PROYECTO PLAZA APARCAMIENTO

Integrantes:
Adrián Pérez Domínguez (Encargado de comunicación).
Claudia Martí Bernal.
David Rodríguez Dagas (Jefe de proyecto).
Iván Brigos Vigo.
Sarah Povedano Barquilla.

Formato de escritura y tipografía:
La realización de la parte escrita de este proyecto se escribirá usando la tipografía "Times New Roman", con tamaño 10 en el cuerpo, 12 títulos secundarios y 14 en el título principal. Los encabezados de partes se escribirán en negrita y los títulos principales subrayados.

Formato de imágenes y tamaño:
El tamaño estándar de todas las imágenes en la memoria será 800x800 en formato ".png".

Índice

1.Introducción.

Imagina llegar a un aparcamiento y que un sistema te diga exactamente dónde puedes estacionar, sin necesidad de perder tiempo buscando un espacio libre o preguntándote si tu coche cabrá. Este es el objetivo de un sistema de aparcamiento inteligente: ofrecer información clara y en tiempo real sobre las plazas disponibles, las ocupadas y las reservadas, para que los conductores puedan estacionar de forma rápida y eficiente. Este tipo de sistema no mueve los coches automáticamente, pero sí actúa como una herramienta que facilita el proceso al guiar a las personas hacia las plazas libres más adecuadas para su vehículo.

En este proyecto, queremos crear un prototipo que permita visualizar una planta de aparcamiento con toda esta información, además de mostrar el movimiento de los vehículos mientras acceden o abandonan las plazas.

El problema desde la perspectiva del cliente:
Para los conductores que usan aparcamientos, los problemas más frecuentes suelen ser:

-Encontrar una plaza libre rápidamente: especialmente en momentos de alta ocupación o en aparcamientos grandes.
-Perder tiempo recorriendo el aparcamiento sin orientación: algo que puede ser frustrante y generar tráfico interno.
-No saber si el coche cabe en una plaza: lo que puede ser complicado para vehículos grandes o específicos.
-Maniobrar en espacios estrechos o poco accesibles: lo que puede generar estrés, especialmente para los conductores menos experimentados.
- Gestionar plazas reservadas: cuando hay áreas específicas para ciertos usuarios, como personas con movilidad reducida.

Un sistema de sensores que muestre con claridad dónde están las plazas disponibles y guíe a los conductores puede resolver estos problemas, haciendo que el proceso sea más sencillo, rápido y organizado.

El problema desde la perspectiva del desarrollo:
Desde el punto de vista técnico, crear un sistema de este tipo no es tarea sencilla. Hay varios desafíos importantes a superar, como:

-Representación del aparcamiento en tiempo real: es necesario diseñar un plano interactivo que refleje el estado de todas las plazas y las vías de acceso.
-Gestión de la información de las plazas: cada plaza debe indicar claramente su estado (libre y ocupado) y el tipo de vehículo que puede estacionar en ella, a través de sensores.
-Control de vehículos: el sistema debe registrar información como la matrícula y la fecha y hora de entrada de cada coche, vinculándolo con su plaza asignada.
-Visualización del movimiento de los vehículos: mostrar cómo los coches se desplazan dentro del aparcamiento, tanto al dirigirse hacia una plaza como al abandonarla.
-Adaptabilidad del sistema: asegurarse de que el prototipo funcione tanto para aparcamientos pequeños como para espacios grandes y complejos.

El objetivo del proyecto:
Con este prototipo, buscamos desarrollar una solución que combine estas funcionalidades, mostrando en un mapa interactivo la planta del aparcamiento, el estado de las plazas, y el movimiento de los coches en tiempo real. Aunque el sistema no estaciona automáticamente los vehículos, sí ofrece información clave para que los conductores puedan orientarse de forma eficiente y sin estrés. Esto mejorará la experiencia de los usuarios, reducirá el tiempo que se invierte en estacionar, y facilitará la gestión general del espacio, optimizando tanto el flujo interno como la ocupación.

Este proyecto, además de ser una herramienta funcional, representa un paso hacia la modernización de los aparcamientos, integrando tecnología que haga el día a día más sencillo para los conductores.

2. Estado del Arte.

El importante crecimiento de las ciudades inteligentes y la urgente necesidad de mejorar el uso eficiente del espacio en las zonas urbanas ha llevado a la gestión y señalización de las plazas de aparcamiento. Se han desarrollado diversos métodos para mejorar la experiencia de usuario y administrador, como por ejemplo: sistemas de sensores, aplicaciones móviles que determinan en tiempo real qué plazas de determinados aparcamientos están libres... Por otro lado, han surgido otras soluciones innovadoras de simulación como Sistemas de Información Geográfica (SIG), Internet de las Cosas (IoT), visión artificial…

Sensores para identificar disponibilidad de plazas (libres u ocupadas), aplicaciones móviles para interacción y gestión del usuario. La optimización del aparcamiento es sólo uno de los muchos elementos de un sistema de aparcamiento inteligente. Actualmente son necesarias una serie de mejoras en los sistemas utilizados, como el control en tiempo real del movimiento de los vehículos en los aparcamientos;la optimización de rutas dentro de parkings, etc...

Las principales tecnologías y métodos más importantes utilizados en el estacionamiento inteligente son los sistemas de gestión de estacionamiento inteligente (sistemas de sensores individuales y visión por computadora), los sistemas de información geográfica (GIS) de estacionamiento, la simulación y el modelado del movimiento de vehículos, aplicaciones móviles de gestión de aparcamientos, etc.

2.1 Sistemas de Gestión de Aparcamientos Inteligentes

El aparcamiento inteligente es una técnica que se basa en el Internet de las Cosas (IoT), en la cuál se usan sensores o cámaras para controlar y recoger información sobre las plazas libres de un parking. Los sistemas de gestión de aparcamientos inteligentes podemos dividirlos principalmente en dos: sistemas de sensores individuales y la visión por computadora. Ambos nos permiten averiguar el estado de las plazas (disponible u ocupada), pero presentan diferencias acerca de su manejo y procesamiento de la información.

2.1.1 Sistemas de Sensores Individuales

Este tipo de sistemas hacen uso de sensores, instalados en cada plaza de aparcamiento, para detectar si está ocupada por un vehículo o está libre y cambios en el entorno; por ejemplo el que podemos observar en la imágen proporcionada. Los tipos de sensores más utilizados son los de ultrasonido y los magnéticos. *ParkSense* y *Libelium* (una empresa española de IoT que ya ha implementado soluciones de este tipo en ciudades como Montpellier y en el Aeropuerto Internacional de Atenas), son ejemplos de sistemas que emplean esta tecnología. Una plataforma central recibe la información enviada por los sensores, a partir de la cual se distribuye entre los usuarios en tiempo real. No obstante, estos sistemas sólo pueden detectar ocupación y no informan acerca del movimiento dinámico de los vehículos dentro del aparcamiento.

2.1.2 Visión por Computadora

Otra alternativa interesante para controlar el aparcamiento son las cámaras de vigilancia y los algoritmos de visión por computadora. En vez de sensores físicos en cada plaza, las cámaras ofrecerán una representación visual del espacio y así determinar qué plazas hay libres y cuáles no. *ParkPlus* y *Parkopedia* son algunos de los sistemas que usan la visión por computadora para mejorar la gestión de los aparcamientos. Además, esta opción permite una descripción detallada de lo que está dentro de cada plaza (tipo vehículo, modelo vehículo, matrícula…); lo cuál puede ser bastante necesario en el caso de rotos y problemas en las distintas plazas para descubrir el motivo. Cabe destacar, que esto no es posible con los sensores tradicionales, pero tiene algunas limitaciones, como la alta demanda computacional y una dependencia total hacia la iluminación para un buen rendimiento.

En definitiva, los sistemas de gestión de aparcamientos tienen muchas ventajas tanto para los usuarios como para los administradores. Sorprendentemente, funciona de manera muy eficiente porque hay muy pocos trabajadores en el estacionamiento, lo que reduce en gran medida la posibilidad de error humano. Además, el proceso se vuelve más flexible al no existir colas unidireccionales en la entrada y el estacionamiento es completamente autónomo y bajo control el 100% del tiempo. Para los usuarios, esto significa que siempre recibirán información actualizada en tiempo real. Para los administradores aquí todo es mucho más sencillo porque el sistema genera automáticamente informes y registros. Otra gran ventaja es que los usuarios nunca se enfrentan al problema de encontrar espacio libre, minimizando así el espacio desperdiciado y reduciendo la congestión y, por tanto, la contaminación.

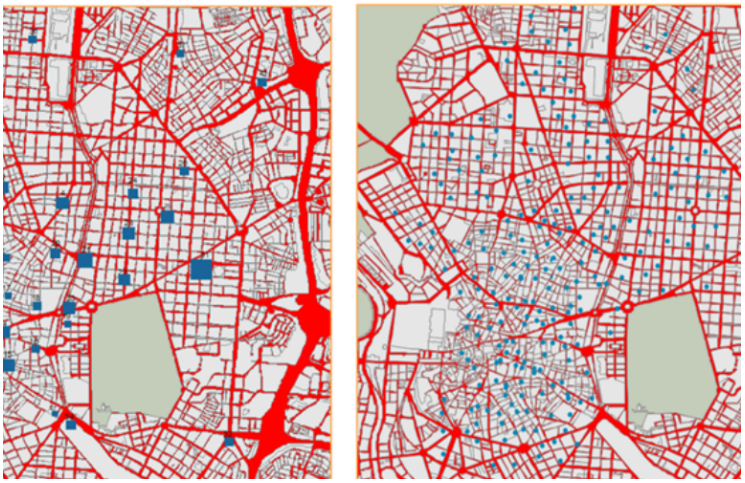
Por otro lado, se deben tener en cuenta ciertas desventajas: los costos de instalación son bastante altos, pero se amortizará con el tiempo debido al ahorro en los pagos regulares y la reducción de los costos de mantenimiento; Si los controladores no están familiarizados con el sistema, el rendimiento será deficiente; las averías suelen ser un problema, pero pueden solucionarse con un buen contrato de mantenimiento...

En resumen, las ventajas de un sistema de gestión de aparcamiento superan las desventajas y pueden superar cualquier inconveniente con una preparación adecuada. Estos sistemas proporcionan opciones eficientes, rentables y seguras para mejorar la gestión del estacionamiento en entornos urbanos o comerciales.

2.2 Sistemas de Información Geográfica (SIG) para Aparcamientos

Los Sistemas de Información Geográfica también han sido utilizados para representar y monitorear información espacial en áreas de aparcamiento. Estas soluciones permiten visualizar en tiempo real el estado de las plazas. Entre las herramientas más populares se encuentran el *QGIS* y el *ArcGIS*, estas se encargan de crear mapas interactivos que contienen información en tiempo real acerca del estado de las plazas. Además, estas herramientas pueden ser empleadas en conjunto con sensores o cámaras para representar gráficamente la cantidad de plazas ocupadas y la distribución del propio aparcamiento.

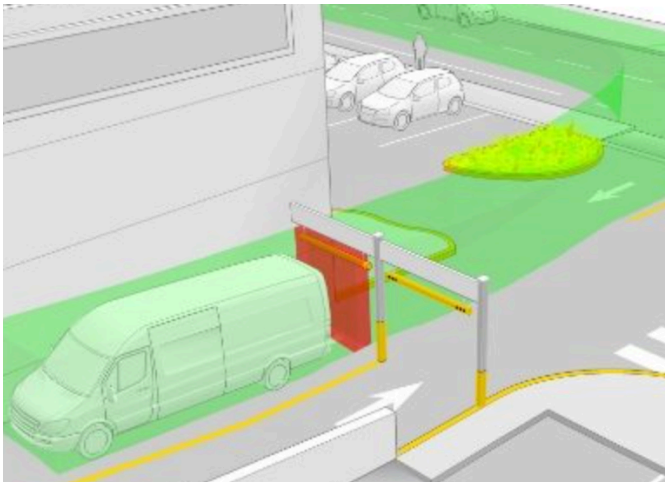
El beneficio más importante del uso de un SIG es la capacidad de administrar grandes cantidades de datos espaciales y crear representaciones visuales claras y detalladas. Por otro lado, la mayor desventaja es que no se adapta para simular el movimiento de los vehículos en aparcamientos en tiempo real. Además, un SIG es probablemente un sistema costoso y complicado para emplear en tantos establecimientos de tamaño más pequeño y con dificultades financieras.



