Universitat Oberta de Catalunya

uoc.edu

Gestión de aparcamiento en el trabajo

Beatriz Domínguez Callero
Máster en desarrollo de aplicaciones móviles
Desarrollo de aplicación

Nombre Profesor/a responsable de la asignatura:
Carles Garrigues Olivella

Fecha de entrega:
03/01/2020

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Gestión de aparcamiento en el trabajo*
Nombre del autor:	*Beatriz Domínguez Callero*
Nombre del consultor/a:	*Pau Dominkovics Coll*
Nombre del PRA:	*Carles Garrigues Olivella*
Fecha de entrega (mm/aaaa):	01/2019
Titulación::	*Máster en desarrollo de aplicaciones móviles*
Área del Trabajo Final:	*Trabajo de Fin de Máster*
Idioma del trabajo:	*Español*
Palabras clave	*Comparte aparcamiento*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

El trabajo consiste en el diseño e implementación de una aplicación móvil Android para la gestión de plazas de aparcamiento en el trabajo debido a la dificultad de comunicación entre los empleados para ello.

Se utiliza una metodología en cascada ya que los objetivos, requisitos y fechas están perfectamente definidos desde el principio del proyecto. Dividiéndose el proyecto en las siguientes fases, que encajan con las entregas parciales del proyecto: requisitos, diseño, implementación, verificación y mantenimiento.

Como producto se obtiene una aplicación Android (un archivo .apk compilado) que cumple los objetivos y requisitos.

Además de este archivo compilado, se entregará el código fuente, un manual de ejecución y compilación, un manual de uso, una memoria con la explicación de cada una de las fases y un video explicativo de los aspectos más importantes.

Con este trabajo se consiguen los objetivos propuestos: investigar y conocer las tecnologías necesarias para una aplicación móvil, aprender a planificar y gestionar el proyecto de inicio a fin, diseñar e implementar la aplicación.

Para finalizar, se proponen diferentes cambios a futuro derivados del escaso tiempo de implementación.

Abstract (in English, 250 words or less):

The projects consists in the design and implementation of an Android mobile application for the management of parking at work due to the difficulty of communication between employees for it.

A cascade methodology is used because the objectives, requirements and dates are defined from the beginning of the project. The project is divided into the following phases, which are part of the project's milestones: requirements, design, implementation, verification and maintenance.

You get an Android application (a compiled .apk file) that fits the objectives and requirements.

In addition to this compiled file, the source code, an execution and compilation manual, a user manual, a memory with the explanation of each of the phases and an explanatory video of the most important aspects are delivered.

With this project, the proposed objectives are achieved: research and know the necessary technologies for a mobile application, learn to plan and manage the project from start to finish, design and implement the application.

Finally, future changes are proposed due to the short time of implementation.

Índice

1. Introducción

Contexto y justificación del Trabajo

Se propone crear una aplicación que resuelva el actual problema de las plazas de aparcamiento en el trabajo.

Las plazas de aparcamiento de la empresa son compartidas por dos personas.
Normalmente, los trabajadores suelen elegir la persona con la que comparten plaza de aparcamiento. Suele ser otro trabajador con el que tienen más relación. Esta gestión de plaza no es complicada, ya que suele haber bastante comunicación entre compañeros, por lo que la aplicación no resolverá esta gestión a día de hoy (se propondrá como solución a futuro).

En esta empresa es común que los trabajadores tengan reuniones en clientes, estén desplazados durante varios días o tengan teletrabajo. Si sumamos también los periodos de vacaciones, nos encontramos con que muchos días ninguno de los dos compañeros necesita la plaza y esta se queda vacía durante mucho tiempo.

Si se accede al aparcamiento de la empresa, se puede observar cómo más de la mitad de los aparcamientos están vacíos. Sin embargo, muchas de las personas con plaza asignada están aparcando fuera del recinto porque la plaza le pertenecía a su compañero ese día.

Como la comunicación entre las demás personas que no son el compañero de plaza es complicada, puesto que somos muchos trabajadores, la aplicación pretende resolver este problema.

Se crea por ello una aplicación móvil que gestione las plazas de aparcamiento entre las personas que no son compañeros de plaza.

Los teléfonos móviles corporativos son Android, por lo que se creará una aplicación Android.

Contexto

Como se ha comentado en el apartado anterior, el objetivo del trabajo es crear una aplicación Android de gestión de aparcamiento para los trabajadores con los que no se comparte plaza. De esta forma, se pretende:

- Asegurar plaza de aparcamiento el mayor número de días posible.
- Reducir los tiempos en ir a trabajar
- Reducir el número de plazas sin usar

A continuación, se enumeran los requisitos de la aplicación:

Requisitos funcionales	
RF1	La aplicación permitirá a los usuarios registrarse
RF2	La aplicación permitirá a los usuarios iniciar sesión con su email y contraseña
RF3	La aplicación permitirá a los usuarios introducir su plaza de aparcamiento asignada por la empresa
RF4	La aplicación permitirá liberar la plaza de aparcamiento asignada a un usuario y su compañero para un día específico
RF5	La aplicación permitirá reservar la plaza de aparcamiento de otro usuario para un día específico
RF6	La aplicación permitirá cerrar sesión del usuario del usuario
Requisitos no funcionales	
RNF1	La aplicación debe ser una aplicación para móviles Android
RNF2	La aplicación debe ser compatible con Android 8.0 o superior (versión móvil corporativo)
RNF3	La aplicación debe funcionar y verse correctamente en un Galaxy J6 (móvil corporativo), aunque deberá adaptarse a todos los dispositivos posibles del mercado
RNF4	La aplicación debe ser fácil e intuitiva
RNF5	La aplicación debe ser fácil de desarrollar y mantener
Requisitos Software para el proyecto	
RS1	Android Studio para el desarrollo
RS2	API + base de datos (o Firebase)
Requisitos Hardware para el proyecto:	
RH1	Ordenador con posibilidad de instalar Android para el desarrollo
RH2	Teléfono móvil Android 8.0 Galaxy J6 para pruebas

Enfoque y método seguido

Estudio de aplicaciones:

Existen numerosas aplicaciones para compartir plazas de aparcamiento particulares, buscar *parking* privados, etcétera. Sin embargo, es difícil encontrar aplicaciones que solucionen el problema de compartir plazas de aparcamiento en empresas o edificios.

En la búsqueda, se obtiene una empresa llamada SCLConsulting que ofrece una solución para la gestión de plazas en oficinas: *App SAP Fiori para la gestión de plazas de aparcamiento.*
Esta aplicación, se basa en que cada plaza de aparcamiento está asignada a un usuario y, por defecto, esta plaza está ocupada por el usuario siempre a no ser que éste la libere. Si la plaza es liberada, otro usuario podrá ocupar su plaza. En caso de que el usuario quiera rectificar y ocupar de nuevo una plaza ese día, la aplicación automáticamente intentará reservar su plaza. Si está ocupada, buscará otra. Si no hay plazas libres, no podrá reservar.

La aplicación, además, también permite plazas sin dueño.

El diseño de la aplicación consta de una pantalla en la que se puede elegir el día en el que liberar/reservar plaza y dos botones: "Reservar" o "Liberar".

El diseño inicial y funcionamiento que se presentaba para el proyecto era algo más complejo y no contemplaba la posibilidad de que la aplicación compruebe si la plaza del usuario está ocupada o no a la hora de reservar, sino que forzaba al usuario a verlo en dos pantallas.
El diseño y usabilidad de la aplicación SAP Fiori también es mucho más sencillo, por lo que lo usará como base para comenzar el diseño de la aplicación a desarrollar.

En la nueva aplicación, es necesario que la plaza pertenezca a dos personas y no es necesario gestión de plazas libres ya que todas las plazas están asignadas a dos trabajadores. Cuando se quedan vacías, es la empresa quién busca otros dos trabajadores que opten a ella.

App SAP Fiori es una solución para empresas y es necesario contactar con ellos para obtener un usuario, por tanto, no se ha podido probar su funcionamiento.

En la *bibliografía*, se incluye un video de la herramienta que explica todo el funcionamiento y diseño.

Creación de producto

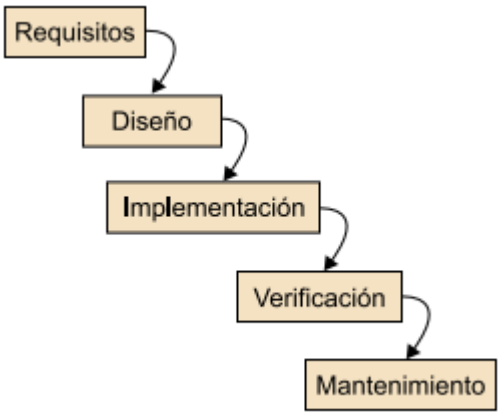
Puesto que es una aplicación a medida con unos requisitos específicos y no se dispone de código fuente, ni producto que pueda servir de base para el desarrollo, se va a desarrollar una nueva aplicación, aunque se utilice *App SAP Fiori* para recoger las ideas de diseño y funcionalidad aportadas anteriormente.

Metodología

Se plantea una metodología en cascada ya que partimos de un entorno no volátil, donde los requisitos y objetivos se definirán al inicio del proyecto y no sufrirán grandes cambios durante el ciclo de vida de desarrollo. Además, debido a las entregas parciales y las fechas, es necesario dividir el proyecto en fases totalmente secuenciales y dar mucho énfasis a la planificación, los tiempos y fechas límites.

Se descartan por ello otras metodologías, como la ágil, en la que los requisitos no están tan claros y se debe ir dando respuesta a medida que se va desarrollando el producto; o las metodologías rápidas, en las que se busca entregar un prototipo en corto plazo para, después, ir mejorándolo.

Por todas estas razones, se elige la metodología en cascada y que se dividirá en las siguientes fases:



Requisitos:

Se estudian las necesidades de los futuros usuarios de la aplicación para determinar todas las funcionalidades que debe cubrir el producto.
De aquí obtendremos una lista de funcionalidades a cubrir.

Diseño:

En esta etapa se busca definir la tecnología a utilizar, la arquitectura y diseño de los distintos elementos que intervienen y las relaciones entre ellas.
Esta fase debe definir todo lo que se implementará en la siguiente fase.

Implementación:

Se utilizará el diseño de la fase anterior para implementar y poner en producción todos los elementos que intervienen.

Verificación:

Aunque la etapa de implementación también supone la realización de pruebas, en esta etapa se busca la ejecución del producto en un entorno real, de manera que se encuentre la mayor parte de errores o incompatibilidades y se puedan resolver.
Una vez pasada esta fase, el producto estará listo y pasará a la fase de mantenimiento.

Mantenimiento:

Se distribuye el producto a los usuarios y entra en fase de mantenimiento.
La tecnología va cambiando y pueden surgir nuevas necesidades, nuevos bugs por nuevas versiones, cambios en el diseño, etcétera.

Planificación del Trabajo

En el *ANEXO 1.* se puede observar la planificación que se va a llevar a cabo.

A la izquierda del anexo, aparece cada tarea dividida en subtareas con la fecha de inicio y fin de cada actividad. También aparece el número de horas dedicado.
A la derecha, en verde, se muestra la estimación de las tareas en el tiempo. En esta estimación aparecen números, que corresponden con el número de horas dedicado cada día para poder identificar los días festivos y los laborables.

Cómo se puede observar, se intenta dedicar 8 horas los viernes y días festivos (sábados y domingos), y dos horas los días laborables, debido a mi horario laboral de Lunes a Viernes.

Creo que podría haber dificultades o imprevistos sobre todo en las fases de diseño y desarrollo del API y base de datos por desconocimiento de tecnologías, por lo que, en caso de no poder cumplir con la planificación prevista, se aumentarán las horas de trabajo aumentando así los días laborables de 2 a 8-10 horas de trabajo. Esto será posible gracias a que aún dispongo de vacaciones en mi trabajo.

Otra opción posible es reducir la funcionalidad de registro de usuarios en la aplicación dando directamente los usuarios registrados en la base de datos de Firebase. Se intentará no realizar esta reducción porque se considera importante para la aplicación.

Breve sumario de productos obtenidos

De acuerdo con la planificación, podemos extraer los siguientes productos o entregables en cada una de las etapas:

Etapa de planificación

El objetivo de esta etapa es obtener una memoria con los siguientes productos:

- El concepto de la aplicación
- Planificación
- Listado de requisitos a alto nivel
- Una idea de las tecnologías que vamos a utilizar

Etapa de diseño

En esta etapa obtendremos una memoria con el diseño de todos los aspectos que componen la aplicación. Esta memoria contendrá:

- Una descripción de la aplicación
- Un análisis de las aplicaciones que se asemejan a esta en el mercado
- Encuestas a los usuarios, que nos permitirán detectar requisitos o puntos de vista que no habíamos detectado
- Descripción de los usuarios que van a usar la aplicación
- Casos de uso
- Listado de requisitos más detallado
- Esbozo de las pantallas y la navegación de la aplicación
- Prototipo del diseño mediante alguna herramienta
- UML o diagrama con las estructuras de datos que compondrán la aplicación.
- UML o diagrama con las estructuras que compondrán el servidor
- Diagrama del diseño del API
- Descripción del modelo que se usará en la aplicación

En cada apartado se justificarán las decisiones que se han tomado y los argumentos.

Etapa de implementación:

En esta etapa se obtendrá:

- El zip con el código fuente del servidor en caso de que exista. Posiblemente se use Firebase, por lo que este apartado no será entregable
- El zip con el código fuente de la aplicación a entregar
- El apk de la aplicación para su instalación
- Las instrucciones para la instalación y el uso de la aplicación
  Memoria con los problemas o decisiones que hayamos ido tomando en esta etapa

Etapa entrega final:

En esta etapa se obtendrá:

- El código fuente, apk e instrucciones totalmente terminadas
- Memoria con los ajustes o justificaciones que haya sido necesario respecto a la entrega anterior
- Video de máximo 20 minutos con la presentación

Breve descripción de los otros capítulos de la memoria

Se añaden tres capítulos a los ya existentes:

- Conceptualización
- Arquitectura
- Implementación

Los dos primeros corresponden con el diseño de la aplicación y, el último, con la implementación.

En el apartado siguiente se realiza una pequeña descripción de los capítulos a incluir.

2. Conceptualización

En este apartado se definen los aspectos relativos a la parte del concepto de la aplicación.

Descripción de la aplicación

Se desarrollará una aplicación Android para la gestión de plazas de aparcamiento en mi trabajo actual.
La aplicación permitirá a los usuarios registrarse, iniciar sesión, registrar su plaza y elegir el día en el que pueden liberar su plaza de aparcamiento o reservar una plaza.
Para ello, se conectará con un servidor que almacenará los datos de los usuarios, las plazas asignadas y el estado de la plaza (libre o reservado)

Análisis de aplicaciones similares

En el apartado "Estudio de aplicaciones" de *"Enfoque y método seguido"* se realizó el análisis de aplicaciones similares.
Cómo se comentó, no existen aplicaciones que se encarguen de esto, salvo App SAP Fiori, una aplicación que ofrece una empresa privada para la gestión de plazas de aparcamiento. Su funcionamiento se explica en el apartado citado.
De esta aplicación obtendremos ideas para el diseño y el funcionamiento de nuestra aplicación.

Encuestas a usuarios

Se crea una encuesta a través de Google Formularios y se distribuye a los usuarios para que puedan opinar sobre la aplicación. Puede consultarse en el siguiente enlace:

https://docs.google.com/forms/d/e/1FAIpQLSfDReE5wz2tt8BpPoa0ZQBDYmRPAfHIXuIjh8nDYKos_dJx0Q/viewform?usp=sf_link

En el *"ANEXO 2. Encuestas a usuarios sobre la aplicación"* se pueden consultar las capturas de pantalla de las encuestas.

En el "*ANEXO 3. Respuestas de las encuestas de los usuarios*" se pueden consultar las respuestas de los usuarios y las gráficas obtenidas a partir de ellas.

Estudio de respuestas:

Del estudio de respuestas se obtienen las siguientes conclusiones:

1. **Edad**

La edad no influye en nuestra aplicación.

2. **Marca y modelo de tu teléfono personal**

De once usuarios que han respondido a la encuesta, dos usan como teléfono personal iPhone.

3. **Marca y modelo de tu teléfono de empresa**

Aunque las respuestas sobre el modelo y tipo de terminal móvil de trabajo son diferentes (por el formato), todos los usuarios encuestados usan el Samsung Galaxy J6.

Anteriormente, se usaba Nokia con su sistema operativo, pero la mayoría ya no funcionaban en buenas condiciones, por lo que existirán muy pocos usuarios tenían este teléfono de empresa. La mayoría de los usuarios que lo tienen seguramente no lo utilizan.

4. **Existen trabajadores que incluyen la SIM del teléfono de empresa en su teléfono personal o redirigen las llamadas a su teléfono personal. ¿Es tu caso?**

El 72 % de los usuarios utilizan el móvil de empresa frente al 27,3 % de usuarios que utilizan su móvil personal como móvil de empresa. Esto no debería ser preocupante si los usuarios usasen un teléfono Android porque la aplicación debería funcionar sobre todos los teléfonos Android a partir de la versión elegida. Sin embargo, supone un problema en los usuarios que usan otro sistema operativo.

En este caso, de once usuarios que responden a la encuesta, solo un usuario utiliza su iPhone cómo móvil personal y de trabajo.

Es un riesgo que se puede asumir y permitir ya que, aunque use su iPhone cómo móvil personal, todos los empleados tienen Android como teléfono corporativo. Si el usuario decide usar la aplicación, deberá hacerlo con su teléfono corporativo.

Aun así, se registra este problema para un posible futuro desarrollo en iOS.

5. **¿Compartes plaza de aparcamiento?**

El 90,9% de usuarios comparte su plaza de aparcamiento, frente al 9,9%. Este 9,9% es un usuario que no tiene coche, por lo que no tiene plaza. Se puede no tener en cuenta.

6. **¿Cuántos días a la semana sueles aparcar en tu plaza?**

El 45% de usuarios suele aparcar tres días a la semana en su plaza; el 27,3 %, una vez; el 9,1% dos veces; el 18,2%, cinco.

Los usuarios que solo aparcan un día usan moto, no tienen coche, o su coche no cabe en la plaza.

Los usuarios que utilizan moto o no tienen coche no deben tenerse en cuenta. El usuario cuyo coche no cabe en su plaza se analizará a continuación.

Se entiende que los usuarios que aparcan dos y tres veces comparten plaza con normalidad.

7. **¿Cuántos días a la semana sueles aparcar fuera del recinto?**

El 45% de usuarios suele aparcar tres veces a la semana fuera; el 27,3 %, una vez; el 9,1%, dos veces; el 18,2%, cinco.

Los usuarios que aparcan una vez tienen un compañero sin coche, no tienen coche (su respuesta no es válida) o su compañero aparca en otra plaza alquilada por ambos fuera del recinto. Por tanto, de estos, solo interesa la persona que tiene plaza alquilada.

Se entiende que los usuarios que aparcan dos y tres veces fuera comparten plaza con normalidad.

8. **¿Tu plaza está libre algunos días porque tu/tu compañero no la usáis?**

   El 90% de usuarios considera que su plaza se queda libre con normalidad porque él o su compañero no la usan. Este es un dato muy importante.

9. **Del 1 al 5, ¿Cómo de importante ves una aplicación que te permita reservar las plazas que no están ocupadas los días que no puedes aparcar en tu plaza?**

El 81,8% de usuarios considera que la aplicación de reservas de aparcamiento es importante. El otro porcentaje usa moto o su compañero no tiene coche.

10. **Del 1 al 5, ¿Usarías la aplicación para liberar tu plaza y que otros puedan usarla?**

El 100% de usuarios liberaría su plaza para que otros puedan usarla.

11. **Del 1 al 5, ¿Cómo de necesario ves que la aplicación también te ayude a gestionar la plaza con tu compañero de plaza?**

El 63,6% de usuarios opina que no necesita que la aplicación le ayude a gestionar la plaza con su compañero, frente a los que marcan el 2, 3 y 5 como que sí.

Un usuario que marca el 5 no tiene coche, el otro sí.

12. ¿Qué te gustaría que tuviese esta aplicación?

La respuesta 12 es importante para sacar conclusiones de funcionalidades que no hayamos contemplado.

En general, los usuarios apuntan que es importante que la aplicación permita reservar y liberar la plaza (esto ya lo teníamos contemplado).

Hay un usuario que indica que su coche no cabe en su plaza. Se podría dar la posibilidad de indicar si la plaza admite coches grandes o no, o la gestión de propuesta de cambio de plaza por estos motivos.

Existe un usuario que indica que sería interesante poder entrar en el aparcamiento con el móvil. Esto no es posible por normas de la empresa.

Otro usuario indica que se haga extensible a otros edificios. Podría plantearse para opciones futuras. Habría que incluir el registro del edificio en la aplicación.

También proponen reservar aparcamiento para visitas. Esto no es posible por normas de la empresa.

13. ¿Tienes algo más que comentar...? Puedes hacerlo aquí:

Los comentarios aportados por los usuarios no toman importancia para el estudio.

Conclusiones:

1. Se anota el problema de los usuarios que no pueden aparcar en sus plazas por dificultades de espacio. Se intentará resolver permitiendo a los usuarios indicar si en su plaza caben coches grandes o no a través de un “check”. Se propone como opcional el intercambio de plaza si se dispusiese de tiempo.

2. Se anota la idea de compartirlo con otros edificios, aunque se dejará para acción a futuro por las necesidades de fechas del proyecto.

Descripción de usuarios

Estudio de usuarios

El usuario principal de la aplicación será el siguiente:

Usuario 1: Trabajador de la empresa con coche con plaza compartida.
El usuario principal de esta aplicación sería un trabajador de la empresa con coche y con plaza compartida asignada. Este usuario normalmente usa su plaza la mitad de la semana, un mes si u otro no, etcétera. La otra mitad la usa su compañero. Las necesidades de este usuario son:

- Poder reservar plazas los días que no puede aparcar.
- Poder liberar su plaza los días que nadie la use.
- Puede indicar si en su plaza caben o no coches grandes.

Existen además usuarios con otras características:

Usuario 2: Trabajador de la empresa con coche y con compañero sin coche:
Este usuario suele disfrutar todos los días de su plaza de garaje, por tanto, la necesidad de este usuario es solo la siguiente

- Poder liberar su plaza los días que nadie la use.
- Puede indicar si en su plaza caben o no coches grandes.

Usuario 3: Trabajador la una empresa con coche grande:
Este usuario no puede aparcar en su plaza porque no es apta para su coche. Sus necesidades son las siguientes:

- Poder liberar su plaza los días que nadie la use.
- Poder reservar plazas grandes.
- Puede indicar que su plaza no es apta para coches grandes.

Usuario: Trabajador de una empresa con moto:
Se puede asumir que es un Usuario 1 que usa menos su plaza.

