previo adecuado de las condiciones que presenta el estado de la mar ayuda a la toma de decisiones del tipo de unidad que será desplegada, ya que para ciertas unidades como corbetas o fragatas esto puede ser un factor determinante. Así mismo es posible evaluar la seguridad para la navegación de cualquier tipo de unidad y por consiguiente analizar si la seguridad de la unidad es la adecuada para el desarrollo de la operación [23].

2.4.1 Fuentes abiertas

De la misma forma que es posible obtener información de fuentes oficiales, existen multitud de fuentes abiertas y de acceso libre que informan sobre las condiciones meteorológicas en diferentes puntos de la península ibérica, y del mundo.



Figura 2-10 Buque Hidrográfico Hespérides atravesando mala mar. Fuente: [22].

Una de estas fuentes abiertas es “Puertos del Estado”. Este organismo, dependiente directamente del Ministerio de Transportes, Movilidad y Agenda Urbana, cuenta con presencia en cuarenta y seis puertos diferentes [24]. Sus responsabilidades son, entre otras, la formación y promulgación de investigación y desarrollo tecnológico relacionados con las actividades portuarias, así como de la gestión del correspondiente sistema de señalización marítima española [6].

Puertos del estado cuenta con un sistema cuyo objetivo principal es el proporcionar los datos meteorológicos esenciales para el diseño y explotación de los centros portuarios nacionales. Sin embargo, al mismo tiempo exporta estos datos ofreciendo un servicio activo a diferentes públicos. El sistema está compuesto por una red de sistemas, entre los que se incluyen boyas, mareógrafos o incluso radares de altas frecuencias. Junto con esto existen también una serie de servicios de predicción de diferentes fenómenos meteorológicos [24].

Para la obtención de los datos de interés, puntos SIMAR (datos modelados), es posible seleccionar las boyas que aportaran datos de la zona deseada, y descargar los ficheros en el formato CSV, para su posterior conversión y utilización.

De la misma forma que Puertos del Estado, la Red Europea de Observación y Datos Marinos (EMODnet) [25], es una red apoyada por las políticas marítimas de la UE, la cual colabora con otras organizaciones en una continua monitorización del entorno marino y su posterior análisis conforme a los requisitos establecidos. Los resultados obtenidos son publicados de manera que es posible su obtención de forma gratuita.

Actualmente, EMODnet ofrece información sobre siete disciplinas bien diferenciadas, entre las que destacan por su aportación a la elaboración de un “Gemelo Digital Oceanográfico”:

- los datos relacionados con batimetría, los datos relacionados con la profundidad del océano, sus costas y posibles accidentes submarinos;
- la geología ya que gracias al estudio de esta es posible determinar ciertos comportamientos de la costa que pueden llegar a afectar a, por ejemplo, las mareas; y
- la conocida como disciplina física, la cual es capaz de aportar diferentes datos relacionados con los comportamientos físicos del fluido en el que operan las unidades marítimas. De esta manera es posible obtener datos como la salinidad del agua, su temperatura, vientos o corrientes.

Para poder acceder a esta información EMODnet, ha creado una serie de servidores que, organizados en diferentes capas, almacenan toda la información y la acercan al usuario final de una forma clara y ordenada.

Actualmente EMODnet ofrece un servicio desarrollado y en vías de mejora con la colaboración de ciento veinte organizaciones [25]. Algunos de estos objetivos que persiguen los programas en desarrollo son la recolección de datos, elaborando históricos de manera que estos se puedan utilizar de manera recurrente, procesar y validar datos en diferentes niveles. Es decir, actualmente es posible llevar esta acción a cabo a nivel nacional, sin embargo, el poder hacer esta operación a nivel internacional está en vías de desarrollo. Y, por último, ayudar en la elaboración de una herramienta que ayude en la toma de decisiones y que al mismo tiempo suponga una experiencia guiada para el usuario.

La obtención de los datos necesarios de esta fuente se realiza a través de su página web, donde tiene la opción de escoger los diferentes datos de interés para el usuario y descargarlos en diferentes formatos, de manera que sea posible trabajar cómodamente con los mismos [6]. La batimetría es uno de los diferentes fenómenos importantes en el estudio del medio marino. Los diferentes modelos de predicción cuentan con un servicio amplio de diferentes fenómenos meteorológicos. Los cuales sumados a la batimetría forman los cimientos del “Gemelo Digital Oceanográfico”.

Es importante tener en cuenta que a la hora de realizar un seguimiento de los fenómenos meteorológicos que afectan a regiones específicas, las fuentes que mayor precisión tienen en sus predicciones son las de carácter local. Es por esto por lo que MeteoGalicia, se ha posicionado como una de las fuentes principales a la hora de la obtención de datos meteorológicos de las Rías Baixas.

El cometido principal de esta agencia es la elaboración de una predicción meteorológica completa del territorio gallego, así como la correcta explotación y mantenimiento de todas las estaciones meteorológicas distribuidas por el territorio [26]. Entre los servicios que ofrece destacan la predicción operativa y la predicción numérica de los datos. De la misma forma, es posible encontrar una serie de modelos meteorológicos, de oleaje y oceanográficos de muy alta precisión, lo que permite realizar predicciones de muy elevada exactitud.

Gracias a la elevada precisión en las predicciones, así como su continua monitorización, MeteoGalicia será la fuente principal de datos para la elaboración del “Gemelo Digital Oceanográfico”, de esta manera es posible obtener datos que oscilen temporalmente entre datos mensuales, diarios o con precisión de minutos [6]. Así mismo, una vez elegidos los datos que se desean monitorizar, es posible obtener un registro de datos históricos que abarcan hasta el año de antigüedad.

Una de las múltiples ventajas que ofrece esta fuente, es la posibilidad de la descarga de datos en el formato JSON, lo cual permitiría la subida de la información a la base de datos sin la necesidad de llevar a cabo ningún tipo de tratamiento sobre los mismos [6]. Sin embargo, en el marco del presente trabajo, es importante establecer una serie de relaciones entre los diferentes datos.

Debido a la gran cantidad de información que es posible obtener de este tipo de fuentes, es importante resaltar cual es el objetivo principal que se busca conseguir en cada modelo de “Gemelo Digital Oceanográfico” gracias a lo que es posible llevar a cabo un filtrado de datos. De esta manera se obtendrán únicamente los datos relacionados con la seguridad a la navegación, obtenidos de las diferentes fuentes, tanto nacionales como abiertas.

De la misma forma que MeteoGalicia permite un filtrado de información, esta operación es posible repetirla para la localización geográfica de las distintas estaciones que son de interés, ya que el presente trabajo se centrará en un área geográfica muy concreta, tal y como aparece en la Figura 2-11.

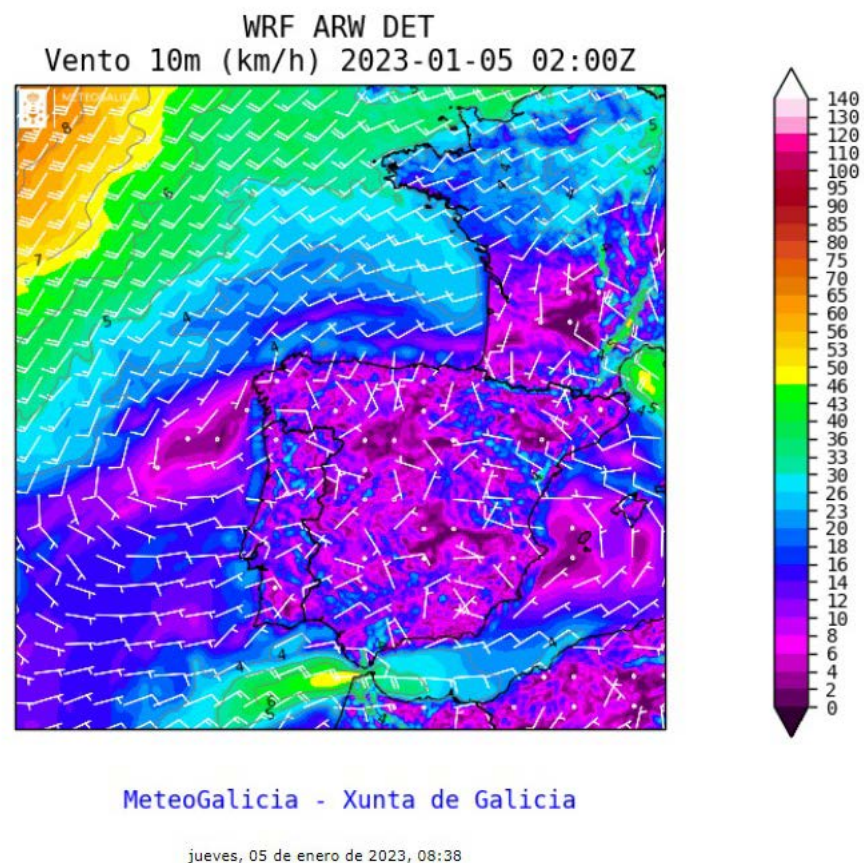


Figura 2-11 Modelo Numérico MeteoGalicia. Fuente: [51].

2.5 Bases de datos

Las bases de datos son, en su definición técnica, un programa o archivo que contiene diferentes tipos de información organizada y estructurada de tal forma que pueda ser utilizada por el usuario de manera ágil y rápida. Su creación e implementación ha ido de la mano de la evolución humana. En sus inicios, las bases de datos eran utilizadas por los campesinos que las utilizaban para llevar un control de sus cosechas. Posteriormente, en los últimos años del siglo XIX, se introdujo un sistema de tarjetas perforadas [27], inventado por Herman Hollerith que fue utilizado para llevar un censo en diferentes localidades de EE. UU.

Posteriormente en 1950, introducidas por Oberlin Smith, aparecieron las cintas magnéticas [27], las cuales permitían almacenar una gran cantidad de información que podía ser consultada únicamente de manera secuencial. Esto conllevaba el problema de que, una vez finalizada la búsqueda de un dato, para realizar la búsqueda de otro se requiere volver al inicio de la cinta. Es decir, cada vez que se quería iniciar una nueva búsqueda era necesario volver al inicio.

Finalmente, en las últimas décadas del siglo XX, han acontecido una serie de hechos que han llevado las bases de datos al siguiente nivel [27]. En los primeros años del siglo XXI las bases de datos eran orientadas a objetos y permitían llevar a cabo búsquedas de información de una manera mucho más rápida y fiable. Sin embargo, durante la última década se han desarrollado una serie de mejoras que han llevado a las bases de datos a los que hoy se conoce como “No Solo SQL” (NoSQL, Structured Query Language), las cuales nacen como respuesta al enorme crecimiento de información derivado del uso de internet, al mismo tiempo que de la necesidad de llevar a cabo un procesamiento y almacenamiento de datos mejorado [28].

2.5.1 Formato JSON y bases de datos

Existen multitud de bases de datos a disposición del usuario en función de cuales sean sus objetivos finales con la información que se va a almacenar en dichos ficheros. Desde las bases de datos relacionales, en las que la información se presenta en forma de columnas, de manera que es fácil accesible y adecuada para bases de datos estructuradas. Hasta las bases de datos NoSQL, las cuales permiten al usuario almacenar, procesar y trabajar con una serie de datos no estructurados. Para poder trabajar con este tipo de bases de datos se requiere de un tipo específico de lenguaje que, en este caso, sería el formato “*JavaScript Object Notation*” (JSON) [28].



Figura 2-12 Ejemplo datos desordenados Puerto de Marín, en formato JSON. Fuente: [6].

El formato JSON es un formato de gran versatilidad, ya que es capaz de ser interpretado por diferentes entornos fuera de su formato original. Mediante la implementación de este formato es posible, partiendo de una serie de datos que a primera vista pueden parecer desordenados, como se observa en la Figura 2-12 pasar a un orden más claro como el que se observa en la Figura 2-13. En esta última, todos los datos aparecen con un orden parecido, ya que todos comparten una estructura similar [6].

```
{
  "predConcello": {
    "idConcello": "36038",
    "listaPredDiaConcello": [
      {
        "ceo": {
          "manha": 103,
          "noite": 203,
          "tarde": 103,
          "dataPrediccion": "2023-01-05T00:00:00",
          "nivelAviso": 0,
          "pchoiva": 1
        },
        "manha": {
          "5": {
            "noite": 5,
            "tarde": 5,
            "tMax": 16,
            "tMin": 6,
            "tmaxFranxa": null,
            "tminFranxa": null,
            "uvMax": 1,
            "viento": {
              "manha": 299,
              "noite": 304,
              "tarde": 300
            },
            "ceo": {
              "manha": 184,
              "noite": 211,
              "tarde": 104,
              "dataPrediccion": "2023-01-06T00:00:00",
              "nivelAviso": 0,
              "pchoiva": 1
            },
            "manha": {
              "5": {
                "noite": 90,
                "tarde": 140,
                "tMax": 14,
                "tMin": 8,
                "tmaxFranxa": null,
                "tminFranxa": null,
                "uvMax": 1,
                "viento": {
                  "manha": 304,
                  "noite": 313,
                  "tarde": 313
                },
                "ceo": {
                  "manha": 108,
                  "noite": 211,
                  "tarde": 118,
                  "dataPrediccion": "2023-01-07T00:00:00",
                  "nivelAviso": 1,
                  "pchoiva": 1
                },
                "manha": {
                  "80": {
                    "noite": 95,
                    "tarde": 80,
                    "tMax": 15,
                    "tMin": 12,
                    "tmaxFranxa": null,
                    "tminFranxa": null,
                    "uvMax": 1,
                    "viento": {
                      "manha": 314,
                      "noite": 322,
                      "tarde": 314
                    },
                      "ceo": {
                        "manha": 188,
                        "noite": 208,
                        "tarde": 107,
                        "dataPrediccion": "2023-01-08T00:00:00",
                        "nivelAviso": null,
                        "pchoiva": 1
                      },
                      "manha": {
                        "80": {
                          "noite": 70,
                          "tarde": 80,
                          "tMax": 14,
                          "tMin": 10,
                          "tmaxFranxa": null,
                          "tminFranxa": null,
                          "uvMax": 9999,
                          "viento": {
                            "manha": 307,
                            "noite": 307,
                            "tarde": 315
                          },
                          "nome": "Pontevedra"
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    ]
  }
}
```

Figura 2-13 Ejemplo datos ordenados formato JSON. Fuente: [6].

Este tipo de formato para los datos se utiliza mayoritariamente en entornos en los que el flujo de datos entre el usuario y el servidor es de vital importancia, al mismo tiempo que se requiere una interacción por parte del usuario con la máquina. Este formato a su vez reduce el número de elementos que se representan a la hora de su utilización, y junto a su simpleza hacen que estos sean fácilmente entendibles por el ser humano, lo que ha llevado a este formato a convertirse en uno de los estándares más utilizados actualmente. Además de esto, el formato JSON requiere de mucha menos memoria, lo que lo hace más fácilmente procesable y por lo tanto aumenta la rapidez con la que los datos pueden ser refrescados en la base de datos [29].

Como muestra la Figura 2-12 existe la posibilidad de almacenar datos de una forma aparentemente no estructurada, estos se almacenan en las conocidas como bases de datos no estructuradas. Este tipo de bases de datos han evolucionado hasta llegar a lo que hoy se conoce como NoSQL, estas son sistemas de almacenamiento de información que no requiere que se cumpla el requisito esquemático entidad-relación [30]. Por lo que, para el almacenamiento de datos, hacen uso de otros métodos como el método de clave-valor o el mapeo de columnas. De esta manera se consiguen resolver problemas de escalabilidad y rendimiento en ciertas aplicaciones.

Este tipo de bases de datos ofrece una serie de ventajas frente a las bases de datos estructuradas como la utilización de pocos recursos, es decir, no requieren de una conmutación excesiva por lo que pueden ser ejecutadas sobre sistemas menos costosos. Al mismo tiempo este tipo de bases de datos permite añadir más nodos a la red, con el único requisito de indicar al sistema que nodos son aquellos que se encuentran disponibles, esto se conoce como escalabilidad horizontal [30]. Sin embargo, el principal problema que estas bases de datos resuelven frente a las estructuradas es la generación de cuellos de botella. Los sistemas SQL requieren un método de ejecución complejo, ya que hay que transcribir cada sentencia para que esta pueda llegar a ser ejecutada, lo que hace que el sistema pueda llegar a producir pérdidas de datos o incluso saturarse en caso de que se den simultáneamente multitud de peticiones de ejecución.

2.5.2 Tipos de bases de datos NoSQL

Dentro de las bases de datos no estructuradas existen diferentes modelos, los cuales difieren unos de otros en función de los mecanismos que emplean para almacenar la información, estos son, el modelo de clave-valor, bases de datos documentales, bases de datos de grafos y las bases de datos columnares.

El modelo conocido como “Clave-Valor” es el modelo más utilizado en la mayoría de las bases de datos, los sistemas de la nube o incluso en muchas aplicaciones para el análisis del “*Big Data*”, esto se debe a su simplicidad a la hora de establecer la base de datos, además de las garantías que ofrece en cuanto a consistencia, persistencia y rendimiento. La estructura básica de este modelo consiste en identificar uno o varios datos con una clave única, de tal manera que la información es fácilmente accesible una vez conocida la clave asociada a la información que se desea buscar [30, 31].

Esta información se representa en columnas, donde aparece en una columna la clave y a continuación toda la información relacionada con esa clave. Esto a su vez dará lugar a tablas, y estas a su vez se podrán dividir en celdas, y en cada celda habrá contenidos un tipo de información específica.

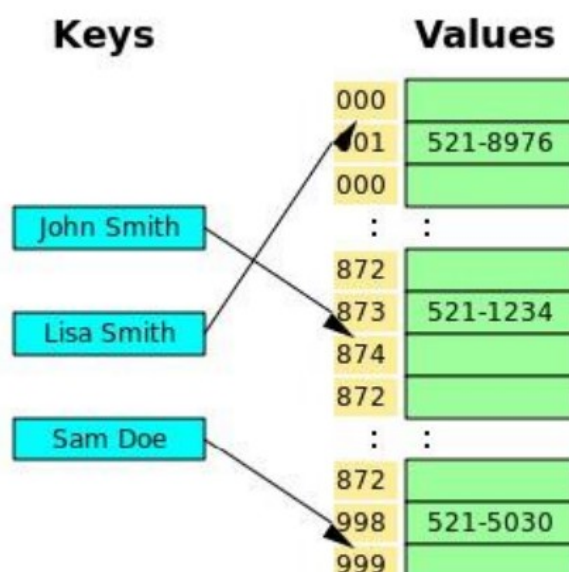


Figura 2-14 Modelo "Clave-Valor" Fuente: [29].

Esta información puede ser de naturaleza similar o completamente distinta. Dentro de estas tablas, el modelo “Clave-Valor” permite que, dependiendo de la necesidad del usuario, existan claves repetidas. Esta forma de almacenamiento de la información presenta el inconveniente de que no realiza funciones de comprobación de la integridad de los datos, delegando este tipo de funciones a la capa aplicación [32]. Estos modelos están pensados para proporcionar alta disponibilidad y separación de datos.

En segundo lugar, se encuentran las bases de datos de grafos, las cuales estructuran la información almacenándola como nodos de una red, y estableciendo la relación entre la información como los nexos entre los nodos. De esta manera el resultado final son dos columnas en las que los nexos únicamente pueden conectar dos celdas de cada columna. Este modelo de base de datos requiere de una normalización específica, ya que, para poder acceder a la información almacenada en la misma, mediante la teoría de grafos, es necesario conocer la estructura de la red, ofreciendo una mayor eficiencia a la hora de realizar búsquedas de información.

Este modelo de base de datos requiere que las relaciones entre los datos tengan la cualidad de poder ser representadas como nexos, es decir, elementos interconectados con un número indeterminado de relaciones entre estos. En otras palabras, esta información contiene un nexo directo a los nodos adyacentes, lo que hace que no sea necesaria una consulta por índices [32].

En la actualidad se trabaja con un modelo que cumple tres funcionalidades necesarias:

- En primer lugar, es un modelo etiquetado, es decir, cada nodo contiene una etiqueta en la que aparece información que lo identifica adecuadamente.
- En segundo lugar, es un modelo dirigido, en otras palabras, los nexos entre los diferentes nodos determinan el sentido del flujo de información, de tal manera que son los propios nexos entre nodos los que indican el camino a seguir cuando sea necesario recorrerlo en busca de información. En último lugar, este modelo de grafos es atribuido, lo que significa que cada nodo permite una lista de valores asociados a los diferentes nombres, de manera que la estructura del grafo se simplifique considerablemente.

Además de todos estos, los modelos multi-grafo deben permitir que el mismo nodo sea capaz de conectarse por diferentes nexos a todos los nodos necesarios, es decir, el número mínimo de nexos de los que disponga un nodo no debe ser un factor limitante.

- En tercer lugar, se encuentran las bases de datos documentales, estos datos se denominan datos “semiestructurados” como pueden ser JSON o “*eXtensible Markup Language*” (XML) [30]. Este método de almacenamiento permite una búsqueda de información que además de organizar los datos de una forma muy similar a la empleada por el modelo “Clave-Valor”, es posible realizar funciones avanzadas con dicha información ya que este modelo permite en cierta manera encapsular la información procedente del modelo “Clave-Valor” y dar lugar a salidas denominadas documentos. Este tipo de bases de datos son las más versátiles, ya que actualmente es posible emplearlas con sistemas que requieran tanto de datos no estructurados como aquellas que requieran datos relaciones.

Finalmente se encuentran las bases de datos columnares, su principal ventaja es la velocidad de procesamiento debido a la forma vectorial en la que se comprimen y almacenan los datos [33]. Este formato de bases de datos permite mantener un tiempo constante en la manipulación de la información, mientras que en otro tipo de bases de datos como en MySQL estos tiempos se ven aumentados a medida que el volumen de datos aumenta, posicionándose como candidatas principales a la hora de solucionar la problemática que supone el procesamiento del “*big data*”.

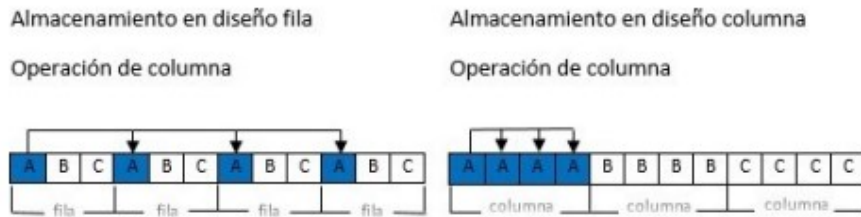


Figura 2-15 Modelo base de datos columnar. Fuente: [33].

A modo de resumen, para poder comprobar el comportamiento entre diferentes modelos de bases de datos según su tipo y según la cantidad de información que esta base de datos es capaz de procesar obtenemos la Tabla 2-3. En la que se puede comprobar como las bases de datos NoSQL presentan un procesamiento más veloz frente a las SQL (MySQL y PostgreSQL). Además, las bases de datos MonetDB y Cassandra siguen el modelo columnar, mientras que MongoDB se orienta hacia el modelo clave-valor.

Primera consulta – clave-valor				
# registros				
Base de datos	1 MM	10 MM	25 MM	50 MM
MySQL	2	2,4	2,8	3,2
PostgreSQL	9	10,2	12,6	13
MongoDB	2,8	3	2,8	2,8
MonetDB	4	4,6	5	4,2
Cassandra	3,4	5	4,8	7,4

Tabla 2-3 Tiempo de procesamiento de diferentes bases de datos. Fuente: [33].

3 MATERIALES Y MÉTODOS

Una vez finalizado el análisis sobre el desarrollo del Gemelo Digital, así como los diferentes factores que afectan en el desarrollo de este. En el siguiente apartado se expondrán la arquitectura básica del concepto inicial de Gemelo Digital Oceanográfico, asentando las bases iniciales para entender y poder solucionar el problema presentado.

En segundo lugar, se expondrán las herramientas que han sido utilizadas para abordar la problemática y poder conseguir los objetivos propuestos, tanto el lenguaje de programación propuesto, Python, el cual servirá para llevar a cabo todas las acciones necesarias en el desarrollo del trabajo. Así como de la plataforma de bases de datos MongoDB, la cual será empleada para la creación y almacenamiento en la nube de los datos descargados.

Durante el desarrollo del presenta apartado de expondrán los diferentes códigos empleados para la solución de los problemas encontrados, con el objetivo de proporcionar al Gemelo Digital de la información que este necesita.

3.1 Arquitectura

La arquitectura principal empleada en el siguiente trabajo se basa en una comunicación a través de la red para la consulta de la información y su posterior representación gráfica en un tiempo actualizado y de manera dinámica. Para ello es importante analizar previamente las dos partes principales del Gemelo digital Oceanográfico, el cual cuenta con un equipo basado físicamente en tierra que sirven de apoyo al equipo ubicado en las unidades desplegadas, las cuales son las encargadas de hacer una solicitud de la información y posteriormente representarla. Por tanto, se trata de una modalidad cliente-servidor.

La razón que persigue esta división es la facilitar el acceso a la información con el mínimo equipo necesario a bordo de las diferentes unidades, de esta manera cuando se esté llevando a cabo la preparación de una navegación en un puerto extranjero o durante la misma navegación, será posible observar gráficamente el entorno marino y por lo tanto analizar los posibles peligros a la navegación derivado de los diferentes factores meteorológicos. Así mismo supondría una herramienta fundamental para elaborar posibles soluciones a las alteraciones derivadas de la meteorología, es decir, este modelo de recopilación y representación de información ayudará al análisis de la situación y a la propuesta de diferentes alternativas en caso de complicaciones a la hora de realizar los diferentes ejercicios que se realizan durante las navegaciones, así como proponer o encontrar diferentes alternativas a las derrotas planeadas.

Por un lado, el servidor en tierra será el encargado de almacenar toda la información relativa a los diferentes parámetros de interés, no solo en tiempo real y a nivel previsión, sino siendo capaz también de almacenar datos pasados para permitir la elaboración de históricos. Los datos deberán ser descargados de nuevo de acuerdo con la frecuencia con la que estos son actualizados por la fuente, de tal forma que sea posible ofrecer al usuario los datos más actualizados. Es importante tener en cuenta que, dependiendo del fenómeno que se desee conocer, este afectará a una superficie variable. Por esta razón, se procederá a implementar un mallado multinivel que permita seleccionar diferentes tamaños de malla.

Es importante reseñar que en este servidor se almacenará toda la información sin importar el área geográfica específica en la que se encuentre la unidad solicitante de información, ya que este filtro lo realizará el programa implementado en el presente proyecto.

Por otro, del lado del cliente, se encuentra la unidad desplegada, y será quien solicite la información que desea conocer al equipo. Proporcionándole a los servidores unas coordenadas que permitan la localización y por lo tanto representación correcta de los datos en ese momento. Para poder acceder a esta información, el buque deberá indicar una serie de datos que permitan conocer su localización, entre los que se encuentran, por ejemplo, la calidad del mallado deseado y el radio de acción.

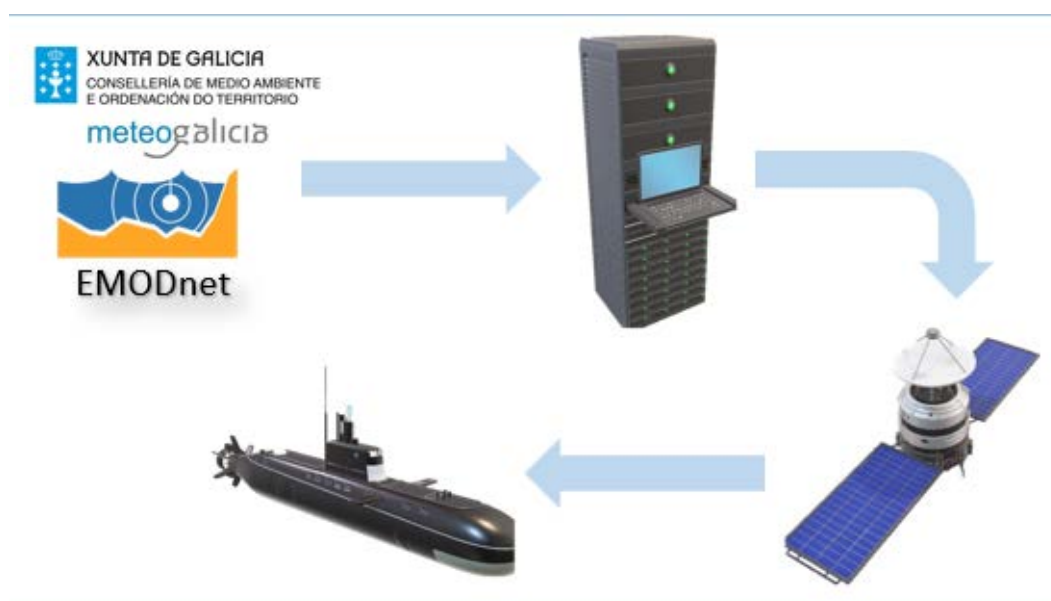


Figura 3-1 Arquitectura Global GDO. Fuente: Propia.

De manera general el buque desplegado hará una solicitud de información a la base de datos en la que le indique el fenómeno meteorológico, la hora, el radio de acción, la resolución del mallado y la localización en la que se desea conocer cierta información meteorológica. El servidor responderá a la unidad con la información más actualizada que disponga y en función del movimiento de este continuará enviando la información requerida.

Uno de los principales problemas que enfrenta este modelo de Gemelo Digital Oceanográfico, es el de las diferentes resoluciones posibles debidas a las características de los diferentes parámetros meteorológicos. Esto se debe a que, dependiendo de los modelos empleados en la generación de la información descargada, esta tendrá mayor o menor área de acción. El resultado de esta problemática resulta en que no es posible la creación de un único mallado en el que se representen todos estos datos con una misma precisión, sin embargo, es posible la creación de un modelo multinivel en el que se representen diferentes mallados hexagonales dependiendo de la variable meteorológica que se necesite. A modo de ejemplo, un mallado para el viento tendrá una resolución menor que un mallado para la temperatura y por lo tanto el tamaño de las celdas cambiará en consonancia.

3.2 Software empleado

3.2.1 Python

A la hora de desarrollar aplicaciones, es necesario introducir las herramientas necesarias para su construcción. Python es uno de los lenguajes más populares y más utilizado por los programadores.

Python es un lenguaje conocido como interpretado o de script, es decir, que para que este programa funcione, requiere de una serie de funcionalidades de interprete y lo traduzca de manera que el lenguaje utilizado pueda ser interpretado y entendido por la máquina. Esta característica hace que este lenguaje sea más flexible y ofrezca una mayor portabilidad.

Python es un lenguaje multiplataforma, lo que permite ejecutar el programa en los diferentes sistemas operativos con la condición de que no se estén utilizando librerías específicas que así lo impidan. Este tipo de lenguaje permite una interpretación sencilla, como se puede observar en la Figura 3-2, a la vez que es capaz de desarrollar complejas aplicaciones de manera rápida y aportando una gran cantidad de bibliotecas [34].

```
import csv
class FormatData:
    def __init__(self, Name="", Age=0, Married=False):
        self.Name = Name
        self.Age = Age
        self.Married = Married
    def __str__(self):
        OutString = "{0}, {1}, {2}".format(
            self.Name,
            self.Age,
            self.Married)
        return OutString
def SaveData(Filename = "", DataList = []):
    with open(Filename, "w", newline='\n') as csvfile:
        DataWriter = csv.writer(
            csvfile,
            delimiter='\n',
            quotechar=" ",
            quoting=csv.QUOTE_NONNUMERIC)
        DataWriter.writerow(DataList)
        csvfile.close()
        print("Data saved!")
def ReadData(Filename = ""):
    with open(Filename, "r", newline='\n') as csvfile:
        DataReader = csv.reader( csvfile, delimiter="\n", quotechar=" ", quoting=csv.QUOTE_NONNUMERIC)
        Output = []
        for Item in DataReader:
            Output.append(Item[0])
        csvfile.close()
        print("Data read!")
        return Output
```

Figura 3-2 Ejemplo aplicación Python. Fuente: Propia.

Una de las grandes ventajas con las que cuenta Python es su entorno integrado IDLE (de su nombre en inglés “Integrated DeveLopment Enviroment”) [35], el cual es un entorno que facilita la escritura de códigos y tiene buenas características de rendimiento para aplicaciones pequeñas, además de contar con un resaltado de código y detector de errores.

Actualmente Python es una herramienta comúnmente utilizada en el campo de la ciencia, tanto es así que es una de las más populares en centro como la Organización Europea para la investigación nuclear (CERN), junto con otros campos como la física, biología o química [35]. Todo esto es debido a que Python es capaz de aportar a los científicos una gran cantidad de bibliotecas en el ámbito de procesamiento de señales o estadística, junto con una muy buena documentación.

Entre estas herramientas destacan los paquetes Python Numérico (PyNum) y Python Científico (SciPy), gracias a los cuales es posible emular acciones similares a las de programas como Matlab, y leer los archivos generados por esa misma aplicación. De la misma forma el Toolkit Modular para Procesamiento de Datos (MPD), ofrece funcionalidades avanzadas en el campo de procesamiento de señales avanzadas. Todas estas herramientas junto con netCDF y librerías específicas, hacen mucho más fácil la estandarización y el intercambio de información [35].

3.2.2 MongoDB

Dentro de los diferentes tipos de bases de datos, se encuentra la conocida MongoDB, la cual actualmente es una de las bases de datos más utilizada por los desarrolladores ya que se trata de un modelo orientado a los documentos y que permite que cada entrada de datos no deba tener ningún tipo de relación con el resto de las entradas ya almacenadas, es decir, es una base de datos de esquema libre [30]. Esta base de datos es de código libre y escrita en C++, flexible, ofreciendo alta escalabilidad y rendimiento.

Esta base de datos, a su vez, emplea un lenguaje actualizado del formato JSON, el conocido como Binary JSON (BSON), el cual permite acelerar el acceso a cualquier tipo de información contenida en un documento, siendo el driver de la propia base de datos el encargado de llevar a cabo esta conversión entre el formato JSON y el BSON ya que este último no es legible por el ser humano [36].

Esta base de datos es capaz de proporcionar una navegación a través de sus datos de la manera más eficiente posible. A la hora de elaborar bases de datos encargadas de almacenar mucha información y de naturaleza diferente es posible establecer una relación entre estas y almacenarla así en bases de datos basadas en grafos [37]. Es por esto por lo que MongoDB ofrece la posibilidad de transformarse en una base de datos orientada a ese modelo. De esta forma la información se almacena en nodos y las diferentes relaciones entre estos nodos se expresa en las aristas, permitiendo a la base de datos priorizar las diferentes relaciones existentes entre los nodos.

Esta base de datos tiene una serie de cualidades que la hacen excelente para poder manejar la información en el ámbito de este trabajo de la mejor manera posible. Entre las características más importantes destacan [38]:

- En primer lugar, se encuentran las consultas ad hoc, esto significa que es posible realizar consultas con diferentes objetivos, es decir, se pueden solicitar partes concretas de la información almacenada o incluso se puede solicitar la devolución de varios de los mismos. Además, es posible acceder y solicitar esta información de diversas formas, según sea el objetivo deseado.
- En segundo lugar, se encuentra la capacidad del balanceo de carga, es decir, la capacidad de ser accesible y poder ejecutarse desde diferentes servidores de manera simultánea, de esta forma MongoDB ofrece un servicio de replicación de los datos, consiguiendo al mismo tiempo no interrumpir su funcionamiento en caso de que se produzca error en alguno de los diferentes servidores.

Una de las cualidades complementarias más importantes para el desarrollo del presente trabajo, es la de la seguridad de los datos una vez hayan sido cargados en la base de datos, por esto es importante establecer una serie de medidas de seguridad que desde la misma plataforma de MongoDB es posible seleccionar.

Para ello se trabajará con MongoDB Atlas, una aplicación de MongoDB que permite controlar y monitorizar diferentes aspectos de la base de datos en la red de manera ágil y sencilla. Esto permite simplificar los procedimientos de seguridad, así como asistir en la mejora de esta.

A continuación, se seleccionará el método de acceso a la base de datos, siendo esta mediante usuario y contraseña. Para llevar a cabo este acceso, el administrador de la base de datos proporcionará al usuario un nombre y una clave que deberá utilizar para acceder a la misma. Sin embargo, existen otras opciones disponibles para poder acceder a la base de datos, tales como el uso de certificados o seguridad AWS IAM.

Tras asignar el usuario y la contraseña es posible seleccionar una serie de características que tendrán los diferentes usuarios de la base de datos. De esta forma es posible seleccionar aquellos usuarios que tendrán permiso para modificar cualquier aspecto de la base de datos, así como las diferentes colecciones y la gestión de los datos de estas. Al mismo tiempo existen aquellos usuarios que únicamente tienen acceso a leer la información que en la base de datos se almacena. Por último, es posible asignar una serie de privilegios a ciertos usuarios dependiendo de las necesidades del gestor.

Así mismo, es posible establecer un acceso temporal a la información, impidiendo el acceso de esos usuarios a la base de datos después de un tiempo entre 1 y 6 horas.

Otra de las medidas posibles es la restricción de direcciones IP a la base de datos, aunque debido al objetivo final de este proyecto es recomendable permitir el acceso a cualquier IP, restringiendo el acceso mediante los métodos anteriores.

3.2.3 DJANGO

A la hora de llevar a cabo una correcta representación de la información y crear un interfaz mediante el cual el usuario sea capaz de acceder a la información, se ha optado por la creación de una página web, ya que cumple con los criterios de la arquitectura cliente-servidor. Para ello la herramienta seleccionada es “Django” [39].

Este *framework* es de código abierto y gratuito, gracias a lo cual es posible realizar múltiples tareas de manera sencilla, llevando a cabo un diseño limpio y pragmático.

Entre las cualidades por las que destaca Django y por las que se ha escogido como principal soporte para el desarrollo de la web destaca, la rapidez con la que es posible desarrollar las diferentes tareas necesarias. Así como la sencillez a la hora de ejecutar las diferentes acciones. Debido a su modelo repetitivo, es posible aprovecharse de ciertas ventajas como la gran escalabilidad que ofrece, consiguiendo al mismo tiempo una gran flexibilidad.



Figura 3-3 Logo Django. Fuente: [40].