2.3 Simulaciones y Modelado del Movimiento de Vehículos

En los últimos años, ha sido muy estudiada la simulación del movimiento dentro de los parking. Herramientas como *VISSIM* y *AIMSUN*, se utilizan para modelar el tráfico urbano y son adecuadas para simular el movimiento de vehículos en aparcamientos. Estas simulaciones permiten anticipar el flujo de vehículos en las vías de acceso y las plazas, optimizando las rutas de entrada y salida y reduciendo los tiempos de espera.

La precisión de las simulaciones en tiempo real es el principal desafío en este método. La eficiencia del sistema se basará en que la creación de los modelos reflejen de manera exacta el comportamiento real de los vehículos. Además, las simulaciones en tiempo real requieren altos recursos computacionales, lo que puede suponer un gran obstáculo para su aplicación generalizada.



2.4 Aplicaciones Móviles de Gestión de Aparcamiento

Actualmente, herramientas muy populares que ayudan a los usuarios a encontrar plazas de aparcamiento de forma rápida y en tiempo real son las aplicaciones móviles para la gestión de aparcamientos: ParkMe, Waze, Elparking… Estas son sólo algunas de las aplicaciones que proporcionan información sobre la disponibilidad de plazas de aparcamiento en parkings públicos o privados, utilizando datos y sensores de GPS en tiempo real. Además, algunas de estas apps permiten reservar plazas de aparcamiento, pagar electrónicamente… lo cual resulta muy cómodo para los usuarios.

Sin embargo, estas aplicaciones suelen estar diseñadas para estacionamiento al aire libre o en la calle, no para estacionamiento interior o subterráneo. Allí existen diferentes condiciones de señalización y acceso. Además, existen pocas simulaciones del movimiento de vehículos en los aparcamientos, lo que limita su funcionalidad.

2.5 Tecnologías de Base de Datos y Backend para Gestión de Información en Tiempo Real
Para poder gestionar eficazmente la información en tiempo real, es importante contar con una base de datos eficaz que permita actualizar rápidamente la información recopilada por sensores, cámaras u otros equipos de vigilancia mencionados anteriormente. Algunas de las tecnologías más comunes para gestionar comunicaciones en tiempo real entre dispositivos y plataformas centrales incluyen *Firebase*, *Socket.io* y *MQTT*. Proporcionan información actualizada sobre la disponibilidad de plazas de aparcamiento.

En términos de gestión de información espacial, existen herramientas muy prácticas para almacenar y analizar datos geográficos, como *PostGIS* y *PostgreSQL*. Con su ayuda, podremos ver más fácilmente los espacios de estacionamiento en el área y rastrear vehículos.

2.6 Conclusión del estado del arte
A pesar de los desafíos, los sistemas inteligentes de administración de aparcamientos presentan una solución potencial para aumentar la eficiencia en las zonas urbanas. Su instalación supondrá una mejora en la calidad de vida urbana y un efecto positivo en la sostenibilidad. Es necesario continuar progresando en la escalabilidad, interoperabilidad y en la reducción de costos para promover así una mayor adquisición. En el futuro, las ciudades inteligentes necesitan estas tecnologías para lograr sus objetivos de movilidad y sostenibilidad.

2.7 Crítica al estado del arte
Tras estudiar diferentes tipos de sistemas, las distintas tecnologías involucradas y sus implementaciones procederemos a hacer una breve crítica. En primer lugar, aunque existe una gran cantidad de artículos que estudian la optimización de espacios de aparcamiento, son pocos los que lo hacen en entornos abiertos y no controlados. Se ha de hacer un inciso también en la falta de conciencia por parte de los autores sobre la importancia de escalar estos sistemas y hacerlos fácilmente accesibles a través de servicios como la nube. Tampoco se preocupan por el tratamiento de los datos y su adaptación entre dispositivos de diferentes marcas. Para finalizar, aún siendo claramente beneficioso a nivel ecológico pocas ciudades han apostado por este tipo de propuestas inteligentes.

2.8 Propuesta
Partiendo de este análisis, este prototipo pretende añadir e innovar a las propuestas normales de sistemas relacionados con la gestión de aparcamiento con sus tecnologías hegemónicas y estandarizadas, creando una conveniente relación entre el usuario de la visualización y el vehículo que accede al parking, permitiendo esto funcionalidades como reconocer la minusvalía del dueño de un coche por su matrícula.

3. Análisis del problema.

Al desenvolver y dividir a nivel atómico la complejidad de la realización de la propuesta, nos encontramos con que las piedras angulares del prototipo serían las funcionalidades de registro y acceso a la aplicación (además de la modificación de los datos de usuario), la representación en la aplicación de los diferentes eventos que ocurren en planta y su relación con la realidad, la gestión de pagos y de los datos que el sistema reciba, la diferencia de capacidades de la aplicación dependiendo de si el usuario está a nivel de administrador. Siguiendo con estas ideas como funcionalidades y incluso añadiendo otras para asegurar una completa comprensión de todas las casuísticas que este prototipo pueda presentar, se desenvolvieron las historias de usuario que se presentan a continuación:

FEATURE: Registrarse en el sistema

SCENARIO: Registrar nuevo usuario
GIVEN: El usuario pulsó la opción "Registrarse como usuario"
WHEN: El nuevo usuario introduce sus credenciales y todas son válidas
THEN: El usuario es registrado satisfactoriamente

SCENARIO: Registrar nuevo administrador del sistema
GIVEN: El administrador pulsó la opción "Registrarse como administrador"
WHEN: El nuevo administrador introduce sus credenciales y todas son válidas
THEN: El administrador es registrado satisfactoriamente

SCENARIO: Registrar nuevo administrador del sistema
GIVEN: No existe administrador registrado
WHEN: El primer administrador pasa por el proceso de registro introduciendo como clave la clave especial única "2PV4SD35YU658ESX90", habilitada durante el desarrollo de la aplicación
THEN: El primer administrador es registrado satisfactoriamente

FEATURE: Introducir datos de registro

SCENARIO: Introducir nombre válido
GIVEN: El usuario o administrador está pasando por el registro
WHEN: El usuario escribe una cadena de caracteres en la sección de nombre
THEN: El sistema reconoce la cadena introducida como nombre como válida

SCENARIO: Introducir email válido
GIVEN: El usuario o administrador está pasando por el registro
WHEN: El usuario escribe una cadena de caracteres continuada de "@" y un servicio de correo electrónico (ejemplo: sarahpovedanobarquilla@gmail.com)
THEN: El sistema comprueba que el email no ha sido registrado ya y la reconoce como válida

SCENARIO: Introducir contraseña válida
GIVEN: El usuario o administrador está pasando por el registro o cambiando su contraseña
WHEN: El usuario escribe una cadena de caracteres, números y símbolos especiales, sin espacios (ejemplo: contraseña 1234$ingenieriasoftware)
THEN: El sistema comprueba que la contraseña no ha sido registrada ya y la reconoce como válida

SCENARIO: Introducir matrícula válida

GIVEN: El usuario está pasando por el registro o añadiendo matrículas a su perfil

WHEN: El usuario introduce 4 dígitos y 3 caracteres que excluyen la Ñ y la Q

THEN: El sistema reconoce la matrícula como válida

SCENARIO: Marcar minusvalía

GIVEN: El usuario está pasando por el registro

WHEN: El usuario marca la opción “Si” en la sección de minusvalía

THEN: Se abre una ventana emergente en la que se realiza el proceso de verificación

SCENARIO: Marcar no minusvalía

GIVEN: El usuario está pasando por el registro

WHEN: El usuario marca la opción “No” en la sección de minusvalía

THEN: El sistema reconoce al usuario como no minusválido

SCENARIO: Verificar minusvalía

GIVEN: El usuario marca la opción “Si” en la sección de minusvalía durante el registro

WHEN: El usuario envía un documento que verifique su minusvalía a través de la ventana emergente

THEN: La verificación queda en espera hasta que el administrador del sistema la apruebe

SCENARIO: Introducir clave de administrador

GIVEN: El administrador está pasando por el registro

WHEN: El administrador escribe una combinación de números y caracteres habilitada por otro administrador al momento del registro

THEN: El sistema reconoce la clave como válida

SCENARIO: Terminar de introducir datos de registro

GIVEN: El usuario o administrador está pasando por el registro

WHEN: El usuario pulsa el botón “Terminar Registro”

THEN: El sistema analiza que los contenidos de los campos sean todos válidos y se hayan rellenado todos

SCENARIO: Introducir datos de registro con formato no válido

GIVEN: El usuario o administrador pulsó “Terminar Registro”

WHEN: El formato de alguno de los campos no cumple con lo estipulado por el sistema

THEN: Salta el aviso “Error con el formato, inténtelo de nuevo” y borra el contenido del campo no válido

SCENARIO: No introducir todos los datos de registro

GIVEN: El usuario o administrador pulsó “Terminar Registro”

WHEN: Alguno de los campos está vacío

THEN: Salta el aviso “Es necesario rellenar todos los campos”

SCENARIO: Introducir los datos de registro correctamente

GIVEN: El usuario o administrador pulsó “Terminar Registro”

WHEN: El sistema reconoce todos los campos como válidos

THEN: Los datos de registro se guardan

FEATURE: Introducir credenciales

SCENARIO: Introducir email válido

GIVEN: El usuario o administrador está accediendo a la aplicación

WHEN: El usuario o administrador rellena el campo del email

THEN: El sistema comprueba que el email esté registrado

SCENARIO: Introducir contraseña válida

GIVEN: El usuario o administrador está accediendo a la aplicación

WHEN: El usuario o administrador rellena el campo de la contraseña

THEN: El sistema comprueba que la contraseña esté registrada

SCENARIO: Introducir clave de administrador

GIVEN: El administrador está accediendo a la aplicación

WHEN: El administrador escribe rellena la clave de administrador

THEN: El sistema comprueba que la clave esté registrada

SCENARIO: Introducir credenciales correctamente

GIVEN: El usuario o administrador está pasando por el proceso de acceso

WHEN: El sistema asocia todas las credenciales correctamente a un perfil

THEN: El usuario o administrador accede a la aplicación

SCENARIO: No introducir credenciales correctamente

GIVEN: El usuario o administrador está pasando por el proceso de acceso

WHEN: El sistema determina que las credenciales no están registradas o asociadas

THEN: El sistema notifica "Credenciales incorrectas"

SCENARIO: Repetir error en credenciales múltiples veces

GIVEN: El usuario ha introducido erróneamente sus credenciales 3 veces consecutivas

WHEN: El usuario vuelve a introducir erróneamente sus credenciales

THEN: El sistema no recibe peticiones de acceso desde ese dispositivo por 10 minutos

FEATURE: Acceder a la aplicación

SCENARIO: Acceder al sistema de visualización como usuario

GIVEN: El usuario está registrado

WHEN: El usuario concreta su email y contraseña

THEN: El usuario accede al sistema con las funcionalidades básicas

SCENARIO: Acceder al sistema de visualización como administrador

GIVEN: El administrador del sistema está registrado

WHEN: El administrador concreta sus credenciales

THEN: El administrador accede al sistema con su funcionalidad completa

SCENARIO: Dar error en el acceso

GIvEN: El usuario pasa por el proceso de acceso

WHEN: El usuario coloca un email y/o contraseña incorrectos

THEN: Salta el mensaje "Los datos de acceso no son correctos, por favor inténtelo de nuevo

