ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
Grado en Ingeniería Informática (Mención en Tecnologías de la Información)

Control Automatizado de Parkings

Automated Control Parking

Realizado por
Antonio Cantos Pérez
Tutorizado por
Luis Manuel Llopis Torres
Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO 2016

Fecha defensa:
El Secretario del Tribunal

Resumen:

En el presente documento se detalla el diseño y la implementación de un sistema de control para parkings, con el fin de hacer llegar la tecnología a este sector en el que pienso que la posibilidad de implementar un sistema como el detallado aquí, puede beneficiar tanto a clientes como a administradores. Para ello, al primer grupo, se le da la posibilidad de conocer en tiempo real su gasto actual y la plaza de garaje ocupada mediante una aplicación Android, así como diversos datos de interés a través de una plataforma web (gastos, tarifas aplicadas, etc.). Al segundo grupo, se le da la posibilidad de visualizar estadísticas y otros asuntos de interés para el administrador. Además de esto, se usará un microcontrolador, el cual mediante tecnología RFID será capaz de relacionar usuarios con plazas de garaje evitando así la problemática surgida de olvidar la plaza donde se ha aparcado.

Palabras clave: Microcontrolador, Intel Edison, JavaEE, Parking, Automatizado, Bootstrap, Omnifaces.

Abstract:

In this paper, is detailed the design and the implementation of a control parking system, in order to get the technology to this sector, because I think the possibility of implementing a system like the detailed here, can benefit both customers and administrators. For this, to first ones, it gives the possibility to know his/her actual spending and occupied parking place in real time using an Android application as well as various interesting data through a web platform (general spending, applied rates, etc.). To second ones, it gives the possibility to see statistics and other matters of interest for the administrator. In addition to this, it will be used a microcontroller which through RFID technology, will be able to relate users with parking places, avoiding the problems arising of forgetting the place where it was parked.

Keywords: Microcontroller, Intel Edison, JavaEE, Parking, Automatizado, Bootstrap, Omnifaces.

Índice de Contenidos

0. Introducción

Una característica común a la gestión de cualquier parking público o privado es la gestión de entradas, salidas y cobros de ocupaciones (téngase en cuenta que, en el presente documento, se denomina ocupación a un tramo temporal de inicio y fin definido en el cual un vehículo está estacionado en el parking). Pese a ser esta la principal característica, se pueden tener en cuenta otra serie de características interesantes como pueden ser una gestión financiera y estadística del aparcamiento controlando los índices de ocupación, los ingresos, etc. o desde el punto de vista de los clientes un mayor control del gasto actual o mensual y un control de la plaza ocupada, todo ello de forma totalmente telemática.

El principal diagrama de flujo de un aparcamiento puede ser tal como se refleja en el siguiente esquema:

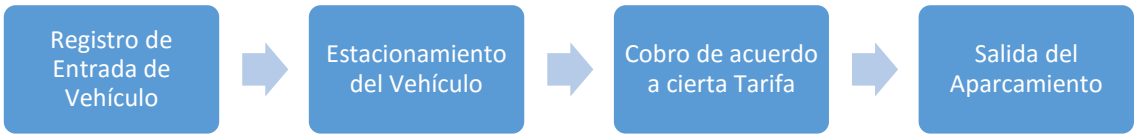


Figura 1: Flujo Principal de un aparcamiento.

La automatización de este flujo y la extracción de información relativa a este proceso (Data Mining) dará otro punto de vista al administrador del parking, permitiéndole tomar decisiones con una mayor seguridad gracias al apoyo de los datos suministrados. Además, se le permitirá al trabajador llevar un control en tiempo real del parking (controlando las plazas ocupadas, el estado del parking, solventar incidencias, etc.). Todo esto, se controlará mediante un servidor web construido mediante el uso de Java EE, mientras que el cliente, además de contar con acceso a este servidor web mediante una página web (como el resto de usuarios), también tendrá a su disposición el uso de una aplicación móvil diseñada para Android, con la cual podrá controlar su estacionamiento.

0.1 Objetivos

Los objetivos que se desean alcanzar con este Trabajo de Fin de Grado son los siguientes:

- Implementar un sistema capaz controlar un aparcamiento público o privado y gestionar todas las acciones básicas del mismo.

- Demostrar los conocimientos adquiridos durante la formación académica

- Adquisición de conocimientos técnicos relativos a la programación de Intel Edison y de sus librerías.

- Familiarización con las técnicas de análisis, diseño, construcción y gestión de proyectos

En cuanto a los objetivos fijados en lo que se refiere propiamente al control de un aparcamiento encontramos los siguientes:

- Posibilitar el acceso a estadísticas e información del parking por parte del Administrador del mismo.

- Permitir la creación de trabajadores, los cuales tengan tareas administrativas sobre el parking.

- Integrar a los aparcamientos en los Sistemas de Información, y con expectativa de futuro, permitir su integración en las “Smart Cities”

Así, se conseguirá una mejora sustancial en términos de facilidad; tanto usuarios como administradores del aparcamiento verán mejorado el servicio, accesibilidad; todos los datos serán accesibles desde cualquier punto bien con un ordenador o con un dispositivo móvil y publicitación; ya que al permitir el acceso vía servicios REST, se podrá integrar el sistema con aplicaciones móviles las cuales ayudan al usuario a la búsqueda de estacionamientos (Ej.: Wazypark).

0.2 Contenido de la Memoria

Llegados a este punto, detallaré la estructura seguida en la memoria a fin de establecer los distintos capítulos que la componen:

Capítulo 0: Introducción

En este capítulo se introduce la solución a adoptar e implementar en el presente texto, así como la estructura que se seguirá en el mismo.

Capítulo 1: Planificación

Se especificará la planificación llevada a cabo en el proyecto y si se cumplieron o no los plazos previstos.

Capítulo 2: Diseño y Arquitectura

Se detallarán los requisitos exigidos y los distintos pasos seguidos en el diseño de la arquitectura, además de las tecnologías usadas.

Capítulo 3: Implementación

Especificación clara y concisa de la implementación realizada, así como los problemas surgidos durante esta fase y sus soluciones.

Capítulo 4: Testeo y Mejora

Durante este capítulo se expondrán las metodologías seguidas para la búsqueda y solución de errores de implementación, así como las mejoras incluidas al sistema tras la obtención de la aplicación base.

Capítulo 5: Conclusiones y Líneas Futuras

Se expondrán las conclusiones del proyecto además de las posibles líneas de mejora/ampliación posibles.

Capítulo 6: Referencias

Referencias bibliográficas o de páginas de internet en las cuales me he basado al realizar la memoria.

Capítulo 7: Anexos

Documentos o esquemas adicionales del sistema.

1. Planificación

Durante esta fase se realizaron tareas de planificación temporal, estableciendo una serie de objetivos y metas a cumplir. Esta fase tiene bastante importancia a fin de conseguir entregar el proyecto antes de la fecha límite, dejando cierto tiempo "sobrante" para cualquier modificación necesaria de última hora.

Un punto importante en cualquier planificación de un proyecto, es conocer el proceso a seguir. En el caso de este proyecto, se siguió una metodología de desarrollo iterativo [12].

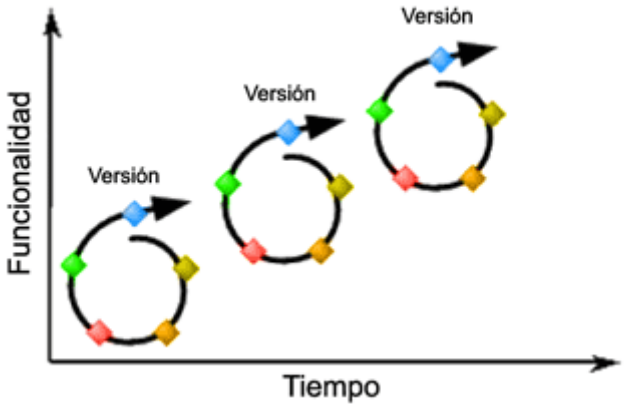


Figura 2: Desarrollo Incremental

Se define esta metodología, como el proceso por el cual se planifica el proyecto en diversos bloques temporales llamados iteraciones. Cada iteración se puede entender como un "miniproyecto" el cual una vez acabado, proporciona una funcionalidad añadida sobre el producto final. Antes de la primera iteración, se suele establecer un periodo de inicialización.

En el caso específico de este proyecto, el periodo de inicialización consistió en establecer las directrices de diseño y arquitectura, así como un desarrollo muy "primario" de lo que sería el sistema más tarde. A continuación, pueden verse las diversas fases así como su temporización:

- ✓ 1 de Febrero al 15 de Febrero: Elaboración del anteproyecto y presentación el mismo en la secretaría del centro.

- ✓ 16 de Febrero al 29 de Febrero: Elaboración de esta planificación para poder llevar un control del tiempo dedicado a la confección de este proyecto.

- ✓ 1 de Marzo al 15 de Marzo: Análisis de requisitos, elaboración de los casos de uso y diseño de la arquitectura que tendrá el sistema, teniendo en cuenta las tecnologías que se usarán.

- ✓ 16 de Marzo al 30 de Abril: Tiempo dedicado a la implementación del sistema inicial, teniendo en cuenta que deberán realizarse pruebas unitarias para cada método, así como una final para comprobar el correcto funcionamiento del sistema así como la cumplimentación de los requisitos principales establecidos en el apartado anterior.

- ✓ 1 de Mayo al 15 de Mayo: Realización de las sucesivas evoluciones incrementales del sistema, finalizando cada evolución con su correspondiente fase de pruebas y cumplimentación del requisito incrementado. Esta fase se alargó en el tiempo debido a que ciertas mejoras fueron subestimadas al realizar la planificación, causando un retraso notable en las fases posteriores.

- ✓ 16 de Mayo al 31 de Mayo: Elaboración formal de esta documentación.

- ✓ 1 de Junio al 30 de Junio: Corrección de posibles bugs, posibles mejoras extra, etc. y entrega del proyecto.

2. Diseño y Arquitectura

Como se ha avanzado anteriormente, en este apartado se especificarán el diseño, la arquitectura y las tecnologías usadas.

En primer lugar, como cualquier fase de diseño requiere, se elaboró un análisis de requisitos [1]. Recibe este nombre, la determinación concisa de las funcionalidades y/o condiciones que cierto software debe cumplir. En el caso de este proyecto, se determinaron una serie de requisitos principales y otros secundarios. Con requisitos principales, en el presente documento, se refiere al conjunto de requisitos que han de cumplirse obligatoriamente en la versión inicial del sistema (versión realizada durante la primera fase de este proyecto), mientras que por secundarios se refiere a los requisitos que serán cumplimentados durante el desarrollo incremental del proyecto. El informe de requisitos puede verse a continuación.

2.1 Requisitos

2.1.1 Requisitos Principales

Estos requisitos como he mencionado anteriormente, se refieren a aquellos de obligatorio cumplimiento en la versión inicial del sistema. Describen la funcionalidad general del sistema refiriéndose a la gestión del aparcamiento de forma que se cumplimente lo expuesto en el documento de *Anteproyecto*. Estos requisitos son:

- Tres tipos de usuarios: Administrador, Trabajador y Usuario, esto servirá para dar más seguridad al sistema estableciendo el *principio del mínimo privilegio*, principio por el cual sólo se permite el acceso a cada usuario a aquello para lo que legítimamente está autorizado.
- CRUD de Usuarios, se permite la creación, actualización, lectura y borrado de usuarios teniendo en cuenta el principio del mínimo privilegio antes nombrado. Así pues, un usuario sólo podrá visualizar y actualizar sus propios datos, mientras que un trabajador podrá visualizar, crear, borrar y actualizar usuarios y trabajadores (incluyendo sus propios datos). Finalmente, un administrador podrá visualizar, borrar, actualizar y crear cualquier usuario.
- CRD de Tarjetas de Acceso, cualquier trabajador podrá crear, visualizar y borrar las tarjetas de un usuario.
- CRD de Plazas del Aparcamiento, cualquier trabajador podrá crear, visualizar y borrar plazas de aparcamiento.
- CR de Noticias, todos los usuarios podrán visualizar noticias, mientras que sólo los trabajadores y administradores podrán crearlas.
- CRU de Facturas, cualquier trabajador podrá facturar a un cliente.
- CRUD de Personas, cualquier trabajador o administrador podrá crear una persona para la base de datos.

- CRU de Reclamaciones, cualquier cliente podrá emitir o ver una reclamación (únicamente las propias), la cual será actualizada y leída por cualquier trabajador.
- CRU de Tarifas, cualquier administrador podrá crear, leer y actualizar tarifas, pudiendo habilitarlas y deshabilitarlas, aunque estás sólo podrán ser de un tipo.
- Gestión de Ocupaciones, cualquier usuario, administrador o trabajador podrá consultar el estado actual del parking, pero dependiendo de su rol, verá un mayor nivel de detalle.
- Lectura de Ocupación, cualquier cliente podrá revisar los datos de su actual ocupación mediante la aplicación Android.
- Tres tipos de dispositivos, dispositivo para registrar entrada, dispositivo para registrar salida y dispositivo para registrar la plaza ocupada.

2.1.2 Requisitos Secundarios

Estos requisitos, como se mencionó anteriormente, serán implementados en las sucesivas etapas incrementales, por lo que, en la versión inicial del sistema, no serán incluidos, pese a que posteriormente si deberán de ser incluidos:

- Visualización de Estadísticas, los administradores podrán visualizar estadísticas globales el sistema como ocupación, beneficios, uso de tarifas, etc.
- Auto-Registro de Dispositivos, no será necesaria la creación manual de dispositivos, sino que estos se auto-registrarán mediante una contraseña de acceso privada.
- Acceso a datos de ocupación por código QR, no será necesario que el cliente introduzca el código alfanumérico de identificación de la tarjeta, si no que escaneará un código QR con su dispositivo móvil.
- Visualización exacta de la relación de plaza de parking-cliente/tarjeta que la ocupa, ayudará a gestionar el parking, por ejemplo, detectando uso ilegal de las tarjetas (como clientes que ceden la tarjeta a otra persona) o problemas que puedan surgir.
- Tres tipos de Tarifas, se podrán crear tres tipos distintos de tarifa: por minuto, por minuto con mínimo y tarifa plana.
- Imágenes de Perfil para los Usuarios, se les permitirá a los usuarios a subir su propio avatar facilitando así la identificación de la persona.
- Imágenes en Noticias, se podrán añadir imágenes a las noticias a fin de potenciar la difusión de éstas.
- Histórico de Ocupaciones, las ocupaciones permanecerán en la base de datos aun cuando la tarjeta que las generó desaparezca para poder así ver estadísticas históricas (consumo anual, por ejemplo).
- Log de Eventos, se realizará un log revisable únicamente por los administradores para poder llevar un control de los movimientos del sistema.

2.2 Arquitectura

Finalizado el análisis de requisitos, se comenzó a pensar la arquitectura del sistema a rasgos generales. Dado que uno de los objetivos del proyecto es el poner en práctica lo aprendido durante el periodo académico, se eligió desarrollar el sistema web bajo Java Enterprise Edition (Java EE, estudiado en la asignatura Sistemas de Información para Internet). Tras esclarecer el sistema principal, faltaba aclarar cómo se efectuarían las conexiones entre los distintos dispositivos Intel Edison y el servidor web. Para ello, y gracias a las facilidades dadas por Java EE para integrar Servicios REST [13], se determinó que esta sería la forma más válida para comunicarlo, usando JSON para la estructuración de los datos enviados en dichas peticiones (dada su facilidad de integración con Java y Android). Tras este paso, el sistema quedó definido de la siguiente forma:

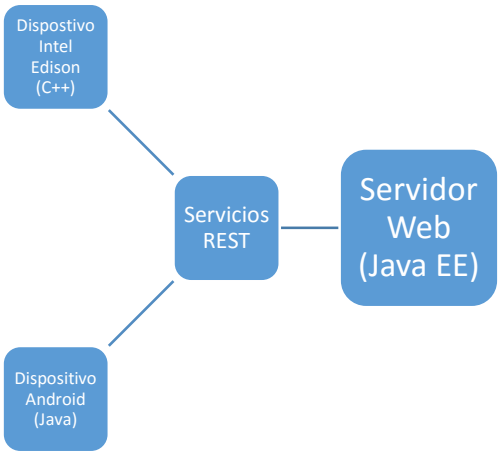
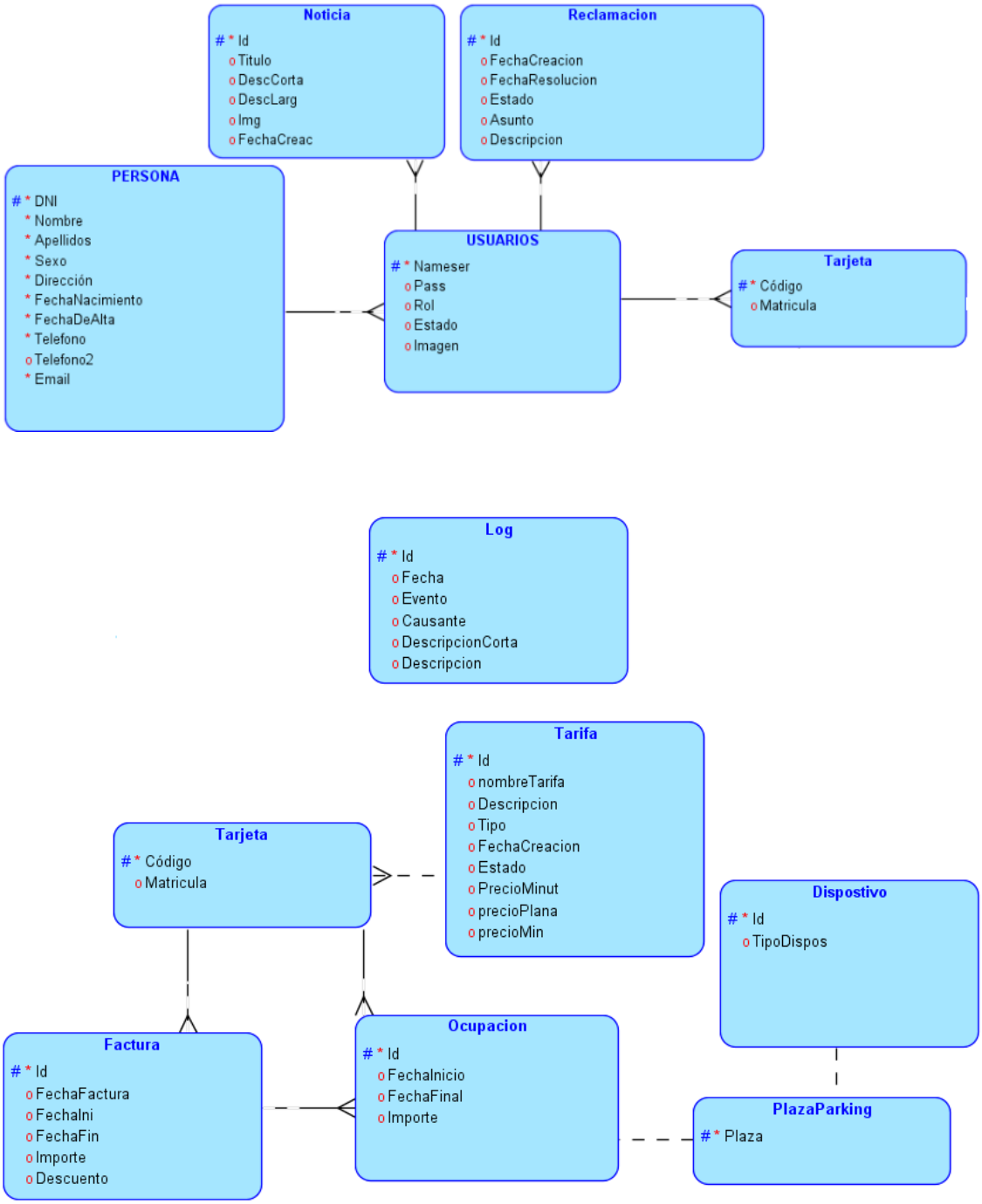


Figura 2: Arquitectura Simplificada del Sistema

2.3 Diseño

Tras realizar el diseño de la arquitectura, se comenzó a pensar la estructuración que debía tener la base de datos teniendo en cuenta que debía permitir todo lo expuesto en el análisis de requisitos. De este estudio, se determinaron un total de 11 entidades, que pueden observarse junto con sus relaciones a continuación:



Figuras 3 y 4: Estructura Base de Datos

Una vez, se determinó el diseño de la base de datos y la arquitectura general del sistema, se pasó a crear los *Casos de Uso [3].* Caso de Uso es definido como una secuencia de interacciones desarrolladas entre un sistema y un actor, sirviendo para especificar de forma general el comportamiento de un sistema mediante su interacción con los usuarios y/o otros sistemas. Los casos de uso forman parte del Lenguaje Unificado De Modelado (UML) , definido posteriormente en tecnologías usadas. De acuerdo a lo expuesto, los casos de uso del sistema aquí definido son:

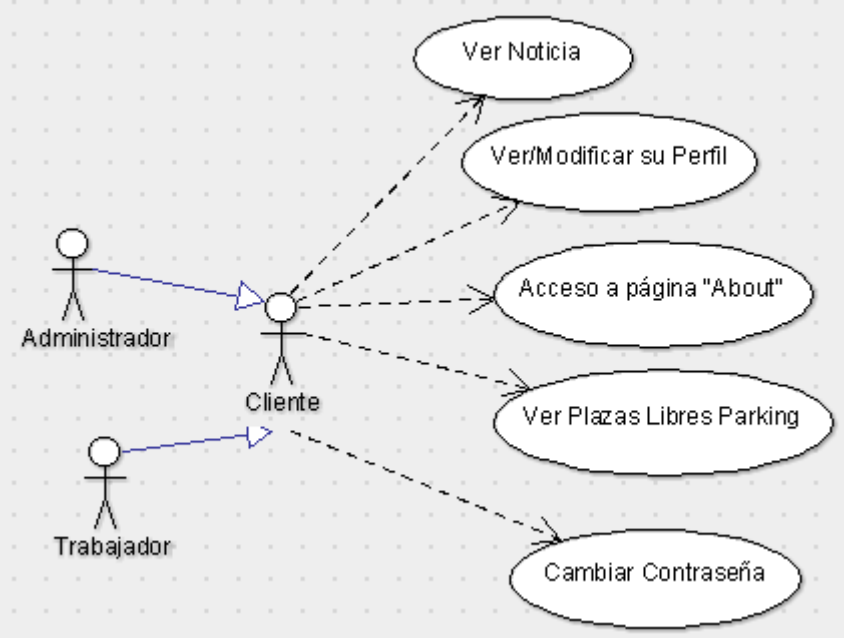


Figura 5: Funciones Comunes Cliente, Administrador y Trabajador

Como se muestra, a todos los usuarios se les permitirá ver y modificar su perfil, visualizar las noticias publicadas, ver la cantidad de plazas libres que tiene el aparcamiento, cambiar su contraseña y acceder a la página “About”.

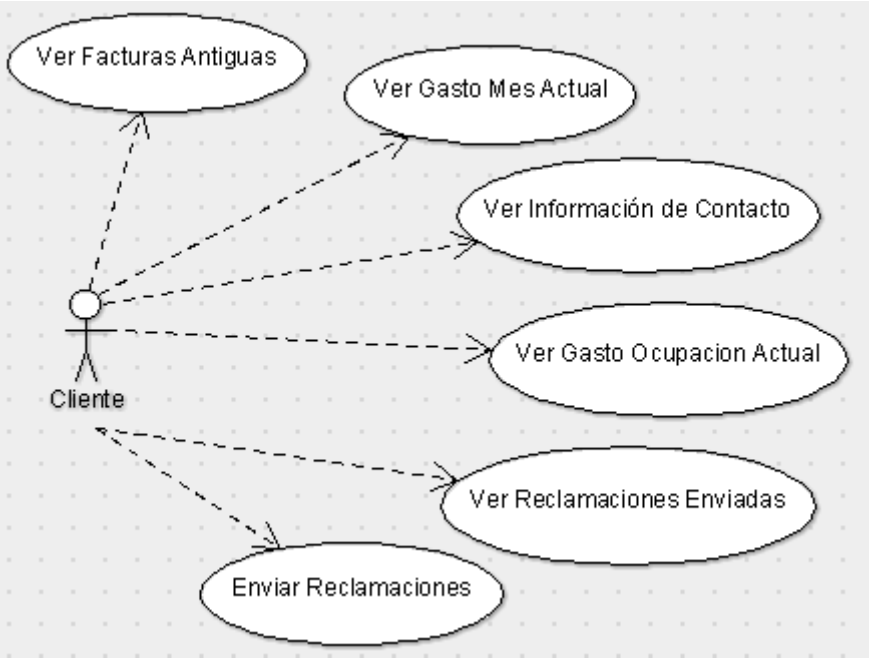


Figura 6: Funciones Cliente

Como se puede observar, un cliente podrá ver las facturas de meses pasados con todo detalle, ver el gasto acumulado en el mes actual, la Información de contacto con el aparcamiento, controlar su actual ocupación (mediante la app móvil) y por último ver y enviar reclamaciones (para problemas que puedan tener con su uso de la página web).

