UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Graduado en Ingeniería del Software

Gestión controlada de estaciones de carga para vehículos eléctricos dentro de un *Smart Campus* – *ControlCS-SC*

Controlled management of charging stations for electric vehicles within an *Smart Campus* – *ControlCS-SC*

Realizado por
José Luis Bueno Pachón

Tutorizado por
María Cristina Alcaraz Tello

Departamento
Lenguajes y Ciencias de la Computación
UNIVERSIDAD DE MÁLAGA

MÁLAGA, MAYO DE 2023

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUDADO EN INGENIERÍA DEL SOFTWARE

Gestión controlada de estaciones de carga para vehículos eléctricos dentro de un *Smart Campus – ControlCS-SC*

Controlled management of charging stations for electric vehicles within an *Smart Campus – ControlCS-SC*

Realizado por
José Luis Bueno Pachón

Tutorizado por
María Cristina Alcaraz Tello

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, MAYO DE 2023

Fecha defensa: junio de 2023

Gracias a mi familia y amigos, por alentarme y apoyarme en los momentos más difíciles de este trayecto académico.

Gracias a mi tutora, Cristina, por mostrarme una de los campos más apasionantes de esta profesión, guiarme y comprenderme durante el desarrollo de este trabajo.

Constancia, trabajo y paciencia, como bien decías, abuelo, va por ti.

Abstract

The development of web applications has become a fundamental tool to facilitate access to different services and products, and on this occasion, the objective of this thesis is the creation of a web application called *ControlCS-SC*, which will allow users of an Smart Campus to manage the reservation, use and management of electric chargers available on a campus through the *(Open Charge Point Protocol) OCPP* protocol.

This tool, which has two different interfaces, is designed both for the general public and for the exclusive administration and management of the system. In the case of users, after registering in the application, they will be able to use the electric chargers available in their area upon reservation, with a maximum of two hours per reservation. In addition, each user will be able to keep a detailed record of their reservations and generate incidents in case of any problem before, during or after the use of the application or an electric charger.

As for the interface for the administration and management of the system, which we will call *AdminCS-SC*, it will allow the designated administrators to keep a detailed record of all reservations, chargers, records and statistics of use and cost of the system, which will allow a comprehensive control of the operation of the system.

In short, *ControlCS-SC* and *AdminCS-SC* are complete and versatile tools that will allow an efficient and simple management of the electric chargers available in an Smart Campus, thus improving the quality of service and contributing to the transition towards a more sustainable mobility.

Keywords: Smart Campus, Electric Vehicles Charging Infrastructures, OCPP, Control, Bookings Controlled Management

Resumen

El desarrollo de aplicaciones web se ha convertido en una herramienta fundamental para facilitar el acceso a distintos servicios y productos, y en esta ocasión, el objetivo del presente trabajo de fin de grado es la creación de una aplicación web denominada *ControlCS-SC*, que permitirá a los usuarios de un *Smart Campus* gestionar la reserva, uso y gestión de los cargadores eléctricos disponibles en dicho campus, a través del protocolo *OCPP*.

Esta herramienta, que cuenta con dos interfaces diferenciadas, está diseñada tanto para el público general como para la administración y gestión exclusiva del sistema. En el caso de los usuarios, tras el registro en la aplicación, podrán hacer uso de los cargadores eléctricos que haya disponibles en su zona previa reserva, con un máximo de dos horas por reserva. Además, cada usuario tendrá la posibilidad de llevar un registro detallado de sus reservas realizadas y generar incidencias en caso de que ocurra algún problema antes, durante o tras el uso de la aplicación o un cargador eléctrico.

En cuanto a la interfaz destinada a la administración y gestión del sistema, a la que llamaremos *AdminCS-SC*, permitirá a los administradores designados llevar un registro detallado de todas las reservas, cargadores, registros y estadísticas de uso y coste del sistema, lo que permitirá llevar un control exhaustivo del funcionamiento del sistema.

En definitiva, *ControlCS-SC* y *AdminCS-SC* son herramientas completas y versátiles que permitirán una gestión eficiente y sencilla de los cargadores eléctricos disponibles en un *Smart Campus*, mejorando así la calidad del servicio y contribuyendo a la transición hacia una movilidad más sostenible.

Palabras clave: Smart Campus, Infraestructuras de Carga de Vehículos Eléctricos, Control, OCPP, Gestión Controlada de Reservas

Índice

1

Acrónimos

2
Introducción

2.1. Motivación

Este trabajo surge a partir de la continua evolución en cuanto a tecnologías de los vehículos eléctricos. Cada vez más personas son propietarias de uno de ellos y nos vemos en la necesidad de aumentar la cantidad de estaciones de carga, especialmente dentro de los *Smart* Campus Universitarios como la de la Universidad de Málaga. Así bien se puede observar que universidades en España se ha comenzado a implantar este tipo de tecnologías, como es el caso de la Universidad de Cádiz, en la cual se implantó la primera estación de carga el pasado año [1], esperando así llegar a aumentar la cantidad hasta 25 puestos dobles, los cuales darán servicio a dos vehículos por estación.

No sólo son las universidades aquellos emplazamientos donde se están llevando a cabo este tipo de proyectos, sino también en ciudades como Málaga, en la cual la empresa *Endesa* ya ha habilitado un total de 22 estaciones de recarga, abasteciendo de energía 100 % verde a los vehículos, ya que aseguran que la energía utilizada en las estaciones procede de fuentes de generación renovable [2]. Numerosas ciudades a lo largo de nuestro país se están poniendo manos a la obra en el desarrollo de estaciones de carga, para así poder permitir viajes de larga distancia con vehículos 100 % eléctricos, así lo prevé la empresa *Iberdrola* con su plan *Smart Mobility*, la cual pretende instalar por toda la península Ibérica 2.500 estaciones de carga, garantizando así una estación de carga cada 50 kilómetros [3].

Es una obviedad que los vehículos eléctricos son el futuro de la movilidad: tanto personal como masiva, por ello, estudiando el número de estaciones de recarga en España, se puede observar que han crecido prácticamente de manera exponencial durante los últimos años [4], adaptándose así a uno de los objetivos de la *Agenda 2030* realizada por la *Unión Europea*, ya que según estudios las reservas totales de petróleo y gas comenzarán a disminuir de manera

muy acusada a partir del año 2025 [5], impulsando así aún más a las nuevas tecnologías de movilidad eléctrica a su despliegue a todo tipo de público.

Por tanto, es necesaria una aplicación web – denotada aquí como *ControlCS-SC* por el título en inglés *"CONTROLled management of Charging Stations for electric vehicles within a Smart Campus"* - que ayude a la gestión coherente y equitativa de las estaciones y sus recursos, facilitando a los usuarios una forma atractiva de realizar reservas a través de la web, registrar sus peticiones de forma segura, facilitando que dichas peticiones se realicen en base a aquellas estaciones disponibles y cerca de sus localizaciones. En este caso, la web está diseñada para soportar el protocolo *Open Charge Point Protocol (OCPP)*, cuya implementación permite la comunicación entre las estaciones de carga de vehículos eléctricos y un sistema central de gestión, también conocido como red de estaciones de carga, similar al sistema que conforman los teléfonos móviles y las redes de teléfonos móviles" [16], para tramitar todas las peticiones y gestión de reserva, y bajo principios de autenticación, confidencialidad a nivel de comunicaciones y registro de peticiones en la web, e integridad frente a posibles modificaciones locales de dichas peticiones.

2.2. Objetivos

Este trabajo de fin de grado consiste en el desarrollo y puesta en línea de la aplicación web *ControlCS-SC* y su análoga de administración, *AdminCS-SC*, facilitando a los usuarios una forma atractiva de realizar reservas a través de la web, registrar sus peticiones de forma segura, facilitando que dichas peticiones se realicen en base a aquellas estaciones disponibles y cerca de sus localizaciones. Se pretenden tener en cuenta las siguientes consideraciones tales como:

- **Servicio de autenticación y autorización**: los usuarios deben estar autenticados, a la vez que necesitarán un *token* que señale su validez para tramitar el proceso y estar autorizados al uso de estaciones de carga en un entorno concreto, restringiendo así su uso sólo a los usuarios que realmente los vayan a utilizar.

- **Tiempo máximo de uso**: se limitará el uso de las estaciones por usuario a un máximo de minutos al día, para así permitir la carga de vehículos eléctricos al máximo de usuarios

posibles. Por otro lado, si un usuario comienza a cargar su vehículo y decide terminar su carga antes de llegar a sus minutos asignados, perderá el tiempo restante.

- **Reserva de estaciones de carga**: otra idea que se busca plasmar en el resultado final es la opción de reservar una estación a una hora determinada siempre que se cumplan unas restricciones horarias marcadas.

- **Localización de estaciones de carga**: la aplicación web busca tener en cuenta el tiempo de desplazamiento del usuario a la estación y la disponibilidad de éste en un rango corto de tiempo para así facilitar su uso. Para ello, se hará suso de mapas y de alguna *Application Programming Interface (API)* que nos permita embeber un mapa dentro de la aplicación web, facilitando así al usuario poder tener un resumen visual de aquellas estaciones que se encuentren a su alrededor.

- **Sistema gestor de incidencias**: estará disponible para todo usuario que tenga algún problema con la reserva realizada o con la estación sugerida, enviando dicho reporte al operario de mantenimiento correspondiente y responsable por área o zona.

- **Servicio de privacidad de los datos**, a nivel de comunicaciones considerando la última versión de *Transport Layer Security (TLS)*.

- **Gestión de las reservas** por parte de la administración, permitiendo así un control exhaustivo de todas las transacciones llevadas a cabo por los usuarios.

- **Gestión de los cargadores** por parte de la administración, permitiendo crear, modificar y eliminar cargadores disponibles para los usuarios dentro de la aplicación web.

- **Estadísticas de uso y coste del sistema** en la web de administración, la cual monitoriza todas las reservas realizadas y efectivas de los usuarios y muestra datos relativos a las horas totales de uso de los cargadores, precio de la luz máximo registrado durante las cargas, precio medio y total de las cargas, etc.

2.3. Estructura del documento

El presente documento aborda de manera exhaustiva diversos aspectos concernientes al estudio, diseño, desarrollo y pruebas del presente trabajo de fin de grado, siguiendo el ciclo

natural de ingeniería del software.

En primer lugar, en el capítulo 3, se realiza una contextualización para el lector acerca del estado actual de las infraestructuras relacionadas con el uso de estaciones de carga de vehículos eléctricos y los vehículos eléctricos en sí mismos. Asimismo, se profundiza en aspectos técnicos referentes al protocolo de comunicación *OCPP*, así como otros protocolos de comunicación de relevancia.

Seguidamente, en el capítulo 4, se aborda en detalle todo lo relativo a la ingeniería de software de las aplicaciones web, haciendo hincapié en los requisitos funcionales y no funcionales, y describiendo el diseño y modelado de sus arquitecturas e interfaces.

Asimismo, en el capítulo 5, se examinan minuciosamente todas las tecnologías utilizadas para el desarrollo de software de las aplicaciones web, desde los contenedores que albergan la infraestructura general, la base de datos, hasta los servidores que gestionan la lógica y las interfaces de usuario de las aplicaciones web.

Además, en el capítulo 6, se llevan a cabo rigurosas validaciones de los requisitos previamente mencionados para comprobar la correcta implementación de las aplicaciones, con el fin de concluir el presente documento y vislumbrar posibles líneas futuras de estudio relacionadas con el desarrollo de estas aplicaciones.

Finalmente, en el capítulo 7, se realiza una conclusión sobre el presente trabajo de fin de grado y se proponen líneas de investigación y desarrollo futuras basadas en el trabajo realizado durante este proyecto.

2.4. Metodología

Para garantizar una correcta ejecución del proyecto, se ha seguido una metodología tradicional basada en el ciclo de vida del software, la cual ha permitido estructurar el proceso de desarrollo en diferentes etapas, desde el análisis inicial hasta la validación final. Durante todo este proceso se ha seguido un enfoque iterativo, lo que ha permitido la revisión y el refinamiento progresivo de la aplicación de forma constante.

En aras de una planificación más exhaustiva y eficiente, se ha empleado un enfoque metodológico riguroso en forma de un diagrama *Gantt* para delinear y estructurar adecuadamente la ejecución integral de este proyecto, véase figura 1. Este instrumento, de reconocida utilidad y amplia aplicación en la gestión de proyectos, ha permitido visualizar de manera clara y con-

cisa todas las tareas, sus respectivas dependencias y plazos, así como el flujo temporal de las actividades. Mediante esta herramienta, se ha logrado una programación detallada y secuencial de las actividades clave, fomentando la asignación adecuada de recursos y facilitando la identificación de posibles solapamientos o cuellos de botella.

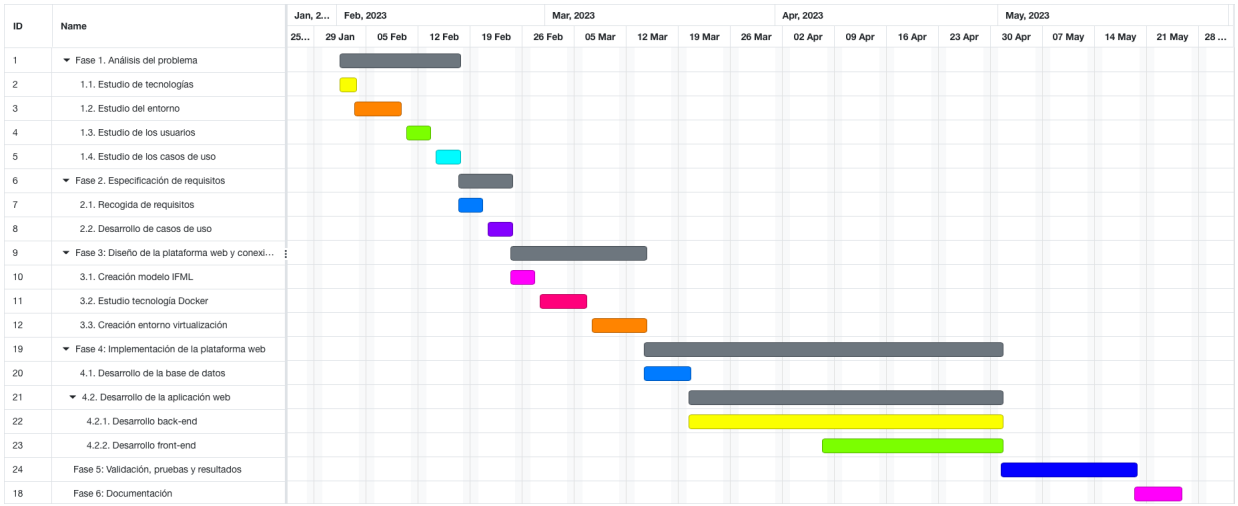

Figura 1: Diagrama *Gantt* de planificación de las tareas del presente proyecto

En el inicio del proyecto se llevó a cabo un análisis detallado del problema a resolver, con el fin de establecer los objetivos y las metas a alcanzar. Posteriormente, se procedió a la recopilación de requisitos y diseño, lo que permitió definir la arquitectura y los componentes necesarios para el correcto funcionamiento de la aplicación.

En el proceso de desarrollo se ha llevado a cabo una serie de iteraciones, cada una con su correspondiente revisión y refinamiento de la aplicación. Es decir, se ha ido trabajando en diferentes versiones de la aplicación, cada vez más completas y mejoradas. Este enfoque ha permitido detectar y solucionar problemas en fases tempranas.

Con el fin de garantizar el éxito del proyecto, se han llevado a cabo numerosas reuniones virtuales con la tutora, las cuales se han realizado de forma casi semanal. Estas reuniones han sido fundamentales para analizar los problemas que han surgido durante el proceso de desarrollo y establecer las soluciones más adecuadas. Además, se han revisado los avances y se ha definido el plan de trabajo para la siguiente fase.

3

Estado del arte

3.1. Situación actual de los vehículos eléctricos e infraestructuras de carga

De manera simplificada, podría afirmarse que la dinamización actual de la movilidad eléctrica se debe a dos factores: por un lado, las restricciones de circulación en las grandes ciudades y, por otro, la percepción social del conductor hacia el vehículo eléctrico.

En primer lugar, es importante destacar que desde mediados del siglo pasado y a nivel global, se ha producido una importante migración de la población de las zonas rurales hacia las ciudades, lo que ha generado una mayor demanda de infraestructuras para el transporte privado y público. Sin embargo, en muchos países, la capacidad de las infraestructuras para el transporte privado ha llegado a un punto crítico, lo que se ha reflejado en índices preocupantes de congestión y contaminación.

Es por ello que las restricciones de circulación en las grandes ciudades han adquirido un papel fundamental en la dinamización de la movilidad eléctrica. Las regulaciones impuestas por las autoridades de tráfico en las zonas urbanas para reducir la circulación de vehículos contaminantes han incentivado la utilización de vehículos eléctricos, contribuyendo así a reducir la contaminación del aire y mejorar la calidad de vida de los ciudadanos.

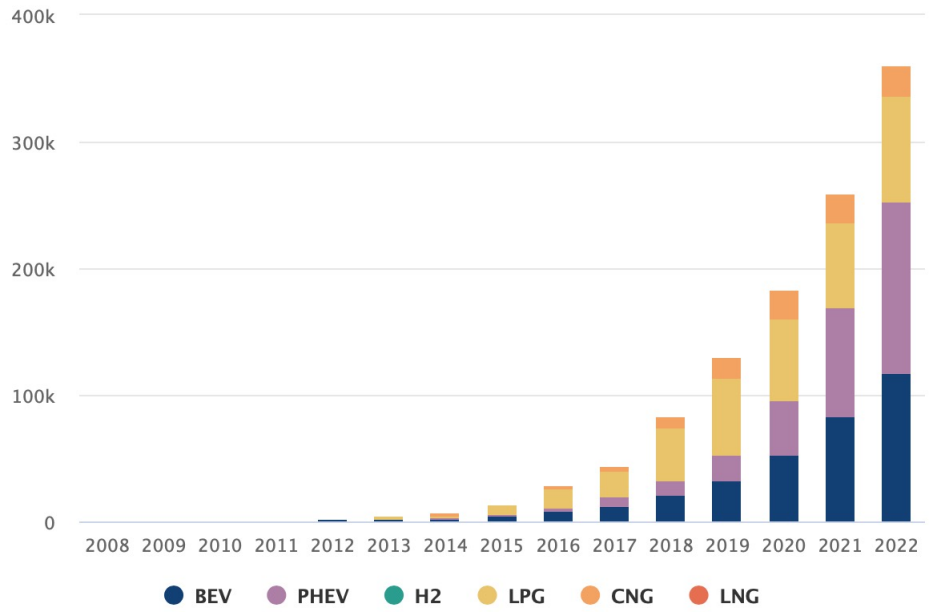

Figura 2: Número total de vehículos matriculados con combustibles alternativos en España desde 2008 a 2022, (fuente: [9])

Por otro lado, la percepción social del conductor hacia el vehículo eléctrico también ha contribuido a su dinamización. En los últimos años, la preocupación por el medio ambiente y la sostenibilidad ha aumentado en gran medida, lo que ha generado una mayor concienciación en la sociedad sobre la necesidad de adoptar un estilo de vida más respetuoso con el planeta. En este contexto, los vehículos eléctricos se han presentado como una alternativa más ecológica y sostenible a los vehículos de combustión interna, lo que ha incentivado su utilización, reflejándose así su popularización en la figura 2 [9].

Cabe en el contexto de este trabajo explicar detalladamente cuáles son los tipos de vehículos eléctricos que existen hoy en día en el mercado, dando lugar a seis modalidades distintas [17]:

- **Vehículos eléctricos de baterías *(Battery Electric Vehicle (BEV))***: un vehículo eléctrico puro es un tipo de vehículo que emplea únicamente la energía almacenada en sus baterías para su movimiento. Para cargar dichas baterías, es necesario conectar el vehículo a un cargador, y la velocidad de carga dependerá de la potencia máxima que pueda soportar el cargador utilizado.

  Estos vehículos utilizan técnicas de frenada regenerativa para maximizar su autonomía

durante las deceleraciones. Según la investigación de la Universidad de Cambridge, los vehículos eléctricos puros son más eficientes en su funcionamiento que otros tipos de vehículos. [23]

Dentro de esta categoría de vehículos eléctricos, se encuentran modelos como el *Opel Corsa*, el *Renault ZOE* y el *Hyundai Ioniq*. Estos vehículos no emiten contaminantes durante su uso, lo que les ha otorgado la etiqueta medioambiental *CERO emisiones* de la *Dirección General de Tráfico (DGT)*. Sin embargo, es importante mencionar que el proceso de fabricación de estos vehículos tiene un costo ambiental más elevado que el de un vehículo equivalente de combustión.

Figura 3: Distintivo medioambiental *CERO emisiones* de la *DGT*, *(fuente: [27])*

Con respecto al distintivo medioambiental cero de la *DGT* (véase figura 3), tendrán derecho a la etiqueta aquellos vehículos que se clasifican como eléctricos de batería (*BEV*), eléctricos de autonomía extendida (*REEV*), eléctricos híbridos enchufables (*Plug-in Hybrid Electric Vehicles (PHEV)*) con una autonomía mínima de 40 kilómetros o vehículos de pila de combustible. [18]

- **Vehículos eléctricos con pila de hidrógeno *(Fuel Cell Electric Vehicle (FCEV))***: los vehículos de hidrógeno no utilizan baterías, por lo que utilizan un sistema diferente al de los vehículos eléctricos puros. Los vehículos con batería de hidrógeno, en particular,

utilizan tecnología de electrólisis de hidrógeno para generar electricidad, por lo que solo emiten agua como residuo.

Es importante tener en cuenta que, aunque el hidrógeno es el elemento químico más abundante en la tierra, no existe en estado puro y, por lo tanto, su producción es costosa desde el punto de vista energético. Pero grandes empresas como *Toyota*, *BMW* y *Hyundai* también confían en este tipo de vehículos como parte de su visión de futuro de la movilidad sostenible.

Actualmente, desafíos importantes para este tipo de vehículos, ya que su coste de fabricación es muy elevado, pero ya existen modelos tales como el *Toyota Mirai* o el *Hyundai Nexo*. También gozan de la etiqueta mediambiental *CERO emisiones* otorgada por la *DGT*.

- **Vehículo eléctrico de autonomía extendida *(Extended Range Electric Vehicle (EREV))***: estos son vehículos que utilizan uno o más motores eléctricos para mover las ruedas utilizando la energía almacenada en baterías, pero también cuentan con un motor térmico que actúa como generador.

  Un requisito previo para que estos vehículos eléctricos de largo alcance no se clasifiquen como híbridos es que el motor de combustión interna carezca de la capacidad de impulsar las ruedas y pueda funcionar como vehículos eléctricos puros en la mayoría de los casos. Su batería se carga de la misma forma que un vehículo eléctrico convencional y la autonomía del motor de combustión interna no debe ser mayor que la del motor eléctrico.

  Un vehículo merece la etiqueta *CERO emisiones* de la *(DGT)* si el motor eléctrico tiene una autonomía de 40 kilómetros o más sin el apoyo de un motor de combustión interna. Los vehículos de este tipo incluyen al *Opel Ampera*, el *BMW i3 Range Extender* y las próximas variantes del *Nissan Qashqai*.

- **Vehículo híbrido enchufable *(PHEV)***: poseen un sistema de propulsión que combina un motor de combustión interna con uno o más motores eléctricos que pueden operar de forma independiente o conjunta. De manera similar, los motores eléctricos se pueden colocar en varios lugares del vehículo, como ruedas, ejes y transmisiones.

  La capacidad de las baterías de estos motores eléctricos no es muy elevada, pero si estos

vehículos tienen una autonomía superior a los 40 kilómetros, cumplen los requisitos para recibir la etiqueta *CERO emisiones* de la *DGT*. De lo contrario, se les etiquetará con *ECO*. La batería se carga mediante la conexión a la red eléctrica. Los conductores tienen la oportunidad de elegir el modo de conducción que más les convenga, pudiendo, incluso combinar ambos motores para una mayor autonomía. Algunos modelos de vehículos pueden utilizar el motor de combustión interna para cargar la batería del motor eléctrico. Los vehículos que usan esta tecnología incluyen al *Renault Captur E-Tech*, el *MG EHS* o el *SEAT León e-HYBRID*.

Figura 4: Distintivo medioambiental *ECO* de la *DGT*, (fuente: [27])

Con respecto al distintivo medioambiental *ECO* de la *DGT*, (véase figura 4) la obtienen vehículos con tecnología híbrida, ya sea mediante la combinación de motores eléctricos y de combustión interna, o mediante la utilización de gas. Aquellos vehículos que pueden conectarse a la red eléctrica y poseen una autonomía inferior a los 40 kilómetros, así como los híbridos no enchufables *(Hybrid Electric Vehicle (HEV))*, los vehículos propulsados por gas natural y *(Gas Natural Comprimido (GNC) y Gas Natural Licuado (GNL))* o *(Gas Licuado de Petróleo (GLP))* también podrán obtener la etiqueta correspondiente. Para ello, deben cumplir los criterios establecidos para la etiqueta *C* (véase figura 5) por las autoridades competentes [18], siendo la etiqueta *C* aquella que obtienen los turismos y furgonetas ligeras de gasolina matriculadas a partir de enero de 2006 y diésel a partir de septiembre de 2015. También, se incluyen los vehículos de más de 8 plazas, excluido

el conductor, y los vehículos pesados tanto de gasolina como diésel, matriculados desde 2014.

Figura 5: Distintivo medioambiental *C* de la *DGT*, (fuente: [27])

- **Vehículo híbrido *(HEV)***: los híbridos fueron los primeros vehículos eléctricos vendidos en masa, principalmente después de que se introdujera el *Toyota Prius* en 1997. Su diseño es más sencillo que el de los vehículos híbridos enchufables *(PHEV)*. Un tren motriz que asiste al motor de combustión interna y reduce su rendimiento para reducir el consumo de combustible.

  Un vehículo híbrido puede conducirse en modo 100 % eléctrico (a diferencia de un *mild hybric* o microhíbrido, explicado más adelante), pero tiene una autonomía limitada debido a la baja capacidad de su batería. En estos casos también se utiliza para la carga la frenada regenerativa y el propio motor térmico.

  Debido a que sus motores de combustión interna funcionan la mayor parte del tiempo y su autonomía eléctrica es inferior a 40 km, estos vehículos cuentan con el distintivo medioambiental *ECO* de la *DGT*. Ejemplos de autos híbridos son el *Honda HR-V* o el *Hyundai Kona*.

- **vehículos microhíbridos *(Mild Hybrid Electric Vehicle (MHEV))***: los automóviles *mild hybrid*, también conocidos como microhíbridos o híbridos de 48 voltios, son automóviles térmicos con la adición de un pequeño sistema de asistencia eléctrica. Estos

vehículos utilizan un motor de arranque/generador para recuperar la energía almacenada en una pequeña batería de 48 V. Esta energía se utiliza para apoyar al motor de combustión durante las fases de aceleración o para suplir algún consumo de energía del sistema. Esto libera el motor de combustión e incluso puede detenerlo prematuramente por inercia. Pero un *mild hybrid* no puede funcionar al 100 % con electricidad. Los *mild hybrid* reciben etiquetas *ECO* de *DGT*. Aunque la electrificación de estos vehículos es mínima, su prevalencia ha dado lugar a las siguientes contradicciones: vehículos de alto rendimiento con etiqueta ecológica como el *Audi RS 6 Avant* y un enorme motor *V8 biturbo* de 600 CV.

Las actuales situaciones de congestión y contaminación han impulsado a los ayuntamientos a establecer medidas de control de tráfico rodado de combustibles fósiles en muchas ciudades del mundo. Estas medidas incluyen restricciones al tráfico, establecimiento de zonas de aparcamiento regulado o restringido y creación de peajes por congestión, tal como ocurre en ciudades como Londres o Estocolmo. Por otro lado, la percepción social del conductor hacia el vehículo eléctrico se ha convertido en otro elemento dinamizador de la movilidad eléctrica. La estrategia de *Tesla* ha revitalizado claramente la percepción del vehículo eléctrico, no sólo desde el punto de vista tecnológico, sino también en la imagen actual de los propietarios de vehículos eléctricos. Actualmente, la imagen de los propietarios de vehículos eléctricos está asociada a personas jóvenes, con recursos económicos, amantes de la tecnología y con una clara responsabilidad medioambiental. [10]

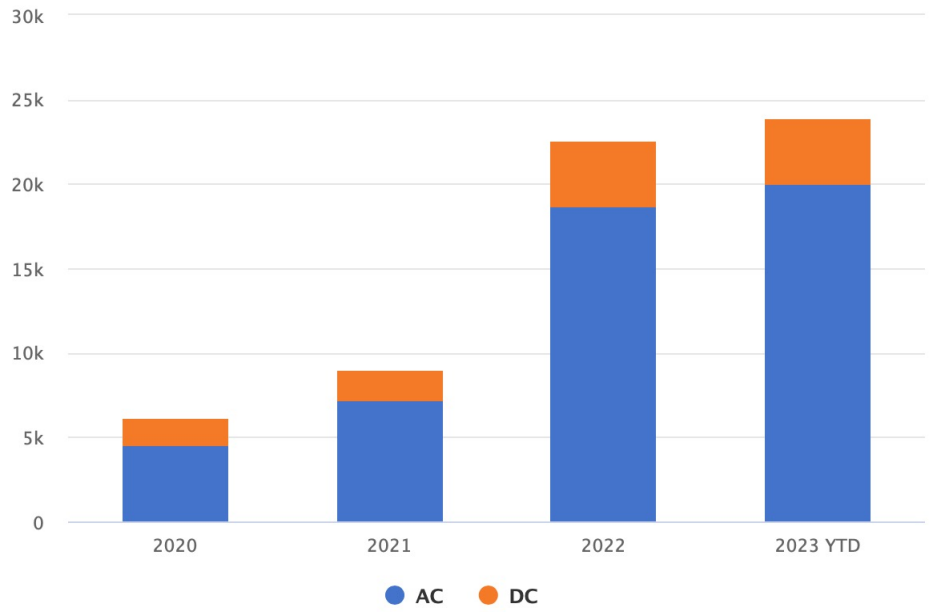

Figura 6: Número total de estaciones de carga en España desde 2020 hasta 2023, (fuente: [11])

Ambos elementos, restricción de circulación en grandes urbes junto con una imagen atractiva, han hecho avanzar a una nueva etapa de desarrollo de los vehículos eléctricos. Esta revitalización no solo afecta a los vehículos eléctricos de uso privado, sino también al uso de otros medios como el vehículo eléctrico compartido, como vehículos, motocicletas, bicicletas o patinetes. Es por todas estas razones que numerosas ciudades de la Unión Europa han invertido en la mejora y puesta a punto de infraestructuras de carga para todo tipo de transportes eléctricos, como podemos observar en la figura 6 [11], el número de estaciones de carga ha aumentado de manera muy significativa durante los últimos años. Cabe destacar la distinción entre las estaciones de carga que se clasifican como de tipo *Alternate Current (AC)* y *Direct Current (DC)*, la cual reside en la ubicación donde ocurre la transformación de la energía de corriente alterna, *AC*, ya sea dentro o fuera del automóvil. En contraposición con los dispositivos de carga de *AC*, un cargador de tipo *DC* dispone del convertidor dentro del propio cargador, lo que permite la introducción directa de la corriente eléctrica en la batería del vehículo sin necesidad de un cargador adicional para llevar a cabo dicha conversión. Además, los cargadores de *DC* son de mayor tamaño, más rápidos y representan un notable avance en el ámbito de los vehículos eléctricos. [12]

3.2. Infraestructura hardware y software de las estaciones de carga

Después de contextualizar al lector sobre la situación actual en España en lo que respecta a los vehículos eléctricos y las infraestructuras de carga, resulta apropiado proporcionar información detallada acerca del funcionamiento de las estaciones de carga. Uno de los valores más preciados en la vida es el tiempo, un factor que influye significativamente en la recarga de cualquier tipo de vehículo eléctrico. Por tanto, existen distintos modos de carga que requieren períodos de carga específicos para recargar el vehículo:

- **Modo 1**: el automóvil se acopla a una fuente de energía eléctrica domiciliaria sin establecer un protocolo de comunicación específico ni hacer uso de dispositivos intermedios. Dicha práctica adolece de insuficiencias en materia de seguridad, razón por la cual suele reservarse al suministro de carga eléctrica de vehículos de dos ruedas, en tanto que los automóviles no son susceptibles de ser cargados bajo esta modalidad.

- **Modo 2**: los vehículos de propulsión eléctrica suelen estar equipados con un convertidor interno que les permite acoplarse a un enchufe de corriente alterna de uso doméstico, con el fin de recargar las baterías mediante la intermediación de un dispositivo específico que garantiza la eficacia de la carga. A pesar de que es posible emplearlo ocasionalmente para la recarga de automóviles, los fabricantes no lo recomiendan. Además, en el ámbito domiciliario, los enchufes convencionales tienen una limitación de 16 amperios y la carga eléctrica demanda entre seis y ocho horas de tiempo.

- **Modo 3**: se instaura una conexión comunicativa entre el automóvil y el punto de recarga, el cual puede estar representado por un cargador de uso domiciliario, un espacio habilitado en el estacionamiento de una edificación comercial, o una estación de carga de acceso público. La manguera empleada en la operación de carga cuenta con diversos contactos que efectúan un control del proceso y verifican que la carga se efectúe de manera adecuada. La energía eléctrica que se suministra es alterna, lo que posibilita una carga a mayor potencia y el doble de velocidad en comparación con la que se obtiene por medio de una toma de corriente convencional.

- **Modo 4**: la carga rápida constituye una alternativa de mayor costo que se encuentra accesible en áreas de servicio, centros comerciales y otros puntos de acceso público.

Estas estaciones se caracterizan por disponer de un convertidor de corriente externo que posibilita la recarga en corriente continua y hacen uso de una conexión distinta al modo 3, lo que permite una carga notablemente más veloz. Este tipo de estaciones suele ostentar una potencia nominal igual o superior a 50 kW, lo que se traduce en la capacidad de cargar la batería de un vehículo en un lapso inferior a 30 minutos. Además, dentro de esta modalidad de carga, se encuentran disponibles unos puntos de recarga de ultrarrápida capacidad que pueden suministrar hasta 700 kW de energía eléctrica, permitiendo recargar el 80 % de la capacidad total en un lapso de tiempo cercano a seis minutos. Ciertos países, como Israel, Holanda o China, han llevado a cabo pruebas con estaciones de recarga en las que se reemplaza la batería agotada por otra previamente cargada en un periodo reducido de minutos, no obstante, esta tecnología no ha obtenido éxito en virtud de los elevados costes que implica y la falta de un estándar unificado en cuanto a las baterías de los vehículos eléctricos.

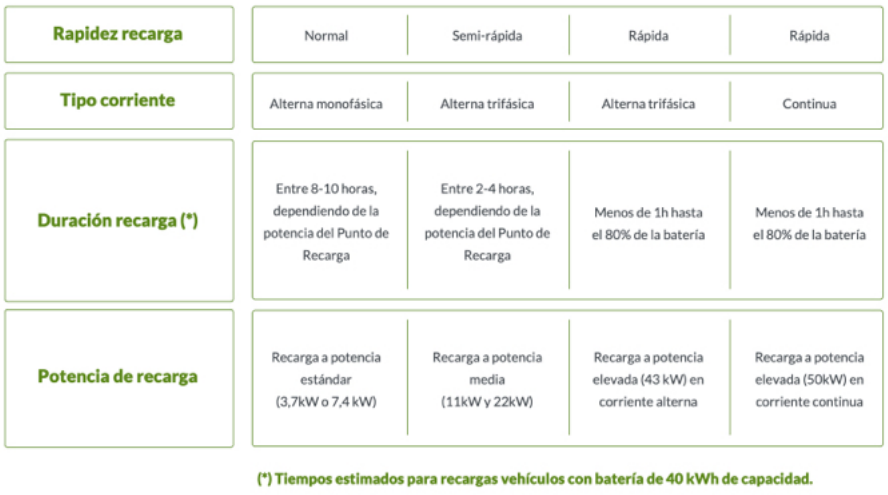

Figura 7: Rapidez, tipo, duración y potencia de los distintos tipos de recarga, (fuente: [13])

Así pues, se puede observar que según el modo de carga utilizado el lapso de tiempo requerido para la recarga del vehículo puede variar desde unos pocos minutos hasta ocho, o incluso, diez horas, como se puede inferir de la figura 7. [13]

Por otro lado, las estaciones de carga disponen de varios tipos de enchufes según si sean del tipo *AC* o *DC*, existiendo dos variantes de enchufes para cada tipo de estación de carga, los

cuales se explican a continuación:

- **Estaciones de carga del tipo *AC***:

Figura 8: Enchufe de tipo 1, (fuente: [28])

- **Enchufe tipo 1**: siendo una interfaz de carga monofásica, se establece como la norma en los vehículos eléctricos de América y Asia. Este conector permite el abastecimiento del vehículo con una capacidad de carga de hasta 7,4 kW, acorde a la potencia de carga del vehículo y a la capacidad de suministro eléctrico de la red en la que se conecte. Véase figura 8.