SCENARIO: Olvidar contraseña

GIVEN: El usuario está registrado en el sistema y pasa por el proceso de acceso

WHEN: El usuario pulsa el botón "Olvidé mi contraseña"

THEN: El usuario modifica su contraseña respecto al procedimiento de modificación de contraseña (mencionado posteriormente)

FEATURE: Modificar datos posteriores al registro

SCENARIO: Modificar contraseña:

GIVEN:El usuario está registrado y ha accedido dentro del sistema a su perfil

WHEN: El usuario cambia su contraseña por una nueva

THEN: Se manda un email de comprobación de contraseña, con el que se valida ese cambio de contraseña

SCENARIO: Añadir matrículas

GIVEN: El usuario está registrado y ha accedido dentro del sistema a su perfil

WHEN: El usuario añade otra matrícula a su perfil

THEN: La matrícula añadida se guarda en el perfil

FEATURE: Representar cambios en los elementos del plano

SCENARIO: Discernir tipo de vehículo según su tamaño

GIVEN: El vehículo llega a la barrera de entrada

WHEN: Los sensores dedicados a aproximar tamaños capta la información necesaria

THEN: Al vehículo se le asocia una representación gráfica de su tamaño

SCENARIO: Visualizar un vehículo entrante

GIVEN: El vehículo llega a la barrera de entrada

WHEN: El sensor de la entrada capta al vehículo entrante

THEN: El vehículo se muestra en pantalla

SCENARIO: Representar vehículo en movimiento

GIVEN: El vehículo se desplaza hacia su plaza por el pasillo de circulación

WHEN: El vehículo pasa por los sensores distribuidos por el pasillo

THEN: El sistema estima a través de la distancia entre los sensores y el tiempo que tardan en activarse la velocidad a la que va el vehículo, y la simula en tiempo real en el visor

SCENARIO: Representar plaza ocupada:

GIVEN: La plaza está libre por defecto, lo que se representa con el color verde

WHEN: Un vehículo está aparcado en la plaza

THEN: La plaza está ocupada, representada con color rojo

SCENARIO: Representar plaza asignada al vehículo

GIVEN: El vehículo accede a la entrada del aparcamiento

WHEN: El sistema le asigna al vehículo una plaza específica

THEN: El número de la plaza se muestra sobre la representación gráfica del vehículo

FEATURE: Visualizar plano de aparcamiento completo

SCENARIO: Visualizar plano

GIVEN: El usuario o administrador accedió a la aplicación

WHEN: Los escenarios de la subfuncionalidad “Visualizar elementos del plano de aparcamiento” funcionan en conjunto

THEN: Se muestra el plano del aparcamiento de forma dinámica a tiempo real

FEATURE: Visualizar el plano del aparcamiento con las funcionalidades extras de administrador

SCENARIO: Acceder al terminal

GIVEN: El administrador ha accedido al visor desde una cuenta de administrador

WHEN: El visor se inicializa

THEN: El visor añade un terminal en el que se muestra información interna del sistema

SCENARIO: Mostrar datos del vehículo entrante

GIVEN: Un vehículo pasa por el procedimiento de entrada

WHEN: El sistema guarda los datos temporales del vehículo

THEN: Se muestra automáticamente en terminal “ {tamaño},{matricula} IN AT {hora entrada}, PARKING SPACE:{plaza asignada}

SCENARIO: Mostrar datos de vehículo

GIVEN: El vehículo cuya matrícula es introducida está en el parking

WHEN: El administrador escribe la matrícula del vehículo de interés en el terminal

THEN: Se muestra en terminal: "{matrícula}, {plaza asignada}, {hora entrada}"

SCENARIO: Mostrar datos del vehículo saliente

GIVEN: Un vehículo pasa por el procedimiento de salida

WHEN: El vehículo paga el importe y sale por la barrera de salida

THEN: Antes de que el sistema borre los datos, se muestra automáticamente en el terminal: "{matricula} OUT AT {hora actual}"

SCENARIO: Habilitar clave especial

GIVEN: El administrador ha accedido al sistema como administrador

WHEN: El administrador escribe "SET KEY {clave especial}"

THEN: la clave {clave especial} queda habilitada para el registro de un nuevo administrador del sistema

FEATURE: Gestionar los datos de los vehículos

SCENARIO: Introducir datos temporales de un vehículo

GIVEN: Un vehículo llega a la barrera de entrada

WHEN: Los sensores y el reconocedor de matrículas captan la información necesaria

THEN: Se guarda la matrícula, el tamaño, la hora de entrada y la plaza asignada del vehículo

SCENARIO: Reconocer si el vehículo tiene privilegio por minusvalía

GIVEN: La matrícula está registrada en un perfil que haya acreditado tener minusvalía

WHEN: El sistema asocia la matrícula a un perfil existente con minusvalía

THEN: El sistema incluye las plazas exclusivas para minusválidos a las posibilidades de elección de la plaza que asignar

SCENARIO: Asignar plaza al vehículo entrante

GIVEN: Los datos del vehículo se han tomado correctamente en la entrada

WHEN: El sistema busca una plaza que cumpla con las características del vehículo

THEN: El sistema le asigna una plaza al vehículo

SCENARIO: Introducir estado del vehículo

GIVEN: El vehículo tiene el estado "En movimiento" por defecto

WHEN: El vehículo es detectado por el sensor de la plaza y el reconocedor de matrículas comprueba que es el vehículo correcto

THEN: El vehículo pasa a tener el estado "Aparcado"

SCENARIO: Eliminar los datos temporales de un vehículo

GIVEN: El vehículo ha pasado por el trámite de entrada al aparcamiento

WHEN: El vehículo paga el importe y atraviesa la barrera de salida

THEN: El sistema borra los datos de estadía del vehículo

FEATURE: Gestionar los pagos:

SCENARIO: Calcular el importe total

GIVEN: El vehículo pasa por el trámite de entrada

WHEN: El vehículo es registrado en el sistema

THEN: El sistema guarda la hora de entrada del vehículo en el aparcamiento

SCENARIO: Efectuar el pago

GIVEN: El vehículo se sitúa en la barrera de salida

WHEN: El lector de matrículas lee la matrícula del coche que se propone salir

THEN: El sistema recupera la hora de entrada y calcula el pago

FEATURE: Utilizar la aplicación

SCENARIO: Acceder al plano completo de la planta de aparcamiento

GIVEN: El usuario está registrado

WHEN: El usuario accede al sistema

THEN: Se muestra en pantalla el plano del aparcamiento con todas las funcionalidades incluidas (movimiento de vehículos, plazas libres y ocupadas, …)

SCENARIO: Acceder al perfil

GIVEN: El usuario se encuentra en el hub principal (plano de planta)

WHEN: El usuario pulsa el botón “Perfil”

THEN: El usuario accede al perfil

SCENARIO: Volver al plano de planta

GIVEN: El usuario ha accedido al perfil

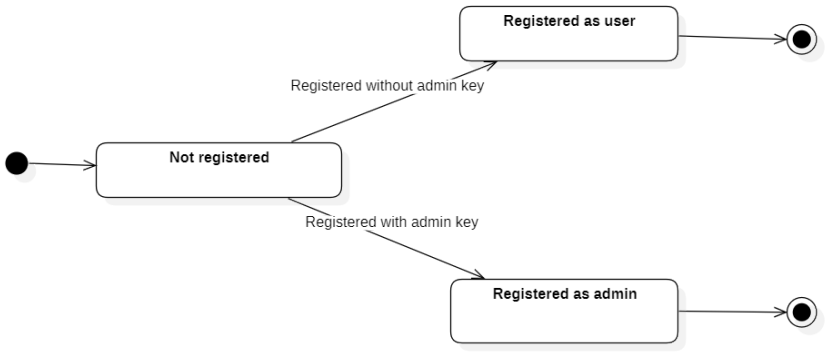
WHEN: El usuario pulsa el botón “Volver”

THEN: El usuario vuelve al plano de planta

4. Diseño de la solución.

Diagramas de estados

-Diagrama de estados de la clase ‘Usuario’:



-El proceso comienza con un usuario que está en el estado **‘**Not registered**’**, dependiendo de cómo se realice el registro:

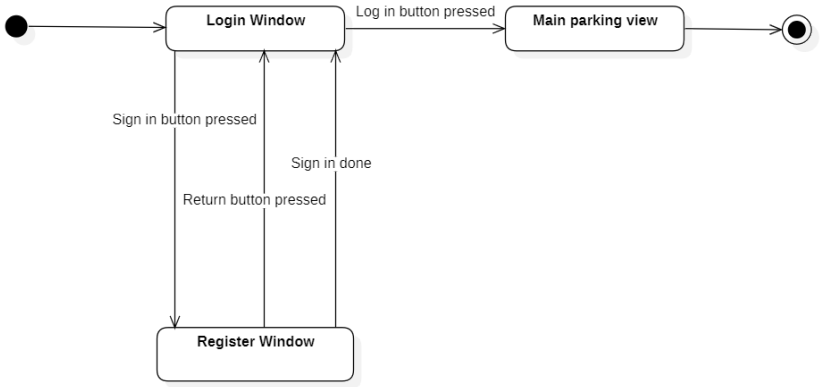
-Si no se utiliza una clave del administrador, el usuario cambia al estado **‘**Registered as user’.

-Si se utiliza una clave del administrador (validada por un sistema de verificación), el usuario cambia al estado ‘Registered as admin**’**.

Una vez registrado, el proceso termina en el estado final correspondiente.

-Diagramas de estados de la clase ‘Login Interface’:

1)



El flujo comienza en la ‘Login Window’, aquí el usuario tiene dos opciones:

-Iniciar sesión, presionando el botón correspondiente.

-Ir al registro, presionando el botón de registro.

Si el usuario selecciona la opción de registro:

-Se cambia a la ‘Register Window’, donde el usuario puede completar el registro.

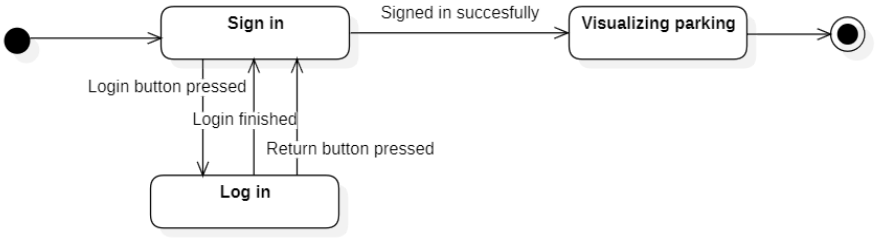
-Tras completar el registro exitosamente, el usuario vuelve automáticamente a la ‘Login Window’.

-Si decide no registrarse, puede regresar manualmente a la ‘Login Window’.

Una vez en la ‘Login Window’, si el usuario presiona el botón de inicio de sesión correctamente:

-Se accede a la ‘Main Parking View’, que marca el estado final.

2)



El flujo comienza en el estado ‘Sign in’ (Registro):

-Aquí el usuario puede completar el registro y, tras finalizarlo con éxito, cambiar directamente al estado de ‘Visualizing parking**’**.

-Alternativamente, el usuario puede optar por ir a la ventana de inicio de sesión **(‘**Log in’**)** presionando el botón de inicio de sesión.