Usuario: Trabajador de la empresa con coche con plaza compartida y con plaza alquilada fuera
Se puede asumir que es un Usuario 1 que usa más su plaza.

Conclusión

Se asumirá que todos los usuarios pueden ser Usuarios de tipo 1 que usan más o menos días su plaza (incluso ningún día en caso del trabajador que no puede aparcar por los problemas de espacio).

Por tanto, existirá solo un tipo de usuario común que tendrá las siguientes necesidades:

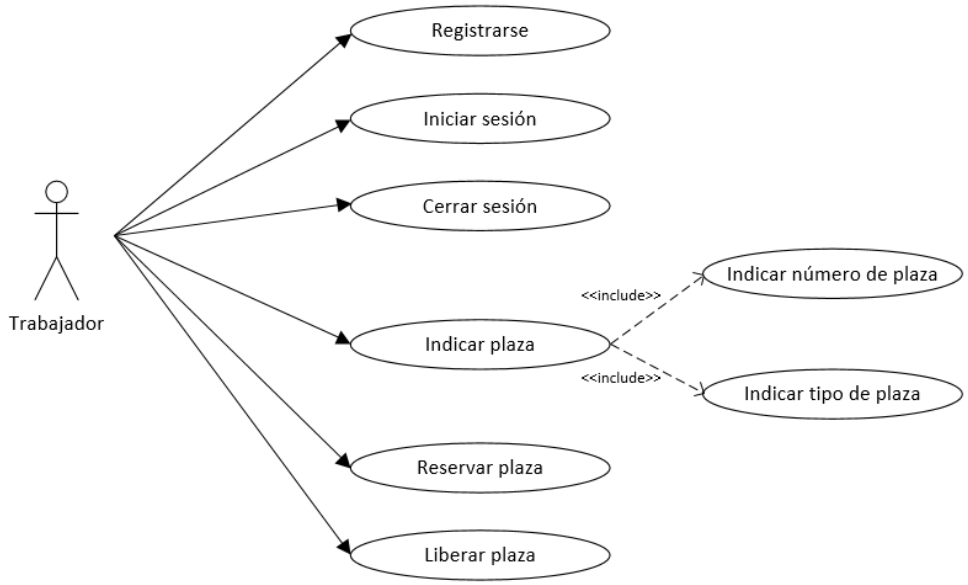
- Reservar plaza.
- Liberar plaza.
- Indicar si en su plaza caben coches grandes o no.

Además, tendrá las necesidades propias de una aplicación con usuarios:

- Registrarse con sus datos.
- Indicar su número de plaza.
- Iniciar sesión.
- Cerrar sesión.

Casos de uso

Se incluye a continuación el diagrama de casos de uso de la aplicación:



Y la definición de estos en las siguientes tablas:

Registrarse	
Nombre	Registrarse
Actores	Trabajador
Objetivos	Registrar al usuario en la aplicación
Precondiciones	
Postcondiciones	Usuario registrado
Escenario básico	El trabajador abre la aplicación y se registra

Iniciar sesión	
Nombre	Iniciar sesión
Actores	Trabajador
Objetivos	Iniciar sesión en la aplicación
Precondiciones	El usuario debe haberse registrado
Postcondiciones	El usuario habrá iniciado sesión en la aplicación
Escenario básico	1. El usuario se registra en la aplicación 2. El usuario inicia sesión

Cerrar sesión	
Nombre	Cerrar sesión
Actores	Trabajador
Objetivos	Cerrar sesión en la aplicación
Precondiciones	El usuario debe haber iniciado sesión
Postcondiciones	El usuario no tiene sesión en la aplicación
Escenario básico	1. El usuario inicia sesión en la aplicación 2. El usuario cierra sesión

Indicar plaza	
Nombre	Indicar plaza
Actores	Trabajador
Objetivos	Indicar el número de plaza y el tipo en la aplicación
Precondiciones	El usuario debe haber iniciado sesión
Postcondiciones	La plaza se asigna al usuario
Escenario básico	1. El usuario inicia sesión en la aplicación 2. El usuario indica el número de plaza y si es apta para coches grandes

Reservar Plaza	
Nombre	Reservar plaza
Actores	Trabajador
Objetivos	Reservar plaza
Precondiciones	El usuario debe haber iniciado sesión
Postcondiciones	Se asigna una plaza al usuario (la suya o la de otro usuario)
Escenario básico	1. El usuario inicia sesión en la aplicación 2. El usuario reserva una plaza

Liberar plaza	
Nombre	Liberar plaza
Actores	Trabajador
Objetivos	Reservar plaza
Precondiciones	El usuario debe haber iniciado sesión e indicado su plaza
Postcondiciones	Se libera la plaza asignada actualmente al usuario (la suya o la de otro usuario)
Escenario básico	1. El usuario inicia sesión en la aplicación 2. El usuario indica su plaza 3. El usuario libera su plaza

Listado de requisitos

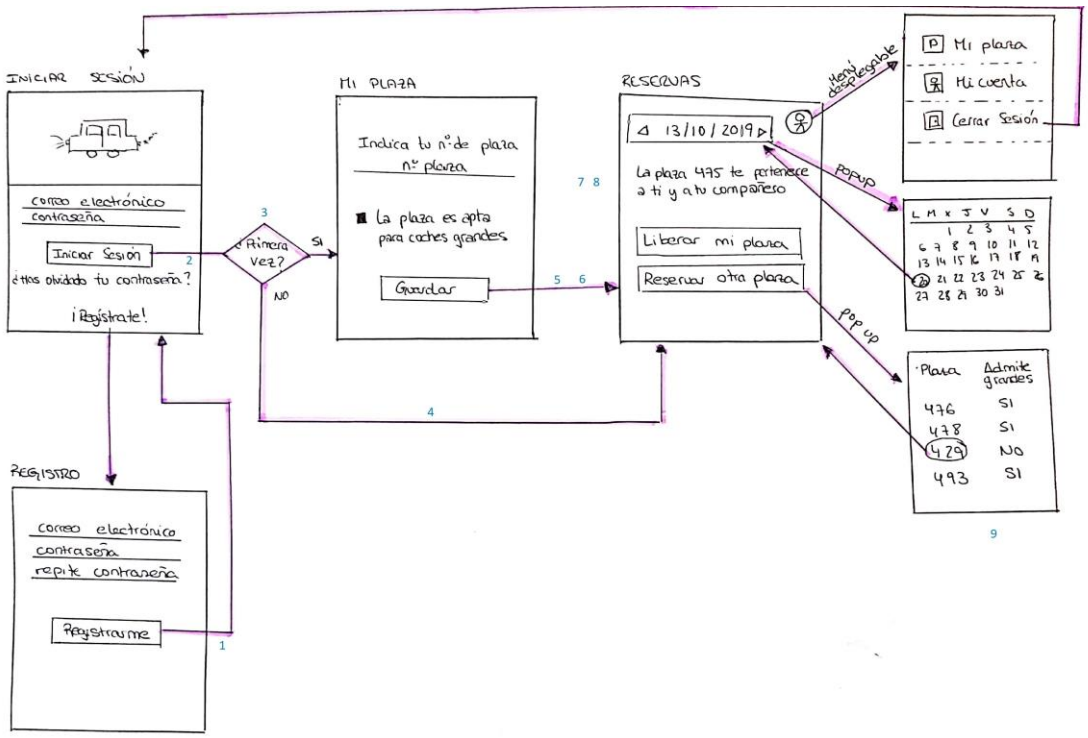
Se incluye la lista de requisitos del apartado *"Contexto"* con una descripción más a fondo y los requisitos encontrados durante el análisis:

Requisitos funcionales	
RF1	La aplicación permitirá a los usuarios **registrarse** con un usuario y contraseña.
RF2	La aplicación permitirá a los usuarios ya registrados **anteriormente iniciar sesión** con el usuario y contraseña utilizados.
RF3	La aplicación permitirá a los usuarios que hayan iniciado sesión incluir el **número de plaza asignada**
RF4	La aplicación permitirá a los usuarios que hayan iniciado sesión indicar si la plaza es **apta para coches grandes**
RF5	La aplicación permitirá a los usuarios que hayan iniciado sesión **reservar la plaza de aparcamiento de otro usuario** para un día específico.
RF6	La aplicación permitirá a los usuarios que hayan iniciado sesión **liberar la plaza de aparcamiento asignada** a un usuario y su compañero para un día específico
RF7	La aplicación permitirá a los usuarios que hayan iniciado sesión y hayan reservado una plaza que no es suya **liberar la plaza que no es suya**
RF8	La aplicación permitirá a los usuarios que hayan iniciado sesión y hayan liberado su plaza **reservar su plaza si está disponible**
RF9	La aplicación permitirá a los usuarios que hayan iniciado sesión **cerrar sesión**
Requisitos no funcionales	
RNF1	La aplicación debe ser una aplicación para móviles **Android**
RNF2	La aplicación debe ser compatible con **Android 8.0 o superior** (versión móvil corporativo)
RNF3	La aplicación debe **verse correctamente en todos los dispositivos Android**, por lo que debe tener un diseño que se adapte a todo tipo de pantallas.
RNF4	La aplicación debe ser probada en **un Samsung Galaxy J6 versión 8.0** o superior ya que es el móvil corporativo. Hay que estar seguro de que funciona correctamente en este modelo
RNF5	La aplicación debe ser **fácil e intuitiva**
RNF6	La aplicación debe ser **fácil de desarrollar y mantener**
Requisitos Software para el proyecto	
RS1	**Android Studio** para el desarrollo
RS2	**API + base de datos** (o Firebase) para consulta de datos.
RS3	**Publicación de servidor** (o Firebase) para que las aplicaciones puedan acceder a él.
Requisitos Hardware para el proyecto:	
RH1	Ordenador con posibilidad de instalar **Android Studio** para el desarrollo
RH2	Teléfono móvil **Android 8.0 Galaxy J6** para pruebas

3. Arquitectura

Navegación

A continuación, se presenta el esquema de navegación de la aplicación:



La aplicación cuenta con tres pantallas y varios “Pop-Up” que forman parte de la pantalla de reservas. Las fechas en color morado indican hacia dónde se puede navegar desde cada pantalla o botón. Se explica por separado cada pantalla:

Pantalla “Inicio de sesión”:

El usuario podrá iniciar sesión en ella.
Se plantea como opcional la implementación de “¿Has olvidado tu contraseña?”. Se realizará si se dispone de tiempo necesario para realizarlo.

Pantalla “Registro”:

El usuario podrá registrarse en la aplicación con su usuario y contraseña.

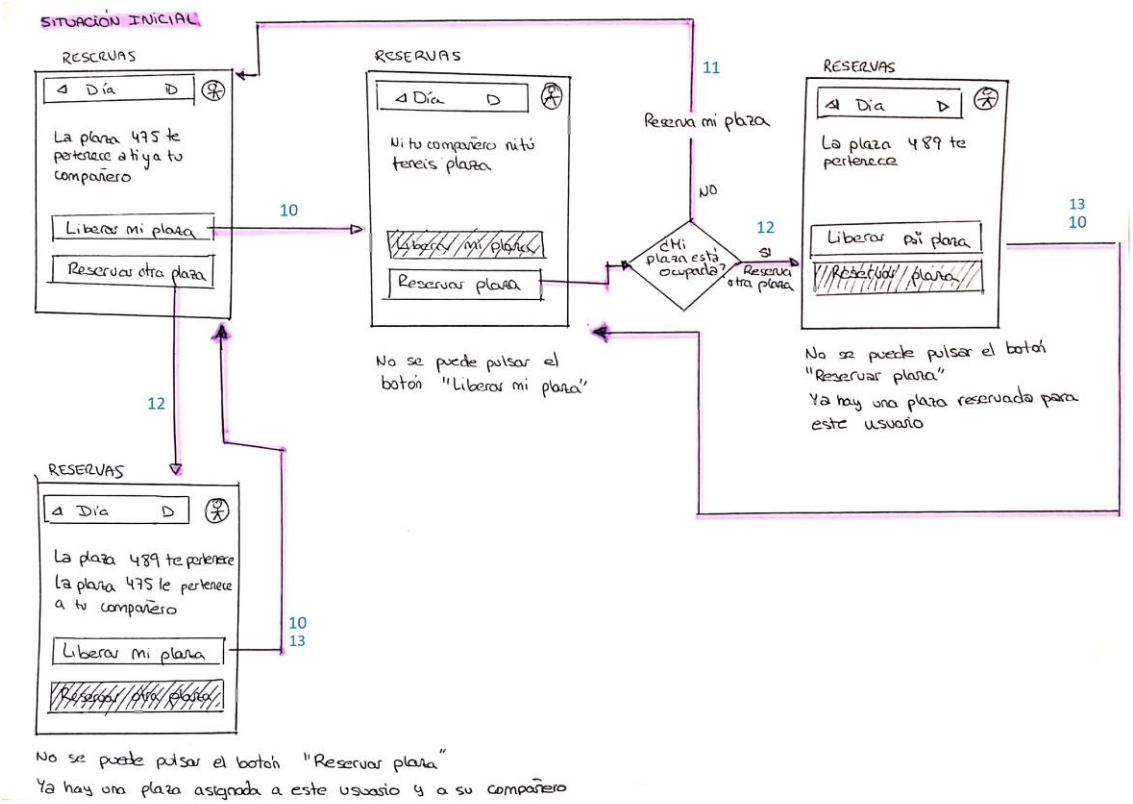
Pantalla “Reservas”:

Esta pantalla permitirá al usuario reservar o liberar la plaza
Dentro de esta pantalla, se incluirán los siguientes elementos:

- Pop Up “Mi plaza”
  Se mostrará únicamente si el usuario no ha indicado ya su plaza anteriormente.
- Menú desplegable:
  Desde él, el usuario podrá cerrar sesión.
  Se plantea como opcional la implementación de “Mi plaza” para modificar la plaza del usuario y de “Mi cuenta” para modificar la cuenta del usuario.
- Pop Up “Calendario”
  Se mostrará un calendario para la elección del día.
- Pop Up “Plazas”
  Mostrará las plazas disponibles para ese día.

Se puede observar todas las acciones de los *Casos de uso* elegidos anteriormente: la pantalla Inicio de Sesión se corresponde con el caso de uso Iniciar Sesión; la pantalla Registro con el caso de uso Registro, la pantalla Reservas contiene los casos de uso Liberar plaza y Reservar Plaza; La pantalla Mi plaza con el caso de uso Indicar plaza; el menú desplegable contiene un botón de Cerrar sesión correspondiente con el caso de uso Cerrar Sesión. Por tanto, los botones de cada una de las pantallas implementan cada caso de uso.

La funcionalidad de la aplicación está centrada en la pantalla “Reservas”. A continuación, se muestra un esquema de cómo irá cambiando esta pantalla según las opciones que vaya indicando el usuario y su estado:



Reservas 1:

Es la situación inicial del usuario para cualquier día. Al usuario y a su compañero le pertenecen la plaza que le han asignado por defecto.
Tendrán dos opciones indicadas con dos botones: Liberar mi plaza, que le llevaría al estado "Reservas 2"; o Reservar otra plaza, por lo que iría al estado "Reservas 4", tal y cómo indican las flechas de navegación en morado.

Reservas 2

El usuario ha liberado su plaza y no tiene otra reservada, por tanto, ni el, ni su compañero tienen plaza. Solo puede reservar plaza.
Automáticamente, la aplicación comprobará si su plaza sigue libre y le asignará su plaza, volviendo al estado "Reservas 1".
Si su plaza ya está ocupada le permitirá elegir otra mediante el "Pop-Up" e ir al estado "Reservas 3".

Reservas 3:

El usuario dispone de una plaza de otra persona ya que liberó la suya y, en el momento de la reserva, no estaba disponible.
No puede reservar otra plaza, solo podría liberarla.

Reservas 4:

El usuario tiene dos plazas: la que le asignó la empresa que usará su compañero y la suya.
La aplicación no realizará la comprobación de si el compañero ha reservado otra plaza o no, se deja cómo opcional si se dispone de tiempo.

Los números que aparecen en el esquema en azul se corresponden con los números de las llamadas del API explicadas en el apartado *0 API.*

Prototipo

En el *ANEXO 4. Prototipo* se muestra el prototipo del diseño de la aplicación realizado con la herramienta de prototipado JustMine.
En este prototipo se incluyen los colores, medidas, tipos de fuente, imágenes y todos los aspectos de diseño que se incluirán en la aplicación.

Estructuras de datos

Modelo de base de datos

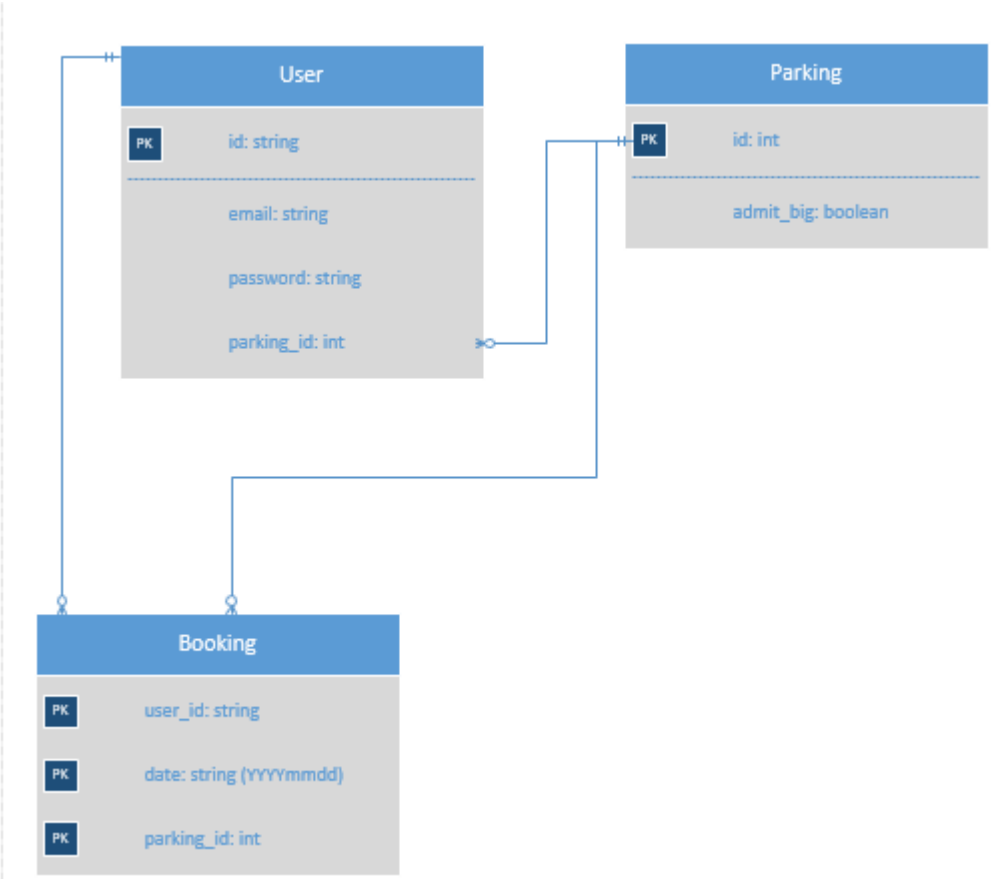
Se analiza la aplicación y se concluye que son necesarias tres entidades para su correcto funcionamiento:

- Usuarios
  Los usuarios estarán identificados por un id de usuario. Cada usuario podrá tener un email y una plaza de aparcamiento asignada.
- Plazas de aparcamiento
  Las plazas estarán identificadas por un id y podrán almacenar si una plaza admite coches grandes o no.
- Reservas
  Las reservas están identificadas por un id de usuario, una fecha y una plaza.

Las relaciones entre las entidades son las siguientes:

- Un usuario puede tener una plaza de aparcamiento asignada.
  Una plaza puede pertenecer a dos usuarios.
  La relación entre la tabla User y Parking es 1:N
- Un usuario puede hacer numerosas reservas de parking según el día.
  Un parking puede estar reservado por numerosos usuarios según el día.
  La relación entre las tablas User y Parking es N:M, por lo que creamos una entidad Booking para esta relación.

Por tanto, se piensa en un esquema de base de datos siguiente para la aplicación:



Cómo el estado inicial de los usuarios es que tengan asignada su plaza, se supondrá que la tabla Booking está vacía en la situación inicial. Nadie ha reservado ninguna plaza aunque le corresponde la suya.

Se utilizará la tabla Booking para indicar cuándo un usuario libera su plaza. Se incluirá un nuevo registro en la tabla cuyo user_id sea null. Por ejemplo, si el usuario_1 libera su plaza 35 el día 23/10/2019, añadiremos el registro:

User_id	Date	Parking_id
null	20191024	35

Si este usuario vuelve a reservar su plaza, se eliminará este registro, volviendo a la situación inicial.
Si, por el contrario, es otro usuario (usuario_2) el que reserva la plaza, el registro aparecerá de la siguiente forma:

User_id	Date	Parking_id
Usuario_2	20191024	35

Por esta razón se ha incluido que parking_id también sea clave primaria de la tabla booking. Puede haber muchas reservas con usuario null y la misma fecha. Se necesita incluir el número de parking para que las claves primarias no sean iguales.

Elección de tecnología de base del servidor

El modelo anterior sirve para entender la estructura de la base de datos, ver qué entidades dependen de otras, cuáles son las claves primarias que deben ser únicas en cada entidad y los tipos de datos de cada dato.

Sin embargo, se utilizará Realtime Database de Firebase para crear la base de datos y API de la aplicación. Realtime Database de Firebase utiliza un json para el almacenamiento de datos. La consulta y modificación de estos se hace a través de la navegación por el mismo. Es imposible plasmar el modelo anterior exactamente igual en un json, por lo que se realizará una implementación lo más próxima posible.

Se decide utilizar esta tecnología frente a otras tecnologías cómo podrían ser MySQL, MariaDB, MongoDB, u otras, debido al escaso tiempo de implementación del proyecto. Estas bases de datos necesitarán un servidor publicado en internet para que los usuarios de las aplicaciones puedan acceder a él. Debido al desconocimiento de estas tecnologías, resulta imposible realizar su implantación en el periodo que debe durar el trabajo de fin de máster. Se propone como solución a futuro ya que estas tecnologías son más fáciles de mantener y simplifican la consulta y modificación de sus datos.

Modelo de base de datos del servidor

Firebase ofrece gestión de usuarios mediante correo electrónico y contraseña, por lo que podemos olvidarnos de almacenar el correo y la contraseña en la tabla “User”. Este registro de usuarios devuelve un ID, que si que utilizaremos en nuestra tabla en el json.

Como Firebase no tiene control de claves únicas o primarias, se decide usar la concatenación de los atributos que componen la clave primaria de cada tabla como clave del json de cada objeto. Por ejemplo, en la tabla user, usaremos el valor del atributo user_id como clave del json. De esta forma, evitamos repetidos y agilizamos las búsquedas accediendo al objeto de la siguiente forma https://<url_firebase>/user/<user_id>.