Figura 9: Enchufe de tipo 2, (fuente: [29])

- **Enchufe tipo 2**: se distinguen por ser enchufes trifásicos, en razón de que utilizan tres conductores para la transferencia de corriente eléctrica, lo que les permite recargar un vehículo eléctrico con mayor rapidez. En un ámbito domiciliario, la velocidad de carga que se alcanza con este tipo de enchufe puede llegar a los 22 kW, mientras que en estaciones de carga de acceso público, la potencia de carga puede llegar a los 43 kW. En ambos casos, la capacidad de carga depende de la potencia de carga que soporte el vehículo y de la capacidad de la red eléctrica disponible. Véase figura 9.

- **Estaciones de carga del tipo *DC***:

Figura 10: Enchufe de tipo *CHarge de MOve (CHAdeMO)*, (fuente: [30])

- **Enchufe de tipo *CHAdeMO***: originario de Japón, fue diseñado para ofrecer capacidades de carga de gran magnitud, además de la posibilidad de carga bidireccional. En la actualidad, los fabricantes de automóviles de Asia están liderando el mercado de los vehículos eléctricos que son compatibles con el enchufe *CHAdeMO*, permitiendo una carga de hasta 100 kW. Véase figura 10.

Figura 11: Enchufe de tipo *Combined Charging System (CCS)*, (fuente: [31])

- **Enchufe de tipo *CCS***: en términos técnicos, el enchufe *CCS* representa una evolución del enchufe de tipo 2 utilizado en Europa, que incorpora dos contactos de alimentación adicionales para la carga rápida de corriente continua en estaciones de carga rápida. Esto permite la carga tanto en corriente alterna como en corriente continua y es compatible con una amplia variedad de vehículos eléctricos. El enchufe *CCS* puede alcanzar una velocidad de carga de hasta 350 kW, lo que representa una de las opciones más rápidas disponibles en la actualidad. Véase figura 11.

Por otro lado, las estaciones de carga que en este proyecto se tratan se encuentran dentro de una topología compuesta por cuatro niveles distintos:

- **Nivel 1**: en la estratificación inicial, se ubican los sistemas centrales de monitorización y control, denominados *Charging Station Management System (CSMS)*. Estos mecanismos desempeñan una función primordial en la supervisión y control remoto de las diversas estaciones de carga que se vinculan a ellos.

- **Nivel 2**: en este escalafón, se hallan las estaciones de carga, las cuales subsumen todas las estratificaciones subyacentes y ostentan la responsabilidad de brindar soporte al usuario.

- **Nivel 3**: en esta categoría, se ubican los *Electric Vehicle Supply Equipment (EVSE)*, cada uno de los módulos que aportan energía en una estación de carga. Es crucial que se ajusten al modo y tipo de carga que requiere cada vehículo eléctrico.

- **Nivel 4**: por último, en la estratificación más baja y final, se hallan los conectores, los cuales tienen como objetivo proporcionar la energía requerida para llevar a cabo la carga del vehículo eléctrico.

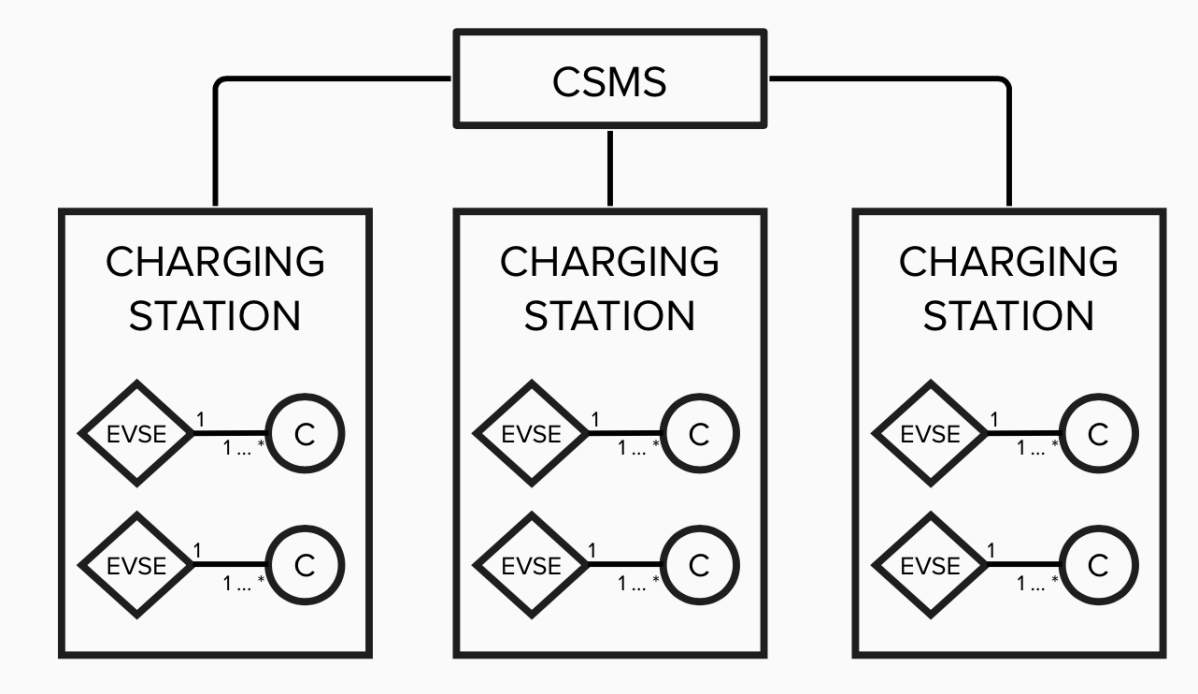

Figura 12: Topología de las conexiones entre las distintas capas referentes a las estaciones de carga

En este sentido, cabe destacar que una *CSMS* tiene la capacidad de monitorizar y supervisar varias estaciones de carga. A su vez, cada una de estas estaciones de carga controla uno o varios módulos *EVSE*, los cuales pueden contar con uno o varios tipos de conectores disponibles para el usuario. De esta manera, se establece una jerarquía en cascada que abarca desde los sistemas centrales de monitorización y control hasta los conectores, con el propósito de garantizar un

suministro eficiente y adecuado de energía eléctrica para la carga de los vehículos eléctricos [15]. Véase figura 12.

3.3. Protocolo de comunicación *OCPP*

Figura 13: Logotipo *OCPP*, (fuente: [16])

El *OCPP* es un protocolo de comunicación de código abierto desarrollado por la *Open Charge Alliance (OCA)* [16] para permitir la interoperabilidad entre las estaciones de carga de vehículos eléctricos y los sistemas de gestión de carga. El objetivo de *OCPP* es proporcionar una interfaz común para la comunicación entre los puntos de carga y los sistemas de gestión de carga, independientemente de su fabricante, lo que permite a los usuarios de vehículos eléctricos cargar sus vehículos en cualquier estación de carga, independientemente del fabricante o la tecnología utilizada.

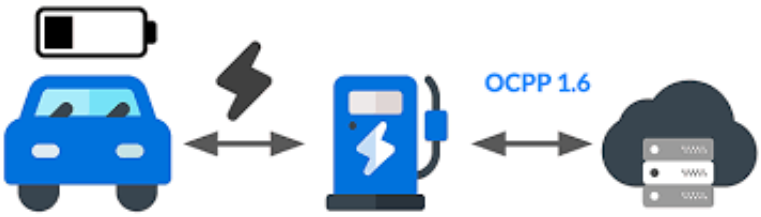

Figura 14: Gráfico general de aplicación del protocolo *OCPP* a las transacciones de recarga de vehículos eléctricos, (fuente: [32])

OCPP ha evolucionado a lo largo del tiempo, con diferentes versiones que ofrecen diferentes características y mejoras en la funcionalidad. *OCPP 1.6*, publicado en 2011, es la primera

versión del protocolo y ofrece un conjunto básico de comandos y notificaciones para la comunicación entre los puntos de carga y los sistemas de gestión de carga. *OCPP 2.0*, lanzado en 2018, es una versión más avanzada del protocolo que incluye nuevas características y mejoras de seguridad, como autenticación mutua y cifrado de datos. *OCPP 2.0.1*, lanzado en 2020, es una actualización menor de *OCPP 2.0* que corrige errores y agrega pequeñas mejoras de funcionalidad.

En este trabajo de fin de grado se ha utilizado una implementación del protocolo *OCPP* basada en el lenguaje de programación *Python*, llamada *Open Charge Point Protocol - Asynchronous Server Gateway Interface (OCPP-ASGI)* [19] la cual es una implementación del protocolo utilizando el marco de trabajo *ASGI* de *Python*, una especificación para servidores web que permite a los desarrolladores crear aplicaciones web *Python* que manejan solicitudes *Hyper Text Transfer Protocol (HTTP)* de manera asíncrona. La principal ventaja de utilizar *ASGI* es que permite una mejor escalabilidad y rendimiento de las aplicaciones web en comparación con otras tecnologías web en *Python* como *Web Server Gateway Interface (WSGI)* y *Common Gateway Interface (CGI)*. *Asynchronous Server Gateway Interface (ASGI)* es utilizado por varios *frameworks* web en *Python* como *Django*, *FastAPI* y *Starlette*, entre otros. La implementación de *OCPP-ASGI* proporciona una forma sencilla de construir servidores *OCPP*, lo que permite una fácil integración de estaciones de carga y vehículos eléctricos en redes de carga eléctrica, permitiendo además una ejecución multi-versión, aceptando así conexiones *OCPP* 1.6, 2.0 y 2.0.1.

3.4. Otros protocolos de comunicación

Aunque el enfoque principal de este proyecto ha sido la recolección de datos basados en la especificación del estándar *OCPP*, las estaciones de carga eléctrica no se limitan únicamente a este protocolo de comunicación para establecer conexiones con las centrales de control. Antes de la adopción de *OCPP*, el mercado estaba dominado por redes y protocolos de comunicación propietarios, lo que dificultaba la implementación de estaciones de carga en masa, ya que cada estación debía ser adaptada a los requisitos específicos de hardware y software de cada protocolo.

Con la aparición del protocolo *OCPP*, las estaciones de carga eléctrica ahora tienen la capacidad de adaptarse fácilmente a los cambios y no necesitan de software ni hardware propietario

para operar, lo que facilita su expansión en el mercado. No obstante, existen otros protocolos de comunicación menos conocidos que también pueden ser utilizados para estos fines.

3.4.1. Protocolo *OSCP*

El protocolo *Open Smart Charging Protocol (OSCP)* es un protocolo de comunicación abierto desarrollado por la *OCA*. Es un protocolo entre un sistema de gestión de puntos de carga *(Charge Point Management System (CPMS))* y un proveedor de servicios de carga inteligente. Estos suelen ser operadores de red, agregadores de respuesta a la demanda o sistemas de gestión de energía.

La función básica del protocolo *OSCP* es comunicar la capacidad neta física del *Distribution System Operator (DSO)* (o propietario del sitio) a la oficina central del operador del punto de carga. El protocolo se puede utilizar para comunicar una predicción de 24 horas de la capacidad local disponible al operador del punto de carga.

Los Operadores de Sistemas de Distribución *(DSO)* son las entidades encargadas de distribuir y gestionar la energía desde las fuentes de generación hasta los consumidores finales. La digitalización es clave para asegurar el modelo de *DSO*, lo que requiere inversiones en automatización, medidores inteligentes, sistemas en tiempo real, *big data* y análisis de datos.

El modelo de *DSO* utiliza medidores inteligentes que permiten la lectura bidireccional del flujo de energía y la comunicación en tiempo real. Esto hace posible detectar interrupciones y restablecer automáticamente el suministro, así como facilitar el seguimiento del consumo diario de los clientes a través de plataformas de consulta digital disponibles para ellos [20].

La versión 2.0 del protocolo *OSCP* se lanzó oficialmente en octubre de 2020 y describe casos de uso en los que los mensajes se aplican en términos más genéricos que *OSCP* 1.0, que estaba específicamente dirigido a la carga inteligente de vehículos eléctricos por parte de un operador del sistema de distribución, la *(DSO)*. La razón para usar términos más genéricos es que esta especificación no quiere limitar las posibilidades del protocolo a la carga inteligente de vehículos eléctricos. Esto se debe a la integración de los vehículos eléctricos en ecosistemas energéticos más grandes, que incluyen fotovoltaica, baterías estacionarias, bombas de calor y otros dispositivos. [21]

3.4.2. Protocolo *IEC-63110*

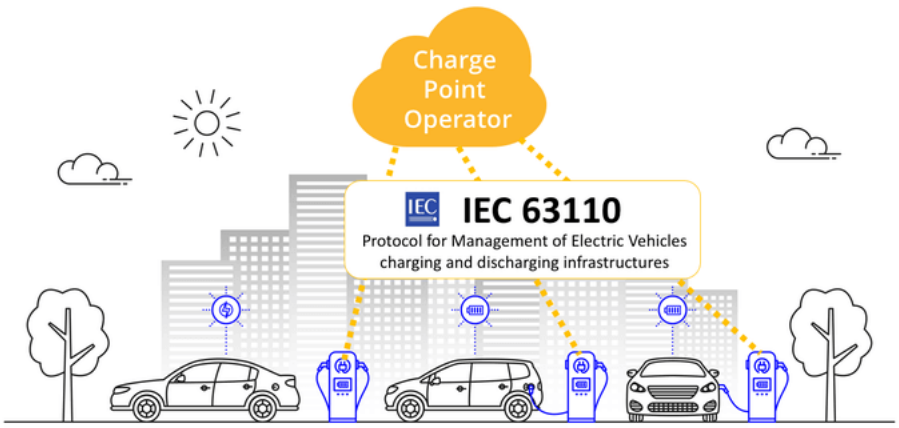

Figura 15: Protocolo *IEC-63110*, (fuente: [22])

El *IEC-63110*, también conocido como el protocolo para la gestión de infraestructuras de carga y recarga de vehículos eléctricos, proporciona enlaces de comunicación entre las estaciones de carga conductiva y los vehículos eléctricos. Junto con otros estándares *ISO*, el *IEC-63110* es responsable de un mayor y rápido nivel de interoperabilidad en la comunicación de *front-end* y la distribución de señales entre infraestructuras de red inteligentes como los vehículos eléctricos y las estaciones de carga conductiva.

El *IEC-63110* es complejo por diseño. Sin embargo, su estándar asegura la ciberseguridad, interoperabilidad, integración en la red y escalabilidad. El modelo de diseño comienza con una interfaz que permite que todos los actores en un mercado específico interactúen entre sí.

OCPP e *IEC-63110* son sistemas de protocolos necesarios para una comunicación eficaz. Muchos usuarios y entusiastas podrían preguntarse: *"¿hay que descartar uno de ellos o pueden coexistir los dos?"*. Aunque *OCPP* e *IEC-63110* comparten numerosas similitudes, son únicos en sus funcionalidades y especificaciones.

OCPP e *IEC-63110* son como hermanos con alguna diferencia entre ellos. Sin embargo, tienen profundas similitudes en cuanto a capacidad de gestión de la red, autenticación y autorización de pagos (teniendo *IEC-63110* más funciones), gestión de transferencias de energía y gestión de equipos de vehículos eléctricos. Tanto *OCPP* como *IEC-63110* ofrecen ventajas simi-

lares en la eliminación del tiempo de espera durante la integración y en los retos relacionados con *Information Technology (IT)*. Pero, sobre todo, ofrecen una interoperabilidad activa que es vital para todos los operadores, empresas, propietarios y conductores de vehículos eléctricos. Aunque *OCPP* y el *IEC-63110* tienen funciones bastante similares, algunos expertos creen que el *IEC-63110* es una actualización del protocolo *OCPP* y tiene un sistema de interacción y comunicación de movilidad eléctrica mejorado. [22]

4

Requisitos y arquitectura

4.1. Requisitos funcionales y no funcionales

En el desarrollo de cualquier aplicación, es fundamental establecer una lista clara de requisitos que describan las funcionalidades y características necesarias para satisfacer las necesidades de los usuarios. Estos requisitos se dividen en dos categorías principales: funcionales y no funcionales.

Los requisitos funcionales se centran en las acciones y tareas que el sistema debe realizar. Estos requisitos describen las funcionalidades específicas que los usuarios podrán utilizar, como el registro de usuarios, la gestión de reservas, la visualización de información, entre otros. Cada requisito funcional se identifica con un código único y se proporciona una descripción extendida para una comprensión más detallada de la funcionalidad requerida.

Por otro lado, los requisitos no funcionales se refieren a los atributos y características del sistema que no están directamente relacionados con las funcionalidades específicas, pero que son igualmente importantes para su correcto funcionamiento. Estos requisitos abarcan aspectos como la seguridad, el rendimiento, la usabilidad y la compatibilidad. Se definen criterios específicos que el sistema debe cumplir, como tiempos de respuesta, requisitos de hardware, estándares de uso, entre otros.

Tanto los requisitos funcionales como los no funcionales son elementos esenciales en la planificación y desarrollo de la aplicación. Establecen las bases para el diseño y la implementación, asegurando que el sistema cumpla con los objetivos establecidos y brinde una experiencia de usuario satisfactoria. Al definir y comprender estos requisitos desde el inicio del proceso de desarrollo, se facilita la toma de decisiones y se mejora la eficiencia en el desarrollo del

producto final.

A continuación, se presentan detalladamente los requisitos funcionales y no funcionales de la aplicación, con el objetivo de proporcionar una visión clara y completa de las funcionalidades y características de ambos sitemas: *Control-CSSC* y *Admin-CSSC*.

4.1.1. Requisitos funcionales de *Control-CSSC*

- **CRF-01**: el sistema permite al usuario registrarse en el sistema.
  El sistema proporciona una funcionalidad que permite a los usuarios registrarse en el sistema ingresando la información requerida, como nombre, dirección de correo electrónico y contraseña. Una vez que el usuario completa el proceso de registro, se crea una cuenta asociada a sus credenciales, lo que le permite acceder a las funcionalidades del sistema.

- **CRF-02**: el sistema permite al usuario iniciar sesión en el sistema.
  El sistema proporciona una funcionalidad que permite a los usuarios iniciar sesión en el sistema utilizando sus credenciales previamente registradas. Al ingresar su usuario y contraseña, el sistema verifica la autenticidad de la información y concede el acceso al usuario a su cuenta personalizada y a las características correspondientes.

- **CRF-03**: el sistema permite al usuario consultar los cargadores disponibles a través de un mapa.
  El sistema ofrece a los usuarios la capacidad de acceder a un mapa interactivo que muestra la ubicación de los cargadores disponibles. Los usuarios pueden visualizar los cargadores en diferentes áreas geográficas y obtener información adicional, como la disponibilidad actual y los cargadores ofrecidos en cada ubicación.

- **CRF-04**: el sistema permite al usuario consultar los tramos horarios disponibles de un cargador específico en un día específico.
  El sistema brinda a los usuarios la opción de verificar la disponibilidad de los tramos horarios para un cargador específico en un día determinado. Los usuarios pueden seleccionar un cargador en particular y ver los horarios disponibles, lo que les permite planificar su carga en función de la disponibilidad de los cargadores.

- **CRF-05**: el sistema permite al usuario realizar la reserva de un cargador en un tramo horario específico en un día especfíco.
  Los usuarios tienen la posibilidad de realizar una reserva para utilizar un cargador en un tramo horario específico en un día determinado. Al seleccionar un cargador disponible y un horario deseado, el sistema registra la reserva y reserva el puesto para el usuario durante el período especificado.

  - **CRF-05.1**: el sistema envía por correo electrónico una confirmación de la reserva efectuada por parte del sistema permite al usuario.
    Después de que el usuario haya realizado una reserva de un cargador en un tramo horario específico, el sistema envía automáticamente un correo electrónico al usuario con los detalles de la reserva, incluyendo la información referente a la misma.

- **CRF-06**: el sistema permite al usuario consultar sus reservas futuras y expiradas.
  Los usuarios tienen la capacidad de consultar una lista de sus reservas actuales y pasadas. Esta funcionalidad les proporciona una visión general de las reservas programadas y las que ya han expirado, lo que les permite administrar y realizar un seguimiento de su historial de reservas

  - **CRF-06.1**: el sistema permite al usuario consultar sus reservas filtrándolas por fecha.
    Los usuarios tienen la opción de filtrar sus reservas por fecha para visualizar únicamente las reservas que ocurrieron en un período específico. Esto les brinda flexibilidad para revisar su historial de reservas en un rango de fechas determinado.
  - **CRF-06.2**: el sistema permite al usuario consultar sus reservas filtrándolas por cargador.
    Los usuarios pueden aplicar un filtro por cargador para ver únicamente las reservas asociadas a un cargador en particular. Esto les ayuda a organizar y revisar sus reservas según la ubicación de los cargadores utilizados.
  - **CRF-06.3**: el sistema permite al usuario consultar sus reservas ordenándolas por fecha.

Los usuarios tienen la opción de ordenar sus reservas en función de la fecha en la que se realizaron. Esta función proporciona una forma conveniente de visualizar y revisar las reservas en un orden cronológico.

- **CRF-06.4**: el sistema permite al usuario consultar sus reservas ordenándolas por cargador.
  Los usuarios pueden ordenar sus reservas según el cargador utilizado. Esto facilita la identificación y revisión de las reservas asociadas a un cargador específico en particular por orden de cargador.
- **CRF-06.5**: el sistema permite al usuario consultar sus reservas ordenándolas por orden ascendente o descendente.
  Los usuarios tienen la capacidad de ordenar sus reservas en orden ascendente o descendente, ya sea por fecha, cargador u otro criterio relevante. Esto les brinda flexibilidad para adaptar la visualización de sus reservas según sus preferencias.

- **CRF-07**: el sistema permite al usuario eliminar una reserva realizada antes de la fecha y hora de su ejecución.
  Los usuarios tienen la opción de cancelar una reserva antes de la fecha y hora programadas para su uso. Al realizar esta acción, el sistema libera el cargador reservado y permite que otros usuarios lo utilicen. Además, se envía automáticamente un correo electrónico de confirmación al usuario notificando la cancelación de la reserva.

  - **CRF-07.1**: el sistema envía por correo electrónico una confirmación de la cancelación de la reserva efectuada por parte del usuario.
    Después de que el usuario haya cancelado una reserva, el sistema envía automáticamente un correo electrónico de confirmación al usuario. Este correo electrónico incluye el identificador de la reserva cancelada.

- **CRF-08**: el sistema permite al usuario comenzar la carga en un puesto bajo reserva en una fecha y franja de tiempo determinadas.
  Los usuarios tienen la capacidad de iniciar la carga en un puesto de carga que hayan reservado previamente. Esto implica conectar su vehículo eléctrico al cargador y activar la carga en el momento programado de acuerdo con la reserva realizada. El sistema

valida la reserva y permite que el usuario comience la carga en el período específico reservado.

- **CRF-09**: el sistema permite al usuario terminar la carga en un puesto bajo reserva siempre que ésta ya haya comenzado en un tiempo pasado.
  Los usuarios tienen la opción de finalizar la carga en un puesto de carga que hayan reservado previamente, siempre que la carga ya haya comenzado en un momento anterior. Esto les brinda flexibilidad para ajustar la duración de la carga según sus necesidades y les permite liberar el puesto de carga para que otros usuarios lo utilicen.

- **CRF-10**: el sistema finaliza la carga de un usuario automáticamente si éste no la ha finalizado dentro del período de carga y su período útil ha expirado.
  El sistema tiene la capacidad de finalizar automáticamente la carga de un usuario si éste no ha completado el proceso de carga dentro del tiempo designado y su período útil ha expirado. El período de carga se define como el tiempo asignado para que un usuario utilice un cargador determinado. Si el usuario no ha finalizado la carga antes de que expire este período, el sistema toma la acción de finalizar la carga de manera automática.

- **CRF-11**: el sistema permite al usuario eliminar una reserva expirada tras la fecha y hora de su validez.
  Los usuarios tienen la capacidad de eliminar una reserva que ha expirado, es decir, una reserva cuya fecha y hora de validez han pasado. Esta acción proporciona una forma de gestionar y limpiar las reservas que ya no son relevantes.

- **CRF-12**: el sistema permite al usuario crear un tiquet de incidencia.
  Los usuarios pueden crear un tiquet de incidencia para informar sobre cualquier problema o inconveniente que hayan experimentado durante el proceso de carga o en el propio uso del sistema. Esto permite que los usuarios notifiquen al equipo de soporte técnico del sistema sobre situaciones inesperadas o necesidades de asistencia. El tiquet de incidencia captura los detalles del problema y se envía al equipo de administración para su resolución.

  - **CRF-12.1**: el sistema envía por correo electrónico una confirmación de la creación

del tiquet de incidencia.

Después de que el usuario haya creado un tiquet de incidencia, el sistema envía automáticamente un correo electrónico de confirmación al usuario. Este correo electrónico incluye los detalles del tiquet de incidencia.

- **CRF-12.2**: el sistema envía por correo electrónico la respuesta del administrador al tiquet creado por el usuario.

  Una vez la incidencia haya sido gestionada y tratada por un personal perteneciente a la adminstración, éste marcará como resuelto el mismo, notificando vía correo electrónico al usuario.

4.1.2. Requisitos no funcionales de *Control-CSSC*

- **CRNF-01**: para que un usuario se registre en el sistema debe aportar un nombre de usuario y correo electrónico que no existan aún en el sistema, siendo el correo electrónico del dominio *@uma.es* y una contraseña que cumpla los requerimientos mínimos de seguridad.

  Este requisito es fundamental para garantizar la integridad y autenticidad de los usuarios en el sistema, así como para evitar duplicidad de cuentas y asegurar que solo los usuarios autorizados tengan acceso a la aplicación.

- **CRNF-02**: para que un usuario inicie sesión en el sistema debe aportar un nombre de usuario y contraseña válidos.

  Esta validación es necesaria para verificar la identidad del usuario y asegurar que solo las personas autorizadas puedan acceder a la aplicación. Al solicitar un nombre de usuario y contraseña válidos, se garantiza un nivel adicional de seguridad y confidencialidad en el sistema.

- **CRNF-03**: el sistema guarda la sesión del usuario a través de *cookies*, usando *tokens* del tipo *JSON Web Tokens (JWT)* con dos *cookies*: sesión (*refresh*) y acceso.

  Las *cookies* son pequeños archivos de texto que se almacenan en el dispositivo del usuario y contienen información sobre su sesión. Esta funcionalidad permite mantener la sesión activa, incluso después de cerrar el navegador o reiniciar el dispositivo. Al utilizar *cookies*, se mejora la experiencia del usuario al evitar la necesidad de iniciar sesión

repetidamente, lo que brinda comodidad y eficiencia en el uso del sistema.

- **CRNF-04**: la franja máxima otorgada para cada reserva es de dos horas y la mínima es de 15 minutos por usuario y cargador.
  Estas limitaciones garantizan una asignación equitativa de los cargadores y evitan el bloqueo prolongado de los mismos. Al establecer una duración máxima y mínima para las reservas, se promueve la disponibilidad y la eficiencia en el uso de los recursos, permitiendo que más usuarios tengan acceso a los cargadores en un período determinado.

- **CRNF-05**: en el sistema sólo se podrán realizar reservas en un horario de 08:00h a 22:00h.
  Esta restricción se implementa para asegurar que las reservas se realicen dentro de un período de tiempo razonable y acorde con las políticas establecidas. Al limitar el horario de reservas, se evitan solicitudes fuera del horario de funcionamiento habitual o en momentos de mantenimiento del sistema, lo que contribuye a un mejor control y planificación de los recursos disponibles.

- **CRNF-06**: para visualizar las franjas horarias disponibles, El sistema permite al usuario seleccionar un día en el calendario.
  Esta interacción permite mostrar al usuario las opciones de reserva disponibles para el día seleccionado, lo que facilita la planificación de su carga. Al proporcionar un calendario interactivo, se mejora la usabilidad y se brinda al usuario la capacidad de elegir la fecha que mejor se ajuste a sus necesidades.

- **CRNF-07**: para visualizar las horas posibles de fin de reserva disponibles, El sistema permite al usuario seleccionar un día y una fecha de comienzo.
  Con esta información, se presentan las opciones de franjas horarias disponibles para finalizar la reserva, lo que permite al usuario elegir el momento más conveniente para finalizar su carga. Esta funcionalidad mejora la experiencia del usuario al proporcionarle información precisa sobre las opciones de finalización de la reserva y permitirle ajustar su planificación según sus necesidades específicas.

- **CRNF-08**: para que una reserva se realice correctamente, ni la fecha ni la franja horaria de la misma puede ser en el pasado.

Esta validación es necesaria para garantizar que las reservas se realicen para momentos próximos y evitar solicitudes de reserva en el pasado. Al verificar que las reservas sean para fechas y horarios futuros, se garantiza que los recursos estén disponibles y que las operaciones de carga se realicen de manera oportuna.

- **CRNF-09**: para que una reserva se etiquete como expirada, debe no haber sido utilizada y encontrarse en el pasado.
  Esta funcionalidad permite identificar y gestionar adecuadamente las reservas que no han sido utilizadas dentro de su período de validez. Al marcar las reservas expiradas, se liberan los recursos para que otros usuarios puedan aprovecharlos y se mantiene un sistema ordenado y eficiente.

- **CRNF-10**: para desbloquear el cargador el sistema permite al usuario hacerlo en la franja horaria reservada para ello, utilizando el botón designado para esa función, no permitiendo la aplicación realizar esta transacción en una franja de tiempo distinta a la especificada en la reserva del sistema permite al usuario al cargador.
  No se permite que la aplicación realice esta transacción en una franja de tiempo distinta a la especificada en la reserva. Esta restricción garantiza que el acceso a los cargadores esté controlado y que los usuarios utilicen los cargadores según las reservas realizadas, evitando conflictos y asegurando un uso adecuado de los recursos.

- **CRNF-11**: el sistema sólo permite la entrada de usuarios de tipo estándar a la aplicación.
  Esto significa que se limita el acceso a usuarios con permisos o roles específicos, permitiendo el uso de la aplicación exclusivamente a usuarios estándar, no a administradores. Esta restricción se implementa para mantener la seguridad y la integridad del sistema, evitando el acceso no autorizado o el uso indebido de las funcionalidades por parte de usuarios no autorizados.

- **CRNF-12**: el sistema sólo permite a los usuarios realizar reservas en una fecha y franja horaria si no tiene otra reserva activa.
  Esto significa que, si un usuario ya ha realizado una reserva en un cargador en una fecha y franja horaria determinada, el sistema no permite realizar cualquier otra reserva en cualquier otro cargador en la misma fecha y con una franja horaria que sea la misma o

que genere conflicto con la ya existente.

4.1.3. Requisitos funcionales de *Admin-CSSC*

- **ARF-01**: el sistema permite al administrador iniciar sesión en el sistema con una cuenta válida de administrador.
  El sistema permite al administrador acceder al sistema mediante el uso de credenciales válidas de administrador. Esto garantiza que solo los administradores autorizados tengan acceso a las funcionalidades y características reservadas para ellos. Al iniciar sesión, el administrador podrá realizar tareas de gestión y supervisión dentro del sistema.

- **ARF-02**: el sistema permite al administrador ver todas las reservas efectuadas en el sistema por cualquier usuario.
  El sistema proporciona al administrador la capacidad de visualizar todas las reservas realizadas por los usuarios en el sistema. Esta funcionalidad es importante para que el administrador tenga una visión general de las reservas realizadas, lo que le permite supervisar y gestionar el uso de los cargadores. Al ver todas las reservas, el administrador puede tomar decisiones informadas sobre la asignación de recursos y la resolución de posibles conflictos.

  - **ARF-02.1**: el sistema permite al administrador eliminar reservas existentes en el sistema siempre y cuando no estén en uso.
    Permite al administrador eliminar reservas que consideren inválidas, no autorizadas o que incumplan las políticas establecidas. Al eliminar una reserva, se liberan los recursos del cargador correspondiente y se permite que otros usuarios realicen nuevas reservas. Esta funcionalidad brinda al administrador el control necesario para garantizar un uso eficiente y adecuado de los cargadores.

  - **ARF-02.2**: el sistema permite al administrador filtrar las reservas efectuadas en el sistema por *ID* de usuario.
    Proporciona al administrador la capacidad de filtrar las reservas realizadas en el sistema según el *ID* de usuario. Al aplicar este filtro, el administrador puede obtener información específica sobre las reservas realizadas por usuarios particulares. Esto resulta útil para realizar un seguimiento de las actividades de un usuario específico

o para resolver problemas relacionados con reservas específicas.

- **ARF-02.3**: el sistema permite al administrador filtrar las reservas efectuadas en el sistema por fecha.
  El administrador puede aplicar un filtro basado en la fecha para visualizar las reservas realizadas en el sistema. Al filtrar por fecha, el administrador puede obtener una lista de reservas específicas realizadas en un rango de fechas determinado. Esto facilita la identificación de patrones, tendencias o posibles problemas relacionados con las reservas en un período de tiempo específico.

- **ARF-02.4**: el sistema permite al administrador filtrar las reservas efectuadas en el sistema por cargador.
  Permite al administrador filtrar las reservas realizadas en el sistema según el cargador específico. Al aplicar este filtro, el administrador puede ver todas las reservas asociadas a un cargador en particular. Esto facilita el seguimiento del uso de cada cargador, su disponibilidad y la capacidad de detectar cualquier problema relacionado con un cargador en particular.

- **ARF-02.5**: el sistema permite al administrador filtrar las reservas efectuadas eligiendo si listas aquellas que están eliminadas o usadas o no.
  Permite al administrador aplicar un filtro adicional a las reservas realizadas en el sistema, con la opción de listar aquellas que estén eliminadas o aún no han sido utilizadas. Esta funcionalidad brinda al administrador la capacidad de obtener una visión detallada de las diferentes categorías de reservas, lo que puede ser útil para el seguimiento de las reservas canceladas, el análisis del uso de los cargadores y la identificación de posibles problemas o inconsistencias en el sistema.

- **ARF-02.6**: el sistema permite al administrador ordenar las reservas efectuadas en el sistema por *ID* de usuario.
  El administrador tiene la capacidad de ordenar las reservas realizadas en el sistema en función del ID de usuario. Al aplicar este ordenamiento, las reservas se presentan de manera secuencial, facilitando la identificación de las reservas asociadas a usuarios específicos y permitiendo un análisis más detallado de las actividades de cada usuario.

- **ARF-02-7**: el sistema permite al administrador ordenar las reservas efectuadas en el sistema por fecha.

  Permite al administrador ordenar las reservas realizadas en el sistema según la fecha en que se realizaron. Al aplicar este ordenamiento, las reservas se presentan en secuencia cronológica, lo que facilita el seguimiento del historial de reservas y la identificación de patrones o tendencias temporales.

- **ARF-02.8**: el sistema permite al administrador ordenar las reservas efectuadas en el sistema por cargador.

  El administrador puede ordenar las reservas realizadas en el sistema según el cargador al que están asociadas. Al aplicar este ordenamiento, las reservas se presentan agrupadas por cargador, lo que facilita el seguimiento del uso de cada cargador y proporciona una visión general de la demanda de los diferentes cargadores.

- **ARF-02.9**: el sistema permite al administrador ordenar las reservas efectuadas en el sistema por orden ascendente o descendente.