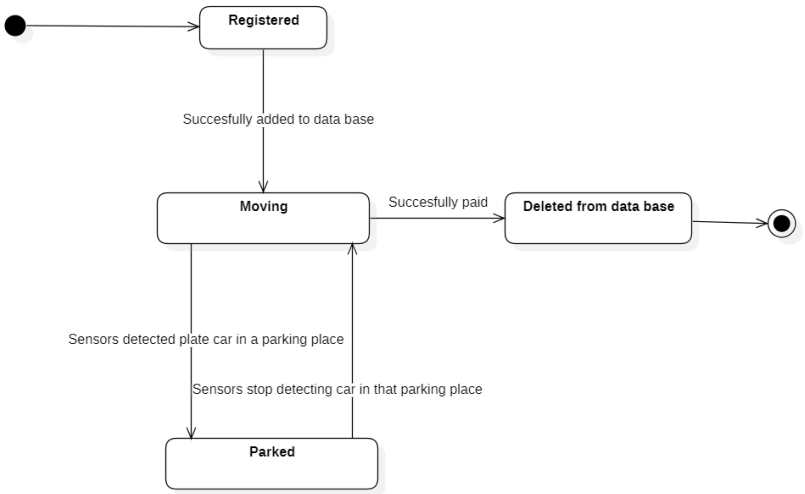
En el estado ‘Log in’, el usuario puede:

-Completar el inicio de sesión, lo que lo regresa al estado inicial ‘Sign in’ una vez finalizado.

-Regresar manualmente a la ventana de registro presionando el botón de regreso.

Una vez que el proceso de registro e inicio de sesión es exitoso, el usuario pasa al estado final, ‘Visualizing parking’, donde puede acceder al sistema principal del aparcamiento.

<u>-Diagrama de estado de la clase ‘Vehicle’:</u>



Estados principales:

-’Registered’: Este estado representa cuando un vehículo ha sido agregado exitosamente a la base de datos.

-’Moving’ : Este estado indica que el vehículo está en tránsito, posiblemente dentro del área monitoreada.

-’Parked’ : El vehículo está en un lugar específico, detectado por sensores en una plaza de aparcamiento.

-’Deleted from database’ : Este estado final ocurre cuando la información del vehículo ha sido eliminada del sistema.

Transiciones entre estados:

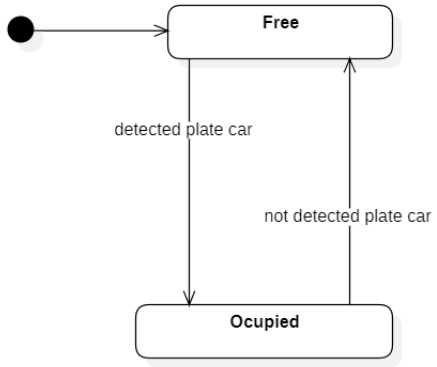
-De ‘Registered’ a ‘Moving’: Esto ocurre tras haber sido registrado exitosamente en la base de datos.

-De ‘Moving’ a ‘Parked’: Los sensores detectan la placa del vehículo en la plaza de aparcamiento, lo que activa el cambio a este estado.

-De ‘Parked’ a ‘Moving’: Los sensores dejan de detectar el vehículo en el lugar de aparcamiento, indicando que ha salido de la plaza.

-De ‘Moving’ a 'Deleted from database’: Esto ocurre tras realizar un pago exitoso, lo que elimina al vehículo del sistema.

-Diagrama de estados de la clase ‘Parking_place’:



El diagrama comienza en el estado ‘Free’ como estado predeterminado, indicando que el lugar está inicialmente disponible.

Estados principales:

-‘Free’: El lugar de estacionamiento está disponible y no se detecta ningún vehículo.

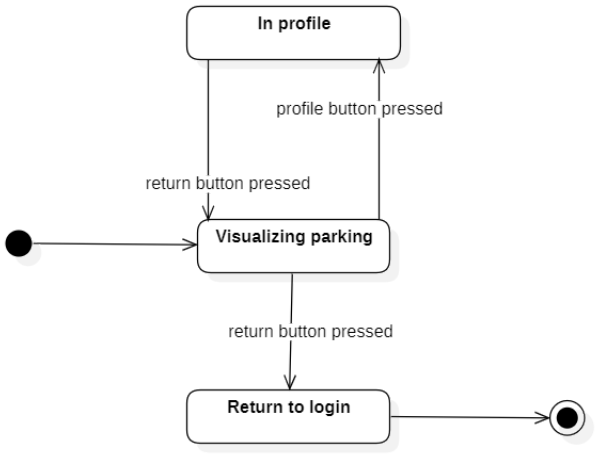
-’Occupied’ : El lugar está ocupado porque se detectó un vehículo estacionado.

Transiciones entre estados:

-De ‘Free’ a ‘Occupied’: Ocurre cuando se detecta la placa de un vehículo en el lugar de estacionamiento (detected plate car).

-De ‘Occupied’ a ‘Free’: Ocurre cuando el sistema deja de detectar la placa del vehículo, indicando que el espacio está vacío (not detected plate car).

-Diagrama de estados de la clase ‘Visualize_parking’:



El estado inicial es ‘Visualizing parking’, lo que indica que, al abrir el sistema, el usuario comienza viendo la información sobre estacionamientos.

Estados principales:

-’In profile’ : El usuario está dentro de la sección de su perfil.

-’Visualizing parking’ : El usuario está viendo información relacionada con el estado de los estacionamientos o las plazas disponibles.

-’Return to login’ : El usuario ha salido de las secciones y regresa a la pantalla de inicio de sesión.

Transiciones entre estados:

-De ‘Visualizing parking’ a ‘In profile’: Esto ocurre cuando el usuario presiona el botón para acceder a su perfil (profile button pressed).

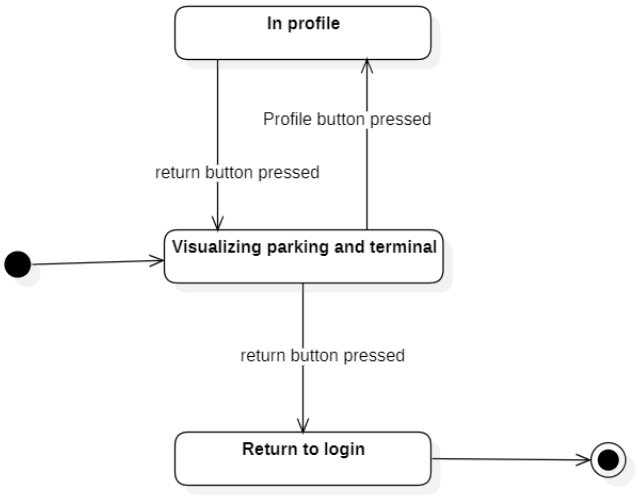
-De ‘In profile’ a ‘Visualizing parking’: El usuario regresa a la vista de estacionamiento al presionar el botón de regresar (return button pressed).

-De ‘Visualizing parking’ a ‘Return to login’: El usuario decide salir y regresar a la pantalla de inicio de sesión al presionar el botón de regresar (return button pressed).

Estado final:

-El estado ‘Return to login’ es el estado final del sistema, representado por un círculo negro rodeado.

-Diagrama de estados de la clase ‘Visualize_as_admin’:



Estados principales:

-‘In profile’: El administrador está dentro de la sección de su perfil, donde puede gestionar su información personal o realizar otras acciones administrativas.

-‘Visualizing parking and terminal’: El administrador está visualizando información relacionada con el estado de los estacionamientos o las plazas disponibles, probablemente para monitorear o gestionar los recursos.

-‘Return to login’: El administrador ha decidido salir de las secciones administrativas y regresa a la pantalla de inicio de sesión, finalizando su sesión activa.

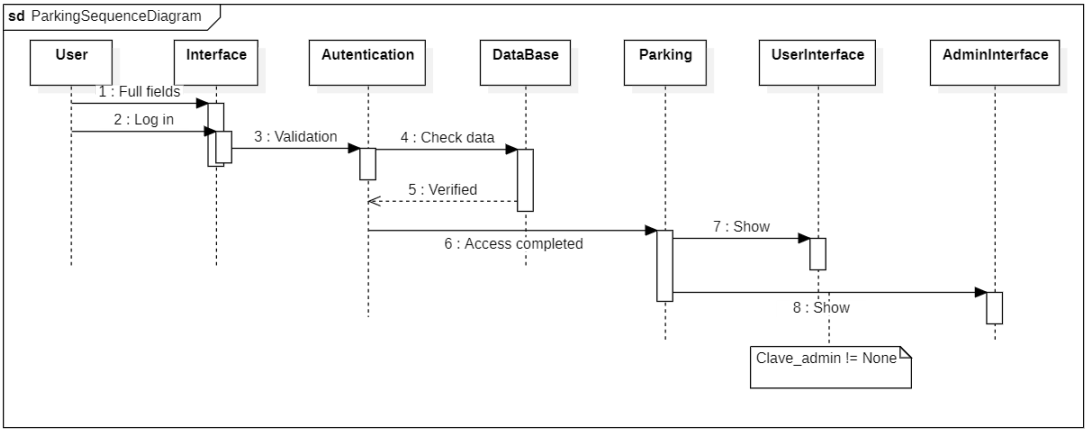
Transiciones entre estados:

-De ‘Visualizing parking and terminal’ a ‘In profile’: Esto ocurre cuando el administrador presiona el botón para acceder a su perfil desde la vista de estacionamientos y terminales (Profile button pressed).

-De ‘In profile’ a ‘Visualizing parking and terminal’: El administrador regresa a la vista de estacionamiento y terminal al presionar el botón de retorno en la interfaz de perfil (Return button pressed).

-De ‘Visualizing parking and terminal’ a ‘Return to login’: El administrador decide salir del sistema y regresa a la pantalla de inicio de sesión al presionar el botón de retorno (Return button pressed).

Diagrama de secuencia



User: ‘Full fields’ (El usuario completa los campos):

-El usuario introduce sus datos en la interfaz de inicio de sesión (nombre de usuario y contraseña).

User: ‘Log in’ (El usuario hace clic en "Iniciar sesión"):

-El usuario intenta iniciar sesión, y se envían los datos ingresados a la Interface.

Interface: ‘Validation’ (Validación de la interfaz):

-La interfaz recibe los datos del usuario e inicia el proceso de validación enviándolos al componente de autenticación.

Autentication: ‘Check data’ (Verificación de datos):

-El sistema de autenticación verifica las credenciales del usuario consultando los datos almacenados en la base de datos.

DataBase: ‘Verified’(Datos verificados):

-La base de datos responde confirmando si las credenciales son correctas o no.

Autentication: Access completed (Acceso completado):

-Una vez verificada la identidad del usuario, se completa el acceso. Aquí se determina si el usuario es administrador o usuario regular.

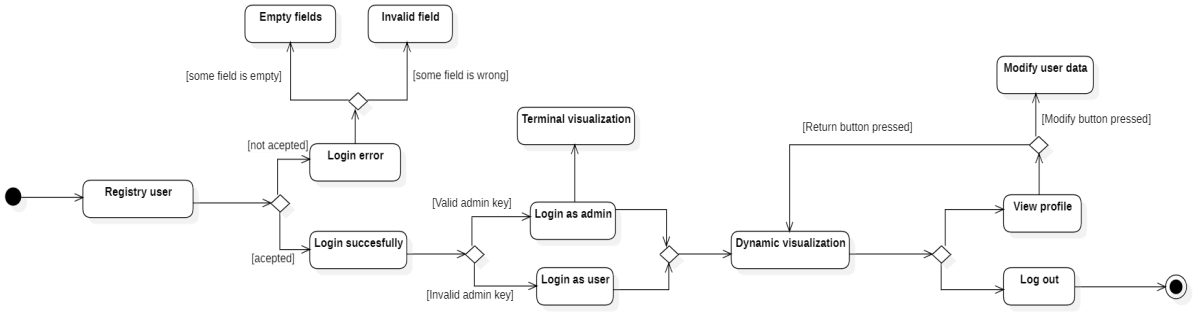
Parking: ‘Show’ (Mostrar información del estacionamiento):

-El módulo de estacionamiento comienza a mostrar la información relevante relacionada con el estacionamiento, como plazas disponibles u otras características. Esta información se muestra en la UserInterface.

