



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Diseño y desarrollo de una aplicación Android para la gestión de amarres en puertos deportivos

Grado en Ingeniería Mecánica

ALUMNO: Carmen Názara Besada

DIRECTORES: Belén Barragáns Martínez

Norberto Fernández García

CURSO ACADÉMICO: 2019-2020

Universida_{de}Vigo



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Diseño y desarrollo de una aplicación Android para la gestión de amarres en puertos deportivos

Grado en Ingeniería Mecánica
Intensificación en Tecnología Naval
Cuerpo General

Universida_{de}Vigo

RESUMEN

Desde que las comunicaciones VHF se introdujeron obligatoriamente para todas las embarcaciones del espacio marítimo internacional, se establecieron como medio de comunicación único entre los puertos deportivos y los patrones de los buques para realizar diferentes gestiones, como puede ser el reservar una plaza de amarre. Pero, aunque este es el medio de comunicación que se ha asumido para este cometido, presenta algunos inconvenientes, como son la falta de privacidad, la vulnerabilidad ante interferencias o el limitado alcance.

En este TFG se propone diseñar, desarrollar y validar una aplicación móvil en Android que permita implementar un método alternativo para dichas comunicaciones. Con esta aplicación se pretende poder gestionar el amarre del barco del usuario, de manera similar a cómo se reserva una habitación de hotel o un vuelo cuando vamos a viajar, pero adaptándolo a las necesidades particulares de los barcos y sus patrones.

PALABRAS CLAVE

Aplicación móvil, Android, Amarre, Puerto deportivo, Comunicación

AGRADECIMIENTOS

En primer lugar, agradecer a mi familia el apoyo continuo que recibí de su parte, no solo durante los meses trabajando en este proyecto, sino también durante los cinco años en la ENM, que supusieron un reto tan grande para ellos como para mí. Gracias.

En segundo lugar, quería agradecer a la profesora D^a Belén Barragáns Martínez el haber aceptado mi idea para este TFG y ayudarme a llevarlo a cabo. Y agradecerle enormemente también el haber decidido tutorizarme junto con el profesor D. Norberto Fernández García, pieza clave para salir de todos los atascos con los que me encontré durante el desarrollo de la aplicación. Gracias.

Gracias a mi promoción, por estar mano a mano cuando en algunas semanas se nos oscureció el cielo. Y por hacerme sentir, este año y siempre, como una hermana para ellos. Gracias.

Por Luisete.

CONTENIDO

Contenido	1
Índice de Figuras	3
Índice de Tablas.....	5
1 Introducción y objetivos	7
1.1 Los puertos deportivos en la Ría de Pontevedra	7
1.2 Comunicaciones entre los puertos deportivos y los patrones.....	8
1.3 Objetivos del TFG.....	9
1.4 Organización de la memoria	9
2 Estado del arte	11
2.1 Revisión de aplicaciones existentes	11
2.1.1 ASNAUGA - Guía Náutica de Galicia	11
2.1.2 Navily	12
2.1.3 Otras aplicaciones	13
2.2 Comparativa	14
2.3 Herramientas usadas en este proyecto.....	15
2.3.1 Aplicación móvil	15
2.3.2 Android	15
2.3.3 Android Studio.....	19
2.3.4 Java	20
2.3.5 Parse Server	21
2.3.6 Motorola G7 Play	22
3 Diseño de la aplicación.....	23
3.1 Diseño de la interfaz.....	23
3.1.1 Diseño de la interfaz para el primer acceso a la aplicación	23
3.1.2 Diseño de la interfaz para los accesos consecutivos a la aplicación.....	24
3.2 Diseño de la base de datos	26
3.3 Instalación de Parse Server	28
3.4 Comunicación con Parse Server.....	29
4 Desarrollo de la aplicación	31
4.1 Estructura de una aplicación Android	31
4.2 Ficheros de la aplicación.....	32
4.2.1 AndroidManifest.....	32
4.2.2 Build.gradle	33

4.2.3 Ficheros de recursos	34
4.2.4 Activities.....	36
5 Prueba y validación	49
5.1 Propósito de la aplicación	49
5.2 Métodos de comprobación	50
5.2.1 Creación de reservas ficticias	50
5.2.2 Comandos Logcat	53
6 Conclusiones y líneas futuras	55
6.1 Conclusiones	55
6.2 Líneas futuras	55
7 Bibliografía.....	57
Anexo I: <i>Scripts</i> de Python para la comprobación de datos de la base de datos de Parse Server	61
Anexo II: Código Java de la aplicación.....	65

ÍNDICE DE FIGURAS

Figura 1-1 Vista aérea de la Ría de Pontevedra [Autoría propia]	7
Figura 1-2 Hombre utilizando un equipo VHF en su embarcación de recreo [6].....	8
Figura 2-1 Opción de reserva para el Real Club de Mar de Agüete (opción no operativa) [Autoría propia]	12
Figura 2-2 Opción de reserva para el Real Club Náutico de La Coruña (opción aparentemente operativa) [Autoría propia]	12
Figura 2-3 Actividades principales de <i>Navily</i> [Autoría propia]	13
Figura 2-4 Pantalla principal de <i>MarinaAlicante</i> y formulario para reservar amarre [Autoría propia]..	13
Figura 2-5 Pantalla principal de <i>Greenwich</i> y opción de llamar al marinero de guardia [Autoría propia]	14
Figura 2-6 Cuota de uso de sistemas operativos móviles en España [18]	16
Figura 2-7 HTC Dream: primer smartphone Android [25].....	17
Figura 2-8 Arquitectura del sistema operativo Android [29].....	18
Figura 2-9 Vista <i>Project</i> de Android Studio [33]	19
Figura 2-10 James Gosling, creador de Java [41]	21
Figura 2-11 Parse como BaaS [47]	21
Figura 2-12 Motorola G7 Play cedido por el CUD [Autoría propia].....	22
Figura 3-1 Diseño de la interfaz para el primer acceso a la aplicación [Autoría propia]	23
Figura 3-2 Diseño de la interfaz para los siguientes accesos a la aplicación [Autoría propia].....	24
Figura 3-3 Embarcaciones atracadas <i>a la mediterránea</i> [49]	25
Figura 3-4 Diseño de la interfaz con los retrocesos incluidos [Autoría propia]	26
Figura 4-1 Vistas <i>Android</i> y <i>Project</i> de la aplicación [Autoría propia]	32
Figura 4-2 <i>AndroidManifest.xml</i> de la aplicación [Autoría propia]	33
Figura 4-3 Elementos más importantes del <i>build.gradle</i> [Autoría propia]	34
Figura 4-4 Ficheros de recursos de la aplicación [Autoría propia]	34
Figura 4-5 Vistas <i>design</i> y <i>blueprint</i> de <i>layout</i> [Autoría propia]	35
Figura 4-6 Código de <i>layout</i> [Autoría propia]	35
Figura 4-7 <i>Strings</i> de la aplicación [Autoría propia]	36
Figura 4-8 Imagen utilizada en la actividad <i>Splash</i> [Autoría propia]	37
Figura 4-9 Extracto del código de la actividad <i>Splash</i> [Autoría propia]	37
Figura 4-10 Búsquedas en la base de datos local dentro de la actividad <i>Splash</i> [Autoría propia]	38
Figura 4-11 Código de la actividad <i>App</i> [Autoría propia]	38
Figura 4-12 Extracto del código de la actividad <i>Configuración</i> [Autoría propia].....	39
Figura 4-13 Código para comprobar e imprimir los datos del patrón de la base de datos local [Autoría propia]	39

Figura 4-14 <i>Layout</i> de la actividad <i>Registrar Patrón</i> [Autoría propia]	40
Figura 4-15 <i>Toast</i> de la actividad <i>Registrar Patrón</i> [Autoría propia]	40
Figura 4-16 <i>Layout</i> de la actividad <i>Registrar Embarcación</i> [Autoría propia].....	41
Figura 4-17 <i>Toast</i> de la actividad <i>Registrar Embarcación</i> [Autoría propia]	41
Figura 4-18 Extracto del código con las búsquedas iniciales en la actividad <i>Principal</i> [Autoría propia]	42
Figura 4-19 <i>Layout</i> de la actividad <i>Principal</i> [Autoría propia]	42
Figura 4-20 <i>Layout</i> de la actividad <i>Selección de fechas</i> [Autoría propia]	43
Figura 4-21 Código para implementar uno de los <i>DatePicker</i> de la actividad <i>Selección de fechas</i> [Autoría propia].....	43
Figura 4-22 <i>Layout</i> de la actividad <i>Filtrado de servicios</i> [Autoría propia]	44
Figura 4-23 Funcionamiento de uno de los filtros de la actividad <i>Filtrado de servicios</i> [Autoría propia]	44
Figura 4-24 <i>Layout</i> de la actividad <i>Listado de plazas disponibles</i> [Autoría propia]	45
Figura 4-25 Extracto del código de la actividad <i>Listado de plazas disponibles</i> [Autoría propia].....	45
Figura 4-26 Extracto de un <i>intent</i> de la actividad <i>Listado de plazas disponibles</i> [Autoría propia].....	45
Figura 4-27 Extracto del código de un <i>getIntent</i> de la actividad <i>Información de la plaza seleccionada</i> [Autoría propia].....	46
Figura 4-28 <i>Layout</i> de la actividad <i>Información de la plaza</i> [Autoría propia]	46
Figura 4-29 Base de datos de los amarres en Parse Server (extracto del <i>script</i> enviar.py) [Autoría propia]	46
Figura 4-30 Extracto del código de la actividad <i>Información del puerto al que pertenece la plaza</i> [Autoría propia].....	47
Figura 4-31 <i>Layout</i> de la actividad <i>Información del puerto al que pertenece la plaza</i> [Autoría propia]	47
Figura 4-32 Base de datos de los puertos en Parse Server (extracto del <i>script</i> enviar.py) [Autoría propia]	48
Figura 4-33 <i>Layout</i> de la actividad <i>Final</i> [Autoría propia].....	48
Figura 5-1 Extracto del código de la actividad <i>Splash</i> que permite borrar la base de datos local de Parse Server [Autoría propia]	50
Figura 5-2 Comparación del diseño del <i>layout</i> (vistas <i>design</i> y <i>blueprint</i>) con la interfaz visualizada en el emulador para la actividad <i>Información del puerto al que pertenece la plaza</i> [Autoría propia]	51
Figura 5-3 Comprobación de correcta impresión en el <i>EditText</i> de la fecha de salida seleccionada en el <i>DatePicker</i> [Autoría propia]	52
Figura 5-4 Extracto del código de la actividad <i>Fechas</i> [Autoría propia].....	53
Figura A1-1 Script <i>consultar.py</i> ejecutado en el terminal	63
Figura A1-2 Script <i>borrarID.py</i> ejecutado en el terminal	63

ÍNDICE DE TABLAS

Tabla 2-1 Comparativa entre aplicaciones	15
Tabla 2-2 Ficha técnica Motorola G7 Play.....	22
Tabla 3-1 Base de datos de embarcaciones [Autoría propia]	27
Tabla 3-2 Base de datos de amarres [Autoría propia]	27
Tabla 3-3 Base de datos de las reservas [Autoría propia]	27
Tabla 3-4 Base de datos de los puertos deportivos [Autoría propia]	28

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Los puertos deportivos en la Ría de Pontevedra

La Ría de Pontevedra (Figura 1-1) es una de las tres principales entradas del mar de las Rías Baixas y es la más grande de todas las de Galicia, con 145 km² de superficie [1]. Esta Ría alberga nueve puertos deportivos y clubs náuticos. Las cinco instalaciones náuticas principales (Sanxenxo, Combarro, Portonovo, Pontevedra y Agüete) suman más de 1100 plazas de amarre, consiguiendo alcanzar cuotas de ocupación de hasta un 90% en temporada alta [2]. Sabiendo que el tráfico anual portuario en las Rías Baixas alcanza cifras superiores a los 9000 [3], se percibe la importancia que tiene este sector del tráfico marítimo en esta parte de la costa gallega.



Figura 1-1 Vista aérea de la Ría de Pontevedra [Autoría propia]

Se debe puntualizar que, por la cercanía a la Escuela Naval Militar y por conocimiento de la Ría de Pontevedra y sus puertos, se emplearán estos puertos como referencia a lo largo de este documento. Esto no significa que la aplicación se circunscriba a dichos puertos de la Ría de Pontevedra.

1.2 Comunicaciones entre los puertos deportivos y los patrones

Actualmente, la mayoría de los puertos deportivos, incluidos los de la Ría de Pontevedra, tienen como método de comunicación (entre el patrón del barco que llega a puerto y el personal de dicho puerto) las comunicaciones vía VHF (*Very High Frequency*), en ocasiones difíciles de llevar a cabo por las interferencias o por el idioma, además de que son comunicaciones abiertas carentes de privacidad.

Según el Real Decreto 1185/2006 del Ministerio de Fomento [4], todas las embarcaciones deben llevar a bordo al menos un equipo VHF (como el de la Figura 1-2) para poder hacer llamadas de emergencia, comunicarse con otros buques o para poder hablar por el canal 16 (canal establecido por el Sistema Mundial de Socorro y Seguridad Marítima como el canal mundial de uso exclusivo al socorro, urgencia, seguridad y llamadas). También quedan establecidos otros canales predeterminados, como el canal 70 (codificado para la emisión y recepción en exclusiva de alertas mediante Llamada Selectiva Digital), el canal 6 (internacionalmente establecido para las comunicaciones de seguridad y salvamento en las que intervienen embarcaciones y aviones) así como el canal 9, que es el canal cuyo uso está limitado a establecer comunicaciones entre los clubes náuticos y deportivos y las embarcaciones [5].



Figura 1-2 Hombre utilizando un equipo VHF en su embarcación de recreo [6]

Así pues, desde la introducción de estos sistemas de comunicación en las embarcaciones de recreo, las comunicaciones puerto-barco han sido siempre por este medio. Estas comunicaciones presentan una serie de inconvenientes reseñables:

- Son comunicaciones abiertas, en las cuales no existe la privacidad, pues cualquiera que sintonice la frecuencia del canal utilizado desde su radio, puede escuchar la conversación que se está manteniendo.
- Están afectadas por las interferencias, tanto del ruido ambiente como de otras personas que hablen a la vez.
- Tienen un alcance aproximado de la línea visual (hasta el horizonte), el cual supone unas siete millas náuticas.

- Hacen que, en ocasiones, mantener una conversación se complique si emisor y receptor no hablan el mismo idioma.

Podemos inferir que las comunicaciones VHF son necesarias y útiles para las distintas situaciones en las que se puede ver la dotación de un barco, pero que no son las ideales para comunicarse con el puerto de destino cuando, por ejemplo, se desea atracar en el mismo.

1.3 Objetivos del TFG

Justificándolo con lo expuesto en el apartado anterior, en este TFG se propone un método alternativo para las citadas comunicaciones que se establecen entre las administraciones de los clubes náuticos o puertos deportivos y los patrones de las embarcaciones con el fin de amarrar en el puerto de destino. Este TFG consiste en diseñar, desarrollar y validar una aplicación móvil en Android en la que se pueda gestionar el amarre de una embarcación de manera similar a cómo se reserva una habitación de hotel o un vuelo antes de viajar, pero adaptándolo a las necesidades particulares de los barcos y sus patrones.

1.4 Organización de la memoria

La presente memoria se organiza de la manera que se explica a continuación.

Comienza presentando la introducción y objetivos, apartado en el que se contextualiza al lector para adentrarlo en la materia a tratar. En el capítulo 2, se expone el estado del arte, donde se presentan las diferentes aplicaciones existentes en el mercado que tienen funcionalidades similares a las que tendrá la aplicación que se pretende desarrollar. Asimismo, se presentan los recursos materiales empleados para llevar a cabo la aplicación. Los capítulos 3 y 4 se dedican a exponer el diseño de la aplicación (tanto la interfaz como la base de datos) y el desarrollo de la aplicación, donde se materializa mediante código lo que previamente se ha diseñado. En el capítulo 5, se detalla la validación de la aplicación, donde se demuestra que cumple los objetivos marcados. El capítulo 6 presenta las conclusiones y líneas futuras. Finalmente, se recogen las referencias bibliográficas consultadas durante la elaboración del presente trabajo.

Los anexos incluidos en esta memoria incluyen los *scripts* utilizados para conectar Android Studio con Parse Server y el código de cada actividad de la aplicación desarrollada.

2 ESTADO DEL ARTE

Como parte del proceso de documentación para realizar este TFG, se ha investigado qué hay en la actualidad que pueda asemejarse a la aplicación que se pretende desarrollar. Asimismo, se ha buscado la mejor plataforma para llevarla a cabo. Los resultados de ambos estudios se exponen en el presente capítulo, que finaliza presentando los recursos materiales empleados para desarrollar el trabajo, tanto *hardware* como *software*.

2.1 Revisión de aplicaciones existentes

Se exponen y detallan a continuación una serie de aplicaciones existentes en el mercado que tienen fines similares a los que se buscan con la aplicación a desarrollar.

2.1.1 ASNAUGA - Guía Náutica de Galicia

Esta aplicación (que se puede obtener en [7] para dispositivos Android y en [8] para dispositivos iOS) es creación de ASNAUGA (Asociación de Náuticos de Galicia) [9] y, con el fin principal de brindar información actualizada sobre los puertos deportivos de Galicia, consta de tres actividades:

- *Clubes náuticos y puertos deportivos*. Presenta todos los clubes náuticos y puertos deportivos pertenecientes a la asociación. Al seleccionar cada uno de ellos, presenta información sobre el mismo (dirección, teléfono, correo electrónico, página web, coordenadas GPS, cartas náuticas que lo incluyen, amplitud de la bocana, calado máximo y tipo de puerto) además de un listado de servicios (vestuarios, cafetería, etc.), la información disponible en el derrotero correspondiente, información turística (hoteles cercanos y sitios para visitar), previsión meteorológica (que nos direcciona al apartado de pronóstico meteorológico de [9]) y el apartado de reservas, que es el que nos interesa. Al seleccionar este último apartado, nos permite indicar, primero, la fecha de llegada y la fecha de salida y, a continuación, el tipo de plaza de amarre (según la longitud de la plaza). Cabe reseñar que esta opción de reserva no está operativa para ninguno de los puertos deportivos de la Ría de Pontevedra, como se puede observar en el ejemplo de la Figura 2-1. Probando para reservar un amarre en el Puerto Deportivo de La Coruña, nos permite cubrir todos los datos (véase la Figura 2-2) pero, tras llamar por teléfono a la sede del puerto, concluyen que ni reciben ni tramitan reservas de amarre a través de esta aplicación móvil.
- *Permisos Parque Nacional Illas Atlánticas*, que direcciona a la página web en [10], donde se tramitan los permisos para poder fondear en alguna de las islas que forman el Parque Nacional de las Islas Atlánticas de Galicia.
- *Rutas náutico turísticas*, opción no operativa por el momento.

Seleccione la fecha de llegada:

31	octubre	2017
1	noviembre	2018
2	diciembre	2019
3	enero	2020
4	febrero	2021
5	marzo	2022
6	abril	2023

Seleccione la fecha de salida:

1	octubre	2017
2	noviembre	2018
3	diciembre	2019
4	enero	2020
5	febrero	2021
6	marzo	2022
7	abril	2023

En las fechas seleccionadas disponemos de los siguientes tipos de plazas:

Siguiente

Figura 2-1 Opción de reserva para el Real Club de Mar de Agüete (opción no operativa) [Autoría propia]

Seleccione la fecha de llegada:

31	octubre	2017
1	noviembre	2018
2	diciembre	2019
3	enero	2020
4	febrero	2021
5	marzo	2022
6	abril	2023

Seleccione la fecha de salida:

1	octubre	2017
2	noviembre	2018
3	diciembre	2019
4	enero	2020
5	febrero	2021
6	marzo	2022
7	abril	2023

En las fechas seleccionadas disponemos de los siguientes tipos de plazas:

- Plazas de 12 m.
- Plazas de 15 m.
- Plazas de 18 m.
- Plazas de 30 x 7.05 metros
- Plazas de 30 x 11.25 metros

Siguiente

Paso 4: Datos personales

*Nombre:

*Apellidos:

NIF:

Teléfono:

*Correo electrónico:

Nombre de barco:

Matricula de barco:

Siguiente

Figura 2-2 Opción de reserva para el Real Club Náutico de La Coruña (opción aparentemente operativa) [Autoría propia]

2.1.2 Navily

Esta aplicación (que se puede descargar en [11] para dispositivos Android y en [12] para dispositivos iOS) es de descarga gratuita al igual que la anterior y ofrece al usuario la posibilidad de reservar plazas de amarre en 650 puertos de toda Europa. Esta aplicación presenta cuatro actividades principales (véase Figura 2-3):

- *Mapa.* Muestra los puertos y fondeaderos que están suscritos a la aplicación. De los 9 puertos deportivos de la Ría de Pontevedra solo presenta a 3 de ellos.
- *Reservas.* Según la ubicación actual, muestra al usuario los puertos deportivos más cercanos, así como el precio del amarre según las dimensiones de la embarcación (datos que previamente introduce el usuario). Al hacer *click* en cada uno de los puertos ofrecidos, ofrece información meteorológica, de servicios prestados por el puerto, distancia a la que se encuentra, teléfono de contacto, calado máximo... así como comentarios de usuarios que accedieron a ese puerto previamente.

Cuando se selecciona un puerto para realizar la reserva, permite escoger los días y, tras haber introducido la tarjeta de crédito con la que se quiere pagar, finaliza la reserva.

- *Offline*. Ofrece una suscripción anual en la que presta servicios *offline* (acceso a información de los fondeaderos o puertos que el usuario hubiese guardado previamente) pagando cierta cantidad de dinero.
- *Perfil*. Permite modificar los datos del usuario, los datos de su embarcación, de su tarjeta de crédito, así como ayuda para manejar la aplicación.

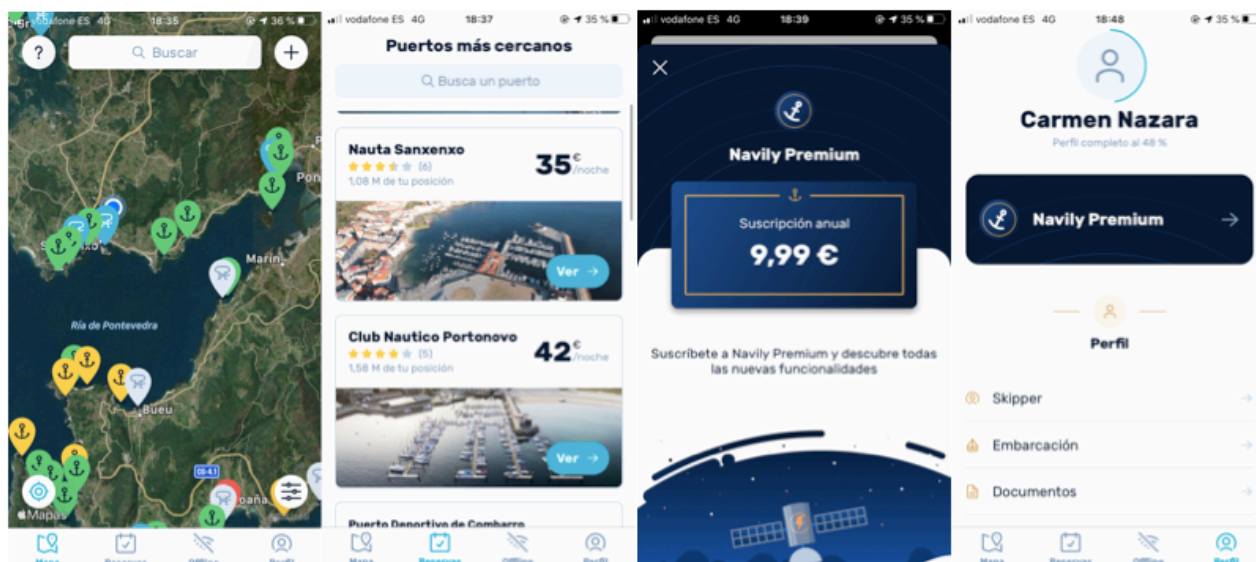


Figura 2-3 Actividades principales de *Navily* [Autoría propia]

2.1.3 Otras aplicaciones

Existen otras aplicaciones relacionadas con puertos deportivos concretos. Destacan:

- *MarinaAlicante* (que se puede descargar en [13] para dispositivos Android y en [14] para dispositivos iOS). Es la aplicación oficial del puerto deportivo de Alicante. Además de presentar los servicios del puerto o noticias relacionadas con el mismo, permite obtener información personalizada sobre los amarres introduciendo las medidas de la embarcación (véase la Figura 2-4).



Figura 2-4 Pantalla principal de *MarinaAlicante* y formulario para reservar amarre [Autoría propia]

- *Greenwich*. Aplicación del puerto deportivo Marina Greenwich (que se puede descargar en [15] para dispositivos Android y en [16] para dispositivos iOS) de Altea (Alicante). Da información general del puerto deportivo (servicios, avisos a los navegantes...) y de sus alrededores. Permite llamar directamente al marinerero de guardia (véase Figura 2-5).