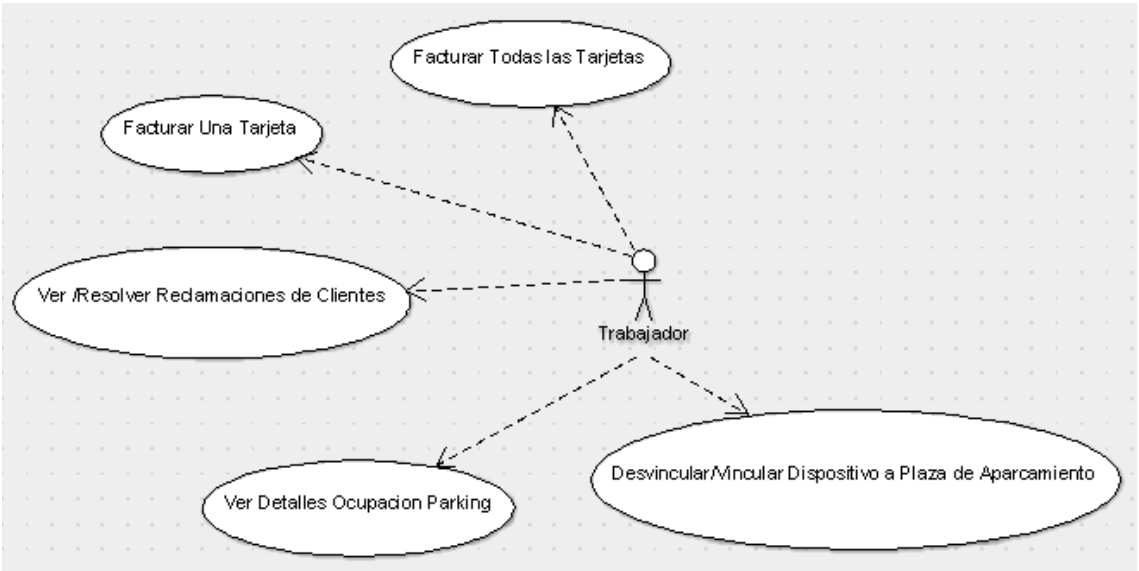


Figura 7: Funciones Trabajador (1)

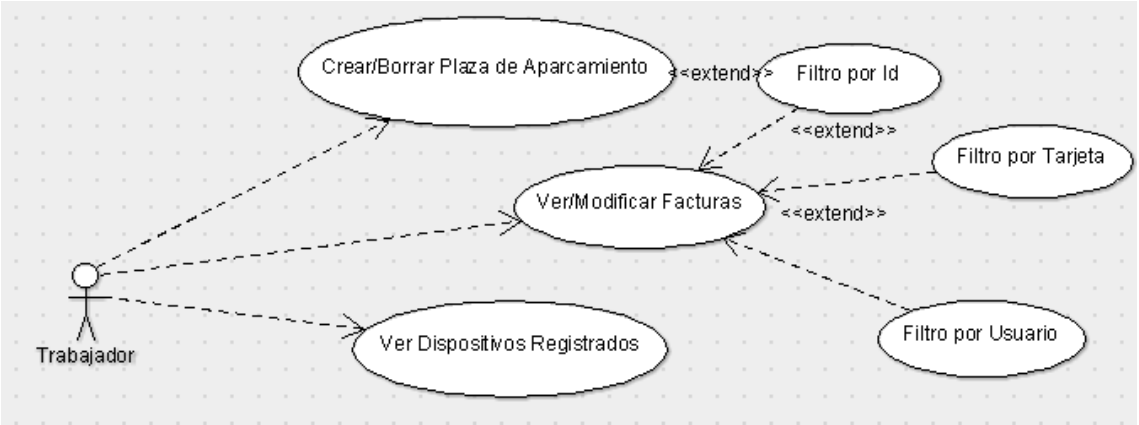


Figura 8: Funciones Trabajador (2)

Como se puede ver un trabajador podrá elaborar la factura correspondiente a una tarjeta (o varias tarjetas a la vez), ver las reclamaciones enviadas por clientes y modificar sus estados (pendiente, en revisión o solucionada), ver los detalles de ocupación del parking (relación de plazas con tarjetas/clientes que la ocupan), vincular o desvincular los dispositivos de plaza a una plaza de aparcamiento, crear o borrar las plazas de aparcamiento, ver los dispositivos registrados (de los diferentes tipos, plaza, entrada o salida) y ver todas las facturas del sistema pudiendo filtrarlas por ID, tarjeta o usuario.

Figura 9: Funciones Administrador

Un administrador podrá crear, modificar y habilitar o deshabilitar las tarifas del sistema, ver los eventos acaecidos en el sistema (pudiéndolos filtrar por tipo y fecha en el que sucedió) y visualizar estadísticas globales del sistema, tanto en tabla como en gráficos.

Finalmente, para concluir el apartado de diseño y arquitectura, se establecieron con la ayuda de una estudiante de Ingeniería de Diseño Industrial, el logo y la paleta de colores a usar, tanto en la página web como en la aplicación Android. Tras reflexionarlo, se decidió establecer un naranja claro para las barras de menú (relacionado con los "conos" presentes habitualmente en los aparcamientos soterrados y el gris oscuro para fondos y rellenos (relacionado con el color de las carreteras). Finalmente, el logo se diseñó de tres tipos: uno ampliado (para portadas), un icono (para la aplicación) y un icono con leyenda:

Figura 10: Paleta de Colores y Logos.

2.4 Tecnologías a Usar

Para finalizar este apartado, y tras completar tanto el diseño como la arquitectura, se empezó a valorar las posibles tecnologías a usar. Esta etapa (a menudo infravalorada) contiene una gran importancia, ya que el rendimiento del sistema dependerá de las tecnologías usadas y sus capacidades de integración y/o rendimiento.

2.4.1 ArgoUML

Herramienta de código abierto líder [4]. Soporta el estándar UML 1.4 y funciona bajo Java Runtime Enviroment. Esta utilidad es usada en el presente proyecto para crear los Casos de Uso y así describir el sistema, aunque permite muchas más operaciones como aplicación de la ingeniería inversa a proyectos finalizados. Otros detalles relevantes de este software de modelado son:

- Sencillo e Intuitivo
- Generación automática de código en lenguaje Java, PHP o C++ entre otros.
- Exportación a diferentes formatos.

2.4.2 Netbeans

Netbeans es un entorno de desarrollo integrado (IDE) de software libre ideado en sus inicios (diciembre de 2000) específicamente para el lenguaje Java [5]. Su diseño (extensible mediante módulos) hace que actualmente sea uno de los entornos de desarrollo más usados actualmente.

Algunas de las características que han expandido su uso entre los desarrolladores son:

- Multiplataforma
- Integración con control de versiones
- Potente editor de código con sugerencias de sintaxis
- Gracias a su amplificación mediante módulos, permite adaptar su uso a varios lenguajes.
- Permite la integración con varios gestores de base de datos.

El motivo por el cual se ha usado este IDE es que su versión con el plugin de Java EE integrado, hace que las configuraciones iniciales del proyecto se generen de forma automática y muestre unos interfaces gráficos muy intuitivos para poder cambiarlas de acuerdo al proyecto concreto. Además de este punto, la integración con diversas bases de datos y con servidores de aplicaciones como Glassfish, declinaron la selección por este IDE.

2.4.3 Sublime Text 3 + Plugin Emmet

Sublime Text es un potente editor de texto lanzado en Enero de 2008 como extensión del popular editor Vim. Las principales características de este editor son:

- Multi-cursor: Permite la escritura en varias líneas.
- Soporta múltiples lenguajes: Permite la escritura con soporte para varios lenguajes como PHP, HTML, XHTML, Java, C, etc.
- Comandos: Permiten acceder a de forma rápida a ciertas acciones como la inserción de texto o el autocompletado

- Soporte para plugins: Esta característica ha sido esencial, dado que gracias al plugin Emmet, la escritura de código HTML, CSS y JS ha sido más sencilla, debido a sus “atajos” para la escritura de código o su autocompletado de etiquetas (muy importante en XHTML).

El uso de este editor se ha centrado en la escritura de código XHTML, necesario para la creación de las páginas web que posteriormente se detallarán durante la fase de implementación.

2.4.4 Arduino IDE

Arduino IDE es el entorno de desarrollo elegido para el desarrollo de código C++ ejecutable en Intel Edison. En esta elección había otra posibilidad a usar, Eclipse IDE. El motivo de elección de Arduino IDE ha sido principalmente las facilidades dadas por éste para la programación del dispositivo, ya que Eclipse exige el desarrollo de librerías propias para el uso librerías [18] ya escritas para manejar el Wifi o las entradas/salidas del dispositivo. Otra opción, es la configuración manual de las librerías estándar de Arduino, aunque esto es una tarea complicada por lo que la facilidad de uso de Arduino IDE y la compatibilidad total con Intel Edison han sido determinantes.

2.4.5 SQL Data Modeler

Herramienta desarrollada por Oracle desde marzo de 2006.Es un entorno integrado de desarrollo para la creación de bases de datos basadas en SQL Oracle mediante la construcción de esquemas relacionales y su posterior ingeniería a sentencias SQL. En este proyecto, fue usado para la construcción del diagrama de clases mostrado en el apartado de Diseño y Arquitectura. El motivo de elección de esta herramienta, es que fue estudiado durante la asignatura Base de Datos, y dado que era objetivo del presente proyecto el poner en práctica los conocimientos adquiridos durante la formación académica, se puso en práctica esta herramienta.

2.4.6 Android Studio

Entorno de desarrollo oficial para Android [6]. Provee el entorno ideal para el desarrollo de aplicaciones Android con su propio editor de interfaces gráficas, sugerencias de código, simulador virtual de un dispositivo Android e integración total con el SDK de Android.

2.4.7 Lenguaje Unificado de Modelado (UML)

Es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. UML ofrece un estándar para el modelado de sistemas respaldado por el Object Management Group (OMG). UML fue declarado estándar a partir de 2005, cuando fue aprobado por la ISO.

Este lenguaje ha sido usado para la creación de los casos de uso mostrados en el apartado Diseño y Arquitectura del presente documento, aunque permite crear muchos más tipos de diagramas como los de clases. Este lenguaje también ha sido estudiado durante los estudios de grado, por lo que ha sido un factor determinante junto con ser un estándar de modelado de sistemas.

2.4.8 Lenguaje de Marcas de HiperTexto (HTML)

Es un tipo de lenguaje compuesto por etiquetas cuyo intérprete, el navegador, es el encargado de darle forma y mostrar por pantalla el resultado.

HTML sigue un estándar y es usado para el desarrollo de páginas web. En este caso W3C es la entidad encargada de unificar todas las reglas y lograr esta estandarización de la tecnología que comprende la web.

Es el lenguaje más importante y expandido en la actualidad debido a que cualquier navegador actual lo soporta. Las principales especificaciones de HTML aparecen en 1991, escritas por Tim Bemers-Lee.

El formato de las instrucciones es como sigue:

```
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <title>Document</title>
</head>
<body>
        <p id="texto">Hello World</p>
</body>
</html>
```

Figura 11: Código HTML básico

Como se puede observar, el formato del código se corresponde con etiquetas encerradas entre los símbolos ‘<’ y ‘>’, añadiendo dentro de las mismas, atributos del estilo ‘atributo = “valor”’.

2.4.9 Hoja de Estilos en Cascada (CSS)

Lenguaje usado para definir la presentación de un documento estructurado en HTML o XML (y por extensión XHTML). La idea principal de CSS es la separación de estructura y representación de un documento, ya que los estilos pueden separarse en ficheros aparte o entre la etiqueta <style>. Su estandarización es llevada a cabo por la organización W3C.

Como ventajas principales, encontramos la mejora del ancho de banda de conexión, ya que el fichero CSS es descargado y usado en multitud de documentos y la mejora de la accesibilidad del documento, ya que evita antiguas prácticas como el control del diseño mediante tablas.

2.4.10 JavaScript (JS)

Lenguaje orientado a objetos con sintaxis parecida a C. Aporta a los documentos HTML ciertas funcionalidades y animaciones muy atractivas de cara al enriquecimiento de la página web. En el caso de este proyecto, se usará para realizar validaciones en el lado del cliente, evitando así la sobrecarga del servidor web con comprobaciones de datos, las cuales pueden ser realizadas en el navegador del cliente.

2.4.11 Twitter Bootstrap

Más conocido como Bootstrap, es un framework HTML, CSS y JS para el desarrollo de web “responsive” (adaptable a dispositivos móviles). Fue desarrollado por Mark Otto y Jacbod Thornton, ambos de Twitter, compañía que decidió liberar el código en 2011. Se basa en los siguientes fundamentos:

- Diseño responsivo: el documento es adaptable a cualquier tipo y tamaño de pantalla.
- Compatible con la mayoría de navegadores.
- Integración con JQuery y otros elementos de interfaz de usuario.
- Facilidad de uso: con un manejo básico, se consiguen grandes resultados.

El motivo principal por el que me he decantado por el uso de este framework (descartando otros como Primefaces) es su independencia total de la tecnología web

del servidor, lo que permite hacer independiente el diseño web de su gestión posterior (capa de negocio).

2.4.12 Omnifaces

Librería de componentes para JSF que surge para solucionar los principales problemas de JSF [22]. En este proyecto se hace uso de esta librería para la gestión de las imágenes de perfil, permitiendo de forma fácil, la subida y muestra de imágenes desde HTML, simplificando en gran medida el uso de esta funcionalidad.

2.4.13 Java EE

Plataforma de programación, parte de la plataforma Java, para el desarrollo de software de aplicaciones web [2]. Esta plataforma permite la integración de todo tipo de servicios web como REST. Además, y como característica determinante a la hora de elegir esta plataforma frente a otras como PHP, ha sido la fácil distribución del software en N capas, determinante como se comentará en el apartado de líneas de ampliación futuras, para la separación del sistema en varios servidores (división de carga). Esta división en N capas, hace posible el uso del patrón Modelo-Vista-Controlador (MVC).

El patrón MVC, consiste en la separación de la lógica de datos de la presentación, comunicándose entre ellas mediante un “controlador” o lógica de negocio. El esquema puede verse en la siguiente figura:

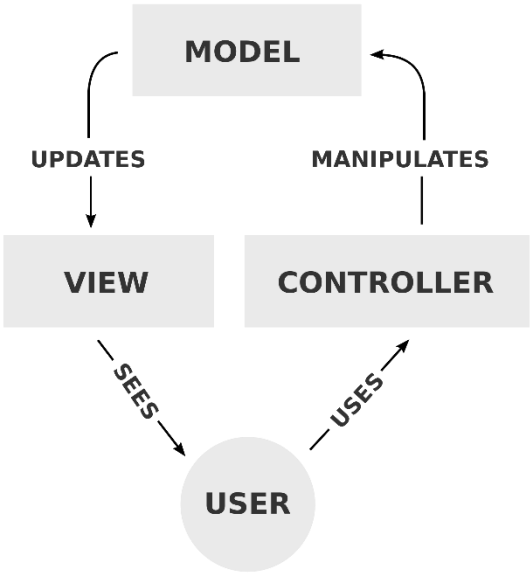


Figura 12: Patrón MVC

En concreto, este modelo aplicado a Java EE da lugar a tres componentes distintos:

- Java Persistence API (JPA): Api de persistencia de Java EE. Es un framework para el manejo de datos relacionales de aplicaciones.
- Enterprise Java Beans (EJB): Encargado de controlar las transacciones y la concurrencia en el acceso a la base de datos.
- JavaServer Faces (JSF): Permite y simplifica el desarrollo de interfaces de usuario para aplicaciones sobre Java EE, enriqueciendo las páginas con contenidos dinámicos.

La relación entre este modelo de N capas y el modelo MVC puede verse de forma clara en la siguiente figura:

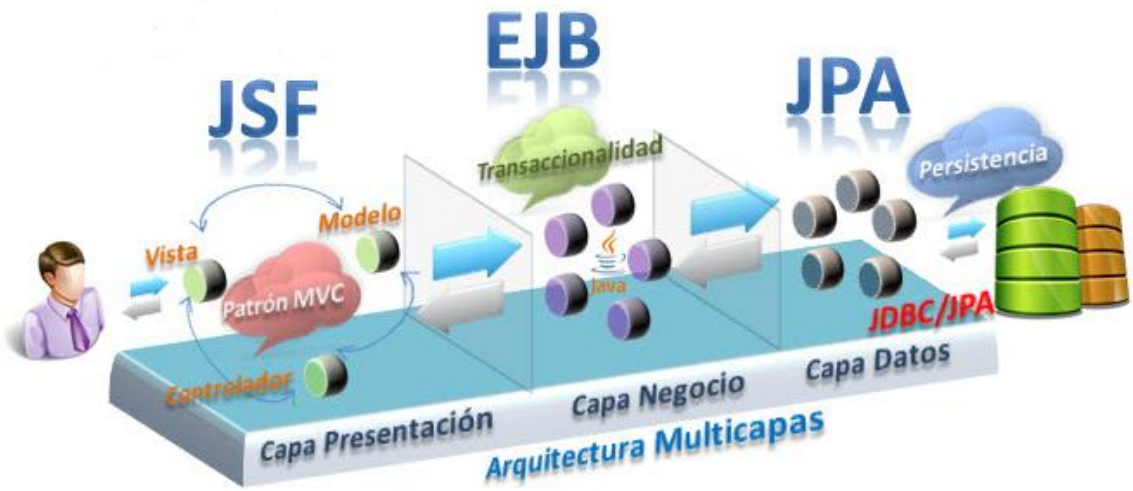


Figura 13: Comparación MVC y Arquitectura Java EE

2.4.14 C++

Lenguaje elegido para la programación del dispositivo empotrado Intel Edison. Este lenguaje surge a mediados de 1980 como extensión de C, dando soporte a la manipulación de objetos. El motivo de elección de este lenguaje radica en que su uso es obligado en caso de usar Arduino IDE. Para poder programar el dispositivo en otro lenguaje como Java o Python, hubiera sido necesario usar otro IDE como Eclipse, pero como se expuso en el apartado correspondiente, esto era complicado.

2.4.15 Base de Datos Apache Derby

Consiste en un subproyecto de Apache DB. Es una pequeña base de datos [7], la cual consume poco espacio para almacenar el motor y el driver de acceso (apenas 2.6 MB), está escrita en Java y viene preinstalada con el Java Development Kit, lo que, en este caso, facilita la instalación en clientes. Estos dos aspectos, han sido claves a la hora de elegirla frente a otras bases de datos más potentes como son MySQL.

2.4.16 Glassfish Server

Servidor número uno en uso para aplicaciones Java EE 7 [8]. Propiedad de Sun Microsystems (compañía perteneciente a Oracle), es gratuito y libre y su integración con Netbeans, han hecho que sea el servidor elegido por delante de otros como Apache TomEE.

2.4.17 Java

Lenguaje de propósito general, concurrente y orientado a objetos diseñado por James Gosling y Sun Microsystems en 1995. Su gran expansión y su diseño multiplataforma ha hecho que haya sido el elegido por Google para el desarrollo de aplicaciones Android, mediante el uso de Android Software Development Kit (Android SDK), es por ello que será necesario el uso de Java (lenguaje aprendido durante el periodo formativo del grado) para la programación de la aplicación Android.

2.4.18 JavaScript Object Notation (JSON)

Es un formato para el intercambio de datos, básicamente JSON describe los datos con una sintaxis dedicada que se usa para identificar y gestionar los datos [9]. JSON nació como una alternativa a XML, el fácil uso en JavaScript ha generado un gran número de seguidores de esta alternativa. Una de las mayores ventajas que tiene el uso de JSON es que puede ser leído por cualquier lenguaje de programación. Por lo tanto, puede ser usado para el intercambio de información entre distintas tecnologías. Este último hecho, ha sido el principal por el que se ha usado para codificar los mensajes entre los dispositivos empotrados, la aplicación móvil y el servidor web REST.

2.4.19 Identificación por Radio Frecuencia (RFID)

RFID es una tecnología “contactless” desarrollada a principios del siglo XX (el año exacto crea confusión, pues diversas tecnologías primarias parecidas a esta, fueron desarrolladas alrededor de esos años por diversas organizaciones, la mayoría bélicas) [10].

Esta tecnología posee muchas variantes, ya que las etiquetas RFID pueden ser pasivas o activas y los lectores pueden ser de baja, alta u “ultra alta” frecuencia. En este proyecto usaremos unas etiquetas pasivas (por su bajo coste) y un lector de “ultra alta” frecuencia.

En un principio, se pensó en usar un lector industrial, el cual permitiría leer la tarjeta sin necesidad de que el cliente tuviera que pasar la tarjeta por un lector, simplemente debería llevarla en el vehículo. Esta opción fue descartada dado que exigía realizar un trabajo electrónico de adaptación a dispositivo empotrado que ocuparía un trabajo de esta o mayor extensión, por lo que se decidió usar un lector más pequeño, aunque supuso también complicaciones que luego veremos en la fase de implementación.

2.4.20 Dispositivo Intel Edison

Dispositivo integrado del tipo “System on Chip” (SoC), el cual posee un procesador Dual Core Intel Atom, con un 1 GB de RAM, controlador Bluetooth, Wifi y controlador SD [19]. Además de esto, posee un controlador (del mismo tipo que los dispositivos Arduino) lo cual lo hace totalmente compatible con el IDE de Arduino y sus librerías.

Los motivos principales por los que me he decantado por esta opción frente a un Arduino UNO convencional son:

- Facilidad de Uso de Wifi: Al estar integrado, permite su uso de forma fácil y rápida mediante la librería estándar de Arduino.
- Infinitas posibilidades de cara al futuro: Aunque se describirá más adelante en el apartado *Conclusiones y Líneas Futuras,* las posibilidades de integración del propio servidor web dentro del dispositivo permitirían un ahorro en cuanto a adquisición de un equipo informático se refiere.

2.4.21 Diversas Librerías Para Intel Edison

Aunque en el apartado de *Implementación* se detallará de forma más precisa, se han usado diversas librerías, algunas de ellas estándar (como la controladora del Wifi [21] o la de JSON [20]) y otras creadas por mí (como la de RFID).

2.4.22 Librería para Escaneo de Códigos QR ZXING

Esta librería open source y libre, es habitualmente usada en proyectos para el escaneo de códigos QR en Android [11].

3. Implementación

Como se detalló en el *Capítulo 2: Diseño y Arquitectura*, el sistema principal aquí expuesto se desarrolló bajo una programación por capas. La programación por capas, posee innumerables ventajas frente a otros tipos de programación, pero destacan dos: escalabilidad; pues el sistema puede subdividirse de forma física en subsistemas más pequeños donde cada equipo se ocupa de una o varias tareas (un equipo de presentación y negocio y otro de datos, por ejemplo) y fomenta el trabajo en equipo, pues cada capa es "casi dependiente" del resto, pudiendo desarrollarse por separado bajo unos requisitos comunes mínimos.

En cuanto a la aplicación Android, se usó un modelo parecido a la programación por capas, aunque en este caso concreto, la capa de negocio y la de datos están unidas en una misma capa (lo que en Android recibe el nombre de "Activity").

Por último, en cuanto a la programación del dispositivo Intel Edison, se trata de un código C++ mono-hebra.

3.1 Subsistema Principal: Servidor Web

Como se ha detallado, su programación se efectuó por capas, por lo que se detallarán la implementación de cada una de forma independiente y en el orden temporal en que se efectuó su implementación, dando especial importancia a los detalles que marcaron su importancia.

3.1.1 Capa de Datos

Tras diseñar la arquitectura y las tecnologías a usar, el primer paso, fue implementar la arquitectura a capas. Para ello, se usó Java Persistence API (en adelante, JPA), framework que abstrae la gestión de las bases de datos relacionales, simplificándola como si al tratamiento de objetos se tratara. Este comportamiento se puede ver en la siguiente figura:

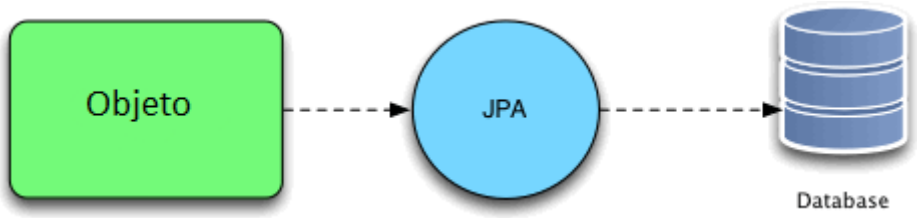


Figura 14: Funcionamiento JPA.

Además de este, tiene otra utilidad añadida: es posible la creación de esquemas de Base de Datos Relacionales mediante una notación específica en objetos (los cuales luego usará para extraer los datos de la base de datos. Los principales elementos de esta notación son: *@Entity, @Id, @Column, @ManyToMany, @ManyToOne y @OneToMany*. En el siguiente ejemplo, podemos ver su uso:

```
@Entity
public class Usuarios implements Serializable {

   @Id
   @Column(length = 20)
   private String nameuser;
   @Column(nullable=false, updatable=true,length = 32)
   private String pass;
   @Column(nullable=false, updatable=false,length = 20)
   private String rol;
   @Column(nullable=false, updatable=true,length = 20)
   @Enumerated(EnumType.STRING)
   private EstadElem stado;
   @Column(nullable=true, updatable=true)
   private byte[] img;

   @ManyToOne
   private Persona pers;

   public Usuarios() {
   }
```

Figura 15: Ejemplo de JPA

La utilización de cada etiqueta es simple:

- @Entity: Indica al compilador que el objeto tendrá correspondencia con una tabla de la base de datos.
- @Id: Indica cual es la clave primaria de la tabla.
- @Column: Etiqueta no obligatorio, pero que permite añadir funcionalidad a la tabla creada en el esquema de base de datos. En el ejemplo, se puede observar por ejemplo establecer la posibilidad de valor nulo, de que el campo sea actualizable y la longitud del mismo
- @ManyToOne: Como el resto de etiquetas de este tipo, designa una relación con otra tabla (en este caso con la tabla Persona).