Una de las cualidades que convierten a Django en la herramienta ideal para la realización de la página web es que se trata de un patrón “Modelo-Vista-Controlador” (MVT). Es decir, integra en su funcionamiento la interacción con bases de datos. En este patrón, el modelo corresponde con la información almacenada en la base de datos, la vista es la interfaz que se le presenta al usuario en la página web y por último el controlador corresponde con la ejecución de las funciones.

Además, cuando se crea el proyecto, automáticamente se crea una base de datos denominada `db.sqlite3` tal y como aparece en la Figura 3-27. Esta es la encargada de almacenar todos los datos que introducirá el usuario para su registro, y sobre la que se analizará si los datos ya existen o no. Esta función la realiza almacenando los datos relacionados entre sí, en forma de columnas, de tal manera que el acceso sea rápido y sencillo, ofreciendo un servicio óptimo al usuario.

3.2.4 Bootstrap

Para poder llevar a cabo una correcta representación de los datos y que estos sean accesibles al usuario de manera correcta es necesaria la creación de un interfaz sencillo de entender. Para ello la herramienta “Bootstrap” ha sido la seleccionada [41]. Bootstrap permite la aplicación de estilos a archivos html, y proporciona tanto ejemplos de interfaces diseñadas en html como las diferentes funcionalidades que en estas aparecen. Es una herramienta de gran utilidad a la hora de diseñar una web.

La gran ventaja que ofrece esta herramienta es la facilidad de aprendizaje, es decir, al ser de código abierto y proporcionar ejemplos de cómo son las diferentes interfaces, es posible consultar en los foros de programación cuales son los principales errores y por lo tanto como solventarlos. Así como presentar modificaciones que en un futuro mejoren tanto el rendimiento como el estilo de la web.



Figura 3-4 Logo Bootstrap. Fuente: [43].

3.3 Metodología

Todos los códigos que se describen en el siguiente apartado se encuentran reflejados en el Anexo IV: Códigos, donde además de los mencionados se encontrarán también todos los necesarios para la réplica del siguiente proyecto. Estos códigos contienen credenciales personales y privadas por lo que han sido sustituidas por “*****”. A continuación, se procederá a describir aquellas partes del código fundamentales para el entendimiento del proyecto.

3.3.1 Base de datos

Uno de los pilares básicos en la elaboración del presente trabajo es la creación de una base de datos que permita almacenar toda la información necesaria, además de facilitar el acceso a esta en el futuro.

Con el objetivo de conseguir esto de la forma más eficiente posible, se ha implementado una base de datos en MongoDB. Una vez el programa ha sido instalado es posible crear una base de datos en la que posteriormente se añadan las diferentes colecciones para almacenamiento de los diferentes datos.

Es por este motivo que, para cada variable obtenida de MeteoGalicia se ha creado una colección individual para cada parámetro meteorológico, de tal forma que una vez sea necesario implementar una llamada a la información para la representación en el mallado el proceso sea más rápido y no sea necesario realizar una búsqueda en toda la base de datos. Al mismo tiempo que es posible organizar la información y en caso de que se quiera añadir alguna variable extra, esto sea posible sin mezclar datos diferentes.

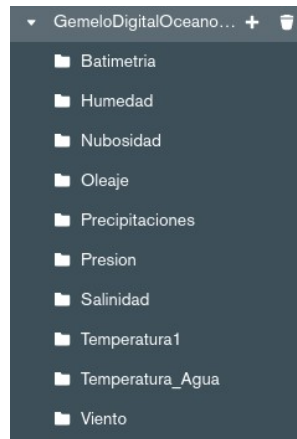


Figura 3-5 Estructura Data Base en MongoDB. Fuente: Propia.

Para poder facilitar el acceso a la información desde cualquier lugar, se procede a la creación de una base de datos en la nube. De esta manera es posible acceder a la información únicamente con los equipos propios del buque y por tanto permitiendo que las labores de descarga y procesamiento de datos se realice en tierra.

Para la realización del presente trabajo, y con el objetivo de la creación de una base de datos que sea accesible para todo aquel que lo desee, se han tenido que modificar los permisos en la base de datos, es decir, se ha habilitado la IP “0.0.0.0/0” como IP válida para el acceso a la misma. De esta manera cualquier dirección IP que posea la dirección de la base de datos (tanto principal como secundaria) podrá acceder a la base de datos.

Network Access

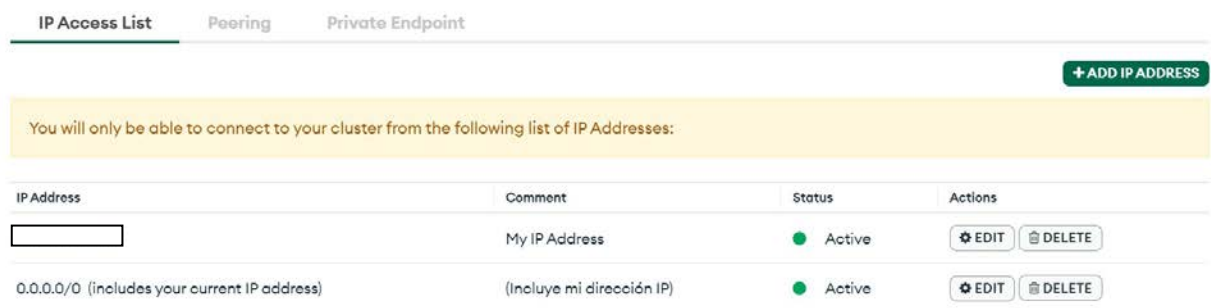


Figura 3-6 Ajustes acceso IP. Fuente: Propia.

Sin embargo, esta medida no implica que cualquiera pueda acceder a los datos y modificarlos, ya que la misma dirección IP tendría permiso para hacerlo. Como medida de seguridad se ha establecido que esta base de datos sea de carácter privada, lo que implica que para su acceso se requerirá un usuario y una contraseña que habrá que indicar en la base de datos.

3.3.2 Descarga de datos batimétricos

La base sobre la que se fundamenta el Gemelo Digital Oceanográfico es la de proporcionar una simulación en tiempo real del estado del entorno marino. Sin embargo, para que esto pueda tener utilidad a la hora de la navegación, es importante también poder proporcionar al usuario una serie de datos sobre aquellas zonas en la que será capaz de navegar.

Para poder suplir con esta información a la base de datos empleada en este gemelo digital, es necesario hacer es descargar los datos en la zona de interés de desarrollo del presente proyecto. Para ello en primer lugar es necesario acceder a la página de EMODnet, en concreto a la sección de batimetría ([EMODnet Map Viewer \(europa.eu\)](https://emodnet.eu/)), donde es posible mediante el menú desplegable que aparece en la parte derecha acceder a diferentes tipos de información dependiendo de las necesidades, tal y como aparece en la Figura 3-7. En lo que atañe al presente trabajo es necesario descargar la profundidad de la zona sin incluir todas las zonas de tierra, en las que el valor de la elevación será “NaN”, es decir, una variable a la que no se le asigna número.

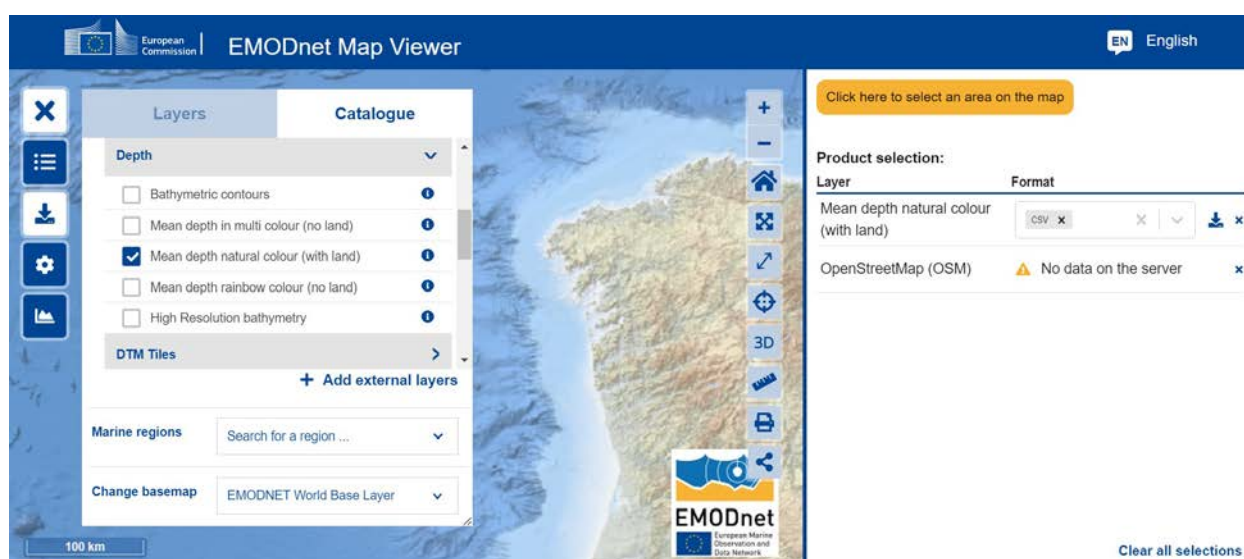


Figura 3-7 Acceso a la descarga de batimetría. Fuente: [42].

Para ello se selecciona la opción correspondiente en el menú emergente en la parte izquierda y posteriormente se procede a establecer en el mapa central un área de interés, de donde se descargarán los datos de batimetría. Para ello, una vez seleccionada el área, se le indicara a EMODnet cuál es el formato del documento que se va a descargar con esta información. de esta forma, en este caso se ha seleccionado un formato JSON, posibilitando una descarga automática a la base de datos empleada para ello. Gracias a que es posible hacer esto de forma seguida, en el mismo código se le puede indicar a Python que proceda a guardar la información en la colección de la base de datos correspondiente.

```
client = pymongo.MongoClient("mongodb+srv://EserName:Password@cluster0.9cawz8x.mongodb.net/GemeloDigitalOceanográfico")
db = client["GemeloDigitalOceanográfico"]
collection = db["Temperatura"]
collection.insert_one(data)
```

Figura 3-8Código para almacenar la batimetría en la base de datos. Fuente: Propia.

El código de la Figura 3-8, realiza una llamada a MongoDB, donde le indica que se está accediendo a un usuario en la red, por lo que a continuación habrá que indicar tanto el nombre de usuario como la contraseña de este. Este procedimiento añade seguridad a la hora del manejo de la base de datos ya que se limita el acceso a la misma. Y a continuación procede a indicarle, dentro del usuario en que base de datos está trabajando, ya que un mismo usuario puede tener diferentes bases de datos. En segundo lugar, se especifica la colección en la que se quiere verter toda la información descargada. Por último, el código inserta esta información en la base de datos.

3.3.3 Descarga de datos MeteoGalicia

La gran mayoría de los datos empleados en la realización del presente trabajo se obtendrán a través de una API REST del servicio que ofrece MeteoGalicia [43]. Una API REST no es más que una relación entre el proveedor de información y el cliente que la solicita. De esta forma se establecen al mismo tiempo una serie de requerimientos que se solicitan al cliente para poder devolver la información que el cliente desea, es decir, las API permiten una serie de datos proporcionados por el cliente para poder comunicar un sistema informático con un ordenador y conseguir algún tipo de información.

Es decir, mediante las API, los diferentes proveedores de información son capaces de llevar a cabo un intercambio de información con los usuarios asegurándose una serie de medidas de seguridad mediante la aplicación de ciertos protocolos y definiciones. Formando así un contrato entre proveedor y cliente, en el que se establece la información que solicita el cliente y a la que se puede acceder.

De esta forma la API_Rest&API_KEY empleada para descargar la información de MeteoGalicia es:

`"https://servizos.meteogalicia.gal/apiv4/getNumericForecastInfo?variables={Variables_deseadas}&coords={LONG},{LAT};&format=application/json&API_KEY=**CONFIDENCIAL**&lang=gl"`

Como se puede apreciar, en un primer momento el servidor requiere que se le indiquen unas coordenadas geográficas para poder llevar a cabo la entrega de la información. al mismo tiempo que es importante decirle al servidor cuales son las variables que deseamos obtener, así como el formato en el que se nos entregará dicha información. En el apartado final, se requiere una API_KEY, esta es de carácter personal e intransferible, por lo que para poder hacer uso de ella será necesaria su solicitud al servicio de administración de MeteoGalicia y rellenar un formulario.

Dentro de esta API el servidor de MeteoGalicia permite descargar la información en diferentes formatos, así como ofrece una serie de variables meteorológicas extensa. Al mismo tiempo y aunque no figure en esta API, es posible seleccionar el modelo de referencia de malla para la descarga de información, ya que la precisión de los datos puede variar según el modelo que se esté utilizando. Sin embargo, si no se especifica ningún modelo específico, el servidor resolverá la solicitud de información con el modelo WRF_Novo_1km, este es el modelo de referencia de Weather Research and Forecasting (WRF), modelo utilizado por defecto en MeteoGalicia y que está pensado que siga siendo así en los próximos años.

3.4 Implementación

3.4.1 Captura de datos de fuentes externas y almacenamiento en MongoDB

Una vez explicado el funcionamiento de la API implementada en la obtención de datos de MeteoGalicia, el siguiente paso es establecer una base para la descarga de datos. Para poder llevar a cabo este proceso de manera correcta, es importante indicar al modelo de simulación cuales son los puntos de los que tiene que obtener la diferente información relevante. De esta manera, se procede a la creación de una malla sobre las Rías Baixas.

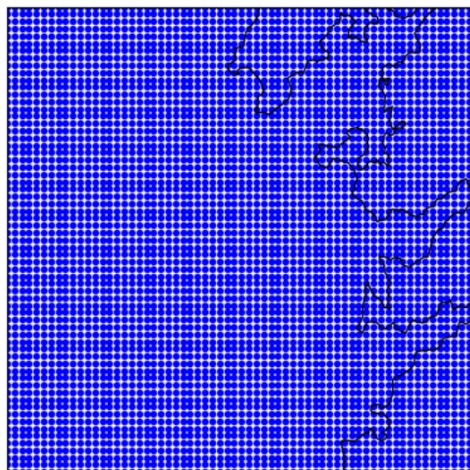


Figura 3-9 Ejemplo Mallado Rías Baixas GDO. Fuente: Propia.

De esta forma, el tamaño seleccionado para la malla es el que aparece, de forma aproximada en la Figura 3-9, estableciendo una serie de puntos que servirán para la descarga de los datos. Estos puntos establecen un mallado de referencia, es decir, en función de los datos obtenidos es posible variar la resolución de los hexágonos empleados y obtener así una representación u otra para la descarga de datos. Al mismo tiempo, al descargar estos los datos necesarios de la meteorología e insertarlos en la base de datos se procederá a realizar un filtrado, eliminando así los puntos que no se desean para el modelo del Gemelo Digital Oceanográfico, de esta manera, aquellos puntos establecidos en tierra serán eliminados.

Es importante destacar que, para llevar a cabo una correcta descarga de los datos, los puntos GPS de los cuales se desea obtener información meteorológica deberán estar almacenados en un archivo en formato específico en “CSV” de esta forma es posible descargar datos de puntos a criterio del usuario. La principal ventaja que este modelo de mallado puede presentar es la de realizar una descarga más genérica en las zonas de menor importancia y donde la información de los factores meteorológicos sea más uniforme. Al mismo tiempo es posible llevar a cabo una descarga en puntos más específicos en los que navegará el buque. De esta forma es posible crear una similitud a los conocidos mallados de control local. De esta forma, los hexágonos que contengan los puntos cercanos a costa tendrán un mayor número de datos y por lo tanto el valor final entregado al usuario será más preciso.

Para la descarga de los datos se llevará a cabo la implementación de un código que sea capaz de realizar todas las funciones necesarias y que al mismo tiempo permita mejorar el modelo empleado. Es decir, aumentar tanto el número de fuentes como el de los parámetros meteorológicos disponibles para descargar.

De esta forma, se seguirá una implementación basada en “herencias”, en el que el objetivo principal será el de crear una clase principal denominada base, de la cual emanaran diferentes clases conocidas como derivadas. El objetivo es que por cada fuente el modelo que mejore cualidades y capacidades que aporta a la hora de gestionar los datos y, llegado el momento, permita ampliar con más y más fuentes de información o incluso permita incluir nuevos parámetros. Este es un mecanismo utilizado en la programación orientado a objetos, que es capaz de solucionar la problemática de tener que reescribir código, ya que las clases derivadas simplemente tendrán que tomar las diferentes líneas de código, y adaptarlas a sus necesidades más específicas. Estas funciones derivadas, en definitiva, implementan todos los atributos de la clase base, además de incluir algunos específicos de cada función.

Además, otra de las cualidades de la orientación a objetos de Python es que permite realizar herencias múltiples, es decir, que sea posible que una clase derivada pueda heredar métodos de varias clases padre. Por lo tanto, lo primero es crear una clase base, denominada “MeteoGalicia”, junto con las funciones que serán comunes a todas las derivadas. En esta clase no se realizará la implementación, pero sí en sus clases derivadas en las que se especificarán los diferentes parámetros meteorológicos disponibles para la obtención de datos, como se puede observar en la Figura 3-10. De esta forma, se podrán definir tantas clases derivadas como fuentes, incluyendo en estas los diferentes datos meteorológicos disponibles.

```
def Temperatura(self):  
    with open('BatiRiasBaixas.csv', 'r') as file:  
        reader = csv.reader(file)  
        next(reader)  
        next(reader)  
        for row in reader:  
            LAT, LONG, elevacion = row
```

Figura 3-10 Ejemplo definición diferentes clases derivadas. Fuente: Propia.

Finalmente, para ejecutar las funciones previamente creadas de los diferentes factores meteorológicos, solamente hay que indicar la clase base a la que pertenecen y dentro de esta clase cual es la función que deseamos ejecutar. De esta forma, al ejecutar el comando que aparece en la Figura 3-11, se hace una llamada a la clase MeteoGalicia, la cual tendrá creadas diferentes funciones para obtener los datos deseados, concretamente en este caso se hace llamada a la función Temperatura. Una vez se ejecute esta función, Python pasa a realizar las acciones que se reflejan en la Figura 3-12, de tal manera que no es necesario reescribir toda la función.

```
import csv  
import json  
import requests  
from MeteoGalicia import MeteoGalicia  
from MeteoClass import MeteoClass  
  
MeteoG = MeteoGalicia()  
MeteoG.Temperatura()
```

Figura 3-11 Llamada a la función "Temperatura". Fuente: Propia.

El objetivo principal perseguido con este tipo de programación orientada a objetos es poder disponer de un script principal, el cual no este excesivamente cargado y sea fácil de entender por el usuario. De tal manera que cuando se quiera llamar a una serie de funciones predefinidas solamente haga falta indicar, la fuente de la que se quieren obtener esos datos.

Además, otro de los objetivos es crear diferentes clases según vaya siendo posible añadir nuevas fuentes de datos, de esta forma, no será necesario modificar el código principal, únicamente será realizar la implementación de las funciones heredadas y posteriormente llamarlas dentro del script en el que nos encontramos trabajando.

Para descargar la información de los parámetros necesarios es necesaria la creación de un fichero “CSV” en el que se le indiquen con anterioridad las coordenadas geográficas de los puntos donde se requiere conocer dicha información. De esta manera, es posible descargar un fichero independiente por cada punto y, al mismo tiempo es posible almacenarlos en la base de datos de manera independiente.

El primer paso para la creación del mallado deseado será la descarga de los datos de batimetría de la página de EMODnet, ya que los puntos que se utilicen para la descarga de batimetría (por la gran cantidad de puntos que esta página proporciona), serán los mismos puntos que se utilizarán para la descarga de los demás datos meteorológicos.

De esta forma y tal como muestra la Figura 3-12, para la obtención de los datos de diferentes puntos contenidos en un fichero CSV, el primer paso requerido es abrir el fichero que contiene las coordenadas y leerlo. Para ello se le indica a Python que abra el documento de nombre “BatiRiasBaixas.csv” en modo lectura como un fichero, y a continuación que lo lea fila por fila. Es importante resaltar que para que el código implementado no genere un error, hay que saltar las primeras dos filas en la que aparecen los títulos de las columnas, para ello se emplea el comando `next()`. Como se puede observar en la Figura 3-12 es necesaria la implementación de un bucle `for`, permitiendo que una vez leída la primera coordenada y el fichero generado haya sido almacenado en la base de datos, se pueda pasar al siguiente punto GPS del fichero y repetir el código las veces que sea necesario hasta que no queden coordenadas. Así mismo se hace uso de la función `if not` de tal manera que en caso de que no se cumpla la condición formulada a continuación, se saltará esa línea y se procederá a la siguiente, tal y como aparece en la Figura 3-13.

```
def Temperatura(self):
    with open('BatiRiasBaixas.csv', 'r') as file:
        reader = csv.reader(file)
        next(reader)
        next(reader)
        for row in reader:
            LAT, LONG, elevation = row
            if not math.isnan(float(elevation)):
                solditems = requests.get(f'APIREST&API_KEY')
                data = solditems.json()
                with open(f'Temperatura_{LONG}_{LAT}.json', 'w') as f:
                    json.dump(data, f)
                client = pymongo.MongoClient("mongodb+srv://UserName:Password5@cluster0.9cawz8x.mongodb.net/GemeloDigitalOceanográfico")
                db = client["GemeloDigitalOceanográfico"]
                collection = db["Temperatura"]
                collection.insert_one(data)
```

Figura 3-12 Código para obtención de datos. Fuente: Propia.

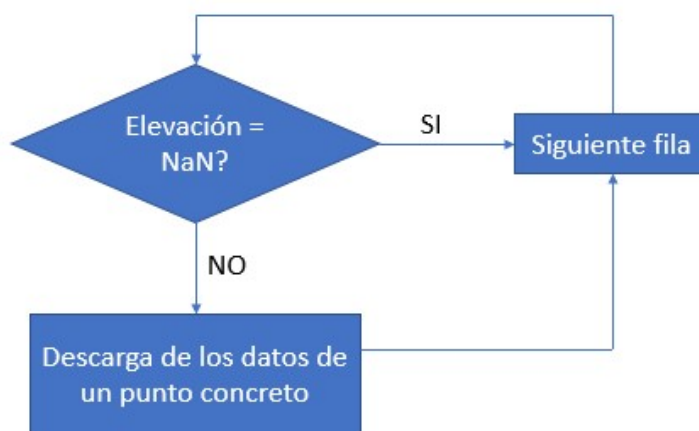


Figura 3-13 Diagrama de flujo para el filtrado de puntos por elevación. Fuente: Propia.

A la hora de almacenar la información en la base de datos, la estructura necesaria es idéntica a la utilizada para el almacenaje de los datos de batimetría. Sin embargo, es importante cambiar la colección en la que se van a almacenar los diferentes archivos, de tal forma que, en el momento de hacer la llamada a la base de datos, no exista confusión y de lugar a errores en la representación o incluso en la información entregada al usuario.

Una vez ejecutado el código de obtención de datos, estos se descargarán uno a uno y al mismo tiempo, antes de pasar al siguiente punto se procederá a almacenar la información contenida en el documento en formato JSON en la base de datos. De esta forma se consigue automatizar tanto la descarga como el almacenamiento de los datos. Al finalizar la ejecución de este código, se obtendrá el resultado tal y como aparece en la Figura 3-14 y al comprobar la base de datos obtendremos el resultado que aparece en la Figura 3-15.

```
pablo@pablo-VirtualBox:~/Escritorio/TFG Valera$ /bin/python3 "/home/pablo/Escritorio/TFG Valera/Ejecucion.p
y"
MeteoGalicia
```

Figura 3-14 Texto indicando final de la descarga. Fuente: Propia.

#	Temperatura	id	objectid	type String	crs object	features Array
1		63e2ad55ce7ab4f6e9f27b9		"FeatureCollection"	(2 fields)	[1 elements]
2		63e2ad55ce7ab4f6e9f27b6		"FeatureCollection"	(2 fields)	[1 elements]
3		63e2ad55ce7ab4f6e9f27b5		"FeatureCollection"	(2 fields)	[1 elements]
4		63e2ad55ce7ab4f6e9f27b4		"FeatureCollection"	(2 fields)	[1 elements]
5		63e2ad55ce7ab4f6e9f27b1		"FeatureCollection"	(2 fields)	[1 elements]
6		63e2ad55ce7ab4f6e9f27c3		"FeatureCollection"	(2 fields)	[1 elements]
7		63e2ad55ce7ab4f6e9f27c5		"FeatureCollection"	(2 fields)	[1 elements]
8		63e2ad55ce7ab4f6e9f27c7		"FeatureCollection"	(2 fields)	[1 elements]
9		63e2ad55ce7ab4f6e9f27c9		"FeatureCollection"	(2 fields)	[1 elements]
10		63e2ad55ce7ab4f6e9f27cb		"FeatureCollection"	(2 fields)	[1 elements]
11		63e2ad55ce7ab4f6e9f27cd		"FeatureCollection"	(2 fields)	[1 elements]
12		63e2ad55ce7ab4f6e9f27cf		"FeatureCollection"	(2 fields)	[1 elements]
13		63e2ad55ce7ab4f6e9f27d1		"FeatureCollection"	(2 fields)	[1 elements]
14		63e2ad55ce7ab4f6e9f27d3		"FeatureCollection"	(2 fields)	[1 elements]
15		63e2ad55ce7ab4f6e9f27d5		"FeatureCollection"	(2 fields)	[1 elements]
16		63e2ad55ce7ab4f6e9f27d7		"FeatureCollection"	(2 fields)	[1 elements]
17		63e2ad55ce7ab4f6e9f27d9		"FeatureCollection"	(2 fields)	[1 elements]

Figura 3-15 Base de datos con los documentos almacenados. Fuente: Propia.

La manera de mantener esta base de datos actualizada es mediante la programación de tareas mediante una herramienta llamada *crontab*, es decir, indicar en un fichero de texto cada cuanto tiempo debe ejecutar de manera automática la descarga de datos configurado en el *cron* del ordenador. De esta manera es posible tener una base de datos actualizada ejecutando estos códigos en segundo plano.

3.4.2 Extracción de datos y representación en mapa según el mallado H3

En el siguiente apartado se describe el código encargado de la consulta a los datos y la representación de estos en un mapa.

En primer lugar, se procede a conectar con la base de datos en la nube proporcionando de nuevo el mismo nombre de usuario y clave asignadas. De esta manera será posible realizar una búsqueda a través de los diferentes ficheros de la colección indicada en cada momento.

```
# Conectarse a la base de datos
client = pymongo.MongoClient("mongodb+srv://*****:****@cluster0.9cawz8x.mongodb.net/GemeloDigitalOceanográfico")
db = client["GemeloDigitalOceanográfico"]
collection = db["Temperatura"]
```

Figura 3-16 Inicio GDO. Fuente: Propia.

El siguiente paso una vez establecida la conexión con la colección indicada en la base de datos es el de analizar la información y extraer de esta únicamente los datos que son de utilidad para el usuario. De esta forma es necesaria la creación de una lista vacía en la que posteriormente se irán escribiendo los datos necesarios para la creación del mapa final como son las coordenadas GPS y el valor del fenómeno meteorológico deseado. Una vez esta lista ha sido creada se implemente un bucle `for` repetidas veces para poder iterar los diferentes apartados del archivo JSON. La peculiaridad del archivo empleado en la descarga de estos datos es que están organizados por días. Es decir, la previsión de los diferentes días está organizada por las conocidas como “clave-valor”. Por lo que es importante en primer lugar, establecer que día será necesario iterar para extraer la información.

Con el objetivo de conseguir esto, en el apartado *day* del mensaje se realiza una comprobación con el uso de la función `if`, de esta forma únicamente se iterará el día. Como el mensaje puede contener información de diferentes parámetros meteorológicos es necesario realizar al mismo tiempo, implementando otra vez la función `if` una comprobación del parámetro que se desea iterar, buscando la variable por el nombre asignado en el mensaje de MeteoGalicia con el método `variable['name']`, de tal forma que no se produzcan errores que alteren el resultado final. Es importante destacar que, con el fin de evitar estos errores, dentro de la función `soldditems` en la Figura 3-12, se le indicara mediante la APIREST a MeteoGalicia que descargue únicamente una variable por archivo.

A continuación, si los criterios anteriores han sido cumplidos, se procede a buscar los valores de las coordenadas y asignarles el nombre correspondiente. Es importante resaltar que para el almacenamiento correcto de esta variable hay que indicar en qué posición se encuentran. Es decir, mediante el empleo de “[0]” y “[1]” se le indica que en primer lugar se encuentra la longitud y en segundo lugar, la latitud. Esto es importante para la representación posterior de los datos en el mapa, así como una correcta consulta en la base de datos.

Con el objetivo de evitar archivos de gran tamaño que dificulten la comunicación entre el usuario embarcado y los servidores en tierra, así como permitir acotar la zona en la que se desea conocer el fenómeno meteorológico, una vez asignadas las coordenadas a las respectivas variables se procede a realizar una comprobación de distancia. Es decir, mediante el rango de acción proporcionado por el usuario, se comprueba que puntos están dentro de ese rango, de tal manera que solamente se almacenaran en la lista `data` aquellos valores de interés. De esta forma se añade un filtro que consigue reducir el tamaño del fichero generado, además de permitir al usuario acceder únicamente a la información de su área correspondiente.

Para realizar este filtrado de la información se hace uso de la fórmula de Haversine, este método permite calcular la distancia entre dos puntos por círculo máximo, según su latitud y longitud, donde φ representa la latitud y λ la longitud:

$$d = 2 * R * \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos\varphi_1 * \cos\varphi_2 * \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \quad (1)$$

El resultado obtenido tras ejecutar esta fórmula es la distancia más corta entre dos puntos, de tal manera que mediante su implementación sea posible seleccionar únicamente los puntos cuya distancia con el punto donde se encuentre el usuario sea menor que la especificada por el mismo. Es decir, la información queda acotada aun día en concreto y a un radio determinado.

```
time_to_search = moment_indicado
data = []
features = collection.find({"features.properties.days.variables.values.timeInstant": moment_indicado})
for feature in features:
    for day in feature['features'][0]['properties']['days']:
        if day['timePeriod']['begin']['timeInstant'] <= time_to_search <= day['timePeriod']['end']['timeInstant']:
            for variable in day['variables']:
                if variable['name'] == 'temperature':
                    longitude = float(feature['features'][0]['geometry']['coordinates'][0])
                    latitude = float(feature['features'][0]['geometry']['coordinates'][1])
                    distance = calculate_distance(float(longitude), float(longitude), float(Lati), float(Longi))
                    if float(distance) <= float(radio_accion):
                        for value in variable['values']:
                            if value['timeInstant'] == time_to_search:
                                temperature = int(value['value'])
                                data.append({'longitude': longitude, 'latitude': latitude, 'temperature': temperature})
```

Figura 3-17 Búsqueda de datos en MongoDB. Fuente: Propia.

A continuación, se busca el valor de la variable deseada, para lo cual será necesario comprobar mediante el momento indicado anteriormente, cual es la hora de la que se desea extraer la información. para ello se implementa un bucle `for`, el cual permite recorrer dentro de la variable `values` los diferentes días en busca del valor deseado.

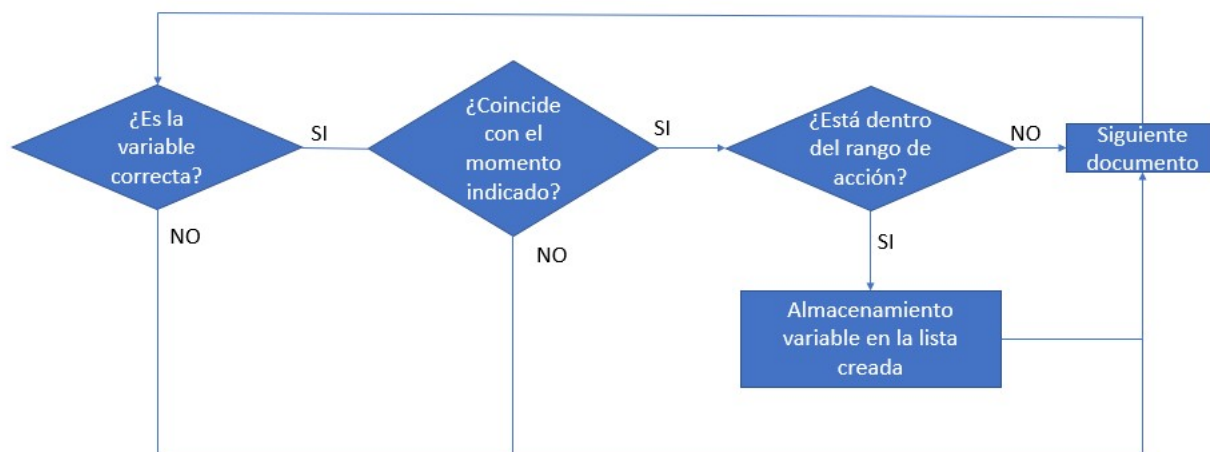


Figura 3-18 Diagrama de flujo filtrar datos mensaje. Fuente: Propia.

Finalmente, se almacena junto con los valores de las coordenadas asociadas a este valor escribiendo los datos en la lista `data` previamente creada. En su inicio esta lista se creó vacía, pero a medida que el primer bucle `for` se ejecuta en los diferentes archivos se va creando una lista final que contiene los diferentes puntos geográficos con el valor de la magnitud buscada, de tal manera que sea posible utilizarla posteriormente para la representación en un mapa de esta información empleando el modelo de mallado H3 de Uber.

En función de los datos que se quieran representar y del formato en el que el archivo JSON con la información pertinente es descargado, la estructura del código anterior puede variar. De esta forma para la consulta del viento únicamente es necesario iterar una vez dentro de la variable `variables` del archivo JSON ya que tanto el valor de dirección como el de intensidad vienen en el mismo apartado.

Sin embargo, en el caso del oleaje, debido a que la solicitud que se hace a MeteoGalicia es de tres variables independientes, estas vendrán en el mismo mensaje, pero en diferentes apartados, por lo que será necesario iterar tantas veces como variables tenga el mensaje, además de crear listas en las que almacenar esta información para después poder acceder de forma independiente a cada una evitando así errores en el manejo de la información.

```
data1 = []
data2 = []
data3 = []
features = collection.find({"features.properties.days.variables.values.timeInstant": moment_indicado})
for feature in features:
    for day in feature['features'][0]['properties']['days']:
        if day['timePeriod']['begin']['timeInstant'] <= time_to_search <= day['timePeriod']['end']['timeInstant']:
            for variable in day['variables']:
                if variable['name'] == 'significant wave height':
                    longitude = float(feature['features'][0]['geometry']['coordinates'][0])
                    latitude = float(feature['features'][0]['geometry']['coordinates'][1])
                    distance = calculate_distance(float(latitude), float(longitude), float(Lati), float(Longi))
                    if float(distance) <= float(radio_accion): # x es el radio deseado en kilómetros
                        for value in variable['values']:
                            if value['timeInstant'] == time_to_search:
                                significant_wave_height = int(value['value'])
                                data1.append({'longitude': longitude, 'latitude': latitude, 'significant_wave_height': significant_wave_height})

for variable in day['variables']:
    if variable['name'] == 'mean wave direction':
        longitude = float(feature['features'][0]['geometry']['coordinates'][0])
        latitude = float(feature['features'][0]['geometry']['coordinates'][1])
        distance = calculate_distance(float(latitude), float(longitude), float(Lati), float(Longi))
        if float(distance) <= float(radio_accion): # x es el radio deseado en kilómetros
            for value in variable['values']:
                if value['timeInstant'] == time_to_search:
                    mean_wave_direction = int(value['value'])
                    data2.append({'longitude': longitude, 'latitude': latitude, 'mean_wave_direction': mean_wave_direction})

for variable in day['variables']:
    if variable['name'] == 'relative peak period':
        longitude = float(feature['features'][0]['geometry']['coordinates'][0])
        latitude = float(feature['features'][0]['geometry']['coordinates'][1])
        distance = calculate_distance(float(latitude), float(longitude), float(Lati), float(Longi))
        if float(distance) <= float(radio_accion): # x es el radio deseado en kilómetros
            for value in variable['values']:
                if value['timeInstant'] == time_to_search:
                    relative_peak_period = int(value['value'])
                    data3.append({'longitude': longitude, 'latitude': latitude, 'relative_peak_period': relative_peak_period})
```

Figura 3-19 Iteración parámetros oleaje. Fuente: Propia.

```
# Obtener los datos de MongoDB
data = list(collection.find( {}, { "latitude": 1, "longitude": 1, "elevation": 1}))
```

Figura 3-20 Extracción datos batimetría. Fuente: Propia.

En el caso de la batimetría el proceso de búsqueda se simplifica debido al formato del archivo JSON del que se extrae la información, debido a esto el uso de la función `find()` se puede aplicar directamente sobre el archivo sin la necesidad de establecer filtros previos o indicar variable meteorológica o indicar una fecha exacta.

```
# Crear un mallado h3 con la resolución indicada
resolution = float(resolucion)
hexagons = []
for point in data:
    lat = point['latitude']
    lng = point['longitude']
    hex_index = h3.geo_to_h3(lat, lng, resolution)
    hexagons.append(hex_index)
```

Figura 3-21 Creación hexágonos. Fuente: Propia.

Una vez se ha extraído la información de la base de datos según los criterios del usuario, y esta ha sido almacenada en la lista `data` es necesario realizar una conversión de coordenadas. Esto es asignar, según las coordenadas geográficas de cada punto, un índice de hexágono a cada coordenada. De esta manera es posible georreferenciar todos los puntos de los cuales se ha obtenido la información siguiendo un modelo de mallado de hexágonos regulares.

Para proceder de esta manera en primer lugar se indica la resolución de los hexágonos, de tal manera que cuando se represente la información en el mapa esta se ajuste a los requisitos del usuario. En segundo lugar, de la misma forma que para los datos de cada punto, se crea una lista denominada hexagons, cuya finalidad será almacenar los diferentes índices de cada hexágono.

A continuación, se itera en la lista previamente creada con los datos meteorológicos y, mediante los valores de latitud, longitud y resolución se asignan los diferentes índices de hexágono. Finalmente, todos estos datos se vuelcan en la lista creada para el almacenamiento de estos.