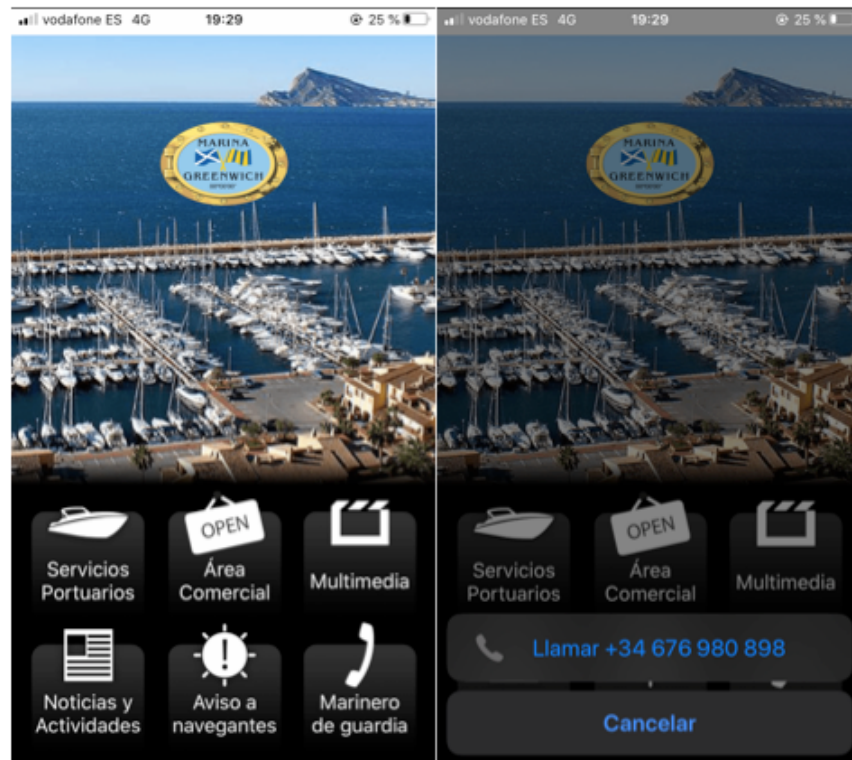


Figura 2-5 Pantalla principal de *Greenwich* y opción de llamar al marinerero de guardia [Autoría propia]

2.2 Comparativa

Teniendo en cuenta que las aplicaciones más destacables (por sus características y funciones) son ASNAUGA y *Navily*, se expone en la Tabla 2-1 una comparativa de las funcionalidades que presenta cada una de ellas. Se comparan ambas aplicaciones con la que se pretende desarrollar (de nombre *Appmarre*).

Valorando los resultados de la tabla comparativa podemos concluir que, globalmente y valorando con el mismo peso cada punto comparando, las aplicaciones evaluadas están prácticamente igualadas. Centrándonos en los puntos que no se abordarán en la aplicación a desarrollar en este TFG:

- Previsión meteorológica: no se considera necesario incluirla, pues es de fácil y mejor consulta en otras aplicaciones específicas.
- Idiomas: inicialmente se desarrollará la aplicación en un único idioma, pero se valorará la opción de traducirla a otros idiomas una vez que se consiga validar con éxito.
- Comentarios: se puede considerar en caso de que la aplicación alcance cierto auge en el número de usuarios.

Funcionalidades	ASNAUGA	Navily	Appmarre
Información básica sobre los puertos ¹	SÍ	SÍ	SÍ
Información básica de la plaza seleccionada ²	SÍ	NO	SÍ
Información ampliada de la plaza seleccionada ³	NO	NO	SÍ
Previsión meteorológica	SÍ	SÍ	NO
Pide fechas de llegada y salida para reservar	SÍ	SÍ	SÍ
Permite elegir la plaza de amarre que prefiera el usuario	SÍ	NO	SÍ
Pide datos del usuario y de su barco para realizar la reserva ⁵	SÍ	SÍ	SÍ
Permite usar la aplicación en más de un idioma ⁶	SÍ	SÍ	NO
Muestra comentarios y valoraciones de usuarios previos	NO	SÍ	NO
Permite pagar con tarjeta online	NO	SÍ	NO

Tabla 2-1 Comparativa entre aplicaciones

¹Información básica de los puertos: dirección, servicios y contacto.

²Eslora y manga de la plaza de amarre.

³Calado, situación respecto al club náutico, resguardo del viento, tipo de amarre (normal / a la mediterránea).

⁴Las solicitudes de amarre son recibidas y tramitadas por la administración de todos los puertos que ofrece la aplicación.

⁵Pide nombre, apellidos, DNI y teléfono/correo del usuario y nombre y matrícula del barco para realizar la reserva.

⁶ASNAUGA ofrece la aplicación en español, inglés y francés. *Navily* permite usar la aplicación en los mismos idiomas y también en italiano y alemán. *Amárrate* se desarrollará sólo en español.

2.3 Herramientas usadas en este proyecto

En este apartado se explicarán las herramientas utilizadas en el presente TFG.

2.3.1 Aplicación móvil

Desde el primer momento en el que se pensó en el desarrollo de esta aplicación, sólo se contempló llevarla a cabo para su uso en dispositivos móviles, aunque es cierto que hoy en día los negocios tienen dos formas para estar en los dispositivos móviles: mediante aplicaciones web o aplicaciones móviles. Sin embargo, las ventajas que tienen las aplicaciones móviles con respecto a las webs son numerosas [17], destacando las siguientes:

- Un usuario puede estar conectado a la aplicación en cualquier lugar y momento sin necesidad de usar un navegador, escribir dirección web o usar buscadores, lo que aumenta la eficiencia y reduce costes.
- Si la aplicación ofrece de forma sencilla y cómoda un servicio a sus clientes, sirve para generar fidelidad, frente a otras en las que el acceso a estos servicios sea más complicado o no dispongan de aplicación móvil.

2.3.2 Android

Teniendo claro lo expuesto en el apartado anterior, debemos justificar por qué se decidió desarrollar la aplicación para dispositivos con sistema operativo Android.

Existen múltiples opciones de sistemas operativos móviles, siendo los más reconocidos Windows Mobile, iOS y Android. A nivel nacional, son los dos últimos los que más se utilizan, como se puede observar en la Figura 2-6, que muestra los sistemas operativos móviles más utilizados en España en el último año, entre los que destacan iOS y Android.

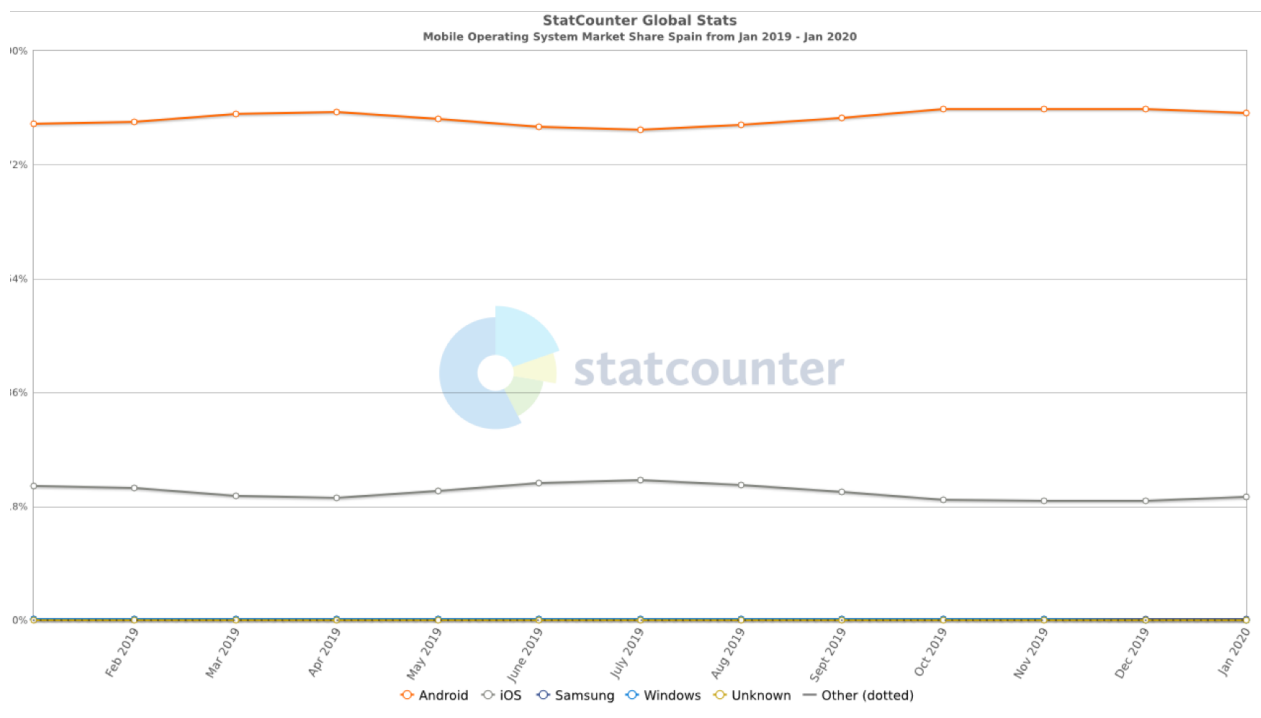


Figura 2-6 Cuota de uso de sistemas operativos móviles en España [18]

Centrándonos en España, la cuota de uso del sistema operativo Android se ha visto reducida en el 2019, quedándose en el 80,83% del total de los *smartphones* utilizados a nivel nacional. iOS, por su parte, llega al 18,92% de cuota de uso en este último año [18].

Android no sólo es el sistema operativo móvil más utilizado a nivel nacional, sino también a nivel internacional, con un 74,13% de cuota de uso – por encima de iOS (con un 24,79% de cuota de uso) [19]. El elemento más diferenciador de Android es que es una plataforma de código abierto y cualquier elemento puede ser provisionado por el conjunto de usuarios de su comunidad.

Su extensivo uso a nivel nacional e internacional es lo que lleva a que nos decantemos por desarrollar la aplicación para dispositivos con sistema operativo Android antes que para dispositivos con el sistema operativo iOS. Pero veamos algunas características [20] que los diferencian:

- El lenguaje de programación. Mientras que Android utiliza Java [21] o Kotlin [22], iOS usa Objective-C [23] o Swift [24].
- Fase de validación. Aunque el emulador iOS es mucho más rápido que su equivalente para Android, este último es más realista debido a su máquina virtual con una CPU virtualizada.
- Interfaz gráfica. Mientras Apple prioriza las animaciones suaves, complejas y potentes, Google tiene el hardware como principal objetivo. A nivel técnico, el desarrollador monta ficheros XML (*Extensible Markup Language*) en las interfaces de Android y ficheros XIB (*XML Interface Builder*) en iOS.
- ‘Volver atrás’, funcionalidad en el desarrollo que existe en Android, pero no en iOS.

Es por lo previamente expuesto por lo que esta aplicación se enfocará a dispositivos con el sistema operativo Android.

Entrando en la historia del sistema operativo elegido, Android Inc. nace a finales de 2003 [25] con cuatro miembros fundadores: Andy Rubin, Rich Miner, Nick Sears y Chris White, los cuales

presentaban Android como un sistema operativo avanzado para cámaras digitales que permitía conectarlas con el PC sin cables; pero viendo que el tema no tenía mucho futuro, se decantaron por los teléfonos móviles. A los dos años, momento en el que Symbian y Windows Mobile tenían proyectos similares, Google decide comprar Android Inc. por 50 millones de dólares, erigiéndose como tercer actor en esta competición. La idea era contar con un software flexible y actualizable que se adecuara a distintos dispositivos. En 2007 ocurre lo inesperado: Apple se adelanta a los tres desarrolladores y lanza el iPhone con su propio sistema operativo iOS que incluía reproductor de música, navegador web, gestor de correo y gestor de mensajes de texto y voz. El primer *Android phone* nace en 2008 con el HTC Dream (Figura 2-7), convirtiéndose en un rival directo del iOS de Apple y que integraba servicios populares de Google como Google Maps o Youtube.



Figura 2-7 HTC Dream: primer smartphone Android [25]

En cuanto a las características del sistema operativo Android, destaca que utiliza código abierto, posee un núcleo basado en Linux; es adaptable a muchas pantallas y resoluciones; utiliza SQLite [26] para el almacenamiento de datos; ofrece diferentes formas de mensajería, posee soporte de Java pero también de HTML [27] o Adobe Flash Player [28]; incluye un emulador de dispositivos, herramientas de memoria y análisis del rendimiento del software; incluye también un catálogo de aplicaciones para descargar en el dispositivo; posee Bluetooth y multitarea real de aplicaciones.

Su arquitectura [29] se basa, como se puede apreciar en la Figura 2-8, en:

- Núcleo Linux. Para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores, Android depende de Linux.
- *Runtime* de Android. Incluye un conjunto de bibliotecas base que proporciona la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje Java. Cada aplicación Android tiene su propio proceso, con su propia instancia de la máquina virtual.
- Librerías. Incluye un conjunto de bibliotecas de C/C++ usadas por varios componentes del sistema.
- Armazón de aplicaciones. Los desarrolladores tienen acceso a las mismas APIs (Interfaces de Programación de Aplicaciones) del *framework* usado por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes: cualquier aplicación puede publicar sus capacidades y cualquier otra puede hacer uso de las mismas.
- Aplicaciones. Incluye el correo electrónico, servicio de SMS (*Short Message Service*), calendario, mapas, navegador, contactos..., todas ellas escritas en Java.



Figura 2-8 Arquitectura del sistema operativo Android [29]

El sistema operativo Android ha ido evolucionando desde su creación hasta el día de hoy, pasando por diferentes versiones que han ido mejorando el funcionamiento de los dispositivos Android. Destacan las siguientes versiones [30]:

- **Android 1.6 Donut.** Esta versión puso la información fácilmente accesible con el cuadro de búsqueda rápida.
- **Android 2.1 Eclair.** Esta versión permitió que las pantallas de alta densidad mostrasen impresionantes fondos de pantalla animados que respondían al tocarlos. Incluyó, además, la navegación detallada para la conducción e información sobre el tráfico en tiempo real directamente en el teléfono.
- **Android 2.2 Froyo.** Esta versión introdujo los teléfonos ultrarrápidos que se podían controlar por voz y que podían tener conexión a Internet en cualquier situación gracias a las zonas WiFi.
- **Android 2.3 Gingerbread,** que ofrecía una experiencia Android más sencilla y rápida tanto para los usuarios como para los desarrolladores. Se implementaron nuevos métodos novedosos a los juegos, la duración de la batería era mayor y la compatibilidad con NFC (*Near-Field Communication*) dio lugar a una nueva categoría de aplicaciones.
- **Android 3.0 Honeycomb.** Esta versión marcó el inicio de la era de las tabletas con una interfaz sencilla que incluía imágenes grandes y navegación en pantalla fluida.
- **Android 4.0 Ice Cream Sandwich.** Esta versión incrementó la personalización y el control de usuario, permitiendo personalizar la pantalla de inicio, definir la cantidad de datos que el usuario quisiese utilizar y compartir contenido al instante en cualquier momento.
- **Android 4.1 Jelly Bean,** con la cual se inició la asistencia móvil personalizada con *Google Now*. Permitía interactuar aún más con las notificaciones y utilizar varias cuentas de usuario en un mismo dispositivo.
- **Android 4.4 Kitkat.** Esta versión introdujo el control por voz. Diciendo *Ok Google*, inicia una búsqueda por voz, envía un mensaje de texto, obtiene indicaciones o reproduce canciones.
- **Android 5.0 Lollipop,** con la que Android llega a pantallas de otros dispositivos distintos de teléfonos móviles o tabletas, como relojes, televisores o coches. Esta versión incluye *Material Design*, que ofrece un diseño atractivo y una respuesta táctil fluida.

- Android 6.0 *Marshmallow*. Esta versión ofrece accesos directos sencillos y respuestas inteligentes, la batería dura mucho más tiempo y los nuevos permisos de las aplicaciones permiten tener un mayor control de la información.
- Android 7.0 *Nougat*, con la que llegan la multiventana, un mejor sistema de notificaciones y la pantalla encendida al levantar el dispositivo móvil.
- Android 8.1 *Oreo*. Esta versión ofrecía notificaciones mejoradas (posibilidad de ordenarlas por categorías o definir la frecuencia con que se reciben) e innovaciones en el diseño (como la unificación de los iconos de los desarrolladores o la simplificación para añadir sonidos de timbre, alarma o notificación).
- Android 9.0 *Pie*, que presentó novedades como la inteligencia artificial, mejoras en la cámara, batería adaptativa, soporte para pantallas con *notch*, nuevos métodos de autenticación o la desactivación de notificaciones.
- Android 10 que, lanzada en septiembre de 2019, incluye novedades [31] como el modo oscuro en todo el sistema, el *Focus Mode*, el *Live Caption*, el control por intensidad de presión en pantalla o el soporte nativo de reconocimiento facial 3D.
- Android 11, cuyos avances de desarrollo se empezaron a publicar en febrero de 2020 y cuya liberación para el consumidor final se espera para julio de 2020 [32].

2.3.3 Android Studio

Android Studio [33] es el entorno principal de desarrollo de aplicaciones para Android, basado en IntelliJ IDEA, pero incluyendo más funcionalidades que aumentan la productividad cuando se desarrollan aplicaciones para Android. En este entorno de desarrollo, se encuentran todas las herramientas necesarias para crear aplicaciones, desde el código hasta el diseño de la interfaz de usuario [34].

Cada proyecto de Android Studio incluye módulos con archivos de código fuente y archivos de recursos. Existen diferentes tipos de módulos: módulos de aplicaciones de Android, módulos de biblioteca y módulos de *Google App Engine*. De manera predeterminada, se muestran los archivos en la vista de *Project* de Android (véase Figura 2-9), la cual está organizada en módulos para que se pueda acceder rápidamente a los archivos fuente más importantes del proyecto.

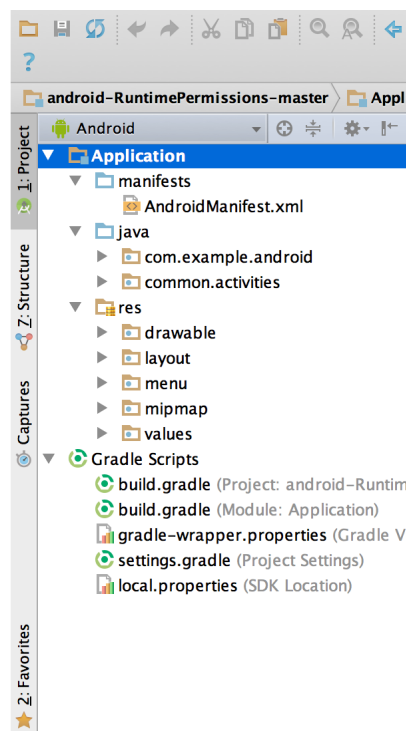


Figura 2-9 Vista *Project* de Android Studio [33]

Cada módulo de aplicación contiene las siguientes carpetas:

- *Manifests*: Contiene el archivo *AndroidManifest.xml*, que describe información esencial de la aplicación para las herramientas de creación de Android, el sistema operativo Android y Google Play. Debe declarar el nombre del paquete de la aplicación, sus componentes, los permisos que necesita la aplicación para acceder a las partes protegidas del sistema o a otras aplicaciones, etc.
- *Java*: Contiene los archivos de código fuente Java separados por nombres de paquetes.
- *Res*: Contiene todos los recursos (archivos adicionales y contenido estático) sin código, como diseños XML y *strings*.

Android Studio ofrece un sistema de compilación flexible basado en Gradle [35], que se ejecuta en una herramienta integrada desde el menú de Android Studio para personalizar, configurar y extender el proceso de compilación; crear diferentes APK (*Android Application Package*) para una misma aplicación o volver a usar códigos y recursos en conjuntos de archivos fuente.

Ayuda también a depurar y mejorar el rendimiento del código. Usa la depuración integrada para verificar referencias, expresiones y valores de variables y ofrece generadores de perfiles de rendimiento para realizar un seguimiento más fácil del uso de la CPU y de la memoria de la aplicación, encontrar objetos asignados, ubicar pérdidas de memoria, optimizar el rendimiento de los gráficos y analizar las solicitudes de red.

2.3.4 Java

Dentro del software de trabajo, Android Studio, se puede trabajar con lenguaje Java o Kotlin. Java es un lenguaje estable, estándar, orientado a objetos, ampliamente utilizado en el mundo de la programación y, además, Android está escrito en Java junto con C y C++. Kotlin es un lenguaje de programación estático que se ejecuta en una máquina virtual Java (JVM, *Java Virtual Machine*) y que se presenta más conciso, eficaz y rápido que Java [36]. Aunque Kotlin ahorra tiempo y esfuerzos en el desarrollo de una aplicación, Java lleva mucho más tiempo en el mercado, lo que proporciona más confianza con una amplia comunidad de desarrolladores consolidados que ofrecen documentación e información.

El lenguaje de programación Java es un lenguaje moderno, publicado por Sun Microsystems [37] en la última mitad de 1995. A pesar de esta juventud, ha ganado muchos adeptos rápidamente por muy diversas razones, como es la posibilidad de añadir programas a una página web de Internet [38].

Java comenzó siendo un proyecto conocido como *Oak*, creado por James Gosling (Figura 2-10) en 1991. El objetivo de Gosling era crear una máquina virtual con un lenguaje con notación similar al lenguaje C, pero que fuese más uniforme y sencillo que C o C++. La primera implantación fue Java 1.0 en 1995. Con la promesa de *Write Once, Run Anywhere* (*escríbelo una vez y ejecútalo en cualquier lugar*), ofrecía períodos de prueba en plataformas de gran popularidad. Los navegadores de mayor importancia lo incorporaron rápidamente como un *applet* (programa que puede incrustarse en un documento HTML [39]) de seguridad configurable. Poco después, se diseñaron nuevas versiones orientadas a plataformas empresariales (Java2EE) y dispositivos limitados (Java2ME) que acabaron siendo conocidos como Java2 [40].

Pero Java2, además de un lenguaje, era una plataforma de desarrollo de programas [38] que constaba de:

- Un lenguaje de programación: el lenguaje Java, del mismo nombre que la plataforma.
- Un conjunto de bibliotecas (*JavaCore*) estándar que se incluyen con la plataforma y que debe existir en cualquier entorno Java. Estas bibliotecas comprenden: *strings*, procesos, entrada y salida, fecha y hora, *applets*, seguridad, acceso a bases de datos, etc.
- Un conjunto de herramientas para el desarrollo de programas. Entre ellas cabe citar el compilador de Java a *bytecodes* o el *javadoc*.

- Un entorno de ejecución cuyo principal componente es una máquina virtual para poder ejecutar los programas en código de bytes.



Figura 2-10 James Gosling, creador de Java [41]

La versión más actual de Java es Java 8, desde abril de 2015, encontrándose actualmente en su actualización número 241 [42].

2.3.5 Parse Server

Para el desarrollo de la parte correspondiente al servidor, en esta aplicación se utiliza Parse Server [43]. Parse Server es un servidor de aplicaciones capaz de simplificar diversas funcionalidades necesarias para el *backend* de aplicaciones móviles y aplicaciones web, ofreciéndonos lo que se conoce como BaaS (*Backend as a Service*), es decir, un *backend* que ahorra muchas de las tareas de programación del lado del servidor (Figura 2-11). Es compatible con iOS, Android, JavaScript y Windows, entre otros; y su motor está basado en el *framework* Express [44] de Node, por lo que requiere tener instalado NodeJS [45] (entorno de ejecución para JavaScript) y MongoDB [46] (base de datos distribuida) en el servidor.

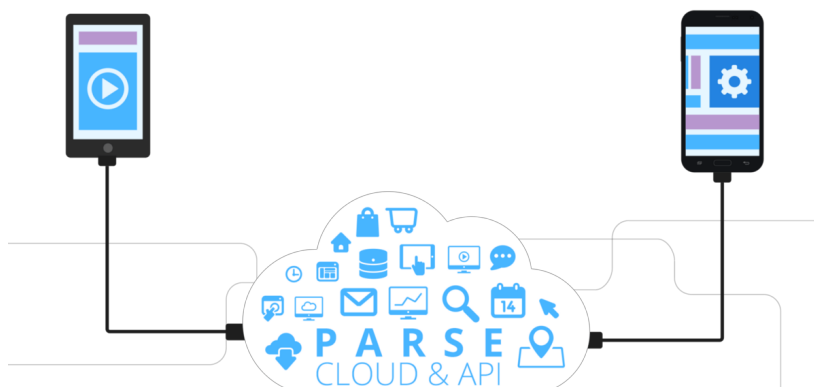


Figura 2-11 Parse como BaaS [47]

Las funcionalidades más atractivas de Parse Server son:

- Una base de datos en tiempo real (*LiveQuery*).
- Un sistema de autenticación de usuarios y la posibilidad de configurar la seguridad de la información.
- Servidor de ficheros.
- Servidor para notificaciones.
- Diversos SDK para programar aplicaciones en múltiples plataformas, como Web JavaScript, iOS (Objective-C/Swift) o Android (Java).
- Implementación de la API REST (API que utiliza el protocolo REST para intercambiar y manipular datos en los servicios de Internet) para el acceso a la información mediante cualquier medio que soporte conexiones HTTP.

2.3.6 Motorola G7 Play

El teléfono móvil donde se realizarán las pruebas y validaciones (además de en el emulador de Android Studio) será el proporcionado por el CUD: el Motorola G7 Play (Figura 2-12), cuyas características se presentan a continuación.



Figura 2-12 Motorola G7 Play cedido por el CUD [Autoría propia]

El Motorola G7 Play es el teléfono más básico de la gama media que Motorola presentó en febrero de 2019. Con *notch* convencional, cuenta con una sola cámara trasera y otra frontal, todo controlado por el procesador Snapdragon 632 [48]. Su ficha técnica se puede ver en la Tabla 2-2.

Dimensiones y peso	147,3x71,5x7,99 / 149 g
Pantalla	5,7' con resolución HD+
Procesador	Snapdragon 632
Memoria RAM	2GB
Almacenamiento interno	32 GB ampliables a 512 GB
Cámara trasera	13 MP
Cámara delantera	8 MP
Batería	3000mAh
Sistema operativo	Android 9 Pie
Conectividad	WiFi, Bluetooth
Otros	Lector de huellas trasero, USB tipo C
Precio	149 euros

Tabla 2-2 Ficha técnica Motorola G7 Play

De esta ficha técnica cabe destacar que su sistema operativo es Android 9 Pie, por lo que el mínimo SDK que se seleccionará para la aplicación debe ser igual o menor a este nivel de API.

3 DISEÑO DE LA APLICACIÓN

En el presente capítulo se presenta el diseño de la interfaz de la aplicación, así como el diseño de la base de datos del servidor.

3.1 Diseño de la interfaz

La aplicación presenta dos interfaces principales: la interfaz para la primera vez que el usuario accede a la aplicación y la interfaz para los accesos consecutivos.