Además de estas notaciones, es necesario que el objeto cumpla los requisitos de poseer un constructor vacío, métodos get y set para cada atributo, equals y hashcode y método toString.

Una vez se construyeron todos los objetos de acuerdo a las especificaciones/requisitos anteriores, se procedió a la configuración del fichero "persistence.xml" el cual posee una gran cantidad de configuraciones como la estrategia a seguir en cada ejecución del proyecto (crear esquema, borrar esquema, etc.), entidades a crear, método de cacheo a seguir, base de datos a conectar, etc. Este fichero es el encargado de configurar el comportamiento del "Contexto de Persistencia", el cual es el encargado de manejar las peticiones hechas por la capa de negocio en lo que a datos se refiere.

Tras tener lista la capa de datos, se procedió a ejecutar una serie de pruebas que incluyeron:

- Prueba de Cumplimentación de Requisitos: Consistió en comprobar de manera rápida pero eficaz, que el actual esquema de base de datos permitía todos los requisitos establecidos (tanto los primarios como secundarios). Esta prueba tuvo que repetirse en varias ocasiones, ya que ciertos requisitos no se cumplimentaban lo que supondría un gran atraso a la hora de implementar la lógica de negocio.
- Prueba de Introducción/Extracción de Datos: Se probó mediante la creación de un pequeño EJB (se explicará en el subapartado *Capa de Negocio)*. Al cual se le incluyó un objeto de la clase "Entity Manager" para el acceso a la base de datos.
- Prueba Correcto Funcionamiento de las Relaciones: Se probaron a meter datos relacionados, comprobando que estos mantenían la relación el base de datos

y tras eliminaciones o modificaciones mantenían la base de datos en un estado estable.

Una vez finalizadas las pruebas, se pasó a la implementación de la capa de presentación, pues la capa de negocio es la controladora de la de datos y de la de presentación y, por tanto, la última en implementar.

3.1.2 Capa de Presentación

La implementación de la capa de presentación consistió en un total de tres fases: Diseño HTML, adaptación HTML a XHTML y finalmente creación de los "manipuladores" de las vistas, es decir, los Java Server Faces (en adelante, JSF o beans).

La primera fase, se elaboraron los diseños sobre HTML. El motivo principal por el que se hizo este paso (a priori innecesario) fue la ayuda que supone usar el plugin Emmet con Sublime Text, ya que ayuda al auto-completado de sentencias o la escritura por atajos como si de un entorno de desarrollo para HTML se tratase. Previo a comenzar con la implementación, se descargaron las librerías *"Twitter Bootstrap", "JQuery" y "Omnifaces"*, a fin de poderlas usar luego en la implementación HTML. Una vez descargadas, se comenzó la implementación de las diversas páginas que componen el sistema web. Esta tarea fue relativamente sencilla gracias a la efectuación de un curso sobre *"Bootstrap",* anterior a la fase de implementación. Como podemos ver en la figura posterior, se usó en todo momento la paleta de colores descrita en el apartado "*Diseño y Arquitectura":*

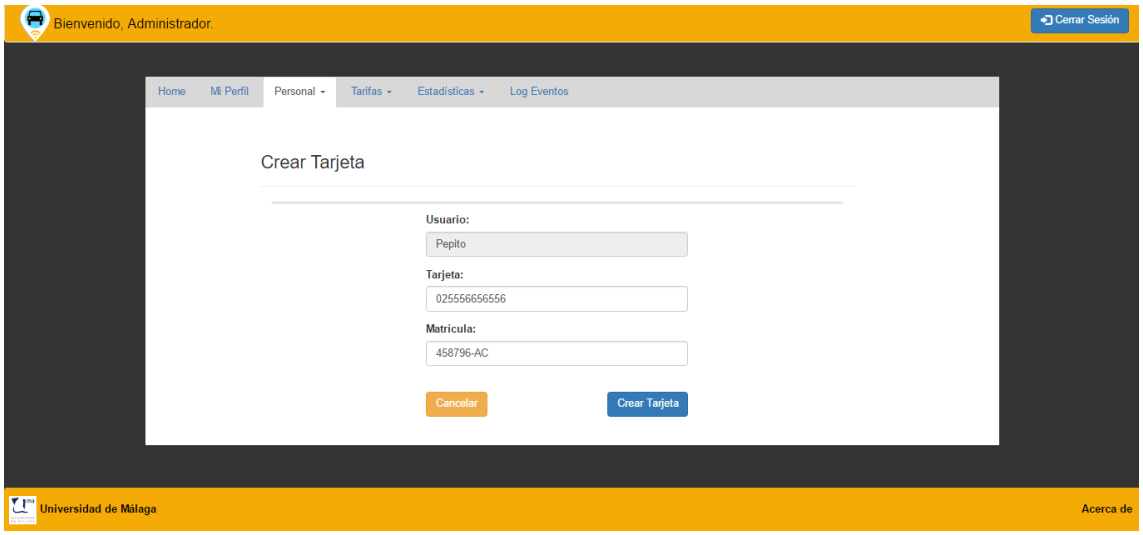


Figura 16: Ejemplo de Página Web

Una vez todas las páginas fueron creadas, se comenzó con la migración a páginas XHTML. Esta migración consistió en importar las librerías necesarias y efectuar una comprobación sobre la obligatoriedad de cierre sobre las etiquetas, es decir, que toda etiqueta deba acabar con "/>" o con una etiqueta de cierre.

```
xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://xmlns.jcp.org/jsf/html"
```

```
<br>
<input type="text" id="user" placeholder="Usuario">

<br />
<input type="text " id="user" placeholder="Usuario"></input>
```

Figura 17: HTML a XHTML

El segundo par de etiquetas cumple los requisitos para ser XHTML y HTML. Sin embargo, el primero, no cumple los requisitos para ser XHTML, por lo que el navegador daría error al intentar interpretar los elementos.

Finalmente, y para finalizar la capa de presentación, es necesaria la implementación del "controlador de las vistas" o JSF. En el caso de este proyecto y con el motivo de facilitar el posterior crecimiento incremental, se implementaron un total de 37 clases JSF. Cada clase JSF se corresponde con al menos una página XHTML, aunque algunas se definen para las páginas de funcionalidad parecida, como pueden ser todas las páginas de cambiar contraseña de los distintos usuarios. Para la implementación de los JSF hay que tener en cuenta las siguientes directrices:

- Establecer el ámbito del JSF: Cuando hablamos de ámbito nos referimos a durante cuánto tiempo debe estar activo el JSF. Un JSF es un objeto el cual estará activo durante un tiempo determinado a partir del cual desaparecerá, este tiempo puede ser definido principalmente como *@ViewScoped (activo mientras la vista a la que maneja lo esté), @RequestScoped (activo durante la petición HTTP que llame a la vista correspondiente), @AplicationScoped (activo mientras la aplicación lo esté) y @SessionScoped (activo mientras que la sesión el navegador esté activa).* En el caso particular de este proyecto, se han definido todos los Beans con ámbito *@ViewScoped* debido a que manejan una única página a la vez. Excepcionalmente se ha definido uno adicional el cual almacena elementos de la sesión necesarios para el correcto funcionamiento de la aplicación web como son el nombre de usuario, el rol, el nombre de la persona, etc.

- Métodos Get y Set por cada variable: Por estándar, cada vean debe poseer los métodos get and set necesarios para acceder a todos sus atributos. Estos atributos son definidos para poder almacenar los valores enviados en la petición HTTP. Así, por ejemplo, si se quisiera recibir una cadena de texto de un cuadro de texto, se debería definir un elemento “String” y luego referenciarlo desde la vista XHTML correspondiente.
- Constructor vacío: Al igual que los métodos Get y Set, debe estar definido por estándar.
- Crear métodos para los elementos dinámicos: Estos métodos son definidos a fin de poder mostrar los elementos dinámicos, como pueden ser listas de objetos definidos en los JPA. Un ejemplo de esto, puede ser la devolución de una lista de *Usuarios* para luego ser mostrados en una tabla.
- Sustituir elementos XHTML por elementos JSF, los cuales muestren el contenido dinámico: Estos elementos sirven para referenciar a los atributos o métodos antes mencionados, como se puede ver en el siguiente ejemplo:

```
<div class="form-group" style="width: 100%;">
   <label class="pull-left" for="oldpass">Contraseña Actual:* </label>
   <h:inputSecret id="oldpass" value="#{cambiarPass.passOld}" class="form-control"></h:inputSecret>
</div>
<div class="form-group" style="width: 100%;">
   <label class="pull-left" for="newpass">Contraseña Nueva:* </label>
   <h:inputSecret id="newpass" value="#{cambiarPass.passNew}" class="form-control"></h:inputSecret>
</div>
```

Figura 18: Ejemplo de métodos JSF.

Esta sustitución pese a parecer una tarea trivial, requirió de un gran esfuerzo por diversos problemas surgidos. Los principales fueron la dificultad para trascribir algunos elementos HTML o la importación y visualización de imágenes. Respecto al primero, se encontró dificultad al transcribir los atributos “placeholder” y “size” de los campos de texto, por ejemplo, teniendo como solución la importación de librarías específicas. Respecto al segundo, se usó la librería *Omnifaces* la cual se encarga de abstraer el manejo de imágenes (tanto la importación como la visualización) como si de flujos de bytes estándar se tratara.

Una vez finalizadas las tareas descritas, la capa de presentación está preparada para ser unida a la *Capa de Presentación*. Esto se realizará mediante la implementación de la *Capa de Negocio,* capa que se encargará de gestionar las peticiones por parte de los *Beans* y extraer los datos pedidos de la *Capa de Datos*.

3.1.3 Capa de Negocio

La Capa de Negocio es implementada mediante los llamados *Enterprise Java Beans (en adelante EJB).* Se denomina EJB al objeto encargado de acceder al contexto de persistencia (antes nombrado y detallado) y realizar consultas mediante un "*EntityManager"*. Un "*EntityManager"* es una interfaz para el acceso al contexto de persistencia. Además de esto, debemos definir el tipo de acceso mediante las etiquetas *@Singleton o @Stateless*. Con el primero, establecemos un EJB el cual crea una instancia por aplicación mientras que con el segundo se crea una instancia por cada cliente (independencia entre clientes). En nuestro caso, definiremos todos los EJB con la etiqueta @Stateless. Además de esto, deberemos "inyectar" el "*EntityManager"* antes nombrado, de acuerdo al contexto de persistencia que creamos al principio del proyecto. En la siguiente figura se puede ver lo definido en este párrafo:

```
@Stateless
public class DispositivosEJB implements Serializable {

   @PersistenceContext(unitName = "ACPejbPU")
   private EntityManager em;
```

Figura 19: Implementación base de un EJB.

Como detalles a destacar de esta capa se pueden destacar los siguientes:

- Conexión JSF-EJB-JPA: Para realizar esta conexión, hubo que solventar una serie de problemas. Estos problemas fueron de índole muy distinta:
  - Almacenamiento seguro de credenciales: Para añadir algo más de seguridad al sistema, se ha usado un sistema de huella o "hash" para el almacenado de contraseñas, más concretamente, el algoritmo MD5.
  - Problemas en la transición entre pestañas con datos heredados: Un ejemplo de este problema se da al abrir una noticia desde el home de cualquier usuario. Para posteriormente verla detallada, es necesario almacenar el id en el Bean con ámbito de sesión, para al acceder a la página de visualización de noticia, ésta sepa la noticia a la que recurrir.
  - Inyección de los EJB en los JSF: Para acceder a las capas de negocio, es necesario inyectar los EJB mediante la etiqueta *@EJB*.
- Consultas TypedQuery: En ciertas ocasiones, es necesario la extracción de una lista de objetos JPA, lo que hace provoca la necesidad de realizar consultas amplias. Esto puede verse en la siguiente figura:

```
TypedQuery<Dispositivo> query= em.createQuery("Select n from Dispositivo n", Dispositivo.class);
```

Figura 20: TypedQuery

- Log de Eventos: Dado que en la aplicación se ha integrado un Log para controlar ciertos eventos de importancia como son la creación, la actualización o el borrado, ha sido necesaria la inyección del EJB correspondiente (LogEJB) en el resto de EJBs para así poder insertar la fila correspondiente al evento en la tabla de Log.
- Servicios REST: Los servicios REST son implementados mediante la opción correspondiente de NetBeans. Estas clases, deben tener un ámbito establecido el cual en este caso será @RequestScoped y al igual que en los JSF, deben inyectarse los EJB correspondientes además de realizar una serie de notificaciones estableciendo el tipo de petición HTTP que llamará a ese método (GET, POST, UPDATE, etc.) así como la entrada y salida que usará el método (json, xml, etc.). Esto podemos verlo en la siguiente figura:

Figura 21: Ejemplo de REST.

*Este método será ejecutado tras recibir el servidor una petición HTTP de tipo POST, y consumirá y enviará un objeto tipo JSON.

Finalizada esta etapa, el servidor web queda completamente implementado y funcional, preparado para la conexión con el resto de subsistemas.

3.2 Subsistema Principal: Intel Edison

Desde el punto de vista del aprendizaje durante el proyecto, esta parte fue la que supuso un mayor desafío debido a que la plataforma Intel Edison es poco conocida actualmente debido a su relativa juventud (2014) y actualmente muchas librerías compatibles con Arduino, no lo son con Edison.

La primera complicación surgida fue relativa al uso del dispositivo RFID, ya que se trataba de un dispositivo poco usado en la actualidad, por lo que no existía ninguna librería creada. Por este motivo, fue necesario en primera instancia la creación de una librería expresa para este dispositivo. Tras revisar ciertas webs y blogs, se comenzó la implementación de la librería "*RFID.h"*, lo cual supuso un desafío. Esto conllevó un tiempo más extenso del pensando en primera instancia, incumpliendo la meta correspondiente, aunque finalmente y tras muchos inconvenientes surgidos de las incompatibilidades antes mencionadas, se consiguió la correcta implementación de la misma.

Otra complicación, como se avanzó en el anterior apartado, fue la adaptación de librerías. Puntualizando, las librerías *"aJson"[20]* y *"RestClient"[21]* debieron ser editadas convenientemente para poder ser ejecutadas en Intel Edison. Respecto a la librería para el manejo de objetos "*aJson",* ha sido necesaria una modificación de las librerías secundarias de la cual se abastece para el manejo de los objetos. En concreto, la librería relativa *"pgmspace"* tuvo que ser adecuada a Intel Edison, pues la estándar de Arduino era incompatible. Respecto a "*RestClient"* fue necesaria una adaptación más profunda, ya que hubo que adaptar su funcionalidad para el uso de la tecnología Wifi en lugar de la tecnología Ethernet cableada. Otro cambio significativo, fue el cambio de la implementación del método POST correspondiente, ya que no estaba adaptado al manejo de elementos JSON.

Por último, se debieron hacer unos cambios en lo que se refiere al funcionamiento normal del dispositivo Edison. Precisando, el funcionamiento normal del dispositivo hace que tras programarlo y ser reiniciado, se borre el esquema programado de la memoria, provocando la pérdida de esa programación efectuada. Para solventar el problema, se configuró la imagen Linux del dispositivo llamada "Yocto" para ejecutar un *Bash Script"* al inicio que dejara sin efectos este funcionamiento. Otro aspecto fue el uso del puerto Serie, ya que éste viene deshabilitado por defecto. La solución fue igual que al anterior problema, además del uso de la variable "*Serial1"* en lugar de "*Serial"* como ocurre en un dispositivo "Arduino UNO" convencional.

En cuanto al montaje físico del sistema, se utilizaron 3 leds (Rojo, Verde y Azul), un piezo (para emitir sonidos), 4 resistencias (3 de 220 Ohmios y 1 de 10 KOhmios), el dispositivo Intel Edison, el lector RFID de 125 Khz con Antena, un pulsador y cableado. La conexión de estos elementos se realizó siguiendo el siguiente orden:

1. Ensamblaje Intel Edison y Lector RFID: En primer lugar, se colocó el Lector RFID sobre el dispositivo empotrado, teniendo en cuenta que el lector necesita de una shield de adaptación para poder ser conectado. Esta shield, transmite los pines 0 y 1 (conexión UART) así como el pin de alimentación de 5 voltios [15].

2. Conexión Leds de Estado: Estos leds se encargarán de iluminarse dependiendo el estado en el que se encuentre el dispositivo: Rojo para error, azul para ocioso y verde para éxito. Estos leds, se conectarán en serie a una resistencia de 220 Ohmios cada uno (que irá a tierra) y por el otro extremo a los pines 3, 11 y 12 (para azul, verde y rojo, respectivamente) [17].

3. Conexión Piezo: El piezo es un elemento el cual mediante vibraciones emite sonidos. En nuestro caso, usaremos este elemento para emitir una respuesta

afirmativa o negativa dependiendo si la lectura fue correcta o errónea. Para ello se conectará al pin 9.

4. Conexión Botón Multifunción: Debido a que se pretendía simular el uso de los tres tipos de dispositivos, fue necesaria la inclusión de este botón para poder cambiar la función que ejecutaba el dispositivo.

Tras seguir los pasos enunciados, el sistema debería quedar así:

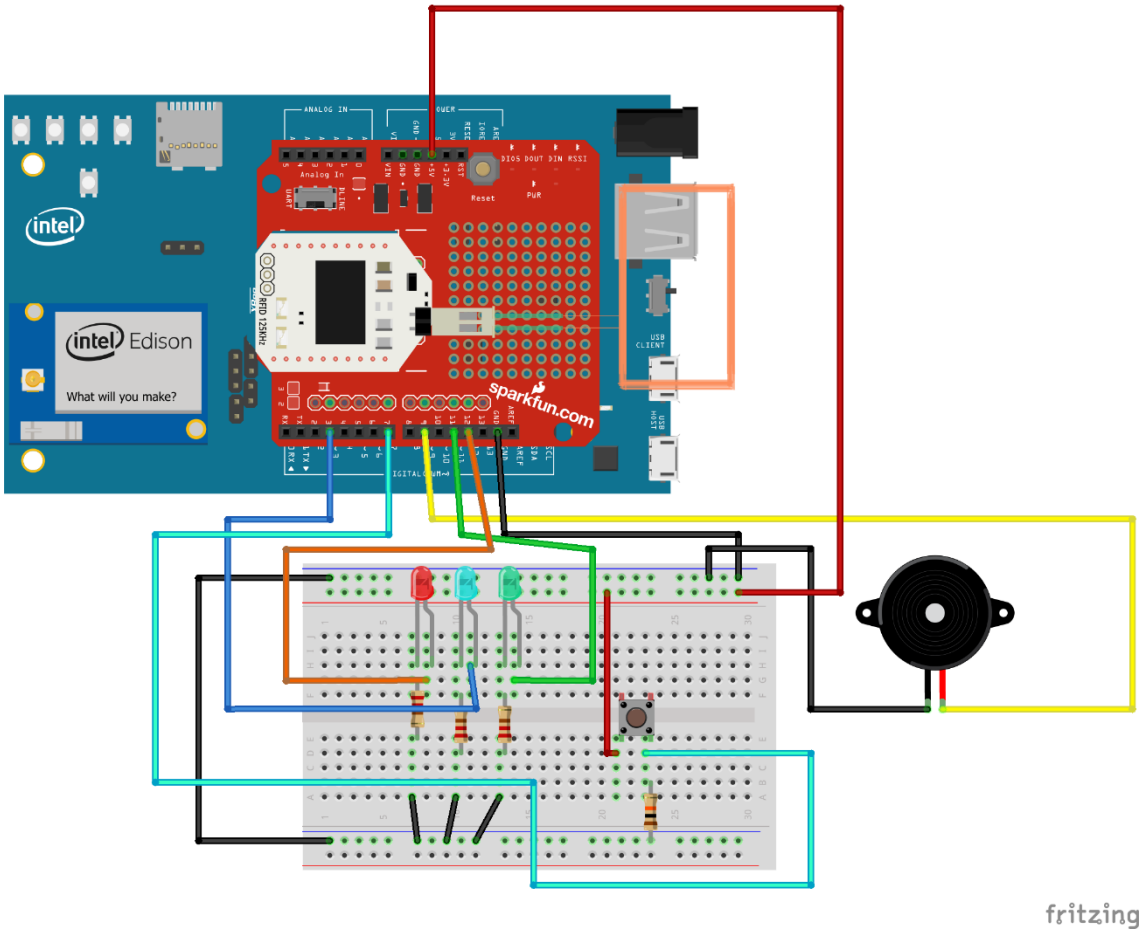


Figura 22: Diagrama de Conexión Intel Edison.

Una vez listo el sistema y comprobada su correcta conexión al servidor principal, se pasó a la implementación de la Aplicación Android.

3.3 Subsistema Secundario: Aplicación Android

El desafío en la implementación de este sistema vino por el uso de la conexión a Internet, así como el uso de la cámara como escáner de códigos QR.

La conexión a Internet en Android no es una tarea trivial, ya que, en primer lugar, deben establecerse los permisos necesarios en el archivo "*AndroidManifest.xml" [16]*. Esto se implementa mediante etiquetas predefinidas con la siguiente estructura:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Figura 23: Estructura Permisos Android.

Tras esto, debe comprobarse la conexión a internet mediante la clase "*ConnectivityManager*" con la que se puede obtener los adaptadores de Internet (Wifi y Red Móvil) y comprobar si alguno posee actualmente conexión a Internet. Una vez se comprueba la existencia de conexión, se puede pasar a realizar las peticiones, las cuales se harán usando la librería externa "*Volley" [24]*. Esta librería supone una gran ayuda a la hora de realizar este tipo de peticiones ya que su uso consiste en generar objetos del tipo "*JsonObjectRequest"* en este caso, establecer el tipo de petición (GET, POST, etc.), establecer las tareas a realizar en caso de respuesta correcta y errónea (mostrar datos de ocupación en el primer caso, y mostrar notificación de error en el segundo) y finalmente incluir la petición en la cola de peticiones.

Finalizadas esto, la aplicación Android queda finalizada, siendo totalmente utilizable.

4. Testeo y Mejora

En este capítulo se describirán tanto la fase de testeo del sistema, como la fase de mejora incremental. Por ello, se ha visto conveniente la separación de ambas fases en subapartados distintos a fin de poder diferenciarlos.

4.1 Testeo

En este apartado se detallarán las distintas pruebas realizadas sobre el sistema a fin de comprobar el correcto funcionamiento del mismo, su estabilidad y su ajuste a los requisitos establecidos. Para ello, se realizaron pruebas de diversa índole [14].

4.1.1 Pruebas Unitarias

Se denominan pruebas unitarias a aquellas que tiene por fin, probar el correcto funcionamiento de un conjunto reducido de líneas de código (normalmente de un método o función). En el caso específico de este proyecto, se efectuó una prueba tras la implementación de cada método probando su correcto funcionamiento. Si el método incluía una llamada a la Base de Datos, antes de ejecutar la prueba, se procedió a la inserción de datos de prueba (con salida conocida), evaluando así, si la salida aportada era correcta.

4.1.2 Pruebas de Integración

Una vez se completaban las pruebas unitarias sobre los métodos interventores de un elemento más grande (por ejemplo, la visualización de una página de la web completa), se somete a prueba este elemento mayor, recibiendo estas pruebas el nombre de Pruebas de Integración. La ejecución de estas pruebas se realizó de manera similar a las unitarias, pues en cierta forma, son iguales salvo por que el sistema probado no es sólo un bloque de código, sino un elemento algo mayor.

4.1.3 Pruebas de Sistema

Una vez el sistema estaba listo para su uso, se realizaron diversas trazas para explorar todos los caminos posibles y verificar el cumplimiento de los requisitos. Esto se hizo de forma automática gracias a la herramienta Selenium, ya que posteriormente sería necesario volver a comprobarlo cuando se realizarán los distintos incrementos.

4.1.4 Pruebas de Carga

Las pruebas de carga consisten en evaluar el sistema bajo diversas condiciones de entorno posibles. En el caso de este proyecto, estas pruebas no han sido ejecutadas debido a que pese a tratarse de un sistema web (para los cuales estas pruebas son recomendables), este sistema está pensado para funcionar en el sistema de un cliente, por lo que el rendimiento dependerá de lo potente de este sistema.

4.1.5 Pruebas de Aceptación y Usabilidad

Pese a que los resultados de estas pruebas no han podido ser evaluados a ciencia cierta, debido al pequeño grupo sobre el que se realizaron los test (4 personas), todos destacaron su gran manejabilidad gracias a la interfaz por pestañas, su categorización de contenidos y su cómoda gama de colores no molesta a la vista.

