



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Desarrollo de una plataforma software para la gestión en tiempo real de la ocupación de parkings en zonas de alto impacto turístico

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Guardiola Pastor, Manuel

Tutor/a: Valderas Aranda, Pedro José

CURSO ACADÉMICO: 2022/2023

Resumen

Actualmente en muchas poblaciones existe un gran problema de masificación. Algunas poblaciones multiplican su población en verano debido a sus famosas playas, pero en cambio, no aumentan los recursos. Esta masificación provoca problemas tanto de contaminación como de pérdida de tiempo para aquellas personas que se dirigen a una playa y encuentren congestionada la carretera. Además del problema de la congestión de carreteras, ciertas playas disponen de parkings con un número de plazas limitado, por lo que llegar a una playa no garantiza que haya plazas libres.

El objetivo es crear una plataforma en la que los usuarios puedan consultar en tiempo real el estado de los parkings. Para ello se desarrollará una aplicación web donde los usuarios puedan consultar las plazas libres de dichos parkings y un lector de matrículas. De esta manera se situará el lector a la entrada y salida del parking con el fin de detectar cuando entra y sale un coche para poder aumentar o disminuir el número de plazas libres.

Palabras clave: parking, detector de matrículas, plataforma, metodología DCU, plazas libres

Resum

Actualment a moltes poblacions hi ha un gran problema de massificació. Algunes poblacions multipliquen la població a l'estiu a causa de les seues famoses platges, però en canvi no augmenten els recursos. Aquesta massificació provoca problemes tant de contaminació com de pèrdua de temps per a aquelles persones que es dirigeixen a una platja i troben congestionada la carretera. A més del problema de la congestió de carreteres, certes platges disposen de pàrquings amb un nombre de places limitat, de manera que arribar a una platja no garanteix que hi haja places lliures.

L'objectiu és crear una plataforma on els usuaris puguen consultar en temps real l'estat dels pàrquings. Per fer-ho, es desenvoluparà una aplicació web on els usuaris puguen consultar les places lliures d'aquests pàrquings i un lector de matrícules. D'aquesta manera se situarà el lector a l'entrada i a l'eixida del pàrquing per tal de detectar quan entra i quan ix un cotxe per poder augmentar o disminuir el nombre de places lliures.

Paraules clau: pàrquing, detector de matrícules, plataforma, metodologia DCU, places lliures

Abstract

Nowadays, in many populations there is a great problem of overcrowding. Some towns multiply their population in summer due to their famous beaches, but on the other hand, resources do not increase. This overcrowding causes problems of both pollution and loss of time for those people who go to a beach and find the road congested. In addition to the problem of road congestion, certain beaches have car parks with a limited number of spaces, so arriving at a beach does not guarantee that there are free spaces.

The objective is to create a platform where users can check the status of the car parks in real time. To do this, a web application will be developed where users can check the free spaces in said car parks and a license plate reader. In this way, the reader will be located at the entrance and exit of the car park in order to detect when a car enters and leaves in order to increase or decrease the number of free spaces.

Key words: parking, license plate detector, platform, DCU methodology, free spaces

Índice general

Índice general	V
Índice de figuras	VII
<hr/>	
1 Introducción	1
1.1 Motivación	1
1.2 Objetivos	2
1.3 Metodología	2
1.4 Estructura de la memoria	2
2 Estado del arte	5
2.1 App Parkings (UPV)	5
2.2 Plataforma València al minut	5
2.3 App Parking Madrid	6
2.4 Conclusión	7
3 Análisis de necesidades	9
3.1 Cuestionario	9
3.2 Definición de persona	10
3.3 Definición de escenarios	11
4 Análisis conceptual y diseño	13
4.1 Diagrama de clases	13
4.2 Modelo de Base de Datos	14
4.3 Boceto de la plataforma	15
5 Desarrollo de la solución	21
5.1 Arquitectura	21
5.2 Contexto tecnológico	22
5.3 Código de la Aplicación Web	26
5.4 Código del lector de matrículas	31
6 Producto desarrollado	35
6.1 Alfonso registra un parking	35
6.2 Alfonso consulta el estado de un parking	40
6.3 Otras capturas	42
7 Validación	45
8 Conclusión	47
9 Trabajo futuro	49
Bibliografía	51
<hr/>	
Apéndices	
A Detalle del cuestionario	53
B Objetivos de Desarrollo Sostenible	59

Índice de figuras

2.1	Captura de la plataforma València al minut (I)	6
2.2	Captura de la plataforma València al minut (II)	6
2.3	Captura de la App Parking Madrid (I)	7
2.4	Captura de la App Parking Madrid (II)	7
3.1	Información de la persona modelo	10
4.1	Diagrama de clases UML	14
4.2	Modelo de la base de datos	15
4.3	Prototipo pantalla inicial	16
4.4	Prototipo pantalla inicial con menú desplegado	16
4.5	Prototipo formulario de inicio de sesión	17
4.6	Prototipo formulario de registro	17
4.7	Prototipo pantalla inicial con sesión iniciada	18
4.8	Prototipo pantalla inicial con sesión iniciada y menú desplegado	18
4.9	Prototipo formulario de registro de parkings	19
5.1	Arquitectura de la plataforma	22
5.2	Código que establece conexión con la base de datos con mysqli	26
5.3	Código que establece conexión con la base de datos con PDO	27
5.4	Código que almacena en la base de datos el nombre y contraseña de un usuario	27
5.5	Código que comprueba si hay una sesión iniciada	28
5.6	Código que crea la tabla de parkings (I)	28
5.7	Código que crea la tabla de parkings (II)	28
5.8	Código que crea la tabla de parkings (III)	28
5.9	Código de la función highlightCell()	30
5.10	Código de creación del formulario de registro de parkings	30
5.11	Detección de contornos de la imagen	31
5.12	Selección del contorno de la matrícula	31
5.13	Escritura de la matrícula leída	32
5.14	Comprobación de la existencia de una matrícula	32
5.15	Actualización del número de plazas libres	32
5.16	Resultado de la ejecución del lector de matrículas	33
6.1	Pantalla inicial	35
6.2	Pantalla inicial con el menú desplegado	36
6.3	Pantalla de inicio de sesión	36
6.4	Mensaje error	37
6.5	Pantalla principal con sesión iniciada (I)	37
6.6	Pantalla principal con sesión iniciada con menú desplegado	38
6.7	Pantalla de registro de un parking	39
6.8	Pantalla principal con sesión iniciada (II)	39
6.9	Pantalla inicial móvil	40

6.10 Pantalla inicial móvil con menú desplegado	40
6.11 Pantalla de inicio de sesión móvil	41
6.12 Pantalla error inicio de sesión móvil	41
6.13 Pantalla principal con sesión iniciada móvil (I)	42
6.14 Pantalla principal con sesión iniciada móvil (II)	42
6.15 Pantalla principal con sesión iniciada móvil y menú desplegado	43
6.16 Pantalla con lista de parkings favoritos móvil	43
6.17 Pantalla de registro móvil	44

CAPÍTULO 1

Introducción

Como se sabe, hay bastantes localidades en las que en verano aumentan considerablemente su población. Esto afecta directamente al tráfico, por ejemplo, ya que algunas de sus carreteras y parkings están pensadas para cierto número de vehículos y al aumentar el número de estos se saturan.

Esto mismo le ocurre a Xàbia, una localidad situada en el noreste de la provincia de Alicante. Según el Instituto Nacional de Estadística (INE), en 2022 Xàbia tenía una población de 28.731[1], pero según una noticia[2] del 2019, la población en ese verano registró un aumento del 331 % y alcanzó los 116.000 habitantes. Al tratarse de una noticia de hace ya algunos años, se puede llegar a pensar que en el año 2022 incluso supero esos 116.000 habitantes. Esto supone un problema para la población que viva en la localidad y los visitantes que reciba.

Este es el caso de playas y calas como, por ejemplo, la Platja de la Barraca y la Platja de la Granadella. Al tener un número reducido de plazas de parking, desde hace ya unos años, se ha instalado una barrera a cierta distancia de algunas calas para evitar que los visitantes aparquen en zonas donde está prohibido. Al ser un número bastante reducido, estos parkings gratuitos se llenan con facilidad, obligando a los visitantes que se encuentren la barrera bajada a aparcar a unos 15 minutos andando. Esto supone tener que andar unos 15 minutos para ir a la playa, y donde viene el problema para la mayoría de gente, otros 15 minutos por una cuesta bastante empinada para volver al coche.

Además de la lejanía del aparcamiento, otro problema evidente y probablemente el más importante es el resultado de la masificación de vehículos, la contaminación y el deterioro de ciertas zonas naturales provocado por el mal estacionamiento de algunos vehículos. Esta masificación de vehículos también provoca continuamente una congestión vehicular por las carreteras de Xàbia que no están preparadas para albergar tal cantidad de vehículos.

1.1 Motivación

Como habitante de Xàbia, tengo la suerte de poder disfrutar las playas durante todo el año, pero soy consciente de que si en verano quiero acudir a alguna de las playas, tengo que acudir muy temprano, y aun así, existe la posibilidad de encontrarme la barrera bajada, lo que te obliga a aparcar bastante lejos. Por lo que la única manera de saber si se puede bajar a la playa o no es yendo.

Teniendo esto en cuenta, me parece interesante hacer una plataforma software que resulte útil para aquellas personas que quieran acudir a estas playas, y que informe en tiempo real del estado de los parkings sin la necesidad de tener que acudir hasta donde

se encuentra la barrera para saber si podrán aparcar más cerca o no y poder decidir a qué playa quieren o les conviene ir.

1.2 Objetivos

El propósito principal de este Trabajo de Fin de Grado es diseñar y crear una plataforma software para que los usuarios puedan consultar el estado en que se encuentran los parkings registrados en el sistema y decidan al que quieren acudir.

Las características que debe ofrecer la aplicación son las siguientes:

- **Objetivos generales:**
 - Permitir la consulta del estado de los parkings dados de alta.
 - Desarrollar un módulo con el cual se detecte una matrícula a partir de una imagen para que posteriormente se descuente o se añada una plaza libre en función de si la ha detectado en la entrada o la salida del parking.
- **Objetivos específicos:**
 - Permitir a los usuarios ver los parkings en forma de listado.
 - Permitir registrarse a los usuarios en nuestra plataforma.
 - Permitir dar de alta un parking a los usuarios registrados.
 - Permitir a los usuarios registrados marcar los parkings que deseen como sus favoritos.

1.3 Metodología

Para el correcto desarrollo de nuestra plataforma se hará uso de la metodología de Desarrollo Centrado en el Usuario (DCU), ya que lo verdaderamente difícil es desarrollar una plataforma útil y usable y esta metodología nos ayudará debido a que se centra en las necesidades y requerimientos del usuario final.

Para ello empezaremos realizando un cuestionario cualitativo por el cual se averigüe ciertos datos, como, por ejemplo, como suelen los usuarios ir a las playas, si utilizan el navegador web, los meses en los que suelen ir a la playa, etc.

Con los resultados que obtengamos del cuestionario, se creará un usuario de la plataforma usando la Técnica Persona y posteriormente se crearán posibles escenarios mediante el Análisis de Escenario. Una vez hecho esto llegaremos a la fase de diseño, en la que toda la información que se ha recopilado nos ayudará a crear un prototipo o *mockup*.

Una vez se haya creado el prototipo, procederemos a desarrollarlo. Esto será la fase de implementación de nuestra plataforma y a la cual, una vez terminada esta fase, someteremos a una evaluación final.

1.4 Estructura de la memoria

Con el fin de basar este trabajo en la metodología de Desarrollo Centrado en el Usuario como se ha mencionado en el apartado anterior, la presente memoria se estructurará en diferentes apartados en los que se detallará el proceso que se ha llevado a cabo para conseguir desarrollar una plataforma útil y usable para el usuario final.

La memoria se estructurará en los apartados que se detallan a continuación:

1. **Introducción:** pequeña introducción al problema que se va a tratar y de la solución propuesta.
2. **Estado del arte:** se comentará el contexto tecnológico actual y se documentarán algunas aplicaciones relacionadas a la que se ha propuesto.
3. **Análisis de necesidades:** este apartado se centrará en estudiar las necesidades que tienen los usuarios y se explicará cómo se ha obtenido la información.
4. **Análisis Conceptual y Diseño:** una vez definidas las necesidades que tienen los usuarios, empezaremos diseñando la estructura que tendrá la base de datos, así como los prototipos de nuestra plataforma.
5. **Desarrollo de la solución:** en este apartado empezará la fase de implementación. Se explicará que arquitectura tendrá nuestra plataforma y se explicará que herramientas y que lenguajes se han utilizado en el desarrollo de la solución. También en este apartado se mostrarán algunos fragmentos de código que se consideren representativos.
6. **Producto desarrollado:** cuando ya se haya diseñado y desarrollado la plataforma, en este apartado se procederá a mostrarla.
7. **Validación:** en este apartado un usuario real realizará diferentes tareas con el fin de evaluar si la plataforma cumple o no con los objetivos y expectativas y se intentarán detectar posibles mejoras o fallos de esta.
8. **Conclusión:** una vez se hayan realizado todos los pasos anteriores, se expondrán una serie de conclusiones que se extraerán del trabajo realizado y una opinión personal.
9. **Trabajo futuro:** en este último apartado se plantearán posibles mejoras y adiciones con tal de mejorar el proyecto.

CAPÍTULO 2

Estado del arte

A día de hoy, existen muchas aplicaciones y plataformas donde se puede consultar el estado de ciertos parkings. A continuación, se muestran algunas alternativas

2.1 App Parkings (UPV)

Parkigns es una aplicación creada en el año 2016 por el Área de Sistemas de la Información y las Comunicaciones de la UPV y está disponible tanto para Android como para IOS.

Esta aplicación permite consultar la ubicación de los parkings que pertenecen a la UPV, consultar el estado en tiempo real de los parkings y también permite añadir parkings como favoritos para facilitar posteriormente su consulta.

La aplicación permite seleccionar el campus deseado (Vera, Gandía o Alcoy) y también el tipo de usuario (Alumno, PAS y PDI, otros miembros de la UPV y externo a la UPV).

2.2 Plataforma València al minut

Valencia al minut es una plataforma desarrollada por el Ayuntamiento de València a la cual se puede acceder desde un navegador web, aunque también se puede acceder mediante la App AppValència disponible para Android e IOS.

Esta plataforma tiene información en tiempo real de todo tipo, desde el estado de las carreteras, la calidad del aire, la ocupación de las plazas de parking de movilidad reducida, la ubicación de ecoparques móviles, la cantidad de bicicletas que están disponibles y las que están ocupadas, cámaras de tránsito, gráficos sobre la contaminación del aire, etc.

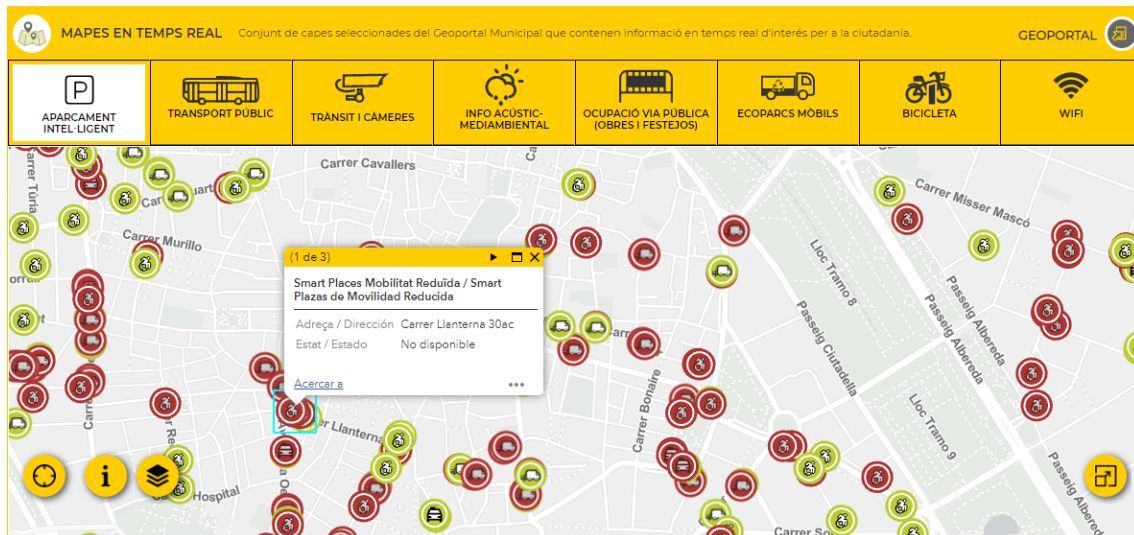


Figura 2.1: Captura de la plataforma València al minut (I)

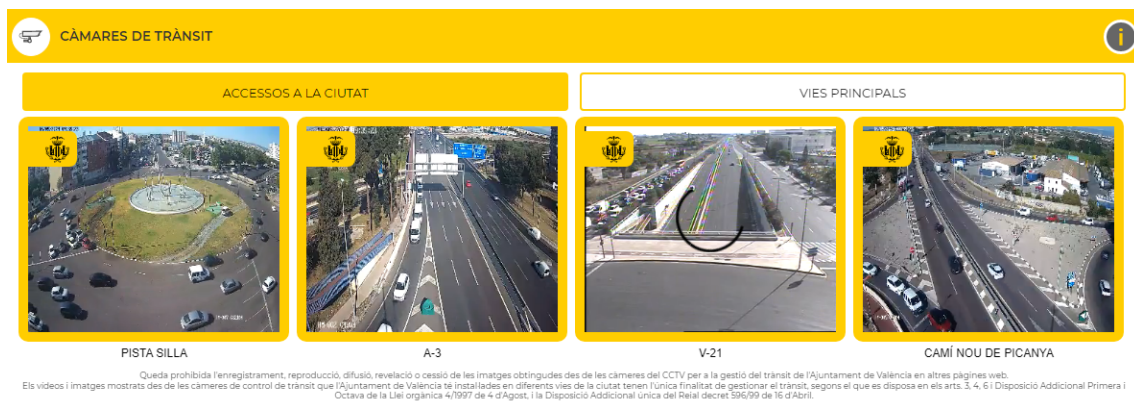


Figura 2.2: Captura de la plataforma València al minut (II)

2.3 App Parking Madrid

Parking Madrid es una aplicación desarrollada por EMT Madrid. Esta aplicación está disponible tanto para Android como para IOS. Su primera versión se desarrolló en el año 2015.