AdminInterface: ‘Show’ (Mostrar interfaz de administrador):

-Si el usuario tiene permisos de administrador (condición: "Clave_admin != None"), se muestra la AdminInterface en lugar de la interfaz de usuario regular. Es decir, la clave o credencial de administrador permite acceder a esta sección.

Diagrama de actividad

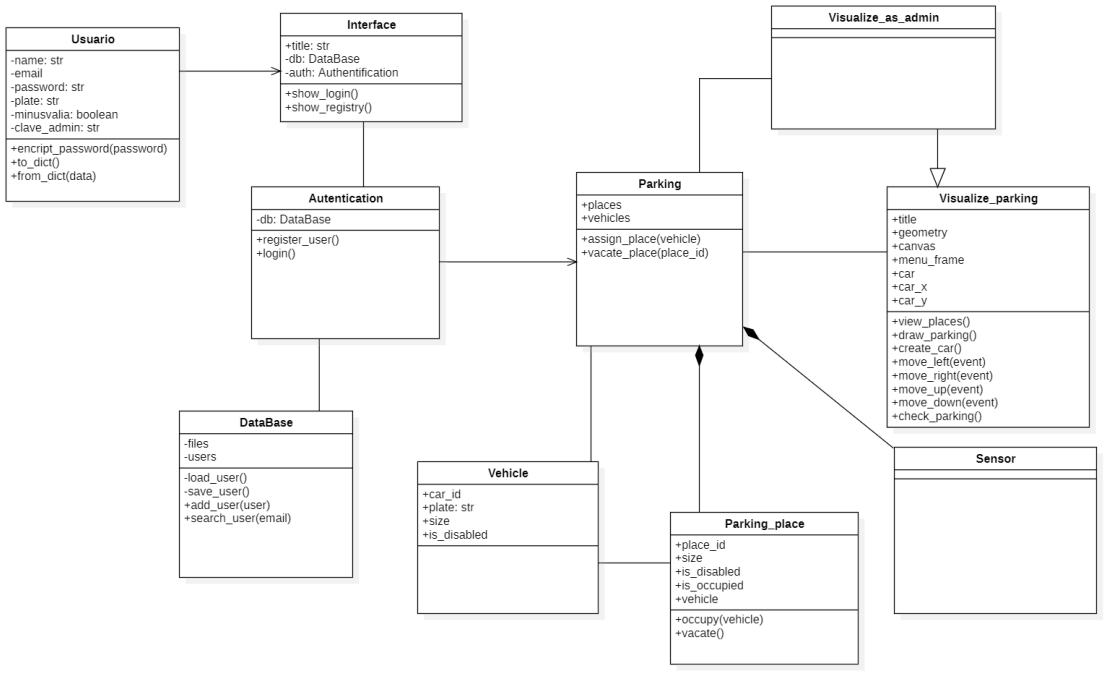


-El flujo comienza con el intento de inicio de sesión, el cual puede fallar si los campos están vacíos o incorrectos.

-Si el inicio de sesión es exitoso, se determina si el usuario tiene acceso como administrador o como usuario normal, lo que afecta las opciones a las que puede acceder.

-Dependiendo de su rol (administrador o usuario), el usuario puede ver la visualización de terminales, modificar datos de usuario, ver su perfil o cerrar sesión.

Diagrama de clases



El sistema gestiona el aparcamiento, usuarios y vehículos, con funcionalidades de autenticación y visualización gráfica.

-Usuario: Representa al usuario del sistema. Contiene información como nombre, email, contraseña (encriptada), placa del vehículo, y si tiene necesidades especiales. Permite convertir datos a diccionario o crear instancias desde uno.

-Interface: Maneja la interacción gráfica con el usuario, mostrando pantallas para registro e inicio de sesión.

-Autentication: Administra el registro e inicio de sesión, utilizando la clase DataBase para validar usuarios.

-DataBase: Guarda y gestiona usuarios en archivos. Permite cargar, guardar, agregar y buscar usuarios por email.

-Parking: Administra los lugares (Parking_place) y vehículos estacionados. Puede asignar o liberar lugares.

-Vehicle: Representa los vehículos con atributos como matrícula, tamaño y si pertenece a una persona con discapacidad.

-Parking_place: Representa los lugares de estacionamiento, indicando tamaño, si está ocupado, y si es para discapacitados. Permite ocupar o liberar el lugar.

-Visualize_parking: Proporciona una interfaz gráfica para mostrar y gestionar el estacionamiento. Permite ver la movilidad de los vehículos y verificar el estado de la plaza de aparcamiento.

-Sensor y Visualize_as_admin: No tienen atributos definidos, pero se intuyen como apoyo para monitoreo y visualización con permisos administrativos.

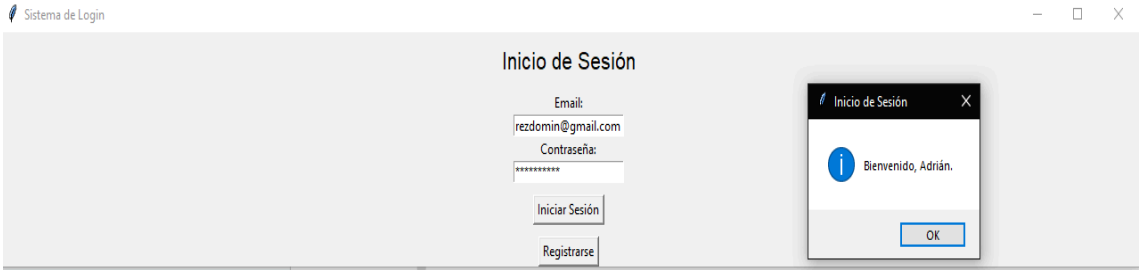
5. Pruebas.

Para probar que la lógica de las diferentes funcionalidades es correcta, se enfrentó al prototipo con una serie de situaciones que en las historias de usuario ya fueron convenientemente consideradas, permitiendo ver si el enfoque del prototipo es correcto si este responde de manera esperada a estos escenarios.

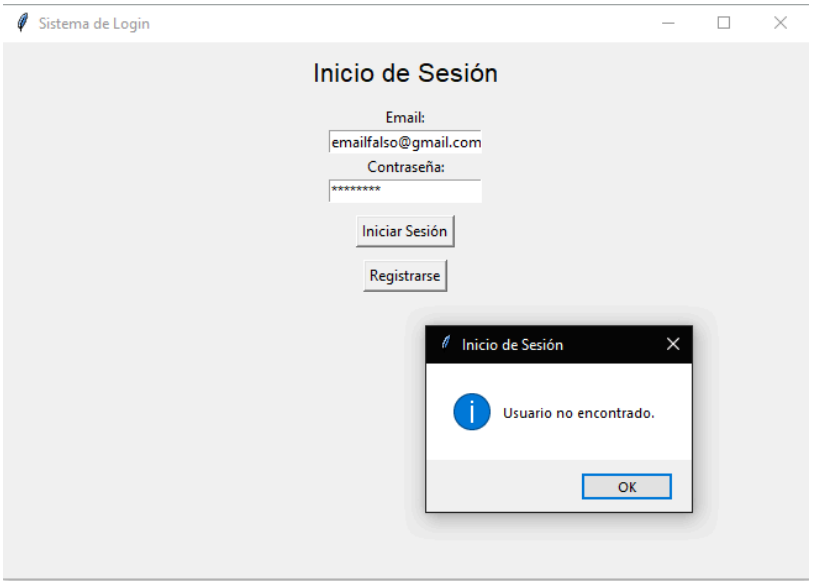
Respecto al acceso y el registro, el prototipo respondió de forma esperada a las siguientes pruebas:

Un registro de usuario exitoso:

El acceso de un usuario ya registrado:



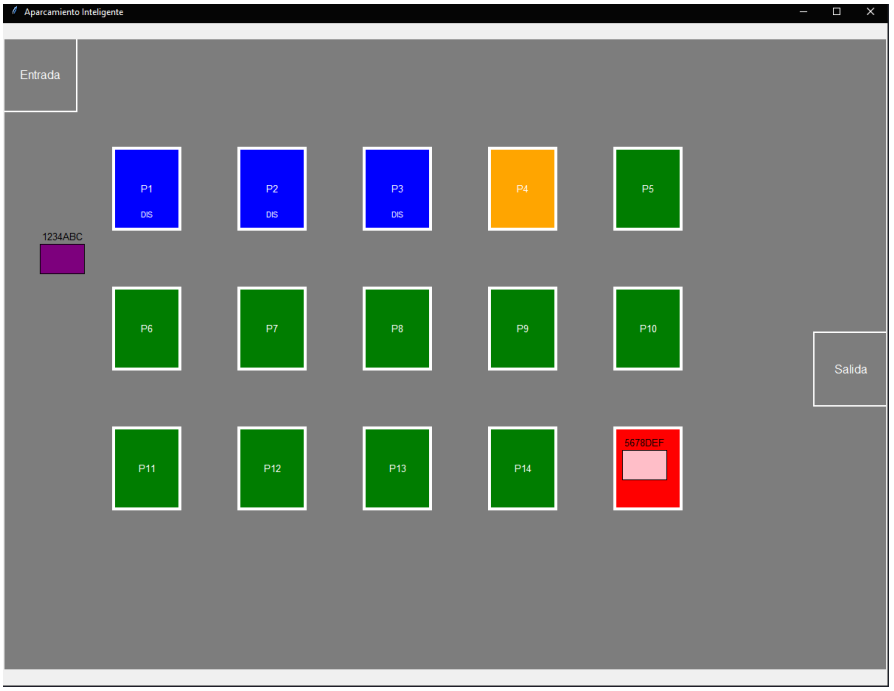
El intento de acceso de un usuario no registrado:



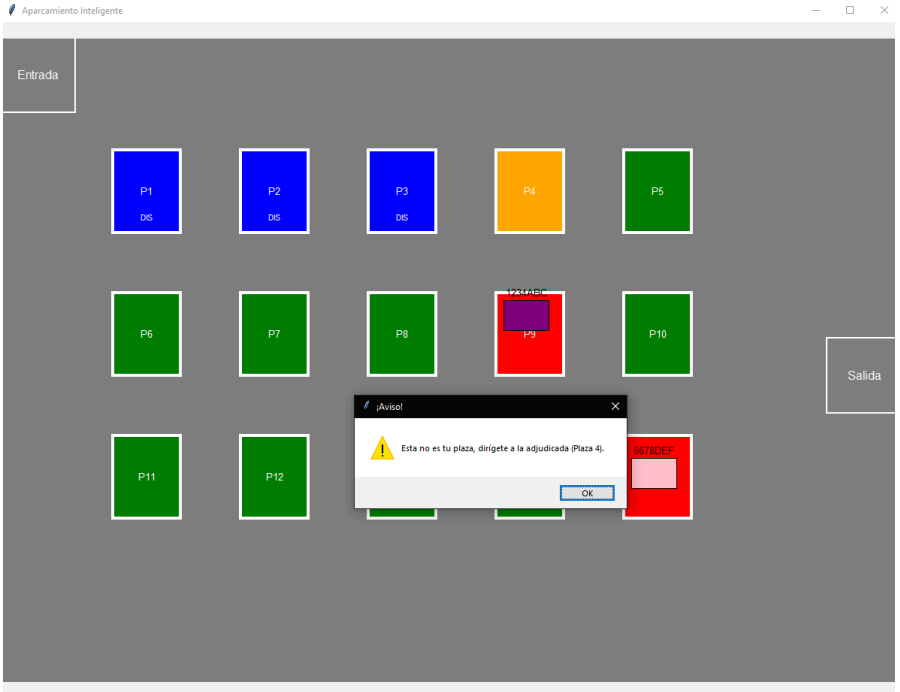
El intento de registro de un usuario con datos con formato incorrecto (control de errores):