En el caso de booking, se tienen dos claves: una es la fecha para agilizar las consultas en un determinado día y otra es la concatenación de las otras dos claves.

Firebase permite consultas por atributo, por ejemplo, podemos obtener todos los usuarios cuyo parking_id sea 245.

De esta forma, se obtiene el siguiente json en el que los valores entre “<>” son los valores variables:

```
{
 "user" : {
    <id_user>:{
        "parking_id" : <parking_id>
    }
  }
  "parking" : {
    <parking_id>:{
        "admit_big" : <admit_big>
    }
  },
  "booking" : {
  <date>:{
    <user_id_parking_id>:
        {
          "parking_id" : <numero_plaza>,
          "admit_big":<valor>,
          "user_id" : <user_id>
        },
    }
  }
}
```

En la tabla booking se ha añadido el atributo admit_big para agilizar la consulta en el “Pop-up” de plazas disponibles. Esto es debido a que se necesita obtener todas las plazas liberadas y si admiten grandes o no para cierto día, es decir, todos los registros de la tabla booking cuyo user_id=null y date=<valor>. Incluyendo el atributo admit_big nos ahorramos hacer una llamada a la tabla parking por cada plaza libre.

Estructuras de datos de la aplicación

La aplicación solo funcionará “on-line”, por lo que no incluiremos en ella base de datos. Realizaremos las consultas al API y plasmaremos el resultado directamente en la vista.
En el apartado *0 Modelo* podemos encontrar la estructura y los objetos que usará la aplicación.

API

A continuación, se explica la tecnología y las decisiones tomadas en el diseño del API:

Tecnología elegida

La tecnología elegida para el API es Firebase. Se elige esta tecnología frente a un servidor Python o php, entre otros, y una base de datos con su respectivo servidor, por limitaciones de tiempo y desconocimiento de tecnologías de servidor.

Uso en la aplicación

Firebase dispone de un SDK que permite a las aplicaciones Android acceder a su API. Este SDK simplifica la implementación de peticiones, por lo que accederemos a los datos a través del él.

El SDK dispone de métodos para la gestión de usuarios y el acceso a la base de datos, entre los que se encuentra la lectura, escritura, modificación o consultas:

Lectura de objetos de la base de datos:

Para la lectura de datos, el SDK dispone de un "listener" que notifica cada vez que la base de datos ha cambiado algún valor del json. En algunas llamadas resulta útil saber cuándo se modifican los objetos para no tener que estar continuamente realizando llamadas, pero en la mayoría de ellas no es necesario, por lo que, siempre que no sea necesario, se eliminará el listener y se obtendrá el valor del elemento una sola vez cómo se indica a continuación:

```
//Obtener datos
myRef.child("user").child("datos")
        .addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                    ...codigo
            }

            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {

            }
        });
```

Más adelante, en la explicación de cada una de las llamadas, se indicará cuando la aplicación deberá mantenerse escuchando a que los datos cambien.

Cómo se puede observar, cada "child" es una clave del json.

Modificación de los datos:

Debemos hacerlo de la siguiente forma:

```
//Modificar datos
myRef.child("user").child("datos").setValue(234)
        .addOnSuccessListener(new OnSuccessListener<Void>() {
            @Override
            public void onSuccess(Void aVoid) {
                ...codigo
            }
        })
        .addOnFailureListener(new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
                ...codigo
            }
        });
```

Firebase permite modificar tanto los hijos del json como el valor de los atributos. Es decir, si quisiéramos modificar un usuario, podríamos realizar una única llamada para incluir el usuario junto con su plaza de aparcamiento:

```
"user_id" : {
  "parking_id" : 248
}
```

Se usará esta posibilidad para reducir las llamadas al API al realizar la modificación de datos.

Eliminación de los datos:

Se realizará de la siguiente forma:

```
myRef.child("user").child("datos").removeValue().addOnSuccessListener(new OnSuccessListener<Void>() {
    @Override
    public void onSuccess(Void aVoid) {

    }
});
```

Consultas a la base de datos:

Se pueden realizar consultas a la base de datos, por ejemplo, devuelve todos los usuarios cuyo user_id=1234

```
myRef.child("booking").orderByChild("user_id").equalTo("1234")
        .addListenerForSingleValueEvent(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
    }

    @Override
    public void onCancelled(@NonNull DatabaseError databaseError) {

    }
});
```

Interfaz Firebase / Aplicación

Con todas las conclusiones anteriores sobre esta tecnología, se obtiene la siguiente interfaz entre el API (o SDK) y aplicación:

1. Registro	
Descripción	El usuario se registrará en la aplicación
Método SDK	createUserWithEmailAndPassword
Parámetros de entrada	- Correo de usuario - Contraseña
Parámetros de salida	- Usuario (identificador, correo)

2. Login	
Descripción	El usuario inicia sesión
Método SDK	signInWithEmailAndPassword
Parámetros de entrada	- Correo de usuario - Contraseña
Parámetros de salida	- Usuario (identificador, correo)

3. Obtener plaza de un usuario	
Descripción	Se obtiene la plaza de un usuario
Método SDK	Database.child(“user”).child(<id_usuario>).child(<parking_id>)
Parámetros de entrada	- Identificador de usuario
Parámetros de salida	- Plaza o Null

4. Obtener información de la plaza	
Descripción	Se obtiene la información de una plaza: si admite grandes o no
Método SDK	Database.child(“parking”).child(<id_plaza>).child(“admit_big”)
Parámetros de entrada	- Identificador de la plaza
Parámetros de salida	- Admit big es verdadero o falso

5. Guardar información de la plaza de un usuario	
Descripción	Se almacena la información de la plaza de un usuario
Método SDK	Database.child(“user”) .child(<id_usuario>).child(<parking_id >).setValue(<id_plaza>)
Parámetros de entrada	- Identificador de usuario - Identificador de plaza
Parámetros de salida	

6. Guardar información de la plaza	
Descripción	Se almacena la información de la plaza: si admite grandes o no
Método SDK	Database.child(“parking”).child(<id_plaza >).child(“admit_big”).setValue(<admit_big>)
Parámetros de entrada	- Identificador de plaza - Admit_big: true o false
Parámetros de salida	

7. Saber si la plaza de un usuario ha sido liberada o reservada por otro	
Descripción	Se obtiene si la plaza de un usuario ha sido liberada o no para un día determinado y si alguien ya la ha reservado. En esta llamada no se eliminará el listener, ya que así podremos saber si cambia el estado en el que se encuentra el usuario durante la ejecución.
Método SDK	Database.child(“booking”).child(<fecha>).orderBy(“parking_id”).equalTo(<parking>)
Parámetros de entrada	- Día - Identificador de la plaza
Parámetros de salida	- Devuelve null si la plaza no fue liberada - Devuelve el objeto booking si la plaza fue liberada - Si user_id=null, la plaza no ha sido reservada por otro usuario. Si tiene valor, la plaza ha sido reservada por otro usuario.

8. Obtener información de las reservas de un usuario para un determinado día	
Descripción	Se obtiene la plaza reservada para un usuario un determinado día (si tiene). En esta llamada no se eliminará el listener, ya que así podremos saber si cambia el estado en el que se encuentra el usuario durante la ejecución.
Método SDK	Database.child(“booking”).child(<fecha>).orderBy(“user”).equalTo(<user>)
Parámetros de entrada	- Día - Identificador del usaurio
Parámetros de salida	- Devuelve las reservas de un usuario para un determinado día

9. Obtener plazas disponibles	
Descripción	Se obtienen las plazas disponibles un determinado día y la información de si admiten grandes o no
Método SDK	Database.child(“booking”).child(<fecha>).orderby(“user”).equalTo(“null)
Parámetros de entrada	- Día
Parámetros de salida	- Listado de plazas libres un determinado día. Se puede obtener si admite grandes o no.

10. Liberar una plaza un determinado día	
Descripción	Se libera una plaza para un determinado día
Método SDK	Database.child(“booking”).child (<dia>).child (<null_parking_id>).setValue(<booking>)
Parámetros de entrada	- Día - Identificador de booking: null_parking_id - Booking: parking_id, admit_big, user_id
Parámetros de salida	

11. Eliminar la liberación de una plaza cierto día (Reservar mi plaza)	
Descripción	Elimina la liberación de una plaza un determinado día
Método SDK	Database.child(“booking”).child(<dia>)child(<null_parking_id>).removeValue()
Parámetros de entrada	- Día - Identificador de la plaza
Parámetros de salida	

12. Reservar una plaza un determinado día	
Descripción	Reservar una plaza un día determinado.
Método SDK	Database.child(“booking”).child(<dia>).child(<user_parking_id>).setValue(<booking>)
Parámetros de entrada	- Día - Concatenación de user y parking - Booking con todos los parámetros
Parámetros de salida	

13. Eliminar reserva un determinado día	
Descripción	Elimina la reserva de la base de datos para un determinado día
Método SDK	Database.child(“booking”).child(<dia>).child(<user_parking_id>).removeValue ()
Parámetros de entrada	- Día - Concatenación de user y parking
Parámetros de salida	

Modelo

Se presenta un modelo MVC (Modelo-Vista-Controlador) ya que se desea que la aplicación tenga separadas las tres funcionalidades:

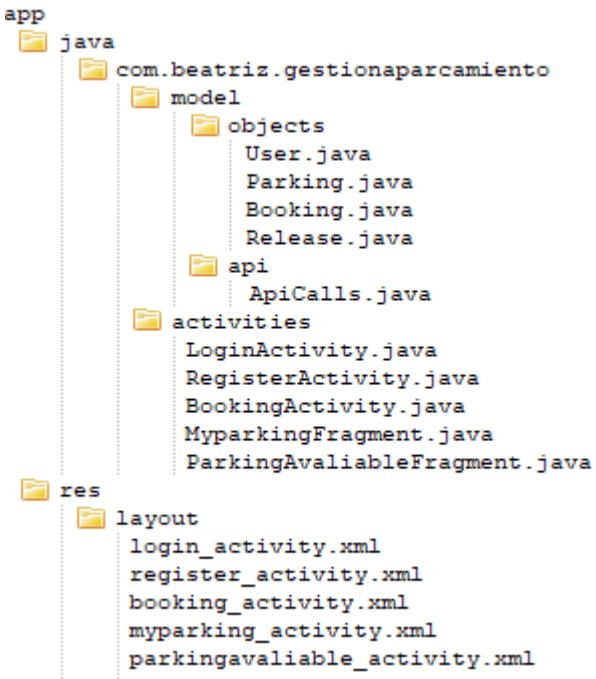
1. Por una parte, tendremos el modelo que representa la capa de datos. En nuestro caso, la capa de datos estará compuesta por los objetos que utilizan la aplicación y la capa que gestiona los datos, es decir, la que se encarga de realizar las llamadas al API, recuperar la información y tratarla.

2. Por otra parte, estará la vista. En Android, hay una clara diferenciación de la vista ya que es necesario separarla en los xml que se compone de las pantallas, botones y todo lo relacionado con la interfaz de usuario.

3. Por último, el controlador, que se encarga de la lógica en sí de la aplicación. Recogerá los eventos de la vista, se conectará con el modelo y realizará la funcionalidad que sea necesaria para cada evento. El controlador lo formarán las activities y fragment de la aplicación

Se elige este modelo frente a otros como puede ser MMVC ya que es una aplicación con sencilla y el desarrollador es la misma persona que realiza el diseño, no se requiere una separación total entre vista y contralador a través de un Binding. Por tanto, se ve innecesario aplicar este modelo que complicaría el desarrollo. Por esta misma razón se descarta un modelo MVP.

Durante el desarrollo, se puede ver el modelo MVC plasmado en la propia estructura de la aplicación, que tendrá un estilo similar al siguiente:



4. Implementación

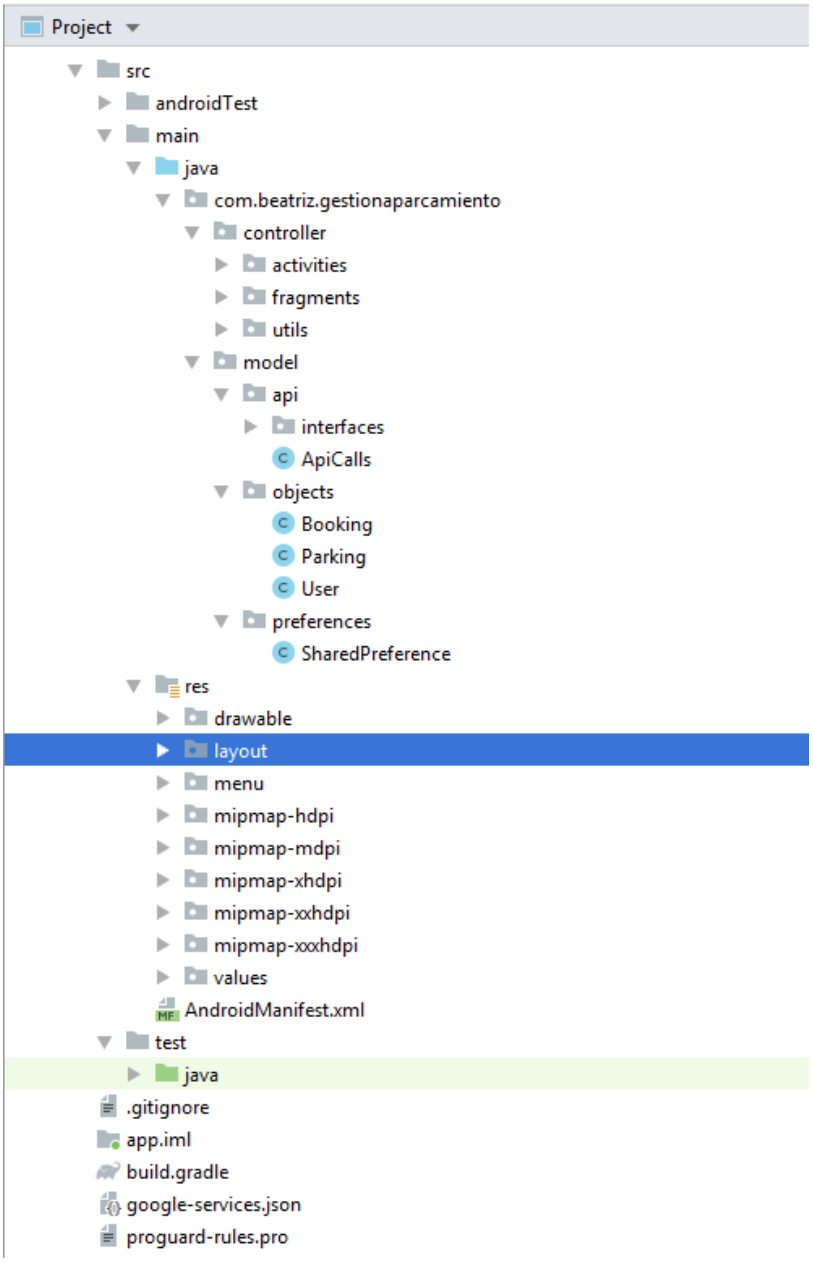
En este apartado se justifican las decisiones de implementación y se documentan las pruebas realizadas.

Decisiones e implementación de la aplicación

A continuación, se explican los detalles, decisiones y justificación de la implementación.
Se explicará la estructura de los directorios y, después, los detalles de cada apartado.

Estructura de directorios

En el proyecto Android, se puede observar la siguiente estructura:



Si comenzamos de arriba abajo, el primer elemento importante que nos encontramos es la carpeta *app*. Dentro de ella, nos encontramos la carpeta *src*, que es dónde se encuentran las clases y archivos de la aplicación.

En el interior de java, existen dos paquetes hijos:

- **Model**: contiene las clases necesarias para la implementación del modelo de la aplicación. Está compuesta a su vez por tres paquetes:

    - **Api**: lo forma un archivo llamado APICalls con todas las llamadas al servidor y la traducción de objetos, además de un paquete interfaces con todas las interfaces utilizadas para la traducción.

    - **Objects**: comprende los objetos que se van a utilizar en la aplicación

    - **Preferences**: contiene una clase que gestiona el almacenamiento de datos en preferencias.

- **Controller**: está formado por las clases que constituyen el controlador de aplicación, es decir, la lógica, las clases que comunican el modelo con la vista y que indican la funcionalidad llevada a cabo por la aplicación.
  Estas clases inicializarán las vistas correspondientes, la lógica de los botones, y su comportamiento.
  Dentro de este paquete, podemos diferencias tres paquetes:

    - **Activities**: forman la lógica de las pantallas de la aplicación

    - **Fragment**: forman la lógica de algunos fragmentos de las pantallas de la aplicación

    - **Utils**: se compone de una clase con métodos que son comunes a toda la aplicación y pueden ser usadas desde diferentes partes del código, como el tratado de fechas o de correo electrónico.

Las vistas, imágenes y todo lo necesario para la interfaz de la aplicación se encuentra en res.

- **Drawable**: está formada por las imágenes que usará la aplicación

- **Layout**: en este paquete se encuentran la interfaz de las pantallas y componentes de la aplicación

- **Menú**: contiene los elementos del menú de la aplicación.

- **Mipmap**: en él, se almacenan los iconos de la aplicación. Como se puede observar, existen varias carpetas con este nombre y un código al final. Este código final indica el tipo de pantalla del dispositivo en el que se ejecutará la aplicación. Se incluyen diferentes iconos para que el contenido pueda visualizarse bien en todos los dispositivos.

- **Values**: contiene los textos, colores, dimensiones y estilos que se utilizarán en la aplicación.

Otro elemento importante es el archivo **AndroidManifest.xml**, en el que se define el icono de la aplicación, el nombre, el tema, las pantallas, los permisos si fuesen necesarios y otros aspectos importantes de la aplicación.

Igual de importante es el archivo **build.gradle**, que contiene las versiones de Android de la aplicación, la versión del código y las dependencias del proyecto, entre otros.

El archivo **Google-services.json** es el archivo de configuración necesario para que la aplicación pueda acceder a Firebase.

Modelo

Objetos:

Cómo hemos comentado en el apartado anterior, se crea un paquete denominado object que almacena los objetos de la aplicación, con sus propiedades, constructores y atributos. Estos objetos son los mismos que utilizamos en la base de datos: User, Parking y Booking.

Los tipos de datos utilizados para cada atributo del modelo son los indicados en el diseño de la base de datos, es decir, los mismos que también se almacenarán en Firebase. Una peculiaridad es que para las fechas se utiliza un String con el formato YYYYmmdd, dónde YYYY es el año, mm el mes y dd el día. Esto es debido a que, en firebase, las claves de los objetos del diccionario deben ser String. Se ha decidido usar también en la aplicación porque no se ha considerado necesaria la gestión de obetos tipo Date cómo tal, ya que no usamos horas que pueden tener desplazamiento. Utilizar String facilita la implementación.

Se valora usar Int o Integer para los números de las plazas de aparcamiento. Se toma la decisión de usar Integer ya que admite nulos. De esta forma, podremos almacenar un id=null cuando, por ejemplo, el usuario no tenga plaza de aparcamiento asignada. Podríamos haber utilizado int con un número de plaza negativa para almacenar los null ya que no existen plazas negativas, pero es más claro para el desarrollador usar null.

Firebase utiliza *json* para el almacén de los objetos y el SDK devuelve los datos como un objeto de tipo *HashMap*. Se podrían haber implementados los objetos de esta forma, pero es más difícil aplicarles lógica. Lo ideal es que la aplicación sea independiente del servicio utilizado y la devolución de datos del mismo. Por ello, se crean los objetos con sus respectivos tipos y se realiza la traducción a los mismos en las llamadas al API.

Llamadas al API

Dentro del paquete API, se crea la llamada al API de Firebase y la traducción de objetos de Firebase a nuestros objetos de la aplicación.

Estas llamadas podrían haberse incluido directamente en la Activity que gestiona la pantalla. Por ejemplo, se podría haber incluido la llamada y traducción de login en LoginActivity.
Sin embargo, se decide crear una clase aparte que realice esto, aislando toda la parte del servidor y datos del controlador. De esta forma, tenemos una aplicación más fácil de mantener y desarrollar.

El paquete interfaces contiene las interfaces de respuestas de la clase APICalls. Una por cada llamada. En todas las interfaces se devuelven uno o dos valores: el objeto pedido en la llamada y un objeto de tipo String llamado error. En caso de que la llamada se ejecute correctamente,

esta devolverá el objeto necesario y error=null. En caso de que haya un error al solicitar el dato, esta devolverá el objeto=null y error="<texto del error>". Si la llamada solo consistiera en guardar un dato en Firebase, la interfaz solo devolverá error=null, sin el objeto, o error="<texto>"

Los errores que devuelve Firebase están en inglés y, a veces, son poco descriptivos. Por esta razón, dentro de APICalls, se incluye el método *translateExceptionIfIsPosible*, que traduce los errores devueltos por Firebase al idioma de la aplicación.
Se ha intentado buscar en el SDK de Firebase algún método que cambie el idioma de los mensajes, pero no se ha encontrado.

APICalls utiliza un patrón *Singleton*, para que toda la aplicación pueda acceder a la instancia ya creada.

Preferences

Para no estar realizando continuamente las mismas llamadas al *API* y dar al usuario mejor experiencia, se almacenas los datos del usuario y de su plaza de aparcamiento en preferencias de la aplicación. Igualmente, se crea un patrón *Singleton* para que toda la aplicación acceda a la misma instancia.

Controllers

Se utiliza una actividad para cada pantalla del diseño de la aplicación: LoginActivity para la pantalla de login, RegisterActivity para la pantalla de registro y BookingActivity para la pantalla de reservas.

Se utiliza un *Fragment* para el *Pop-Up* de la elección de plaza de aparcamiento del usuario y otro para el *Pop-Up* de la reserva de las plazas. Se podría haber incluido también el código dentro de la actividad BookingActivity ya que la lógica se incluye dentro de la pantalla de reservas, pero se decide hacer en *Fragment* separados para simplificar el código y que sea más fácil su lectura e implementación.

LoginActivity

La encargada de mostrar el *layout* del login y de gestionar su lógica.

Tanto en esta cómo en todas las demás actividades, en el método *OnCreate* se inicializan todos los elementos de la interfaz, es decir, en el momento en el que se crea la actividad. Los textos, botones, etcétera.

Lo primero que se realiza en esta actividad es la comprobación del usuario en preferencias. Si el usuario ya ha iniciado sesión una vez, sus datos son almacenados en preferencias, por lo que no le volvemos a pedir que inicie sesión. Esto es así en muchas aplicaciones, y se ha tomado esta idea para mejorar la experiencia del usuario.

Una vez ejecutado el método *onCreate*, se queda

esperando a recibir los eventos del usuario: bien sea pulsar algún botón, introducir texto o pulsar el botón de atrás para salir de la aplicación.