La aplicación ofrece mucha información sobre los parkings, como, por ejemplo, su ubicación, plazas disponibles, tarifas y formas de pago, servicios adicionales del parking, altura máxima y si dispone de punto de recarga para vehículos eléctricos, etc. También ofrece la opción de consultar la ubicación de distintos puntos de interés, como pueden ser las bocas de metro, el teleférico de Madrid, instalaciones deportivas, etc.

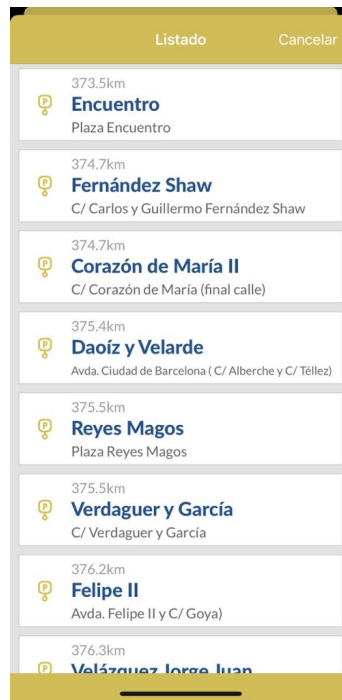


Figura 2.3: Captura de la App Parking Madrid (I)

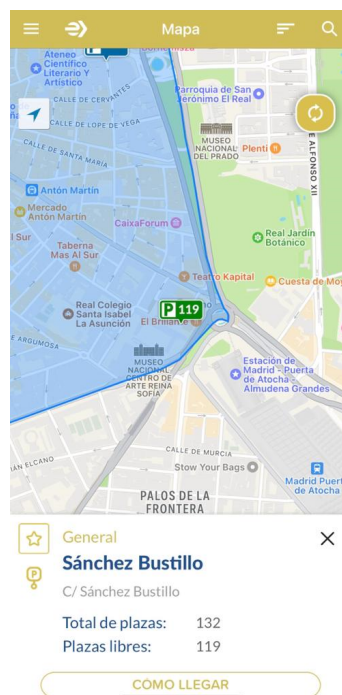


Figura 2.4: Captura de la App Parking Madrid (II)

2.4 Conclusión

Como se puede observar, hay aplicaciones que realizan funciones similares a las que buscamos. La aplicación de la UPV sería la más parecida en cuanto a funcionalidad, pero en este caso no se centraría solo en una organización. En cuanto a la plataforma Valencia

Al Minut y Parking Madrid se observa lo mismo, solo se centra en una localidad, València y Madrid respectivamente.

Por esto, desarrollaremos una plataforma a la que se pueda acceder desde cualquier dispositivo que tenga un navegador web y no se necesite descargar ninguna aplicación. Así mismo, el objetivo de la plataforma será contener todos los parkings que se quieran adherir y no restringirla a un municipio en concreto. Por estas razones, aunque nuestra plataforma tenga bastante competencia, a medida que vaya creciendo irá haciendo frente al mercado actual.

CAPÍTULO 3

Análisis de necesidades

A la hora de crear una plataforma, aplicación o página web es conveniente realizar un análisis de necesidades. Este análisis ayudará a conocer un poco más al usuario final, es decir, que es lo que quiere conseguir mediante el uso de nuestra plataforma, en que contexto va a utilizarla, qué conocimientos tiene ya el usuario (en cuanto al uso de algunos dispositivos se refiere), qué dispositivos y con qué frecuencia los usa, etc.

Para obtener toda esta información se ha decidido usar cuestionarios mediante la herramienta *Formularios de Google*, ya que de esta manera se daba un poco de libertad al usuario para contestar cuando y donde pudiera y además es una forma rápida y sencilla de recopilar la información.

3.1 Cuestionario

Con el fin de recopilar información para conocer mejor a los usuarios y así poder realizar un perfil de persona que vaya a utilizar nuestra plataforma, como se ha comentado anteriormente, se ha decidido usar la herramienta *Formularios de Google*.

El cuestionario se ha distribuido y contestado por un total de 13 personas, número que consideramos suficiente para llevar a cabo las posteriores tareas. Cabe destacar que se ha intentado distribuir y llegar a personas de diferente rango de edad.

Algunas de las preguntas que se han realizado en el cuestionario se centran en datos demográficos y otras se especializan más en conocer las necesidades y capacidades que tiene el usuario.

Aunque a continuación solo aparecen algunas preguntas, en el Apéndice A se muestran todas las preguntas con sus respectivas respuestas obtenidas.

- ¿Cuándo sueles ir a la playa?

Aunque para esta pregunta hubiera sido conveniente recibir más cantidad de respuestas, se realiza para saber si realmente hay un problema debido a la concentración de gente en ciertos meses del año.

- En caso de tener que aparcar a unos 15-20 min andando de la playa, ¿Qué haces?

Con esta pregunta queremos conocer que haría el usuario si cuando llega a la playa la barrera automática del parking esta bajada y debe aparcar a una distancia relativamente lejana de la playa.

- ¿Si pudieras consultar el estado del parking de las playas a las que vas antes de ir, lo harías?

Con esta pregunta, y puede que la más importante de nuestro cuestionario, se quiere conocer si teniendo esta herramienta, como es nuestra plataforma, el usuario en cierto momento la usaría o no.

Una vez obtenidos y analizados los resultados de nuestra encuesta, se puede decir que el rango de edad del usuario de nuestra plataforma esta entre 35 y 55 años, sería un hombre que usa diariamente un dispositivo con conectividad a internet. También se puede decir que va a la playa mayoritariamente en los meses de julio y agosto, que normalmente no va a la misma playa, que suele ir en coche y que si pudiera consultar el estado del parking de las playas, lo consultaría.

3.2 Definición de persona

Una vez obtenidas las respuestas al cuestionario, podemos proceder a definir una persona mediante la Técnica persona. Según la Universitat Oberta de Catalunya (UOC), *“La técnica de personas se desarrolló originalmente como una técnica de ayuda al diseño, propuesta por Alan Cooper en su aproximación al desarrollo de sistemas que tienen en cuenta al usuario”*[4].

Con toda la información obtenida, y mediante la Técnica persona se ha creado la siguiente persona:



ALFONSO OLMOS

BIOGRAFÍA

- Hombre
- 50 años
- Vive con su mujer en Jávea
- Es concejal en el ayuntamiento
- Tiene vacaciones en agosto

OBJETIVOS

- No quiere perder el tiempo buscando aparcamiento cuando vaya a la playa
- Aprovechar el tiempo de trabajo al máximo
- Consultar información sobre el estado de los parkings

HABILIDADES TECNOLÓGICAS

- Usa el dispositivo móvil y el ordenador a diario
- No tiene dificultades a la hora de navegar por internet

Figura 3.1: Información de la persona modelo

El objetivo de esta definición de persona es no olvidarnos de los objetivos y necesidades que tiene y una vez que ya se ha definido el usuario potencial de nuestra plataforma podemos avanzar a la construcción de escenarios.

3.3 Definición de escenarios

Según la UOC, “Un escenario es una técnica de modelado que consiste en describir de manera narrativa cómo utiliza un usuario el producto para lograr sus objetivos”[5]. El usuario que utilizará nuestra plataforma para lograr sus objetivos será la persona que se creó en el apartado anterior.

Dependiendo del momento en el que se realicen los escenarios, se pueden distinguir 3 tipos de escenarios:

- Escenarios de contexto
Este tipo de escenario son los que se crean antes de haber definido que funciones realiza nuestra plataforma.
- Escenario principal
Este es el escenario que usaremos. En este tipo de escenario se define al completo la interacción de un usuario (en nuestro caso la persona del apartado anterior) con nuestra plataforma.
- Escenario de validación
Estos escenarios normalmente se usan para comprobar que un sistema cumple con todas las necesidades que tienen los usuarios.

Una vez definidos los tipos de escenario, se van a definir los escenarios principales de uso:

- **Alfonso registra un parking**
Son las 9 de la mañana de un día laborable para Alfonso. Alfonso, que es concejal de playas del ayuntamiento de Xàbia, recibe una solicitud por parte de un ciudadano de agregar el parking de la Platja de la Barraca a la plataforma **PlayAparca**. Como Alfonso ya tiene el ordenador encendido, abre el navegador web y entra en la página de PlayAparca. Aquí inicia sesión y selecciona del menú la opción *Registrar parking*, donde rellena un formulario con los datos del parking y lo da de alta.
- **Alfonso consulta el estado de un parking**
Son las 10:45 del día 1 de agosto. Es el primer día de vacaciones de Alfonso y quiere ir con su pareja a la Platja de la Barraca a bañarse. Como no quiere perder el tiempo yendo a una playa en la que no pueda aparcar cerca, mientras su mujer conduce, él coge el teléfono móvil y accede a la web **PlayAparca**. Una vez dentro se fija en la Platja de la Barraca y consulta las plazas que quedan libres en el parking de esta playa, como ve que quedan pocas libres, decide consultar cuantas plazas libres quedan en el parking de la Platja de la Granadella y al ver que en este parking hay muchas libres, deciden ir a esta playa. A las 12:30 mientras están volviendo al coche, como a Alfonso y su pareja les ha gustado la playa accede a la web, inicia sesión y añade este parking a favoritos.

CAPÍTULO 4

Análisis conceptual y diseño

4.1 Diagrama de clases

Aunque la plataforma tiene una estructura interna no demasiado compleja, suele ser conveniente realizar un diagrama de clases que ofrezca una visión global. El uso de estos diagramas ofrece una serie de ventajas y alguna de ellas son:

- Como se ha mencionado anteriormente, ayuda a tener una visión global del proyecto.
- Expresar visualmente las necesidades específicas de un sistema.

Como se aprecia en la siguiente imagen, el diagrama de clases de la plataforma *PlayAparca* tiene 2 clases, los **usuarios** y los **parkings**.

Los usuarios pueden estar registrados o no y aun así pueden consultar el estado de los parkings. La clase usuario almacena el nombre de usuario y la contraseña de inicio de sesión y la clase parking almacena el nombre, una descripción, la dirección del parking, las plazas totales que tiene el parking y las plazas que actualmente están libres.

Entre estas 2 clases hay una relación, ya que los parkings solo se podrán registrar por usuarios que hayan iniciado sesión, aunque el parking no tiene ningún campo que contenga quien ha registrado ese parking. Los usuarios que no estén registrados pueden consultar los parkings que hay dados de alta en la plataforma, pero no podrán registrar ninguno.

Para realizar el siguiente diagrama se ha hecho uso de la herramienta online *Lucidchart*[6].

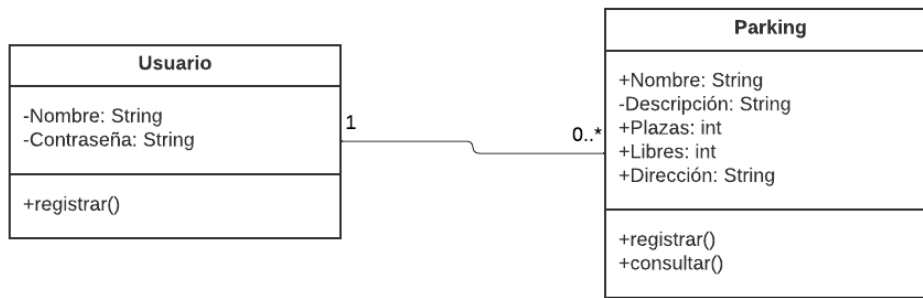


Figura 4.1: Diagrama de clases UML

4.2 Modelo de Base de Datos

Una vez hecho el diagrama de clases del apartado anterior, procedemos a diseñar las tablas de la base de datos. En la imagen que hay al final de este apartado, se muestran el esquema de la base de datos.

Para nuestra base de datos, se han creado 3 tablas, una de ellas para almacenar los usuarios que se registran con sus respectivas contraseñas, otra para almacenar los parkings y los datos de estos y otra para almacenar los parkings que los usuarios marcan como sus favoritos.

En la tabla **parkings** hay una columna llamada **id** que es la clave principal de ella. Al ser la clave principal, no podrá haber en la tabla otras entradas con el mismo id, por lo tanto, este campo será único y no podrá ser nulo. Para facilitar el uso e inserción de estas tablas, se ha hecho que esta columna sea autoincrementable.

El **id** de la tabla **parkings** nos servirá para poder identificar con un número único a cada parking, ya que a lo largo de la vida de la plataforma puede que en algún momento haya 2 parkings que tengan el mismo nombre. Para evitar que haya 2 entradas en la tabla para el mismo parking se ha establecido la columna **Dirección** como campo único. Las columnas **Plazas** y **Libres** se definirán como columnas de tipo INT.

La tabla **users** contendrá solamente 2 columnas. La columna **Nombre** será la clave principal de esta tabla y será única para evitar problemas y por lo que no tendremos más de 1 usuario registrado en nuestra plataforma con el mismo nombre. En cuanto a la columna **Contraseña** no sería conveniente almacenar las contraseñas de los usuarios en texto plano y lo más seguro sería utilizar una función hash, utilizar la técnica de "salting", etc. para que en caso de ataque a nuestra base de datos estas contraseñas no se filtren tan fácilmente. A pesar de ello, en este caso no se considera necesario.

La tabla **favoritos** también contendrá solamente 2 columnas. La primera de ellas será **id_usuario**. En esta columna se almacenará el id de los usuarios, que en este caso es el nombre. Esto es porque, ya que no habrá ningún nombre de usuario repetido en nuestra base de datos, se establece el nombre como id para simplificar la tabla. Y la segunda columna será **id_parking**. En este caso sí que almacenara el id de los parkings que cada usuario establezca como sus favoritos.



Figura 4.2: Modelo de la base de datos

Como se aprecia en la Figura 4.2, existe una relación entre las tablas. Las 2 columnas existentes de la tabla favoritos (`id_usuario` e `id_parking`) son claves foráneas que se corresponden a la columna `Nombre` de la tabla `users`, y a la columna `id` de la tabla `parkings` respectivamente. Cabe destacar que se ha establecido como acción ante cualquier actualización o eliminación de datos de las tablas de `parkings` y `users`, el borrado y la actualización en cascada. Esto es porque si se borra un usuario no sirve de nada seguir manteniendo sus `parkings` favoritos y si se borra un `parking` no sirve de nada que un usuario lo mantenga como favorito si se ha borrado.

4.3 Boceto de la plataforma

Una vez que ya se ha diseñado la base de datos y se ha propuesto que estructura tendrá nuestra plataforma, con el fin de mejorar la experiencia del usuario se van a crear unos prototipos de nuestra web. Para el diseño de estos prototipos se hará uso de la herramienta *Justinmind*[7].

Como nuestra web será accesible desde cualquier tipo de dispositivo con acceso a internet se han creado prototipos para dispositivos móviles. No se considera necesario hacer prototipos también para ordenadores ya que la web se adaptará al tamaño de la pantalla del dispositivo desde la que se accede.

Como se puede ver en la figura 4.3, en cuanto entremos a la web nos saldrá un listado con los `parkings` que se hayan registrado en nuestra plataforma. Aparecerán en una tabla con 4 columnas. En la primera aparecerá el nombre del `parking`. En la siguiente columna aparecerá la dirección. Esta dirección vendrá en forma de enlace por lo que si los usuarios de nuestra plataforma hacen clic en dicho enlace se les abrirá una pestaña nueva de *Google Maps* con el `parking` seleccionado, en el caso de que se acceda a la web con ordenador, o se les abrirá la aplicación de *Google Maps* en caso de que la tengan instalada y accedan desde un dispositivo móvil. Esto facilitará a los usuarios a llegar al `parking` si no saben ya cómo hacerlo. Por último, las 2 últimas columnas indicarán el número de plazas del `parking` y el número de plazas libres. El color del número de plazas libres variará en función del porcentaje de ocupación, como se explicará en el siguiente apartado.