En cuanto a la parte de visualización en sí misma, se comprueban los siguientes aspectos:

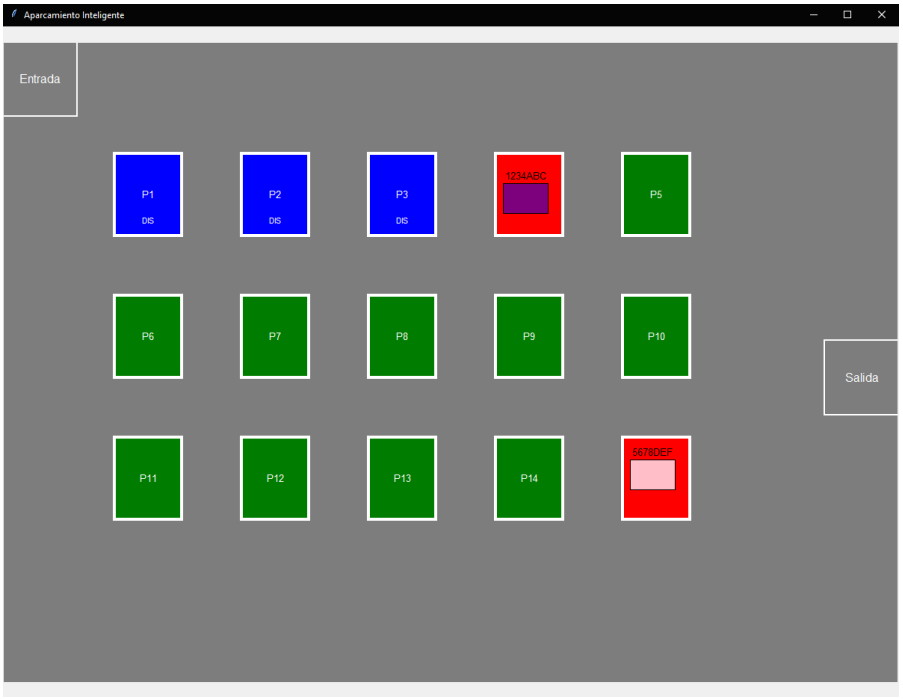
Vehículo entrante con su plaza asignada, otro vehículo marcando una plaza como aparcada:



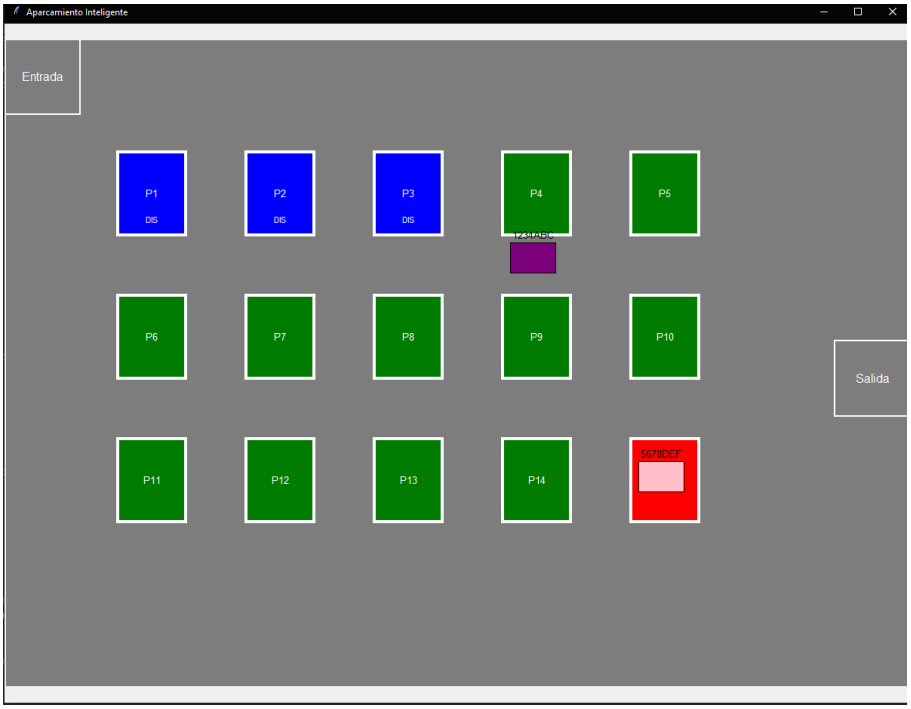
Vehículo entra en una plaza no asignada:



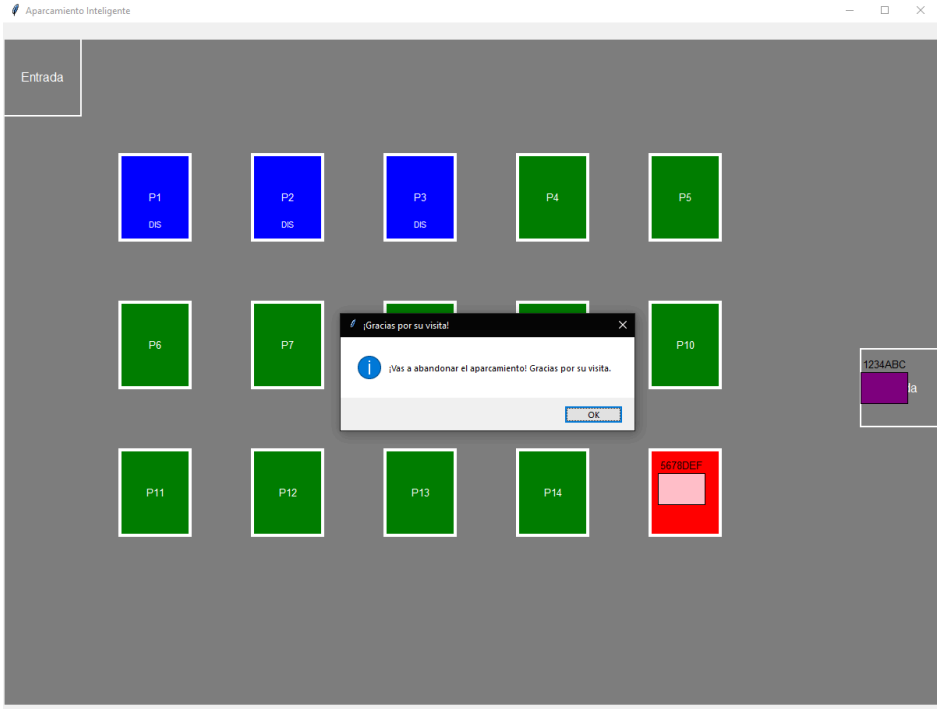
Vehículo entra en su plaza asignada y la marca como ocupada:

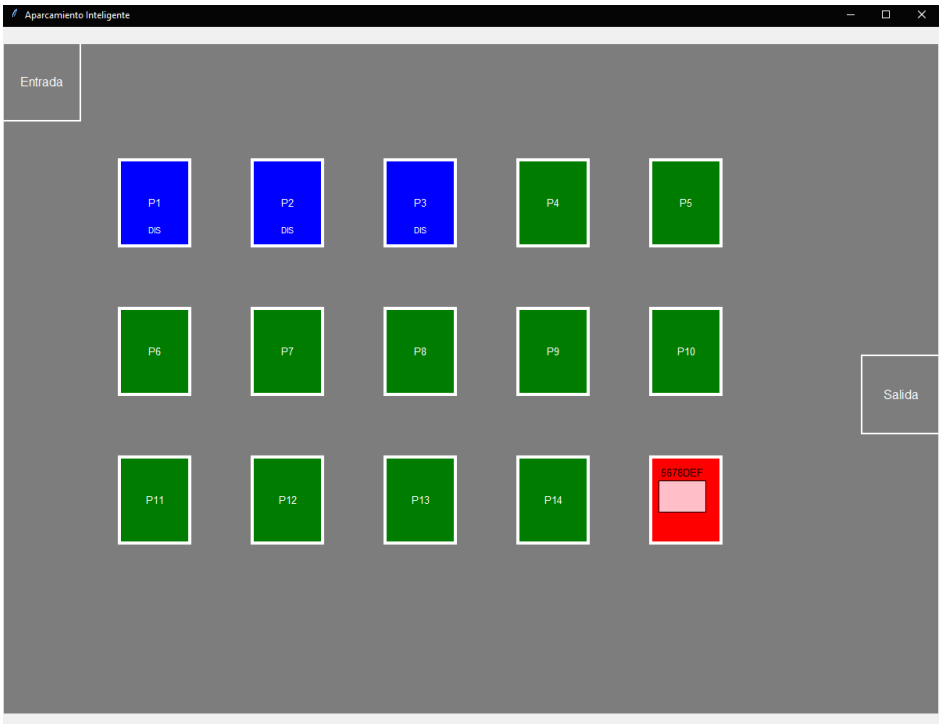


Vehículo sale de la plaza ocupada y esta ya no está asignada (ya no se marca en amarillo):



Vehículo sale del parking (primero salta la notificación y después desaparece el coche):





6. Conclusiones.

El prototipo desarrollado para el sistema de visualización de planta de aparcamiento inteligente representa un avance significativo en la optimización del uso de espacios de estacionamiento. Su diseño enfocado en la interacción en tiempo real y en la mejora de la experiencia del usuario garantiza una solución funcional y versátil para diferentes tipos de aparcamientos.

Entre sus principales ventajas destacan:

- La integración de funcionalidades clave como la asignación dinámica de plazas, la visualización de movimientos de vehículos y el acceso diferenciado para usuarios y administradores.
- La capacidad del sistema para manejar datos en tiempo real, proporcionando una experiencia fluida y eficiente tanto para usuarios como para administradores.
- La modularidad y escalabilidad de la solución, que permite su aplicación en aparcamientos de diferentes tamaños y complejidades.

No obstante, existen áreas que pueden ser perfeccionadas. La interfaz gráfica, aunque funcional, carece de atractivo visual, lo que podría dificultar la aceptación por parte de usuarios finales. Además, la lógica de ciertas funcionalidades, como la validación de matrículas en formatos no estándar, requiere ajustes para garantizar un uso universal.

Propuestas de mejora:

1. **Rediseño de la interfaz gráfica:** Incorporar elementos visuales modernos y más intuitivos para facilitar la interacción del usuario.

2. **Ampliación de la lógica de validación:** Adaptar el sistema a formatos internacionales de matrículas y mejorar la flexibilidad en las restricciones de entrada.

3. **Integración con tecnologías emergentes:** Explorar la incorporación de inteligencia artificial para mejorar la predicción de disponibilidad de plazas y optimizar rutas internas.

4. **Mejora en la accesibilidad:** Diseñar opciones más avanzadas para personas con movilidad reducida, como la reserva automática de plazas preferenciales.

En conclusión, este proyecto demuestra un fuerte potencial para mejorar la gestión de aparcamientos en entornos urbanos y comerciales. Las limitaciones identificadas son menores y solucionables, lo que refuerza la viabilidad de implementar esta solución en escenarios reales. Con las mejoras sugeridas, el sistema podría no solo modernizar la experiencia de estacionamiento, sino también contribuir a la sostenibilidad urbana al reducir el tiempo de búsqueda de plazas y las emisiones asociadas.