Si el usuario presiona el botón de Iniciar Sesión, antes de realizar la llamada, se comprueba que los datos sean correctos: si los campos están vacíos, si el correo electrónico es válido, si la contraseña tiene la longitud correcta… y se muestra el mensaje correspondiente en el cuadro de texto que corresponda, cómo se ve en la imagen anterior ("Correo electrónico inválido").
Si, finalmente, todos los campos son correctos, se realiza la llamada al API. Si el usuario está registrado, se guardarán sus datos en preferencias para que no sea necesario adquirirlos más veces del servidor y se irá a la pantalla de Reservas. En caso contrario, se muestra un mensaje al usuario para indicar que el usuario no es correcto.

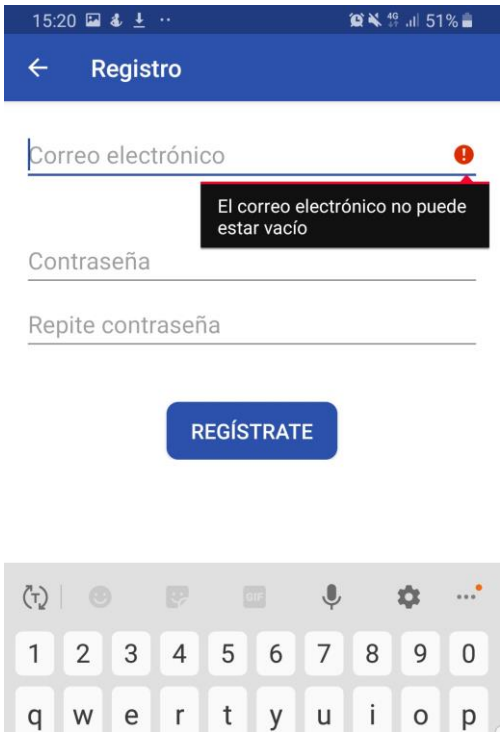
El botón ¿Has olvidado tu contraseña? se deja cómo implementación a futuro.
El botón de Registro lleva a la pantalla de registro.

En esta y en todas las actividades y *fragments*, se muestra un *progressBar*, o icono de progreso, cada vez que se está realizando una llamada al API. De esta forma, el usuario tiene conciencia de que la aplicación está trabajando. Cuando la llamada termina, desaparece.
Esto se consigue mostrando un layout por encima del layout principal en la vista. En la actividad, se indica que sea visible o invisible según se realiza la llamada o no.

La llamada de Login devuelve un error del SDK de Firebase cuándo no tiene conexión. Este mensaje se devuelve al usuario para que sea consciente de que la aplicación no es capaz de recuperar los datos.

RegisterActivity

Esta actividad contiene la lógica de la pantalla de Registro.



Al igual que en la pantalla de login, el método *onCreate* inicializa la vista y los botones, después, se mantiene a la espera de los eventos del usuario.

Al pulsar el botón de Registro, se comprueban los campos introducidos por el usuario, se realiza la llamada al API si se cree conveniente mostrando el icono de progreso y se muestra el resultado.

El SDK de Firebase también proporciona un mensaje de error cuándo no se obtiene conexión. Este mensaje se traduce en APICalls y se muestra al usuario en esta actividad.

La única peculiaridad de esta *Activity* es que, al completar el registro, se muestra un mensaje al usuario con un botón de aceptar. En el momento que se pulsa el botón de aceptar, la aplicación redirige a la pantalla de Reservas. En un principio, se implementó que redirigiese a la pantalla de Login por petición de un usuario, pero se modifica para que realice el Login automáticamente por recomendación del profesor en la fase de pruebas.

BookingActivity

Esta actividad contiene la lógica principal de la aplicación, ya que gestiona la parte de reservas, el Pop-Up que introduce la plaza del usuario, el Pop-Up de la reserva de plazas y el menú superior de la derecha.

Para entender mejor la implementación de esta actividad, se incluye el fragmento y la explicación del método *onCreate*, el método ejecutado cuando se crea la Actividad:

```
@Override
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_booking);
    Toolbar toolbar = findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    //Inicializa la vista de booking
    BT_calendar  = findViewById(R.id.BT_calendar);
    TV_info = findViewById(R.id.TV_info);
    BT_release = findViewById(R.id.BT_release);
    BT_booking = findViewById(R.id.BT_booking);
    date = Utils.getDateToday();
    initBTCalendar();
    RL_progress_bar = findViewById(R.id.RL_progress_bar);
    boolean load_booking = true;

    //Inicializa el fragment my parking si es necesario
    LL_popUp = findViewById(R.id.layout_popUp);
    user = (User) getIntent().getSerializableExtra( name: "user");
    if(user.getParking_id()==null){
        load_booking=false;
        getUserParkingFromServer();
    }else{
        parking=SharedPreference.getInstance(getApplicationContext()).getParkingPreferences();
    }

    //Carga datos
    if(load_booking){
        loadData();
    }

}
```

En primer lugar, tenemos una serie de inicializaciones de elementos de la vista y variables.

En segundo lugar, tenemos un comentario *"Inicializa el fragment my parking si es necesario*". En este fragmento de código, se obtiene el usuario de la actividad anterior (del login) a través del *Intent* (la forma utilizada en Android para pasar parámetros entre actividades).
Se comprueba si este usuario tiene un parking asignado a través de *user.getParking_id()==null.* En caso de que no lo tenga, se realizará la llamada al servidor. Si lo tiene, se obtendrá su valor y propiedades de preferencias y se cargarán los datos correspondientes a las reservas (*loadData()*).

getParkingFromServer intenta obtener la plaza de aparcamiento de un usuario en el servidor. Es posible que aún no la haya registrado, en este caso, este método llamará a otro método denominado *showMyParkingFragment*, que cargará el *fragment MyParkingFragment* y mostrará el *Pop-Up MyParking.*

La lógica de este *fragment* es independiente de la actividad. Al finalizar las acciones del usuario, llamará a *saveUserWithParkingAndCloseFragment* para almacenar la plaza de aparcamiento en preferencias y hacer desaparecer el fragment anterior.

Se ha hecho una separación en el código por comentarios para encontrar rápidamente los métodos relacionados con la obtención de la plaza y la funcionalidad de mostrar y cerrar el *fragment*. Podemos encontrar estos métodos debajo de los comentarios *"LOAD PARKING METHODS"*. Serían los siguientes:

- *getUserParkingFromServer*
- *getParkingFromServer*
- *showMyParkingFragment*
- *saveUserWithParkingAndCloseFragment.*

La aplicación no permite salir de *MyParkingFragment* sin haber elegido la plaza del usuario, ya que todo usuario que tiene acceso al aparcamiento tiene asignada una plaza. Si no, no puede acceder al recinto. No tendría sentido el uso de la aplicación a un usuario sin plaza asignada. Por esta razón, no se incluye la opción de cerrar el Pop-Up sin guardar.

Una vez que se recoge la plaza y se almacena, la actividad ya muestra al usuario el *layout* propio de esta pantalla:

Al igual que con los métodos del *fragment*, se ha intentado diferencias los métodos propios de las reservas incluyéndolos debajo de los comentarios *"BOOKING PARKING METHODS":*

- *initBTCalendar*: inicializa el botón del calendario

- *loadData*: realiza las llamadas necesarias para saber cuáles son las plazas reservadas para el usuario. Las llamadas realizadas pueden consultarse a través de los números dibujados en el esquema del apartado de *Arquitectura .* Este método se llama desde diferentes partes de la actividad para refrescar los datos siempre que hay un cambio. Se podría haber evitado realizar la llamada y actualizarlos según las entradas del usuario, pero se considera más conveniente hacerlo así para tener los datos lo más actualizados posibles y evitar posibles errores de sincronización.

- *Booking1, Booking2, Booking3 y Booking4:* se corresponden con los estados de la pantalla de reservas, que también pueden consultarse en el esquema del apartado *Arquitectura*. Estos métodos se encargan de mostrar el color correspondiente del botón cuando sea pulsable (azul) o no (gris), mostrar los textos correspondientes a cada estado y actualizar la lógica de cada botón en función del estado en el que se encuentre el usuario en ese momento.
  Dependiendo del estado, el usuario podrá liberar plaza o no, reservar plaza o no, podrá liberar su plaza u otra plaza, etcétera.

- *releaseAndDeleteOtherParking*: llamada al API para eliminar la reserva de un aparcamiento.

- *releaseMyParking*: llamada al API para liberar el aparcamiento asignado a un usuario.

- *deleteReleaseMyParking*: llamada al API para eliminar la liberación de la plaza del usuario.

Cuando el usuario pulsa el botón de Reserva y la aplicación considera que tiene que tiene que reservar una plaza que no es la suya, entra el juego el *fragment ParkingFreeFragment*, encargado de mostrar la lista de plazas disponibles ese día.

Al igual que con el otro *fragment*, la lógica del fragmento no se encuentra en esta actividad, pero si podemos encontrar los métodos encargados de mostrar y ocultar el *fragment* cuando sea necesario debajo de los comentarios *"FREE PARKING METHODS":*

- *showMyFreeParkingFragment*
- *closeFreeFragment*

Podemos encontrar también los métodos correspondientes al menú superior derecho debajo de los comentarios "*MENU*":

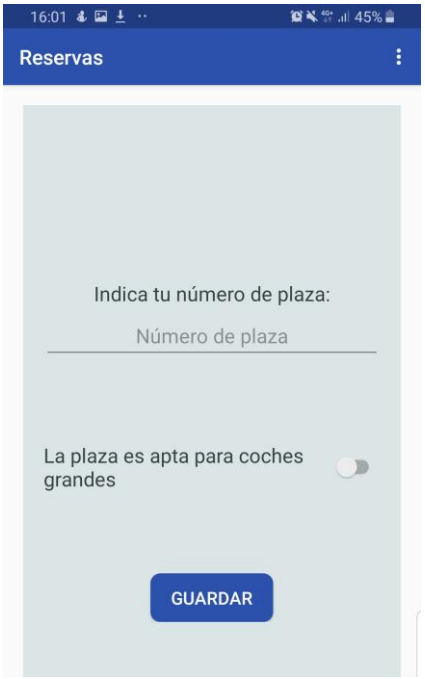
- *onCreateOptionMenu*
- *onOptionItemSelected*

Y, por último, otros métodos:

- *showMessages*: para mostrar mensajes al usuario
- *onBackPressed*: este método es el que gestiona el botón de atrás de Android. Se sobrescribe para que la aplicación se cierre cuando el usuario pulsa el botón de atrás. Se realiza esta acción ya que no queremos que el usuario vuelva a la pantalla de login, ya hemos almacenado anteriormente sus preferencias.

El SDK de Firebase gestiona la perdida de conexión y se queda esperando recuperar la conexión en caso de pérdida de la misma para todas las llamadas que no son Login y Registro.

MyParkingFragment



El *fragment* encargado de mostrar el *layout* perteneciente a la imagen de la derecha y de recoger los eventos del usuario.

Al igual que en las actividades, cuando el usuario pulsa el botón de guardar, se comprueba que el número de plaza no está vacío. Si es correcto, se muestra una alerta para preguntar si está seguro de haber introducido su plaza correctamente. Esto se hace porque, actualmente, no existe la posibilidad de cambiar el dato si no es contactando con el administrador de la aplicación. Se ha planteado cómo funcionalidad a futuro.

Si el usuario acepta, se realiza la llamada al API y se llama al método de *BookingActivity* que cierra el freagmento y almacena los datos en preferencias.

ParkingFreeFragment

El encargado de mostrar la lista de plazas disponibles para que el usuario pueda reservarlas.

El método *onCreateView* inicializa los elementos y botones de la vista. En este caso, podemos observar el botón "X" que, al ser pulsado, llamaba al método de *BookingActivity* para cerrar este Fragment.

En la inicialización podemos ver también la creación de dos nuevos elementos: un elemento del tipo *RecyclerView*, que es la parte de la interfaz correspondiente a la lista, y un *listener* que captura cuándo se pulsa un elemento de la lista.

Dentro del *OnCreateView* se llama también al método *getFreeParkingAnsSetAdapter*, que realiza la llamada al API para obtener la lista de plazas. Si no obtiene ninguna plaza, muestra un *layout* indicando que no hay plazas libres ese día.

Si obtiene plazas, inicializa el *adapter* de la lista, que gestiona la carga de los elementos.
Este adapter podemos encontrarlo en una clase diferente, denominada *MyItemRecyclerViewAdapter*, que incluye una lista de objetos Parking a mostrar. Se relaciona con el *fragment* a través del *listener*.
Los demás aspectos de funcionalidad son propios de Android.

Elementos de la vista

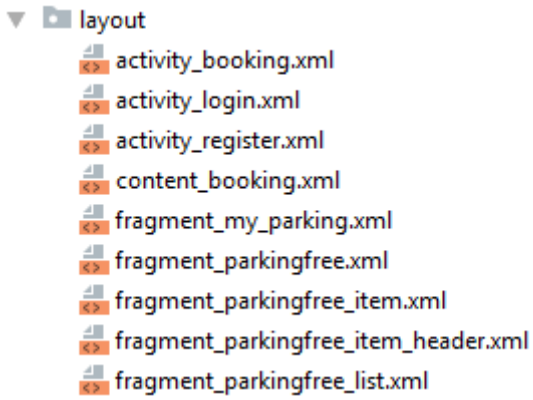
Todos los elementos de la interfaz se encuentran en la carpeta res.

Ya se ha explicado para qué se utiliza cada uno de ellos de forma global. En este apartado, se explicarán las peculiaridades más importantes.

Layout

Dentro de la carpeta *layout*, se encuentran las vistas de la aplicación. En todos los *layout* se ha intentado realizar un diseño que se adapte lo mejor posible a las diferentes pantallas. Para ello, se intenta introducir medidas relativas a la pantalla. Además, se intenta hacer que el contenido esté siempre ajustado a los márgenes derecho, superior e izquierdo de la pantalla de tal manera que, si el contenido crece, sea en vertical hacia abajo. Controlando esto en todos los elementos de la vista podemos incluir un *ScrollView* Vertical que permita al usuario hacer *Scroll* de arriba abajo sobre la pantalla cuando el contenido no quepa en su pantalla.
Así, aunque se ha elegido un modelo específico de dispositivo para realizar la prueba, nos aseguramos que pueda verse correctamente en todos los dispositivos posibles.

En los *layout* de las *activities* y *fragments* se incluye un *RelativeLayout* con dos contenedores en el interior: el primero, el contenido en sí de la vista ocupando toda la pantalla; el segundo, un *RelativeLayout* que contiene un *ProgressBar* que ocupa igualmente toda la pantalla (*match*_parent). En un principio, invisible. El *controller* será encargado de mostrar este último cuando se estén realizando llamadas al API.
Los *layout* incluidos son los siguientes:



- **Activity_booking**: se corresponde con la pantalla de booking. Este *layout* solo contiene la barra de navegación, realiza una inclusión del *layout content_booking*, que es el que en realidad tiene el contenido, y un *frameLayout* dónde irá el *fragment* de los Pop-Up.

- **Activity_login**: contiene los elementos de la vista necesarios para mostrar la pantalla de Login.

- **Activity_register**: comprende los elementos de la vista de registro.

- **Content_booking**: contenido real de la pantalla de reservas.

- **Fragment_my_parking**: contenido de la vista del Pop-Up de la elección de plaza

- **Fragment_parkingfree**: como se ha comentado, está formado por un *RelativeLayout* que contiene un *LinearLayout* con el contenido en sí y un *RelativeLayout* con la barra de progreso.
  Dentro del *LinearLayout*, podemos encontrar el botón de cerrar; un *LinearLayout* que incluye los *layouts* de la cabecera y la lista; y un *LinearLayout* denominado *LL_empyList*. Este último, lo muestra el contenedor cuando no existen plazas.
  Por tanto, los *layout* restantes, son subvistas de esta vista.

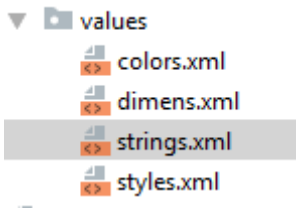
- **Fragment_parkingfree_item**: vista del elemento de la lista de aparcamientos libres.

- **Fragment_parkinfree_header**: vista del elemento de la cabecera de aparcamientos libres.

- **Fragment_parkingfree_list**: contiene únicamente la lista.

Values

El paquete *values* tiene gran importancia. En el deberían definirse la mayor parte de los colores, dimensiones, y estilos de la aplicación. De esta forma, están centralizados en un solo fichero y podemos hacerlos globales para toda la aplicación.



Por ello, hemos incluido los colores base de la aplicación dentro del archivo *colors*, las dimensiones más usadas dentro de *dimens* y los estilos dentro de *styles*.

Dentro de esta carpeta existe también el archivo *string*, importante para tener los textos centralizados, pero también para localizar los textos de la aplicación.

En este caso, la aplicación solo estará disponible en español ya que no se cree necesario el uso de otro idioma. Sin embargo, se considera una buena práctica separar los textos que se mostrarán al usuario del código en sí. De esta manera, se consigue que la aplicación sea más fácil de mantener y facilitamos la traducción a otro idioma en el futuro.
Al tener solo un idioma, nos bastará con tener una carpeta denominada *values* (carpeta por defecto), con el fichero *strings.xml*. En este fichero encontraremos todos los textos que se muestran en la aplicación.

Si en el futuro se quisiera añadir otro idioma, por ejemplo el inglés, podría hacerse creando otra carpeta denominada *values-en* y copiando el archivo *strings* con los textos traducidos al

inglés. La aplicación elegirá automáticamente en qué carpeta debe buscar los textos dependiendo del idioma del dispositivo.

AndroidManifest

En él se incluye el nombre y el icono de la aplicación. Se incluyen también las actividades, su nombre y el atributo *Android:screenOrientation="portrait"* para que la aplicación solo se vea en vertical. Se decide incluir este atributo porque no es una aplicación que necesariamente tenga que usarse en horizontal y cuya funcionalidad es más importante que el diseño. De esta forma, nos aseguramos de que nuestro diseño es capaz de adaptarse mejor a todos los dispositivos y nos evitamos los errores de cambio de orientación.

Build.gradle de la aplicación

Establecemos la propiedad *minSdkVersion* 26, que indica que la aplicación requerirá *Android API 26 (versión 8)* para funcionar, tal y cómo habíamos establecido en el diseño inicial.

Establecemos también *targetSdkVersion* a 26, que indica que la versión utilizada para realizar las pruebas ha sido la 26. Aunque también se han realizado pruebas en otras versiones, como Android 9.

En este fichero, también se añaden las dependencias propias de Firebase para que pueda funcionar correctamente.

Decisiones e implementación del servidor

Para la creación del servidor, se crea un proyecto Android en Firebase a través de su web:

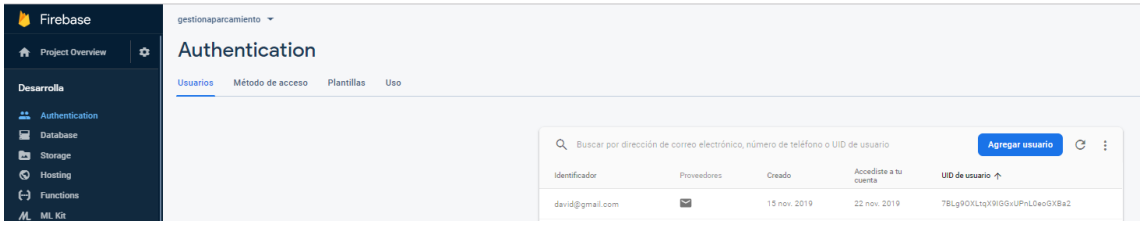
https://firebase.google.com/

Y se siguen las instrucciones indicadas en la misma web para que nuestra aplicación pueda acceder a ella:

1. Se debe indicar el nombre del paquete de la aplicación Android.
2. Descargar un archivo de configuración e incluirlo en la aplicación.
3. Modificar los archivos build.gradle para indicar las dependencias de Firebase.

Una vez realizado esto, la aplicación puede acceder correctamente al servidor.
En la web de Firebase existe bastante documentación y ejemplos de cómo ejecutar las llamadas y cómo desarrollar correctamente el código de acceso a esta herramienta, consultada previamente en el diseño de la parte del servidor.

Una vez comprobado que la aplicación puede acceder correctamente al SDK, accedemos a la parte de Authenticación de la consola para definir el método de autenticación de los usuarios. Existen varios métodos: correo electrónico, teléfono, anónimo, acceso mediante las cuentas de Google, Facebook, etcétera. Cómo ya se había definido en el diseño, se elegirá el método de email y contraseña.

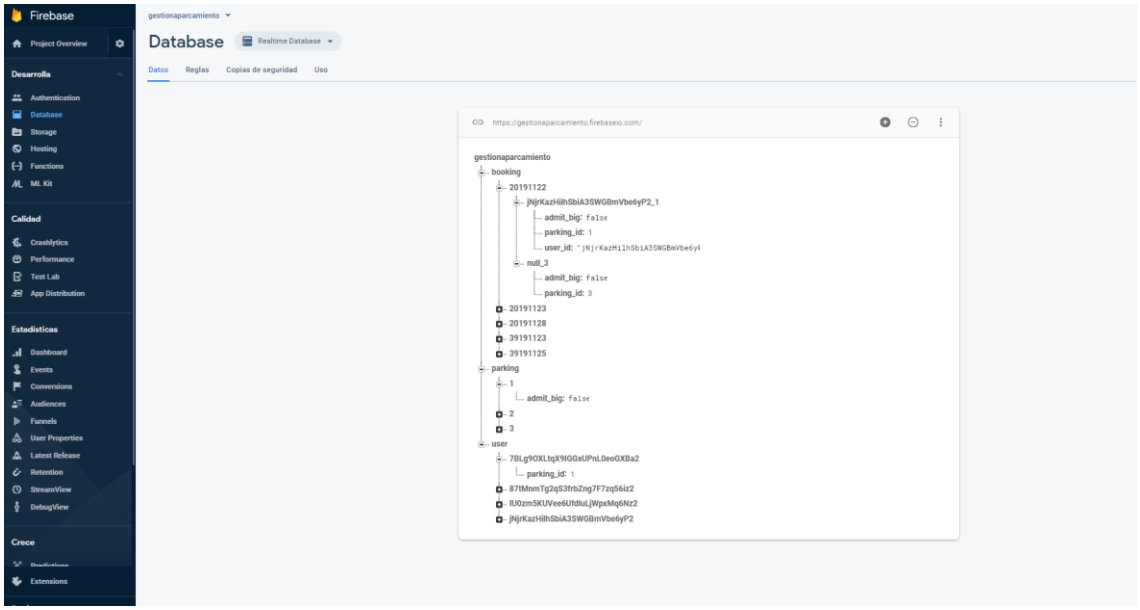


Todos los usuarios que vayan realizando el registro de la aplicación a través del SDK aparecerán en este panel con su identificador.