En resumen, en esta parte se está creando el mallado que posteriormente será empleado, es decir, se le están asignando los puntos a los hexágonos, de tal forma que posteriormente se trabaje sobre esta base.

```
# Asignar cada punto de los datos a un hexágono h3
hexagons = list(set(hexagons))
hex_data = {}
for hex_index in hexagons:
    hex_data[hex_index] = []

for point in data:
    lat = point['latitude']
    lng = point['longitude']
    hex_index = h3.geo_to_h3(lat, lng, resolution)
    value = point['temperature']
    hex_data[hex_index].append(value)
```

Figura 3-22 Asignación puntos a la malla. Fuente: Propia.

Uno de los principales problemas que pueden surgir en este proceso de almacenamiento de datos es que la precisión con la que se obtienen los datos de EMODnet no coincida con la precisión de los datos que puede proporcionar MeteoGalicia. Esto supone que se hagan aproximaciones a la hora de extraer información meteorológica de diferentes puntos, de esta forma es posible que exista el mismo valor repetido para el mismo punto geográfico, lo que alteraría el resultado final de la previsión y por lo tanto el modelo final no tendría la precisión necesaria.

Es por esto por lo que es necesario establecer un sistema que permita comprobar, y en caso de que se produzcan, elimine los datos que se encuentran duplicados. Para ello se ha utilizado la lista denominada hexagons, la cual se establece una serie de valores únicos que impide que estos estén duplicados dentro de un mismo hexágono.

A continuación, al igual que en la creación del mallado asociando los diferentes puntos, se establece una lista cuya finalidad es la de almacenar los datos asignados a cada hexágono, es decir la temperatura asociada a cada punto geográfico. En primer lugar, mediante el primer bucle for se crea una lista vacía en la que se asocian los diferentes índices creados a los datos de temperatura de cada punto. Estos datos se extraen mediante la implementación del segundo bucle for el cual, en primer lugar, establece cual es el índice de cada punto que está analizando dentro del hexágono y posteriormente le asocia el valor de la magnitud meteorológica deseada. Finalmente, esta asociación se almacena en la lista creada en el primer bucle for.

Una vez los valores han sido asignados a cada hexágono y se ha establecido el mallado sobre la zona geográfica correspondiente es necesaria la creación de un sistema que sea capaz de representar los datos de forma correcta y entendible por el usuario.

```
# Calcular la media de los datos en cada hexágono
for hex_index in hexagons:
    values = hex_data[hex_index]
    mean_value = sum(values) / len(values)
    hex_data[hex_index + "_mean"] = mean_value

min_value = np.min([hex_data[hex_index + "_mean"] for hex_index in hexagons])
max_value = np.max([hex_data[hex_index + "_mean"] for hex_index in hexagons])
value_range = max_value - min_value
```

Figura 3-23 Cálculo de la media. Fuente: Propia.

El primer paso en este proceso es el de establecer un criterio para analizar los datos. Es por esto por lo que es necesario establecer un criterio correcto para la representación de los datos en pantalla. Para ello en primer lugar se calcularán las medias de todos los valores asociados a un mismo hexágono, es decir, se calculará el valor medio de todos los datos que estén asignados a cada hexágono. Esto es importante para la representación ya que, dependiendo de la resolución indicada por el usuario, el valor final mostrado puede variar.

Para realizar este cálculo en primer lugar es necesario iterar en la lista creada previamente y extraer los valores deseados, los cuales se asignan a la variable `values` para posteriormente calcular su media.

A continuación, se realiza una búsqueda de cuáles son los valores máximos y mínimos existentes en las medias anteriormente calculadas, así como calcular la diferencia entre estos valores. Este cálculo final posibilita la asignación de colores en el posterior mapa de calor, de forma que sea más fácil realizar un análisis de los datos

En los casos en los que haya que representar varias variables dentro de los diferentes factores meteorológicos, estos pasos deberán repetirse tantas veces como sea necesario. Sin embargo, es necesario escoger la cualidad que se quiere asociar al mapa de calor, de tal forma que sea solamente una la que sea representada.

```
color_hex_data = {}
cmap = matplotlib.cm.get_cmap("RdYlBu_r")
for hex_index in hexagons:
    value = hex_data[hex_index + "_mean"]
    color_value = (value - min_value) / value_range
    color = cmap(color_value)
    color = matplotlib.colors.to_rgba(color)
    hex_index = hex_index.replace("_mean", "")
    color_hex_data[hex_index] = color

m = folium.Map(location=[Lati, Longi], tiles='cartodbpositron', zoom_start=13)
```

Figura 3-24 Asignación colores. Fuente: Propia.

El siguiente paso por tanto es la asignación de los colores a cada hexágono para su posterior representación. Para lo cual, se crea una lista vacía en la cual posteriormente se almacenará la información que relaciona cada hexágono con su color asociado.

Para poder asignar los diferentes colores es importante importar una biblioteca que cumpla con los requisitos que necesitamos para su representación. De esta forma, mediante la función `cmap` es posible descargar diferentes bibliotecas de colores. En el caso de la temperatura, la biblioteca `RdYlBu_r`, la cual es una escala de verdes, permite representar aquellos hexágonos con valores medios de los diferentes factores más elevados de color rojo y aquellos valores menores de color azul, de manera que los valores medios tendrán un color intermedio.

El siguiente paso consisten en iterar mediante un bucle `for` toda la lista de hexágonos junto con los valores de temperatura medios que estos tienen asignados extrayendo el valor necesario. A continuación, tiene lugar una normalización, en la que se obtiene un valor entre 0 y 1 el cual será usado posteriormente en la asignación de colores a cada hexágono. Para esto se hace uso de la función `cmap` de nuevo en la que se busca en este caso el valor que corresponde. A continuación, el color asociado a cada hexágono se asocia a una tupla RGBA, que no es más que un conjunto de caracteres, los cuales se encargan de determinar la cantidad de color que estará presente en cada hexágono.

Así mismo el último de estos 4 caracteres es el encargado de facilitar la opacidad del hexágono, este valor oscila entre 0 y 1 siendo este último completamente opaco. Finalmente, se agrega el color al índice hexagonal para poder llevar a cabo una correcta representación de los datos en el mapa de color.

Finalmente, haciendo uso de la biblioteca `folium` se crea un archivo `html` en el cual se representará toda la información obtenida anteriormente. Para ello es importante indicar en el código las coordenadas en las que se encuentra el buque, de tal manera que el mapa se centre en esta posición. Así como un zoom inicial y el tipo de mapa que se desea. Esto se hace ya que existen diferentes representaciones que se adaptan en mejor o peor medida a los datos que en el mapa se representan.

```
for hex_index in hexagons:
    hex_vertices = h3.h3_to_geo_boundary(hex_index)
    lat = [vertex[0] for vertex in hex_vertices]
    lng = [vertex[1] for vertex in hex_vertices]
    color = color_hex_data[hex_index]
    mean_value = hex_data[hex_index + "_mean"]
    fill_color = matplotlib.colors.rgb2hex(color[:3])
    folium.Polygon(locations=[(lat_i, lng_i) for lat_i, lng_i in zip(lat, lng)],
                    color='black',
                    weight = 1,
                    fill=True,
                    fill_color=fill_color,
                    opacity=0.9,
                    popup="Temperatura: {}".format(round(mean_value, 2))).add_to(m)
```

Figura 3-25 Asignación polígonos. Fuente: Propia.

En último lugar, se procede a la representación de los datos en el mapa anteriormente creado. Para esto se hace uso de un bucle `for` el cual recorre todos los datos almacenados en la lista `hexagons`. Una vez se inicia este proceso lo primero que se ejecuta es un cálculo de los vértices de cada hexágono y se le asigna un color en función del valor medio que se asocia a este hexágono.

Es importante remarcar que para conseguir una mejor representación final del mapa de calor, se crea también un borde negro alrededor de los diferentes hexágonos. De esta manera, en aquellos lugares donde la variable representada no presente cambios, será posible apreciar las diferentes celdas hexagonales.

De la misma forma se extrae del índice del hexágono el valor medio del factor meteorológico y se añade en forma de `popup`, es decir, que en el momento que se seleccione el hexágono del que se desea conocer la información aparece una etiqueta proporcionando el valor medio calculado aproximado a dos decimales.

```
tooltip = unidad
my_icon = folium.features.CustomIcon('templates/icono.png', icon_size = (50,50))
folium.Marker([Lati, Longi], icon = my_icon, tooltip=tooltip).add_to(m)
minimap = MiniMap(toggle_display=True, position="topleft", tile_layer="stamenterrain")

m.add_child(minimap)
m.save("iframe.html")
```

Figura 3-26 Finalización código y guardado. Fuente: Propia.

Antes de finalizar la ejecución es posible realizar una serie de añadidos sobre el mapa final generado. Lo que se realiza es la creación de un marcador, es decir, un icono que aparecerá en la posición indicada al principio del código, donde se representará mediante un icono aquella variable que está midiendo el mapa. En esta situación aparecerá el nombre de la unidad que se asigne al principio del código debido a que el elemento `tooltip` se asigna a la variable con el nombre asignado por el usuario. Lo siguiente que se indica son las características principales del marcador, es decir, su color y el icono que se representará, el cual puede variar, cambiando únicamente el parámetro que se encuentra en la parte `icon`, estos se pueden sacar de la biblioteca abierta que posee `folium`, por lo que se puede cambiar según las posibilidades. A continuación se añadirá el marcador mediante la función `folium.Marker`, indicando la posición en la que se encontrará.

Finalmente, se ha añadido un mini mapa indicando únicamente la ubicación y el tipo de mapa deseado. Al finalizar este código el fichero creado por la biblioteca `folium` se almacena en un fichero `html` y podrá ser ejecutado para comprobar los resultados finales.

3.4.3 Interfaz: la página web

Una vez la base de datos ha sido creada y los datos han sido cargados en la misma es posible acceder a ellos mediante una representación un mapa, de acuerdo con factores especificados por los usuarios. Con el objetivo de poder llevar a cabo esto de manera rápida y que la información a transmitir entre el buque y los servidores sea mínima se implementará una página web donde, mediante una interfaz se pueda acceder a esta información.

Para llevar a cabo la creación de la página web, el primer paso es la instalación de Django, para poder trabajar utilizando las herramientas que este *framework* proporciona. A continuación, se lanzará el programa, creando como administrador un proyecto denominado en este caso `mysite`. Este proyecto constituye la base sobre la cual se va a construir la página web para la utilización del gemelo digital.

Una vez creado el proyecto, se creará automáticamente un fichero dentro del propio usuario del ordenador. En este fichero se encuentran diferentes archivos del formato Python, así como una pequeña base de datos en la que almacenará la información necesaria.

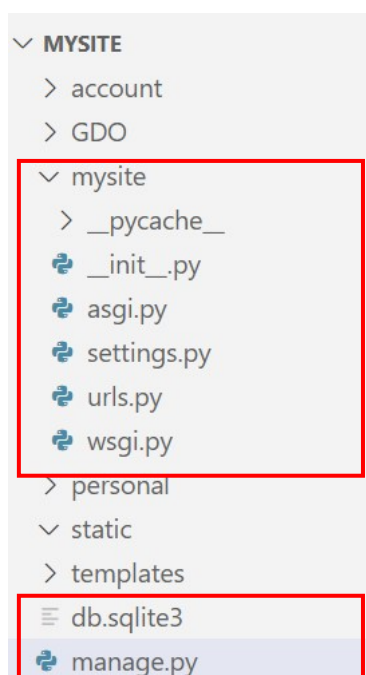


Figura 3-27 Carpetas creadas automáticamente en el proyecto. Fuente: Propia.

Estas carpetas son las necesarias para poner en funcionamiento la página web, ya que en ellas habrá que reflejar todas las rutas de acceso a los diferentes archivos html que se quieran representar en la página web, como indicar aquellas modificaciones necesarias en los ajustes del proyecto para poder otorgar ciertos permisos necesarios.

Una vez se ha creado correctamente el proyecto, es necesario tener en cuenta cual es el propósito de este trabajo para poder diseñar las diferentes vistas necesarias para crear las diversas funcionalidades del sistema dentro del proyecto anteriormente creado. Para que el proceso de diseño de la interfaz sea menos complejo, se ha utilizado la herramienta “*Bootstrap*”, la cual, como se comentó en la sección Bootstrap proporciona diferentes ejemplos de diseño con el objetivo de ayudar a la composición estética de la página web.

Es importante resaltar que dentro del proyecto creado, se albergarán diferentes carpetas, estas son las encargadas de almacenar todos los ficheros que gestionarán, de forma individual, las diferentes funcionalidades de la página. Para ello, dentro de ellas aparecerán una serie de archivos, de los cuales, los únicos que interesa modificar son los de `views.py`, `forms.py` y `models`, atendiendo al modelo MVC de Django. Así mismo será necesario crear una carpeta auxiliar denominada `templates` en la que se guardarán los archivos html que recogen el formato que se representará en pantalla, es decir, son los que guardan la información del formato y la estructura de la página web para ciertas funcionalidades.

Sin embargo, previamente a la creación de estas vistas, es necesario crear la plantilla que será común a todas las interfaces y que, se mantendrá invariable en las diferentes pestañas accesibles por el usuario. De esta forma, en la carpeta del proyecto denominada `templates` se crea un archivo html denominado `base.html`. Este fichero recoge la estructura general de la página web, es decir, se trata de un documento html, con un encabezado (`head`) y un pie de página (`footer`). El objetivo de separar estos espacios es el de generar un archivo final lo más dividido posible sencillo de editar, tal y como aparece en la Figura 3-28. Al mismo tiempo se consigue una separación de errores, es decir, que un error de diseño producido en cualquiera de estos espacios no afecte a los espacios contiguos.

```
<!-- Required meta tags -->
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
<hr>
{% include 'snippets/head.html' %}
</html>
<body>
  <div class="background">

    <!--Cuerpo principal de la pagina-->

    {% block content %}

    {% endblock content %}
    <!--Fin Cuerpo principal de la pagina-->

  </div>
  {% include 'snippets/footer.html' %}
</body>

</html>
```

Figura 3-28 Código html base de la página web. Fuente: Propia.

Una vez creada una plantilla general se procede a la creación de diferentes vistas permitirán al usuario interactuar con los diferentes modelos digitales. Debido al uso final que se pretende que tenga este proyecto, en el que el uso de estos modelos estará limitado al ámbito del adiestramiento de las unidades militares, el primer paso es crear un sistema que permita a los buques de estas características crear su propio usuario, de manera que esta plataforma cuente con más seguridad y al mismo tiempo permita a los encargados de este adiestramiento llevar un control de todos los alumnos que se registren.

3.4.3.1 Gestión de usuarios

El primer paso es implementar todo lo necesario para realizar la gestión de usuarios (aquellos que tendrán acceso al sistema). Para ello, es necesario crear una carpeta denominada `account`, que es la que se va a encargar únicamente de permitir a los buques militares acceder a los diferentes modelos en caso de que tengan una cuenta asignada, modificar tanto el correo como la contraseña en caso de necesidad, cerrar la sesión en curso o crear una cuenta de los diferentes modelos.

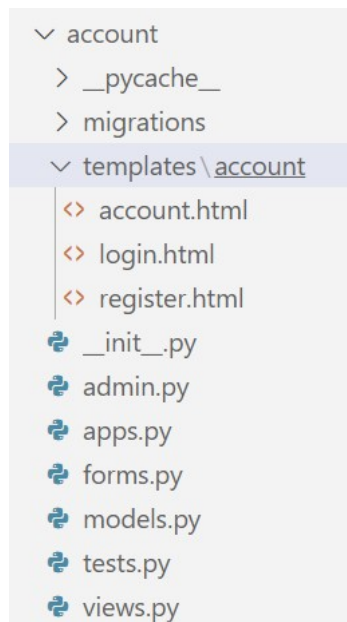


Figura 3-29 Aplicación "account": Fuente: Propia.

Atendiendo al modelo MVC que sigue Django, se crea otra carpeta auxiliar denominada `templates` en la que se guardarán los archivos `html` que recogen el formato que se representará en pantalla, es decir, son los que guardan la información del formato y la estructura de la página web para ciertas funcionalidades, es decir, se crea la parte de la interfaz con el usuario (vista), tal y como aparece en la Figura 3-30.

```
{% block content %}

<h2>Account</h2>
<form method="post">{% csrf_token %}
  {% for field in account_form %}
    <p>
      {{field.label_tag}}
      {{field}}

      {% if field.help_text %}
        <small style="color: grey;"> {{field.help_text}}</small>
      {% endif %}

      {% for error in field.errors %}
        <p style="color: red;"> {{error}}</p>
      {% endfor %}

      {% if account_form.non_field_error %}
        <div style="color: red;">
          <p>{{account_form.non_field_errors}}</p>
        </div>

      {% endif %}
    </p>
  {% endfor %}
  <button type="submit">Guardar</button>
</form>
{% endblock content %}
```

Figura 3-30 Account.html. Fuente: Propia.

Una vez creada la vista, se pasa a crear la parte del modelo, es decir, una carpeta dentro del proyecto que incluye diferentes archivos Python, así como una carpeta “caché”. La función principal del modelo será la de gestionar la información introducida por el usuario respecto a sus credenciales. De esta manera, se comprobará que los datos introducidos por el usuario para su registro no han sido utilizados previamente. En caso de que no hayan sido utilizados para registrar a otro usuario, los almacenará, y si por el contrario ya están en la base de datos se lo indicará al usuario.

```
class MyAccountManager(BaseUserManager):
    def create_user(self, email, username, password=None):
        if not email:
            raise ValueError('Los usuarios deberán tener un correo valido')
        if not username:
            raise ValueError('Todos los usuarios deben tener un nombre de usuario')

        user = self.model(
            email=self.normalize_email(email),
            username=username,
        )

        user.set_password(password)
        user.save(using=self._db)
        return user
```

Figura 3-31 Modelo.py. Fuente: Propia.

El fichero `views.py` que junto con el fichero `forms.py` son los controladores, es decir, son los encargados de llevar a cabo todas las acciones como por ejemplo, registrar, autenticar o cerrar la sesión del usuario. De esta forma, una vez el usuario los haya completado y se hayan enviado al modelo correspondiente (en este caso la base de datos `db.sqlite3`), estos tendrán una forma y se almacenarán correctamente. Al mismo tiempo, si el super usuario, es decir, el administrador deseara conocer cualquier tipo de información acerca de estos usuarios, podría acceder a la misma.

Al mismo tiempo, una vez realizadas las acciones, según lo que se indique para cada función, el controlador entregará al usuario final una respuesta diferente.

```
def registration_view(request):
    context = {}
    if request.POST:
        form = RegistrationForm(request.POST)
        if form.is_valid():
            form.save()
            email = form.cleaned_data.get('email')
            raw_password = form.cleaned_data.get('password1')
            account = authenticate(email=email, password=raw_password)
            login(request, account)
            return redirect('home')
        else:
            context['registration_form'] = form
    else:
        form = RegistrationForm()
        context['registration_form'] = form
    return render(request, 'account/register.html', context)
```

Figura 3-32 Controlador. Fuente: Propia.

De esta manera es posible conocer quienes tienen acceso a la información proporcionada. Así mismo, el gestor de la página web tiene acceso a una serie de opciones en las que puede modificar ciertos aspectos de los usuarios, tal como asignar privilegios.

Toda la información a la que puede acceder el administrador la proporciona “Django” a través de la página del administrador. En esta se puede observar en la parte derecha, un historial con todos los movimientos realizados, y da la oportunidad al mismo tiempo de revertir acciones en caso de que sea necesario. En la parte izquierda se le ofrece la opción de acceder a la información de los usuarios o los grupos creados tal y como aparece en la Figura 3-33.

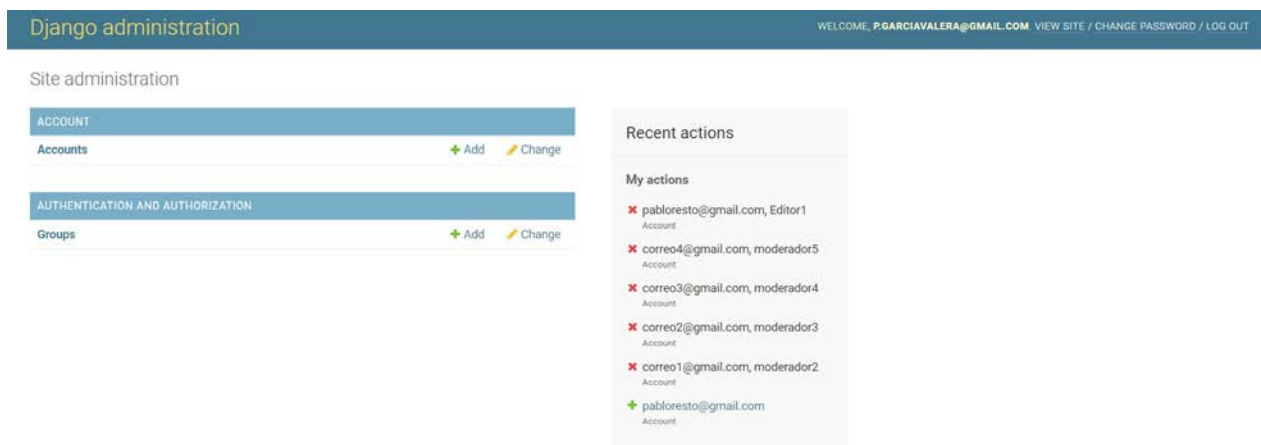


Figura 3-33 Página principal del administrador. Fuente: Propia.

El interfaz que permite interactuar con la información de los usuarios permite crear usuarios directamente como administrador, sin necesidad de rellenar el formulario por parte del usuario. Al mismo tiempo, permite conocer cierta información de los usuarios, filtrar los usuarios según la necesidad o buscar un usuario en concreto.

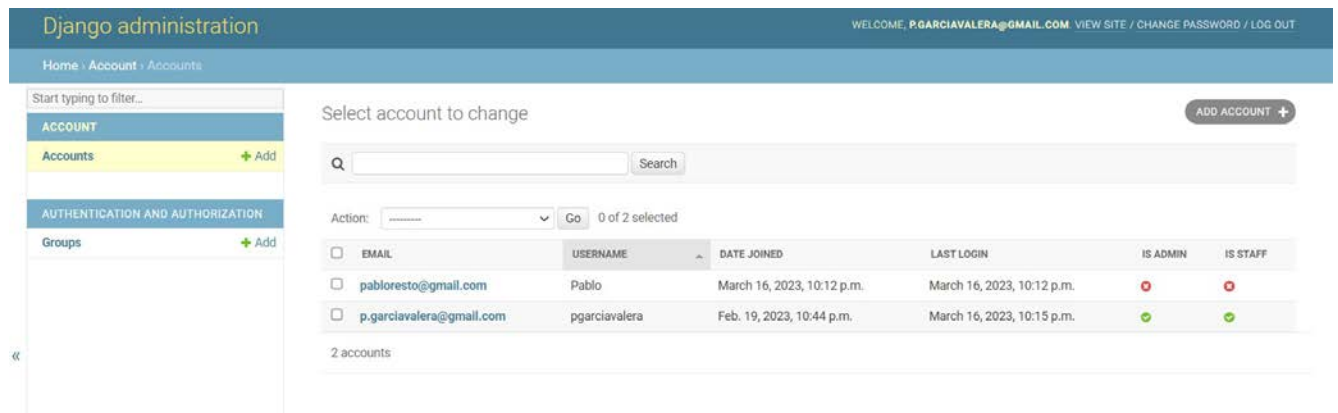


Figura 3-34 Administrar usuarios. Fuente: Propia.

Dentro de la información que “Django” proporciona con respecto a los diferentes usuarios es importante resaltar que la contraseña se encuentra cifrada por razones de protección de datos, de tal manera que solamente las personas autorizadas sean aquellas que puedan conocer esta información. Además, la aplicación permite conocer el historial de todas las acciones que ha realizado el usuario, así como conocer los permisos de los que goza, su fecha de registro y la última vez que el usuario inició sesión en la plataforma.

Finalmente, es posible eliminar el usuario, guardar este y otro con los cambios realizados, guardar los cambios realizados y seguir editando o guardar la información que cambie definitivamente la configuración del usuario.

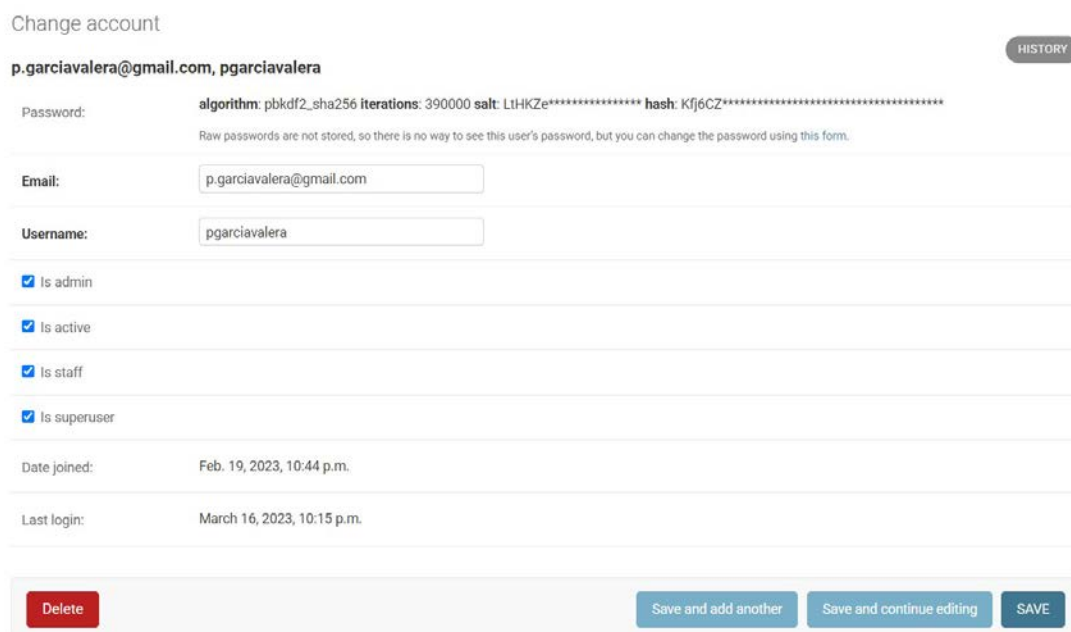


Figura 3-35 Información del usuario. Fuente: Propia.

Esta información la almacenará el administrador de esta, el cual tendrá acceso a la información de los usuarios, sin embargo, por motivos de privacidad, todas las claves estarán ocultas y no podrán ser observadas.

3.4.3.2 Consulta de la información del Gemelo Digital Oceanográfico

Una vez creada la funcionalidad de gestión de usuarios, es necesario centrarse en crear una funcionalidad que posibilite la consulta de la información del Gemelo Digital Oceanográfico y sirva de interfaz entre el usuario y los datos, de tal manera que estos sean accesibles de manera rápida y sencilla. De esta manera se crea una funcionalidad denominada *personal*, en la que, siguiendo un procedimiento similar se creará una funcionalidad que cumpla con el objetivo principal del proyecto.

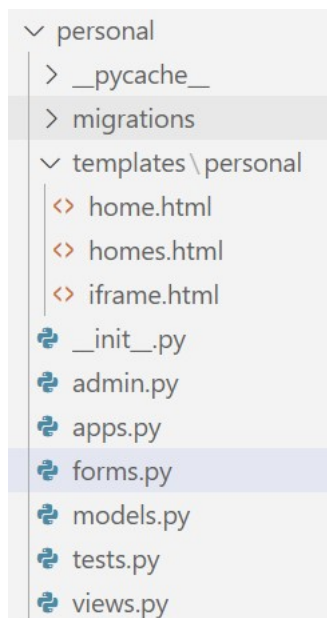


Figura 3-36 Aplicación "Personal". Fuente: Propia.

De la misma forma que en el caso anterior, al crear una nueva aplicación se generan automáticamente una serie de ficheros Python, así como se requiere de la creación de una carpeta para almacenar los ficheros `html` que se utilizarán para crear el interfaz.

Las principales diferencias entre estas dos aplicaciones son las de rellenar un formulario y que en este caso no será necesario hacer uso de la base de datos existente en el proyecto. Por esta razón, en esta aplicación será necesario modificar únicamente los archivos `views.py` y las diferentes vistas.

La razón por la que en este caso no se requiere de la utilización de la base de datos `db.sqlite3` es que la información a la que el usuario está intentando acceder se encuentra en una base de datos no estructurada, a la cual se accede a través de las acciones realizadas por el controlador.

Dentro del archivo `views.py` se definirán todas aquellas funciones que sean de utilidad para el tratamiento de la información, además de definir las funciones que permitirán al usuario acceder al interfaz en el que tendrá que escribir la información y donde se mostrarán los resultados obtenidos.

En la carpeta de `templates` se puede observar que aparecen diferentes ficheros `html`, estos son los diferentes interfaces que observará el usuario, siendo `Home` el interfaz que se observará en caso de que el usuario esté registrado correctamente `Homes`, el interfaz en caso de no estar registrado, quedando el archivo `iframe`, el cual no es un interfaz al que el usuario tenga que indicarle ningún tipo de datos si no que se trata del fichero `html` que se genera con los datos. Es decir, es el archivo que contiene la información final de interés para el usuario.



Figura 3-37 Página principal de la web. Fuente: Propia.

Una vez se accede a la página principal tras el registro del usuario, es posible acceder a las diferentes simulaciones según la información deseada. La disposición general de la página web permite, al estar dividida en 2 columnas, en un lateral indicar los datos necesarios y en el otro representar la información deseada. Para conseguir visualizar unos datos concretos es necesario rellenar los campos del formulario que aparecen en pantalla, indicando: la unidad en la que nos encontramos, latitud y longitud del punto en que queremos conocer la previsión, el momento de los datos que queremos conocer (siguiendo el formato de ejemplo), el rango de acción y por último la resolución del mallado H3. Estos últimos datos se requieren para poder tener una representación que abarque una mayor o menos cantidad de datos y al mismo tiempo con una mayor o menor precisión.

Figura 3-38 Pagina web, recopilación datos. Fuente: Propia.

Para poder recopilar los datos de la Figura 3-38 es necesario crear un fichero html en el que se indique a que función se van a enviar los datos, el método por el que los datos son recogidos, que dato se recoge en cada espacio y que modelo ha de ejecutarse en cada caso. De esta manera, en el siguiente código se indica mediante el valor `method = post` el método por el cual se va a recoger la información entregada por el usuario. Y en el apartado `action` se indica a que función se tiene que enviar esa información. para poder realizar esto correctamente, es necesario crear una ruta a esta función asignándole una `url`, tal y como aparece en la

```
<div class="row">
  <div class="d-flex flex-column justify-content-end align-items-center col-md-9">
    <h2>Gemelo Digital:</h2>
    <iframe src="{% url 'iframe' %}" name="iframe_principal" width="100%" height="100%"></iframe>
  </div>
  <div class="d-flex flex-column justify-content-end align-items-center col-md-3">
    <h3>Datos del buque</h3>
    <form method="post" action="{% url 'todo' %}">
      {% csrf_token %}
    </form>
  </div>
</div>
```

Figura 3-39 Código inicial del formulario. Fuente: Propia.

El siguiente paso es crear una serie de espacios dedicados a la recopilación de la información, creando así una etiqueta en la que se indica el tipo de información que se requiere e imprimiendo en pantalla los datos necesarios para que el usuario sepa donde insertar correctamente los valores.

Debido a las características del modelo planteado, y como medida preventiva para evitar el mal funcionamiento de la página web y posibles errores derivados del mal uso de este, a la hora de crear el formulario, se establecerán una serie de filtros para poder llevar a cabo una validación de los datos insertados por el usuario antes de enviarlos al ejecutor. Este filtro es individual de cada espacio en el formulario, adecuándose a las necesidades específicas de cada campo.

En primer lugar se crea el espacio necesario para que el usuario indique el nombre de la unidad en la que se encuentra. Este campo no requiere de ningún filtro ya que es únicamente informativo para que el usuario al colocar el ratón encima de la unidad sepa de cual se trata. De esta forma, tal y como aparece en la Figura 3-40, se indicará el tipo de texto que se introducirá, asignándole una identificación específica, un tamaño para que se ajuste al ancho de la página y finalmente el nombre del valor con el que esta información se almacenará.

```
<div class="form-group">
  <label for="{% form.unidad.id_for_label %}">Unidad:</label><br>
  {{ form.unidad }}
  <input type="text" name="unidad" id="unidad" size="25" value="{% unidad %}">
</div>
```

Figura 3-40 Formulario unidad. Fuente: Propia.

El siguiente campo que es importante crear es el de la latitud y la longitud. Estos campos son de especial importancia ya que su mala utilización puede derivar en errores de interpretación por parte del código como errores a la hora de acceder a la información. Teniendo esto en cuenta, tanto para la latitud como para la longitud se establecen los filtros que aparecen en la Figura 3-41.

```
<div class="form-group">
  <label for="{% form.latitud.id_for_label %}">Latitud:</label><br>
  {{ form.latitud }}
  <input type="number" name="latitud" id="latitud" style="width: 100%" value="{% latitud %}"
    pattern="/^\d+$/" min="40" max="50" step="any"/>
</div>
```

Figura 3-41 Formulario latitud y longitud. Fuente: Propia.

Los filtros aplicados en este espacio empiezan definiendo el tipo de texto que se quiere insertar en este campo, además del patrón que se seguirá, es decir, la forma en la que se insertan los datos (/^\d+\$/). De esta manera, al indicar que será un número, el formulario lo entenderá como si fuese una variable de tipo `float` consiguiendo así evitar errores a la hora de utilizar puntos y comas en la separación de los decimales. Es decir, en este campo debido al tipo de texto solamente se podrán introducir números, siendo indiferente el tipo de separador, ya que lo interpretará correctamente en ambos casos. Otro de los aspectos importantes es establecer el número de decimales que serán aceptados en este campo. Para ello el filtro `step="any"` es el encargado de indicar que no existe límite a la hora de introducir decimales, mejorando así la precisión a la hora de establecer la situación.

Finalmente, para evitar los errores que puedan llevar al usuario a introducir unas coordenadas incorrectas y por lo tanto impedir el acceso adecuado a la información, se establece un rango de valores entre los que el usuario puede elegir, estableciendo únicamente los valores máximos y mínimos.

A diferencia del caso anterior, debido al tipo de texto que se está insertando en este caso, para poder ajustar el tamaño de la celda al máximo posible dentro de la página web, se establece el filtro al 100%.

Una de las características que se aprecian debido a los filtros establecidos, es la posibilidad de variar el número mediante unas flechas que aparecen en el lateral, tal y como aparece en la Figura 3-42. De esta manera es posible realizar ajustes de manera más rápida sin tener que introducir todos los datos de nuevo. Esta opción de utilizar las flechas aparecerá en todos aquellos campos en los que se establece como tipo de texto un valor numérico.

Figura 3-42 Filtros numéricos. Fuente: Propia.

En el caso de la hora en la que se desea conocer el estado del entorno marino, no es posible establecer estos filtros tan específicos debido al tipo de información que se ha de rellenar en este campo. Es por esto por lo que el formato de esta información tendrá un diseño similar al del nombre de la unidad. Cambiando únicamente los valores relacionados con la identificación de los datos.

El siguiente valor es el del radio de acción. Los filtros necesarios persiguen evitar que el usuario introduzca radios que impidan una correcta representación de los estos. Para ello simplemente se establecen aquellos valores máximos y mínimos entre los que el usuario puede elegir, tal y como aparece en la Figura 3-43.

```
<div class="form-group">
  <label for="{{ form.radio.id_for_label }}">Radio de accion (km):</label><br>
  {{ form.radio }}
  <input type="number" name="radio" id="radio" style="width: 100%" value="{{ radio }}"
    max="5" min="1" pattern="/^\d+$/"/>
</div>
```

Figura 3-43 Formulario radio de acción. Fuente: Propia.

Finalmente, y de la misma forma que ocurre con el radio de acción es posible acotar la resolución del mallado. Esto busca conseguir que el buque no elija una resolución que pueda derivar en problemas de procesamiento de la información o que debido al gran tamaño de las celdas estas no sean representativas de lo que realmente está ocurriendo en una zona. Es por esto por lo que, los filtros serán iguales que los aplicados en la Figura 3-43, cambiando únicamente los valores máximos y mínimos.

El siguiente paso es el de crear la lista de botones en el que se indica el nombre del modelo necesario, de esta manera, al ejecutar el código de la función se podrá seguir una estructura `elif` tal y como aparece en la Figura 3-46 que ejecute únicamente el código seleccionado por el usuario. Para llevar a cabo esta acción en el fichero `html` se indica que los modelos se representen como botones, dándoles un valor que será el utilizado posteriormente para la identificación del modelo seleccionado por el usuario.

```
<a><button value="Batimetria" type="submit" class="list-group-item list-group-item-action" name="submit" >Batimetria</button></a>
<a><button value="Humedad" type="submit" class="list-group-item list-group-item-action" name="submit">Humedad</button></a>
<a><button value="Nubosidad" type="submit" class="list-group-item list-group-item-action" name="submit">Nubosidad</button></a>
<a><button value="Oleaje" type="submit" class="list-group-item list-group-item-action" name="submit" >Oleaje</button></a>
<a><button value="Precipitaciones" type="submit" class="list-group-item list-group-item-action" name="submit">Precipitaciones</button></a>
<a><button value="Presion" type="submit" class="list-group-item list-group-item-action" name="submit">Presion</button></a>
<a><button value="Salinidad" type="submit" class="list-group-item list-group-item-action" name="submit">Salinidad</button></a>
<a><button value="Temperatura" type="submit" class="list-group-item list-group-item-action" name="submit">Temperatura</button></a>
<a><button value="Temperatura agua" type="submit" class="list-group-item list-group-item-action disabled" name="submit">Temperatura agua</button></a>
<a><button value="Viento" type="submit" class="list-group-item list-group-item-action" name="submit">Viento</button></a>
```

Figura 3-44 Html Botones. Fuente: Propia.

Una vez todos estos datos han sido cubiertos, se deberá elegir el modelo necesario. De esta manera, seleccionando el modelo en la lista que aparece en la Figura 3-45, se procederá al acceso y representación de los datos. Tal y como aparece en esa figura, puede ocurrir que algún modelo no represente los datos correctamente debido a que el valor buscado es igual en todos los puntos. De esta forma cuando se ejecuta el código necesario, este devuelve un error, por ello estos modelos han sido desactivados de la lista, indicando en el código `html` de la Figura 3-44 aquellos modelos que no pueden ser elegidos.