3.1.1 Diseño de la interfaz para el primer acceso a la aplicación

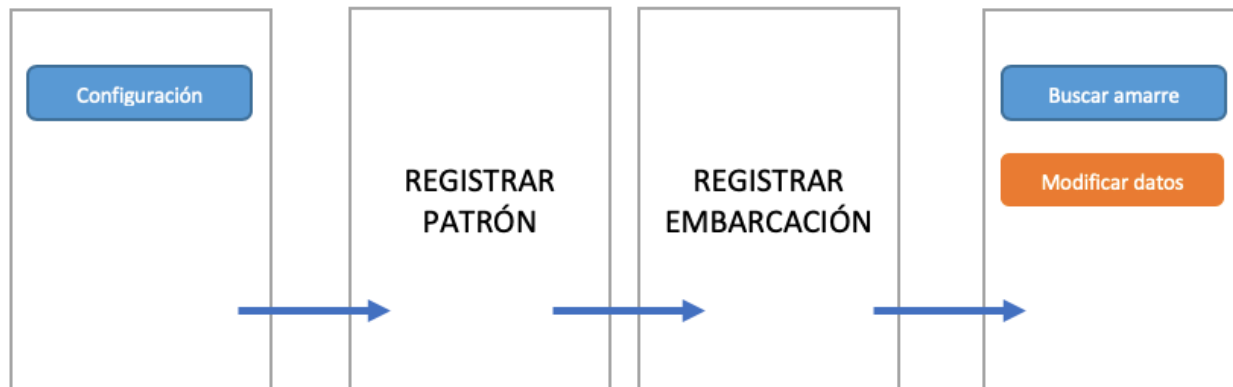


Figura 3-1 Diseño de la interfaz para el primer acceso a la aplicación [Autoría propia]

Como se puede observar en la Figura 3-1, la primera vez que el usuario abre la aplicación, solo tendrá la posibilidad de configurarla. Esta configuración consiste en introducir los datos del patrón y de su embarcación:

- Datos del patrón. El usuario deberá introducir el nombre, los apellidos, el DNI, el correo electrónico y el número de teléfono móvil.
- Datos de la embarcación. Se requiere el nombre de la embarcación, su eslora, manga y calado y su matrícula.

A continuación, se le direcciona a la pantalla principal que le recibirá en las visitas posteriores que haga a la aplicación. Esta pantalla presenta dos opciones: *buscar amarre* (proceso que se explica en el *Diseño de la interfaz para los accesos consecutivos a la aplicación*) y *modificar datos* (volvería a los

registros del patrón y la embarcación), donde puede tanto comprobar que los datos se han guardado con éxito como modificarlos si se ha equivocado introduciendo algún dato.

3.1.2 Diseño de la interfaz para los accesos consecutivos a la aplicación

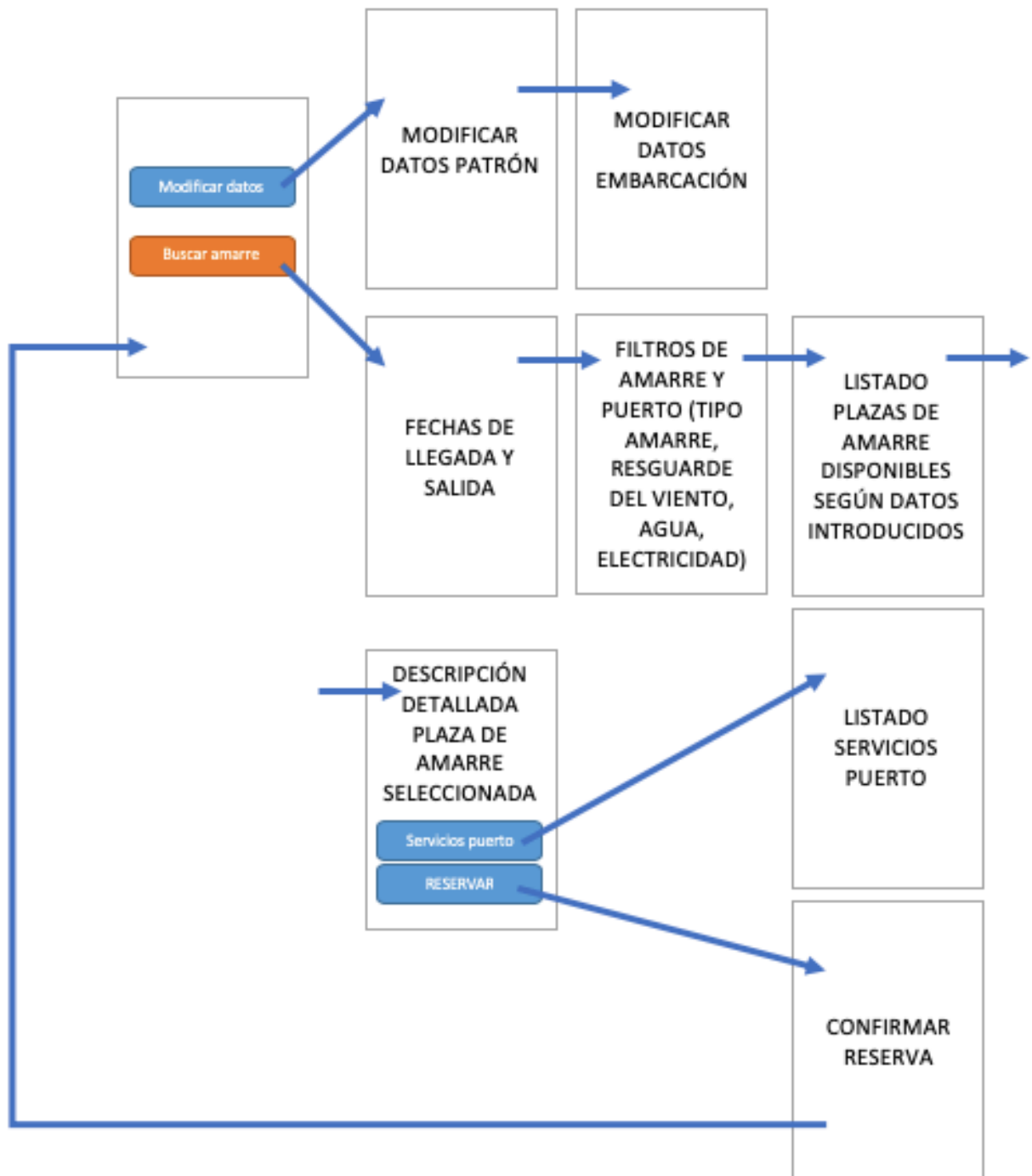


Figura 3-2 Diseño de la interfaz para los siguientes accesos a la aplicación [Autoría propia]

Como se explicó en el apartado anterior y como se puede apreciar en la Figura 3-2, tras el primer uso de la aplicación, el usuario tendrá dos opciones:

- *Modificar datos.* Al seleccionar esta opción, el usuario vuelve al registro del patrón y de la embarcación, donde podrá modificar los datos de ambos o comprobar que son correctos.

- *Buscar amarre*. Al seleccionar esta opción, el usuario pasa al proceso de reserva que se describe a continuación.

Cuando el usuario selecciona la opción de *Buscar amarre*, dicha búsqueda utilizará los datos del patrón y de la embarcación que se introdujeron la primera vez que se abrió la aplicación – o que se modificaron posteriormente – y direcciona al usuario a la selección de las fechas de llegada y salida del puerto.

Después de seleccionar ambas fechas, se le presenta en pantalla una serie de filtros para su búsqueda: deberá seleccionar si desea o no las siguientes características para su plaza de amarre:

- Plaza de amarre con tipo de amarre *a la mediterránea* (tipo de amarre que se puede observar en la Figura 3-3).
- Plaza de amarre resguardada del viento, para aquellos puertos que no tengan espigones que resguarden del viento a todas sus plazas de amarre.
- Plaza de amarre con aprovisionamiento cercano de agua.
- Plaza de amarre con aprovisionamiento cercano de corriente eléctrica.



Figura 3-3 Embarcaciones atracadas a la mediterránea [49]

Tras realizar la búsqueda, se mostrará en pantalla una lista de las plazas de amarre disponibles teniendo en cuenta los datos introducidos y los filtros seleccionados.

Cuando el usuario selecciona una de las plazas de amarre que se le ofrecen, se le presenta una descripción detallada de la misma. Esta descripción incluye: el número de la plaza y el puerto donde se encuentra, la eslora, manga y calado de la plaza, las características de la plaza de amarre (las que se citaron en el filtrado) y la distancia al Club Náutico (para que el usuario sepa si tendrá que andar mucho para salir de los pantalanes). Además, podrá visualizar también la lista de características específicas del puerto en el que se encuentra (si posee WiFi, vigilancia, vestuarios, etc.), que son servicios comunes a todas sus plazas de amarre.

Si el usuario está conforme con todo lo que le ofrece la plaza de amarre, seleccionará la opción de *Reservar*. Esta acción supondrá el fin de todo el proceso de reserva e implicará el envío de todos los datos de la reserva al servidor de la aplicación.

En todo este proceso, el usuario podrá retroceder a la actividad anterior, tal como se muestra a través de las flechas rojas en la Figura 3-4. Asimismo, tras confirmar la reserva, se le redirigirá directamente a la pantalla inicial.

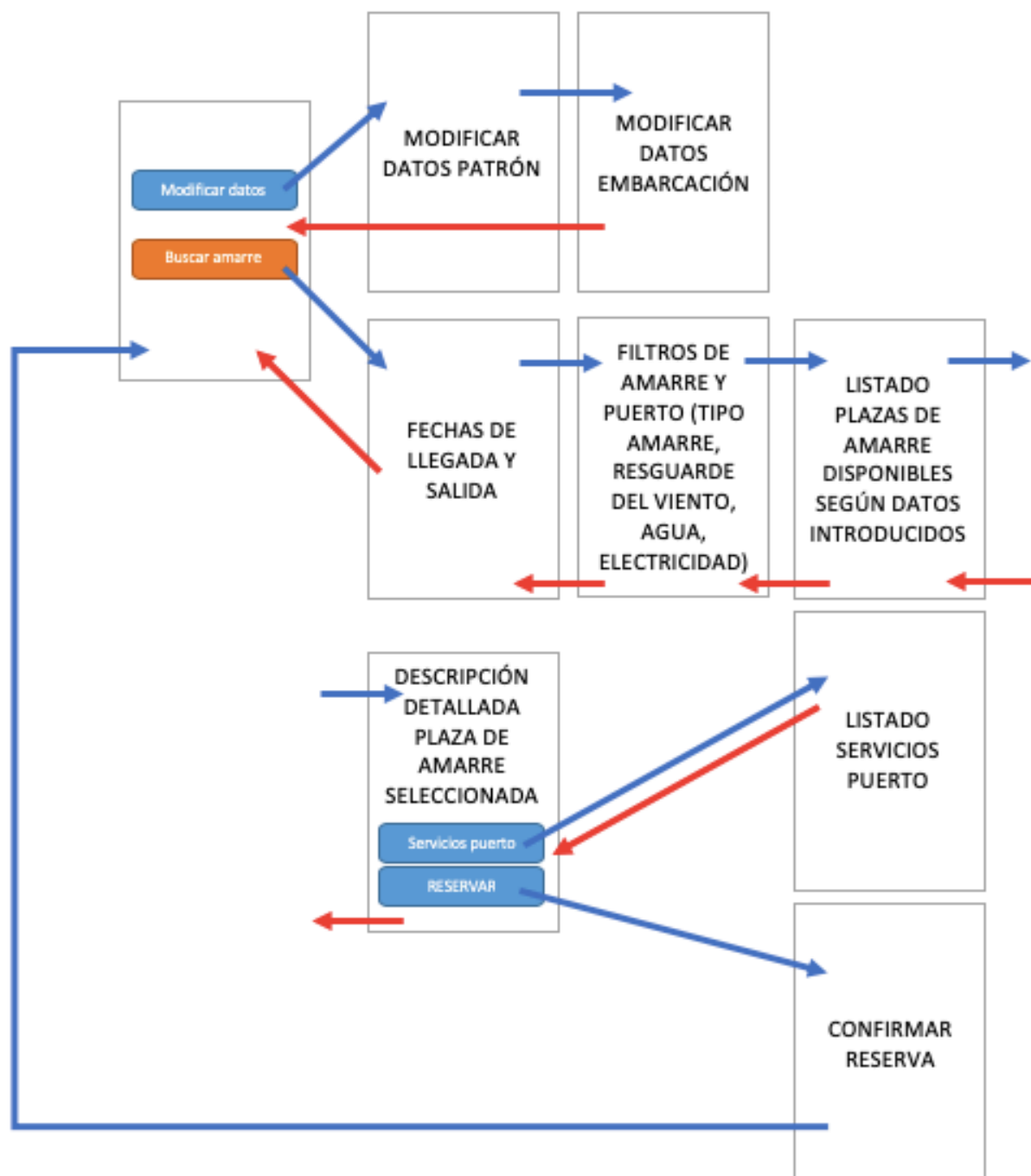


Figura 3-4 Diseño de la interfaz con los retrocesos incluidos [Autoría propia]

3.2 Diseño de la base de datos

Para el correcto desarrollo de la aplicación, es fundamental un diseño adecuado de la base de datos que almacena toda la información relevante. Este diseño consiste en la elaboración de diferentes tablas que incluyen los datos de cuatro entidades: patrón, embarcación, puerto y amarre. Así, habrá una tabla para la información de cada embarcación que haga una reserva (Tabla 3-1), una tabla para la información de la plaza de amarre (Tabla 3-2), otra tabla para la información cada una de las reservas (Tabla 3-3) y otra tabla para la información de los puertos en los que se encuentren las plazas de amarre (Tabla 3-4).

Nombre	Matrícula ¹	Eslora ² [m]	Manga ³ [m]	Calado ⁴ [m]

Tabla 3-1 Base de datos de embarcaciones [Autoría propia]

¹La clave primaria de la embarcación será su matrícula. La matriculación de las embarcaciones está regulada por el Real Decreto 1027 del 28 de julio de 1989 [50] y establece que este identificativo único de cada barco es un conjunto alfanumérico de ocho caracteres (dos letras y seis números que indican la lista de registro, la provincia marítima, el distrito marítimo, el folio donde consta y el año de matriculación). Así pues, el valor a introducir será una cadena de ocho caracteres.

²La eslora vendrá definida por un número entero [m].

³La manga vendrá definida por un número entero [m].

⁴El calado vendrá definido por un número entero [m].

Nº de la plaza ¹	Puerto ¹	Eslora ²	Manga ²	Calado ²	Distancia al Club Náutico ³ [m]	Agua ⁴	Electricidad ⁴	Resguardado del viento ⁴	Ataque a la mediterránea ⁴

Tabla 3-2 Base de datos de amarres [Autoría propia]

¹La identificación del amarre será doble: el número de la plaza y el puerto donde se encuentre (cadena de caracteres).

²La eslora, manga y calado de cada plaza de amarre vendrán expresados por números enteros [m].

³La distancia al Club Náutico será un número entero [m].

⁴Estas características se representarán en la tabla con variables de tipo booleano.

Nombre barco ¹	Matrícula ²	Fecha inicio ³	Fecha fin ³	Nombre ⁴	Apellidos ⁴	DNI ⁵	Teléfono ⁶	Correo usuario ⁷

Tabla 3-3 Base de datos de las reservas [Autoría propia]

¹El nombre del barco será el introducido en el registro.

²Al igual que el nombre, la matrícula estará relacionada con los datos que haya sobre la embarcación en la base de datos de las embarcaciones.

³Las fechas de inicio y fin de la reserva serán números de tipo *long*.

⁴El nombre y los apellidos serán cadenas de caracteres, introducidos en el registro del patrón.

⁵El DNI será una cadena de ocho dígitos (ejemplo: 123456789).

⁶El número de teléfono será un número entero guardado en forma de cadena.

⁷El correo será una cadena de caracteres.

Nombre puerto ¹	Dirección ²	Servicios ³				

Tabla 3-4 Base de datos de los puertos deportivos [Autoría propia]

¹La clave primaria de cada puerto será el nombre del propio puerto, que será una cadena de caracteres.

²La dirección vendrá dada por una cadena de un máximo de cincuenta caracteres.

³Cada uno de los 11 servicios definidos tendrá asociado un valor booleano (verdadero o falso) según si dicho servicio sea proporcionado o no por el puerto: aseos, duchas, carburante, WiFi, reciclaje, recuperación de aguas residuales, vídeo vigilancia, vigilante nocturno, lavandería, venta de hielo, acastillaje.

3.3 Instalación de Parse Server

Para poder hacer uso de Parse Server, fue necesario instalar una serie de elementos en el ordenador donde se desarrolló la aplicación.

Primeramente, fue necesaria la instalación de MongoDB [51] y de NodeJS [52]. Para comprobar que la instalación del primero fue correcta se ejecuta en el terminal el siguiente comando:

```
mongo.exe
```

Para comprobar que NodeJS se instaló correctamente se ejecuta en el terminal el siguiente comando:

```
node -v
```

La ejecución de este comando imprimirá en el terminal la versión de NodeJS instalada en el ordenador.

A continuación, se procedió a la instalación del servidor de Parse [53]. Se comprueba también la correcta instalación del programa ejecutándolo en el terminal los siguientes comandos (el terminal debe estar ubicado en la carpeta donde se descomprimió el archivo ZIP descargado):

```
npm install
```

```
npm audit fix
```

Posteriormente, se instala el paquete NPM de Parse haciendo:

```
npm install -g parse-server
```

La siguiente operación es abrir con un editor de texto el fichero *index.js* que se encuentra entre los documentos incluidos en el ZIP descargado. Casi al principio de este archivo aparece lo siguiente:

```
appId: process.env.APP_ID || 'myAppId',  
masterKey: process.env.MASTER_KEY || '',  
serverURL: process.env.SERVER_URL || 'http://localhost:1337/parse'
```

Se debe configurar el APP_ID y la MASTER_KEY, ya que estos datos serán los utilizados para relacionar la base de datos de Parse Server con la aplicación creada en Android Studio. Una vez configurado, se lanza el servidor (desde la carpeta donde están guardados los archivos de Parse Server) con el comando:

```
npm start
```


3.4 Comunicación con Parse Server

Como se citó anteriormente, las bases de datos utilizadas en esta aplicación se almacenan en Parse Server. Con el fin de modificar las bases de datos, así como comprobar que la información introducida se almacena con éxito y sin fallos, se crearon *scripts* de Python [54] que, a través del terminal del ordenador que se utiliza como servidor, permiten la comunicación del terminal con las bases de datos de Parse Server.

El código de los *scripts* utilizados se encuentra en el Anexo I: *Scripts* de Python para la comprobación de datos de la base de datos de Parse Server.

4 DESARROLLO DE LA APLICACIÓN

En este capítulo se expondrá la estructura típica de una aplicación Android, particularizándola para la aplicación diseñada en este proyecto, detallándose el proceso de codificado de cada actividad con referencias al código utilizado y al diseño empleado. Todo el código que compone la aplicación se encuentra en el Anexo II: Código Java de la aplicación.

4.1 Estructura de una aplicación Android

Los directorios que forman un proyecto de Android se pueden visualizar de diferentes maneras, siendo las vistas *Android* y *Project* [33] las más utilizadas. La primera, cuyo diseño se puede apreciar en la parte izquierda de la Figura 4-1, fue la más utilizada durante el desarrollo de la aplicación. Esta vista está organizada en módulos para que el creador de la aplicación pueda acceder rápidamente a los archivos fuente clave del proyecto. Cada módulo app contiene las siguientes carpetas:

- *Manifests*: contiene el archivo *AndroidManifest.xml*, que describe la información esencial de la aplicación.
- *Java*: contiene los archivos de código fuente Java.
- *Res* (carpeta de recursos): contiene todos los recursos sin código. Algunos de los recursos que contiene este directorio son:
 - *Drawable*: son archivos de imágenes (.png, .jpg, .gif, etc.).
 - *Layout*: son archivos XML que definen el diseño de una interfaz de usuario.
 - *Values*: son archivos XML que contienen valores simples, como *strings*, valores enteros y colores.

La estructura real del proyecto de Android en el disco se presenta en la vista *Project* (imagen de la derecha de la Figura 4-1), cuyos directorios más importantes son:

- *Build*: contiene los resultados de la compilación.
- *Libs*: contiene bibliotecas privadas.
- *Src*: contiene todos los archivos de código y recursos para el módulo en los siguientes directorios:
 - *AndroidTest*: contiene código para las pruebas de instrumentación que se ejecutan en un dispositivo Android, es decir, son pruebas que se ejecutan en la JVM.
 - *Main*: contiene los archivos fuente principales, es decir, el código y recursos de Android como el *AndroidManifest.xml*, *java* o *res*.
 - *Test*: contiene código para pruebas locales. Estas son pruebas que se ejecutan en un dispositivo o emulador de *hardware*.

- *Build.gradle*: ficheros que definen la configuración de compilación influyendo, por ejemplo, en la mínima versión SDK de la aplicación o en el nombre de la versión de la aplicación.

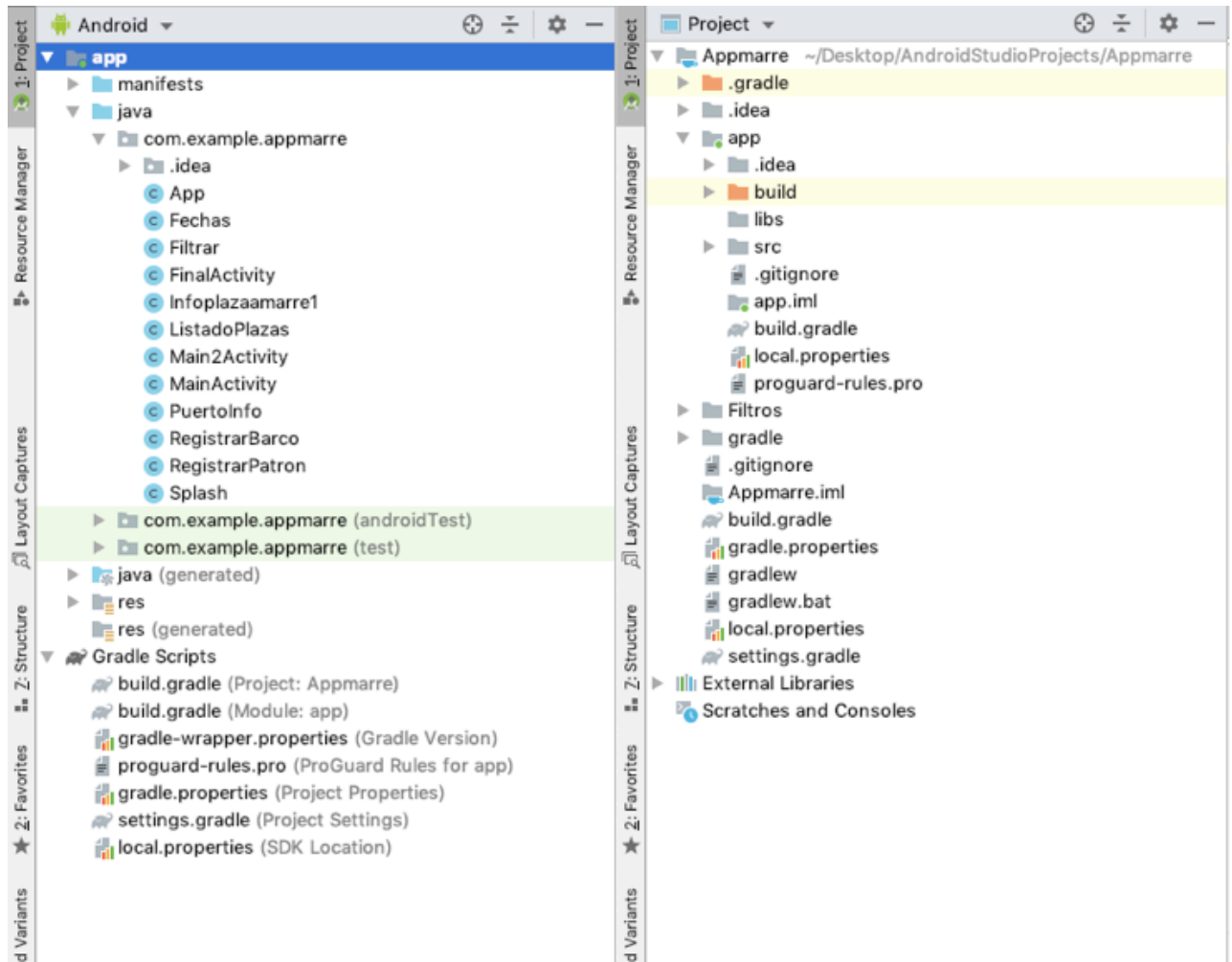


Figura 4-1 Vistas *Android* y *Project* de la aplicación [Autoría propia]

4.2 Ficheros de la aplicación

En este apartado se detallan los ficheros más importantes de la presente aplicación.

4.2.1 *AndroidManifest*

Todos los proyectos de aplicaciones tienen que tener un archivo *AndroidManifest.xml* en el directorio raíz de la fuente del proyecto. El archivo de manifiesto [33] describe, como se citó en el apartado anterior, la información esencial de la aplicación. Los componentes más importantes del archivo son:

1. El nombre del paquete de la aplicación, utilizado por las herramientas de compilación de Android para determinar la ubicación de las entidades de código cuando se compila el proyecto.
2. Los componentes de la aplicación, que incluyen todas las actividades creadas.
3. Los permisos que necesita la aplicación para acceder a las partes protegidas del sistema o a otras aplicaciones, así como los permisos que otras aplicaciones deben tener si quieren acceder al contenido de nuestra aplicación.

El archivo *AndroidManifest.xml* (Figura 4-2) se genera automáticamente al crear la aplicación. La mayoría de los elementos esenciales de este archivo se van agregando a medida que se compila la aplicación.