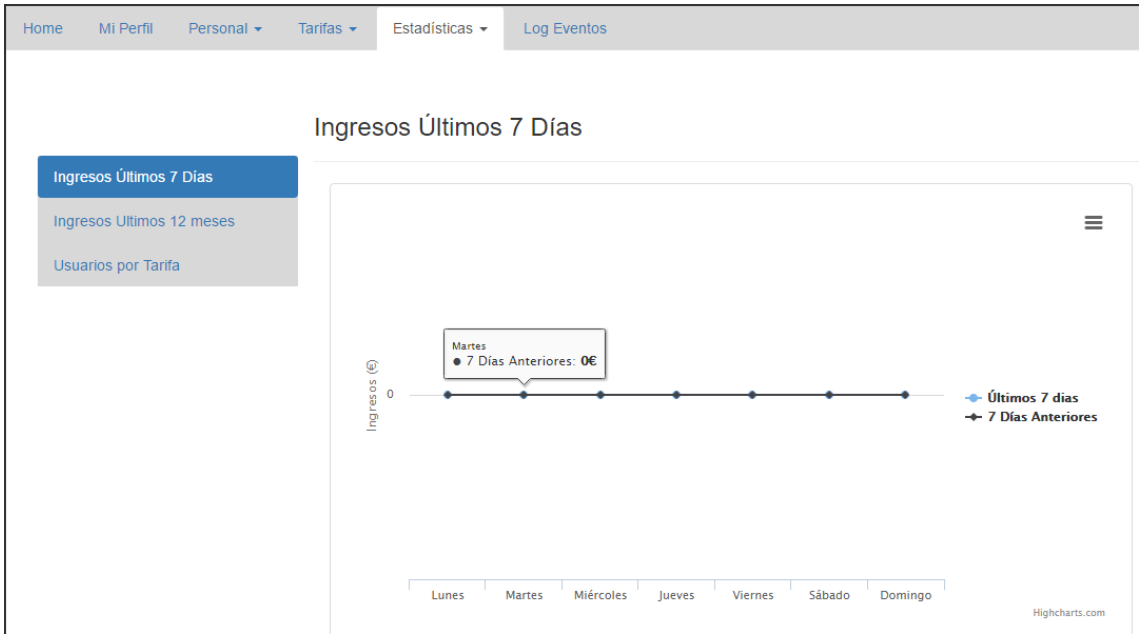
4.2 Mejora

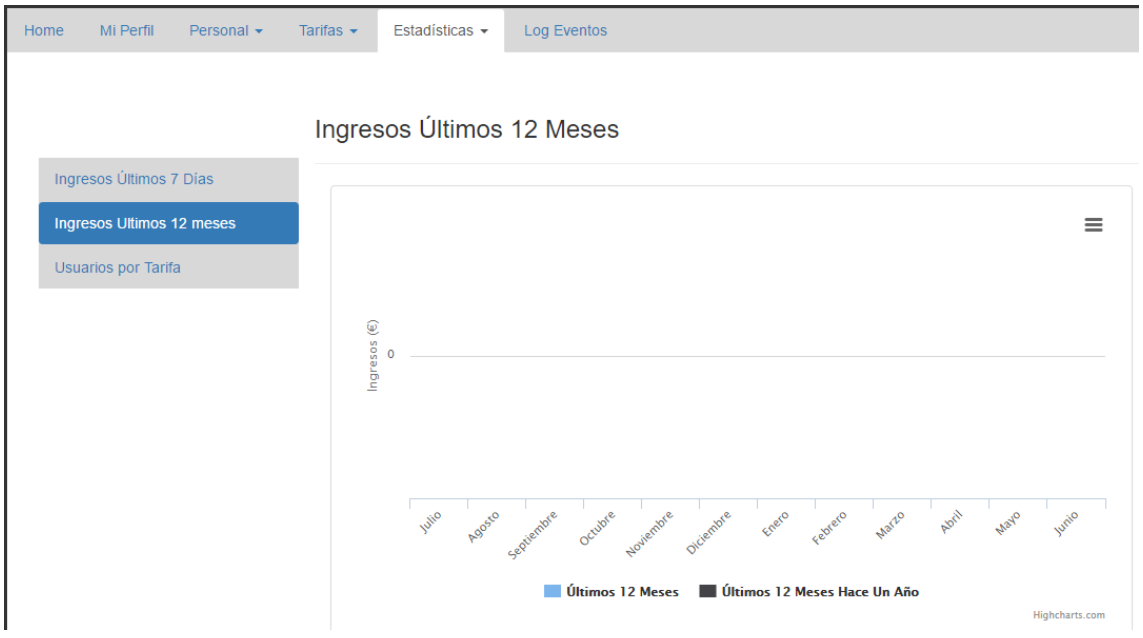
En esta fase, se fueron implementando los distintos requisitos nombrados en el subapartado de Requisitos Secundarios, en el apartado Arquitectura y Diseño.

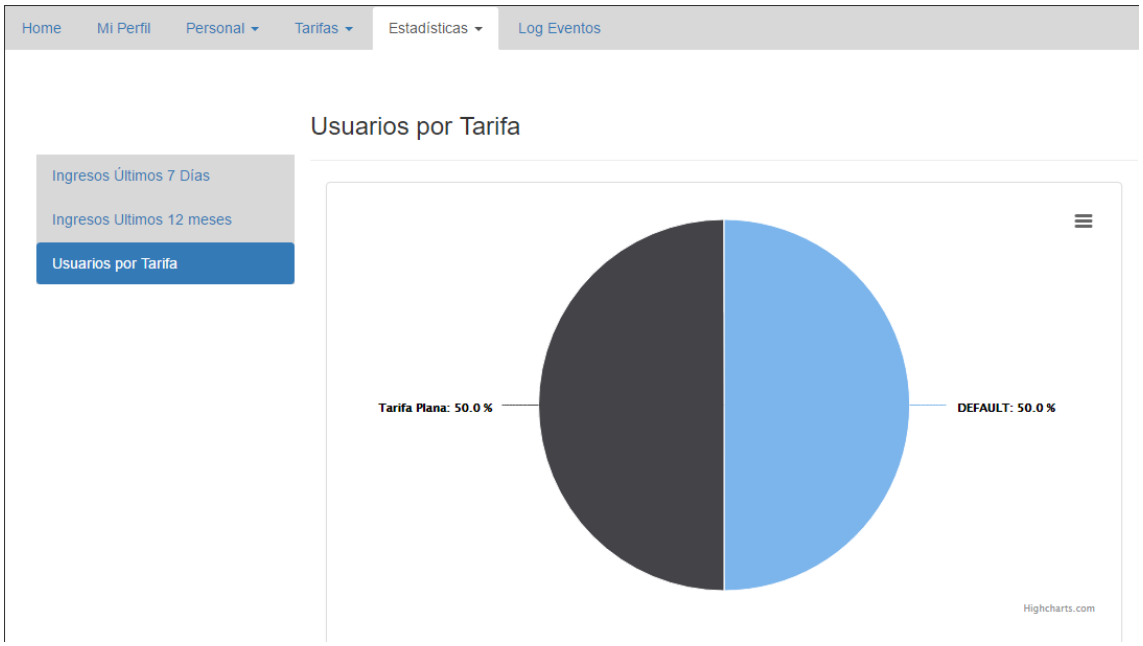
Para esclarecer el desarrollo efectuado durante esta fase, a continuación, se exponen los pasos seguidos, así como el requisito implementado en esa fase y las pruebas realizadas para comprobar el correcto funcionamiento del sistema tras el cambio. Llegados a este punto, conviene detallar que para esta fase fue muy útil el uso de la una herramienta de versionado, en el caso de este proyecto, se usó la herramienta integrada de NetBeans, es decir, *Git.* El control de versiones permite la posibilidad de realizar cambios, registrarlos y en caso de un fallo de implementación, volver a la versión anterior.

4.2.1 Visualización de Estadísticas

Dado que el administrador más allá de la gestión de tarifas, no poseía otra función específica, se vio interesante la idea de añadir algunas funcionalidades más a este perfil, más cercanas a la gestión financiera y numérica del aparcamiento, así, el administrador puede gestionar y llevar un control mayor del aparcamiento desde la distancia. Para ello, se incluyó la librería *HighCharts,* librería preparada para la creación de todo tipo de gráficos. Tras la importación de esta librería y posterior uso de sus elementos JavaScript, se pasó a la adaptación de las clases JSF así como la creación de nuevos métodos en los EJB correspondientes. Tras esto, se obtuvieron los siguientes resultados:







Figuras 24, 25 y 26: Estadísticas HighCharts

Tras implementar estas estadísticas gráficas, se pasó a implementar otra serie de estadísticas sobre tabla, las cuales pueden ser modificadas añadiendo o borrando, mediante la modificación de un método (buscando así la personalización si fuera necesaria).

Figura 27: Estadísticas Textuales

Para comprobar estas modificaciones, se realizaron una serie de pruebas, añadiendo datos de ocupaciones y facturas controladas, conociendo los datos que deberían proporcionar las estadísticas y comprobando si realmente estas estadísticas proporcionan los datos correctos.

4.2.2 Auto-Registro de Dispositivos en el Sistema

El sistema base permite la inclusión de dispositivos de forma manual, por lo que cada dispositivo debe ser agregado mediante el Id por un trabajador. Esto supondría un gran esfuerzo en caso de aparcamientos con un gran número de plazas. Para dar solución a este inconveniente, se decidió la inclusión de un sistema de auto-registro de dispositivos mediante contraseña (configurada en el servidor). Para permitir esta posibilidad, se tuvo que crear un nuevo servicio REST, al cual se conectaran los dispositivos para comprobar su registro y en caso contrario, efectuarlo.

La comprobación de esta funcionalidad, se realizó mediante la prueba de registro de estos tres tipos de dispositivo, ya que se trata de una funcionalidad simple.

4.2.3 Acceso a Datos de Ocupación mediante Código QR

En el sistema base, el acceso a los datos de ocupación mediante la aplicación móvil, se efectúa mediante la escritura manual del Id en la aplicación. Se pensó, que esta acción podría dar una imagen muy contraria a la esperada de un sistema como éste, el cual pretende innovar y actualizar tecnológicamente a los aparcamientos. Por ello, se decidió la inclusión de códigos QR en las tarjetas entregadas a clientes, así como la introducción de un escáner de códigos QR integrado en la aplicación Android.

Para el análisis de códigos QR, me serví de la librería libre y open source llamada "*ZXING"*. Para el uso de esta librería, fue necesaria la importación del "*core",* ya que, si solo se usara la propia librería, obligaríamos al cliente a descargar otra aplicación para poder realizar la lectura de códigos QR, acción que provocaría un gran descontento además de una molestia significativa al cliente. Esta inclusión debió realizarse modificando las dependencias del compilador "*Gradle"*, obligándole a la inclusión del fichero que antes se comentaba.

Como pruebas de implementación, se efectuaron una serie de escaneos sobre códigos QR válidos (los cuales tiene Id de tarjetas registradas) y códigos QR inválidos (con cadenas de texto al azar). Estas pruebas fueron satisfactorias.

4.2.4 Visualización de la Relación Plazas-Clientes

Esta utilidad se basa en la relación existente entre las plazas ocupadas y los clientes que la ocupan. Esta implementación fue relativamente sencilla debido a que no entrañaba mayor complicación que la de esclarecer esa relación.

Como pruebas, se realizaron las varias ocupaciones ficticias, comprobando que estas efectivamente se mostraban en la plaza adecuada.

4.2.5 Tres tipos de Tarifa

Esta característica se refiere a la posible tarificación del aparcamiento. En la versión inicial, sólo se permitía la creación de un tipo de tarifa, aquella que por cada minuto fija el importe. Esto se vio insuficiente para la fidelización de clientes, ya que no permitía unas tarifas más flexibles como pueden ser las Tarifas Planas. Es por este motivo que se incorporó al sistema dos nuevos tipos de tarifa: la Tarifa Plana y la Tarifa por Minuto con Mínimo. El comportamiento de estas nuevas tarifas, consiste en la Tarificación Mensual, Semanal o Diaria de un importe exacto (Tarifa Plana) o de un importe mínimo (Tarifa por Minuto con Mínimo). Para la inclusión de estas tarifas, bastó con la modificación de la página XHTML, el correspondiente JSF y finalmente el método cerrarOcupación de la clase Configuration (el encargado de facturar las ocupaciones).

Para comprobar el correcto funcionamiento de estas nuevas tarificaciones, se probó a realizar distintas ocupaciones con distintas tarifas viendo si en todo momento mostraba correctamente el importe.

4.2.6 Permitir Imágenes en el Sistema

Como se ha mencionado varias veces a lo largo del presente documento, esta carga de imágenes fue posible gracias al uso de la librería Omnifaces. Esta librería permite la abstracción de la presentación o subida de imágenes, simplificando su gestión a la gestión de un flujo de bytes (para la subida) y un array de bytes (en la presentación).

La inclusión de esta librería en el proyecto se realizó mediante la importación de los elementos en los documentos XHTML y su correspondiente JAR en las importaciones del proyecto.

Las pruebas realizadas consistieron en la subida y mostrado de imágenes de diversos tamaños y formatos, comprobando que no se alteraran.

4.2.7 Histórico de Ocupaciones

Tal vez, esta modificación fue la que más reforzó el valor comercial de esta herramienta. En la primera versión del sistema, al borrar una tarjeta o usuario, sus ocupaciones desaparecían del sistema, haciendo que fuera imposible llevar una cuenta clara de lo ganado en ese año o mes. Para evitar esto, al borrar lo antes mencionado, las ocupaciones prevalecen en la base de datos, aunque desvinculadas de cualquier tarjeta, con esto, conseguimos mostrar estadísticas globales reales y no basadas en el alta o no de tarjetas y/o usuarios.

Al igual que las anteriores modificaciones de código, se probó mediante el uso de un conjunto de datos de prueba, así como de la creación de tarjetas y su posterior eliminación.

4.2.8 Log de Eventos

Esta mejora se trata de un añadido más para el administrador del sistema. Consiste, en llevar un control de las modificaciones importantes que se van realizando en el sistema, para así controlar los errores que puedan suceder en el mismo. Esta mejora, como he mencionado sólo estará disponible en el perfil del administrador y será posible sólo la visualización del Log, y no su modificación o borrado.

Para comprobar su funcionamiento, bastó con ejecutar todas las modificaciones posibles y comprobar que efectivamente quedan registradas.

Una vez se finalizaron todas las mejoras incrementales, fue necesaria la realización de *Pruebas de Regresión*. Este tipo de pruebas son ejecutadas cuando se realizan modificaciones sobre el programa, confirmando que estas funcionan correctamente y que su inclusión, no alteró el resto de funcionalidades dejándolas obsoletas o con errores. Para realizar estas pruebas bastó con realizar un conjunto de pruebas de Selenium (parte del cual ya estaba creado de la fase de pruebas anterior), y realizar su ejecución comprobando que todas arrojaran resultados correctos.

5. Conclusiones y Líneas Futuras

En este apartado, se recabará de manera clara y concisa las conclusiones obtenidas tras el desarrollo del Trabajo de Fin de Grado, así como enumerar posibles líneas de ampliación futuras.

5.1 Conclusiones

Para llevar a cabo la realización de este trabajo, se han tratado de cumplir todos los objetivos expuestos en el apartado correspondiente de esta memoria. En particular, con la realización de este trabajo se pretendió afianzar los conocimientos teóricos y prácticos aprendidos a lo largo del periodo académico (como Java EE o HTML), así como adquirir algunos nuevos en tecnologías emergentes (caso particular de Intel Edison y Servicios REST). El empeño y la motivación de aprendizaje han sido piezas fundamentales en el desarrollo del presente proyecto en todas sus fases.

En general, las fases que entrañaron mayor dificultad fueron sin duda la fase de análisis y la fase de planificación. En cuanto a la dificultad de la primera, residió en analizar correctamente las funcionalidades que debía tener el sistema para ser un sistema estable y útil (lo que técnicamente podemos llamar valor comercial y diferencial del producto). Respecto a la segunda, la dificultad principal residió en evitar caer en el exceso de confianza al establecer plazos demasiado cortos en ciertas tareas, y en analizar cada tarea de forma individual para así decidir correctamente los plazos de ejecución de las mismas y las pruebas que evaluarían su correcta implementación.

En cuanto a las dificultades técnicas encontradas, se pueden citar, a modo de resumen, como más relevantes las siguientes:

- Adaptación de librerías para Edison: Como se mencionó anteriormente, el uso de las librerías nativas de Arduino sobre Intel Edison, supuso un gran desafío debido a que no son compatibles en su totalidad y en algunos de los casos conllevó la revisión y adaptación manual de éstas.

- Creación de la librería RFID: Debido a que el módulo RFID usado en este proyecto no es muy frecuente (la mayoría de proyectos usan lectores basados en conexiones SPI), se tuvo que crear una librería propia (muy básica) a fin de poder usarlo posteriormente.

- Subida y Carga de Imágenes en la Web: Aunque este problema se simplificó con el uso de la librería Omnifaces, encontrar la solución no fue sencilla ya que esta librería no es muy conocida.

- Uso de Bootstrap junto con JSF: En un principio, se pensó en realizar confirmaciones de borrado mediante el uso de ventanas modales, sin embargo, al realizar el XHTML correspondiente, esto provocaba que el envío de formularios no se hiciese de forma correcta, por lo que se tuvieron que suprimir en favor de ventanas de confirmación de JavaScript, mucho más fáciles de usar y configurar.

- Afianzar conocimientos en codificación HTML: Pese a que en el grado se han estudiado HTML y sus tecnologías "amigas" (CSS, JS), éstas han sido estudiadas de forma muy superficial, ya que, por ejemplo, no dimos ningún tipo de framework. Para solventar esta carencia, fue necesario realizar una breve formación anterior a este proyecto a fin de aprender el manejo de Bootstrap así como de la codificación HTML.

- Uso de JSON desde Edison: Dado que los dispositivos empotrados tienen una memoria muy reducida, los elementos JSON deben ser manejados de forma cuidadosa para evitar el llenado (y bloqueo) de la memoria del dispositivo.

- Elaboración de Servicios REST: La complicación encontrada en este punto, fue principalmente la elaboración de las respuestas por parte del servidor y la inclusión de los elementos JSON de respuesta en las mismas, ya que se debe hacer de forma compleja.

Finalmente, mencionar que gracias a los conocimientos adquiridos durante el grado (especialmente en las fases de diseño y planificación) facilitaron la elaboración de este proyecto y consiguieron que ésta fuera satisfactoria.

5.2 Líneas Futuras

Como trabajo futuro y posibles ampliaciones del sistema, podemos mencionar las siguientes:

- Permitir más tipos de tarifa: Esto aportaría mayor versatilidad al aparcamiento, adaptándose a necesidades que puedan surgir como son fechas señaladas o intervalos de mayor afluencia.

- Mudar el Sistema a la Propia Intel Edison: Mudando el sistema, permitiríamos al aparcamiento ahorrarse los costos de adquisición del equipo informático que

actuaría de servidor, haciendo que éste funcione sobre Intel Edison dividiendo las carga entre varias de ellas.

- Mudar el Sistema a la Nube: Con esto subsanaríamos por completo el problema de la adquisición de servidores propios, ya que sería un único servidor en la nube controlando todos los aparcamientos. Con esto, se facilitaría en gran medida la instalación del sistema, así como ahorraríamos el costo hardware al cliente y daríamos la posibilidad al cliente de administrar varios aparcamientos (si el cliente fuera una gran empresa, por ejemplo).

- Incorporar al Sistema los elementos habitualmente Usados en Aparcamientos: Elementos como vayas, lectores, etc. de uso habitual en aparcamientos serían de gran interés integrarlos a fin de que los clientes puedan reutilizar sus equipos.

- Desarrollar más la aplicación Móvil: Un mayor desarrollo de la aplicación móvil haría que el cliente pudiese recibir más información de la misma sin necesidad de entrar en la página web.

- Cambiar la tecnología RFID por NFC: Esto aportaría un salto de calidad en el servicio, pues se abriría la puerta al uso de las “tarjetas virtuales” las cuales podría el cliente llevar en el móvil y usarlas a través del NFC presente cada vez más en estos dispositivos.

Como se puede observar, el sistema tiene una gran cantidad de líneas de futuro por las que seguir su desarrollo, siendo algunas de ellas muy interesantes de cara al uso comercial.

6. Referencias

[1] «Análisis de Requisitos: Wikipedia»
https://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_requisitos

[2] «Java EE: Oracle»
http://www.oracle.com/technetwork/java/javaee/overview/index.html

[3] «Casos de Uso: Wikipedia»
https://es.wikipedia.org/wiki/Caso_de_uso

[4] «ArgoUML: ArgoUML»
http://argouml.tigris.org

[5] «Netbeans: Wikipedia»
https://es.wikipedia.org/wiki/NetBeans

[6] «Android: Android Developer»
https://developer.android.com/develop/index.html

[7] «Apache Derby: Apache»
https://db.apache.org/derby/

[8] «GlassFish: Wikipedia»
https://es.wikipedia.org/wiki/GlassFish

[9] «JSON: Wikipedia»
https://es.wikipedia.org/wiki/JSON

[10] «RFID: Wikipedia»
https://es.wikipedia.org/wiki/RFID

[11] «Librería Lector QR: ZXing»
https://github.com/zxing/zxing

[12] «Desarrollo Iterativo Incremental: Proyectos Agiles»
https://proyectosagiles.org/desarrollo-iterativo-incremental/

[13] «REST: Wikipedia»
https://es.wikipedia.org/wiki/Representational_State_Transfer

[14] «Tipos de Pruebas Software: Ing-sw»
http://ing-sw.blogspot.com.es/2005/04/tipos-de-pruebas-de-software.html

[15] «Implementación RFID: Cooking-Hacks»
https://www.cooking-hacks.com/documentation/tutorials/rfid-125-khz-module-arduino-raspberry-pi-tutorial/

[16] «Programación Android: Sgoliver»
http://www.sgoliver.net/blog/curso-de-programacion-android/indice-de-contenidos/

[17] «Arduino Projects Book: Arduino»

[18] «Arduino Libraries: Arduino.org»
https://www.arduino.cc/en/Reference/Libraries

[19] «Intel Edison: Intel Software»
https://software.intel.com/es-es/iot/hardware/edison?cid=sem43700008151004940&intel_term=intel+%2Bedison+software&gclid=CjwKEAjwy6O7BRDzm-Tdub6ZiSASJADPNzYrRAX660OjsdAKF6tjONCHH0KuAu50mmundAhzitv3jBoCVhDw_wcB&gclsrc=aw.ds

[20] «Librería aJSON: GitHub»
https://github.com/interactive-matter/aJson

[21] «RestClient con WiFi: GitHub»
https://github.com/DaKaZ/esp8266-restclient

[22] «Librería Omnifaces: Omnifaces»
http://showcase.omnifaces.org/

[23] «Diversos Apuntes sobre Java EE y Servidores REST aportados por profesores durante el Grado»

[24] «Librería Volley: Android Developers»
https://developer.android.com/training/volley/index.html?hl=es

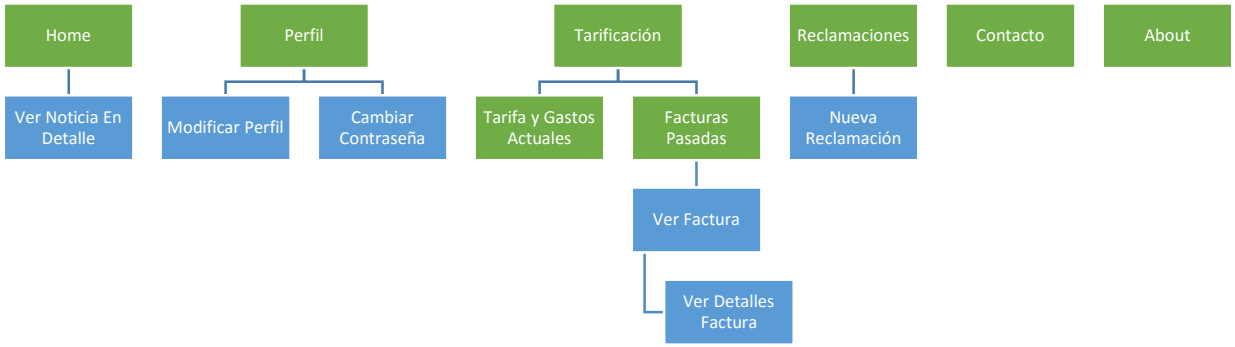
7. Anexos

7.1 Diagramas de Navegación

Estos diagramas reflejan los posibles “caminos” a seguir en cada pestaña de la aplicación. Como nota aclaratoria, mencionar que para acceder a los caminos aquí descritos, antes será necesario pasar por la página de *Login*, a través de la cual accederemos al sistema mediante un usuario y una contraseña.