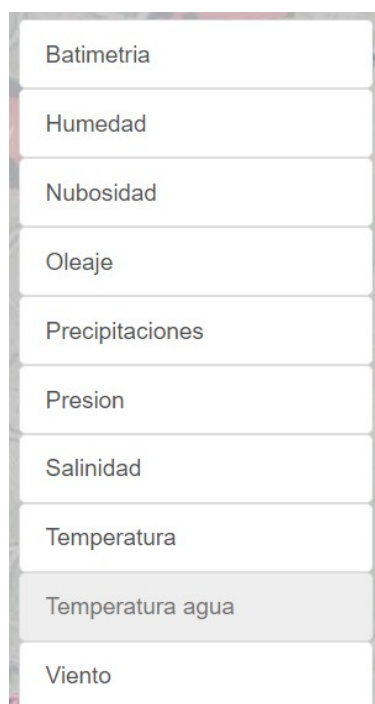


Figura 3-45 Diseño botones. Fuente: Propia.

```
if request.method == 'POST':
    if 'Batimetria' in request.POST.get('submit'):

    elif 'Humedad' in request.POST.get('submit'):

    elif 'Nubosidad' in request.POST.get('submit'):

    elif 'Oleaje' in request.POST.get('submit'):

    elif 'Precipitaciones' in request.POST.get('submit'):

    elif 'Presion' in request.POST.get('submit'):

    elif 'Salinidad' in request.POST.get('submit'):

    elif 'Temperatura' in request.POST.get('submit'):

    elif 'Viento' in request.POST.get('submit'):

    else:
        return redirect('home')
```

Figura 3-46 Elif. Fuente: Propia.

Finalmente, para ejecutar el Gemelo Digital correctamente el primer paso es acceder al código indicado, y comprobar que el método para indicar la información es el correcto, empleando para ello el elemento `if`. En caso de que esta condición sea correcta se continuarán ejecutando las siguientes líneas, mientras que, si el método no coincide, la página nos redirigirá automáticamente a la página principal donde se requerirá que se indiquen los datos de nuevo.

A continuación, mediante el uso de `elif`, se buscará el modelo seleccionado por el usuario, comparando el nombre asignado a cada función con el valor que se ha asignado a la variable `submit`, el cual se asocia automáticamente al pulsar sobre el botón. De esta manera es posible indicarle al Gemelo Digital Oceanográfico que modelo de simulación es el requerido por el usuario.

Una vez se ha seleccionado el modelo y se ejecuta el código se accede a la información que el usuario ha insertado en los campos anteriores, para ello es necesario hacer uso del método `request.POST.get()`, de esta forma estamos tomando el valor asociado a cada `id` en la página principal y asociándolo a una nueva variable que será útil para acceder a la información y representarla de forma correcta.

```
elif 'Temperatura' in request.POST.get('submit'):
    unidad = request.POST.get('unidad')
    Lati = request.POST.get('latitud')
    Longi = request.POST.get('longitud')
    moment_indicado = request.POST.get('hora')
    radio_accion = request.POST.get('radio')
    resolucion = request.POST.get('resolucion')
```

Figura 3-47 Asignación de variables. Fuente: Propia.

Antes de finalizar la función y devolver al usuario a la página home de la página web, el código cambia de ubicación el archivo final creado. Esto se debe a que la manera escogida para la representación de los datos es mediante un `iframe`. Y para que este se ejecute correctamente es necesario que tanto el archivo creado como el interfaz donde se representará estén almacenados en la misma carpeta.

Una vez que se finaliza el código final, el usuario es dirigido a la página principal de nuevo. Con el objetivo de que el usuario no deba insertar de nuevo todos los datos necesarios y pueda ejecutar otro modelo de forma rápida, es necesario enviar los datos de entrada.

```
m.add_child(minimap)
m.save("iframe.html")
shutil.move('iframe.html', 'personal/templates/personal/iframe.html')
return render(request, 'personal/home.html', {'unidad' : unidad, 'latitud' : Lati, 'longitud' ... })
```

Figura 3-48 Añadir marcas. Fuente: Propia.

El objetivo final de la página web es conseguir que el usuario no tenga que cambiar de página o refrescar la página para mostrar el mapa con la información que buscaba en el espacio especificado. Por esto todos los archivos descargados de los diferentes mapas con la información de diferentes fenómenos tienen el mismo nombre. De esta forma al descargar un archivo y cambiarlo de ubicación estos se sobrescribirán, de tal manera que no se pueda producir errores en la interpretación y la extracción de los datos.

Consiguiendo, al igual que se hizo en el diseño de la web y se estableció una columna para la entrada de los datos por parte del usuario, la otra columna está destinada a la representación de los resultados finales. Los únicos factores importantes para este caso son la definición del tamaño de la columna y asignar correctamente el nombre a la fuente de donde se obtendrá el `iframe` de tal manera que no se produzcan errores en la representación de la información.

```
<div class="row">
  <div class="d-flex flex-column justify-content-end align-items-center col-md-9">
    <h2>Gemelo Digital:</h2>
    <iframe src="{% url 'iframe' %}" name="iframe_principal" width="100%" height="100%"></iframe>
  </div>
```

Figura 3-49 Establecimiento “iframe”. Fuente: Propia.

Finalmente, para evitar que el usuario tenga que estar continuamente introduciendo los datos en el formulario, en el apartado `return`, se indicará a la página a la que se enviarán los datos obtenidos, así como cierta información que la página conoce. Es decir, una vez el ejecutor ha terminado de realizar todas las tareas que tenía asignadas, devolverá a la página principal, no solo el archivo `html` generado sino también los datos que ha utilizado para generarlo, es decir, los introducidos por el usuario. De esta manera se consigue que se represente la información solicitada y se rellene automáticamente el formulario, permitiendo al usuario realizar aquellos cambios que considere necesarios.

Además, tal y como aparece en la Figura 3-39, se habilita la opción de adaptar el tamaño y contenido de la página web a diferentes tamaños de pantalla. Esto es, que mediante `{% csrf_token %}` el contenido de la página se adaptara, mostrando siempre el mayor contendí posible. De esta forma no será necesario realizar ajustes que impliquen pérdidas de la información en la visualización de los resultados.

4 PRUEBA Y VALIDACIÓN DE RESULTADOS

4.1 Descripción funcionalidades de la página web

4.1.1 Gestión de usuarios: registro y acceso

Una vez se entra por primera vez en la página que sirve como interfaz, se puede observar que se indica al usuario que para poder hacer uso del sistema será necesario estar registrado previamente en la misma. La Figura 4-1 muestra ese mensaje. De esta forma, si es la primera vez que el usuario accede, podrá crear una cuenta que será utilizada para futuras consultas, o en caso de ya poseer credenciales, podrá registrarse con las mismas.

En caso de que el usuario quiera registrarse, este tendrá que rellenar un formulario tal y como aparece en la Figura 4-2, con datos para su identificación. Además de esto, una vez insertados, la aplicación comprobará que los datos indicados por el usuario son los correctos, es decir, el correo electrónico cumple con la arquitectura de estos y la contraseña cumple con los requisitos de seguridad y que no han sido utilizados previamente.



Figura 4-1 Interface de la página principal. Fuente: Propia.

Valera Engineering



[Inicio](#) | [Login](#) | [Registrate](#)

Crea tu cuenta!

Email: Obligatorio. Añade un email valido

Username:

Password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation: Enter the same password as before, for verification.

Figura 4-2 Formulario de registro. Fuente: Propia.

En caso de que el usuario ya posea una cuenta, el único procedimiento para acceder a los diferentes modelos será introducir los datos de usuario y contraseña, siguiendo los pasos como en la Figura 4-3.

Valera Engineering



[Inicio](#) | [Login](#) | [Registrate](#)

Login

Email:

Password:

By: Pablo Garcia Valera | 2023

Con la colaboración de:

Dña. Milagros Fernández Gavilanes

D. Carlos Pérez Collazzo

Figura 4-3 Inicio de sesión. Fuente: Propia.

Una vez se ha accedido, la vista del sitio cambia. Tomando como referencia la Figura 4-4, se observa como aparecen, los diferentes campos para introducir los datos de interés y poder así, obtener los resultados de las simulaciones.



Figura 4-4 Página principal. Fuente: Propia.

Además del acceso y registro del usuario, existe la posibilidad de que el usuario modifique algunos de los datos que habrá insertado previamente. Para ello, en la parte superior se encuentra Account donde se mostrarán aquellos campos que se pueden modificar, tal y como aparece en la Figura 4-5, siendo así el correo electrónico proporcionado y el nombre de usuario asociado a la cuenta. De la misma forma que sucedía en el apartado anterior, si se intenta cambiar el nombre de usuario introduciendo uno que existiese previamente la plataforma impedirá esta acción.



Figura 4-5 Cambios al usuario. Fuente: Propia.

4.1.2 Acceso a datos y su representación



Figura 4-6 Página principal. Fuente: Propia.

Una vez el usuario se ha registrado y autenticado, el siguiente paso es el de introducir cuales son los datos necesarios, así como las coordenadas geográficas para obtener la información asociada a una fuente concreta, para posteriormente visualizarla en un mapa. Para ello en la parte derecha de la página existirá un apartado en el que el usuario tendrá que rellenar un formulario, además de seleccionar el modelo que desee, tal y como aparece en la Figura 3-45.

Datos del buque

Unidad:
GM SALAS

Latitud:
42.36197

Longitud:
-8.80011

Hora (2023-02-07T23:00:00+01):

Radio de accion:
4

Resolución mallado H3 (0-16):
10

Figura 4-7 Indicar datos necesarios. Fuente: Propia.

Uno de los primeros test realizados está relacionado con probar el correcto funcionamiento del código en lo que a la representación de la batimetría se refiere. Es importante resaltar que, a diferencia de la mayoría de los fenómenos meteorológicos, la batimetría no es algo que esté en constante cambio (la información cambia cada cuatro años, cuando se realizan campañas hidrográficas), es por esto por lo que la frecuencia de actualización de estos datos es tan baja que se puede considerar como constante en el tiempo.

De esta manera, siempre que el usuario necesite obtener datos de la batimetría no será necesario indicar la hora en la que se desea conocer el dato ya que este parámetro se considere invariable en el tiempo.

Una vez se selecciona el parámetro de la batimetría seleccionando el modelo según la Figura 3-45, se obtendrá en el espacio indicado para ello el mapa interactivo representando el dato solicitado.

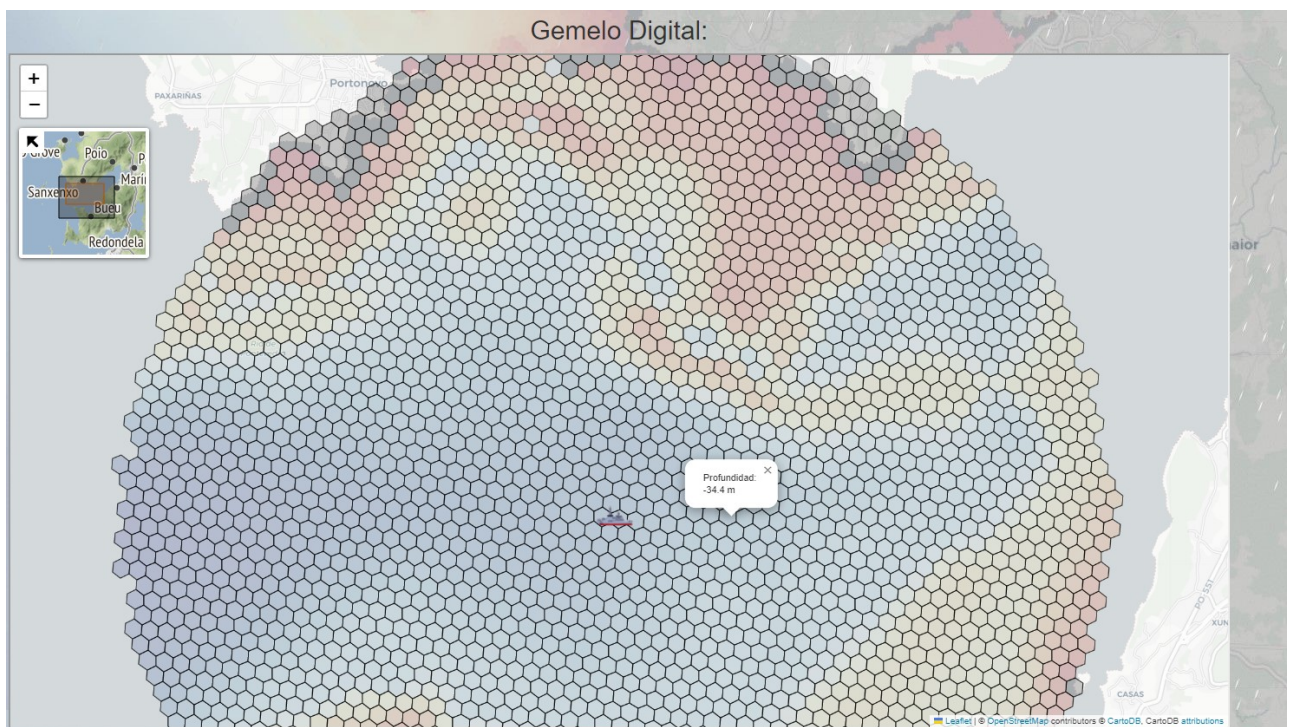


Figura 4-8 Resultado batimetría. Fuente: Propia.

Tal y como se observa en Figura 4-8, se ha obtenido como resultado un mapa interactivo de tal manera que es posible consultar la información requerida por el usuario. Los hexágonos que aparecen en la imagen presentan colores diferentes en el mapa de calor. Esto es debido a la creación de una escala en la que, en función del valor medio de los puntos que recaen en cada hexágono. De esta forma aquellos valores con profundidades menores presentan colores rojizos, mientras que aquellos más profundos colores azulados. Esto da una idea al usuario de aquellas zonas en las que será más seguro navegar.

Al mismo tiempo es posible observar como aquellos puntos sobre tierra no contienen ningún valor, esto es debido al filtrado previo que se hace para únicamente recopilar aquellos datos sobre los puntos que recaen en la mar, así como debido a que el mallado se ha de realizar sin posibilidad de eliminar aquellas zonas que no son de interés para el usuario.

Si se desean conocer los valores de otros parámetros meteorológicos en este caso será necesario indicar la hora concreta en la que se desea obtener el valor de los datos, indicándolos de la forma específica para no dar lugar a un error en los datos.

De esta forma, antes de proceder a realizar las simulaciones para aquellos valores que afecten de una manera más directa a las condiciones de navegación del buque. Es necesario calcular el tamaño de celda adecuado. Esto se debe a las diferencias entre las resoluciones de las celdas. Tal y como aparece en la Tabla 2-3, la resolución de una celda de tamaño 15 es extremadamente pequeña, ofreciendo por el contrario una precisión excepcional. Por el contrario, aquellas celdas de menor resolución tienen un área que no es realista, es decir, su tamaño es tan grande que no aportan ningún tipo de información útil.

Para conseguir establecer un criterio de selección de este tamaño se ha calculado el área de una celda teniendo en cuenta en valor de la apotema, estableciendo el valor de este como la eslora del buque.

De esta manera, el cálculo realizado determina que para buques de eslora de 20 metros aproximadamente, tal y como son las lanchas de instrucción (A-121, A-122, A-123 y A-124) el tamaño de la celda deberá oscilar entre resoluciones de 9 y 10. De esta manera, la lancha podrá estar dentro de una única celda en caso de que la resolución sea de 9, dejando un pequeño margen al límite de cada celda. En caso de elegir en este caso la resolución de 10, la lancha tendrá una eslora mayor a la longitud del hexágono, sin embargo, esta diferencia es ínfima, por lo que sería aquella que mejor se ajustaría en caso de querer una mayor precisión.

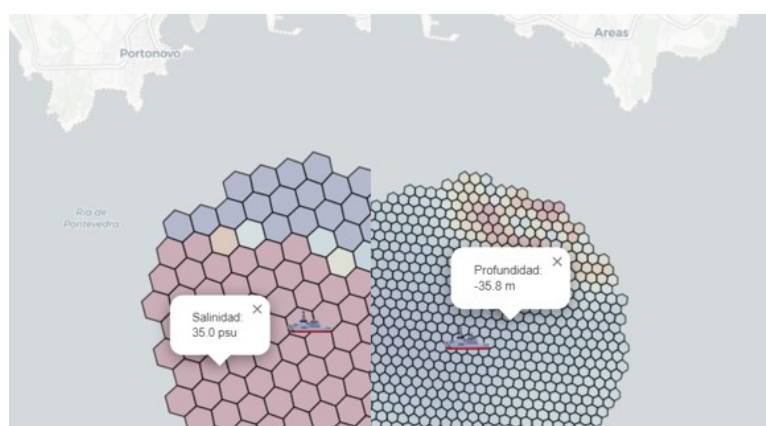


Figura 4-9 Comparación entre diferentes resoluciones (9/10). Fuente: Propia.

Realizando esta comparación se puede comprobar cómo es posible realizar mallados diferentes y como uno permite una mayor precisión, aunque su tamaño sea menor que la eslora de la lancha. Mientras que, por otro lado, se puede observar que las celdas abarcan un área mayor, permitiendo abarcar a la lancha en su área.

De la misma forma se ha optado por calcular cual debería ser el tamaño adecuado para aquellos barcos de esloras de mayor tamaño. Para ello se ha estimado la resolución necesaria para el LHD Juan Carlos I, debido a que es un barco con una eslora suficientemente grande para abarcar a aquellos de menor porte. Para este caso el tamaño de celda adoptado es de 7 aunque no permite obtener exactitud en la simulación. El área de las celdas se incrementa de manera exponencial, por lo que es difícil encontrar una celda que se ajuste al tamaño de cada buque.

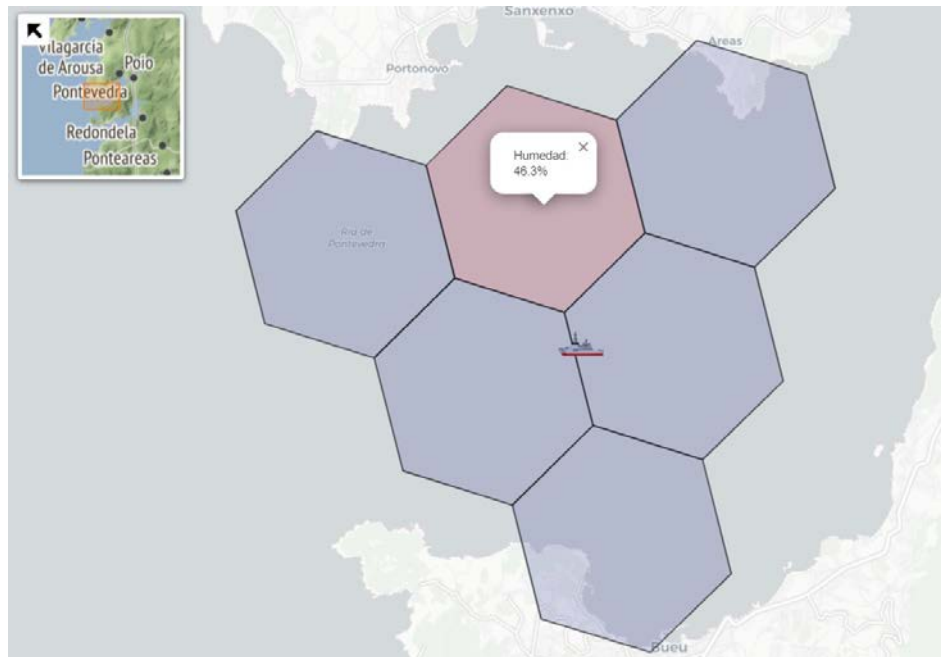


Figura 4-10 Mallado de resolución 7. Fuente: Propia.

Es importante resaltar que los cálculos realizados para estos mallados son aproximaciones, ya que no existe la posibilidad de ajustar el tamaño del hexágono para adecuarlo a un buque en concreto.

Una de las principales ventajas que ofrece este gemelo digital, es la de poder acceder a información relacionada con un mismo fenómeno únicamente accediendo a la información de un hexágono. Es decir, si se requiere conocer información acerca del oleaje, este modelo proporciona los valores tanto de dirección, altura y periodo en la misma simulación. De la misma forma, es posible acceder a la información de la dirección e intensidad del viento en el mismo modelo.



Figura 4-11 Oleaje y viento. Fuente: Propia.

4.2 Simulaciones

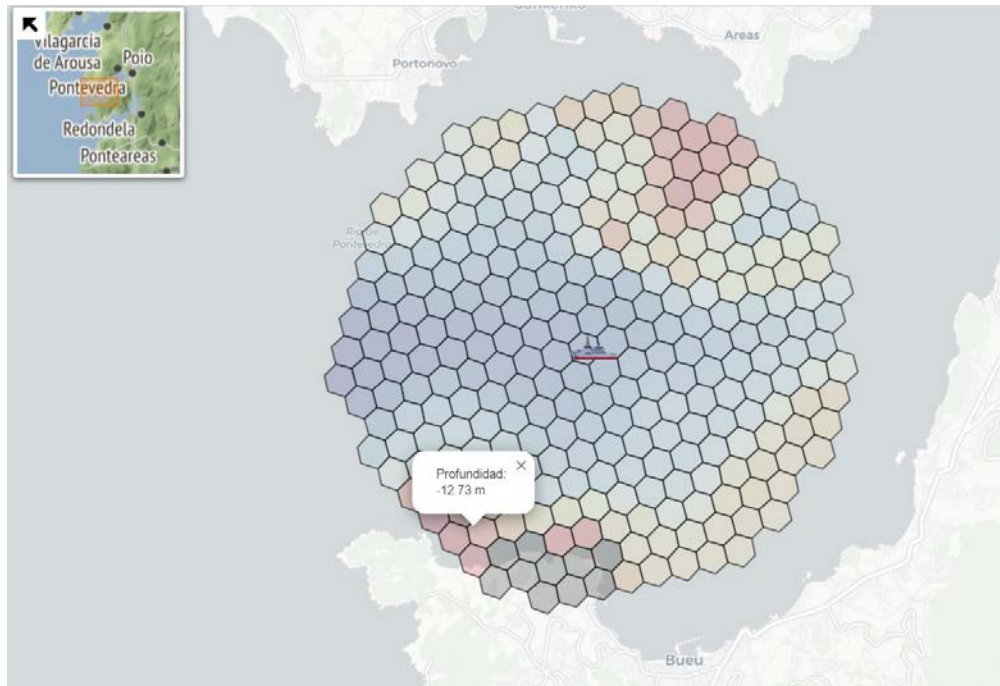


Figura 4-12 Batimetría. Fuente: Propia.

Simulación de la batimetría en las Rías Baixas para un radio de acción de 3km y una resolución de 9, en un punto de latitud 42,36197° N 8,80011° W.

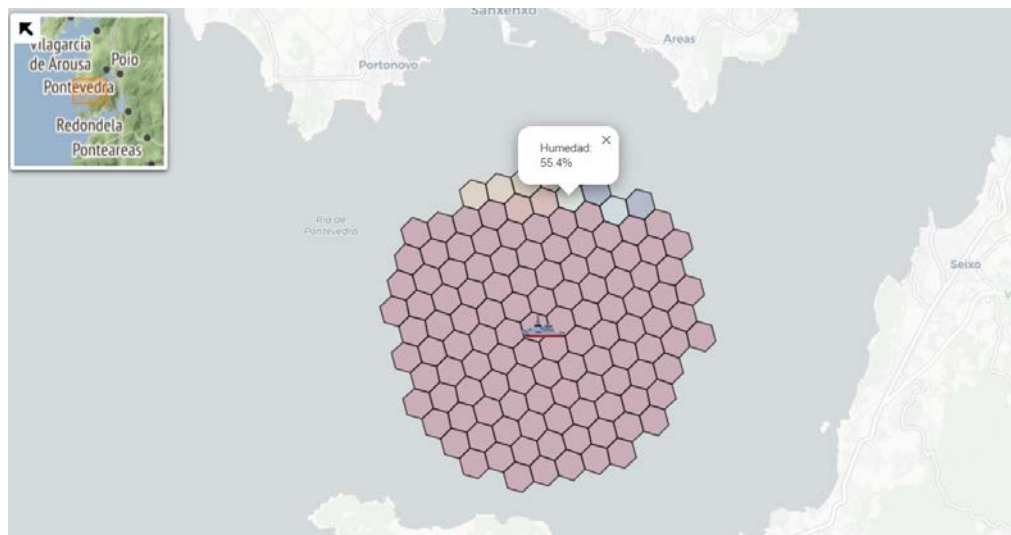


Figura 4-13 Humedad (2023-02-27T08:00:00+01). Fuente: Propia.

Simulación de la humedad en las Rías Baixas para un radio de acción de 2km y una resolución de 9, en un punto de latitud 42,36197° N 8,80011° W, para el día 2023-02-27T08:00:00+01.

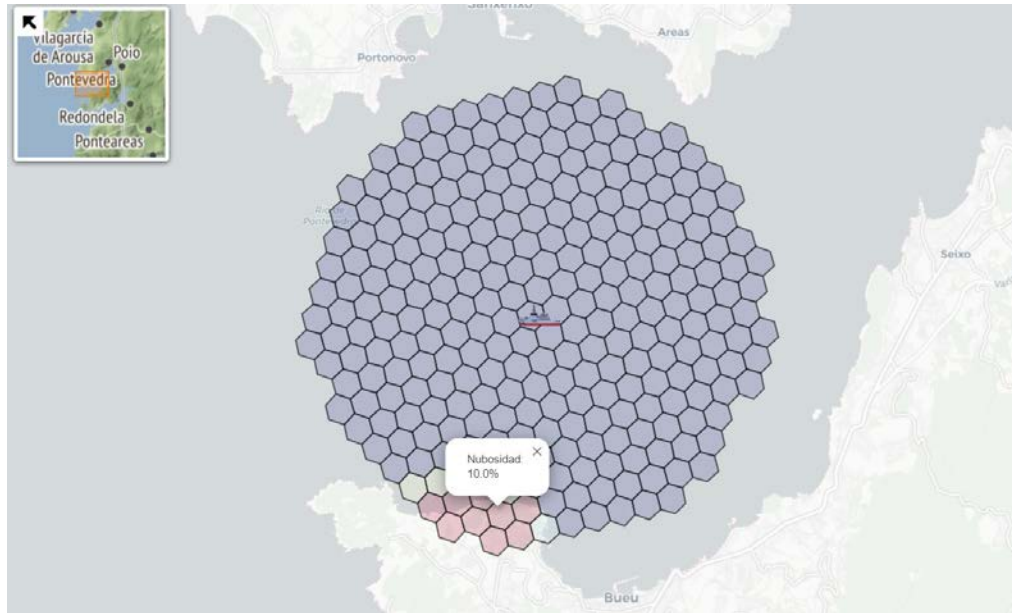


Figura 4-14 Nubosidad (2023-02-27T08:00:00+01). Fuente: Propia.

Simulación de la nubosidad en las Rías Baixas para un radio de acción de 3km y una resolución de 9, en un punto de latitud 42,36197° N 8,80011° W, para el día 2023-02-27T08:00:00+01.

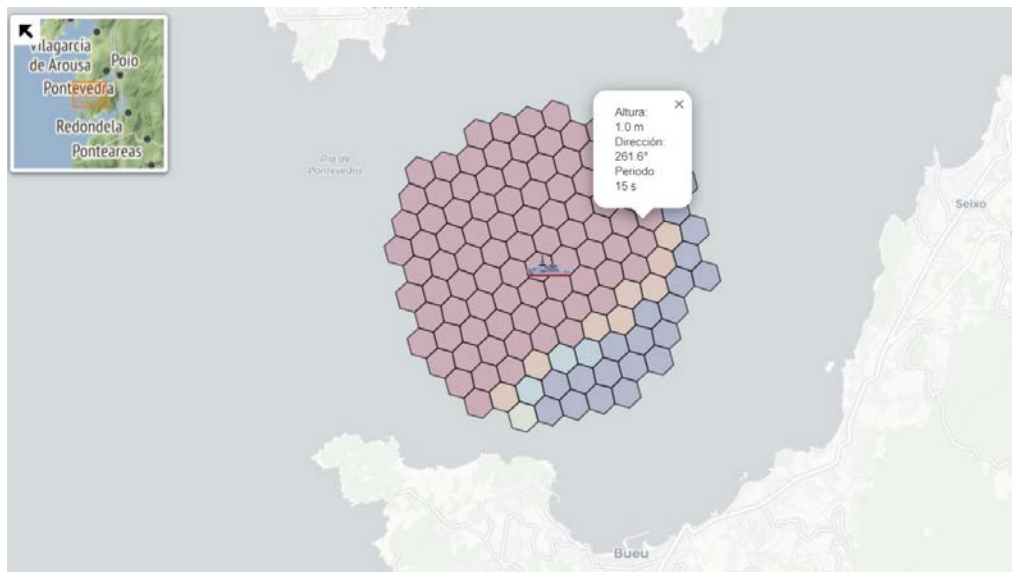


Figura 4-15 Oleaje (2023-03-01T17:00:00+01). Fuente: Propia.

Simulación del oleaje en las Rías Baixas para un radio de acción de 2km y una resolución de 9, en un punto de latitud 42,36197° N 8,80011° W, para el día 2023-03-01T17:00:00+01.

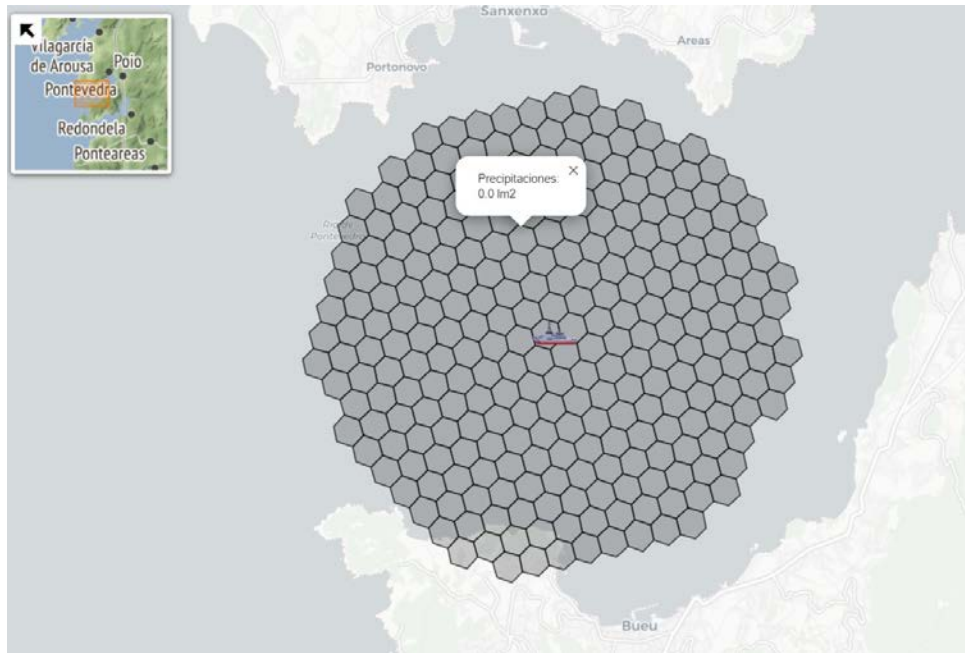


Figura 4-16 Precipitaciones (2023-03-01T23:00:00+01). Fuente: Propia.

Simulación de las precipitaciones en las Rías Baixas para un radio de acción de 3km y una resolución de 9, en un punto de latitud 42,36197° N 8,80011° W, para el día 2023-03-01T23:00:00+01.

En los casos en los que no existe una diferencia entre ningún valor medio de cada hexágono, no será posible calcular un rango de valores a los que asignar un color según el mapa de calor asignado, por lo que en estos casos la zona de estudio aparecerá sin color.

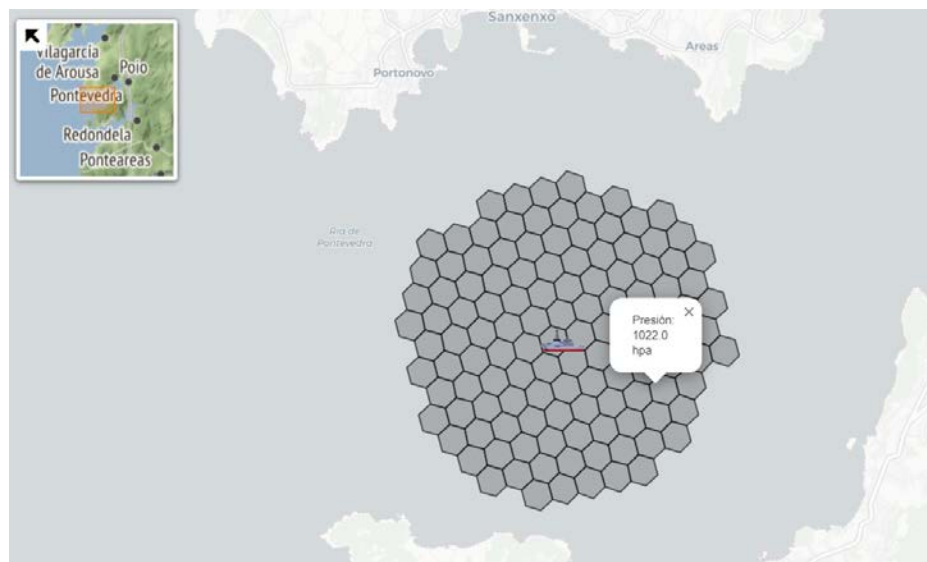


Figura 4-17 Presión (2023-03-02T02:00:00+01). Fuente: Propia.

Simulación de la presión en las Rías Baixas para un radio de acción de 2km y una resolución de 9, en un punto de latitud 42,36197° N 8,80011° W, para el día 2023-03-02T02:00:00+01.

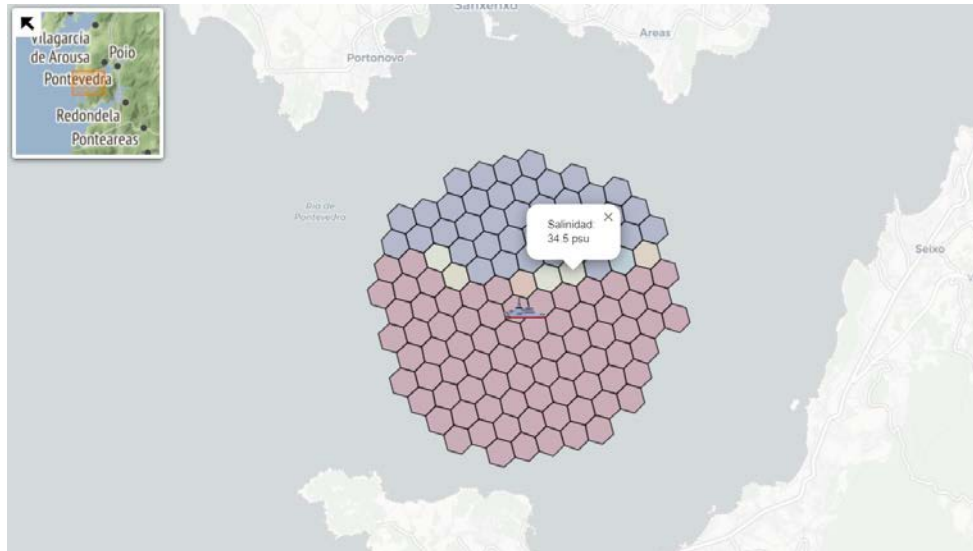


Figura 4-18 Salinidad (2023-02-27T12:00:00+01). Fuente: Propia.

Simulación de la salinidad en las Rías Baixas para un radio de acción de 2km y una resolución de 9, en un punto de latitud 42,36197° N 8,80011° W, para el día 2023-02-27T12:00:00+01.

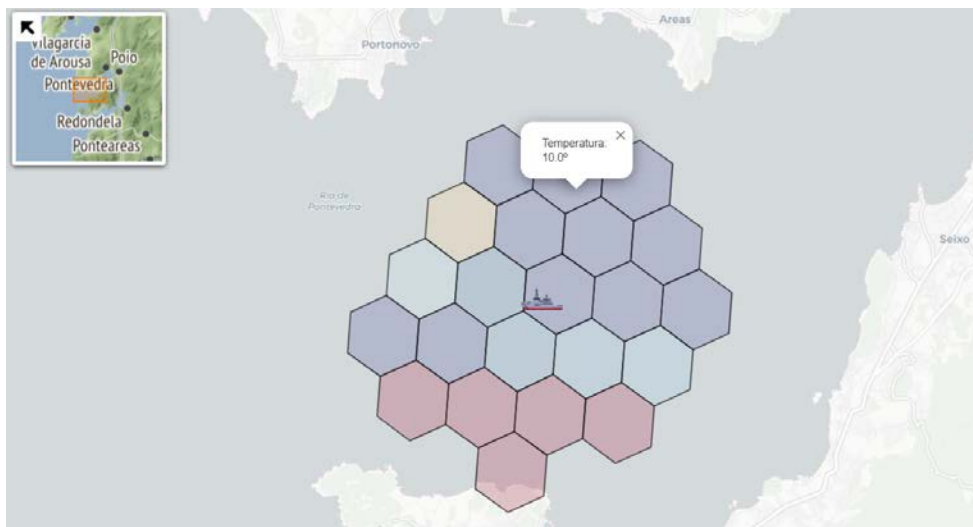


Figura 4-19 Temperatura (2023-02-08T03:00:00+01). Fuente: Propia.

Simulación de la temperatura en las Rías Baixas para un radio de acción de 2km y una resolución de 8, en un punto de latitud 42,36197° N 8,80011° W, para el día 2023-02-08T03:00:00+01.