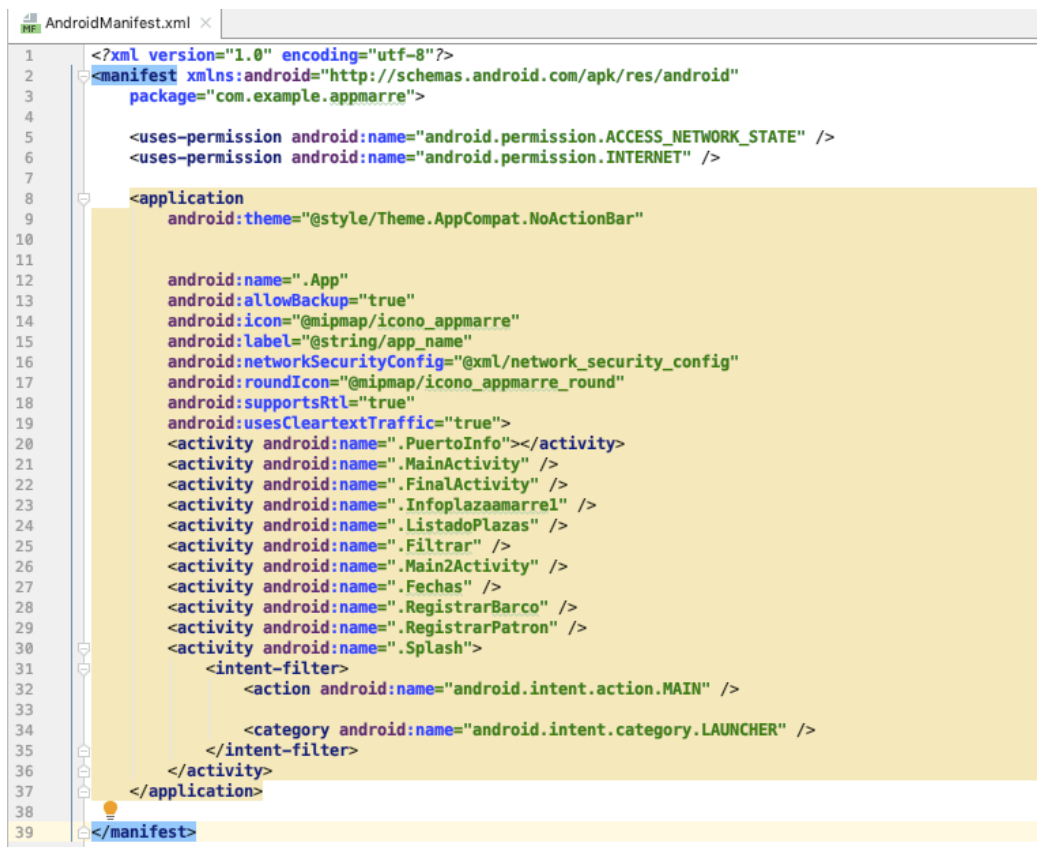


Figura 4-2 *AndroidManifest.xml* de la aplicación [Autoría propia]

Cabe resaltar que en este archivo se define también la primera actividad que se cargará al ejecutar la aplicación (conocida como *Launcher Activity*). En la presente aplicación, la actividad *Launcher* es la actividad *Splash*.

4.2.2 Build.gradle

El archivo *build.gradle* de módulo permite configurar ajustes de compilación. Algunos elementos y ajustes básicos (Figura 4-3) de este archivo son:

- *compileSdkVersion*: especifica el nivel de API en el que se compila la aplicación. Además de las funciones de la API seleccionada, se incluyen también las de APIs inferiores. En el caso de nuestra aplicación, el valor seleccionado es 29, correspondiente al SDK más moderno. El nivel seleccionado aquí coincide con el del *targetSdkVersion*, que especifica el nivel de API utilizado para hacer pruebas de la aplicación.
- *minSdkVersion*: define el mínimo nivel de API que permite la aplicación para funcionar cuando se ejecuta. Para esta aplicación se eligió como mínimo nivel de API el 21, pues es el que mejor relación tiene entre el número de mejoras presentadas (mejoras de rendimiento, respuesta de las aplicaciones o uso de estadísticas, entre otras) y el número de usuarios que podrían ejecutar nuestra aplicación (85%) [55].
- *versionCode* y *versionName*: la primera define el número de versión de la aplicación y la segunda el nombre de la versión que verán los usuarios. En nuestro caso se han dejado los que se establecen por defecto: 1 y "1.0".

```

android {
    compileSdkVersion 29
    buildToolsVersion "29.0.2"
    defaultConfig {
        applicationId "com.example.appmarre"
        minSdkVersion 21
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
}

```

Figura 4-3 Elementos más importantes del *build.gradle* [Autoría propia]

4.2.3 Ficheros de recursos

Los ficheros de recursos (Figura 4-4) son los archivos adicionales y el contenido estático que usa el código de la aplicación, como imágenes, definiciones de diseño o instrucciones de animación.

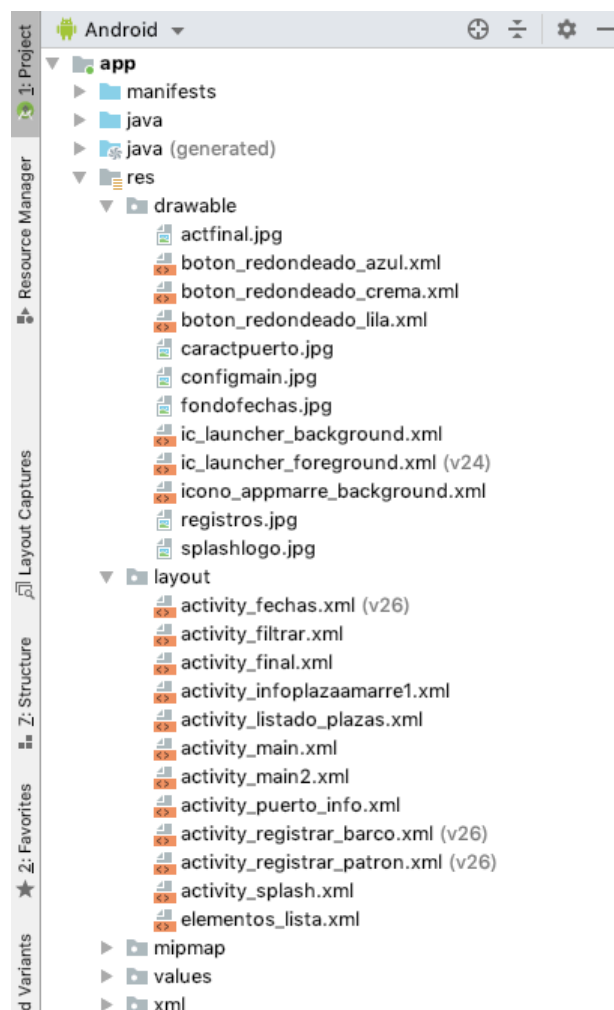


Figura 4-4 Ficheros de recursos de la aplicación [Autoría propia]

4.2.3.1 Drawable

Como se dijo anteriormente, este tipo de recurso agrupa a los archivos de imagen o archivos XML que se han incluido como elementos de diseño.

Los archivos de este tipo que se cargaron en la aplicación son la imagen .jpg que se utilizó para el fondo de la actividad *Splash* y las imágenes que se utilizaron como fondo del resto de actividades.

4.2.3.2 Layout

Son los archivos XML que definen el diseño de la interfaz de usuario de una actividad. Con dos posibles vistas (Figura 4-5) de la actividad a crear (*design* y *blueprint*), se pretende diseñar la parte visual de la aplicación.

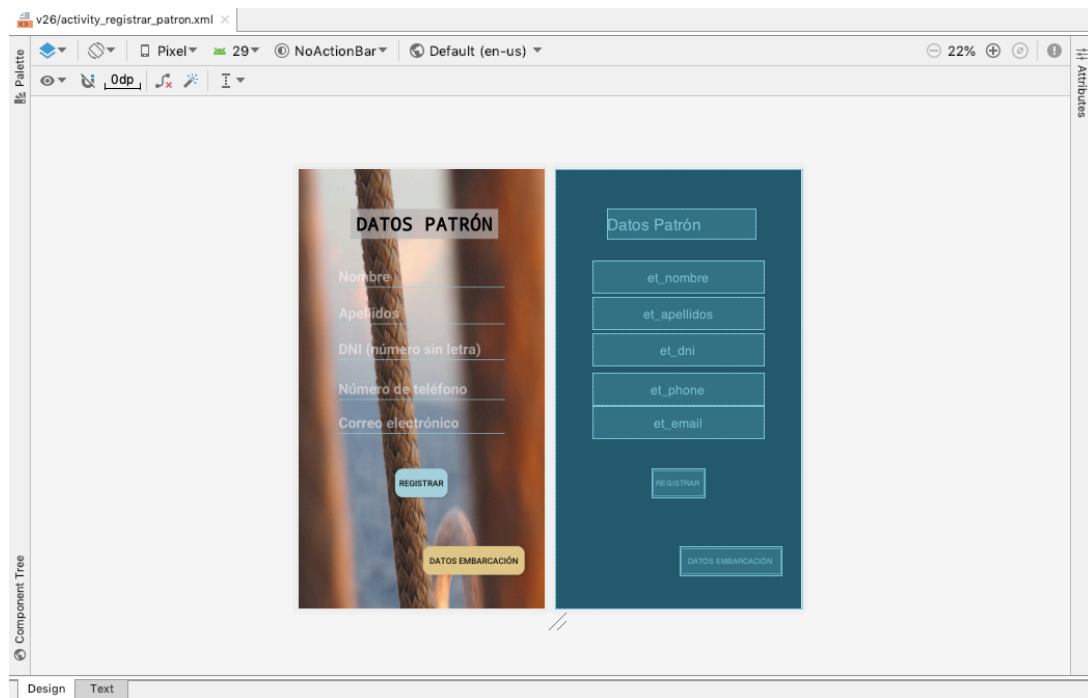


Figura 4-5 Vistas *design* y *blueprint* de *layout* [Autoría propia]

Asimismo, la parte visual se traduce en un código XML (Figura 4-6) que se genera automáticamente al crear el diseño y que se puede modificar o comprobar en la pestaña *text*.

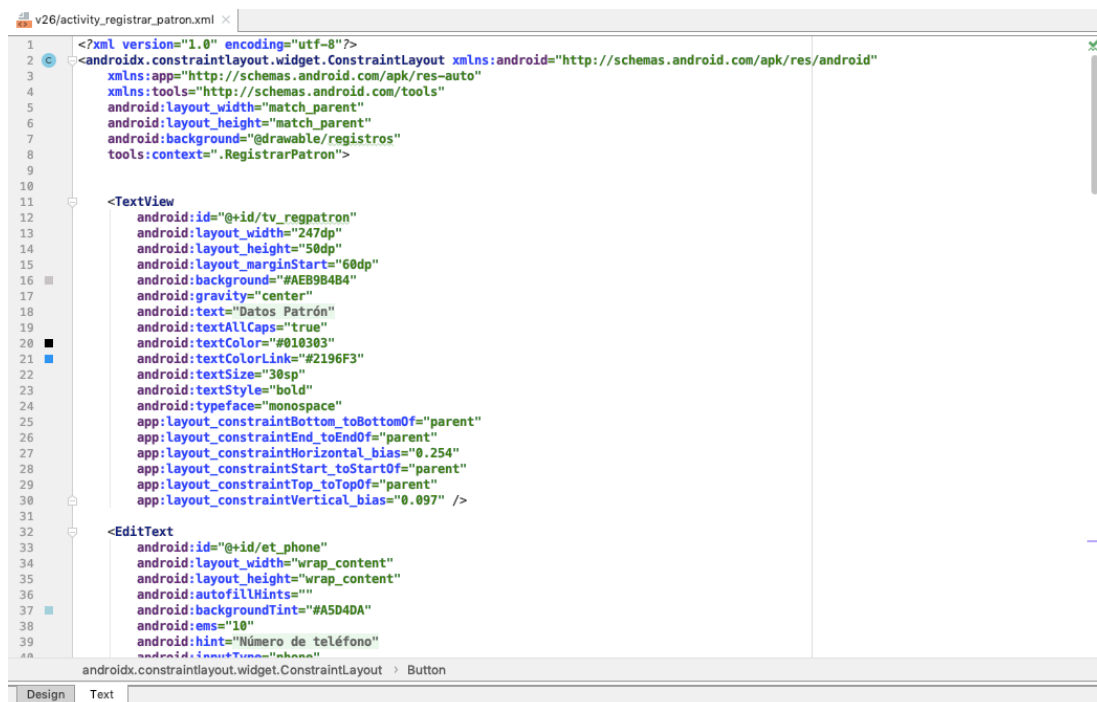


Figura 4-6 Código de *layout* [Autoría propia]

Existen multitud de elementos para darle forma a las actividades de una aplicación Android. Los utilizados en *Appmarre* son:

- *TextView*: es un control que imprime un texto cuando se ejecuta la aplicación o que se muestra permanentemente en una actividad, independientemente de las acciones que ocurran dentro de ella.
- *EditText*: es un control que espera la entrada de texto por parte del usuario. En nuestro caso, se utiliza para recoger los datos del patrón y de la embarcación. También se puede configurar para que imprima un texto cuando se ejecuta la aplicación.
- *Button*: botón que ejecuta una acción cuando se pulsa. En la presente aplicación se utiliza para acciones como pasar de una actividad a otra o para guardar datos.
- *Checkbox*: es un botón de doble estado (seleccionado o no seleccionado) que permite elegir una o varias opciones de un conjunto. En la presente aplicación se utiliza para filtrar la búsqueda, seleccionando qué servicios quiere el usuario en su plaza de amarre.
- *Listview*: es un elemento que visualiza una lista vertical de varios elementos. En nuestra aplicación se utiliza para listar las plazas disponibles que tendrá el usuario a su disposición tras introducir sus datos y seleccionar fechas y servicios.

4.2.3.3 Values

Son archivos XML que contienen valores simples, como cadenas de caracteres, valores enteros y colores. De este fichero de recursos, cabe destacar los de tipo *string* (Figura 4-7).

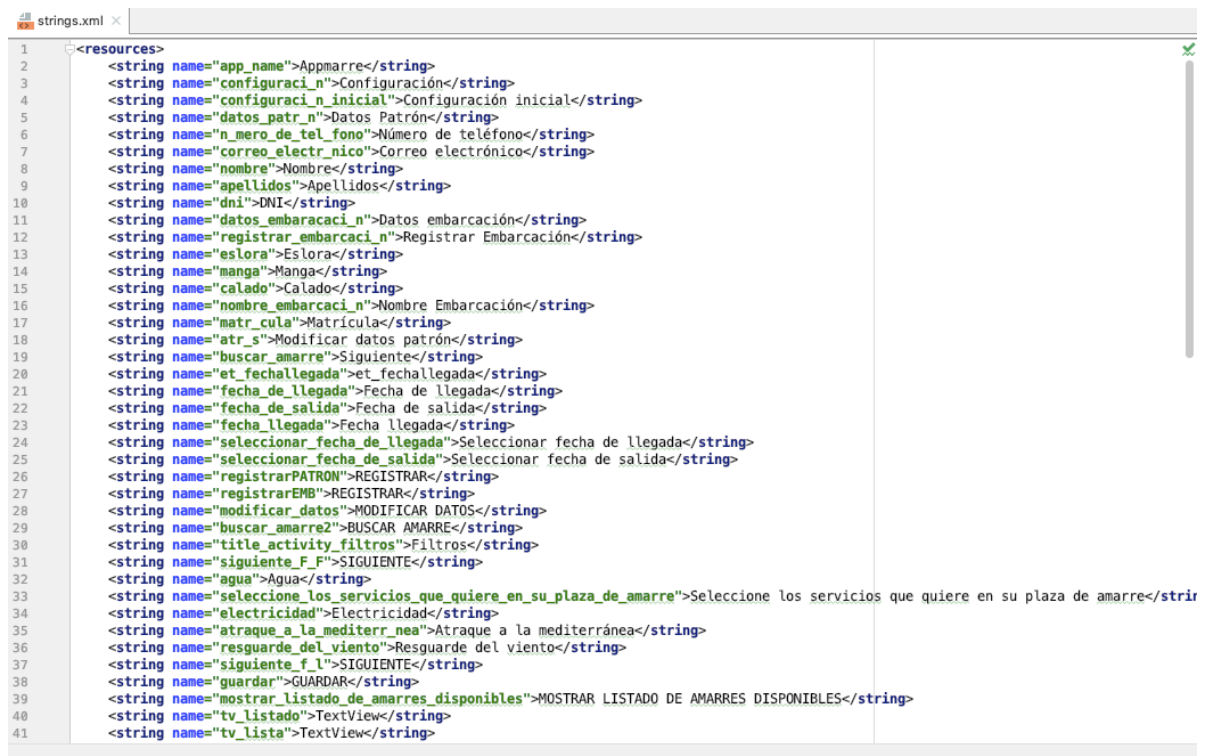


Figura 4-7 Strings de la aplicación [Autoría propia]

Cada vez que se crea un elemento en el *layout* de una actividad, se tiene que crear un *string* asociado al elemento creado. El fichero *strings.xml* almacena las cadenas, quedando definidas en un solo sitio para su posterior uso en otras partes de la aplicación. Así, se consigue una aplicación más eficiente, en la que no se repiten cadenas referidas a un mismo elemento.

4.2.4 Activities

Existirá una clase *Activity* [33] por cada una de las pantallas con las que interactúa el usuario, un componente clave de una aplicación de Android. La mayoría de las aplicaciones contienen varias pantallas, lo que significa que incluyen varias actividades.

4.2.4.1 Actividad *Splash*

En la actividad *Splash* se genera una pantalla que se le muestra al usuario durante unos pocos segundos cuando abre la aplicación. No contiene interacción y sirve para tener un recurso visual para el usuario mientras que la aplicación se ejecuta y llega hasta la actividad principal. Así, cuando finaliza ese breve periodo de tiempo, ya se ha hecho una precarga de datos para iniciar la aplicación.

Durante el transcurso de estos segundos, en *Appmarre* se visualiza en pantalla la imagen de la Figura 4-8.



Figura 4-8 Imagen utilizada en la actividad *Splash* [Autoría propia]

En el código de esta actividad se pueden diferenciar dos partes: la parte de código que provoca la aparición de la imagen mientras el usuario espera a que cargue la aplicación (véase Figura 4-9) y la parte de código que hace búsquedas para averiguar si ya hay un patrón y embarcación registrados en la base local de Parse Server del dispositivo en el que se inicializó la aplicación.

```

121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
    }

    setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
    this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowManager.LayoutParams.FLAG_FULLSCREEN);

    setContentView(R.layout.activity_splash);

    new Handler().postDelayed(() -> {
        ParseQuery<ParseObject> query = ParseQuery.getQuery("Configuracion");
        query.fromLocalDatastore();

        query.findInBackground((configuracion, e) -> {
            if (e==null){...}else{
                Log.w( tag: "bdlocalsplasherror", e.getMessage());
            }
        });
    }, DURATION_SPLASH);
    
```

Figura 4-9 Extracto del código de la actividad *Splash* [Autoría propia]

En la segunda parte de código citada (véase Figura 4-10), la de las consultas, se pretende dirigir al usuario a rellenar sus datos de patrón y embarcación en caso de que no lo haya hecho (es decir, dirigirlo a la actividad principal) o dirigirlo directamente a la selección de fechas y servicios de la

reserva en caso de que sus datos ya estén guardados en la base de datos local (es decir, dirigirlo a la actividad *Principal*).



```

93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118

ParseQuery<ParseObject> borrar=ParseQuery.getQuery("Configuracion");
borrar.whereEqualTo("patron","patron");
borrar.fromLocalDatastore();
borrar.findInBackground(new FindCallback<ParseObject>() {
    @Override
    public void done(List<ParseObject> borrarpatron, ParseException e) {
        if (e==null){
            for (ParseObject delete:borrarpatron){
                delete.deleteInBackground();
            }
        }
    }
});

ParseQuery<ParseObject> borrar2=ParseQuery.getQuery("Configuracionbarco");
borrar2.whereEqualTo("barco","barco");
borrar2.fromLocalDatastore();
borrar2.findInBackground((borrarbarco, e) -> {
    if (e==null){
        for (ParseObject delete2:borrarbarco){
            delete2.deleteInBackground();
        }
    }
});

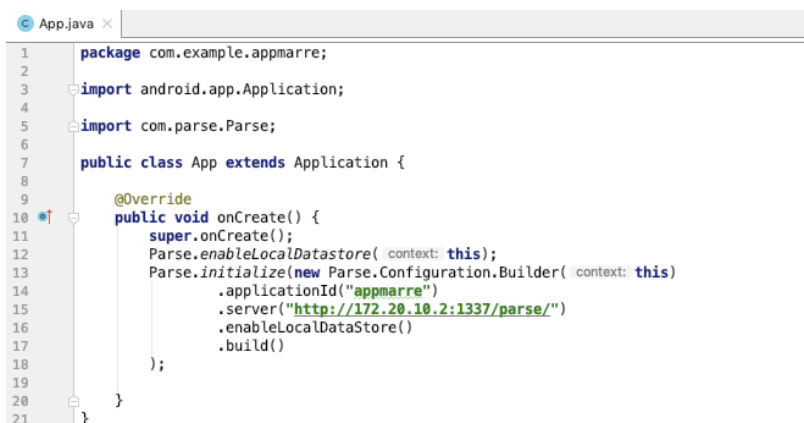
```

Figura 4-10 Búsquedas en la base de datos local dentro de la actividad *Splash* [Autoría propia]

Según el resultado de esta búsqueda, el usuario será dirigido bien a la actividad *Configuración* para registrarse, bien a la actividad *Principal* a continuar con la reserva de amarre con los datos que ya hay guardados en la base de datos local.

4.2.4.2 Actividad *App*

En la actividad *App*, la cual no tiene un *layout* asociado, se inicializan las conexiones con la base de datos remota de Parse, así como con la base de datos local, también de Parse. La herencia de esta actividad es *Application*, como se puede observar en la Figura 4-11. Este tipo de herencia hace que la clase se instancie antes que cualquier otra clase cuando se ejecuta la aplicación. El resto de actividades tienen la herencia *AppCompatActivity* (que es la clase base para las actividades que utilizan las funciones de *ActionBar*, y la que genera Android Studio por defecto cuando se crea una actividad nueva).



```

App.java x
1 package com.example.appmarre;
2
3 import android.app.Application;
4
5 import com.parse.Parse;
6
7 public class App extends Application {
8
9     @Override
10    public void onCreate() {
11        super.onCreate();
12        Parse.enableLocalDatastore( context: this);
13        Parse.initialize(new Parse.Configuration.Builder( context: this)
14            .applicationId("appmarre")
15            .server("http://172.20.10.2:1337/parse/")
16            .enableLocalDatastore()
17            .build()
18        );
19    }
20 }
21

```

Figura 4-11 Código de la actividad *App* [Autoría propia]

Esta clase se creó directamente, sin crear una actividad (cuando se crea una actividad se generan por defecto tanto el archivo de código o *class* como el archivo de interfaz de usuario o *layout*), por lo que se tuvo que incluir manualmente en el *AndroidManifest.xml*, como se puede observar en la Figura 4-2.

En el código se aprecia una primera acción en la línea 12 que corresponde al inicio de conexión entre la aplicación Android y la base de datos local de Parse en el dispositivo donde se está ejecutando la aplicación. El código desde la línea 13 a la 18 inicializa la conexión entre la aplicación Android y el servidor online de Parse. Se puede apreciar que hay que incluir el nombre de la aplicación que se

guardó en el *index.js* de los archivos de Parse (que es donde se configura el servidor de Parse), así como la dirección IP del ordenador que funciona como servidor. En nuestro caso, se utilizó un portátil particular.

4.2.4.3 Actividad Configuración

Esta actividad es la antesala de la configuración de datos básicos del usuario. Esto es, es un simple *intent* (elemento de Android que permite enviar componentes o dirigir al usuario a otra actividad, véase la Figura 4-12) que lleva a las actividades en las que el usuario registra al patrón y a la embarcación que se van a utilizar en la reserva.

```

9 public class MainActivity extends AppCompatActivity {
10
11     @Override
12     protected void onCreate(Bundle savedInstanceState) {
13         super.onCreate(savedInstanceState);
14         setContentView(R.layout.activity_main);
15     }
16
17     public void inicio_regpatron (View view){
18         Intent inicio_regpatron = new Intent( packageContext: this, RegistrarPatron.class);
19         startActivity(inicio_regpatron);
20     }
21 }
    
```

Figura 4-12 Extracto del código de la actividad Configuración [Autoría propia]

4.2.4.4 Actividad Registrar Patrón

En esta actividad, el usuario introduce los datos personales del patrón (nombre, apellidos, DNI, teléfono y correo electrónico) a través de distintos *EditText*. A esta actividad se puede llegar tras la Actividad Configuración o a través de la Actividad Principal.

El código comienza con una búsqueda en la base de datos local de Parse para comprobar si ya se habían introducido datos del patrón previamente en el dispositivo utilizado. En caso afirmativo, estos datos se imprimen en cada uno de los *EditText* (como en la Figura 4-13). En caso negativo, en los *EditText* aparecerá el dato que se tiene que introducir en cada uno de los controles. Ambos resultados de la búsqueda (campos a rellenar vacíos o con los datos de la base de datos local) se aprecian en la Figura 4-14.

```

32 final ParseQuery<ParseObject> query = ParseQuery.getQuery("Configuracion");
33 query.fromLocalDatastore();
34 query.findInBackground((config, e) -> {
35     if (e==null){
36         Log.i( tag: "Appmarre", msg: "pegar1");
37
38         String[]configuracionString = new String[config.size()];
39
40         Log.i( tag: "Appmarre", Integer.toString(config.size()));
41
42         if(configuracionString!=null){
43             query.whereEqualTo("patron","patron");
44             query.findInBackground((configlist, e) -> {
45                 if (e==null){
46                     for (ParseObject configuracioninicial : configlist){
47                         Log.i( tag: "Appmarre", msg: "pegando");
48
49                         EditText et_nombre=(EditText)findViewById(R.id.et_nombre);
50                         EditText et_apellidos=(EditText)findViewById(R.id.et_apellidos);
51                         EditText et_dni=(EditText)findViewById(R.id.et_dni);
52                         EditText et_telefono=(EditText)findViewById(R.id.et_phone);
53                         EditText et_correo=(EditText)findViewById(R.id.et_email);
54
55                         String nombre = configuracioninicial.getString( key: "nombre");
56                         et_nombre.setText(nombre);
57                         String apellidos =configuracioninicial.getString( key: "apellidos");
58                         et_apellidos.setText(apellidos);
59                         String dni =configuracioninicial.getString( key: "DNI");
60                         et_dni.setText(dni);
61                         String correo =configuracioninicial.getString( key: "correo");
62                         et_correo.setText(correo);
63                         String phone =configuracioninicial.getString( key: "telefono");
64                         et_telefono.setText(phone);
65                     }
66                 }
67             });
68         }
69     }
70 });
    
```

Figura 4-13 Código para comprobar e imprimir los datos del patrón de la base de datos local [Autoría propia]

Figure 4-14 shows two side-by-side screenshots of the 'Registrar Patrón' activity. The left screenshot shows the form with empty input fields for Name, Surnames, DNI, Phone Number, and Email. The right screenshot shows the same form with the fields filled out: Carmen Nazara, DNI 77418229, Phone 626499272, and Email armennazara@gmail.com. Both screens have a 'REGISTRAR' button and a 'DATOS EMBARCACIÓN' button at the bottom.