7.1.1 Diagrama de Usuario

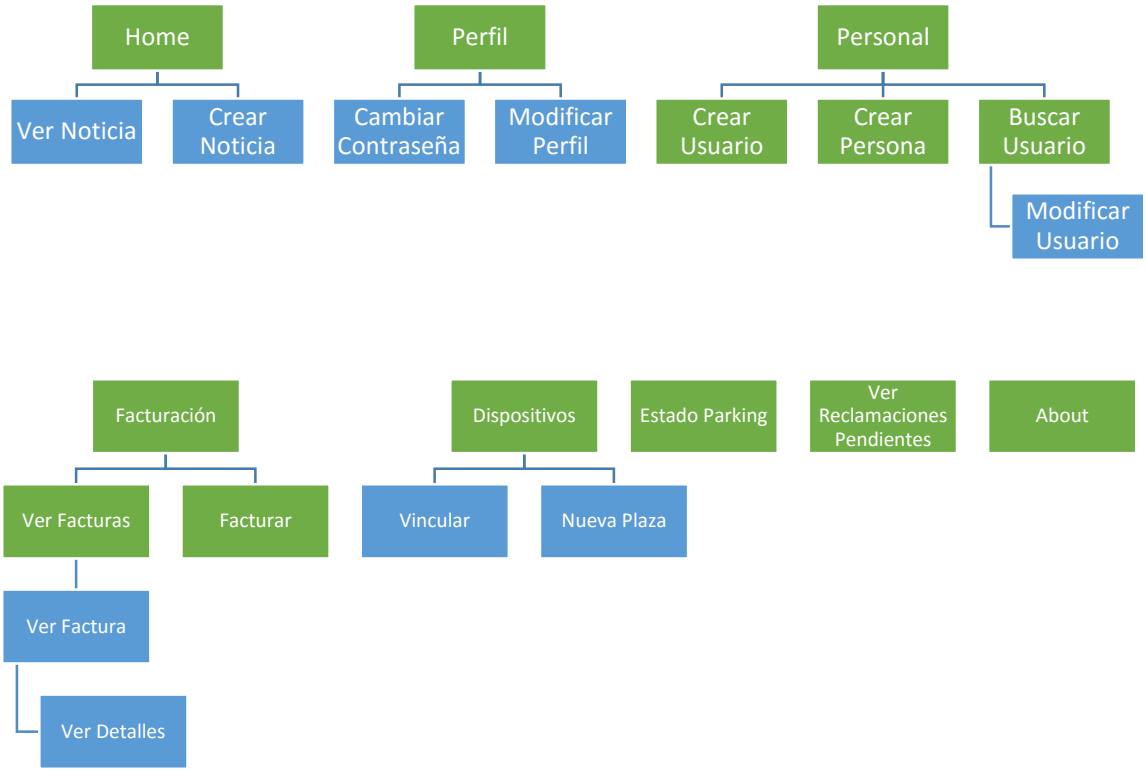
A continuación, se muestra el diagrama de navegación del Cliente, es decir, las páginas a las que podrá acceder desde su propio perfil en la página web.



*Las pestañas marcadas en verde, son pestañas a las cuales, por el esquema de pestañas usado en la implementación, se pueden acceder desde cualquier otra pestaña.

7.1.2 Diagrama de Trabajador

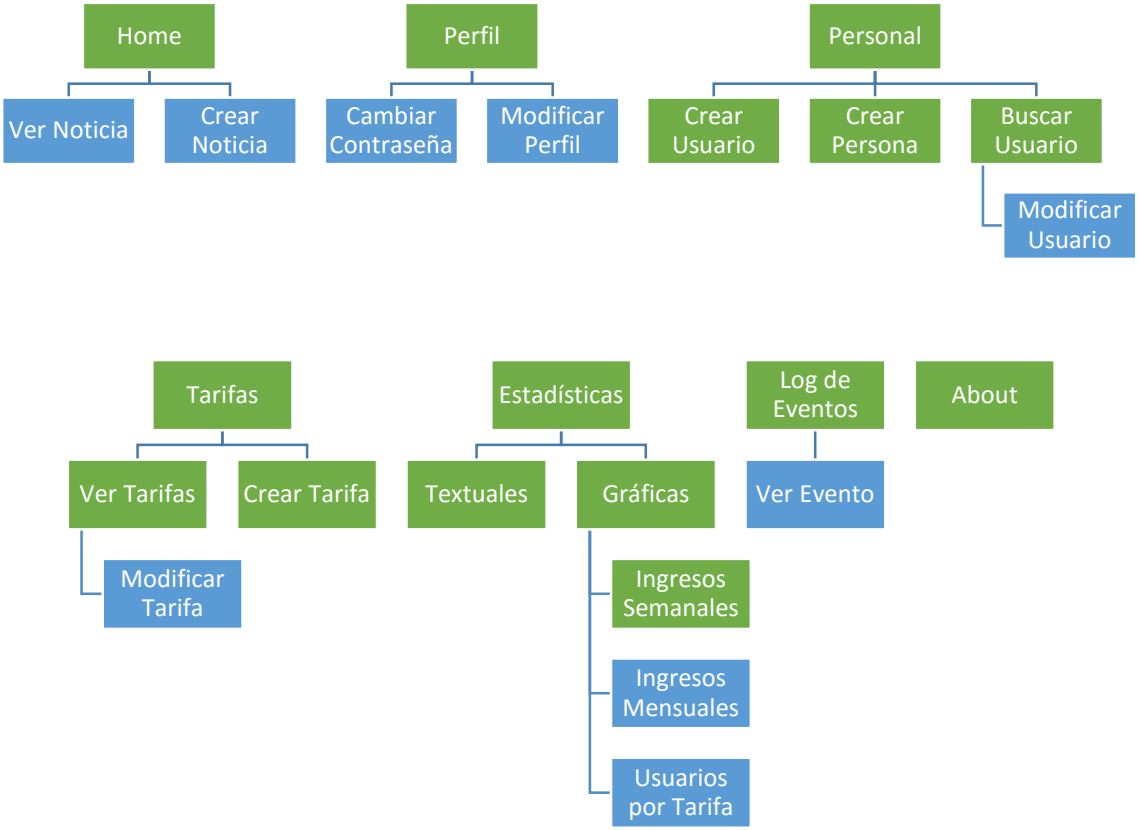
A continuación, se muestra el diagrama de navegación del Trabajador, es decir, las páginas a las que podrá acceder desde su propio perfil en la página web.



*Las pestañas marcadas en verde, son pestañas a las cuales, por el esquema de pestañas usado en la implementación, se pueden acceder desde cualquier otra pestaña.

7.1.3 Diagrama de Administrador

A continuación, se muestra el diagrama de navegación del Administrador, es decir, las páginas a las que podrá acceder desde su propio perfil en la página web.

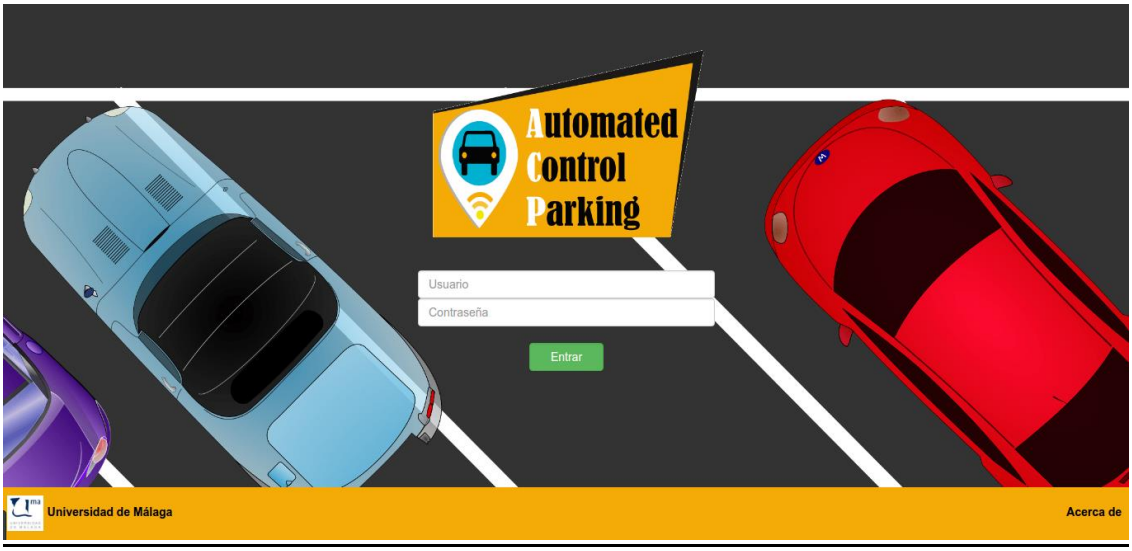


*Las pestañas marcadas en verde, son pestañas a las cuales, por el esquema de pestañas usado en la implementación, se pueden acceder desde cualquier otra pestaña.