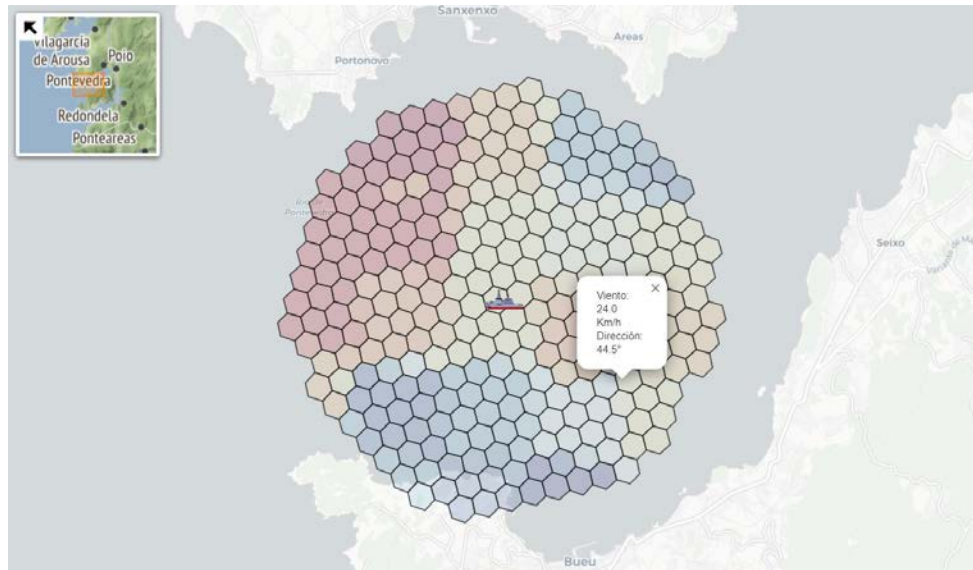


Figura 4-20 Viento (2023-02-26T23:00:00+01). Fuente: Propia.

Simulación del viento en las Rías Baixas para un radio de acción de 3km y una resolución de 9, en un punto de latitud 42,36197° N 8,80011° W, para el día 2023-02-26T23:00:00+01.

4.3 Mensajes de error

Los mensajes de error que pueden llegar a producir en la página web son aquellos que vienen producidos por la realización de acciones incorrectas.

4.3.1 Mensajes de error durante la administración de la cuenta de usuario

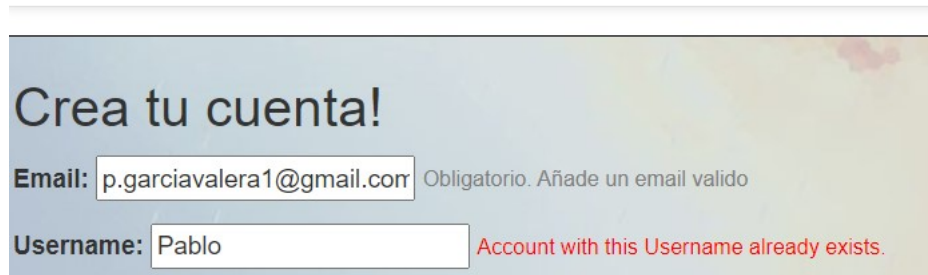
Un ejemplo de error es el que se produce durante la creación de un usuario, cuando el email que se está intentando registrar ya ha sido utilizado con anterioridad. En este caso aparece un mensaje en color rojo, tal y como aparece en la Figura 4-21, indicando al usuario el error. En este caso, el usuario no podrá continuar con la creación de la cuenta y tendrá que cambiar el correo electrónico.

[Inicio](#) | [Login](#) | [Regístrate](#)

Figura 4-21 Email ya existente en la base de datos. Fuente: Propia.

Al mismo tiempo cuando el usuario trata de insertar un nombre de usuario que ya existe, aparecerá un mensaje de las mismas características indicando al usuario el problema, tal y como aparece en la Figura 4-22. De la misma forma que con el caso del correo electrónico, el proceso no podrá finalizar si estos datos no se insertan correctamente.

[Inicio](#) | [Login](#) | [Regístrate](#)



Crea tu cuenta!

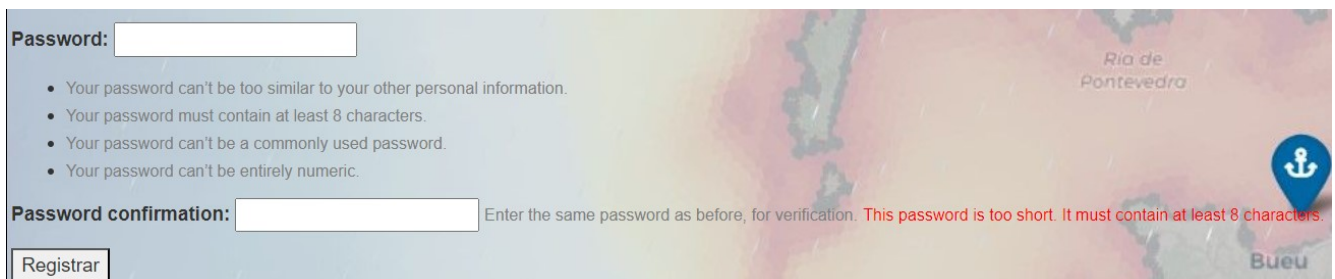
Email: Obligatorio. Añade un email valido

Username: Account with this Username already exists.

Figura 4-22 Nombre de usuario ya existente. Fuente: Propia.

Finalmente en este apartado aparecen los errores relacionados con la creación de la contraseña. De esta manera se presentan los siguientes problemas:

- En primer lugar, las contraseñas introducidas por el usuario no cumplen con alguna de las características que se indican y por lo tanto estas no serán válidas. Impidiendo el procedimiento de creación de la cuenta, tal y como aparece en la Figura 4-23.



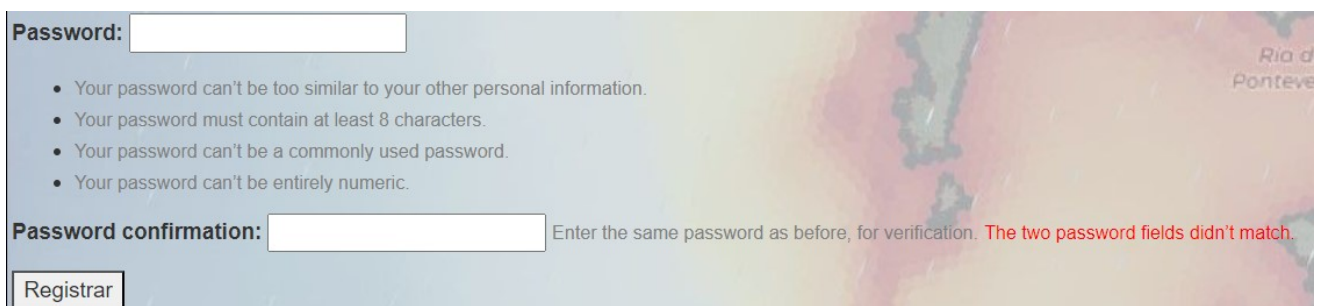
Password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation: Enter the same password as before, for verification. This password is too short. It must contain at least 8 characters.

Figura 4-23 Error en la configuración de la contraseña. Fuente: Propia.

- El otro posible error, es cuando las contraseñas introducidas no coinciden. En este caso y de la misma forma que en los casos anteriores aparecerá un mensaje automáticamente indicando el tipo de error y se interrumpirá el proceso, tal y como aparece en la Figura 4-24.



Password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation: Enter the same password as before, for verification. The two password fields didn't match.

Figura 4-24 Error coincidencia de contraseñas. Fuente: Propia.

De la misma forma que el sistema implementado avisa de los errores de formato en estos casos, también lo hará si se deja alguno de los formularios obligatorios en blanco. De esta manera aparecerá un *pop-up* con el símbolo adecuado indicando cual es el campo que no se ha introducido por el usuario y que se deberá introducir, tal y como aparece en la Figura 4-25.

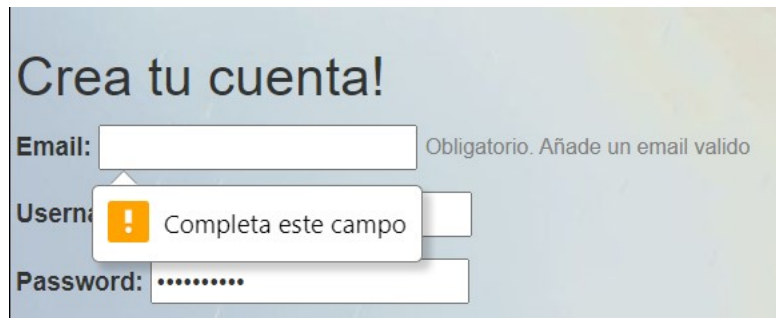


Figura 4-25 Formulario en blanco. Fuente: Propia.

A la hora de iniciar sesión, si se introducen los dato erróneos de correo o contraseña, el usuario no podrá entrar y validar sus credenciales. Sin embargo, en este caso no se proporciona ningún tipo de información al usuario, solamente se vuelve a la página para volver a rellenar el formulario y se borra la contraseña anteriormente introducida.

Por último, en caso de intentar cambiar los datos de una cuenta ya existente, pueden darse dos situaciones diferentes. La primera es que el nombre de usuario que se quiere establecer en la cuenta ya esté en uso, aparecerá un mensaje en rojo indicando al usuario cual es el motivo por el cual no se puede llevar a cabo el procedimiento, tal y como aparece en la Figura 4-26.



Figura 4-26 Nombre de usuario ya utilizado. Fuente: Propia.

Por último, en caso de querer cambiar el correo electrónico asignado a esa cuenta y el correo ya esté en uso en otra cuenta aparecerá el mensaje de la Figura 4-27 indicando al usuario el error.



Figura 4-27 Email ya en uso. Fuente: Propia.

4.3.2 Mensajes de error durante la realización de simulaciones



The image shows a web form titled "Datos del buque" (Ship Data). It contains several input fields with the following labels and values:

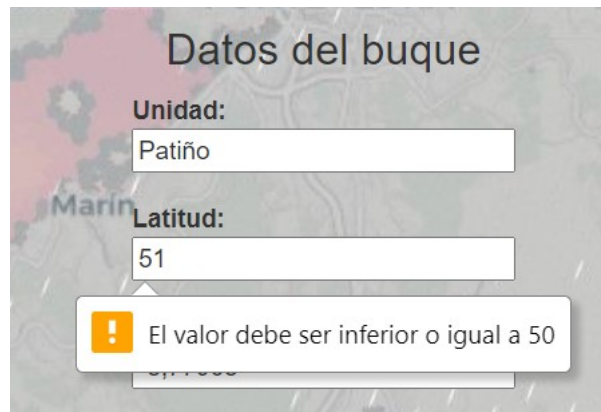
Label	Value
Unidad:	GM SALAS
Latitud:	42.36197
Longitud:	-8.80011
Hora (2023-02-07T23:00:00+01):	2023-03-02T02:00:00+01
Radio de accion (km):	3
Resolución mallado H3 (7-10):	8

Figura 4-28 Insertar datos. Fuente: Propia.

Tal y como se ha mencionado en los apartados anteriores el formato de los datos que inserta el usuario es de vital importancia para el correcto funcionamiento del sistema. Es por esto por lo que, a excepción de un exceso de tiempo de comunicación entre el acceso a la información y la aplicación que causará un *time out*, los errores que se pueden producir son aquellos que impidan acceder a la información de manera correcta.

Es por esto por lo que, antes de enviar los datos del formulario a la función en concreto, se realiza una validación de los mismos, indicando en el momento al usuario cuales han sido los fallos e impidiendo que este continúe con el proceso de simulación hasta que estos sean corregidos.

Siguiendo el orden lógico en el que se introducen los datos, el nombre de la unidad es informativo, por lo que no tendrá efecto en el código. Sin embargo, las coordenadas que se insertan deberán indicar tanto la latitud como la longitud, indicando el número de decimales que se considere oportuno. Así mismo, es importante indicar la longitud siguiendo un criterio de signos de tal manera que el signo negativo representa las coordenadas de poniente y las positivas oriente. En caso de no seguir estos formatos, aparecerá en la pantalla un mensaje al usuario en el que se le indica cual es el error que está cometiendo, e impidiendo continuar con el proceso hasta que los errores sean subsanados, tal y como aparece en la Figura 4-29 para la latitud y la Figura 4-30 para la longitud.



Datos del buque

Unidad:
Patiño

Latitud:
51

! El valor debe ser inferior o igual a 50

Figura 4-29 Aviso error en la latitud. Fuente: Propia.



Datos del buque

Unidad:
Patiño

Latitud:
42,36197

Longitud:
-10,77968

! El valor debe ser superior o igual a -10

Figura 4-30 Aviso error en la longitud. Fuente: Propia.

De la misma forma, en caso de introducir una fecha incorrecta, es decir, una fecha que no corresponde con ninguna de las almacenadas en la base de datos, el error es el mismo, no es posible calcular el valor mínimo asociado a ningún hexágono ya que estos no existen. En este caso la página web redirige al usuario a otra página de error mostrando el resultado que aparece en la Figura 4-31.



Datos del buque

Unidad:
F-102

Latitud:
42.36197

Longitud:
8.80011

Hora (2023-02-07T23:00:00+01):
2023-02-26T23:00:00+01

Radio de acción (km):
3

Resolución malla H3 (7-10):
9

ValueError at /todo/
min() arg is an empty sequence
Request Method: POST
Request URL: http://127.0.0.1:8000/todo/
C:\Users\Pablo Garcia Valera\mysite\personal\views.py, line 107, in todo
107. min_value = min(hex_data.values())
Local vars

Figura 4-31 Error en la consulta a los datos. Fuente: Propia.

Estos errores pueden solucionarse ampliando el catálogo de puntos almacenados en la base de datos, así como aumentar la frecuencia con la que los datos se actualizan de manera que sea posible disponer de un modelo que ofrezca una cobertura mayor. Para poder llevar a cabo estas mejoras se precisa de ordenadores con capacidad suficiente para procesar una gran cantidad de datos, así como servidores capaces de almacenar y manipular grandes flujos de datos, lo que se conoce como *big data*.

El caso del radio de acción, el problema surge cuando no existen datos a la distancia indicada. En estos casos el modelo seleccionado únicamente devolverá aquellos datos que posea, por lo que, aunque se indiquen radios de acción elevados, se representarán únicamente aquellos existentes en la base de datos, quedando una representación final como la representada en la Figura 4-32, quedando una línea recta que indica donde el modelo digital no dispone de más datos.

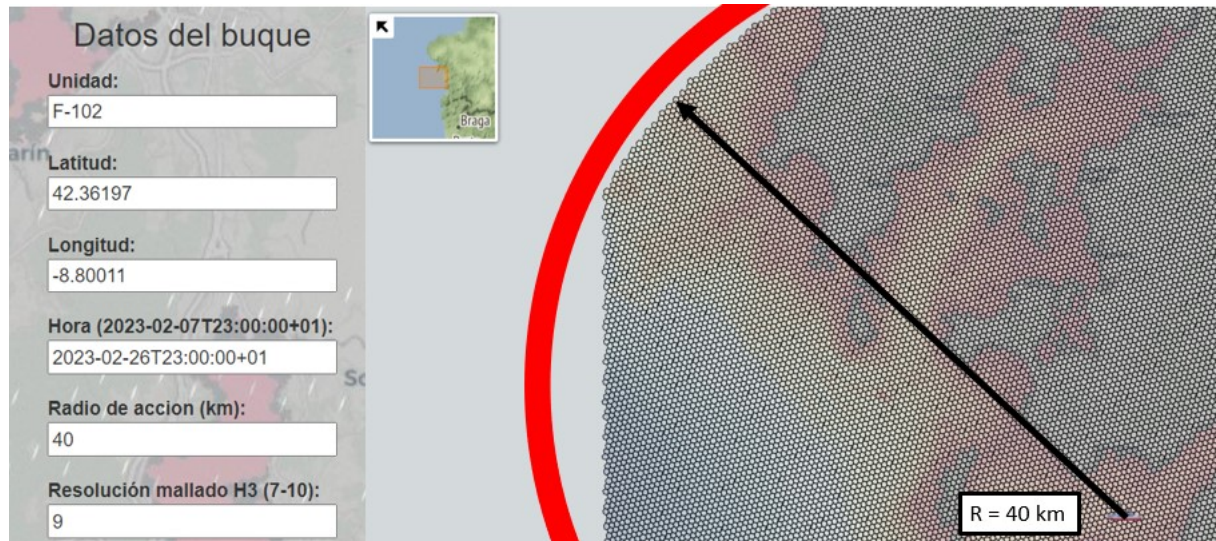


Figura 4-32 Error radio acción. Fuente: Propia.

Para poder solucionar este tipo de problemas, se hará uso de la validación que se viene haciendo en los campos anteriores. De esta manera se han fijado unos valores máximos y mínimos de radio de acción a la hora de seleccionar el valor del radio de acción, por lo que si se introducen valores excesivamente altos y que estén fuera de este rango, obtendremos un valor tal que Figura 4-33.

Figura 4-33 Error en el establecimiento del radio de acción. Fuente: Propia.

Finalmente, la resolución del mallado juega un papel importante a la hora de la representación de los datos. De esta forma y como se explicó en apartados anteriores la resolución del mallado se ha estimado para diferentes buques, indicando que cuanto mayor sea el tamaño del mallado, menor será el tamaño de la celda y por consiguiente mayor la precisión. Para evitar problemas con la resolución, estos valores se han acotado, indicando los valores permitidos en la parte superior. De esta forma en caso de introducir valores erróneos, se obtendrá un mensaje tal y como aparece en la

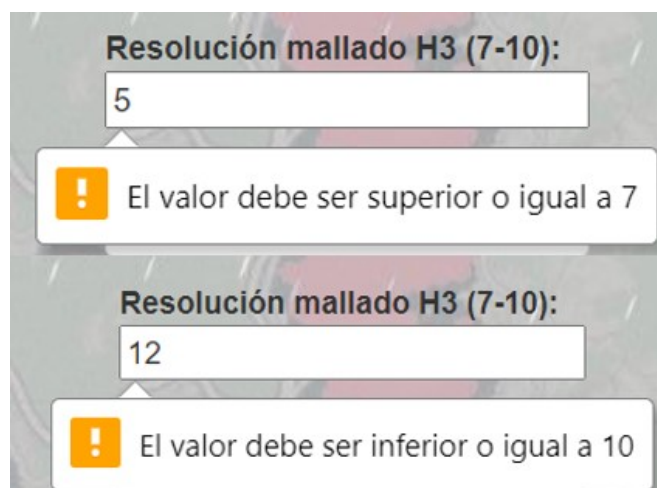


Figura 4-34 Error de formato en la resolución del mallado. Fuente: Propia.

Otro de los problemas que se consigue solucionar mediante la acotación de la resolución y el radio de acción es la de generar archivos html demasiado pesados y que estos no puedan ser representados en la página web o que incluso estos no puedan ser enviados al buque.

El valor que se representa en cada hexágono depende del valor medio asignado a ese hexágono, de esta forma, si en un hexágono no existe un valor para ese hexágono no será representado. A excepción de la batimetría, ya que tal y como se comentó en el apartado Extracción de datos y representación en mapa según el mallado H3, aquellos puntos que correspondan con tierra, su elevación será “NaN”, por lo que al representarlos estos no tendrán ningún color y se verán con un fondo negro transparente.

Existe una posible solución para eliminar esta problemática, y es aquella de eliminar aquellos valores cuya elevación se considera como “NaN”, sin embargo, haciendo esto no es posible identificar aquellos valores que no han sido filtrados, por lo que para acceder a la información sería necesario introducir todas las coordenadas que se quieren representar. En otras palabras, al descargar el archivo CSV de EMODnet, este archivo permite almacenar todos los puntos identificados según sean latitud, longitud o elevación, mientras que, al filtrarlos, esta clasificación se pierde, haciendo que el proceso de almacenaje y posterior representación sea complejo. Además, la clasificación no atiende a criterios ya que cada valor identifica al de la fila siguiente, dando lugar a errores como el de la Figura 4-35.

```
_id: ObjectId('63e5099c5eb84a735fc065a0')
latitude: "42.045312499992455"
longitude: "-9.199479166661485"
elevation: "-143.6"

_id: ObjectId('6405c8a27426c8440dae0c96')
42.045312499992455: 42.045312499992455
-9.21406249999483: -9.213020833328162
-145.8: -145.6
```

Figura 4-35 Diferentes almacenamientos. Fuente: Propia.

En caso de que no se trate de la batimetría y sea cualquier otro valor el que se necesite representar, si no se dispone de la información de ese punto, y por lo tanto existen hexágonos dentro del radio de acción cuyo valor medio no exista (el valor “NaN” en este caso no existe), estos hexágonos no se representarán. Tal y como aparece en la Figura 2-8 y como se menciona para los casos donde el rango de acción es elevado en comparación con la cantidad de datos de la que dispone la base de datos. si la resolución es demasiado pequeña puede darse el caso de que se produzcan huecos dentro del mallado. De esta forma, aunque la precisión buscada sea mayor, existirá una pérdida de información que inutilizará el modelo.

Pudiendo observar en la Figura 4-36, que se están produciendo varios errores simultáneamente, en primer lugar, el error tras haber seleccionado un rango de acción demasiado alto. Y eligiendo una resolución tan elevada que impida visualizar correctamente la información, perdiendo utilidad el modelo.

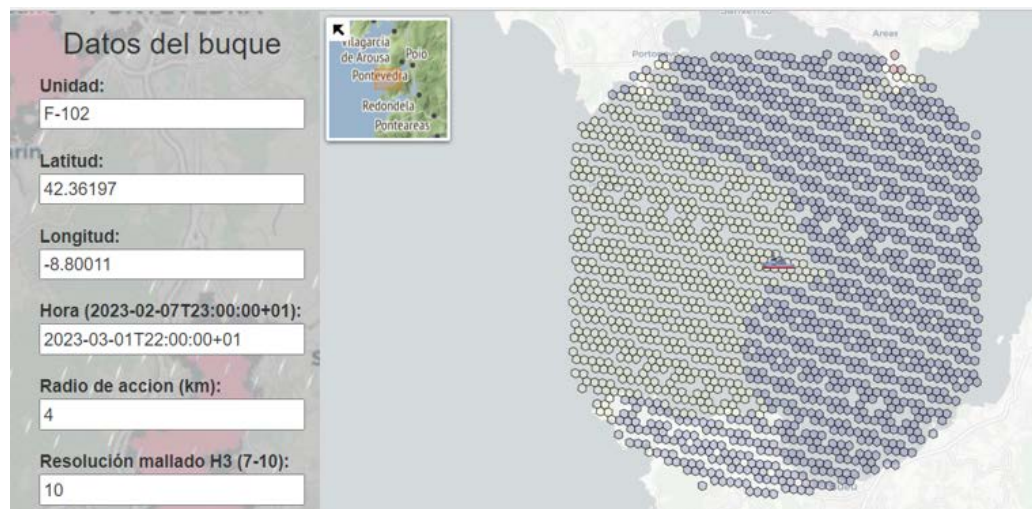


Figura 4-36 Error mallado. Fuente: Propia.

En último lugar se encuentran los errores de precisión, que son aquellos errores que dependen de la localización y posicionamiento indicados al modelo. Debido a que el método de posicionamiento de la biblioteca de folium en Python, así como la descarga de información de MeteoGalicia utilizan precisiones diferentes. Es decir, en algunos casos la representación de datos meteorológicos en el mapa puede estar desfasada, dando valores a hexágonos que no deberían y viceversa.

Esto se debe a que cuando se busca descargar la información, MeteoGalicia es capaz de ofrecer una precisión determinada, almacenando los datos según las especificaciones de precisión del usuario reflejadas en el fichero en el que se indican las coordenadas de los puntos donde se descargarán los valores de los diferentes fenómenos meteorológicos. Fichero que en este caso ha sido generado por EMODnet, dando una precisión muy elevada a los diferentes nodos del mallado.

Sin embargo, folium no es capaz de resolver esa precisión y por lo tanto redondea ciertos valores de coordenadas. Como resultado algunos de los valores para un punto pueden llegar a cambiar de posición, generando una representación errónea de los datos.

Este error es posible solucionarlo de varias maneras. En primer lugar, aumentando el número de nodos del mallado de tal forma que sea posible aproximar de mejor manera las coordenadas y por lo tanto utilizar mallados más exactos para representar la información. Al mismo tiempo es posible solventar este error eligiendo un valor de rango de acción que elimine estos valores de tal manera que se representen únicamente aquellos que se encuentran sobre tierra.

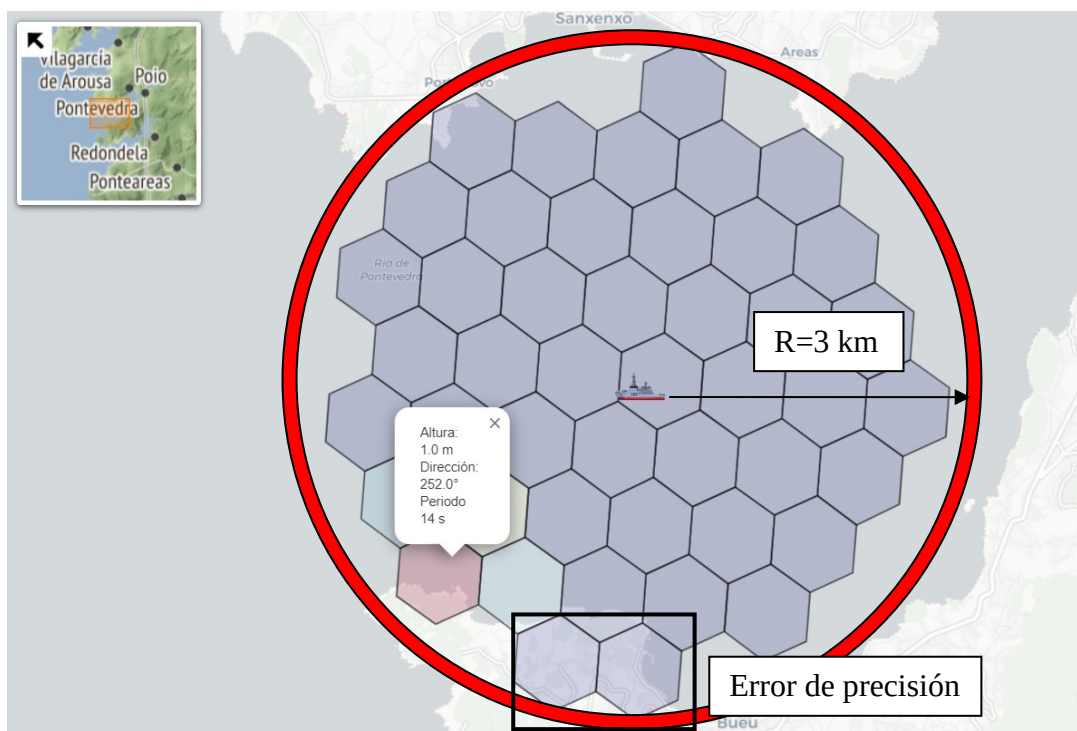


Figura 4-37 Oleaje. Fuente: Propia.

Como se puede comprobar en la imagen anterior y de acuerdo con lo mencionado en cuanto al error de precisión, existe la posibilidad de realizar un mallado de control de área local. Esta herramienta tiene una gran ventaja, pero al mismo tiempo presenta un inconveniente. El código implementado se encarga de asignar los diferentes puntos a cada hexágono dependiendo del tamaño de la malla y posteriormente calcula el valor medio dentro de cada hexágono. Esta acción la realiza sin importar el número de datos que posea cada hexágono y sin importar las coordenadas específicas dentro del hexágono.

Como resultado puede dar lugar a errores de precisión, tal y como aparece en la Figura 4-37, puede darse el caso en que alguno de los hexágonos que aparentemente se encuentran sobre tierra contenga un punto sobre la superficie del agua. Por lo que el valor medio calculado sobre un único hexágono se obtendrá únicamente de ese valor. De esta forma el color y por tanto el valor que se asume dentro del hexágono no es preciso. La solución en caso de que se produzcan este tipo de errores es la de seleccionar una resolución para el mallado H3 más precisa.

La principal diferencia entre los mallados H3 es el área de los diferentes hexágonos, de esta manera, aquellos hexágonos más pequeños, a pesar de tener un menor número de datos en su interior proporcionarían un mapa final con mayor precisión. Por esto es necesario llevar a cabo un balance entre la selección de la resolución del mallado y la precisión en la representación de los datos.

5 CONCLUSIONES Y LÍNEAS FUTURAS

5.1 Conclusiones

Es algo irrefutable que la cuarta revolución industrial es un proceso que ya ha comenzado, a la vez que los avances de la tecnología posibilitan un conocimiento instantáneo de cualquier parámetro medible del entorno y, de su conocimiento futuro. Con el fin de poder aprovechar esta capacidad al máximo es de vital importancia colocarse en la vanguardia del avance tecnológico buscando un desarrollo que conduzca a las organizaciones a desarrollar sistemas capaces de manejar este tipo de innovaciones.

Una de las principales conclusiones que derivan del presente proyecto es la necesidad de utilizar equipos capaces de procesar grandes flujos de datos (*Big Data*), al mismo tiempo que implementar redes capaces de conectar y soportar estos equipos. Ejemplo de esto pueden ser los ordenadores cuánticos, los cuales son capaces de procesar una gran cantidad de datos sin presentar problemas como aquellos equipos ordinarios empleados en el presente trabajo.

Otra de las conclusiones a las que se ha llegado con la creación del siguiente proyecto es la importancia de tener acceso a la información en cualquier lugar de manera precisa y en vivo, es decir, monitorizando cualquier elemento en tiempo real. Al mismo tiempo de disponer de una plataforma capaz de ofrecer una serie de datos en tiempo real dentro del mismo interfaz sin tener que cambiar de página.

Al mismo tiempo es de vital importancia el peso de los archivos generados. Durante la fase de prueba del presente trabajo se han obtenido diferentes documentos con diferente tamaño de almacenamiento y se ha visto como aquellos documentos menos pesados tardan menos en cargar. Este aspecto se torna de gran importancia a la hora de establecer comunicaciones con una unidad desplegada, la cual depende de la calidad de las comunicaciones. De esta manera, con el modelo presentado se ha conseguido reducir el tamaño de los archivos consiguiendo unas comunicaciones más fluidas.

En cuanto a los objetivos establecidos en Objetivos, al inicio de la presente memoria, el primer objetivo ha sido completado satisfactoriamente. En el presente trabajo se han diseñado una serie de códigos capaces de establecer conexión con bases de datos y las colecciones existentes en estos o de crear aquellas colecciones necesarias. De tal manera que aquella información que se desee almacenar en la base de datos se almacenará de forma automática sin necesidad de introducir los datos manualmente por el usuario.

Para la consecución de este objetivo, el estudio de bases de datos, así como la utilización de la API proporcionada por MeteoGalicia han sido fundamentales, posibilitando la automatización del proceso. Además, la posibilidad de disponer de una fuente de puntos como EMODnet permite disponer de mallados de buena calidad y precisión de manera rápida y sencilla, obteniendo además la batimetría asociada a esos puntos.

El segundo objetivo ha sido alcanzado satisfactoriamente al igual que el primero. Para el cumplimiento de este objetivo, la forma de realizar la consulta a la información ha tenido un papel vital, permitiendo una organización de los datos fundamental para evitar errores que impidiesen la exactitud de los datos. De la misma forma separar los datos en diferentes listas a la hora de manipular la información ha ayudado a la claridad de la interpretación e integración de los datos en el mismo modelo.

Finalmente, como objetivo final se planteaba la implementación de un modelo para las Rías Baixas, haciendo tangente la creación de un Gemelo Digital Oceanográfico. Este objetivo ha conseguido satisfacer los requisitos mínimos, siendo un modelo de demostración debido a las limitaciones encontradas por el material empleado.

Esto deriva de que las principales complicaciones encontradas durante el desarrollo del presente trabajo se han debido de la gran cantidad de datos a manejar. Esto se ha visto reflejado debido a la escasa capacidad de procesamiento y almacenamiento de los equipos empleados. El principal problema se traduce en la imposibilidad de descargar datos capaces de satisfacer las necesidades de información que un GDO necesita para un correcto funcionamiento. Teniendo solución mediante la utilización de equipamiento adecuado como los ordenadores cuánticos.

Para concluir me gustaría añadir a título personal que con la realización de este trabajo he descubierto el gran abanico de posibilidades que la tecnología es capaz de brindar a la Armada. Así como la importancia táctica del conocimiento en tiempo real del entorno marino. De la misma forma, me he dado cuenta de la importancia que tiene el proyecto actual de la Armada con el gemelo digital de las futuras F-110 y el proyecto S-101 del Instituto Hidrográfico de la Marina.

5.2 Líneas futuras

Una vez conseguido implementar un modelo capaz de monitorizar en tiempo real y con posibilidad de elaborar predicciones es importante tener en cuenta que este proyecto está siendo investigado por parte del Instituto Hidrográfico de la Marina (IHM). Este proyecto cuenta con la colaboración de marinas como la canadiense y la americana. Inicialmente el proyecto se divide en cuatro fases, encontrándose actualmente en la última de ellas. Siendo el orden de estas la introducción de los datos, la geolocalización de estos, la integración de diferentes datos y por último la creación del producto final. [44]

Así mismo es posible disponer de varios niveles, portuarios, costeros y general. Este proyecto estudia además como pueden afectar los diferentes fenómenos meteorológicos a magnitudes como la batimetría, de tal manera que sea posible calcular en tiempo real la variación de esta. [44]

Los resultados obtenidos por el IHM son similares a los obtenidos en el presente proyecto y confirman que, teniendo un sistema que proporcione los datos en cantidad y calidad necesarios, es posible realizar unos modelos que realmente sean capaces de proporcionar información relevante [44].

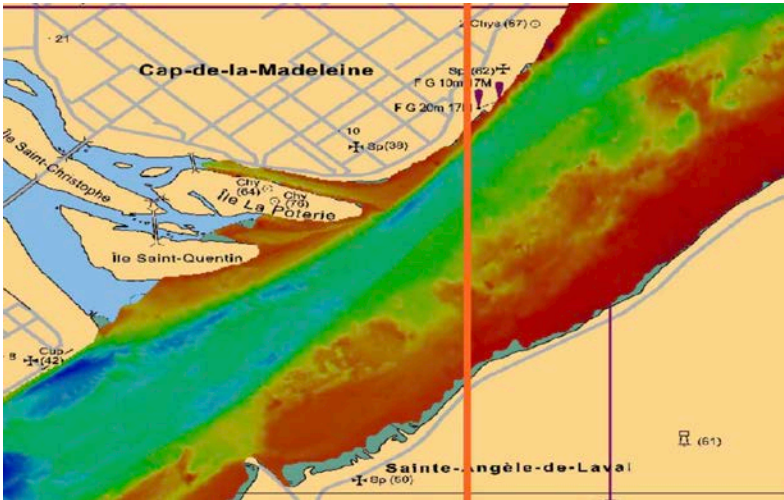


Figura 5-1 Ejemplo batimetría S-100. Fuente: [45].

Estos son los modelos que se encuentra desarrollando por parte del IHM, así como el estado en el que se encuentran, el grupo de trabajo en el que se encuentran dentro del Organización Hidrográfica Internacional (OIH), y la sección dentro del IHM en la que se enmarca el proyecto.

Núm.	Especificación de producto	Estado	Grupo Trabajo de la OHI	Sección IHM
S-102	Bathymetric Surface	Abril 2012	TSMAD	Hidrografía
S-101	Electronic Navigational Chart	4ª Fase	TSMAD	Cartografía
S-10X	Inland ENC	Proyecto	TSMAD	Cartografía
S-10X	Magnetic Variation information	Proyecto	HSSC	Cartografía
S-10X	ETSI Ice	Proyecto	TSMAD	Cartografía
S-10X	Auxiliary Information Layer Integration to ENC	Proyecto	TSMAD DIPWG	Cartografía
S-10X	Maritime Boundary Data	Proyecto	TSMAD	Cartografía
S-10X	Route Information	Proyecto	TSMAD	Cartografía
S-10X	Additional Maritime layer	Proyecto	----	Cartografía
S-10X	Potentially Polluting Marine Sites	Proyecto	MSDIWG	Oceanografía
S-10X	Ocean Forecasts	Proyecto	TSMAD	Oceanografía
S-10X	Surface currents	Proyecto	SCWG	Oceanografía
S-10X	Tidal data	Proyecto	TWLWG	Oceanografía
S-10X	Dynamic tides in ECDIS	Proyecto	TWLWG	Oceanografía
S-10X	Transmission of real-time tidal data	Proyecto	TWLWG	Oceanografía
S-10X	Marine Protected Area.	Proyecto	SNPWG	Cartografía
S-10X	Digital Routing Guide Information	Proyecto	SNPWG	Navegación
S-10X	Nautical Information	Proyecto	SNPWG	Navegación
S-10X	Notice of Arrival and Pilot Requests	Proyecto	TSMAD	Navegación

Tabla 5-1 Proyectos del S-10X. Fuente: [44].

Finalmente se proponen una serie de líneas futuras que permitirán continuar con el presente trabajo y aplicar mejoras de este.

- Utilización de equipos que permitan la descarga y el posterior manejo de los datos de manera que sea posible ampliar el área de acción del modelo presentado. Incrementando no solo el radio de acción sino el número de nodos en el mallado con el fin de conseguir un mejor rendimiento.
- Integración de nuevas fuentes de datos que permitan al usuario elaborar diferentes mapas con mallados en los que se represente la información, de tal manera que sea posible comparar las distintas fuentes y obtener resultados más precisos.
- Estudiar las mejoras que supondría utilizar servidores en local en vez de bases de datos en línea de tal manera que sea posible agilizar los procesos de comunicación entre el usuario y los servidores. Al mismo tiempo que se mantiene un mayor grado de confidencialidad debido a la finalidad del modelo.
- Finalmente, en lo que respecta a la página web, una de las mejoras a realizar sería la de crear un modelo que permita la verificación del usuario. Esto es, enviar un mensaje al correo electrónico proporcionado por el usuario y que desde ese mensaje pueda activar la cuenta. De esta manera será posible verificar la identidad de los posibles usuarios.