Desde este panel se puede eliminar las cuentas, inhabilitar la cuenta de un usuario o cambiar su contraseña, por esta razón se decide que, para cambiar la cuenta o la contraseña del usuario en la aplicación, es necesario contactar con el administrador. Se deja como acción a futura la implementación del cambio de cuenta en la aplicación.

En el menú de Firebase existe otro apartado denominado Database, dónde se incluirá el json que contiene los datos.

La ejecución de la aplicación comienza con un json vacío, sin datos. Será la aplicación la encargada de ir rellenando este json y su estructura, de tal manera que, al finalizar las ejecuciones de los usuarios, podremos ver un json similar al siguiente:


Firebase
Project Overview
Desarrolla
Authentication
Database
Storage
Hosting
Functions
ML Kit
Calidad
Crashlytics
Performance
Test Lab
App Distribution
Estadísticas
Dashboard
Events
Conversions
Audiences
Funnels
User Properties
Latest Release
Retention
StreamView
DebugView
Crece
Extensions
gestionaparcamiento
Database
Realtime Database
Datos
Reglas
Copias de seguridad
Uso
https://gestionaparcamiento.firebaseio.com/
gestionaparcamiento
booking
20191122
jNjrKazHilhSbiA3SWGBmVbe6yP2_1
admit_big: false
parking_id: 1
user_id: "jNjrKazHilhSbiA3SWGBmVbe6yP
null_3
admit_big: false
parking_id: 3
20191123
20191128
39191123
39191125
parking
1
admit_big: false
2
3
user
7BLg9OXLtqX9IGGxUPnL0eoGXBa2
parking_id: 1
87tMnmTg2qS3frbZng7F7zq56iz2
IUOzm5KUVee6UfdluLjWpxMq6Nz2
jNjrKazHilhSbiA3SWGBmVbe6yP2

Pruebas

En este apartado se define el plan de pruebas a realiza, una matriz que identifica dónde se prueba cada requisito funcional para ver que los requisitos se cumplen correctamente y el resultado de las pruebas.

Plan de pruebas

Se muestra cada prueba a realizar en una tabla:

P-01	
Identificador	P-01
Descripción	En la pantalla de inicio de sesión, se verifica que el formulario comprueba los valores correctamente: - Si los valores están vacíos. - Si el correo electrónico es correcto. - Si la contraseña tiene una longitud mayor de 6 . Y que permite el login una vez los formatos son correctos: - Si los valores no coinciden con los valores del servidor, se muestra la alerta indicando que el login es incorrecto. - Si los son correctos, se accede a Booking.
Pre-condiciones	El usuario debe estar registrado.
Pasos	1. Introducir email y contraseña vacíos y comprobar que aparece el error correspondiente. 2. Introducir email inválido y comprobar que aparece el error correspondiente. 3. Introducir contraseña menor de seis caracteres y comprobar que aparece el error correspondiente. 4. Introducir los formatos correctos y valores inválidos para ver que el login no fue exitoso. 5. Introducir los formatos y valores correctos y ver que se procede al login.
Resultado	Se lanzará la llamada de login solo cuándo los datos sean correctos.

P-02	
Identificador	P-02
Descripción	En la pantalla de Inicio de sesión, se comprueba que el botón ¿Has olvidado tu contraseña? devuelve la alerta deseada.
Pre-condiciones	
Pasos	1. Pulsar el botón ¿Has olvidado tu contraseña? y ver que se muestra la alerta correspondiente.
Resultado	Se muestra la alerta que indica que se debe contactar con el administrador.

P-03	
Identificador	P-03
Descripción	En la pantalla de Inicio de sesión, se comprueba que la llamada de inicio de sesión se ejecuta correctamente.
Pre-condiciones	El usuario debe estar registrado.
Pasos	1. Introducir usuario y contraseña correcto 2. Sin conectividad, pulsar el botón de iniciar sesión y comprobar que se muestra el mensaje correspondiente. 3. Con conectividad, pulsar el botón y comprobar que se inicia sesión de forma correcta y se accede a la pantalla de Booking.
Resultado	Sin conectividad, alerta de error. Con conectividad, inicio de sesión y acceso a la pantalla de Booking.

P-04	
Identificador	P-04
Descripción	Comprobar que, al iniciar sesión una vez, no vuelve a pedir el login la segunda vez que se inicia sesión.
Pre-condiciones	
Pasos	1. Iniciar sesión en la aplicación. 2. Desde la pantalla de Booking, pulsar botón de atrás para cerrar la aplicación. 3. Abrir la aplicación y comprobar que no pide credenciales.
Resultado	La aplicación no pide credenciales la segunda vez que se ha iniciado sesión.

P-05	
Identificador	P-05
Descripción	En la pantalla de Inicio de sesión, se comprueba que se accede correctamente al registro.
Pre-condiciones	
Pasos	4. Pulsar el botón de Registrar. 5. Comprobar que se accede a la pantalla de Registro.
Resultado	Acceso a la pantalla de registro.

P-06	
Identificador	P-06
Descripción	En la pantalla de registro, se verifica que el formulario comprueba los valores correctamente: - Si los valores están vacíos. - Si el correo electrónico es correcto. - Si la contraseña tiene una longitud mayor de 6. - Si las contraseñas coinciden. Si los valores son correctos, se procede al registro: - Si el email ya existe, debe mostrar la alerta indicándolo. - Si no existe, se procede al registro.
Pre-condiciones	Estar en la pantalla de Registro.
Pasos	6. Introducir email y contraseña vacíos y comprobar que aparece el error correspondiente al pulsar el botón de registro. 7. Introducir email inválido y comprobar que aparece el error correspondiente al pulsar el botón. 8. Introducir contraseña menor de seis caracteres y comprobar que aparece el error correspondiente al intentar el registro. 9. Introducir las contraseñas diferentes y comprobar que aparece el error correspondiente al pulsar el botón. 10. Introducir los formatos válidos pero un email repetido, comprobar que no deja formalizar el registro 11. Introducir los formatos y valores correctos. Comprobar que se registra al pulsar el botón de registro.
Resultado	Se lanzará la llamada de registro solo cuándo los datos sean correctos.

P-07	
Identificador	P-07
Descripción	En la pantalla de Registro, se comprueba que la llamada se ejecuta correctamente.
Pre-condiciones	Estar en la pantalla de Registro.
Pasos	1. Introducir usuario y contraseñas correctas. 2. Sin conectividad, pulsar el botón de registro y comprobar que se muestra el mensaje correspondiente. 3. Con conectividad, comprobar que se inicia sesión de forma correcta y se accede a la pantalla de Booking.
Resultado	Sin conectividad, alerta de error. Con conectividad, registro y acceso a la pantalla de Login.

P-08	
Identificador	P-08
Descripción	Comprobar que, al hacer login por primera vez, aparece el pop-Up de elección de aparcamiento.
Pre-condiciones	
Pasos	1. Hacer login en la aplicación por primera vez. 2. Comprobar que aparece el Pop-Up de elección de aparcamiento.
Resultado	PopUp de elección de plaza de aparcamiento.

P-09	
Identificador	P-09
Descripción	En el Pop-Up de elección de plaza de aparcamiento, comprobar que se pueden introducir los datos correctamente: - Si la plaza es nula aparece un error. - La plaza solo permite introducir valores numéricos. - Se puede introducir que se admitan coches grandes. Cuando los valores son correctos, se accede a la llamada de registrar la plaza de aparcamiento para ese usuario.
Pre-condiciones	Inicio de sesión
Pasos	1. No introducir plaza y pulsar Guardar, aparece un error indicando que la plaza es nula. 2. Intentar introducir valores no numéricos y verificar que no es posible. 3. Marcar el check de admite grandes. 4. Cuando los valores son correctos, registrar la plaza.
Resultado	Se lanzará la llamada de registro solo cuándo los datos sean correctos.

P-10	
Identificador	P-10
Descripción	En el Pop-Up de elección de plaza, se comprueba que se ejecuta correctamente la llamada al servidor y se accede a la pantalla de Booking.
Pre-condiciones	Estar en el Pop-Up de elección de plaza de aparcamiento.
Pasos	1. Introducir los datos correctos de la plaza. 2. Pulsar el botón Guardar sin conexión y comprobar que se mantiene esperando. 3. Recuperar conexión, comprobar que la llamada se ejecuta correctamente y que se accede a la pantalla de Booking.
Resultado	Sin conectividad, se mantiene esperando Con conectividad, se ejecuta la llamada y se registra la plaza

P-11	
Identificador	P-11
Descripción	Comprobar que el Pop-Up de indicar el número de plaza no vuelve a aparecer si hemos indicado la plaza anteriormente.
Pre-condiciones	Introducir plaza de aparcamiento completa.
Pasos	1. Hacer login. 2. Comprobar que el PopUp de plaza de aparcamiento no aparece.
Resultado	No aparece el Pop-Up de indicación de plaza de aparcamiento.

P-12	
Identificador	P-12
Descripción	Desde Booking, comprobar que se accede correctamente al calendario y se permite elegir el día.
Pre-condiciones	Estar en la pantalla de Reservas.
Pasos	1. Pulsar el botón del calendario. 2. Pulsar el día. 3. Comprobar que el botón aparece el día correctamente.
Resultado	El botón del calendario cambia correctamente de día.

P-13	
Identificador	P-13
Descripción	Desde Booking, comprobar se accede correctamente a los diferentes estados: - Reservas 1: permite liberar la plaza e ir a Reservas 2 y reservar la plaza e ir a Reservas 4. - Reservas 2: no permite liberar plaza y permite reservar plaza e ir a Reservas 1 o Reservas 3 dependiendo de si la plaza está ocupada o no. - Reservas 3: permite liberar plaza e ir a Reservas 2, no permite reservar plazas. - Reservas 4: Permite liberar plaza e ir a Reservas 1, no permite Reservar plaza.
Pre-condiciones	Estar en la pantalla de Reservas
Pasos	1. Desde el estado Reservas 1, comprobar que los textos son correctos. 2. Desde el estado Reservas 1, liberar plaza e ir al estado Reservas 2. 3. Desde el estado Reservas 1, reservar la plaza e ir al estado Reservas 4. 4. Desde el estado Reservas 2, comprobar que los textos son correctos. 5. Desde el estado Reservas 2, comprobar que el botón Liberar plaza está en gris y no es pulsable. 6. Desde el estado Reservas 2, sabiendo que la plaza del usuario está libre, comprobar que se accede a Reservas 1. 7. Desde el estado Reservas 2, sabiendo que la plaza del usuario no está libre, comprobar que se accede a Reservas 3. 8. Desde el estado Reservas 3, comprobar que los textos son correctos. 9. Desde el estado Reservas 3, comprobar que el botón de Reservar está en gris y no es pulsable. 10. Desde el estado de Reservas 3, liberar plaza y comprobar que se accede al estado Reservas 2. 11. Desde el estado Reservas 4, comprobar que los textos son correctos. 12. Desde el estado Reservas 4, comprobar que el botón Reservar aparece en gris y no es pulsable. 13. Desde el estado Reservas 4, comprobar que al liberar plaza se accede a Reseras 1.

Resultado	El texto de la pantalla Booking, los botones y las transiciones entre estados son las marcadas en el prototipo inicial.

P-14	
Identificador	P-14
Descripción	En la pantalla de Booking, se comprueba que el botón de Liberar plaza realiza las llamadas correctamente en todos los estados.
Pre-condiciones	Estar en la pantalla de Booking
Pasos	1. Pulsar el botón de Liberar plaza desde cada uno de los estados sin conexión, comprobar que la aplicación se mantiene esperando. 2. Recuperar la conexión y comprobar que la llamada se ejecuta.
Resultado	Las llamadas de liberar plaza se ejecutan correctamente: - Sin conectividad, se mantiene esperando. - Con conectividad, realiza la llamada.

P-15	
Identificador	P-15
Descripción	En la pantalla de Booking, se comprueba que el botón de Reservar otra plaza abre correctamente el PopUp de plazas libres.
Pre-condiciones	Estar en la pantalla de Booking.
Pasos	3. Pulsar el botón de Reservar plaza y comprobar que se abre el popUp de plazas libres.
Resultado	PopUp de plazas libres.

P-16	
Identificador	P-17
Descripción	Se comprueba que el PopUp de plazas libres funciona correctamente.
Pre-condiciones	
Pasos	1. Comprobar que, cuando no hay plazas disponibles, aparece el layout indicando que no hay plazas libres. 2. Verificar que, cuando hay plazas, aparecen las plazas disponibles. Verificar que las plazas son pulsables y que, una vez pulsadas, se realiza la reserva correspondiente. 3. Verificar que el botón de Cerrar cierra correctamente el popUp.
Resultado	Funcionamiento correcto de Pop-Up de plazas libres.

P-17	
Identificador	P-18
Descripción	Se comprueba que el Pop-Up de plazas libres ejecuta correctamente las llamadas al API.
Pre-condiciones	Estar en el PopUp de plazas Libres.
Pasos	1. Reservar una de las plazas sin conexión, comprobar que la aplicación se mantiene esperando. 2. Recuperar conexión, comprobar que la llamada se ejecuta correctamente.
Resultado	Funcionamiento correcto de la llamada de Pop-Up de plazas libres

P-18	
Identificador	P-19
Descripción	Desde Booking, se comprueba que se abre correctamente el menú superior derecho y que sus opciones se ejecutan correctamente.
Pre-condiciones	Estar en Booking
Pasos	1. Abrir menú superior derecho. 2. Pulsar cerrar sesión y comprobar que se cierra correctamente sesión. 3. Pulsar Mi plaza y comprobar que se muestra la alerta correspondiente. 4. Pulsar Mi cuenta y comprobar que se muestra la alerta correspondiente.
Resultado	Correcta funcionalidad del menú superior derecha.

P-19	
Identificador	P-20
Descripción	Pruebas en distintos dispositivos para verificar que: - La aplicación funciona correctamente en el dispositivo corporativo: Samsung Galaxy J6 versión 8.0 - La aplicación funciona y se visualiza correctamente en un dispositivo con versión Android superior. - La aplicación funciona y se visualiza correctamente en una Tablet (diferentes tamaños de pantalla).
Pre-condiciones	
Pasos	1. Realizar las pruebas con un Samsung Galaxy J6 versión 8.0. 2. Realizar las pruebas con un dispositivo con versión de Android superior a 8.0. 3. Realizar las pruebas con una Tablet. Verificar que el comportamiento de la aplicación es correcto y la interfaz es correcta.
Resultado	Correcta funcionalidad e interfaz en todos los dispositivos.

P-20	
Identificador	P-21
Descripción	Comprobar que la aplicación es intuitiva.
Pre-condiciones	
Pasos	Pedir la aplicación a varios usuarios que no conozcan la aplicación, explicarles para qué sirve y dejar que la utilicen, comprobar que son capaces de moverse por la aplicación sin problemas y son capaces de realizar todas las funcionalidades de la aplicación.
Resultado	Comprobar que la aplicación es fácil de usar e intuitiva.

Matriz requisitos/pruebas

En la siguiente matriz se muestran tanto los requisitos funcionales, como los no funcionales, con sus correspondientes pruebas, para verificar que todos los requisitos se han contemplado, implementado y probado:

		Requisitos funcionales									Requisitos no funcionales					
		RF1	RF2	RF3	RF4	RF5	RF6	RF7	RF8	RF9	RNF1	RNF2	RNF3	RNF4	RNF5	RNF6
Pruebas	P-01		X													
	P-02															
	P-03		X													
	P-04															
	P-05	X														
	P-06	X														
	P-07	X														
	P-08			X	X											
	P-09			X	X											
	P-10			X	X											
	P-11			X	X											
	P-12					X	X	X	X							
	P-13					X	X	X	X							
	P-14						X	X								
	P-15					X			X							
	P-16					X	X	X	X							
	P-17					X	X	X	X							
	P-18					X	X	X	X							
	P-19									X						
	P-20										X	X	X	X		
	P-21														X	

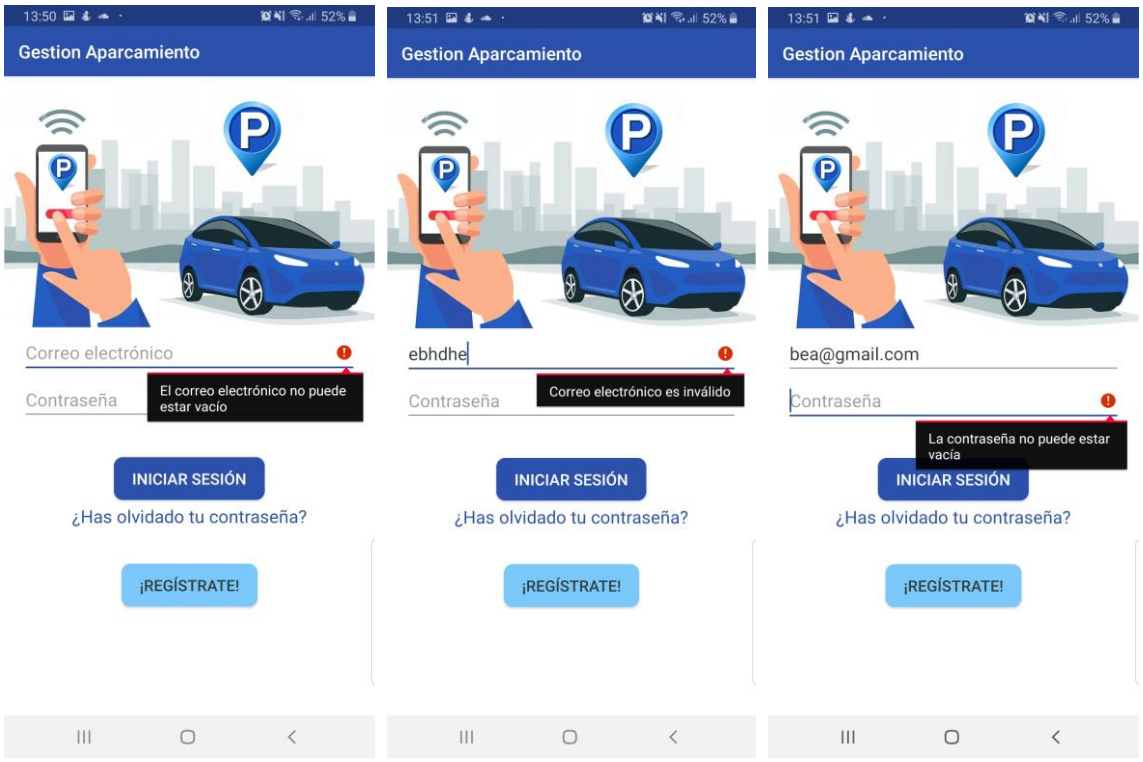
Cómo se puede observar, algunas pruebas no tienen correspondencia con su requisito. Esto es porque se han añadido algunas funcionalidades que no están ligadas a ellos pero que, igualmente, deben ser probadas.

El requisito no funcional RNF6 indica que la aplicación debe ser fácil de desarrollar y mantener, no es posible probar este requisito cómo tal, por eso no se corresponde con ninguna prueba, aunque si se ha intentando que se cumpla durante todo el desarrollo.

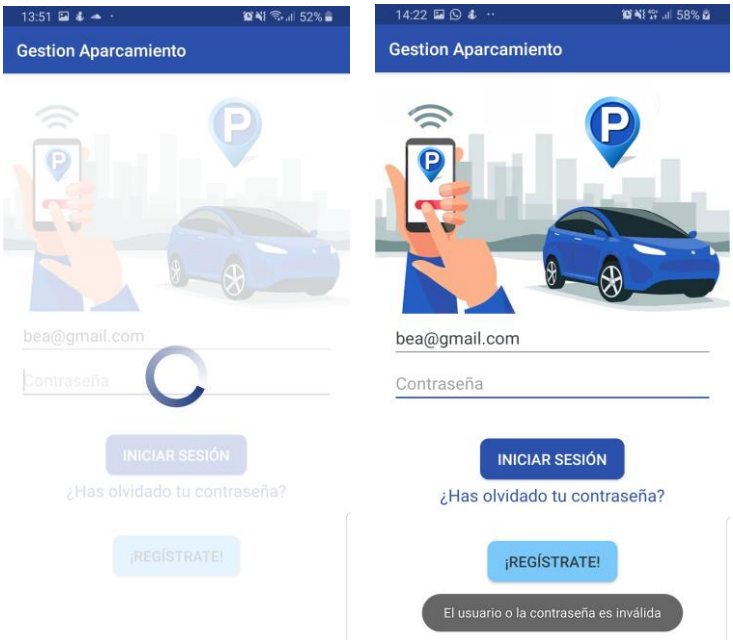
Resultado de las pruebas

P-01

Se comprueban que aprecen los errores correspondientes:

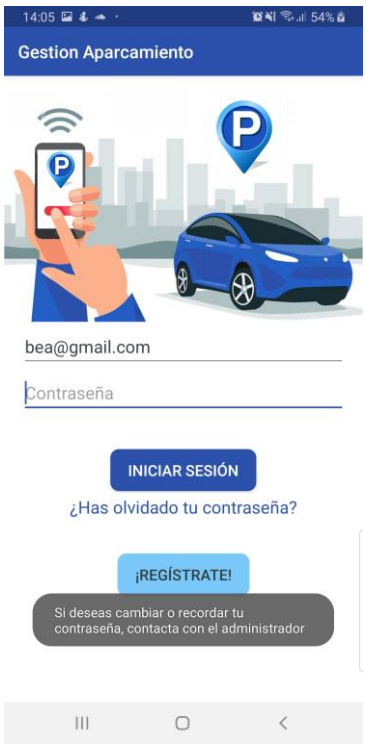


Una vez introducidos los formatos correctos, se procede a intentar el inicio de sesión: si el usuario es inválido porque no se ha registrado aún, se muestra el mensaje. Si el usuario es válido, se inicia sesión.



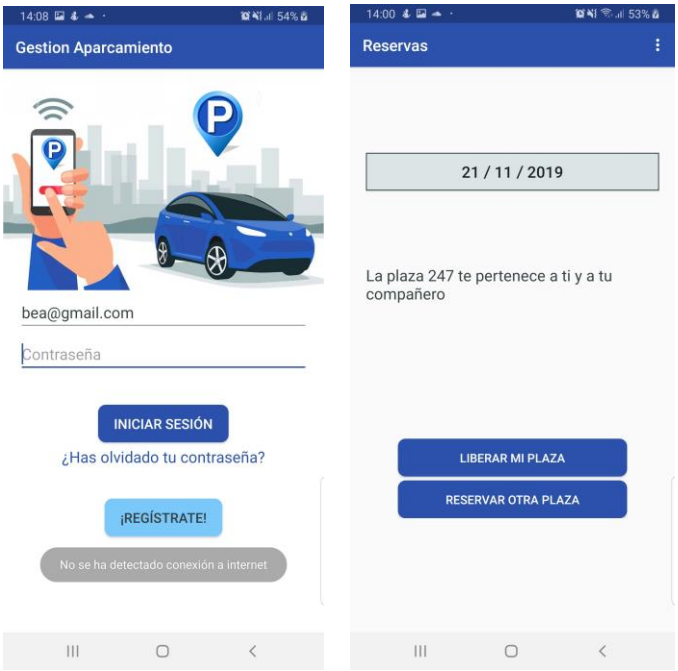
P-02

Se comprueba que el botón ¿has olvidado tu contraseña? tiene el funcionamiento esperado: muestra la alerta.