Figura 4-14 Layout de la actividad *Registrar Patrón* [Autoría propia]

Para poder seguir el proceso de registro (los datos se registran cuando el usuario pulsa el botón *Registrar*) el usuario debe rellenar todos los campos. Si así lo ha hecho, al pulsar el botón aparecerá un *Toast* (notificación que se superpone en la parte inferior de la pantalla) que le informa de que el registro ha sido exitoso. En caso contrario, en el *Toast* se le informará de que debe rellenar todos los datos pedidos por la aplicación. Ambos casos se pueden observar en la Figura 4-15.

Figure 4-15 shows two side-by-side screenshots of the 'Registrar Patrón' activity. The left screenshot shows a 'Debe rellenar todos los campos.' (Must fill all fields) error message. The right screenshot shows a 'Registro exitoso' (Registration successful) success message. Both screens have the 'REGISTRAR' and 'DATOS EMBARCACIÓN' buttons at the bottom.

Figura 4-15 *Toast* de la actividad *Registrar Patrón* [Autoría propia]

Para pasar a la siguiente actividad, el usuario deberá pulsar el botón de *Datos Embarcación*. Al pulsar este botón, además de dirigirlo a la actividad siguiente, se envían también los datos que se han introducido a la siguiente actividad, ya que estos no serán guardados en la base de datos del servidor de Parse hasta que el usuario llegue al final de la aplicación y pulse el botón *Reservar*. Sin embargo, estos datos sí se guardarán en la base de datos local de Parse Server cuando el usuario pulse el botón *Registrar*.

4.2.4.5 Actividad *Registrar Embarcación*

Esta actividad tiene una estructura idéntica a la de la actividad anterior, a diferencia de que aquí se introducen el nombre del barco, su eslora, manga y calado y la matrícula del mismo, como se puede apreciar en la Figura 4-16.

Figura 4-16 Layout de la actividad *Registrar Embarcación* [Autoría propia]

El botón *Registrar* tiene la misma configuración que el botón de la actividad anterior, mostrando diferentes *Toast* dependiendo de si el usuario cubre bien o no los datos de la embarcación a registrar, como se puede apreciar en la Figura 4-17.

Figura 4-17 Toast de la actividad *Registrar Embarcación* [Autoría propia]

Al igual que en la actividad *Registrar Patrón*, al pulsar el botón *Registrar*, los datos de la embarcación se guardarán en la base de datos local de Parse Server, pero no en la del servidor *online* de Parse. Esta acción ocurrirá cuando el usuario pulse el botón *Reservar* al final del proceso de reserva.

En este caso, el botón *Siguiente* dirigirá al usuario a la actividad *Principal*.

4.2.4.6 Actividad *Principal*

El código de esta actividad comienza con una búsqueda en la base de datos local de Parse para averiguar si hay datos guardados. Esto carece de sentido si venimos de registrarnos, pero es esencial si nos encontramos en la segunda (o posterior) vez que el usuario entra en la aplicación ya que es, tras esta búsqueda, cuando la aplicación recolecta dichos datos para que se vayan enviando mediante *intents* en las sucesivas actividades hasta llegar a la última actividad, en la que se enviarán al servidor de la aplicación para poder gestionar la reserva. Estas búsquedas se pueden apreciar en la Figura 4-18.

```

35      final ParseQuery<ParseObject> query = ParseQuery.getQuery("Configuracionbarco");
36      query.whereEqualTo("barco", "barco");
37      query.fromLocalDatastore();
38      query.findInBackground((config, e) -> {
41          if (e==null){
42              if (config.size()!=0){
43                  for(ParseObject configbarco:config){
44                      nombrebarco = configbarco.getString( key: "nombrebarco");
45                      eslorastring = configbarco.getString( key: "eslorabarco");
46                      mangastring = configbarco.getString( key: "mangabarco");
47                      caladostring = configbarco.getString( key: "caladobarco");
48                      matricula = configbarco.getString( key: "matricula");
49                      Log.d( tag: "busqueda eslora",eslorastring);
50                  }
51              }
52          }
53      });
54

```

Figura 4-18 Extracto del código con las búsquedas iniciales en la actividad *Principal* [Autoría propia]

Tras estas búsquedas, se le dirigirá al usuario a donde elija, dependiendo de si pulsa el botón *Modificar Datos* (por el cual irá a las actividades de registro del patrón y del barco para comprobar/modificar los datos ya guardados) o el de *Buscar Amarre* (por el cual se irá a elegir las fechas de llegada y salida de puerto para la reserva). El *layout* de esta actividad se puede observar en la Figura 4-19.



Figura 4-19 *Layout* de la actividad *Principal* [Autoría propia]

4.2.4.7 Actividad *Selección de fechas*

En esta actividad se seleccionan las fechas de comienzo y fin de la reserva a través de un *DatePicker* que se activa con botones (véase la imagen de la izquierda de la Figura 4-20), que presentan un calendario por cada una de las fechas (como se puede ver en la imagen de la derecha de la Figura 4-20, donde se aprecia el calendario desplegado, después de que el usuario pulsase el botón *Seleccionar Fecha Llegada*). Cuando se despliegan los calendarios, el usuario puede seleccionar cada una de las fechas haciendo *click* en los días que desee.

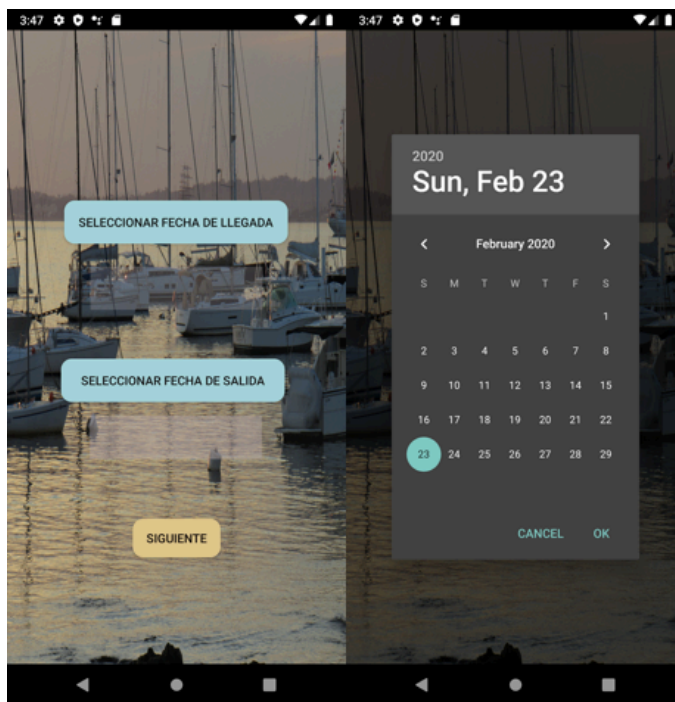


Figura 4-20 Layout de la actividad *Selección de fechas* [Autoría propia]

Para que el calendario aparezca posicionado en la fecha en la que se está utilizando la aplicación, se debe introducir el fragmento de código que se ven en las líneas 73-76 de la Figura 4-21.

```

72
73
74
75
76
77
78
79
80
81
82
86
87
88
89
90
91
92
93
95
96
97

if (view == butfechallegada) {
    final Calendar c = Calendar.getInstance();
    dia = c.get(Calendar.DATE);
    mes = c.get(Calendar.MONTH);
    ano = c.get(Calendar.YEAR);
    c.set(Calendar.HOUR_OF_DAY,0);
    c.set(Calendar.MINUTE,0);
    c.set(Calendar.SECOND,0);
    c.set(Calendar.MILLISECOND,0);

    final DatePickerDialog datePickerDialog = new DatePickerDialog( context: this, (view, year, month, dayOfMonth) -> {
        String date =(dayOfMonth + "/" + (month + 1) + "/" + year);
        tvfechallegada.setText(date);
        Calendar calendar =Calendar.getInstance();
        calendar.set(year,month,dayOfMonth);
        fllegadamilis =calendar.getTimeInMillis();
        fllegadamilistring=Long.toString(fllegadamilis);
        Log.d( tag: "fechallegada en milis",fllegadamilistring);
    }, ano, mes, dia);
    datePickerDialog.show();
}
    
```

Figura 4-21 Código para implementar uno de los *DatePicker* de la actividad *Selección de fechas* [Autoría propia]

Con el fin de poder comprobar la ocupación de las plazas de amarre, todas las fechas utilizadas se pasan a milisegundos, como se puede ver en la línea 90 de la Figura 4-21. Asimismo, en la base de datos se guardan también en este formato.

4.2.4.8 Actividad *Filtrado de servicios*

En esta actividad, el usuario selecciona, a través de varios *checkbox*, los diferentes servicios o características que quiere que su plaza de amarre tenga. Los servicios o características que se ponen en esta aplicación son el de electricidad, agua, resguardo del viento y el de atraque a la mediterránea (véase Figura 4-22).



Figura 4-22 Layout de la actividad *Filtrado de servicios* [Autoría propia]

Cada vez que el usuario selecciona alguno de los *checkbox*, el valor del filtro se pone a '1', como se puede observar en el código que se ve en la Figura 4-23.

```

65
66
67
68
69
    if(checkagua.isChecked()== true) {
        filtro_agua=1;
    }else{
        filtro_agua=0;
    }

```

Figura 4-23 Funcionamiento de uno de los filtros de la actividad *Filtrado de servicios* [Autoría propia]

Una vez el usuario ha elegido los servicios o características que quiere, pulsa el botón *Guardar*, guardándose todos los filtros seleccionados. Cuando el usuario pulsa el botón *Siguiente*, se le dirige a la actividad en la que se muestra el listado de plazas disponibles según los datos introducidos en esta actividad y en las anteriores.

4.2.4.9 Actividad *Listado de plazas disponibles*

Con los datos recolectados en las actividades anteriores se realiza una búsqueda en la base de datos de amarres y reservas de Parse Server. Aquellas plazas que cumplan los requerimientos solicitados por el cliente (y que no estén reservadas en el periodo temporal que abarquen las fechas de llegada y salida seleccionadas por el usuario) aparecerán en pantalla, presentados a través de un *Listview* (como se puede apreciar en la imagen de la izquierda de la Figura 4-24). En caso de que no haya ninguna plaza que cumpla todos los requerimientos del cliente, la lista aparecerá vacía y se le avisará al usuario a través de un *Toast* (como se puede apreciar en la imagen del centro de la Figura 4-24). Si el motivo por el cual no hay ninguna plaza disponible es por las fechas seleccionadas, se le informará al cliente mediante un *Toast* como el que aparece en la imagen de la derecha de la Figura 4-24.



Figura 4-24 Layout de la actividad *Listado de plazas disponibles* [Autoría propia]

Para ver las características de cada una de las plazas, así como para reservar, tendrá que presionar sobre la plaza que le interese. El paso a la actividad en la que se muestran las características de la plaza se hace a través de un *OnClickListener* (véase línea 142 del código de la Figura 4-25).

```

128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146

if (plazas.size() != 0) {
    queryAmarre.whereNotContainedIn( key: "numero", plazas);
}
queryAmarre.findInBackground((amarres, e) -> {
    if (e == null) {
        int i = 0;
        final String[] resultsAsString = new String[amarres.size()];

        Log.d( tag: "Query", Integer.toString(amarres.size()));
        for (ParseObject amarre : amarres) {
            final int numero = (Integer) amarre.get("numero");
            final String post = (String) amarre.get("post");
            listView.setOnItemClickListener((parent, view, position, id) -> {
                Object numeroplaza = parent.getItemAtPosition(position);
                numeroplazastring = String.valueOf(numeroplaza);
            });
        }
    }
});
    
```

Figura 4-25 Extracto del código de la actividad *Listado de plazas disponibles* [Autoría propia]

4.2.4.10 Actividad *Información de la plaza seleccionada*

En esta actividad se mostrarán las características de la plaza de amarre que seleccione el usuario de las que se ofrezcan en el *ListView* de la actividad anterior. Las características de las plazas de amarre están almacenadas en la base de datos de Parse Server. El acceso a dicha base de datos se consigue realizando una búsqueda con el número de plaza seleccionado, que se exporta desde la actividad anterior hasta la presente actividad. Este traspaso de información se realiza a través de un *getExtra* con un *intent* desde la actividad emisora (mostrado en la Figura 4-26) y se recoge a través de un *getIntent* en la actividad receptora (mostrado en la Figura 4-27).

```

150
151
152
153
154
155
156
157
158

Intent intent = new Intent( packageContext: ListadoPlazas.this, Infoplazaamarre1.class);
intent.putExtra( name: "numeroplazaseleccionada", numeroplazastring);
intent.putExtra( name: "fechallegada", fechallegada);
intent.putExtra( name: "fechasalida", fechasalida);
intent.putExtra( name: "filtro_agua", filtro_agua);
intent.putExtra( name: "filtro_electricidad", filtro_electricidad);
intent.putExtra( name: "filtro_viento", filtro_viento);
intent.putExtra( name: "filtro_medi", filtro_medi);
intent.putExtra( name: "eslora", eslora);
    
```

Figura 4-26 Extracto de un *intent* de la actividad *Listado de plazas disponibles* [Autoría propia]

```

53 Bundle datos=getIntent().getExtras();
54 fechallegada=datos.getString( key: "fechallegada");
55 fechasalida=datos.getString( key: "fechasalida");
56 filtro_agua=datos.getString( key: "filtro_agua");
57 filtro_electricidad=datos.getString( key: "filtro_electricidad");
58 filtro_viento=datos.getString( key: "filtro_viento");
59 filtro_medi=datos.getString( key: "filtro_medi");
60 eslora=datos.getString( key: "eslora");

```

Figura 4-27 Extracto del código de un *getIntent* de la actividad *Información de la plaza seleccionada* [Autoría propia]

Como se muestra en la Figura 4-28, debajo de la lista de características de la plaza (que están presentadas en pantalla a través de diferentes *TextView*), hay dos botones: *Ver servicios del puerto* y *Reservar*. Si el usuario quiere conocer los servicios que el puerto ofrece a todas las plazas, deberá pulsar el primer botón citado. Si quiere reservar directamente la plaza, deberá pulsar el botón *Reservar*.

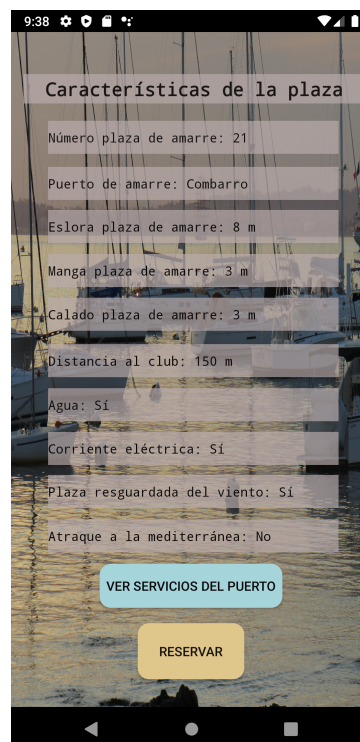


Figura 4-28 *Layout* de la actividad *Información de la plaza* [Autoría propia]

Se debe reseñar que los servicios que se muestran en esta actividad, están guardados en la base de datos de Parse Server como se muestra en la Figura 4-29.

```

{
  "method": "POST",
  "path": "/parse/classes/Amarre",
  "body": {
    "post": "Plaza 20 (Sanxenxo)",
    "numero": 20,
    "puerto": "Sanxenxo",
    "eslora": 12,
    "manga": 3,
    "calado": 4,
    "distancia_al_club": 100,
    "agua": 1,
    "electricidad": 1,
    "resguardo_viento": 1,
    "atraque_mediterranea": 0,
    "ocupacion": 0
  },

```

Figura 4-29 Base de datos de los amarres en Parse Server (extracto del *script* enviar.py) [Autoría propia]

Cuando el usuario pulsa el botón *Reservar* que se encuentra bajo la lista, será dirigido a la última actividad de la aplicación. Esta acción, además, supondrá el envío de todos los datos necesarios para la reserva a la base de datos de Parse Server.

4.2.4.11 Actividad *Información del puerto al que pertenece la plaza*

En caso de que el usuario quiera obtener más detalles de los servicios del puerto en el que se encuentra la plaza en la que está interesado, pulsará el botón *Ver servicios del puerto* de la actividad descrita anteriormente. En ese caso, se ejecutará la búsqueda que se muestra en la Figura 4-30 para extraer los datos del puerto que están almacenados en la base de datos de Parse Server. Los datos obtenidos se presentan en pantalla con diferentes *TextView*, como se puede mostrar en la Figura 4-31.

```

61
62
63
66
67
68
69
70
71
72
75
76
77
78
79
80
81
82
    ParseQuery<ParseObject> query = ParseQuery.getQuery("Amarre");
    query.whereEqualTo("numero", Integer.parseInt(numeroobtenido));
    query.findInBackground((objects, e) -> {
        if (e==null){
            for (ParseObject plazaelegida: objects){
                puerto=plazaelegida.getString( key: "puerto");
                Log.d( tag: "puerto", puerto);
                ParseQuery<ParseObject> query2 = ParseQuery.getQuery("Puerto");
                query2.whereEqualTo("nombre", puerto);
                query2.findInBackground((plazaspuertos, e) -> {
                    if (e==null){
                        Log.d( tag: "if", msg: "dentro"+plazaspuertos.size());

                        for (ParseObject puertos : plazaspuertos){
                            Log.d( tag: "for", msg: "dentro");

                            tv_nombrepuerto.setText("Puerto de "+puerto);
                        }
                    }
                });
            }
        }
    });

```

Figura 4-30 Extracto del código de la actividad *Información del puerto al que pertenece la plaza* [Autoría propia]

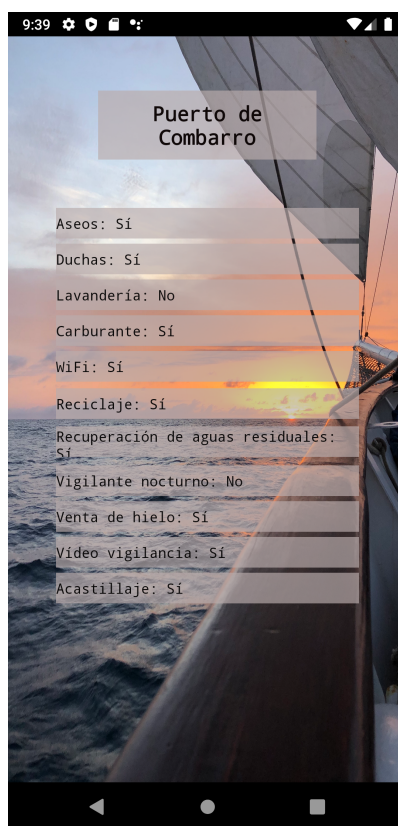


Figura 4-31 Layout de la actividad *Información del puerto al que pertenece la plaza* [Autoría propia]

La información del puerto donde se encuentra la plaza de amarre seleccionada está guardada en la base de datos de puertos de Parse Server como se observa en Figura 4-32.

```

{
  "method": "POST",
  "path": "/parse/classes/Puerto",
  "body": {
    "nombre": "Combarro",
    "latitud": 42.428124,
    "longitud": -8.704579,
    "aseos": 1,
    "duchas": 1,
    "carburante": 1,
    "WiFi": 1,
    "reciclaje": 1,
    "recuperacion_aguas_residuales": 1,
    "video_vigilancia": 1,
    "vigilante_nocturno": 0,
    "lavanderia": 0,
    "venta_hielo": 1,
    "acastillaje": 1
  }
}

```

Figura 4-32 Base de datos de los puertos en Parse Server (extracto del *script* enviar.py) [Autoría propia]

4.2.4.12 Actividad *Final*

Cuando el usuario pulsa el botón *Reservar* en la actividad *Información de la plaza seleccionada*, al usuario se le presentará esta actividad, en la que se muestra un *TextView* con el mensaje que se puede ver en la Figura 4-33, particularizado con el correo que el usuario hubiese introducido al registrarse.



Figura 4-33 Layout de la actividad *Final* [Autoría propia]

Como se citó anteriormente, cuando el usuario pulsa el botón *Reservar* (el que conduce al usuario hasta esta actividad), es entonces también cuando todos los datos de patrón, embarcación y reserva son enviados a las bases de datos de Parse Server, creando un nuevo objeto en las clases correspondientes de la base de datos.

En esta actividad se presenta también el botón *Volver al menú principal*, que llevará al usuario a la actividad *Principal*, ya que en ese momento los datos que introdujo al principio del proceso ya se han guardado en local. Así pues, podría volver a realizar otra reserva desde esa actividad.

5 PRUEBA Y VALIDACIÓN

En este capítulo se detallan las pruebas realizadas para comprobar que la aplicación funciona correctamente, así como algunos de los fallos obtenidos y las soluciones dadas.

Se debe reseñar que la validación de la aplicación ha sido constante durante todo el proceso de desarrollo. Esto es debido a que, para añadir nuevas actividades o implementar nuevos métodos, se debía comprobar que los que ya estaban desarrollados funcionaban correctamente.

5.1 Propósito de la aplicación

El fin último de todo el proceso de gestión de la reserva de un amarre en la presente aplicación es que los datos de reserva que se introducen lleguen correctamente al servidor de la aplicación.

Para llegar a ese hito final, se deben conseguir otros hitos intermedios:

- Registro del patrón y de la embarcación. La primera acción que tiene que realizar el usuario es rellenar sus datos para realizar la reserva. Estos datos no se guardan en la base de datos de Parse Server hasta que se pulsa el botón *Reservar*, pero sí se guardan en la base de datos local de Parse en el momento en que el usuario pulsa *Registrar* en cada una de las actividades de registro (*Registrar Patrón* y *Registrar Embarcación*). Así pues, la manera de comprobar que se han guardado con éxito estos datos (además de asegurarse de que se imprime el *Toast*) es que el usuario se dirija a la actividad *Principal* (que se encuentra a continuación de la actividad *Registrar Embarcación*) y pulse *Modificar Datos*. Si se han guardado correctamente, se encontrarán impresos en cada uno de los *EditText* en los que anteriormente introdujo los datos.
- Selección de las fechas. Las fechas de llegada y salida al puerto, como se citó anteriormente, se escogen a través de un *DatePicker*. Para comprobar que cada una de las fechas elegidas se extrae correctamente para el proceso de búsqueda, dichas fechas se imprimen en dos *EditText* en la misma actividad donde se encuentra el *DatePicker*.
- Filtro de servicios de la plaza de amarre. Tras seleccionar los servicios que el usuario quiere en su plaza a través de los *checkbox*, deberá pulsar el botón *Guardar*, momento en el que le aparecerá un *Toast* si no hubo ningún problema en el proceso de guardado.

Las siguientes actividades (*Listado de plazas disponibles*, *Información de la plaza seleccionada* e *Información del puerto de la plaza seleccionada*) son resultados de la búsqueda, que solo se pueden comprobar conociendo previamente las plazas que hay disponibles y las características de las mismas.

Esta aplicación está diseñada para que el usuario la utilice desde su teléfono móvil y que los datos de la base de datos de Parse Server se comprueben desde un ordenador (el cual funciona como el servidor de la aplicación).

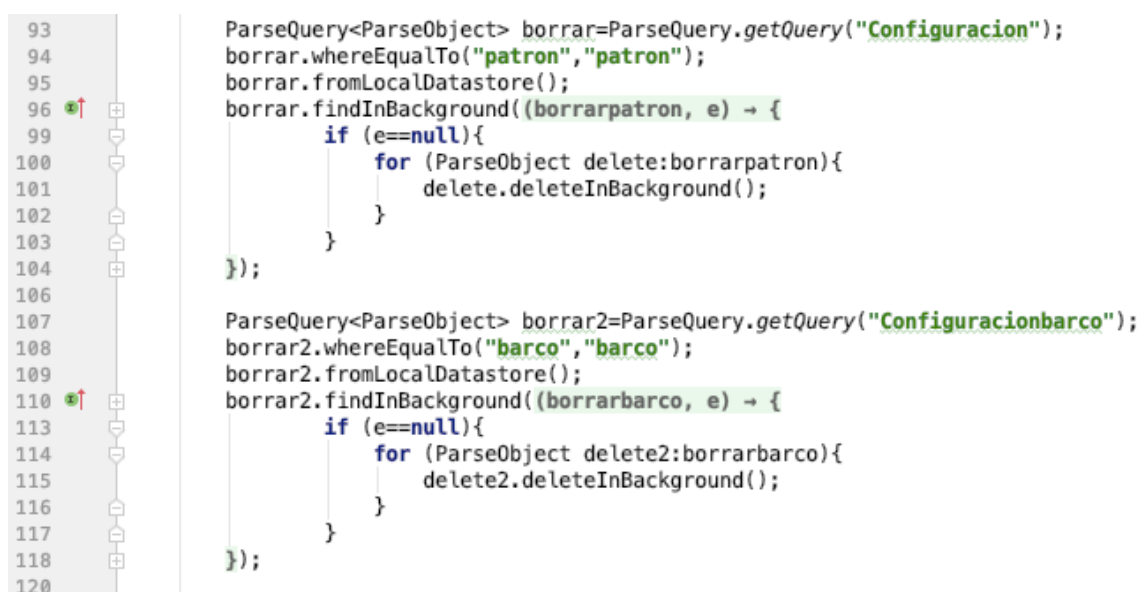
5.2 Métodos de comprobación

Para comprobar que cada nueva funcionalidad implementada funcionaba correctamente y que las reservas solicitadas se guardaban con éxito, se utilizaron dos métodos. En primer lugar, se fueron creando reservas ficticias de diferentes características que tocasen todos los ámbitos que pudiesen dar problemas, comprobando qué impedía la reserva si algo estaba mal y comprobando también que la base de datos se iba actualizando con cada reserva realizada correctamente. En segundo lugar, se hizo uso de los comandos *Logcat*, que permiten volcar mensajes internos en el sistema. Todas las comprobaciones se llevaron a cabo tanto en el emulador de Android Studio como en el teléfono móvil (descrito en el apartado 2.3.6) cedido por el CUD.