  Permite al administrador seleccionar el orden en que se muestran las reservas realizadas en el sistema. Puede optar por ordenarlas de forma ascendente (de la más antigua a la más reciente) o descendente (de la más reciente a la más antigua). Esta funcionalidad proporciona flexibilidad al administrador al presentar la información de acuerdo con sus necesidades y preferencias.

- **ARF-03**: el sistema permite al administrador crear cargadores para el uso de los mismos dentro de la aplicación.

  El administrador tiene la capacidad de crear nuevos cargadores en el sistema. Al crear un cargador, el administrador puede definir sus atributos y configuraciones específicas, como su ubicación, imagen y descripción. Esto permite al administrador gestionar y ampliar la lista de cargadores disponibles para los usuarios dentro de la aplicación.

- **ARF-04**: el sistema permite al administrador ver todos los cargadores existentes y eliminados existentes en el sistema.

  El administrador tiene la capacidad de ver tanto los cargadores existentes y en uso como aquellos que han sido eliminados del sistema. Esta funcionalidad brinda al administrador una visión completa de todos los cargadores, lo que facilita la supervisión y gestión

de los recursos disponibles.

- **ARF-05**: el sistema permite al administrador eliminar cargadores existentes en el sistema.

  Permite al administrador eliminar cargadores existentes en el sistema. Al eliminar un cargador, se actualiza la disponibilidad de los recursos y se garantiza que ya no esté disponible para nuevas reservas. Esta funcionalidad proporciona al administrador el control necesario para gestionar los cargadores y ajustar su disponibilidad según sea necesario.

- **ARF-06**: el sistema permite al administrador modificar los datos de los cargadores existentes en el sistema.

  El administrador tiene la capacidad de realizar modificaciones en los datos de los cargadores existentes en el sistema. Estas modificaciones pueden incluir cambios en la ubicación, nombre, descripción, imagen o cualquier otra información relevante asociada a los cargadores. Esto permite al administrador mantener actualizada y precisa la información de los cargadores dentro del sistema.

- **ARF-07**: el sistema permite al administrador ver todos los tiquets de incidencias existentes en el sistema.

  Establece que el administrador tiene la capacidad de visualizar todos los tiquets de incidencias registrados en el sistema. Al ver todos los tiquets de incidencias, el administrador puede realizar un seguimiento de los problemas reportados por los usuarios y tomar las acciones correspondientes para su resolución.

  - **ARF-07.1**: el sistema permite al administrador filtrar los tiquets de incidencias existentes el sistema por resuelto o no resulto.

    El administrador tiene la capacidad de aplicar un filtro para mostrar solo los tiquets de incidencias que han sido resueltos o aquellos que aún no han sido resueltos. Esto permite al administrador priorizar y gestionar eficientemente las incidencias pendientes y garantizar una resolución oportuna.

  - **ARF-07.2**: el sistema permite al administrador filtrar los tiquets de incidencias existentes el sistema por *ID* de tiquet.

    proporciona al administrador la capacidad de buscar un tiquet de incidencia es-

pecífico a partir de su identificador. Al aplicar este filtro, el administrador puede buscar y acceder rápidamente a un tiquet de incidencia específico, lo que facilita el seguimiento y la resolución de problemas individuales.

- **ARF-07.3**: el sistema permite al administrador ordenar los tiquets de incidencia por orden ascendiente o descendiente de fecha de creación.
  El administrador tiene la capacidad de ordenar los tiquets de incidencia en función de su fecha de creación. Puede optar por mostrarlos en orden ascendente (desde los más antiguos) o descendente (desde los más recientes). Esto facilita la organización y el seguimiento de los tiquets de incidencia en función de su cronología.

- **ARF-08**: el sistema permite al administrador resolver una incidencia escribiendo o no un mensaje para el usuario.
  Permite al administrador resolver una incidencia registrada en el sistema. El administrador tiene la opción de proporcionar un mensaje explicativo o de confirmación al usuario afectado. Al resolver una incidencia, se indica que se ha tomado acción sobre el problema reportado y se brinda una respuesta o solución al usuario correspondiente.

- **ARF-09**: el sistema permite al administrador ver estadísticas de uso y coste del sistema referentes a las reservas.
  El administrador tiene acceso a estadísticas relacionadas con el uso y el costo del sistema en lo que respecta a las reservas realizadas. Esto incluye información sobre el número total de reservas, la utilización de los cargadores, los tiempos promedio de reserva, los costos asociados, entre otros datos relevantes. Estas estadísticas proporcionan al administrador una visión general del rendimiento y la eficiencia del sistema.

- **ARF-10**: el sistema permite al administrador ver todos los registros o *logs* del sistema.
  El administrador tiene la capacidad de acceder y visualizar todos los registros o *logs* generados por el sistema. Estos registros contienen información detallada sobre las actividades, eventos y transacciones realizadas en el sistema. Al ver los registros del sistema, el administrador puede monitorizar la actividad, identificar posibles problemas o anomalías y realizar análisis de seguimiento.

  - **ARF-10.1**: el sistema permite al administrador descargar los registros del sistema

en formato *JavaScript Object Notation (JSON)*.

El administrador tiene la opción de descargar los registros del sistema en formato *JSON*. Esto permite la exportación de los registros para su análisis posterior, almacenamiento o integración con otras herramientas o sistemas.

- **ARF-10.2**: el sistema permite al administrador filtrar los registros del sistema por *ID* de usuario.

  Proporciona al administrador la capacidad de aplicar un filtro para visualizar únicamente los registros relacionados con un usuario específico, identificado por su *ID* de usuario. Al filtrar los registros por *ID* de usuario, el administrador puede analizar de manera más precisa las actividades y acciones realizadas por usuarios individuales.

4.1.4. Requisitos no funcionales de *Admin-CSSC*

- **ARNF-01**: para crear un nuevo cargador en la aplicación, debe existir previamente un cargador con el *ID* de *OCPP* que se vaya a utilizar, en la latitud y longitud exactas.

  Establece una condición para la creación de nuevos cargadores en la aplicación. Antes de poder crear un cargador, es necesario que ya exista un cargador registrado en el sistema *OCPP* con el mismo *ID* de *OCPP* que se va a utilizar. Además, la ubicación del nuevo cargador debe coincidir exactamente con la latitud y longitud del cargador. Esta restricción garantiza que los cargadores se creen de manera coherente y evita duplicados o inconsistencias en la información de los cargadores.

- **ARNF-02**: el sistema guarda la sesión del usuario a través de *cookies*.

  Se establece que el sistema utiliza *cookies* para almacenar y gestionar la sesión del usuario. Las *cookies* son pequeños archivos de texto que se almacenan en el navegador del usuario y se utilizan para identificar al usuario y mantener su sesión activa durante su interacción con la aplicación. Al utilizar *cookies* para gestionar las sesiones, el sistema puede proporcionar una experiencia de usuario personalizada y permitir al usuario acceder a funciones y recursos específicos según su estado de inicio de sesión.

- **ARNF-03**: el sistema sólo permite la entrada de usuarios de tipo administrador a la aplicación.

Establece una restricción en el acceso a la aplicación. Solo se permite la entrada y autenticación de usuarios con un tipo de usuario específico, en este caso, el tipo administrador. Esto significa que solo aquellos usuarios que hayan sido designados como administradores tendrán la capacidad de iniciar sesión y acceder a las funciones y características destinadas a los administradores. Esta medida de seguridad garantiza que solo los usuarios autorizados tengan acceso a las funcionalidades y capacidades de administración del sistema.

Se ruega al lector consultar el anexo C para revisar los diagramas de casos de uso de los requisitos más relevantes del sistema.

4.2. Diseño de la arquitectura y flujos de interacción

4.2.1. Diseño de la arquitectura

Para el presente trabajo de fin de grado se han desarrollado dos aplicaciones *full-stack*, *Control-CSSC* y *Admin-CSSC*, distintos frontales para un mismo servidor, las cuales siguen la misma arquitectura general.

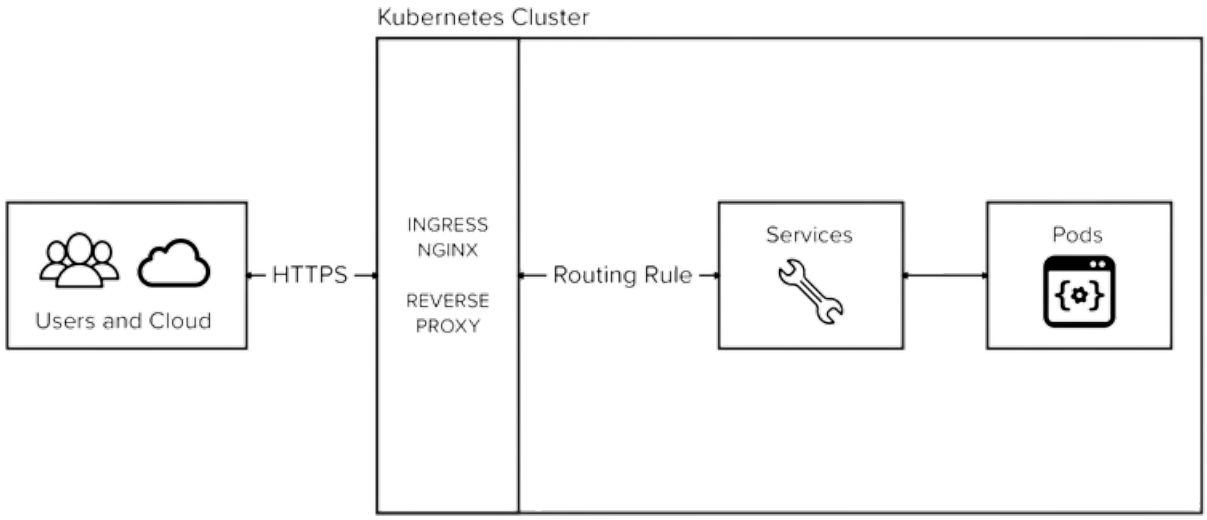

Figura 16: Arquitectura general de las aplicaciones

Desde un punto de vista situado en alto nivel la aplicación se encuentra encapsulada en su totalidad en un clúster de *Kubernetes* [37], el cual nos permite crear un *proxy* inverso (el cual

es un servidor intermediario que actúa como un punto de entrada para los clientes y redirige las solicitudes a los servidores internos correspondientes) y así garantizar el uso de *Hyper Text Transfer Protocol Secure (HTTPS)* en toda la aplicación de cara al usuario, véase figura 16. Así pues, tanto el usuario como las *SaaS* usadas, *Cloudinary* y *MongoDB Atlas*, se comunicarán exclusivamente en *HTTPS* con el clúster, especializando el caso de *MongoDB Atlas*, el cual se comunica por un puerto propietario a través del *driver* usado por los microservicios del *backend*. Entonces, una vez atravesado el *proxy* inverso la conexión se enruta a través de reglas definidas para acceder a los servicios.

En el marco de este proyecto, se ha hecho necesario utilizar ciertos servicios *SaaS* debido a las limitaciones de recursos disponibles en la máquina local. No obstante, en vista de la crítica importancia de la aplicación, se recomienda encarecidamente desplegarla en servicios *cloud* privados.

El uso de *SaaS* ha proporcionado una solución temporal y conveniente para satisfacer las necesidades del proyecto en términos de escalabilidad, disponibilidad y funcionalidad. Sin embargo, es importante reconocer que el uso de servicios en la nube pública puede plantear ciertos riesgos en cuanto a la seguridad y la confidencialidad de los datos.

En un entorno *cloud* privado, se puede ejercer un mayor control sobre la infraestructura y los recursos utilizados. Esto implica implementar medidas de seguridad adicionales, como cifrado de datos, autenticación de usuarios y auditoría de accesos, lo cual es fundamental para garantizar la integridad y la confidencialidad de la información sensible.

Además, al utilizar servicios *cloud* privados, se tiene la posibilidad de adaptar la infraestructura a las necesidades específicas de la aplicación, asignando recursos de manera eficiente y optimizando el rendimiento. Esto se vuelve particularmente relevante en aplicaciones críticas, donde la disponibilidad y la respuesta rápida son aspectos fundamentales.

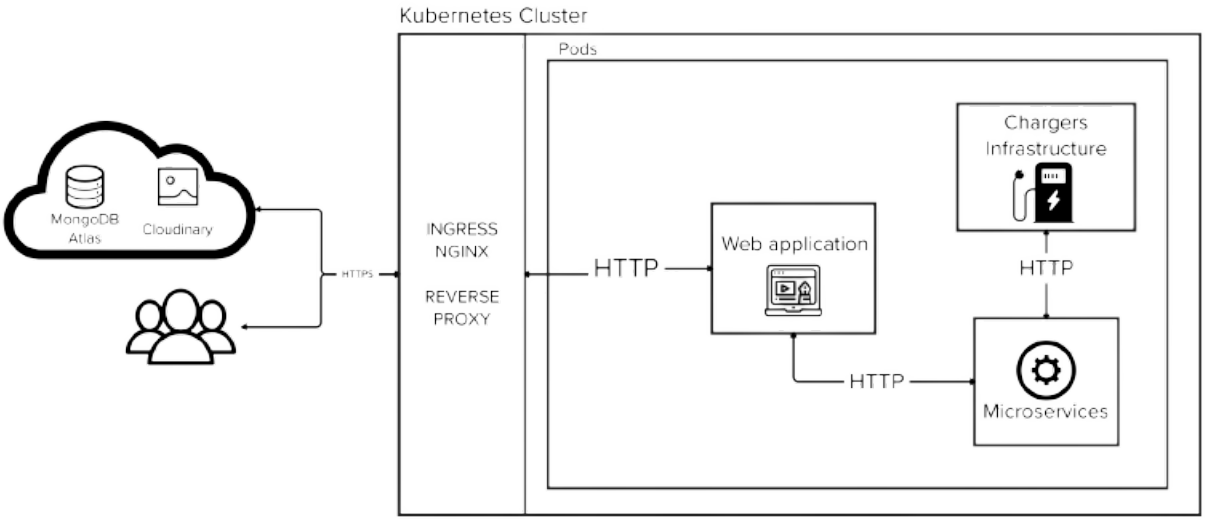

Figura 17: Arquitectura interna de los *pods*

Indagando en la arquitectura de los *pods*, concepto que será explicado en un siguiente apartado, se encuentra la arquitectura interna de la aplicación, dividida en la ya convencional arquitectura binaria de *frontend* y *backend*, véase figura 17.

Con respecto a la arquitectura del *backend*, se ha seguido una basada en microservicios, siendo ésta un enfoque de desarrollo de aplicaciones de software que se compone de un conjunto de pequeños servicios autónomos que brindan funcionalidades de negocio completas. En este modelo, cada microservicio es un código que puede estar desarrollado en un lenguaje de programación distinto y cumple una función específica. Los microservicios se comunican entre sí mediante interfaces de programación de aplicaciones *(API)*.

La decisión de no utilizar una arquitectura monolítica se ha tomado debido a las múltiples desventajas que presenta. En primer lugar, la alta complejidad que se presenta a la hora de mantener la aplicación. Si el monolito no está diseñado de forma modular y desacoplada, a medida que crece la aplicación, el coste del mantenimiento del código puede aumentar significativamente debido a la complejidad técnica (deuda técnica).

Además, la escalabilidad también se ve afectada por esta arquitectura. Para escalar un monolito, se debe desplegar toda la aplicación, lo que implica un aumento en la infraestructura necesaria, incluso cuando la escalabilidad solo es necesaria para unos pocos casos de uso. Además, la aplicación debe ser diseñada para ser desplegada en *clustering*, lo que significa que debe

estar libre de sesiones pegajosas y procesos concurrentes que se solapen.

Otra limitación de la arquitectura monolítica es que está atada a una única tecnología o *framework*. Cuando se desarrolla un monolito, se establece una tecnología y todo el equipo queda atado a dicha tecnología, lo que imposibilita la utilización de diferentes soluciones tecnológicas en función de cada caso de uso y sus requisitos.

Por último, pero no menos importante, la arquitectura monolítica también presenta un fallo crítico en términos de la integridad de la aplicación: el punto único de fallo. Si algo falla en el monolito, podría afectar a toda la aplicación, lo que se conoce como un *Single Point of Failure (SPOF)*. [24]

Por tanto, la adopción de una arquitectura basada en microservicios ha permitido la creación de infraestructuras de tecnología de la información más adaptables y flexibles, ya que modificar un solo servicio no requiere la alteración del resto de la infraestructura. Cada uno de los servicios puede ser implementado y modificado sin afectar a otros servicios o aspectos funcionales de la aplicación. Así pues, cada uno de los microservicios se comunica de manera bidireccional con el *frontend* de la aplicación, con la base de datos y con la infraestructura de los cargadores. [25]

Por otro lado, una parte esencial de este trabajo de fin de grado es la conexión a las estaciones de carga usando el protocolo *OCPP*, en el cual ya se ha especificado el uso de *OCPP-ASGI*, el cual posee la estructura visible en la figura 18.

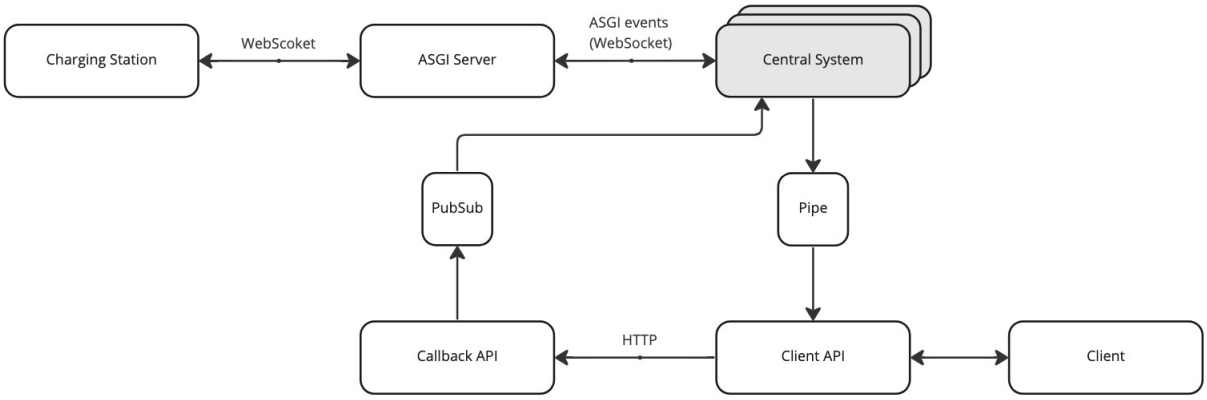

Figura 18: Arquitectura de *OCPP-ASGI*

En la presente arquitectura, se ha designado al cliente como los *backends* de las aplicaciones *Control-CSSC* y *Admin-CSSC*, los cuales se comunican directamente con el sistema *OCPP-ASGI*. Las solicitudes son enviadas a través de la *API* del cliente, la cual ofrece ciertas rutas para llevar a cabo acciones tanto en la central de cargadores como en las estaciones de carga. Una vez que la solicitud es recibida por la *API* del cliente, ésta se dirige hacia la *API* de devolución de llamada (*callback*), la cual tiene como función principal evitar que el sistema central tenga que estar consultando continuamente la red en busca de eventos entrantes. En lugar de esto, la *API* de devolución de llamada envía eventos al sistema central a través de un servicio de mensajería asíncrono denominado *Publish/Subscribe Messaging (PubSub)*, el cual facilita la comunicación de eventos entre servicios de forma asíncrona, evitando que los datos sean visibles para el cliente [26].

El sistema central se encarga de procesar todos los eventos entrantes, y ya sea que los procese y devuelva la respuesta a la *API* del cliente mediante una tubería o *pipe*, o que requiera enviar el evento a una estación de carga mediante un servidor *ASGI* intermedio. La comunicación entre el servidor intermedio y la estación de carga se realiza a través de un *WebSocket*, lo que permite abrir una sesión de comunicación interactiva entre cliente y servidor, en este caso, sistema central y estación de carga. De esta manera, se pueden recibir respuestas controladas por eventos sin necesidad de consultar al servidor para obtener una respuesta. Una vez que la estación de carga recibe y procesa el evento, éste se envía de vuelta al sistema central, y éste a su vez lo envía a la *API* del cliente.

4.2.2. Flujos de interacción

En este apartado se proponen los flujos de interacción de las aplicaciones web desarrolladas durante el proyecto: *Control-CSSC* y *Admin-CSSC* usando *Interaction Flow Modeling Language (IFML)*, el cual es un lenguaje de modelado estandarizado utilizado para describir la interacción entre usuarios y sistemas informáticos a través de interfaces de usuario. Proporciona una notación gráfica y semántica para representar de manera precisa y estructurada los flujos de interacción, eventos, acciones y elementos de visualización relacionados con una interfaz. El flujo de interacción de la página web se describe a través de una serie de pasos lógicos que los usuarios siguen para interactuar con la interfaz y acceder a diferentes funcionalidades

IFML permite modelar tanto la estructura como el comportamiento de una interfaz de usua-

rio, abarcando diferentes aspectos clave de la interacción. A través de su sintaxis formal, *IFML* define elementos y relaciones que ayudan a capturar y representar la lógica subyacente de cómo los usuarios interactúan con un sistema. Para facilitar la lectura de los diagramas *IFML*, se explican a continuación los distintos componentes que forman este tipo de diagramas:

- ***View Container***: se trata de un elemento que actúa como un contenedor visual para otros elementos de interfaz de usuario en un diagrama de flujo de interacción. Un *View Container* proporciona una estructura organizativa y jerárquica para los elementos de la interfaz de usuario, lo que permite agruparlos y organizarlos de manera lógica.

  Un *View Container* puede representar una página, una ventana, un panel o cualquier otro tipo de contenedor visual que alberga elementos de la interfaz de usuario, como botones, campos de entrada, etiquetas, etc. Actúa como un recipiente que encapsula y define el ámbito de visibilidad y la ubicación de los elementos contenidos.

  Posee dos etiquetas: *[L]* y *[D]*, si un *View Container* está marcado con la etiqueta *[L]*, abreviación de *landmark*, significa que desde cualquier otro *View Container* se puede acceder a él sin tener que definir un flujo al mismo de manera explícita, es decir, es la página principal. Esta analogía se puede comparar con el logotipo de las aplicaciones web ubicado en la barra de navegación, donde al hacer clic se regresa a la página principal. Por otro lado, si un *View Container* está marcado con la etiqueta *[D]*, abreviación de *default*, significa que es la página principal, aquella que se mostrará al usuario cuando acceda por primera vez a la aplicación web.

- ***Form View Component***: el componente *Form* se refiere a un elemento utilizado para representar un formulario o un conjunto de campos de entrada en un diagrama de flujo de interacción. Este componente permite al usuario ingresar datos o realizar selecciones mediante diferentes tipos de campos, como casillas de verificación, botones de opción, campos de texto, menús desplegables, entre otros.

- ***List View Component***: el componente de vista *List View* se refiere a un elemento utilizado para representar una vista de lista o una visualización tabular de datos en un diagrama de flujo de interacción. El componente de permite mostrar una colección de elementos de datos en forma de filas o entradas en una estructura tabular.

- ***Event***: representa un evento desencadenante que ocurre durante la interacción del usuario con la aplicación. Un evento puede ser una acción realizada por el usuario, como hacer clic en un botón, ingresar texto en un campo de entrada, seleccionar una opción de un menú, etc. También puede ser un evento generado por el sistema, como una notificación o un temporizador. Se colocan en los extremos de los *View Container* y en los componentes de tipo *Form* si son acciones desencadenadas por el usuario, si por el contrario son acciones realizadas por el sistema, éstos van posicionados en los extremos del componente *Action*, el cual se explica a continuación.

- ***Action***: representa una acción o respuesta que se desencadena como resultado de un evento. Una acción puede ser cualquier acción realizada por el sistema en respuesta a un evento, como mostrar una nueva página, ejecutar una función, realizar una validación de datos, enviar un mensaje, entre otras.

Así pues, como se muestra en la figura 54, la aplicación web *Control-CSSC* ofrece flujos de interacción acorde a los requisitos requeridos para su correcto funcionamiento, al igual que lo expuesto en la figura 55 para la aplicación web *Admin-CSSC*.

5

Tecnologías usadas

5.1. Contenedores

En el presente apartado se proporcionará una explicación detallada y rigurosa sobre las tecnologías de contenedores y orquestación utilizadas en el desarrollo del proyecto. Se abordarán aspectos fundamentales como los contenedores y su papel en la virtualización ligera, así como las tecnologías de orquestación que permiten gestionar y coordinar eficientemente la ejecución de múltiples contenedores. Se examinarán en profundidad las herramientas y plataformas utilizadas destacando sus características, funcionalidades y beneficios en términos de escalabilidad, disponibilidad y administración de aplicaciones distribuidas.

5.1.1. *Docker*

Docker es una plataforma de virtualización de aplicaciones de código abierto que permite a los desarrolladores y administradores de sistemas crear, distribuir y ejecutar aplicaciones de manera eficiente y portátil. Proporciona un entorno aislado y ligero, conocido como contenedor, que encapsula una aplicación y todas sus dependencias, incluidas bibliotecas, herramientas y configuraciones, en una unidad estandarizada.

Los contenedores *Docker* son instancias de imágenes *Docker*, que son plantillas de solo lectura que contienen todo lo necesario para ejecutar una aplicación, como el sistema operativo, las bibliotecas y el código fuente. Cada contenedor se ejecuta de forma aislada en un entorno sandbox y utiliza recursos del sistema de manera eficiente, lo que permite ejecutar múltiples contenedores en un mismo host físico.

Docker utiliza la tecnología de virtualización a nivel de sistema operativo, también conocida como contenedorización, para brindar un enfoque ligero y rápido para el despliegue de aplicaciones. A diferencia de las máquinas virtuales tradicionales, los contenedores no requieren

un sistema operativo completo para cada instancia, lo que reduce la sobrecarga y proporciona tiempos de inicio y rendimiento más rápidos.

La popularidad de *Docker* se debe a varias ventajas clave. En primer lugar, ofrece portabilidad, ya que los contenedores *Docker* se pueden ejecutar en cualquier entorno compatible con *Docker*, independientemente del sistema operativo subyacente o la infraestructura de hardware. Esto facilita la migración de aplicaciones entre diferentes entornos, como máquinas locales, servidores en la nube o clústeres de contenedores.

En segundo lugar, *Docker* simplifica el proceso de desarrollo y despliegue de aplicaciones al proporcionar un entorno consistente y reproducible. Los desarrolladores pueden crear imágenes *Docker* con todas las dependencias necesarias y compartir fácilmente esas imágenes con otros miembros del equipo, lo que garantiza que todos trabajen en el mismo entorno. Además, *Docker* facilita la implementación y escalado de aplicaciones en producción, ya que los contenedores se pueden administrar y orquestar de manera eficiente utilizando herramientas como *Docker Compose* o *Kubernetes*, explicado a continuación. [33]

Para crear imágenes de *Docker* nos apoyamos en archivos del tipo *Dockerfile*, los cuales actúan como un conjunto de instrucciones paso a paso que le indica a *Docker* cómo construir la imagen. Cada instrucción en el *Dockerfile* realiza una tarea específica, como copiar archivos, instalar dependencias, configurar variables de entorno y exponer puertos de red. Estas instrucciones se ejecutan en secuencia para crear una imagen final coherente y reproducible.

El formato del *Dockerfile* es sencillo y sigue una estructura básica. Comienza con una línea de instrucción FROM, que especifica la imagen base a partir de la cual se construirá la nueva imagen. A continuación, se pueden incluir otras instrucciones como COPY, RUN, ENV, EXPOSE y CMD, entre otras, que definen las acciones a realizar durante la construcción de la imagen.

Una vez que se ha creado el *Dockerfile*, se utiliza el comando docker build junto con la ubicación del *Dockerfile* para iniciar el proceso de construcción de la imagen. *Docker* interpreta las instrucciones del *Dockerfile* y genera una nueva imagen que encapsula la aplicación y su entorno, la cual es usada en la orquestación llevada a cabo en *Kubernetes*, explicado a continación. [35]

5.1.2. *Kubernetes*

Kubernetes es una plataforma de orquestación de contenedores de código abierto desarrollada por *Google* que facilita la gestión, escalabilidad y despliegue de aplicaciones en contenedores a gran escala. Proporciona un entorno para automatizar y coordinar la implementación, escalado y administración de aplicaciones contenerizadas en clústeres de máquinas físicas o virtuales.

En *Kubernetes*, los contenedores o *pods* son la unidad fundamental de implementación. Un contenedor es una forma ligera y portátil de empaquetar una aplicación con todas sus dependencias y configuraciones, lo que garantiza que la aplicación se ejecute de manera coherente en diferentes entornos. *Kubernetes* permite administrar eficientemente un gran número de contenedores distribuidos en múltiples nodos o máquinas.

El objetivo principal de *Kubernetes* es proporcionar una plataforma para la orquestación de contenedores que ofrezca características como alta disponibilidad, escalabilidad, recuperación ante fallos y distribución de carga. Está diseñado para manejar de forma automática y dinámica la asignación de recursos, la monitorización, el despliegue y la escalabilidad de las aplicaciones contenerizadas.

Kubernetes utiliza una arquitectura de clúster compuesta por un conjunto de nodos de trabajo y un nodo maestro. Los nodos de trabajo son las máquinas donde se ejecutan los contenedores, mientras que el nodo maestro es responsable de coordinar y gestionar todo el clúster. El nodo maestro incluye componentes clave que colaboran para proporcionar funcionalidades avanzadas de orquestación no usadas en este proyecto.

Entre las principales características de *Kubernetes* se encuentran:

- **Escalabilidad automática**: permite escalar automáticamente los contenedores según la demanda de la aplicación, ajustando el número de réplicas en función de las métricas definidas.

- **Gestión de almacenamiento**: proporciona una forma flexible de gestionar el almacenamiento de datos para los contenedores, como volúmenes persistentes y almacenamiento en la nube.

- **Despliegue y actualización de aplicaciones**: facilita el despliegue y actualización de aplicaciones de forma declarativa, lo que permite especificar el estado deseado y dejar que *Kubernetes* se encargue de mantener ese estado.

- **Descubrimiento de servicios**: proporciona un mecanismo para descubrir y acceder a los servicios de una aplicación dentro del clúster mediante una dirección *Internet Protocol (IP)* y un nombre *Domain Name System (DNS)*.

- **Tolerancia a fallos**: ofrece mecanismos de recuperación ante fallos, como la reprogramación automática de contenedores en caso de fallos y la distribución de carga entre los nodos sanos.

Kubernetes se ha convertido en una herramienta popular para la gestión de aplicaciones en contenedores, debido a su enfoque flexible, escalable y altamente automatizado. Es utilizado ampliamente en entornos de producción para implementar y administrar aplicaciones en contenedores en una variedad de infraestructuras, incluyendo nubes públicas, privadas e híbridas.

Centrando el enfoque en el desarrollo del presente proyecto, para llevar a cabo la integración de *Kubernetes* a partir de las imágenes de *Docker* previamente explicadas se ha llevado a cabo el uso de diferentes bloques de configuración propios de *Kubernetes*:

- ***Deployment***: este fichero es utilizado para definir y configurar el despliegue de una aplicación en *Kubernetes*. En este archivo se especifican los detalles del contenedor, como la imagen a utilizar, los recursos asignados, las variables de entorno y las opciones de escalado. Además, permite definir la estrategia de actualización, como el número de réplicas deseadas y las políticas de tolerancia a fallos. Esta configuración es esencial para desplegar y gestionar la aplicación de forma declarativa en el clúster de *Kubernetes*.

- ***Service***: este archivo se utiliza para definir un servicio en *Kubernetes*. Un servicio es un objeto que expone una aplicación o conjunto de *pods* dentro del clúster, permitiendo el acceso a ellos desde otros componentes. Aquí se especifican detalles como el tipo de servicio *(ClusterIP, NodePort, LoadBalancer)*, los puertos a exponer y la selección de *pods* a los que el servicio está dirigido. Este archivo es fundamental para establecer la comu-

nicación entre los distintos componentes de una aplicación y permitir el descubrimiento de servicios dentro del clúster.

- ***Ingress***: esta configuración se utiliza para configurar las reglas de enrutamiento y acceso externo a los servicios dentro del clúster de *Kubernetes*. *Ingress* actúa como un controlador de tráfico y permite redirigir las solicitudes a diferentes servicios en función de reglas específicas, como la ruta del *Uniform Resource Identifier (URI)* o el nombre del *host*. Aquí se definen las reglas de enrutamiento, los certificados *Secure Socket Layer (SSL)/TLS* y otras opciones de configuración relacionadas con la gestión del tráfico externo hacia los servicios del clúster.

- ***Kustomize***: esta herramienta está incluida en *Kubernetes* para la gestión y personalización de configuraciones en aplicaciones. *Kustomize* permite modificar y personalizar archivos de configuración sin tener que editar directamente los archivos originales. Aquí se especifican las transformaciones y sobreescrituras que se aplicarán a los recursos definidos en otros archivos de configuración. Esto incluye la modificación de nombres, la adición o eliminación de etiquetas, la configuración de variables de entorno y otras personalizaciones necesarias para ajustar la configuración de la aplicación a diferentes entornos o requisitos específicos. [37]

Así pues, se ha utilizado los *Deployments* para arrancar todas las instancias tanto de los distintos microservicios pertenecientes al *backend* como de ambos *frontends*, además de la base de datos. Por otro lado, se han usado los *Services* para llevar a cabo la exposición dentro del clúster de todas las instancias para permitir la intercomunicación entre ellas. Luego, con *Ingress* se ha permitido el acceso a los *frontends* a través de sus respectivas *Uniform Resource Locators (URL)* añadiendo una capa de seguridad con el uso de certificados *SSL/TLS*. Finalmente, se ha incluido el uso de *Kustomization* para agrupar en un sólo fichero la gestión de todo el clúster de *Kubernetes* y tener a un simple vistazo todas las instancias que se van a requerir en el clúster.

Así pues, se han utilizado los *Deployments* para iniciar y gestionar todas las instancias de los distintos microservicios pertenecientes al *backend*, así como de ambos *frontends*, y también de la base de datos.

Por otro lado, se han empleado los *Services* para exponer y facilitar la comunicación entre las diferentes instancias dentro del clúster, tal y como se indica en la arquitectura de la aplicación.

Luego, con el uso de *Ingress*, se ha habilitado el acceso a los *frontends* a través de sus respectivas *URL*. *Ingress* actúa como un *proxy* reverso y permite enrutar las solicitudes hacia los *frontends* dentro del clúster. Además, se ha añadido una capa de seguridad adicional mediante el uso de certificados *SSL/TLS*, asegurando así la confidencialidad y la integridad de las comunicaciones.

Finalmente, se ha incorporado el uso de *Kustomization* para centralizar y gestionar de manera más eficiente la configuración del clúster de *Kubernetes*.

5.2. Servidor

En el presente apartado se detallarán las tecnologías empleadas en el desarrollo del *backend* de la aplicación, con el objetivo de proporcionar una visión exhaustiva y formal sobre las herramientas y *frameworks* utilizados en la capa de servidor. Se abordarán aspectos clave como el lenguaje de programación, los *frameworks* y bibliotecas empleadas. Mediante esta exposición detallada, se busca brindar una comprensión integral de las tecnologías que respaldan el funcionamiento y la implementación eficiente del *backend*, sentando así las bases para un análisis exhaustivo y una evaluación crítica de su rendimiento y su idoneidad en el contexto específico del proyecto.

5.2.1. *Python*

Python es un lenguaje de programación de alto nivel, interpretado y multipropósito. Fue creado por Guido van Rossum en 1991 y se ha vuelto extremadamente popular en la comunidad de desarrollo de software debido a su simplicidad, legibilidad y versatilidad. *Python* se caracteriza por su sintaxis clara y concisa, lo que facilita la escritura y comprensión del código, así como su mantenimiento a largo plazo.

Python se basa en una filosofía de diseño conocida como *El Zen de Python* (también conocido como el *PEP 20*) [39], que promueve la legibilidad, la simplicidad y la claridad del código. El lenguaje está diseñado para ser fácil de aprender y usar, lo que lo hace especialmente adecuado

tanto para principiantes como para programadores experimentados.

Python ofrece una amplia gama de características y funcionalidades, lo que lo convierte en un lenguaje muy versátil. Es utilizado en una variedad de dominios, como desarrollo web, análisis de datos, inteligencia artificial, automatización de tareas, *scripting*, entre otros. Una de las ventajas clave de *Python* es su gran cantidad de bibliotecas y módulos de terceros, lo que permite a los desarrolladores aprovechar una amplia gama de funcionalidades preexistentes y acelerar el desarrollo de aplicaciones.