P-03

Se confirma que, si se inicia sesión sin conectividad, se obtiene el mensaje indicando que no hay conexión. En caso contrario, se realiza la llamada y se accede a Booking:



P-04

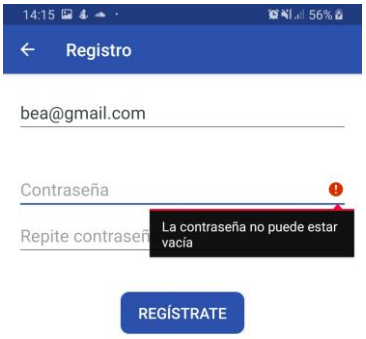
Se inicia sesión, se cierra la aplicación y se vuelve a entrar en ella. Se verifica el comportamiento. La segunda vez no pide credenciales.

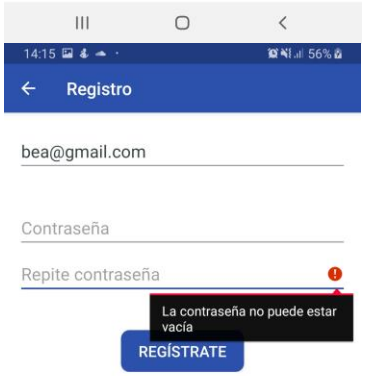
P-05

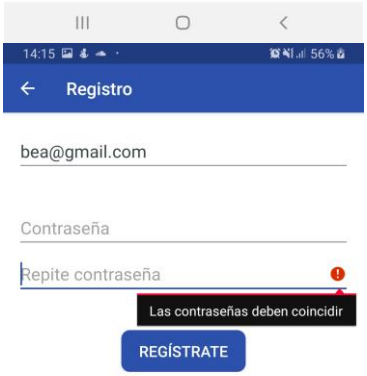
Se pulsa el botón de Registrar de la pantalla de Login. Se accede a la pantalla de Registro.

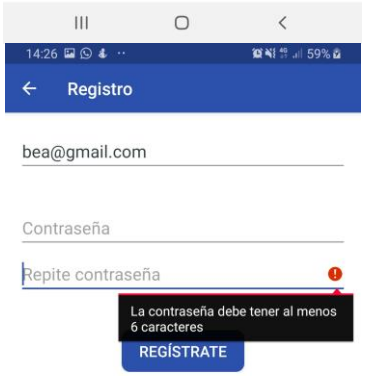
P-06

Se accede a la pantalla de Registro y verifican los mensajes de error cuando el formato de los campos no es correcto:

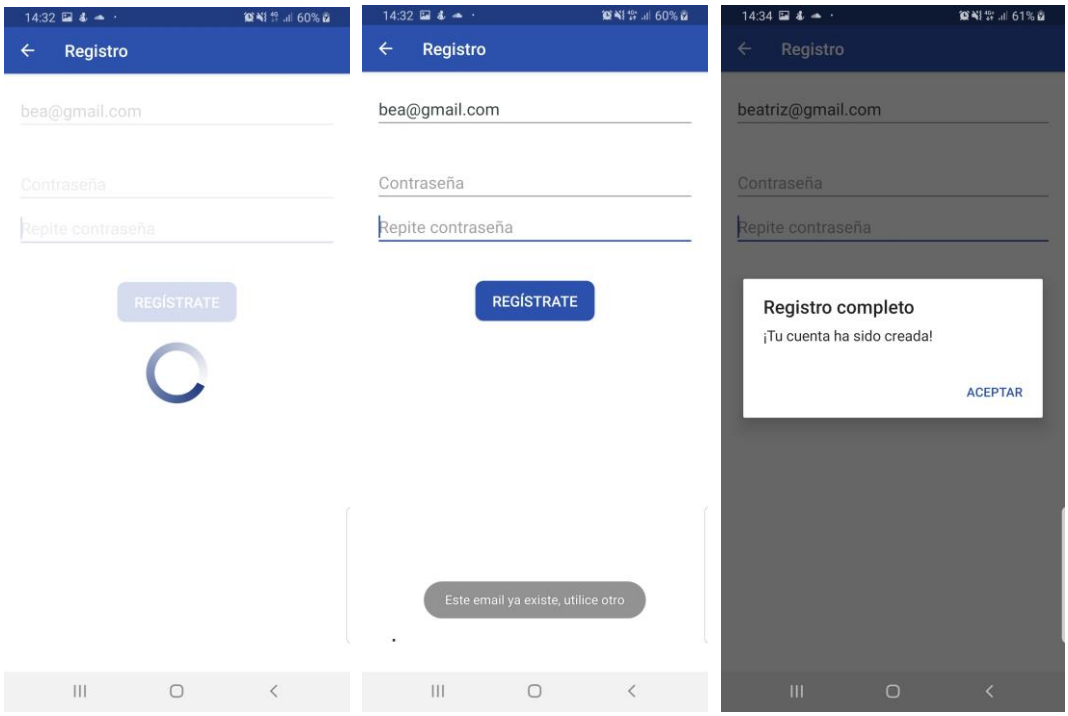






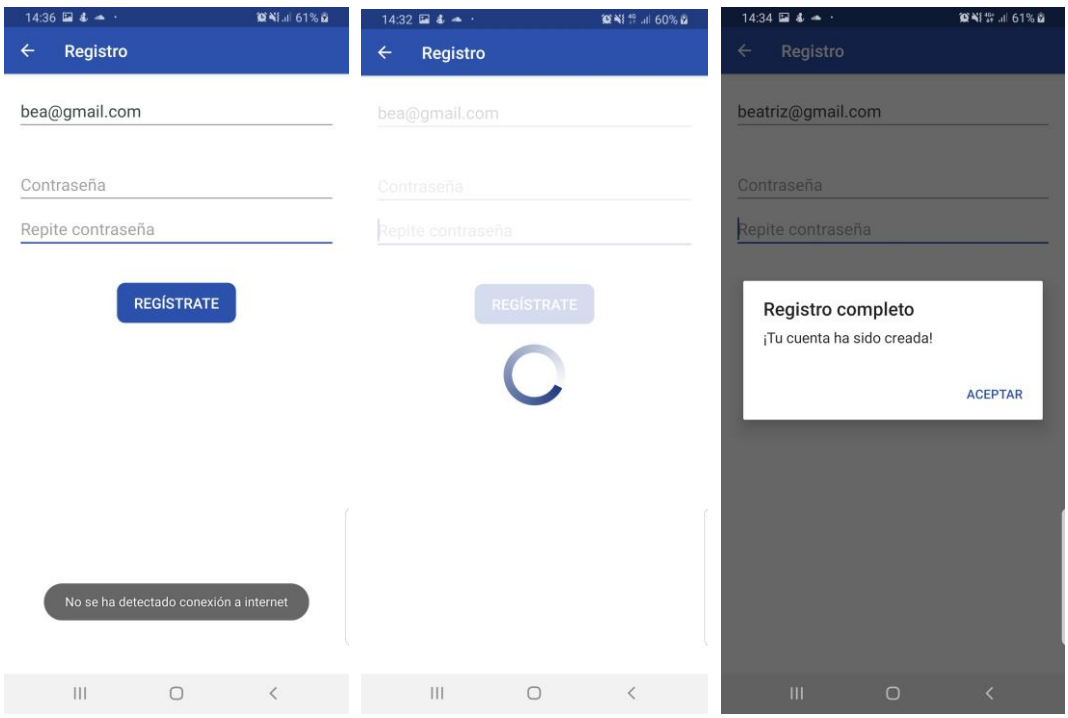


Una vez el formato es correcto, se procede a la llamada de Registro. Si el email ya se ha registrado, devuelve una alerta indicándolo. Si no, se registra el usuario correctamente:



P-07

Se accede a la pantalla de Registro y se intenta realizar el registro sin conexión. Se obtiene un mensaje indicando que no hay conexión. Se realiza la llamada con conexión y se comprueba que el registro se realiza correctamente. Al pulsar Aceptar, se accede a la pantalla de Booking.

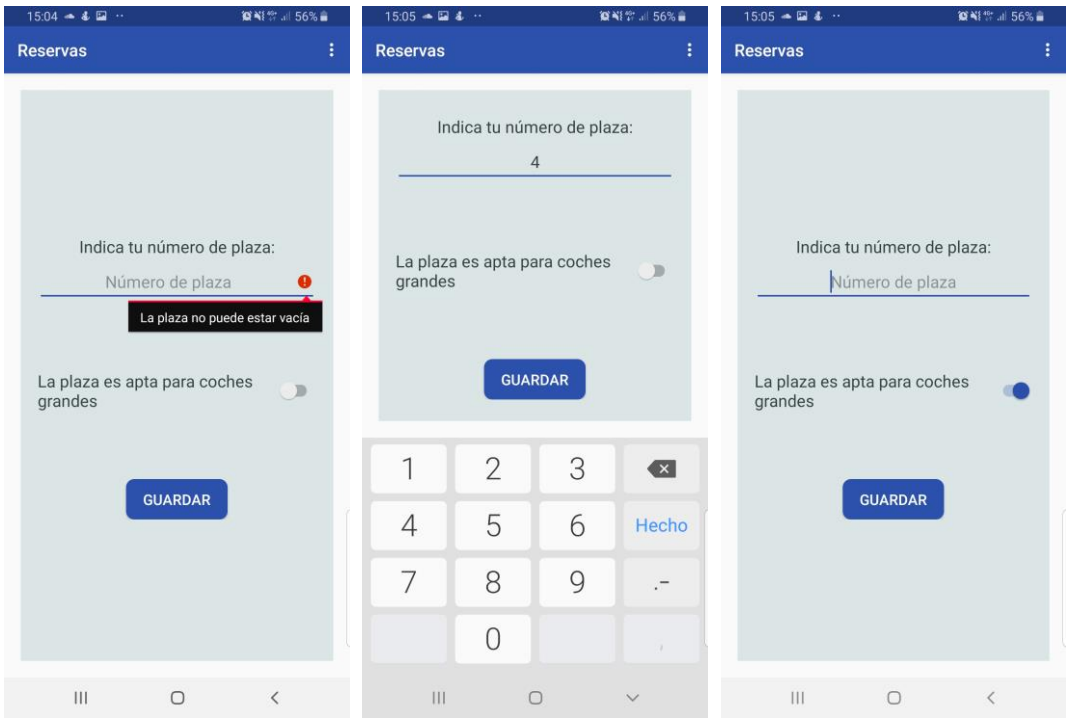


P-08

Se inicia sesión por primera vez con un usuario recién registrado y se comprueba que aparece el Pop-Up de elección de aparcamiento:

P-09

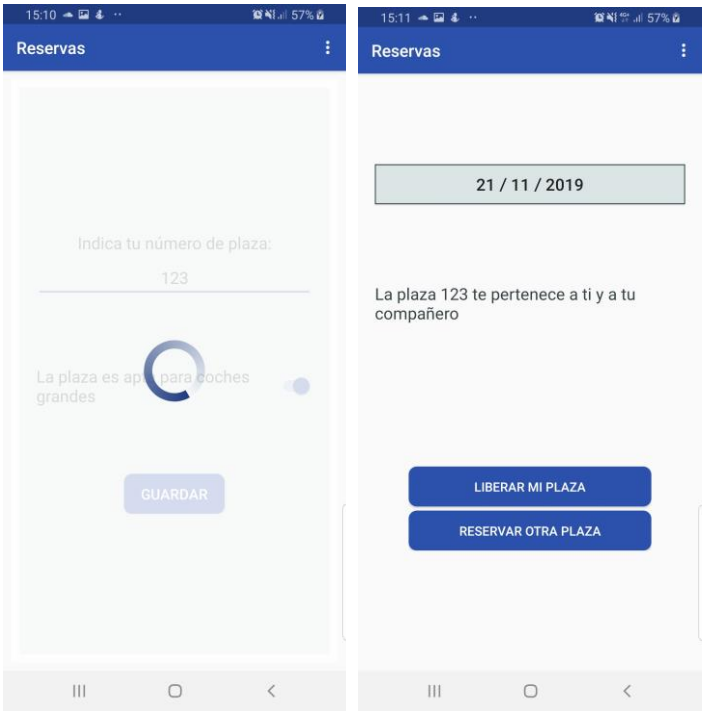
En el Pop-Up de elección de plaza de aparcamiento, se verifica que la plaza no puede ser nula, que solo admite valores numéricos (teclado numérico) y que se puede marcar el check de coches grandes:



Cuando los valores son correctos, se procede a guardar la plaza.

P-10

En el Pop-Up de elección de plaza, se comprueba que la llamada se realiza correctamente cuando los valores son correctos. Si no hay conexión, se mantiene a la espera. Si hay conexión, automáticamente reanuda la llamada se accede a Reservas.

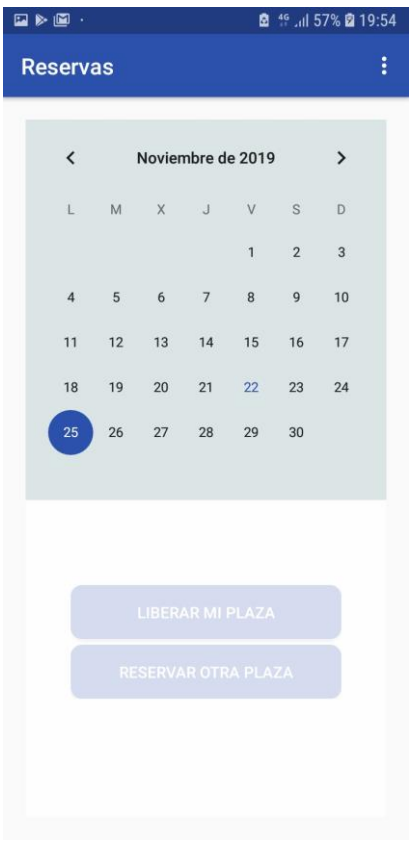


P-11

Se verifica que, una vez introducida la plaza en el Pop-Up de introducir plaza, ya no vuelve a aparecer. Ni aunque desinstalemos e instalemos la aplicación.

P-12

Desde la pantalla de Booking, al pulsar el botón de la fecha, se abre el calendario. Si elegimos, por ejemplo, el día 25, lo vemos reflejado en el botón.



P-13

Para la realización de esta prueba, se utilizan cuatro usuarios, con sus respectivas plazas:

Usuario1	Plaza 1
Usuario2	Plaza 2
Usuario3	Plaza 3
Usuario4	Plaza 1

Se utiliza al Usuario 1 para mostrar todas las transiciones de la pantalla Booking, con todos sus estados. En cada estado se comprobarán los mensajes, colores de los botones, si se pueden pulsar o no y el efecto al pulsarlos.

A continuación, las acciones de los usuarios:

1. Usuario1 está en Reservas 1:

Los textos y colores son los adecuados. Se comprobarán las transiciones.

2. Usuario1 en Reservas1, libera su plaza, accediendo así a Reservas2:

3. Usuario 1 en Reservas 2, con su plaza liberada y sin que nadie aún la haya reservado, pulsa el botón de Reservar plaza.
   Como su plaza está libre, la aplicación automáticamente reserva su plaza, volviendo así a Reservas 1, es decir, a la pantalla del paso 1.
   Vuelve a liberar su plaza para volver a Reservas 2 y seguir comprobando las transiciones

4. Ahora entra en juego el Usuario2. Usuario2 reserva la plaza liberada por el Usuario1. Usuario1 vuelve a pulsar el botón de Reservar Plaza. Como su plaza ahora sí está ocupada, aparece el Pop-Up de reservar plaza:

En esta ocasión, vacío, porque nadie liberó ninguna plaza más.
Un nuevo usuario, el Usuario 3, libera la plaza número 3.
Usuario1 vuelve a pulsar el botón de Reservar plaza. Esta vez le aparece el botón con la plaza 3:

La reserva, accediendo así a Reservas 3:

5. En Reservas 3, el Usuario1 libera su plaza, volviendo a Reservas 2 y comprobando el correcto funcionamiento de esta transición.

6. Mediante el calendario, el Usuario1 elige otro día, por ejemplo, el 28. El usuario se vuelve a encontrar en la situación inicial, es decir, en Reservas 1.
   El usuario intenta reservar plaza. Como no hay plaza plazas disponibles, sale el Pop-Up indicando que no hay plazas disponibles ese día.
   Usuario3 libera su plaza número 3 para ese día.
   Usuario1 vuelve a intentar reservar plaza. En este caso, aparece la 3. La reserva y accede así a Reservas 4:

7. Una vez en Reservas 4, libera la plaza y vuelve a Reservas 1. Se observa que la plaza liberada es la última que reservó, no su propia plaza.

8. En Reservas 1, el Usuario1 vuelve a liberar su plaza. Usuario4 inicia sesión y comprueba que su plaza también está liberada, ya que es el mismo número. Usuario4 vuelve a reservar la plaza. Usuario1 la ve reservada.

P-14

Se comprueba que, al pulsar el botón Liberar plaza de la pantalla de Booking se mantiene esperando en todos los estados (aparece la barra de progreso).
Si se vuelve a activar la conexión, se realiza la llamada correspondiente.

P-15

Al pulsar el botón Reservar otra plaza desde la pantalla Booking, se presenta el Pop-Up de plazas libres.

P-16

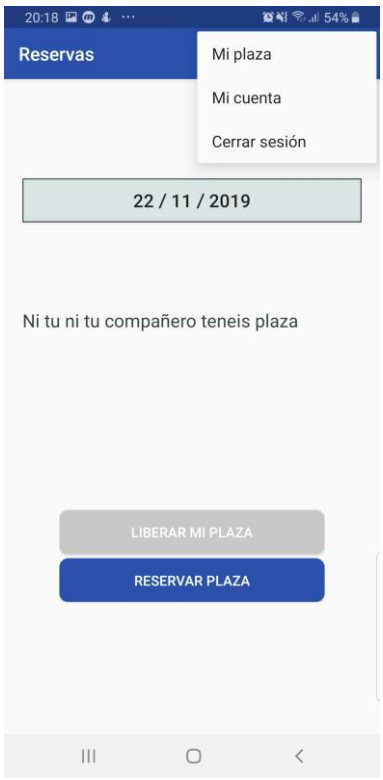
Se accede al Pop-Up de reserva de plazas. Se verifica que el *layout* cambia según haya plazas libres o no, que se puede cerrar el Pop-Up con el botón de "X" y que las plazas se pueden pulsar. Al pulsar una plaza, se reserva dicha plaza.

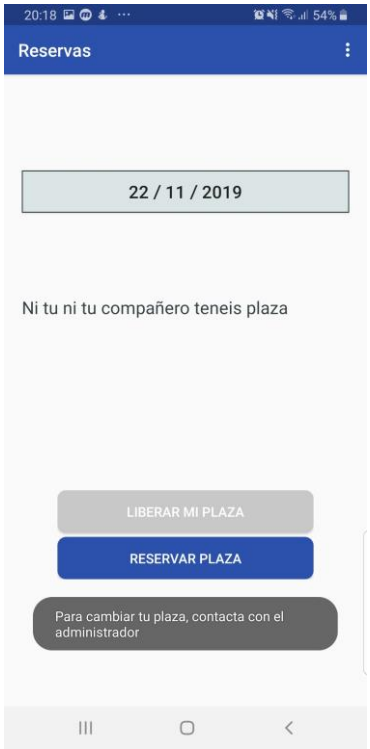
P-17

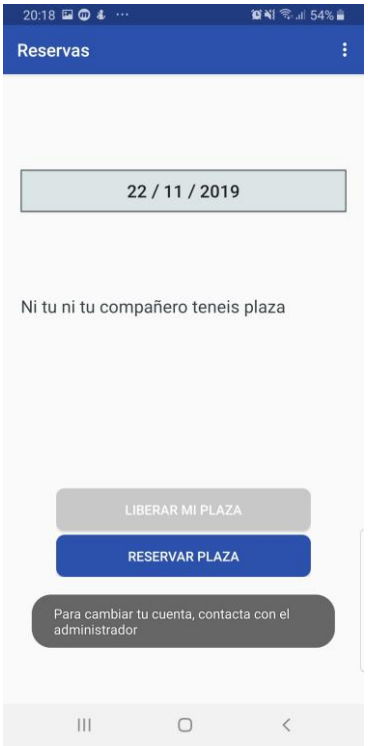
En el Pop-Up de plazas libres, se comprueba que, al desactivar la conexión a internet del dispositivo y pulsar sobre una plaza, se mantiene esperando.
Si se vuelve a activar la conexión, se realiza la llamada correspondiente.

P-18

Se comprueba el acceso al menú superior derecho. Se verifica que al pulsar el botón de cerrar sesión, la sesión se cierra; al pulsar Mi plaza, aparece una alerta. Al pulsar Mi cuenta, aparece una alerta.