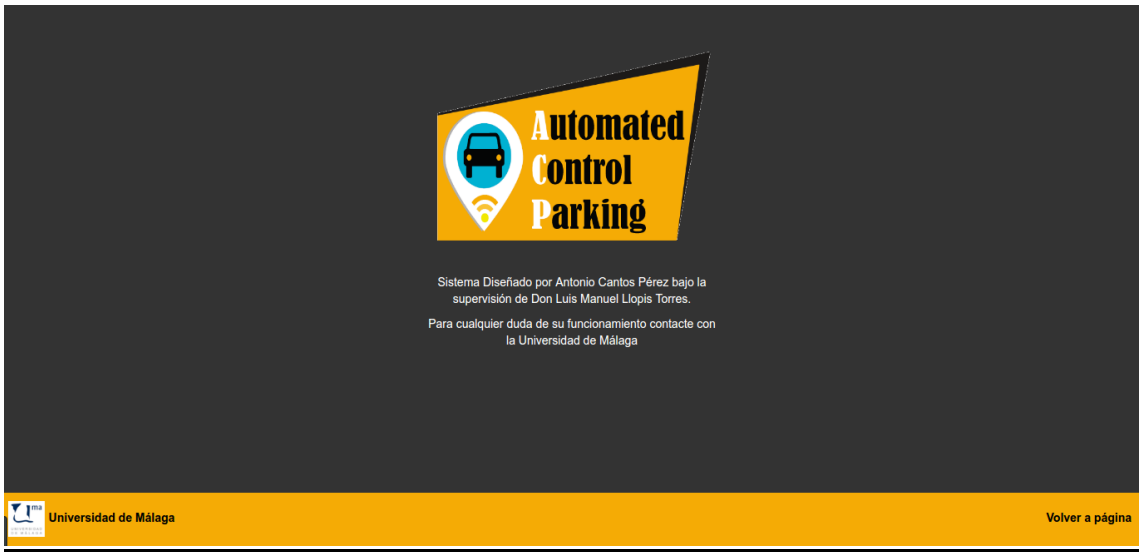
7.2 Capturas de Pantalla

7.2.1 Generales

7.2.1.1 Login

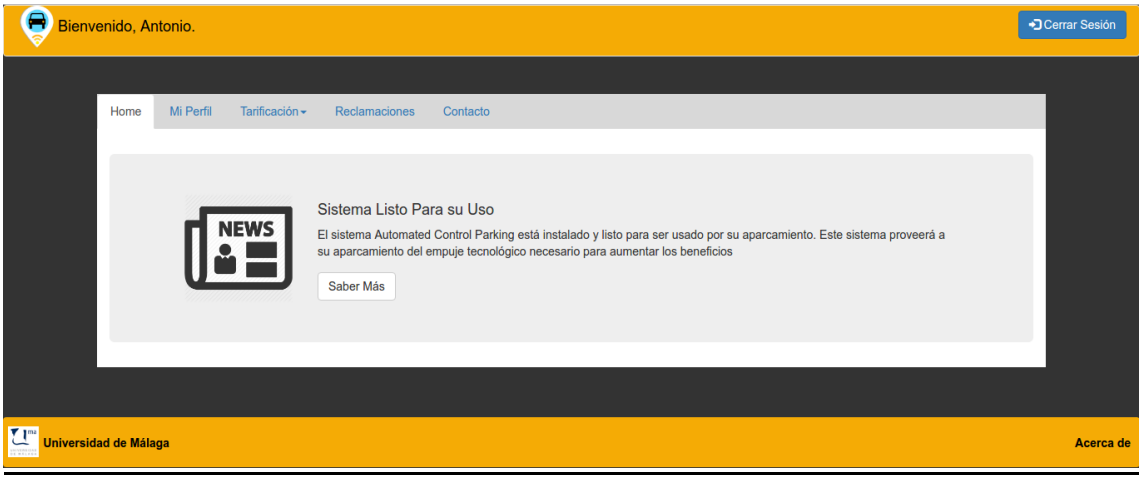


7.2.1.2 About

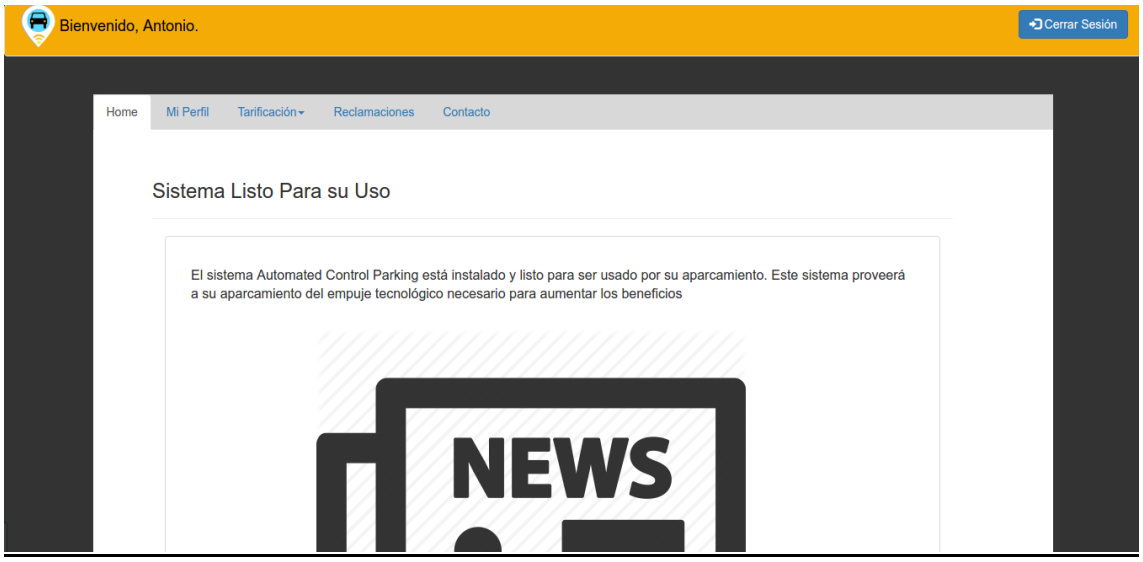


7.2.2 Usuario

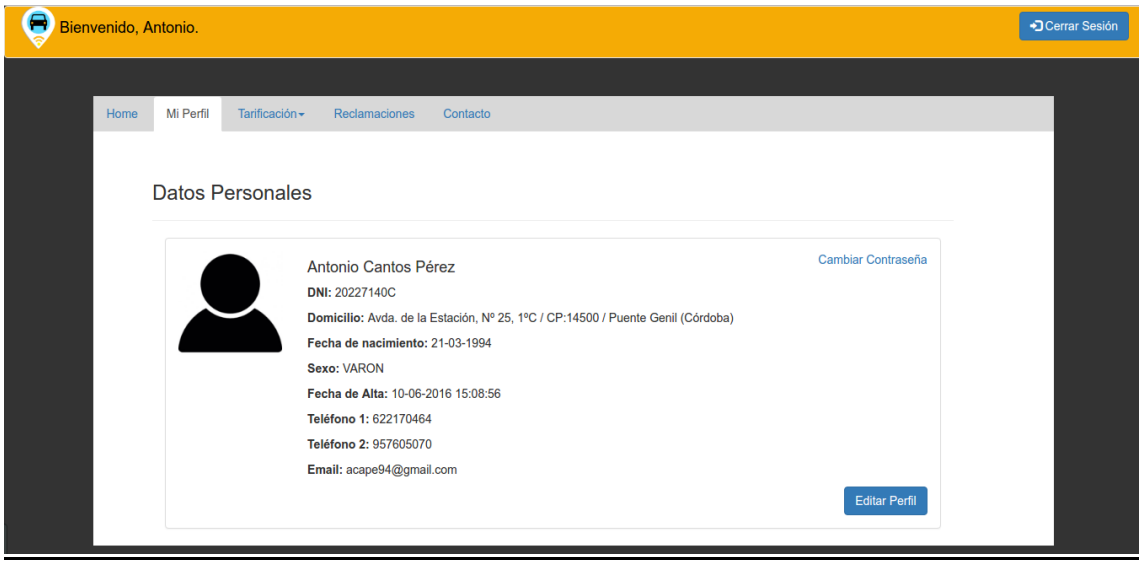
7.2.2.1 Home



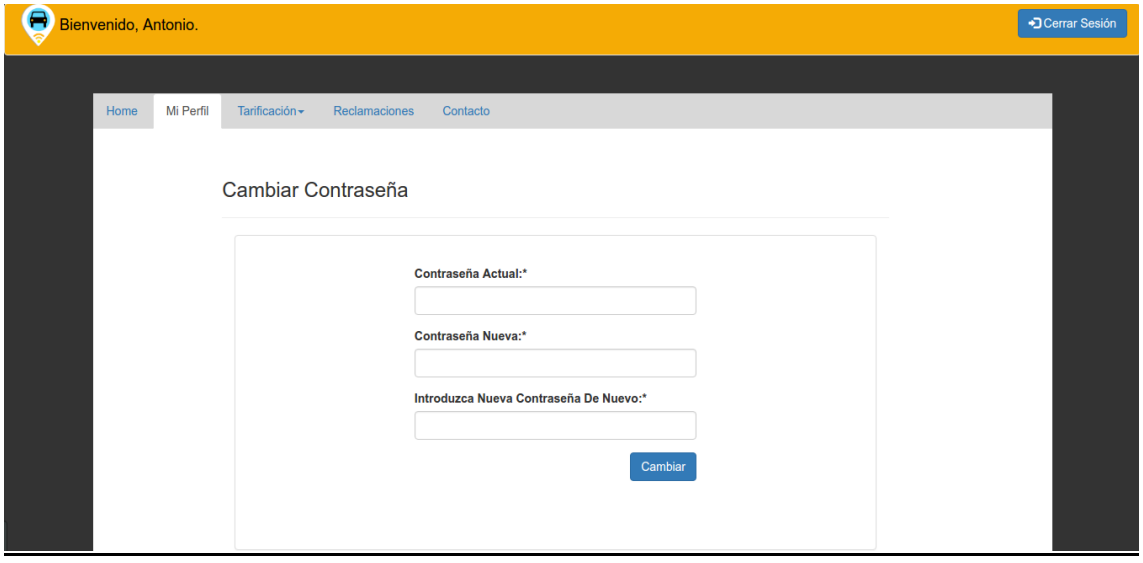
7.2.2.2 Ver Noticia



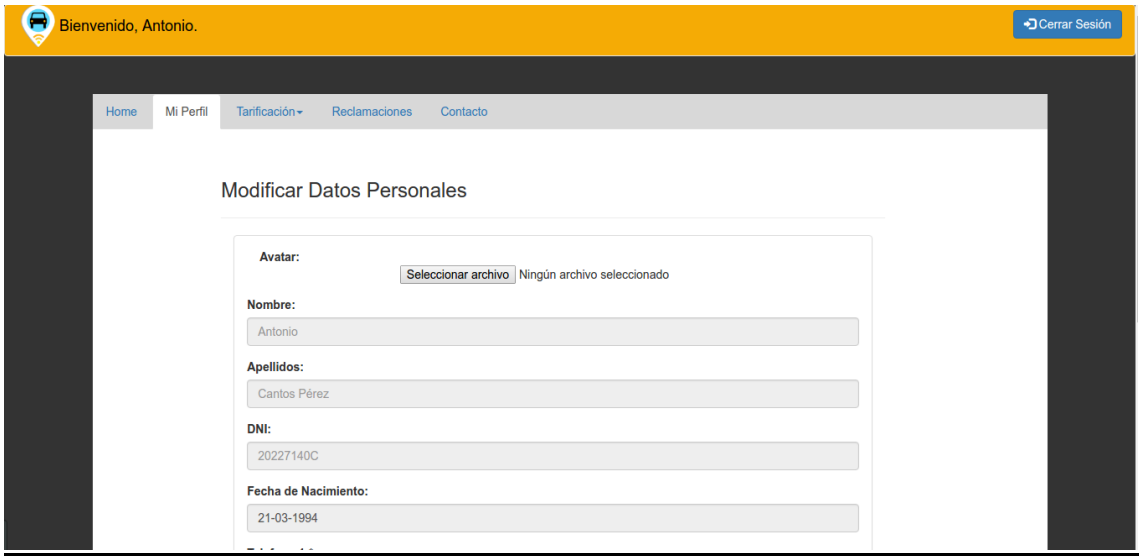
7.2.2.3 Ver Perfil



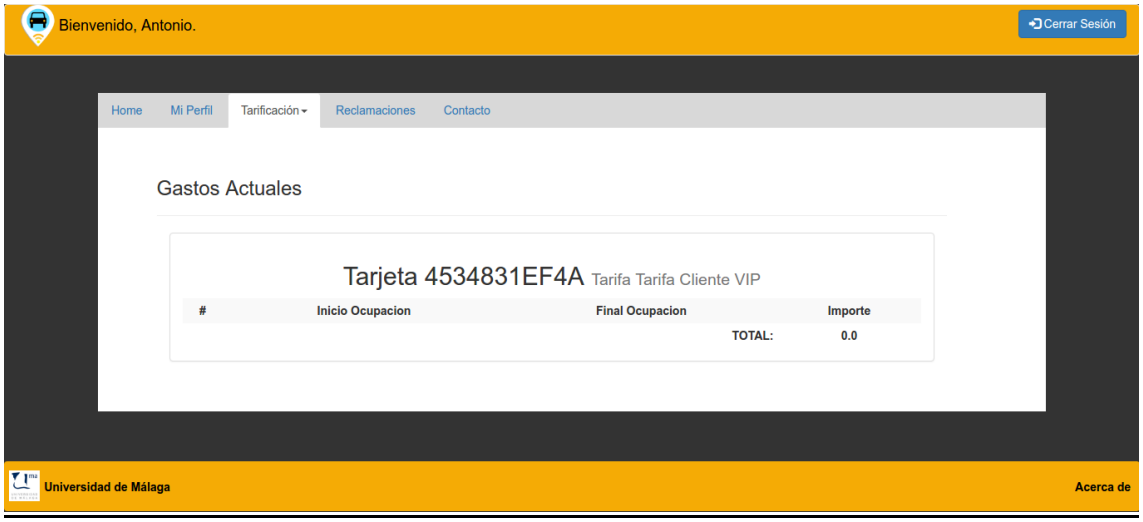
7.2.2.4 Cambiar Contraseña



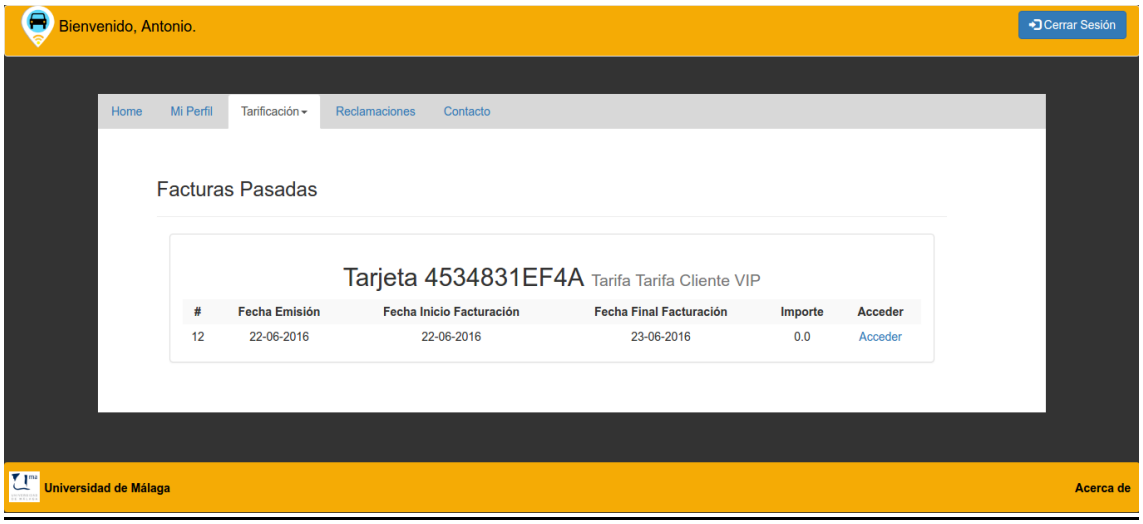
7.2.2.5 Modificar Perfil



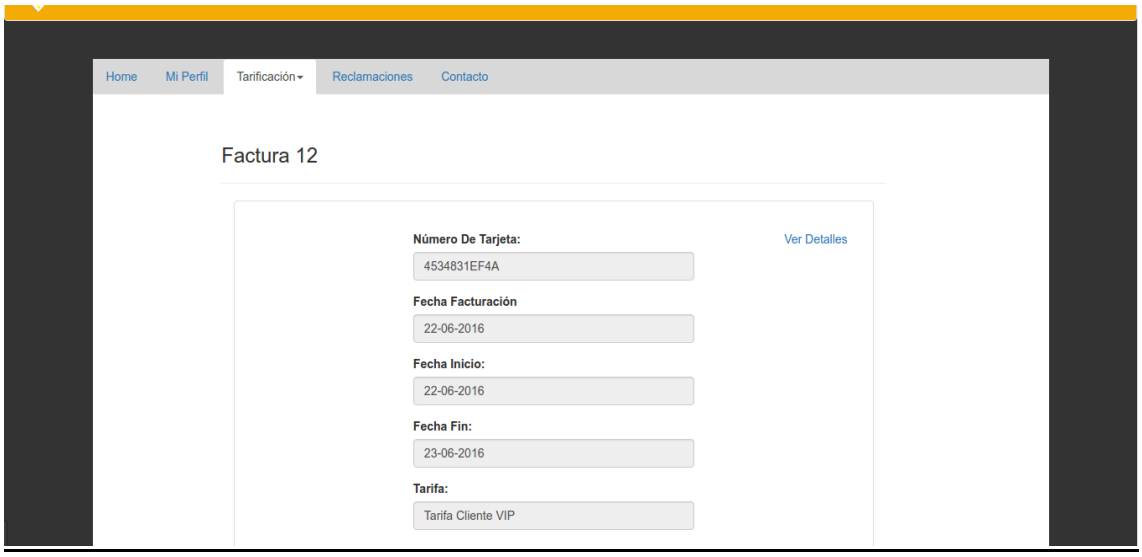
7.2.2.6 Gastos Actuales



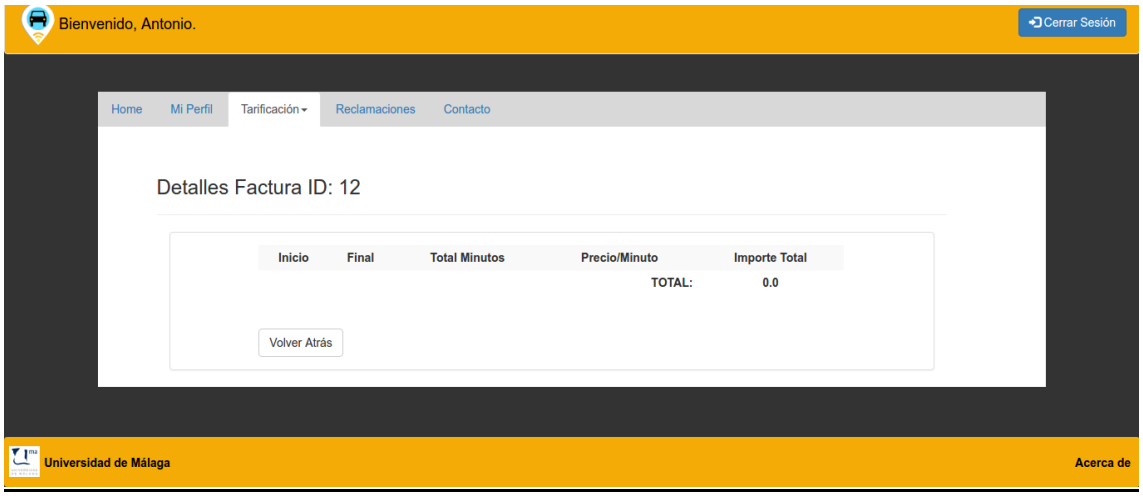
7.2.2.7 Facturas Pasadas



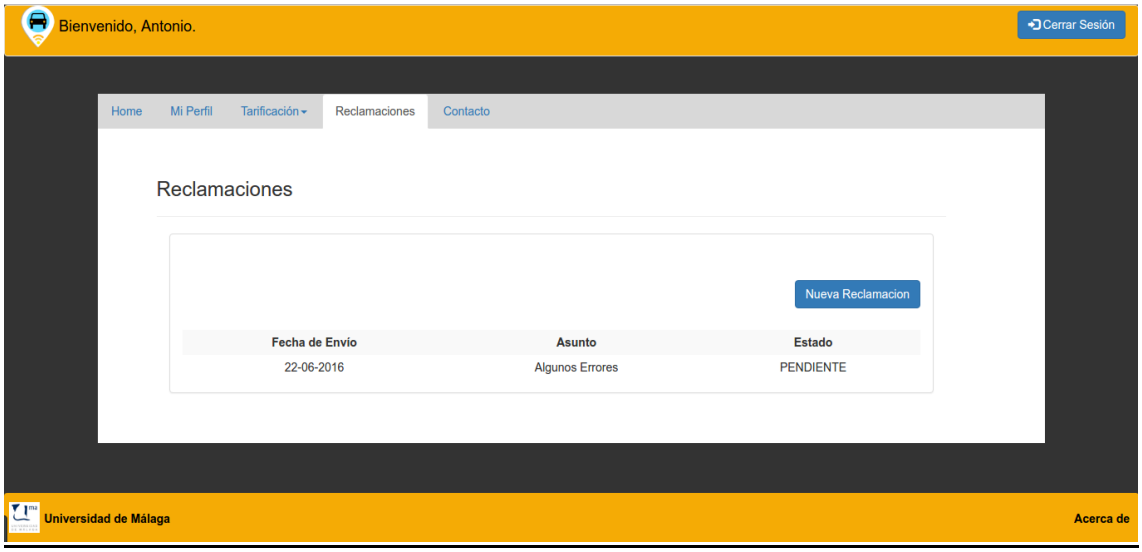
7.2.2.8 Ver Factura



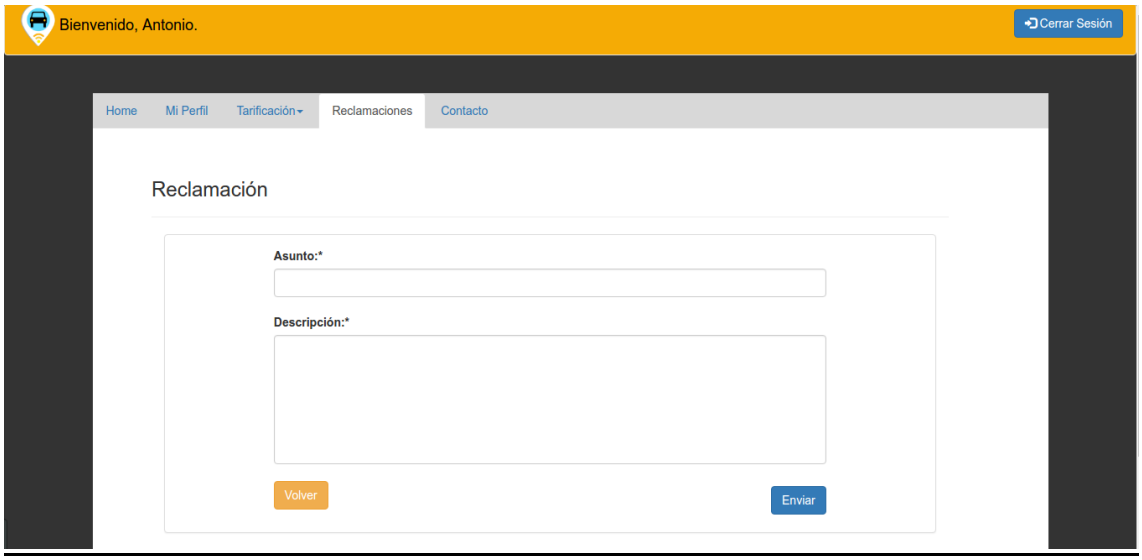
7.2.2.9 Detalles Factura



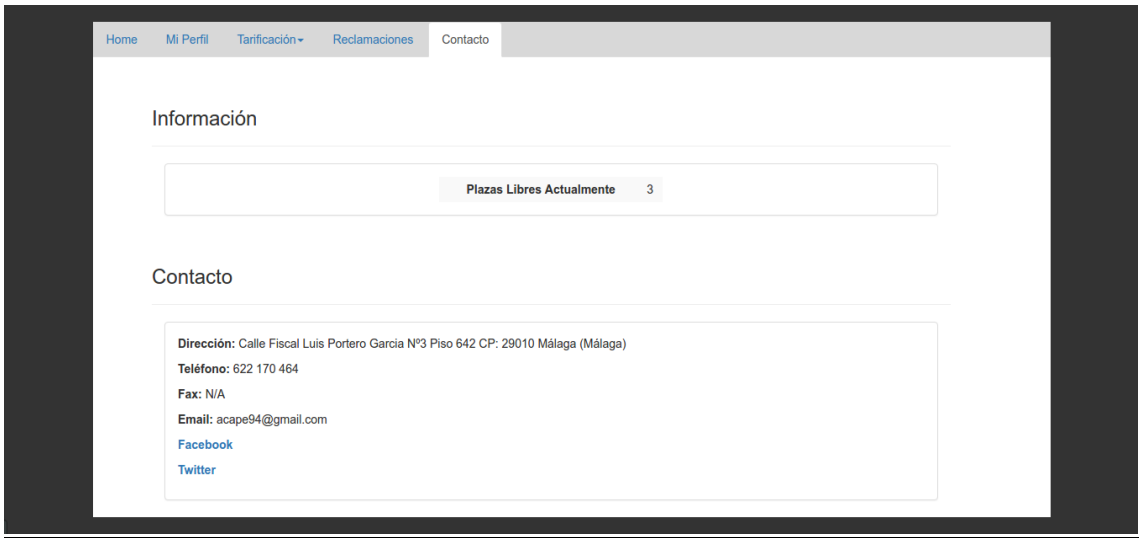
7.2.2.10 Ver Reclamaciones



7.2.2.11 Nueva Reclamación

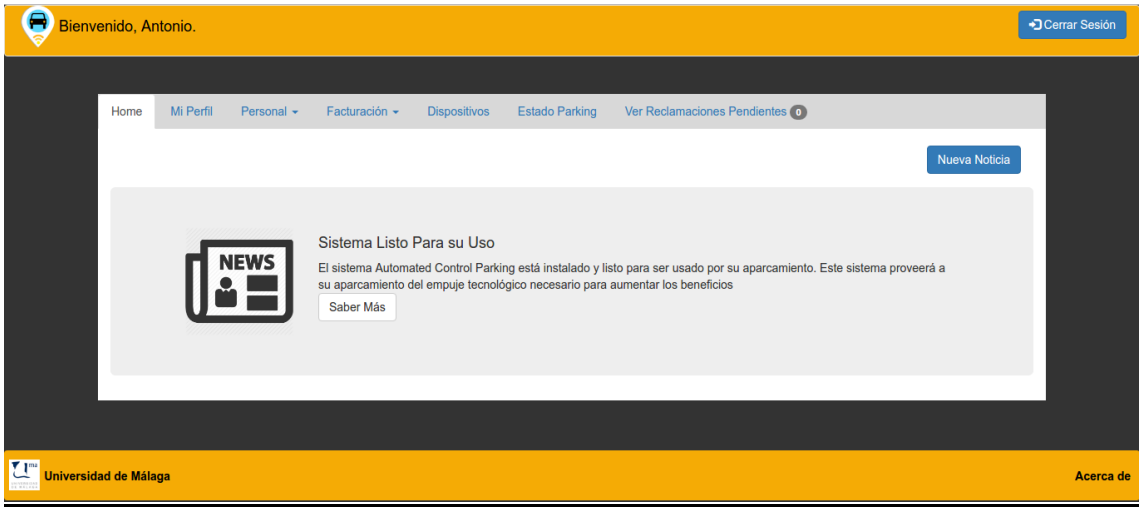


7.2.2.12 Contacto



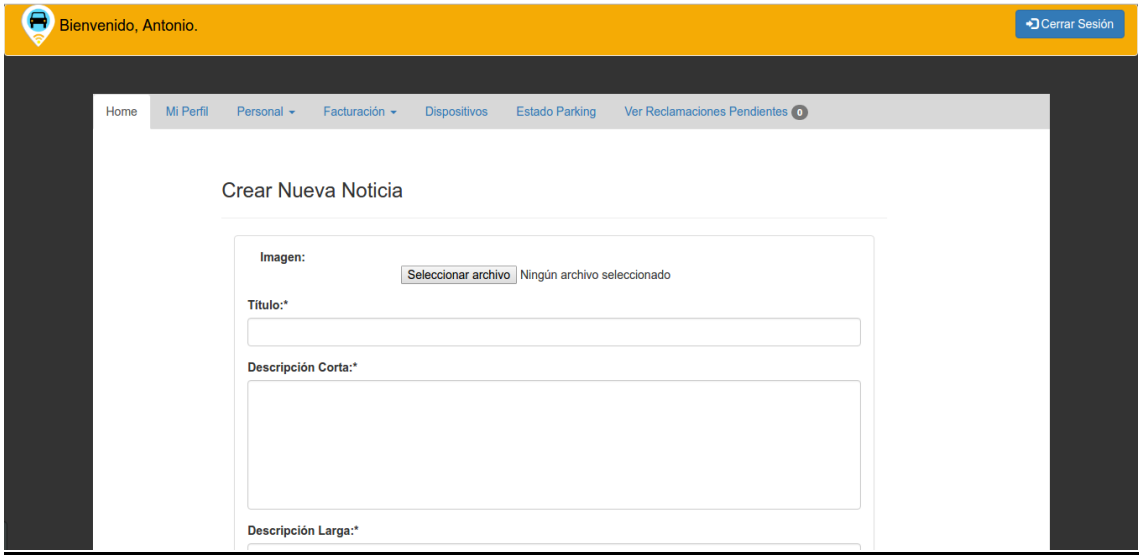
7.2.3 Trabajador

7.2.3.1 Home

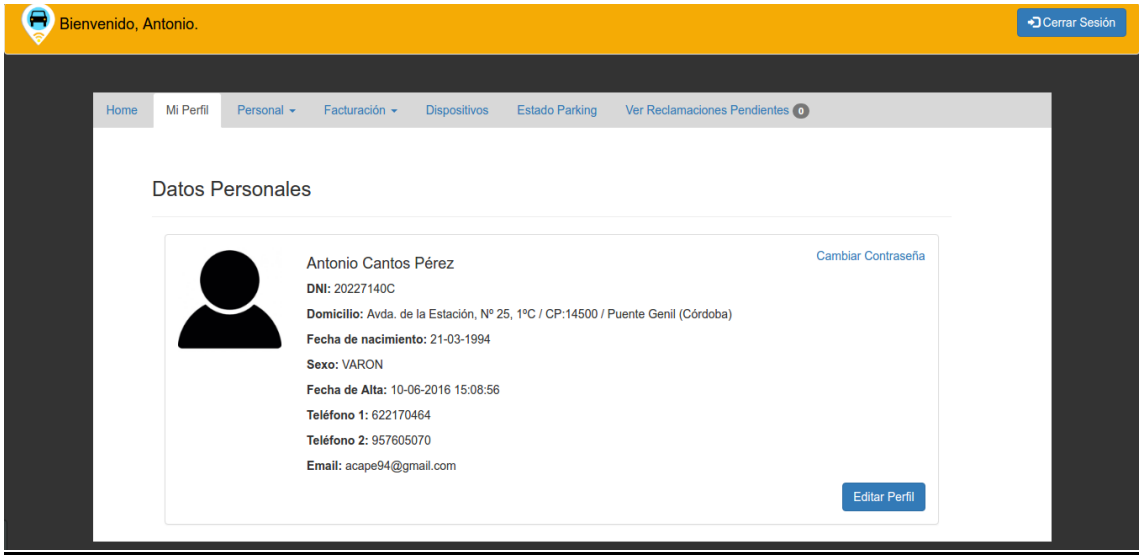


7.2.3.2 Ver Noticia

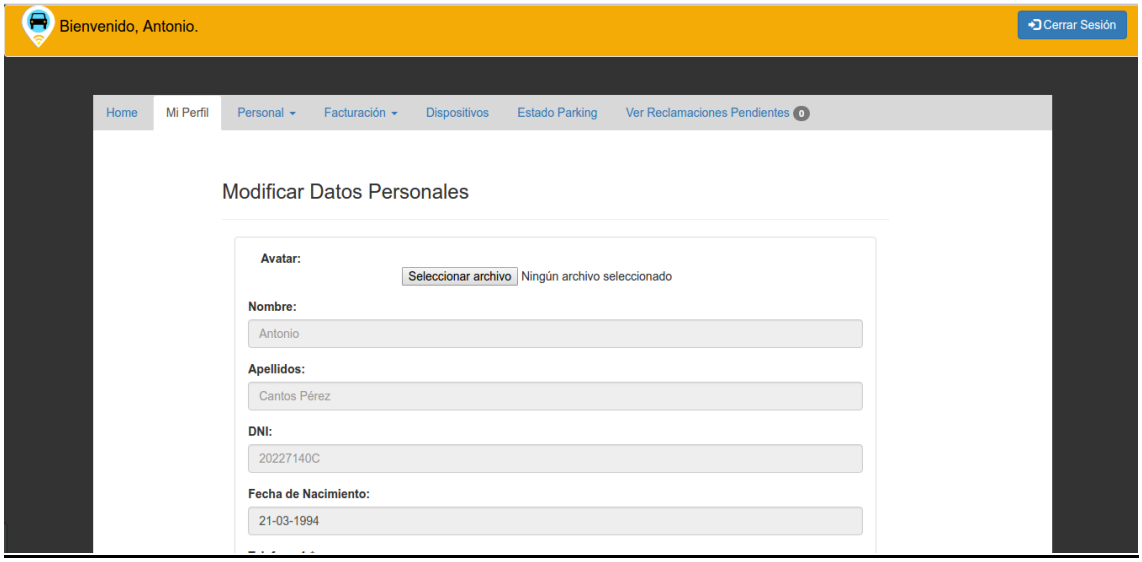
7.2.3.3 Nueva Noticia



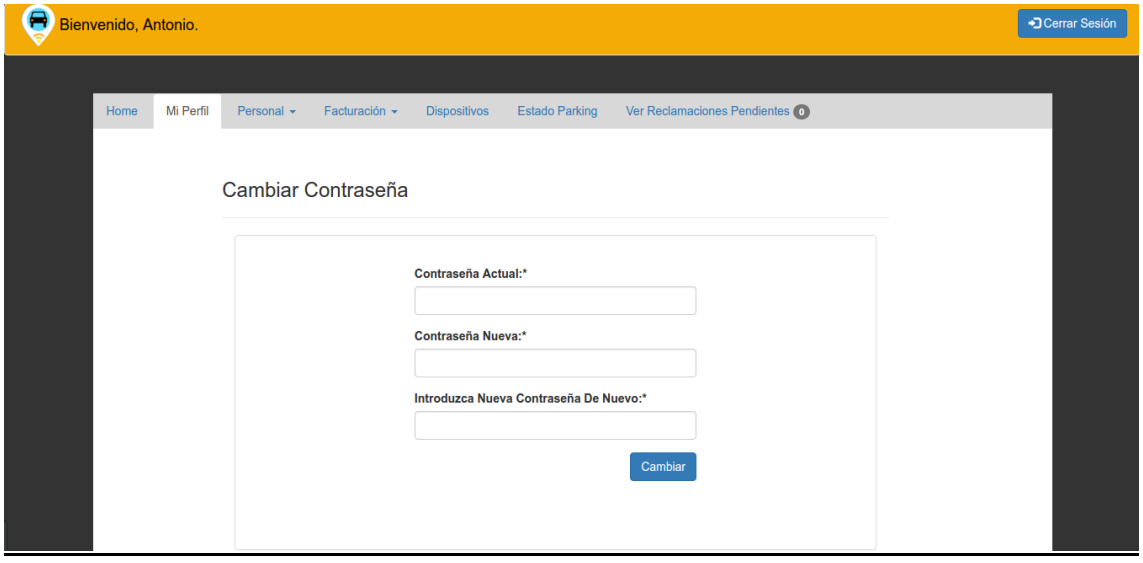
7.2.3.4 Ver Perfil



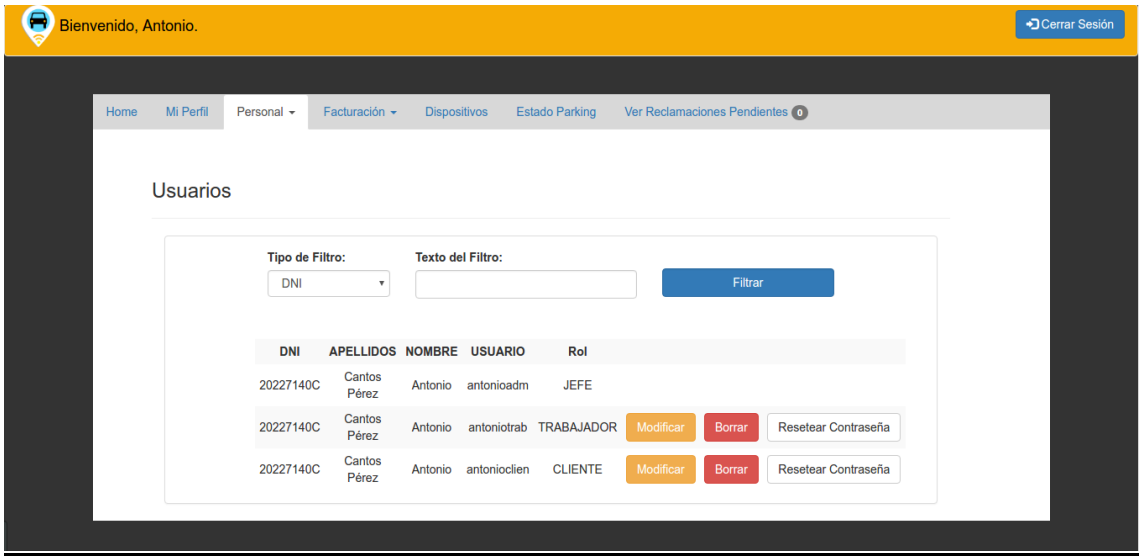
7.2.3.5 Modificar Perfil



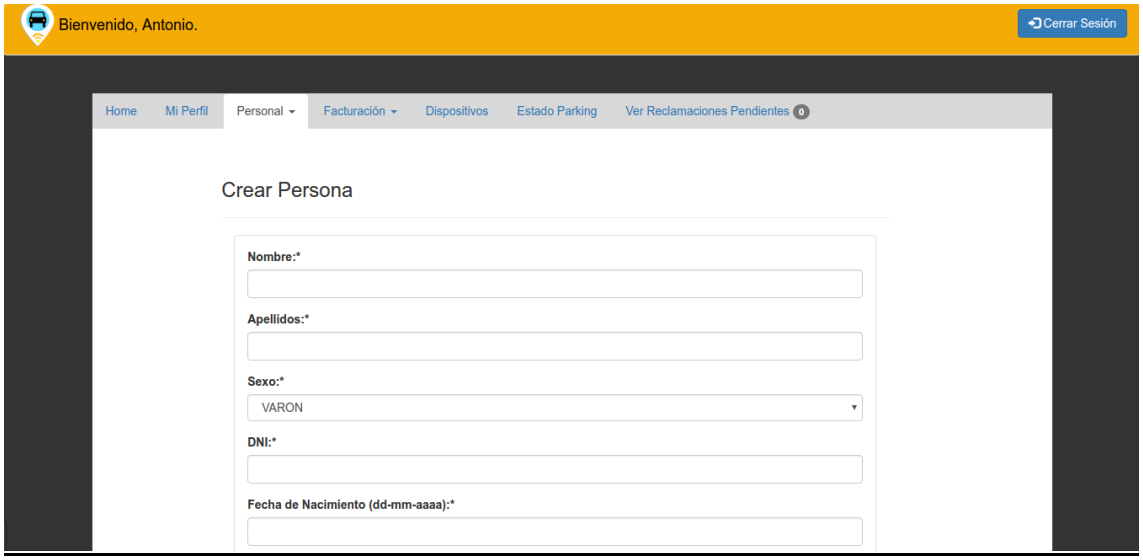
7.2.3.6 Cambiar Contraseña



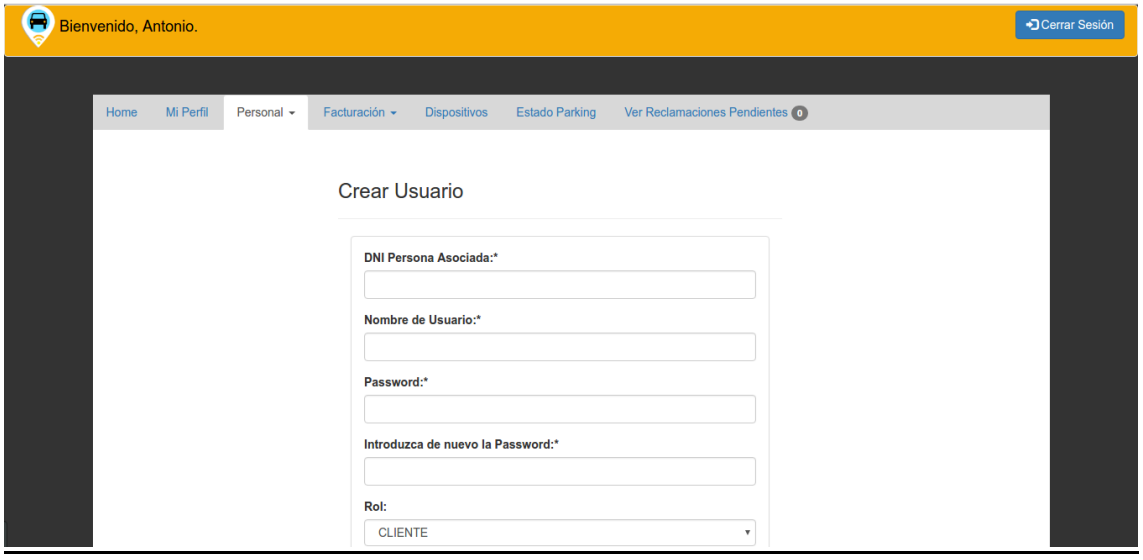
7.2.3.7 Buscar Usuario



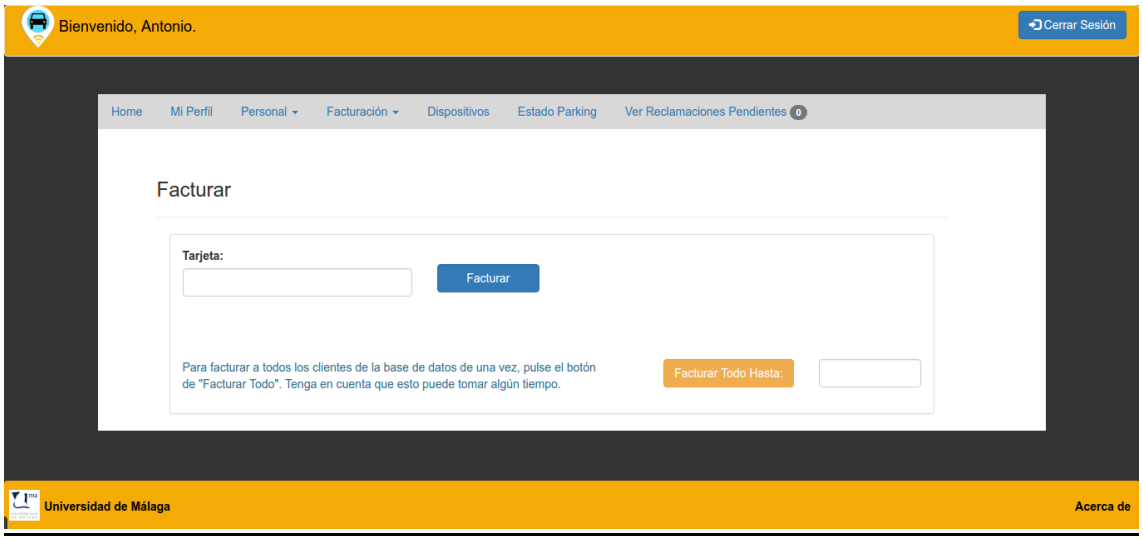
7.2.3.8 Crear Persona



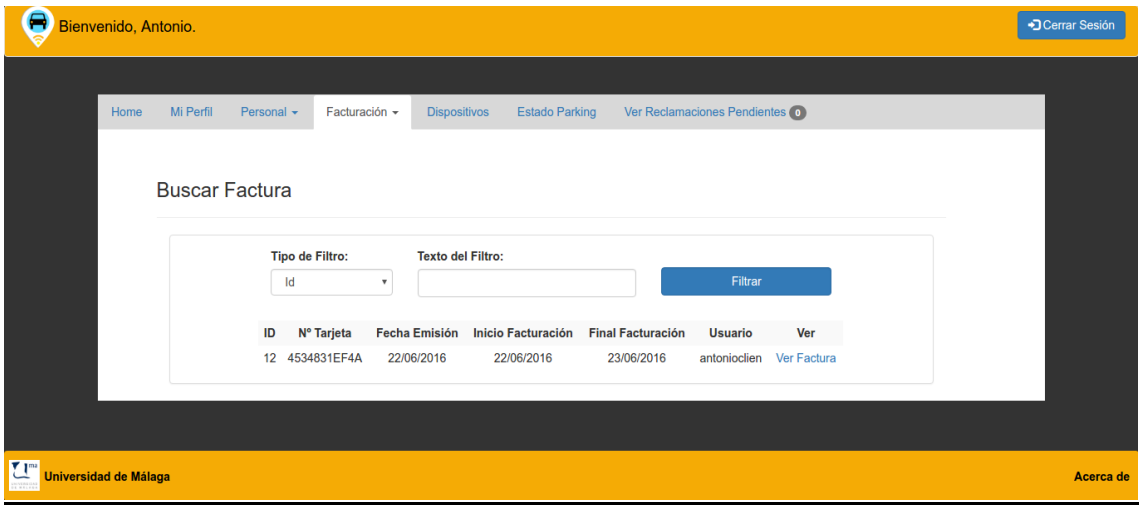
7.2.3.9 Crear Usuario

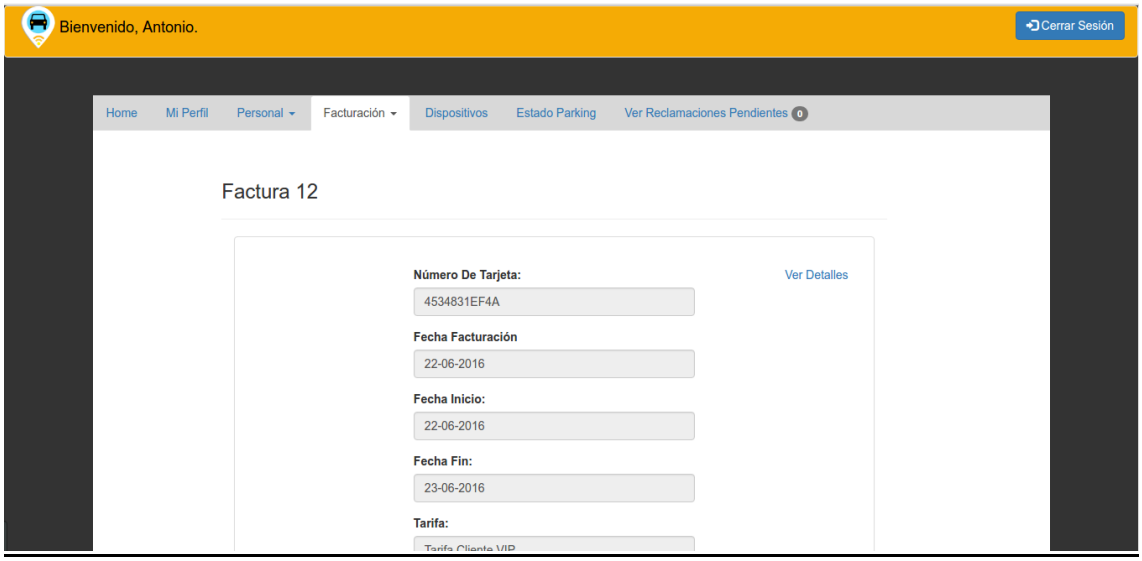


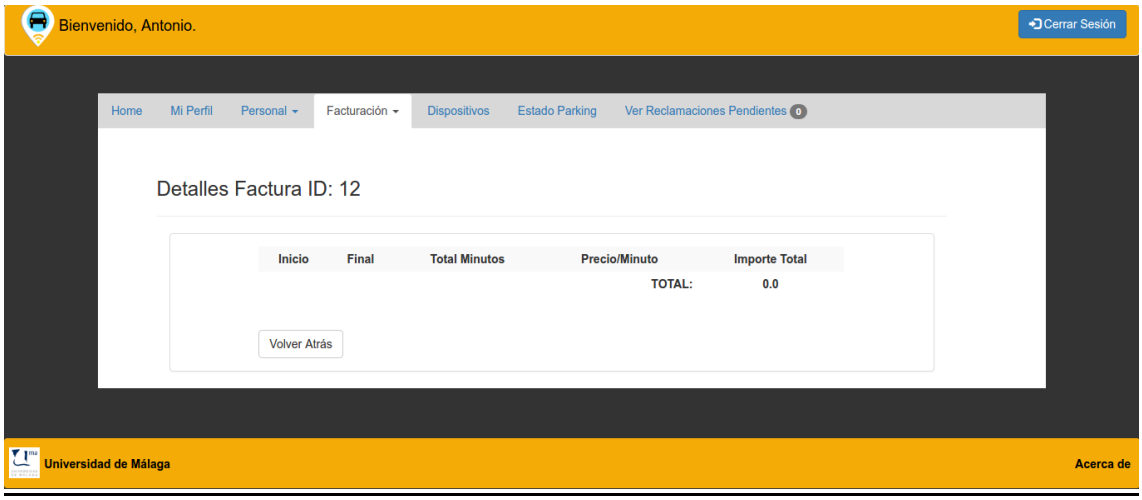
7.2.3.10 Facturar



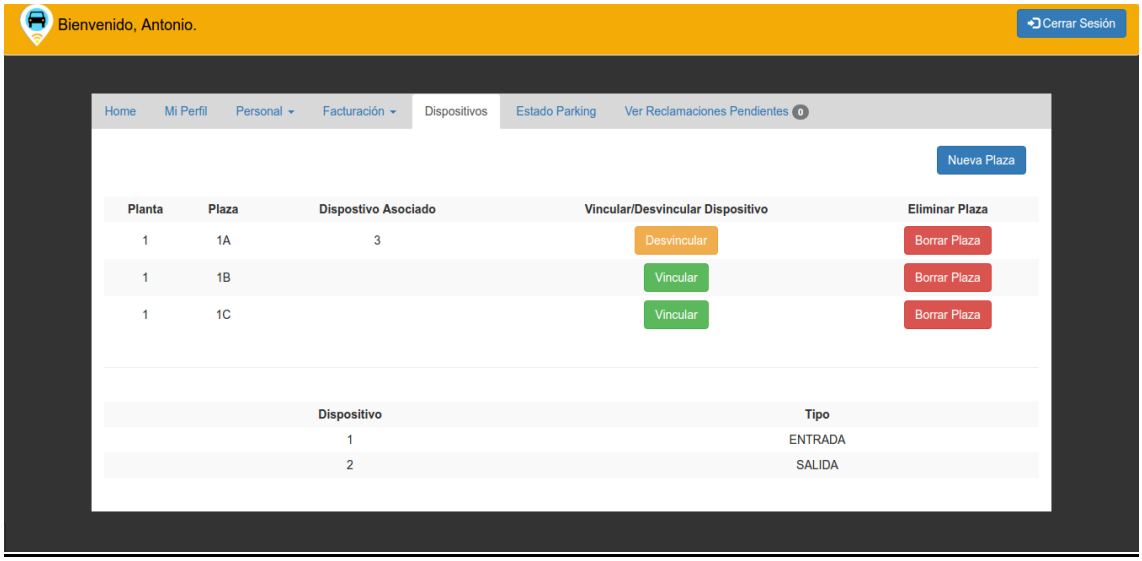
7.2.3.11 Ver Facturas



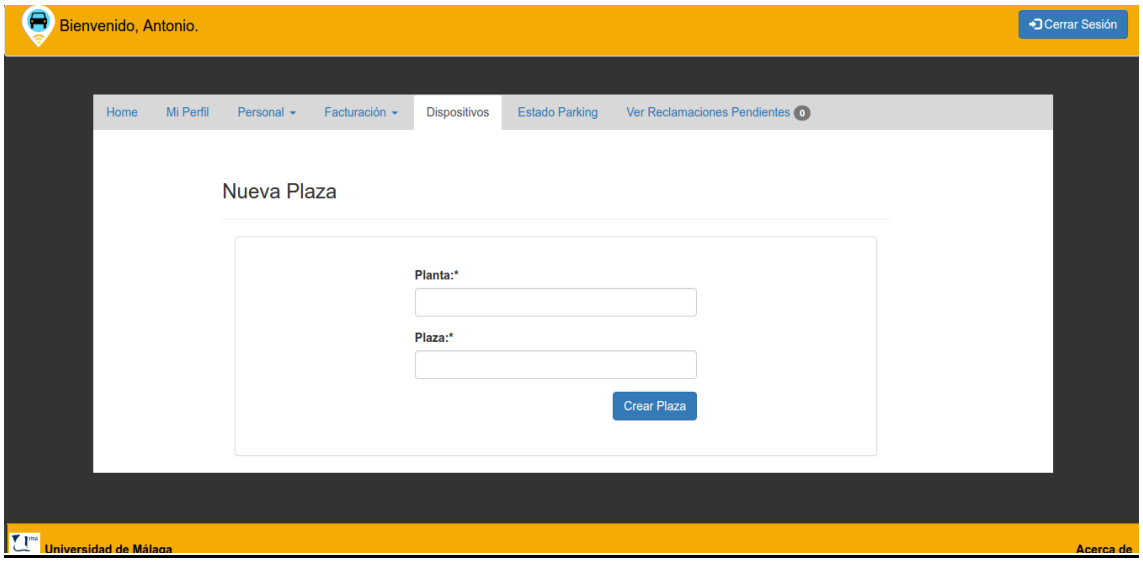




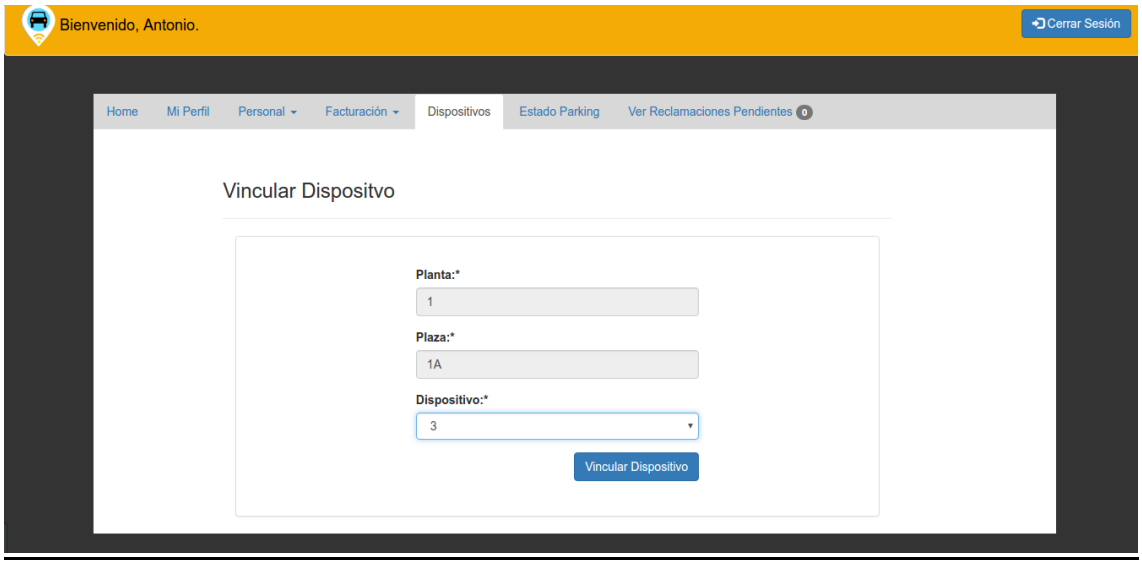
7.2.3.12 Dispositivos



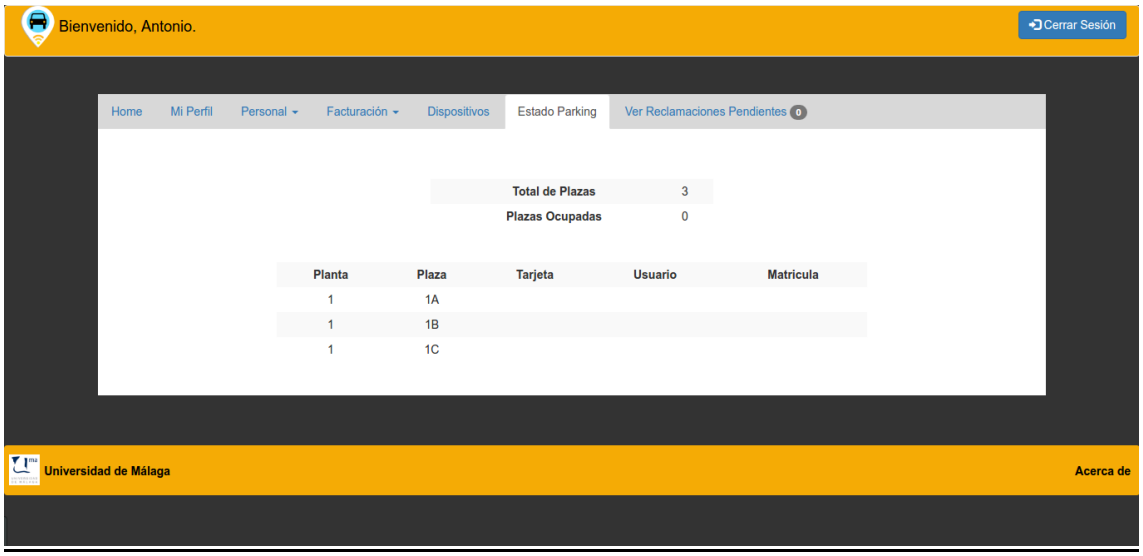
7.2.3.13 Nueva Plaza



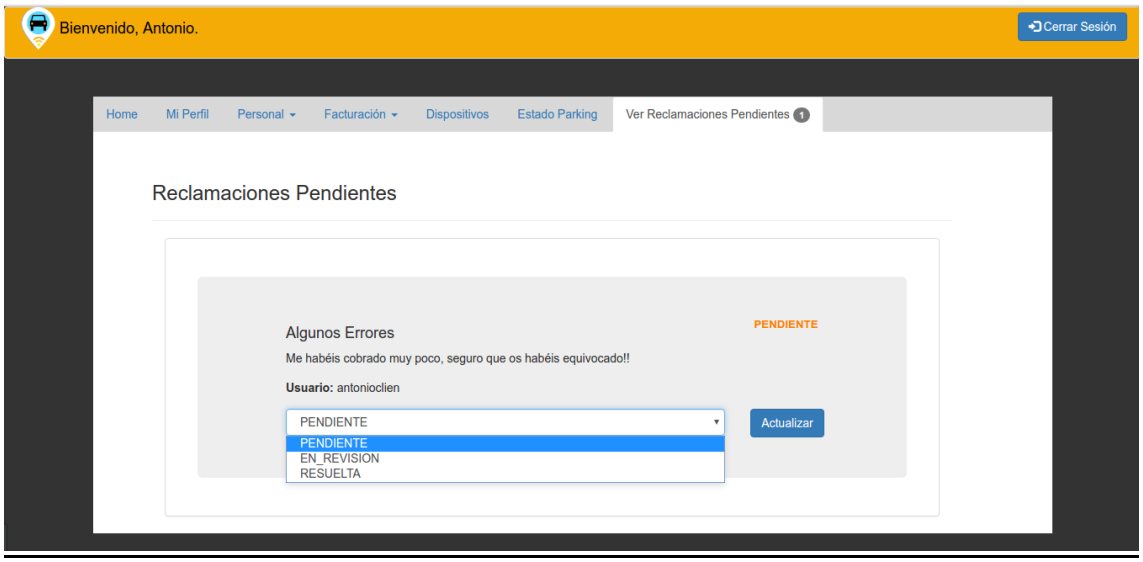
7.2.3.14 Vincular Plaza



7.2.3.15 Estado Parking



7.2.3.16 Reclamaciones Pendientes

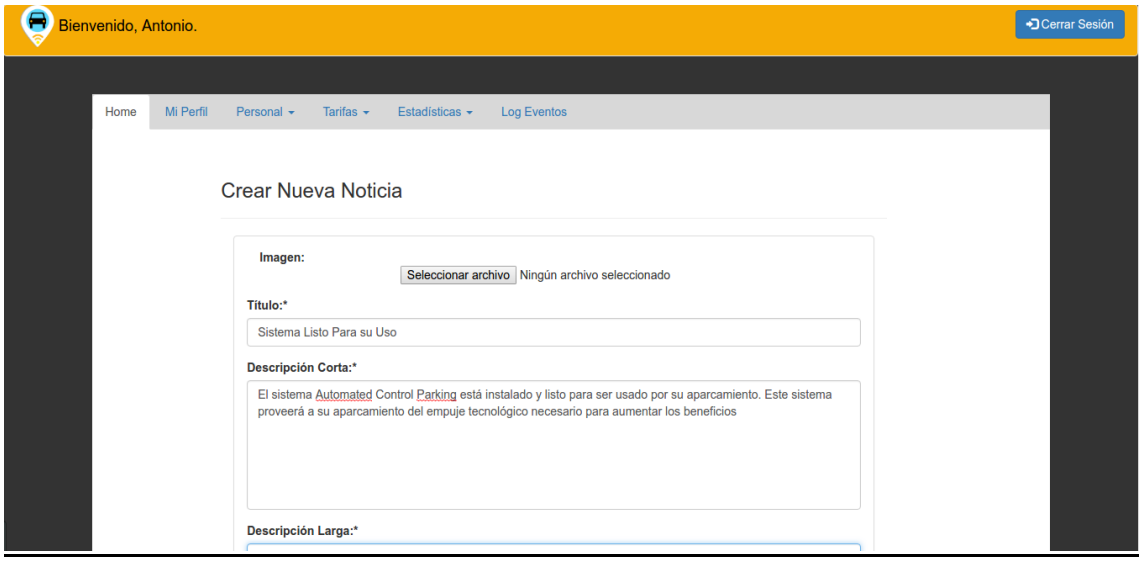


7.2.4 Administrador

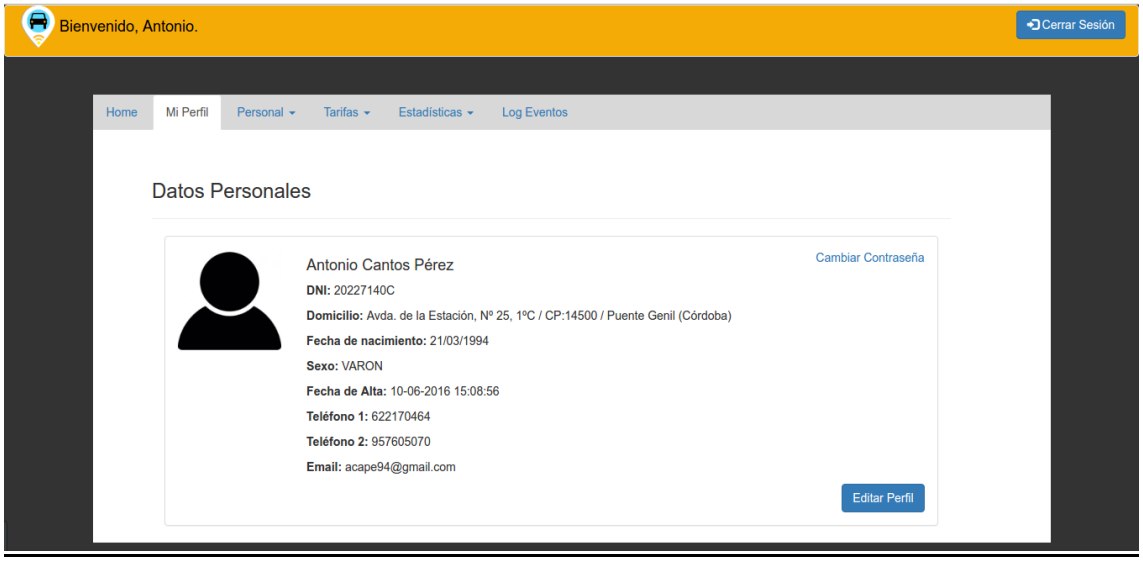
7.2.4.1 Home

7.2.4.2 Ver Noticia

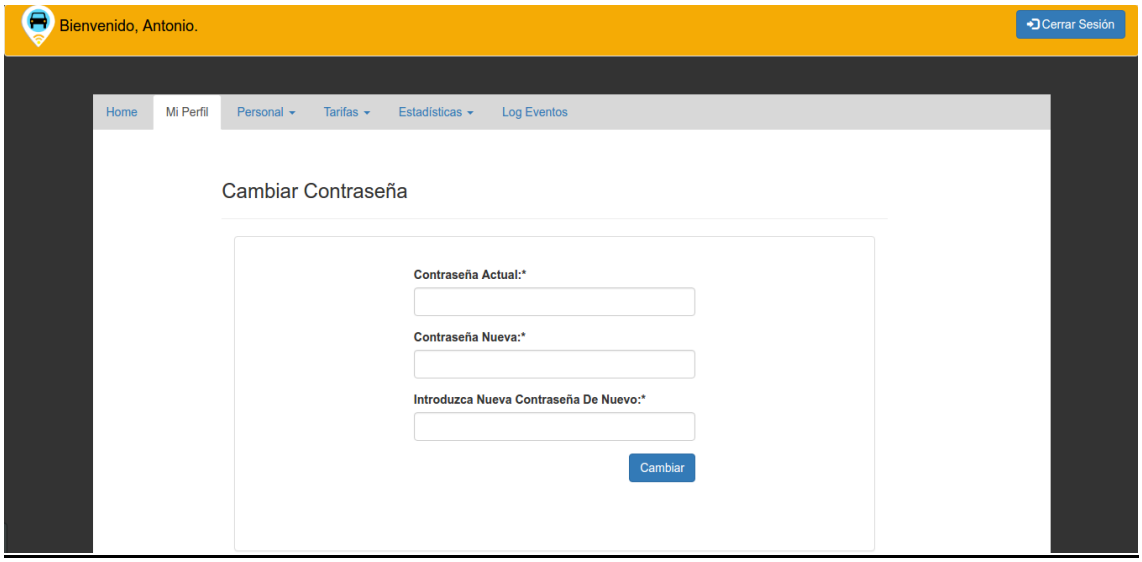
7.2.4.3 Nueva Noticia



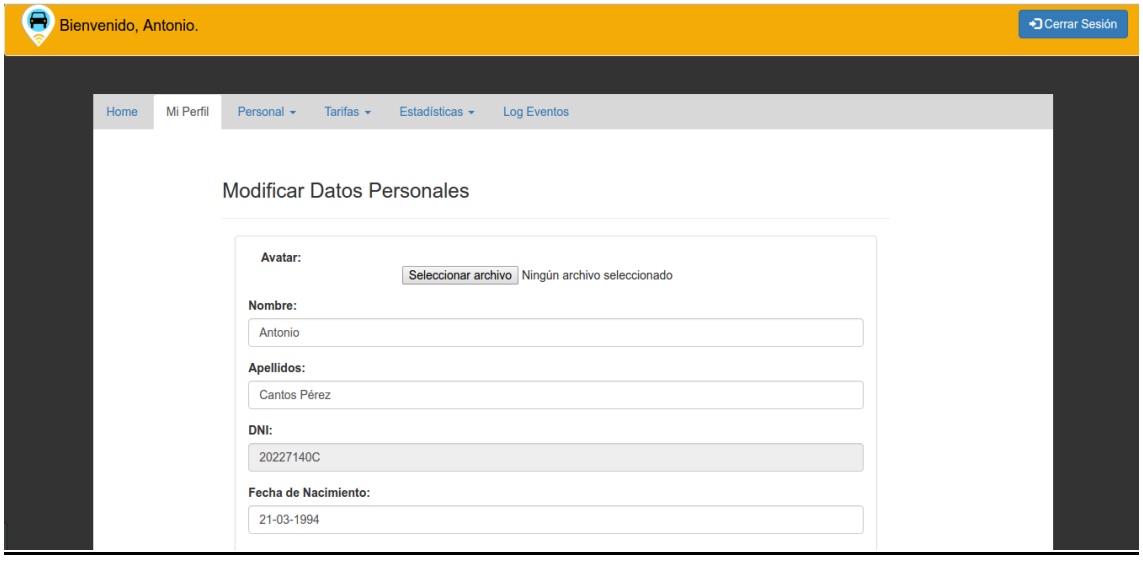
7.2.4.4 Perfil



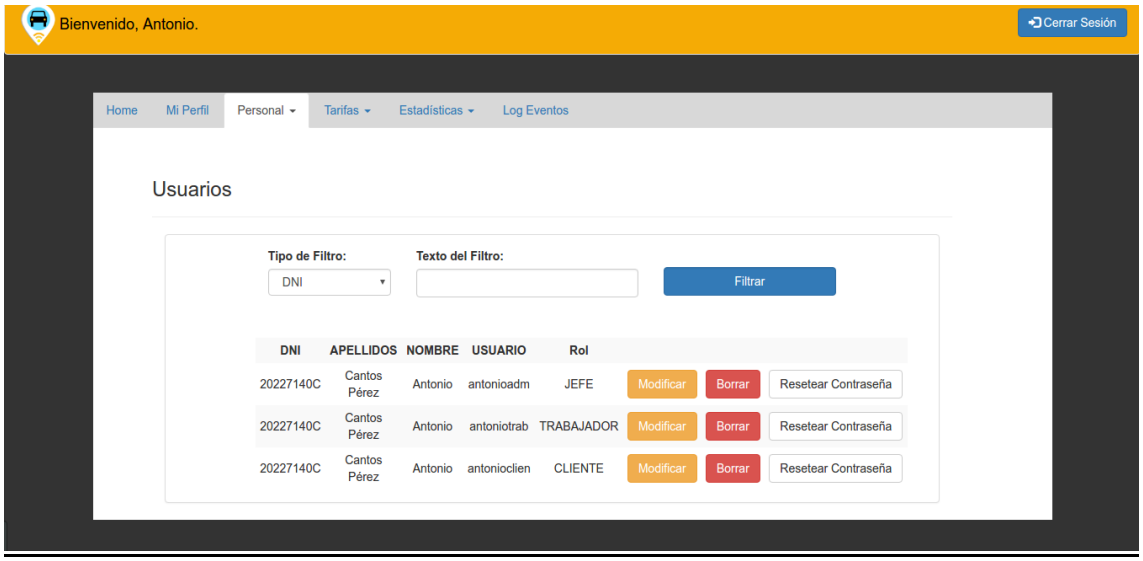
7.2.4.5 Cambiar Contraseña



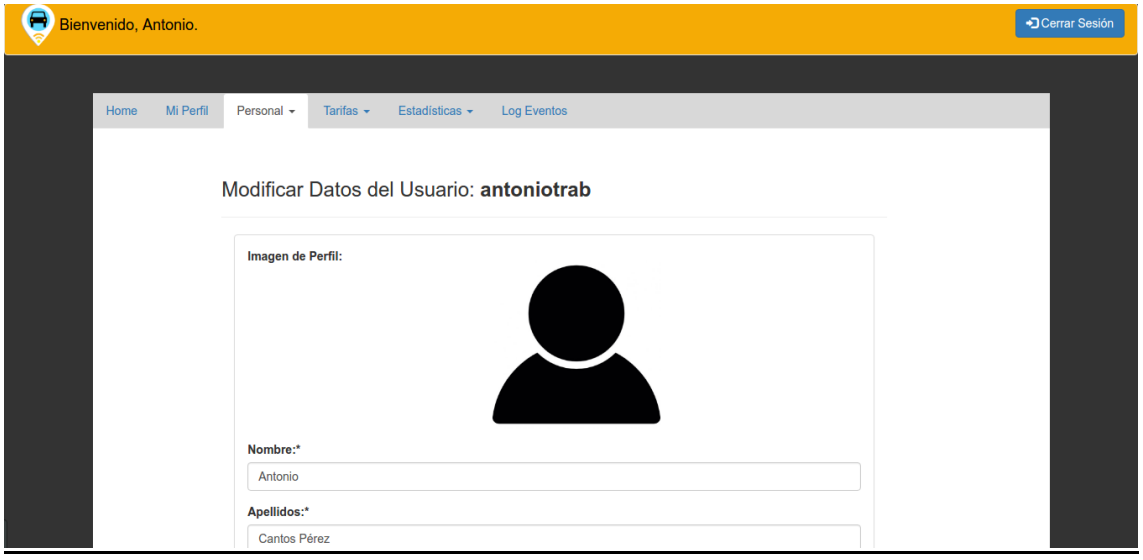
7.2.4.6 Modificar Perfil



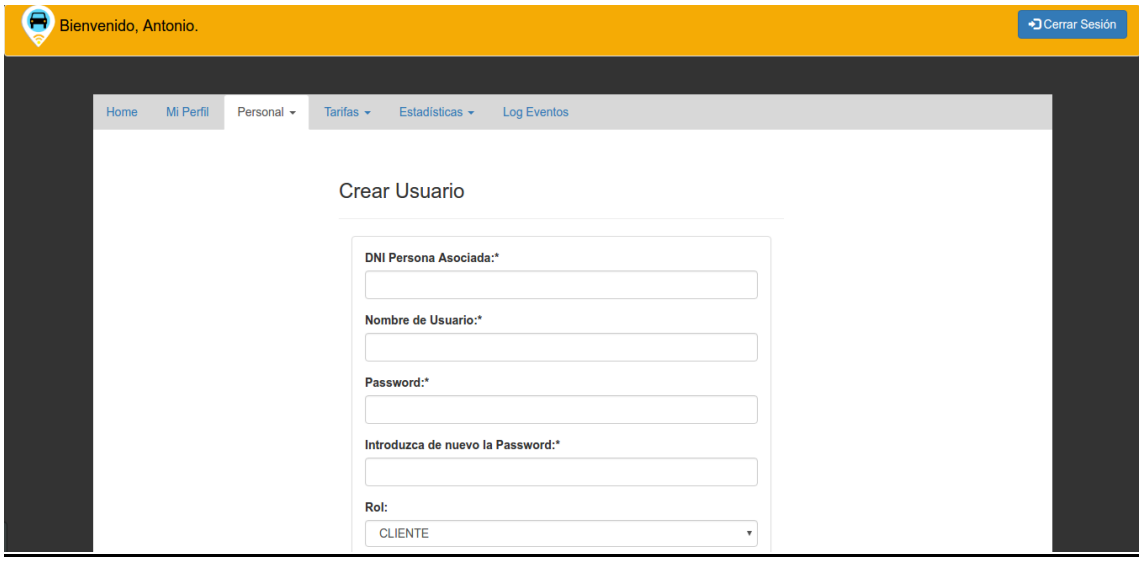
7.2.4.7 Ver Usuarios



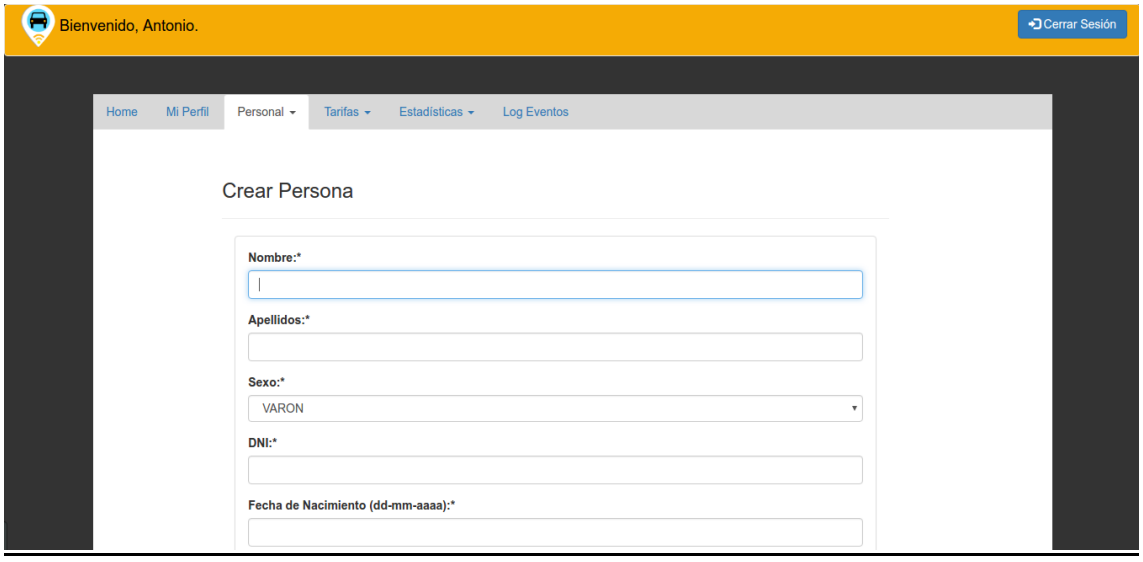
7.2.4.8 Modificar Usuario



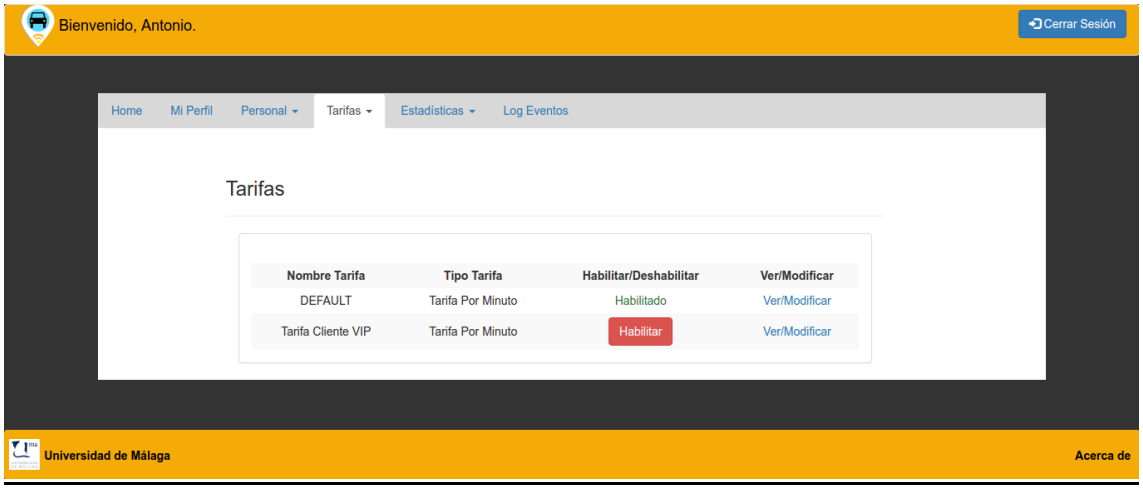
7.2.4.9 Crear Usuario



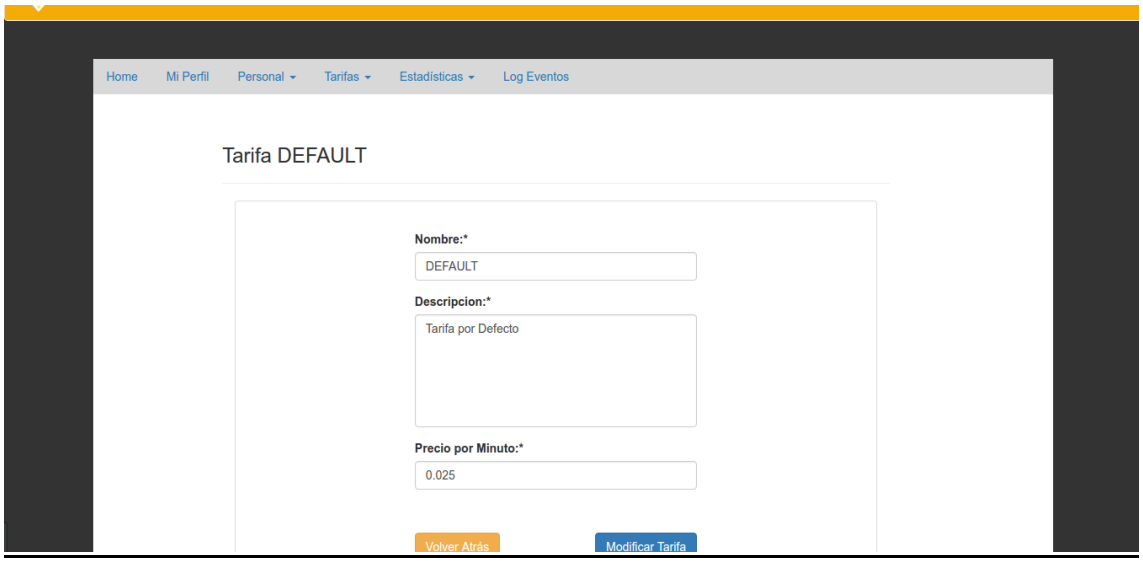
7.2.4.10 Crear Persona



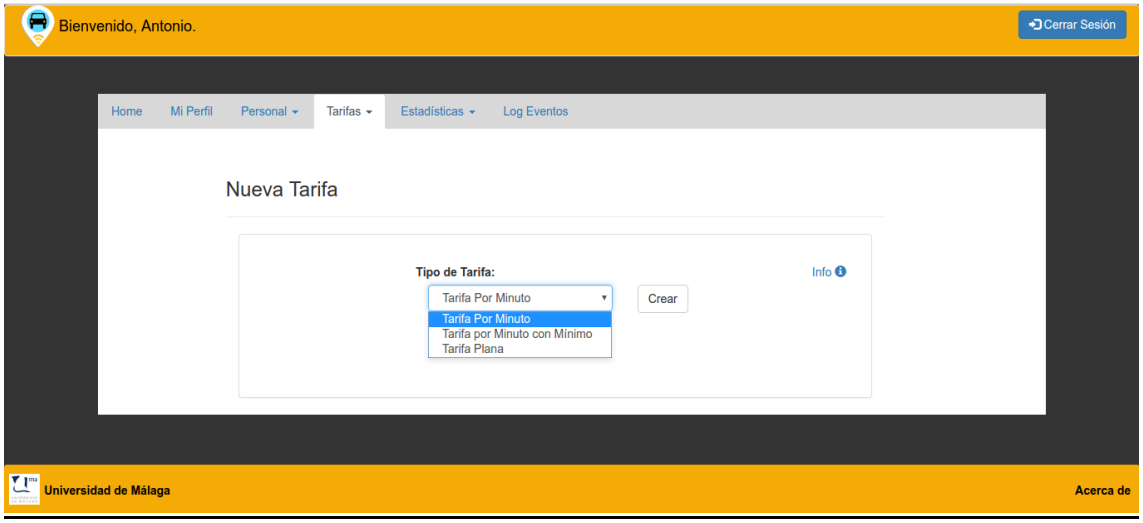
7.2.4.11 Ver Tarifas



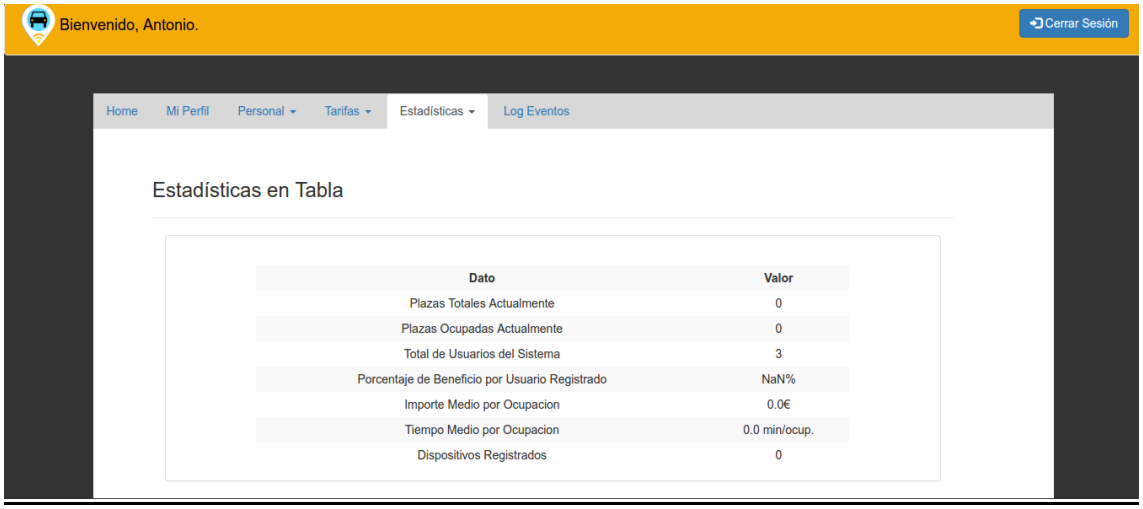
7.2.4.12 Modificar Tarifa



7.2.4.13 Crear Tarifa

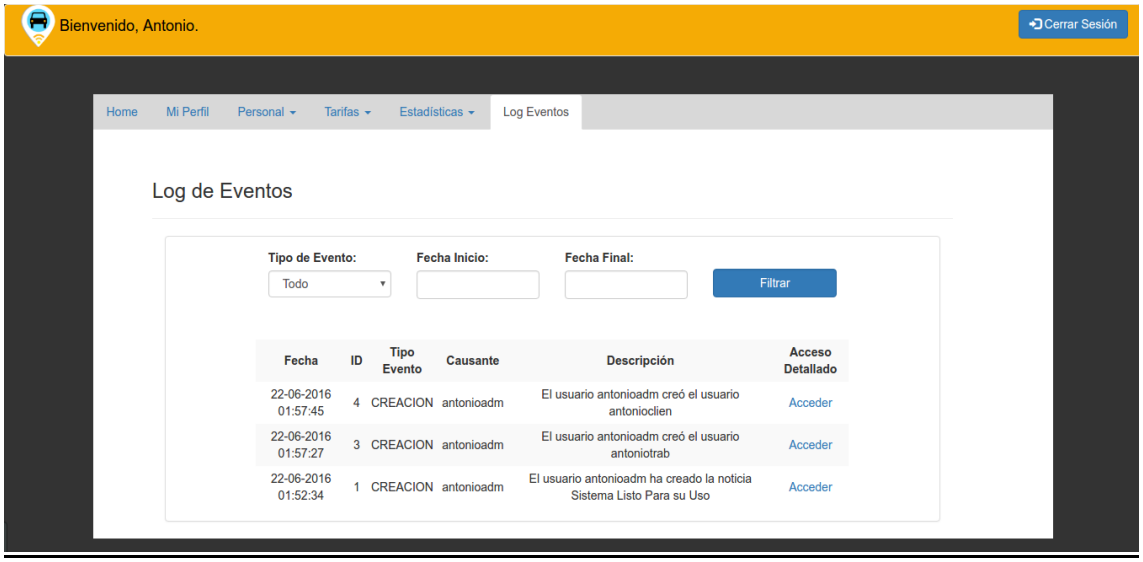


7.2.4.14 Estadísticas Textuales

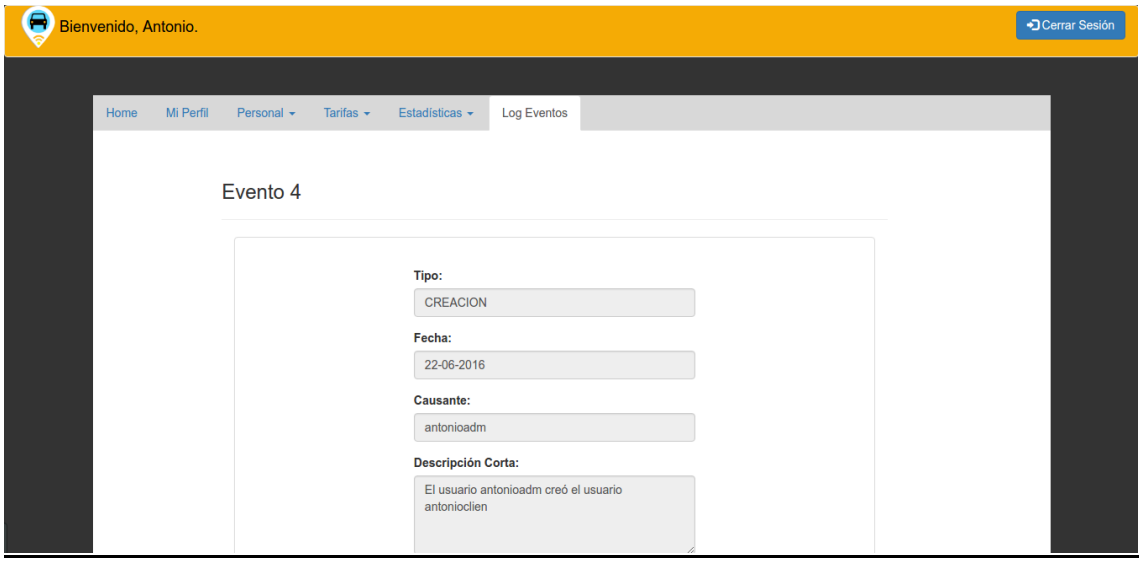


7.2.4.15 Estadísticas Gráficas

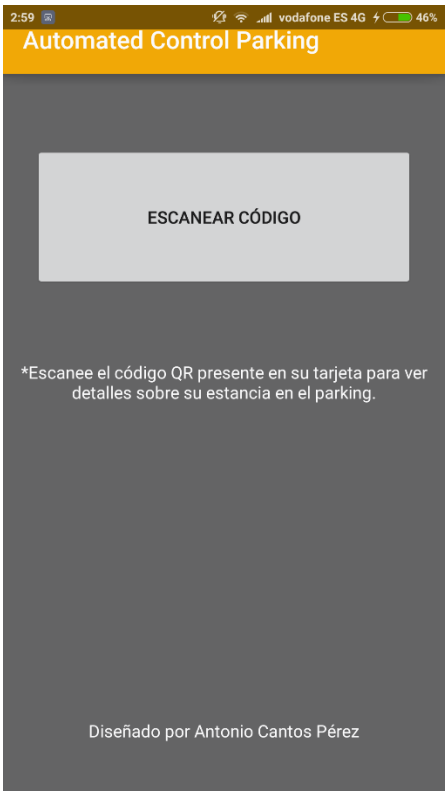
7.2.4.16 Log de Eventos

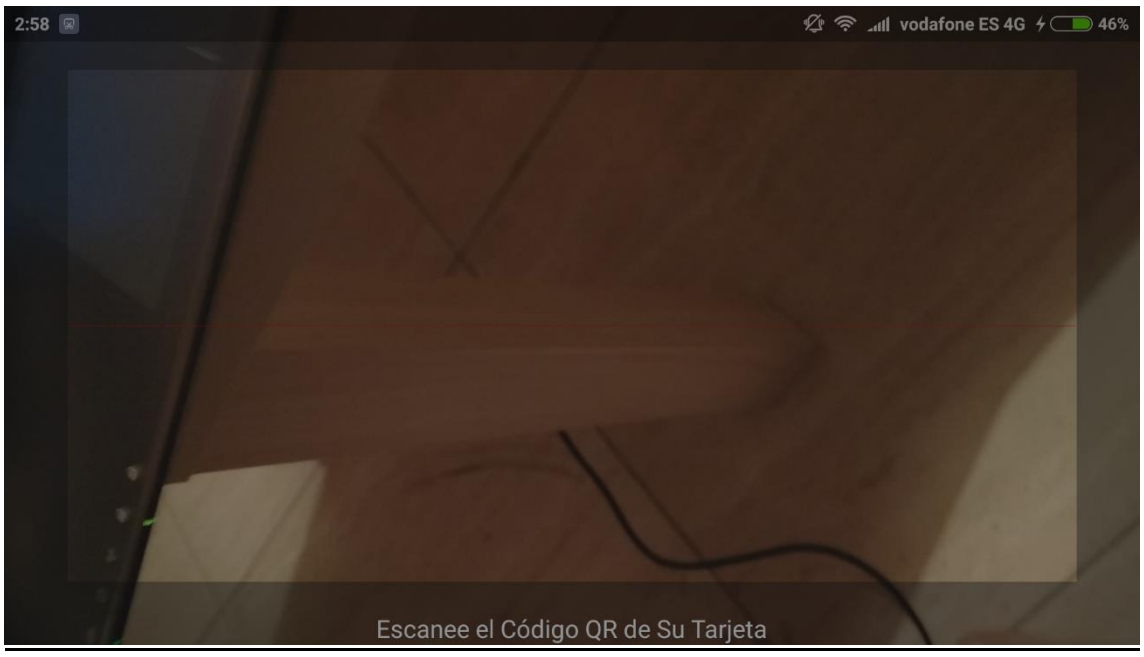


7.2.4.17 Ver Evento



7.2.5 Capturas Aplicación Android






2:58
vodafone ES 4G
46%
Automated Control Parking
Código Tarjeta:
4534831EF4A
Tarifa Aplicada:
Tarifa Cliente VIP
Plaza Ocupada:
Planta: 1, Plaza: 1A
Importe Actual:
0.91€