En la figura 4.4 se muestra el resultado de hacer clic sobre el icono del menú. Se ha decidido usar un menú hamburguesa ya que el resultado de la encuesta mostraba que los usuarios están familiarizados con los dispositivos móviles, por tanto, como actualmente se está utilizando mucho este menú es bastante probable que también estén familiarizados con este icono. A parte de esto, el menú hamburguesa aporta sencillez y facilidad

de navegación. Mediante este menú se podrá acceder al formulario de inicio de sesión o, simplemente, volver al inicio.



Figura 4.3: Prototipo pantalla inicial

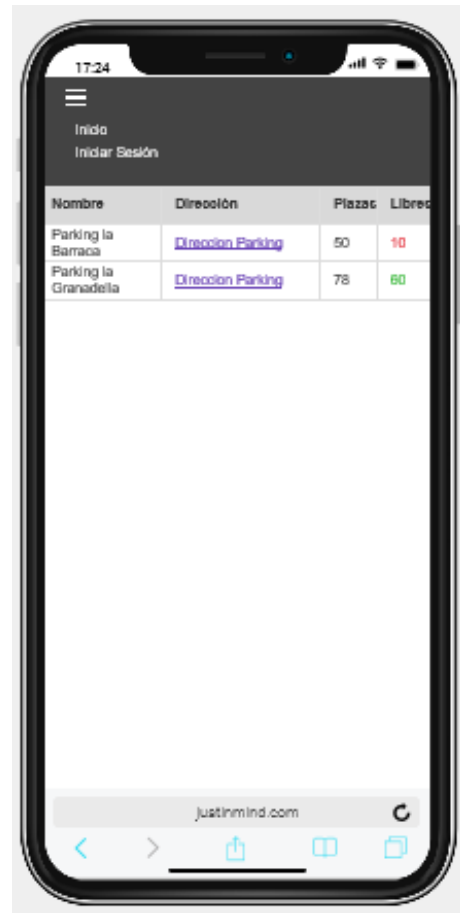


Figura 4.4: Prototipo pantalla inicial con menú desplegado

Las siguientes figuras, las figuras 4.5 y 4.6 son los prototipos de los formularios de inicio de sesión y de registro. Son formularios idénticos, pues para registrarse no hace falta nada más que un nombre de usuario y una contraseña. El botón de registrarse añadirá a la base de datos *users* el nombre de sesión y contraseña introducidos, aunque previamente se comprobará si ya existe algún usuario con ese nombre mostrándole un aviso al usuario si ya existiera un usuario con ese nombre.

El usuario podrá acceder al formulario de registro mediante un enlace que habrá bajo el botón de iniciar sesión.



Figura 4.5: Prototipo formulario de inicio de sesión

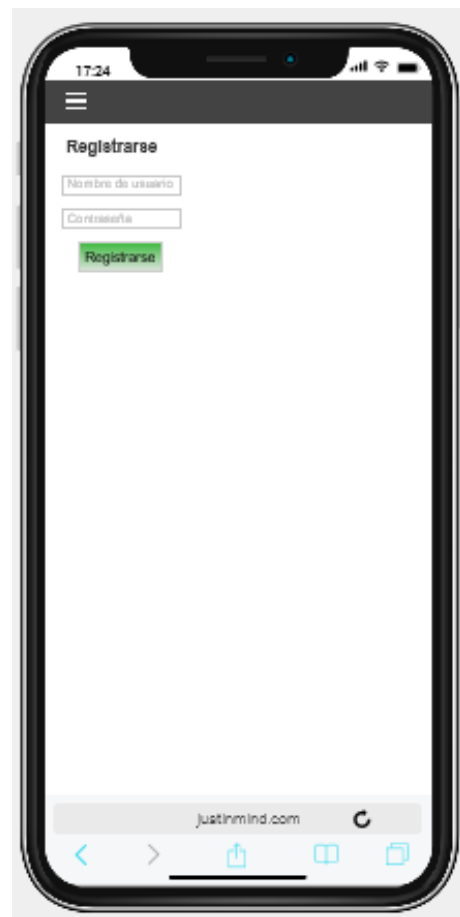


Figura 4.6: Prototipo formulario de registro



Figura 4.7: Prototipo pantalla inicial con sesión iniciada

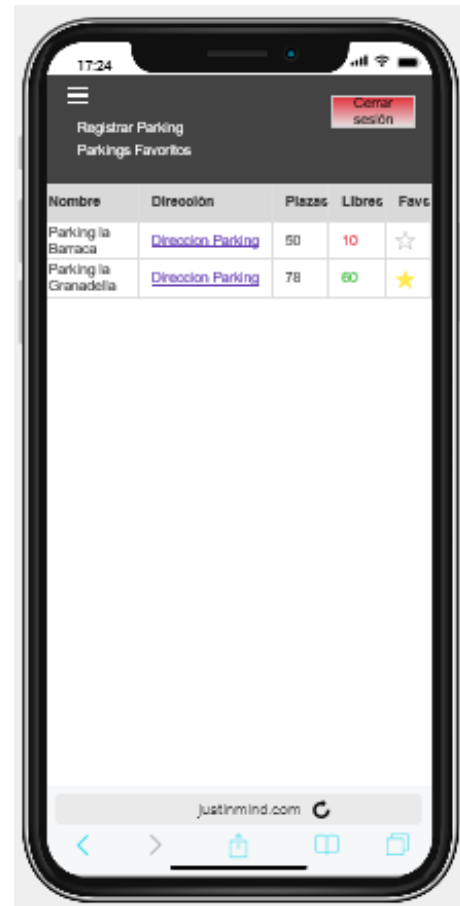


Figura 4.8: Prototipo pantalla inicial con sesión iniciada y menú desplegado

Una vez el usuario ya haya iniciado sesión se le llevará a una página parecida a la de la figura 4.3. La diferencia será que ahora podrá ver una columna más (figura 4.7), esta columna indicará que parkings ha añadido el usuario a favoritos. El usuario que se haya registrado y haya iniciado sesión tendrá algunas ventajas frente a los que no se hayan registrado. Una de ellas es, como se acaba de mencionar, añadir a favoritos los parkings que desee el propio usuario. Desplegando el menú (figura 4.8) podrá acceder a un listado donde aparezcan solo los parkings que previamente haya añadido a favoritos y también podrá acceder al formulario para registrar un parking.

Como es evidente, si ha iniciado sesión puede que en algún momento el usuario quiera cerrar sesión, es por esto que se ha añadido un botón de cerrar sesión. Este botón solo será visible cuando se despliegue el menú. Una vez más, facilitando la asociación que tiene el usuario con otras páginas donde inicie sesión, el botón será de color rojo a diferencia de los de iniciar sesión o registrarse, que serán verdes.

El prototipo del apartado *Parkings favoritos* no se mostrará, pues es idéntico al prototipo de la figura 4.7, con la única diferencia que solo se mostrarán los parkings que el usuario añada a favoritos.

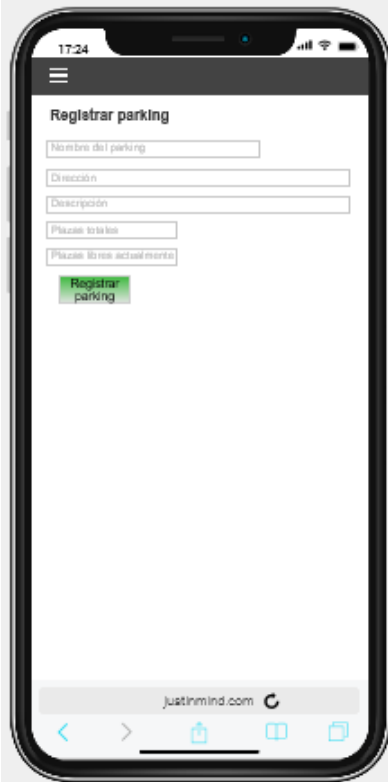
El prototipo muestra una pantalla de un smartphone con el título "Registrar parking". El formulario contiene cinco campos de entrada: "Nombre del parking", "Dirección", "Descripción", "Plazas totales" y "Plazas libres actualmente". Debajo de los campos hay un botón verde con el texto "Registrar parking". En la parte superior de la pantalla se muestra el tiempo "17:24" y los iconos de señal, Wi-Fi y batería. En la parte inferior se ve la barra de navegación del sistema operativo con los iconos de inicio, aplicaciones y recien.

Figura 4.9: Prototipo formulario de registro de parkings

Por último, en la figura 4.9 se muestra el formulario al que se puede acceder desde el menú una vez se haya iniciado sesión. Este formulario añadirá a la base de datos el parking con los datos del formulario aunque, al igual que con el registro de usuarios, previamente comprobará si existe algún parking con esa dirección y, en caso de existir, no se añadirá y avisará al usuario. Como es evidente, también exigirá al usuario que rellene el formulario que el número de plazas libres sea menor o igual que el número de plazas totales.

CAPÍTULO 5

Desarrollo de la solución

En apartados anteriores se ha puesto atención a que necesidades tienen los usuarios y que aplicaciones o páginas web existen que puedan parecerse a lo que se busca ofrecer. Se ha realizado una encuesta, se ha definido un usuario y unos escenarios en los que este usuario deberá usar la plataforma, y en el apartado anterior se ha diseñado la base de datos en la que guardaremos la información de los parkings y las cuentas de usuario de nuestros usuarios finales.

Este apartado se va a centrar en explicar que arquitectura tiene nuestra plataforma, así como que lenguajes se usarán en su desarrollo y algunos fragmentos de código que se consideren representativos de cada parte de nuestro sistema. Por todo ello, en este apartado empezaremos a desarrollar la plataforma final, es decir, empieza la fase de implementación.

5.1 Arquitectura

Si se tiene una visión global de la arquitectura de nuestra plataforma (Figura 5.1) se podría dividir en 3 capas. La primera sería la **capa de presentación**, es decir, la interfaz de usuario con la que el usuario interactuará con nuestra plataforma (el navegador con la aplicación web). Para las capas 2 y 3 se ha utilizado un paquete de software libre, **XAMPP**. La capa 2, la **capa de aplicación** contiene el servidor web, al usar XAMPP este servidor será un servidor web Apache, esta capa contendrá Scripts PHP para comunicarse con la capa 3. Por último, la capa 3 será la **capa de datos**. En esta capa se ubicará el servidor de base de datos MariaDB, como se explicará en el siguiente apartado, donde se almacenará todos los datos que se quieran mostrar en nuestra página web.

La mayor ventaja que tiene el uso de una arquitectura de 3 capas es la gran flexibilidad que ofrece, cada capa se podría ejecutar, por ejemplo, en un sistema operativo. Además, se podría modificar cualquiera de las capas sin que afecte a las otras.

Mediante la herramienta **phpMyAdmin** se ha creado la base de datos, estructura de la cual aparece en la Figura 4.2. A esta base de datos se harán consultas y actualizaciones para que los usuarios puedan realizar las funciones que debe ofrecer nuestra plataforma.

Como se ha comentado anteriormente, para poder obtener y/o actualizar la base de datos se deberá establecer una conexión con ella. Este establecimiento de conexión se hará mediante el lenguaje **PHP**, y los archivos que establezcan estas conexiones deberán ubicarse en la carpeta en la que se configure durante la instalación de XAMPP. Cabe destacar que este establecimiento se puede realizar en un archivo dedicado solo a esto o también se puede realizar dentro de la etiqueta `<main>` del archivo que contenga el código HTML. Aunque dicho archivo contenga código en HTML, deberá tener la extensión **.php**.

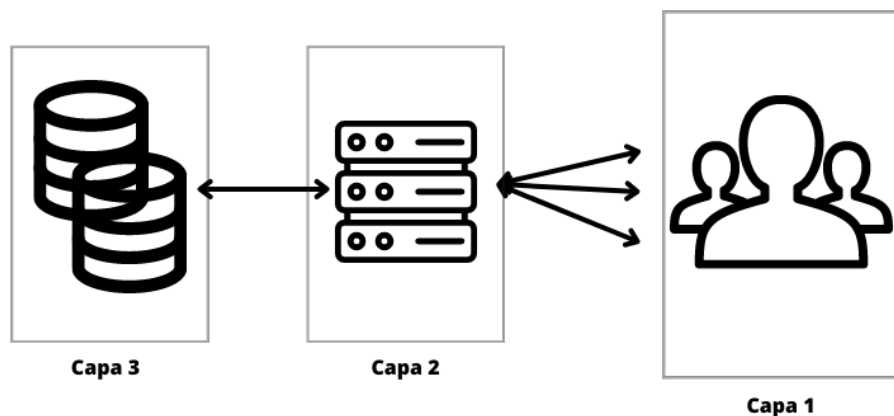


Figura 5.1: Arquitectura de la plataforma

Por tanto, cuando un cliente quiera consultar las plazas que haya libres en un parking se seguirán los siguientes pasos. El usuario entrará en nuestra web, esto es, el usuario ejecutará la capa 1. Después el servidor web (capa 2) realizará una consulta a la base de datos (capa 3). La base de datos responderá al servidor web y, por último, este mostrará al usuario cuantas plazas libres hay.

5.2 Contexto tecnológico

En esta sección se van a listar todas las tecnologías utilizadas durante el desarrollo de la plataforma y por qué se han elegido estas. Empezaremos con los lenguajes de programación y después se detallará el software utilizado.

1. PHP



PHP es un lenguaje de programación fácil de usar y que además es de código abierto. Normalmente PHP se usa conjuntamente con HTML ya que facilita la conexión entre servidor y página web o aplicación.

Se ha decidido usar este lenguaje porque es compatible con MariaDB que, precisamente, es el sistema de gestión de base de datos que tiene XAMPP. A parte de esto, PHP es un lenguaje fácil de entender y aprender y hay una gran cantidad de programadores en foros en los que si hace falta te resuelven cualquier duda.

Mediante el uso de PHP se obtienen todos los datos de nuestra plataforma, es decir, se obtienen todos los parkings para que se muestren en la web y también se almacenan los usuarios y sus contraseñas en una tabla existente en nuestra base de datos.

2. HTML



HTML5 son las siglas de *HyperText Markup Language* y podríamos decir que es la base de la web que se conoce hoy en día. El 5 no es nada más y nada menos que la versión.

HTML, gracias a sus etiquetas, permite estructurar la página web y agrupar cierta información dándole así un cierto significado.

Como solo nos permite estructurar la página web, si se quiere dar algún tipo de estilo a la web se deberá utilizar otro lenguaje que permita que la web sea visualmente más agradable y, como este trabajo lleva a cabo la metodología DCU, que sea más usable.

3. CSS



CSS son las siglas de *Cascading Style Sheets* y es un lenguaje que permite diseñar y describir la presentación de las páginas web.

Mediante el uso de CSS, el programador puede dar un color específico, animar, redimensionar, otorgarle una posición específica, todo esto, a cualquier elemento que haya en la web o aplicación.

El uso de este lenguaje puede ser un poco complejo ya que es fácil que se cometa algún error o que, simplemente, se introduzca un fragmento de código repetido o que sea inservible.

Como su propio nombre indica, es un lenguaje que se aplica en cascada, por lo que si hay más de un estilo definido para un mismo elemento, solo se aplicará el último.

Gracias a CSS se consigue proporcionar al usuario de nuestra plataforma una interfaz más usable.

4. JavaScript



Una vez que ya se tiene la estructura y el diseño de la página web, JavaScript ayudará a que, además, sea funcional.

JavaScript es un lenguaje de programación interpretado, débilmente tipado, imperativo y dinámico.

Aunque solo se ha utilizado JavaScript para definir algunas funciones, JavaScript a día de hoy puede usarse para, por ejemplo, sistemas en tiempo real.

Cabe destacar que JavaScript no tiene nada que ver con Java, simplemente son marcas registradas de Oracle.

5. Python



Python es un lenguaje de programación de alto nivel, interpretado y de propósito general. Python se ha convertido en uno de los lenguajes más populares en la industria del software y la comunidad de desarrollo. Hay que destacar Python por su legibilidad y simplicidad debido a su sencilla sintaxis.

Una de las grandes ventajas de usar Python y por lo que se ha elegido como lenguaje para el módulo de detección de matrículas, es porque como es uno de los lenguajes más usados, tiene una gran cantidad de desarrolladores por lo que existe una amplia biblioteca.

Para el desarrollo de este módulo se ha hecho uso de algunas bibliotecas de Python, **Numpy**, **imutils**, **pytesseract** y **PIL**. Estas bibliotecas se importan para: Numpy, esta biblioteca la utilizamos para realizar operaciones numéricas en la máscara y obtener las coordenadas de la matrícula. Imutils es una biblioteca de utilidades para simplificar las operaciones comunes de procesamiento de imágenes utilizando OpenCV. Pytesseract es una biblioteca de Python que brinda acceso a la funcionalidad OCR (Reconocimiento Óptico de Caracteres). Mediante esta biblioteca se consigue extraer el texto de una imagen, en nuestro caso se utiliza para extraer el número de matrícula de la imagen. Y, por último, PIL (Python Imaging Library), es una biblioteca de imágenes de Python que permite manipular imágenes, por ejemplo, se usa la biblioteca para convertir la imagen leída en un formato compatible con OpenCV.

6. OpenCV



OpenCV (Open Source Computer Vision Library) es una biblioteca de código abierto diseñada para procesamiento de imágenes y visión por computadora y está desarrollada por Intel.