A demás de emplear el modelo de Gemelo Digital Oceanográfico, se podría estudiar la posibilidad de emplear “*machine learning*” para realizar una serie de funciones entre las que se encuentran, el análisis de los datos representados de tal forma que sea posible recomendar al usuario cual es la derrota correcta que seguir en función de los valores límite establecidos. Es decir, elaborar una serie de algoritmos que sean capaces de avisar al usuario cuando los valores de los diferentes factores meteorológicos superen el umbral establecido y por lo tanto supongan un peligro para la navegación.

Al mismo tiempo, utilizar esta tecnología para elaborar “*Patterns of life*” (POL) de los diferentes buques, de tal manera que, en un futuro, la máquina sea capaz de recomendar de manera autónoma cuales son estos umbrales y el proceso quede mucho más automatizado.

6 BIBLIOGRAFÍA

En esta sección figuran todas las referencias que se han utilizado en la realización del presente trabajo:

- [1] A. J. E. M. d. I. A. AJEMA, «Lineas Generales de la Armada 2022,» Ministerio de Defensa, Madrid, 2022.
- [2] M. d. Defensa, «Estrategia de Tecnologia e Innovacion para la Defensa,» 2020.
- [3] Ministerio de Defensa , «Jornada tecnológica: simulación en el ámbito de la Defensa,» 2018.
- [4] «Redshift by Autodesk; Gemelo digital: el modelo inteligente de datos, futuro de la edificación,» Autodesk, [En línea]. Available: <https://redshift.autodesk.es/gemelo-digital/>. [Último acceso: 30 12 2022].
- [5] C. d. B. Peón, «Los gemelos digitales en la industria 4.0,» Facultad de Ciencias Económicas y Empresariales, Universidad de Valladolid, Valladolid, 2019.
- [6] B. P. Enrique, «Prueba de concepto de un modelo de sistema de información para su implantación en un gemelo digital oceanográfico.,» Centro Universitario de la Defensa, Pontevedra, 2021.
- [7] Cesar de Pradaa, Santos Galán-Casadoc, Jose L. Pitarchd, Daniel Sarabiae, Anibal Galána, Gloria Gutiérrez, «Gemelos digitales en la industria de procesos,» Revista Iberoamericana de Automatica e Informatica Industrial, 2022.
- [8] M. V. Chiquito, J. C. G. Plua, M. B. Chong y C. B. Chong, «Gemelos digitales y su evolución en la industria (pg 300-308),» Saberes del Conocimiento, 2020.
- [9] M. Singh, E. Fuenmayor, E. P. Hinchy, Y. Qiao, N. Murray y D. Devine, «Digital Twin: Origin to Future,» MDPI, 2021.
- [10] Navantia, «Guia de orientacion sobre Gemelo Digital».
- [11] P. García González, «Gemelo Digital: Concepto y Aplicación en la Armada Española.».
- [12] «A tale of two oceans: Scientists are building digital twins of the ocean,» European Comision, [En línea]. Available: <https://ec.europa.eu/research-and-innovation/en/horizon-magazine/tale-two-oceans-scientists-are-building-digital-twins-ocean>. [Último acceso: 30 12 2022].

- [13] MERCATOR OCEAN 2022, «DIGITAL TWIN OCEAN,» [En línea]. Available: <https://digitaltwinoccean.mercator-ocean.eu/>. [Último acceso: 30 12 2022].
- [14] O. C. Zienkiewicz, El metodo de los elementos finitos, Reverte.
- [15] A. D. MORCILLO, MÉTODOS DE MALLADO Y ALGORITMOS ADAPTATIVOS EN DOS Y TRES DIMENSIONES PARA LA RESOLUCIÓN DE PROBLEMAS ELECTROMAGNÉTICOS CERRADOS MEDIANTE EL MÉTODO DE LOS ELEMENTOS FINITOS, UNIVERSIDAD POLITÉCNICA DE VALENCIA.
- [16] A. R. D. C. y A. L. F., «MALLADO PARA SIMULACIONES».
- [17] L. M. García-Cuevas González, A. Gil Megías, R. Navarro García y P. M. Quintero Igeño, Mecanica de fluidos computacional: tipos de mallas y calidad del mallado.
- [18] MeteoGalicia, «Guía de la API del MeteoSIX,» 2021.
- [19] I. Brodsky, «Uber,» Engineering, 27 7 2018. [En línea]. Available: <https://www.uber.com/en-ES/blog/h3/>. [Último acceso: 1 1 2023].
- [20] «h3geo.org,» [En línea]. Available: <https://h3geo.org/docs/core-library/overview/>. [Último acceso: 1 1 2023].
- [21] «AEMET,» [En línea]. Available: <https://aemetblog.es/2018/05/26/meteorologia-y-defensa/>. [Último acceso: 5 1 2023].
- [22] Instituto Hidrográfico de la Marina, «Armada Española,» [En línea]. Available: <https://armada.defensa.gob.es/ArmadaPortal/page/Portal/ArmadaEspannola/cienciai1/pr efLang-es/01Presentacion--03mision>. [Último acceso: 5 1 2023].
- [23] Revista Española de Defensa , «Los meteorologos de las fuerzas armadas,» 2015.
- [24] Puertos del Estado, «Puertos del Estado,» [En línea]. Available: <https://www.puertos.es/es-es/nosotros/puertos/Paginas/Nosotros.aspx>. [Último acceso: 5 1 2023].
- [25] EMODnet, «What is EMODnet?,» [En línea]. Available: https://emodnet.ec.europa.eu/en/about_emodnet. [Último acceso: 5 1 2023].
- [26] Universidad de Santiago de Compostela, «Unidad de Observación y Predicción Meteorológica (MeteoGalicia),» [En línea]. Available: <https://www.usc.gal/es/servizos/meteogalicia>. [Último acceso: 5 1 2023].
- [27] R. Peiró. [En línea]. Available: <https://economipedia.com/definiciones/base-de-datos.html>. [Último acceso: 3 1 2023].
- [28] «Oracle España,» [En línea]. Available: <https://www.oracle.com/es/database/what-is-database/>. [Último acceso: 3 1 2023].
- [29] F. P. Sourd, «XML, JSON y el intercambio de información,» Universidad Agraria de la Habana.
- [30] acenswhitepapers, «Bases de datos NoSQL. Qué son y tipos que nos podemos encontrar».

- [31] E. F. B. Gaibor, «Evaluacion de rendimiento de sistemas de almacenamiento clave-valor utilizando reproduccion de cargas de trabajo reales,» Escuela Superior Politecnica del Litoral.
- [32] O. Y. E. Hansel Gracia del Busto, «Bases de datos NoSQL p. 21-33,» Revista Telem@tica.
- [33] E. J. T.-G. M. R. M.-M. S. M. C. Jhonatan W. Durán-Cazar, «Rendimiento de bases de datos,» Ingenius Revista de ciencia y tecnologia, 2019.
- [34] R. G. Duque, «Python para todos».
- [35] I. Challenger-Pérez, Y. Díaz-Ricardo y R. A. Becerra-García, «El lenguaje de programación Python,» Centro de Información y Gestión Tecnológica de Santiago de Cuba, Santiago de Cuba, 2014.
- [36] Universidad de Veracruz, «Manejadores de bases de datos».
- [37] «MongoDB,» [En línea]. Available: <https://www.mongodb.com/databases/mongodb-graph-database>. [Último acceso: 4 1 2023].
- [38] Angel Robledano, «OpenWebinars,» 28 Octubre 2019. [En línea]. Available: <https://openwebinars.net/blog/que-es-mongodb/>. [Último acceso: 28 1 2023].
- [39] «DJANGO,» [En línea]. Available: <https://www.djangoproject.com/>. [Último acceso: 03 03 2023].
- [40] «1000marcas,» 01 06 2022. [En línea]. Available: <https://1000marcas.net/django-logo/>. [Último acceso: 27 2 2023].
- [41] «Bootstrap,» [En línea]. Available: <https://getbootstrap.com/>. [Último acceso: 03 03 2023].
- [42] EMODnet, «EMODnet,» [En línea]. Available: EMODnet Map Viewer (europa.eu). [Último acceso: 14 02 1998].
- [43] «RED HAT,» 8 Mayo 2020. [En línea]. Available: <https://www.redhat.com/es/topics/api/what-is-a-rest-api>. [Último acceso: 28 1 2023].
- [44] F. Y. Guerrero, «Introduccion a las S-100 "Modelo Universal de Datos Hidrográficos",» Instituto Hidrográfico de la Marina, 2013.
- [45] Tokio School, «Tokio School,» 11 02 2022. [En línea]. Available: <https://www.tokioschool.com/noticias/sueldo-programador-python-descubre-su-salario/>. [Último acceso: 07 03 2023].
- [46] Talent.com, «Talent.com,» [En línea]. Available: <https://es.talent.com/salary?job=programador+web#:~:text=El%20salario%20programador%20web%20promedio,hasta%20%E2%82%AC%2033.850%20al%20a%C3%B1o..>. [Último acceso: 07 03 2023].
- [47] R. Caubilla, «HUFFPOST,» 01 09 2022. [En línea]. Available: https://www.huffingtonpost.es/entry/bahia-algeciras-gasolinera-buque-varado-problema-ambiental_es_6310cbf4e4b0dc23bbefc55b.html?outputType=amp. [Último acceso: 07 03 2023].
- [48] «Job Wizards,» [En línea]. Available: <https://job-wizards.com/es/gemelos-digitales-duplicando-el-potencial-de-innovacion/>. [Último acceso: 29 12 2022].

- [49] Weather, Passage, «Passage Weather,» [En línea]. Available: <https://www.passageweather.com/>. [Último acceso: 29 12 2022].
- [50] «tinakoziol.com,» [En línea]. Available: <https://t1nak.github.io/blog/2020/h3intro/>. [Último acceso: 1 1 2023].
- [51] L. R. Yunta, «BASES DE DATOS DOCUMENTALES: ESTRUCTURA Y PRINCIPIOS DE USO,» CINDOC-CSIC.
- [52] J. N. Castillo, J. R. Garcés, M. P. Navas y D. F. J. Segovia, «Base de Datos NoSQL: MongoDB vs. Cassandra en operaciones CRUD,» Revista Publicando, 4 No 11. (1). 2017.
- [53] «MeteoGalicia,» [En línea]. Available: https://www.meteogalicia.gal/modelos/inicio.action?request_locale=es. [Último acceso: 5 1 2023].
- [54] Navantia, «Navantia Sistemas,» [En línea]. Available: <https://www.navantia.es/es/lineas-de-negocio/sistemas/sicp/>. [Último acceso: 5 1 2023].
- [55] «Bootstrap,» [En línea]. Available: <https://getbootstrap.com/docs/5.3/getting-started/download/>. [Último acceso: 2023 03 02].
- [56] N. Mayhew, «Organismo Internacional de Energia Atómica,» 17 01 2019. [En línea]. Available: <https://www.iaea.org/es/newscenter/news/el-oiea-y-el-cambio-climatico-adaptacion-monitorizacion-y-mitigacion>. [Último acceso: 07 03 2023].

ANEXO I: IMPLICACIONES ECONÓMICAS Y AMBIENTALES

A.1.1 Implicaciones económicas

Uno de los fines del presente trabajo es poder desarrollar una aplicación capaz de suplir y mejorar las deficiencias y prestaciones de la demostración diseñada. Para poder llevar a cabo dicho proyecto es importante tener en cuenta los factores económicos del mismo.

Para poder hacer una estimación del coste económico que supondría elaborar dicho proyecto así como el mantenimiento de este es importante tener en cuenta que, la parte más costosa será la de suprimir los errores y llevar a cabo los ajustes necesarios. Esto se debe a que la parte posterior de mantenimiento de servidores, en cuanto a costes directos será la parte más económica.

De esta forma es posible observar que para la creación del presente proyecto se ha llevado a cabo una división de las horas empleadas de manera que sea posible distinguir los gastos que este proyecto puede llegar a suponer.

En primer lugar, se encuentra la fase de formación, esta fase es necesario tenerla en cuenta ya que se ha partido de cero y por lo tanto ha sido necesaria una formación en aspectos técnicos sobre programación en lenguaje de Python. Es importante resaltar que existen cursos gratuitos en diversas plataformas en línea, sin embargo, es importante una correcta formación para poder hacer frente a las diferentes situaciones que se presenten de manera correcta y sin pérdidas de tiempo.

En segundo lugar, fase de programación, siendo esta una de las dos fases principales de la creación del proyecto ocupando el 42,4% del tiempo empleado en el desarrollo del TFG. Esta fase es de especial importancia por la complejidad de esta y las horas que esta lleva asociadas. Así mismo es importante resaltar que esta fase supone un coste variable, ya que es posible que debido a las complejidades de la programación esta fase se alargue en el tiempo. Con el fin de aproximar al máximo posible el coste total del proyecto se tendrá en cuenta que este puede ser llevado a cabo por programadores nóveles, y que por lo tanto no se necesita un conocimiento elevado en el ámbito de la programación.

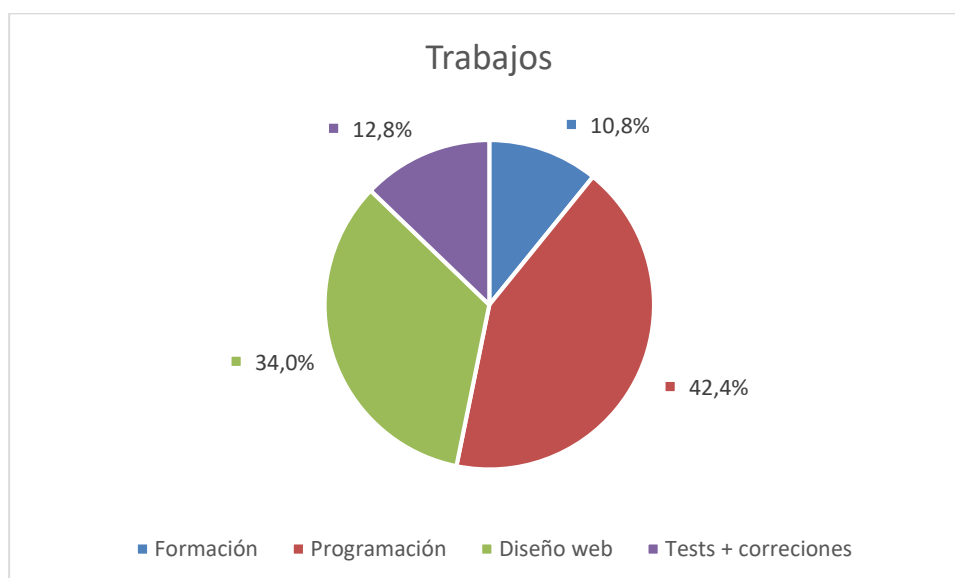


Figura A1 1 Distribución horas. Fuente: Propia.

La siguiente fase engloba el diseño de la página web, siendo el segundo pilar de este proyecto de acuerdo con los criterios mencionados anteriormente. De esta forma, y debido también a la novedad de este campo, supone uno de los costes principales en la construcción del presente proyecto.

Finalmente, la fase de pruebas y corrección de errores, fase de corta duración pero al mismo tiempo de las más importantes. Esta fase tiene su importancia en la puesta a punto de la aplicación, previo a su lanzamiento de cara al público. Además, la correcta realización de esta fase significa el ahorro a futuro de la gestión y corrección de los posibles errores derivados del uso continuado de la aplicación.

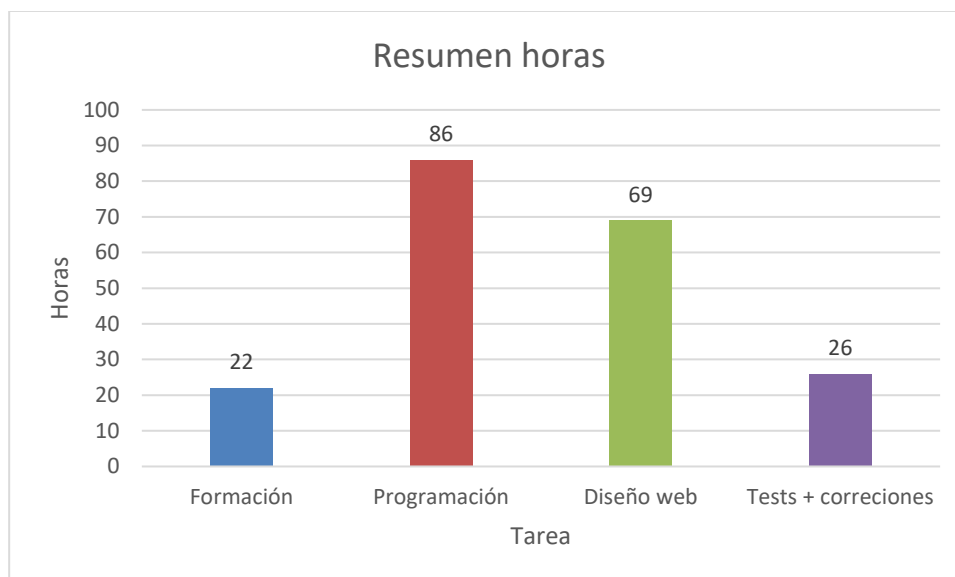


Figura A1 2 Horas totales. Fuente: Propia.

Una vez analizadas las diferentes fases de este proyecto, asignando una importancia relativa según la complejidad de cada fase, es posible llevar a cabo una estimación económica de los costes totales del proyecto. De esta manera, y atendiendo al comportamiento del mercado actual:

Fase	Horas (h)	Costes/Hora (€/h)	Costes totales (€)
Formación	22	15	330€
Programación	86	10,15	873,44
Diseño Web	69	12,44	858,36
Pruebas + Correcciones	26	22,59	587,34
Totales	203	-	2.341,14

Tabla A1 1 Costes elaboración GDO. Fuente: Propia.

Para llevar a cabo estas estimaciones se han tenido en cuenta los salarios asociados a programadores y diseñadores web junior, tal y como se menciona en las referencias [45, 46]. Así mismo se estima que para llevar a cabo la última de las fases será necesario llevar a cabo pruebas finales en las que tanto programadores como diseñadores web tendrán que estar involucrados y por lo tanto se asume que el coste por hora es el asociado a los salarios por hora de ambos trabajadores.

De esta forma es posible estimar que el coste de la creación de una aplicación de estas características ronda aproximadamente los 4990€ sin tener en cuenta los gastos asociados al mantenimiento de los servidores.

Debido a las características del ámbito en el que se pretende desarrollar el TFG, es necesario indicar que cabe la posibilidad de suprimir el coste de mantenimiento de los servidores debido a que estos pueden ser mantenidos por los diferentes servicios informáticos de la Armada. Sin embargo, existen diferentes empresas encargadas de realizar mantenimientos informáticos. De acuerdo con los costes medios de estas empresas los costes de mantenimiento de los servidores podrían oscilar entre:

Tipo servicio	Precio (€/mes)
Básico	30
Completo	50
Premium	70

Tabla A1 2 Costes mantenimiento servidores: Fuente: [48].

Según las especificaciones de cada tipo de servicio, para el presente proyecto se recomendaría un mantenimiento premium. Esto se debe a que los servidores almacenarán información que debe estar disponible en cualquier momento, además de estar continuamente actualizada. Esto genera un consumo de capacidades del servidor que hacen que una automatización de copias de seguridad, así como visitas periódicas a los servidores para detectar errores se conviertan en necesidades básicas que ayuden a conseguir un rendimiento óptimo del servidor.

A modo de resumen, este proyecto puede clasificar sus costes de diferentes maneras, diferenciando por fases y horas trabajadas. De esta manera la inversión inicial del proyecto, sin tener en cuenta costes asociados, asciende a 4.990€ aproximadamente, a lo que hay que sumar una serie de gastos periódicos de 70 €/mes para un correcto mantenimiento de este.

Como se ha podido comprobar, este proyecto supone una gran inversión inicial, destinando los gastos posteriores a mantenimientos en los servidores. Es importante resaltar que es posible que debido al uso de la aplicación se descubran posibles fallos en el sistema que haya que subsanar, lo que elevaría el coste en un determinado momento. Sin embargo, estos gastos por considerarse puntuales no han sido tenidos en cuenta en la elaboración del presente proyecto.

Como conclusión derivada de las implicaciones económicas es posible deducir que, mediante el uso de este modelo de gemelo digital, será posible alterar las derrotas o los puntos de fondeo de los buques según las diferentes condiciones meteorológicas. Esto supone un ahorro en cuanto a combustible y piezas para repuestos de las diferentes unidades de la Armada, ya que, conociendo los lugares con condiciones adversas, será posible evitarlas y por lo tanto someter al buque a esfuerzos menores, evitando de esta forma esfuerzos excesivos en los motores, así como en las diferentes piezas que pueden derivar en rotura.

A 1.2 Implicaciones medioambientales

A pesar de que el propósito principal del presente trabajo sea el de elaborar un modelo que permita predecir la meteorología a futuro y en tiempo real y de esta manera asistir a los buques en sus navegaciones, existen usos derivados que, con algunas modificaciones al proyecto pueden convertir en una herramienta útil al gemelo digital oceanográfico.

En primer lugar, es posible aumentar el número de colecciones, al mismo tiempo que modificar el código para llevar un registro del histórico meteorológico de las diferentes áreas de interés. De esta manera será posible monitorizar el cambio climático en tiempo real y evaluar cuales son las consecuencias de los cambios sucedidos. De esta manera será posible, en base a las predicciones realizadas con el mismo modelo, deducir cuales pueden llegar a ser las posibles consecuencias de este cambio. Al mismo tiempo, será posible establecer una serie de medidas que permitan corregir o impedir algunos efectos negativos derivados del cambio climático antes de que estos se lleguen a producir.



Figura A1 3 Ramas acción contra el cambio climático. Fuente: [49].

De la misma forma este modelo aplicado a la navegación civil al igual que la militar puede ser empleado para la detección de condiciones meteorológicas que puedan suponer un peligro para la navegación y pongan en peligro al propio buque. De esta manera, no solo se reduce la contaminación derivada por las emisiones de los motores al estar sometidos a menos esfuerzos y por lo tanto necesitar de menos combustible y de menor tiempo de uso, sino que es posible evitar posibles accidentes que deriven en catástrofes naturales como fue el caso del “Prestige”.



Figura A1 4 Buque vertiendo líquidos contaminantes al mar. Fuente: [47].

Es por esto por lo que se puede concluir que este TFG aporta, además de un sistema de simulación que permitirá planificar la derrotas para adecuarlas a las condiciones de seguridad en la navegación, la capacidad de prever cuáles serán las condiciones meteorológicas que evitar para impedir desastres como el que aparece en la

Figura A1 4. Además, aporta la capacidad de elaborar una serie de históricos y predicciones que permitirán monitorizar de manera correcta no solo la evolución del ecosistema marino sino también de cuáles pueden ser los efectos que se producirán en función de las condiciones meteorológicas existentes en un área determinada. Permitiendo a los diferentes organismos internacionales tomar acción y solucionando de manera correcta los efectos negativos derivados del cambio climático.

ANEXO II: REFLEXIONES ÉTICAS Y SOCIALES

Es importante tener en cuenta que en el desarrollo del TFG se ha llevado a cabo la creación de una página web en la que es necesario el registro del usuario previamente a la autorización de los modelos disponibles. Así mismo, la base de datos en MongoDB cuenta con la posibilidad de autorizar a los usuarios a modificar la información en ella presente o simplemente leerla. Para lo que es necesario un usuario con contraseña, además de autorizar o denegar el acceso a direcciones IP específicas.

El conocimiento de estos datos implica una responsabilidad social y ética en cuanto al tratamiento de datos personales, además de estar regulado por la “*Ley Orgánica 3/2018 de 5 de Protección de Datos Personales*” en la que se regulan todos los aspecto relacionados con el tratamiento de dicha información.

De la misma forma es importante resaltar que una de las características principales del presente trabajo es la de almacenamiento de información que podrá ser consultada no solo por usuarios militares sino también por usuarios civiles, los cuales basarán ciertos aspectos de la seguridad a la navegación en sus buques en los resultados derivados de las búsquedas realizadas con los modelos creados.

Es por esto por lo que el tratamiento de la información conlleva una responsabilidad para asegurar que la información almacenada en la base de datos es correcta y no ha sido manipulada. Ya que, si esto sucede, no solo se está poniendo en peligro el ecosistema ante posibles catástrofes naturales, sino que también se ponen en riesgo vidas humanas en la mar.

Por estas razones es importante actuar de una forma éticamente correcta con respecto a la información presentada al usuario final, de tal manera que exista una transparencia en cuanto a los responsables del tratamiento de la información, así como la responsabilidad asociada a la veracidad de la misma.

Este TFG supone un adelanto y una mejora para la sociedad marítima en cuanto a seguridad, capaz de asegurar una precisión lo suficientemente elevada como para que sea fiable y al mismo tiempo sirva de herramienta en diferentes ámbitos. No solo en la navegación, sino también para la pesca, en cuanto al análisis de la salinidad o temperatura del agua, y por último al control del cambio climático.

Finalmente, el TFG de ámbito de telecomunicaciones, tiene asociadas una serie de responsabilidades derivadas principalmente del tratamiento de la información. A su vez, el buen uso de esta plataforma puede ser una gran ayuda para el mundo marítimo, aportando conocimiento del entorno, históricos que permitan la monitorización del medio y predicciones a futuro que ayuden con el conocimiento y la planificación de las actividades necesarias para evitar catástrofes o mejorar el aprovechamiento de los océanos.

ANEXO III: INSTALACIONES

En el siguiente apartado se recogen los diferentes procedimientos seguidos para poder llevar a cabo una correcta instalación de los diferentes sistemas y programas necesarios para el desarrollo del trabajo. Así como ciertas especificaciones a la hora de llevar a cabo la instalación de estos.

A.3.1 Instalaciones

Debido a las facilidades ofrecidas por el sistema operativo Linux a la hora de llevar a cabo tareas de programación, así como establecer relaciones entre los diferentes programas se ha optado por la creación de un entorno virtual que soporte este sistema operativo.

Para poder llevar a cabo este proceso en el ordenador personal de cualquier usuario es necesario escoger previamente una máquina virtual que permita tener un entorno de simulación que soporte diversos sistemas operativos y los almacene como si fuesen ordenadores diferentes e independientes.

A.3.1.1 VirtualBox

En este caso se ha optado por utilizar VirtualBox, de esta forma para su instalación es necesario dirigirse a la web oficial (<https://www.virtualbox.org/wiki/Downloads>), concretamente a la parte de descargas (“Downloads”), donde tendremos acceso a las diferentes versiones creadas en los últimos años, así como a la versión más reciente, la cual está indicada en un apartado separada de las demás.

Una vez finalizada la descarga se abrirá el terminal de VirtualBox, podremos seleccionar las diferentes opciones, así como la posibilidad de configurar máquinas virtuales ya existentes.

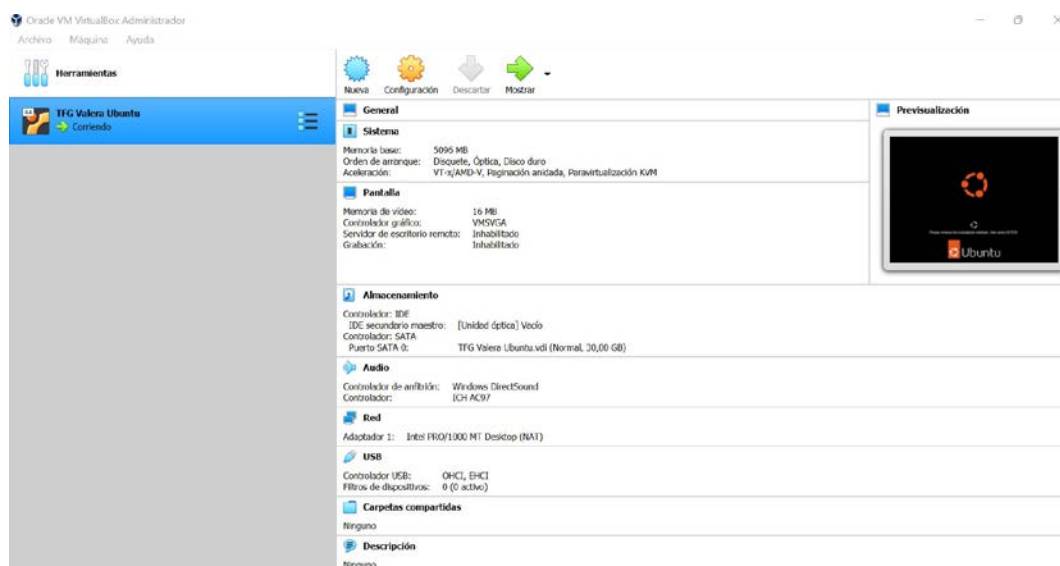


Figura A3 1 Interfaz VirtualBox Fuente: Propia.

Una vez instalado VirtualBox, se procede a la instalación del sistema operativo Linux. Para ello el primer paso es elegir el sistema operativo basado en Linux, este será Ubuntu. Para poder realizar la instalación correctamente es necesario descargar la versión “Ubuntu 22.04 LTS”, el termino LTS hace referencia a “Long-Term Support”, lo que garantiza un mantenimiento a largo plazo (aproximadamente 5 años) de esta versión de Ubuntu, esta descarga se puede llevar a cabo desde la página oficial de Ubuntu (<https://ubuntu.com/download/desktop>)

Una vez finalizada la descarga, es necesario abrir la interfaz de VirtualBox, y seleccionar dentro de la barra superior de herramientas el icono donde se especifica la palabra “Nueva”, de tal manera que sea posible empezar la creación del nuevo sistema operativo independiente.

Una vez abierto VirtualBox solicita cierta información para la creación de la máquina virtual, en primer lugar, se requiere un nombre para poder llevar a cabo una correcta identificación, al mismo tiempo que exige que se seleccione el sistema operativo que se desea. En este caso el sistema operativo seleccionado ha sido Ubuntu (64-bit). Una vez establecido el sistema operativo es importante establecer otros datos importantes para el funcionamiento de la máquina, entre estos valores se encuentran la memoria RAM que se le pretende dar, en este caso la memoria RAM seleccionada es de 5096 MB, y la memoria o capacidad del disco duro, que en este caso es de 30GB, estos valores se han asignado con el objetivo de evitar futuras complicaciones debido a la gran cantidad de información que se maneja en el sistema operativo.

El siguiente paso es darle al icono donde aparece “Iniciar”, y esperar a que aparezca el interfaz inicial donde el primer paso es indicarle al programa que queremos instalar Ubuntu. A continuación, aparece una ventana donde se da la opción de probar el sistema operativo para ver si es el adecuado o instalarlo directamente. En el caso de este trabajose procede a instalar el sistema operativo directamente. Para ello se dejan todos los valores seleccionados que aparecen por defecto, y tras una instalación inicial es sistema operativo queda instalado y configurado en el sistema operativo deseado dentro del programa VirtualBox, permitiendo su almacenamiento como si fuese un ordenador completamente diferente.

A.3.1.2 Visual Studio Code y Python

El lenguaje seleccionado para la programación que se ha seleccionado es Python. Para poder llevar a cabo una correcta escritura y poder identificar con mayor facilidad los diferentes errores o partes del código se ha optado por la descarga e implementación de Visual Studio Code (VSC). Una vez abierto el sistema operativo configurado en la máquina virtual, se accede a la tienda, donde es posible descargar diferentes aplicaciones de manera directa.

Una vez dentro, escribimos la palabra “Code” y buscamos el icono relacionado con dicha aplicación, y una vez localizado instalamos el programa en el entorno virtual.

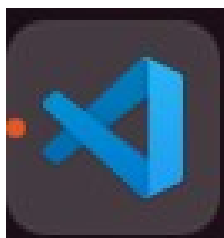


Figura A3 2 Logo Visual Studio Code. Fuente: Propia.

Debido a que estamos utilizando el lenguaje Python y para que VSC sea capaz de interpretar y entender lo que estamos haciendo, así como de ejecutarlo, es importante instalar la extensión necesaria de Python. Para ello en el menú de opciones de la izquierda se selecciona el apartado de extensiones, y en el buscador se realiza la búsqueda de la extensión necesaria, en este caso: Python, y se procede a su instalación.

Una vez llevados a cabo estos pasos, el programa VSC es capaz de entender y ejecutar todos los códigos que se realicen y que sean de utilidad para el desarrollo del trabajo.

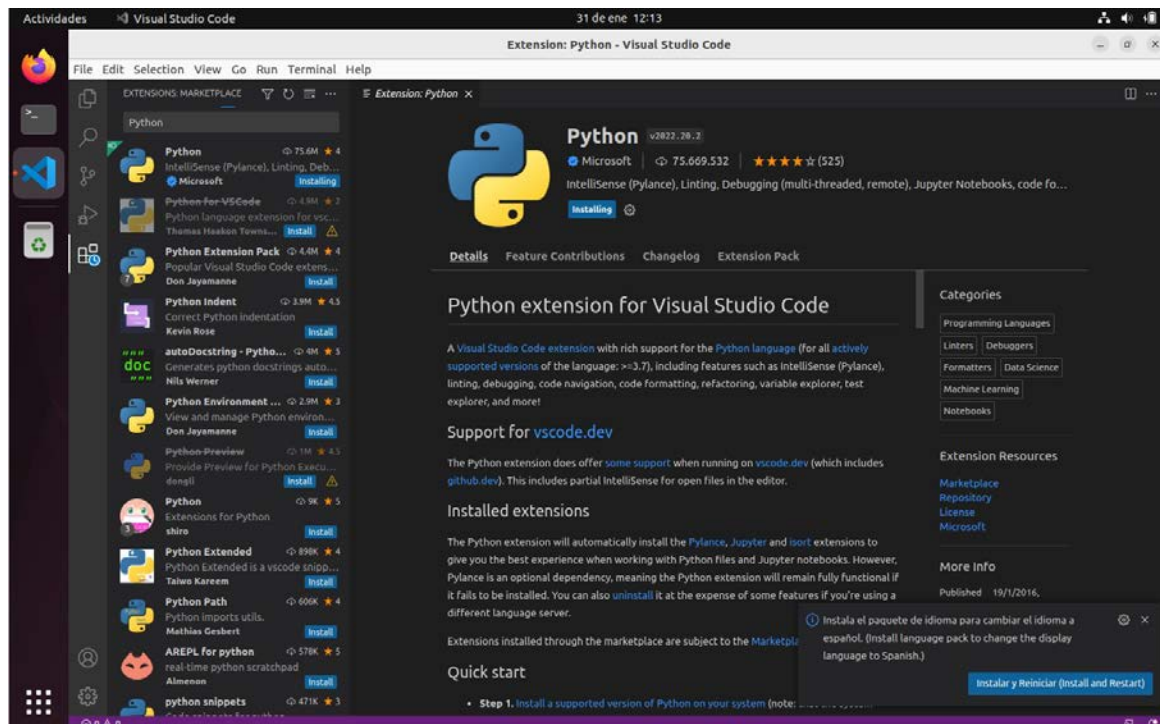


Figura A3 3 Entorno Visual Studio Code. Fuente: Propia.

A.3.1.2 MongoDB

Debido a que es necesaria la implementación de una base de datos para el almacenamiento de toda la información que posteriormente será utilizada por el Gemelo Digital, es necesario crear una base de datos, para ello se ha optado por la instalación de la base de datos MongoDB.

Para la instalación correcta de MongoDB en el entorno de Ubuntu es suficiente con ejecutar de manera individual los códigos de instalación en un terminal, de esta manera no hace falta descargar ningún archivo de internet.

Sin embargo, todavía no está instalado el interfaz que nos permitirá acceder virtualmente a la información, así como la configuración de los diferentes aspectos de la base de datos. Para ello se instalará el paquete Compass de MongoDB. Para ello es necesario únicamente acceder a la sección de descargas de MongoDB (<https://www.mongodb.com/try/download/compass>), y descargar en la máquina virtual la versión adecuada al sistema operativo. Una vez descargado se ejecuta el archivo descargado y de esta manera queda instalada la base de datos MongoDB.

Finalmente, para poder empezar a utilizar la base de datos, es importante ejecutar el programa MongoDB. Para ello en un terminal es necesario ejecutar el comando “Sudo systemctl start mongod”. De esta manera es posible acceder y crear una base de datos específica para el desarrollo del presente trabajo. Desde esta misma interfaz es posible añadir datos a la base de datos de manera manual, sin embargo, es preferible hacerlo través de la herramienta de Python de tal forma que se pueda hacer de manera automática.

Al mismo tiempo es importante configurar los diferentes “Clusters” en la página de bases de datos de MongoDB Atlas

A.3.1.2 DJANGO & BOOTSTRAP

Estas herramientas tienen una instalación sencilla. El primer paso para poder instalar Django es ejecutar en la barra de comandos del ordenador que se esté empleando el siguiente comando:

```
C:\Users\Pablo Garcia Valera>pip install Django==4.1.7
```

Figura A3 4 Instalación Django. Fuente: Propia.

Una vez realizada la acción indicando la versión deseada, únicamente es necesario seguir las instrucciones que aparecen en la página oficial de Django para ejecutar correctamente la creación del proyecto y las diferentes aplicaciones que pueda tener.

En cuanto a Bootstrap, no es necesario descargar ninguna aplicación adicional si no es necesario, para la correcta aplicación del programa únicamente es necesario insertar unas líneas de código en la cabecera y en el pie de página creados en la página web, en el código html. Estas líneas son accesibles desde su página web.

CDN via jsDelivr

Skip the download with [jsDelivr](#) to deliver cached version of Bootstrap's compiled CSS and JS to your project.

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css"
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.js">
```

If you're using our compiled JavaScript and prefer to include Popper separately, add Popper before our JS, via a CDN preferably.