7. Anexo: Código del proyecto.

MAIN.PY

```
from src import login
import tkinter as tk


#Initialization of the program, execute this file to run the program

if __name__=="__main__":
    root = tk.Tk()
    app = login.LoginInterface(root)
    login.LoginInterface(root)
    root.mainloop()
```

LOGIN.PY

```
import tkinter as tk
from tkinter import messagebox
import json
from src import visualization
import os

# Ruta dinámica hacia dataBase/users.json
DATABASE_FILE  =  os.path.join(os.path.dirname(os.path.dirname(__file__)),
"dataBase", "users.json")


# Clase Usuario
class Usuario:
    def __init__(self, nombre, email, contraseña, matricula, minusvalia,
clave_usuario):
        self.nombre = nombre
        self.email = email
        self.contraseña = contraseña
        self.matricula = matricula
        self.minusvalia = minusvalia
        self.clave_usuario = clave_usuario


    def to_dict(self):
        """Convierte el objeto Usuario en un diccionario."""
        return {
            "nombre": self.nombre,
```

```
            "email": self.email,
            "contraseña": self.contraseña,
            "matricula": self.matricula,
            "minusvalia": self.minusvalia,
            "clave_usuario": self.clave_usuario,
        }

    @staticmethod
    def from_dict(data):
        """Crea un objeto Usuario a partir de un diccionario."""
        return Usuario(
            nombre=data["nombre"],
            email=data["email"],
            contraseña=data["contraseña"],  # Ya cifrada
            matricula=data["matricula"],
            minusvalia=data["minusvalia"],
            clave_usuario=data["clave_usuario"],
        )


# Clase BaseDeDatos
class BaseDeDatos:
    def __init__(self, archivo):
        self.archivo = archivo
        self.usuarios = self.cargar_usuarios()

    def cargar_usuarios(self):
        """Carga los usuarios desde el archivo JSON."""
        try:
            with open(self.archivo, "r") as f:
                datos = json.load(f)
                return [Usuario.from_dict(u) for u in datos]
        except FileNotFoundError:
            return []
        except json.JSONDecodeError:
            return []

    def guardar_usuarios(self):
        """Guarda los usuarios en el archivo JSON."""
        with open(self.archivo, "w") as f:
            json.dump([u.to_dict() for u in self.usuarios], f, indent=4)

    def agregar_usuario(self, usuario):
```

```
        """Agrega un usuario a la base de datos."""
        self.usuarios.append(usuario)
        self.guardar_usuarios()

    def buscar_usuario(self, email):
        """Busca un usuario por email."""
        for usuario in self.usuarios:
            if usuario.email == email:
                return usuario
        return None


# Clase Autenticación
class Autenticacion:
    def __init__(self, db):
        self.db = db

      def registrar_usuario(self, nombre, email, contraseña, matricula,
minusvalia, clave_usuario):
        """Registra un nuevo usuario en el sistema."""
        if self.db.buscar_usuario(email):
            return False, "El email ya está registrado."

           nuevo_usuario = Usuario(nombre, email, contraseña, matricula,
minusvalia, clave_usuario)
        self.db.agregar_usuario(nuevo_usuario)
        return True, "Usuario registrado exitosamente."

    def iniciar_sesion(self, email, contraseña):
        """Inicia sesión verificando las credenciales."""
        usuario = self.db.buscar_usuario(email)
        if not usuario:
            return False, "Usuario no encontrado."

        if usuario.contraseña == contraseña:
            return True, f"Bienvenido, {usuario.nombre}."
        else:
            return False, "Contraseña incorrecta."


# Interfaz gráfica con Tkinter
class LoginInterface:
    def __init__(self, root):
```

```
        self.root = root
        self.root.title("Sistema de Login")
        self.db = BaseDeDatos(DATABASE_FILE)
        self.auth = Autenticacion(self.db)
        self.mostrar_login()

    def mostrar_login(self):
        """Pantalla de inicio de sesión."""
        for widget in self.root.winfo_children():
            widget.destroy()

            tk.Label(self.root, text="Inicio de Sesión", font=("Arial",
16)).pack(pady=10)
        tk.Label(self.root, text="Email:").pack()
        email_entry = tk.Entry(self.root)
        email_entry.pack()
        tk.Label(self.root, text="Contraseña:").pack()
        password_entry = tk.Entry(self.root, show="*")
        password_entry.pack()

        def iniciar_sesion():
            email = email_entry.get()
            contraseña = password_entry.get()
            exito, mensaje = self.auth.iniciar_sesion(email, contraseña)
            messagebox.showinfo("Inicio de Sesión", mensaje)
            if exito:
                self.show_visualization()

                        tk.Button(self.root,   text="Iniciar   Sesión",
command=iniciar_sesion).pack(pady=10)
                            tk.Button(self.root,   text="Registrarse",
command=self.mostrar_registro).pack()

    def mostrar_registro(self):
        """Pantalla de registro con validaciones."""
        for widget in self.root.winfo_children():
            widget.destroy()

        # Títulos y campos de entrada
          tk.Label(self.root, text="Registro de Usuario", font=("Arial",
16)).pack(pady=10)
        tk.Label(self.root, text="Nombre:").pack()
        nombre_entry = tk.Entry(self.root)
```

```
        nombre_entry.pack()
        tk.Label(self.root, text="Email:").pack()
        email_entry = tk.Entry(self.root)
        email_entry.pack()
        tk.Label(self.root, text="Contraseña:").pack()
        password_entry = tk.Entry(self.root, show="*")
        password_entry.pack()
        tk.Label(self.root, text="Matrícula:").pack()
        matricula_entry = tk.Entry(self.root)
        matricula_entry.pack()

        # Checkbutton para minusvalía
        minusvalia_var = tk.BooleanVar()
        tk.Label(self.root, text="¿Tienes minusvalía?").pack()
        tk.Checkbutton(self.root, variable=minusvalia_var).pack()

        # Campo opcional para clave de usuario
        tk.Label(self.root, text="Clave de Usuario (opcional):").pack()
        clave_usuario_entry = tk.Entry(self.root)
        clave_usuario_entry.pack()

        # Función para validar los datos
        def validar_datos():
            nombre = nombre_entry.get()
            email = email_entry.get()
            contraseña = password_entry.get()
            matricula = matricula_entry.get()
            minusvalia = minusvalia_var.get()
            clave_usuario = clave_usuario_entry.get()

            # Validaciones
            if not nombre.isalpha():
                        messagebox.showerror("Error", "El nombre solo debe
contener letras.")
                return False
            if not ("@" in email and "." in email.split("@")[-1]):
                      messagebox.showerror("Error", "El email no tiene un
formato válido.")
                return False
            if len(contraseña) < 4:
                  messagebox.showerror("Error", "La contraseña debe tener
al menos 4 caracteres.")
                return False
```

```
            if not (len(matricula) == 7 and matricula[:4].isdigit() and
matricula[4:].isalpha()):
                messagebox.showerror("Error", "La matrícula debe ser de 4
números seguidos de 3 letras.")
                return False

            return True

        # Función de registro con validación
        def registrar():
            if validar_datos():
                nombre = nombre_entry.get()
                email = email_entry.get()
                contraseña = password_entry.get()
                matricula = matricula_entry.get()
                minusvalia = minusvalia_var.get()
                clave_usuario = clave_usuario_entry.get()
                exito, mensaje = self.auth.registrar_usuario(
                        nombre, email, contraseña, matricula, minusvalia,
clave_usuario
                )
                messagebox.showinfo("Registro", mensaje)
                if exito:
                    self.mostrar_login()

        # Botones
                                tk.Button(self.root,    text="Registrar",
command=registrar).pack(pady=10)
                                   tk.Button(self.root,    text="Volver",
command=self.mostrar_login).pack()


    def show_visualization(self):
        """Pantalla de bienvenida tras iniciar sesión."""
        for widget in self.root.winfo_children():
            widget.destroy()
        root = tk.Tk()
        visualization.ParkingGUI(root)


# Ejecutar la aplicación
if __name__ == "__main__":
    root = tk.Tk()
```

```
    app = App(root)
    root.mainloop()
```