Con la ayuda de las demás bibliotecas usadas, OpenCV se encarga de, por ejemplo, cargar la imagen, cambiar el color de la imagen a escala de grises para facilitar la detección de la matrícula, recortar la imagen, etc.

7. XAMPP



XAMPP es un paquete que integra diferentes herramientas con el fin de facilitar la creación y prueba de, por ejemplo, páginas web y aplicaciones. XAMPP es un

paquete de software completamente gratuito por lo que también facilita el acceso a él y fue desarrollado por Apache Friends.

El nombre XAMPP es realmente un acrónimo de las herramientas que contiene:

- **X:** representa que es paquete multiplataforma, es decir, se puede instalar en Windows, Linux y macOS.
- **A:** Apache. Es el servidor web.
- **M:** MariaDB. Es el sistema de gestión de base de datos (SGBD). Cabe destacar que en versiones más antiguas de XAMPP se utilizaba MySQL. Este SGBD combinado con el servidor web Apache y los lenguajes que se detallan a continuación, sirve para el almacenamiento de datos de servicios web.
- **P:** PHP. Es el lenguaje que permite crear web dinámicas y es especialmente adecuado en cuanto al desarrollo web se refiere.
- **P:** Perl. Es otro lenguaje de programación que contiene el paquete, pero en este caso, está orientado a la programación de la red y la administración de los sistemas.

Como es evidente, si el paquete XAMPP es soportado en diferentes plataformas (como indica la X), sus herramientas también los serán.

A pesar de existir diferentes alternativas a XAMPP, se ha decantado por este paquete debido a su facilidad de instalación y uso. Además, al haber trabajado con XAMPP en alguna asignatura, es un paquete bastante conocido y existe una gran cantidad de información sobre él en internet.

8. Apache



Apache es un servidor HTTP de código abierto. Apache se usa, básicamente, para enviar páginas web estáticas y dinámicas dentro de internet.

Se ha decantado por el uso de Apache además de porque es la herramienta que integra XAMPP porque mediante Apache se pueden ejecutar los archivos PHP con los que se accede a la base de datos que se ha creado en el servidor para consultarla y/o modificarla.

9. MariaDB y phpMyAdmin



MariaDB es un sistema de gestión de base de datos que deriva de MySQL.

PhpMyAdmin es una herramienta de software libre y que se usa con el fin de administrar las bases de datos MariaDB del servidor. Trabajar con esta herramienta es bastante sencillo ya que permite la creación y administración de base de datos, ya sea crear tablas, modificarlas, borrarlas y un largo etc. Permite realizar estas operaciones desde su interfaz web, y también la capacidad de ejecutar instrucciones SQL.

En phpMyAdmin se han creado todas las tablas necesarias en el desarrollo de la plataforma y mediante esta misma plataforma se han insertado algunos datos en la tabla.

10. Visual Studio Code



Visual Studio Code es un editor de código desarrollado por Microsoft. Visual Studio Code ofrece una interfaz sencilla y agradable lo que lo hace uno de los editores de código más usado por desarrolladores.

El uso de esta herramienta tiene múltiples ventajas, posee una gran capacidad de personalización, tiene gran cantidad de extensiones creadas por usuarios, y por lo que se ha decantado la balanza a la hora de utilizar esta herramienta, permite programar con la mayoría de lenguajes.

5.3 Código de la Aplicación Web

En este apartado se va a mostrar algunos de los fragmentos de código que se consideran representativos.

Para empezar, como se mencionada en reiteradas ocasiones, en cada página que queramos obtener o almacenar datos de nuestra base de datos se deberá realizar una conexión a esta. Esta conexión se puede hacer de 2 formas distintas, en un archivo separado que solo sirva para conectarse a la base de datos, o como se realiza en nuestro caso, dentro de una etiqueta `<main>` dentro del cuerpo del documento HTML.

```
<?php
    $servername = "localhost";
    $usernamebd = "root";
    $passwordbd = "";
    $dbname = "tfg";

    $conn = new mysqli($servername, $usernamebd, $passwordbd, $dbname);

    if ($conn->connect_error) {
        die("Conexión fallida: " . $conn->connect_error);
        echo "conexion fallida" ;
    }
```

Figura 5.2: Código que establece conexión con la base de datos con mysqli

Como se aprecia en la figura 5.2, a la función *mysqli* se le pasan 4 argumentos, el primero es el nombre del servidor, localhost, el segundo es el usuario, root, el tercero es la contraseña (no se ha establecido contraseña) y, por último, el nombre de la base de datos, tfg. Si durante el establecimiento de la conexión hubiera algún problema o simplemente introdujéramos algún parámetro incorrectamente, devolvería una cadena indicando que se ha producido un error.

El establecimiento de conexión se puede realizar con 2 extensiones de PHP, la primera es la que se ha mostrado en la figura 5.2, mediante la extensión **mysqli**, pero también se puede utilizar la extensión **PDO** como se muestra en la figura 5.3.

```
$servername = "localhost";
$usernamebd = "root";
$passwordbd = "";
$dbname = "tfg";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $usernamebd, $passwordbd);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
}
```

Figura 5.3: Código que establece conexión con la base de datos con PDO

Una vez ya se ha establecido la conexión con la base de datos, ya podemos realizar operaciones con ella, por ejemplo, insertar en la tabla **users** el nombre de usuario y la contraseña de un usuario que se registra en nuestra web, como se aprecia en la figura 5.4.

```
$consulta = "SELECT * FROM users WHERE Nombre = :nombreUsuario";
$stmt = $conn->prepare($consulta);
$stmt->bindParam(':nombreUsuario', $username);
$stmt->execute();

if ($stmt->rowCount() == 0) {
    $insertar = "INSERT INTO users (Nombre, Contraseña) VALUES (:username, :password)";
    $stmt = $conn->prepare($insertar);
    $stmt->bindParam(':username', $username);
    $stmt->bindParam(':password', $password);
    $stmt->execute();

    $_SESSION["username"] = $username;
    header("Location: sesionIniciada.php");
    exit();
} else {
    echo '<script language="javascript">alert("El nombre de usuario ya existe"); window.location="registro.php";</script>';
    exit();
}
```

Figura 5.4: Código que almacena en la base de datos el nombre y contraseña de un usuario

Lo primero que debemos hacer es consultar si el nombre de usuario con el que se está intentado registrar ya existe en nuestra base de datos. Esta comprobación la realiza el primer fragmento del código comparando los nombres que hay en la base de datos con el nombre de usuario que se recibe del formulario (valor *\$username*). Si en la ejecución de la consulta no hay ninguna fila significa que no hay ningún usuario con el mismo nombre de usuario por lo que, tanto el nombre como la contraseña (que también se recibe del formulario como *\$password*) se insertarán en la tabla. Si en cambio, el resultado de la consulta es que ya hay algún usuario con ese nombre, el navegador mostrará un cuadro de diálogo en el que se nos avisará de esto y tras pulsar en aceptar volveremos al formulario de registro.

Como se ve al final del *if*, se almacena el nombre de usuario en una variable de sesión. Esto lo usaremos para comprobar que cuando se accede a la web si existe una sesión iniciada el usuario poseerá privilegios (podrá añadir a sus favoritos los parkings que desee y podrá acceder al formulario de registro de parkings). Por tanto, para asegurarnos de que existe una sesión iniciada, en cada página que queramos comprobarlo deberemos añadir el código PHP de la figura 5.5 al inicio del documento.

```

<?php
session_start();

if (!isset($_SESSION["username"])) {
    header("Location: InicioDeSesion.php");
    exit();
}

$username = $_SESSION["username"];
?>

```

Figura 5.5: Código que comprueba si hay una sesión iniciada

Este código lo primero que hace es iniciar sesión para que las variables de sesión se puedan utilizar a lo largo del documento. Después se comprueba que la variable de sesión *username* está definida y si no lo está se redirige al usuario al formulario de inicio de sesión. Finalmente, si existe una sesión iniciada, almacena en una variable el nombre de usuario de la sesión existente y se continúa “ejecutando” el documento.

Una vez que el usuario ha iniciado sesión y se ha comprobado que hay una sesión existente, se creará la tabla con los datos que se muestran los parkings.

```

$sql = "SELECT id, Nombre, Direccion, Plazas, Libres FROM parkings";
$result = $conn->query($sql);
if ($result->num_rows > 0) {
    echo "<table>";
    echo "<tr><th>Nombre</th><th>Direccion</th><th>Plazas</th><th>Libres</th><th>Favs</th></tr>";

    while($row = $result->fetch_assoc()) {

        echo "<tr>";
        echo "<td> . $row['Nombre'] . "</td>";
        echo "<td><a href='https://www.google.com/maps?q=" . urlencode($row['Direccion']) . "' target='_blank'> . $row['Direccion'] . "</a></td>";

        $plazas = $row["Plazas"];
        $libres = $row["Libres"];
        $libresStyle = "";
        if ($libres <= $plazas/4) {
            $libresStyle = "style='color: red;'";
        } elseif ($libres > $plazas/4 && $libres < $plazas/2) {
            $libresStyle = "style='color: orange;'";
        } elseif ($libres >= $plazas/2) {
            $libresStyle = "style='color: green;'";
        }
        echo "<td> . $row["Plazas"] . "</td>";
        echo "<td $libresStyle> . $libres . "</td>";
    }
}

```

Figura 5.6: Código que crea la tabla de parkings (I)

```

echo "<td>";
echo "<form method='POST' action=''>";
echo "<input type='hidden' name='usuario_id' value='" . $username . "'>";
echo "<input type='hidden' name='parking_id' value='" . $row['id'] . "'>";

```

Figura 5.7: Código que crea la tabla de parkings (II)

```

$consu = "SELECT * FROM favoritos WHERE id_usuario = '$username' AND id_parking = " . $row['id'];
$resultado_favorito = $conn->query($consu);
if ($resultado_favorito->num_rows > 0) {
    echo "<button onclick='highlightCell(this)' type='submit' class='favorite-button' style='background-color: yellow;'>";
    echo "<i class='far fa-star'></i></button>";
} else {
    echo "<button onclick='highlightCell(this)' type='submit' class='favorite-button'><i class='far fa-star'></i></button>";
}
echo "</form>";
echo "</td>";
echo "</tr>";

```

Figura 5.8: Código que crea la tabla de parkings (III)

En la figura 5.6 se puede ver como se crean las columnas **Nombre**, **Dirección**, **Plazas** y **Libres**. Los valores de estas se obtienen de la tabla `parkings` de la base de datos. Como se puede ver en la definición de columnas de la tabla hay una última columna, la columna **Favs**, esta columna solo será visible para los usuarios que hayan iniciado sesión por lo que el código de creación de la tabla para los usuarios no autenticados será el mismo que el de la imagen excluyendo la definición de la columna `Favs` (`<th>Favs</th>`). En esta misma figura (la 5.6) se puede apreciar como en la columna **Dirección** en vez de escribir un valor, la dirección se inserta como enlace a *Google Maps*. Cuando se pulse en dicho enlace se abrirá en una pestaña nueva. También en esta imagen se puede apreciar cómo se le da un color diferente a cada fila de la columna **Libres** en función de este valor. Si la mitad o más de la mitad de las plazas del parking están libres, el número de plazas libres tendrá un color verde. Si hay entre un 26 % y un 49 %, ambos inclusive, el número será de color naranja. Por último, si el número de plazas libres es de un 25 % o menos, aparecerá de color rojo.

En la figura 5.7 y 5.8 está el código que genera la columna **Favs**. Para generarlo lo primero que se hace es obtener el nombre de usuario con la sesión iniciada y el id del parking de la fila actual. Una vez que ya se tienen estos 2 valores, se consulta la tabla de la base de datos favoritos, ya que los usuarios pueden haber iniciado sesión previamente y pueden haber añadido algún parking ya como favorito. Aunque no se enseñe el código, se ha hecho un apartado accesible desde el menú en el que solo se muestren los parkings favoritos. Esta adaptación se realizará cambiando la posición de la consulta y el `if` de la tabla favoritos al inicio de la creación de la tabla.

En cada fila de esta columna se inserta un botón con forma de estrella, se elige esta forma porque es bastante probable que los usuarios estén familiarizados con este tipo de botón y asocien la estrella a favoritos. Para cada fila de la tabla de nuestra web, es decir, para cada parking, se hace una consulta a la tabla favoritos de la base de datos en la que si en esta tabla existe una entrada con ese usuario y el id de parking del parking de la fila actual. Si existe, es porque anteriormente el usuario ya ha añadido el parking a favoritos, por lo que se resaltará el botón con un fondo amarillo. Si en cambio no hay ninguna entrada en la tabla con esa dupla de valores, el botón no deberá tener ningún fondo ya que no es un parking favorito del usuario.

Ahora que ya se ha creado la tabla en nuestra web, se debe proporcionar al usuario la opción de añadir o eliminar de favoritos un parking. Esto lo haremos mediante la función `highlightCell()`. Esta función (figura 5.9), a la que se le pasa el botón como argumento, establece que acción se debe ejecutar en función del color del fondo del botón, es decir, si el botón tiene un fondo amarillo significa que el parking está en la tabla favoritos de la base de datos, por lo que si se pulsa otra vez sobre él quiere decir que el usuario quiere borrar este parking de sus favoritos y llamará al archivo `eliminar_favorito.php`. Por otro lado, si el botón no tiene fondo de color y el usuario pulsa sobre él significa que el usuario quiere añadirlo a favoritos, por lo que se llamará a `guardar_favorito.php`. Estos 2 archivos ejecutarán un código parecido al que almacena el usuario y la contraseña en el registro (figura 5.4) por lo que se cree que no es necesario mostrar el código.

```
function highlightCell(button) {
    var form = button.parentNode;
    if (button.style.backgroundColor === 'yellow') {
        form.action = 'eliminar_favorito.php';
    } else {
        form.action = 'guardar_favorito.php';
    }
    form.submit();
}
```

Figura 5.9: Código de la función highlightCell()

Cuando se llama a `guardar_favorito.php`, se realiza una consulta en la tabla de la base de datos favoritos, y si no hay ninguna entrada con el nombre de usuario y el id del parking se hace un INSERT en la tabla con esos valores. Después de esto se redirige al usuario a la página anterior por lo que ahora este parking tendrá el botón con el fondo amarillo, y por lo tanto, será un parking favorito. Todo lo contrario para `eliminar_favorito.php`. Aquí también se hará una consulta, pero esta vez si existe alguna entrada en la tabla con el nombre de usuario que hace la petición y el id del parking sobre el que se ha pulsado el botón, se realizará un DELETE de esa fila de la tabla de la base de datos. Después también se le redirigirá a la página anterior y a diferencia de antes, ahora el botón pulsado no tendrá el color de fondo.

Hay que destacar que siempre que se establezca una conexión con la base de datos, habrá que cerrarla una vez se termine realizar operaciones con ella. Dependiendo de si se ha utilizado la función `mysqli` o PDO se deberá cerrar con `$conn = null`, para el establecimiento de conexión de la figura 5.3, y con `$conn ->close()` para el establecimiento de la figura 5.2.

Para acabar con este apartado, en la figura 5.10 se muestra el código perteneciente al formulario de registro de un parking.

```
<form id="login-form" action="registroParking.php" method="POST">
  <input type="text" class="direccion" name="name" placeholder="Nombre del Parking" required><br>
  <div class="input-container">
    <input type="text" class="direccion" name="direccion" placeholder="Direccion" required><br>
    <span class="input-icon"><i class="fas fa-question-circle"></i></span>
    <span class="hover-text">El formato de la dirección no debe contener comas: Calle Numero Codigo_Postal Ciudad Provincia</span>
  </div>
  <input type="text" class="descripcion" name="descripcion" placeholder="Descripcion" ><br>
  <input type="number" class="num" name="plazas" placeholder="Plazas totales" required><br>
  <input type="number" class="num" name="libres" placeholder="Plazas libres actualmente" required><br>
  <input type="submit" value="Registrar">
</form>
```

Figura 5.10: Código de creación del formulario de registro de parkings

Como se aprecia, todos los campos menos la descripción serán obligatorios. Esto se define con el parámetro `required` al final del `input`. Los `input` de las plazas libres y las plazas totales se establecerán como `type="number"` de forma que solo se permitirán valores numéricos y en el extremo derecho de los campos aparecerán 2 pequeños botones con los que el usuario podrá aumentar o disminuir de 1 en 1 el valor introducido. También en el extremo derecho, pero esta vez del campo dirección, aparecerá un símbolo en forma de interrogante en el que, si el usuario pone el puntero del ratón encima, aparecerá una pequeña descripción sobre cómo se debe introducir la dirección. Se realiza esto con el fin de unificar formato de las direcciones para poder compararlas con las direcciones de los parkings ya existentes en la base de datos.

Por último, el archivo `registroParking.php` será el encargado de comprobar si ya existe un parking con esa dirección y en caso de que no exista realizar la inserción en la tabla,

aunque primero se comprobará que el número de plazas libres sea menor o igual que el de plazas totales.

5.4 Código del lector de matrículas

Una vez ya se han enseñado los fragmentos de código pertenecientes a la aplicación web, se mostrará algún fragmento de código del módulo desarrollado para la detección de matrículas para el posterior incremento o decremento del valor de plazas libres de la tabla parkings existente en nuestra base de datos.