En términos de sintaxis, *Python* se destaca por su enfoque en la legibilidad del código. Utiliza una indentación significativa (espacios en blanco) en lugar de llaves o palabras clave para definir bloques de código. Esto fomenta una estructura clara y coherente, y evita la necesidad de caracteres especiales o sintaxis complicada. Además, *Python* tiene una amplia biblioteca estándar que proporciona funcionalidades esenciales y comunes, lo que simplifica el desarrollo de aplicaciones al no tener que empezar desde cero.

5.2.2. *FastAPI*

FastAPI es un moderno *framework* de desarrollo web de alto rendimiento diseñado para construir aplicaciones *API* en *Python*. Fue creado por Sebastián Ramírez y se basa en las capacidades de tipado estático y en las características asíncronas de *Python 3.7* en adelante. *FastAPI* se ha vuelto muy popular debido a su eficiencia, facilidad de uso y su capacidad para manejar un alto volumen de solicitudes de manera concurrente.

Una de las principales características distintivas de *FastAPI* es su enfoque en el tipado estático, utilizando las anotaciones de tipos introducidas en *Python 3*. Las anotaciones de tipos permiten especificar los tipos de datos esperados en las rutas, los parámetros y las respuestas de las *API*, lo que brinda una mayor claridad y seguridad al desarrollo de aplicaciones. Además, *FastAPI* utiliza el sistema de tipos de *Python* para realizar validaciones automáticas en tiempo de ejecución, ayudando a evitar errores y facilitando el mantenimiento del código.

FastAPI también aprovecha las características asíncronas de *Python*, lo que significa que puede manejar solicitudes de forma concurrente y aprovechar la eficiencia de las operaciones asíncronas. Esto se logra utilizando la sintaxis de corutinas de Python y la compatibilidad con el marco de trabajo *asyncio*, lo que permite que las aplicaciones *FastAPI* sean altamente escalables y eficientes en términos de rendimiento.

Otra característica importante de *FastAPI* es su sistema de documentación automática e interactiva. *FastAPI* genera automáticamente documentación detallada para las *API* creadas, utilizando la información proporcionada por las anotaciones de tipos y los comentarios en el código. Esta documentación es generada en formato *OpenAPI* (anteriormente conocido como *Swagger*), lo que facilita su visualización y exploración, y permite a los desarrolladores probar las *API* directamente desde la documentación generada.

Además, *FastAPI* ofrece una sintaxis declarativa y sencilla para definir las rutas y las rutas o *endpoints* de las *API*. Mediante el uso de decoradores, los desarrolladores pueden especificar las operaciones *HTTP* admitidas (como GET, POST, PUT, DELETE), las rutas, los parámetros, los encabezados y las respuestas de manera intuitiva y eficiente. Esto facilita el desarrollo rápido de aplicaciones *API* con una estructura clara y coherente.

FastAPI también es compatible con otras bibliotecas y herramientas de *Python*, lo que permite integrarlo fácilmente con bases de datos, sistemas de autenticación, cachés y otras funcionalidades adicionales. Además, cuenta con una activa comunidad de desarrolladores que contribuye con bibliotecas y complementos para ampliar aún más las capacidades del *framework*. [42]

5.2.3. *AsyncIO*

AsyncIO, también conocido como *Coroutines and Asynchronous IO* (Entrada/Salida Asíncrona y Corrutinas), es un módulo integrado en la biblioteca estándar de *Python* a partir de la versión 3.4. Proporciona un conjunto de herramientas y características para facilitar la programación asíncrona y concurrente en *Python*.

El objetivo principal de *AsyncIO* es permitir la escritura de código asíncrono, donde múltiples tareas pueden ejecutarse de forma concurrente sin bloquear el hilo principal de ejecución. Esto es especialmente útil en situaciones en las que se requiere realizar operaciones de entrada/salida intensivas, como solicitudes a bases de datos, llamadas a servicios web o lecturas/escrituras en archivos.

El paradigma de programación asíncrona se basa en el uso de corutinas, que son funciones especiales que pueden suspenderse y reanudarse en puntos específicos de ejecución. Con *AsyncIO*, se pueden definir corutinas utilizando la sintaxis `async def`, lo que permite una escritura más concisa y clara del código asíncrono. Estas corutinas pueden invocarse de manera

asíncrona utilizando la sintaxis `await`, lo que permite que otras tareas continúen su ejecución mientras se espera la finalización de una operación.

AsyncIO proporciona un bucle de eventos *(event loop)* que actúa como un administrador de tareas y coordina la ejecución de las corutinas. Este bucle de eventos se encarga de programar y ejecutar las corutinas de manera eficiente, asegurando que las tareas se ejecuten de forma concurrente y sin bloqueos. Además, el bucle de eventos también maneja la gestión de eventos, como eventos de red, entrada/salida o temporizadores.

Una de las características clave de *AsyncIO* es su compatibilidad con operaciones de entrada/salida no bloqueantes. Esto significa que, en lugar de esperar de forma bloqueante a que una operación de entrada/salida se complete, se pueden realizar múltiples operaciones de entrada/salida de manera concurrente, aprovechando los recursos disponibles de manera eficiente. Esto mejora significativamente el rendimiento y la capacidad de respuesta de las aplicaciones asíncronas.

Otra característica importante de *AsyncIO* es la capacidad de realizar tareas de manera concurrente utilizando el concepto de tareas o *tasks*. Una tarea representa una unidad de trabajo asíncrono y puede ser creada a partir de una corutina. Las tareas se pueden ejecutar de manera simultánea y *AsyncIO* se encarga de administrar su programación y ejecución de forma eficiente.

AsyncIO también proporciona herramientas para el manejo de excepciones y el control de flujo en entornos asíncronos. Permite capturar y manejar excepciones específicas de corutinas, así como implementar patrones de control de flujo como *try-except-finally* y *for-loops* en el contexto asíncrono. [44]

5.2.4. *Motor*

Motor es una biblioteca de *Python* diseñada específicamente para interactuar con bases de datos *MongoDB* utilizando el paradigma de programación asíncrona. Fue desarrollada por la comunidad de *Python* para aprovechar las capacidades de *AsyncIO* y proporcionar una capa de abstracción eficiente y fácil de usar para realizar operaciones de base de datos asíncronas.

MongoDB es una base de datos *NoSQL* muy popular que se basa en documentos en lugar de en tablas y filas. Permite el almacenamiento flexible y escalable de datos y es ampliamente utilizado en aplicaciones modernas. Sin embargo, la biblioteca cliente oficial de *MongoDB* para

Python no era compatible con *AsyncIO* y no ofrecía una experiencia asíncrona completa. Fue en este contexto que surgió *Motor*.

Motor utiliza la biblioteca oficial de *Python* para *MongoDB*, *PyMongo*, como base, pero lo envuelve con funcionalidades adicionales para admitir operaciones asíncronas. *Motor* permite realizar operaciones de lectura y escritura en la base de datos sin bloquear el hilo principal de ejecución, lo que resulta en una mayor capacidad de respuesta y eficiencia en las aplicaciones.

Una de las ventajas clave de utilizar *Motor* es su integración perfecta con el paradigma asíncrono de *Python*. Permite definir corutinas y utilizar la sintaxis `await` para esperar la finalización de las operaciones de base de datos, lo que facilita la escritura de código asíncrono claro y conciso. Además, *Motor* aprovecha el bucle de eventos de *AsyncIO* para coordinar las operaciones de manera eficiente y permitir la ejecución concurrente de tareas.

Otra característica destacada de *Motor* es su compatibilidad con características avanzadas de *MongoDB*, como índices, agregaciones y réplicas. Proporciona métodos y funciones específicas para trabajar con estas características, lo que facilita su uso en aplicaciones que requieren operaciones más complejas en la base de datos.

Además, *Motor* ofrece una capa de abstracción adicional que simplifica la interacción con *MongoDB*. Proporciona una interfaz de programación consistente y fácil de usar para realizar operaciones de *Create, Remove, Update, Delete (CRUD)* en la base de datos, lo que reduce la complejidad y la cantidad de código necesario para realizar tareas comunes.

Motor también se beneficia de la amplia comunidad de *Python* y *MongoDB*, lo que garantiza un soporte y una documentación sólidos. [43]

5.2.5. *PyTest*

PyTest es un marco de pruebas ampliamente utilizado y poderoso para aplicaciones de *Python*. Proporciona un conjunto completo de características y funcionalidades que facilitan la creación, ejecución y análisis de pruebas en *Python*.

Una de las principales ventajas de PyTest es su simplicidad y facilidad de uso. Permite escribir pruebas de manera clara y concisa utilizando una sintaxis sencilla y legible. Esto permite a los desarrolladores enfocarse en el diseño de pruebas efectivas y no en detalles técnicos innecesarios.

PyTest también es altamente flexible y extensible. Permite a los desarrolladores aprovechar

una amplia gama de *plugins* y extensiones para adaptar el marco a las necesidades específicas del proyecto. Este marco también ofrece una amplia gama de características y funcionalidades avanzadas. Esto incluye la capacidad de generar informes detallados de pruebas, gestionar y reutilizar datos de prueba, además de establecer configuraciones específicas para diferentes entornos.

Una característica destacada de *PyTest* es su sistema de descubrimiento automático de pruebas. Esto significa que no es necesario seguir una estructura de directorios o convenciones específicas para identificar y ejecutar las pruebas. *PyTest* busca automáticamente todos los archivos de prueba dentro del proyecto y los ejecuta, lo que hace que el proceso de prueba sea eficiente y sin problemas.

Además, *PyTest* se integra perfectamente con otras herramientas y bibliotecas populares utilizadas en el ecosistema de *Python*, como en este caso, con *FastAPI*. Esto facilita la integración de *PyTest* en el flujo de trabajo de desarrollo y asegura una cobertura de prueba exhaustiva y una entrega continua de software de alta calidad.

Así pues, *PyTest* es un marco de pruebas potente, flexible y fácil de usar para aplicaciones de *Python*. Con su sintaxis clara, funcionalidades avanzadas y capacidad de personalización, *PyTest* se ha convertido en una opción popular entre los desarrolladores para garantizar la calidad y confiabilidad del software desarrollado en *Python*.

5.3. Interfaz de usuario

En el siguiente documento se presentará una explicación detallada y precisa acerca de las tecnologías utilizadas en el desarrollo de la interfaz de usuario o *frontend* de la aplicación. Se abordarán diversos aspectos clave, como los lenguajes de programación y *frameworks* empleados, así como las herramientas y bibliotecas utilizadas para la creación de interfaces interactivas y atractivas. Se explorarán conceptos relacionados con el diseño web, la maquetación de páginas, la interacción con el usuario y la implementación de componentes visuales. Mediante esta exposición detallada, se busca proporcionar una visión completa y rigurosa de las tecnologías empleadas en el *frontend*, sentando las bases para un análisis crítico y una implementación exitosa de la interfaz de usuario.

5.3.1. *Node.js*

Node.js es un entorno de ejecución de código abierto basado en el motor de *JavaScript V8* de *Google*. Fue creado por Ryan Dahl en 2009 con el objetivo de permitir la ejecución de código *JavaScript* fuera del navegador, en el lado del servidor. A diferencia de los entornos de ejecución tradicionales, que utilizan lenguajes como *PHP* o *Java*, *Node.js* utiliza *JavaScript*, un lenguaje ampliamente conocido y utilizado en el desarrollo web.

La principal característica de *Node.js* es su capacidad para realizar operaciones de entrada/salida de manera asíncrona y no bloqueante. Esto significa que puede manejar múltiples solicitudes simultáneamente sin tener que esperar a que una operación se complete antes de procesar la siguiente. Utiliza un modelo de programación basado en eventos y *callbacks*, lo que le permite ser altamente eficiente y escalable en entornos de alto rendimiento.

Una de las ventajas clave de *Node.js* es su arquitectura de un solo subproceso e hilo, lo que significa que puede manejar muchas conexiones concurrentes con un consumo mínimo de recursos. Esto es posible gracias a la naturaleza asíncrona de *JavaScript* y a la capacidad de *Node.js* para realizar operaciones de I/O de manera eficiente, como la lectura y escritura de archivos, las solicitudes y respuestas *HTTP*, y las operaciones de base de datos.

Además de su eficiencia y escalabilidad, *Node.js* cuenta con una amplia y activa comunidad de desarrolladores que contribuyen con módulos y paquetes de código abierto a través de *Node Package Manager (NPM)*. Esto facilita la reutilización de código y acelera el proceso de desarrollo, ya que se pueden aprovechar las numerosas bibliotecas disponibles para agregar funcionalidades adicionales a las aplicaciones.

Node.js es especialmente adecuado para aplicaciones en tiempo real, como aplicaciones de *chat*, juegos en línea y aplicaciones colaborativas, donde la capacidad de manejar múltiples conexiones simultáneas es crucial, siendo así elegido para el desarrollo del presente proyecto.[46]

5.3.2. *Astro*

Astro es una herramienta y marco de trabajo moderno para el desarrollo de aplicaciones web estáticas y dinámicas. Se destaca por su enfoque en la eficiencia, la productividad y la flexibilidad al construir sitios y aplicaciones web.

Astro posee tres

En esencia, *Astro* se basa en la idea de crear componentes reutilizables y componibles que se pueden combinar para construir interfaces de usuario complejas y dinámicas. Está diseñado para aprovechar el poder de las tecnologías web existentes, como *JavaScript* y *Hypertext Markup Language (HTML)*, al tiempo que ofrece una sintaxis simple y clara para el desarrollo.

Una de las principales características de *Astro* es su capacidad para generar sitios y aplicaciones web estáticas y dinámicas. Esto significa que se pueden generar páginas estáticas en tiempo de compilación para obtener un rendimiento óptimo, mientras se conserva la capacidad de agregar interactividad y actualizaciones en tiempo real cuando sea necesario.

Además, *Astro* permite el uso de diferentes lenguajes de plantilla, como *React*, *Vue.js* y *Svelte* el cual ha sido usado en este proyecto, lo que brinda a los desarrolladores la libertad de elegir la tecnología que mejor se adapte a sus necesidades y conocimientos previos. Esto facilita la creación de componentes y la gestión del estado de la aplicación.

Así pues, podemos enumerar algunos beneficios que aporta el uso de *Astro* en el desarrollo web:

- **Eficiencia de rendimiento**: *Astro* utiliza la generación de sitios estáticos en tiempo de compilación para lograr un rendimiento óptimo. Esto significa que las páginas web se pueden generar como archivos *HTML* estáticos, lo que resulta en tiempos de carga más rápidos para los usuarios. Además, *Astro* tiene una capacidad única para combinar elementos estáticos y dinámicos en un solo sitio, lo que permite una experiencia fluida y receptiva.

- **Flexibilidad de enfoque**: *Astro* no impone restricciones estrictas sobre las tecnologías o lenguajes de plantilla que se deben utilizar. Los desarrolladores pueden elegir entre una variedad de lenguajes de plantilla populares según sus preferencias y conocimientos previos. Esto brinda una gran flexibilidad y permite a los desarrolladores utilizar las herramientas y bibliotecas que mejor se adapten a sus necesidades.

- **Desarrollo rápido**: *Astro* está diseñado para facilitar el desarrollo rápido de aplicaciones web. Con su sintaxis clara y concisa, los desarrolladores pueden escribir código de manera eficiente y productiva. Además, *Astro* proporciona una serie de componentes

predefinidos y funciones integradas que aceleran el proceso de desarrollo y permiten la reutilización de código.

- **Compatibilidad con múltiples fuentes de datos**: *Astro* facilita la integración de diferentes fuentes de datos en una aplicación web. Puede conectarse a *API RESTful*, bases de datos y servicios de terceros para obtener y manipular datos de manera eficiente. Esto permite a los desarrolladores crear aplicaciones web dinámicas y personalizadas que se actualizan en tiempo real según la información proporcionada por diversas fuentes.

- **Comunidad activa y ecosistema de paquetes**: *Astro* cuenta con una comunidad activa de desarrolladores que contribuyen con módulos y paquetes a través del sistema de gestión de paquetes *NPM*. Esto proporciona una amplia gama de funcionalidades adicionales y herramientas que se pueden utilizar para mejorar y extender las capacidades de *Astro*. La comunidad también brinda soporte y recursos para ayudar a los desarrolladores a resolver problemas y aprender mejores prácticas.

- **Mantenimiento y escalabilidad**: *Astro* ofrece una forma estructurada y organizada de desarrollar aplicaciones web, lo que facilita su mantenimiento y escalabilidad a medida que el proyecto crece. Los componentes modulares y la arquitectura clara permiten una fácil reutilización de código y facilitan la incorporación de nuevos desarrolladores al proyecto. [48]

5.3.3. *Svelte*

Svelte es un marco de trabajo de desarrollo web de código abierto que se destaca por su enfoque en la eficiencia y la velocidad de ejecución. A diferencia de otros marcos de trabajo como *React* o *Vue.js*, *Svelte* se diferencia por su enfoque en la compilación anticipada *(ahead-of-time compilation)* en lugar de utilizar una biblioteca de tiempo de ejecución en el navegador. Esto significa que el código *Svelte* se transforma en código optimizado y altamente eficiente durante el proceso de compilación, lo que resulta en un mejor rendimiento y una menor carga en el navegador del usuario final.

Una de las principales virtudes de *Svelte* es su enfoque en la reactividad. *Svelte* permite actualizar automáticamente la interfaz de usuario en respuesta a los cambios en los datos subyacentes. A diferencia de otros marcos de trabajo que realizan actualizaciones manuales

o utilizan el sistema de *virtual Document Object Model (DOM)*, *Svelte* logra esto en tiempo de compilación, generando código *JavaScript* optimizado que actualiza la interfaz de usuario de manera eficiente. Esto resulta en una experiencia de usuario fluida y receptiva, incluso en aplicaciones complejas.

Otra ventaja significativa de *Svelte* es su tamaño compacto. Debido a su enfoque en la compilación anticipada, las aplicaciones *Svelte* tienden a ser más pequeñas en tamaño en comparación con otras alternativas. Esto es beneficioso tanto para los desarrolladores como para los usuarios finales, ya que las aplicaciones se cargan más rápido y ocupan menos espacio en el dispositivo del usuario.

Svelte también se destaca por su simplicidad y facilidad de uso. La sintaxis de *Svelte* es limpia y concisa, lo que facilita la escritura y comprensión del código. Además, *Svelte* evita la necesidad de escribir código adicional para manejar la actualización del *DOM*, lo que simplifica el desarrollo y reduce la posibilidad de errores.

Una característica distintiva de *Svelte* es su enfoque en la optimización del rendimiento. Durante el proceso de compilación, *Svelte* analiza el código y aplica diversas técnicas de optimización, como la eliminación de código muerto y la minimización de operaciones innecesarias. Esto resulta en aplicaciones web altamente eficientes y rápidas, lo que mejora la experiencia del usuario final.

Además, *Svelte* cuenta con una comunidad activa de desarrolladores que contribuyen con módulos y componentes reutilizables a través del ecosistema de paquetes *NPM*. Esto permite a los desarrolladores aprovechar una amplia gama de funcionalidades preconstruidas y acelerar el desarrollo de aplicaciones. [50]

5.3.4. *JavaScript* y *TypeScript*

JavaScript y *TypeScript* son dos lenguajes de programación ampliamente utilizados en el desarrollo web y aplicaciones en el lado del cliente. Ambos lenguajes comparten una base común, pero tienen diferencias significativas en términos de sintaxis, características y forma de trabajar. A continuación, se explicarán de manera formal y extensa cada uno de ellos, destacando sus virtudes y beneficios de uso, y se realizará una comparación entre ambos.

JavaScript es un lenguaje de programación interpretado, dinámico y orientado a objetos que se utiliza principalmente para agregar interactividad y funcionalidad a las páginas web. Es

compatible con todos los navegadores web modernos y es uno de los lenguajes más utilizados en el desarrollo web. *JavaScript* cuenta con una amplia gama de bibliotecas y *frameworks*, que facilitan el desarrollo de aplicaciones web complejas. Algunas de las virtudes y beneficios de *JavaScript* son:

- **Versatilidad**: *JavaScript* es un lenguaje versátil que se puede utilizar tanto en el lado del cliente (navegador web) como en el lado del servidor *(Node.js)*. Esto permite a los desarrolladores utilizar el mismo lenguaje en ambos entornos, lo que facilita el desarrollo y la reutilización de código.

- **Interactividad**: *JavaScript* permite agregar interactividad y dinamismo a las páginas web. Con *JavaScript*, es posible manipular el *DOM* para cambiar y actualizar el contenido de una página de forma dinámica, responder a eventos del usuario, validar formularios y realizar peticiones *Asynchronous JavaScript and XML (AJAX)* para interactuar con servicios web.

- **Amplia comunidad y recursos**: *JavaScript* cuenta con una gran comunidad de desarrolladores y una amplia gama de recursos, como bibliotecas, *frameworks*, tutoriales y documentación. Esto facilita el aprendizaje y el desarrollo de aplicaciones web, ya que los desarrolladores pueden aprovechar las soluciones y las mejores prácticas establecidas por la comunidad.

- **Integración con tecnologías web**: *JavaScript* se integra perfectamente con *HTML* y *Cascading Style Sheets (CSS)*, lo que permite una fácil manipulación y personalización de los elementos de la página web. Además, *JavaScript* es compatible con *API* web modernas, como las *API* de geolocalización, notificaciones de tipo *push* y almacenamiento en el navegador, lo que permite crear aplicaciones web avanzadas.

Por otro lado, *TypeScript* es un derivado de *JavaScript* que agrega características de lenguajes de programación de tipado estático, lo que significa que se pueden definir tipos de datos para las variables y parámetros. *TypeScript* se compila a *JavaScript* y es compatible con todos los navegadores web modernos. Algunas de las virtudes y beneficios de *TypeScript* son:

- **Tipado estático**: una de las principales ventajas de *TypeScript* es su sistema de tipos estáticos. Permite detectar errores de tipo en tiempo de compilación y ofrece un desarrollo

más seguro y robusto. Además, el tipado estático facilita el mantenimiento del código a medida que el proyecto crece en tamaño y complejidad.

- **Autocompletado y verificación de errores**: *TypeScript* proporciona un sólido soporte de herramientas de desarrollo, como autocompletado y verificación de errores en tiempo real. Esto ayuda a los desarrolladores a escribir código más rápido y reduce los errores al detectar problemas en el código mientras se escribe.

- **Mejor escalabilidad**: *TypeScript* es especialmente útil en proyectos grandes y complejos, ya que ofrece una mejor escalabilidad. Con el uso de tipos estáticos, se pueden establecer contratos claros entre los componentes del código, lo que facilita la colaboración en equipos grandes y la gestión de proyectos a largo plazo. Además, *TypeScript* permite la modularización del código y el uso de clases y módulos, lo que favorece una estructura más organizada y mantenible.

- **Mayor productividad**: *TypeScript* proporciona características adicionales a *JavaScript*, como la inferencia de tipos, el uso de interfaces y la capacidad de definir tipos personalizados. Estas características ayudan a los desarrolladores a escribir un código más limpio y legible, reduciendo la cantidad de errores y permitiendo una mayor productividad en el desarrollo.

- **Compatibilidad con *JavaScript***: *TypeScript* es compatible con *JavaScript*, lo que significa que se puede utilizar el código *JavaScript* existente en un proyecto *TypeScript* sin problemas. Esto facilita la adopción gradual de *TypeScript* en proyectos existentes y permite a los desarrolladores aprovechar las ventajas de *TypeScript* sin tener que reescribir todo el código.

Así pues, *JavaScript* y *TypeScript* son lenguajes complementarios y tienen diferentes fortalezas dependiendo del contexto de desarrollo. *JavaScript* es más flexible y fácil de aprender, lo que lo hace adecuado para proyectos más pequeños y rápidos. Por otro lado, *TypeScript* ofrece ventajas en proyectos más grandes y complejos, ya que proporciona un sistema de tipos estáticos y herramientas de desarrollo más avanzadas. [51]

5.4. Base de datos

En este apartado, se abordará en detalle la base de datos utilizada como componente fundamental del sistema. La base de datos desempeña un papel crucial en la gestión, almacenamiento y recuperación de la información necesaria para el funcionamiento de la aplicación. A lo largo de esta exposición, se describirán exhaustivamente características y ventajas de la base de datos, destacando su importancia y relevancia en el contexto del proyecto. Además, se analizarán los aspectos técnicos y conceptuales que la hacen idónea para satisfacer las necesidades específicas del proyecto, proporcionando un entorno seguro, eficiente y confiable para la gestión de los datos.

5.4.1. *MongoDB*

MongoDB es *Sistema Gestor de Bases de Datos (SGBD)* de código abierto, orientado a documentos y no relacional. A diferencia de las bases de datos *Structured Query Language (SQL)* tradicionales, que se basan en tablas y relaciones, *MongoDB* se basa en un modelo de datos flexible y escalable conocido como almacenamiento de documentos.

En *MongoDB*, los datos se almacenan en documentos *JSON*, que son estructuras de datos flexibles y autocontenidas que pueden representar cualquier tipo de información. Cada documento puede tener un esquema diferente, lo que permite una mayor flexibilidad en comparación con las bases de datos *SQL*, donde se requiere un esquema fijo y predefinido.

Una de las principales virtudes de *MongoDB* es su capacidad para escalar horizontalmente. Esto significa que puede manejar grandes volúmenes de datos y un alto rendimiento al distribuir la carga de trabajo en varios servidores o nodos. A medida que crece la cantidad de datos y tráfico, se pueden agregar más nodos a la configuración de *MongoDB* para mantener el rendimiento y la disponibilidad del sistema. [53]

Así pues, *MongoDB* ofrece una serie de beneficios en comparación con las bases de datos *SQL*:

- **Flexibilidad y agilidad**: *MongoDB* permite agregar, modificar o eliminar campos en los documentos sin afectar el rendimiento o la estructura del esquema. Esto facilita la adaptación a cambios en los requisitos y evita la necesidad de realizar modificaciones complejas en la estructura de la base de datos.

- **Escalabilidad**: *MongoDB* es altamente escalable y puede manejar grandes volúmenes de datos distribuyendo la carga de trabajo en varios nodos. Esto permite un crecimiento fluido a medida que aumentan los requisitos de almacenamiento y rendimiento.

- **Rendimiento**: al eliminar la necesidad de realizar operaciones de unión complejas, *MongoDB* ofrece un alto rendimiento en consultas y operaciones de lectura y escritura. Además, su modelo de datos desnormalizado permite recuperar los datos de manera eficiente sin la necesidad de realizar múltiples consultas.

- **Facilidad de desarrollo**: *MongoDB* utiliza un modelo de datos similar a los objetos en lenguajes de programación, lo que facilita el desarrollo de aplicaciones. Esto elimina la necesidad de mapear objetos a tablas relacionales, lo que simplifica el desarrollo y reduce la cantidad de código necesario.

- **Replicación y tolerancia a fallos**: *MongoDB* admite la replicación automática de datos en varios nodos, lo que proporciona alta disponibilidad y tolerancia a fallos. En caso de que un nodo falle, otros nodos pueden tomar su lugar y garantizar la continuidad del servicio.

  A partir de lo expuesto anteriormente, *MongoDB* ofrece una alternativa flexible y escalable a las bases de datos *SQL* tradicionales. Su modelo de datos orientado a documentos, su capacidad de escalabilidad horizontal y su enfoque en la agilidad y el rendimiento hacen que sea una elección popular para aplicaciones modernas que requieren almacenamiento y procesamiento eficiente de grandes volúmenes de datos. [54]

5.5. *SaaS*

En el presente informe se detallarán las *SaaS* utilizadas en el desarrollo y despliegue del proyecto. Estas soluciones, disponibles en la nube, desempeñan un papel fundamental al proporcionar servicios y funcionalidades específicas sin requerir una infraestructura local. A lo largo de esta exposición, se explorarán en profundidad las *SaaS* implementadas. Asimismo, se analizará la pertinencia de su utilización en función de los recursos disponibles y las necesidades críticas de la aplicación. En definitiva, se presentará una visión completa de las *SaaS* utilizadas, resaltando su relevancia y el valor añadido que aportan al proyecto en términos de

eficiencia, agilidad y capacidad de adaptación a entornos dinámicos.

5.5.1. *MongoDB Atlas*

MongoDB Atlas es un servicio de base de datos en la nube, diseñado específicamente para la gestión de bases de datos *MongoDB*. Proporciona una solución completa y escalable para almacenar, gestionar y acceder a datos en entornos *cloud*.

MongoDB Atlas se basa en la tecnología de *MongoDB*, una base de datos *NoSQL* altamente flexible y orientada a documentos. Con *MongoDB Atlas*, los desarrolladores y las empresas pueden aprovechar todas las funcionalidades de *MongoDB* sin tener que preocuparse por la configuración y el mantenimiento de la infraestructura subyacente.

Una de las principales ventajas de *MongoDB Atlas* es su enfoque en la escalabilidad. Permite escalar verticalmente y horizontalmente de manera sencilla, lo que significa que se puede aumentar la capacidad de almacenamiento y la potencia de procesamiento de la base de datos de forma rápida y eficiente, a medida que las necesidades del proyecto o la aplicación van creciendo.

Además, *MongoDB Atlas* ofrece una alta disponibilidad y durabilidad de los datos. Utiliza replicación automática y distribución geográfica para garantizar que los datos estén siempre accesibles y protegidos contra posibles fallos o interrupciones. Esto es especialmente importante en entornos de producción, donde la continuidad del servicio es esencial.

Otra característica destacada de *MongoDB Atlas* es su integración con otras herramientas y servicios de la nube. Permite la integración con proveedores de servicios en la nube como *Amazon Web Services (AWS)*, *Google Cloud Platform (GCP)* y *Microsoft Azure*, lo que facilita la configuración y gestión de la infraestructura necesaria para el despliegue de aplicaciones.

En cuanto a la seguridad, *MongoDB Atlas* ofrece varias capas de protección para los datos almacenados. Proporciona opciones de cifrado de datos en reposo y en tránsito, así como autenticación y control de acceso basado en roles. Esto garantiza la confidencialidad e integridad de los datos, cumpliendo con los estándares de seguridad más exigentes. [56]

5.5.2. *Cloudinary*

Cloudinary es una plataforma de gestión de medios en la nube que ofrece una amplia gama de servicios y herramientas para almacenar, manipular y entregar contenido multimedia

de manera eficiente y escalable. Diseñada específicamente para satisfacer las necesidades de aplicaciones web y móviles que manejan grandes volúmenes de imágenes y vídeos, *Cloudinary* simplifica y agiliza el proceso de gestión de activos multimedia.

En esencia, *Cloudinary* actúa como un intermediario entre los desarrolladores de aplicaciones y los recursos multimedia, permitiéndoles almacenar sus archivos en la nube y acceder a ellos de manera rápida y segura. Uno de los principales beneficios de *Cloudinary* es su capacidad para adaptar y optimizar automáticamente los recursos multimedia según los requisitos de la aplicación y el dispositivo del usuario final.

Entre las funcionalidades clave de *Cloudinary* se encuentran la carga y almacenamiento de archivos, la manipulación y transformación de imágenes y vídeos, la entrega rápida y eficiente de contenido, y la integración con otras herramientas y servicios de la nube. La plataforma es compatible con una amplia variedad de formatos de archivo y ofrece opciones avanzadas de transformación, como recortar, redimensionar, comprimir, aplicar filtros y efectos, entre otros.

Una de las ventajas distintivas de *Cloudinary* es su capacidad para ofrecer una experiencia de entrega de contenido optimizada a través de una red de distribución de contenido *Content Delivery Network (CDN)*. Esto permite entregar de manera rápida y eficiente los recursos multimedia a los usuarios finales en cualquier parte del mundo, reduciendo la latencia y mejorando el rendimiento de la aplicación.

Además, *Cloudinary* ofrece características adicionales para la gestión de activos multimedia, como etiquetas y metadatos personalizados, capacidades de búsqueda y filtrado avanzadas, y control de acceso y seguridad a nivel de activo. También proporciona métricas y análisis detallados para evaluar el rendimiento y la utilización de los recursos multimedia. [58]

6

Pruebas y validaciones

En el contexto del desarrollo de aplicaciones, es de vital importancia llevar a cabo un proceso riguroso de pruebas y validación para garantizar que el producto final cumpla con los requisitos funcionales y no funcionales establecidos. En esta etapa, se busca demostrar la correcta implementación del sistema y verificar su conformidad con las especificaciones previamente definidas.

El objetivo principal de las pruebas y validación es identificar y corregir posibles errores, fallos de funcionamiento y brechas en la seguridad antes de que el sistema se ponga en producción. Además, se busca evaluar la implementación del sistema, asegurando que cumpla con las expectativas y necesidades de los usuarios finales.

En este contexto, el presente trabajo se enfocará en la demostración de los requisitos funcionales y no funcionales previamente definidos para validar la correcta implementación del sistema. A través de la ejecución de pruebas exhaustivas y la comparación de los resultados obtenidos con los criterios de aceptación establecidos, se evaluará el grado de cumplimiento de dichos requisitos. Además, se analizarán los resultados de las pruebas para identificar posibles mejoras o ajustes necesarios antes de la puesta en producción del sistema.

Con el objetivo de garantizar la calidad y confiabilidad del sistema, se emplearán técnicas y herramientas de prueba apropiadas, siguiendo metodologías de pruebas reconocidas y buenas prácticas de desarrollo de software. A través de este enfoque sistemático y metódico, se busca asegurar que el sistema cumpla con los estándares de calidad y se encuentre apto para su uso en el entorno operativo previsto.

Así pues, se ha empleado la herramienta *PyTest* en conjunción con la librería proporcionada por *FastAPI* para llevar a cabo pruebas de integración. Estas pruebas se enfocan en verificar el

correcto funcionamiento de los diferentes componentes del sistema cuando interactúan entre sí, asegurando así la integración adecuada de los distintos módulos y servicios.

PyTest es un *framework* de pruebas flexible y fácil de usar que ofrece una amplia gama de funcionalidades para la creación y ejecución de pruebas unitarias y de integración. Su sintaxis sencilla y su capacidad para descubrir automáticamente las pruebas hacen que sea una herramienta muy popular en el desarrollo de software.

Al combinar *PyTest* con la librería de pruebas de *FastAPI*, se ha logrado implementar casos de prueba de integración que cubren diferentes escenarios y flujos de ejecución del sistema. Estas pruebas evalúan la comunicación entre los diferentes componentes del sistema, la correcta respuesta de las *API* y la validación de los datos procesados.