```
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js" int
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.min.js">
```

Figura A3 5 Aplicación Bootstrap. Fuente: [46].

ANEXO IV: CÓDIGOS

A.4.1 Descarga datos

En este primer bloque aparecen los diferentes códigos empleados para la descarga y almacenamiento de los datos en la base de datos de MongoDB.

A.4.1.1 Almacenamiento batimetría en MongoDB

```
import pymongo
import pandas as pd

client =
pymongo.MongoClient("mongodb+srv://*****:*****@cluster0.9cawz8x.mongodb.net/Gemelo
DigitalOceanográfico")
db = client["GemeloDigitalOceanográfico"]
collection = db["Batimetria"]

data = pd.read_csv('BatiRiasBaixas.csv').to_dict('records')
collection.insert_many(data)

client.close()
```

A.4.1.2 Definición clases

```
import math
class MeteoClass(object):
    def __init__(self, name):
        self.nombre = name

    def Temperatura(self):
        pass

    def Precipitaciones(self):
        pass

    def Viento(self):
        pass

    def Humedad(self):
        pass

    def Nubosidad(self):
        pass

    def Presion(self):
        pass
```

```
def Temperatura_Agua(self):  
    pass  
  
def Salinidad(self):  
    pass  
  
def Oleaje(self):  
    pass
```

A.4.1.3 Descarga y almacenamiento de datos en MongoDB

```
import csv  
import json  
import requests  
import pymongo  
import math  
import dns  
from MeteoClass import MeteoClass  
  
class MeteoGalicia():  
    def __init__(self):  
        MeteoClass.__init__(self, "MeteoGalicia")  
  
    def Temperatura(self):  
        with open('BatiRiasBaixas.csv', 'r') as file:  
            reader = csv.reader(file)  
            next(reader)  
            next(reader)  
            for row in reader:  
                LAT, LONG, elevacion = row  
                if not math.isnan(float(elevacion)):  
                    solditems =  
requests.get(f'https://servizos.meteogalicia.gal/apiv4/getNumericForecastInfo?variabl  
es=temperature&coords={LONG},{LAT}&format=application/json&API_KEY=CONDIFENCIAL')  
                    data = solditems.json()  
                    with open(f'Temperatura_{LONG}_{LAT}.json', 'w') as f:  
                        json.dump(data, f)  
                    client =  
pymongo.MongoClient("mongodb+srv://*****:*****@cluster0.9cawz8x.mongodb.net/GemeloDig  
italOceanográfico")  
                    db = client["GemeloDigitalOceanográfico"]  
                    collection = db["Temperatura1"]  
                    collection.insert_one(data)  
  
    def Precipitaciones(self):  
        with open('BatiRiasBaixas.csv', 'r') as file:
```

```
        reader = csv.reader(file)
        next(reader)
        next(reader)
        for row in reader:
            LAT, LONG, elevacion = row
            if not math.isnan(float(elevacion)):
                solditems =
requests.get(f'https://servizos.meteogalicia.gal/apiv4/getNumericForecastInfo?variabl
es=precipitation_amount&coords={LONG},{LAT}&format=application/json&API_KEY=
CONFIDENCIAL')

                data = solditems.json()
                with open(f'Precipitaciones_{LONG}_{LAT}.json', 'w') as f:
                    json.dump(data, f)
                client = pymongo.MongoClient("mongodb+srv://
*****:****@cluster0.9cawz8x.mongodb.net/GemeloDigitalOceanográfico")
                db = client["GemeloDigitalOceanográfico"]
                collection = db["Precipitaciones"]
                collection.insert_one(data)

def Viento(self):
    with open('coordsTemp.csv', 'r') as file:
        reader = csv.reader(file)
        next(reader)
        next(reader)
        for row in reader:
            LAT, LONG, elevacion = row
            if not math.isnan(float(elevacion)):
                solditems =
requests.get(f'https://servizos.meteogalicia.gal/apiv4/getNumericForecastInfo?variabl
es=wind&coords={LONG},{LAT}&format=application/json&API_KEY= CONFIDENCIAL')
                data = solditems.json()
                with open(f'Viento_{LONG}_{LAT}.json', 'w') as f:
                    json.dump(data, f)
                client = pymongo.MongoClient("mongodb+srv://
*****:****@cluster0.9cawz8x.mongodb.net/GemeloDigitalOceanográfico")
                db = client["GemeloDigitalOceanográfico"]
                collection = db["Viento1"]
                collection.insert_one(data)

def Humedad(self):
    with open('BatiRiasBaixas.csv', 'r') as file:
        reader = csv.reader(file)
        next(reader)
        next(reader)
        for row in reader:
            LAT, LONG, elevacion = row
            if not math.isnan(float(elevacion)):
                solditems =
requests.get(f'https://servizos.meteogalicia.gal/apiv4/getNumericForecastInfo?variabl
```

```
es=relative_humidity&coords={LONG},{LAT};&format=application/json&API_KEY=
CONFIDENCIAL')

    data = solditems.json()
    with open(f'Humedad_{LONG}_{LAT}.json', 'w') as f:
        json.dump(data, f)
    client = pymongo.MongoClient("mongodb+srv://
*****:****@cluster0.9cawz8x.mongodb.net/GemeloDigitalOceanográfico")
    db = client["GemeloDigitalOceanográfico"]
    collection = db["Humedad"]
    collection.insert_one(data)

def Nubosidad(self):
    with open('BatiRiasBaixas.csv', 'r') as file:
        reader = csv.reader(file)
        next(reader)
        next(reader)
        for row in reader:
            LAT, LONG, elevacion = row
            if not math.isnan(float(elevacion)):
                solditems =
requests.get(f'https://servizos.meteogalicia.gal/apiv4/getNumericForecastInfo?variabl
es=cloud_area_fraction&coords={LONG},{LAT};&format=application/json&API_KEY=
CONFIDENCIAL')

    data = solditems.json()
    with open(f'Nubosidad_{LONG}_{LAT}.json', 'w') as f:
        json.dump(data, f)
    client = pymongo.MongoClient("mongodb+srv://
*****:****@cluster0.9cawz8x.mongodb.net/GemeloDigitalOceanográfico")
    db = client["GemeloDigitalOceanográfico"]
    collection = db["Nubosidad"]
    collection.insert_one(data)

def Presion(self):
    with open('BatiRiasBaixas.csv', 'r') as file:
        reader = csv.reader(file)
        next(reader)
        next(reader)
        for row in reader:
            LAT, LONG, elevacion = row
            if not math.isnan(float(elevacion)):
                solditems =
requests.get(f'https://servizos.meteogalicia.gal/apiv4/getNumericForecastInfo?variabl
es=air_preassure_at_sea_level&coords={LONG},{LAT};&format=application/json&API_KEY=
CONFIDENCIAL')

    data = solditems.json()
    with open(f'Presion_{LONG}_{LAT}.json', 'w') as f:
        json.dump(data, f)
    client = pymongo.MongoClient("mongodb+srv://
*****:****@cluster0.9cawz8x.mongodb.net/GemeloDigitalOceanográfico")
```

```
db = client["GemeloDigitalOceanográfico"]
collection = db["Presion"]
collection.insert_one(data)

def Temperatura_Agua(self):
    with open('BatiRiasBaixas.csv', 'r') as file:
        reader = csv.reader(file)
        next(reader)
        next(reader)
        for row in reader:
            LAT, LONG, elevacion = row
            if not math.isnan(float(elevacion)):
                solditems =
requests.get(f'https://servizos.meteogalicia.gal/apiv4/getNumericForecastInfo?variabl
es=sea_water_temperature&coords={LONG},{LAT}&format=application/json&API_KEY=
CONFIDENCIAL')

                data = solditems.json()
                with open(f'Temperatura_Agua_{LONG}_{LAT}.json', 'w') as f:
                    json.dump(data, f)
                client = pymongo.MongoClient("mongodb+srv://
*****:*****@cluster0.9cawz8x.mongodb.net/GemeloDigitalOceanográfico")
                db = client["GemeloDigitalOceanográfico"]
                collection = db["Temperatura_Agua"]
                collection.insert_one(data)

def Salinidad(self):
    with open('BatiRiasBaixas.csv', 'r') as file:
        reader = csv.reader(file)
        next(reader)
        next(reader)
        for row in reader:
            LAT, LONG, elevacion = row
            if not math.isnan(float(elevacion)):
                solditems =
requests.get(f'https://servizos.meteogalicia.gal/apiv4/getNumericForecastInfo?variabl
es=sea_water_salinity&coords={LONG},{LAT}&format=application/json&API_KEY=
CONFIDENCIAL')

                data = solditems.json()
                with open(f'Salinidad{LONG}_{LAT}.json', 'w') as f:
                    json.dump(data, f)
                client = pymongo.MongoClient("mongodb+srv://
*****:*****@cluster0.9cawz8x.mongodb.net/GemeloDigitalOceanográfico")
                db = client["GemeloDigitalOceanográfico"]
                collection = db["Salinidad"]
                collection.insert_one(data)

def Oleaje(Self):
    with open('BatiRiasBaixas.csv', 'r') as file:
        reader = csv.reader(file)
        next(reader)
```

```
next(reader)
for row in reader:
    LAT, LONG, elevacion = row
    if not math.isnan(float(elevacion)):
        solditems =
requests.get(f'https://servizos.meteogalicia.gal/apiv4/getNumericForecastInfo?variables=significant_wave_height,mean_wave_direction,relative_peak_period&coords&coords={LONG},{LAT};&format=application/json&API_KEY= CONFIDENCIAL')
        data = solditems.json()
        with open(f'Oleaje_{LONG}_{LAT}.json', 'w') as f:
            json.dump(data, f)
        client = pymongo.MongoClient("mongodb+srv://
*****:*****@cluster0.9cawz8x.mongodb.net/GemeloDigitalOceanográfico")
        db = client["GemeloDigitalOceanográfico"]
        collection = db["Oleaje"]
        collection.insert_one(data)
```

A.4.1.4 Ejecución código para obtención de datos.

```
import csv
import json
import requests
from MeteoGalicia import MeteoGalicia
from MeteoClass import MeteoClass
```

```
MeteoG = MeteoGalicia()
MeteoG.Temperatura()
print(MeteoG.nombre)
```

A.4.2 Diseño web “Gemelo Digital Oceanográfico”

En el siguiente bloque se reflejan todos los códigos, así como los ficheros html presentes en el desarrollo e implementación de la página web.

A.4.2.1 MYSITE.

A.4.2.1.1 Settings.

```
"""
```

```
Django settings for mysite project.
```

```
Generated by 'django-admin startproject' using Django 4.1.7.
```

```
For more information on this file, see
https://docs.djangoproject.com/en/4.1/topics/settings/
```

```
For the full list of settings and their values, see
https://docs.djangoproject.com/en/4.1/ref/settings/
"""
```

```
from pathlib import Path
import os
# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-_)$$8#!*d0!0@g8uv*cnzlw4_ibzy+=o3jd%+)44z!1f&u##yp'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = [
```

```
'127.0.0.1',
]

# Application definition

INSTALLED_APPS = [
    #mis app
    'personal',
    'account',
    'GDO',

    #djangoapps
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

X_FRAME_OPTIONS = 'SAMEORIGIN'

ROOT_URLCONF = 'mysite.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

```
    },
]

AUTH_USER_MODEL = 'account.Account'

WSGI_APPLICATION = 'mysite.wsgi.application'

# Database
# https://docs.djangoproject.com/en/4.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

# Password validation
# https://docs.djangoproject.com/en/4.1/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

# Internationalization
# https://docs.djangoproject.com/en/4.1/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True
```

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.1/howto/static-files/

STATIC_URL = 'static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static')

# Default primary key field type
# https://docs.djangoproject.com/en/4.1/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

A.4.2.2 URLS.

```
"""mysite URL Configuration
```

```
The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/4.1/topics/http/urls/
```

```
Examples:
```

```
Function views
```

1. Add an import: from my_app import views
2. Add a URL to urlpatterns: path('', views.home, name='home')

```
Class-based views
```

1. Add an import: from other_app.views import Home
2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')

```
Including another URLconf
```

1. Import the include() function: from django.urls import include, path
2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))

```
"""
```

```
from django.contrib import admin
from django.views.generic import TemplateView
from django.urls import path
from django.conf import settings
from personal.views import (
    home_screen_view,
    home1_screen_view,
    memoria_screen_view,
    todo,
    iframe_screen_view,
)
from account.views import (
    registration_view,
    logout_view,
    login_view,
    account_view,
)
from django.conf.urls.static import (
    static,
)
urlpatterns = [
```

```
path('admin/', admin.site.urls),
path('home/', home_screen_view, name='home'),
path('register/', registration_view, name='register'),
path('logout/', logout_view, name='logout'),
path('login/', login_view, name='login'),
path('account/', account_view, name='account'),
path('memoria/', memoria_screen_view, name='memoria'),
path('home1/', home1_screen_view, name='home1'),
path('todo/', todo, name='todo'),
path('iframe/', iframe_screen_view, name='iframe'),
]
```

A.4.2.2 Templates.

A.4.2.2.1 Base.

```
<!DOCTYPE html>
<html>
<head>
    <div class="pricing-header px-3 py-3 pt-md-5 pb-md-4 mx-auto text-center">
        <h1 class="display-4">Valera Engineering </h1>
    </div>

    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
fit=no">
    <hr>
    {% include 'snippets/head.html' %}
    <style type="text/css">
        .background {
            box-sizing: border-box;
            width: 100%;
            height: auto;
            padding: 3px;
            background-size: auto;
            background-image:
url(https://i.pinimg.com/originals/66/8d/06/668d065a78310b8f1ae77adb70e612fd.jpg);
            background-size: cover;
            background-attachment: scroll;
            border: 1px solid black;
            background-size: auto;
        }
    </style>
```

```
</html>
<body>
  <div class="background">

    <!--Cuerpo principal de la pagina-->

    {% block content %}

    {% endblock content %}
    <!--Fin Cuerpo principal de la pagina-->

  </div>
  {% include 'snippets/footer.html' %}

</body>
```

```
</html>
```

A.4.2.2 Snippets.

A.4.2.2.4.1 Header.

```
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.3.1/dist/css/bootstrap.min.css"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.4.0/jquery.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-
validate/1.19.0/jquery.validate.min.js"></script>

<style type="text/css">
  @media (min-width: 768px) {
    html {
      font-size: 16px;
    }
  }

  .container {
    max-width: 960px;
  }
```

```

.pricing-header {
    max-width: 700px;
}

.card-deck .card {
    min-width: 220px;
}
</style>
<!--Encabezado-->
<div class="d-flex flex-column flex-md-row align-items-center p-3 px-md-4 mb-3 bg-
white border-bottom shadow-sm">
    <h4 class="my-0 mr-md-auto font-weight-normal">
        {% if request.user.is_authenticated %}
        <p>Bienvenido, {{request.user.username}}!</p>
    </h4>
    {% endif %}
    <nav class="my-2 my-md-0 mr-md-3">
        {% if request.user.is_authenticated %}
        <p>
            <a class="p2 text-dark" href="{% url 'home' %}">Inicio |</a>
            <a class="p2 text-dark" href="{% url 'account'
%}">|
            Account |</a>
            <a class="p2 text-dark" href="{% url 'logout' %}">| Logout</a>
        </p>
        {% else %}
        <p>
            <a class="p2 text-dark" href="{% url 'home' %}">Inicio |</a>
            <a class="p2 text-dark" href="{% url 'login' %}">| Login |</a>
            <a class="btn btn-outline-primary" href="{% url 'register'
%}">|
            Regístrate</a>
        </p>
        {% endif %}
    </nav>
</div>
<!-- Fin Encabezado-->

```

A.4.2.2.4.2 Footer.

```

<style type="text/css">
    .footer{
        min-height: 100px;
    }
</style>
<!--Pie de pagina-->
<hr>
<div class="d-flex flex-column align-items-center footer m-auto">
    <h5 class="m-auto">By: Pablo Garcia Valera | 2023</h5><br>
    <p class="m-auto">Con la colaboración de:</p><br>

```

```
<a href="https://cud.uvigo.es/pdi/name/milagros-fernandez-gavilanes/" >Dña.
Milagros Fernández Gavilanes</a><br>
<a href="https://cud.uvigo.es/pdi/name/carlos-perez-collazo/">D. Carlos Pérez
Collazzo</a><br>
</div>

<!--Fin Pie de pagina-->
```

```
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-
q8i/X+965Dz00rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.7/dist/umd/popper.min.js"
integrity="sha384-U02eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@4.3.1/dist/js/bootstrap.min.js"
integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"
crossorigin="anonymous"></script>
```

A.4.2.3 Account.

A.4.2.2.1 Views.

```
from django.shortcuts import render, redirect
from django.contrib.auth import login, authenticate, logout
from account.forms import RegistrationForm, AccountAuthenticationForm,
AccountUpdateForm
import account

def registration_view(request):
    context = {}
    if request.POST:
        form = RegistrationForm(request.POST)
        if form.is_valid():
            form.save()
            email = form.cleaned_data.get('email')
            raw_password = form.cleaned_data.get('password1')
            account = authenticate(email=email, password= raw_password)
            login(request, account)
            return redirect('home')
        else:
            context['registration_form'] = form
    else:
        form = RegistrationForm()
        context['registration_form'] = form
    return render(request, 'account/register.html', context)

def logout_view(request):
```

```
logout(request)
return redirect('home1')

def login_view(request):

    context = {}

    user= request.user
    if user.is_authenticated:
        return redirect('home')

    if request.POST:
        form = AccountAuthenticationForm(request.POST)
        if form.is_valid():
            email = request.POST['email']
            password = request.POST['password']
            user = authenticate(email=email, password=password)

            if user:
                login(request, user)
                return redirect ('home')
        else:
            form = AccountAuthenticationForm
    context['login_form'] = form
    return render(request, 'account/login.html', context)

def account_view(request):

    if not request.user.is_authenticated:
        return redirect('login')

    context = {}

    if request.POST:
        form = AccountUpdateForm(request.POST, instance=request.user)
        if form.is_valid():
            form.save()
    else:
        form = AccountUpdateForm(
            initial= {
                'email': request.user.email,
                'username': request.user.username,
            }
        )

    context['account_form']= form
    return render(request, 'account/account.html', context)
```

A.4.2.2.2 Models.

```
from django.db import models
from django.contrib.auth.models import AbstractBaseUser, BaseUserManager

class MyAccountManager(BaseUserManager):
    def create_user(self, email, username, password=None):
        if not email:
            raise ValueError('Users must have an email address')
        if not username:
            raise ValueError('Users must have a username')

        user = self.model(
            email=self.normalize_email(email),
            username=username,
        )

        user.set_password(password)
        user.save(using=self._db)
        return user

    def create_superuser(self, email, username, password):
        user = self.create_user(
            email=self.normalize_email(email),
            password=password,
            username=username,
        )
        user.is_admin = True
        user.is_staff = True
        user.is_superuser = True
        user.save(using=self._db)
        return user

class Account(AbstractBaseUser):
    email = models.EmailField(verbose_name="email", max_length=60,
unique=True)
    username = models.CharField(max_length=30, unique=True)
    date_joined = models.DateTimeField(verbose_name='date joined',
auto_now_add=True)
    last_login = models.DateTimeField(verbose_name='last login',
auto_now=True)
    is_admin = models.BooleanField(default=False)
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)
    is_superuser = models.BooleanField(default=False)

    USERNAME_FIELD = 'email'
```

```
REQUIRED_FIELDS = ['username']

objects = MyAccountManager()

def __str__(self):
    return self.email + ", " + self.username

def has_perm(self, perm, obj=None):
    return self.is_admin

def has_module_perms(self, app_label):
    return True
```

A.4.2.2.3 Forms.

```
from django import forms
from django.contrib.auth.forms import UserCreationForm
from account.models import Account
from django.contrib.auth import authenticate

class RegistrationForm(UserCreationForm):
    email = forms.EmailField(max_length=60, help_text= ' Required. Añade un email valido')

    class Meta:
        model = Account
        fields = ('email', 'username', 'password1', 'password2')

class AccountAuthenticationForm(forms.ModelForm):

    password = forms.CharField(label= 'Password', widget=forms.PasswordInput)

    class Meta:
        model = Account
        fields = ('email', 'password')

    def clean(self):
        email = self.cleaned_data['email']
        password = self.cleaned_data['password']
        if not authenticate(email=email, password=password):
            raise forms.ValidationError("Login erroneo")

class AccountUpdateForm(forms.ModelForm):

    class Meta:
        model = Account
        fields = ('email', 'username')
```

```

def clean_email(self):
    if self.is_valid():
        email = self.cleaned_data['email']
        try:
            account =
Account.objects.exclude(pk=self.instance.pk).get(email=email)
        except Account.DoesNotExist:
            return email
        raise forms.ValidationError('El email: "%s", ya esta en uso.' %
account.email)

def clean_username(self):
    if self.is_valid():
        username = self.cleaned_data['username']
        try:
            account =
Account.objects.exclude(pk=self.instance.pk).get(username=username)
        except Account.DoesNotExist:
            return username
        raise forms.ValidationError('El nombre de usuario: "%s", ya está en uso.'
% account.username)

```

A.4.2.2.4 Templates.

A.4.2.2.4.1 Account.

```

{% extends 'base.html' %}

{% block content %}

<h2>Account</h2>
<form method="post">{% csrf_token %}
    {% for field in account_form %}
        <p>
            {{field.label_tag}}
            {{field}}

            {% if field.help_text %}
                <small style="color: grey;"> {{field.help_text}}</small>
            {% endif %}

            {% for error in field.errors %}
                <p style="color: red;"> {{error}}</p>
            {% endfor %}

            {% if account_form.non_field_error %}
                <div style="color: red;">
                    <p>{{account_form.non_field_errors}}</p>
                </div>

            {% endif %}

```

```
    </p>
{% endfor %}
<button type="submit">Guardar</button>
```

```
</form>
{% endblock content %}
```

A.4.2.2.4.2 Login

```
<h2>Login</h2>
<form method="post">{% csrf_token %}
  {% for field in login_form %}
    <p>
      {{field.label_tag}}
      {{field}}

      {% if field.help_text %}
        <small style="color: grey;"> {{field.help_text}}</small>
      {% endif %}

      {% for error in field.errors %}
        <p style="color: red;"> {{error}}</p>
      {% endfor %}

      {% if login_form.non_field_error %}
        <div style="color: red;">
          <p>{{login_form.non_field_errors}}</p>
        </div>

      {% endif %}
    </p>
  {% endfor %}
  <button type="submit">Log in</button>

</form>
{% endblock content %}
```

A.4.2.2.4.3 Register.

```
{% extends 'base.html' %}

{% block content %}

<h2>Register</h2>
<form method="post">{% csrf_token %}
  {% for field in registration_form %}
    <p>
      {{field.label_tag}}
      {{field}}

      {% if field.help_text %}
        <small style="color: grey;"> {{field.help_text}}</small>
      {% endif %}

      {% for error in field.errors %}
        <small style="color: red;"> {{error}}</small>
      {% endfor %}
    </p>
  {% endfor %}
  <button type="submit">Register</button>

</form>
{% endblock content %}
```

A.4.2.4 Personal.

A.4.2.4.1 Views.

```
from django.shortcuts import render, redirect
from GDO.forms import DatosForm
from django.http import HttpRequest, HttpResponse
import requests
import pymongo
import numpy as np
import h3
import folium
import matplotlib.cm
import matplotlib.colors
from folium import plugins
from folium.plugins import MiniMap
from folium import Map, Marker
from folium.features import CustomIcon
from folium import Icon
import webbrowser
from account.models import Account
import math
import shutil

def home_screen_view(request):
    context = {}

    return render(request, 'personal/home.html', context)

def iframe_screen_view(request):
    context = {}

    return render(request, 'personal/iframe.html', context)

def home1_screen_view(request):
    context = {}

    return render(request, 'personal/home1.html', context)

def memoria_screen_view(request):
    context = {}
    return render(request, 'GDO/memoria.html', context)

def calculate_distance(lat1, lon1, lat2, lon2):
    # Formula de Haversine para calcular la distancia entre dos coordenadas
    R = 6371 # Radio de la Tierra en kilómetros
    dlat = math.radians(lat2 - lat1)
    dlon = math.radians(lon2 - lon1)
    a = math.sin(dlat / 2) * math.sin(dlat / 2) + math.cos(math.radians(lat1)) \
```

```

        * math.cos(math.radians(lat2)) * math.sin(dlon / 2) * math.sin(dlon / 2)
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    distance = R * c
    return distance

def todo(request):
    if request.method == 'POST':
        if 'Batimetria' in request.POST.get('submit'):
            unidad = request.POST.get('unidad')
            Lati = request.POST.get('latitud')
            Longi = request.POST.get('longitud')
            moment_indicado = request.POST.get('hora')
            radio_accion = request.POST.get('radio')
            resolucion = request.POST.get('resolucion')

            client = pymongo.MongoClient("mongodb+srv://
*****:*****@cluster0.9cawz8x.mongodb.net/GemeloDigitalOceanográfico")
            db = client["GemeloDigitalOceanográfico"]
            collection = db["Batimetria"]

            datos=[]
            datos = list(collection.find( {}, { "latitude": 1, "longitude": 1,
"elevation": 1}))
            data = []
            for item in datos:
                latitude = item["latitude"]
                longitude = item["longitude"]
                distance = calculate_distance(float(latitude), float(longitude),
float(Lati), float(Longi))
                if float(distance) <= float(radio_accion): # x es el radio deseado en
kilómetros
                    data.append(item)

            resolution = float(resolucion)
            hexagons = []
            for point in data:
                lat = point['latitude']
                lng = point['longitude']
                long = float(lng)
                lati = float(lat)
                hex_index = h3.geo_to_h3(lati, long, resolution)
                hexagons.append(hex_index)

            hexagons = list(set(hexagons))
            hex_data = {}
            for hex_index in hexagons:
                hex_data[hex_index] = []

            for point in data:

```

```
lat = point['latitude']
lng = point['longitude']
long = float(lng)
lati = float(lat)
hex_index = h3.geo_to_h3(lati, long, resolution)
value = point['elevation']
values = float(value)
hex_data[hex_index].append(values)

for hex_index in hexagons:
    values = hex_data[hex_index]
    mean_value = np.mean(values)
    hex_data[hex_index] = mean_value

min_value = min(hex_data.values())
max_value = max(hex_data.values())
value_range = max_value - min_value

color_hex_data = {}
for hex_index, value in hex_data.items():
    color_value = (value - min_value) / value_range
    color = matplotlib.cm.get_cmap("RdYlBu_r")(color_value)
    color = matplotlib.colors.to_hex(color)
    color_hex_data[hex_index] = color
    #valores más pequeños como rojo: RdYlBu_r
    #valores más pequeños como verde: YlGn

m = folium.Map(location=[Lati, Longi], zoom_start=13, tiles =
'cartodbpositron')

for hex_index in hexagons:
    hex_vertices = h3.h3_to_geo_boundary(hex_index)
    lat = [vertex[0] for vertex in hex_vertices]
    lng = [vertex[1] for vertex in hex_vertices]
    color = color_hex_data[hex_index]
    mean_value = hex_data[hex_index]
    folium.Polygon(locations=[(lat_i, lng_i) for lat_i, lng_i in
zip(lat, lng)],

                                color='black',
                                weight = 1,
                                fill=True,
                                fill_color=color,
                                opacity=0.8,
                                popup="Profundidad: {}".format(round(mean_value, 2))).add_to(m)

    tooltip = unidad
    my_icon = folium.features.CustomIcon('templates/icono.png', icon_size =
(50,50))
    folium.Marker([Lati, Longi], icon = my_icon, tooltip=tooltip).add_to(m)
```

```

        minimap =
MiniMap(toggle_display=True,position="topleft",tile_layer="stamenterrain")

        m.add_child(minimap)
        m.save("iframe.html")
        shutil.move('iframe.html', 'personal/templates/personal/iframe.html')
        return render(request, 'personal/home.html', {'unidad' : unidad,
'latitud' : Lati, 'longitud' : Longi, 'hora' : moment_indicado, 'radio' :
radio_accion, 'resolucion' : resolucion})

    elif 'Humedad' in request.POST.get('submit'):
        unidad = request.POST.get('unidad')
        Lati = request.POST.get('latitud')
        Longi = request.POST.get('longitud')
        moment_indicado = request.POST.get('hora')
        radio_accion = request.POST.get('radio')
        resolucion = request.POST.get('resolucion')

        # Conectarse a la base de datos
        client = pymongo.MongoClient("mongodb+srv://
*****:*****@cluster0.9cawz8x.mongodb.net/GemeloDigitalOceanográfico")
        db = client["GemeloDigitalOceanográfico"]
        collection = db["Humedad"]

        time_to_search = moment_indicado
        data = []
        features =
collection.find({"features.properties.days.variables.values.timeInstant":
moment_indicado})
        for feature in features:
            for day in feature['features'][0]['properties']['days']:
                if day['timePeriod']['begin']['timeInstant'] <=
time_to_search <= day['timePeriod']['end']['timeInstant']:
                    for variable in day['variables']:
                        if variable['name'] == 'relative_humidity':
                            longitude =
float(feature['features'][0]['geometry']['coordinates'][0])
                            latitude =
float(feature['features'][0]['geometry']['coordinates'][1])
                            distance = calculate_distance(float(latitude),
float(longitude), float(Lati), float(Longi))
                            if float(distance) <= float(radio_accion): # x es
el radio deseado en kilómetros

                                for value in variable['values']:
                                    if value['timeInstant'] ==
time_to_search:

```

```
relative_humidity =
int(value['value'])
data.append({'longitude': longitude,
'latitude': latitude, 'relative_humidity': relative_humidity})

# Crear un mallado h3 con una resolución de 8
resolution = float(resolucion)
hexagons = []
for point in data:
    lat = point['latitude']
    lng = point['longitude']
    hex_index = h3.geo_to_h3(lat, lng, resolution)
    hexagons.append(hex_index)

# Asignar cada punto de los datos a un hexágono h3
hexagons = list(set(hexagons))
hex_data = {}
for hex_index in hexagons:
    hex_data[hex_index] = []

for point in data:
    lat = point['latitude']
    lng = point['longitude']
    hex_index = h3.geo_to_h3(lat, lng, resolution)
    value = point['relative_humidity']
    hex_data[hex_index].append(value)

# Calcular la media de los datos en cada hexágono
for hex_index in hexagons:
    values = hex_data[hex_index]
    mean_value = sum(values) / len(values)
    hex_data[hex_index + "_mean"] = mean_value

min_value = np.min([hex_data[hex_index + "_mean"] for hex_index in
hexagons])
max_value = np.max([hex_data[hex_index + "_mean"] for hex_index in
hexagons])
value_range = max_value - min_value

color_hex_data = {}
cmap = matplotlib.cm.get_cmap("RdYlBu_r")
for hex_index in hexagons:
    value = hex_data[hex_index + "_mean"]
    color_value = (value - min_value) / value_range
    color = cmap(color_value)
    color = matplotlib.colors.to_rgba(color)
    hex_index = hex_index.replace("_mean", "")
    color_hex_data[hex_index] = color
```

```

m = folium.Map(location=[Lati, Longi], tiles= 'cartodbpositron',
zoom_start=13)

for hex_index in hexagons:
    hex_vertices = h3.h3_to_geo_boundary(hex_index)
    lat = [vertex[0] for vertex in hex_vertices]
    lng = [vertex[1] for vertex in hex_vertices]
    color = color_hex_data[hex_index]
    mean_value = hex_data[hex_index + "_mean"]
    fill_color = matplotlib.colors.rgb2hex(color[:3])
    folium.Polygon(locations=[(lat_i, lng_i) for lat_i, lng_i in
zip(lat, lng)],
                    color='black',
                    weight = 1,
                    fill=True,
                    fill_color=fill_color,
                    opacity=0.9,
                    popup="Humedad: {}".format(round(mean_value,
2))).add_to(m)

tooltip = unidad
my_icon = folium.features.CustomIcon('templates/icono.png', icon_size =
(50,50))
folium.Marker([Lati, Longi], icon = my_icon, tooltip=tooltip).add_to(m)
minimap =
MiniMap(toggle_display=True,position="topleft",tile_layer="stamenterrain")

m.add_child(minimap)
m.save("iframe.html")
shutil.move('iframe.html', 'personal/templates/personal/iframe.html')
return render(request, 'personal/home.html', {'unidad' : unidad,
'latitud' : Lati, 'longitud' : Longi, 'hora' : moment_indicado, 'radio' :
radio_accion, 'resolucion' : resolucion})

elif 'Nubosidad' in request.POST.get('submit'):
    unidad = request.POST.get('unidad')
    Lati = request.POST.get('latitud')
    Longi = request.POST.get('longitud')
    moment_indicado = request.POST.get('hora')
    radio_accion = request.POST.get('radio')
    resolucion = request.POST.get('resolucion')

# Conectarse a la base de datos
client = pymongo.MongoClient("mongodb+srv://
*****:*****@cluster0.9cawz8x.mongodb.net/GemeloDigitalOceanográfico")
db = client["GemeloDigitalOceanográfico"]
collection = db["Nubosidad"]

```

```
        time_to_search = moment_indicado
        data = []
        features =
collection.find({"features.properties.days.variables.values.timeInstant":
moment_indicado})
        for feature in features:
            for day in feature['features'][0]['properties']['days']:
                if day['timePeriod']['begin']['timeInstant'] <=
time_to_search <= day['timePeriod']['end']['timeInstant']:
                    for variable in day['variables']:
                        if variable['name'] == 'cloud_area_fraction':
                            longitude =
float(feature['features'][0]['geometry']['coordinates'][0])
                            latitude =
float(feature['features'][0]['geometry']['coordinates'][1])
                            distance = calculate_distance(float(latitude),
float(longitude), float(Lati), float(Longi))
                            if float(distance) <= float(radio_accion): # x es
el radio deseado en kilómetros
                                for value in variable['values']:
                                    if value['timeInstant'] ==
time_to_search:
                                        cloud_area_fraction =
int(value['value'])
                                        data.append({'longitude': longitude,
'latitude': latitude, 'cloud_area_fraction': cloud_area_fraction})

# Crear un mallado h3 con una resolución de 8
resolution = float(resolucion)
hexagons = []
for point in data:
    lat = point['latitude']
    lng = point['longitude']
    hex_index = h3.geo_to_h3(lat, lng, resolution)
    hexagons.append(hex_index)

# Asignar cada punto de los datos a un hexágono h3
hexagons = list(set(hexagons))
hex_data = {}
for hex_index in hexagons:
    hex_data[hex_index] = []

for point in data:
    lat = point['latitude']
    lng = point['longitude']
```

```

        hex_index = h3.geo_to_h3(lat, lng, resolution)
        value = point['cloud_area_fraction']
        hex_data[hex_index].append(value)

# Calcular la media de los datos en cada hexágono
for hex_index in hexagons:
    values = hex_data[hex_index]
    mean_value = sum(values) / len(values)
    hex_data[hex_index + "_mean"] = mean_value

min_value = np.min([hex_data[hex_index + "_mean"] for hex_index in
hexagons])
max_value = np.max([hex_data[hex_index + "_mean"] for hex_index in
hexagons])
value_range = max_value - min_value

color_hex_data = {}
cmap = matplotlib.cm.get_cmap("RdYlBu_r")
for hex_index in hexagons:
    value = hex_data[hex_index + "_mean"]
    color_value = (value - min_value) / value_range
    color = cmap(color_value)
    color = matplotlib.colors.to_rgba(color)
    hex_index = hex_index.replace("_mean", "")
    color_hex_data[hex_index] = color

m = folium.Map(location=[Lati, Longi], tiles= 'cartodbpositron',
zoom_start=13)

for hex_index in hexagons:
    hex_vertices = h3.h3_to_geo_boundary(hex_index)
    lat = [vertex[0] for vertex in hex_vertices]
    lng = [vertex[1] for vertex in hex_vertices]
    color = color_hex_data[hex_index]
    mean_value = hex_data[hex_index + "_mean"]
    fill_color = matplotlib.colors.rgb2hex(color[:3])
    folium.Polygon(locations=[(lat_i, lng_i) for lat_i, lng_i in
zip(lat, lng)],

                                color='black',
                                weight = 1,
                                fill=True,
                                fill_color=fill_color,
                                opacity=0.9,
                                popup="Nubosidad: {}".format(round(mean_value,
2))).add_to(m)

tooltip = unidad

```

```
my_icon = folium.features.CustomIcon('templates/icono.png', icon_size =
(50,50))
folium.Marker([Lati, Longi], icon = my_icon, tooltip=tooltip).add_to(m)
minimap =
MiniMap(toggle_display=True,position="topleft",tile_layer="stamenterrain")

m.add_child(minimap)
m.save("iframe.html")
shutil.move('iframe.html', 'personal/templates/personal/iframe.html')
return render(request, 'personal/home.html', {'unidad' : unidad,
'latitud' : Lati, 'longitud' : Longi, 'hora' : moment_indicado, 'radio' :
radio_accion, 'resolucion' : resolucion})

elif 'Oleaje' in request.POST.get('submit'):
    unidad = request.POST.get('unidad')
    Lati = request.POST.get('latitud')
    Longi = request.POST.get('longitud')
    moment_indicado = request.POST.get('hora')
    radio_accion = request.POST.get('radio')
    resolucion = request.POST.get('resolucion')

    client = pymongo.MongoClient("mongodb+srv://
*****:*****@cluster0.9cawz8x.mongodb.net/GemeloDigitalOceanográfico")
    db = client["GemeloDigitalOceanográfico"]
    collection = db["Oleaje"]

    time_to_search = moment_indicado
    data1 = []
    data2 = []
    data3 = []
    features =
collection.find({"features.properties.days.variables.values.timeInstant":
moment_indicado})
    for feature in features:
        for day in feature['features'][0]['properties']['days']:
            if day['timePeriod']['begin']['timeInstant'] <=
time_to_search <= day['timePeriod']['end']['timeInstant']:
                for variable in day['variables']:
                    if variable['name'] == 'significant_wave_height':
                        longitude =
float(feature['features'][0]['geometry']['coordinates'][0])
                        latitude =
float(feature['features'][0]['geometry']['coordinates'][1])
                        distance = calculate_distance(float(latitude),
float(longitude), float(Lati), float(Longi))
                        if float(distance) <= float(radio_accion): # x es
el radio deseado en kilómetros

                            for value in variable['values']:
```

```

        if value['timeInstant'] ==
time_to_search:
        significant_wave_height =
int(value['value'])
        data1.append({'longitude': longitude,
'latitude': latitude, 'significant_wave_height': significant_wave_height})

        for variable in day['variables']:
            if variable['name'] == 'mean_wave_direction':
                longitude =
float(feature['features'][0]['geometry']['coordinates'][0])
                latitude =
float(feature['features'][0]['geometry']['coordinates'][1])
                distance = calculate_distance(float(latitude),
float(longitude), float(Lati), float(Longi))
                if float(distance) <= float(radio_accion): # x es
el radio deseado en kilómetros
                    for value in variable['values']:
                        if value['timeInstant'] ==
time_to_search:
                            mean_wave_direction =
int(value['value'])
                            data2.append({'longitude': longitude,
'latitude': latitude, 'mean_wave_direction': mean_wave_direction})

                            for variable in day['variables']:
                                if variable['name'] == 'relative_peak_period':
                                    longitude =
float(feature['features'][0]['geometry']['coordinates'][0])
                                    latitude =
float(feature['features'][0]['geometry']['coordinates'][1])
                                    distance = calculate_distance(float(latitude),
float(longitude), float(Lati), float(Longi))
                                    if float(distance) <= float(radio_accion): # x es
el radio deseado en kilómetros
                                        for value in variable['values']:
                                            if value['timeInstant'] ==
time_to_search:
                                                relative_peak_period =
int(value['value'])
                                                data3.append({'longitude': longitude,
'latitude': latitude, 'relative_peak_period': relative_peak_period})

# Crear un mallado h3 con una resolución de 8
resolution = float(resolucion)
hexagons = []
for point in data1:
    lat = point['latitude']