Lo primero que deberemos hacer es importar todas las librerías que se vayan a usar y que ya se han explicado en el apartado 5.2. Una vez que ya hayamos importado las librerías ya se puede empezar a trabajar con ellas.

El siguiente paso será cargar la imagen que contenga una matrícula para poder empezar a manipularla. Una vez que ya tengamos la imagen cargada, la redimensionaremos para evitar posibles errores ya que trabajaremos con imágenes descargadas, así nos aseguraremos de que todas tengan el mismo tamaño. Luego se convertirá la imagen a escala de grises ya que facilita la detección del contorno de la matrícula y no aporta nada trabajar con una foto a color. Una vez hecho esto, se reducirá el ruido que contenga la imagen mediante el uso de la función de OpenCV `bilateralFilter()`. Por la simplicidad del código realizado hasta ahora no se añade ninguna captura.

Una vez ya tengamos la imagen preparada, vamos a detectar los contornos de esta y seleccionar algunos de ellos (figura 5.11). Con la primera línea, gracias a la función `Canny()` de OpenCV se dibujan los bordes de la imagen (por esto se reducía el ruido anteriormente, para reducir los bordes). Ahora que ya tenemos todos los bordes detectados, los ordenamos de mayor a menor y se seleccionan los mayores 10 contornos que tengan una superficie cerrada.

```
edged = cv2.Canny(gray, 30, 200)
cnts = cv2.findContours(edged.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
cnts = sorted(cnts, key = cv2.contourArea, reverse = True)[:10]
```

Figura 5.11: Detección de contornos de la imagen

Entre estos 10 contornos debería estar la matrícula. Se recorrerán estos 10 resultados mediante un bucle buscando uno que tenga forma de rectángulo y como ya se ha hecho una preselección antes, seguro tendrá una superficie cerrada. Si el resultado de esta búsqueda es positivo, se confirmará que hay una matrícula en nuestra imagen y ya sería suficiente para incrementar o decrementar el valor de plazas libres, pero cómo es posible que en un futuro sea necesario detectar el número de matrícula para, por ejemplo, imprimir un ticket a la entrada del parking, se implementará parte del código que permita leer la matrícula.

```
mask = np.zeros(gray.shape,np.uint8)
new_image = cv2.drawContours(mask,[screenCnt],0,255,-1,)
new_image = cv2.bitwise_and(img,img,mask=mask)

(x, y) = np.where(mask == 255)
(topx, topy) = (np.min(x), np.min(y))
(bottomx, bottomy) = (np.max(x), np.max(y))
Cropped = gray[topx:bottomx+1, topy:bottomy+1]
```

Figura 5.12: Selección del contorno de la matrícula

En esta parte del código, la figura 5.12, se hace uso de la librería Numpy para seleccionar solo la parte que contiene la matrícula. Para ello lo primero que se hace es crear una matriz del mismo tamaño que la imagen que más tarde se usará como máscara, luego se dibuja el contorno de la imagen y por último se hace una operación **AND** entre la imagen y la máscara con el resultado de que todos los píxeles de la imagen que no estén dentro del rectángulo dibujado se establecerán en negros. Con la segunda parte conseguiremos crear una nueva imagen. Esta imagen será un recorte de la imagen inicial usando las coordenadas obtenidas y con el resultado final de una imagen de la matrícula.

```
text = pytesseract.image_to_string(Cropped, config='--psm 11')

with open('matricula.txt', 'w') as file:
    file.write(text)
```

Figura 5.13: Escritura de la matrícula leída

Por último y gracias a la librería de Python **pytesseract** y su función **image_to_string()** extraeremos el texto que contenga la imagen creada en la figura 5.12 y almacenaremos la matrícula en un archivo para poder utilizarla en caso de necesitarla (figura 5.13).

Una vez ya hayamos obtenido el número de matrícula y lo tengamos almacenado en un archivo procedemos a actualizar la base de datos.

Para ello lo primero que se debe hacer es ejecutar el archivo Python que se ha descrito. Una vez que se haya ejecutado comprobaremos si existe el archivo **matricula.txt** y si existe leeremos el contenido y borraremos el archivo para evitar confusiones con futuras ejecuciones (figura 5.14). Si el archivo no está vacío significa que se ha leído una matrícula y por lo que se deberá proceder a actualizar la tabla parkings.

```
$command = "python " . __DIR__ . "/detect_license_plate.py";
exec($command);

if (file_exists("matricula.txt")){
    $text = trim(file_get_contents('matricula.txt'));
    unlink("matricula.txt");
}

if (!empty($text)) {
```

Figura 5.14: Comprobación de la existencia de una matrícula

Cuando se hayan realizado todas las comprobaciones se hará un *UPDATE* de la tabla y en función de si ha sido a la entrada o a la salida del parking habrá que sumar o restar 1 al número de plazas libres. Como se aprecia en la figura 5.15, el detector de matrículas al que pertenece este fragmento de código estaría instalado en la salida de la Platja de la Barraca, por lo que, al detectar una matrícula, el número de plazas libres debería de incrementarse en 1.

```
$update = "UPDATE parkings SET libres = libres + 1 WHERE Nombre = 'Parking Playa de la Barraca'";
```

Figura 5.15: Actualización del número de plazas libres

Cabe destacar que en el desarrollo de este módulo de detección de matrículas puede haber fallos con alguna imagen. Se ha intentado evitar al máximo que falle estableciendo valores medios en, por ejemplo, la función **bilateralFilter()** y la función **Canny()**. También cabe destacar que en ocasiones la biblioteca Tesseract no reconoce correctamente los valores de las matrículas. O incluso, en ocasiones no se llega a detectar el contorno de la

matrícula. Para solucionar estos errores se podría hacer uso de *Machine Learning* basado en imágenes de coches o incluso mejorar la calidad de las imágenes. A continuación se muestra el resultado de ejecutar correctamente el lector de matrículas.

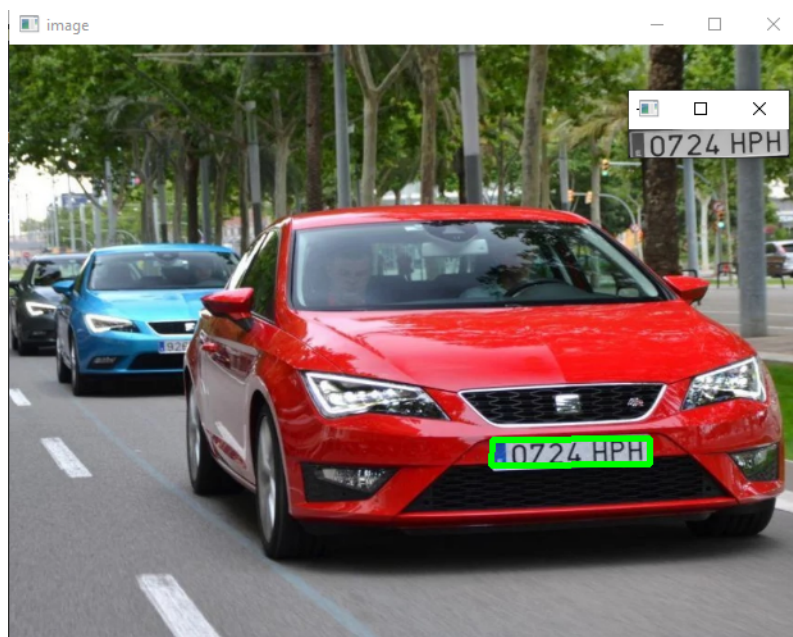


Figura 5.16: Resultado de la ejecución del lector de matrículas

Para poder apreciar visualmente el resultado se ha hecho uso de la función `imshow()` de OpenCV. En la figura 5.16 se ven 2 ventanas, la más grande es la imagen real con el contorno dibujado sobre lo que ha detectado como matrícula (en este caso correctamente), y la imagen más pequeña es el resultado de recortar la imagen principal dejando solo el rectángulo de la matrícula y sobre la que se aplicará la función `image_to_string()`.

CAPÍTULO 6

Producto desarrollado

En el siguiente apartado se presentará la aplicación web desarrollada y como usarla. Para ello, se va a proceder a mostrar como la persona definida en el apartado 3.2 realiza los escenarios creados en el 3.3.

6.1 Alfonso registra un parking

Como Alfonso se dispone a registrar un parking lo primero que deberá hacer será acceder a nuestra web. Cuando acceda lo primero que verá será una lista de los parkings, con sus direcciones y sus plazas totales y libres (figura 6.1). Como para poder registrar un parking hay que iniciar sesión previamente, hace clic en el icono de la parte superior izquierda y se despliega un menú (figura 6.2).

Nombre	Direccion	Plazas	Libres
Parking Platja de la Granadella	Ctra. de la Granadella 5 03738 Xàbia Alicante	80	40
Parking Cala Blanca	Glorieta de Guillermo Pons 2 03738 Xàbia Alicante	30	25
Parking el Arenal	Av. de la Llibertat 19 03738 Platja del Arenal Alicante	100	35

Figura 6.1: Pantalla inicial

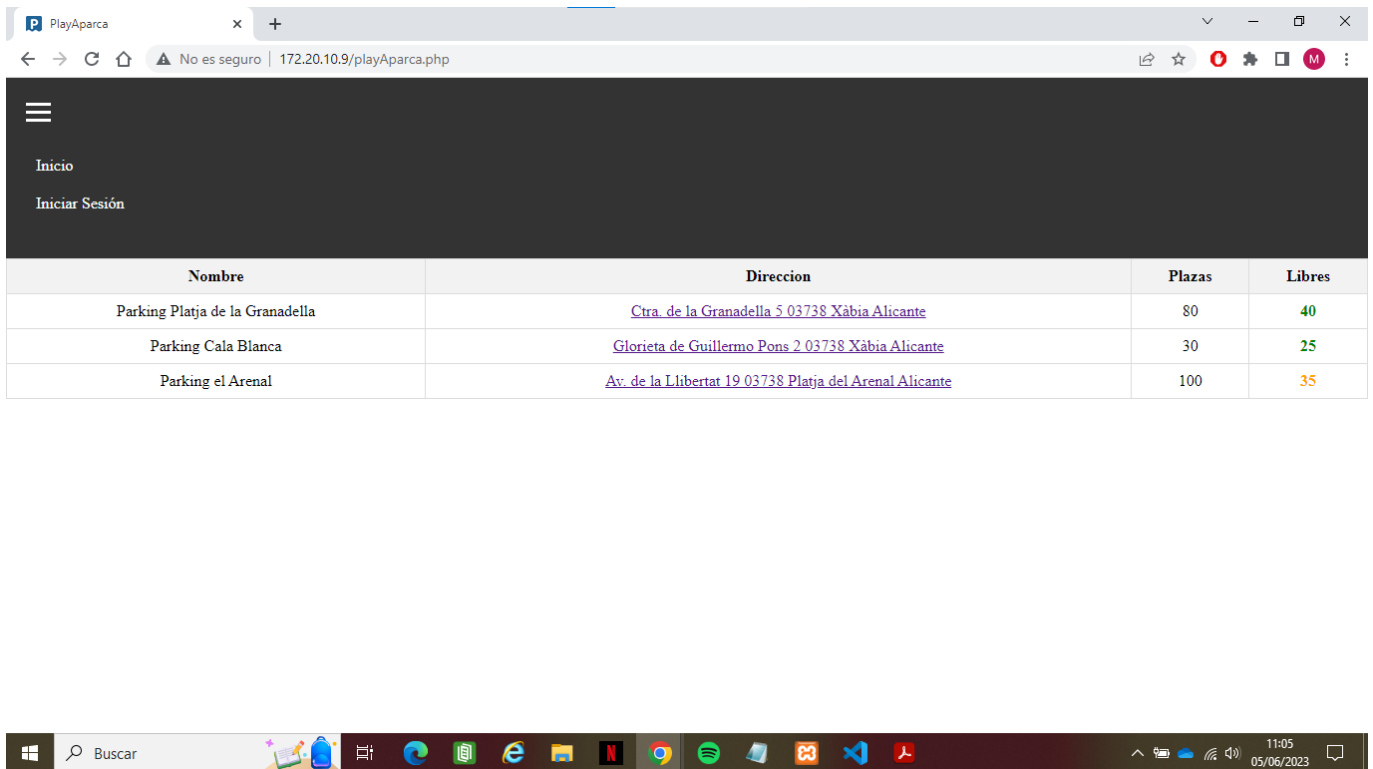


Figura 6.2: Pantalla inicial con el menú desplegado

Con el menú desplegado de la figura anterior hace clic en el apartado *Iniciar sesión* y le aparece un formulario para iniciar sesión (figura 6.3). Como Alfonso ya tiene una cuenta introduce su nombre de sesión, su contraseña y hace clic en el botón de iniciar sesión.

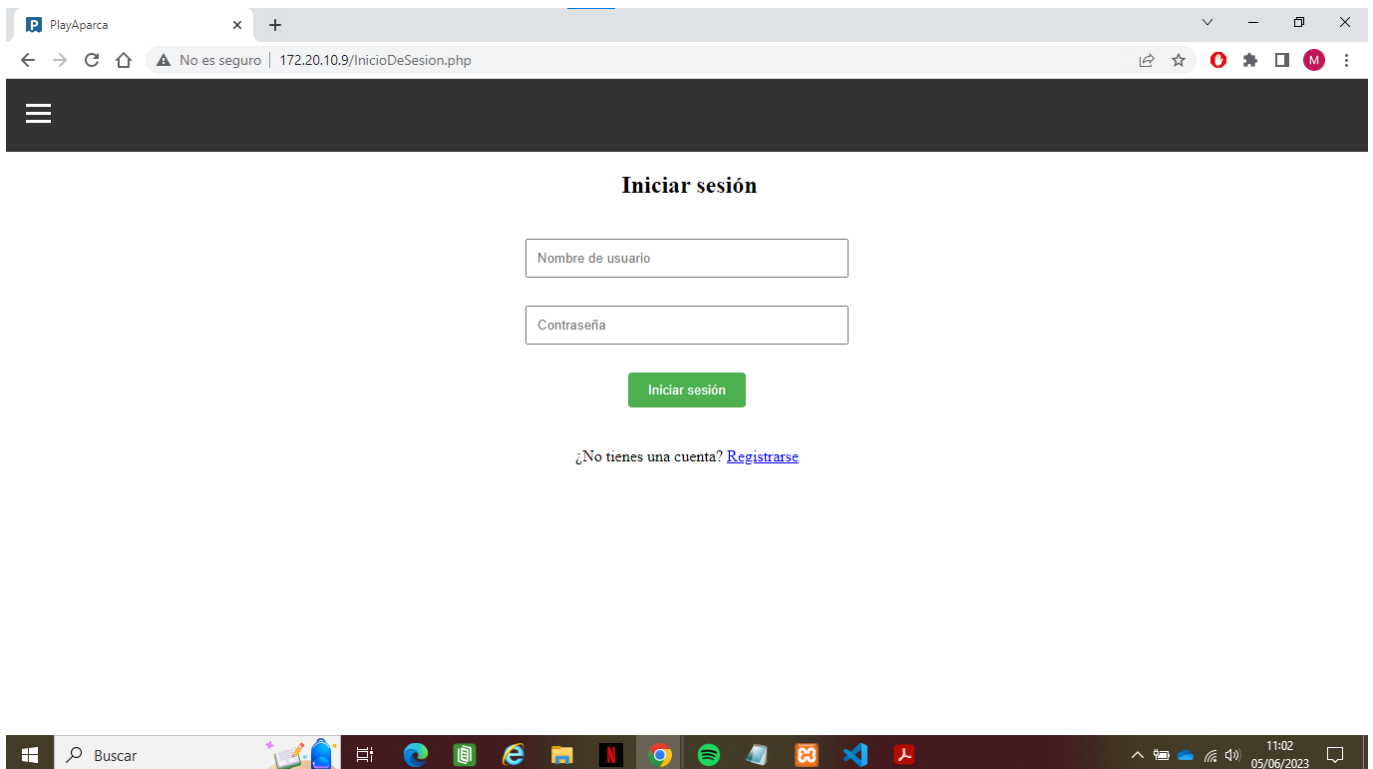


Figura 6.3: Pantalla de inicio de sesión

Sin darse cuenta, Alfonso ha introducido una contraseña incorrecta, por lo que le aparece una alerta (figura 6.4) y cuando hace clic en Aceptar, vuelve al formulario de inicio de sesión donde esta vez introduce los datos correctamente y, nuevamente, pulsa en el botón de iniciar sesión.

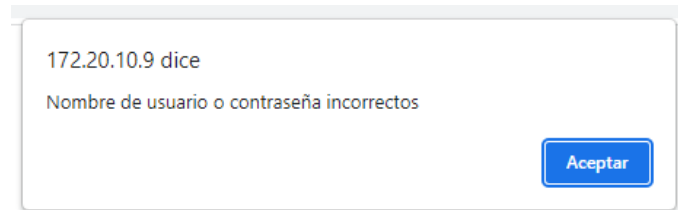


Figura 6.4: Mensaje error

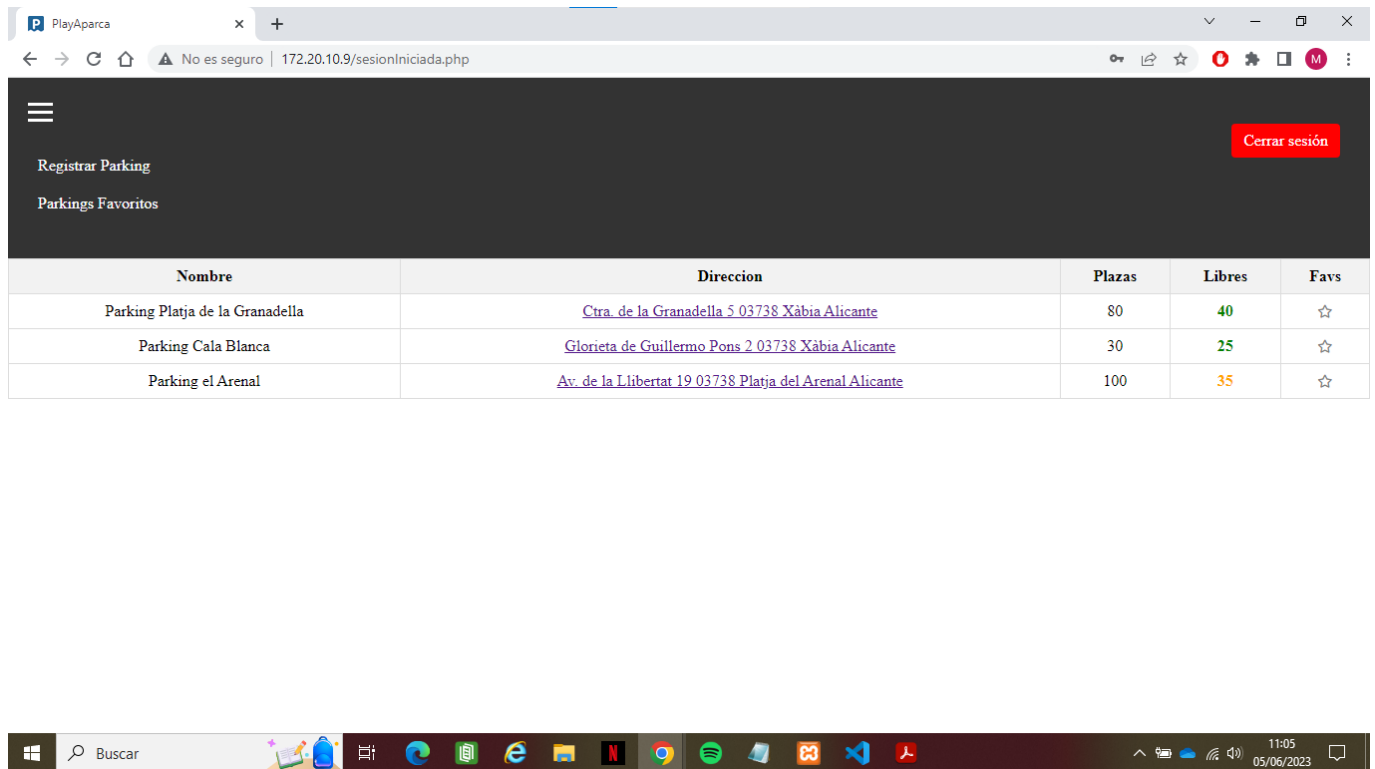
Ahora que ya tiene la sesión iniciada se le vuelve a mostrar la tabla con los parkings y sus datos, pero esta vez aparece una columna que previamente no aparecía. Como Alfonso quiere registrar un parking hace clic en el icono de la parte superior izquierda para desplegar el menú y como se aprecia en la figura 6.6, aparecen 2 opciones.

Una captura de pantalla de un navegador web que muestra la interfaz de usuario de PlayAparca. La barra de direcciones muestra la URL "172.20.10.9/sesionIniciada.php". La página principal contiene una tabla con los siguientes datos:

Nombre	Direccion	Plazas	Libres	Favs
Parking Platja de la Granadella	Ctra. de la Granadella 5 03738 Xàbia Alicante	80	40	☆
Parking Cala Blanca	Glorieta de Guillermo Pons 2 03738 Xàbia Alicante	30	25	☆
Parking el Arenal	Av. de la Llibertat 19 03738 Platja del Arenal Alicante	100	35	☆

La interfaz también incluye un menú desplegable en la parte superior izquierda y una barra de tareas de Windows en la parte inferior.

Figura 6.5: Pantalla principal con sesión iniciada (I)



The screenshot shows a web browser window with the URL `172.20.10.9/sesionIniciada.php`. The page has a dark header with a hamburger menu icon, the text 'Registrar Parking' and 'Parkings Favoritos', and a red 'Cerrar sesión' button. Below the header is a table with the following data:

Nombre	Direccion	Plazas	Libres	Favs
Parking Platja de la Granadella	Ctra. de la Granadella 5 03738 Xàbia Alicante	80	40	☆
Parking Cala Blanca	Glorieta de Guillermo Pons 2 03738 Xàbia Alicante	30	25	☆
Parking el Arenal	Av. de la Llibertat 19 03738 Platja del Arenal Alicante	100	35	☆

The Windows taskbar at the bottom shows the search bar with 'Buscar', several application icons, and the system tray with the time '11:05' and date '05/06/2023'.

Figura 6.6: Pantalla principal con sesión iniciada con menú desplegado

Al hacer clic en el apartado de Registrar Parking, Alfonso deja de ver la tabla con los parkings y ahora le aparece un formulario, Alfonso introduce los datos que tiene del parking, *Parking Platja de la Barraca, Carrer de la Barraca 45 03738 la Mar Blava Alacant*, 50 plazas totales y 10 plazas libres en ese momento. Alfonso revisa los datos y hace clic en el botón de Registrar.

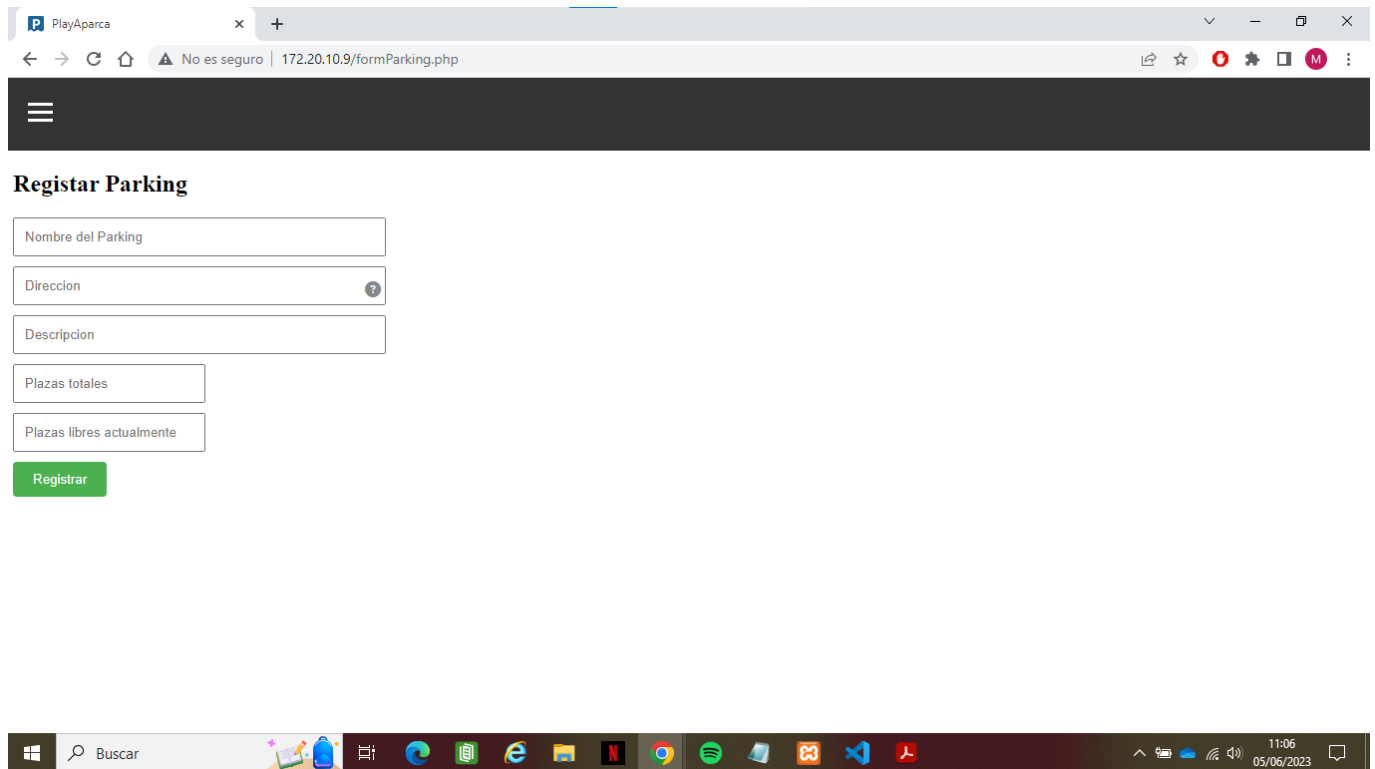


Figura 6.7: Pantalla de registro de un parking

Ahora Alfonso ya puede ver en la tabla el parking que acaba de registrar.

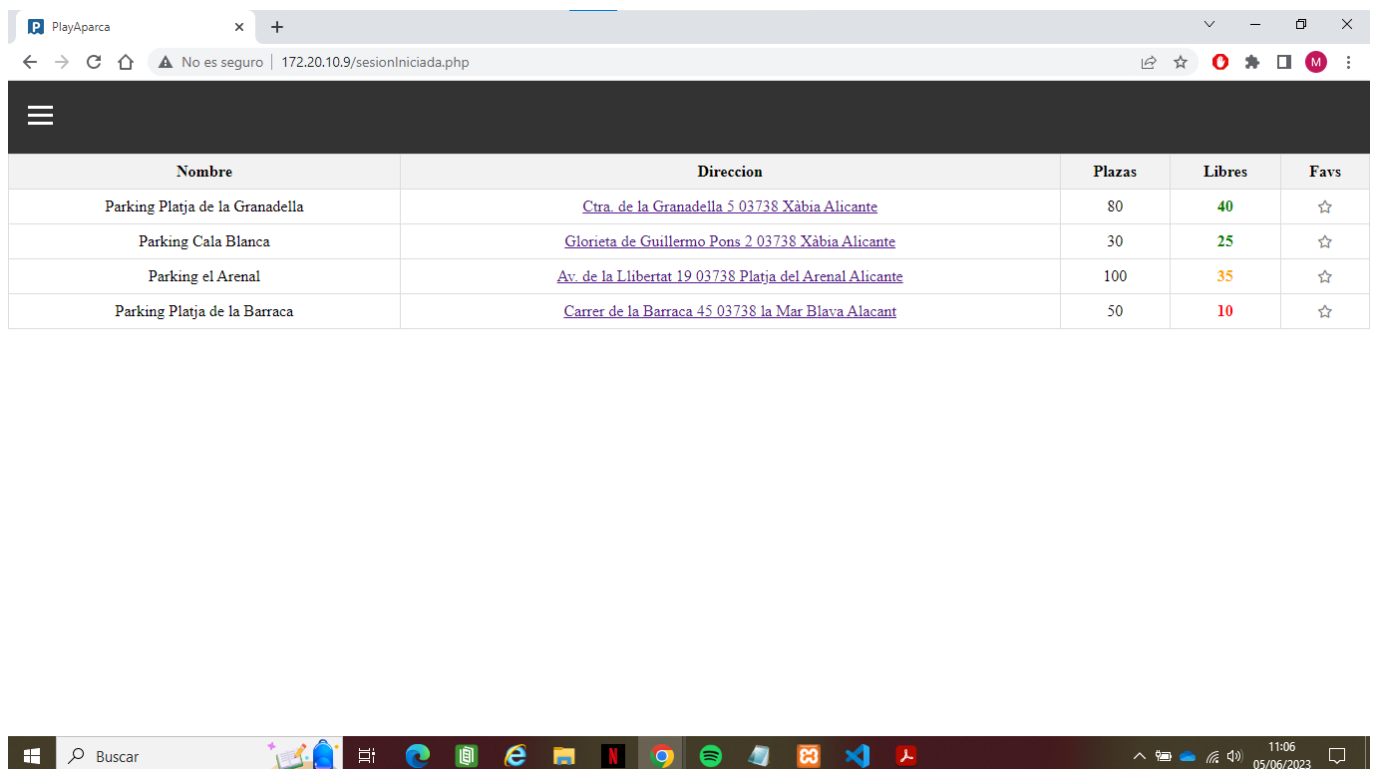



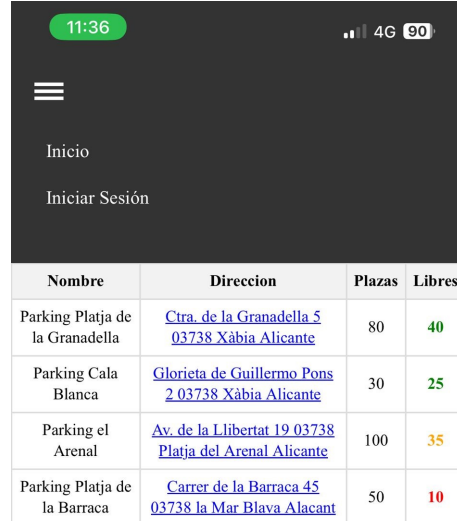
Figura 6.8: Pantalla principal con sesión iniciada (II)

6.2 Alfonso consulta el estado de un parking

Alfonso quiere consultar el estado de un parking. Para ello desde el móvil accede a la web **PlayAparca** y le aparece la pantalla que se muestra en la figura 6.9. Como Alfonso en un principio solo quiere consultar el estado del parking y desde esta pantalla puede hacerlo, no inicia sesión.



Nombre	Direccion	Plazas	Libres
Parking Platja de la Granadella	Ctra. de la Granadella 5 03738 Xàbia Alicante	80	40
Parking Cala Blanca	Glorieta de Guillermo Pons 2 03738 Xàbia Alicante	30	25
Parking el Arenal	Av. de la Llibertat 19 03738 Platja del Arenal Alicante	100	35
Parking Platja de la Barraca	Carrer de la Barraca 45 03738 la Mar Blava Alacant	50	10



Nombre	Direccion	Plazas	Libres
Parking Platja de la Granadella	Ctra. de la Granadella 5 03738 Xàbia Alicante	80	40
Parking Cala Blanca	Glorieta de Guillermo Pons 2 03738 Xàbia Alicante	30	25
Parking el Arenal	Av. de la Llibertat 19 03738 Platja del Arenal Alicante	100	35
Parking Platja de la Barraca	Carrer de la Barraca 45 03738 la Mar Blava Alacant	50	10

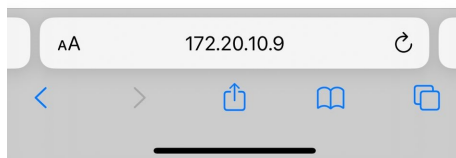


Figura 6.9: Pantalla inicial móvil

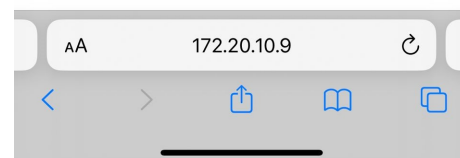


Figura 6.10: Pantalla inicial móvil con menú desplegado

Ahora que Alfonso y su pareja ya están volviendo al coche, a Alfonso le parece buena idea añadir el parking al que han ido a favoritos. Para ello vuelve a acceder a la aplicación web, hace clic en el icono de la parte superior izquierda y se le despliega el menú (figura 6.10). Cuando hace clic sobre el apartado de *Iniciar sesión* le aparece la página con un formulario para iniciar sesión (figura 6.11) y al introducir los datos se equivoca con la contraseña y le aparece una alerta que se lo indica, y al pulsar sobre Cerrar vuelve a la pantalla con el formulario para iniciar sesión.

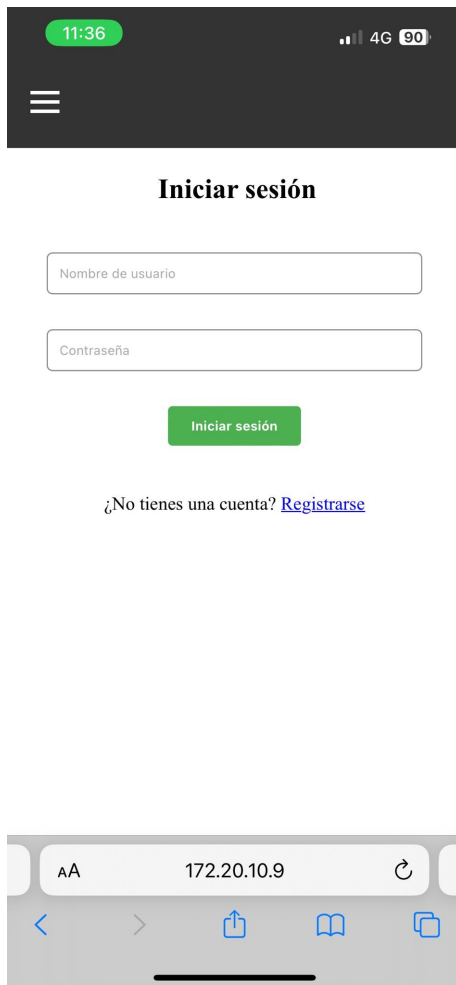


Figura 6.11: Pantalla de inicio de sesión móvil

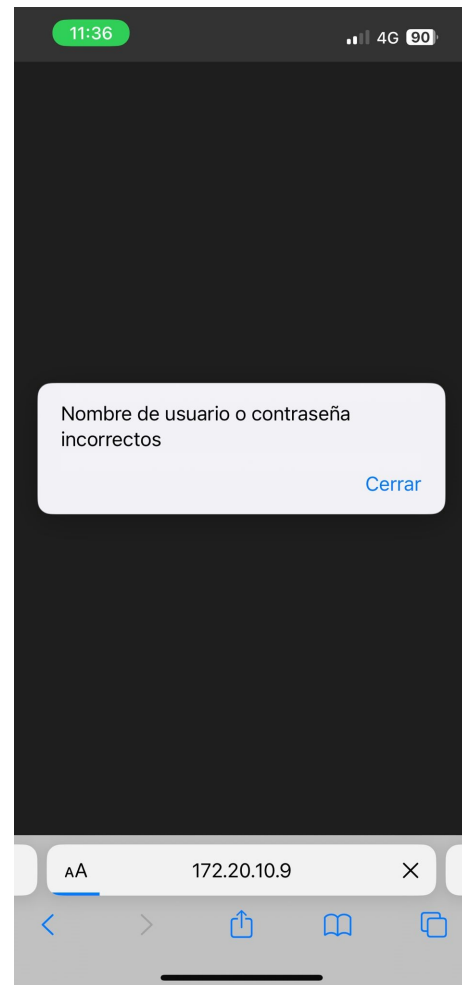



Figura 6.12: Pantalla error inicio de sesión móvil



Nombre	Direccion	Plazas	Libres	Favs
Parking Platja de la Granadella	Ctra. de la Granadella 5 03738 Xàbia Alicante	80	40	☆
Parking Cala Blanca	Glorieta de Guillermo Pons 2 03738 Xàbia Alicante	30	25	☆
Parking el Arenal	Av. de la Llibertat 19 03738 Platja del Arenal Alicante	100	35	☆
Parking Platja de la Barraca	Carrer de la Barraca 45 03738 la Mar Blava Alacant	50	10	☆



Nombre	Direccion	Plazas	Libres	Favs
Parking Platja de la Granadella	Ctra. de la Granadella 5 03738 Xàbia Alicante	80	40	☆
Parking Cala Blanca	Glorieta de Guillermo Pons 2 03738 Xàbia Alicante	30	25	☆
Parking el Arenal	Av. de la Llibertat 19 03738 Platja del Arenal Alicante	100	35	☆
Parking Platja de la Barraca	Carrer de la Barraca 45 03738 la Mar Blava Alacant	50	10	★

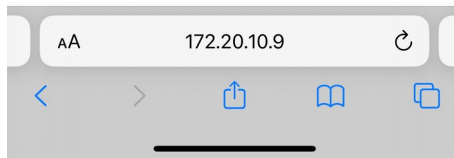


Figura 6.13: Pantalla principal con sesión iniciada móvil (I)

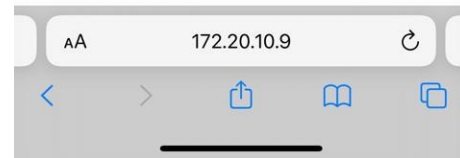


Figura 6.14: Pantalla principal con sesión iniciada móvil (II)

Cuando ha iniciado sesión correctamente le aparece la lista de parkings con una columna para añadir los parkings a favoritos (figura 6.13). Para añadir el parking a favoritos lo único que debe hacer es hacer clic en el botón en forma de estrella que se corresponde con el parking que quiere añadir. Una vez haya hecho clic, este se volverá amarillo (figura 6.14).

6.3 Otras capturas

Una vez se ha descrito como se realizan los escenarios, las figuras 6.15, 6.16 y 6.17 muestran capturas de interfaces móviles que no se han utilizado durante los escenarios.

La figura 6.15 muestra una pantalla con el menú desplegado en el que un usuario ya ha iniciado sesión. Desde aquí los usuarios autenticados pueden acceder al apartado de Parkings Favoritos (figura 6.16). En esta pantalla solo aparecerán los parkings que los usuarios hayan añadido a favoritos.



Figura 6.15: Pantalla principal con sesión iniciada móvil y menú desplegado



Figura 6.16: Pantalla con lista de parkings favoritos móvil

La figura 6.17 muestra la interfaz a la que se accede desde el enlace que aparece en la interfaz de inicio de sesión. Una vez se ha introducido el nombre de usuario y la contraseña y se haga clic en el botón de registrar se redireccionará al usuario a la pantalla principal (figura 6.13). Si se introduce un nombre de usuario que ya existe en la base de datos aparecerá un mensaje de alerta como el de la figura 6.4 (en el caso de que se acceda con ordenador) y el de la figura 6.12 (en caso de que se acceda con móvil).

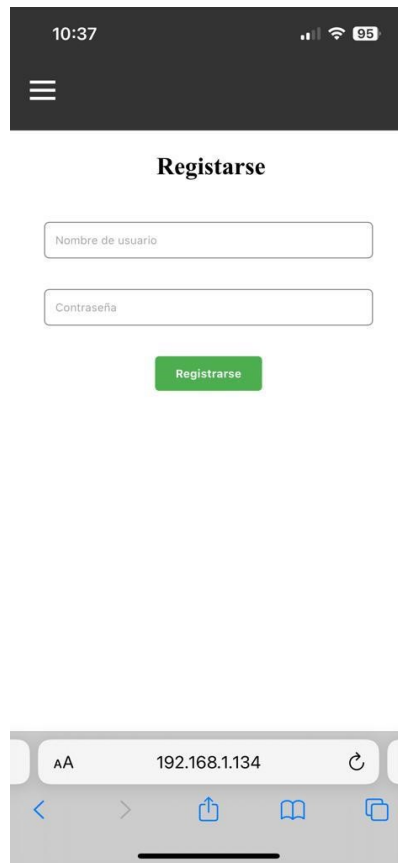


Figura 6.17: Pantalla de registro móvil

CAPÍTULO 7

Validación

Como se mencionó al inicio de esta memoria, cuando se explicaba la metodología a seguir, una vez terminada la fase de implementación de la plataforma, se someterá a evaluación. Aunque hay diferentes formas de evaluarla, se va a llevar a cabo una evaluación con un usuario lo más parecido al usuario modelo, descrito en el apartado 3.2. El usuario que participará en la validación de la plataforma será un hombre de 52 años que frecuentemente usa el navegador de los diferentes dispositivos que posee y que va a la playa cuando tiene algún rato libre.

Para realizar la evaluación se le proporcionará al usuario un dispositivo con acceso a la aplicación web y se le pedirá que interactúe con ella realizando diferentes tareas.

Lo primero que se le pide es que acceda a la web, que consulte el estado de varios parkings y que, si no sabe cómo llegar a alguno de ellos, que realice lo que crea conveniente para averiguarlo. El usuario detecta rápidamente los parkings en los que quedan pocas plazas libres y en los que, en cambio, hay gran cantidad de plazas libres. El usuario comenta que el color del número de plazas libres facilita la detección de los parkings más y menos ocupados, incluso comenta que no le importaría que no se mostrara el número de plazas totales ya que a él solo le interesan las libres. Por último, el usuario hace clic intencionadamente sobre la dirección de uno de los parkings y comenta que es muy cómodo que se abra directamente *Google Maps*.

Ahora se le pide al usuario que inicie sesión. El usuario coge el dispositivo y tras un vistazo rápido, realiza correctamente y sin ningún error el proceso. Al llegar a la pantalla de inicio de sesión el usuario se da cuenta de que no tiene cuenta por lo que procede sin pensarlo a clicar en el botón de registrarse y rellenar el formulario. Cuando el usuario hace clic en el botón de registrarse, enseguida ve la tabla con los parkings y la nueva columna.

Como ya se ha registrado, se le pide al usuario que añada a favoritos un par de parkings y que luego consulte la lista de sus parkings favoritos. El usuario añade 2 parkings a favoritos y comenta que el botón en forma de estrella le resulta familiar de otras plataformas. El usuario ahora despliega el menú y selecciona el apartado *Parkings Favoritos*. Como pega, el usuario comenta que le gustaría poder ordenar los parkings por nombre, o por número de plazas libres.

Como última acción se le pide al usuario que registre un parking con unos datos que se le proporcionan y que cuando lo haga, que cierre sesión. El usuario realiza sin problemas la tarea y comenta que es bastante fácil dar de alta un parking y que si en algún momento algún usuario no registraría un parking ya existente. Se le indica que en caso de intentar registrar un parking con una misma dirección le mostraría un mensaje de error y no lo añadiría. También comenta que al igual que con el color amarillo y el

botón en forma de estrella de la columna favoritos, el color rojo y el color verde de los botones de cerrar e iniciar sesión le recuerdan a muchas otras plataformas que usa.

Ahora que ya se ha validado la aplicación web, se va a probar el detector de matrículas que se ha implementado.

Para la validación de este detector se han realizado ejecuciones con diferentes imágenes. Como se ha comentado, en ocasiones no se detecta la matrícula o incluso la lectura de esta no es correcta. Para comprobar su funcionamiento se ha usado la función **imshow()** mostrando en la imagen principal el contorno de la matrícula que se ha detectado y transcribiendo esta matrícula a un archivo de texto. Es en este punto donde se ha detectado que en ocasiones no se detecta el contorno de la matrícula o que incluso en ocasiones no detecta correctamente un carácter.

CAPÍTULO 8

Conclusión

Una vez ya se ha desarrollado la solución al problema planteado al principio de la memoria y se ha validado, en este apartado se expondrán las conclusiones que se extraen del trabajo realizado.

En este Trabajo de Fin de Grado se ha realizado una plataforma desde cero, y tal y como se comentó al inicio del documento, se ha seguido la metodología DCU. Seguir esta metodología ha ayudado a entender lo importante que es conocer lo máximo posible a los usuarios a los que va dirigida la plataforma y conocer sus necesidades y los objetivos que tiene.

En la metodología DCU hay un par de fases que pueden parecer menos importantes a la hora de realizar una aplicación web pero que realmente son muy útiles. Estas fases son la fase de análisis y la fase de evaluación. Estas fases son interesantes pues los desarrolladores de la plataforma pueden estar más familiarizados con algunas características y/o funcionalidades que los usuarios finales, que no tienen por qué tener los mismos conocimientos que los desarrolladores. También gracias a la última fase de esta metodología el usuario puede proporcionarle ideas a los desarrolladores ya sea porque estos no consideraban que fueran importantes o porque no las habían pensado.

Una vez desarrollada la plataforma y validada, se ha comprobado que la plataforma es capaz de cumplir todos los objetivos declarados en el apartado 1.2 y que, por tanto, se considera que es una aplicación útil y usable para el usuario ya que se ha seguido la metodología mencionada anteriormente.

Durante la realización de este trabajo se han llevado a cabo diferentes conocimientos adquiridos en diferentes asignaturas del grado. Aunque esta plataforma no tiene una base de datos muy compleja, se han utilizado conocimientos adquiridos en la asignatura **Bases de Datos y Sistemas de Información**, en cuanto al diseño y desarrollo de las interfaces ha ayudado haber cursado las asignaturas **Interfaces Persona Computador**, **Desarrollo Centrado en el Usuario** y **Diseño de Sitios Web**, y como es obvio también tiene una gran relación con **Desarrollo Web**, pues aportan conocimientos de JavaScript, CSS, HTML...

Como conclusión, el desarrollo de este TFG ha sido un desafío pues se han puesto en común por primera vez algunos de los conocimientos que se han mencionado en el párrafo anterior. En ciertos elementos ha sido un proceso iterativo pues han estado en constantes mejoras. Por último, destacar que, aunque ha sido un desafío, ha sido sobre todo, una experiencia muy gratificante.

CAPÍTULO 9

Trabajo futuro

Actualmente la plataforma cumple con la totalidad de los objetivos que se plantearon al inicio del documento, aun así, existen varias funcionalidades extras que se pueden añadir y mejorar.

Una de las posibles mejoras se detectó en la fase de validación, se podrían añadir filtros de ordenación en la lista de parkings.

Ahora mismo no se ofrece demasiada información sobre los parkings registrados. En la base de datos se almacena información que no se muestra al usuario, como, por ejemplo, la descripción de estos. En un futuro se podría actualizar la base de datos para que también almacene algunas fotos del parking y realizar una interfaz en la que se muestren detalladamente todos los datos del parking.

Una de las principales funcionalidades que se podrían añadir sería relacionada con el detector de matrículas. Aunque ahora es capaz de detectar y leer algunas matrículas, con otras es incapaz de hacerlo, por tanto, lo mejor sería entrenarlo con algoritmos de aprendizaje automático para asegurarnos la correcta lectura y detección de las matrículas. Para llevar a cabo la implementación en barreras reales de este lector podríamos instalar este detector en una barrera en la que se realice una foto cuando el usuario pulse el botón que haya en la barrera (como se hace en muchos parkings) y luego se podría realizar la gestión de diferentes maneras, imprimiendo un tique con el número de matrícula detectado y actualizar las plazas libres o, simplemente, actualizar el valor de plazas libres.

Bibliografía

- [1] INE (2022). Datos demográficos de Xàbia. Recuperado de: <https://www.ine.es/jaxiT3/Datos.htm?t=2856>.
- [2] Verónica Blasco (2019, Agosto 08). Verano en Xàbia: Un 331% más de población y mismos recursos Xàbia.com, Recuperado de: <https://www.javea.com/verano-en-xabia-un-331-mas-de-poblacion-y-mismos-recursos/>.
- [3] Jordi Sánchez. En busca del Diseño Centrado en el Usuario (DCU): definiciones, técnicas y una propuesta. No Solo Usabilidad, Recuperado de: http://www.nosolousabilidad.com/articulos/dcu.htm?utm_source=iNeZha.com&utm_medium=im_robot&utm_campaign=iNeZha.
- [4] Universitat Oberta de Catalunya (UOC). *La técnica de personas*. Recuperado de: <http://design-toolkit.uoc.edu/es/persona/>.
- [5] Universitat Oberta de Catalunya (UOC). *Escenarios*. Recuperado de: <http://design-toolkit.uoc.edu/es/escenarios/>.
- [6] Lucidchart. Página oficial de Lucidchart. Recuperado de: <https://www.lucidchart.com/>.
- [7] Justinmind. Página oficial de Justinmind. Recuperado de: <https://www.justinmind.com/>.
- [8] OpenCV Documentation. *OpenCV: Image Filtering*. Recuperado de: https://docs.opencv.org/4.x/d4/d86/group_imgproc__filter.html#ga9d7064d478c95d60003cf839430737ed.
- [9] OpenCV Documentation. *OpenCV: Image Processing*. Recuperado de: https://docs.opencv.org/3.4/d2/d96/tutorial_py_table_of_contents_imgproc.html.
- [10] PHP.net Documentation. *PHP: PDO - Manual*. Recuperado de: <https://www.php.net/manual/es/book.pdo>.
- [11] PHP.net Documentation. *PHP: MySQLi - Manual*. Recuperado de: <https://www.php.net/manual/es/book.mysqli.php>.

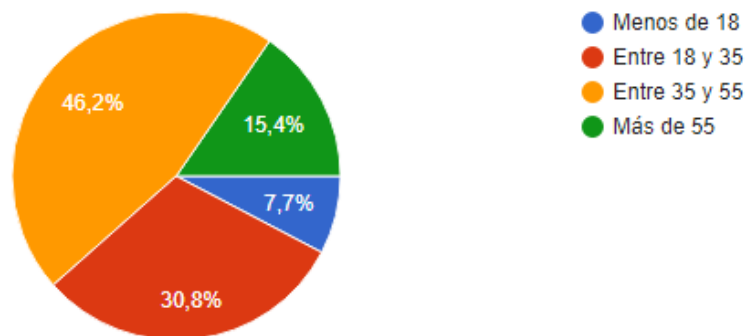
APÉNDICE A

Detalle del cuestionario

Como se ha mencionado en el apartado 3.1, se ha realizado un cuestionario con el fin de conocer mejor al usuario y sus necesidades. En dicho apartado solo aparecían una pocas preguntas, es por eso que en este apéndice se recogen todas las preguntas realizadas, así como las respuestas obtenidas.

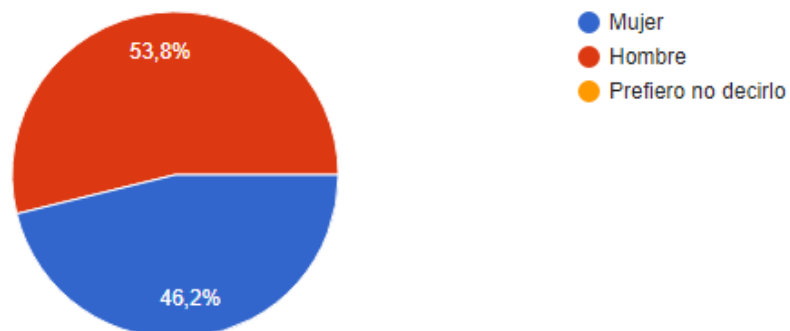
¿Cuál es tu edad?

13 respuestas



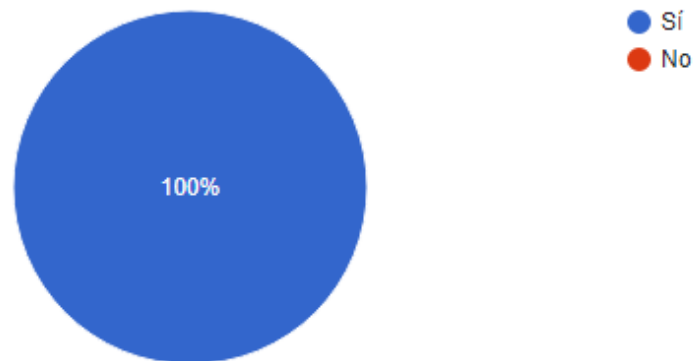
¿Con qué género te identificas más?

13 respuestas



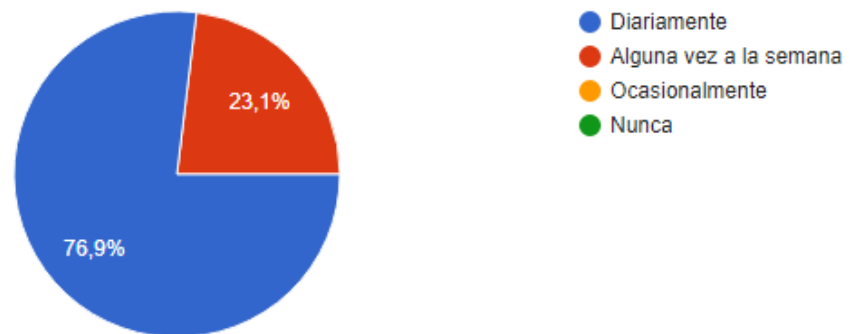
¿Dispones de algún dispositivo con conectividad a internet?

13 respuestas



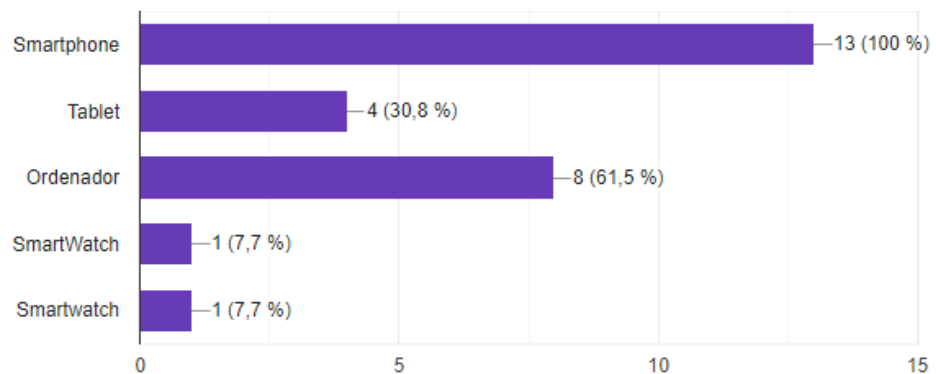
¿Con que frecuencia usas el navegador del dispositivo?

13 respuestas



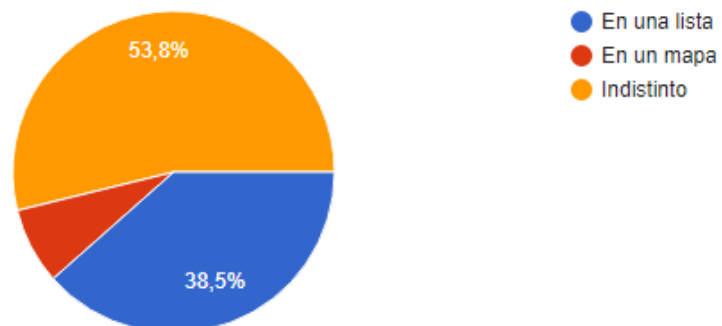
¿Qué dispositivo usas para conectarte a internet?

13 respuestas



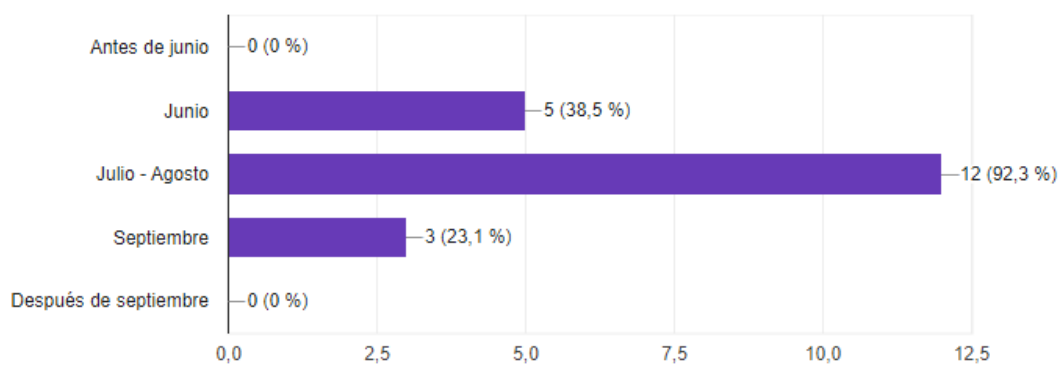
Si por ejemplo, estas buscando un hotel en TripAdvisor, ¿Cómo prefieres obtener los resultados?

13 respuestas



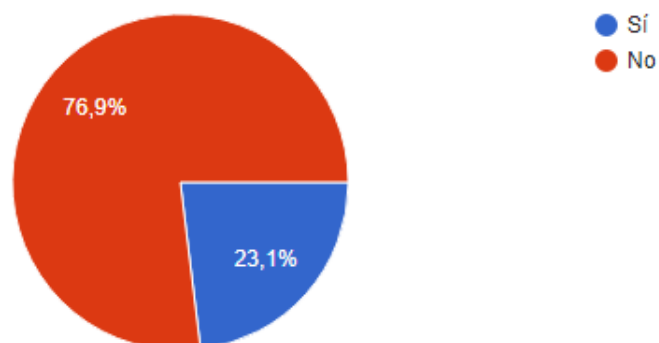
¿Cuándo sueles ir a la playa?

13 respuestas



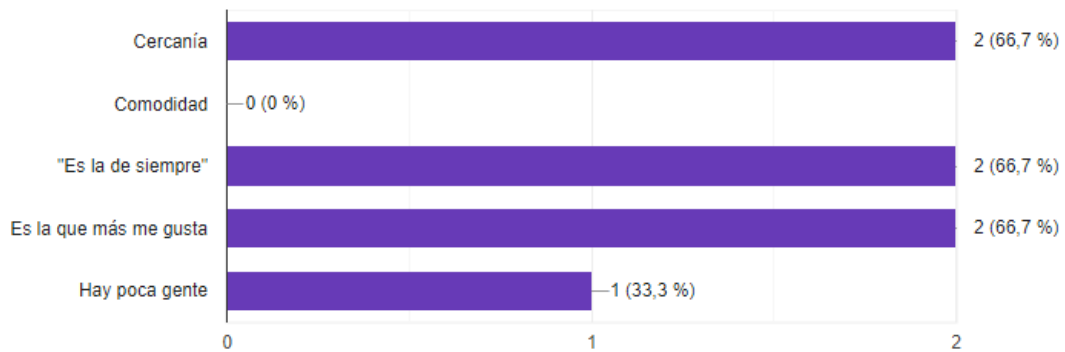
¿Vas siempre a la misma playa?

13 respuestas



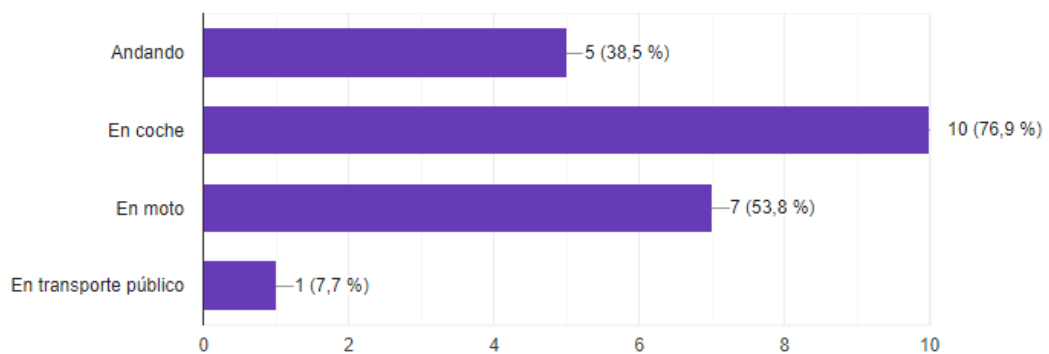
En caso de ir siempre a la misma playa, ¿Por qué repites siempre?

3 respuestas



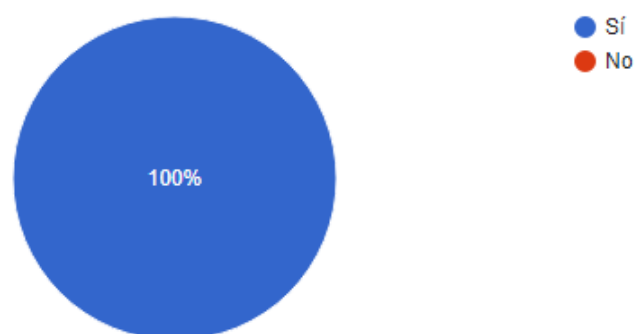
¿Cómo sueles ir a la playa?

13 respuestas



¿Supone algún inconveniente para ti tener que aparcar lejos de la playa? (supón que debes andar unos 15-20 minutos)

13 respuestas



¿Qué inconveniente?

5 respuestas

Si voy a la playa no me apetece volver al coche y llegar sudando

Me cuesta andar mucho y voy a la que tengo al lado de casa

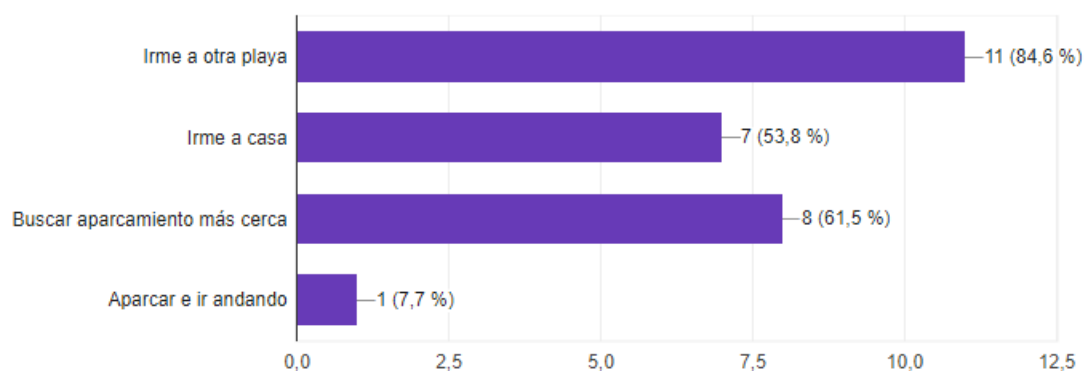
Me gusta ir a la playa para refrescarme, no me gusta irme de allí sudando

Si voy a refrescarme no quiero volver sudando

Normalmente suelo ir a la playa a las 11:30 de la mañana y salgo de la playa sobre las 12:30-13:00, he estado toda la mañana refrescandome y justa cuando voy a volver a casa no quiero empezar a sudar. Encima a la 13 es cuando más calor hace

En caso de tener que aparcar a unos 15-20 min andando de la playa, ¿Qué haces?

13 respuestas



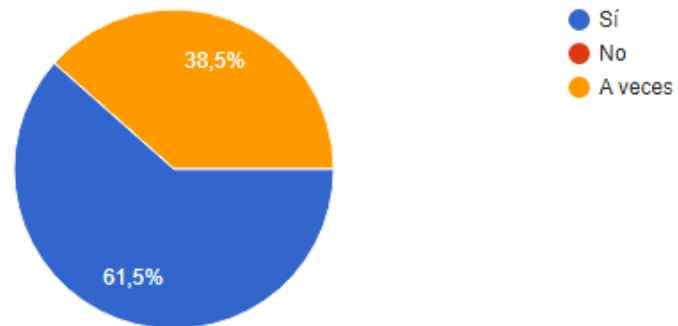
¿Alguna vez has decidido no ir a alguna playa por la dificultad a la hora de encontrar sitio para aparcar o la lejanía del parking?

13 respuestas



¿Si pudieras consultar el estado del parking de las playas a las que vas antes de ir, lo harías?

13 respuestas



¿Te gustaría añadir algo en relación a las preguntas anteriores?

0 respuestas

APÉNDICE B

Objetivos de Desarrollo Sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	Alto	Medio	Bajo	No procede
ODS 1. Fin de la pobreza.				X
ODS 2. Hambre cero.				X
ODS 3. Salud y bienestar.			X	
ODS 4. Educación de calidad.				X
ODS 5. Igualdad de género.				X
ODS 6. Agua limpia y saneamiento.				X
ODS 7. Energía asequible y no contaminante.				X
ODS 8. Trabajo decente y crecimiento económico.				X
ODS 9. Industria, innovación e infraestructuras.				X
ODS 10. Reducción de las desigualdades.				X
ODS 11. Ciudades y comunidades sostenibles.		X		
ODS 12. Producción y consumo responsables.				X
ODS 13. Acción por el clima.	X			
ODS 14. Vida submarina.		X		
ODS 15. Vida de ecosistemas terrestres.		X		
ODS 16. Paz, justicia e instituciones sólidas.				X
ODS 17. Alianzas para lograr objetivos.				X

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

En 2015 la ONU estableció 17 Objetivos de Desarrollo Sostenible con los que impulsar el crecimiento económico, el compromiso con las necesidades sociales y la protección del medio ambiente.

La intención de este TFG es intentar ayudar al usuario en la elección del parking al que acudir e incluso guiarlo si fuera necesario con la ayuda de *Google Maps*.

El verano de 2022 fue el verano más caluroso en España y en Europa desde que se registran las temperaturas. Esto fue provocado, en parte, por la emisión de gases que producen un efecto invernadero provocando un calentamiento global.

Por eso, y como se puede apreciar en la tabla, el objetivo que más afín tiene con este TFG es el **ODS 13. Acción por el medio ambiente**. Mediante el desarrollo de esta plataforma se pretende que el usuario antes de dirigirse a un parking consulte nuestra plataforma para ver que parkings tienen plazas libres y cuantas de ellas hay. Con esto

logramos reducir la cantidad de kilómetros realizados en coche y por tanto la contaminación que este provoca. También con la opción que se ofrece de abrir *Google Maps* con la dirección del parking se evita que el usuario que no sepa llegar al parking se desvíe o, como en el caso anterior, haga kilómetros innecesarios.

Por esta misma razón, se pueden asociar los objetivos **ODS 11. Ciudades y comunidades sostenibles**, **ODS 14. Vida submarina** y **ODS 15. Vida de ecosistemas terrestres** a este TFG. Disminuir la contaminación generada por los coches afectará directamente con lograr una ciudad más sostenible. Por tanto, las emisiones de gases de efecto invernadero también afectarán al cambio climático y con este a la vida submarina y a los ecosistemas terrestres por lo que la reducción de estos gases beneficiará a ambos ecosistemas. Además, el establecimiento de las plazas de parking en sitios habilitados para ello reducirá la cantidad de coches aparcados en zonas prohibidas y que, por tanto, perjudiquen en ocasiones la vida de ecosistemas terrestres y lleguen, incluso, a provocar accidentes.

Podemos relacionar todos estos objetivos mencionados con el objetivo **ODS 3. Salud y bienestar**. Con la reducción de la contaminación, la mejora del ecosistema submarino y terrestre repercutirá en nuestra salud y bienestar.

Por último, esta plataforma no tiene relación con el objetivo **ODS 1. Fin de la pobreza** pero al ser una plataforma gratuita no perjudicará a aquellas personas con menor poder adquisitivo. Además, la reducción de kilómetros realizados disminuirá el consumo de combustible por lo que también provocará una mejora en la economía del usuario. Lo mismo pasa con el objetivo **ODS 5. Igualdad de género**, a pesar de no tener relación con este, en la plataforma no se distingue entre géneros por lo que, aunque no logrará la igualdad de género en la sociedad, sí que lo logrará en la plataforma. Tampoco tiene relación directa con el objetivo **ODS 2. Hambre cero** aunque la reducción de contaminación provocará una mejora de los ecosistemas y, por tanto, una mejora en la cantidad y calidad de los alimentos de estos, en resumen, aunque no pondrá fin al hambre al aumentar la cantidad de, por ejemplo, peces, aumentará la posibilidad de llegar a más población.

Aunque los objetivos restantes no tienen prácticamente o no tienen ninguna relación con este trabajo, son igual de importantes que los que se han mencionado en este apartado.