5.2.1 Creación de reservas ficticias

Para comprobar el buen funcionamiento de la aplicación se crearon reservas ficticias que, a la vez o por separado, permitiesen comprobar las siguientes cuestiones:

- Comprobar que, cuando se ejecuta la aplicación, aparece la imagen elegida para la actividad *Splash*. Comprobar también que, si la base de datos local de Parse Server tiene información guardada del patrón y su embarcación, dirige al usuario a la actividad *Principal*. Una vez se comprobó esto, se estableció al principio de esta actividad el código que se muestra en la Figura 5-1, el cual borra la base de datos local de Parse Server. Al borrarse la base de datos local de Parse Server cada vez que ejecutamos la aplicación, la actividad *Splash* nos dirigirá siempre a la actividad *Configuración* y, en consecuencia, a rellenar los datos de patrón y embarcación. Esto último se mantuvo para la comprobación del resto de cuestiones, para poder realizar reservas con datos de diferentes personas y barcos.



```

93  ParseQuery<ParseObject> borrar=ParseQuery.getQuery("Configuracion");
94  borrar.whereEqualTo("patron","patron");
95  borrar.fromLocalDatastore();
96  borrar.findInBackground((borrarpatron, e) -> {
97      if (e==null){
98          for (ParseObject delete:borrarpatron){
99              delete.deleteInBackground();
100          }
101      }
102  });
103
104  ParseQuery<ParseObject> borrar2=ParseQuery.getQuery("Configuracionbarco");
105  borrar2.whereEqualTo("barco","barco");
106  borrar2.fromLocalDatastore();
107  borrar2.findInBackground((borrarbarco, e) -> {
108      if (e==null){
109          for (ParseObject delete2:borrarbarco){
110              delete2.deleteInBackground();
111          }
112      }
113  });
114
115  });
116
117  });
118
119  });
120

```

Figura 5-1 Extracto del código de la actividad *Splash* que permite borrar la base de datos local de Parse Server [Autoría propia]

- Comprobar, para todas las actividades, que los elementos instalados (botones, *EditText*, *TextView*...) se visualizan tal y como se diseñaron en sus respectivos *layouts* (en la Figura 5-2 se puede observar la comprobación para la actividad Información del puerto al que pertenece la plaza). De la misma manera, comprobar que las imágenes puestas como *background* de las diferentes actividades se visualizan íntegras y sin daños.

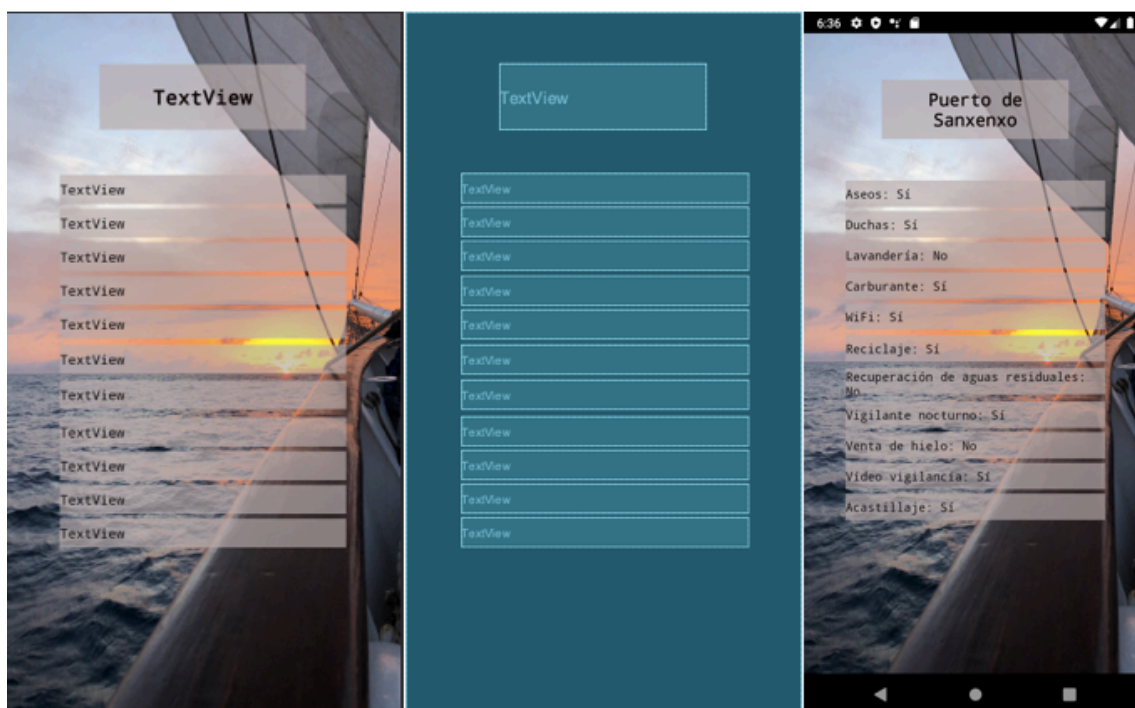


Figura 5-2 Comparación del diseño del *layout* (vistas *design* y *blueprint*) con la interfaz visualizada en el emulador para la actividad Información del puerto al que pertenece la plaza [Autoría propia]

- Comprobar, para todas las actividades, que los *intent* incluidos en el código se ejecutan correctamente (tanto los que transportan al usuario de una actividad a otra como los que traspasan datos de una actividad a otra).
- Comprobar, para todas las actividades, que, si se pulsa el botón ‘volver atrás’ del emulador o del teléfono móvil, se dirige al usuario a la actividad que se encuentra inmediatamente antes de la actividad que se muestra en pantalla.
- Comprobar que, tras introducir correctamente los datos que se piden en las actividades *Registrar Patrón* y *Registrar Embarcación* y una vez pulsado el botón *Registrar*, se muestra en pantalla el *Toast* con el mensaje *Registro exitoso*. Comprobar también que, si, por el contrario, se deja uno de los datos solicitados en blanco, cuando se pulsa el botón *Registrar*, se muestra en pantalla el *Toast* con el mensaje *Debe rellenar todos los campos*.
- Comprobar que, si se introducen dimensiones de la embarcación que son mayores que las dimensiones de las plazas de amarre de la base de datos de Parse Server, dichas plazas de amarre no aparecen en el listado de plazas de la actividad *Listado de plazas disponibles*.
- Comprobar que si se vuelve hacia atrás desde la actividad *Principal* (hasta las actividades de *Registrar Patrón* y *Registrar Embarcación*), los datos previamente introducidos en el registro se encuentran impresos en sus respectivos *EditText*. Asimismo, si en la actividad *Principal*, se pulsa el botón *Modificar datos*, también se encuentran los datos introducidos en el registro impresos en sus respectivos *EditText*.
- Comprobar que, si se modifican los datos de *Registrar Patrón* y *Registrar Embarcación* y se guardan, al volver a acceder a estas actividades, se encuentran impresos los datos que se introdujeron más recientemente.
- Comprobar, en la actividad de *Selección de fechas*, que las fechas seleccionadas en los *DatePicker* se imprimen correctamente en los *TextView* correspondientes. Un ejemplo de esta comprobación se muestra en la Figura 5-3.

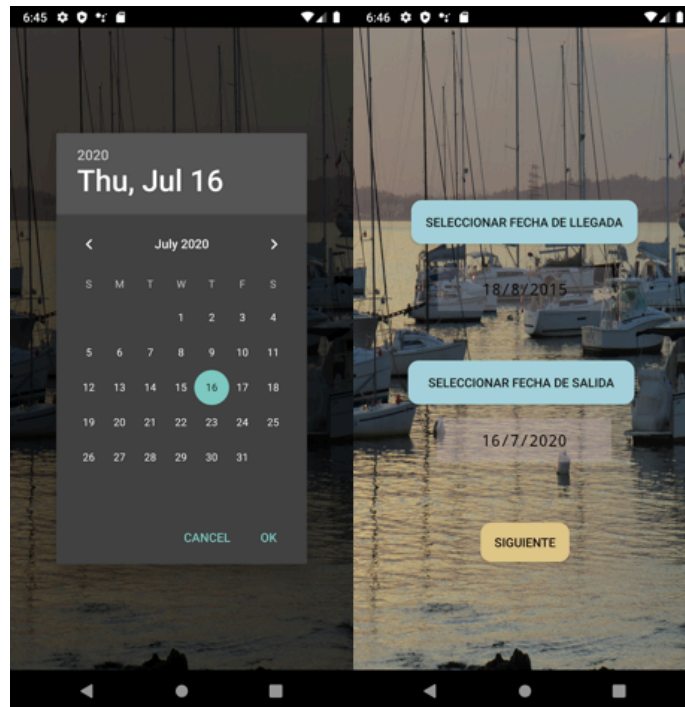


Figura 5-3 Comprobación de correcta impresión en el *EditText* de la fecha de salida seleccionada en el *DatePicker* [Autoría propia]

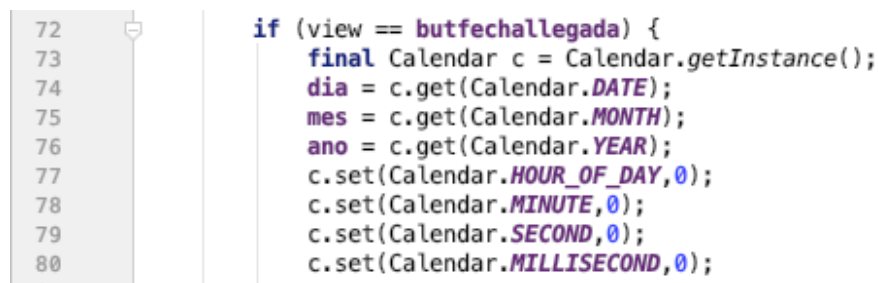
- Comprobar que, si se introducen fechas que coinciden, en sí mismas o el periodo que conforman, con las de otras reservas realizadas previamente, las plazas de amarre afectadas no aparecen en el listado de plazas disponibles.
- Comprobar que los servicios que se desean para la plaza de amarre (seleccionados en la actividad *Filtrado de servicios*), coinciden con los servicios que tienen las plazas que salen en el listado de plazas disponibles. De la misma manera, comprobar que, aunque el usuario no seleccione que quiere agua o electricidad en su plaza, se le ofrezcan igualmente las plazas que sí tengan acceso a esos dos servicios.
- Comprobar que, cuando se pulsa el botón *Guardar* de la actividad *Filtrado de servicios*, se muestra en pantalla un *Toast* con el mensaje ‘Guardado’.
- Comprobar que las plazas que aparecen en el *ListView* de la actividad *Listado de plazas disponibles*, cumplen con todos los datos, fechas y filtros introducidos previamente por el usuario.
- Comprobar, para las plazas que salgan en el listado de plazas disponibles, que las características de la plaza que se muestran en la actividad *Información de la plaza seleccionada*, coinciden con los datos de esta plaza que se encuentran guardados en la base de datos de Parse Server.
- Comprobar que las características del puerto que se muestran en la actividad *Información del puerto* cuando el usuario pulsa el botón *Ver servicios del puerto* en la actividad anterior, coinciden con los datos guardados en la base de datos de Parse Server. Comprobar también que el puerto que se muestra es el puerto donde se encuentra la plaza seleccionada.
- Comprobar, en la actividad *Final*, que el correo electrónico que se muestra en el *TextView*, es el mismo correo electrónico que introdujo el usuario en el registro del patrón.

Una vez se comprobaron todas estas cuestiones, se realizaron otras reservas para comprobar que las búsquedas hechas en Parse Server se ejecutaban correctamente cuando se introducen nuevas reservas.

5.2.2 Comandos Logcat

Este método resultó muy útil para comprobar el funcionamiento de métodos que, visualmente en la pantalla del emulador o del teléfono móvil, no se aprecian. Los usos que fueron determinantes para el desarrollo de la aplicación fueron:

- Que las cadenas de caracteres o números enteros extraídos de la base de datos de Parse Server o de la base de datos local, se obtenían íntegros y sin daños. Gracias a esto, se pudo apreciar que algunos errores provenían de intentar extraer un número de la base de datos como si se tratase de una cadena.
- Para comprobar que el paso de las fechas al formato en milisegundos se hacía correctamente. Gracias a esto, se pudo apreciar que las fechas extraídas del *DatePicker* llevaban la hora del ordenador del momento en que se hacía la reserva, en vez de una hora estándar. Al ocurrir esto, fechas iguales no resultaban iguales al compararlas porque las horas con las que se habían guardado eran diferentes. Para solucionarlo, se estableció una hora fija (00:00:00) para todas las fechas que se obtuviesen del *DatePicker*, como se puede apreciar en las líneas 77-80 de la Figura 5-4.



```

72
73
74
75
76
77
78
79
80
81

if (view == butfechallegada) {
    final Calendar c = Calendar.getInstance();
    dia = c.get(Calendar.DATE);
    mes = c.get(Calendar.MONTH);
    ano = c.get(Calendar.YEAR);
    c.set(Calendar.HOUR_OF_DAY,0);
    c.set(Calendar.MINUTE,0);
    c.set(Calendar.SECOND,0);
    c.set(Calendar.MILLISECOND,0);
}
    
```

Figura 5-4 Extracto del código de la actividad *Fechas* [Autoría propia]

- Para comprobar que los datos transportados mediante *intent* llegaban a la actividad de destino de manera íntegra y sin daños.
- Para comprobar hasta donde llegaba a ejecutarse la aplicación cuando se interrumpía por algún error.
- Para que el sistema nos notificase mediante un mensaje si ocurría algún error durante la ejecución de la aplicación.
- Para conocer cuántas filas componían las listas que se creaban tras las búsquedas en la base de datos de Parse Server.

Realizadas todas las pruebas anteriormente descritas, podemos afirmar que la aplicación ha sido validada con éxito.

6 CONCLUSIONES Y LÍNEAS FUTURAS

En este capítulo, se hace balance de lo conseguido teniendo en cuenta lo que se pretendía. También se plantean líneas futuras para continuar este proyecto presentando tanto ideas que mejorarían la aplicación, como ideas que la complementarían.

6.1 Conclusiones

Si revisamos los objetivos establecidos en el capítulo 1, podemos afirmar que, efectivamente, se han conseguido desarrollar la aplicación pretendida.

También, recapitulando la comparativa hecha en el apartado 2.2, se puede afirmar que se han conseguido implementar las características propuestas en dicha comparativa. Estos objetivos eran: incluir información básica de los puertos en los que se realizan las reservas, incluir información básica y ampliada de la plaza seleccionada, pedir fechas de llegada y salida, permitir elegir la plaza de amarre que prefiera el usuario y pedir datos del patrón y de la embarcación para realizar la reserva.

El resto de cuestiones que se presentaban en el citado apartado, aunque no se proponían para esta aplicación, se proponen como líneas futuras en el siguiente apartado.

6.2 Líneas futuras

Las líneas futuras que se plantean a continuación se dividen en dos categorías: líneas futuras que mejoran las funcionalidades actuales de la aplicación y líneas futuras complementarias.

1. Líneas futuras que mejoran las funcionalidades actuales de la aplicación:
 - Establecer un límite de caracteres para los datos introducidos en los registros (sobre todo para aquellos datos que sigan un patrón, como los números de teléfono, el DNI del patrón o la matrícula de la embarcación).
 - Enviar el correo electrónico de confirmación de la reserva de manera directa al mismo tiempo que le aparece al usuario el mensaje en pantalla informándole de que recibirá dicho correo.
 - Intentar que ciertas fechas de los calendarios del *DatePicker* se bloqueen: las fechas donde el puerto esté cerrado, inoperativo o completo, así como las fechas previas al día en el que se realiza la reserva. De la misma manera, bloquear en el *DatePicker* de la fecha de salida las fechas anteriores a la fecha de llegada seleccionada.
 - Incluir la posibilidad de pagar con tarjeta de manera *online* una vez que el usuario confirma la reserva. Al implementar esta opción se debería incluir el precio de cada plaza de amarre en la información de cada una de ellas.

- Crear una interfaz de empleo por los puertos deportivos que se conecte al servidor Parse y muestre la información que incumba al puerto en cuestión.
 - Validar la aplicación para un conjunto de datos de entrada (puertos, plazas de amarre, etc.) mayor y más variado. Si se incluyesen puertos localizados en emplazamientos muy lejanos, se deberían ofrecer resultados teniendo en cuenta la ubicación GPS actual del usuario que está realizando la reserva.
2. Líneas futuras complementarias:
- Adaptación de la misma aplicación móvil a dispositivos Android que no sean teléfonos móviles.
 - Permitir la obtención de un *feedback* de los usuarios, es decir, que los usuarios de la aplicación puedan dejar comentarios o valoraciones acerca del puerto donde realizaron la reserva.
 - Internacionalizar la aplicación para que esté disponible en otros idiomas.
 - Incluir la previsión meteorológica de cada puerto deportivo que trabaje con esta aplicación.
 - Diseño y desarrollo de una versión de la aplicación móvil para dispositivos iOS.

7 BIBLIOGRAFÍA

- [1] Turismo Rías Baixas, «La Ría de Pontevedra en las Rías Baixas» [En línea]. Available: <https://turismoriasbaixas.com/recursopan1?content=280377165>. [Último acceso: 28 diciembre 2019].
- [2] M. Abilleira, «La náutica de recreo recupera espacio en la Ría de Pontevedra» 11 julio 2016. [En línea]. Available: <https://www.diariodepontevedra.es/articulo/noticias/la-nautica-de-recreo-recupera-espacio-en-la-ria-donde-solo-un-15-de-los-amarres-esta/20160711000000305107.html>. [Último acceso: 28 diciembre 2019].
- [3] SASEMAR, «Transparencia Salvamento Marítimo» 2018. [En línea]. Available: <http://transparencia.salvamentomaritimo.es>. [Último acceso: 3 marzo 2020].
- [4] Ministerio de Fomento, «Real Decreto 1185/2006 sobre el Reglamento que regula las radiocomunicaciones marítimas de los buques civiles españoles.» 2006. [En línea]. Available: <https://www.boe.es/buscar/act.php?id=BOE-A-2006-18968>. [Último acceso: 3 marzo 2020].
- [5] Sail and Trip, «Frecuencias, canales y alcance del VHF marino» 9 octubre 2016. [En línea]. Available: <https://sailandtrip.com/vhf-marino-frecuencias-canales/>. [Último acceso: 11 enero 2020].
- [6] «Náutica y Yates» [En línea]. Available: nauticayyates.com. [Último acceso: 10 enero 2020].
- [7] Play Store, «Enlace Play Store para descargar la aplicación ASNAUGA» 2015. [En línea]. Available: <https://play.google.com/store/apps/details?id=com.asnauga.guianautica>. [Último acceso: 19 enero 2020].
- [8] App Store, «Enlace App Store para descargar la aplicación ASNAUGA» 2015. [En línea]. Available: <https://apps.apple.com/es/app/asnauga-gu%C3%ADa-de-n%C3%A1utica-de-galicia/id821619061>. [Último acceso: 19 enero 2020].
- [9] «Web Asociación de Clubes Náuticos de Galicia» [En línea]. Available: <http://asnauga.com>. [Último acceso: 10 enero 2020].
- [10] «Illas Atlánticas» [En línea]. Available: <https://www.iatlanticas.es>. [Último acceso: 24 diciembre 2019].
- [11] Play Store, «Enlace Play Store para descargar la aplicación Navily» 2019. [En línea].

- Available: <https://play.google.com/store/apps/details?id=com.ckcnet.android.navily&hl=es>. [Último acceso: 19 enero 2020].
- [12] App Store, «Enlace App Store para descargar la aplicación *Navily*» 2019. [En línea]. Available: <https://apps.apple.com/es/app/navily/id881103205>. [Último acceso: 19 enero 2020].
- [13] Play Store, «Enlace Play Store para descargar la aplicación *MarinaAlicante*» 2015. [En línea]. Available: <https://play.google.com/store/apps/details?id=com.entornodigital.marinaalicante>. [Último acceso: 19 enero 2020].
- [14] App Store, «Enlace App Store para descargar la aplicación *MarinaAlicante*» 2015. [En línea]. Available: <https://apps.apple.com/es/app/marina-alicante/id923958795>. [Último acceso: 19 enero 2020].
- [15] Play Store, «Enlace Play Store para descargar la aplicación *Greenwich*» 2015. [En línea]. Available: <https://play.google.com/store/apps/details?id=com.entornodigital.marinagreenwich>. [Último acceso: 19 enero 2020].
- [16] App Store, «Enlace App Store para descargar la aplicación *Greenwich*» 2015. [En línea]. Available: <https://apps.apple.com/es/app/marina-greenwich/id904107186>. [Último acceso: 19 enero 2020].
- [17] Be First, comunicación y eventos, «Ventajas Aplicaciones Móviles con respecto a las Web» [En línea]. Available: <http://www.befirst.es/aplicaciones-moviles/>. [Último acceso: 19 enero 2020].
- [18] StatCounter from GlobalStats, «*Mobile Operating System Market Share Spain*» [En línea]. Available: <https://gs.statcounter.com/os-market-share/mobile/spain/#monthly-201901-202001>. [Último acceso: 23 enero 2020].
- [19] StatCounter from GlobalStats, «*Mobile Operating System Market Share Worldwide*» [En línea]. Available: <https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-201901-202001>. [Último acceso: 23 enero 2020].
- [20] YeePLY, «Crear apps móviles: diferencias entre Android y iOS» [En línea]. Available: <https://www.yeeply.com/blog/crear-apps-moviles-diferencias-android-e-ios/>. [Último acceso: 23 enero 2020].
- [21] Java, «La tecnología Java» [En línea]. Available: https://www.java.com/es/download/faq/whatis_java.xml. [Último acceso: 3 marzo 2020].
- [22] Kotlin, «Sobre Kotlin» [En línea]. Available: <https://kotlin.es/sobre-kotlin/>. [Último acceso: 2 marzo 2020].
- [23] Lenguajes de programación, «Lenguaje de programación Objective-C» [En línea]. Available: <https://lenguajesdeprogramacion.net/objective-c/>. [Último acceso: 2 marzo 2020].
- [24] Apple, «Swift. Un lenguaje potente y abierto a todos para crear apps increíbles.» [En línea]. Available: <https://www.apple.com/es/swift/>. [Último acceso: 2 marzo 2020].
- [25] J. M. López, «Cuando Google conoció a Android» Hipertextual, 23 diciembre 2018. [En línea]. Available: <https://hipertextual.com/2018/12/historia-origen-android>. [Último acceso: 19 enero 2020].

- [26] SQLite, «*What is SQLite?*» [En línea]. Available: <https://www.sqlite.org/index.html>. [Último acceso: 2 marzo 2020].
- [27] M. J. L. Lapuente, «HTML» [En línea]. Available: <http://www.hipertexto.info/documentos/html.htm>. [Último acceso: 2 marzo 2020].
- [28] Adobe, «*Flash Player*» [En línea]. Available: <https://www.adobe.com/es/products/flashplayer.html>. [Último acceso: 9 marzo 2020].
- [29] Androidos, «Características y arquitectura de Android OS» [En línea]. Available: <https://androidos.readthedocs.io/en/latest/data/caracteristicas/>. [Último acceso: 20 enero 2020].
- [30] Android Inc., «Versiones de Android OS» [En línea]. Available: https://www.android.com/intl/es_es/history/#/donut. [Último acceso: 21 enero 2020].
- [31] The Verge, «*Android 10 officially released for Google Pixel phones*» 3 septiembre 2019. [En línea]. Available: <https://www.theverge.com/2019/9/3/20842507/google-android-10-q-pixel-release-download-availability>. [Último acceso: 21 enero 2020].
- [32] S. P. Martí, «Android 11: fecha de lanzamiento, novedades y rumores.» 4 marzo 2020. [En línea]. Available: <https://www.pcworld.es/articulos/software/android-11-3783447/>. [Último acceso: 6 marzo 2020].
- [33] Android Developers, «Introducción a Android Studio» [En línea]. Available: <https://developer.android.com/studio/intro?hl=es-419>. [Último acceso: 20 enero 2020].
- [34] El androide libre, «Android Studio: esto es lo que puedes hacer con este programa» 17 enero 2020. [En línea]. Available: <https://elandroidelibre.lespanol.com/2020/01/android-studio-que-es-y-para-que-se-utiliza.html>. [Último acceso: 20 enero 2020].
- [35] Android Developers, «Sugerencias y recetas de Gradle» [En línea]. Available: <https://developer.android.com/studio/build/gradle-tips?hl=es-419>. [Último acceso: 2 marzo 2020].
- [36] Platzi, «Java VS. Kotlin» [En línea]. Available: <https://platzi.com/blog/java-vs-kotlin/>. [Último acceso: 21 enero 2020].
- [37] Oracle, «*Oracle and Sun Microsystems*» [En línea]. Available: <https://www.oracle.com/es/sun/>. [Último acceso: 2 marzo 2020].
- [38] Jesús Sánchez Allende, Gabriel Huecas Fernández-Toribio, Baltasar Fernández Manjón y Pilar Moreno Díaz, *JAVA 2*, Madrid: MCGRAW-HILL, 2006.
- [39] ViaFirma, «¿Qué es un *Applet*?» [En línea]. Available: <https://www.viafirma.com/faq/es/que-es-un-applet/>. [Último acceso: 19 enero 2020].
- [40] Free Java Guide, «*History of Java programming language*» [En línea]. Available: <https://www.freejavaguide.com/history.html>. [Último acceso: 19 enero 2020].
- [41] Evaluamos, «Google se llevó al padre de Java: James Gosling» 28 marzo 2011. [En línea]. Available: <http://www.evaluamos.com/2011/internal.php?load=detail&id=12273>. [Último acceso: 26 enero 2020].
- [42] Java, «Versiones de Java» [En línea]. Available: https://www.java.com/es/download/faq/release_dates.xml. [Último acceso: 6 marzo 2020].