6.1. Validación de requisitos funcionales y no funcionales de *Control-CSSC*

6.1.1. El sistema permite al usuario registrarse en el sistema.

Para validar este requisito funcional, se ha definido el test test_register_user_successfully en el cual se registra a un usuario con un nombre anteriormente no utilizado, un correo electrónico que tampoco está en uso en el sistema y una contraseña segura:

```
@pytest.mark.asyncio
async def test_register_user_successfully(client):
    registerDict = {
        "username": "testUser",
        "password": "SeCuRePaSsWoRd&123",
        "email": "testUser@uma.es"
    }
    response = client.post("/register", json=registerDict)
    assert response.status_code == 200
    assert response.json()["res"] == "User created successfully
  "
```

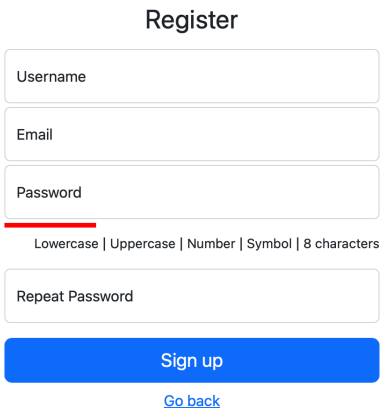

Figura 19: Interfaz web mostrando el formulario de registro

Para probar casos de error, se ha definido varios tests, entre los que se encuentran test_register_user_already_existing_username, para así comprobar la no duplicidad de nombres de usuario en el sistema:

```
@pytest.mark.asyncio
async def test_register_user_already_existing_username(client):
    registerDict = {
        "username": "testUser",
        "password": "SeCuRePaSsWoRd&123",
        "email": "testUser1@uma.es"
    }
    response = client.post("/register", json=registerDict)
    assert response.status_code == 400
    assert response.json()["detail"] == "User or email already
   exists"
```

Además, se ha probado también el caso de la duplicidad de correos electrónicos con el test test_register_user_already_existing_email, para así comprobar la no duplicidad de correos electrónicos en el sistema:

```
@pytest.mark.asyncio
async def test_register_user_already_existing_email(client):
```

```
    registerDict = {
        "username": "testUser1",
        "password": "SeCuRePaSsWoRd&123",
        "email": "testUser@uma.es"
    }
    response = client.post("/register", json=registerDict)
    assert response.status_code == 400
    assert response.json()["detail"] == "User or email already
   exists"
```

En ambos casos, la interfaz web informa al usuario del error, como se puede ver en la figura 20

Figura 20: Interfaz web mostrando error de duplicidad de usuario o correo electrónico

Por otro lado, se ha probado el caso del uso de una contraseña no segura, para así asegurar la robustez de las mismas dentro del sistema con el test test_register_password_not_secure, no permitiendo el uso de contraseñas débiles:

```
@pytest.mark.asyncio
async def test_register_password_not_secure(client):
    registerDict = {
        "username": "newUser",
        "password": "notsecure",
        "email": "newemail@uma.es"
    }
    response = client.post("/register", json=registerDict)
    assert response.status_code == 400
    assert response.json()["detail"] == "PASSWORD NOT SECURE"
```

También se puede comprobar el error lanzado al usuario a través de la interfaz web para este caso, como se puede observar en la figura 21

Figura 21: Interfaz web mostrando error de contraseña insegura

Por último, se ha comprobado a través de tres tests, test_register_user_missing_username, test_register_user_missing_password y test_register_user_missing_email que el servidor no permite el registro de usuarios cuyos formularios tienen datos sin rellenar:

```
@pytest.mark.asyncio
async def test_register_user_missing_username(client):
    registerDict = {
        "password": "SeCuRePaSsWoRd&123",
        "email": "newemail@newemail.com"
    }
    response = client.post("/register", json=registerDict)
    assert response.status_code == 422

@pytest.mark.asyncio
async def test_register_user_missing_password(client):
    registerDict = {
        "username": "newuser",
        "email": "newemail@newemail.com"
    }
    response = client.post("/register", json=registerDict)
    assert response.status_code == 422

@pytest.mark.asyncio
async def test_register_user_missing_email(client):
    registerDict = {
```

```
        "username": "newuser",
        "password": "SeCuRePaSsWoRd&123"
    }
    response = client.post("/register", json=registerDict)
    assert response.status_code == 422
```

Así pues, al ejecutar el comando pytest se comprueba que los tests arrojan un resultado positivo, quedando validados CRF-01 y CRNF-01.

6.1.2. El sistema permite al usuario iniciar sesión en el sistema

Para validar este requisito funcional, se ha definido el test test_login_user_valid, de manera que si un usuario está registrado, puede iniciar sesión con sus credenciales válidas, obteniendo así dos *tokens*: acceso y sesión que serán almacenados en el navegador en una *cookie* por la interfaz web, y además comprobando además que sus permisos y parámetros son los adecuados para ser usados dentro de la implementación y correcto uso del sistema:

```
@pytest.mark.asyncio
async def test_login_user_valid(client):
    response = client.post("/login",
    data={
        "username": "testUser",
        "password": "SeCuRePaSsWoRd&123"
    })
    assert response.status_code == 200
    assert response.json()["access_token"] is not None
    assert response.json()["refresh_token"] is not None
    accessToken = response.json()["access_token"]
    decoded = decode(accessToken, Settings().authjwt_secret_key
   , algorithms=["HS256"])
    assert decoded["isAdmin"] is False
    assert decoded["email"] == "testUser@uma.es"
    assert decoded["authUnlock"] is False
```

```
    assert decoded["ocppId"] is -1
```

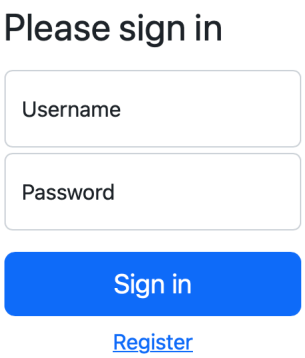

Figura 22: Interfaz web mostrando formulario de login

Por otro lado, se ha validado el caso en el que usuario o contraseña no sean válidos con los tests test_login_user_invalid_username y test_login_user_invalid_password, no dejando así el sistema iniciar sesión y arrojando un código de error:

```
@pytest.mark.asyncio
async def test_login_user_invalid_username(client):
    response = client.post("/login", data={
        "username": "testUser1",
        "password": "SeCuRePaSsWoRd&123"
    })
    assert response.status_code == 404

@pytest.mark.asyncio
async def test_login_user_invalid_password(client):
    response = client.post("/login", data={
        "username": "testUser",
        "password": "SeCuRePaSsWoRd&1234"
    })
    assert response.status_code == 404
```

Así pues, se puede comprobar en la interfaz web el error mostrado la usuario en ambos casos, como se puede ver en la figura 23

Figura 23: Interfaz web mostrando error referido a credenciales inválidas de usuario

Además, la interfaz web valida el *token* de usuario recibido por el sistema de autenticación, y si éste no es de tipo usuario, le deniega el acceso, como se puede observar en la figura 24.

Figura 24: Interfaz web mostrando error de autenticación a administrador en interfaz destinada a usuarios comunes

Por último, se ha comprobado además que el sistema no deja iniciar sesión a los usuarios que no aporten alguno de los datos requeridos para ello con los tests test_login_user_missing_username y test_login_user_missing_password, impidiendo el inicio de sesión arrojando un código de error:

```
@pytest.mark.asyncio
async def test_login_user_missing_username(client):
    response = client.post("/login", data={
        "password": "SeCuRePaSsWoRd&123"
    })
    assert response.status_code == 422

@pytest.mark.asyncio
async def test_login_user_missing_password(client):
    response = client.post("/login", data={
        "username": "testUser"
    })
    assert response.status_code == 422
```

Así pues, al ejecutar el comando pytest se comprueba que los tests arrojan un resultado positivo, quedando validados CRF-02, CRNF-02, CRNF-03 y CRNF-11.

6.1.3. El sistema permite al usuario consultar los cargadores disponibles a través de un mapa.

Para validar este requisito funcional tenemos que dividir las pruebas en dos entornos distintos: interfaz web y servidor.

Comenzando por la parte del servidor se ha de validar que los usuarios pueden obtener una lista de los cargadores del sistema, para ello se ha creado un test test_get_chargers, el cual usa un *token* de autenticación y autorización previamente otorgado a un usuario que ha iniciado sesión en el sistema:

```
@pytest.mark.asyncio
async def test_get_chargers(client):
    accessToken = Settings().validToken
    response = client.get(
        "/chargers",
        headers={
            "Authorization": f"Bearer {accessToken}"
    })
    assert response.status_code == 200
    assert response.json() != []
    print(response.json())
    for charger in response.json():
        assert charger["_id"] != None
        assert charger["name"] != None
        assert charger["latitude"] != None
        assert charger["longitude"] != None
        assert charger["image"] != None
        assert charger["locationDescription"] != None
        assert charger["ocppId"] != None
```

Además, se puede obtener información sobre cargadores en particular, para ello se han creado los tests test_get_charger_by_id para el caso en el que el identificador exista, test_get_charger_by_id_not_found para el caso en el que el identificador no exista, y test_get_charger_by_id_invalid_id para el caso en el que el identificador no sea válido:

```
@pytest.mark.asyncio
async def test_get_charger_by_id(client):
    accessToken = Settings().validToken
    response = client.get("/chargers/63e384ad3ba66bd7e65b5620",
    headers={"Authorization": f"Bearer {accessToken}"})
    assert response.status_code == 200
    assert response.json() != []
    assert response.json()["_id"] == "63e384ad3ba66bd7e65b5620"

@pytest.mark.asyncio
async def test_get_charger_by_id_not_found(client):
    accessToken = Settings().validToken
    response = client.get("/chargers/63e384ad3ba66bd7e65b5621",
    headers={"Authorization": f"Bearer {accessToken}"})
    assert response.status_code == 404

@pytest.mark.asyncio
async def test_get_charger_by_id_invalid_id(client):
    accessToken = Settings().validToken
    response = client.get("/chargers/invalidId", headers={"
   Authorization": f"Bearer {accessToken}"})
    assert response.status_code == 422
```

Por otro lado, verificamos la parte del mapa en la propia interfaz web, obteniendo así el resultado deseado reflejado en la figura 25:

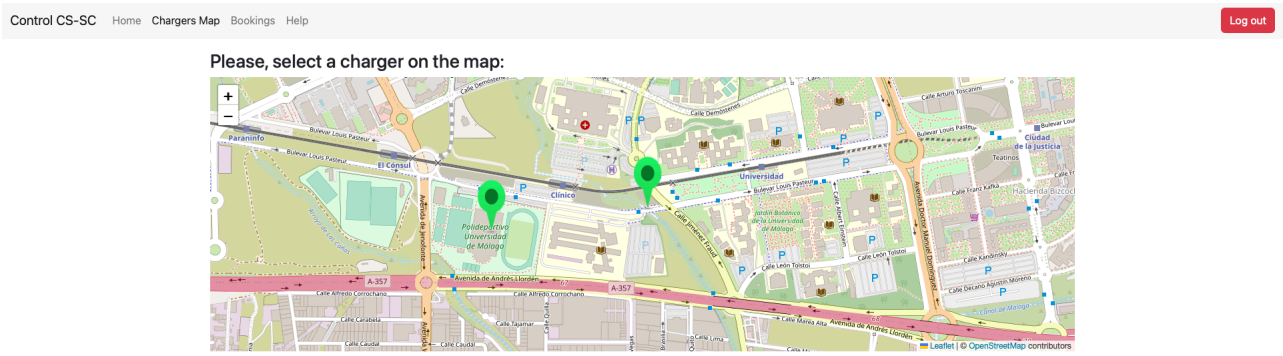

Figura 25: Interfaz web mostrando los cargadores existentes en el sistema

Así pues, al ejecutar el comando pytest se comprueba que los tests arrojan un resultado positivo, queda validado el requisito CRF-03.

6.1.4. El sistema permite al usuario consultar los tramos horarios disponibles de un cargador específico en un día específico.

Para validar este requisito es necesario centrarnos tanto en la parte del cliente web, donde se ofrece al usuario una interfaz usable en la que, tras elegir el cargador que desea utilizar, se despliegan una serie de elementos que se pueden observar en la figura 26, como en el servidor, donde reside la lógica.

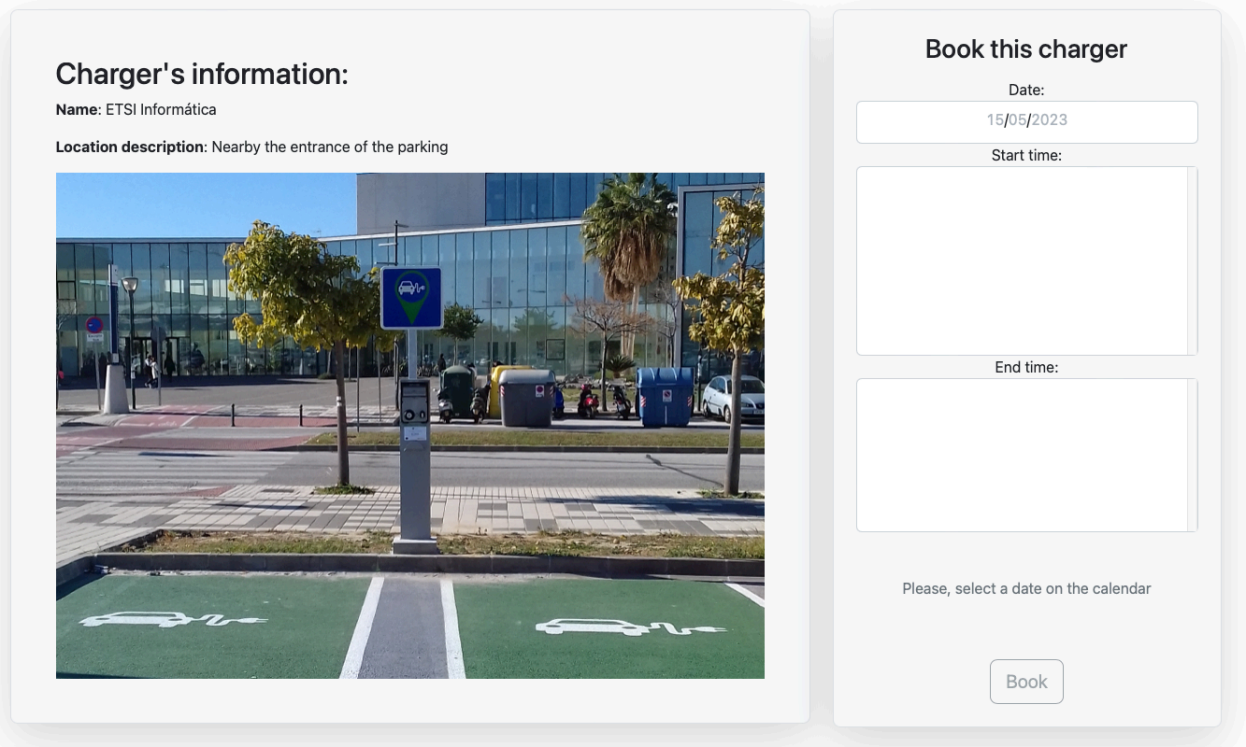

Figura 26: Interfaz web mostrando los distintos elementos tras elegir un cargador

Tras elegir una fecha determinada, se puede observar como el selector de hora de inicio se rellena con aquellas horas que se encuentran disponibles para realizar una reserva, como se puede observar en la figura 27. Los tramos de reserva comienzan a las 08:00 de la mañana y finalizan a las 22:00 de la noche, definidos así en el sistema. Para validar esta funcionalidad se ha creado un test, llamado test_get_available_start_times, para así probar la correcta funcionalidad del servidor para obtener los intervalos horarios disponibles para el comienzo de una reserva:

```
@pytest.mark.asyncio
async def test_get_available_start_times(client):
    accessToken = Settings().validToken
    response = client.get("/availableStartTimes?startDate
=2023-05-16T00:00:00&ocpp=1",
        headers={"Authorization": f"Bearer {accessToken}"})
    assert response.status_code == 200
```

```
    assert response.json() != []
```

Además, se han creado dos tests: test_get_available_start_times_missing_date y test_get_available_start_times_missing_ocpp para comprobar estados de error:

```
@pytest.mark.asyncio
async def test_get_available_start_times_missing_date(client):
    accessToken = Settings().validToken
    response = client.get("/availableStartTimes?ocpp=1",
        headers={"Authorization": f"Bearer {accessToken}"})
    assert response.status_code == 422

@pytest.mark.asyncio
async def test_get_available_start_times_missing_ocpp(client):
    accessToken = Settings().validToken
    response = client.get("/availableStartTimes?startDate
   =2023-05-16T00:00:00",
        headers={"Authorization": f"Bearer {accessToken}"})
    assert response.status_code == 422
```

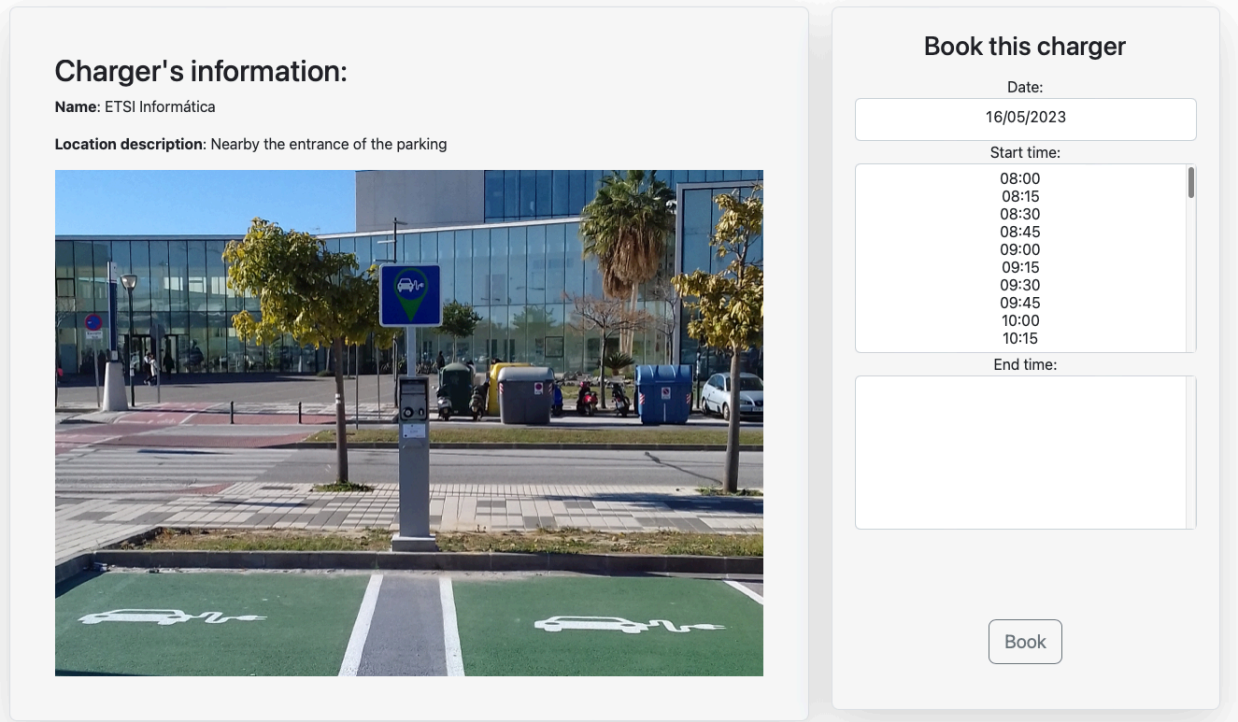

Figura 27: Interfaz web mostrando los tramos horarios de inicio de reserva disponibles para un cargador determinado en un día determinado

Habiendo seleccionado una fecha de inicio, la interfaz web deja ver aquellos tramos horarios de fin disponibles para el día y hora de inicio previamente seleccionados, como se puede observar en la figura 28. Cada reserva puede tener una longitud de, como mínimo, 15 minutos y como un máximo de dos horas. Para validar esta funcionalidad se ha creado un test, llamado test_get_available_end_times, para así probar la correcta funcionalidad del servidor para obtener los intervalos horarios disponibles para el fin de una reserva:

```
@pytest.mark.asyncio
async def test_get_available_end_times(client):
    accessToken = Settings().validToken
    response = client.get("/availableEndTimes?startDate
=2023-05-16T00:00:00&ocpp=1&startTime=1",
        headers={"Authorization": f"Bearer {accessToken}"})
    assert response.status_code == 200
    assert response.json() != []
```

Además, se han creado dos tests: test_get_available_start_times_missing_date y test_get_available_start_times_missing_ocpp para comprobar estados de error:

```
@pytest.mark.asyncio
async def test_get_available_end_times_missing_date(client):
    accessToken = Settings().validToken
    response = client.get("/availableEndTimes?ocpp=1&startTime
   =1",
        headers={"Authorization": f"Bearer {accessToken}"})
    assert response.status_code == 422

@pytest.mark.asyncio
async def test_get_available_end_times_missing_ocpp(client):
    accessToken = Settings().validToken
    response = client.get("/availableEndTimes?startDate
   =2023-05-16T00:00:00&startTime=1",
        headers={"Authorization": f"Bearer {accessToken}"})
    assert response.status_code == 422
```

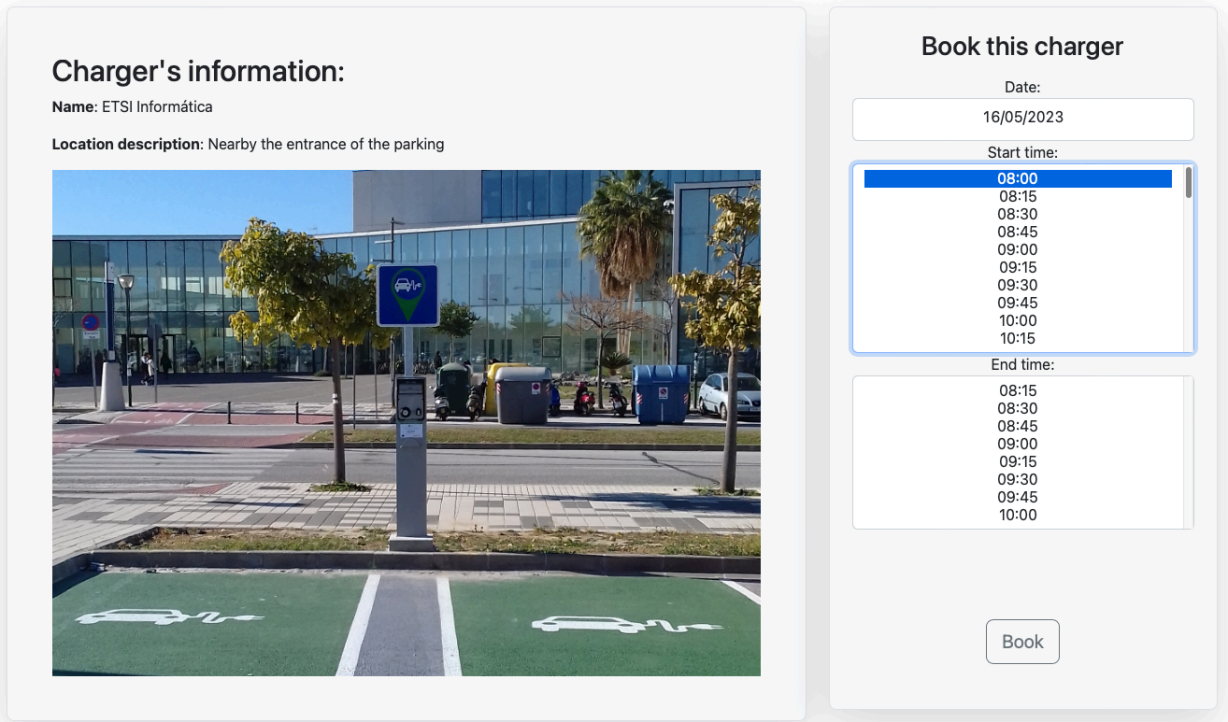

Figura 28: Interfaz web mostrando los tramos horarios de fin de reserva disponibles para un cargador determinado en un día determinado para una hora de inicio determinada

Así pues, al ejecutar el comando pytest se comprueba que los tests arrojan un resultado positivo, quedando validados los requisitos CRF-04, CRF-05, CRNF-04, CRNF-05, CRNF-06, CRNF-07 y CRNF-08.

6.1.5. El sistema permite al usuario realizar la reserva de un cargador en un tramo horario específico en un día especfíco.

Para la validación de este requisito se ha creado un test llamado test_create_bookings_user para comprobar que, dado un usuario válido, cargador válido y demás parámetros correctos, la reserva se crea de manera satisfactoria, se ha utilizado un *mock* (tecnología usada para imitar el comportamiendo de una determinada función) de la llamada a la *API* debido a que ésta requiere de otros microservicios:

```
@pytest.mark.asyncio
async def test_create_bookings_user(restore_client_post, client
    ):
```

```
    accessToken = "validToken"
    client.post = MagicMock(return_value=Response(status_code
=200))
    today = datetime.today()
    response = client.post("/bookings", headers={"Authorization
": f"Bearer {accessToken}"}, json={
        "chargerId": "63e384ad3ba66bd7e65b5620",
        "userId": "6460eb325223bb5768aefdc9",
        "startDate": datetime.today().isoformat(),
        "startTime": 34,
        "endTime": 35,
        "ocppId": 1
    })
    assert response.status_code == 200
```

En la parte de la interfaz, una vez elegida fecha y tramo horario, cuando el usuario decide reservar, obtiene un mensaje por pantalla confirmando la acción realizada, como se puede observar en la figura 29

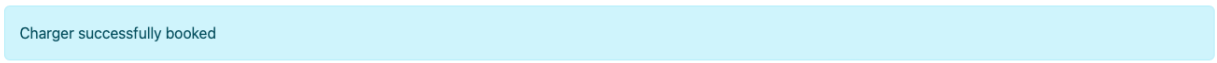

Figura 29: Interfaz web mostrando un mensaje indicando que la reserva se ha realizado de manera satisfactoria

Tras la reserva por parte del usuario, recibe en su bandeja de entrada un correo electrónico confirmando la reserva con el siguiente contenido:

```
Booking number <bookingNumber>.
Your booking has been created, you'll be able to unlock the
   charger on the selected date and time.

Thank you for choosing us.
Control-CSSC Team.
```

Así pues, llega la información de la reserva al servidor *OCPP*:

```
INFO: 10.1.1.232:52774 - "POST /reserve_now/1?
expiry_date_time=2023-05-16T13:45:00.000Z HTTP/1.1" 200 OK
```

Además, se han creado los tests test_create_bookings_invalid_charger_id, test_create_bookings_invalid_user_id y test_create_bookings_invalid_start_date para comprobar la validación correcta de datos:

```
@pytest.mark.asyncio
async def test_create_bookings_invalid_charger_id(
    restore_client_post, client):
    accessToken = Settings().validToken
    client.post = MagicMock(return_value=Response(status_code
    =422))
    today = datetime.today()
    response = client.post("/bookings", headers={"Authorization
    ": f"Bearer {accessToken}"}, json={
        "chargerId": "invalidChargerId",
        "userId": "6460eb325223bb5768aefdc9",
        "startDate": datetime.today().isoformat(),
        "startTime": 34,
        "endTime": 35,
        "ocppId": 1
    })
    assert response.status_code == 422

@pytest.mark.asyncio
async def test_create_bookings_invalid_user_id(
    restore_client_post, client):
    accessToken = Settings().validToken
    client.post = MagicMock(return_value=Response(status_code
    =422))
```

```
    today = datetime.today()
    response = client.post("/bookings", headers={"Authorization
  ": f"Bearer {accessToken}"}, json={
        "chargerId": "63e384ad3ba66bd7e65b5620",
        "userId": "invalidUserId",
        "startDate": datetime.today().isoformat(),
        "startTime": 34,
        "endTime": 35,
        "ocppId": 1
    })
    assert response.status_code == 422

@pytest.mark.asyncio
async def test_create_bookings_invalid_start_date(
  restore_client_post, client):
    accessToken = Settings().validToken
    client.post = MagicMock(return_value=Response(status_code
  =422))
    today = datetime.today()
    response = client.post("/bookings", headers={"Authorization
  ": f"Bearer {accessToken}"}, json={
        "chargerId": "63e384ad3ba66bd7e65b5620",
        "userId": "6460eb325223bb5768aefdc9",
        "startDate": "invalidDate",
        "startTime": 34,
        "endTime": 35,
        "ocppId": 1
    })
    assert response.status_code == 422
```

Si además el usuario intenta realizar una reserva en el pasado, la interfaz web comprueba los datos y devuelve un mensaje de error, como se puede observar en la figura 30.

Figura 30: Interfaz web mostrando un mensaje indicando que la reserva debe realizarse en una fecha futura

Así pues, al ejecutar el comando pytest se comprueba que los tests arrojan un resultado positivo, quedando validados los requisitos CRF-05, CRF-05.1 y CRNF-08.

6.1.6. El sistema permite al usuario consultar sus reservas futuras y expiradas.

Para validar este requisito, se han creado numerosos tests, tanto con filtrado como sin ellos y con ordenación, entre los que se encuentran: test_get_bookings_user para el caso de filtrado por identificador de usuario,
test_get_user_booking_by_id para el caso de filtrado por identificador de reserva,
test_get_user_bookings_date para el caso de filtrado por fecha,
test_get_user_bookings_date_time para el caso de filtrado por fecha y hora de comienzo,
test_get_user_bookings_date_time_chargerId para el caso de filtrado por fecha, hora de comienzo e identificador de reserva,
test_get_user_bookings_sort_date_ascending para el caso de ordenación por fecha de manera ascendente,
test_get_user_bookings_sort_date_descending para el caso de ordenación por fecha de manera descendente,
test_get_user_bookings_sort_charger_ascending para el caso de ordenación por cargador de manera ascendente y
test_get_user_bookings_sort_charger_descending para el caso de ordenación por cargador de manera descendente:

```
@pytest.mark.asyncio
async def test_get_bookings_user(client):
    accessToken = Settings().validToken
    response = client.get("/bookings", headers={"Authorization
  ": f"Bearer {accessToken}"})
```

```
    assert response.status_code == 200
    assert response.json() != []
    for bookings in response.json():
        assert bookings["userId"] == "6411ff39717a9abb082bd6fb"

@pytest.mark.asyncio
async def test_get_user_booking_by_id(client):
    accessToken = Settings().validToken
    response = client.get("/bookings/6463a19e91d6f5fbd2b9ca80",
    headers={"Authorization": f"Bearer {accessToken}"})
    print(response.json())
    assert response.status_code == 200
    assert response.json() != []
    assert response.json()["_id"] == "6463a19e91d6f5fbd2b9ca80"

@pytest.mark.asyncio
async def test_get_user_bookings_date(client):
    accessToken = Settings().validToken
    date = "2023-05-17T00:00:00"
    response = client.get(f"/bookings?date={date}", headers={"
   Authorization": f"Bearer {accessToken}"})
    assert response.status_code == 200
    assert response.json() != []
    for bookings in response.json():
        assert bookings["startDate"] == date

@pytest.mark.asyncio
async def test_get_user_bookings_date_time(client):
    accessToken = Settings().validToken
    date = "2023-05-17T00:00:00"
```

```
    response = client.get(f"/bookings?date={date}&time=0",
   headers={"Authorization": f"Bearer {accessToken}"})
    assert response.status_code == 200
    assert response.json() != []
    for bookings in response.json():
        assert bookings["startDate"] == date
        assert bookings["startTime"] == 0

@pytest.mark.asyncio
async def test_get_user_bookings_date_time_chargerId(client):
    accessToken = Settings().validToken
    date = "2023-05-17T00:00:00"
    response = client.get(f"/bookings?date={date}&time=0&
   chargerId=63e384ad3ba66bd7e65b5620", headers={"Authorization
   ": f"Bearer {accessToken}"})
    assert response.status_code == 200
    assert response.json() != []
    for bookings in response.json():
        assert bookings["startDate"] == date
        assert bookings["startTime"] == 0
        assert bookings["chargerId"] == "63
   e384ad3ba66bd7e65b5620"

@pytest.mark.asyncio
async def test_get_user_bookings_date_sort_date_ascending(
   client):
    accessToken = Settings().validToken
    date = "2023-05-17T00:00:00.000Z"
    response = client.get(f"/bookings",
        headers={"Authorization": f"Bearer {accessToken}"})
```

```
    responseOrdered = client.get(f"/bookings?sort=startDate&
    order=asc",
        headers={"Authorization": f"Bearer {accessToken}"})
    assert response.status_code == 200
    assert response.json() != []
    assert response != responseOrdered

@pytest.mark.asyncio
async def test_get_user_bookings_date_sort_date_descending(
    client):
    accessToken = Settings().validToken
    date = "2023-05-17T00:00:00.000Z"
    response = client.get(f"/bookings?date={date}",
        headers={"Authorization": f"Bearer {accessToken}"})
    responseOrderedAsc = client.get(f"/bookings?date={date}&
    sort=startDate&order=asc",
        headers={"Authorization": f"Bearer {accessToken}"})
    responseOrderedDesc = client.get(f"/bookings?date={date}&
    sort=startDate&order=desc",
        headers={"Authorization": f"Bearer {accessToken}"})
    assert response.status_code == 200
    assert response.json() != []
    assert response != responseOrderedAsc
    assert response != responseOrderedDesc
    assert responseOrderedAsc != responseOrderedDesc

@pytest.mark.asyncio
async def test_get_user_bookings_sort_charger_ascending(client)
    :
    accessToken = Settings().validToken
    response = client.get("/bookings",
```

```
        headers={"Authorization": f"Bearer {accessToken}"})
    responseOrdered = client.get(f"/bookings?sort=charger&order
  =asc",
        headers={"Authorization": f"Bearer {accessToken}"})
    assert response.status_code == 200
    assert response.json() != []
    assert response != responseOrdered

@pytest.mark.asyncio
async def test_get_user_bookings_sort_charger_descending(client
  ):
    accessToken = Settings().validToken
    response = client.get("/bookings",
        headers={"Authorization": f"Bearer {accessToken}"})
    responseOrderedAsc = client.get(f"/bookings?sort=charger&
  order=asc",
        headers={"Authorization": f"Bearer {accessToken}"})
    responseOrderedDesc = client.get(f"/bookings?sort=charger&
  order=desc",
        headers={"Authorization": f"Bearer {accessToken}"})
    assert response.status_code == 200
    assert response.json() != []
    assert response != responseOrderedAsc
    assert response != responseOrderedDesc
    assert responseOrderedAsc != responseOrderedDesc
```

Así pues, en la interfaz web se pone a disposición del usuario filtros para estas acciones, como se puede observar en la figura 31.

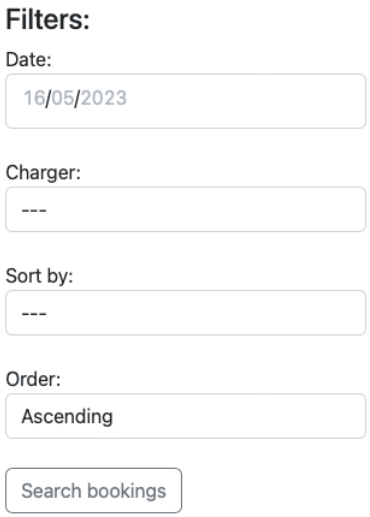

Figura 31: Interfaz web mostrando los filtros disponibles para las reservas del usuario

Para el caso en el que un usuario malintencionado intentase acceder a las reservas de otro usuario, se ha creado un test, test_get_booking_by_id_another_user, el cual asegura la privacidad de los usuarios con respecto a sus datos personales de reservas:

```
@pytest.mark.asyncio
async def test_get_booking_by_id_another_user(client):
    accessToken = Settings().validToken
    anotherUserBooking = client.get("/bookings/6457
   de7dbe544a4157d8a506",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    assert anotherUserBooking.status_code == 404
```

Por otro lado, si un usuario intenta buscar una reserva no existente, se ha creado un test, test_get_user_booking_not_existing, el cual muestra un error informando de la situación al usuario, además se ha creado el test test_get_booking_not_valid_id para el caso en el que se busca un identificador de reserva no válido:

```
@pytest.mark.asyncio
async def test_get_user_booking_not_existing(client):
    accessToken = Settings().validToken
```

```
    response = client.get("/bookings/6460eb325223bb5768aefdc8",
    headers={"Authorization": f"Bearer {accessToken}"})
    assert response.status_code == 404


@pytest.mark.asyncio
async def test_get_booking_not_valid_id(client):
    accessToken = Settings().validToken
    response = client.get("/bookings/invalidId", headers={"
   Authorization": f"Bearer {accessToken}"})
    assert response.status_code == 422
```

Así pues, al ejecutar el comando pytest se comprueba que los tests arrojan un resultado positivo, quedando validados los requisitos CRF-06, CRF-06.1, CRF-06.2, CRF-06.3, CRF-06.4 y CRF-06.5.

6.1.7. El sistema permite al usuario comenzar la carga en un puesto bajo reserva en una fecha y franja de tiempo determinadas.