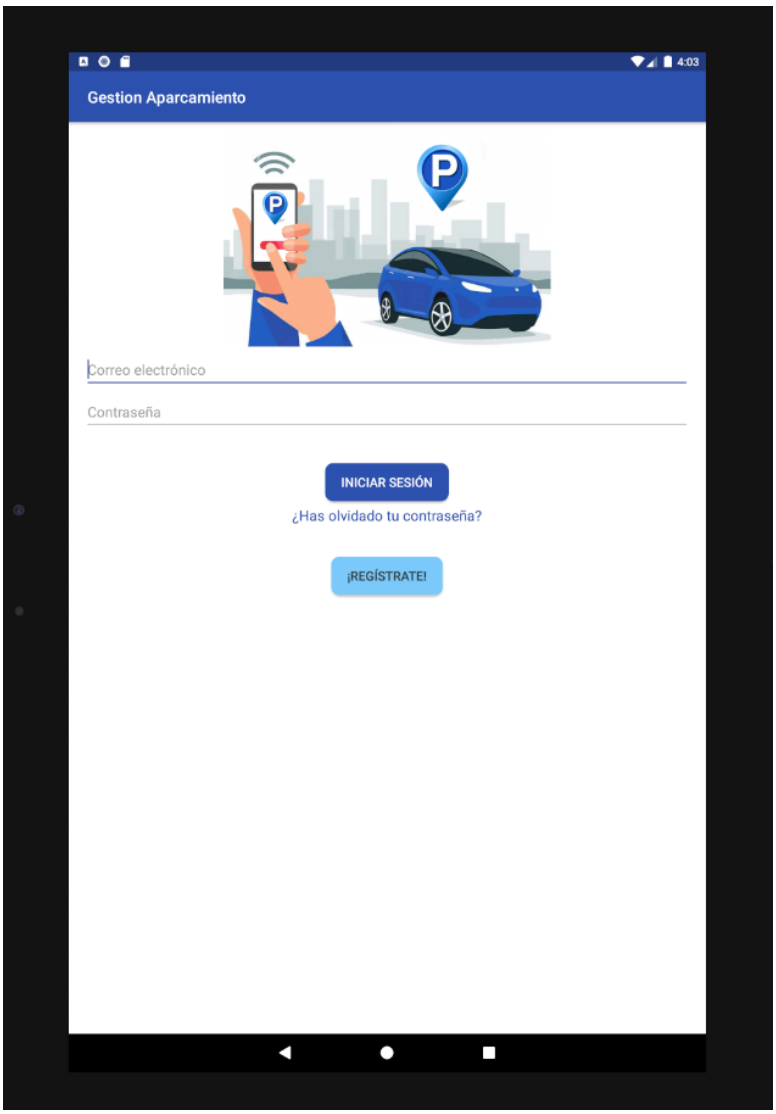


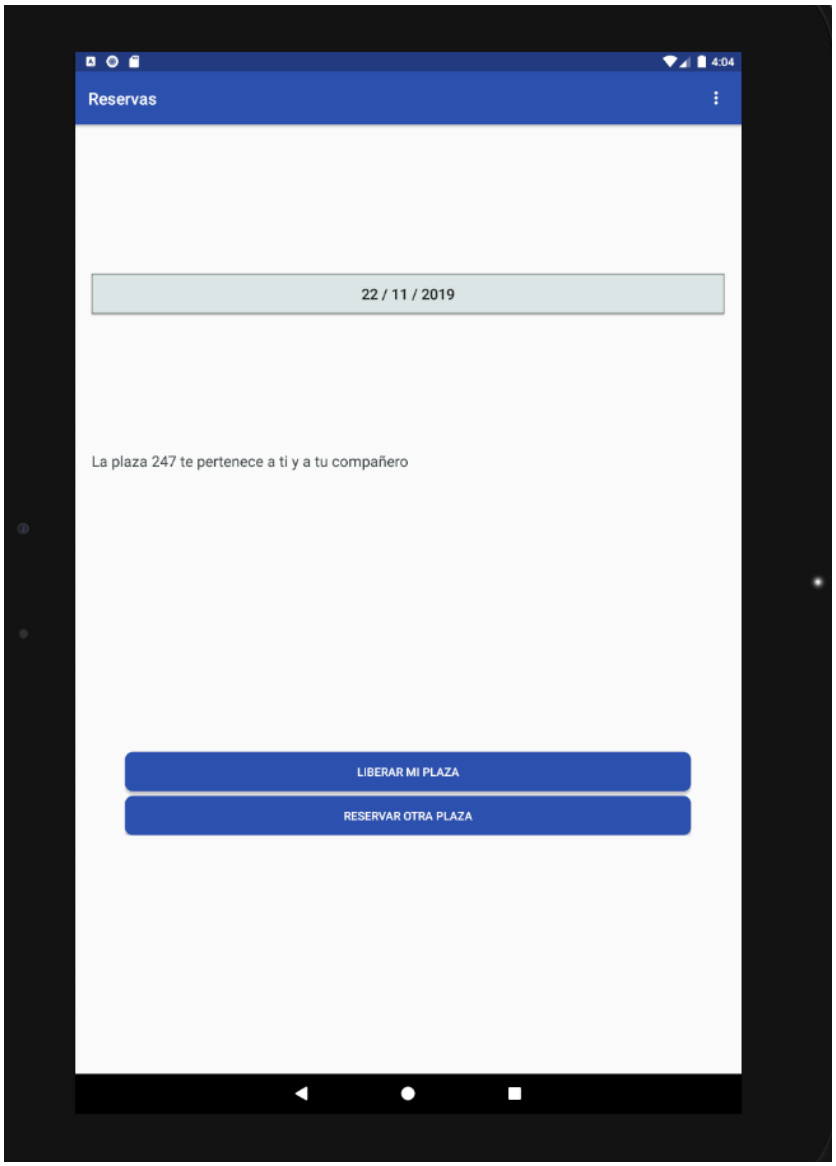
P-19

Se realizan las pruebas con un Samsung Galaxy J6 versión de Android 8, se comprueba que la funcionalidad anterior es correcta y se ven correctamente las pantallas.

Se realizan las pruebas con un Samsung Galaxy S8 + versión de Android 9, se comprueba que la funcionalidad y el diseño es correcto.

Se realizan las pruebas con una Tablet Nexus 10 con Android 8, entre otras, y se aprecia un funcionamiento y diseño correctos:





P-20

Se utiliza a un usuario trabajador de la empresa para realizar pruebas. Se le indica que la aplicación sirve para reservas de aparcamiento en el edificio y se le invita a probar la aplicación. Se comprueba que el usuario es capaz de moverse por la aplicación sin problemas. En una primera versión de la aplicación, el registro no volvía a la pantalla de Login. El usuario se queda extrañado por este comportamiento, por lo que en la versión siguiente se corrigió y se añadió este comportamiento.
Por todo lo demás, el usuario es capaz de utilizar todas las funcionalidades de la aplicación.
Se realizan estas pruebas con varios usuarios.

5. Conclusiones

A continuación, se explican conclusiones del trabajo.

Objetivos personales y aprendizaje

La evolución de la tecnología y los dispositivos móviles han hecho que se abra otra rama en el sector tecnológico, el desarrollo de aplicaciones móviles. Se requieren cada vez más aplicaciones que los usuarios puedan consultar en cualquier lugar o momento y que, además, interaccionen con el contexto, los usuarios de alrededor, los lugares que visitan, etcétera.

Debido a esto, creo que es importante la necesidad de especialistas que sean capaces de abordar un proyecto móvil de principio a fin. Esta fue la razón principal por la que comencé el máster en Desarrollo de aplicaciones Móviles y decidí hacer este proyecto.

Este proyecto me ha aportado la capacidad de investigar sobre las tecnologías móviles, las tecnologías necesarias en el desarrollo de esta, como el servidor y la base de datos, así como la comunicación entre ellos. También me ha aportado la capacidad de planificar y llevar la gestión del proyecto en unas fechas, de diseñarlo e implementarlo.

Por tanto, creo que he alcanzado los objetivos personales que había marcado. Es cierto que, con este proyecto, no he sido capaz de conocer todos los aspectos de la tecnología Android, algo que también considero que es muy difícil ya que la tecnología va cambiando continuamente, pero he conseguido las bases para poder seguir investigando y desarrollándome en este sector.

Planificación y metodología

Desde el inicio de este trabajo he intentado seguir la planificación aportada, aunque he ido realizando algunos cambios sobre la marcha, sobre todo en la fase de implementación. Dediqué más tiempo a esta fase en las semanas del 4, 11, 18 y 25 de Noviembre, adelantando las pruebas a la semana del 30 de Noviembre y dejando la semana del 2 de Noviembre para la documentación y ajustes en el código que pudieran ir saliendo de las pruebas de usuario. Esto me permitió entregar la tercera entrega a tiempo, incluso dejar los últimos cinco días para posibles correcciones encontradas.

Pienso que se ofrece poco tiempo para la implementación, la mayoría de ideas para el proyecto tenían parte del servidor y no solo se debe que diseñar e implementar la aplicación, sino también esta parte. Me hubiese gustado tener más tiempo en esta fase para poder realizar más funcionalidades y una correcta implementación de las ya existentes, pero creo que es suficiente para realizar al menos una primera versión funcional de la aplicación.

La metodología en cascada ha ayudado bastante a dividir la planificación y a poder seguir las etapas. A pesar de que el periodo de implementación fuese corto, realizar todo el diseño y análisis con detalle antes de implementar ayuda a que la implementación sea rápida y no se pierda tiempo en la toma de decisiones.

Por tanto, creo que la planificación aportada y la metodología elegida han ayudado a cumplir plazos de entrega y al desarrollo en sí del proyecto.

Trabajo a Futuro

Debido al escaso tiempo de implementación, la aplicación tiene bastantes aspectos a mejorar que se proponen cómo trabajo a futuro.

En primer lugar, el cambio de la tecnología del servidor. En este trabajo se ha elegido Firebase porque permitía la obtención de un API y almacenamiento de datos rápido y fácil. Su base de datos orientada a documentos no es la mejor opción para una aplicación de este estilo. Una mejor alternativa sería una base de datos relacional como MySQL, por ejemplo.

Se propone cómo trabajo a futuro el cambio de esta base de datos y, por tanto, también la creación de un API REST en un servidor.

A nivel de aplicación, también se han ido proponiendo diferentes funcionalidades a futuro lo largo de la memoria:

- En el registro, se debe comprobar que el correo electrónico pertenece al usuario a través del envío de un correo electrónico a su cuenta y un enlace de aceptación.
- Se debe incluir la opción de recordar contraseña.
- Se debe incluir la modificación del perfil de usuario: cambiar su correo electrónico y su contraseña.
- Se debe incluir la posibilidad de cambiar la plaza del usuario mediante la aplicación.
- Se propone también funcionalidad añadida: el cambio de plazas entre usuarios para solucionar el problema de las plazas grandes.
- Se propone la relación entre los usuarios que comparten plaza y el aviso mediante una notificación push al compañero, de tal forma que deba aceptar que su compañero va a liberar la plaza compartida.
- Se propone también una mejor gestión de las reservas

Todas estas funcionalidades no se han desarrollado por la complejidad que supone utilizar la base de datos de Firebase, se propone cómo cambio a futuro una vez la nueva Base de Datos y API Rest esté funcionando.

6. Glosario

App: Aplicación móvil.

Pop-Up: Ventana emergente.

Activity/Actividad: Componente de una aplicación Android que contiene una pantalla con la que los usuarios pueden interactuar.

Fragment/Fragmento: Componente de una aplicación Android que contiene una porción de funcionalidad o interfaz de una Actividad.

Progress bar: Componente visual de la aplicación móvil que sirve para mostrar al usuario que un determinado proceso se está llevando a cabo. Se suele utilizar para mantener al usuario en espera.

API: Interfaz de programación. Permite que los servicios se comuniquen unos con otros. En este proyecto, es utilizado para definir las llamadas existentes en el Servidor.

Login: Inicio de sesión de un usuario en la aplicación.

7. Bibliografía

URL: http://cv.uoc.edu/nplincampus
Fecha de consulta: 27/09/2019
Descripción: Contenido de las asignaturas del máster en desarrollo de aplicaciones móviles: Android I, Android II, Diseño de productos multidispositivo y Trabajo de final de máster.

URL: https://firebase.google.com/docs?hl=es-419
Fecha de consulta: 29/09/2019
Descripción: Creación de API y estructura de datos con Firebase

URL: https://www.youtube.com/watch?v=1vCe0RhfegA
URL: https://www.scl-consulting.com/contacto/
Fecha de consulta: 07/10/2019
Descripción: Aplicación aparcamiento App SAP Fiori

URL: https://ingsotfwarekarlacevallos.wordpress.com/2015/06/04/uml-casos-de-uso/
Fecha de consulta: 12/10/2019
Descripción: Creación de diagrama de casos de uso y relaciones

URL:https://www.lucidchart.com/documents/edit/744c2325-41e1-4178-889e-39a1debbe11c/0_0?beaconFlowId=F23902007A236912
Fecha de consulta: 12/10/2019
Descripción: Herramienta para crear diagrama de casos de uso online

URL:http://ocw.uc3m.es/ingenieria-informatica/diseno-de-software-avanzado/material-de-clase-1/04-Modelado_Basico_con_Casos_de_Uso.pdf
Fecha de consulta: 12/10/2019
Descripción: Creación de diagrama de casos de uso y relaciones

URL: https://firebase.google.com/docs/database/android/read-and-write
Fecha de consulta: 19/10/2019
Descripción: Obtener datos de Firebase Realtime Database con el sdk de Android

URL:https://ingsoftwarei2014.wordpress.com/category/comparacion-de-los-patrones-de-arquitectura-mvc-mv-vm-mvp/
Fecha de consulta: 26/10/2019
Descripción: Modelos de aplicaciones

URL:https://e-archivo.uc3m.es/bitstream/handle/10016/23025/TFG_Rafael-Jesus_Quintana_Francisco.pdf?sequence=1&isAllowed=y
Fecha de consulta: 14/11/2019
Descripción: Ejemplo para documentar pruebas

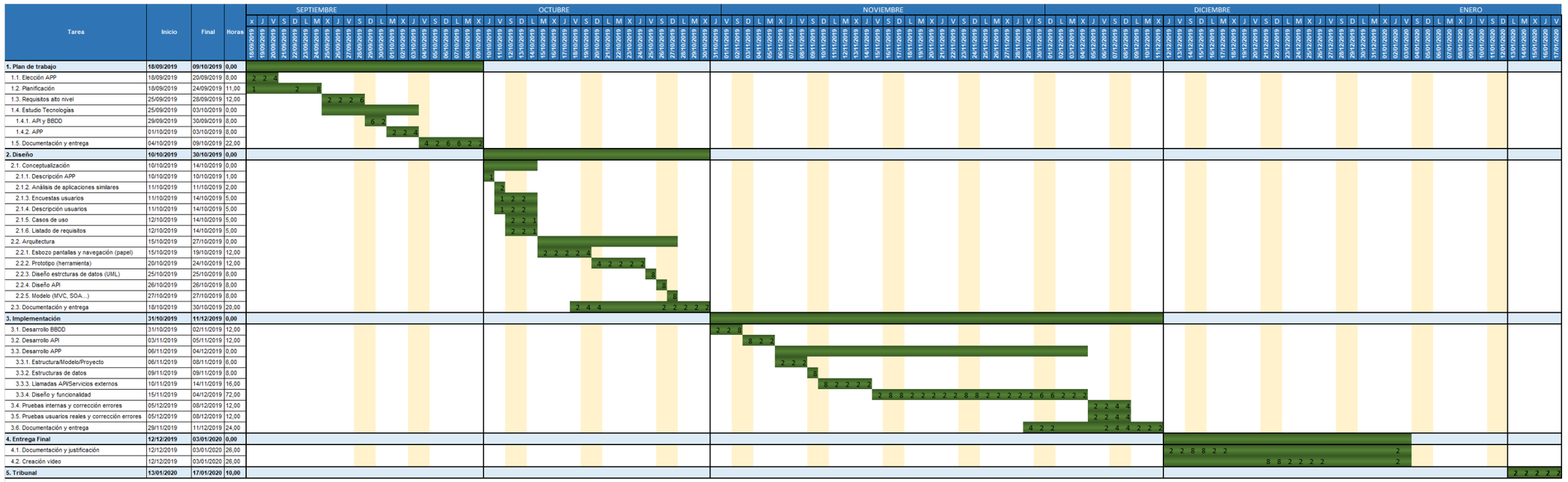
URL: https://stackoverflow.com/
Fecha de consulta: 1/11/2019 – 30/11/2019
Descripción: Foro de ayuda a la implementación. Se consultan distintos posts.

8. Anexos

Lista de anexos:

ANEXO 1. Planificación
ANEXO 2. Encuestas a usuarios sobre la aplicación
ANEXO 3. Respuestas de las encuestas de los usuarios
ANEXO 4. Prototipo

ANEXO 1. Planificación



Tarea	Inicio	Final	Horas
1. Plan de trabajo	**18/09/2019**	**09/10/2019**	**0,00**
1.1. Elección APP	18/09/2019	20/09/2019	8,00
1.2. Planificación	18/09/2019	24/09/2019	11,00
1.3. Requisitos alto nivel	25/09/2019	28/09/2019	12,00
1.4. Estudio Tecnologías	25/09/2019	03/10/2019	0,00
1.4.1. API y BBDD	29/09/2019	30/09/2019	8,00
1.4.2. APP	01/10/2019	03/10/2019	8,00
1.5. Documentación y entrega	04/10/2019	09/10/2019	22,00
2. Diseño	**10/10/2019**	**30/10/2019**	**0,00**
2.1. Conceptualización	10/10/2019	14/10/2019	0,00
2.1.1. Descripción APP	10/10/2019	10/10/2019	1,00
2.1.2. Análisis de aplicaciones similares	11/10/2019	11/10/2019	2,00
2.1.3. Encuestas usuarios	11/10/2019	14/10/2019	5,00
2.1.4. Descripción usuarios	11/10/2019	14/10/2019	5,00
2.1.5. Casos de uso	12/10/2019	14/10/2019	5,00
2.1.6. Listado de requisitos	12/10/2019	14/10/2019	5,00
2.2. Arquitectura	15/10/2019	27/10/2019	0,00
2.2.1. Esbozo pantallas y navegación (papel)	15/10/2019	19/10/2019	12,00
2.2.2. Prototipo (herramienta)	20/10/2019	24/10/2019	12,00
2.2.3. Diseño estrcturas de datos (UML)	25/10/2019	25/10/2019	8,00
2.2.4. Diseño API	26/10/2019	26/10/2019	8,00
2.2.5. Modelo (MVC, SOA...)	27/10/2019	27/10/2019	8,00
2.3. Documentación y entrega	18/10/2019	30/10/2019	20,00
3. Implementación	**31/10/2019**	**11/12/2019**	**0,00**
3.1. Desarrollo BBDD	31/10/2019	02/11/2019	12,00
3.2. Desarrollo API	03/11/2019	05/11/2019	12,00
3.3. Desarrollo APP	06/11/2019	04/12/2019	0,00
3.3.1. Estructura/Modelo/Proyecto	06/11/2019	08/11/2019	6,00
3.3.2. Estructuras de datos	09/11/2019	09/11/2019	8,00
3.3.3. Llamadas API/Servicios externos	10/11/2019	14/11/2019	16,00
3.3.4. Diseño y funcionalidad	15/11/2019	04/12/2019	72,00
3.4. Pruebas internas y corrección errores	05/12/2019	08/12/2019	12,00
3.5. Pruebas usuarios reales y corrección errores	05/12/2019	08/12/2019	12,00
3.6. Documentación y entrega	29/11/2019	11/12/2019	24,00
4. Entrega Final	**12/12/2019**	**03/01/2020**	**0,00**
4.1. Documentación y justificación	12/12/2019	03/01/2020	26,00
4.2. Creación video	12/12/2019	03/01/2020	26,00
5. Tribunal	**13/01/2020**	**17/01/2020**	**10,00**

Horas totales de cada apartado:	
1. Plan de trabajo (PEC 1)	69,00
2. Diseño	91,00
3. Implementación	174,00
4. Entrega Final	52,00
5. Tribunal	10,00
Total:	396,00

ANEXO 2. Encuestas a usuarios sobre la aplicación

Aplicación para gestión de plazas de aparcamiento en tu trabajo

*Obligatorio

Edad *

Tu respuesta

Marca y modelo de tu telefono personal *

Tu respuesta

Marca y modelo de tu telefono de empresa *

Tu respuesta

Existen trabajadores que incluyen la SIM del teléfono de empresa en su teléfono personal o redirigen las llamadas a su teléfono personal. ¿Es tu caso? *

- ◯ SI
- ◯ NO

¿Compartes plaza de aparcamiento? *

- ◯ SI
- ◯ NO

¿Cuántos días a la semana sueles aparcar en tu plaza? *

1	2	3	4	5
◯	◯	◯	◯	◯

¿Cuántos días a la semana sueles aparcar fuera del recinto? *

1 2 3 4 5

¿Tu plaza está libre algunos días porque tu/tu compañero no la usáis? *

- SI
- NO

Del 1 al 5, ¿Cómo de importante ves una aplicación que te permita reservar las plazas que no están ocupadas los días que no puedes aparcar en tu plaza? *

1 2 3 4 5

Del 1 al 5, ¿Usarías la aplicación para liberar tu plaza y que otros puedan usarla? *

1 2 3 4 5

Del 1 al 5, ¿Cómo de necesario ves que la aplicación también te ayude a gestionar la plaza con tu compañero de plaza? *

1 2 3 4 5

¿Qué te gustaría que tuviese esta aplicación? *

Tu respuesta

¿Tienes algo más que comentar...? Puedes hacerlo aquí:

Tu respuesta

Enviar

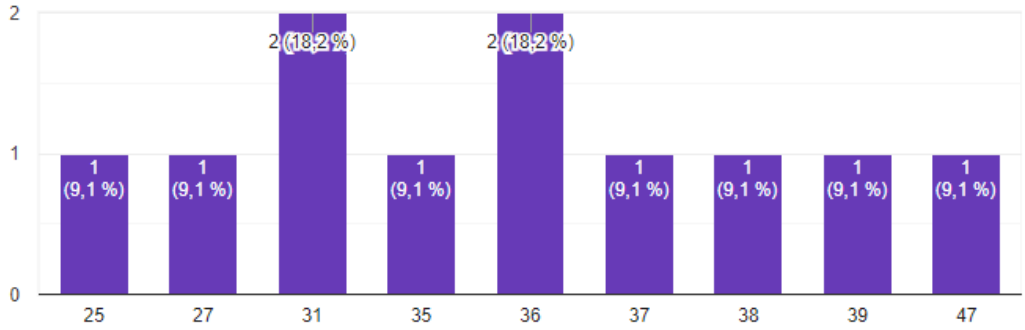
ANEXO 3. Respuestas de las encuestas de los usuarios

	Número de pregunta												
Respuesta		1	2	3	4	5	6	7	8	9	10	11	12	13
	Usuario 1	25	Huawei P9	Samsung Galaxy J6	NO	SI	3	2	SI	5	5	5	Me gustaría poder ver los días que hay plazas libres y poder ver cuándo me toca aparcar.	
	Usuario 2	36	Xiaomi	Samsung	SI	SI	1	5	SI	2	5	1	Uso moto para ir a trabajar. Solo aparco dentro los días de lluvia pero sería interesante saber si hay plazas libres los día que llueve y así no molestar a mi compañera.	
	Usuario 3	31	iPhone 6	Samsung	NO	SI	5	1	SI	1	5	1	La persona que comparte plaza conmigo no tiene coche, así que solo usaría la aplicación para que los demás pudiesen aparcar cuando yo no estoy	
	Usuario 4	36	No uso	Galaxy J6	NO	SI	5	1	SI	5	5	1	Que podamos ver los días que hay plazas libres	En mi caso, tengo alquilado parking con mi compañero fuera. Por eso, aparco todos los días dentro. Me gusta la idea de la aplicación porque quizá podríamos dejar la plaza de fuera y aparcar los dos dentro todos los días.
	Usuario 5	47	Samsung S8	Samsung	NO	SI	1	5	SI	5	5	1	Me gustaría poder cambiar mi plaza en la aplicación. Tengo un coche grande y no puedo aparcar en mi plaza porque no me cabe porque en mi plaza tengo una columna detrás	
	Usuario 6	27	Samsung Note	Samsung	NO	SI	2	3	SI	5	5	1	Poder saber cuándo hay plazas libres para poder aparcar. El parking siempre está vacío.	
	Usuario 7	39	__	galaxy j6	NO	SI	3	3	SI	5	5	1	reservar plazas vacias, informar de que mi plaza se queda libre, poder fichar al entrar al parking con el movil y no con la tarjeta	
	Usuario 8	38	Uso el de empresa	Samsung	NO	SI	3	3	SI	5	5	1	Me gustaría tener una aplicación dónde pueda reservar las plazas vacías y poner los días que se queda la mía libre porque hay veces que no podéis aparcar en la mía porque no sabéis que no vengo.	Espero que podamos usarla pronto! :)
	Usuario 9	31	Huawei	Galaxy	NO	NO	1	1	NO	5	5	5	...	no tengo plaza porque no tengo cohe, pero creo que es una buena idea
	Usuario 10	37	samsung galaxy j6	samsung galaxy j6	SI	SI	3	3	SI	5	5	2	La aplicación podría darnos la opción de compartir nuestra plaza cuando no la usamos y de reservar otras plazas cuando los demás están de vacaciones o en cliente. Estaría bien que sirviese para otros edificios, así tendremos plaza cuando vayamos a reuniones. También podría tener la opción de reservarlas para visitas. La forma de reservar parking para visitas externas es muy compleja y hay muy pocas plazas de visita.	
	Usuario 11	35	iphone	android	SI	SI	3	3	SI	5	5	3	poder reservar plazas	

Índice de preguntas	
Nº	Pregunta
1	Edad
2	Marca y modelo de tu telefono personal
3	Marca y modelo de tu telefono de empresa
4	Existen trabajadores que incluyen la SIM del teléfono de empresa en su teléfono personal o redirigen las llamadas a su teléfono personal. ¿Es tu caso?
5	¿Compartes plaza de aparcamiento?
6	¿Cuántos días a la semana sueles aparcar en tu plaza?
7	¿Cuántos días a la semana sueles aparcar fuera del recinto?
8	¿Tu plaza está libre algunos días porque tu/tu compañero no la usáis?
9	Del 1 al 5, ¿Cómo de importante ves una aplicación que te permita reservar las plazas que no están ocupadas los días que no puedes aparcar en tu plaza?
10	Del 1 al 5, ¿Usarías la aplicación para liberar tu plaza y que otros puedan usarla?
11	Del 1 al 5, ¿Cómo de necesario ves que la aplicación también te ayude a gestionar la plaza con tu compañero de plaza?
12	¿Qué te gustaría que tuviese esta aplicación?
13	¿Tienes algo más que comentar...? Puedes hacerlo aquí:

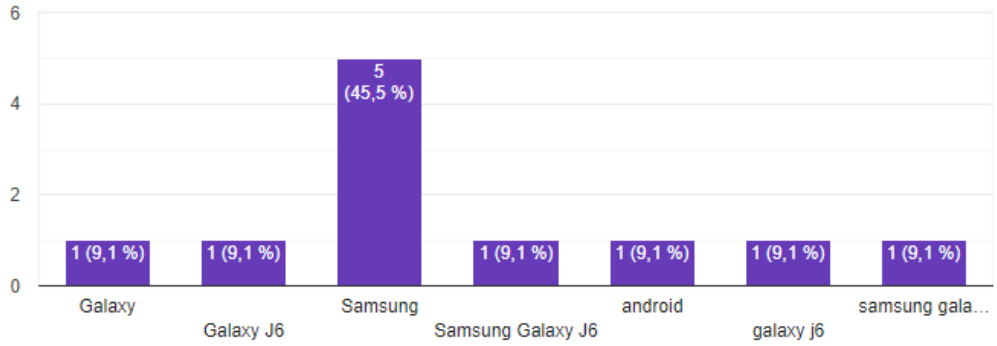
Edad

11 respuestas



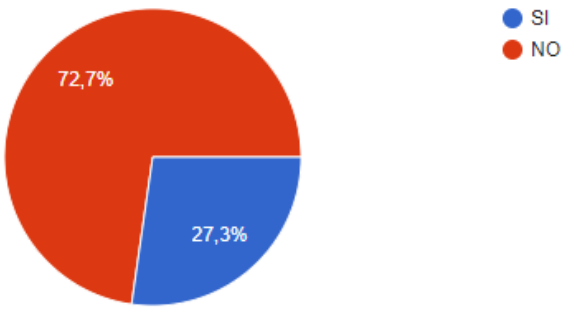
Marca y modelo de tu telefono de empresa

11 respuestas



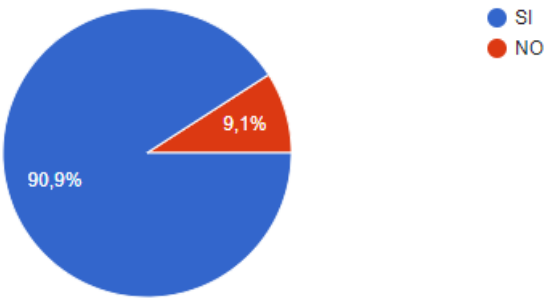
Existen trabajadores que incluyen la SIM del teléfono de empresa en su teléfono personal o redirigen las llamadas a su teléfono personal. ¿Es tu caso?

11 respuestas



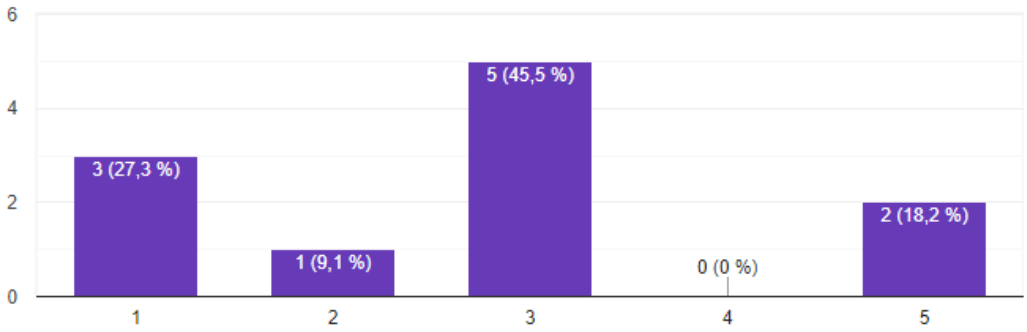
¿Compartes plaza de aparcamiento?

11 respuestas



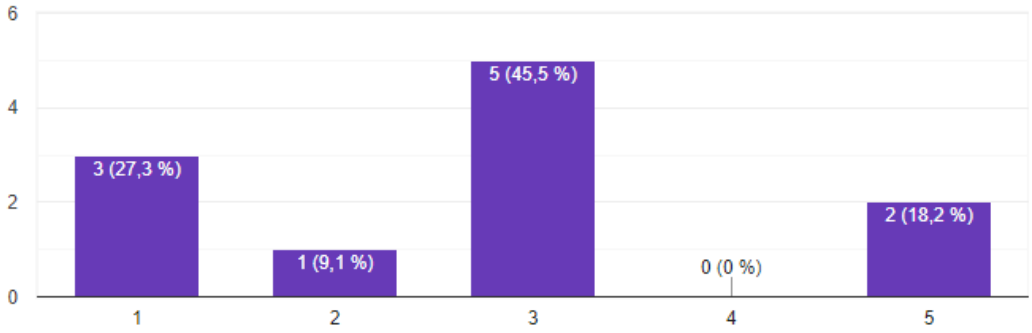
¿Cuántos días a la semana sueles aparcar fuera del recinto?

11 respuestas



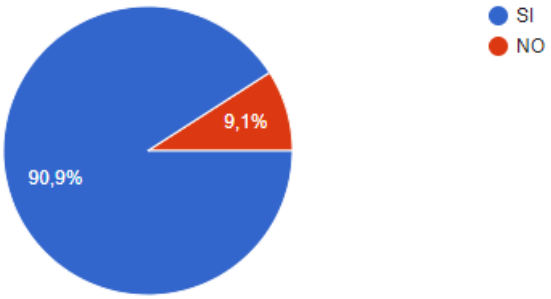
¿Cuántos días a la semana sueles aparcar en tu plaza?

11 respuestas



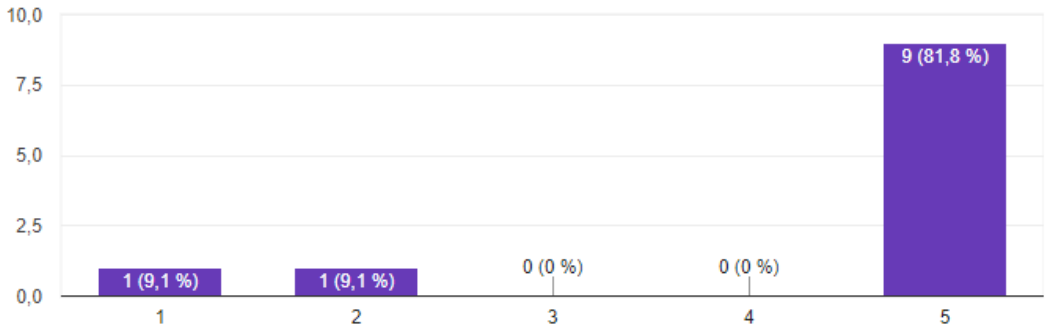
¿Tu plaza está libre algunos días porque tu/tu compañero no la usáis?

11 respuestas



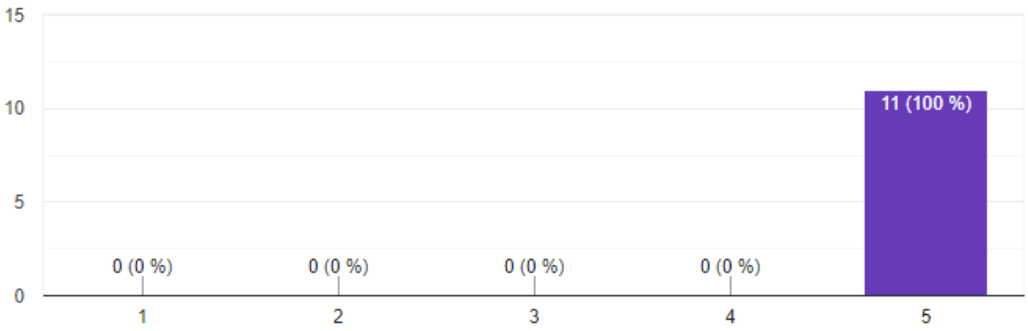
Del 1 al 5, ¿Cómo de importante ves una aplicación que te permita reservar las plazas que no están ocupadas los días que no puedes aparcar en tu plaza?

11 respuestas



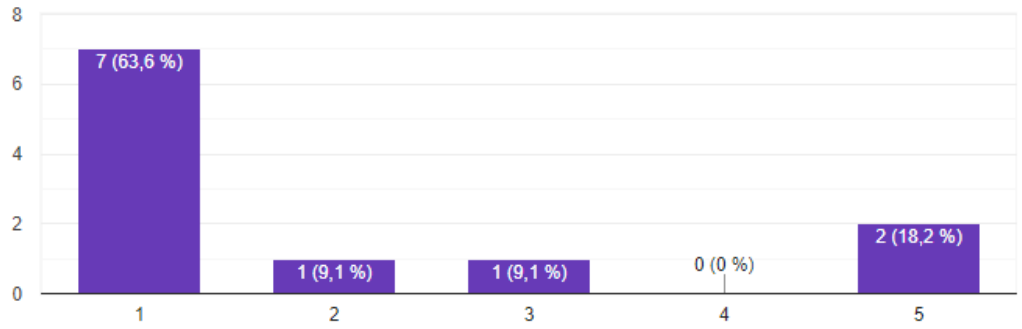
Del 1 al 5, ¿Usarías la aplicación para liberar tu plaza y que otros puedan usarla?

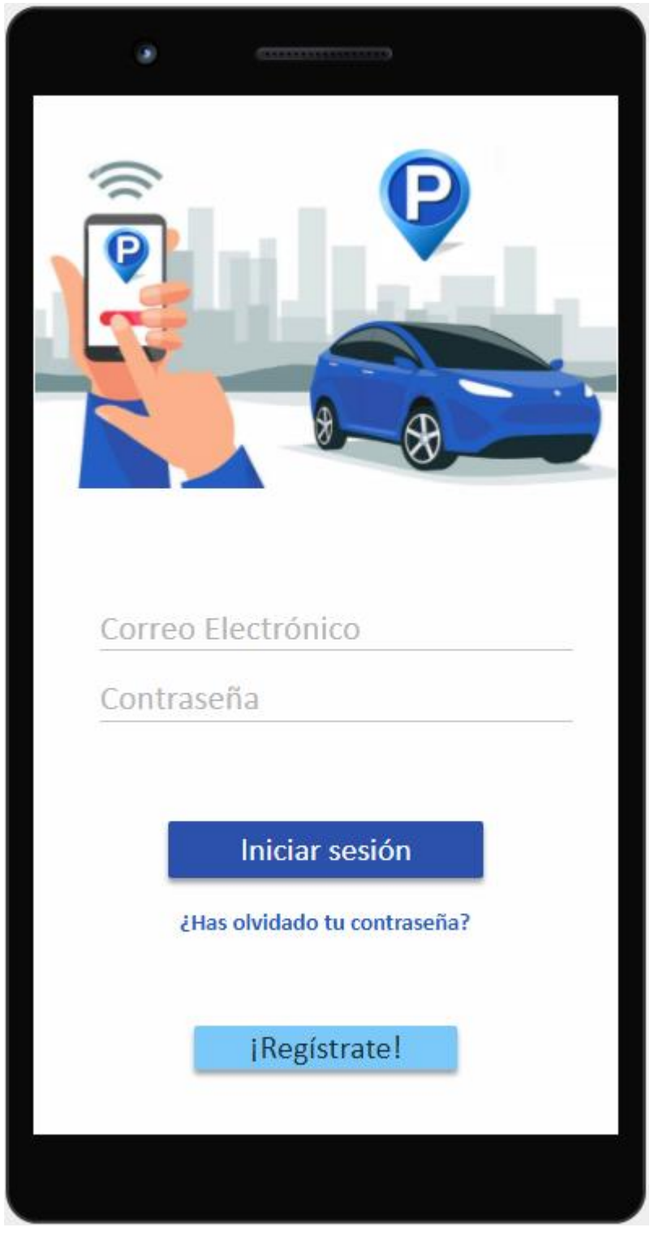
11 respuestas

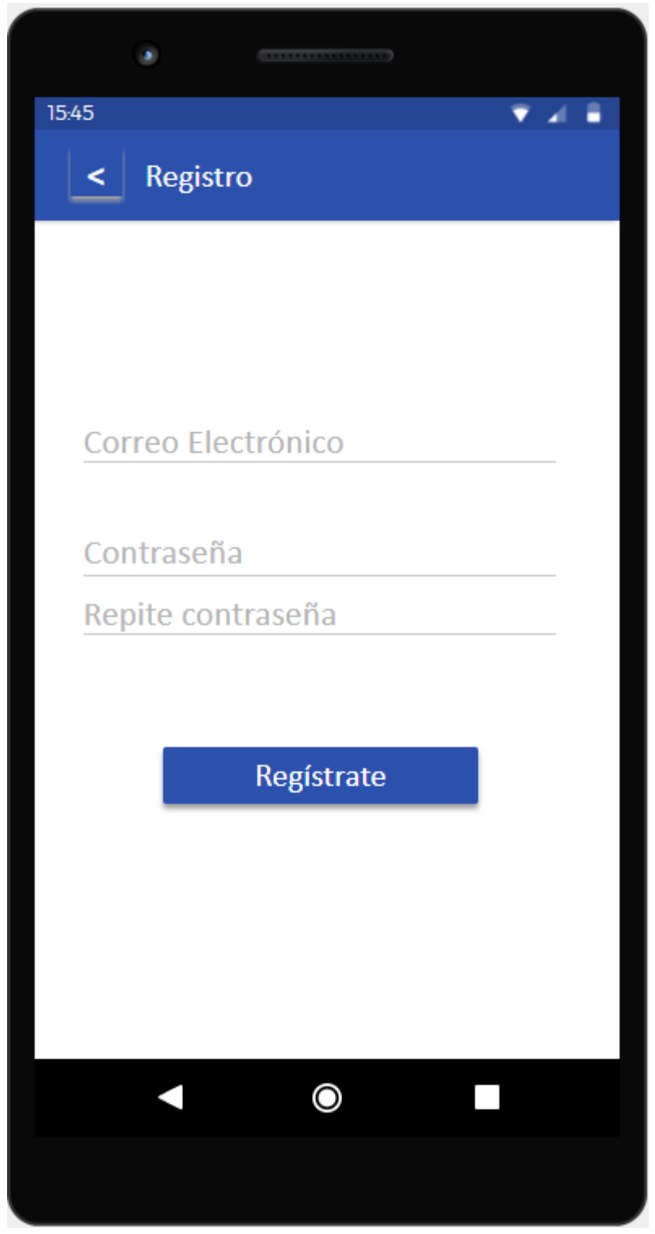


Del 1 al 5, ¿Cómo de necesario ves que la aplicación también te ayude a gestionar la plaza con tu compañero de plaza?

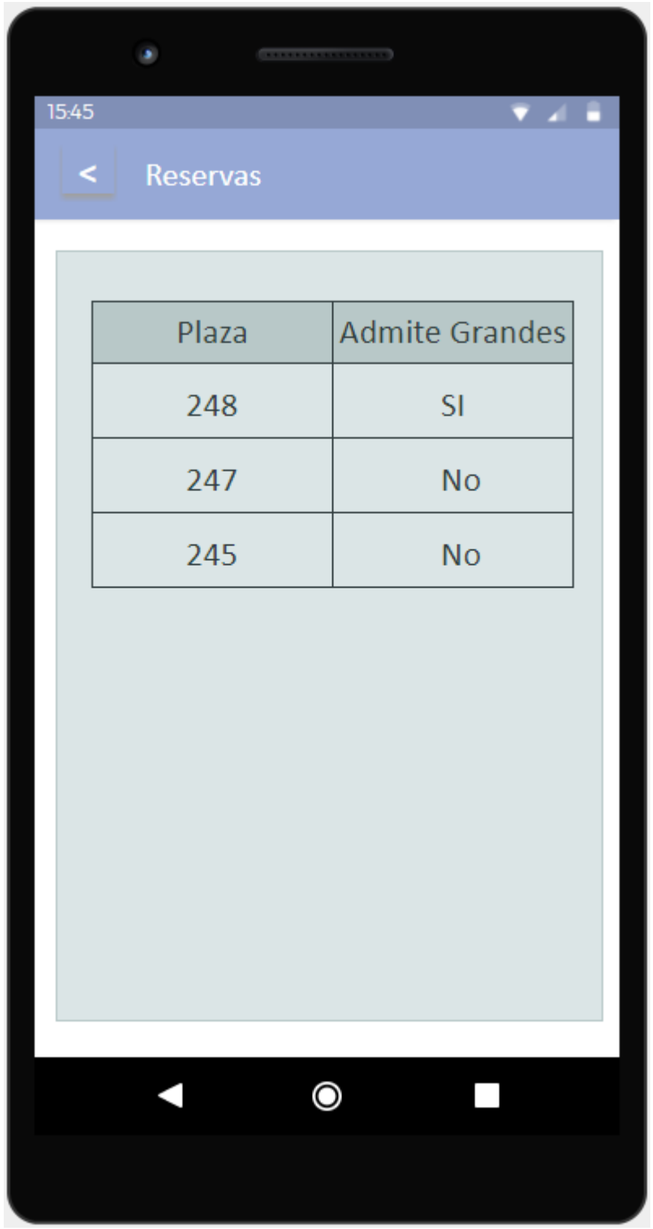
11 respuestas

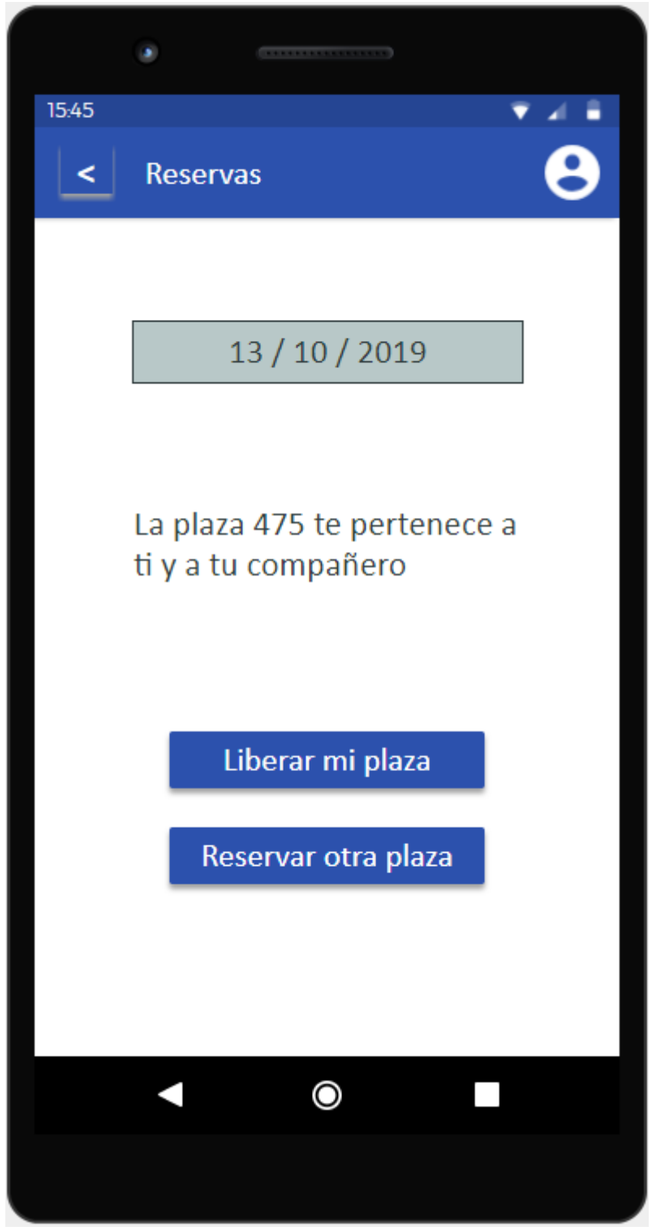



Correo Electrónico
Contraseña
Iniciar sesión
¿Has olvidado tu contraseña?
¡Regístrate!


15:45
Registro
Correo Electrónico
Contraseña
Repite contraseña
Regístrate


15:45
Reservas
Indica tu número de plaza:
Número de plaza
La plaza es apta
para coches grandes
Guardar


15:45
Reservas
Plaza
Admite Grandes
248
SI
247
No
245
No


15:45
Reservas
13 / 10 / 2019
La plaza 475 te pertenece a ti y a tu compañero
Liberar mi plaza
Reservar otra plaza