- [43] Arsys, «Parse Server, uno de los entornos más útiles hoy en día para el desarrollo web» 28 septiembre 2016. [En línea]. Available: <https://www.arsys.es/blog/programacion/parse-server-servidor-cloud/>. [Último acceso: 26 enero 2020].
- [44] Node, «Express: infraestructura web rápida, minimalista y flexible para Node.js» [En línea]. Available: <https://expressjs.com/es/>. [Último acceso: 29 enero 2020].
- [45] Node, «Node.js, a *JavaScript* runtime» [En línea]. Available: <https://nodejs.org/en/>. [Último acceso: 29 enero 2020].
- [46] MongoDB, «MongoDB, la base de datos líder para aplicaciones modernas» [En línea]. Available: <https://www.mongodb.com/es/>. [Último acceso: 29 enero 2020].
- [47] SITECH GROUP, «*How to host and manage Apps with an open source Parse Server*» 5 julio 2017. [En línea]. Available: <https://www.sitechgroup.com/how-to-host-and-manage-apps-with-an-open-source-parse-server/>. [Último acceso: 26 enero 2020].
- [48] B. Macías, «Motorola G7 Play: *notch* convencional y procesador potente para el más básico de la familia Moto» 7 febrero 2019. [En línea]. Available: <https://www.xatakandroid.com/moviles-android/moto-g7-play-caracteristicas-precio-ficha-tecnica>. [Último acceso: 14 enero 2020].
- [49] Island Yachts Broker, «*Island Yachts Broker*» [En línea]. Available: <https://www.islandyachtsbroker.com>. [Último acceso: 2 marzo 2020].
- [50] Ministerio de Relaciones con las Cortes y de la Secretaría del Gobierno, «Real Decreto 1027/1989, de 28 de julio, sobre abanderamiento, matriculación de buques y registro marítimo» 15 agosto 1989. [En línea]. Available: <https://www.boe.es/eli/es/rd/1989/07/28/1027>. [Último acceso: 23 enero 2020].
- [51] MongoDB, «Instalación de MongoDB» [En línea]. Available: <https://www.mongodb.com/download-center/community>. [Último acceso: 3 marzo 2020].
- [52] NodeJS, «Instalación NodeJS» [En línea]. Available: <https://nodejs.org/en/download/>. [Último acceso: 3 marzo 2020].
- [53] GitHub, «Instalación de Parse Server» [En línea]. Available: <https://github.com/parse-community/parse-server-example>. [Último acceso: 3 marzo 2020].
- [54] Python, «Python Software» [En línea]. Available: <https://www.python.org>. [Último acceso: 2 marzo 2020].
- [55] Android Studio, «*Android Platform/API Version Distribution*» [En línea]. Available: <https://i.stack.imgur.com/jT1Gl.png>. [Último acceso: 3 marzo 2020].

ANEXO I: *SCRIPTS* DE PYTHON PARA LA COMPROBACIÓN DE DATOS DE LA BASE DE DATOS DE PARSE SERVER

Se crearon diferentes *scripts* en Python para poder comprobar y modificar el contenido de la base de datos de Parse Server. Los tres más utilizados fueron *enviar.py*, *consultar.py* y *borrarID.py*, los cuales se describen a continuación.

1. *enviar.py*. Este *script* se utiliza para introducir las características de las plazas de amarre y de los puertos donde se encuentran. Se detalla a continuación el contenido introducido para realizar la validación de la aplicación.

```
import json,httplib

connection = httplib.HTTPConnection('localhost', 1337)
connection.connect()
connection.request('POST', '/parse/batch', json.dumps({
    "requests": [
        {
            "method": "POST",
            "path": "/parse/classes/Amarre",
            "body": {
                "post": "Plaza 21 (Combarro)",
                "numero": 21,
                "puerto" : "Combarro",
                "eslora" : 8,
                "manga" : 3,
                "calado" : 3,
                "distancia_al_club" : 150,
                "agua" : 1,
                "electricidad" : 1,
                "resguardo_viento" : 1,
                "atraque_mediterranea" : 0,
                "ocupacion" : 0
            }
        },
        {
            "method": "POST",
            "path": "/parse/classes/Amarre",
            "body": {
                "post": "Plaza 20 (Sanxenxo)",
                "numero": 20,
                "puerto" : "Sanxenxo",
                "eslora" : 12,
                "manga" : 3,
                "calado" : 4,
                "distancia_al_club" : 100,
                "agua" : 1,
                "electricidad" : 1,
                "resguardo_viento" : 1,
                "atraque_mediterranea" : 0,
                "ocupacion" : 0
            }
        },
        {
            "method": "POST",
            "path": "/parse/classes/Puerto",
            "body": {
                "nombre": "Sanxenxo",
                "latitud": 42.398167,
                "longitud": -8.802874,
                "aseos": 1,
                "duchas": 1,
                "carburante": 1,
                "WiFi": 1,
```

```
        "reciclaje": 1,
        "recuperacion_aguas_residuales": 0,
        "video_vigilancia": 1,
        "vigilante_nocturno": 1,
        "lavanderia": 0,
        "venta_hielo": 0,
        "acastillaje": 1
    }
},
{
    "method": "POST",
    "path": "/parse/classes/Puerto",
    "body": {
        "nombre": "Combarro",
        "latitud": 42.428124,
        "longitud": -8.704579,
        "aseos": 1,
        " duchas": 1,
        "carburante": 1,
        "WiFi": 1,
        "reciclaje": 1,
        "recuperacion_aguas_residuales": 1,
        "video_vigilancia": 1,
        "vigilante_nocturno": 0,
        "lavanderia": 0,
        "venta_hielo": 1,
        "acastillaje": 1
    }
}
]),
{"X-Parse-Application-Id": "appmarre", "Content-Type": "application/json"}
)

results = json.loads(connection.getresponse().read())
print results
```

2. *consultar.py*. Este *script* se utiliza para visualizar los datos que hay en una determinada clase de la base de datos de Parse Server.

```
import sys
import json,httplib

if len(sys.argv) != 2:
    print "Uso: python consultar.py NombreClase"
    sys.exit(1)

consulta = '/parse/classes/' + sys.argv[1]
print "Buscando: " + consulta

connection = httplib.HTTPConnection('localhost', 1337)
connection.connect()
connection.request('GET', consulta, "", {"X-Parse-Application-Id": "appmarre"})
results = json.loads(connection.getresponse().read())
print results
```

Al ejecutar este comando en el terminal, se vería lo que se muestra en la Figura A1-1.

```
MacBook-Air-de-Carmen:Parse carmennazara$ python consultar.py Amarre
Buscando: /parse/classes/Amarre
{'u'results': [{u'ataque_mediterranea': 0, u'ocupacion': 0, u'objectId': u'1RoIQGDExv', u'electricidad': 1, u'agua': 1, u'manga': 3, u'calado': 3, u'eslora': 8, u'numero': 21, u'updatedAt': u'2020-03-01T11:18:48.014Z', u'puerto': u'Combarro', u'resguardo_viento': 1, u'distancia_al_club': 150, u'post': u'Plaza 21 (Combarro)', u'createdAt': u'2020-03-01T11:18:48.014Z'}, {u'ataque_mediterranea': 0, u'ocupacion': 0, u'objectId': u'jkiOKBBYT3', u'electricidad': 1, u'agua': 1, u'manga': 3, u'calado': 4, u'eslora': 12, u'numero': 20, u'updatedAt': u'2020-03-01T11:18:48.014Z', u'puerto': u'Sanxenxo', u'resguardo_viento': 1, u'distancia_al_club': 100, u'post': u'Plaza 20 (Sanxenxo)', u'createdAt': u'2020-03-01T11:18:48.014Z'}]}
MacBook-Air-de-Carmen:Parse carmennazara$
```

Figura A1-1 Script *consultar.py* ejecutado en el terminal

3. *borrarID.py*. Al ejecutarse este comando en el terminal, se borra el objeto elegido de cierta clase. Para poder borrar dicho objeto, se debe conocer el ObjectID que el sistema genera automáticamente para cada objeto que se añade a la base de datos.

```
import sys
import json,httplib

if len(sys.argv) != 3:
    print "Uso: python consultar.py NombreClase objectId"
    sys.exit(1)

consulta = '/parse/classes/' + sys.argv[1] + "/" + sys.argv[2]
print "Borrando: " + consulta

connection = httplib.HTTPConnection('localhost', 1337)
connection.connect()
connection.request('DELETE', consulta, "", {"X-Parse-Application-Id": "appmarre"})
output = json.loads(connection.getresponse().read())
print output
```

En el terminal se visualizaría como se muestra en la Figura A1-2.

```
MacBook-Air-de-Carmen:Parse carmennazara$ python borrarID.py Reserva CC9KXA3QB1
Borrando: /parse/classes/Reserva/CC9KXA3QB1
{}
MacBook-Air-de-Carmen:Parse carmennazara$
```

Figura A1-2 Script *borrarID.py* ejecutado en el terminal

ANEXO II: CÓDIGO JAVA DE LA APLICACIÓN

En este anexo se adjunta el código de cada una de las actividades que componen la aplicación.

Actividad *Splash*

```
package com.example.appmarre;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.content.pm.ActivityInfo;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.view.View;
import android.view.WindowManager;
import android.widget.AdapterView;

import com.parse.FindCallback;
import com.parse.GetCallback;
import com.parse.Parse;
import com.parse.ParseException;
import com.parse.ParseObject;
import com.parse.ParseQuery;
import com.parse.SaveCallback;

import java.util.List;

public class Splash extends AppCompatActivity {

    private final int DURACION_SPLASH = 2000;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash);

        ParseQuery<ParseObject> borrar=ParseQuery.getQuery("Configuracion");
        borrar.whereEqualTo("patron","patron");
        borrar.fromLocalDatastore();
        borrar.findInBackground(new FindCallback<ParseObject>() {
            @Override
            public void done(List<ParseObject> borrarpatron, ParseException e) {
                if (e==null){
                    for (ParseObject delete:borrarpatron){
                        delete.deleteInBackground();
                    }
                }
            }
        });

        ParseQuery<ParseObject> borrar2=ParseQuery.getQuery("Configuracionbarco");
        borrar2.whereEqualTo("barco","barco");
        borrar2.fromLocalDatastore();
        borrar2.findInBackground(new FindCallback<ParseObject>() {
            @Override
            public void done(List<ParseObject> borrarbarco, ParseException e) {
                if (e==null){
                    for (ParseObject delete2:borrarbarco){
                        delete2.deleteInBackground();
                    }
                }
            }
        });
    }
}
```

```
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
        this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);

        setContentView(R.layout.activity_splash);

        new Handler().postDelayed(new Runnable(){
            public void run(){
                ParseQuery<ParseObject> query = ParseQuery.getQuery("Configuracion");
                query.fromLocalDatastore();

                query.findInBackground(new FindCallback<ParseObject>() {
                    @Override
                    public void done(List<ParseObject> configuracion, ParseException e) {
                        if (e==null){

                            Log.i("Splash config size",
Integer.toString(configuracion.size()));

                                if(configuracion.size()!=0){
                                    ParseQuery<ParseObject> query1
=ParseQuery.getQuery("Configuracionbarco");
                                    query1.fromLocalDatastore();
                                    query1.findInBackground(new FindCallback<ParseObject>() {
                                        @Override
                                        public void done(List<ParseObject> configuracionbarco,
ParseException e) {
                                            if (e==null){

                                                Log.i("Splash configbarco size",
Integer.toString(configuracionbarco.size()));
                                                if(configuracionbarco.size()==1){
                                                    Intent splash_mainactivity2 = new
Intent(Splash.this, Main2Activity.class);
                                                    startActivity(splash_mainactivity2);
                                                }else {
                                                    Intent splash_mainactivity1 = new
Intent(Splash.this, MainActivity.class);
                                                    startActivity(splash_mainactivity1);
                                                }
                                            }
                                        }
                                    });
                                }else{
                                    Intent splash_mainactivity1 = new Intent(Splash.this,
MainActivity.class);
                                    startActivity(splash_mainactivity1);
                                }

                                }else{
                                    Log.w("bdlocalsplasherror", e.getMessage());
                                }
                            }
                        });
                    }
                }, DURATION_SPLASH);
            }
        }
```


Actividad *App*

```
package com.example.appmarre;

import android.app.Application;
import com.parse.Parse;

public class App extends Application {

    @Override
    public void onCreate() {
        super.onCreate();
        Parse.enableLocalDatastore(this);
        Parse.initialize(new Parse.Configuration.Builder(this)
            .applicationId("appmarre")
            .server("http://172.20.10.2:1337/parse/")
            .enableLocalDataStore()
            .build()
        );
    }
}
```

Actividad *Configuración*

```
package com.example.appmarre;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void inicio_regpatron (View view){
        Intent inicio_regpatron = new Intent(this, RegistrarPatron.class);
        startActivity(inicio_regpatron);
    }
}
```

Actividad *Registrar Patrón*

```
package com.example.appmarre;

import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

import com.parse.FindCallback;
import com.parse.ParseException;
import com.parse.ParseObject;
import com.parse.ParseQuery;
import com.parse.SaveCallback;

import java.util.List;
```

```
import static com.example.appmarre.R.layout.activity_registrar_patron;

public class RegistrarPatron extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(activity_registrar_patron);

        Log.i("Appmarre", "onCreate");

        final ParseQuery<ParseObject> query = ParseQuery.getQuery("Configuracion");
        query.fromLocalDatastore();
        query.findInBackground(new FindCallback<ParseObject>() {
            @Override
            public void done(List<ParseObject> config, ParseException e) {
                if (e==null){
                    Log.i("Appmarre", "pegar1");

                    String[]configuracionString = new String[config.size()];

                    Log.i("Appmarre", Integer.toString(config.size()));

                    if(configuracionString!=null){
                        query.whereEqualTo("patron","patron");
                        query.findInBackground(new FindCallback<ParseObject>() {
                            @Override
                            public void done(List<ParseObject> configlist, ParseException
e) {
                                if (e==null){
                                    for (ParseObject configuracioninicial : configlist){
                                        Log.i("Appmarre", "pegando");

                                        EditText
et_nombre=(EditText)findViewById(R.id.et_nombre);
                                        EditText
et_apellidos=(EditText)findViewById(R.id.et_apellidos);
                                        EditText
et_dni=(EditText)findViewById(R.id.et_dni);
                                        EditText
et_telefono=(EditText)findViewById(R.id.et_phone);
                                        EditText
et_correo=(EditText)findViewById(R.id.et_email);

                                        String nombre =
configuracioninicial.getString("nombre");
                                        et_nombre.setText(nombre);
                                        String apellidos
=configuracioninicial.getString("apellidos");
                                        et_apellidos.setText(apellidos);
                                        String dni =configuracioninicial.getString("DNI");
                                        et_dni.setText(dni);
                                        String correo
=configuracioninicial.getString("correo");
                                        et_correo.setText(correo);
                                        String phone
=configuracioninicial.getString("telefono");
                                        et_telefono.setText(phone);

                                    }
                                }
                            }
                        });
                    }
                }
            }
        });
    }
}
```

```

    });
}

public void Registrar (View view) {

    String nombre = ((EditText) findViewById(R.id.et_nombre)).getText().toString();
    String apellidos = ((EditText) findViewById(R.id.et_apellidos)).getText().toString();
    String dni = ((EditText) findViewById(R.id.et_dni)).getText().toString();
    String correo = ((EditText) findViewById(R.id.et_email)).getText().toString();
    String telefono = ((EditText) findViewById(R.id.et_phone)).getText().toString();

    Log.i("Appmarre", correo);

    if (nombre.length() != 0 && (apellidos.length() != 0) && (dni.length() != 0) &&
        (correo.length() != 0) && (telefono.length() != 0)) {

        ParseObject config = new ParseObject("Configuracion");
        config.put("patron", "patron");
        config.put("nombre", nombre);
        config.put("apellidos", apellidos);
        config.put("correo", correo);
        config.put("DNI", dni);
        config.put("telefono", telefono);
        config.pinInBackground(new SaveCallback() {
            @Override
            public void done(ParseException e) {
                if (e == null)
                    Log.i("localdbbarco", "guardado");
                else {
                    Log.i("localdbbarco", "error");
                    e.printStackTrace();
                }
            }
        });

        Toast.makeText(this, "Registro exitoso", Toast.LENGTH_LONG).show();

    } else {
        Toast.makeText(this, "Debe rellenar todos los campos.", Toast.LENGTH_LONG).show();
    }
}

public void regpatron_regemb(View view) {

    Intent regpatron_regemb = new Intent(this, RegistrarBarco.class);
    startActivity(regpatron_regemb);
}
}

```

Actividad *Registrar embarcación*

```

package com.example.appmarre;

import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

```

```
import com.parse.FindCallback;
import com.parse.ParseException;
import com.parse.ParseObject;
import com.parse.ParseQuery;
import com.parse.SaveCallback;

import java.util.List;

public class RegistrarBarco extends AppCompatActivity {

    String nombrebarco;
    String eslorabarco;
    String mangabarco;
    String caladobarco;
    String matricula;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_registrar_barco);

        Log.i("regbarco", "onCreate");

        final ParseQuery<ParseObject> query = ParseQuery.getQuery("Configuracionbarco");
        query.fromLocalDatastore();
        query.findInBackground(new FindCallback<ParseObject>() {
            @Override
            public void done(List<ParseObject> configbarco, ParseException e) {
                if (e == null) {
                    Log.i("regbarco", "pegar");
                    String[] configbarcoString = new String[configbarco.size()];
                    Log.i("Regbarco", Integer.toString(configbarco.size()));
                    if (configbarcoString != null) {
                        query.whereEqualTo("barco", "barco");
                        query.findInBackground(new FindCallback<ParseObject>() {
                            @Override
                            public void done(List<ParseObject> configbarcolist,
                            ParseException e) {
                                if (e == null) {
                                    for (ParseObject configinicial : configbarcolist) {
                                        EditText et_nombrebarco = (EditText)
                                        findViewById(R.id.et_nombrebarco);
                                        EditText et_eslora = (EditText)
                                        findViewById(R.id.et_eslorabarco);
                                        EditText et_manga = (EditText)
                                        findViewById(R.id.et_mangabarco);
                                        EditText et_calado = (EditText)
                                        findViewById(R.id.et_caladobarco);
                                        EditText et_matricula = (EditText)
                                        findViewById(R.id.et_matricula);

                                        String nombre =
                                        configinicial.getString("nombrebarco");
                                        et_nombrebarco.setText(nombre);
                                        String matricula =
                                        configinicial.getString("matricula");
                                        et_matricula.setText(matricula);
                                        String eslora =
                                        configinicial.getString("eslorabarco");
                                        et_eslora.setText(eslora);
                                        String manga =
                                        configinicial.getString("mangabarco");
                                        et_manga.setText(manga);
                                        String calado =
                                        configinicial.getString("caladobarco");
                                        et_calado.setText(calado);
                                    }
                                }
                            }
                        });
                    }
                }
            }
        });
    }
}
```

```

    });
}

public void RegistrarBarco (View view){

    nombrebearco = ((EditText) findViewById(R.id.et_nombrebearco)).getText().toString();
    eslorabearco = ((EditText) findViewById(R.id.et_eslorabearco)).getText().toString();
    mangabearco = ((EditText) findViewById(R.id.et_mangabearco)).getText().toString();
    caladobearco = ((EditText) findViewById(R.id.et_caladobearco)).getText().toString();
    matricula = ((EditText) findViewById(R.id.et_matricula)).getText().toString();

    if
(!nombrebearco.isEmpty()&&!eslorabearco.isEmpty()&&!mangabearco.isEmpty()&&!caladobearco.isEmpty()
&&!matricula.isEmpty()){

        ParseObject config = new ParseObject("Configuracionbarco");
        config.put("barco", "barco");
        config.put("nombrebearco", nombrebearco);
        config.put("eslorabearco", eslorabearco);
        config.put("mangabearco", mangabearco);
        config.put("caladobearco", caladobearco);
        config.put("matricula", matricula);
        config.pinInBackground(new SaveCallback() {
            @Override
            public void done(ParseException e) {
                if (e == null)
                    Log.i("localdbbarco", "guardado");
                else {
                    Log.i("localdbbarco", "error");
                    e.printStackTrace();
                }
            }
        });

        Toast.makeText(this,"Registro exitoso",Toast.LENGTH_LONG).show();
    }else{
        Toast.makeText(this,"Debe rellenar todos los campos.",Toast.LENGTH_LONG).show();
    }

}

public void regemb_mainact2(View view) {
    Intent regemb_mainact2 = new Intent(this, Main2Activity.class);
    startActivity(regemb_mainact2);
}
}

```

Actividad Principal

```
package com.example.appmarre;  
  
import androidx.appcompat.app.AppCompatActivity;  
  
import android.content.Intent;  
import android.os.Bundle;  
import android.util.Log;  
import android.view.View;  
  
import com.parse.FindCallback;  
import com.parse.ParseException;  
import com.parse.ParseObject;
```

```
import com.parse.ParseQuery;

import java.util.List;

public class Main2Activity extends AppCompatActivity {

    String nombrebarco;
    String eslorastring;
    String mangastring;
    String caladostring;
    String matricula;
    String nombrepatron;
    String apellidos;
    String dni;
    String telefono;
    String correo;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main2);

        final ParseQuery<ParseObject> query = ParseQuery.getQuery("Configuracionbarco");
        query.whereEqualTo("barco", "barco");
        query.fromLocalDatastore();
        query.findInBackground(new FindCallback<ParseObject>() {
            @Override
            public void done(List<ParseObject> config, ParseException e) {
                if (e==null){
                    if (config.size()!=0){
                        for(ParseObject configbarco:config){
                            nombrebarco = configbarco.getString("nombrebarco");
                            eslorastring = configbarco.getString("eslorabarco");
                            mangastring = configbarco.getString("mangabarco");
                            caladostring = configbarco.getString("caladobarco");
                            matricula = configbarco.getString("matricula");
                            Log.d("busqueda esloras", eslorastring);
                        }
                    }
                }
            }
        });
        final ParseQuery<ParseObject> query2 = ParseQuery.getQuery("Configuracion");
        query2.whereEqualTo("patron", "patron");
        query2.fromLocalDatastore();
        query2.findInBackground(new FindCallback<ParseObject>() {
            @Override
            public void done(List<ParseObject> config, ParseException e) {
                if (e==null){
                    if (config.size()!=0){
                        for(ParseObject configpatron:config){
                            nombrepatron = configpatron.getString("nombre");
                            apellidos = configpatron.getString("apellidos");
                            dni = configpatron.getString("DNI");
                            telefono = configpatron.getString("telefono");
                            correo = configpatron.getString("correo");
                        }
                    }
                }
            }
        });
    }

    public void mainact2_fechas (View view) {
        Intent mainact2_fechas = new Intent(this, Fechas.class);
        mainact2_fechas.putExtra("esloras", eslorastring);
        mainact2_fechas.putExtra("manga", mangastring);
    }
}
```

```

        mainact2_fechas.putExtra("calado",caladostring);
        mainact2_fechas.putExtra("nombrebarco",nombrebarco);
        mainact2_fechas.putExtra("matricula",matricula);
        mainact2_fechas.putExtra("nombrepatron",nombrepatron);
        mainact2_fechas.putExtra("apellidos",apellidos);
        mainact2_fechas.putExtra("dni",dni);
        mainact2_fechas.putExtra("telefono",telefono);
        mainact2_fechas.putExtra("correo",correo);

        startActivity(mainact2_fechas);
    }
    public void mainact2_moddatos (View view) {
        Intent mainact2_moddatos = new Intent(this, RegistrarPatron.class);
        startActivity(mainact2_moddatos);
    }
}

```

Actividad *Selección de fechas*

```

package com.example.apppmarre;

import android.app.DatePickerDialog;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.TextView;

import androidx.appcompat.app.AppCompatActivity;

import java.util.Calendar;

public class Fechas extends AppCompatActivity implements View.OnClickListener {

    Button butfechallegada;
    TextView tvfechallegada;
    Button butfechasalida;
    TextView tvfechasalida;

    long fllegadamilis;
    String fllegadamilistring;
    long fsalidamilis;
    String fsalidamilistring;

    String eslora;
    String manga;
    String calado;
    String nombrebarco;
    String matricula;
    String nombrepatron;
    String apellidos;
    String dni;
    String telefono;
    String correo;

    private int dia, mes, ano;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fechas);
        butfechallegada = (Button) findViewById(R.id.but_fechallegada);
        tvfechallegada = (TextView) findViewById(R.id.tv_fechallegada);
        butfechallegada.setOnClickListener(this);
        butfechasalida = (Button) findViewById(R.id.but_fechasalida);
    }
}

```

```
tvfechasalida = (TextView) findViewById(R.id.tv_fechasalida);
butfechasalida.setOnClickListener(this);

Bundle datos=getIntent().getExtras();
eslora=datos.getString("eslora");
manga=datos.getString("manga");
calado=datos.getString("calado");
nombrebearco=datos.getString("nombrebearco");
matricula=datos.getString("matricula");
nombrepatron=datos.getString("nombrepatron");
apellidos=datos.getString("apellidos");
dni=datos.getString("dni");
telefono=datos.getString("telefono");
correo=datos.getString("correo");
}

@Override
public void onClick(View view) {

    if (view == butfechallegada) {
        final Calendar c = Calendar.getInstance();
        dia = c.get(Calendar.DATE);
        mes = c.get(Calendar.MONTH);
        ano = c.get(Calendar.YEAR);
        c.set(Calendar.HOUR_OF_DAY,0);
        c.set(Calendar.MINUTE,0);
        c.set(Calendar.SECOND,0);
        c.set(Calendar.MILLISECOND,0);

        final DatePickerDialog datePickerDialog = new DatePickerDialog(this, new
DatePickerDialog.OnDateSetListener() {

            @Override
            public void onDateSet(DatePicker view, int year, int month, int dayOfMonth)
{

                String date =(dayOfMonth + "/" + (month + 1) + "/" + year);
                tvfechallegada.setText(date);
                Calendar calendar =Calendar.getInstance();
                calendar.set(year,month,dayOfMonth);
                fllegadamilis =calendar.getTimeInMillis();
                fllegadamilistring=Long.toString(fllegadamilis);
                Log.d("fechallegada en milis",fllegadamilistring);
            }
        }, ano, mes, dia);
        datePickerDialog.show();
    }

    if (view == butfechasalida) {
        final Calendar c2 = Calendar.getInstance();
        dia = c2.get(Calendar.DATE);
        mes = c2.get(Calendar.MONTH);
        ano = c2.get(Calendar.YEAR);
        c2.set(Calendar.HOUR_OF_DAY,0);
        c2.set(Calendar.MINUTE,0);
        c2.set(Calendar.SECOND,0);
        c2.set(Calendar.MILLISECOND,0);

        DatePickerDialog datePickerDialog = new DatePickerDialog(this, new
DatePickerDialog.OnDateSetListener() {

            @Override
            public void onDateSet(DatePicker view, int years, int months, int
dayOfMonths) {

                String date2 =(dayOfMonths + "/" + (months + 1) + "/" + years);
                tvfechasalida.setText(date2);
```



```

        Calendar calendar2 =Calendar.getInstance();
        calendar2.set(years,months,dayOfMonths);
        fsalidamilis =calendar2.getTimeInMillis();
        fsalidamilistring=Long.toString(fsalidamilis);
        Log.d("fechasalida en milis",fsalidamilistring);
    }
}
        , ano, mes, dia);
    datePickerDialog.show();
}

}

public void fechas_filtro(View view) {

    Intent fechas_filtro = new Intent(this, Filtrar.class);
    fechas_filtro.putExtra("fechallegada", fllegadamilistring);
    fechas_filtro.putExtra("fechasalida", fsalidamilistring);
    fechas_filtro.putExtra("eslora", eslora);
    fechas_filtro.putExtra("manga", manga);
    fechas_filtro.putExtra("calado", calado);
    fechas_filtro.putExtra("nombrebarco", nombrebarco);
    fechas_filtro.putExtra("matricula", matricula);
    fechas_filtro.putExtra("nombrepatron", nombrepatron);
    fechas_filtro.putExtra("apellidos", apellidos);
    fechas_filtro.putExtra("dni", dni);
    fechas_filtro.putExtra("telefono", telefono);
    fechas_filtro.putExtra("correo", correo);
    startActivity(fechas_filtro);

}
}

```

Actividad *Filtrado de servicios*

```

package com.example.appmarre;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.CheckBox;
import android.widget.Toast;

import com.parse.ParseObject;

public class Filtrar extends AppCompatActivity {

    private CheckBox checkagua, checkelectricidad, checkviento, checkmediterranea;

    String fechallegada;
    String fechasalida;
    int filtro_agua;
    int filtro_electricidad;
    int filtro_viento;
    int filtro_mediterranea;
    String eslora;
    String manga;
    String calado;
    String nombrebarco;
    String matricula;
    String nombrepatron;
    String apellidos;
    String dni;

```

```
String telefono;
String correo;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_filtrar);

    Bundle datos=getIntent().getExtras();
    fechallegada=datos.getString("fechallegada");
    fechasalida=datos.getString("fechasalida");
    eslora=datos.getString("eslora");
    manga=datos.getString("manga");
    calado=datos.getString("calado");
    nombrebardo=datos.getString("nombrebardo");
    matricula=datos.getString("matricula");
    nombrepatron=datos.getString("nombrepatron");
    apellidos=datos.getString("apellidos");
    dni=datos.getString("dni");
    telefono=datos.getString("telefono");
    correo=datos.getString("correo");

    checkagua = (CheckBox)findViewById(R.id.cb_agua);
    checkelectricidad = (CheckBox)findViewById(R.id.cb_electricidad);
    checkviento = (CheckBox)findViewById(R.id.cb_resguardeviento);
    checkmediterranea = (CheckBox)findViewById(R.id.cb_mediterranea);
}

public void guardarfiltros (View view) {

    if(checkagua.isChecked()== true) {
        filtro_agua=1;
    }else{
        filtro_agua=0;
    }
    if(checkelectricidad.isChecked()== true) {
        filtro_electricidad=1;
    }else{
        filtro_electricidad=0;
    }
    if(checkviento.isChecked()== true) {
        filtro_viento=1;
    }else{
        filtro_viento=0;
    }
    if(checkmediterranea.isChecked()== true) {
        filtro_mediterranea=1;
    }else{
        filtro_mediterranea=0;
    }
    Toast.makeText(this,"Guardado",Toast.LENGTH_LONG).show();
}

public void filtros_listado (View view) {
    Intent filtros_lista = new Intent(this, ListadoPlazas.class);
    filtros_lista.putExtra("fechallegada", fechallegada);
```

```

        filtros_lista.putExtra("fechasalida", fechasalida);
        filtros_lista.putExtra("filtro_agua", Integer.toString(filtro_agua));

        filtros_lista.putExtra("filtro_electricidad", Integer.toString(filtro_electricidad));
        filtros_lista.putExtra("filtro_viento", Integer.toString(filtro_viento));
        filtros_lista.putExtra("filtro_medi", Integer.toString(filtro_mediterranea));
        filtros_lista.putExtra("eslora", eslora);
        filtros_lista.putExtra("manga", manga);
        filtros_lista.putExtra("calado", calado);
        filtros_lista.putExtra("nombrebarco", nombrebarco);
        filtros_lista.putExtra("matricula", matricula);
        filtros_lista.putExtra("nombrepatron", nombrepatron);
        filtros_lista.putExtra("apellidos", apellidos);
        filtros_lista.putExtra("dni", dni);
        filtros_lista.putExtra("telefono", telefono);
        filtros_lista.putExtra("correo", correo);

        startActivity(filtros_lista);
    }
}

```

Actividad *Listado de plazas disponibles*

```

package com.example.appmarre;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

import com.parse.FindCallback;
import com.parse.ParseException;
import com.parse.ParseObject;
import com.parse.ParseQuery;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

public class ListadoPlazas extends Activity {

    ListView listView;

    ArrayAdapter<String> adapter;

    String fechallegada;
    String fechasalida;
    String eslora;
    String manga;
    String calado;
    String nombrebarco;
    String matricula;
    String nombrepatron;
    String apellidos;
    String dni;
    String telefono;
    String correo;
    String filtro_agua;
    String filtro_electricidad;
    String filtro_viento;
    String filtro_medi;

```

```
String numeroplazastring;
```

```
@Override
```

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_listado_plazas);
```

```
    Bundle datos=getIntent().getExtras();  
    fechallegada=datos.getString("fechallegada");  
    Long fechallegadalong=Long.parseLong(fechallegada);  
    fechasalida=datos.getString("fechasalida");  
    Long fechasalidalong=Long.parseLong(fechasalida);  
    filtro_agua=datos.getString("filtro_agua");  
    filtro_electricidad=datos.getString("filtro_electricidad");  
    filtro_viento=datos.getString("filtro_viento");  
    filtro_medi=datos.getString("filtro_medi");  
    eslora=datos.getString("eslora");  
    manga=datos.getString("manga");  
    calado=datos.getString("calado");  
    nombreb barco=datos.getString("nombreb barco");  
    matricula=datos.getString("matricula");  
    nombrepatron=datos.getString("nombrepatron");  
    apellidos=datos.getString("apellidos");  
    dni=datos.getString("dni");  
    telefono=datos.getString("telefono");  
    correo=datos.getString("correo");
```

```
    Log.d("eslora",eslora);  
    Log.d("filtroviento",filtro_viento);  
    Log.d("filtromedi",filtro_medi);  
    Log.d("filtroelect",filtro_electricidad);  
    Log.d("fechallegada", "" + fechallegada);  
    Log.d("fechasalida", "" + fechasalida);  
    Log.d("filtro_agua",filtro_agua);
```

```
listView=(ListView)findViewById(R.id.lv_plazas);
```

```
ParseQuery<ParseObject> queryInicio =ParseQuery.getQuery("Reserva");  
queryInicio.whereLessThanOrEqualTo("fecha_llegada", fechallegadalong);  
queryInicio.whereGreaterThanOrEqualTo("fecha_salida", fechallegadalong);
```

```
ParseQuery<ParseObject> queryFin =ParseQuery.getQuery("Reserva");  
queryFin.whereLessThanOrEqualTo("fecha_llegada", fechasalidalong);  
queryFin.whereGreaterThanOrEqualTo("fecha_salida", fechasalidalong);
```

```
List<ParseQuery<ParseObject>> queries = new ArrayList<ParseQuery<ParseObject>>();  
queries.add(queryInicio);  
queries.add(queryFin);
```

```
ParseQuery<ParseObject> queryOr = ParseQuery.or(queries);  
queryOr.findInBackground(new FindCallback<ParseObject>() {
```

```
    @Override
```

```
    public void done(List<ParseObject> objects, ParseException e) {  
        if (e == null) {
```

```
            int vientoint = Integer.parseInt(filtro_viento);  
            int mediint = Integer.parseInt(filtro_medi);  
            int esloraInt = Integer.parseInt(eslora);  
            int mangaint = Integer.parseInt(manga);  
            int caladoint = Integer.parseInt(calado);
```

```
            List<Integer> plazas = new LinkedList<Integer>();  
            for (ParseObject object : objects) {  
                Integer plaza = object.getInt("numero");  
                if (!plazas.contains(plaza)) {
```

```

        plazas.add(plaza);
    }

    Log.d("Appmarres", "Plaza: " + plaza);
}
ParseQuery<ParseObject> queryAmarre = ParseQuery.getQuery("Amarre");
queryAmarre.whereEqualTo("agua", 1);
queryAmarre.whereEqualTo("electricidad", 1);
queryAmarre.whereEqualTo("resguardo_viento", vientoint);
queryAmarre.whereEqualTo("atraque_mediterranea", mediint);
queryAmarre.whereGreaterThan("eslora", esloraint);
queryAmarre.whereGreaterThan("manga", mangaint);
queryAmarre.whereGreaterThan("calado", caladoint);
queryAmarre.whereEqualTo("ocupacion", 0);

if (plazas.size() != 0) {
    queryAmarre.whereNotContainedIn("numero", plazas);
}
queryAmarre.findInBackground(new FindCallback<ParseObject>() {
    @Override
    public void done(List<ParseObject> amarres, ParseException e) {
        if (e == null) {
            int i = 0;
            final String[] resultsAsString = new
String[amarres.size()];

            Log.d("Query", Integer.toString(amarres.size()));
            for (ParseObject amarre : amarres) {
                final int numero = (Integer) amarre.get("numero");
                final String post = (String) amarre.get("post");
                listView.setOnItemClickListener(new
                    @Override
                    public void onItemClick(AdapterView<?> parent, View
                        Object numeroplaza =
                            numeroplazastring =

                            Log.d("Parse post", numeroplazastring);

                            Intent intent = new Intent(ListadoPlazas.this,
                                intent.putExtra("numeroplazaseleccionada",

                                    intent.putExtra("fechallegada", fechallegada);
                                    intent.putExtra("fechasalida", fechasalida);
                                    intent.putExtra("filtro_agua", filtro_agua);
                                    intent.putExtra("filtro_electricidad",

                                        intent.putExtra("filtro_viento",

                                            intent.putExtra("filtro_medi", filtro_medi);
                                            intent.putExtra("eslora", eslora);
                                            intent.putExtra("manga", manga);
                                            intent.putExtra("calado", calado);
                                            intent.putExtra("nombrebarco", nombrebarco);
                                            intent.putExtra("matricula", matricula);
                                            intent.putExtra("nombrepatron", nombrepatron);
                                            intent.putExtra("apellidos", apellidos);
                                            intent.putExtra("dni", dni);
                                            intent.putExtra("telefono", telefono);
                                            intent.putExtra("correo", correo);
                                            startActivity(intent);
                                        }
                                    });
            }
        }
    }
});

```

```
        resultsAsString[i++] = post;

        Log.d("Parse Query", Integer.toString(numero));
    }
    prepareListView(resultsAsString);

    } else {
        Toast.makeText(ListadoPlazas.this, "No hay plazas
disponibles para sus requerimientos", Toast.LENGTH_LONG).show();
        Log.w("Parse Query", e.getMessage());
    }
    }
    });

    }else{
        Toast.makeText(ListadoPlazas.this, "No hay plazas disponibles para las
fechas seleccionadas", Toast.LENGTH_LONG).show();
        Log.w("Parse Query 1", e.getMessage());
    }
    }

    });

    }

    public void prepareListView(String[] resultsAsString) {
        adapter = new
        ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, resultsAsString);
        listView.setAdapter(adapter);
    }
}
```

Actividad Información de la plaza seleccionada

```
package com.example.appmarre;

import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.TextView;

import androidx.appcompat.app.AppCompatActivity;

import com.parse.FindCallback;
import com.parse.ParseException;
import com.parse.ParseObject;
import com.parse.ParseQuery;
import com.parse.SaveCallback;

import java.util.List;

public class Infoplazaamarre1 extends AppCompatActivity {

    TextView tv_numero;
    TextView tv_puerto; TextView tv_eslora; TextView tv_manga; TextView tv_calado; TextView
tv_distancia; TextView tv_agua;
    TextView tv_electricidad; TextView tv_viento; TextView tv_medi;

    String fechallegada;
    String fechasalida;
    String filtro_agua;
    String filtro_electricidad;
    String filtro_viento;
```

```

String filtro_medi;
String eslora;
String manga;
String calado;
String numeroobtenido;
String nombrebarco;
String matricula;
String nombrepatron;
String apellidos;
String dni;
String telefono;
String correo;

int numeroplaza;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_infoplazamarrel);

    Bundle datos=getIntent().getExtras();
    fechallegada=datos.getString("fechallegada");
    fechasalida=datos.getString("fechasalida");
    filtro_agua=datos.getString("filtro_agua");
    filtro_electricidad=datos.getString("filtro_electricidad");
    filtro_viento=datos.getString("filtro_viento");
    filtro_medi=datos.getString("filtro_medi");
    eslora=datos.getString("eslora");
    manga=datos.getString("manga");
    calado=datos.getString("calado");
    numeroobtenido=datos.getString("numeroplazaseleccionada");
    nombrebarco=datos.getString("nombrebarco");
    matricula=datos.getString("matricula");
    nombrepatron=datos.getString("nombrepatron");
    apellidos=datos.getString("apellidos");
    dni=datos.getString("dni");
    telefono=datos.getString("telefono");
    correo=datos.getString("correo");

    tv_numero=findViewById(R.id.tv_numero);
    tv_puerto=findViewById(R.id.tv_puerto);
    tv_eslora=findViewById(R.id.tv_eslora);tv_manga=findViewById(R.id.tv_manga);

    tv_calado=findViewById(R.id.tv_calado);tv_distancia=findViewById(R.id.tv_distancia);

    tv_agua=findViewById(R.id.tv_agua);tv_electricidad=findViewById(R.id.tv_electricidad);
    tv_viento=findViewById(R.id.tv_viento);tv_medi=findViewById(R.id.tv_mediterranea);

    //obtener el numero de la plaza
    ParseQuery<ParseObject> numplaza= ParseQuery.getQuery("Amarre");
    numplaza.whereEqualTo("post", numeroobtenido);
    numplaza.findInBackground(new FindCallback<ParseObject>() {
        @Override
        public void done(List<ParseObject> plaza, ParseException e) {
            if (e==null){
                for (ParseObject buscapost:plaza){
                    numeroplaza=buscapost.getInt("numero");
                    Log.d("plazapulsada",Integer.toString(numeroplaza));
                }
                ParseQuery<ParseObject> query = ParseQuery.getQuery("Amarre");
                query.whereEqualTo("numero", numeroplaza);
                query.findInBackground(new FindCallback<ParseObject>() {
                    @Override
                    public void done(List<ParseObject> amarrelist, ParseException e) {
                        Log.d("done","dentro");
                        if (e==null){
                            Log.d("if","dentro " + amarrelist.size());
                        }
                    }
                });
            }
        }
    });
}

```

```

for (ParseObject amarre : amarrelist) {
    Log.d("for", "dentro");

    int numero = amarre.getInt("numero");
    String numerostring = Integer.toString(numero);
    tv_numero.setText("Número plaza de amarre: " +
numerostring);

    String puerto = amarre.getString("puerto");
    tv_puerto.setText("Puerto de amarre: " + puerto);
    int eslora = amarre.getInt("eslora");
    String eslorastring = Integer.toString(eslora);
    tv_eslora.setText("Eslora plaza de amarre: " +
eslorastring + " m");

    int manga = amarre.getInt("manga");
    String mangastring = Integer.toString(manga);
    tv_manga.setText("Manga plaza de amarre: " +
mangastring + " m");

    int calado = amarre.getInt("calado");
    String caladostring = Integer.toString(calado);
    tv_calado.setText("Calado plaza de amarre: " +
caladostring + " m");

    int distancia = amarre.getInt("distancia_al_club");
    String distanciastring = Integer.toString(distancia);
    tv_distancia.setText("Distancia al club: " +
distanciastring + " m");

    int agua = amarre.getInt("agua");
    if(agua==1){
        tv_agua.setText("Agua: Sí");
    }else{
        tv_agua.setText("Agua: No");
    }

    int electricidad = amarre.getInt("electricidad");
    if(electricidad==1){
        tv_electricidad.setText("Corriente eléctrica: Sí");
    }else{
        tv_electricidad.setText("Corriente eléctrica: No");
    }

    int viento = amarre.getInt("resguardo_viento");
    if(viento==1){
        tv_viento.setText("Plaza resguardada del viento:
Sí");
    }else{
        tv_viento.setText("Plaza resguardada del viento:
No");
    }

    int medi = amarre.getInt("ataque_mediterranea");
    if(medi==1){
        tv_medi.setText("Ataque a la mediterránea: Sí");
    }else{
        tv_medi.setText("Ataque a la mediterránea: No");
    }
}

} else{

    Log.w("Parse Query", e.getMessage());

}

});
} else{
    Log.w("Query numplaza", e.getMessage());
}
});
}

```



```

}

public void reservar_final (View view) {

    ParseObject reserva = new ParseObject("Reserva");
    reserva.put("nombrebarco", nombrebarco);
    reserva.put("matricula", matricula);
    reserva.put("fecha_llegada", Long.parseLong(fechallegada));
    reserva.put("fecha_salida", Long.parseLong(fechasalida));
    reserva.put("nombrepatron",nombrepatron );
    reserva.put("apellidos", apellidos);
    reserva.put("dni", dni);
    reserva.put("telefono",telefono );
    reserva.put("correo", correo);
    reserva.put("plaza_reservada", numeroobtenido );

    reserva.saveInBackground(new SaveCallback() {
        @Override
        public void done(ParseException e) {
            if (e == null)
                Log.i("reserva", "bien");
            else {
                Log.i("reserva", "error");
                e.printStackTrace();
            }
        }
    });
    ParseObject datosPatron = new ParseObject("Patron");
    datosPatron.put("nombre", nombrepatron);
    datosPatron.put("apellidos", apellidos);
    datosPatron.put("DNI", Integer.parseInt(dni));
    datosPatron.put("correo", correo);
    datosPatron.put("telefono", Integer.parseInt(telefono));
    datosPatron.saveInBackground(

        new SaveCallback() {
            @Override
            public void done(ParseException e) {
                if (e == null)
                    Log.i("registropatron", "SALVADO!");
                else {
                    Log.i("registropatron", "NO SALVADO!");
                    e.printStackTrace();
                }
            }
        }
    );

    ParseObject datosbarco = new ParseObject("Embarcacion");
    datosbarco.put("nombrebarco", nombrebarco);
    datosbarco.put("eslorabarco", eslora);
    datosbarco.put("mangabarco", manga);
    datosbarco.put("caladobarco", calado);
    datosbarco.put("matricula", matricula);
    datosbarco.saveInBackground(new SaveCallback() {
        @Override
        public void done(ParseException e) {
            if (e == null)
                Log.i("registrobarco", "bien");
            else {
                Log.i("registrobarco", "error");
                e.printStackTrace();
            }
        }
    });
    Intent reserva_final = new Intent(Infoplazaamarre1.this, FinalActivity.class);
    reserva_final.putExtra("correo", correo);

```

```
        startActivity(reserva_final);
    }
    public void infoplaza_infopuerto(View view) {
        Intent infoplaza_infopuerto = new Intent(this, PuertoInfo.class);
        infoplaza_infopuerto.putExtra("numeroobtenido", Integer.toString(mercadoplaza));
        startActivity(infoplaza_infopuerto);
    }
}
```

Actividad *Información del puerto al que pertenece la plaza*

```
package com.example.appmarre;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

import com.parse.FindCallback;
import com.parse.ParseException;
import com.parse.ParseObject;
import com.parse.ParseQuery;

import java.util.List;

public class PuertoInfo extends AppCompatActivity {

    String numeroobtenido;
    String puerto;

    TextView tv_nombrepuerto;
    TextView tv_aseos;
    TextView tv_duchas;
    TextView tv_carburante;
    TextView tv_wifi;
    TextView tv_reciclaje;
    TextView tv_recaguasresiduales;
    TextView tv_videovig;
    TextView tv_vignocturno;
    TextView tv_lavanderia;
    TextView tv_ventahielo;
    TextView tv_acastillaje;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_puerto_info);

        Bundle datos=getIntent().getExtras();
        numeroobtenido=datos.getString("numeroobtenido");

        Log.d("numero", numeroobtenido);

        tv_nombrepuerto=findViewById(R.id.tv_nombrepuerto);

        tv_aseos=findViewById(R.id.tv_aseos);
        tv_duchas=findViewById(R.id.tv_duchas);
        tv_carburante=findViewById(R.id.tv_carburante);
        tv_wifi=findViewById(R.id.tv_wifi);
        tv_reciclaje=findViewById(R.id.tv_reciclaje);
        tv_recaguasresiduales=findViewById(R.id.tv_recaguasresiduales);
        tv_videovig=findViewById(R.id.tv_videovigilancia);
        tv_vignocturno=findViewById(R.id.tv_vignocturno);
        tv_lavanderia=findViewById(R.id.tv_lavanderia);
        tv_ventahielo=findViewById(R.id.tv_ventahielo);
        tv_acastillaje=findViewById(R.id.tv_acastillaje);
    }
}
```

```

ParseQuery<ParseObject> query = ParseQuery.getQuery("Amarre");
query.whereEqualTo("numero", Integer.parseInt(numeroobtenido));
query.findInBackground(new FindCallback<ParseObject>() {
    @Override
    public void done(List<ParseObject> objects, ParseException e) {
        if (e==null){
            for (ParseObject plazaelegida: objects){
                puerto=plazaelegida.getString("puerto");
                Log.d("puerto",puerto);
                ParseQuery<ParseObject> query2 = ParseQuery.getQuery("Puerto");
                query2.whereEqualTo("nombre", puerto);
                query2.findInBackground(new FindCallback<ParseObject>() {
                    @Override
                    public void done(List<ParseObject> plazaspuertos,
ParseException e) {
                        if (e==null){
                            Log.d("if", "dentro"+plazaspuertos.size());

                            for (ParseObject puertos : plazaspuertos){
                                Log.d("for", "dentro");

                                tv_nombrepuerto.setText("Puerto de "+puerto);

                                int aseos =puertos.getInt("aseos");
                                if(aseos==1){
                                    tv_aseos.setText("Aseos: Sí");
                                }else{
                                    tv_aseos.setText("Aseos: No");
                                }
                                int duchas =puertos.getInt("duchas");
                                if(duchas==1){
                                    tv_duchas.setText("Duchas: Sí");
                                }else{
                                    tv_duchas.setText("Duchas: No");
                                }
                                int carburante =puertos.getInt("carburante");
                                if(carburante==1){
                                    tv_carburante.setText("Carburante: Sí");
                                }else{
                                    tv_carburante.setText("Carburante: No");
                                }
                                int wifi =puertos.getInt("WiFi");
                                if(wifi==1){
                                    tv_wifi.setText("WiFi: Sí");
                                }else{
                                    tv_wifi.setText("WiFi: No");
                                }
                                int reciclaje =puertos.getInt("reciclaje");
                                if(reciclaje==1){
                                    tv_reciclaje.setText("Reciclaje: Sí");
                                }else{
                                    tv_reciclaje.setText("Reciclaje: No");
                                }
                                int recaguas
=puertos.getInt("recuperacion_aguas_residuales");
                                if(recaguas==1){
                                    tv_recaguasresiduales.setText("Recuperación de
aguas residuales: Sí");
                                }else{
                                    tv_recaguasresiduales.setText("Recuperación de
aguas residuales: No");
                                }
                                int videovig =puertos.getInt("video_vigilancia");
                                if(videovig==1){
                                    tv_videovig.setText("VÍdeo vigilancia: Sí");

```

```
}else{  
    tv_videovig.setText("VÍdeo vigilancia: No");  
}  
int vignoc =puertos.getInt("vigilante_nocturno");  
if(vignoc==1){  
    tv_vignocturno.setText("Vigilante nocturno:  
Sí");  
  
No");  
  
}  
else{  
    tv_vignocturno.setText("Vigilante nocturno:  
  
});  
  
});  
  
});  
  
});  
  
});  
  
});  
  
});
```

Actividad Final

```
package com.example.appmarre;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

import com.parse.FindCallback;
import com.parse.GetCallback;
import com.parse.Parse;
import com.parse.ParseException;
import com.parse.ParseObject;
import com.parse.ParseQuery;

import java.util.List;
```

```

public class FinalActivity extends AppCompatActivity {

    private TextView tv_final;

    String correo;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_final);

        Bundle datos=getIntent().getExtras();
        correo=datos.getString("correo");

        tv_final=findViewById(R.id.tv_final);

        tv_final.setText("¡Su reserva ha sido confirmada! ¡Se remitirá un correo a "+correo+"!");
    }

    public void final_menuprincipal (View view) {
        Intent volvermenuprincipal = new Intent(this, Main2Activity.class);
        startActivity(volvermenuprincipal);
    }
}

```