Para validar este requisito, se ha creado un test llamado test_start_charging_valid_id_valid_token para comprobar que el desbloqueo del cargador se realiza de manera satisfactoria. De nuevo, se ha utilizado un *mock* para simular el comportamiento debido a que esta función del servidor depende de otros microservicios.

```
@pytest.mark.asyncio
async def test_start_charging_valid_id_valid_token(
   restore_client_post, client):
    accessToken = Settings().validToken
    client.post = MagicMock(return_value=Response(status_code
   =200))
    response = client.post("/chargers/1/start", headers={"
   Authorization": f"Bearer {accessToken}"})
    assert response.status_code == 200
```

Además, se han creado los tests test_start_charging_invalid_id_valid_token para el

caso en el que el identificador no sea válido y el *token* de acceso sea válido, test_start_charging_valid_id_invalid_token para el caso en el que el identificador sea válido y el *token* de acceso sea inválido, test_start_charging_invalid_id_invalid_token para el caso en el que ambos parámetros sean inválidos:

```
@pytest.mark.asyncio
async def test_start_charging_invalid_id_valid_token(
    restore_client_post, client):
     accessToken = Settings().validToken
     client.post = MagicMock(return_value=Response(status_code
    =403))
     response = client.post("/chargers/99/start", headers={"
    Authorization": f"Bearer {accessToken}"})
     assert response.status_code == 403

@pytest.mark.asyncio
async def test_start_charging_valid_id_invalid_token(
    restore_client_post, client):
     accessToken = "invalidToken"
     client.post = MagicMock(return_value=Response(status_code
    =403))
     response = client.post("/chargers/1/start", headers={"
    Authorization": f"Bearer {accessToken}"})
     assert response.status_code == 403

@pytest.mark.asyncio
async def test_start_charging_invalid_id_invalid_token(
    restore_client_post, client):
     accessToken = "invalidToken"
     client.post = MagicMock(return_value=Response(status_code
    =403))
```

```
    response = client.post("/chargers/99/start", headers={"
Authorization": f"Bearer {accessToken}"})
    assert response.status_code == 403
```

Como podemos observar en la figura 32 la interfaz web permite desbloquear el cargador al usuario si se encuentra en la fecha y hora seleccionadas para la reserva.

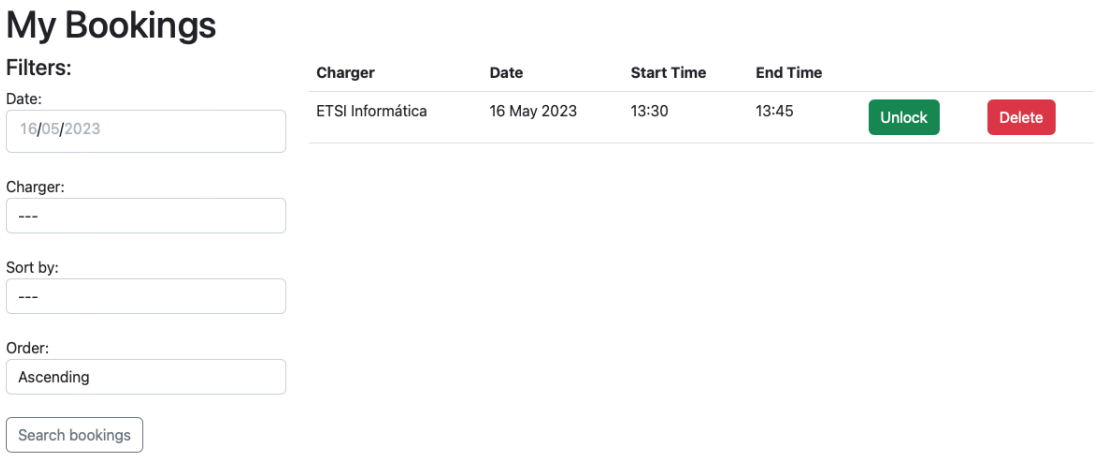

Figura 32: Interfaz web mostrando una reserva lista para ser utilizada

Una vez desbloqueado el cargador, la interfaz web devuelve un mensaje al usuario indicando que la transacción se ha realizado de manera satisfactoria, como se puede observar en la figura 33. Además, el sistema muestra la reserva como desbloqueada al usuario, como se puede observar en la figura 34, recibiendo además el servidor *OCPP* la instrucción de desbloqueo del cargador:

```
    INFO: 10.1.1.216:39630 - "POST /remote_start_transaction/1
HTTP/1.1" 200 OK
```

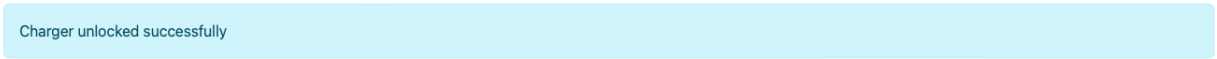

Figura 33: Interfaz web mostrando mensaje de confirmación de desbloqueo de cargador

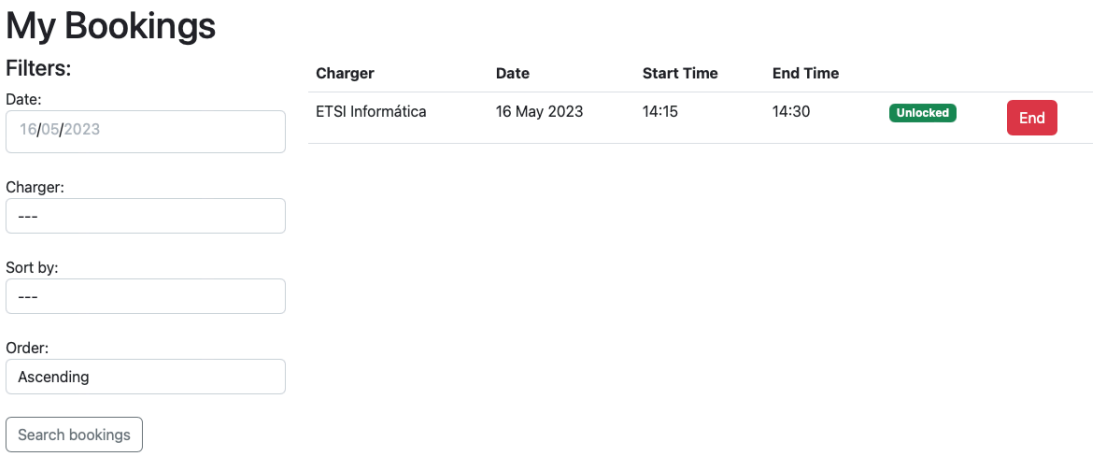

Figura 34: Interfaz web mostrando reserva iniciada por el usuario

Si el usuario decide terminar la reserva de manera anticipada a la hora estipulada para su fin la interfaz web permite realizar dicha acción como se puede observar en la figura 34, obteniendo así un mensaje de confirmación tras realizar la transacción, como se puede observar en la figura 35. Para probar esta funcionalidad, se ha creado el test test_stop_charging_valid_id_valid_token_valid_booking_id, que también usa un *mock* debido a dependencias de otros microservicios:

```
@pytest.mark.asyncio
async def
    test_stop_charging_valid_id_valid_token_valid_booking_id(
    restore_client_post, client):
      accessToken = Settings().validToken
      client.post = MagicMock(return_value=Response(status_code
    =200))
      response = client.post("/chargers/1/stop?bookingId=
    validBookingId",
          headers={"Authorization": f"Bearer {accessToken}"}
      )
      assert response.status_code == 200
```

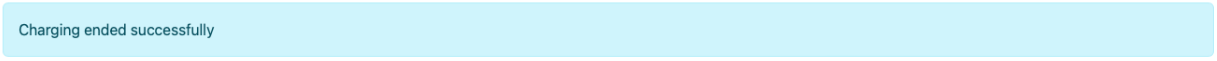

Figura 35: Interfaz web mostrando mensaje de confirmación de fin de reserva

El servidor *OCPP* recibe una indicación del fin de la carga y cancelación de la reserva:

```
INFO: 10.1.1.216:57390 - "POST /remote_stop_transaction/1?
transaction_id=646369f7fa9e420185f07a7d HTTP/1.1" 200 OK
INFO: 10.1.1.242:60308 - "POST /cancel_reservation/1?
reservation_id=646369676195420185607174 HTTP/1.1" 200 OK
```

En adición, se han desarrollado numerosos tests con todas las posibles combinaciones de argumentos válidos e inválidos para comprobar así que siempre que exista algún argumento inválido o inexistente el sistema devuelva un mensaje de error. Se han desarrollado numerosos tests (en concreto ocho, ya que son todas la posibles combinaciones de tres elementos), se citan el primeros y último, tanto para validez como para falta de argumentos: test_stop_charging_valid_id_valid_token_invalid_booking_id, y test_stop_charging_valid_id_invalid_token_no_booking_id:

```
@pytest.mark.asyncio
async def
test_stop_charging_valid_id_valid_token_valid_booking_id(
restore_client_post, client):
accessToken = Settings().validToken
client.post = MagicMock(return_value=Response(status_code
=200))
response = client.post("/chargers/1/stop?bookingId=
validBookingId", headers={"Authorization": f"Bearer {
accessToken}"})
assert response.status_code == 200

@pytest.mark.asyncio
```

```
async def
    test_stop_charging_valid_id_invalid_token_no_booking_id(
    restore_client_post, client):
      accessToken = "invalidToken"
      client.post = MagicMock(return_value=Response(status_code
    =403))
      response = client.post("/chargers/1/stop", headers={"
    Authorization": f"Bearer {accessToken}"})
      assert response.status_code == 403
```

Por otro lado, si el usuario no finaliza su reserva de manera anticipada, el sistema la finaliza de manera automática, cada 15 minutos se comprueba si hay una reserva que debe terminar, y si la hay, es finalizada por el sistema, informando al servidor *OCPP* de la transacción.

Finalmente y de manera automática, si la fecha y hora de la reserva está en el pasado, la interfaz web mostrará la reserva como expirada, como se muestra en la figura 36.

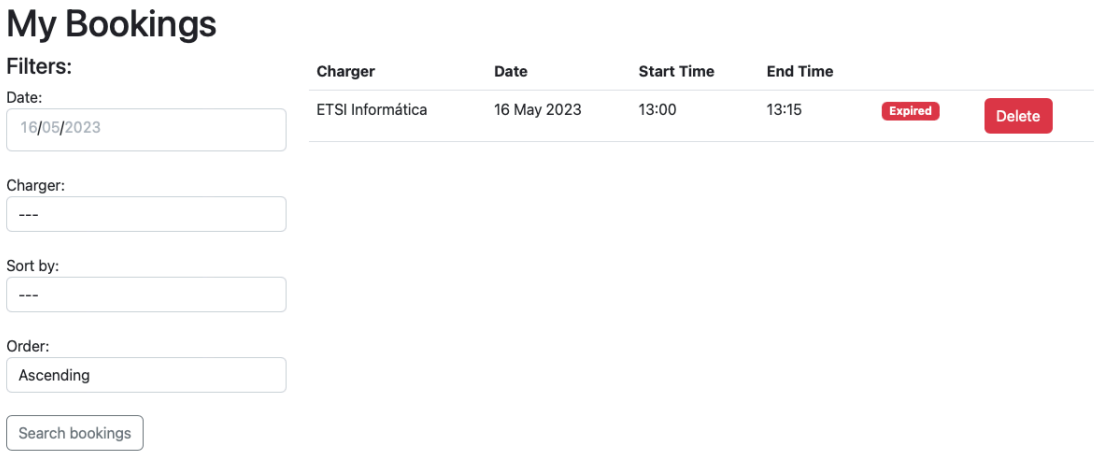

Figura 36: Interfaz web mostrando una reserva expirada

El usuario puede eliminar una reserva, ya sea previo a su utilización o expirada, ya que al usarla y finalizarla se elimina de manera automática. Para probar esta funcionalidad se ha definido el test test_delete_user_booking, usando de nuevo un *mock* por dependencias de otros microservicios:

```
@pytest.mark.asyncio
```

```
async def test_delete_user_booking(restore_client_delete,
    client):
     accessToken = Settings().validToken
     client.delete = MagicMock(return_value=Response(status_code
    =200))
     response = client.delete("/bookings/validUserBooking")
     assert response.status_code == 200
```

También, en el caso de que un usuario de manera malintencionada consigue acceso al servidor e intenta eliminar la reserva de otro usuario se ha probado con el test test_delete_user_booking_another_user:

```
@pytest.mark.asyncio
async def test_delete_user_booking_another_user(
    restore_client_delete, client):
     accessToken = Settings().validToken
     client.delete = MagicMock(return_value=Response(status_code
    =404))
     response = client.delete("/bookings/anotherUserBooking")
     assert response.status_code == 404
```

Por otro lado, para el caso en el que la reserva no exista, se ha creado el test test_delete_user_booking_not_existing:

```
@pytest.mark.asyncio
async def test_delete_user_booking_not_existing(
    restore_client_delete, client):
     accessToken = Settings().validToken
     client.delete = MagicMock(return_value=Response(status_code
    =404))
     response = client.delete("/bookings/notExistingBooking")
     assert response.status_code == 404
```

Por último, se ha desarrollado un test para casos en los que la validación de parámetros no sea correcta: test_delete_user_booking_invalid_id:

```
@pytest.mark.asyncio
async def test_delete_user_booking_invalid_id(
    restore_client_delete, client):
     accessToken = Settings().validToken
     client.delete = MagicMock(return_value=Response(status_code
    =422))
     response = client.delete("/bookings/invalidId")
     assert response.status_code == 422
```

Así pues, al ejecutar el comando pytest se comprueba que los tests arrojan un resultado positivo, quedando validados los requisitos CRF-08, CRF-09, CRF-10, CRF-11, CRNF-09 y CRNF-10.

6.1.8. El sistema permite al usuario crear un tiquet de incidencia.

Para validar este requisito, se ha creado el test test_create_ticket_user para comprobar su correcto funcionamiento:

```
@pytest.mark.asyncio
async def test_create_ticket_user(client):
     accessToken = Settings().validToken
     adminToken = Settings().accessTokenAdmin
     ticketsBefore = client.get("/tickets?solved=false",
          headers={"Authorization": f"Bearer {adminToken}"}
     )
     response = client.post("/tickets",
          headers={"Authorization": f"Bearer {accessToken}"},
          json={
               "userId": "6411ff39717a9abb082bd6fb",
               "userMsg": "ticket test",
               "userEmail": "jlbp@uma.es",
               "creationDate": "2023-05-15T00:00:00.000",
          }
```

```
    )
    ticketsAfter = client.get("/tickets?solved=false",
        headers={"Authorization": f"Bearer {adminToken}"}
    )
    assert response.status_code == 200
    assert response.json() != []
    assert ticketsBefore.status_code == 200
    assert ticketsAfter.status_code == 200
    assert ticketsBefore.json() != []
    assert ticketsAfter.json() != []
    assert len(ticketsAfter.json()) == len(ticketsBefore.json()
) + 1
```

Además, tras su ejecución, el usuario recibe el siguiente correo electrónico:

```
Ticket number <ticketId>.
Your ticket has been created:

<userMessage>

We will contact you as soon as possible.
Control-CSSC Team.
```

Por otro lado, se ha creado otro test llamado test_create_ticket_admin para el caso en el que el identificador de usuario no sea válido:

```
@pytest.mark.asyncio
async def test_create_ticket_admin(client):
    accessToken = Settings().validToken
    response = client.post("/tickets",
        headers={"Authorization": f"Bearer {accessToken}"},
        json={
            "userId": "invalid",
            "userMsg": "ticket invalid user id",
```

```
            "userEmail": "jlbp@uma.es",
            "creationDate": "2023-05-15T00:00:00.000",
        }
    )
    assert response.status_code == 422
```

Como se puede observar en la figura 37 esta funcionalidad se ve reflejada en la interfaz web, permitiendo así a los usuarios crear tiquets de incidencias que serán respondidos por el administrador, recibiendo además un correo electrónico indicando el mensaje de respuesta del mismo:

```
Ticket number <ticketId>.
Your ticket has been solved by an admin:

<adminMessage>

In case of needing further help, create another ticket.
Control-CSSC Team.
```

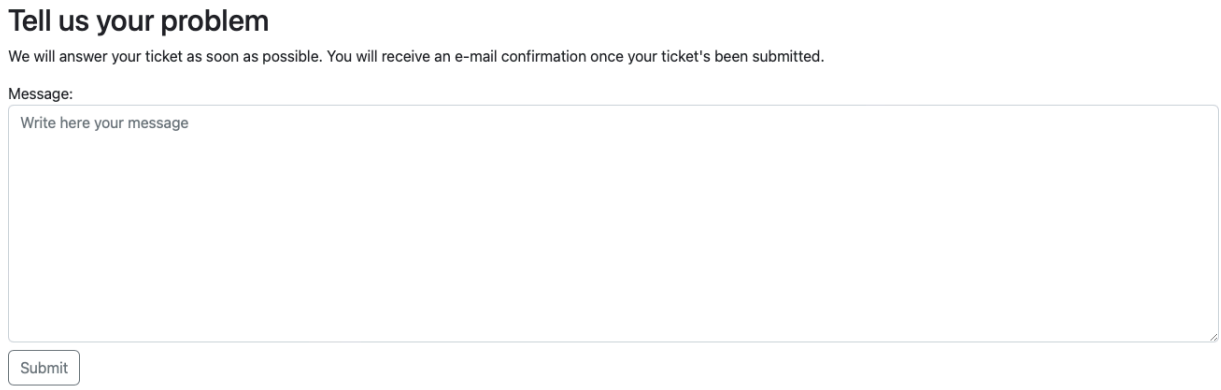

Figura 37: Interfaz web mostrando el formulario de creación de un tiquet de incidencia

Finalmente, una vez el usuario ha creado el tiquet de incidencia, la interfaz web le muestra un mensaje de confirmación de la transacción, como se puede observar en la figura 38

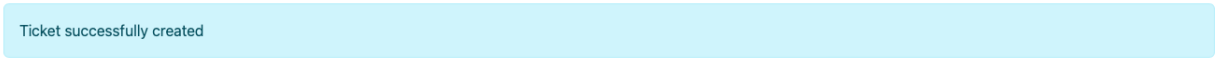

Figura 38: Interfaz web mostrando la confirmación de creación de tiquet de incidencia

Así pues, al ejecutar el comando pytest se comprueba que los tests arrojan un resultado positivo, quedando validados los requisitos CRF-12, CRF-12.1 y CRF-12.2.

6.1.9. El sistema sólo permite a los usuarios realizar reservas en una fecha y franja horaria si no tiene otra reserva activa

Para validar este requisito se ha creado el test test_not_overlapping_bookings de manera que, si ya existe una reserva asignada a un usuario en un día y franja horaria, al crear una nueva reserva, el sistema no permite al usuario que se solapen las franjas horarias. De nuevo, se ha usado un *mock* debido a dependencias con otros microservicios:

```
@pytest.mark.asyncio
async def test_not_overlapping_bookings(restore_client_post,
    client):
     accessToken = "validToken"
     client.post = MagicMock(return_value=Response(status_code
    =200))
     today = datetime.today()
     first = client.post("/bookings", headers={"Authorization":
    f"Bearer {accessToken}"}, json={
          "chargerId": "63e384ad3ba66bd7e65b5620",
          "userId": "6460eb325223bb5768aefdc9",
          "startDate": datetime.today().isoformat(),
          "startTime": 34,
          "endTime": 37,
          "ocppId": 1
     })
```

```
    assert first.status_code == 200

    client.post = MagicMock(return_value=Response(status_code
    =400))
    second = client.post("/bookings", headers={"Authorization":
    f"Bearer {accessToken}"}, json={
        "chargerId": "63e384ad3ba66bd7e65b5620",
        "userId": "6460eb325223bb5768aefdc9",
        "startDate": datetime.today().isoformat(),
        "startTime": 35,
        "endTime": 36,
        "ocppId": 1
    })
    assert second.status_code == 400
```

Esta funcionalidad se puede comprobar en la figura 39, donde se muestra el resultado de esta transacción en la interfaz web.

Figura 39: Interfaz web mostrando un error de solapamiento entre reservas

Así pues, al ejecutar el comando pytest se comprueba que el test arroja un resultado positivo, quedando validado el requisito CRNF-12.

6.2. Validación de requisitos funcionales y no funcionales de *Admin-CSSC*

6.2.1. El sistema permite al administrador iniciar sesión en el sistema con una cuenta válida de administrador.

Para validar este requisito funcional, parte de su validación se ha realizado en el punto 6.1.2 de este documento, para asegurar el inicio de sesión de un administrador se ha definido el test test_login_user_valid_admin, de manera que si un usuario está registrado, puede iniciar

sesión con sus credenciales válidas, obteniendo así dos *tokens*: acceso y sesión que serán almacenados en el navegador en una *cookie* por la interfaz web, y además comprobando además que sus permisos y parámetros son los adecuados para ser usados dentro de la implementación y correcto uso del sistema:

```
@pytest.mark.asyncio
async def test_login_user_valid_admin(client):
    response = client.post("/login", data={"username": "admin",
    "password": "123"})
    assert response.status_code == 200
    assert response.json()["access_token"] is not None
    assert response.json()["refresh_token"] is not None
    accessToken = response.json()["access_token"]
    decoded = decode(accessToken, Settings().authjwt_secret_key
   , algorithms=["HS256"])
    assert decoded["isAdmin"] is True
    assert decoded["email"] == "jlbp@uma.es"
    assert decoded["authUnlock"] is False
    assert decoded["ocppId"] is -1
```

De manera que, si un usuario regular intenta iniciar sesión en la interfaz web destinada a la administración, la propia interfaz procesará el *token* y denegará la transacción, como se puede ver en la figura 40.

Figura 40: Interfaz web mostrando un error a un usuario regular que intenta iniciar sesión en la interfaz web destinada a la administración

Una vez se introduzcan credenciales válidas de administrador, se iniciará sesión en el sistema de manera satisfactoria.

Así pues, al ejecutar el comando pytest se comprueba que los tests arrojan un resultado positivo, quedando validados los requisitos ARF-01, ARNF-02 Y ARNF-03.

6.2.2. El sistema permite al administrador ver todas las reservas efectuadas en el sistema por cualquier usuario.

Para validar este requisito funcional se ha definido el test test_get_bookings_admin, con el cual se comprueba que un administrador puede ver todas las reservas del sistema, mostrándose en la interfaz web como se puede observar en la figura 41:

```
@pytest.mark.asyncio
async def test_get_bookings_admin(client):
    adminId = "6410b5aa51dad87bbce38435"
    accessTokenAdmin = Settings().accessTokenAdmin
    response = client.get("/bookings", headers={"Authorization
  ": f"Bearer {accessTokenAdmin}"})
    print(response.json())
    assert response.status_code == 200
    assert response.json() != []
    assert response.json()[0]["userId"] != adminId
```

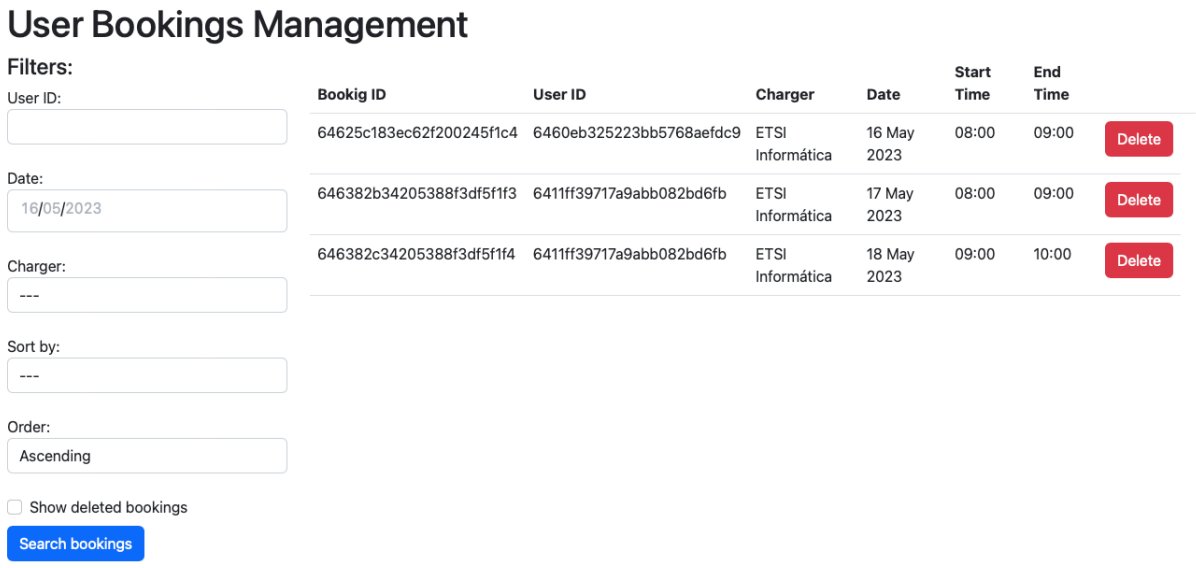

Figura 41: Interfaz web mostrando las reservas en el sistema

Además, el administrador puede acceder directamente a la información de determinadas

reservas, para ello se han creado dos tests: test_get_user_booking_admin para el caso correcto y test_get_user_booking_not_existing_admin para el caso en el que la reserva no existe:

```
@pytest.mark.asyncio
async def test_get_user_booking_admin(client):
    accessTokenAdmin = Settings().accessTokenAdmin
    response = client.get("/bookings/?userId=6411
   ff39717a9abb082bd6fb", headers={"Authorization": f"Bearer {
   accessTokenAdmin}"})
    assert response.status_code == 200
    assert response.json() != []
    print(response.json())
    for booking in response.json():
        assert booking["userId"] == "6411ff39717a9abb082bd6fb"

@pytest.mark.asyncio
async def test_get_user_booking_not_existing_admin(client):
    accessTokenAdmin = Settings().accessTokenAdmin
    response = client.get("/bookings/6460eb325223bb5768aefdc8",
    headers={"Authorization": f"Bearer {accessTokenAdmin}"})
    assert response.status_code == 404
```

Por otro lado, el administrador puede filtrar y ordenar las reservas de los usuarios usando varios criterios, para ello, se han creado varios tests: test_get_user_bookings_date_admin para el filtrado por fecha,
test_get_user_bookings_date_chargerId_admin para el filtrado por identificador de cargador,
test_get_user_bookings_sort_date_asc para el ordenado por fecha en orden ascendente,
test_get_user_bookings_sort_date_desc para el ordenado por fecha en orden descendente,
test_get_users_bookings_sort_charger_asc para el ordenado por cargador en orden ascendente,

test_get_users_bookings_sort_charger_desc para el ordenado por cargador en orden descendente, test_get_users_bookings_sort_user_asc para el ordenado por usuario en orden ascendente y test_get_users_bookings_sort_user_desc para el ordenado por cargador en orden descendente:

```
@pytest.mark.asyncio
async def test_get_user_bookings_date_admin(client):
    accessTokenAdmin = Settings().accessTokenAdmin
    date = "2023-05-16T00:00:00"
    response = client.get(f"/bookings?userId=6460
   eb325223bb5768aefdc9&date={date}", headers={"Authorization":
    f"Bearer {accessTokenAdmin}"})
    assert response.status_code == 200
    assert response.json() != []
    for booking in response.json():
        assert booking["startDate"] == date
        assert booking["userId"] == "6460eb325223bb5768aefdc9"

@pytest.mark.asyncio
async def test_get_user_bookings_date_time_admin(client):
    accessTokenAdmin = Settings().accessTokenAdmin
    date = "2023-05-16T00:00:00"
    response = client.get(f"/bookings?userId=6460
   eb325223bb5768aefdc9&date={date}&time=0", headers={"
   Authorization": f"Bearer {accessTokenAdmin}"})
    assert response.status_code == 200
    assert response.json() != []
    for booking in response.json():
        assert booking["startDate"] == date
        assert booking["userId"] == "6460eb325223bb5768aefdc9"
        assert booking["startTime"] == 0
```

```
@pytest.mark.asyncio
async def test_get_user_bookings_date_time_chargerId_admin(
    client):
     accessTokenAdmin = Settings().accessTokenAdmin
     date = "2023-05-16T00:00:00"
     response = client.get(f"/bookings?userId=6460
    eb325223bb5768aefdc9&date={date}&time=0&chargerId=63
    e384ad3ba66bd7e65b5620", headers={"Authorization": f"Bearer
    {accessTokenAdmin}"})
     assert response.status_code == 200
     assert response.json() != []
     for booking in response.json():
          assert booking["startDate"] == date
          assert booking["userId"] == "6460eb325223bb5768aefdc9"
          assert booking["startTime"] == 0
          assert booking["chargerId"] == "63
    e384ad3ba66bd7e65b5620"


@pytest.mark.asyncio
async def test_get_user_bookings_sort_date_asc(client):
     accessTokenAdmin = Settings().accessTokenAdmin
     response = client.get(f"/bookings", headers={"Authorization
    ": f"Bearer {accessTokenAdmin}"})
     responseAsc = client.get(f"/bookings?sort=date&order=asc",
    headers={"Authorization": f"Bearer {accessTokenAdmin}"})
     assert response.status_code == 200
     assert responseAsc.status_code == 200
     assert response.json() != []
     assert responseAsc.json() != []
     assert response.json() != responseAsc.json()
```

```
@pytest.mark.asyncio
async def test_get_user_bookings_sort_date_desc(client):
    accessTokenAdmin = Settings().accessTokenAdmin
    response = client.get(f"/bookings", headers={"Authorization
   ": f"Bearer {accessTokenAdmin}"})
    responseAsc = client.get(f"/bookings?sort=date&order=asc",
   headers={"Authorization": f"Bearer {accessTokenAdmin}"})
    responseDesc = client.get(f"/bookings?sort=date&order=desc
   ", headers={"Authorization": f"Bearer {accessTokenAdmin}"})
    assert response.status_code == 200
    assert responseAsc.status_code == 200
    assert responseDesc.status_code == 200
    assert response.json() != []
    assert responseAsc.json() != []
    assert responseDesc.json() != []
    assert responseAsc.json() != responseDesc.json()

@pytest.mark.asyncio
async def test_get_users_bookings_sort_charger_asc(client):
    accessTokenAdmin = Settings().accessTokenAdmin
    response = client.get(f"/bookings", headers={"Authorization
   ": f"Bearer {accessTokenAdmin}"})
    responseAsc = client.get(f"/bookings?sort=charger&order=asc
   ", headers={"Authorization": f"Bearer {accessTokenAdmin}"})
    assert response.status_code == 200
    assert responseAsc.status_code == 200
    assert response.json() != []
    assert responseAsc.json() != []
    assert response.json() == responseAsc.json()

@pytest.mark.asyncio
```

```
async def test_get_users_bookings_sort_charger_desc(client):
    accessTokenAdmin = Settings().accessTokenAdmin
    response = client.get(f"/bookings", headers={"Authorization
   ": f"Bearer {accessTokenAdmin}"})
    responseAsc = client.get(f"/bookings?sort=charger&order=asc
   ", headers={"Authorization": f"Bearer {accessTokenAdmin}"})
    responseDesc = client.get(f"/bookings?sort=charger&order=
   desc", headers={"Authorization": f"Bearer {accessTokenAdmin
   }"})
    print(responseAsc.json())
    print(responseDesc.json())
    assert response.status_code == 200
    assert responseAsc.status_code == 200
    assert responseDesc.status_code == 200
    assert response.json() != []
    assert responseAsc.json() != []
    assert responseDesc.json() != []
    assert responseAsc.json() != responseDesc.json()

@pytest.mark.asyncio
async def test_get_users_bookings_sort_user_asc(client):
    accessTokenAdmin = Settings().accessTokenAdmin
    response = client.get(f"/bookings", headers={"Authorization
   ": f"Bearer {accessTokenAdmin}"})
    responseAsc = client.get(f"/bookings?sort=user&order=asc",
   headers={"Authorization": f"Bearer {accessTokenAdmin}"})
    assert response.status_code == 200
    assert responseAsc.status_code == 200
    assert response.json() != []
    assert responseAsc.json() != []
    assert response.json() != responseAsc.json()
```

```
@pytest.mark.asyncio
async def test_get_users_bookings_sort_user_desc(client):
    accessTokenAdmin = Settings().accessTokenAdmin
    response = client.get(f"/bookings", headers={"Authorization
    ": f"Bearer {accessTokenAdmin}"})
    responseAsc = client.get(f"/bookings?sort=user&order=asc",
    headers={"Authorization": f"Bearer {accessTokenAdmin}"})
    responseDesc = client.get(f"/bookings?sort=user&order=desc
    ", headers={"Authorization": f"Bearer {accessTokenAdmin}"})
    assert response.status_code == 200
    assert responseAsc.status_code == 200
    assert responseDesc.status_code == 200
    assert response.json() != []
    assert responseAsc.json() != []
    assert responseDesc.json() != []
    assert responseAsc.json() != responseDesc.json()
```

Por otro lado, el administrador también puede eliminar reservas efectuadas por cualquier usuario, para probarlo, se ha creado el test test_delete_user_booking_admin para el caso en el que la reserva existe y test_delete_user_booking_not_found_admin para el caso en el que la reserva no existe, además, usa la herramienta *mock*, ya que existen dependencias con otros microservicios. Esta funcionalidad es visible en los botones destinados para ello como se puede ver en la figura 41.

```
@pytest.mark.asyncio
async def test_delete_user_booking_admin(restore_client_delete,
    client):
    accessTokenAdmin = Settings().accessTokenAdmin
    bookingsListLengthBefore = MagicMock(return_value=5)
    bookingsListLenthAfter = MagicMock(return_value=4)
    client.delete = MagicMock(return_value=Response(status_code
```

```
    =200))
    response = client.delete("/bookings/63
    e384ad3ba66bd7e65b5620")
    assert response.status_code == 200
    assert bookingsListLengthBefore != bookingsListLenthAfter


@pytest.mark.asyncio
async def test_delete_user_booking_not_found_admin(
    restore_client_delete, client):
    accessTokenAdmin = Settings().accessTokenAdmin
    client.delete = MagicMock(return_value=Response(status_code
    =404))
    response = client.delete("/bookings/63
    e384ad3ba66bd7e65b5621")
    assert response.status_code == 404
```

Por otro lado, si una reserva está en uso, es decir, el usuario ha desbloqueado un cargador, el administrador no puede eliminarla, para comprobar esta funcionalidad se ha creado el test test_delete_user_booking_already_in_use, de nuevo, se ha usado un *mock* debido a dependencias con otros microservicios:

```
@pytest.mark.asyncio
async def test_delete_user_booking_already_in_use(
    restore_client_delete, client):
    accessTokenAdmin = Settings().accessTokenAdmin
    client.delete = MagicMock(return_value=Response(status_code
    =400))
    response = client.delete("/bookings/63
    e384ad3ba66bd7e65b5622")
    assert response.status_code == 400
```

Esta funcionalidad se puede observar en la figura 42, donde el administrador recibe el mensaje de error en la interfaz de usuario.