```

```
        lng = point['longitude']
        hex_index = h3.geo_to_h3(lat, lng, resolution)
        hexagons.append(hex_index)

# Asignar cada punto de los datos a un hexágono h3
hexagons = list(set(hexagons))
hex_data = {}
for hex_index in hexagons:
    hex_data[hex_index] = []

for point in data1:
    lat = point['latitude']
    lng = point['longitude']
    hex_index = h3.geo_to_h3(lat, lng, resolution)
    value = point['significant_wave_height']
    hex_data[hex_index].append(value)

# Calcular la media de los datos en cada hexágono
for hex_index in hexagons:
    values = hex_data[hex_index]
    mean_valuea = sum(values) / len(values)
    hex_data[hex_index + "_mean"] = mean_valuea

min_value = np.min([hex_data[hex_index + "_mean"] for hex_index in
hexagons])
max_value = np.max([hex_data[hex_index + "_mean"] for hex_index in
hexagons])
value_range = max_value - min_value

#media de la direccion
hexagons = list(set(hexagons))
hex_data1 = {}
for hex_index in hexagons:
    hex_data1[hex_index] = []

for point in data2:
    lat = point['latitude']
    lng = point['longitude']
    hex_index = h3.geo_to_h3(lat, lng, resolution)
    value = point['mean_wave_direction']
    hex_data1[hex_index].append(value)

# Calcular la media de los datos en cada hexágono
for hex_index in hexagons:
    valuesd = hex_data1[hex_index]
    mean_valued = sum(valuesd) / len(valuesd)
    hex_data1[hex_index + "_mean"] = mean_valued

min_value1 = np.min([hex_data1[hex_index + "_mean"] for hex_index in
hexagons])
```

```

max_value1 = np.max([hex_data1[hex_index + "_mean"] for hex_index in
hexagons])
value_range1 = max_value1 - min_value1

        #media del periodo
hexagons = list(set(hexagons))
hex_data2 = {}
for hex_index in hexagons:
    hex_data2[hex_index] = []

for point in data3:
    lat = point['latitude']
    lng = point['longitude']
    hex_index = h3.geo_to_h3(lat, lng, resolution)
    value = point['relative_peak_period']
    hex_data2[hex_index].append(value)

# Calcular la media de los datos en cada hexágono
for hex_index in hexagons:
    values = hex_data2[hex_index]
    mean_values = sum(values) / len(values)
    hex_data2[hex_index + "_mean"] = mean_values

min_value2 = np.min([hex_data2[hex_index + "_mean"] for hex_index in
hexagons])
max_value2 = np.max([hex_data2[hex_index + "_mean"] for hex_index in
hexagons])
value_range2 = max_value2 - min_value2

color_hex_data = {}
cmap = matplotlib.cm.get_cmap("RdYlBu_r")
for hex_index in hexagons:
    value = hex_data[hex_index + "_mean"]
    color_value = (value - min_value) / value_range
    color = cmap(color_value)
    color = matplotlib.colors.to_rgba(color)
    hex_index = hex_index.replace("_mean", "")
    color_hex_data[hex_index] = color

m = folium.Map(location=[Lati, Longi], tiles='cartodbpositron',
zoom_start=13)

for hex_index in hexagons:
    hex_vertices = h3.h3_to_geo_boundary(hex_index)
    lat = [vertex[0] for vertex in hex_vertices]
    lng = [vertex[1] for vertex in hex_vertices]
    color = color_hex_data[hex_index]

```

```
mean_valuea = hex_data[hex_index + "_mean"]
mean_valued = hex_data1[hex_index + "_mean"]
mean_values = hex_data2[hex_index + "_mean"]
fill_color = matplotlib.colors.rgb2hex(color[:3])
folium.Polygon(locations=[(lat_i, lng_i) for lat_i, lng_i in
zip(lat, lng)],
                    color='black',
                    weight = 1,
                    fill=True,
                    fill_color=fill_color,
                    opacity=0.8,
                    popup="Altura: {} m Dirección: {}° Período {}
s".format(round(mean_valuea, 2), round(mean_valued, 2),
round(mean_values))).add_to(m)

tooltip = unidad
my_icon = folium.features.CustomIcon('templates/icono.png', icon_size =
(50,50))
folium.Marker([Lati, Longi], icon = my_icon, tooltip=tooltip).add_to(m)
minimap =
MiniMap(toggle_display=True,position="topleft",tile_layer="stamenterrain")

m.add_child(minimap)
m.save("iframe.html")
shutil.move('iframe.html', 'personal/templates/personal/iframe.html')
return render(request, 'personal/home.html', {'unidad' : unidad,
'latitud' : Lati, 'longitud' : Longi, 'hora' : moment_indicado, 'radio' :
radio_accion, 'resolucion' : resolucion})

elif 'Precipitaciones' in request.POST.get('submit'):
    unidad = request.POST.get('unidad')
    Lati = request.POST.get('latitud')
    Longi = request.POST.get('longitud')
    moment_indicado = request.POST.get('hora')
    radio_accion = request.POST.get('radio')
    resolucion = request.POST.get('resolucion')

    # Conectarse a la base de datos
    client = pymongo.MongoClient("mongodb+srv://
*****:*****@cluster0.9cawz8x.mongodb.net/GemeloDigitalOceanográfico")
    db = client["GemeloDigitalOceanográfico"]
    collection = db["Precipitaciones"]

    time_to_search = moment_indicado
    data = []
    features =
collection.find({"features.properties.days.variables.values.timeInstant":
moment_indicado})
```

```

        for feature in features:
            for day in feature['features'][0]['properties']['days']:
                if day['timePeriod']['begin']['timeInstant'] <=
time_to_search <= day['timePeriod']['end']['timeInstant']:
                    for variable in day['variables']:
                        if variable['name'] == 'precipitation_amount':
                            longitude =
float(feature['features'][0]['geometry']['coordinates'][0])
                            latitude =
float(feature['features'][0]['geometry']['coordinates'][1])
                            distance = calculate_distance(float(latitude),
float(longitude), float(Lati), float(Longi))
                            if float(distance) <= float(radius_accion): # x es
el radio deseado en kilómetros
                                for value in variable['values']:
                                    if value['timeInstant'] ==
time_to_search:
                                        precipitation_amount =
int(value['value'])
                                        data.append({'longitude': longitude,
'latitude': latitude, 'precipitation_amount': precipitation_amount})

# Crear un mallado h3 con una resolución de 8
resolution = float(resolucion)
hexagons = []
for point in data:
    lat = point['latitude']
    lng = point['longitude']
    hex_index = h3.geo_to_h3(lat, lng, resolution)
    hexagons.append(hex_index)

# Asignar cada punto de los datos a un hexágono h3
hexagons = list(set(hexagons))
hex_data = {}
for hex_index in hexagons:
    hex_data[hex_index] = []

for point in data:
    lat = point['latitude']
    lng = point['longitude']
    hex_index = h3.geo_to_h3(lat, lng, resolution)
    value = point['precipitation_amount']
    hex_data[hex_index].append(value)

# Calcular la media de los datos en cada hexágono
for hex_index in hexagons:
    values = hex_data[hex_index]

```

```
        mean_value = sum(values) / len(values)
        hex_data[hex_index + "_mean"] = mean_value

    min_value = np.min([hex_data[hex_index + "_mean"] for hex_index in
hexagons])
    max_value = np.max([hex_data[hex_index + "_mean"] for hex_index in
hexagons])
    value_range = max_value - min_value

    color_hex_data = {}
    cmap = matplotlib.cm.get_cmap("RdYlBu_r")
    for hex_index in hexagons:
        value = hex_data[hex_index + "_mean"]
        color_value = (value - min_value) / value_range
        color = cmap(color_value)
        color = matplotlib.colors.to_rgba(color)
        hex_index = hex_index.replace("_mean", "")
        color_hex_data[hex_index] = color

    m = folium.Map(location=[Lati, Longi], tiles= 'cartodbpositron',
zoom_start=13)

    for hex_index in hexagons:
        hex_vertices = h3.h3_to_geo_boundary(hex_index)
        lat = [vertex[0] for vertex in hex_vertices]
        lng = [vertex[1] for vertex in hex_vertices]
        color = color_hex_data[hex_index]
        mean_value = hex_data[hex_index + "_mean"]
        fill_color = matplotlib.colors.rgb2hex(color[:3])
        folium.Polygon(locations=[(lat_i, lng_i) for lat_i, lng_i in
zip(lat, lng)],

                                color='black',
                                weight = 1,
                                fill=True,
                                fill_color=fill_color,
                                opacity=0.9,
                                popup="Precipitaciones: {}".
format(round(mean_value, 2))).add_to(m)

        tooltip = unidad
        my_icon = folium.features.CustomIcon('templates/icono.png', icon_size =
(50,50))
        folium.Marker([Lati, Longi], icon = my_icon, tooltip=tooltip).add_to(m)
        minimap =
MiniMap(toggle_display=True,position="topleft",tile_layer="stamenterrain")

    m.add_child(minimap)
    m.save("iframe.html")
    shutil.move('iframe.html', 'personal/templates/personal/iframe.html')
```

```

        return render(request, 'personal/home.html', {'unidad' : unidad,
'latitud' : Lati, 'longitud' : Longi, 'hora' : moment_indicado, 'radio' :
radio_accion, 'resolucion' : resolucion})

    elif 'Presion' in request.POST.get('submit'):
        unidad = request.POST.get('unidad')
        Lati = request.POST.get('latitud')
        Longi = request.POST.get('longitud')
        moment_indicado = request.POST.get('hora')
        radio_accion = request.POST.get('radio')
        resolucion = request.POST.get('resolucion')

        # Conectarse a la base de datos
        client = pymongo.MongoClient("mongodb+srv://
*****:*****@cluster0.9cawz8x.mongodb.net/GemeloDigitalOceanográfico")
        db = client["GemeloDigitalOceanográfico"]
        collection = db["Presion"]

        time_to_search = moment_indicado
        data = []
        features =
collection.find({"features.properties.days.variables.values.timeInstant":
moment_indicado})
        for feature in features:
            for day in feature['features'][0]['properties']['days']:
                if day['timePeriod']['begin']['timeInstant'] <=
time_to_search <= day['timePeriod']['end']['timeInstant']:
                    for variable in day['variables']:
                        if variable['name'] == 'air_pressure_at_sea_level':
                            longitude =
float(feature['features'][0]['geometry']['coordinates'][0])
                            latitude =
float(feature['features'][0]['geometry']['coordinates'][1])
                            distance = calculate_distance(float(latitude),
float(longitude), float(Lati), float(Longi))
                            if float(distance) <= float(radio_accion): # x es
el radio deseado en kilómetros
                                for value in variable['values']:
                                    if value['timeInstant'] ==
time_to_search:
                                        air_pressure_at_sea_level =
int(value['value'])
                                        data.append({'longitude': longitude,
'latitude': latitude, 'air_pressure_at_sea_level': air_pressure_at_sea_level})

```

```
# Crear un mallado h3 con una resolución de 8
resolution = float(resolucion)
hexagons = []
for point in data:
    lat = point['latitude']
    lng = point['longitude']
    hex_index = h3.geo_to_h3(lat, lng, resolution)
    hexagons.append(hex_index)

# Asignar cada punto de los datos a un hexágono h3
hexagons = list(set(hexagons))
hex_data = {}
for hex_index in hexagons:
    hex_data[hex_index] = []

for point in data:
    lat = point['latitude']
    lng = point['longitude']
    hex_index = h3.geo_to_h3(lat, lng, resolution)
    value = point['air_pressure_at_sea_level']
    hex_data[hex_index].append(value)

# Calcular la media de los datos en cada hexágono
for hex_index in hexagons:
    values = hex_data[hex_index]
    mean_value = sum(values) / len(values)
    hex_data[hex_index + "_mean"] = mean_value

min_value = np.min([hex_data[hex_index + "_mean"] for hex_index in
hexagons])
max_value = np.max([hex_data[hex_index + "_mean"] for hex_index in
hexagons])
value_range = max_value - min_value

color_hex_data = {}
cmap = matplotlib.cm.get_cmap("RdYlBu_r")
for hex_index in hexagons:
    value = hex_data[hex_index + "_mean"]
    color_value = (value - min_value) / value_range
    color = cmap(color_value)
    color = matplotlib.colors.to_rgba(color)
    hex_index = hex_index.replace("_mean", "")
    color_hex_data[hex_index] = color

m = folium.Map(location=[Lati, Longi], tiles= 'cartodbpositron',
zoom_start=13)

for hex_index in hexagons:
    hex_vertices = h3.h3_to_geo_boundary(hex_index)
```

```

lat = [vertex[0] for vertex in hex_vertices]
lng = [vertex[1] for vertex in hex_vertices]
color = color_hex_data[hex_index]
mean_value = hex_data[hex_index + "_mean"]
fill_color = matplotlib.colors.rgb2hex(color[:3])
folium.Polygon(locations=[(lat_i, lng_i) for lat_i, lng_i in
zip(lat, lng)],
                color='black',
                weight = 1,
                fill=True,
                fill_color=fill_color,
                opacity=0.9,
                popup="Presión: {} hpa".format(round(mean_value,
2))).add_to(m)

tooltip = unidad
my_icon = folium.features.CustomIcon('templates/icono.png', icon_size =
(50,50))
folium.Marker([Lati, Longi], icon = my_icon, tooltip=tooltip).add_to(m)
minimap =
MiniMap(toggle_display=True,position="topleft",tile_layer="stamenterrain")

m.add_child(minimap)
m.save("iframe.html")
shutil.move('iframe.html', 'personal/templates/personal/iframe.html')
return render(request, 'personal/home.html', {'unidad' : unidad,
'latitud' : Lati, 'longitud' : Longi, 'hora' : moment_indicado, 'radio' :
radio_accion, 'resolucion' : resolucion})

elif 'Salinidad' in request.POST.get('submit'):
    unidad = request.POST.get('unidad')
    Lati = request.POST.get('latitud')
    Longi = request.POST.get('longitud')
    moment_indicado = request.POST.get('hora')
    radio_accion = request.POST.get('radio')
    resolucion = request.POST.get('resolucion')

    # Conectarse a la base de datos
    client = pymongo.MongoClient("mongodb+srv://
*****:****@cluster0.9cawz8x.mongodb.net/GemeloDigitalOceanográfico")
    db = client["GemeloDigitalOceanográfico"]
    collection = db["Salinidad"]

    time_to_search = moment_indicado
    data = []

```

```
        features =
collection.find({"features.properties.days.variables.values.timeInstant":
moment_indicado})
    for feature in features:
        for day in feature['features'][0]['properties']['days']:
            if day['timePeriod']['begin']['timeInstant'] <=
time_to_search <= day['timePeriod']['end']['timeInstant']:
                for variable in day['variables']:
                    if variable['name'] == 'sea_water_salinity':
                        longitude =
float(feature['features'][0]['geometry']['coordinates'][0])
                        latitude =
float(feature['features'][0]['geometry']['coordinates'][1])
                        distance = calculate_distance(float(latitude),
float(longitude), float(Lati), float(Longi))
                        if float(distance) <= float(radio_accion): # x es
el radio deseado en kilómetros
                            for value in variable['values']:
                                if value['timeInstant'] ==
time_to_search:
                                    sea_water_salinity =
int(value['value'])
                                    data.append({'longitude': longitude,
'latitude': latitude, 'sea_water_salinity': sea_water_salinity})

# Crear un mallado h3 con una resolución de 8
resolution = float(resolucion)
hexagons = []
for point in data:
    lat = point['latitude']
    lng = point['longitude']
    hex_index = h3.geo_to_h3(lat, lng, resolution)
    hexagons.append(hex_index)

# Asignar cada punto de los datos a un hexágono h3
hexagons = list(set(hexagons))
hex_data = {}
for hex_index in hexagons:
    hex_data[hex_index] = []

for point in data:
    lat = point['latitude']
    lng = point['longitude']
    hex_index = h3.geo_to_h3(lat, lng, resolution)
    value = point['sea_water_salinity']
    hex_data[hex_index].append(value)

# Calcular la media de los datos en cada hexágono
```

```

        for hex_index in hexagons:
            values = hex_data[hex_index]
            mean_value = sum(values) / len(values)
            hex_data[hex_index + "_mean"] = mean_value

    min_value = np.min([hex_data[hex_index + "_mean"] for hex_index in
hexagons])
    max_value = np.max([hex_data[hex_index + "_mean"] for hex_index in
hexagons])
    value_range = max_value - min_value

    color_hex_data = {}
    cmap = matplotlib.cm.get_cmap("RdYlBu_r")
    for hex_index in hexagons:
        value = hex_data[hex_index + "_mean"]
        color_value = (value - min_value) / value_range
        color = cmap(color_value)
        color = matplotlib.colors.to_rgba(color)
        hex_index = hex_index.replace("_mean", "")
        color_hex_data[hex_index] = color

    m = folium.Map(location=[Lati, Longi], tiles= 'cartodbpositron',
zoom_start=13)

    for hex_index in hexagons:
        hex_vertices = h3.h3_to_geo_boundary(hex_index)
        lat = [vertex[0] for vertex in hex_vertices]
        lng = [vertex[1] for vertex in hex_vertices]
        color = color_hex_data[hex_index]
        mean_value = hex_data[hex_index + "_mean"]
        fill_color = matplotlib.colors.rgb2hex(color[:3])
        folium.Polygon(locations=[(lat_i, lng_i) for lat_i, lng_i in
zip(lat, lng)],

                                color='black',
                                weight = 1,
                                fill=True,
                                fill_color=fill_color,
                                opacity=0.9,
                                popup="Salinidad: {}".format(round(mean_value, 2))).add_to(m)

    tooltip = unidad
    my_icon = folium.features.CustomIcon('templates/icono.png', icon_size =
(50,50))
    folium.Marker([Lati, Longi], icon = my_icon, tooltip=tooltip).add_to(m)
    minimap =
MiniMap(toggle_display=True,position="topleft",tile_layer="stamenterrain")

```

```
m.add_child(minimap)
m.save("iframe.html")
shutil.move('iframe.html', 'personal/templates/personal/iframe.html')
return render(request, 'personal/home.html', {'unidad' : unidad,
'latitud' : Lati, 'longitud' : Longi, 'hora' : moment_indicado, 'radio' :
radio_accion, 'resolucion' : resolucion})

elif 'Temperatura' in request.POST.get('submit'):
    unidad = request.POST.get('unidad')
    Lati = request.POST.get('latitud')
    Longi = request.POST.get('longitud')
    moment_indicado = request.POST.get('hora')
    radio_accion = request.POST.get('radio')
    resolucion = request.POST.get('resolucion')

    # Conectarse a la base de datos
    client = pymongo.MongoClient("mongodb+srv://
*****:****@cluster0.9cawz8x.mongodb.net/GemeloDigitalOceanográfico")
    db = client["GemeloDigitalOceanográfico"]
    collection = db["Temperatura1"]

    time_to_search = moment_indicado
    data = []
    features =
collection.find({"features.properties.days.variables.values.timeInstant":
moment_indicado})
    for feature in features:
        for day in feature['features'][0]['properties']['days']:
            if day['timePeriod']['begin']['timeInstant'] <=
time_to_search <= day['timePeriod']['end']['timeInstant']:
                for variable in day['variables']:
                    if variable['name'] == 'temperature':
                        longitude =
float(feature['features'][0]['geometry']['coordinates'][0])
                        latitude =
float(feature['features'][0]['geometry']['coordinates'][1])
                        distance = calculate_distance(float(latitude),
float(longitude), float(Lati), float(Longi))
                        if float(distance) <= float(radio_accion): # x es
el radio deseado en kilómetros
                            for value in variable['values']:
                                if value['timeInstant'] ==
time_to_search:
                                    temperature = int(value['value'])
                                    data.append({'longitude': longitude,
'latitude': latitude, 'temperature': temperature})
```

```
# Crear un mallado h3 con una resolución de 8
resolution = float(resolucion)
hexagons = []
for point in data:
    lat = point['latitude']
    lng = point['longitude']
    hex_index = h3.geo_to_h3(lat, lng, resolution)
    hexagons.append(hex_index)

# Asignar cada punto de los datos a un hexágono h3
hexagons = list(set(hexagons))
hex_data = {}
for hex_index in hexagons:
    hex_data[hex_index] = []

for point in data:
    lat = point['latitude']
    lng = point['longitude']
    hex_index = h3.geo_to_h3(lat, lng, resolution)
    value = point['temperature']
    hex_data[hex_index].append(value)

# Calcular la media de los datos en cada hexágono
for hex_index in hexagons:
    values = hex_data[hex_index]
    mean_value = sum(values) / len(values)
    hex_data[hex_index + "_mean"] = mean_value

min_value = np.min([hex_data[hex_index + "_mean"] for hex_index in
hexagons])
max_value = np.max([hex_data[hex_index + "_mean"] for hex_index in
hexagons])
value_range = max_value - min_value

color_hex_data = {}
cmap = matplotlib.cm.get_cmap("RdYlBu_r")
for hex_index in hexagons:
    value = hex_data[hex_index + "_mean"]
    color_value = (value - min_value) / value_range
    color = cmap(color_value)
    color = matplotlib.colors.to_rgba(color)
    hex_index = hex_index.replace("_mean", "")
    color_hex_data[hex_index] = color

m = folium.Map(location=[Lati, Longi], tiles= 'cartodbpositron',
zoom_start=13)

for hex_index in hexagons:
```

```
        hex_vertices = h3.h3_to_geo_boundary(hex_index)
        lat = [vertex[0] for vertex in hex_vertices]
        lng = [vertex[1] for vertex in hex_vertices]
        color = color_hex_data[hex_index]
        mean_value = hex_data[hex_index + "_mean"]
        fill_color = matplotlib.colors.rgb2hex(color[:3])
        folium.Polygon(locations=[(lat_i, lng_i) for lat_i, lng_i in
zip(lat, lng)],
                                color='black',
                                weight = 1,
                                fill=True,
                                fill_color=fill_color,
                                opacity=0.9,
                                popup="Temperatura: {}".format(round(mean_value,
2))).add_to(m)

        tooltip = unidad
        my_icon = folium.features.CustomIcon('templates/icono.png', icon_size =
(50,50))
        folium.Marker([Lati, Longi], icon = my_icon, tooltip=tooltip).add_to(m)
        minimap =
MiniMap(toggle_display=True,position="topleft",tile_layer="stamenterrain")

        m.add_child(minimap)
        m.save("iframe.html")
        shutil.move('iframe.html', 'personal/templates/personal/iframe.html')
        return render(request, 'personal/home.html', {'unidad' : unidad,
'latitud' : Lati, 'longitud' : Longi, 'hora' : moment_indicado, 'radio' :
radio_accion, 'resolucion' : resolucion})

    elif 'Temperatura agua' in request.POST.get('submit'):
        unidad = request.POST.get('unidad')
        Lati = request.POST.get('latitud')
        Longi = request.POST.get('longitud')
        moment_indicado = request.POST.get('hora')
        radio_accion = request.POST.get('radio')
        resolucion = request.POST.get('resolucion')

        # Conectarse a la base de datos
        client = pymongo.MongoClient("mongodb+srv://
*****:*****@cluster0.9cawz8x.mongodb.net/GemeloDigitalOceanográfico")
        db = client["GemeloDigitalOceanográfico"]
        collection = db["Temperatura_Agua"]

        time_to_search = moment_indicado
        data = []
```

```

        features =
collection.find({"features.properties.days.variables.values.timeInstant":
moment_indicado})
    for feature in features:
        for day in feature['features'][0]['properties']['days']:
            if day['timePeriod']['begin']['timeInstant'] <=
time_to_search <= day['timePeriod']['end']['timeInstant']:
                for variable in day['variables']:
                    if variable['name'] == 'sea_water_temperature':
                        longitude =
float(feature['features'][0]['geometry']['coordinates'][0])
                        latitude =
float(feature['features'][0]['geometry']['coordinates'][1])
                        distance = calculate_distance(float(latitude),
float(longitude), float(Lati), float(Longi))
                        if float(distance) <= float(radio_accion): # x es
el radio deseado en kilómetros
                            for value in variable['values']:
                                if value['timeInstant'] ==
time_to_search:
                                    sea_water_temperature =
int(value['value'])
                                    data.append({'longitude': longitude,
'latitude': latitude, 'sea_water_temperature': sea_water_temperature})

# Crear un mallado h3 con una resolución de 8
resolution = float(resolucion)
hexagons = []
for point in data:
    lat = point['latitude']
    lng = point['longitude']
    hex_index = h3.geo_to_h3(lat, lng, resolution)
    hexagons.append(hex_index)

# Asignar cada punto de los datos a un hexágono h3
hexagons = list(set(hexagons))
hex_data = {}
for hex_index in hexagons:
    hex_data[hex_index] = []

for point in data:
    lat = point['latitude']
    lng = point['longitude']
    hex_index = h3.geo_to_h3(lat, lng, resolution)
    value = point['sea_water_temperature']
    hex_data[hex_index].append(value)

```

```
# Calcular la media de los datos en cada hexágono
for hex_index in hexagons:
    values = hex_data[hex_index]
    mean_value = sum(values) / len(values)
    hex_data[hex_index + "_mean"] = mean_value

min_value = np.min([hex_data[hex_index + "_mean"] for hex_index in
hexagons])
max_value = np.max([hex_data[hex_index + "_mean"] for hex_index in
hexagons])
value_range = max_value - min_value

color_hex_data = {}
cmap = matplotlib.cm.get_cmap("RdYlBu_r")
for hex_index in hexagons:
    value = hex_data[hex_index + "_mean"]
    color_value = (value - min_value) / value_range
    color = cmap(color_value)
    color = matplotlib.colors.to_rgba(color)
    hex_index = hex_index.replace("_mean", "")
    color_hex_data[hex_index] = color

m = folium.Map(location=[Lati, Longi], tiles= 'cartodbpositron',
zoom_start=13)

for hex_index in hexagons:
    hex_vertices = h3.h3_to_geo_boundary(hex_index)
    lat = [vertex[0] for vertex in hex_vertices]
    lng = [vertex[1] for vertex in hex_vertices]
    color = color_hex_data[hex_index]
    mean_value = hex_data[hex_index + "_mean"]
    fill_color = matplotlib.colors.rgb2hex(color[:3])
    folium.Polygon(locations=[(lat_i, lng_i) for lat_i, lng_i in
zip(lat, lng)],
                    color='black',
                    weight = 1,
                    fill=True,
                    fill_color=fill_color,
                    opacity=0.9,
                    popup="Temperatura <br> agua:
{:.2f}".format(round(mean_value, 2))).add_to(m)

tooltip = unidad
my_icon = folium.features.CustomIcon('templates/icono.png', icon_size =
(50,50))
folium.Marker([Lati, Longi], icon = my_icon, tooltip=tooltip).add_to(m)
minimap =
MiniMap(toggle_display=True,position="topleft",tile_layer="stamenterrain")
```

```

        m.add_child(minimap)
        m.save("iframe.html")
        shutil.move('iframe.html', 'personal/templates/personal/iframe.html')
        return render(request, 'personal/home.html', {'unidad' : unidad,
'latitud' : Lati, 'longitud' : Longi, 'hora' : moment_indicado, 'radio' :
radio_accion, 'resolucion' : resolucion})

    elif 'Viento' in request.POST.get('submit'):
        unidad = request.POST.get('unidad')
        Lati = request.POST.get('latitud')
        Longi = request.POST.get('longitud')
        moment_indicado = request.POST.get('hora')
        radio_accion = request.POST.get('radio')
        resolucion = request.POST.get('resolucion')

        client = pymongo.MongoClient("mongodb+srv://
*****:*****@cluster0.9cawz8x.mongodb.net/GemeloDigitalOceanográfico")
        db = client["GemeloDigitalOceanográfico"]
        collection = db["Viento"]

        time_to_search = moment_indicado
        data = []
        features =
collection.find({"features.properties.days.variables.values.timeInstant":
moment_indicado})
        for feature in features:
            for day in feature['features'][0]['properties']['days']:
                if day['timePeriod']['begin']['timeInstant'] <=
time_to_search <= day['timePeriod']['end']['timeInstant']:
                    for variable in day['variables']:
                        if variable['name'] == 'wind':
                            longitude =
float(feature['features'][0]['geometry']['coordinates'][0])
                            latitude =
float(feature['features'][0]['geometry']['coordinates'][1])
                            distance = calculate_distance(float(latitude),
float(longitude), float(Lati), float(Longi))
                            if float(distance) <= float(radio_accion): # x es
el radio deseado en kilómetros
                                for value in variable['values']:
                                    if value['timeInstant'] ==
time_to_search:
                                        wind = int(value['moduleValue'])
                                        direction =
int(value['directionValue'])
                                        data.append({'longitude': longitude,
'latitude': latitude, 'wind': wind, 'direction' : direction})

```

```
# Crear un mallado h3 con una resolución de 8
resolution = float(resolucion)
hexagons = []
for point in data:
    lat = point['latitude']
    lng = point['longitude']
    hex_index = h3.geo_to_h3(lat, lng, resolution)
    hexagons.append(hex_index)

# Asignar cada punto de los datos a un hexágono h3
hexagons = list(set(hexagons))
hex_data = {}
for hex_index in hexagons:
    hex_data[hex_index] = []

for point in data:
    lat = point['latitude']
    lng = point['longitude']
    hex_index = h3.geo_to_h3(lat, lng, resolution)
    value = point['wind']
    hex_data[hex_index].append(value)

# Calcular la media de los datos en cada hexágono
for hex_index in hexagons:
    values = hex_data[hex_index]
    mean_valuew = sum(values) / len(values)
    hex_data[hex_index + "_mean"] = mean_valuew

min_value = np.min([hex_data[hex_index + "_mean"] for hex_index in
hexagons])
max_value = np.max([hex_data[hex_index + "_mean"] for hex_index in
hexagons])
value_range = max_value - min_value

#media de la direccion
hexagons = list(set(hexagons))
hex_data1 = {}
for hex_index in hexagons:
    hex_data1[hex_index] = []

for point in data:
    lat = point['latitude']
    lng = point['longitude']
    hex_index = h3.geo_to_h3(lat, lng, resolution)
    value = point['direction']
    hex_data1[hex_index].append(value)
```

```

# Calcular la media de los datos en cada hexágono
for hex_index in hexagons:
    valuesd = hex_data1[hex_index]
    mean_valued = sum(valuesd) / len(valuesd)
    hex_data1[hex_index + "_mean"] = mean_valued

min_value1 = np.min([hex_data1[hex_index + "_mean"] for hex_index in
hexagons])
max_value1 = np.max([hex_data1[hex_index + "_mean"] for hex_index in
hexagons])
value_range1 = max_value1 - min_value1

color_hex_data = {}
cmap = matplotlib.cm.get_cmap("RdYlBu_r")
for hex_index in hexagons:
    value = hex_data[hex_index + "_mean"]
    color_value = (value - min_value) / value_range
    color = cmap(color_value)
    color = matplotlib.colors.to_rgba(color)
    hex_index = hex_index.replace("_mean", "")
    color_hex_data[hex_index] = color

m = folium.Map(location=[Lati, Longi], tiles='cartodbpositron',
zoom_start=13)

for hex_index in hexagons:
    hex_vertices = h3.h3_to_geo_boundary(hex_index)
    lat = [vertex[0] for vertex in hex_vertices]
    lng = [vertex[1] for vertex in hex_vertices]
    color = color_hex_data[hex_index]
    mean_value = hex_data[hex_index + "_mean"]
    mean_valued = hex_data1[hex_index + "_mean"]
    fill_color = matplotlib.colors.rgb2hex(color[:3])
    folium.Polygon(locations=[(lat_i, lng_i) for lat_i, lng_i in
zip(lat, lng)],
                    color='black',
                    weight = 1,
                    fill=True,
                    fill_color=fill_color,
                    opacity=0.9,
                    popup="Viento: {} Km/h Dirección:
{}°".format(round(mean_value, 2), round(mean_valued, 2))).add_to(m)

tooltip = unidad
my_icon = folium.features.CustomIcon('templates/icono.png', icon_size =
(50,50))

```

```
        folium.Marker([Lati, Longi], icon = my_icon, popup=unidad,
tooltip=tooltip).add_to(m)
        minimap =
MiniMap(toggle_display=True,position="topleft",tile_layer="stamenterrain")

        m.add_child(minimap)
        m.save("iframe.html")
        shutil.move('iframe.html', 'personal/templates/personal/iframe.html')
        return render(request, 'personal/home.html', {'unidad' : unidad,
'latitud' : Lati, 'longitud' : Longi, 'hora' : moment_indicado, 'radio' :
radio_accion, 'resolucion' : resolucion})

    else:
        return redirect('home')
else:
    return redirect('home1')
```

A.4.2.4.2 Templates.

A.4.2.4.2.1 Home.

```
{% extends 'base.html' %}

{% block content %}
{% load static %}
<style>
    label{
        width: auto;
        display: contents;
    }
</style>
<div class="row">
    <div class="d-flex flex-column justify-content-end align-items-center col-md-9">
        <h2>Gemelo Digital:</h2>
        <iframe src="{% url 'iframe' %}" name="iframe_principal" width="100%"
height="100%"></iframe>
    </div>
    <div class="d-flex flex-column justify-content-end align-items-center col-md-3">
        <h3>Datos del buque</h3>
        <form method="post" action="{% url 'todo' %}">
            {% csrf_token %}
            <div class="form-group">
                <label for="{% form.unidad.id_for_label %}">Unidad:</label><br>
                {{ form.unidad }}
                <input type="text" name="unidad" id="unidad" size="25" value="{%
unidad %}">
            </div>
            <div class="form-group">
                <label for="{% form.latitud.id_for_label %}">Latitud:</label><br>
                {{ form.latitud }}
                <input type="number" name="latitud" id="latitud" style="width: 100%"
value="{% latitud %}" pattern="/^\d+$/ " min="40" max="50" step="any"/>
            </div>
            <div class="form-group">
                <label for="{% form.longitud.id_for_label %}">Longitud:</label><br>
                {{ form.longitud }}
                <input type="number" name="longitud" id="longitud" style="width:
100%" value="{% longitud %}" pattern="/^\d+$/ " min="-10" max="-6" step="any"/>
            </div>
            <div class="form-group">
                <label for="{% form.hora.id_for_label %}">Hora (2023-02-
07T23:00:00+01):</label><br>
                {{ form.hora }}
                <input type="text" name="hora" id="hora" style="width: 100%"
value="{% hora %}" />
            </div>
        </form>
    </div>
</div>
```

```
<div class="form-group">
  <label for="{ { form.radio.id_for_label } }">Radio de accion
(km):</label><br>
  {{ form.radio }}
  <input type="number" name="radio" id="radio" style="width:
100%"value="{ { radio } }" max="5" min="1" pattern="/^\d+$/">
</div>
<div class="form-group">
  <label for="{ { form.resolucion.id_for_label } }">Resolución mallado H3
(7-10):</label><br>
  {{ form.resolucion }}
  <input type="number" name="resolucion" id="resolucion" style="width:
100%" value="{ { resolucion } }" max="10" min="7" pattern="/^\d+$/">
</div>
  <a><button value="Batimetria" type="submit" class="list-group-item
list-group-item-action" name="submit" >Batimetria</button></a>
  <a><button value="Humedad" type="submit" class="list-group-item list-
group-item-action" name="submit">Humedad</button></a>
  <a><button value="Nubosidad" type="submit" class="list-group-item
list-group-item-action" name="submit">Nubosidad</button></a>
  <a><button value="Oleaje" type="submit" class="list-group-item list-
group-item-action" name="submit" >Oleaje</button></a>
  <a><button value="Precipitaciones" type="submit" class="list-group-
item list-group-item-action" name="submit">Precipitaciones</button></a>
  <a><button value="Presion" type="submit" class="list-group-item list-
group-item-action" name="submit">Presion</button></a>
  <a><button value="Salinidad" type="submit" class="list-group-item
list-group-item-action" name="submit">Salinidad</button></a>
  <a><button value="Temperatura" type="submit" class="list-group-item
list-group-item-action" name="submit">Temperatura</button></a>
  <a><button value="Temperatura agua" type="submit" class="list-group-
item list-group-item-action disabled" name="submit">Temperatura agua</button></a>
  <a><button value="Viento" type="submit" class="list-group-item list-
group-item-action" name="submit">Viento</button></a>
</form>
</div>
</div>

{% endblock content %}
```

A.4.2.4.2.2 Homes.

```
{% extends 'base.html' %}

{% block content %}
<div class="pricing-header px-3 py-3 pt-md-5 pb-md-4 mx-auto text-center">
  <h1 class="display-5">Gemelo Digital Oceanográfico</h1>
  <p class="lead">El Gemelo digital oceanográfico es una herramienta que potencia
la capacidad de prevision y acción del usuario, así como una correcta interpretación
del entorno marino.</p>
</div>

<div class="container">
  <div class=" d-flex flex-column">
    <h3>Bienvenido!</h3>
    <br>
    <h5>Querido usuario, para poder hacer uso de los modelos digitales,
regístrese primero o inicie sesión!</h5>
    <p>
      <a class="p2 text-dark" href="{% url 'register' %}">Regístrate!</a>      |
      <a class="p2 text-dark" href="{% url 'login' %}">Login!</a><br>
    </p>
  </div>
</div>

{% endblock content %}
```

A.4.2.5 *Manage.*

```
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys

def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'mysite.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)

if __name__ == '__main__':
    main()
```