Universidad Internacional de La Rioja
Escuela Superior de Ingeniería y Tecnología

Grado en Ingeniería Informática

Desarrollo de una Aplicación Web para la gestión de creación de *Parkings*

Trabajo fin de estudio presentado por:	Miguel Soto Lombris
Línea de investigación:	Desarrollo de Aplicación Web
Director/a:	D. Luis Pedraza Gómara
Fecha:	Abril 2023
Repositorio:	https://github.com/MSotoL/parking_lot

Agradecimientos

Quiero expresar mi profunda gratitud a todos aquellos que me han apoyado durante mis estudios universitarios.

A todos los profesores que he tenido el placer de conocer, por su dedicación y paciencia al guiarme y enseñarme todo lo que necesitaba saber para llevar a cabo esta aventura.

A mi director de trabajo final de estudios, Luis Pedraza Gómara, por su orientación y apoyo. Gracias por su inestimable ayuda, su paciencia y por tener siempre palabras tranquilizadoras y motivadoras.

Y, por supuesto, a mi familia, en especial a mi TODO, Ana, sin ella esto no habría sido posible, y a mis hijas María y Adriana, sin su paciencia y comprensión no habría podido terminar este viaje con éxito. A las tres, gracias por apoyarme, aguantar mis momentos de tensión, no daros nunca por vencidas y por todos y cada uno de los días que he vivido a vuestro lado, **sois mi luz**.

"El éxito es la suma de pequeños esfuerzos repetidos día tras día"

- Ralph Waldo Emerson-

Cada día, durante mis estudios universitarios, hice pequeños (o grandes) esfuerzos para aprender y crecer, y gracias a las personas que mencioné anteriormente, pude lograr el objetivo que un día comenzó como una aventura y que me ha llevado a conseguir un resultado soñado.

A todos aquellos que han sido parte de mi camino, les agradezco de todo corazón su ayuda y paciencia.

Por último no quiero dejar pasar este momento sin recordar a mis padres, allá donde estéis sé que estáis orgullosos de mí, os echo de menos.

¡GRACIAS!

Resumen

El objetivo del presente proyecto consiste en analizar, desarrollar y validar una aplicación web que permita la gestión integral de la información del negocio de *parkings* y facilite que los usuarios puedan administrar y optimizar su gestión.

La aplicación cuenta con una función de diseño de *layout* de las plantas de los *parkings*, que permite planificar la disposición de los espacios de manera fácil y rápida, optimizando el uso del espacio disponible y aumentando la capacidad de estacionamiento de este.

Para llevar a cabo el mismo se ha aplicado una metodología que toma como base el modelo iterativo incremental que divide el proyecto final en trabajos más pequeños.

Finalmente, para evaluar la aplicación se han realizado pruebas de funcionamiento y pruebas de aceptación por usuarios.

Se puede comprobar que la aplicación cumple con el objetivo de aglutinar en una sola herramienta diferentes funcionalidades para una gestión eficaz del negocio de *parkings.*

Palabras clave: Gestión de *Parkings*, Diseño de *Layout*, Desarrollo Web, *Django*, *Python*, *KonvaJS*

Abstract

The aim of this project is to analyse, develop and validate a web application that allows the integral management of the information of the car park business and facilitates users to administer and optimise its management.

The application has a layout design function for the car park floors, which allows the layout of the spaces to be planned easily and quickly, optimising the use of the available space and increasing its parking capacity.

A methodology based on the incremental iterative model, which divides the final project into smaller jobs, has been applied to carry out the project.

Finally, to evaluate the application, functional tests and user acceptance tests have been carried out.

It can be seen that the application meets the objective of bringing together in a single tool different functionalities for the efficient management of the car park business.

Keywords: *Parking Lot Management, Layout Design, Web Development, Django, Python, KonvaJS*

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

El presente trabajo tiene por objeto analizar, diseñar y desarrollar una aplicación web que permita gestionar la información integral de los negocios de *Parkings*.

El objetivo general de la aplicación consiste en dotar al usuario de una única herramienta que permita gestionar de la información necesaria para el correcto funcionamiento de un negocio de *y* realizar las labores de diseño del *layout* de estacionamiento de este.

Se pretende desarrollar una herramienta que facilite el trabajo y lo simplifique, puesto que para su utilización no serán necesarios conocimientos técnicos específicos.

1.1. Justificación del tema elegido

El desarrollo de una aplicación que englobe diferentes herramientas es esencial en la época en la que nos encontramos, en la que la tecnología está cada vez más presente en nuestras vidas. Esta aplicación tiene como objetivo principal facilitar el trabajo a los usuarios, ya que permite tener acceso a una variedad de funciones y herramientas desde un solo lugar, con una *interfaz* simple e intuitiva, ahorrando tiempo y esfuerzo, dotando al usuario de una rápida adaptación a las diferentes *interfaces* y proporcionando un sistema de gran facilidad de uso.

La aplicación permite al usuario gestionar toda la información relacionada con el *parking* sobre el que esté trabajando. Podrá dar de alta diferentes *parkings*, seguidamente el usuario indicará al sistema la información sobre los edificios que forman parte de este. También podrá indicar las plantas que tiene cada edificio y, en cada planta, indicar la información de las plazas que forman parte de cada zona. Las plazas podrán ser de tres tipos exclusivamente: Grandes, Pequeñas y Discapacitados. Además, la aplicación incorpora una herramienta de diseño de *layout* de cada planta, la cual permite al usuario optimizar el espacio de esta y gestionar los estacionamientos a su conveniencia.

Al tener acceso a diferentes funcionalidades, o herramientas, en un solo lugar, los usuarios pueden realizar su trabajo de una manera más eficiente y rápida, con el ahorro de tiempo que esto conlleva. Por otro lado, el hecho de englobar en una única aplicación varias funcionalidades ayuda a reducir costes al eliminar la necesidad de comprar y mantener diferentes herramientas y programas específicos independientes.

1.2. Problema y finalidad del trabajo

El presente trabajo pretende cubrir las necesidades de un nicho de negocio que requiere el uso de diferentes herramientas para la gestión integral de la información de un *parking*, y además permita realizar el diseño de un plano para ubicar los diferentes estacionamientos.

Gracias a las nuevas tecnologías, y al avance de estas, es necesario dotar a las empresas de herramientas que permitan avanzar en su campo. La falta de herramientas similares en el sector limita la capacidad de la empresa para competir eficazmente y el presente TFE ayudará a superar esta limitación. Dichas nuevas tecnologías han transformado el día a día en la gestión de la información y la manera en la que se gestiona la misma, cada vez es más importante estar conectado y tener acceso a herramientas y recursos en línea.

Tal como define Zofío en su libro Aplicaciones Web:

> "Se denomina aplicación web al software que reside en un ordenador, denominado servidor web, que los usuarios pueden utilizar a través de Internet o de una intranet, con un navegador web, para obtener los servicios que ofrezca." (Zofío, 2013).

La aplicación mantiene esta premisa en consideración, permitiendo a los usuarios acceder a las diferentes funcionalidades desde cualquier lugar y en cualquier momento. El trabajo realizado en este TFE aborda el desarrollo de una aplicación accesible mediante internet, desarrollado en *Django* como lenguaje para el entorno *back-end*, y con HTML como lenguaje base, el uso de CSS y *Javascript* para desarrollo del entorno *front-end*.

Cabe destacar que una aplicación como la desarrollada en el presente trabajo ayuda a una empresa del sector del *parking* a anticiparse en sus estrategias y objetivos a largo plazo, por la flexibilidad que aporta en la gestión del negocio.

1.3. Objetivos del TFE

El presente trabajo tiene por objeto analizar, diseñar y desarrollar una aplicación web, que aglutine en una sola herramienta procesos que hasta ahora se debían realizar por separado, tales como la gestión de la información de un *parking* y el diseño del *layout* de cada planta. Para llevar a cabo el trabajo se plantea un objetivo general del proyecto que se subdividirá en objetivos específicos.

1.3.1. Objetivo general

El objetivo general de este trabajo es desarrollar una aplicación web que facilite la gestión integral de la información relacionada con un *parking* y permita el diseño del *layout* de las diferente plantas.

Para lograr este objetivo la aplicación dispondrá de diferentes *interfaces* de usuario que faciliten el trabajo a realizar, permitiendo, de una manera sencilla, dotar al sistema de la información necesaria que requiera. Dicha aplicación almacenará la información en una base de datos que garantice la integridad y disponibilidad de los datos.

1.3.2. Objetivos específicos

Los objetivos específicos que permiten alcanzar el objetivo general van ligados a las diferentes fases o etapas del desarrollo de una aplicación:

1. Analizar los requisitos de usuario: En esta etapa se recopila información sobre las necesidades del usuario y se define el alcance del proyecto. Se lleva a cabo un análisis detallado de los requisitos funcionales y no funcionales de la aplicación, se establecen los objetivos del proyecto, se identifican los usuarios y se definen las restricciones del proyecto.
2. Diseñar la aplicación: En esta etapa se crea una arquitectura y una *interfaz* de usuario para la aplicación. Se diseñan los componentes de la aplicación, se eligen las tecnologías a utilizar y se definen los procesos de negocio.
3. Desarrollar la aplicación: En esta etapa se escribe el código y se integran las diferentes partes de la aplicación. Se desarrollan las distintas funcionalidades y se implementan los procesos de negocio.
4. Realizar pruebas: En esta etapa se realizan pruebas para asegurar que la aplicación funciona correctamente. Se realizan pruebas unitarias, de integración y de aceptación para comprobar que la aplicación cumple con los requisitos especificados en la etapa de análisis de requisitos.
5. Sacar conclusiones: Esta fase pretende aglutinar las conclusiones obtenidas durante la elaboración del presente TFE y plasmar posibles trabajos futuros y mejoras.

1.4. Estructura del TFE

A partir de este capítulo el resto del presente TFE se estructura de la siguiente manera. En el Capítulo 2 se hace un repaso al MARCO TEÓRICO en el que se basa este trabajo, realizando un recorrido por la evolución de la industria de los parkings, evolución de los diseños de estos, análisis de la metodología utilizada y se incide en la base de los lenguajes y *frameworks* de desarrollo utilizados y gestión de la información mediante bases de datos.

El Capítulo 3, titulado CONTEXTUALIZACIÓN, realiza un estudio de diferentes herramientas existentes en el mercado que hay relacionadas con los negocios de *parkings*.

El Capítulo 4 detalla el DISEÑO DE LA PROPUESTA haciendo hincapié en la metodología utilizada para llevar a cabo el análisis, modelos y diagramas en el que se basa el diseño y, finalmente, se expone el desarrollo de la aplicación, *interfaces* gráficas, etc.

El Capítulo 5, se centra en la validación de la aplicación. Una vez finalizado el desarrollo es básico realizar una EVALUACIÓN DE LA APLICACIÓN, en este capítulo se desglosan las diferentes pruebas que se han realizado para validar el desarrollo.

Se concluye el TFE con el Capítulo 6, denominado CONCLUSIONES Y TRABAJO FUTURO, en el que, como su nombre indica, se reflejan las conclusiones obtenidas durante la elaboración del trabajo, así como una recopilación de posibles mejoras y trabajos futuros a realizar sobre el mismo.

2. MARCO TEÓRICO

En el presente capítulo se detallan los aspectos que van a fundamentar el TFE. El objetivo es dotar a las empresas que gestionen, o sean propietarias de *parkings*, de una aplicación web que les permita realizar el diseño de un *layout* del mismo y trasladar toda esa información a una base de datos, sin la necesidad de tener unos conocimientos avanzados sobre ingeniería o sobre aspectos informáticos de creación de bases de datos. Tal como se define en el artículo "Historia de las bases de datos" de Cultura Informática una base de datos es "un conjunto de información relacionada que se encuentra agrupada o estructurada." (Cultura Informática, 2011). El hecho de utilizar una base de datos facilitará la gestión de la información almacenada en el sistema.

Para estructurar el presente documento se ha realizado el análisis de la evolución de la tecnología utilizada en este ámbito y de la forma de trabajo que se llevaba a cabo, también se hace un repaso por las tecnologías utilizadas en el análisis y desarrollo de la aplicación.

2.1. Evolución de la industria del aparcamiento e importancia del sector

En este apartado se aborda una introducción a la industria del aparcamiento, la evolución histórica de la misma y se plasman unas bases para determinar la importancia del sector en la actualidad.

2.1.1. Evolución de la industria de los *parkings*

La movilidad es la cualidad de movible (Real Academia Española, 2022b), que a su vez se define como un adjetivo y determina "que por sí puede moverse, o es capaz de recibir movimiento por ajeno impulso" (Real Academia Española, 2022a). Dicha movilidad permite desplazarse a las personas a diferentes lugares para satisfacer los propósitos oportunos de cada uno. Esto hace que sea de gran importancia facilitar, potenciar y agilizar la movilidad, pues es un elemento clave para el desarrollo de las personas, así como de la comunidad a la que pertenezcan.

Tal como se detalla en el artículo *The Evolution of Urban Mobility* el problema de la movilidad se inició cuando las personas adquieren automóviles para desplazarse, es en ese momento cuando surgió la necesidad de disponer de espacios adecuados para estacionar y guardar los diferentes medios de transporte que se empezaron a utilizar.

> "Las primeras fases del crecimiento económico urbano conducen a un rápido aumento de la propiedad y el uso del automóvil, con el consiguiente interés político por hacer frente al "inevitable" gran crecimiento del tráfico de vehículos de motor, para evitar que la ciudad "se detenga". Esto se asocia a menudo con el desarrollo o la expansión de una industria automovilística nacional. La solución a este problema se ve en términos científicos y de ingeniería: requiere invertir en un gran programa de construcción de carreteras urbanas y medidas para maximizar la capacidad de los vehículos en las calles urbanas existentes, con el apoyo de grandes aumentos en la provisión de aparcamientos, especialmente en los principales destinos de los viajes". (Jones, 2014)

En los inicios de la aparición del automóvil, este, era considerado un artículo de lujo, al alcance de unos pocos. Es a finales del siglo XIX, cuando empiezan a proliferar los automóviles por las ciudades, cuando surgen las primeras compañías de construcción de estos. (Máxima, 2022)

Con la revolución de la industria automovilista, y la aparición de vehículos asequibles a todos los bolsillos, así como el crecimiento demográfico, la necesidad de aparcamientos especiales se hizo patente. Se daban dos tipos de situaciones, las personas necesitaban un aparcamiento durante el día, cerca de dónde estaban trabajando y por la noche, cerca de su domicilio. En principio se utilizaban establos, debido a que el automóvil sustituyó a los carruajes, pero se daba la circunstancia de que dichos establos no se encontraban donde los conductores querían que estuvieran. Esta situación agravó el desorden en las ciudades, por lo que la necesidad de contar con espacios específicos de estacionamiento, o aparcamiento, se hizo cada vez más acuciante.

A principios del siglo XX surgieron los primeros parkings construidos expresamente para los automóviles: al aire libre, de varios niveles e, incluso, subterráneos.

Tal como se informa en el blog de Metinvest, “en 1901 se construye el primer aparcamiento cerrado en Londres. Se trataba de un edificio de siete plantas con 100 aparcamientos. Debido al éxito de la iniciativa, más adelante, construye dos parkings más para 230 y 200 plazas. En España el primer aparcamiento subterráneo se construye en Barcelona, en la casa Milá.” (Metinvest, 2020).

2.1.2. Importancia del sector

El sector del aparcamiento está muy fragmentado, se puede observar que existen multitud de empresas pequeñas que gestionan un único aparcamiento, pero al tiempo es un sector que concentra unas pocas compañías que operan y controlan la mayoría del mercado (ParkingYa, 2019). Por este motivo es muy difícil obtener datos y conocer con precisión el número de aparcamientos gestionados.

La Asociación Europea de Aparcamientos (EPA), fundada en 1983, es la organización que agrupa a las asociaciones europeas de aparcamientos. Las asociaciones nacionales miembros representan al sector del aparcamiento, formado por empresas privadas y organismos públicos que explotan, y gestionan, estructuras y servicios de aparcamiento en la vía pública y fuera de ella.

Dicha organización realiza estudios e informes estadísticos sobre el sector, en base a ello podemos consultar la oferta de plazas disponibles en los países miembros de la asociación, tal como se muestra en la Figura 1, se observa que en España hay una oferta de aproximadamente dos millones ochocientas mil plazas de aparcamiento, frente a los 47 millones de habitantes (INE. Instituto Nacional de Estadística, 2022) o más concretamente frente a los, aproximadamente, 27 millones de conductores con licencia que están censados en España (Dirección General de Tráfico (DGT), 2021).

Figura 1. Oferta de Plazas de aparcamiento

País	En plazas	En la calle	Total
Austria	361.523	193.735	555.258
Bélgica	664.331	343.480	1.007.811
Croacia	217.624	80.846	298.470
Chipre	58.528	24.003	82.531
Dinamarca	292.693	123.911	416.604
Estonia	68.435	22.108	90.543
Finlandia	445.276	129.222	574.498
Francia	1.676.318	1.028.250	2.704.568
Alemania	4.935.623	2.638.385	7.574.008
Grecia	979.308	472.215	1.451.523
Hungría	534.519	219.457	753.976
Irlanda	268.897	84.259	353.156
Italia	1.409.779	1.810.647	3.220.426
Luxemburgo	20.083	4.604	24.688
Holanda	1.085.257	431.773	1.517.030
Noruega	382.185	103.571	485.756
Polonia	1.704.192	698.848	2.403.040
PORTUGAL	**591.149**	**236.454**	**827.603**
Serbia	395.440	178.158	573.598
Eslovaquia	238.844	57.930	296.774
ESPAÑA	**1.558.712**	**1.190.311**	**2.749.023**
Suecia	933.211	227.577	1.160.788
Suiza	234.114	104.360	338.474
Reino Unido	2.700.000	1.600.000	4.300.000
TOTAL	**21.756.041**	**12.004.105**	**33.760.146**

Fuente: Elaboración de ParkingYa! a partir de datos de la *European Parking Association*

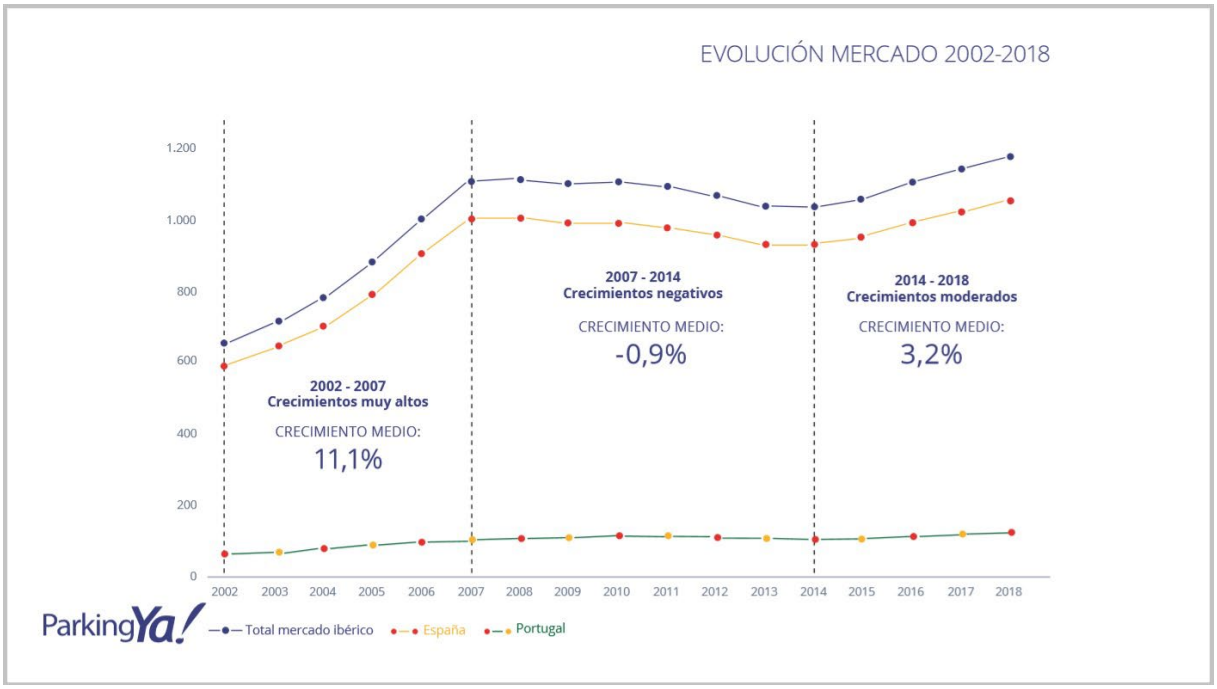
A parte de la oferta y demanda, los aparcamientos son un elemento esencial en la estrategia de movilidad de las ciudades por varias razones. La primera, y más importante, reside en que la falta de plazas de aparcamiento en las grandes ciudades conlleva un grave problema de congestión de tráfico y, por consiguiente, las retenciones y atascos. La solución a este primer problema consiste en aumentar el número de plazas de aparcamiento disponibles, es fundamental para mejorar la movilidad urbana.

El segundo gran problema recae en que la seguridad de las ciudades se ve comprometida. El número de desperfectos que se ocasionan a los vehículos, o el robo de estos, aumentará considerablemente si estos se encuentran estacionados en la calle.

Estos problemas reflejan la importancia que tiene el sector de los aparcamientos respecto a la movilidad, y otros factores, en las grandes ciudades.

Respecto a la parte económica del sector, durante el año 2018, ha facturado en torno a mil cien millones de euros (ParkingYa, 2019). Tal como podemos observar en la Figura 2, se trata de un mercado estable con crecimiento económico sostenido, lo cual nos sitúa frente a un sector con una demanda muy grande y con unos beneficios importantes.

Figura 2. Evolución del sector



Fuente: Elaboración de ParkingYa! a partir de datos de la *European Parking Association*

2.2. Evolución del diseño de *parkings*

Los aparcamientos comenzaron a surgir a principios del siglo XX, no mucho después de la creación del primer automóvil en 1885. Los primeros aparcamientos tenían un valor exclusivamente utilitario: debían funcionar como espacios para guardar coches sin prestar suficiente atención a la faceta estética.

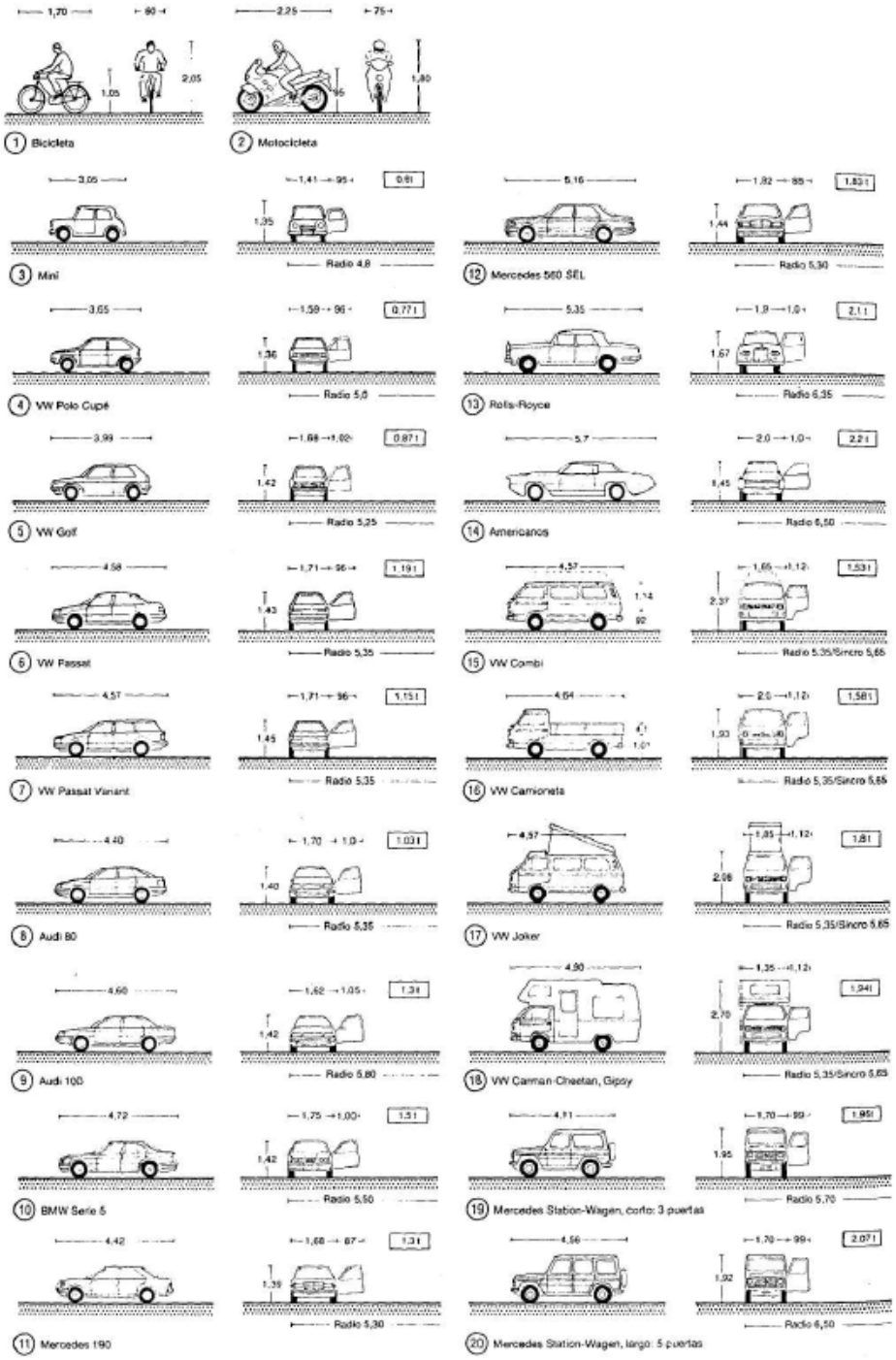
El origen de la palabra *parking*, como edificio de estacionamiento, no en el sentido del verbo aparcar, viene definido por dos palabras que se utilizaban en Estados Unidos: *to rank* (clasificar) y *to park* (aparcar) (*The MIT Press Reader*, 2020).