VISUALIZATION.PY

```
import tkinter as tk
from tkinter import messagebox

# Clases del Sistema
class Plaza:
    def __init__(self, plaza_id, size="medium", is_disabled=False):
        self.plaza_id = plaza_id
        self.size = size
        self.is_disabled = is_disabled
        self.is_occupied = False
        self.vehicle = None

    def occupy(self, vehicle):
        if self.is_occupied:
            raise ValueError(f"Plaza {self.plaza_id} ya está ocupada")
        self.is_occupied = True
        self.vehicle = vehicle

    def vacate(self):
        self.is_occupied = False
        self.vehicle = None


class Vehiculo:
    def __init__(self, plate, size="medium", is_disabled=False):
        self.plate = plate
        self.size = size
        self.is_disabled = is_disabled
        self.car_id = f"Car_{plate}"


class Aparcamiento:
    def __init__(self, num_plazas, num_columnas=6):
        self.num_columnas = num_columnas
        self.plazas = [
              Plaza(f"P{i+1}", size="medium" if i % 3 != 0 else "large",
is_disabled=(i < 3)) for i in range(num_plazas)
        ]
        self.vehicles = []
```

```
# Configuración inicial
aparcamiento = Aparcamiento(15)  # 15 plazas


# Interfaz Gráfica
class ParkingGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Aparcamiento Inteligente")
        self.root.geometry("1200x900")

        # Crear un canvas para el aparcamiento
             self.canvas  =  tk.Canvas(self.root,  width=1200,  height=900,
bg="gray")
        self.canvas.pack(pady=20)

        # Botones
        self.menu_frame = tk.Frame(self.root)
        self.menu_frame.pack()

        # Añadir el botón para ver las plazas
                          tk.Button(self.menu_frame,   text="Ver   Plazas",
command=self.view_plazas).grid(row=0, column=1, padx=10)

        # Dibujar aparcamiento
        self.num_columnas = 5  # Número de columnas en el aparcamiento
        self.draw_parking()

        # Inicializar vehículo (un vehículo al principio)
        self.car = Vehiculo(plate="1234ABC", size="medium")

           # Aquí asignamos las coordenadas de la entrada para el coche
morado
        self.car_x = 50  # Coordenada X de la entrada
        self.car_y = 50  # Coordenada Y de la entrada (ajustable)

        self.create_car()

        # Crear otro coche de color rosa con una ruta predeterminada
        self.rosa_car = Vehiculo(plate="5678DEF", size="medium")
        self.rosa_car_x = 50
        self.rosa_car_y = 480
```

```
        self.create_rosa_car()

        # Definir la ruta predeterminada del coche rosa
        self.rosa_car_route = [(840, 480),  # Posición inicial
                               (840, 480),  # Zona libre horizontal
debajo de las plazas
                               (840, 560)   # Subida directa a P15
                              ]

        self.rosa_car_index = 0

        # Asociar las teclas para mover el coche principal
        self.root.bind("<Left>", self.move_left)
        self.root.bind("<Right>", self.move_right)
        self.root.bind("<Up>", self.move_up)
        self.root.bind("<Down>", self.move_down)

        # Iniciar el movimiento del coche rosa
        self.move_rosa_car()

        # Variable para controlar el parpadeo de la plaza asignada
        self.blinking = False
        self.blinking_plaza = None

        # Asignar plaza al coche morado y comenzar a parpadear
        self.assign_parking_for_purple_car()

    def view_plazas(self):
         # Este es el método que se llama cuando se presiona el botón "Ver
Plazas"
        print("Botón 'Ver Plazas' presionado")

          # Como ejemplo, vamos a cambiar el color de todas las plazas a
amarillo cuando se presiona el botón
        for x1, y1, x2, y2 in self.plaza_coords:
              self.canvas.create_rectangle(x1, y1, x2, y2, fill="yellow",
outline="white", width=4)

    def draw_parking(self):
        x, y = 150, 150
        width_plaza = 90
        height_plaza = 110
```

```
        # Aumentar la separación entre las plazas
        horizontal_gap = 80  # Mayor separación horizontal entre plazas
          vertical_gap = 80  # Mayor separación vertical entre filas de
plazas

        self.plaza_coords = []
        for i, plaza in enumerate(aparcamiento.plazas):
            color = "blue" if plaza.is_disabled else "green"
                  self.plaza_coords.append((x, y, x + width_plaza, y +
height_plaza))
                self.canvas.create_rectangle(x, y, x + width_plaza, y +
height_plaza, fill=color, outline="white", width=4, tags=plaza.plaza_id)
             self.canvas.create_text(x + width_plaza / 2, y + height_plaza
/ 2, text=plaza.plaza_id, font=("Arial", 10), fill="white")
            if plaza.is_disabled:
                      self.canvas.create_text(x + width_plaza / 2, y +
height_plaza - 20, text="DIS", font=("Arial", 8), fill="white")
            x += width_plaza + horizontal_gap
            if (i + 1) % self.num_columnas == 0:
                x = 150  # Volver al margen izquierdo
                   y += height_plaza + vertical_gap  # Salto de fila con
espacio vertical

         # Dibujar la entrada (rectángulo gris claro) en la parte superior
izquierda
              self.canvas.create_rectangle(0, 0, 100, 100, fill="gray",
outline="white", width=2)
           self.canvas.create_text(50, 50, text="Entrada", font=("Arial",
12), fill="white")

         # Dibujar la salida (rectángulo gris oscuro) en el centro de la
parte derecha
        salida_x1 = 1100  # Borde derecho
        salida_y1 = 400   # Centro vertical
        salida_x2 = 1200  # Borde derecho
        salida_y2 = 500   # Tamaño de la salida
            self.canvas.create_rectangle(salida_x1, salida_y1, salida_x2,
salida_y2, fill="gray", outline="white", width=2)
         self.canvas.create_text(salida_x1 + (salida_x2 - salida_x1) / 2,
salida_y1 + (salida_y2 - salida_y1) / 2,
                                          text="Salida", font=("Arial", 12),
fill="white")
```

```
    def create_car(self):
        car_width = 60
        car_height = 40
                self.car_rect = self.canvas.create_rectangle(self.car_x,
self.car_y, self.car_x + car_width, self.car_y + car_height,
fill="purple", outline="black", tags=self.car.car_id)
        self.car_label = self.canvas.create_text(self.car_x + car_width /
2, self.car_y - 10, text=self.car.plate, font=("Arial", 10),
fill="black")

    def create_rosa_car(self):
        car_width = 60
        car_height = 40
                                                self.rosa_car_rect =
self.canvas.create_rectangle(self.rosa_car_x, self.rosa_car_y,
self.rosa_car_x + car_width, self.rosa_car_y + car_height, fill="pink",
outline="black", tags=self.rosa_car.car_id)
        # Crear la matrícula del coche rosa encima del coche
          self.rosa_car_label = self.canvas.create_text(self.rosa_car_x +
car_width / 2, self.rosa_car_y - 10, text=self.rosa_car.plate,
font=("Arial", 10), fill="black")

    def move_left(self, event):
        self.canvas.move(self.car_rect, -10, 0)
        self.canvas.move(self.car_label, -10, 0)
        self.car_x -= 10
        self.check_parking()

    def move_right(self, event):
        self.canvas.move(self.car_rect, 10, 0)
        self.canvas.move(self.car_label, 10, 0)
        self.car_x += 10
        self.check_parking()

    def move_up(self, event):
        self.canvas.move(self.car_rect, 0, -10)
        self.canvas.move(self.car_label, 0, -10)
        self.car_y -= 10
        self.check_parking()

    def move_down(self, event):
        self.canvas.move(self.car_rect, 0, 10)
        self.canvas.move(self.car_label, 0, 10)
```

```
        self.car_y += 10
        self.check_parking()

    def move_rosa_car(self):
        if self.rosa_car_index < len(self.rosa_car_route):
            target_x, target_y = self.rosa_car_route[self.rosa_car_index]

            if self.rosa_car_y < target_y:  # Mover hacia abajo
                self.canvas.move(self.rosa_car_rect, 0, 10)
                self.canvas.move(self.rosa_car_label, 0, 10)
                self.rosa_car_y += 10
            elif self.rosa_car_x < target_x:  # Mover hacia la derecha
                self.canvas.move(self.rosa_car_rect, 10, 0)
                self.canvas.move(self.rosa_car_label, 10, 0)
                self.rosa_car_x += 10
             elif self.rosa_car_x > target_x:  # Mover hacia la izquierda
(si necesario)
                self.canvas.move(self.rosa_car_rect, -10, 0)
                self.canvas.move(self.rosa_car_label, -10, 0)
                self.rosa_car_x -= 10
            elif self.rosa_car_y > target_y:  # Mover hacia arriba
                self.canvas.move(self.rosa_car_rect, 0, -10)
                self.canvas.move(self.rosa_car_label, 0, -10)
                self.rosa_car_y -= 10

            # Actualizar el índice si se alcanza el objetivo
                  if self.rosa_car_x == target_x and self.rosa_car_y ==
target_y:
                self.rosa_car_index += 1

            # Continuar moviendo cada 100 ms
            self.root.after(100, self.move_rosa_car)
        else:
            # Al llegar al destino, ocupar la plaza
            plaza = aparcamiento.plazas[14]  # Plaza 15 es el índice 14
            if not plaza.is_occupied:
                plaza.occupy(self.rosa_car)

self.canvas.itemconfig(self.canvas.find_withtag(plaza.plaza_id),
fill="red")

            # Asignar plaza al coche morado
            self.assign_parking_for_purple_car()
```

```
    def check_parking(self):
          if self.car_x >= 1100 and self.car_x <= 1200 and self.car_y >=
400 and self.car_y <= 500:
                # Mostrar el mensaje cuando el coche esté en la zona de
salida
                  messagebox.showinfo("¡Gracias por su visita!", "¡Vas a
abandonar el aparcamiento! Gracias por su visita.")

            # Eliminar el coche completamente de la pantalla
            self.canvas.delete(self.car_rect)
            self.canvas.delete(self.car_label)
            self.car_rect = None
            self.car_label = None

        else:
            parked = False
            for i, (x1, y1, x2, y2) in enumerate(self.plaza_coords):
                if x1 < self.car_x < x2 and y1 < self.car_y < y2:
                    plaza = aparcamiento.plazas[i]
                    if plaza.is_occupied:
                            # Si la plaza está ocupada por otro vehículo,
mostrar el aviso
                           if plaza.vehicle != self.car:  # Si el vehículo
en la plaza no es el actual
                                  messagebox.showwarning("AVISO!", f"Plaza
{plaza.plaza_id} ocupada por otro vehículo. Elija otra plaza.")
                           else:
                                # Si el vehículo en la plaza es el mismo, no
mostrar el aviso
                                parked = True
                    else:
                        # Si la plaza no está ocupada, aparcar el coche
                        plaza.occupy(self.car)

self.canvas.itemconfig(self.canvas.find_withtag(plaza.plaza_id),
fill="red")
                        parked = True
                    break

            if not parked:
                for plaza in aparcamiento.plazas:
                    if plaza.is_occupied and plaza.vehicle == self.car:
```

```
                        plaza.vacate()
                        color = "blue" if plaza.is_disabled else "green"

self.canvas.itemconfig(self.canvas.find_withtag(plaza.plaza_id),
fill=color)

    def assign_parking_for_purple_car(self):
    # Asignar la plaza 4 como objetivo para el coche morado
     self.blinking_plaza = aparcamiento.plazas[3]  # Plaza 4 (índice 3)
     self.blinking = True
        self.plaza_4_occupied_once = False  # Indicar que aún no se ha
ocupado la plaza 4
     self.blink_plaza()  # Iniciar el parpadeo de la plaza

    def blink_plaza(self):
    # Asegurarse de que el parpadeo de la plaza 4 solo se detiene cuando
se ocupa o cuando el coche se va
     if self.blinking and self.blinking_plaza:
                                                    plaza_rect      =
self.canvas.find_withtag(self.blinking_plaza.plaza_id)
        current_color = self.canvas.itemcget(plaza_rect, "fill")

          # Si la plaza está ocupada por algún coche (incluido el coche
morado), detener el parpadeo y ponerla en rojo
                        if  self.blinking_plaza.is_occupied  and  not
self.plaza_4_occupied_once:
            self.canvas.itemconfig(plaza_rect, fill="red")
            self.blinking = False  # Detener el parpadeo
             self.plaza_4_occupied_once = True  # Marcar que la plaza fue
ocupada al menos una vez
        elif not self.blinking_plaza.is_occupied:
            # Si la plaza no está ocupada, alternar entre naranja ámbar
                new_color = "orange" if current_color != "orange" else
"green"
            self.canvas.itemconfig(plaza_rect, fill=new_color)
                 self.root.after(500, self.blink_plaza)  # Continuar el
parpadeo cada 500 ms

    def check_parking(self):
    # Comprobar si el coche morado está en la zona de salida
     if 1100 <= self.car_x <= 1200 and 400 <= self.car_y <= 500:
         messagebox.showinfo("¡Gracias por su visita!", "¡Vas a abandonar
el aparcamiento! Gracias por su visita.")
```

```
        # Eliminar el coche morado completamente de la pantalla
        self.canvas.delete(self.car_rect)
        self.canvas.delete(self.car_label)
        self.car_rect = None
        self.car_label = None

        # Si la plaza 4 estaba ocupada por el coche morado, desocuparla
        for plaza in aparcamiento.plazas:
            if plaza.is_occupied and plaza.vehicle == self.car:
                plaza.vacate()
                color = "blue" if plaza.is_disabled else "green"

self.canvas.itemconfig(self.canvas.find_withtag(plaza.plaza_id),
fill=color)

        # Detener el parpadeo independientemente del estado
        self.blinking = False

        return  # Evitar que siga ejecutando el código de aparcar

    parked = False
    for i, (x1, y1, x2, y2) in enumerate(self.plaza_coords):
        if x1 < self.car_x < x2 and y1 < self.car_y < y2:
            plaza = aparcamiento.plazas[i]
            if plaza.is_occupied:
                 # Si la plaza está ocupada por otro vehículo, mostrar el
aviso
                  if plaza.vehicle != self.car:  # Si el vehículo en la
plaza no es el coche morado
                                  messagebox.showwarning("AVISO!", f"Plaza
{plaza.plaza_id} ocupada por otro vehículo. Elija otra plaza.")
                else:
                      # Si el vehículo en la plaza es el coche morado, no
mostrar el aviso
                    parked = True
            else:
                # Si la plaza no está ocupada, aparcar el coche
                plaza.occupy(self.car)

self.canvas.itemconfig(self.canvas.find_withtag(plaza.plaza_id),
fill="red")
                parked = True
```

```
                    # Si aparca en una plaza distinta de la 4, detener el
parpadeo y mostrar el mensaje
                if plaza != self.blinking_plaza:
                        self.blinking = False  # Detener el parpadeo de la
plaza 4
                            messagebox.showwarning("¡Aviso!", "Esta no es tu
plaza, dirígete a la adjudicada (Plaza 4).")

                        # Si aparca en la plaza 4, marcarla como ocupada
definitivamente
                elif plaza == self.blinking_plaza:
                    self.blinking = False
                    self.plaza_4_occupied_once = True

            break

    # Si el coche morado no ha aparcado en ninguna plaza
    if not parked:
       for plaza in aparcamiento.plazas:
            if plaza.is_occupied and plaza.vehicle == self.car:
                plaza.vacate()
                color = "blue" if plaza.is_disabled else "green"

self.canvas.itemconfig(self.canvas.find_withtag(plaza.plaza_id),
fill=color)

     # Si la plaza 4 se ha desocupado, reiniciar el parpadeo si no ha
sido ocupada
     if self.blinking_plaza and not self.blinking_plaza.is_occupied and
not self.plaza_4_occupied_once:
        self.blinking = True
        self.blink_plaza()

if __name__=="__main__":
    # Iniciar la aplicación
    root = tk.Tk()
    app = ParkingGUI(root)
    root.mainloop()
```