Figura 42: Interfaz web mostrando un error de intento de eliminación de reserva en uso

Además, el administrador puede ver aquellas reservas que han sido eliminadas o usadas, que a vista del administrador poseen la misma connotación, así pues, se ha creado un test llamado test_get_user_bookings_deleted_admin, funcionalidad visible en la figura 43:

```
@pytest.mark.asyncio
async def test_get_user_bookings_deleted_admin(client):
    accessTokenAdmin = Settings().accessTokenAdmin
    response = client.get(f"/bookings", headers={"Authorization
  ": f"Bearer {accessTokenAdmin}"})
    responseDeleted = client.get(f"/bookings?deleted=true",
  headers={"Authorization": f"Bearer {accessTokenAdmin}"})
    for booking in responseDeleted.json():
        assert booking["exists"] == False
    assert response.status_code == 200
    assert responseDeleted.status_code == 200
    assert response.json() != []
    assert responseDeleted.json() != []
    assert response.json() != responseDeleted.json()
```

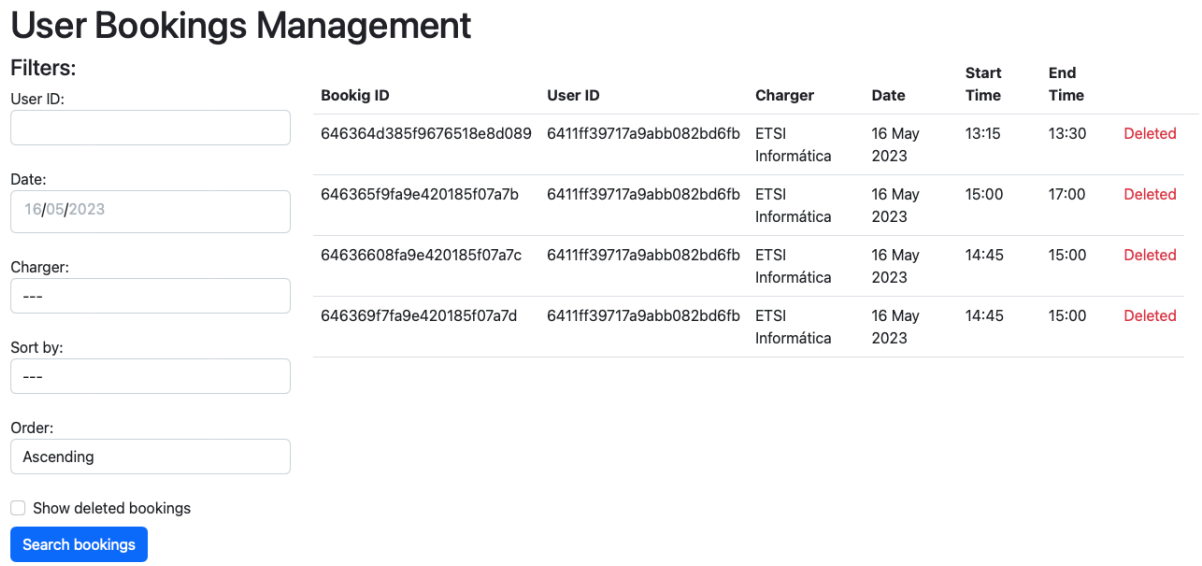

Figura 43: Interfaz web mostrando las reservas eliminadas del sistema

Así pues, al ejecutar el comando pytest se comprueba que los tests arrojan un resultado positivo, quedando validados los requisitos ARF-02, ARF-02.1, ARF-02.2, ARF-02.3, ARF-02.4, ARF-02.5, ARF-02.6, ARF-02.7, ARF-02.8 y ARF-02.9

6.2.3. El sistema permite al administrador crear cargadores para el uso de los mismos dentro de la aplicación.

Para validar el presente requisito, se ha creadon tests llamados test_create_charger_valid_admin para el caso correcto, test_create_charger_same_ocpp_id_admin para el caso en el que se quiera crear un cargador con *ID* de *OCPP* duplicado y test_create_charger_same_ocpp_id_not_admin para el caso en el que un usuario malicioso intentara crear cargadores:

```
@pytest.mark.asyncio
async def test_create_charger_valid_admin(client):
    accessToken = Settings().accessTokenAdmin
    chargersListBefore = client.get("/chargers", headers={"
  Authorization": f"Bearer {accessToken}"})
```

```
    response = client.post("/chargers",
        headers={"Authorization": f"Bearer {accessToken}"},
        json={
            "name": "testCharger",
            "locationDescription": "testLocation",
            "latitude": 0.0,
            "longitude": 0.0,
            "ocppId": 5,
            "image": "testImage"
        }
    )
    chargersListAfter = client.get("/chargers", headers={"
   Authorization": f"Bearer {accessToken}"})
    assert response.status_code == 200
    assert response.json() != []
    assert len(chargersListAfter.json()) == len(
   chargersListBefore.json()) + 1


@pytest.mark.asyncio
async def test_create_charger_same_ocpp_id_admin(client):
    accessToken = Settings().accessTokenAdmin
    response = client.post("/chargers",
        headers={"Authorization": f"Bearer {accessToken}"},
        json={
            "name": "testCharger",
            "locationDescription": "testLocation",
            "latitude": 0.0,
            "longitude": 0.0,
            "ocppId": 5,
            "image": "testImage"
```

```
        }
    )
    assert response.status_code == 400

@pytest.mark.asyncio
async def test_create_charger_same_ocpp_id_not_admin(client):
    accessToken = Settings().validToken
    response = client.post("/chargers",
        headers={"Authorization": f"Bearer {accessToken}"},
        json={
            "name": "testCharger",
            "locationDescription": "testLocation",
            "latitude": 0.0,
            "longitude": 0.0,
            "ocppId": 5,
            "image": "testImage"
        }
    )
    assert response.status_code == 401
```

En la interfaz web se ve reflejada la creación de un nuevo cargador como se puede observar en la figura 44, si se crea con *ID* de *OCPP* duplicado, se obtiene el error de la figura 45.

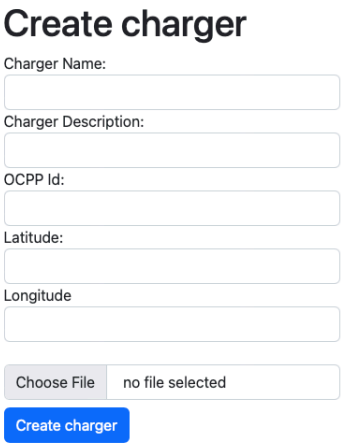

Figura 44: Interfaz web mostrando un formulario para la creación de cargadores en el sistema

Figura 45: Interfaz web mostrando un error de duplicidad de identificadores *OCPP*

Así pues, al ejecutar el comando pytest se comprueba que los tests arrojan un resultado positivo, quedando validados los requisitos ARF-03 y ARNF-01.

6.2.4. El sistema permite al administrador ver todos los cargadores existentes y eliminados existentes en el sistema.

Para validar este requisito se ha creado un test llamado test_get_chargers_admin, además de sus correspondientes tests para obtener información sobre un cargador en concreto: test_get_charger_by_id_admin para el caso en el que existe y test_get_charger_by_id_not_found_admin para el caso en el que no existe. Esta funcionalidad en la interfaz web se puede observar en la figura 46.

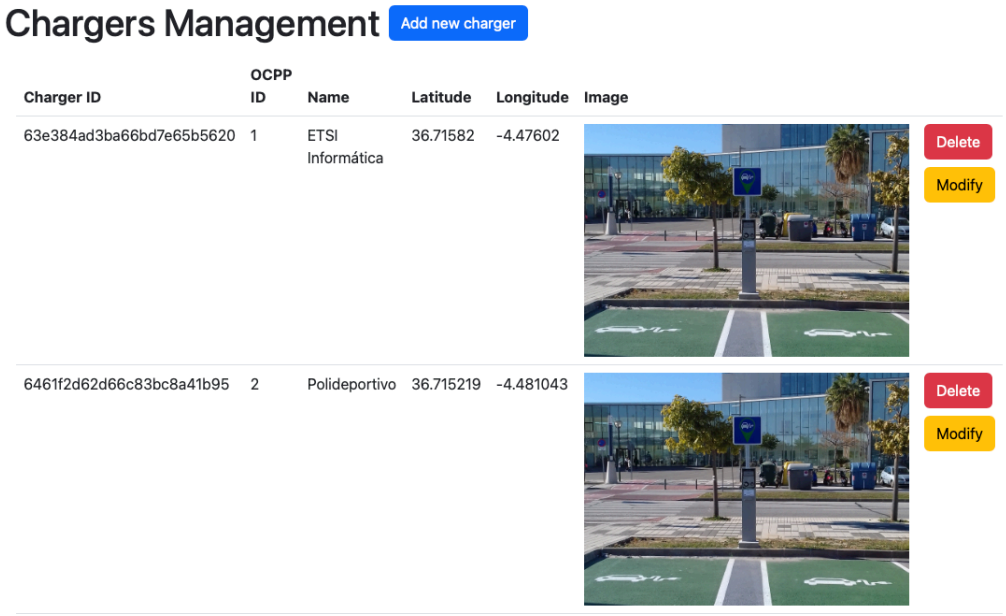

Figura 46: Interfaz web mostrando los cargadores en el sistema

```
@pytest.mark.asyncio
async def test_get_chargers_admin(client):
    accessToken = Settings().accessTokenAdmin
    response = client.get("/chargers", headers={"Authorization
   ": f"Bearer {accessToken}"})
    assert response.status_code == 200
    assert response.json() != []

@pytest.mark.asyncio
async def test_get_charger_by_id_admin(client):
    accessToken = Settings().accessTokenAdmin
    response = client.get("/chargers/63e384ad3ba66bd7e65b5620",
    headers={"Authorization": f"Bearer {accessToken}"})
    assert response.status_code == 200
    assert response.json() != []

@pytest.mark.asyncio
```

```
async def test_get_charger_by_id_not_found_admin(client):
    accessToken = Settings().accessTokenAdmin
    response = client.get("/chargers/63e384ad3ba66bd7e65b5621",
    headers={"Authorization": f"Bearer {accessToken}"})
    assert response.status_code == 404
```

Así pues, al ejecutar el comando pytest se comprueba que los tests arrojan un resultado positivo, quedando validado el requisito ARF-04.

6.2.5. El sistema permite al administrador eliminar cargadores existentes en el sistema.

Para validar este requisito se ha creado el test test_delete_charger_valid_admin para el caso de éxito, test_delete_charger_valid_not_admin para el caso en el que un usuario malintencionado intente eliminar un cargador de manera no autorizada, test_delete_charger_not_found_admin para el caso en el que no exista el cargador que se quiere eliminar y test_delete_charger_invalid_id_admin para el caso en el que el identificador no sea válido. La funcionalidad de eliminar cargadores está disponible en la interfaz web de administradores, tal y como se puede observar en la figura 46.

```
@pytest.mark.asyncio
async def test_delete_charger_valid_admin(client):
    accessToken = Settings().accessTokenAdmin
    chargers = client.get("/chargers",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    chargerId = "6463a84fb41cd9f01f7cd4ad"
    response = client.delete(f"/chargers/{chargerId}",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    chargersBefore = client.get("/chargers",
        headers={"Authorization": f"Bearer {accessToken}"})
```

```
    assert response.status_code == 200
    assert response.json()["message"] == "Charger deleted"
    assert chargersBefore != chargers

@pytest.mark.asyncio
async def test_delete_charger_valid_not_admin(client):
    accessToken = Settings().validToken
    chargers = client.get("/chargers",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    chargerId = "6463a84fb41cd9f01f7cd4ad"
    response = client.delete(f"/chargers/{chargerId}",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    assert response.status_code == 401

@pytest.mark.asyncio
async def test_delete_charger_not_found_admin(client):
    accessToken = Settings().accessTokenAdmin
    response = client.delete("/chargers/63
   e384ad3ba66bd7e65b5621",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    assert response.status_code == 404

@pytest.mark.asyncio
async def test_delete_charger_invalid_id_admin(client):
    accessToken = Settings().accessTokenAdmin
    response = client.delete("/chargers/123",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
```

```
    assert response.status_code == 422
```

Así pues, al ejecutar el comando pytest se comprueba que los tests arrojan un resultado positivo, quedando validado el requisito ARF-05.

6.2.6. El sistema permite al administrador modificar los datos de los cargadores existentes en el sistema.

Para validar este requisito se ha definido un test llamado test_modify_charger_valid_admin para el caso de éxito, test_modify_charger_no_data_admin para el caso en el que se hace la llamada sin datos y un test por cada elemento a modificar dentro del cargador.

```
@pytest.mark.asyncio
async def test_modify_charger_valid_admin(client):
    accessToken = Settings().accessTokenAdmin
    chargers = client.get("/chargers",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    chargerId = chargers.json()[len(chargers.json()) - 1]["_id
   "]
    originalCharger = client.get(f"/chargers/{chargerId}",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    response = client.put(f"/chargers/{chargerId}",
        headers={"Authorization": f"Bearer {accessToken}"},
        json={
            "name": "testChargerModified",
            "locationDescription": "testLocationModified",
            "latitude": 0.1,
            "longitude": 0.1,
            "ocppId": 6,
            "image": "testImageModified"
        }
```

```
    )
    assert response.status_code == 200
    assert response.json() != []
    assert response.json()["name"] != originalCharger.json()["
    name"]
    assert response.json()["locationDescription"] !=
    originalCharger.json()["locationDescription"]
    assert response.json()["latitude"] != originalCharger.json
    ()["latitude"]
    assert response.json()["longitude"] != originalCharger.json
    ()["longitude"]
    assert response.json()["ocppId"] != originalCharger.json()
    ["ocppId"]
    assert response.json()["image"] != originalCharger.json()["
    image"]

@pytest.mark.asyncio
async def test_modify_charger_no_data_admin(client):
    accessToken = Settings().accessTokenAdmin
    chargers = client.get("/chargers",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    chargerId = chargers.json()[len(chargers.json()) - 1]["_id
    "]
    response = client.put(f"/chargers/{chargerId}",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    assert response.status_code == 422

@pytest.mark.asyncio
async def test_modify_charger_name_admin(client):
```

```
    accessToken = Settings().accessTokenAdmin
    chargers = client.get("/chargers",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    chargerId = chargers.json()[len(chargers.json()) - 1]["_id
   "]
    originalCharger = client.get(f"/chargers/{chargerId}",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    response = client.put(f"/chargers/{chargerId}",
        headers={"Authorization": f"Bearer {accessToken}"},
        json={
            "name": "testChargerModified1",
        }
    )
    assert response.status_code == 200
    assert response.json() != []
    assert response.json()["name"] != originalCharger.json()["
   name"]

@pytest.mark.asyncio
async def test_modify_charger_location_description_admin(client
   ):
    accessToken = Settings().accessTokenAdmin
    chargers = client.get("/chargers",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    chargerId = chargers.json()[len(chargers.json()) - 1]["_id
   "]
    originalCharger = client.get(f"/chargers/{chargerId}",
        headers={"Authorization": f"Bearer {accessToken}"}
```

```
    )
    response = client.put(f"/chargers/{chargerId}",
        headers={"Authorization": f"Bearer {accessToken}"},
        json={
            "locationDescription": "testLocationModified1",
        }
    )
    assert response.status_code == 200
    assert response.json() != []
    assert response.json()["locationDescription"] !=
   originalCharger.json()["locationDescription"]

@pytest.mark.asyncio
async def test_modify_charger_latitude_admin(client):
    accessToken = Settings().accessTokenAdmin
    chargers = client.get("/chargers",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    chargerId = chargers.json()[len(chargers.json()) - 1]["_id
   "]
    originalCharger = client.get(f"/chargers/{chargerId}",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    response = client.put(f"/chargers/{chargerId}",
        headers={"Authorization": f"Bearer {accessToken}"},
        json={
            "latitude": 0.2,
        }
    )
    assert response.status_code == 200
    assert response.json() != []
```

```
    assert response.json()["latitude"] != originalCharger.json
    ()["latitude"]

@pytest.mark.asyncio
async def test_modify_charger_longitude_admin(client):
    accessToken = Settings().accessTokenAdmin
    chargers = client.get("/chargers",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    chargerId = chargers.json()[len(chargers.json()) - 1]["_id
    "]
    originalCharger = client.get(f"/chargers/{chargerId}",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    response = client.put(f"/chargers/{chargerId}",
        headers={"Authorization": f"Bearer {accessToken}"},
        json={
            "longitude": 0.2,
        }
    )
    assert response.status_code == 200
    assert response.json() != []
    assert response.json()["longitude"] != originalCharger.json
    ()["longitude"]

@pytest.mark.asyncio
async def test_modify_charger_ocpp_id_admin(client):
    accessToken = Settings().accessTokenAdmin
    chargers = client.get("/chargers",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
```

```
    chargerId = chargers.json()[len(chargers.json()) - 1]["_id
    "]
    originalCharger = client.get(f"/chargers/{chargerId}",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    response = client.put(f"/chargers/{chargerId}",
        headers={"Authorization": f"Bearer {accessToken}"},
        json={
            "ocppId": 7,
        }
    )
    assert response.status_code == 200
    assert response.json() != []
    assert response.json()["ocppId"] != originalCharger.json()
    ["ocppId"]

@pytest.mark.asyncio
async def test_modify_charger_image_admin(client):
    accessToken = Settings().accessTokenAdmin
    chargers = client.get("/chargers",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    chargerId = chargers.json()[len(chargers.json()) - 1]["_id
    "]
    originalCharger = client.get(f"/chargers/{chargerId}",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    response = client.put(f"/chargers/{chargerId}",
        headers={"Authorization": f"Bearer {accessToken}"},
        json={
            "image": "testImageModified1",
```

```
        }
    )
    assert response.status_code == 200
    assert response.json() != []
    assert response.json()["image"] != originalCharger.json()["
image"]
```

Además, se ha creado el test test_modify_charger_same_ocpp_id_admin para comprobar el caso en el que si quiera cambiar el *ID* de *OCPP* de un cargador a otro que ya ha sido asignado a otro cargador. En la interfaz web no se permite la edición de este parámetro, pero se valida en caso de hacerse mediante petición al servidor.

```
@pytest.mark.asyncio
async def test_modify_charger_same_ocpp_id_admin(client):
    accessToken = Settings().accessTokenAdmin
    chargers = client.get("/chargers",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    chargerId = chargers.json()[len(chargers.json()) - 1]["_id
"]
    originalCharger = client.get(f"/chargers/{chargerId}",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    response = client.put(f"/chargers/{chargerId}",
        headers={"Authorization": f"Bearer {accessToken}"},
        json={
            "name": "testChargerModified",
            "locationDescription": "testLocationModified",
            "latitude": 0.1,
            "longitude": 0.1,
            "ocppId": 7,
            "image": "testImageModified"
```

```
        }
    )
    assert response.status_code == 400
```

Por otro lado, se ha contemplado el caso en el que un usuario malicioso intente modificar los datos de un cargador sin los permisos necesarios, para ello, se ha creado el test test_modify_charger_id_not_admin:

```
@pytest.mark.asyncio
async def test_modify_charger_id_not_admin(client):
    accessToken = Settings().validToken
    chargers = client.get("/chargers",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    chargerId = chargers.json()[len(chargers.json()) - 1]["_id
   "]
    originalCharger = client.get(f"/chargers/{chargerId}",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    response = client.put(f"/chargers/{chargerId}",
        headers={"Authorization": f"Bearer {accessToken}"},
        json={
            "name": "testChargerModified",
            "locationDescription": "testLocationModified",
            "latitude": 0.1,
            "longitude": 0.1,
            "ocppId": 6,
            "image": "testImageModified"
        }
    )
    assert response.status_code == 403
```

Por último, también se han contemplado los casos en los que se intente modificar un car-

gador que no existe en el sistema o se proporcione un identificador no válido con los tests test_modify_charger_id_not_found_admin y test_modify_charger_invalid_id_admin:

```
@pytest.mark.asyncio
async def test_modify_charger_id_not_found_admin(client):
    accessToken = Settings().accessTokenAdmin
    response = client.put("/chargers/63e384ad3ba66bd7e65b5621",
        headers={"Authorization": f"Bearer {accessToken}"},
        json={
            "name": "testChargerModified",
            "locationDescription": "testLocationModified",
            "latitude": 0.1,
            "longitude": 0.1,
            "ocppId": 10,
            "image": "testImageModified"
        }
    )
    assert response.status_code == 404

@pytest.mark.asyncio
async def test_modify_charger_invalid_id_admin(client):
    accessToken = Settings().accessTokenAdmin
    response = client.put("/chargers/123",
        headers={"Authorization": f"Bearer {accessToken}"},
        json={
            "name": "testChargerModified",
            "locationDescription": "testLocationModified",
            "latitude": 0.1,
            "longitude": 0.1,
            "ocppId": 10,
            "image": "testImageModified"
        }
```

```
    )
    assert response.status_code == 422
```

Esta funcionalidad se expone en la interfaz web como se puede observar en la figura 46, pudiendo editar cargadores como se puede observar en la figura 47.

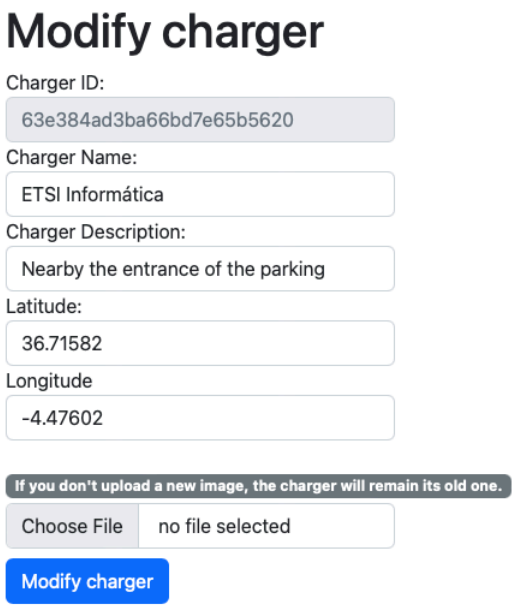

Figura 47: Interfaz web mostrando el formulario de modificación de datos de un cargador concreto

Así pues, al ejecutar el comando pytest se comprueba que los tests arrojan un resultado positivo, quedando validado el requisito ARF-06.

6.2.7. El sistema permite al administrador ver todos los tiquets de incidencias existentes en el sistema.

Para validar este requisito se ha creado un test llamado test_get_tickets_admin, el cual comprueba que el administrador recibe los tiquets existentes en el sistema (se da por supuesta la existencia de tiquets en el sistema). Esta funcionalidad está disponible en la interfaz web, obsérvese la figura 48

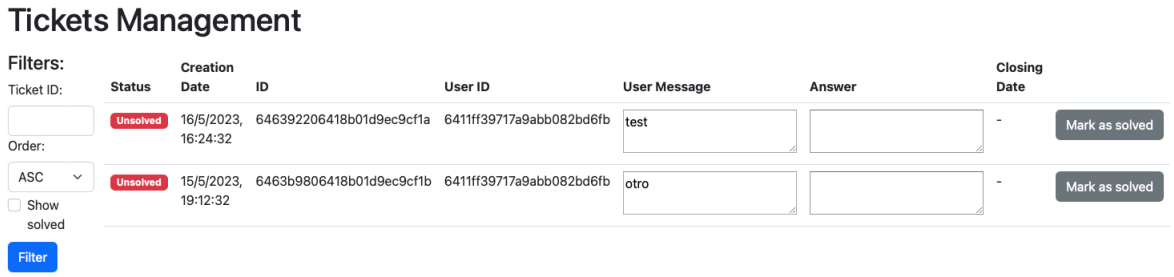

Figura 48: Interfaz web mostrando los tiquets de incidencia en el sistema

```
@pytest.mark.asyncio
async def test_get_tickets_admin(client):
    accessToken = Settings().accessTokenAdmin
    response = client.get("/tickets?solved=false",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    assert response.status_code == 200
    assert response.json() != []
```

Por otro lado, se ha tenido en cuenta el caso en el que un usuario malintencionado intente acceder a los tiquets del sistema sin autorización previa, para ello se ha creado el test test_get_tickets_user, que deniega el acceso a aquellos usuarios que no son administradores:

```
@pytest.mark.asyncio
async def test_get_tickets_user(client):
    accessToken = Settings().validToken
    response = client.get("/tickets?solved=false",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    assert response.status_code == 403
```

Además, se han creado tests para validar el filtrado y ordenado de los tiquets en el sistema con los tests test_get_tickets_filter_solved para filtrar por tiquets cerrados o no,

test_get_tickets_filter_ticket_id para buscar un tiquet en concreto que esté sin cerrar y test_get_tickets_order_asc_desc para ordenarlos en order ascendente o descendente con respecto a su fecha de creación:

```
@pytest.mark.asyncio
async def test_get_tickets_filter_solved(client):
    accessToken = Settings().accessTokenAdmin
    unSolved = client.get("/tickets?solved=false",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    response = client.get("/tickets?solved=true",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    assert response.status_code == 200
    assert response.json() != []
    assert response.json() != unSolved.json()
    for ticket in response.json():
        assert ticket["solved"] == True
    for ticket in unSolved.json():
        assert ticket["solved"] == False

@pytest.mark.asyncio
async def test_get_tickets_filter_ticket_id(client):
    accessToken = Settings().accessTokenAdmin
    response = client.get("/tickets/646392206418b01d9ec9cf1a?
   solved=false",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    assert response.status_code == 200
    assert response.json() != []
    for ticket in response.json():
        assert ticket["_id"] == "646392206418b01d9ec9cf1a"
```

```
@pytest.mark.asyncio
async def test_get_tickets_order_asc_desc(client):
    accessToken = Settings().accessTokenAdmin
    responseAsc = client.get("/tickets?solved=false&order=asc",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    responseDesc = client.get("/tickets?solved=false&order=desc
   ",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    assert responseAsc.status_code == 200
    assert responseDesc.status_code == 200
    assert responseAsc.json() != []
    assert responseDesc.json() != []
    assert responseAsc.json() != responseDesc.json()
```

Por otro lado, el administrador puede resolver incidencias indicando o no un mensaje para el usuario, para validar esta funcionalidad, visible en la interfaz gráfica a través de la figura 48, se han creado varios tests, entre los que se encuentran test_close_ticket_with_message_admin para cerrar un tiquet con mensaje y test_close_ticket_without_message_admin para cerrar un tiquet sin mensaje:

```
@pytest.mark.asyncio
async def test_close_ticket_with_message_admin(client):
    accessToken = Settings().accessTokenAdmin
    ticketList = client.get("/tickets?solved=false",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    close = client.put("/tickets/646392206418b01d9ec9cf1a?
   solved=true&adminId=6410b5aa51dad87bbce38435&adminMsg=test&
   solvingDate=2023-05-16T19:30:00.000",
```

```
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    ticketListAfter = client.get("/tickets?solved=false",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    assert ticketList.status_code == 200
    assert ticketListAfter.status_code == 200
    assert close.status_code == 200
    assert close.json() != []
    assert ticketList.json() != []
    assert ticketListAfter.json() != []
    assert ticketList.json() != ticketListAfter.json()

@pytest.mark.asyncio
async def test_close_ticket_without_message_admin(client):
    accessToken = Settings().accessTokenAdmin
    ticketList = client.get("/tickets?solved=false",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    close = client.put("/tickets/6463b9806418b01d9ec9cf1b?
   solved=true&adminId=6410b5aa51dad87bbce38435&adminMsg=&
   solvingDate=2023-05-16T19:30:00.000",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    ticketListAfter = client.get("/tickets?solved=false",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    assert ticketList.status_code == 200
    assert ticketListAfter.status_code == 200
    assert close.status_code == 200
    assert close.json() != []
```

```
    assert ticketList.json() != []
    assert ticketListAfter.json() != []
    assert ticketList.json() != ticketListAfter.json()
```

En la figura 49 se puede observar la interfaz web para visualizar aquellos tickets que han sido cerrados por un administrador.

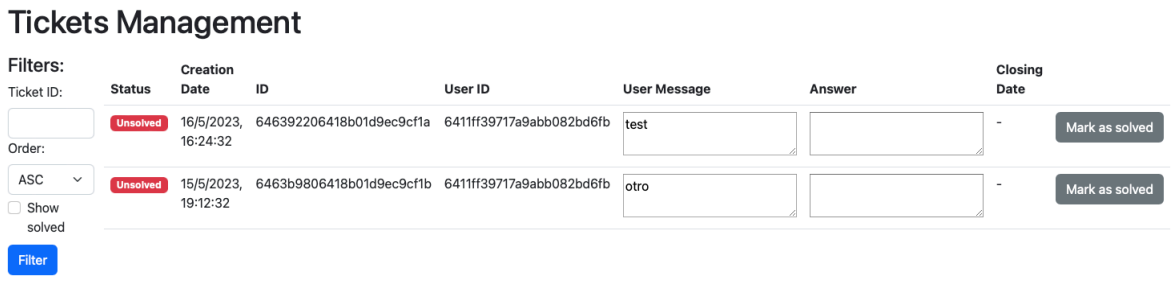

Figura 49: Interfaz web mostrando los tiquets de incidencia cerrados por un administrador en el sistema

Para finalizar, se han tenido en cuenta dos situaciones: se quiere cerrar un tiquet que no existe o un usuario malintencionado quiere cerrar un tiquet sin autorización. Para validar ambas situaciones se han creado los tests test_close_ticket_not_found_admin y test_close_ticket_user:

```
@pytest.mark.asyncio
async def test_close_ticket_not_found_admin(client):
    accessToken = Settings().accessTokenAdmin
    close = client.put("/tickets/6463b9806418b01d9ec9cf2a?
   solved=true&adminId=6410b5aa51dad87bbce38435&solvingDate
   =2023-05-16T19:30:00.000",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    assert close.status_code == 404

@pytest.mark.asyncio
```

```
async def test_close_ticket_user(client):
    accessToken = Settings().validToken
    close = client.put("/tickets/6463b9806418b01d9ec9cf1b?
  solved=true&adminId=6410b5aa51dad87bbce38435&solvingDate
  =2023-05-16T19:30:00.000",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    assert close.status_code == 401
```

Así pues, al ejecutar el comando pytest se comprueba que los tests arrojan un resultado positivo, quedando validados los requisitos ARF-07, ARF-07.1, ARF-07.2, ARF-07.3 y ARF-08.

6.2.8. El sistema permite al administrador ver estadísticas de uso y coste del sistema referentes a las reservas.

Para validar este requisito, se ha creado un test llamado test_get_stats_admin que comprueba que al realizar la llamada al servidor se devuelven datos. Esta funcionalidad se puede comprobar observando la figura 50.

```
@pytest.mark.asyncio
async def test_get_stats_admin(client):
    accessTokenAdmin = Settings().accessTokenAdmin
    response = client.get("/stats", headers={"Authorization": f
  "Bearer {accessTokenAdmin}"})
    lastWeek = [
        (datetime.today() - timedelta(days=i)).strftime('%Y-%m
  -%d')
        for i in range(0, 7)
    ]
    assert response.status_code == 200
    assert response.json() != []
    assert isinstance(response.json()["totalBookings"], int)
    assert isinstance(response.json()["totalHours"], float)
```

```
    assert isinstance(response.json()["totalPrice"], float)
    assert isinstance(response.json()["avgPrice"], float)
    assert isinstance(response.json()["avgHours"], float)
    assert isinstance(response.json()["avgWeeklyBookings"],
float)
    assert response.json()["lastWeekBookingsPrice"] != []
    for date in response.json()["lastWeekBookingsPrice"]:
        assert date in lastWeek
    for date in response.json()["
lastWeekBookingsElectricityPrice"]:
        assert date in lastWeek
```

Figura 50: Interfaz web mostrando las estadísticas de uso y coste del sistema referentes a las reservas

Por otro lado, también se ha comprobado la funcionalidad de selección de fecha con el test test_get_stats_date, que comprueba que los días proporcionados son los seleccionados por el administrador:

```
@pytest.mark.asyncio
```

```
async def test_get_stats_date(client):
    accessTokenAdmin = Settings().accessTokenAdmin
    today = datetime.today()
    today = today.replace(hour=0, minute=0, second=0,
  microsecond=0)
    lastWeek = [
        (today - timedelta(days=i)).strftime('%Y-%m-%d')
        for i in range(0, 7)
    ]
    response = client.get(f"/stats?date={today.isoformat()}",
  headers={"Authorization": f"Bearer {accessTokenAdmin}"})
    lastWeekBookingsPrice = response.json()["
  lastWeekBookingsPrice"]
    assert response.json() != []
    assert response.status_code == 200
    for date in lastWeekBookingsPrice:
        assert date in lastWeek
```

Así pues, al ejecutar el comando pytest se comprueba que los tests arrojan un resultado positivo, quedando validado el requisito ARF-09.

6.2.9. El sistema permite al administrador ver todos los registros o logs del sistema.

Para la validación de este requisito, se han creado varios tests: el primero, test_get_logs_pages_admin para calcular el número de páginas a mostrar en la interfaz web dada un a longitud de página y test_get_logs_admin para obtener los logs dado el número de página y su longitud:

```
@pytest.mark.asyncio
async def test_get_logs_pages_admin(client):
    accessToken = Settings().accessTokenAdmin
    pages = client.get("/logs/pages?pageLength=10",
        headers={"Authorization": f"Bearer {accessToken}"}
```

```
    )
    print(pages.json())
    assert pages.status_code == 200
    assert pages.json() != []
    assert isinstance(pages.json(), int)

@pytest.mark.asyncio
async def test_get_logs_admin(client):
    accessToken = Settings().accessTokenAdmin
    logs = client.get("/logs?pageNumber=1&pageLength=10",
   headers={"Authorization": f"Bearer {accessToken}"})
    assert logs.status_code == 200
    assert logs.json() != []
    for log in logs.json():
        assert log["_id"] is not None
        assert log["userId"] is not None
        assert log["date"] is not None
        assert log["action"] is not None
```

Observando la figura 51 se puede verificar esta funcionalidad en la interfaz web, además de la funcionalidad de descarga por páginas en formato *JSON*.

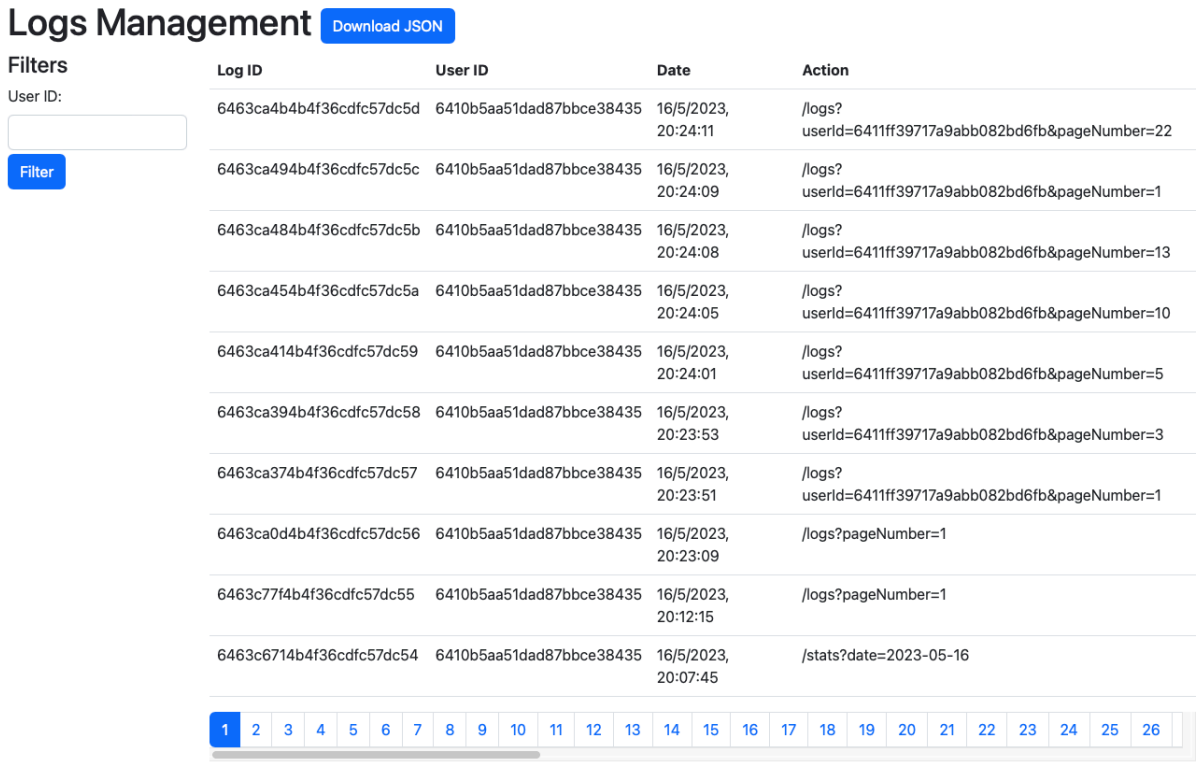

Figura 51: Interfaz web mostrando los registros del sistema

Además, para validar el filtrado de registros por *ID* de usuario, se ha creado el test test_get_logs_by_user_id, que permite verificar que al filtrar los registros, todos los obtenidos son del usuario que se ha requerido:

```
@pytest.mark.asyncio
async def test_get_logs_by_user_id(client):
    accessToken = Settings().accessTokenAdmin
    logs = client.get("/logs?pageNumber=1&pageLength=10&userId
  =6411ff39717a9abb082bd6fb",
        headers={"Authorization": f"Bearer {accessToken}"}
    )
    assert logs.status_code == 200
    assert logs.json() != []
    for log in logs.json():
        assert log["userId"] == "6411ff39717a9abb082bd6fb"
```

Así pues, al ejecutar el comando pytest se comprueba que los tests arrojan un resultado positivo, quedando validados los requisitos ARF-10, ARF-10.1 y ARF-10.2.

7

Conclusiones y líneas futuras de investigación

7.1. Conclusiones

En el presente proyecto, se ha llevado a cabo una investigación exhaustiva y un desarrollo detallado de dos aplicaciones web cuya funcionalidad es la gestión, uso y reservas de los cargadores eléctricos disponibles dentro de un *Smart Campus*, que utilizan el protocolo *OCPP* para la conexión con los cargadores. A lo largo de este proyecto, se ha demostrado la viabilidad y la efectividad de implementar esta solución para satisfacer las necesidades de los usuarios y optimizar la utilización de los recursos energéticos.