2.2.1. Diseño de *Layouts* de *Parkings*

Para realizar el diseño de un aparcamiento se deben tener en cuenta ciertos factores tales como el espacio necesario de maniobras, el tamaño medio de los vehículos, el tipo de edificio o espacio abierto del que se disponga, entre otros. En cualquier caso, hoy en día, es indispensable utilizar un software de diseño arquitectónico, o uno orientado específicamente al diseño de *parkings* y, por supuesto, ser consciente de una serie de información sobre dimensiones y parámetros que se deben tener en cuenta en el diseño.

Para realizar el diseño de un *parking* se debe tener en cuenta la dimensión de los vehículos que van a utilizar el mismo, en la Figura 3 se pueden observar diferentes tamaños de vehículos:

Figura 3. Tamaño de Vehículos



Fuente: Tesis Guía de aparcamiento de vehículos (Balsells, 2004)

Es imposible tener en cuenta todos los tamaños de cada vehículo en particular, por lo tanto, para realizar los diseños de los *parkings* se toman tres medidas genéricas, que engloben a todas las demás, y que pasan a ser: coche normal, coche pequeño y coche grande. De esta forma los *parkings* dispondrán de plazas para dos de esas referencias, el tamaño del coche grande y el tamaño del coche normal, que a su vez engloba a los coches de tamaño pequeño.

2.3. Metodologías de Desarrollo

El presente apartado pretende exponer la metodología de desarrollo software que se ha empleado durante el ciclo de vida del proyecto. Para conseguir los objetivos del presente proyecto se ha planteado el uso del **modelo de desarrollo iterativo incremental**, el cual está basado en el modelo de desarrollo en cascada.

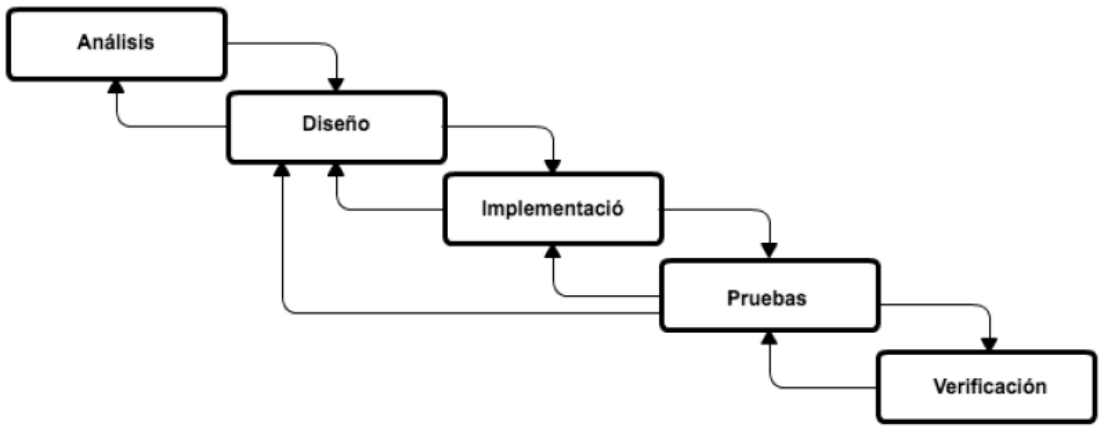
2.3.1. Modelo Iterativo Incremental

Se ha decidido implementar un modelo de desarrollo iterativo incremental para llevar a cabo el desarrollo de la aplicación, dicho modelo está basado en el modelo en cascada.

El desarrollo en cascada es un enfoque metodológico que ordena rigurosamente las etapas del proceso de desarrollo software (Pressman, 2010). Básicamente este modelo implementa un desarrollo basado en que una etapa no comienza hasta que no haya finalizado la anterior. Cuando esto ocurre se lleva a cabo una revisión que permite tomar la decisión de pasar a la siguiente fase o no.

Las fases en que se compone el modelo en cascada se pueden observar en la Figura 4, los errores que se detecten en la fase de validación implican rediseñar la propuesta y volver a codificar.

Figura 4. Modelo de Desarrollo en Cascada

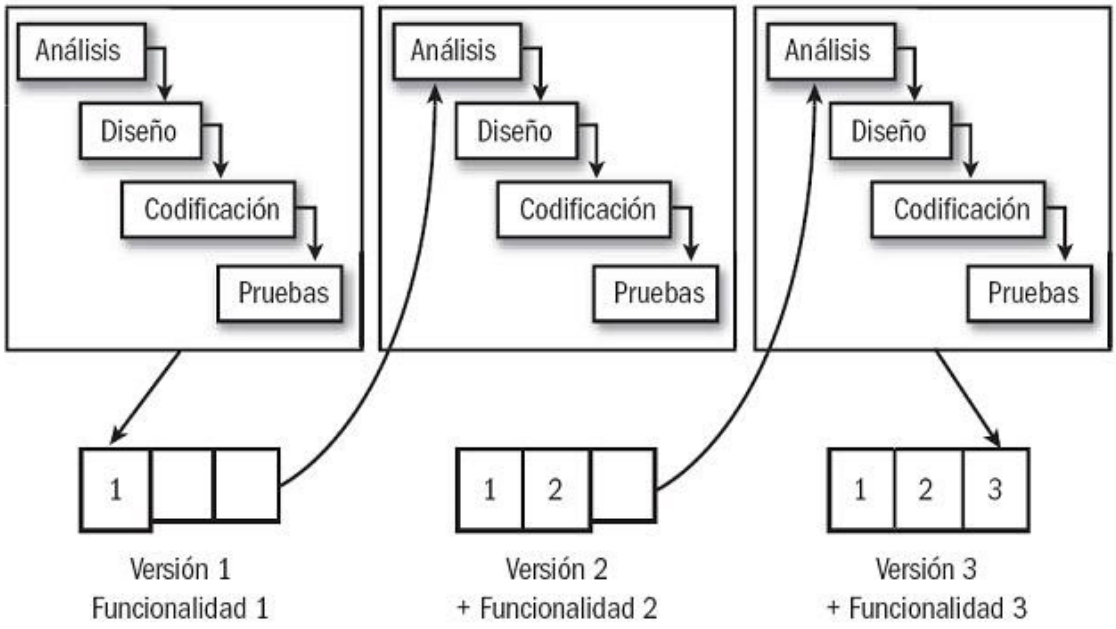


Fuente: Libro Ingeniería del Software: un enfoque práctico. (Pressman, 2010)

El modelo iterativo incremental subsana las debilidades del modelo anterior modificando la rigidez secuencial de las etapas o el tiempo para la detección y corrección de errores (Pressman, 2005).

Un proyecto desarrollado siguiendo este modelo se planifica en diferentes etapas temporales, llamadas iteraciones, en las que se repiten una serie de procesos (análisis, diseño, codificación y pruebas) (ProyectosAgiles.org, 2008).

Figura 5. Modelo de Desarrollo Iterativo Incremental



Fuente: Adaptado del libro Ingeniería del Software: un enfoque práctico. (Pressman, 2010)

Básicamente consiste en dividir el proyecto en mini proyectos más pequeños, y acotados, sobre los que se realizan los trabajos de cada iteración.

2.4. Lenguaje de Desarrollo

La elección del lenguaje de programación que se va a utilizar durante el desarrollo viene motivada por la función que se vaya a implementar en cada módulo, o sección, que forme parte de la aplicación. En este caso se observan dos elementos diferenciados que van a intervenir en la aplicación:

- *Front-End*, o desarrollo del lado cliente. Esta parte del desarrollo se centra en la parte visual de la aplicación, incluye el diseño y la creación de modelos de *interfaces* que componen la aplicación y todo el aspecto visual que engloba. Se encarga de facilitar al usuario la recopilación de la información a almacenar.
- *Back-End*, o desarrollo del lado servidor. Este parte del desarrollo es la encargada de procesar la información que maneja la aplicación, básicamente se trata del desarrollo que se ejecuta en el servidor para almacenar los datos.

2.4.1. Desarrollo *Front-End*

El *Front-End* es la parte del desarrollo web que se dedica a la parte visible de un sitio o aplicación. Incluye todo lo relacionado con la estructura de la aplicación pasando por los estilos, colores, fondos, etc. El desarrollo del lado cliente cubre la parte de la aplicación con la que interaccionan los usuarios.

Para el desarrollo de las *interfaces* de usuario se ha utilizado HTML5 (*HyperText Markup Language*, versión 5). HTML es un lenguaje de marcas, no considerado de programación, y es el código que se utiliza para estructurar, y desplegar, una página web y sus contenidos.

Con respecto a la parte visual de la página web se ha utilizado CSS3 (*Cascading Style Sheets*, versión 3), lo cual va a permitir definir, controlar y crear la parte visual de la *interfaz* de usuario. Dentro del apartado visual se ha utilizado *Bootstrap*[1], se trata de un *framework* CSS de desarrollo que, mediante el uso de una serie de componentes, dota de interactividad a la página web y facilita el diseño *responsive* de esta.

La programación de los módulos y funciones necesarias, para dotar de interactividad a la aplicación, se ha codificado con *Javascript*, el cual es un lenguaje de programación interpretado, se define como orientado a objetos, basado en prototipos, débilmente tipado y dinámico.

Una de las funcionalidades con la que se ha dotado al sistema incluye el uso de AJAX (*Asynchronous Javascript and XML*). *Ajax* permite, a una aplicación ubicada en el entorno cliente, iniciar una comunicación con el servidor web sin necesidad de realizar la recarga de las páginas.

Para el desarrollo de la *interfaz* de diseño del *layout* del *parking* se ha utilizado la librería *KonvaJS*[2], se trata de *framework Javascript* para HTML5 y *Canvas*, que permite realizar dibujos, añadiendo formas a lo que la librería llama escenario, y la utilización de capas.

[1] *Bootstrap*: https://getbootstrap.com/
[2] *KonvaJS*: https://konvajs.org/

2.4.2. Desarrollo *Back-End*

El *front-end* es la capa de aplicación que se ejecuta en el navegador del usuario, el *back-end* se encarga de procesar la información facilitada desde el *front-end*. Se puede decir que es la capa de acceso a los datos e incluye la lógica de programación que se ejecuta en el servidor web.

Para el desarrollo del lado servidor se ha utilizado el *framework* de desarrollo *Django*[3], el cual está basado en *Python*.

"*Django* es un *framework* web *Python* de alto nivel que fomenta el desarrollo rápido y el diseño limpio y pragmático" (*Django Software Foundation*, 2023). *Django* es gratuito y de código abierto.

Trabaja bajo una arquitectura denominada "*Model View Template (MVT)*" o Modelo-Vista-Plantilla. A continuación se detalla qué es cada elemento de esta arquitectura:

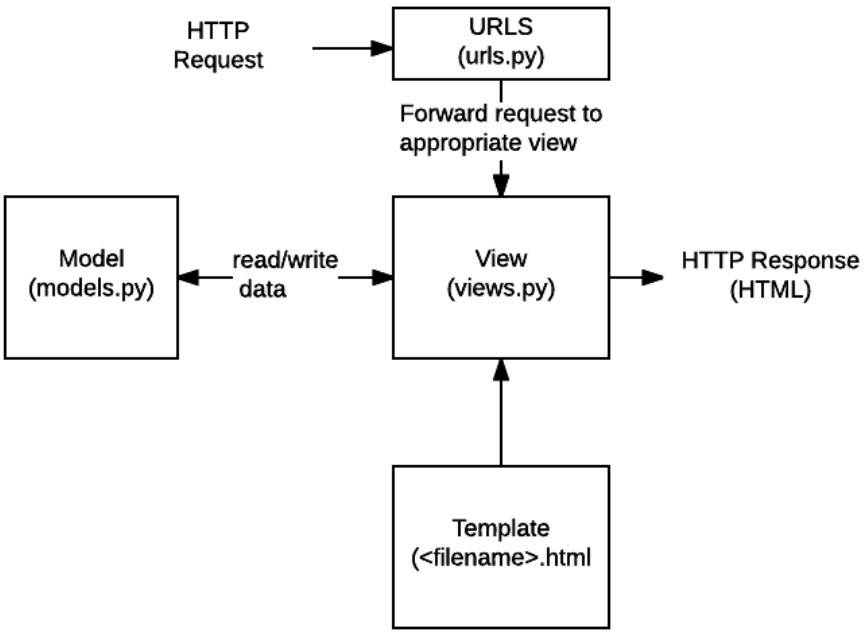
- **URLs:** *Django* utiliza un mapeador URL para redirigir las peticiones HTTP a la vista apropiada.
- **Vistas (*Views*):** Una vista es una función de gestión de peticiones que recibe solicitudes HTTP y devuelve respuestas HTTP. En la programación orientada a objetos clásica las vistas se corresponderían con los métodos de las clases.
- **Modelos (*Models*):** Los modelos son objetos de *Python* que definen la estructura de los datos de una aplicación y proporciona mecanismos para gestionar los mismos.
- **Plantillas (*Templates*):** Una plantilla es un fichero de texto que define la estructura de la página web que se va a visualizar, un documento HTML por ejemplo. La vista creará dinámicamente una página web usando una plantilla y rellenándola con datos de un modelo.

El funcionamiento general de cualquier aplicación web, básicamente, consiste en mantenerse a la espera de recibir solicitudes HTTP desde un navegador. Cuando se hace la petición, esta se resuelve en base a la URL y a la información que solicite. Seguidamente la respuesta que ha generado la aplicación se devuelve como una página web de forma que se muestren los datos

[3] *Django*: https://www.djangoproject.com/

solicitados. En la Figura 6 podemos comprobar este funcionamiento con una aplicación web basada en *Django*.

Figura 6. Arquitectura MVT de *Django*



Fuente: Introducción a *Django* - Aprende sobre desarrollo web | MDN, 2022)

El uso de un *framework* de desarrollo se reduce la carga de trabajo que está asociada al desarrollo web, en el caso del trabajo que nos ocupa se ha optado por *Django*, el cual, como ya se ha mencionado anteriormente, está basado en el lenguaje de programación *Python*.

Python es un lenguaje de alto nivel de programación interpretado, la principal cualidad de este lenguaje es la legibilidad de su código, se utiliza para desarrollar desde *scripts* hasta aplicaciones complejas de escritorio. Se trata de un lenguaje de programación multiparadigma, soporta la orientación a objetos, y la programación imperativa, es un lenguaje interpretado y multiplataforma.

2.5. Base de Datos

El uso de Bases de Datos es necesario para mantener la información organizada y accesible, y proporciona unos sistemas de *backup* (copias de seguridad) necesarios para mantener la disponibilidad de los datos. Cualquier desarrollo que tenga que manejar información, en función del volumen e importancia de esta, es necesario que disponga de una base de datos en la que almacenar la misma y gestionarla de forma que se asegure la integridad y disponibilidad de los datos almacenados.

Se pueden encontrar varias definiciones de lo que es una Base de Datos, una de ellas la define como “una recopilación organizada de información o datos estructurados, que normalmente se almacena de forma electrónica en un sistema informático.” (Oracle, 2022).

Para el desarrollo de la aplicación de *parkings* se ha decidido utilizar el sistema de base de datos *SQLite*[4], el cual se trata de un sistema de gestión de datos relacional, compatible con las características *ACID* de los datos, que permite clasificar las transacciones del sistema gestor de base de datos. ACID es el acrónimo, en inglés, de *Atomicity* (atomicidad), *Consistency* (consistencia), *Isolation* (aislamiento) y *Durability* (durabilidad). Cada una de estas características garantiza que los datos almacenados son correctos y fiables. Cada característica ACID garantiza lo siguiente:

- Atomicidad (*atomicity*): garantiza la ejecución completa de la orden u órdenes solicitadas a la base de datos, es decir, si una transacción está formada por varias instrucciones se garantiza que se ejecutan todas ellas o ninguna, lo que se denomina transacción completa.
- Consistencia (*consistency*): garantiza las reglas de integridad de la base de datos.
- Aislamiento (*isolation*): esta propiedad garantiza que una operación no va a afectar a otras, evitando así posibles errores en transacciones simultáneas.
- Durabilidad *(durability*): por último, esta propiedad, asegura que las operaciones realizadas sobre los datos permanecerán persistentes.

SQLite es una biblioteca de C que provee una base de datos ligera, basada en disco, que no requiere un proceso de servidor separado, enlazándose directamente con la aplicación sobre la que se incluya y formando parte integral con esta. El conjunto de datos (tablas, índices, restricciones y los datos de la aplicación en sí mismos) se almacenan en un único fichero en la máquina que actúa como *host*. Por último, otro factor que ha decantado el uso de este sistema de base de datos ha consistido en que *SQLite* permite el prototipado de la aplicación y, en caso de que el volumen de datos comience a ser voluminoso, transferir con facilidad el código a un sistema gestor de mayor volumen como puede ser *MariaDB*, *Oracle* o *PostgreSQL*.

[4] *SQLite*: https://www.sqlite.org/index.html

2.5.1. Diagrama de Entidad-Relación

Para diseñar correctamente la base de datos del proyecto se ha definido un Diagrama de Entidad-Relación. Este diagrama muestra cómo interactúan las entidades que forman parte del sistema. Es un modelo conceptual que ayuda al analista a ver las relaciones que existen entre los elementos del sistema. Los diferentes símbolos que se pueden encontrar en este tipo de diagramas son:

- Los rectángulos, que representan las entidades, las cuales, al transformar el diagrama al modelo relacional, se convertirán en tablas de la base de datos.
- Los óvalos son los diferentes atributos de cada entidad y describen las características de esta.
- Los rombos muestran las relaciones entre las entidades.
- Las líneas de conexión unen las relaciones con las entidades y expresan las cardinalidades. Dichas cardinalidades indican la cantidad de información que tiene en común una entidad con otra.

El Diagrama de Entidad-Relación permite mapear los elementos clave de la base de datos que almacenará la información del sistema.

3. CONTEXTUALIZACIÓN

El desarrollo del presente trabajo se centra en el diseño e implementación de una aplicación web que permita a un usuario realizar el diseño de un *Layout* de un *Parking* e introducir toda la información que requiera este tipo de negocio en una base de datos, sin necesidad de contar con dos sistemas diferenciados: uno para realizar el diseño y otro para la gestión de los datos.

Con una *interfaz* gráfica muy sencilla, la persona encargada de gestionar un *parking* podrá diseñar el *Layout* que considere más oportuno para el espacio del que disponga, para ello la *interfaz* le dotará de una barra de herramientas desde la que podrá seleccionar aquellos elementos que le interese. El usuario podrá incluir diferentes puntos de interés tales como carreteras o columnas, si es que se trata de un recinto cubierto y dispone de ellas, y, principalmente, la disposición de las plazas de que va a disponer el *parking*. Además se podría dotar al sistema de herramientas para incluir paredes, para delimitar el espacio del que dispone, también se podrían incorporar las salidas y entradas de vehículos, los accesos y salidas peatonales, definición de zonas, etc.

Se dotará al sistema de dos tamaños de plazas para poder cubrir la mayoría de los tamaños de vehículos posible, grande o pequeña, aunque se diferenciará, dentro de las plazas grandes, a unas plazas específicas para discapacitados.

Para cada una de las plazas del *parking* el usuario podrá indicar una serie de información asociada a la misma y unas observaciones. El tamaño de la plaza se almacenará automáticamente en función de la plaza que haya seleccionado en la barra de herramientas, podrá posicionar la plaza en la zona que el usuario quiera dentro del plano.

Toda la información que el usuario vaya introduciendo se almacenará en una base de datos que identificará cada plaza de forma unívoca, y guardará la información asociada a la misma. Esa información se podría utilizar para la gestión del negocio en cuanto a niveles de ocupación, gestión de cobros, gestión de clientes habituales, etc., quedando estos aspectos fuera del ámbito del desarrollo abordado en este TFE.

Todo lo comentado en los párrafos anteriores se detallará en los apartados sucesivos, así como las tecnologías a utilizar durante el desarrollo de la aplicación.

3.1. Herramientas de diseño de *layout* de parkings

En la actualidad existen herramientas que, con los conocimientos previos necesarios, sirven para realizar el diseño del *layout* de un *parking* o para la gestión propia del negocio, si bien, las primeras se tratan de aplicaciones muy técnicas, y las segundas se encargan de algún aspecto específico de la gestión: gestión del aparcamiento, gestión del negocio, cobros. Al tratarse de herramientas independientes unas de otras no dan la facilidad al usuario de realizar un diseño y, a la vez, almacenar la información necesaria en una base de datos.

En los siguientes subapartados se profundiza en algunos de los desarrollos existentes del sector, respecto al diseño de *layout* de plazas, y sus características principales.

3.1.1. Transoft Solutions ParkCAD

Una de las herramientas que existen en el mercado en la actualidad se llama *ParkCAD*[5].

Figura 7. Logotipo de *ParkCAD*

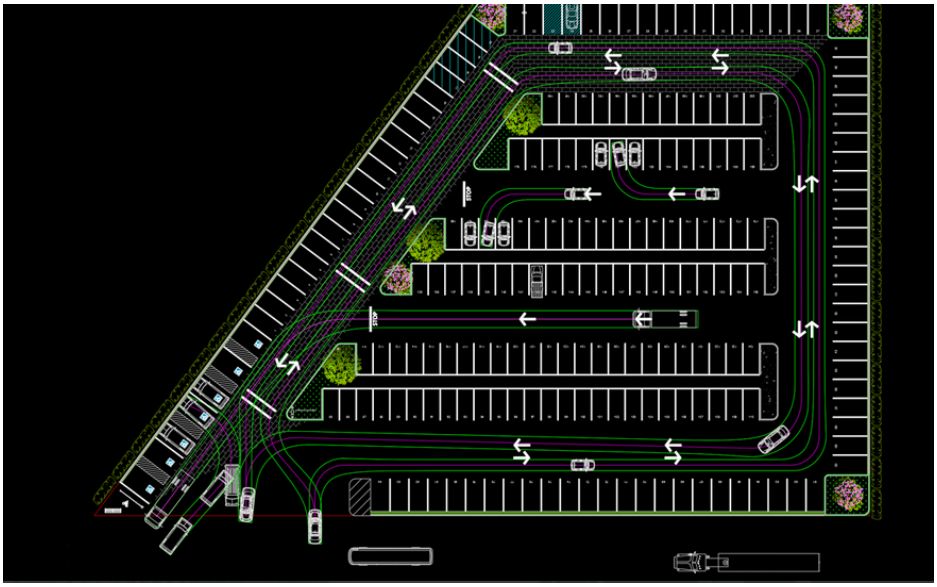
Fuente: *Transoft Solutions*

ParkCAD, como su nombre indica, es una herramienta *CAD (Computer-Aided Design)*. Se trata de una herramienta de diseño asistido por ordenador con la que ingenieros y arquitectos pueden realizar el diseño de una distribución de estacionamiento en base a un espacio disponible, por lo tanto, requiere de unos conocimientos técnicos que no todos los usuarios poseen.

Dicha herramienta únicamente se encarga del diseño, sin posibilidad de dotar de la información, que el usuario quiera dar a cada una de las plazas, para incluirla en su base de datos.

[5] *ParkCAD* - https://www.transoftsolutions.com/es/diseno-de-obras/parkcad/

Figura 8. Imagen de *ParkCAD*

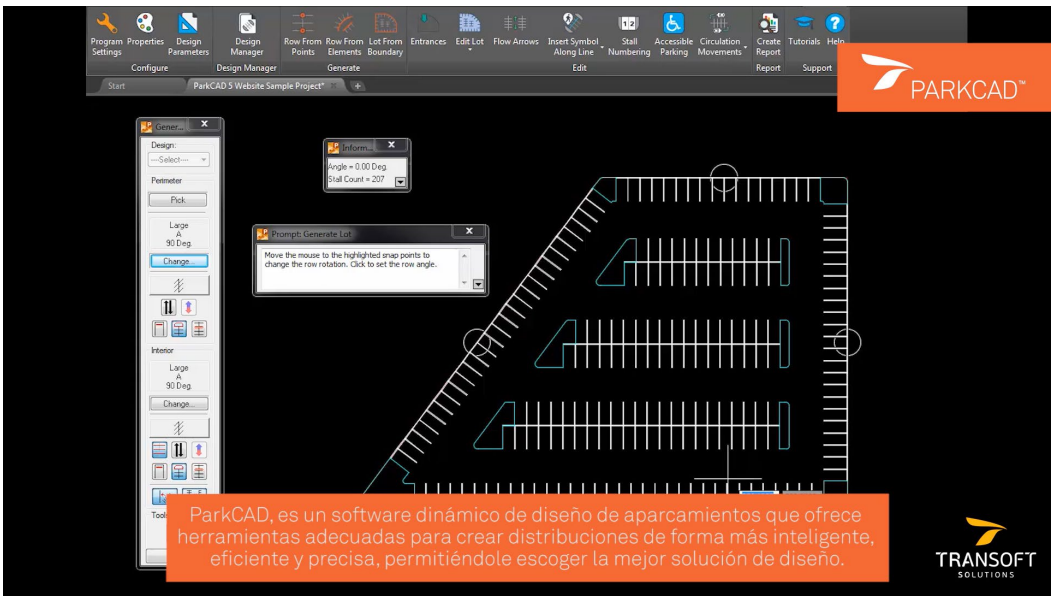


Fuente: *Transoft Solutions*

Las funcionalidades principales de dicho desarrollo son las siguientes:

- Diseño de plazas de aparcamiento: el software permite a un ingeniero o arquitecto definir las plazas de aparcamiento de un recinto. Mediante el software CAD el diseñador puede incorporar plazas de *parking* de diferente tamaño, o incluso específicas para discapacitados, también isletas para delimitar la zona de tránsito de los vehículos, puede incluir elementos ornamentales tales como árboles, etc.

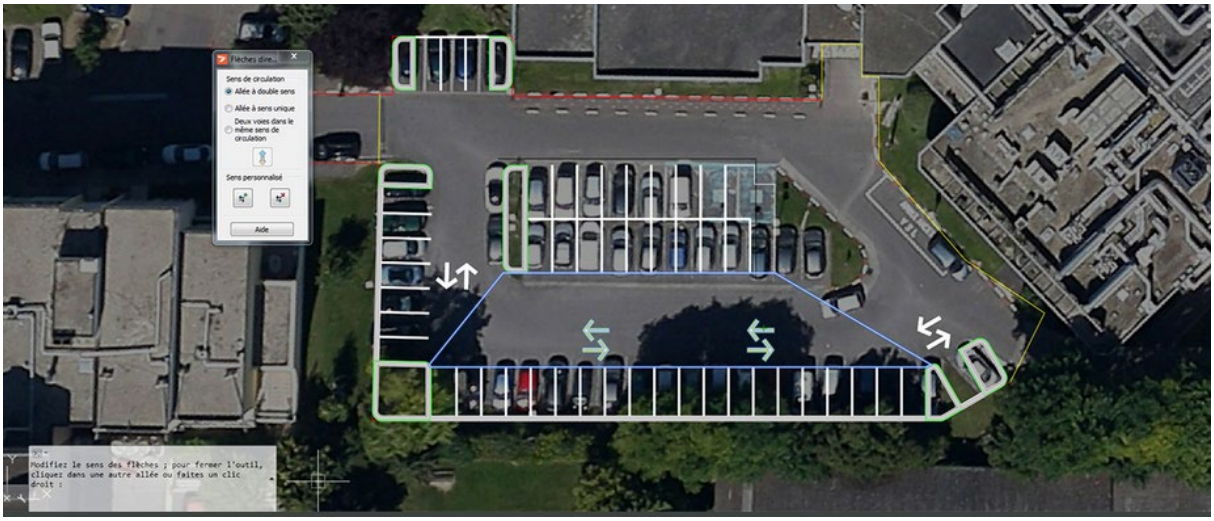
Figura 9. *ParkCAD* - Diseño de plazas



Fuente: *Transoft Solutions*

- Flechas de flujo de tráfico: el sistema permite añadir flechas para indicar la dirección del tráfico en las diferentes vías que se formen al diseñar el *parking*.

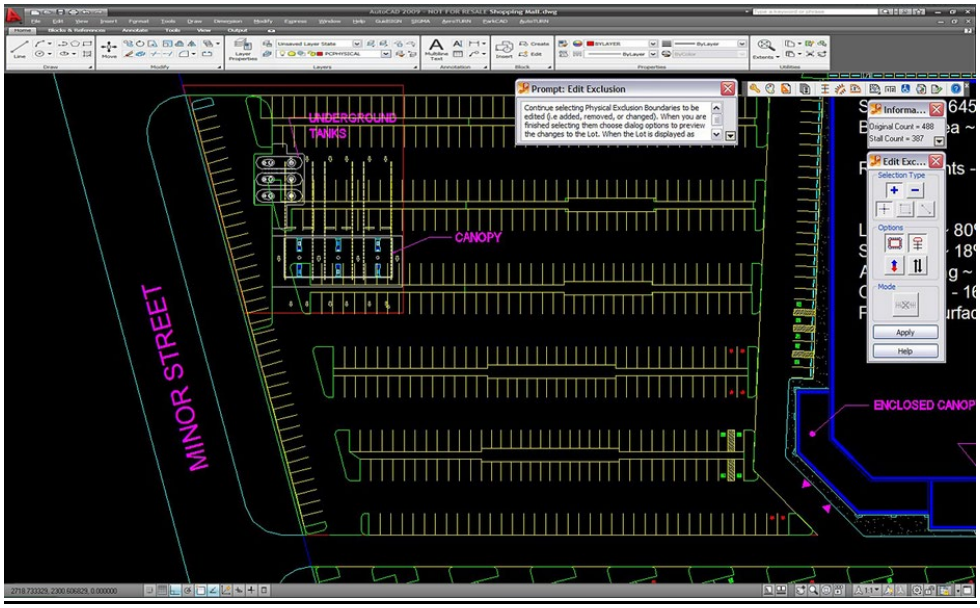
Figura 10. *ParkCAD* - Flechas de flujo de tráfico



Fuente: *Transoft Solutions*

- Áreas de exclusión: el sistema permite definir áreas donde no se definirán plazas de aparcamiento, y que podrán utilizarse para dotar al *parking* final de edificios, isletas y cualquier elemento que no deba utilizarse para el diseño de las plazas.

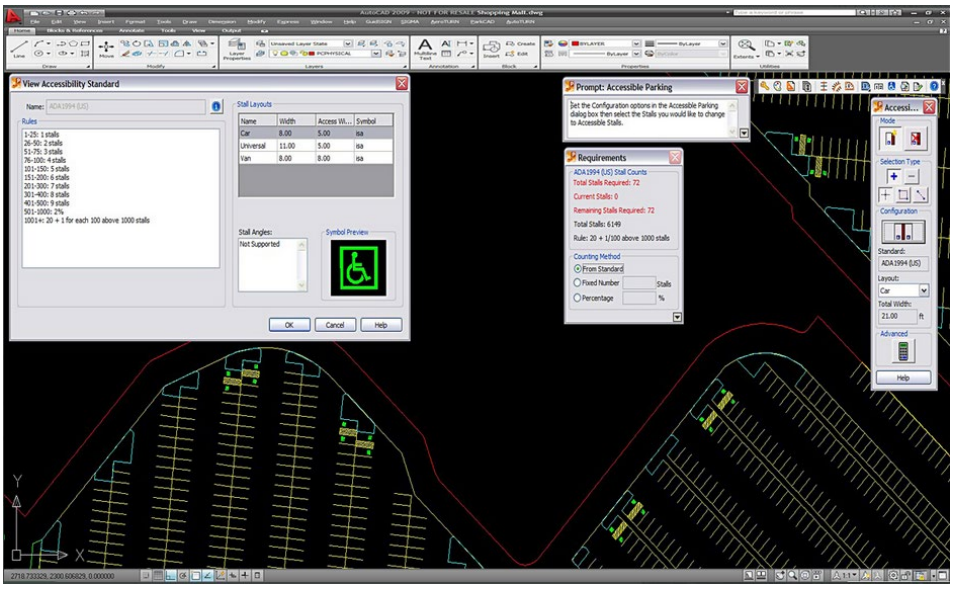
Figura 11. *ParkCAD* - Áreas de Exclusión



Fuente: *Transoft Solutions*

- Opciones de accesibilidad: la aplicación permite definir plazas para usuarios con movilidad reducida y mantener un recuento de estas para poder cumplir la normativa vigente.

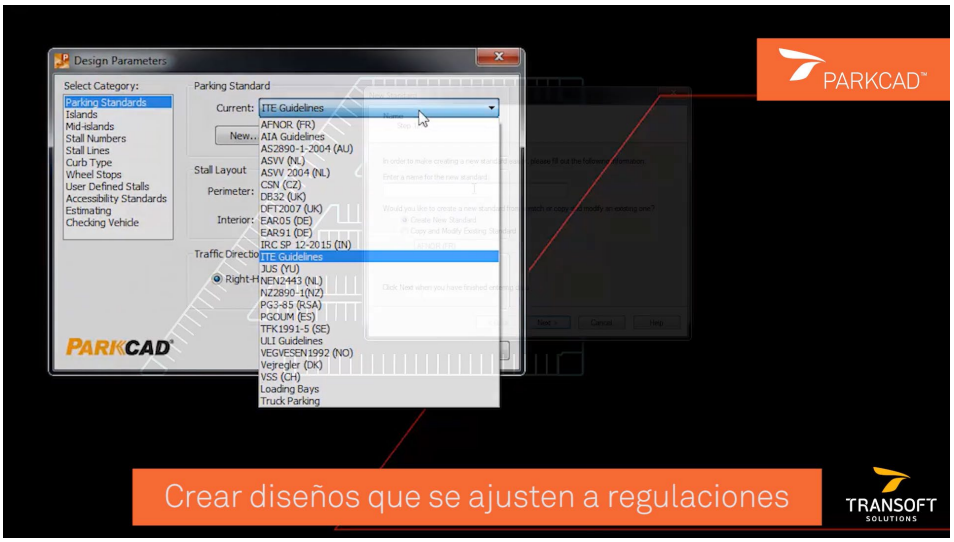
Figura 12. *ParkCAD* - Opciones de accesibilidad



Fuente: *Transoft Solutions*

- Plantillas de regulación normativa: el usuario puede seleccionar la normativa sobre la que va a basar el diseño de su *parking* en función a una serie de plantillas que incluye el desarrollo.

Figura 13. *ParkCAD* - Plantillas de regulación normativa



Fuente: *Transoft Solutions*

Es un sistema muy complejo que, como ya se ha mencionado anteriormente, requiere de unos conocimientos técnicos muy concretos en el campo de la ingeniería de obra civil o en la arquitectura.

ParkCAD requiere de los siguientes requisitos técnicos para su funcionamiento, tal como se puede observar en la Tabla 1:

Tabla 1. *ParkCAD* - Requisitos técnicos de la aplicación

REQUISITOS DE LA PLATAFORMA	**REQUISITOS DEL SISTEMA**
Autodesk® AutoCAD® 2015 – 2023 (excepto AutoCAD LT)	Compatibilidad total con sistema operativo 64 bits
Autodesk® AutoCAD® Civil 3D® 2015 – 2023	Estación de trabajo: *Windows®* 7, 8, 8.1, 10, 11
Bentley® MicroStation® V8i, CONNECT	Red: *Windows® Server* 2012, 2016, 2019, 2022
Bentley® OpenRoads Designer CONNECT	
Bricsys® BricsCAD® V17 – V23 (excepto *BricsCAD Lite*)	

Fuente: *Transoft Solutions*

3.1.2. *ConceptDraw DIAGRAM* v16

ConceptDraw Diagram[6] es otra herramienta que permite realizar el diseño del *layout* de un *parking*.

[6] *ConceptDraw Diagram* - https://www.conceptdraw.com/products/drawing-tool

Figura 14. Logotipo de *ConceptDraw DIAGRAM*

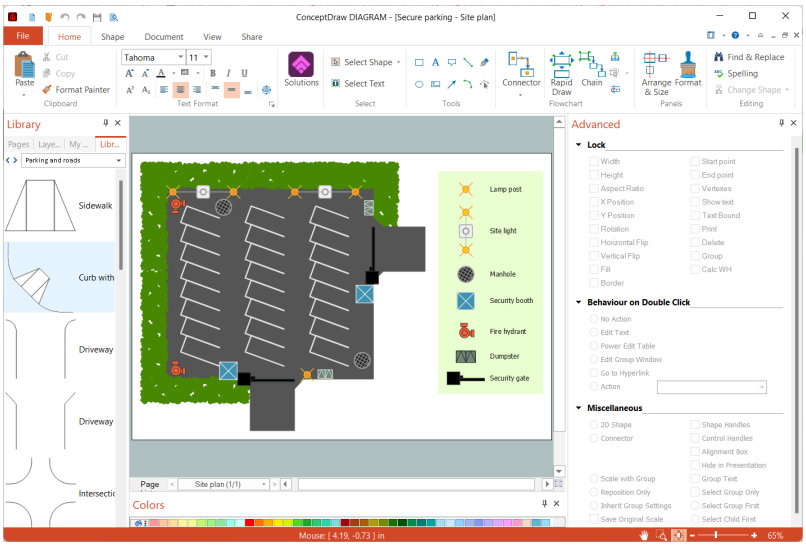
Fuente: *Computer Systems Odessa*

ConceptDraw DIAGRAM es una aplicación desarrollada por *Computer Systems Odessa*, ubicada en San José, California (USA). Actualmente la herramienta se encuentra desarrollada en su versión 16. Se trata de una herramienta de diagramación que permite realizar diseños, compartir estos y realizar trabajos en equipo mediante presentaciones (Computer Systems Odessa, 2023).

ConceptDraw DIAGRAM proporciona un conjunto de herramientas de dibujo, mediante plantillas, que facilita al usuario la labor de realizar el diseño para visualizar diagramas de negocio específicos, además incluye muchas de librerías *stencils*, o estarcido en castellano, las cuales utilizan un método que consiste en estampar diseños pasando la pintura a una superficie a través de una plantilla, (Sánchez, 2021).

En la Figura 15 se muestra el detalle de la pantalla principal de la aplicación, en la misma podemos observar cómo facilita al usuario una serie de elementos que se pueden arrastrar al área de trabajo desde una librería.

Figura 15. Pantalla de trabajo de *ConceptDraw DIAGRAM* v16

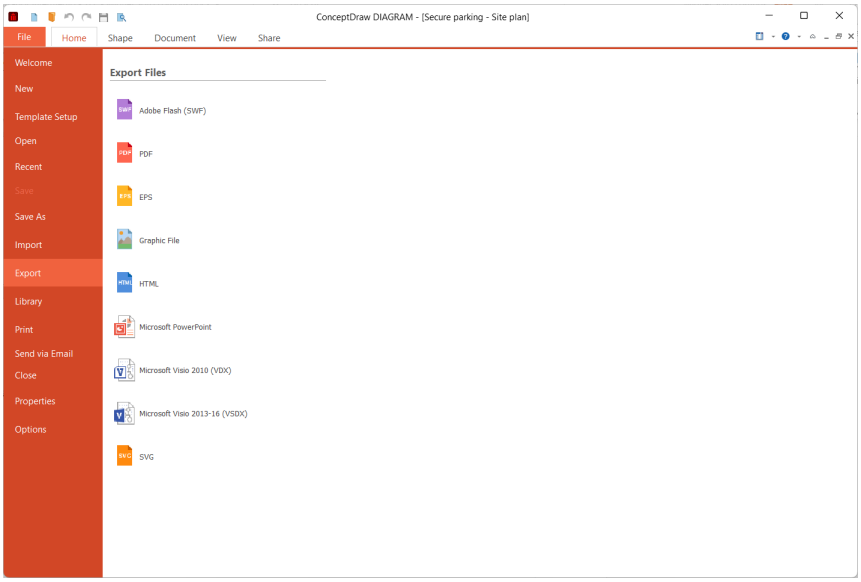


Fuente: *ConceptDraw DIAGRAM* v16 (*CS Odessa*, 2023)

Como se puede comprobar, el software lo único que permite es el diseño de un diagrama, que se realiza en base a unos elementos de una plantilla, es decir, permite realizar una representación gráfica, pero no permite al usuario introducir información de cada plaza o elemento (punto de interés) en una base de datos.

Una vez finalizado el diseño del diagrama, el usuario tiene la posibilidad de exportar su trabajo a diferentes tipos de documentos, tal como se puede comprobar en la Figura 16.

Figura 16. *ConceptDraw DIAGRAM* v16 - Pantalla de Exportación



Fuente: *ConceptDraw DIAGRAM* v16 (*CS Odessa*, 2023)

ConceptDraw DIAGRAM v16 requiere de los siguientes requisitos de sistema, tal como se detalla en la Tabla 2:

Tabla 2. *ConceptDraw DIAGRAM* v16 - Requisitos técnicos de la aplicación

macOS	**Windows**
macOS® 11 (Big Sur), y 12 (Monterey)	Microsoft Windows® 8.1, 10 (64-bit)
CPU: Intel	CPU: Intel o AMD; 1.8 GHz o superior
RAM: mínimo 4 GB	RAM: mínimo 4 GB
HDD: mínimo 1.5 GB de espacio en disco	HDD: mínimo 1.5 GB de espacio en disco

Fuente*: ConceptDraw DIAGRAM* v16 (*CS Odessa*, 2023)

3.2. Herramientas de gestión de información de *parkings*

Otro ámbito en el que se pueden encontrar aplicaciones basadas en este tipo de negocios son aquellas herramientas que sirven para la gestión propia del negocio, las cuales se encargan de algún aspecto específico de dicha gestión: gestión de la capacidad de aparcamiento, gestión del negocio, cobros, etc. A continuación se exponen algunos ejemplos de herramientas que sirven para este fin, el de la gestión de información.

3.2.1. Parking Lot Problem

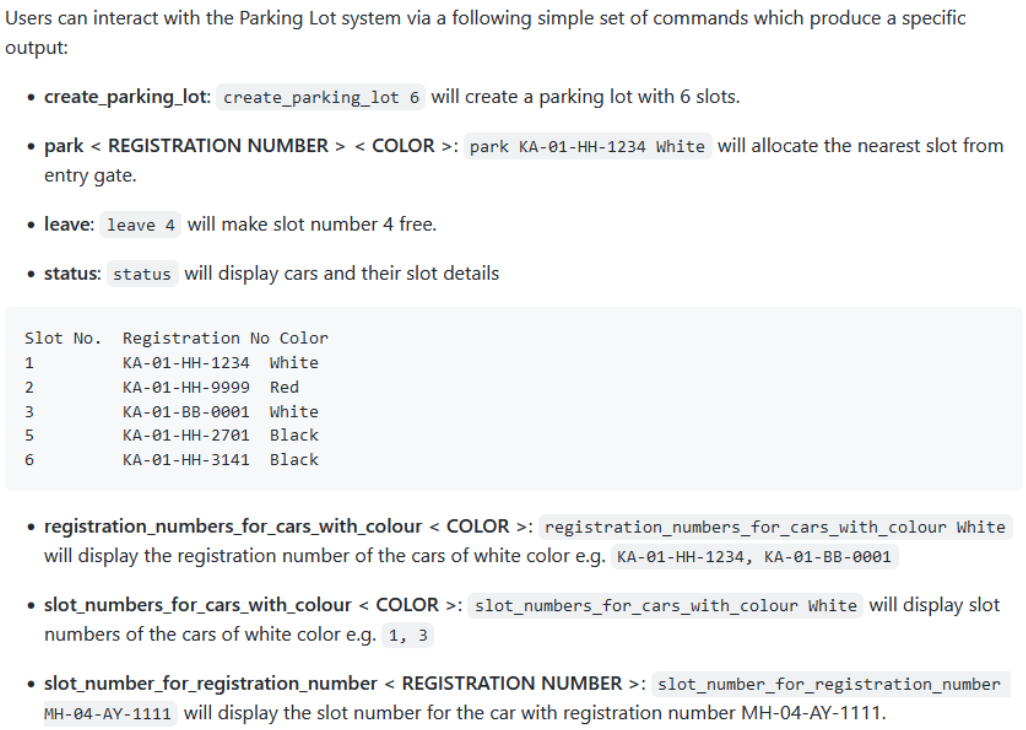
Parking Lot Problem es una aplicación desarrollada por Vinit Shahdeo y codificada en *Node.js*. Se trata de una aplicación básica que permite almacenar información sobre los vehículos que se aparcan en plazas, y, tal como se especifica en el propio desarrollo (Shahdeo, 2019/2022), de esta forma disponer de una herramienta con capacidad para encontrar:

- Matrículas de todos los coches de un color determinado.
- Número de ranura en la que está aparcado un coche con una matrícula determinada.
- Números de ranura de todas las ranuras en las que está aparcado un coche de un color determinado.

El desarrollo aborda la problemática de almacenar información sobre la ocupación de plazas, pero no permite el diseño del *layout* del *parking*.

Mediante el uso de unos comandos específicos el sistema va realizando las operaciones necesarias para crear las plazas de las que va a disponer el sistema de *parking*, va a permitir almacenar la información de las plazas ocupadas, permite ver el nivel de ocupación del *parking*, liberar plazas ocupadas, etc.

Figura 17. Ejemplos de comandos de *Parking Lot Problem*



Users can interact with the Parking Lot system via a following simple set of commands which produce a specific output:

- **create_parking_lot**: `create_parking_lot 6` will create a parking lot with 6 slots.
- **park** < **REGISTRATION NUMBER** > < **COLOR** >: `park KA-01-HH-1234 White` will allocate the nearest slot from entry gate.
- **leave**: `leave 4` will make slot number 4 free.
- **status**: `status` will display cars and their slot details

```
Slot No.  Registration No Color
1         KA-01-HH-1234  White
2         KA-01-HH-9999  Red
3         KA-01-BB-0001  White
5         KA-01-HH-2701  Black
6         KA-01-HH-3141  Black
```

- **registration_numbers_for_cars_with_colour** < **COLOR** >: `registration_numbers_for_cars_with_colour White` will display the registration number of the cars of white color e.g. `KA-01-HH-1234, KA-01-BB-0001`
- **slot_numbers_for_cars_with_colour** < **COLOR** >: `slot_numbers_for_cars_with_colour White` will display slot numbers of the cars of white color e.g. `1, 3`
- **slot_number_for_registration_number** < **REGISTRATION NUMBER** >: `slot_number_for_registration_number MH-04-AY-1111` will display the slot number for the car with registration number MH-04-AY-1111.

Fuente: *Parking Lot Problem* (Shahdeo, 2019/2022)

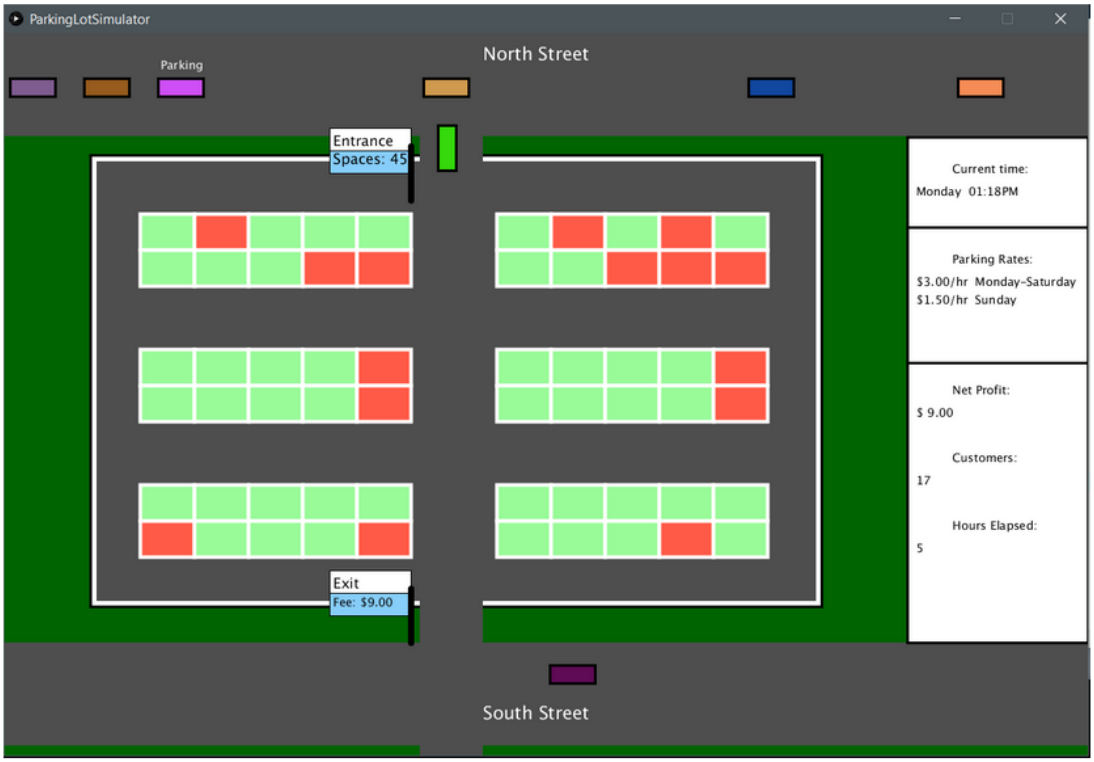
3.2.2. Parking Lot Simulator

Parking Lot Simulator es una aplicación desarrollada por Luke Garland y Michael Enright. La aplicación permite simular el comportamiento de un *parking* de pago. Las plazas que están libres se muestran en pantalla en color verde, al ocuparse se cambian a color rojo. La aplicación simula la entrada y salida de vehículos mediante un tiempo aleatorio, e incluso simula el pago.

La aplicación está desarrollada en *Processing 3*[7].

[7] Processing 3 - https://processing.org/

Figura 18. Ejemplo de *Parking Lot Simulator*



Fuente: *Parking Lot Simulator* (Garland & Enright, 2018/2022)

3.3. Conclusiones de la contextualización

Todos los desarrollos que se han estudiado para la elaboración del presente trabajo se centran más en la gestión propia de un *parking*, ocupación de plazas, pagos, etc., que en diseños y administración de espacio de estos.

Como conclusión, cabe destacar que el objetivo principal del proyecto consiste en incorporar en una única aplicación diferentes herramientas que el dueño de un *parking* debería utilizar para, primero, almacenar toda la información necesaria para la gestión propia del negocio y, en segundo lugar, permitir diseñar la estructura de este y los espacios de estacionamiento.

4. DISEÑO DE LA PROPUESTA

El presente capítulo pretende mostrar cómo se ha realizado el diseño de la propuesta en sus diferentes fases. Para ello se van a detallar aspectos relevantes sobre las decisiones tomadas para el diseño de la arquitectura que da forma al sistema, y por último las decisiones en diseño de software para la implementación del código.

En primer lugar se presenta un análisis general de la aplicación y se exponen los recursos con los que se ha contado para el desarrollo de esta.

A continuación se pasa a detallar el alcance y los casos de uso de la aplicación, más adelante cada uno de estos se desglosa en requisitos funcionales y, para concluir ese apartado se han definido unos requisitos no funcionales.

Para continuar con el diseño de la propuesta se expone el análisis realizado con respecto a los datos que tiene que manejar la aplicación. Para ello se ha diseñado un diccionario de datos en el que se incluye la definición y estructura de los datos que formarán parte del desarrollo. También se ha realizado un diagrama del Modelo Entidad-Relación en el que se detallan las diferentes entidades que forman parte de la estructura de la base de datos, así como las relaciones entre las mismas y la cardinalidad que existe entre ellas.

Después se hace una presentación de la arquitectura de la aplicación, se presentan los diferentes diagramas utilizados para el análisis del diseño, tales como diagramas de casos de uso o diagrama de clases según el modelo de vistas de *Django*, entre otros.

Seguidamente se detalla la metodología utilizada en el proyecto, la cual es el modelo iterativo incremental. Con dicha metodología se divide el desarrollo en unidades de trabajo más pequeñas que las hacen más manejables. Es ahí donde se detallan las diferentes iteraciones que se han realizado en el proyecto.

Un apartado crítico para la correcta finalización de la aplicación es la evaluación y validación, mediante pruebas, de esta. Al considerarse un aspecto básico se ha decidido separar dicha evaluación en un capítulo específico (Capítulo 5. EVALUACIÓN DE LA APLICACIÓN).

4.1. Análisis general

Tal como se ha definido en la introducción del presente trabajo, más concretamente en el apartado 1.3 (Objetivos del TFE), el objetivo general de este consiste en el desarrollo de una aplicación web pasando por todas las fases del ciclo de vida de un desarrollo software. Más allá de las diferentes fases de dicho ciclo de vida podemos especificar los siguientes objetivos:

- Identificar la tecnología más adecuada para el desarrollo de la aplicación en función de los entornos existentes.
- Realizar un análisis de requisitos que, en base a las necesidades del usuario, debe satisfacer la aplicación.
- Refinar los requisitos encontrados para esta aplicación.
- Instalar un entorno de desarrollo para la codificación de la aplicación.
- Diseñar un sistema de base de datos que asegure la disponibilidad e integridad de la información almacenada.
- Diseñar una *interfaz* de usuario práctica que facilite la utilización de la aplicación.
- Producir como resultado el almacenamiento de la información deseada por el usuario.

4.1.1. Software utilizado para el desarrollo

Para abordar el desarrollo se ha utilizado el siguiente software:

- *Visual Studio Code*, versión 1.76.2, es el IDE[8] de desarrollo utilizado.
- *Microsoft Word*, versión 365, como programa de procesamiento de textos para la edición y maquetación del presente documento.
- *Microsoft Visio Profesional*, versión 365, software utilizado para el diseño de diagramas.
- Sistema operativo del equipo de desarrollo: *Windows 11*.

Además del software anterior, se ha instalado el siguiente entorno de desarrollo:

- *Python*, versión 3.11, es el lenguaje de desarrollo del entorno servidor utilizado, en el que está basado *Django*.

[8] *Integrated Development Environment* (Entorno de Desarrollo Integrado)

- *Bootstrap, framework* CSS y *Javascript* utilizado para el diseño de *interfaces* y con diseño *responsive*.
- *KonvaJS,* es un *framework Javascript* utilizado para crear la *interfaz* de diseño de *layout* de *parking*.
- *Django*, *framework* de desarrollo basado en *Python*, se ha utilizado para programar la lógica de servidor de la aplicación.
- *SQLite*, es un sistema de gestión de base de datos, utilizado para almacenar la información de la aplicación.
- *Git*, se trata de un sistema de control de versiones distribuido que permite llevar un control sobre los cambios realizados en los diferentes archivos que forman parte del proyecto.

4.2. Alcance y Casos de Uso de la aplicación

En este apartado se detalla el alcance de la aplicación, así como los casos de uso que se han establecido, con el objetivo de completar los objetivos de la aplicación y sus diferentes fases.

Dichos casos de uso posteriormente se desglosarán en requisitos funcionales. Para llevar a cabo este análisis primero debemos definir lo que se entiende por requisito. Un requisito es un atributo necesario de un sistema, entendido como una declaración que identifica una característica, capacidad o factor de calidad, con el objetivo y finalidad de aportar valor a un cliente o usuario (Young, 2004).

4.2.1. Alcance de la aplicación

El presente proyecto tiene como finalidad el analizar, diseñar, desarrollar y validar de una aplicación web que permita al usuario la gestión de un negocio de *parking*, la propia creación lógica del negocio en una base de datos e implementar un diseño del *layout* del mismo.

Mediante el acceso a una aplicación *web*, el usuario podrá crear la estructura lógica del *parking*, dando una descripción al mismo, ubicación y demás datos que lo identifiquen. También el usuario tendrá la posibilidad de indicar si el parking dispone de diferentes edificios o espacios (en caso de parkings no cubiertos que sólo dispongan de un espacio para estacionar los vehículos), definir las plantas de que dispone cada edificio y por último definir las propias

plazas de aparcamiento mediante dos maneras, a través del diseñador de plantas o en las pantallas específicas de toma de datos.

4.2.2. Casos de uso

Para definir los casos de uso de la aplicación se ha tomado como punto de partida el objetivo general planteado en el apartado 1.3 (Objetivos del TFE) del presente trabajo y más concretamente en el subapartado 1.3.1 (Objetivo general).

Se han definido unos casos de uso que posteriormente se desglosarán en requisitos funcionales. En la Tabla 3 se describen los casos de uso de la aplicación web.

Tabla 3. Casos de uso de la aplicación

Identificador	Descripción
CU-1	El sistema permitirá dar de alta, baja y modificar los datos de negocios de ***parking***.
CU-2	El sistema permitirá dar de alta, baja y modificar los datos de los diferentes **edificios** del *parking*.
CU-3	El sistema permitirá dar de alta, baja y modificar los datos de las **plantas** que tenga cada edificio.
CU-4	El usuario debe ser capaz de diseñar el **layout** del parking.
CU-5	El sistema permitirá dar de alta, baja y modificar los datos de las **plazas** de aparcamiento de cada planta.

4.3. Diseño de la Base de Datos

Toda aplicación que maneje información requiere de una base de datos para sustentarla. Una base de datos es necesaria en una aplicación para almacenar y recuperar información de manera eficiente.

El diseño de la base de datos es crucial para la eficiencia del sistema. Un diseño correcto ayuda a garantizar la integridad de los datos, lo que implica que la información almacenada está organizada de manera lógica, y los datos que la componen se relacionan de manera apropiada para evitar errores o inconsistencias. Por otro lado, un diseño correcto ayuda a maximizar el rendimiento de la base de datos, lo que significa que las consultas y transacciones se ejecutan de manera rápida y eficiente. También es importante destacar que un diseño correcto hace que sea más fácil de mantener y escalar la base de datos a medida que las necesidades de la aplicación cambien con el tiempo.

Para realizar el análisis y diseño de la base de datos se ha llevado a cabo la definición de un diccionario de datos y un diagrama de entidad-relación.

4.3.1. Sistema gestor de base de datos

Tal como se expone en el artículo 2.5, se ha decidido optar por trabajar con *SQLite*. Una de las características especiales del modelo de datos de *Django* es que permite crear objetos mediante una API[9] que realizan las operaciones comunes de CRUD (crear, leer, actualizar y eliminar) en los modelos de datos sin necesidad de escribir sentencias SQL.

Esta forma de trabajar con los modelos de datos dota al sistema de una serie de ventajas, como pueden ser:

- Abstracción de la base de datos: *Django* proporciona una abstracción de alto nivel que permite trabajar con modelos de datos de manera más sencilla, sin preocuparse por detalles de bajo nivel de la base de datos.
- Seguridad: Usar la API de *Django* para operaciones CRUD aumenta la seguridad de la aplicación, ya que proporciona protección contra inyecciones de SQL, evitando que los atacantes puedan manipular la base de datos de la aplicación.
- Productividad: el desarrollo es más rápido y eficiente al eliminar la necesidad de escribir sentencias SQL tediosas y repetitivas. Además de proporcionar muchas funcionalidades adicionales, como manejo de transacciones, validación de datos y gestión de relaciones entre tablas.

[9] *Application Programming Interface*, en español, *interfaz* de programación de aplicaciones.

- Mantenibilidad: Usar la API de *Django* para operaciones CRUD hace que la aplicación sea más fácil de mantener a largo plazo, ya que el código es más legible y mantenible que las sentencias SQL directas, lo que hace que sea más fácil para los desarrolladores entender y modificar el código. Además, la API de *Django* también maneja la migración de esquemas de base de datos, lo que hace que la aplicación sea más fácil de mantener a medida que evoluciona con el tiempo.

Para asegurar una estructura eficiente, coherente y la integridad de las tablas, se ha aplicado el proceso de normalización en el diseño de la base de datos. El nivel de normalización utilizado es la tercera forma normal (3FN), la cual implica las siguientes restricciones:

- Eliminación de dependencias transitivas.
- Todos los atributos son dependientes de la clave principal.
- Todos los atributos no clave deben depender directamente de la clave principal.
- No se permiten transacciones multivaluadas, lo que significa que una fila no puede tener más de un valor para un atributo determinado.

4.3.2. Diccionario de Datos

Un diccionario de datos es «una colección de información detallada sobre las entidades de datos que utiliza una aplicación» (Wiegers & Beatty, 2013).

El diccionario de datos contiene información sobre los tipos de datos, valores permitidos y definición de los datos del sistema. Esta información se utiliza para modelar las tablas de la base de datos y los atributos que tendrá cada una de ellas.

La Tabla 4 muestra el diccionario de datos diseñado para la aplicación.

Tabla 4. Diccionario de Datos

Elemento de Datos	Descripción	Composición o tipo	Longitud	Valores y unidades
Parking	Información general del *parking* sobre el que se vaya a almacenar la información.	Id del *parking* + Descripción + Ubicación + Cubierto + Observaciones + Fecha de creación + Fecha de actualización		
Edificio	Información de los diferentes edificios que formen parte del *parking*. Para ello se asociará el edificio a un *parking* en concreto. Todos los *parkings* deben disponer como mínimo de un edificio.	Id del Edificio + Descripción + Observaciones + Fecha de creación + Fecha de actualización + Id del *parking*		
Planta	Información de las plantas de las que se compone cada edificio. Para ello se asociará la planta con el edificio al que pertenece. Todos los edificios deben disponer como mínimo de una planta.	Id de la Planta + Descripción + Máximo de plazas + Observaciones + Fecha de creación + Fecha de actualización + Id del edificio		
Plaza	Información de las diferentes plazas del *parking*. Para ello se asociará la plaza con la planta en la que se encuentre y además se indicará el tipo de plaza que es.	Id de la Plaza + Descripción + Observaciones + Fecha de creación + Fecha de actualización + Id de la planta + Id del tipo de plaza		
Tipo de plaza	Información sobre los diferentes tipos de plaza que hay en el sistema. Vienen delimitados por tres posibles valores (Grande, Pequeña, Discapacitados).	Id del tipo de plaza + Descripción + Ancho + Largo + Discapacitados + Fecha de creación + Fecha de actualización		
ID del *Parking*	Identificador único del *Parking*	Entero positivo	8	Número secuencial entero generado por la base de datos con cada *parking*, comenzando con el 1.
ID del Edificio	Identificador único del Edificio	Entero positivo	8	Número secuencial entero generado por la base de datos con cada edificio, comenzando con el 1.

ID del Planta	Identificador único del Planta	Entero positivo	8	Número secuencial entero generado por la base de datos con cada planta, comenzando con el 1.
ID del Plaza	Identificador único de la Plaza	Entero positivo	8	Número secuencial entero generado por la base de datos con cada plaza, comenzando con el 1.
ID del Tipo de Plaza	Identificador único del Tipo de Plaza	Entero positivo	8	Número secuencial entero generado por la base de datos con cada tipo de plaza, comenzando con el 1.
Descripción	Texto que describe el elemento al que se le incluya	Caracteres alfabéticos	50	Puede contener espacios en blanco, y cualquier carácter tecleado por el usuario.
Observaciones	Texto que complementa la información del elemento en el que se incluya	Caracteres alfabéticos	300	Puede contener espacios en blanco, y cualquier carácter tecleado por el usuario.
Fecha de creación	Fecha en la que se da de alta por primera vez el elemento en el que se incluya	Fecha	10	Formato de fecha
Fecha de actualización	Fecha en la que se da ha modificado por última vez el elemento en el que se incluya	Fecha	10	Formato de fecha
Ancho	Indica la medida del ancho del Tipo de Plaza	Numérico con decimales	8	
Largo	Indica la medida del largo del Tipo de Plaza	Numérico con decimales	8	
Discapacitados	Indica si la plaza es considerada para uso de personas discapacitadas	Booleano		Puede contener True o False
Ubicación	Información con la ubicación del *parking*	Caracteres alfabéticos	100	Puede contener espacios en blanco, y cualquier carácter tecleado por el usuario
Cubierto	Informa de si el *parking* es cubierto	Booleano		Puede contener True o False
Máximo de plazas	Indica la cantidad máxima de plazas que puede tener una planta	Entero positivo	8	Puede contener el 0.

4.3.3. Diagrama de Entidad-Relación

Para el desarrollo de la aplicación se han detectado las siguientes entidades:

- *Parkings.*

  Contiene la información general del *parking* sobre el que se vaya a almacenar la información.

- Edificios.

  Contiene la información de los diferentes edificios que formen parte del *parking*. Para ello se asociará el edificio a un *parking* en concreto. Todos los *parkings* deben disponer como mínimo de un edificio.

- Plantas.

  Contiene la información de las plantas de las que se compone cada edificio. Para ello se asociará la planta con el edificio al que pertenece. Todos los edificios deben disponer como mínimo de una planta.

- Plazas.

  Contiene la información de las diferentes plazas del *parking*. Para ello se asociará la plaza con la planta en la que se encuentre y además se indicará el tipo de plaza que es.

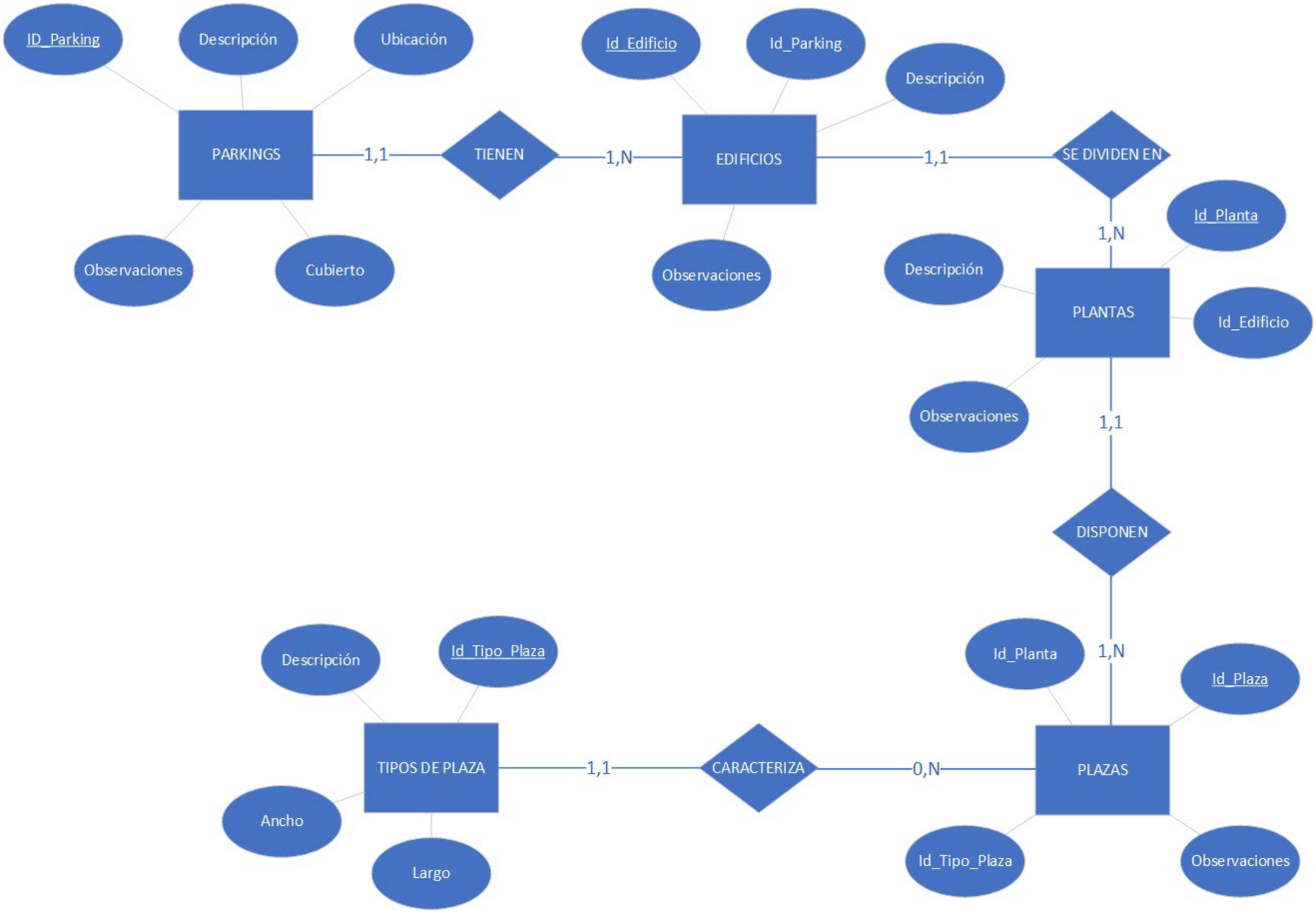
- Tipos de plaza.

  Contiene la información sobre los diferentes tipos de plaza que hay en el sistema. Vienen delimitados por tres posibles valores:

  - Grande.
  - Pequeña.
  - Discapacitados.

En la Figura 19 se puede observar el Diagrama de Entidad-Relación diseñado para el desarrollo de la aplicación:

Figura 19. Diagrama de Entidad-Relación



Fuente: Elaboración propia

4.4. Desarrollo de la aplicación

Para llevar a cabo del desarrollo de la aplicación, y como parte de la fase de análisis previo de cada iteración del modelo iterativo incremental que se ha seguido, expuesto en el apartado 2.3.1, se han desglosado los casos de uso en una serie de requisitos funcionales.

4.4.1. Diagrama de Casos de Uso

El Diagrama de casos de uso pretende describir la interacción de los actores, en este caso el usuario, y el sistema. Para elaborar los requisitos funcionales del proyecto se han desglosado los casos de uso analizados en el apartado 4.2.2.

A continuación se detallan los casos de uso que se definieron en el apartado mencionado y los requisitos funcionales en que se han dividido cada uno de estos.

4.4.1.1. CU-1: Parkings

El presente subapartado muestra en detalle el Caso de Uso 1, relativo a los *Parkings*, y cómo se ha desglosado en requisitos funcionales.

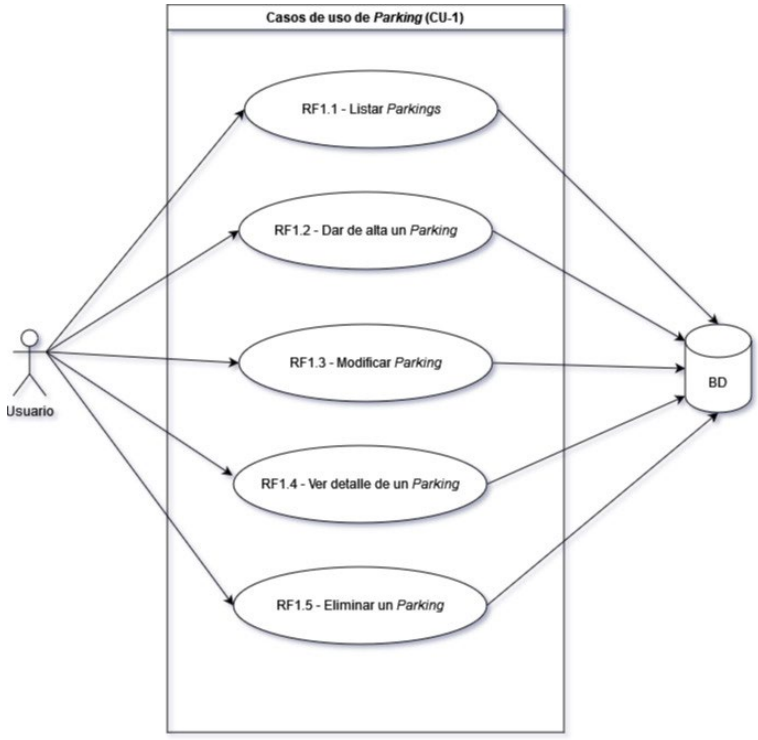
Tabla 5. Requisitos funcionales del CU-1: *Parking*

CU-1	**Caso de Uso 1: *Parkings***
El sistema permitirá dar de alta, baja y modificar los datos del ***parking***, los cuales se componen de: • Identificador único de *parking*, atributo numérico auto incrementable. • Descripción, atributo de tipo cadena de texto. • Ubicación, atributo de tipo cadena de texto. • Cubierto, atributo *booleano* que indica *True* o *False*. • Observaciones, atributo de tipo cadena de texto. • Fecha de alta, atributo de tipo fecha, no editable, se guarda al crear el *parking*. • Fecha de última actualización, atributo de tipo fecha, no editable, se actualiza cada vez que se modifique el *parking*.	

Requisitos funcionales del CU-1
RF1.1 – El sistema permite listar los diferentes *parkings* almacenados.
RF1.2 – El sistema permite dar de alta la información de un *parking*.
RF1.3 – El sistema permite modificar la información de un *parking*.
RF1.4 – El sistema permite ver el detalle de un negocio de un *parking*.
RF1.5 – El sistema permite eliminar toda la información de un *parking*.

La Figura 20 muestra el diagrama de casos de uso diseñado para el desarrollo.

Figura 20. Diagrama de Casos de Uso de Parkings



4.4.1.2. CU-2: Edificios

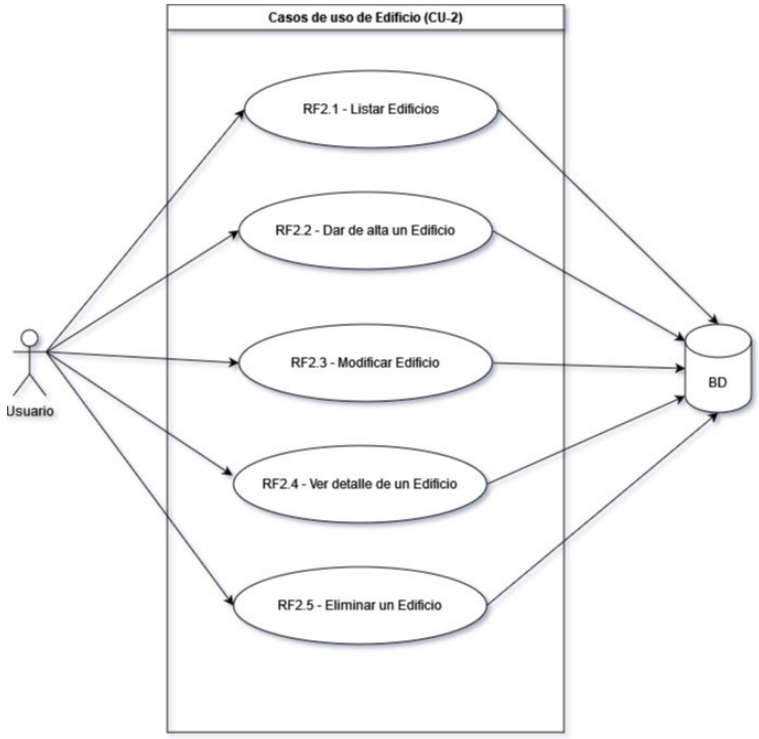
A continuación se detalla el Caso de Uso 2, relativo a los Edificios, y cómo se ha desglosado en requisitos funcionales.

Tabla 6. Requisitos funcionales del CU-2: Edificios

CU-2	**Caso de Uso 2: *Edificios***
El sistema permitirá dar de alta, baja y modificar los datos de los **edificios** que componen el *parking*, los cuales se componen de: • Identificador único de edificio, atributo numérico auto incrementable. • Descripción, atributo de tipo cadena de texto. • Observaciones, atributo de tipo cadena de texto. • Fecha de alta, atributo de tipo fecha, no editable, se guarda al crear el edificio. • Fecha de última actualización, atributo de tipo fecha, no editable, se actualiza cada vez que se modifique el edificio. • Identificador de *parking*, atributo numérico, clave foránea que identifica el *parking* al que pertenece el edificio.	
Requisitos funcionales del CU-2	
RF2.1 – El sistema permite listar los diferentes edificios almacenados.	
RF2.2 – El sistema permite dar de alta la información de un edificio.	
RF2.3 – El sistema permite modificar la información de un edificio.	
RF2.4 – El sistema permite ver el detalle de un edificio.	
RF2.5 – El sistema permite eliminar toda la información de un edificio.	

Seguidamente, la Figura 21 muestra el diagrama de casos de uso de Edificios (CU-2).

Figura 21. Diagrama de Casos de Uso de Edificios



4.4.1.3. CU-3: Plantas

El presente subapartado muestra en detalle el Caso de Uso 3, relativo a las Plantas, y cómo se ha desglosado en Requisitos Funcionales.

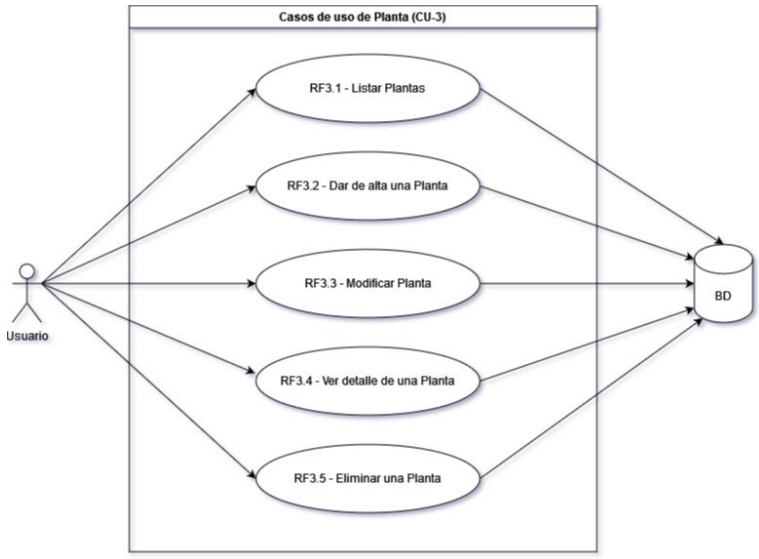
Tabla 7. Requisitos funcionales del CU-3: Plantas

CU-3	**Caso de Uso 3: *Plantas***
El sistema permitirá dar de alta, baja y modificar los datos de las **plantas** que forman parte de cada edificio del *parking*, los cuales se componen de: • Identificador único de planta, atributo numérico auto incrementable. • Descripción, atributo de tipo cadena de texto. • Observaciones, atributo de tipo cadena de texto. • Fecha de alta, atributo de tipo fecha, no editable, se guarda al crear la planta. • Fecha de última actualización, atributo de tipo fecha, no editable, se actualiza cada vez que se modifique la planta.	

• Identificador de *edificio*, atributo numérico, clave foránea que identifica el *edificio* al que pertenece la planta.
Requisitos funcionales del CU-3
RF3.1 – El sistema permite listar las diferentes plantas almacenadas.
RF3.2 – El sistema permite dar de alta la información de una planta.
RF3.3 – El sistema permite modificar la información de una planta.
RF3.4 – El sistema permite ver el detalle de una planta.
RF3.5 – El sistema permite eliminar toda la información de una planta.

La Figura 22 muestra el diagrama de casos de uso de Plantas, tal como se ha desglosado el CU-3.

Figura 22. Diagrama de Casos de Uso de Plantas



4.4.1.4. CU-4: Diseño de *Layout*

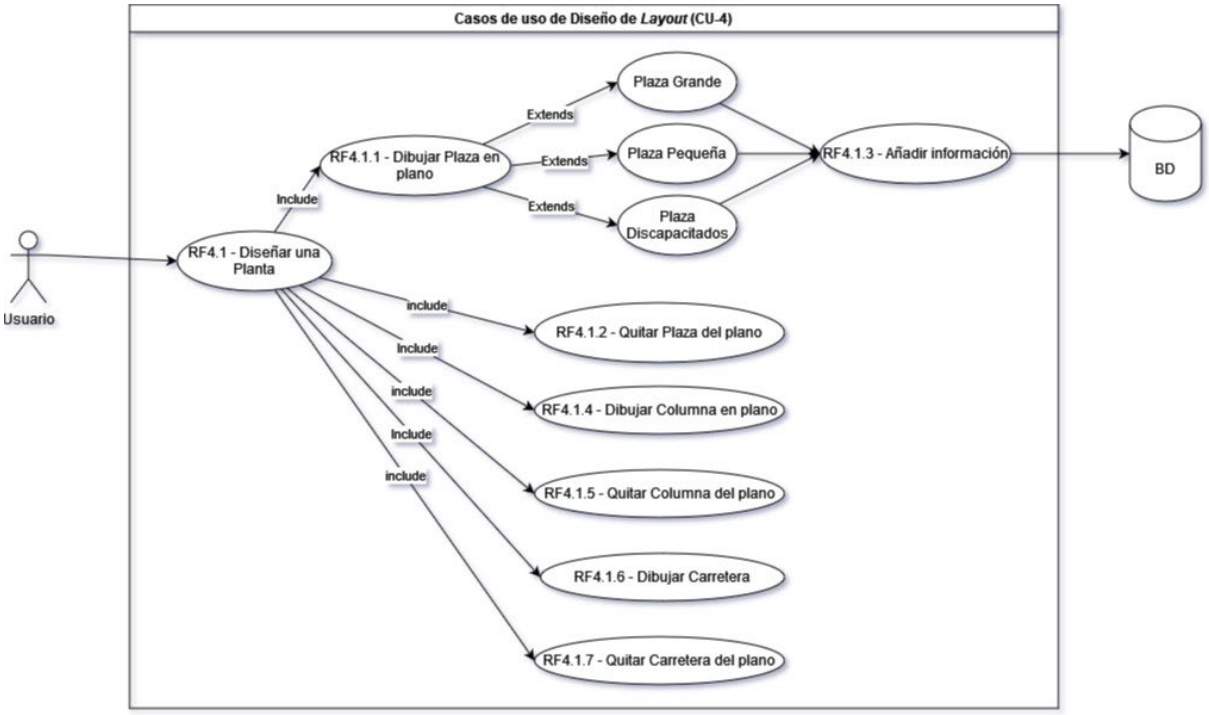
El diseño del *layout* de cada planta es un elemento clave de la aplicación, por este motivo se ha separado en un requisito funcional específico. A continuación se detalla Caso de Uso 4 y cómo se ha desglosado en Requisitos Funcionales.

Tabla 8. Requisitos funcionales del CU-4: Diseño de *Layout*

CU-4	**Caso de Uso 4: *Diseño de Layout***
El usuario debe ser capaz de diseñar el ***layout*** del parking. Para ello definirá el diseño de una planta y podrá introducir la información de cada plaza que asigne en el plano a la base de datos.	
Requisitos funcionales del CU-4	
RF4.1 – El sistema permite diseñar el *layout* de una planta.	
RF4.1.1 – El sistema permite dibujar una plaza en el plano.	
RF4.1.2 – El sistema permite quitar una plaza del plano.	
RF4.1.3 – El sistema permite añadir información de una plaza en la base de datos.	
RF4.1.4 – El sistema permite dibujar una columna en el plano.	
RF4.1.5 – El sistema permite quitar una columna del plano.	
RF4.1.6 – El sistema permite dibujar una carretera en el plano.	
RF4.1.7 – El sistema permite quitar una carretera del plano.	

La Figura 23 muestra el diagrama de casos de uso del Diseño de *layout* de plantas, tal como se ha desglosado el CU-4.

Figura 23. Diagrama de Casos de Uso de Diseño de *Layout*



4.4.1.5. CU-5: Plazas

Por último se muestra en detalle el Caso Funcional 5, relativo a las Plazas, y cómo se ha desglosado en diferentes requisitos funcionales.

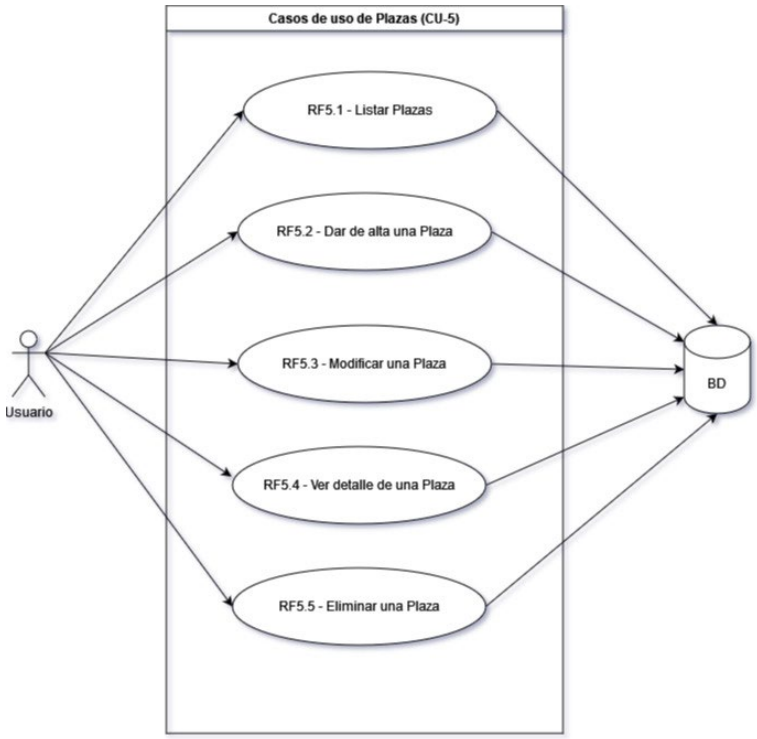
Tabla 9. Requisitos funcionales del CU-5: Plazas

CU-5	**Caso de Uso 5: *Plazas***
El sistema permitirá dar de alta, baja y modificar los datos de las **plazas** que forman parte de cada planta del *parking*, los cuales se componen de: • Identificador único de plaza, atributo numérico auto incrementable. • Descripción, atributo de tipo cadena de texto. • Observaciones, atributo de tipo cadena de texto. • Fecha de alta, atributo de tipo fecha, no editable, se guarda al crear la planta. • Fecha de última actualización, atributo de tipo fecha, no editable, se actualiza cada vez que se modifique la planta.	

• Identificador de planta, atributo numérico, clave foránea que identifica la planta a la que pertenece la plaza. • Identificador de tipo de plaza, atributo numérico, clave foránea que identifica el tipo de plaza.
Requisitos funcionales del CU-5
RF5.1 – El sistema permite listar las diferentes plazas almacenadas.
RF5.2 – El sistema permite dar de alta la información de una plaza.
RF5.3 – El sistema permite modificar la información de una plaza.
RF5.4 – El sistema permite ver el detalle de una plaza.
RF5.5 – El sistema permite eliminar toda la información de una plaza.

La Figura 24 muestra el diagrama de casos de uso de Plantas, tal como se ha desglosado el CU-5.

Figura 24. Diagrama de Casos de Uso de Plazas



4.4.1.6. Requisitos no funcionales

Los requisitos no funcionales de la aplicación definen las restricciones que tiene la aplicación, tanto a nivel de su utilización, como a otros niveles. En la Tabla 10 se detallan los requisitos no funcionales de la aplicación.

Tabla 10. Requisitos no funcionales de la aplicación

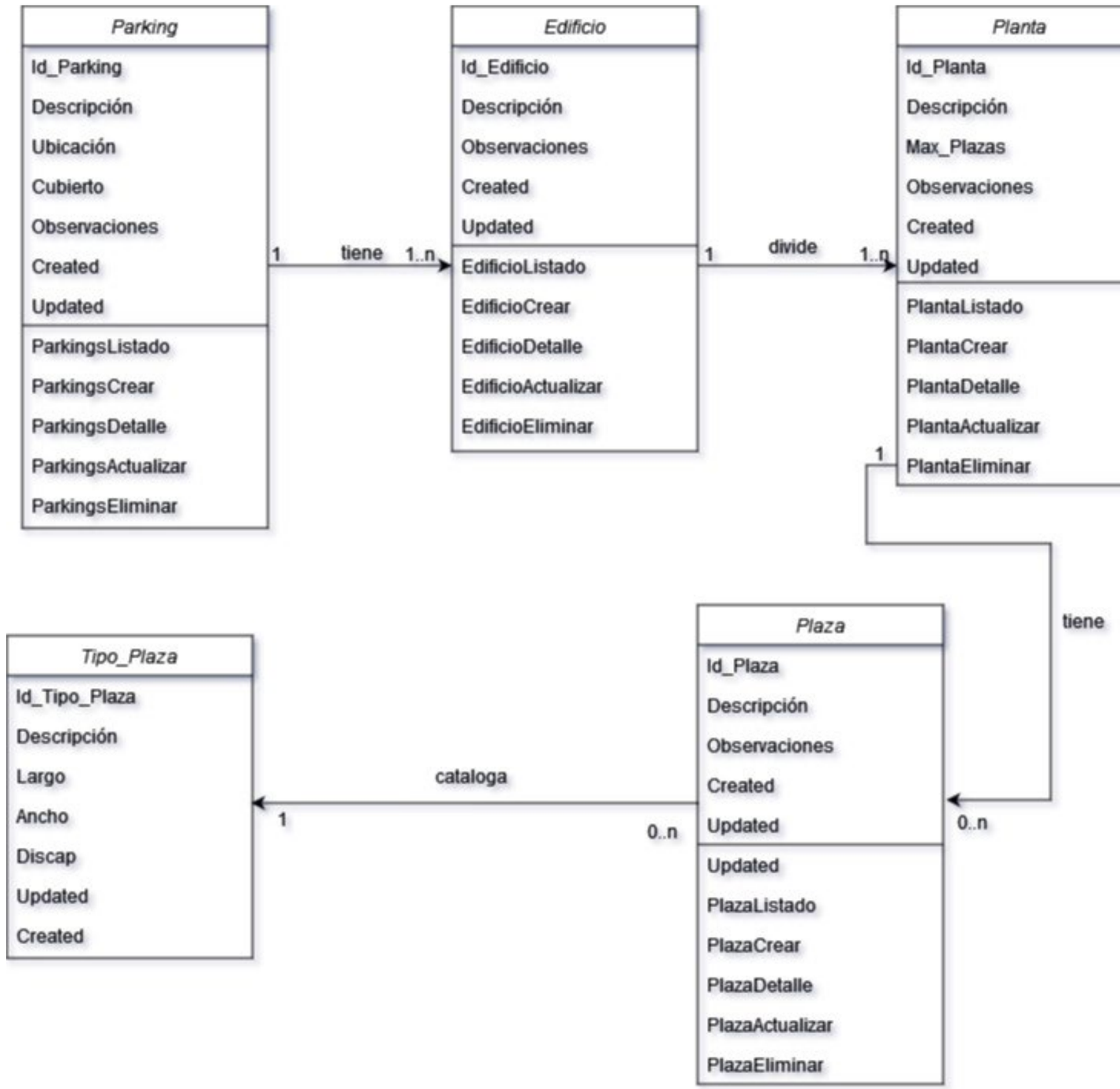
Identificador	Descripción
RNF-1	El usuario debe disponer de conexión a Internet para utilizar la aplicación.
RNF-2	La aplicación deberá utilizarse a través de un navegador web.
RNF-3	La base de datos se almacenará y funcionará bajo un sistema SQLite3.

4.4.2. Diagrama de Clases

En ingeniería de software, un diagrama de clases en Lenguaje Unificado de Modelado (UML) es un tipo de diagrama de estructura que permite observar claramente la estructura del sistema mostrando las clases que forman parte de este, los atributos de las clases, los métodos y las relaciones entre los diferentes objetos.

La Figura 25 muestra un diagrama de clases que se ha elaborado tras realizar el análisis previo del desarrollo. Para definir este se han unificado en un único diagrama el modelo de datos y las vistas de cada aplicación (objeto) de *Django*, se recuerda al lector que, tal como se detalló en el apartado 2.4.2, *Django* trabaja con el modelo *MVT (Model-View-Template),* de forma que separa el modelo de datos y las vistas en elementos separados. El diagrama representa como atributos de cada clase los campos definidos en el modelo de datos de cada aplicación que forma parte del proyecto y como métodos las vistas de estas.

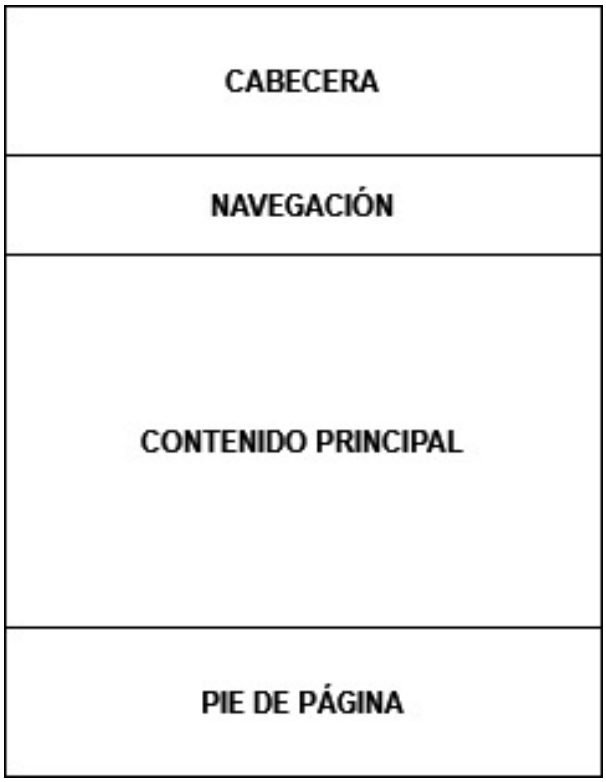
Figura 25. Diagrama de clases



4.4.3. *Interfaces* de la aplicación

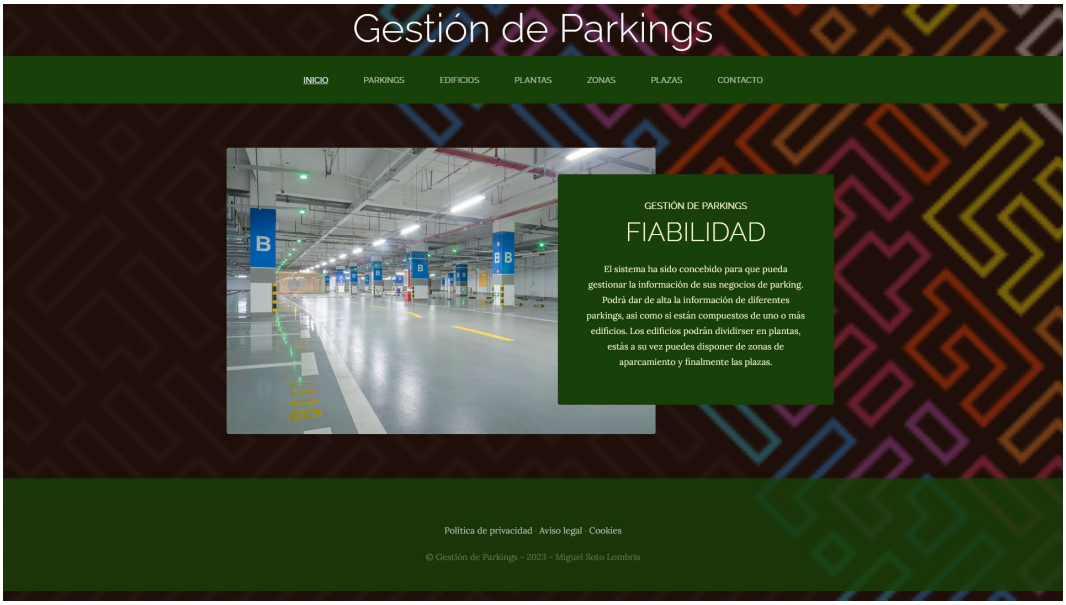
Atendiendo a la simplicidad que se quiere facilitar al usuario, y una vez analizadas las necesidades, los casos de uso, requisitos funcionales y no funcionales, modelado de datos y diagramas de clase, se ha desarrollado una *interfaz* muy sencilla, e intuitiva, en la que se disponen todas las páginas siguiendo un mismo patrón, tal como podemos observar en el prototipo de pantalla, en la Figura 26, todas las *interfaces* disponen de una cabecera, una barra de navegación, un área con el contenido principal y por último un pie de página.

Figura 26. Prototipo de *interfaz* de usuario



En base a este prototipo se han desarrollado una serie de *interfaces* de usuario. La Figura 27 muestra la pantalla principal de la aplicación.

Figura 27. Pantalla principal de la aplicación



4.4.4. Desarrollo de las Clases

Como ya se ha expuesto a lo largo del presente TFE, se ha desarrollado la aplicación en *Django*, *framewok* de desarrollo basado en *Python*, de forma que cada clase tiene sus métodos en lo que en *Django* se denomina **vistas**. El siguiente fragmento de código muestra un ejemplo que detalla las vistas de la clase Planta:

```
class PlantaListado(ListView):
    model = Planta # Llamo a la clase 'Planta' que se encuentra en el archivo
'models.py'

class PlantaCrear(SuccessMessageMixin, CreateView):
    model = Planta # Llamamos a la clase 'Planta' que se encuentra en nuestro
archivo 'models.py'
    form = Planta # Definimos nuestro formulario con el nombre de la clase o
modelo 'Planta'
    fields = ("descripcion", "max_plazas", "observaciones", "id_edificio") #
Le decimos a Django que muestre los campos de la tabla 'Planta' de nuestra
Base de Datos

    success_message = 'Planta Creada Correctamente' # Mostramos este Mensaje
si el Parking se crea correctamente

    # Redireccionamos a la página principal
    def get_success_url(self):
        return reverse('ListarPlanta') # Redireccionamos a la vista principal
'ListarParking'

class PlantaDetalle(DetailView):
```

```
    model = Planta # Llamamos a la clase 'Planta' que se encuentra en nuestro
archivo 'models.py'

class PlantaActualizar(SuccessMessageMixin, UpdateView):
    model = Planta # Llamamos a la clase 'Planta' que se encuentra en nuestro
archivo 'models.py'
    form = Planta # Definimos nuestro formulario con el nombre de la clase o
modelo 'Planta'
    fields = "__all__" # Le decimos a Django que muestre todos los campos de
la tabla 'Planta' de nuestra Base de Datos
    success_message = 'Planta Actualizada Correctamente' # Mostramos este
Mensaje si el Planta se edita correctamente

    # Redireccionamos a la página principal
    def get_success_url(self):
        return reverse('ListarPlanta') # Redireccionamos a la vista principal

class PlantaEliminar(SuccessMessageMixin, DeleteView):
    model = Planta
    form = Planta
    fields = "__all__"

    # Redireccionamos a la página principal
    def get_success_url(self):
        success_message = 'Planta Eliminada Correctamente' # Mostramos este
Mensaje si el Parking se ha eliminado correctamente
        messages.success (self.request, (success_message))
        return reverse('ListarPlanta') # Redireccionamos a la vista principal
```

El siguiente apartado muestra las diferentes iteraciones llevadas a cabo en el desarrollo del proyecto, en cada una de ellas se definieron las vistas de las diferentes aplicaciones, tal como se ha abordado anteriormente, en *Django* los objetos que interactúan entre si en el proyecto son definidos como aplicaciones independientes.

4.5. Metodología de desarrollo

Tal y como se ha expuesto en el Capítulo 2, más concretamente en el apartado 2.3.1 se ha decidido utilizar el modelo iterativo incremental, el cual consiste, a grandes rasgos, en dividir el proyecto en pequeñas partes sobre las que se realizarán una serie de trabajos, de forma que al final de la iteración se obtenga una versión completa del mini proyecto iterado y de esta forma obtener versiones más completas del proyecto final.

4.5.1. Iteración 1: Inicio del proyecto

Como se ha expuesto en el apartado 2.4 (Lenguaje de Desarrollo), la programación de la aplicación se ha llevado a cabo con el *framework Django*, el cual trabaja en base a aplicaciones, es decir, cada objeto que forma parte del proyecto se define como una aplicación y esta es a su vez incluida en una aplicación general que define el proyecto general.

La primera iteración del desarrollo ha consistido en definir la aplicación general, en diseñar la plantilla que sirve de base para todas las *interfaces* de aquella y en desarrollar una página de contacto para que el administrador del sistema pueda recibir correos electrónicos del usuario que necesite contactar con este.

La Figura 28 detalla cómo se han definido las aplicaciones iniciales que forman parte de la iteración:

Figura 28. Iteración 1. Aplicaciones iniciales

```
# Application definition
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'ParkingLotApp',
    'contacto',
]
```

Como parte de la iteración se ha diseñado la pantalla de inicio de la aplicación, tal como se muestra en la Figura 27.

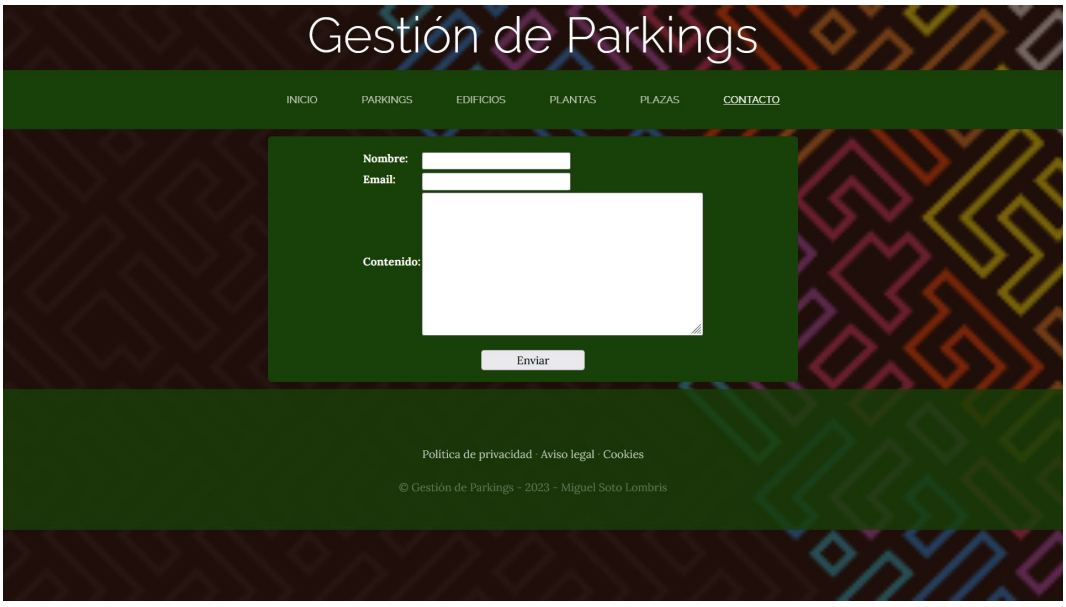
También se ha desarrollado una página de contacto. La Figura 29 muestra el código de programación de dicha página de contacto:

Figura 29. Codificación de la página de contacto

```
{% extends "ParkingLotApp/base.html" %}

{% load static %}

{% block content %}

    <div class="contenedorFormulario">

        {% if "valido" in request.GET %}
            <p style="color:snow">Información enviada correctamente. Gracias.</p>
        {% elif "no_valido" in request.GET %}
            <p style="color:snow">NO se ha podido enviar la información. Inténtelo de nuevo.</p>
        {% endif %}

        <form action="" method="POST" style="text-align: center;">
            {% csrf_token %}
            <table style="color:white; margin:20px auto;">{{formulario.as_table}}</table>
            <input type="submit" value="Enviar" style="width:150px;">
        </form>
    </div>

{% endblock %}
```

La Figura 30 muestra el *interfaz* diseñado para que los usuarios puedan contactar con el administrador del sistema.

Figura 30. Pantalla de Contacto



Al trabajar con el *framework* de desarrollo *Django* las tablas que forman parte de la Base de Datos se definen en cada una de las diferentes aplicaciones que forman parte del proyecto, para ello, cada aplicación debe contener un archivo llamado ***models.py*** que contiene el modelo de datos de la información que gestiona. En esta iteración se definió el modelo de datos de Tipos de Plaza, puesto que son unos datos fijos, necesarios para el correcto

funcionamiento de la aplicación, que no pueden ser manipulados por el usuario final. Dicha definición se muestra en la Figura 31.

Figura 31. Modelo de datos de Tipo de Plaza

```
# Creo los modelos para los Tipos de Plaza
# Los datos de los Tipos de Plaza NO podrán ser modificados por el usuario
# Los datos de los Tipos de Plaza serán fijos y tendrán los valores:
#       - Plaza Grande
#       - Plaza Pequeña
#       - Plaza Discapacitados

class TipoPlaza(models.Model):

    # Creo el modelo de datos de los Tipos de Plaza
    descripcion=models.CharField(max_length=50)
    ancho=models.FloatField()
    largo=models.FloatField()
    discap=models.BooleanField(default=False)
    created=models.DateTimeField(auto_now_add=True)
    updated=models.DateTimeField(auto_now_add=True)

    class Meta:
        verbose_name='tipoPlaza'
        verbose_name_plural='tiposPlaza'

    def __str__(self):
        return self.descripcion
```

4.5.2. Iteración 2: Parkings

La siguiente iteración que se ha llevado a cabo en el proyecto ha consistido en trabajar con el modelo de *Parking*. Se definió el modelo de datos necesario y se codificó el mismo en *Django*, tal como se muestra en la Figura 32.

Figura 32. Modelo de datos de la clase Parking

```
# Creo los modelos para el Parking
# Los datos de los Parkings podrán ser modificados por el usuario

class Parking(models.Model):

    # Creo el modelo de datos del Parking
    descripcion=models.CharField(max_length=50)
    ubicacion=models.CharField(max_length=100)
    cubierto=models.BooleanField(default=False)
    observaciones=models.TextField(max_length=300, null=True, blank=True)
    created=models.DateTimeField(auto_now_add=True)
    updated=models.DateTimeField(auto_now_add=True)

    class Meta:
        verbose_name='parking'
        verbose_name_plural='parkings'

    def __str__(self):
        return self.descripcion
```

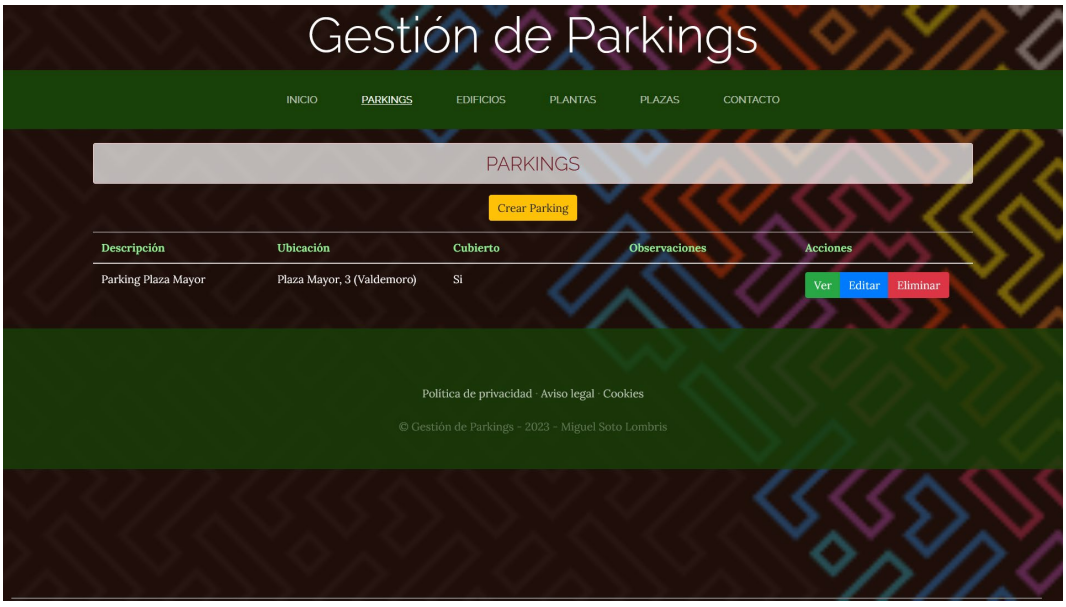
Se definieron las vistas de la aplicación de *Parkings*, tal como se muestra en la Figura 33.

Figura 33. Vistas de la aplicación *Parkings*

```python
from django.urls import path
from parkings.views import ParkingsListado,ParkingsDetalle,ParkingsCrear,ParkingsActualizar, ParkingsEliminar
from django.conf import settings

urlpatterns = [
    # # La ruta 'ListarParking' en donde listamos todos los registros o Parkings de la Base de Datos
    path('', ParkingsListado.as_view(template_name = "parkings/index_parking.html"), name='ListarParking'),

    # # La ruta 'DetalleParking' en donde mostraremos una página con los detalles de un Parking o registro
    path('DetalleParking/<int:pk>', ParkingsDetalle.as_view(template_name = "parkings/detalle_parking.html"), name='DetalleParking'),

    # # La ruta 'crear' en donde mostraremos un formulario para crear un nuevo Parking o registro
    path('CrearParking', ParkingsCrear.as_view(template_name = "parkings/crear_parking.html"), name='CrearParking'),

    # # La ruta 'actualizar' en donde mostraremos un formulario para actualizar un Parking o registro de la Base de Datos
    path('ActualizarParking/<int:pk>', ParkingsActualizar.as_view(template_name = "parkings/actualizar_parking.html"), name='ActualizarParking'),

    # # La ruta 'eliminar' que usaremos para eliminar un Parking o registro de la Base de Datos
    path('ElminarParking/<int:pk>', ParkingsEliminar.as_view(), name='ElminarParking'),
]
```

Y se diseñaron las diferentes *interfaces* para trabajar con el modelos de datos, dichas *interfaces* en *Django* son llamadas *Templates*. La Figura 34 muestra la *interfaz* que lista los diferentes negocios de *parking* almacenados en el sistema.

Figura 34. *Interfaz* para listar *Parkings*



La Figura 35 muestra la *interfaz* para dar de alta un nuevo *parking*, la cual es similar a la diseñada para editar los datos de un *parking*, con la diferencia de que esta última mostraría, al cargarse, los datos de este.

Figura 35. *Interfaz* para crear/editar un parking

4.5.3. Iteración 3: Edificios

Esta iteración ha consistido en trabajar con la aplicación de Edificios. Se definió el modelo de datos necesario y se codificó el mismo en *Django*, tal como se muestra en la Figura 36.

Figura 36. Modelo de datos de la clase Edificios

```
# Creo los modelos para los Edicficios del Parking
# Los datos de los Edificios podrán ser modificados por el usuario

class Edificio(models.Model):

    # Creo el modelo de datos del Edificio
    descripcion=models.CharField(max_length=50)
    observaciones=models.TextField(max_length=300, null=True, blank=True)
    created=models.DateTimeField(auto_now_add=True)
    updated=models.DateTimeField(auto_now_add=True)
    id_parking=models.ForeignKey(Parking, on_delete=models.CASCADE)

    class Meta:
        verbose_name='edificio'
        verbose_name_plural='edificios'

    def __str__(self):
        return self.descripcion
```

Se diseñaron las diferentes *interfaces* para trabajar con esta clase, en la Figura 37 se puede observar la *interfaz* diseñada para ver el detalle de un edificio.

Figura 37. *Interfaz* con el detalle de un edificio

4.5.4. Iteración 4: Plantas

La siguiente iteración del modelo de desarrollo ha consistido en trabajar con la aplicación de Plantas. Se definió el modelo de datos necesario y se codificó el mismo en *Django*, tal como se muestra en la Figura 38.

Figura 38. Modelo de datos de la clase Planta

```
# Creo los modelos para las Plantas
# Los datos de las Plantas podrán ser modificadas por el usuario

# Create your models here.
class Planta(models.Model):

    # Creo el modelo de datos de la Planta
    descripcion=models.CharField(max_length=50)
    max_plazas=models.IntegerField(blank=False, default=0)
    observaciones=models.TextField(max_length=300, null=True, blank=True)
    created=models.DateTimeField(auto_now_add=True)
    updated=models.DateTimeField(auto_now_add=True)
    id_edificio=models.ForeignKey(Edificio, on_delete=models.CASCADE)

    class Meta:
        verbose_name='planta'
        verbose_name_plural='plantas'

    def __str__(self):
        return self.descripcion
```

4.5.5. Iteración 5: Plazas

La quinta iteración se centró en el desarrollo de la aplicación Plazas. De esta forma se aseguraba que el usuario puede dar de alta plazas directamente en la base de datos aún sin realizar un diseño de *layout* de las plantas del parking.

Se definió el modelo de datos de las plazas tal como se puede observar en la Figura 39.

Figura 39. Modelo de datos de la clase Plaza

```
# Creo los modelos para las Plazas
# Los datos de las Plantas podrán ser modificadas por el usuario

# Create your models here.
class Plaza(models.Model):

    # Creo el modelo de datos de la Planta
    descripcion=models.CharField(max_length=50)
    observaciones=models.TextField(max_length=300, null=True, blank=True)
    created=models.DateTimeField(auto_now_add=True)
    updated=models.DateTimeField(auto_now_add=True)
    id_planta=models.ForeignKey(Planta, on_delete=models.CASCADE)
    id_tipo_plaza=models.ForeignKey(TipoPlaza, on_delete=models.CASCADE)

    class Meta:
        verbose_name='plaza'
        verbose_name_plural='plazas'

    def __str__(self):
        return self.descripcion
```

Se definieron las plantillas de las *interfaces* y se desarrollaron los métodos de la clase, que en *Django* se definen como vistas. El siguiente fragmento de código detalla las importaciones que se han tenido que realizar para poder definir las mismas:

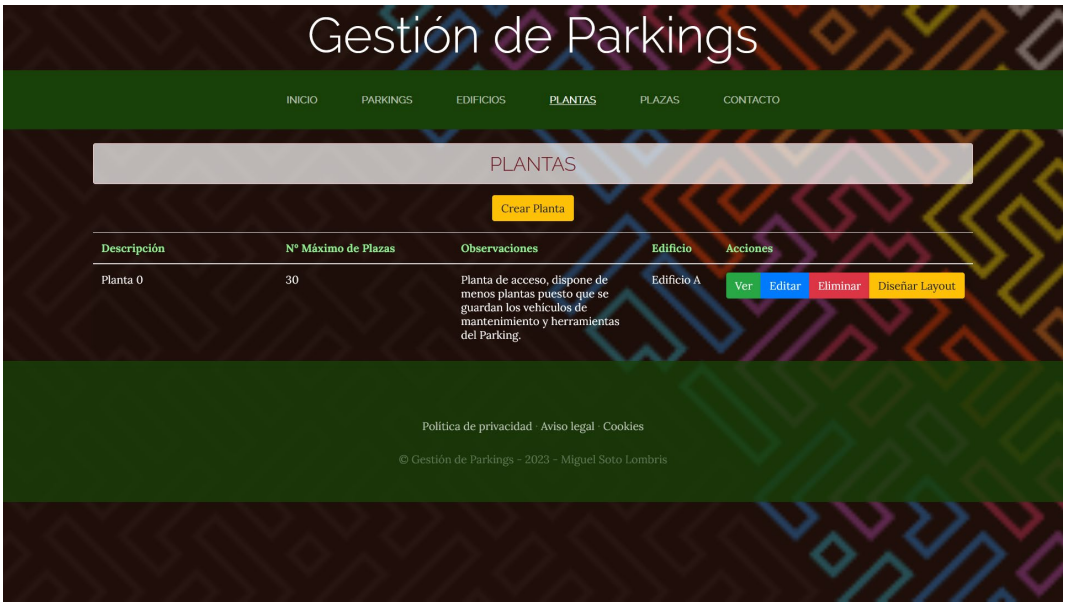
```
from django.shortcuts import render

# Instancio las vistas genéricas de Django
from django.views.generic import ListView, DetailView
from django.views.generic.edit import CreateView, UpdateView, DeleteView

# Instancio los modelos de la BBDD
from .models import Plaza
from plantas.models import Planta
from ParkingLotApp.models import TipoPlaza

# Sirve para redireccionar después de una acción revertiendo patrones de
expresiones regulares
from django.urls import reverse

# Habilitamos el uso de mensajes en Django
```

```
from django.contrib import messages

# Habilitamos los mensajes para class-based views
from django.contrib.messages.views import SuccessMessageMixin

# Habilitamos los formularios en Django
from django import forms
```

4.5.6. Iteración 6: Diseño de Layout

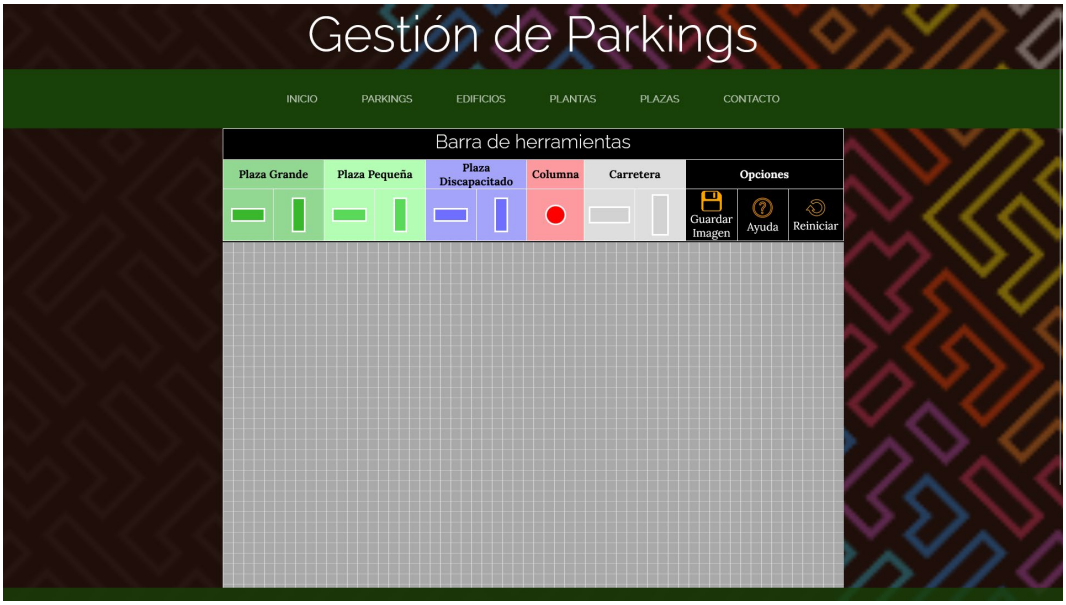
La última iteración se centró en el desarrollo de la funcionalidad de diseño del *layout* de las plantas, y en hacer que desde el plano diseñado se creara en la base de datos la plaza correspondiente en el modelo de datos de estas, que ya se había definido en la iteración anterior. La Figura 40 muestra la *interfaz* que muestra las plantas almacenadas en la base de datos, en la misma se observa que dispone de un botón para acceder al diseñador de planta.

Figura 40. Acceso al diseñador de *layout*



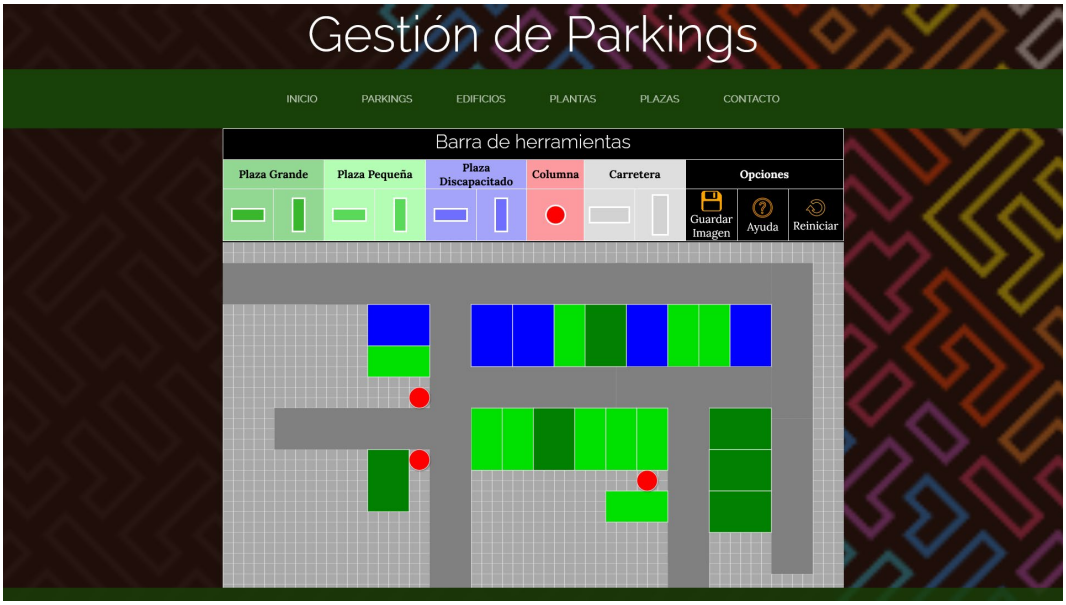
La Figura 41 muestra el resultado final del diseñador de *layout* de planta.

Figura 41. Diseñador de *layout* de planta



Para dotar de funcionalidad al diseñador se decidió crear una barra de herramientas desde la que el usuario pudiera pinchar sobre el elemento que quiera incluir en el plano, tales elementos se diferenciaron en plaza grande, plaza pequeña o plaza para discapacitados. También puede incorporar al diseño un elemento de carretera o un elemento de columna. A la hora de insertar un elemento de plaza o de carretera se le da la opción al usuario de seleccionar la orientación horizontal o vertical de esta. La Figura 42 muestra un ejemplo de diseño realizado con la aplicación.

Figura 42. Ejemplo de diseño de *layout*



La programación del diseñador de *layout* de plantas se encuentra en la ruta `/ParkingLotApp/static/js/grid-plazas.js`

El desarrollo de esta utilidad se ha realizado utilizando la librería *`Konva.js`*. El primer paso que se debe dar para trabajar con dicha librería consiste en crear un escenario (*stage*) donde se irán incorporando diferentes capas y elementos.

```
// ######################################################
// #####  CREO EL ESCENARIO DONDE VOY A DIBUJAR  ######
// ######################################################
var stage = new Konva.Stage({
  container: 'lienzo',
  width: width,
  height: height
});
```

A continuación se crean capas (*layer*) sobre las que se irán incluyendo diferentes elementos. En el desarrollo que nos ocupa el escenario dispone de dos capas diferenciadas.

Una primera capa que dibuja una rejilla (*grid*) de 15x15 píxeles que sirve como referencia al usuario para la colocación de las diferentes figuras. Además esta capa está dotada con una funcionalidad de *snap*, que hace que las formas se ajusten automáticamente a una posición de la cuadrícula de referencia mientras se arrastra el elemento, y de esa forma facilitar al usuario la inclusión de elementos en el plano.

```
// ######################################################
// #########  CREO UN GRID DE REFERENCIA   ############
// ######################################################
var gridLayer = new Konva.Layer();
var padding = blockSnapSize;
console.log(width, padding, width / padding);
for (var i = 0; i < width / padding; i++) {
  gridLayer.add(new Konva.Line({
    points: [Math.round(i * padding) + 0.5, 0, Math.round(i * padding) + 0.5,
height],
    stroke: '#ddd',
    strokeWidth: 1,
  }));
}

gridLayer.add(new Konva.Line({points: [0,0,10,10]}));
for (var j = 0; j < height / padding; j++) {
  gridLayer.add(new Konva.Line({
    points: [0, Math.round(j * padding), width, Math.round(j * padding)],
    stroke: '#ddd',
```

```
    strokeWidth: 0.5,
  }));
}
```

Una segunda capa que sirve de base para que se vayan colocando los elementos del diseño.

```
var layer = new Konva.Layer();

// ######################################################
// #####  AÑADO AL ESCENARIO EL GRID Y LA CAPA  #######
// ######################################################
stage.add(gridLayer);
stage.add(layer);
```

Una vez creadas las capas, básicamente, la funcionalidad del diseñador radica en ir añadiendo las formas que se quieran. Al pulsar con el botón izquierdo del ratón sobre las forma deseada de la barra de herramientas se hace una llamada a una función que crea la forma en la capa del escenario, el siguiente fragmento de código muestra las funciones que crean una Plaza Grande Horizontal y una Columna:

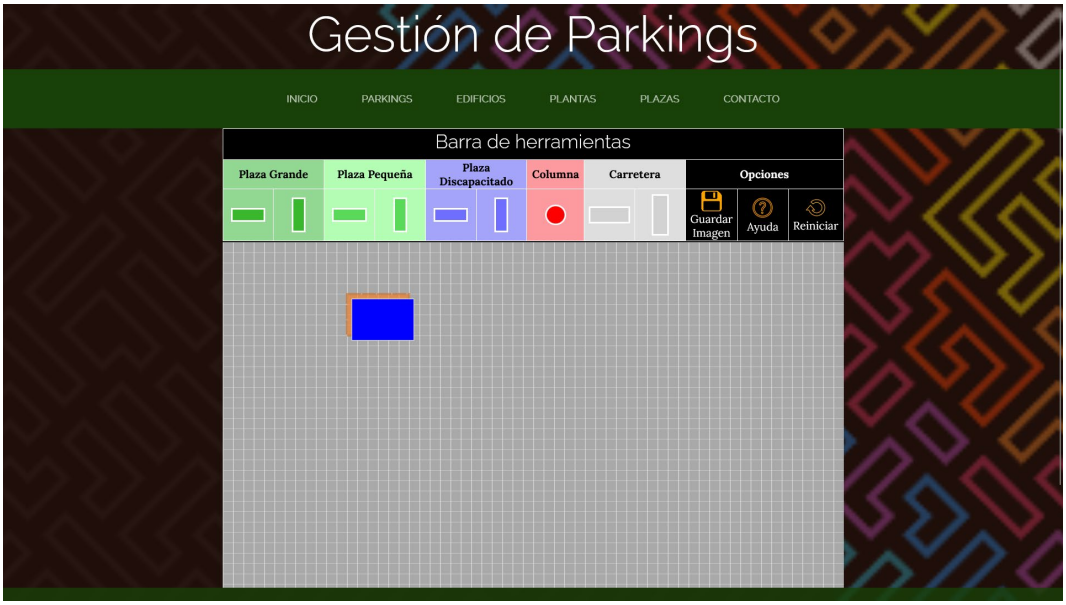
```
// ######################################################
// ########  FUNCIONES DE CREACIÓN DE FIGURAS  ########
// ######################################################

// Plaza grande HORIZONTAL
function nuevaPlazaGr_H(x, y, layer, stage) {
  // Creo un rectángulo de Konva
  let plazaGr_H = new Konva.Rect({
    x: x,
    y: y,
    width: blockSnapSize * 6,
    height: blockSnapSize * 4,
    fill: 'green',
    stroke: '#ddd', // borde
    strokeWidth: 1,
    draggable: true,
    id:'gr_' + plazaNum //identificador de plaza
  });
  // Evento inicio de "drag" (arrastrar)
  plazaGr_H.on('dragstart', (e) => {
    // Al comenzar a arrastrar una plaza muestro la sombra
    sombraPlazaGr_H.show();
    sombraPlazaGr_H.moveToTop();
    plazaGr_H.moveToTop();
  });
  // Evento fin de "drag" (arrastrar)
  plazaGr_H.on('dragend', (e) => {
    plazaGr_H.position({
```

```
      // Recalculo la posición de la plaza para que haga "snap" con el grid
      x: Math.round(plazaGr_H.x() / blockSnapSize) * blockSnapSize,
      y: Math.round(plazaGr_H.y() / blockSnapSize) * blockSnapSize
    });
    stage.batchDraw();
    // Oculto la sombra
    sombraPlazaGr_H.hide();
  });
  // Evento mientras hacemos un movimiento de "drag" (arrastrar)
  plazaGr_H.on('dragmove', (e) => {
    // Recalculo la posición de la sombra para que haga "snap" con el grid
    sombraPlazaGr_H.position({
      x: Math.round(plazaGr_H.x() / blockSnapSize) * blockSnapSize,
      y: Math.round(plazaGr_H.y() / blockSnapSize) * blockSnapSize
    });
    // Repintamos el área de Stage
    stage.batchDraw();
  });
  // Añado la plaza a la capa
  layer.add(plazaGr_H);
  plazaNum+=1;
  stage.batchDraw()
}

// Columna
function nuevaColumna(x, y, layer, stage) {
  // Creo un círculo de Konva
  let columna = new Konva.Circle({
      x: x,
      y: y,
      width: blockSnapSize * 2,
      height: blockSnapSize * 2,
      fill: 'red',
      stroke: '#ddd',
      strokeWidth: 1,
      shadowColor: 'black',
      shadowBlur: 2,
      shadowOffset: {x : 1, y : 1},
      shadowOpacity: 0.4,
      draggable: true,
      id:'co_' + columnaNum
  });
  columna.on('dragstart', (e) => {
      sombraColumna.show();
      sombraColumna.moveToTop();
      columna.moveToTop();
  });
  columna.on('dragend', (e) => {
    columna.position({
```

```
            x: Math.round(sombraColumna.x() / blockSnapSize) * blockSnapSize,
            y: Math.round(sombraColumna.y() / blockSnapSize) * blockSnapSize
        });
        stage.batchDraw();
        sombraColumna.hide();
    });
    columna.on('dragmove', (e) => {
      sombraColumna.position({
            x: Math.round(columna.x() / blockSnapSize) * blockSnapSize,
            y: Math.round(columna.y() / blockSnapSize) * blockSnapSize
        });
        stage.batchDraw();
    });
    layer.add(columna);
    columnaNum+=1;
    stage.batchDraw()
}
```

Tal como se puede observar en el código cuando se arrastra la figura (*drag*) se inician una serie de eventos que son interceptados por la aplicación: `dragstart`, `dragmove`, `dragend`. En ese momento se muestra una sombra que es la que realmente hace los cálculos para buscar la posición correcta y ajustarse a la rejilla. La Figura 43 muestra cómo actúa esta funcionalidad, se puede observar que la sombra naranja de la forma se ajusta a los vértices de la rejilla mientras que la forma azul no está ajustada, cuando el usuario suelte el botón del ratón, y así finalice el evento de arrastrar la forma, esta se coloca correctamente.

Figura 43. Sombra de autoajuste de la forma

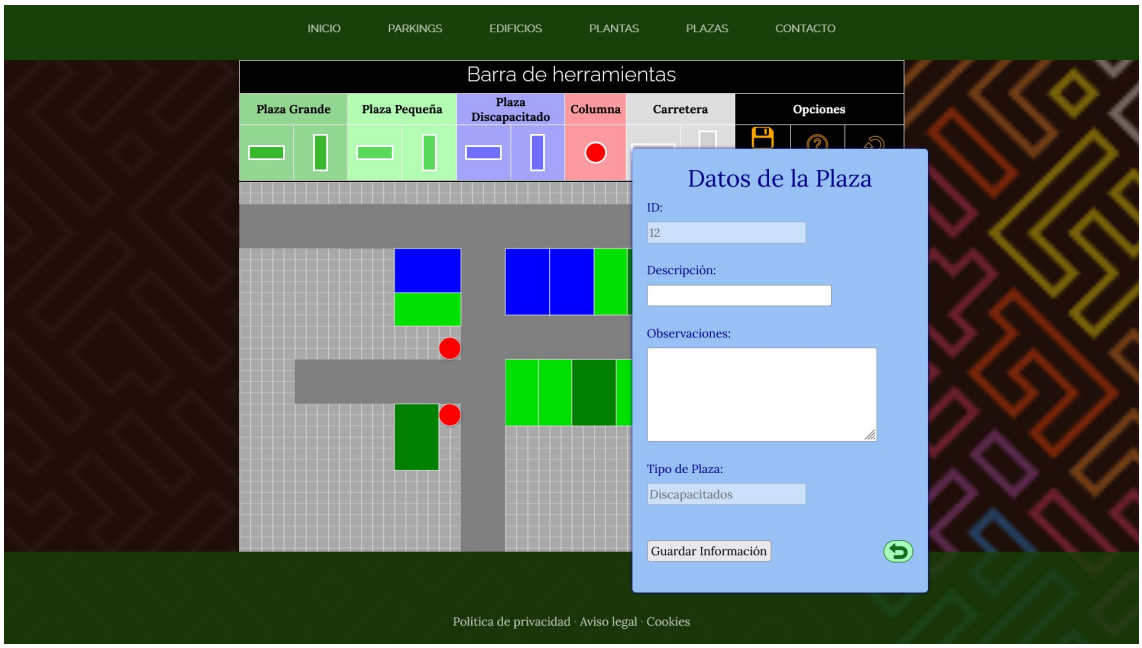


Para que los datos de las plazas se puedan almacenar en la base de datos se ha dotado al diseñador de un formulario en el que se solicita al usuario la información de cada plaza, sin tener que indicar el tipo de plaza puesto que ya lo ha hecho al seleccionarla desde la barra de herramientas. La Figura 44 muestra cómo, al pulsar con el botón derecho del ratón en una plaza, se muestran las opciones que dispone el usuario para actuar sobre esta.

Figura 44. Menú de opciones de las plazas

A continuación la Figura 45 presenta el formulario que se muestra para dotar de información a la plaza.

Figura 45. Incorporación de datos de plaza



El siguiente fragmento de código HTML muestra el formulario para incluir información de la plaza, dicho formulario se oculta y se muestra en función de las necesidades del usuario:

```
<div id="frmPlaza">
    <p class="text-center h2">Datos de la Plaza</p>
    <form>
        {% csrf_token %}
        <label for="plzID">ID:</label><br>
        <input type="text" name="plzID" id="plzID"><br><br>
        <label for="plzDescrip">Descripción:</label><br>
        <input type="text" name="plzDescrip" id="plzDescrip"><br><br>
        <label for="plzObserv">Observaciones:</label><br>
        <textarea name="plzObserv" id="plzObserv" rows="5" cols="30"></textarea>
        <br><br>
        <label for="plzTipo">Tipo de Plaza:</label><br>
        <input type="text" name="plzTipoPlaza" id="plzTipoPlaza"><br><br><br>
        <input type="button" name="btnGuardarInfo" id="btnGuardarInfo"
onclick="guardarInfoPlaza();" value="Guardar Información"> 
        <button type="button" id="btnCerrarAyuda"
onclick="cerrarFormulario();"><img src="{% static 'ParkingLotApp/img/volver.png'
%}"></button>
    </form>
</div>
<script src="{% static 'ParkingLotApp/js/grid-plazas.js' %}"></script>
```

Cuando el usuario pulsa el botón “Guardar Información” del formulario se envían los datos al servidor mediante *AJAX*, tal como se puede observar en el siguiente fragmento del código:

```
// ######################################################
// #####        GUARDAR DATOS DE LA PLAZA       #######
// ######################################################
function guardarInfoPlaza(){

  var objPlaza = {
    id_obj: "",
    descripcion: "",
    observaciones: "",
    planta: 0,
    tipo: 0,
    id_bd: 0,
  }
  var id_plaza=0;

  objPlaza.id_obj=document.getElementById('plzID').value;
  objPlaza.descripcion=document.getElementById('plzDescrip').value;
  objPlaza.observaciones=document.getElementById('plzObserv').value;

  var URLactual = window.location.href;
```

```
  var plantaURL=URLactual.slice(URLactual.lastIndexOf('/')+1,
URLactual.length);
  objPlaza.planta=Number(plantaURL);

  switch(document.getElementById('plzTipoPlaza').value){
    case 'Grande':
      objPlaza.tipo=1;
      break;
    case 'Pequeña':
      objPlaza.tipo=2;
      break;
    case 'Discapacitados':
      objPlaza.tipo=3;
      break;
  }

  // ENVIAR LOS DATOS AL SERVIDOR MEDIANTE AJAX
  // Creo un JSON con el objeto plaza
  const plazaJSON = JSON.stringify(objPlaza);

  // Envío el JSON al servidor con AJAX
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    // Aquí controlo la respuesta del servidor
    const respuesta=JSON.parse(this.responseText)
    console.log(this.responseText)
    if (respuesta.resultado=="OK"){
      alert(respuesta.mensaje);
      objPlaza.id_bd=respuesta.id;
      arrPlazas.push(objPlaza);
    }
    else{
      alert("Ha ocurrido un error al insertar la plaza.");
    }
  }

  //Llamo a la vista guardar_plaza_BD
  xhttp.open("POST", "/guardar_plaza_BD");
  xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
  xhttp.send("plaza=" + plazaJSON +
"&csrfmiddlewaretoken="+document.getElementsByName('csrfmiddlewaretoken')[0].v
alue);

  document.getElementById('frmPlaza').style.visibility='hidden';
}
```

Este código lo que hace es crear un elemento JSON y llamar a una vista de la aplicación principal que ejecuta el siguiente código:

```
def guardarPlazaBD(request):

    # data={'resultado':'OK', 'mensaje':'Dato insertado'}
    json_data=json.loads(request.POST.get('plaza'))

    plantaBD=Planta.objects.get(id=json_data["planta"])
    tipoPlazaBD=TipoPlaza.objects.get(id=json_data["tipo"])

    plzBD = Plaza(descripcion=json_data["descripcion"],
observaciones=json_data["observaciones"], id_planta=plantaBD,
id_tipo_plaza=tipoPlazaBD)
    plzBD.save()

    data={'resultado':'OK', 'mensaje':'Plaza insertada correctamente.', 'id':
plzBD.id}

    return JsonResponse(data, safe=False)
```

5. EVALUACIÓN DE LA APLICACIÓN

Cada una de las iteraciones genera una versión más completa del proyecto definitivo. Una vez finalizado el desarrollo de una aplicación en cada iteración, el siguiente paso debe consistir en validar que la misma cumple con los requisitos definidos y que realiza las tareas que se han diseñado de forma correcta.

En el presente capítulo se detallan las pruebas que se han llevado a cabo con las aplicaciones en cada iteración, con el objetivo de evaluar y comprobar el correcto funcionamiento de esta, así como detectar posibles mejoras.

Se han realizado dos tipos de pruebas:

- Pruebas de funcionamiento general, o pruebas de verificación.
- Pruebas de aceptación, o pruebas de validación.

5.1. Pruebas de funcionamiento general, o de verificación

Las pruebas de funcionamiento general permiten verificar que la ejecución de la aplicación es correcta y muestra los datos que se esperan en cada *interfaz*.

Para llevar a cabo este tipo de pruebas y su validación se han probado todas las *interfaces* de cada clase, cada método de las vistas y el acceso a los datos de los modelos, en la Tabla 11 se detallan las pruebas de funcionamiento general que se han llevado a cabo:

Tabla 11. Pruebas de funcionamiento general

Clase	***Interfaz***	**Prueba realizada**
Parkings	Listar *Parkings*	▪ El sistema muestra la pantalla correctamente si no existen registros en la tabla. ▪ El sistema muestra la pantalla correctamente con registros en la tabla correspondiente.
	Crear *Parking*	▪ El sistema genera registros correctamente desde el formulario de creación.

	Modificar *Parking*	▪ El sistema actualiza registros correctamente desde el formulario de modificación.
	Eliminar *Parking*	▪ El sistema elimina los datos del Parking de la Base de Datos. ▪ El sistema elimina en cascada los datos relacionados con el Parking (Edificios, Plantas y Plazas).
Edificios	Listar Edificios	▪ El sistema muestra la pantalla correctamente si no existen registros en la tabla. ▪ El sistema muestra la pantalla correctamente con registros en la tabla correspondiente.
	Crear Edificio	▪ El sistema genera registros de Edificio correctamente desde el formulario de creación.
	Modificar Edificio	▪ El sistema actualiza el registro de Edificio seleccionado correctamente desde el formulario de modificación.
	Eliminar Edificio	▪ El sistema elimina los datos del Edificio de la Base de Datos. ▪ El sistema elimina en cascada los datos relacionados con el Edificio (Plantas y Plazas) sin eliminar el registro del Parking al que estaba asociado.
Plantas	Listar Plantas	▪ El sistema muestra la pantalla correctamente si no existen registros en la tabla. ▪ El sistema muestra la pantalla correctamente con registros en la tabla correspondiente.

	Crear Planta	▪ El sistema genera registros de la Planta correctamente desde el formulario de creación.
	Modificar Planta	▪ El sistema actualiza el registro de la Planta seleccionado correctamente desde el formulario de modificación.
	Eliminar Planta	▪ El sistema elimina los datos de la Planta de la Base de Datos. ▪ El sistema elimina en cascada los datos relacionados con la Planta (Plazas) sin eliminar el registro del Edificio al que estaba asociado.
Plazas	Listar Plazas	▪ El sistema muestra la pantalla correctamente si no existen registros en la tabla. ▪ El sistema muestra la pantalla correctamente con registros en la tabla correspondiente.
	Crear Plaza	▪ El sistema genera registros de la Plaza correctamente desde el formulario de creación.
	Modificar Plaza	▪ El sistema actualiza el registro de la Plaza seleccionado correctamente desde el formulario de modificación.
	Eliminar Plaza	▪ El sistema elimina los datos de la Plaza de la Base de Datos. ▪ El sistema no elimina los datos relacionados con la Plaza (Tipo de Plazas) y sin eliminar el registro de la Planta a la que estaba asociada.
General	Contacto	▪ El sistema envía un correo electrónico desde el formulario de contacto.

5.2. Pruebas de aceptación, o de validación

En el presente apartado se detallan el conjunto de pruebas, de validación, que se ha elaborado para sean llevadas a cabo por usuarios que no hayan participado en el proceso de análisis y desarrollo del proyecto.

Tabla 12. Prueba de Aceptación-01. Alta, modificación y gestión de *Parkings*

PA-01	**Alta, modificación y gestión de un nuevo *Parking***
Descripción	El usuario debe ser capaz de dar de alta un nuevo *Parking* y modificar sus datos.
Pasos por seguir	El usuario debe acceder a la URL de la aplicación, una vez allí debe acceder al apartado de gestión de *Parkings* y dar de alta un nuevo *parking*, el sistema le mostrará el listado de todos los *parkings* de la base de datos, seguidamente el usuario deberá modificar el *parking* creado.
Resultado esperado	Alta de un nuevo Parking y modificación de sus datos.
Número de usuarios que realizan la prueba	3
Resultado obtenido	Sin errores.

Tabla 13. Prueba de Aceptación-02. Alta, modificación y gestión de Edificio

PA-02	**Alta, modificación y gestión de un nuevo Edificio**
Descripción	El usuario debe ser capaz de dar de alta un nuevo Edificio, asociándolo a un *Parking* y modificar sus datos.
Pasos por seguir	El usuario debe acceder a la URL de la aplicación, una vez allí debe acceder al apartado de gestión de Edificios y dar de alta un nuevo

	edificio, asignándolo al *parking* creado, el sistema le mostrará el listado de edificios de la base de datos, seguidamente el usuario deberá modificar el edificio creado.
Resultado esperado	Alta de un nuevo edificio y modificación de sus datos.
Número de usuarios que realizan la prueba	3
Resultado obtenido	Sin errores.

Tabla 14. Prueba de Aceptación-03. Alta, modificación y gestión de Planta

PA-03	**Alta, modificación y gestión de una nueva Planta**
Descripción	El usuario debe ser capaz de dar de alta una nueva Planta, asociándolo a un Edificio y modificar sus datos.
Pasos por seguir	El usuario debe acceder a la URL de la aplicación, una vez allí debe acceder al apartado de gestión de Plantas y dar de alta una nueva planta, asignándola al edificio creado, el sistema mostrará el listado de plantas de la base de datos, seguidamente el usuario deberá modificar la planta creada.
Resultado esperado	Alta de una nueva planta y modificación de sus datos.
Número de usuarios que realizan la prueba	2
Resultado obtenido	Sin errores.

Tabla 15. Prueba de Aceptación-04. Alta, modificación y gestión de Plaza

PA-04	**Alta, modificación y gestión de una nueva Plaza**
Descripción	El usuario debe ser capaz de dar de alta una nueva Plaza, asociándolo a una planta y modificar sus datos.
Pasos por seguir	El usuario debe acceder a la URL de la aplicación, una vez allí debe acceder al apartado de gestión de Plazas y dar de alta una nueva plaza, asignándola a la planta creada, el sistema le mostrará el listado de plazas de la base de datos, seguidamente el usuario deberá modificar la plaza creada.
Resultado esperado	Alta de una nueva plaza y modificación de sus datos.
Número de usuarios que realizan la prueba	2
Resultado obtenido	Sin errores.

Tabla 16. Prueba de Aceptación-05. Eliminación de Plaza

PA-05	**Eliminación de Plaza**
Descripción	El usuario debe ser capaz de eliminar los datos de una Plaza.
Pasos por seguir	El usuario debe acceder a la URL de la aplicación, una vez allí debe acceder al apartado de gestión de Plazas, en el listado de plazas mostrado por pantalla debe eliminar una a su elección.
Resultado esperado	Eliminación de los datos de la plaza seleccionada.

Número de usuarios que realizan la prueba	2
Resultado obtenido	Sin errores.

Tabla 17. Prueba de Aceptación-06. Eliminación de Planta

PA-06	**Eliminación de Planta**
Descripción	El usuario debe ser capaz de eliminar los datos de una Planta.
Pasos por seguir	El usuario debe acceder a la URL de la aplicación, una vez allí debe acceder al apartado de gestión de Plantas, en el listado de plantas mostrado por pantalla debe eliminar una planta a su elección.
Resultado esperado	Eliminación de los datos de la planta seleccionada y plazas que esta tenga asociadas.
Número de usuarios que realizan la prueba	3
Resultado obtenido	Sin errores.

Tabla 18. Prueba de Aceptación-07. Eliminación de Edificio

PA-07	**Eliminación de Edificio**
Descripción	El usuario debe ser capaz de eliminar los datos de un Edificio.
Pasos por seguir	El usuario debe acceder a la URL de la aplicación, una vez allí debe acceder al apartado de gestión de Edificios, en el listado de

	edificios mostrado por pantalla debe eliminar un edificio a su elección.
Resultado esperado	Eliminación de los datos del edificio seleccionado y de las plantas, y plazas que este tenga asociadas.
Número de usuarios que realizan la prueba	2
Resultado obtenido	Sin errores.

Tabla 19. Prueba de Aceptación-08. Eliminación de Parking

PA-08	**Eliminación de Parking**
Descripción	El usuario debe ser capaz de eliminar los datos de un Parking.
Pasos por seguir	El usuario debe acceder a la URL de la aplicación, una vez allí debe acceder al apartado de gestión de Parkings, en el listado de parkings mostrado por pantalla debe eliminar un *parking* a su elección.
Resultado esperado	Eliminación de los datos del parking seleccionado y de los edificios, plantas y plazas que este tenga asociadas.
Número de usuarios que realizan la prueba	3
Resultado obtenido	Sin errores.

Tabla 20. Prueba de Aceptación-09. Envío de email de contacto

PA-09	**Contacto**
Descripción	El usuario debe ser capaz de enviar un correo electrónico de contacto desde el formulario preparado a tal efecto.
Pasos por seguir	El usuario debe acceder a la URL de la aplicación, una vez allí debe acceder al formulario de Contacto y enviar un correo electrónico.
Resultado esperado	Recepción de un correo electrónico en la cuenta de administración
Número de usuarios que realizan la prueba	2
Resultado obtenido	Sin errores.

Tabla 21. Prueba de Aceptación-10. Diseño de *Layout* de Planta

PA-10	**Diseño de *Layout* de planta.**
Descripción	El usuario debe ser capaz de diseñar el *layout* de una planta.
Pasos por seguir	El usuario debe acceder a la URL de la aplicación, una vez allí debe acceder al apartado de gestión de Plantas, desde el listado de plantas existentes debe poder acceder al diseñador. El usuario podrá añadir o eliminar elementos al plano.
Resultado esperado	Diseño de un *layout* de planta seleccionando elementos de la barra de herramientas.

Número de usuarios que realizan la prueba	2
Resultado obtenido	Sin errores.

Tabla 22. Prueba de Aceptación-11. Inserción de datos de Plaza desde el diseñador

PA-11	**Inserción de datos de Plaza desde el diseñador.**
Descripción	El usuario debe ser capaz de dar de alta la información de una plaza en la base de datos desde el diseñador.
Pasos por seguir	El usuario debe acceder a la URL de la aplicación, una vez allí debe acceder al apartado de gestión de Plantas, desde el listado de plantas existentes debe poder acceder al diseñador, al incluir una plaza en el plano debe poder añadir su información y que esta se almacene en la base de datos.
Resultado esperado	Datos de la plaza almacenados en la tabla de Plazas.
Número de usuarios que realizan la prueba	2
Resultado obtenido	Sin errores.

6. CONCLUSIONES Y TRABAJO FUTURO

El presente apartado pretende plasmar las conclusiones que se han obtenido tras la elaboración del presente TFE y consecuente finalización del trabajo. También se dan unas pautas que puedan servir para realizar mejoras en el proyecto como parte de un trabajo futuro.

6.1. Conclusiones

El presente trabajo ha resultado enriquecedor y motivador. A continuación se exponen las conclusiones que se han obtenido con respecto a cada uno de los objetivos específicos que se detallaron en el apartado 1.3.2.:

1. Analizar los requisitos de usuario.

   La primera fase del trabajo consistió en el estudio de la fundamentación teórica analizando la situación del mercado de los parkings en la actualidad y la evolución de este. Una vez finalizada esa primera fase se pasó a detallar el contexto de la aplicación, en este punto se estudiaron herramientas que existen hoy en día para aplicar al dominio de la aplicación y se realizó un estudio de las tecnologías que se podrían utilizar para el desarrollo de esta, tal como se puede comprobar en el Capítulo 3 (CONTEXTUALIZACIÓN).

   Seguidamente se llevó a cabo un análisis de necesidades, tal como se puede ver en el apartado 4.2 (Alcance y Casos de Uso de la aplicación) se han definido una serie de casos de uso del proyecto, a su vez, estos, se desglosaron en diferentes requisitos funcionales (apartado 4.4.1) y se definieron unos requisitos no funcionales (apartado 4.4.1.6). Dichos requisitos funcionales permitieron hacer que el proyecto fuera más manejable y específico.

2. Diseñar la aplicación.

   Para abordar este objetivo específico se ha realizado un análisis del modelo de datos, con el que se diseñaría la base de datos, en la que la aplicación iba a almacenar la información, el apartado 4.3 detalla toda la gestión realizada en este aspecto. También se diseñó el diagrama de clases que interactúan en el proyecto, el cual puede

consultarse en el apartado 4.4.1.6. Seguidamente se diseñó el modelo de *interfaces* que tendría la aplicación, el cual se expone en el apartado 4.4.3.

3. Desarrollar la aplicación.

   El desarrollo de la aplicación se llevó a cabo siguiendo una metodología de desarrollo iterativa incremental, que se explica en el apartado 2.3.1, y que se culminó mediante una serie de iteraciones expuestas en el apartado 4.5. Cada iteración dio lugar a una versión más completa del proyecto final.

4. Realizar pruebas.

   Se definieron una serie de pruebas que se realizaron en dos fases:

   - Pruebas de funcionamiento general, o de verificación (apartado 5.1).
   - Pruebas de aceptación, o de validación (apartado 5.2).

   Las pruebas dieron como resultado la verificación del correcto funcionamiento de la aplicación y su validación por parte de usuarios.

5. Sacar conclusiones.

   Esta fase se pretende aglutinar las conclusiones obtenidas durante la elaboración del presente TFE y plasmar posibles trabajos futuros y mejoras. El presente capítulo muestra dichas conclusiones.

El proyecto consistía en desarrollar una aplicación de gestión de parkings y diseñar el *layout* de estacionamientos de este, una vez realizadas las pruebas oportunas, se puede considerar que el objetivo general está conseguido.

6.2. Trabajo futuro

La aplicación desarrollada deja abierta la posibilidad de realizar trabajo futuro de mejora del software desarrollado y, por otro lado, ampliar las funcionalidades de la aplicación mediante nuevas líneas de trabajo.

La aplicación se presenta como una primera versión en la que se ha querido plasmar la funcionalidad principal de los objetivos, pero se pueden plantear mejoras que hagan evolucionar y crecer la aplicación.

Haciendo un análisis de la aplicación en su estado actual, posibles trabajos futuros, como mejora de la versión desarrollada, podrían orientarse en las siguientes áreas:

- Creación de un sistema de autenticación y *login* a la aplicación, dotando al sistema de diferentes niveles de permiso para la realización de ciertas acciones.
- Mejora de los *interfaces* de la aplicación.
- Creación de informes.
- Creación de estadísticas.
- Ampliación del sistema para la gestión de entradas y salidas de vehículos, niveles de ocupación y cobro del tiempo de estacionamiento.
- Mejora del sistema de diseño de *layout* del *parking* dando la posibilidad de incluir puntos de interés tales como extintores, medianas, zonas no transitables y/o accesos entre otros.
- Dotar al sistema de la posibilidad de exportar el plano diseñado a un entorno que pueda volver a ser editable.
- Vincular un plano con la planta a la que pertenezca.

Con los ejemplos mostrados se deja abierta la posibilidad de realizar trabajos futuros sobre la aplicación.

6.3. Valoración personal

El proyecto ha supuesto tener que poner en práctica todos los conocimientos adquiridos, y las estrategias aprendidas, para que se pudieran controlar los procesos del ciclo de vida del software y estos permitieran finalizar el proyecto de forma satisfactoria. Durante dicho ciclo de vida del proyecto, desde el análisis hasta su validación, se adquirieron conocimientos importantes sobre la gestión de proyectos de software, la planificación del desarrollo, la implementación de pruebas, la documentación y la resolución de problemas, incluso la gestión de posibles mejoras.

Uno de los retos más importantes fue el de implementar el desarrollo con un *framework* de programación desconocido, en el caso que nos ocupa, *Django*. Supuso un esfuerzo tener que adaptarse a un entorno nuevo pero que, finalmente, ha resultado gratificante y que ha logrado querer profundizar más en el mismo para futuros proyectos. Entre los objetivos se puede comprobar que se pretendía realizar *interfaces* gráficas de uso sencillo y práctico, que no requirieran de unos conocimientos técnicos considerables, sino que, por el contrario,

pudieran ser utilizadas por cualquier persona. Se ha pretendido que las *interfaces* no sean únicamente funcionales sino que además puedan ser visualmente agradables.

Específicamente, se ha constatado que *Django* es un marco de desarrollo web de alta calidad, que permite la creación rápida y eficiente de aplicaciones web complejas. Ofrece una amplia gama de características y funcionalidades útiles, como la administración de bases de datos, el enrutamiento URL, la autenticación de usuarios y la seguridad, entre otras cosas. Además, la comunidad de desarrolladores de *Django* es muy activa y ofrece una amplia gama de recursos y documentación en línea.

En general, el desarrollo de la aplicación web ha sido una experiencia enriquecedora y satisfactoria, que ha permitido adquirir habilidades valiosas en el campo del desarrollo de software y ha proporcionado una base sólida para futuros proyectos.

REFERENCIAS BIBLIOGRÁFICAS

Balsells, J. F. (2004). *Guía de diseño de aparcamientos urbanos*.

Computer Systems Odessa. (2023). *Diagram Software and Drawing Tool | ConceptDraw DIAGRAM | ConceptDraw*. ConceptDraw DIAGRAM. https://www.conceptdraw.com/products/drawing-tool

Cultura Informática. (2011, marzo 1). *Historia de las bases de datos—Culturainformatica.es*. https://culturainformatica.es/articulos/historia-de-las-bases-de-datos/

Dirección General de Tráfico (DGT). (2021). *Anuario estadístico general 2021* (Anuario estadístico general 2021). DGT.

Django Software Foundation. (2023). *Django*. Django Project. https://www.djangoproject.com/

Garland, L., & Enright, M. (2022). *Parking Lot Simulator* [Processing]. https://github.com/lukegarland/ParkingLotSimulator (Original work published 2018)

INE. Instituto Nacional de Estadística. (2022, julio 1). *INE. Instituto Nacional de Estadística*. INE. https://www.ine.es/index.htm

Introducción a Django—Aprende sobre desarrollo web | MDN. (2022, diciembre 5). https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Introduction

Jones, P. (2014). The evolution of urban mobility. *IATSS Research*, *38*(1), 7-13. https://doi.org/10.1016/j.iatssr.2014.06.001

Máxima, J. (2022, octubre 5). *Historia del Automóvil: Resumen, evolución y características*. Enciclopedia Humanidades. https://humanidades.com/historia-del-automovil/

Metinvest. (2020, diciembre 9). Aparcamientos de acero: Historia y perspectivas — Metinvest. *Metinvest*. https://metinvestholding.com/es/media/news/avtomobiljnie-parkovki-iz-stali-istoriya-i-perspektivi

Oracle. (2022). *¿Qué es una base de datos?* https://www.oracle.com/mx/database/what-is-database/

ParkingYa. (2019, octubre 16). *Situación, estructura y características del Mercado de los Aparcamientos*. El Blog de parkingYa! https://www.parkingya.es/blog/situacion-estructura-y-caracteristicas-del-mercado-de-los-aparcamientos-2018-2019/

Pressman, R. S. (2005). *Software Engineering: A Practitioner's Approach* (6ª). Palgrave McMillan.

Pressman, R. S. (2010). *Ingenieria del Software. Un Enfoque Practico* (7ª). McGrawHill.

ProyectosAgiles.org. (2008, septiembre 27). Desarrollo iterativo e incremental. *Proyectos Ágiles*. https://proyectosagiles.org/desarrollo-iterativo-incremental/

Real Academia Española. (2022a). *Movible | Diccionario de la lengua española*. «Diccionario de la lengua española» - Edición del Tricentenario. https://dle.rae.es/movible

Real Academia Española. (2022b). *Movilidad | Diccionario de la lengua española*. «Diccionario de la lengua española» - Edición del Tricentenario. https://dle.rae.es/movilidad

Sánchez, M. (2021, marzo 22). *¿Qué es el stencil y cuáles son los orígenes de la técnica? | Blog*. Domestika. https://www.domestika.org/es/blog/7050-que-es-el-stencil-y-cuales-son-los-origenes-de-la-tecnica

Shahdeo, V. (2022). *Parking Lot Problem* [JavaScript]. https://github.com/vinitshahdeo/ParkingLot (Original work published 2019)

The MIT Press Reader. (2020, septiembre 3). From Chaos to Order: A Brief Cultural History of the Parking Lot. *The MIT Press Reader*. https://thereader.mitpress.mit.edu/brief-cultural-history-of-the-parking-lot/

Wiegers, K. E., & Beatty, J. (2013). *Software Requirements (Developer Best Practices)* (3rd Edition). Microsoft Press.

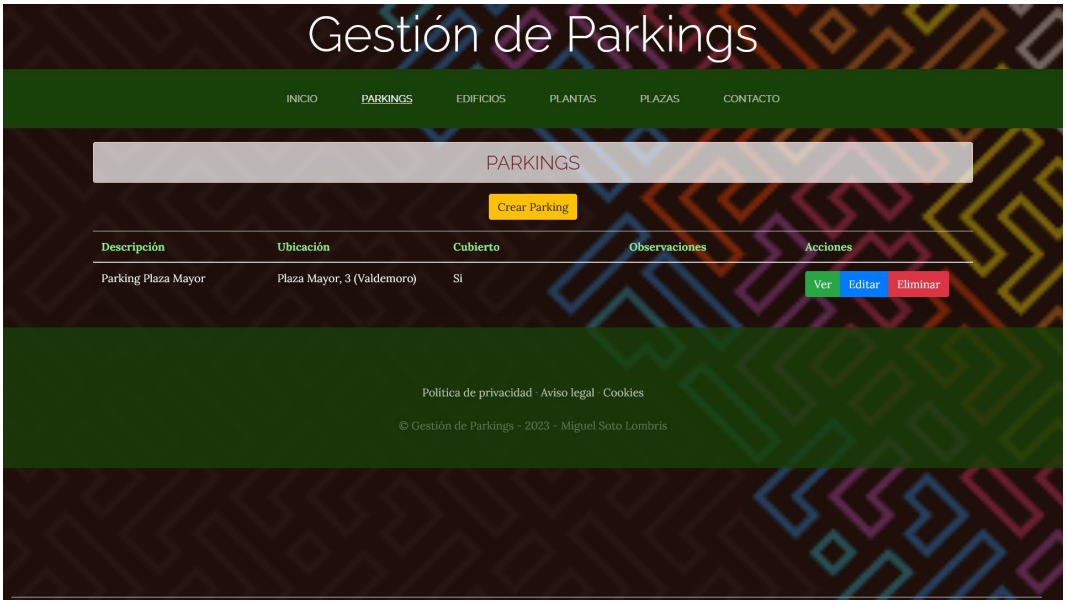
Young, R. R. (2004). *The Requirements Engineering Handbook*. Artech House.

Zofío, J. (2013). *Aplicaciones Web* (1.ª ed.). Macmillan.

Anexo A. *Interfaces* de Usuario

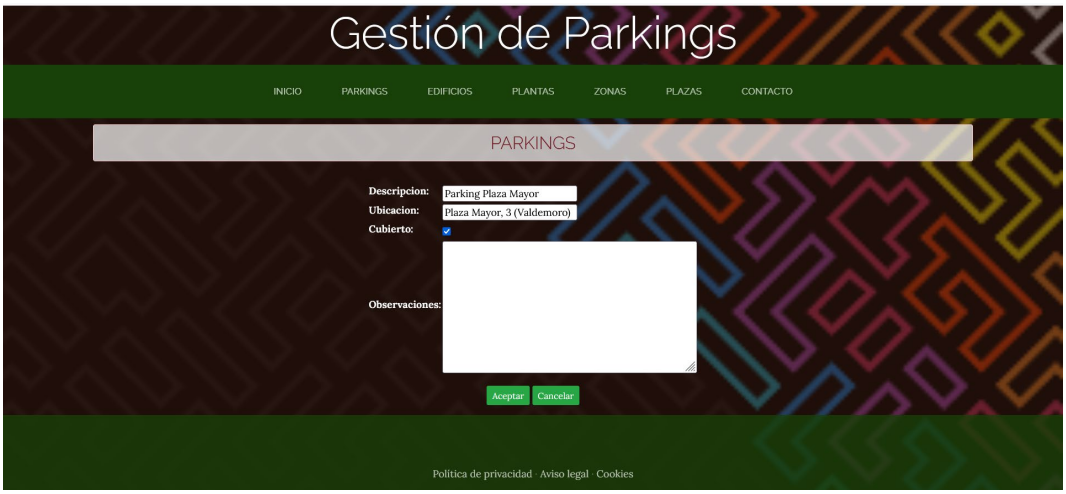
La Figura 46. *Interfaz* gráfica - Listar *Parkings* se centra en la *interfaz* creada para mostrar al usuario el listado de *parkings* almacenados en la aplicación.

Figura 46. *Interfaz* gráfica - Listar *Parkings*



La Figura 47. *Interfaz* gráfica – Crear / Actualizar *Parking* muestra la pantalla de creación o edición de *Parkings*.

Figura 47. *Interfaz* gráfica – Crear / Actualizar *Parking*



En el caso de que se tratase de una creación de un *parking* nuevo la *interfaz* gráfica sería la misma con la diferencia de que los campos del formulario aparecerían vacíos por defecto.

En la Figura 48. *Interfaz* Gráfica - Detalle de *Parking* se puede observar la pantalla diseñada para ver el detalle de información de un *parking*.

Figura 48. *Interfaz* Gráfica - Detalle de *Parking*

El proceso de eliminación de un *parking* se puede comprobar en la Figura 49. *Interfaz* Gráfica - Eliminar *Parking* y, una vez eliminado, en la Figura 50. *Interfaz* Gráfica - *Parking* Eliminado.

Figura 49. *Interfaz* Gráfica - Eliminar *Parking*

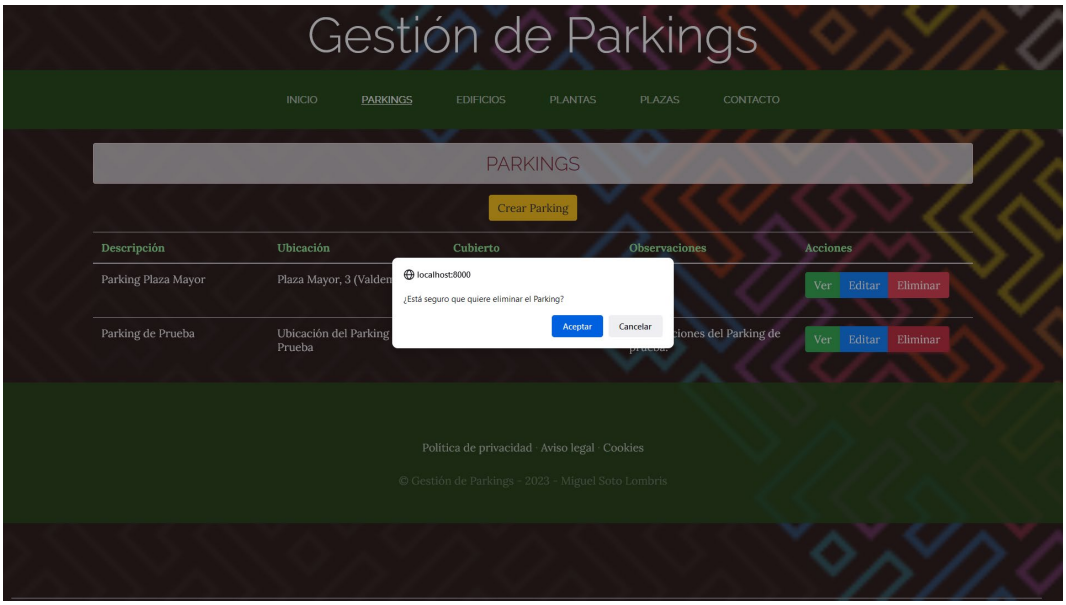
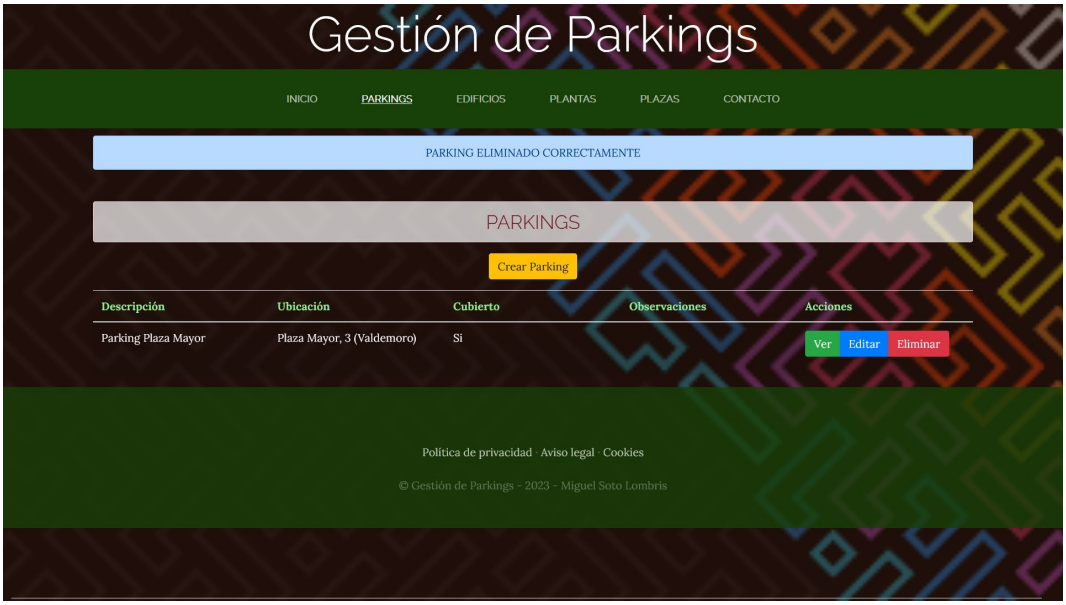


Figura 50. *Interfaz* Gráfica - Parking Eliminado

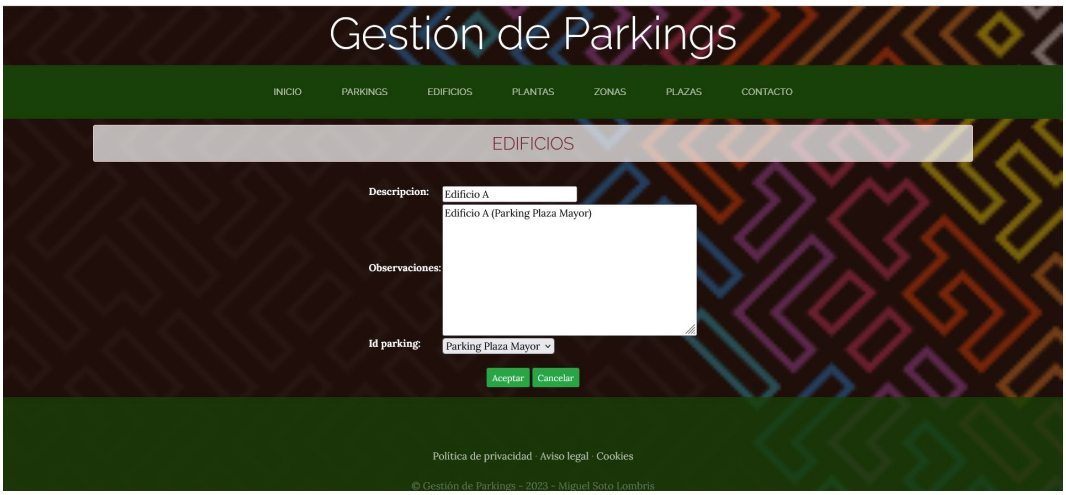


El resto de *interfaces* son similares, dotando al sistema de una consistencia en su apariencia. La gestión de Edificios se puede comprobar en la Figura 51. *Interfaz* Gráfica - Listar Edificios, Figura 52. *Interfaz* Gráfica - Detalle de Edificio, Figura 53. *Interfaz* Gráfica - Crear / Actualizar Edificio, Figura 54. *Interfaz* Gráfica - Eliminar Edificio y en la Figura 55. *Interfaz* Gráfica - Edificio Eliminado.

Figura 51. *Interfaz* Gráfica - Listar Edificios

Figura 52. *Interfaz* Gráfica - Detalle de Edificio

Figura 53. *Interfaz* Gráfica - Crear / Actualizar Edificio



En el caso de que se tratase de una creación de un edificio nuevo la *interfaz* gráfica sería la misma con la diferencia de que los campos del formulario aparecerían vacíos por defecto.

Figura 54. *Interfaz* Gráfica - Eliminar Edificio

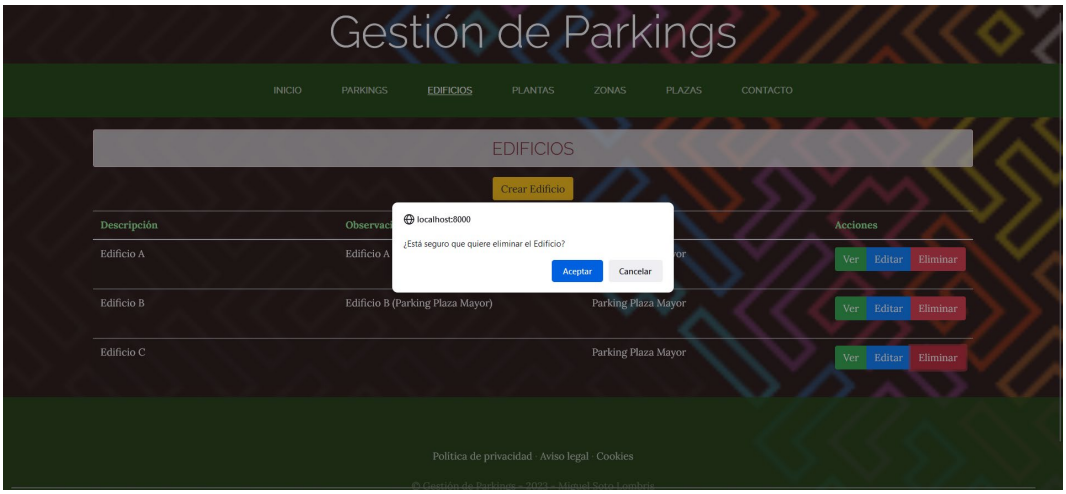
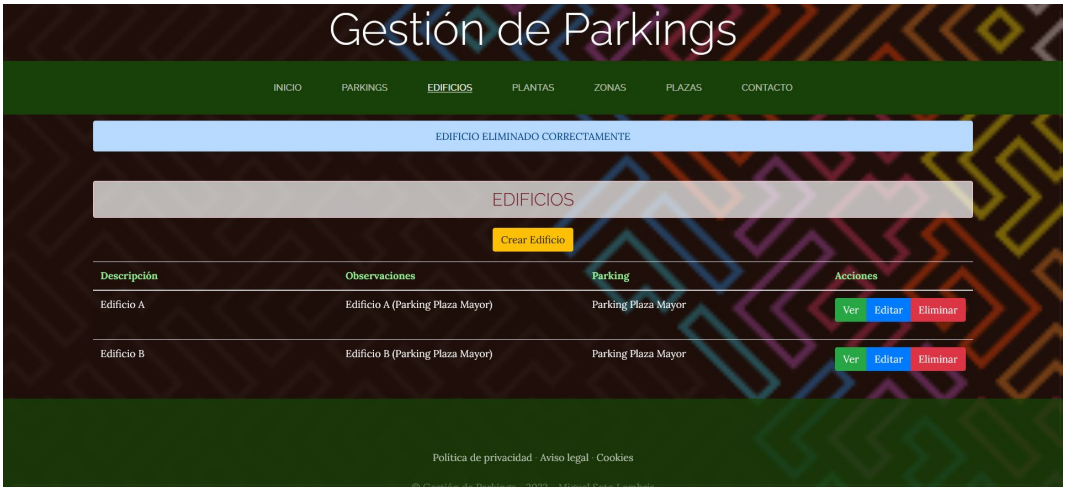


Figura 55. *Interfaz* Gráfica - Edificio Eliminado



La gestión de Plantas es similar a las anteriores, tal como se puede comprobar en la Figura 56. *Interfaz* Gráfica - Listar Plantas, Figura 57. *Interfaz* Gráfica - Crear / Actualizar Plantas y en la Figura 58. *Interfaz* Gráfica - Detalle de Planta

Figura 56. *Interfaz* Gráfica - Listar Plantas

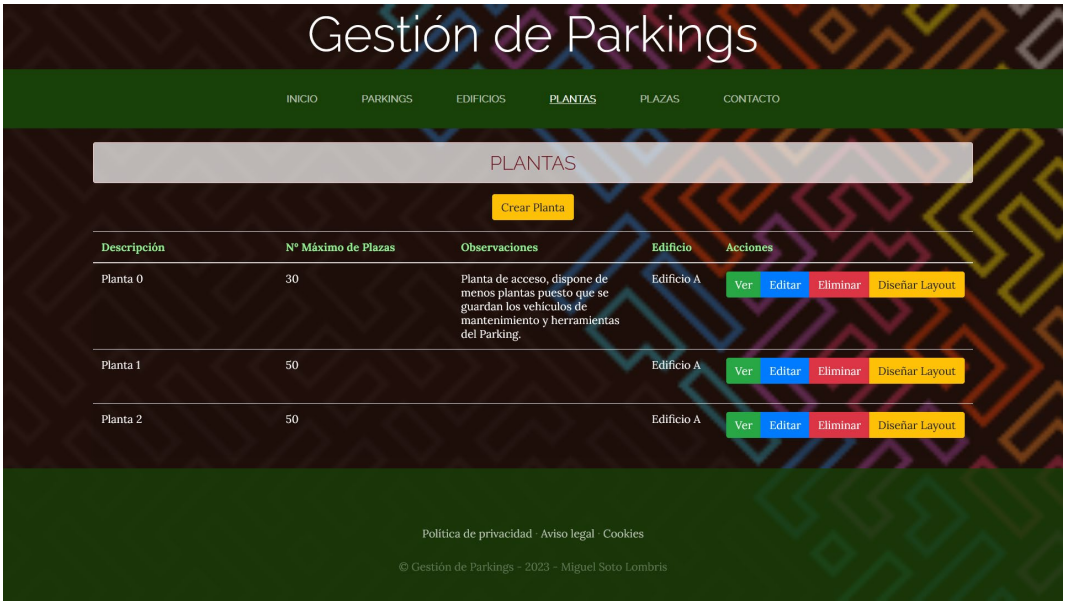


Figura 57. *Interfaz* Gráfica - Crear / Actualizar Plantas

En el caso de que se tratase de una creación de una planta nueva la *interfaz* gráfica sería la misma con la diferencia de que los campos del formulario aparecerían vacíos.

Figura 58. *Interfaz* Gráfica - Detalle de Planta

La gestión de Plazas tiene la particularidad de que se pueden dar de alta tanto de las *interfaces* creadas para ello o desde la *interfaz* generada para crear el diseño del *layout* de la planta. La Figura 59. *Interfaz* Gráfica - Listado de Plazas y la Figura 60. *Interfaz* Gráfica - Actualizar Plaza muestran las *interfaces* genéricas desarrolladas para la gestión de las plazas.

Figura 59. *Interfaz* Gráfica - Listado de Plazas

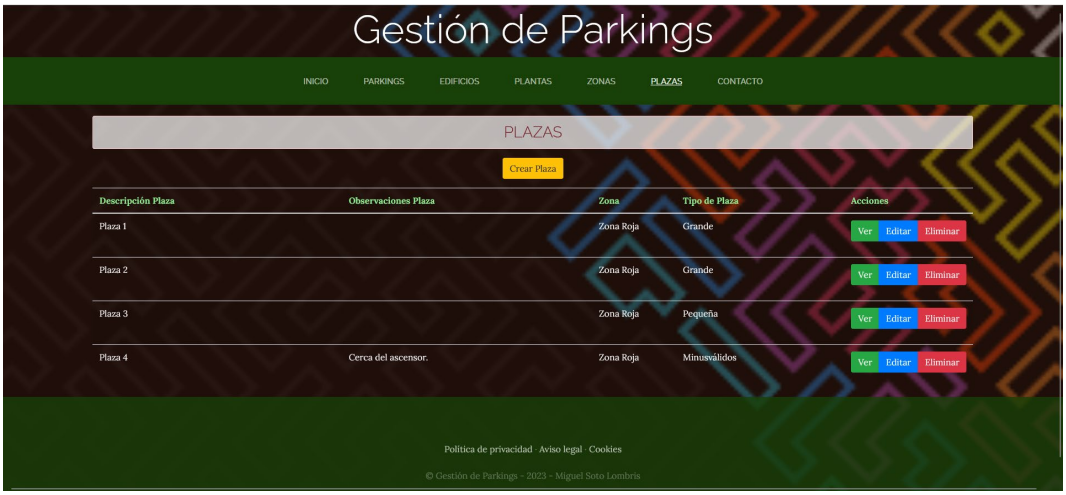
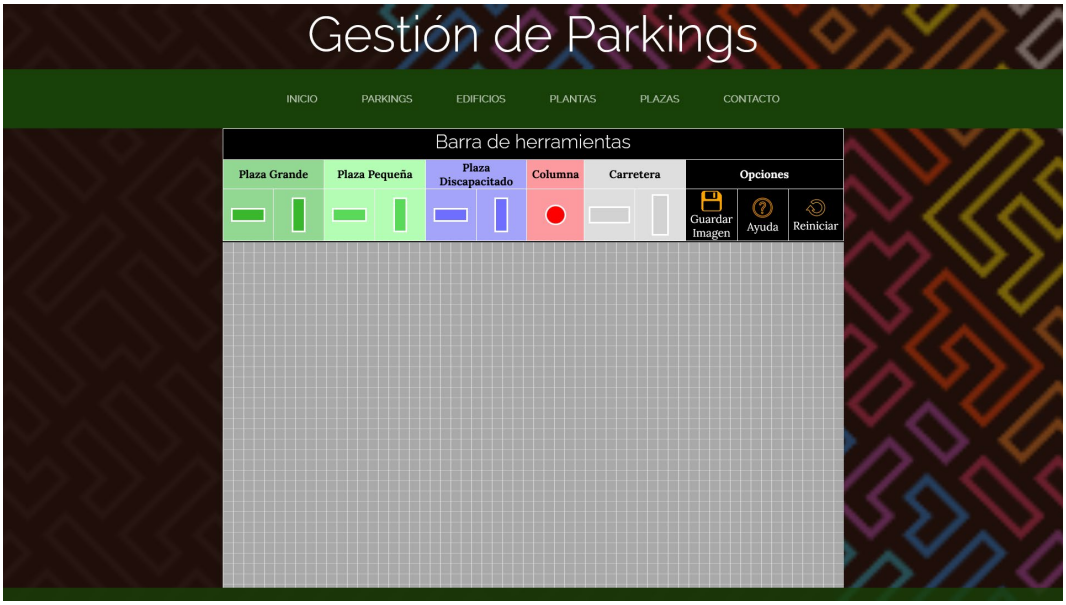


Figura 60. *Interfaz* Gráfica - Actualizar Plaza

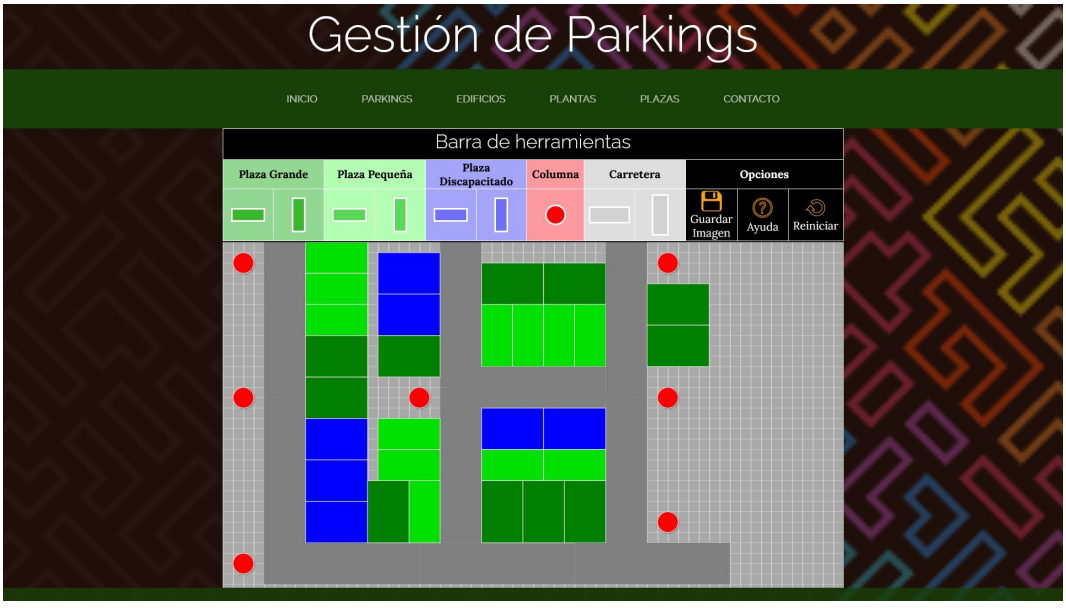
La Figura 61. *Interfaz* gráfica - Diseñador de *layout* de planta muestra el resultado final del diseñador de *layout* de plantas.

Figura 61. *Interfaz* gráfica - Diseñador de *layout* de planta



La Figura 62. *Interfaz* gráfica - Ejemplo de diseño muestra un ejemplo de *layout* diseñado con la aplicación.

Figura 62. *Interfaz* gráfica - Ejemplo de diseño



Para finalizar la Figura 63 muestra la ayuda generada para que el usuario pueda consultar la funcionalidad del diseñador de plantas.

Figura 63. *Interfaz* gráfica - Ayuda del diseñador de plantas