La integración de *OCPP* en las aplicaciones web ha permitido establecer una comunicación fluida y confiable entre los cargadores y el sistema de gestión. Los usuarios ahora tienen la capacidad de visualizar la disponibilidad en tiempo real de los cargadores, realizar reservas, monitorizar el estado de carga y acceder a estadísticas relevantes. Esto ha brindado una mayor comodidad y accesibilidad para los usuarios, así como una planificación más eficiente y una mejor utilización de los recursos energéticos disponibles.

Además, se ha diseñado una interfaz intuitiva y amigable para los usuarios, lo que facilita la navegación y el uso de las diversas funcionalidades proporcionadas por las aplicaciones web. Se han implementado sólidas medidas de seguridad y autenticación para garantizar la privacidad y protección de los datos de los usuarios, así como la integridad del sistema en general.

Los resultados obtenidos de este proyecto demuestran que la implementación del protocolo *OCPP* en la gestión de cargadores dentro de un *Smart Campus* puede generar beneficios

significativos en términos de eficiencia energética, optimización de recursos y satisfacción del usuario. Los usuarios ahora pueden aprovechar al máximo los cargadores disponibles, evitando tiempos de espera innecesarios y maximizando la utilización de la infraestructura de carga.

Además de los beneficios directos para los usuarios, esta solución también contribuye a la promoción de la movilidad sostenible y la reducción de las emisiones de gases de efecto invernadero. Al facilitar el acceso y la gestión eficiente de los cargadores eléctricos, se fomenta la adopción de vehículos eléctricos y se impulsa la transición hacia un transporte más limpio y respetuoso con el medio ambiente.

7.2. Líneas futuras de investigación

A pesar de los avances logrados en este proyecto, existen varias áreas de investigación y desarrollo que podrían expandir aún más las funcionalidades y la eficiencia de las aplicaciones web. Algunas líneas futuras de investigación sugeridas son:

7.2.1. Implementación de técnicas de inteligencia artificial

La implementación de técnicas de inteligencia artificial, como el aprendizaje automático *(machine learning)* y la optimización basada en algoritmos genéticos, puede brindar mejoras significativas en la eficiencia de la gestión de cargadores eléctricos y en la predicción de la demanda de carga. Estas técnicas permiten adaptar el sistema a los patrones de uso cambiantes y optimizar la asignación de recursos de manera dinámica.

Mediante el uso de algoritmos de aprendizaje automático, es posible analizar grandes cantidades de datos históricos relacionados con el uso de los cargadores, como los patrones de carga de los usuarios, la disponibilidad de los cargadores en diferentes momentos del día, la duración de las sesiones de carga, entre otros. Estos algoritmos pueden identificar patrones ocultos y tendencias en los datos, lo que permite realizar predicciones precisas sobre la demanda futura de carga. Con esta información, el sistema puede ajustar de manera proactiva la asignación de recursos, garantizando una distribución óptima de la carga y evitando congestiones en determinados momentos.

Además del aprendizaje automático, la optimización basada en algoritmos genéticos puede

ser aplicada para encontrar soluciones óptimas en la gestión de los cargadores eléctricos. Estos algoritmos imitan el proceso de selección natural y evolución biológica para encontrar combinaciones óptimas de variables en un problema determinado. En el contexto de la gestión de cargadores, los algoritmos genéticos pueden buscar la mejor asignación de cargadores a usuarios, considerando factores como la distancia entre los usuarios y los cargadores, la demanda energética de cada vehículo y la disponibilidad de los recursos.

La implementación de estas técnicas de inteligencia artificial en la gestión de cargadores eléctricos puede llevar a una mayor eficiencia en la utilización de los recursos, una distribución más equitativa de la carga y una mejor experiencia del usuario. Además, al adaptarse a los patrones de uso cambiantes, el sistema puede anticiparse a las necesidades de carga de los usuarios, evitando situaciones de sobrecarga o tiempos de espera prolongados.

No obstante, es importante destacar que la implementación de técnicas de inteligencia artificial requiere una adecuada recopilación y análisis de datos históricos, así como la disponibilidad de recursos computacionales suficientes. Además, es fundamental garantizar la privacidad y la seguridad de los datos de los usuarios durante el proceso de aprendizaje y optimización.

Así pues, la implementación de técnicas de inteligencia artificial, como el aprendizaje automático y la optimización basada en algoritmos genéticos, en la gestión de cargadores eléctricos dentro de un Campus universitario puede proporcionar beneficios significativos en términos de eficiencia, optimización de recursos y experiencia del usuario. Estas técnicas permiten adaptar el sistema a los cambios en la demanda de carga y optimizar la asignación de recursos de manera dinámica, mejorando así la eficiencia y la sostenibilidad del sistema en general.

7.2.2. Implementación de tecnologías de carga inteligente y vehículos eléctricos conectados *(V2G)*

La carga inteligente se refiere a la capacidad de los cargadores eléctricos para comunicarse y coordinarse entre sí, así como con los vehículos eléctricos, a través de una red de comunicación. Esto permite una gestión más eficiente de la carga, ya que los cargadores pueden ajustar su potencia de carga y horarios en función de la disponibilidad de energía y las necesidades de los vehículos conectados.

La tecnología *Vehicle to Grid (V2G)*, por su parte, permite a los vehículos eléctricos no solo recibir carga de los cargadores, sino también devolver energía a la red eléctrica cuando sea

necesario. Esto significa que los vehículos eléctricos pueden actuar como unidades de almacenamiento de energía móviles y contribuir a la estabilización de la red eléctrica, especialmente en momentos de alta demanda o durante apagones.

En el contexto de un Campus universitario, la implementación de tecnologías de carga inteligente y *V2G* puede proporcionar beneficios adicionales, como:

- **Gestión optimizada de la carga**: los cargadores pueden coordinarse entre sí y con los vehículos eléctricos para evitar congestiones y optimizar la asignación de recursos de carga. Esto puede ayudar a minimizar los tiempos de espera y maximizar la utilización de los cargadores disponibles.

- **Integración con energía renovable local**: al tener una comunicación bidireccional con los vehículos eléctricos, los cargadores pueden ajustar su carga para aprovechar al máximo la energía renovable generada localmente en el Campus universitario, como la energía solar o eólica. Esto promovería el uso de energías limpias y reduciría la dependencia de la red eléctrica convencional.

- **Participación activa en la gestión de la red eléctrica**: los vehículos eléctricos conectados pueden contribuir a la estabilización de la red eléctrica al devolver energía a la red en momentos de alta demanda o durante apagones. Esto podría ayudar a reducir los costos de infraestructura y mejorar la resiliencia del sistema eléctrico del Campus universitario.

- **Beneficios económicos para los usuarios**: la tecnología *V2G* permite a los propietarios de vehículos eléctricos recibir ingresos por la energía devuelta a la red eléctrica. Esto podría incentivar aún más la adopción de vehículos eléctricos y promover un modelo de movilidad sostenible en el Campus universitario.

La implementación de tecnologías de carga inteligente y *V2G* requeriría investigaciones adicionales en áreas como la comunicación entre cargadores y vehículos, el desarrollo de algoritmos de gestión de carga avanzados y el estudio de los impactos técnicos y económicos de estas tecnologías en el contexto del Campus universitario.

En resumen, la implementación de tecnologías de carga inteligente y *V2G* en la gestión de cargadores dentro de un Campus universitario puede ofrecer beneficios adicionales en térmi-

nos de gestión optimizada de la carga, integración con energía renovable local, participación activa en la gestión de la red eléctrica y beneficios económicos para los usuarios. Estas áreas representan una línea de investigación futura prometedora para mejorar aún más la eficiencia y la sostenibilidad de la gestión de cargadores eléctricos en entornos universitarios. [59]

Apéndice A
Bibliografía

[1] Universidad de Cádiz (2021, 12 de Noviembre). La UCA impulsa la recarga de vehículos eléctricos e híbridos en el marco de su estrategia de movilidad sostenible: enlace, último acceso: 10/10/2022

[2] Montañez, A. I. (2022, 9 de Febrero). Coches eléctricos en Málaga: 22 puntos de recarga ya tienen licencia: enlace, último acceso: 10/10/2022

[3] Iberdrola (s.f.). Gracias a Iberdrola, podrás recorrer España con un vehículo eléctrico: enlace, último acceso: 10/10/2022

[4] Frías Marín, P., & Román Úbeda, J. (2019). Vehículo eléctrico: situación actual y perspectivas futuras.

[5] Freyssenet, M. (2011). Lo más dudoso no es lo más improbable: el coche eléctrico. La nueva revolución del automóvil. Jornada internacional 'Movilidad sostenible y vehículo eléctrico, el motor de la innovación local, Ayuntamiento de Valladolid, Valladolid, España, Fundación CEU-San Pablo Castilla y León.

[6] Open Charge Alliance (2020). Open Charge Point Protocol: enlace, último acceso: 11/10/2022

[7] UML (2005, Julio). What is UML : enlace, último acceso: 11/10/2022

[8] IFML (s.f.). IFML: enlace, último acceso: 11/10/2022

[9] European Alternative Fuels Observatory (2021). Total number of alternative fuelled (BEV, PHEV, H2, LPG, CNG, LNG) passenger cars (M1) and vans (N1): enlace, último acceso: 28/04/2023

[10] Marín, P. F., & Úbeda, J. R. (2019). Vehículo eléctrico: situación actual y perspectivas futuras. Economía industrial, (411), 11-20.

[11] European Alternative Fuels Observatory. (2021). Total number of recharging points, based on the AFIR classification: enlace, último acceso: 28/04/2023

[12] Wallbox. (s. f.). FAQs corriente de carga EV: ¿Cuál es la diferencia entre CA y CC? Wallbox: enlace, último acceso: 5/05/2023

[13] Iberdrola. (s.f.). Estaciones de carga para vehículos eléctricos: enlace último acceso:

6/05/2023

[14] Wallbox. (s. f.). Tipos de conectores de carga para coches eléctricos: enlace, último acceso: 6/05/2023

[15] Open Charge Alliance. (2020). OCPP 2.0.1: enlace, último acceso: 6/05/2023

[16] Open Charge Alliance. (2020). Open Charge Point Protocol: enlace, último acceso: 7/05/2023

[17] Hita, M. A. (2021, 12 de Noviembre). Conoce qué tipos de coches eléctricos hay, sus características y si estás buscando uno de segunda mano. Motorpasión: enlace, último acceso: 9/05/2023

[18] Dirección General de Tráfico (2020, 18 de Noviembre). Distintivo ambiental: enlace, último acceso: 9/05/2023

[19] Karkkainen, V. (2022, 28 de Noviembre). OCPP-ASGI: enlace, último acceso: 10/05/2023

[20] Iberdrola, (s.f.). DSO - How to convert grid management towards a smarter system?: enlace, último acceso: 10/05/2023

[21] Open Charge Alliance (2020). The Importance Of Open Protocols: enlace, último acceso: 11/05/2023

[22] Hive Power (2021, 20 de Diciembre). OCPP vs IEC-63110 - Open Communication Protocol for V2G: enlace, último acceso: 11/05/2023

[23] Haugen M. J. (2021, 25 de Enero). A switch to battery electric vehicles is the best option for cleaner road transport, study finds. enlace, último acceso: 12/05/2023

[24] Reche, J. M. (2022, 19 de Enero). ¿Monolito o microservicios? Ventajas y desventajas: enlace, último acceso: 12/05/2023

[25] Decide Soluciones (2019, 3 de Septiembre). Arquitectura de microservicios: qué es, ventajas y desventajas: enlace, último acceso: 13/05/2023

[26] Google (2022, 9 de Diciembre). ¿Qué es Pub/Sub?: enlace, último acceso: 14/05/2023

[27] DGT Online (s.f.). Distintivo ambiental B, C, Eco o Cero para tu coche o moto: enlace, último acceso: 14/05/2023

[28] Akyga (s.f.). Enchufe tipo 1 (J1772) AK-SC-E02: enlace, último acceso: 14/05/2023

[29] Akyga (s.f.). Enchufe macho tipo 2 AK-SC-E03 : enlace, último acceso: 14/05/2023

[30] MiDA (s.f.). 125A 200A Chademo Plug Fast EV Cargador Enchufes Conector de carga de CC: enlace, último acceso: 14/05/2023

[31] Cocheselectricos10 (2018, 30 de Julio). Guía de cables para coches eléctricos: Conector Combo o CSS (IEC 62196): enlace, último acceso: 14/05/2023

[32] Hive Power (2021, 20 de Septiembre). 5 cosas que debes saber sobre OCPP 2.0, el protocolo de comunicación V2G: enlace, último acceso: 15/05/2023

[33] Docker Docs (s.f.). Get Started: enlace, último acceso: 15/05/2023

[34] Docker (s.f.). Docker Logos: enlace, último acceso: 15/05/2023

[35] Docker (s.f.). Containerize an Application: enlace, último acceso: 15/05/2023

[36] 1000 Marcas (2022, 20 de Julio). Kubernetes Logo: enlace, último acceso: 15/05/2023

[37] Kubernetes Documentation (s.f.). Concepts: enlace, último acceso: 15/05/2023

[38] Python Software Foundation (s.f.). The Python Logo: enlace, último acceso: 15/05/2023

[39] Python Enhancement Proposals (2004, 19 de Agosto). PEP 20 – The Zen of Python: enlace, último acceso: 15/05/2023

[40] Amazon Web Services (s.f.). ¿Qué es Python?: enlace, último acceso: 15/05/2023

[41] FastAPI (s.f.). FastAPI: enlace, último acceso: 16/05/2023

[42] R. Sebastián (2019, 4 de Febrero). Introducing FastAPI: enlace, último acceso: 16/05/2023

[43] Motor (s.f.). Motor: Asynchronous Python driver for MongoDB: enlace, último acceso: 16/05/2023

[44] Python Docs (s.f.). asyncio - E/S asíncrona: enlace, último acceso: 16/05/2023

[45] Node.js (s.f.). Node.js: enlace, último acceso: 16/05/2023

[46] S. Taha (2023, 3 de Febrero) What is Node.js: A Comprehensive Guide: enlace, último acceso: 16/05/2023

[47] Astro (s.f.). Press Resources: enlace, último acceso: 17/05/2023

[48] Astro (s.f.). Getting Started: enlace, último acceso: 17/05/2023

[49] Bluuweb (s.f.). Svelte (Guía Bluuweb): enlace, último acceso: 17/05/2023

[50] Cruz J. C. (s.f.). ¿Qué es Svelte.js?: enlace, último acceso: 17/05/2023

[51] Kakar S. (2021, 29 de Julio). Typescript vs Javascript : Which one should you use for your next project?: enlace, último acceso: 17/05/2023

[52] MongoDB (s.f.). Brand Resources: enlace, último acceso: 17/05/2023

[53] MongoDB (s.f.). ¿Qué es MongoDB?: enlace, último acceso: 17/05/2023

[54] IntelliPaat (2023, 7 de Abril). Mongodb Vs SQL - Top Key Differences: enlace, último acceso: 17/05/2023

[55] Ubunlog (s.f.). MongoDB Atlas: a DB for multi-cloud clusters: enlace, último acceso: 17/05/2023

[56] MongoDB (s.f.). MongoDB Atlas: enlace, último acceso: 17/05/2023

[57] Cloudinary (s.f.). Cloudinary Brand Assets: enlace, último acceso: 17/05/2023

[58] Vivevirtual (s.f.). Cloudinary: The most powerful image and video API: enlace, último acceso: 17/05/2023

[59] Forococheselectricos (2020, 29 de Diciembre). ¿Qué es, cómo funciona el V2G y por qué es importante para la gestión de flotas?: enlace, último acceso: 18/05/2023

[60] PyTest (2015). Get Started: enlace

Apéndice B
Manual de instalación

B.1. Requisitos previos

Para la correcta ejecución y rendimiento de la aplicación se recomienza al lector alojar la presente aplicación en una máquina con los siguientes requerimientos mínimos:

- **Recursos de CPU**: se recomienda tener al menos 4 núcleos de *Central Processing Unit (CPU)* para garantizar un rendimiento adecuado y manejar la carga de trabajo de la aplicación.
- **Memoria RAM**: se sugiere contar con al menos 8 *GB* de memoria *Random Access Memory (RAM)* para asegurar un funcionamiento fluido del proyecto y evitar problemas de falta de memoria.
- **Espacio de almacenamiento**: se debe disponer de suficiente espacio de almacenamiento para la aplicación, imágenes de contenedores y datos generados por los mismos. Un mínimo de 20GB de espacio de almacenamiento es la cantidad recomendada.
- **Conectividad de red**: es necesario disponer una conexión de red confiable y de alta velocidad para permitir la comunicación fluida con la nube.
- **Sistema operativo**: se recomienda utilizar un sistema operativo compatible con *Docker Desktop* y *Kubernetes*, como *Linux* (por ejemplo, *Ubuntu* o *CentOS*) o *MacOS*.
- ***Software***: es necesario tener instaladas las herramientas *Docker*, *Kubernetes* y *Helm*.

B.1.1. Inicialización de la aplicación

Para inicializar la ejecución de la aplicación, sitúese en la carpeta raíz del proyecto. Una vez allí, ejecute los siguientes comandos:

```
chmod +x ./run.sh;
./run.sh;
```

Si se encuentra en un sistema operativo donde no es posible ejecutar *scripts* de este tipo, deberá crear las imágenes una a una ejecutando los siguientes comandos desde la raíz del proyecto:

```
docker build -t frontend ./frontend;
docker build -t adminfrontend ./adminFrontend;
docker build -t backend-bookingsrest ./backend/bookingsREST
;
docker build -t backend-chargersrest ./backend/chargersREST
;
docker build -t backend-usersrest ./backend/usersREST;
docker build -t backend-ticketsrest ./backend/ticketsREST;
docker build -t backend-scheduler ./backend/schedulerREST;
docker build -t backend-logsrest ./backend/logsREST;
docker build -t ocpp ./backend/ocpp-asgi;
kubectl kustomize --enable-helm ./deployment | kubectl
apply -f -;
```

B.1.2. Modificación del fichero *hosts*

IMPORTANTE: Realizar este paso sólo en caso de no poder ejecutar el *script* run.sh.

Para acceder a la interfaz web de manera satisfactoria, ha de añadir dos entradas al fichero *hosts* de su sistema operativo.

Para sistemas operativos *MacOS* y *Linux*, ejecute los siguientes comandos:

```
cd;
echo "127.0.0.1 admincssc.com\n127.0.0.1 controlcssc.com" |
sudo tee -a /etc/hosts;
```

Para sistemas operativos *Windows*, siga los siguientes pasos:

- Ejecute la aplicación bloc de notas como administrador.
- Abra el archivo *hosts* situado en la ruta C:\Windows\System32\drivers\etc\hosts
- Añada las siguientes líneas al final del documento:

```
127.0.0.1 admincssc.com controlcssc.com
```

B.1.3. Instalación del certificado

Situándose en la raíz del proyecto, navegue hasta la carpeta deployment, una vez allí, navegue hasta la carpeta certs e instale el certificado cert.pem en su máquina.

Se refiere al lector a un tutorial de instalación de este tipo de archivos: enlace

Una vez realizado este paso, debe poder acceder a las aplicaciones web a través de las *URL* controlcssc.com y admincssc.com.

Apéndice C
Casos de uso más relevantes en el sistema

En el ámbito del desarrollo de software, es fundamental comprender los requerimientos y las funcionalidades que un sistema debe cumplir. Una de las técnicas más ampliamente utilizadas para capturar y describir estas funcionalidades es el análisis de casos de uso. En este apartado, se exploran los diagramas de casos de uso más relevantes del sistema.

El diagrama de caso de uso es una representación gráfica que muestra los diferentes casos de uso de un sistema y cómo se relacionan entre sí. Es una forma visual de comunicar las interacciones entre los actores y el sistema, brindando una visión general de las funcionalidades ofrecidas por el sistema y las relaciones entre ellas. En un diagrama de caso de uso, los actores se representan como entidades externas al sistema, mientras que los casos de uso se representan como elipses o rectángulos con etiquetas descriptivas. En un diagrama de caso de uso, los actores se conectan a los casos de uso mediante líneas que representan las interacciones. Estas interacciones pueden ser simples, como una solicitud de información, o más complejas, involucrando múltiples pasos y acciones.

Al explorar los diagramas de casos de uso, se puede obtener una visión más clara de cómo el sistema interactúa con sus actores y cumple con sus objetivos funcionales. Esta herramienta proporciona una forma estructurada y visual de capturar los escenarios de uso del sistema, lo que ayuda a los desarrolladores, diseñadores y usuarios finales a comprender y comunicar de manera efectiva las funcionalidades requeridas.

C.1. Funcionalidades de usuario estándar

A continuación, en la figura 52 se expone el diagrama de casos de uso referentes a las funcionalidades de un usuario estándar, el cual puede navegar libremente en la interfaz web destinada para ello: *Control-CSSC*. Realizando acciones entre las que se encuentran realizar

reservas sobre cargadores y el uso (bloqueo y desbloqueo de los mismos), gestionar sus propias reservas y la creación de tiquets de incidencia.

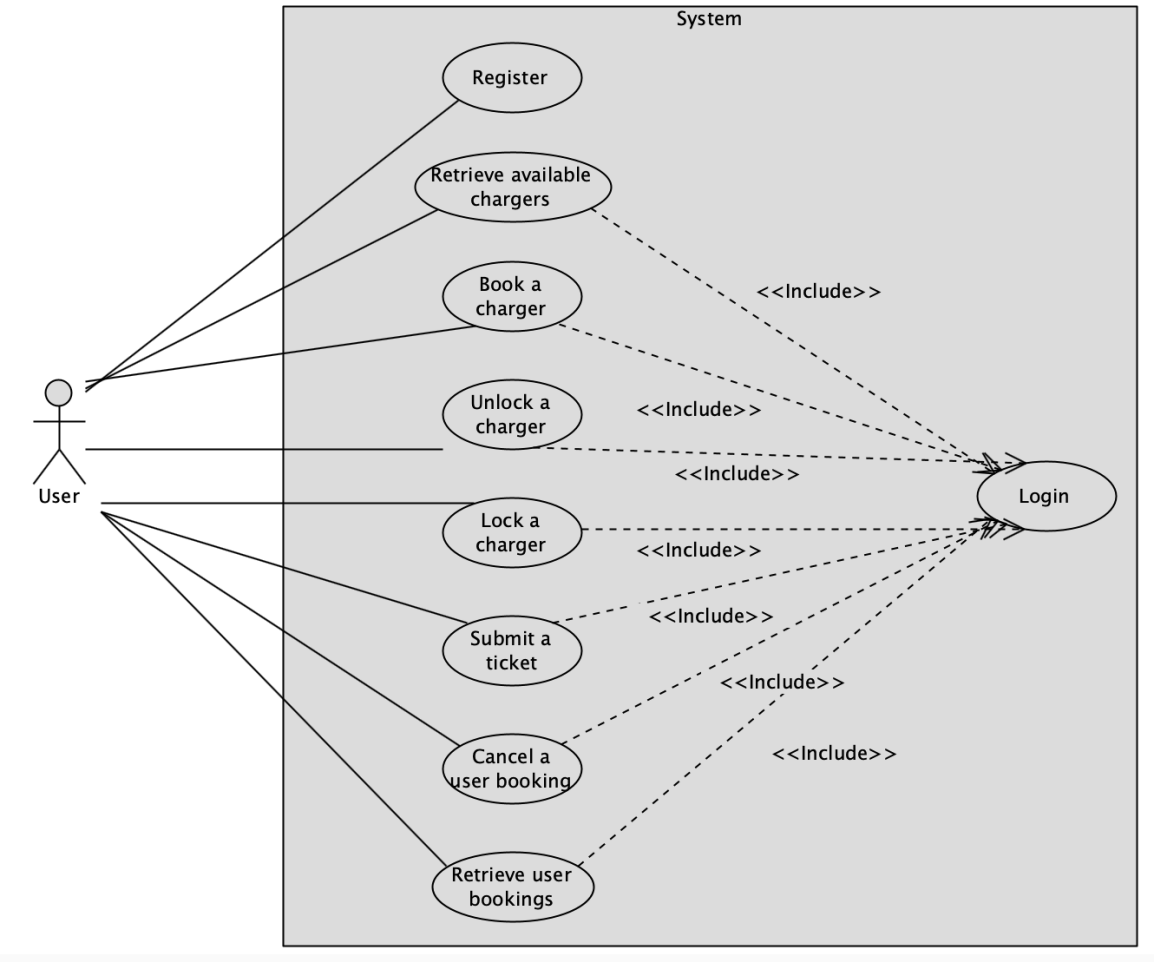

Figura 52: Diagrama de casos de uso referentes a las funcionalidades de un usuario estándar

C.2. Funcionalidades de administrador

A continuación, en la figura 53 se expone el diagrama de casos de uso referentes a las funcionalidades de un administrador, el cual puede navegar libremente en la interfaz web destinada para ello: *Admin-CSSC*. Realizando acciones entre las que se encuentras gestionar las reservas de todos los usuarios, gestión de cargadores y tiquets de incidencia, además de la consulta de registros y estadísticas del sistema.

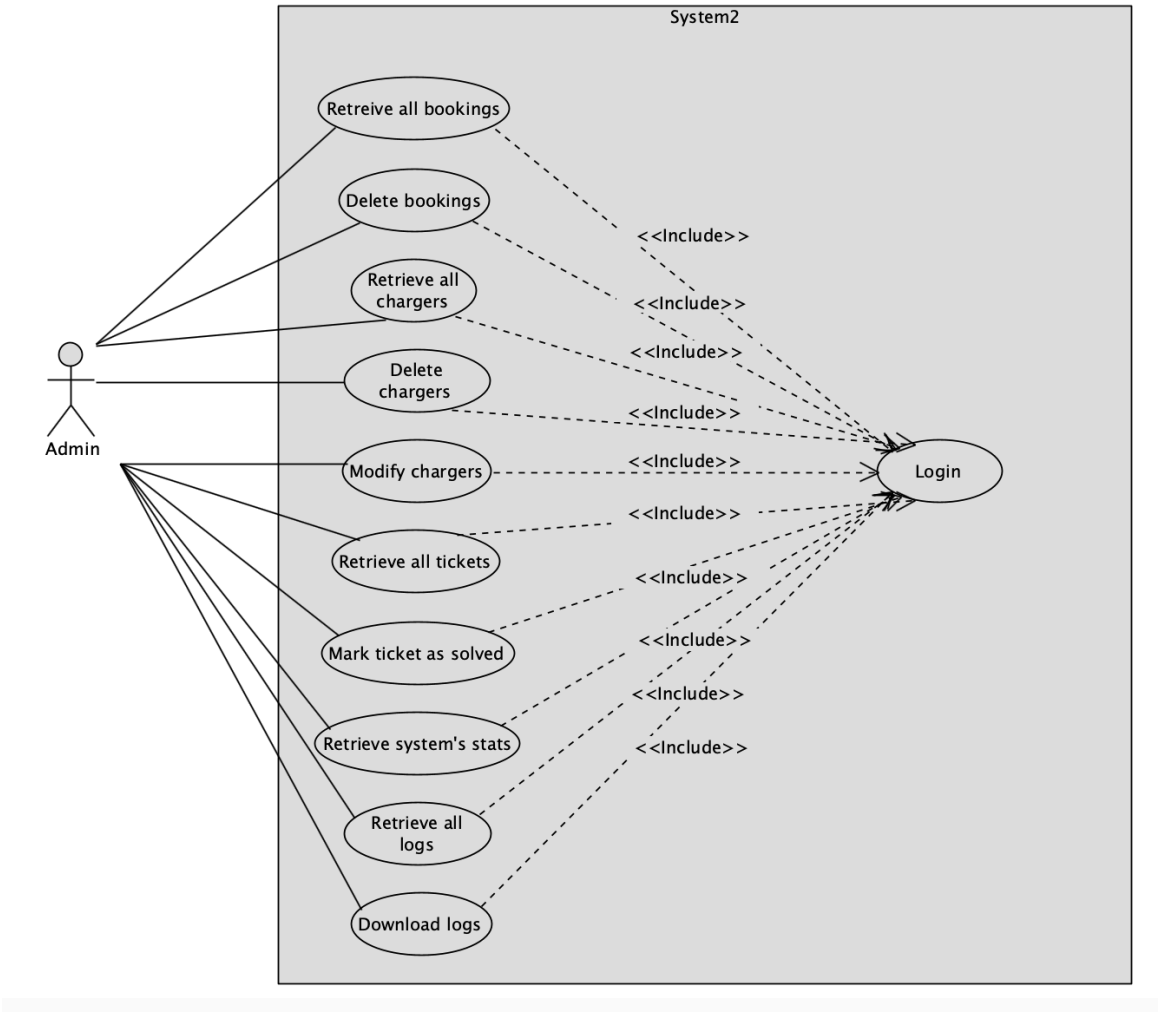

Figura 53: Diagrama de casos de uso referentes a las funcionalidades de un administrador

Apéndice D
Diagramas de flujo de interacción

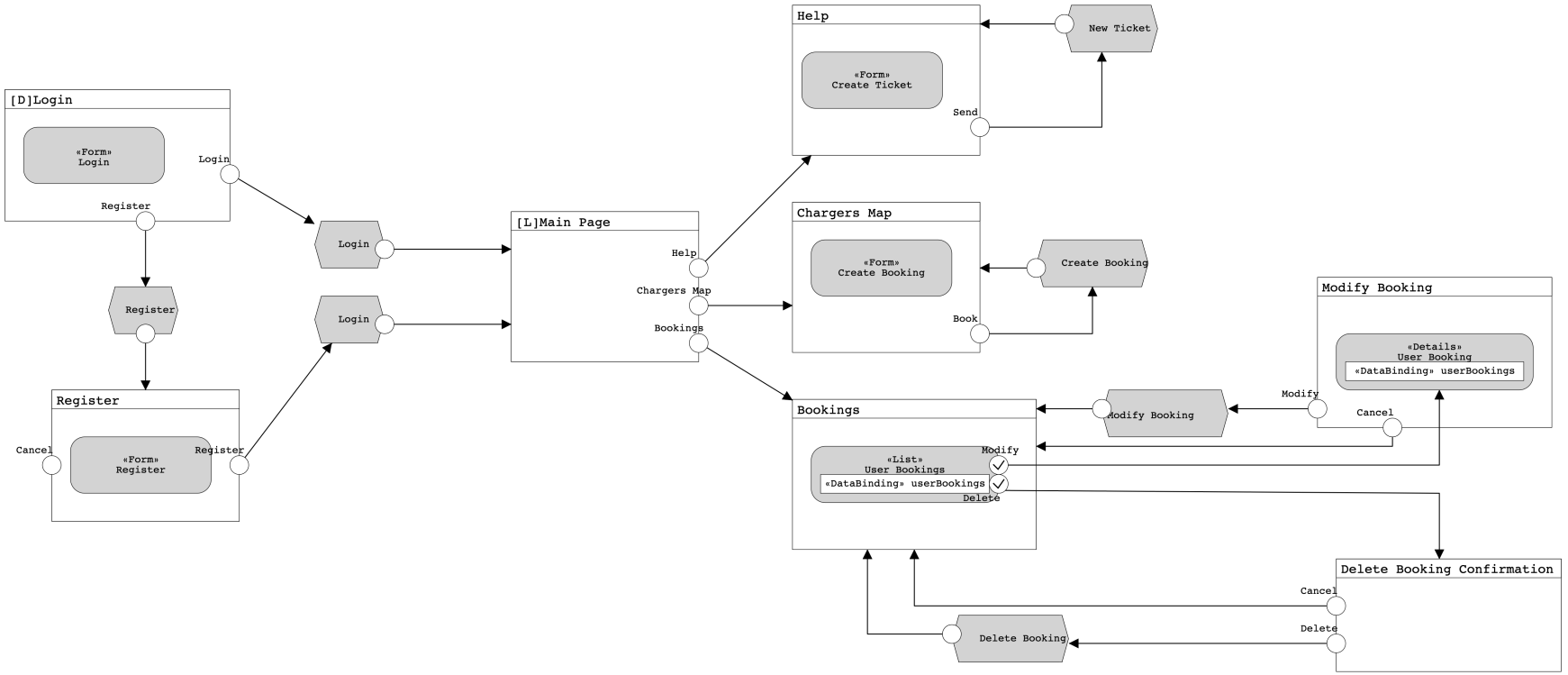

Figura 54: Modelo *IFML* de *Control-CSSC*

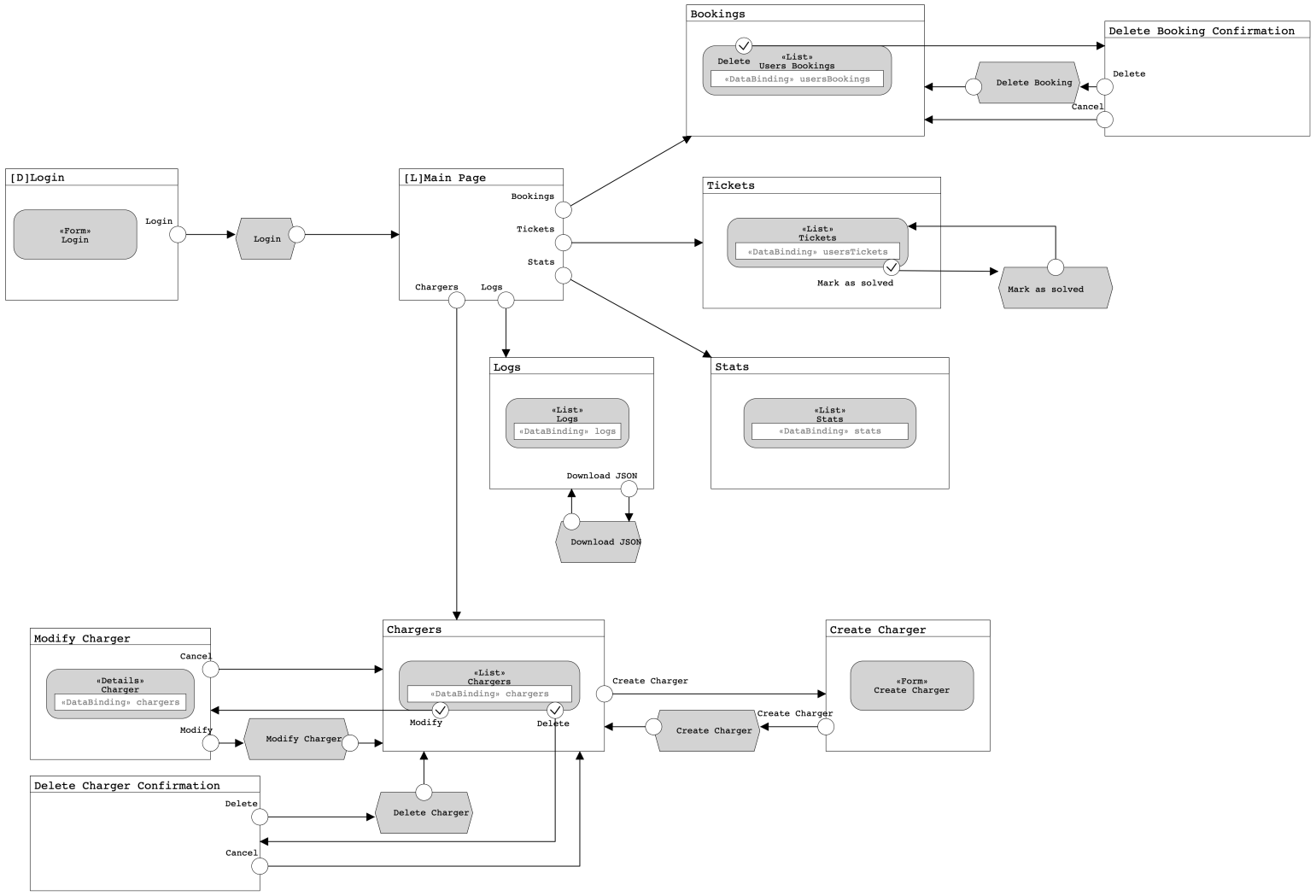

Figura 55: Modelo *IFML* de *Admin-CSSC*

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA