Universitat Oberta de Catalunya

UOCSAT

José Manuel Martínez Roca
Grado de ingeniería informática
Java EE

Consultor: Albert Grau Perisé
Profesor: Santi Caballe Llobet

8 de enero de 2020

FICHA DEL TRABAJO FINAL

Título del trabajo:	*UOCSAT*
Nombre del autor:	*José Manuel Martínez Roca*
Nombre del consultor/a:	*Albert Grau Perisé*
Nombre del PRA:	*Santi Caballe Llobet*
Fecha de entrega (mm/aaaa):	01/2020
Titulación::	*Grado de ingeniería informática*
Área del Trabajo Final:	*Java EE*
Idioma del trabajo:	*Español*
Palabras clave	*TFG, JavaEE, JSF, UOC.*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

El objetivo de este proyecto es reunir los estudios y competencias adquiridos durante el Grado de Ingeniería Informática y su especialización en ingeniería de software. Siendo necesarios conocimientos en bases de datos, programación, análisis y diseño de software, patrones de diseño, frameworks.

La metodología usada ha sido una adaptación de la tecnología ágil, en concreto una adaptación de SCRUM, aplicando las tecnologías JavaEE: Java Server Faces (JSF) en la capa de presentación, ManagedBean en la capa de negocio y JPA en la capa de integración. Se ha mejorado la aplicación web con el uso de JQuery y JavaScript.

La aplicación web implementa la rutina de gestión diaria de un servicio técnico de computadoras tanto para servicios externos como internos. Para ello se incluyen las altas de los usuarios, tanto técnicos como clientes, las altas de dispositivos, las altas de servicios como principales funcionalidades.

Se ha logrado con este proyecto controlar y facilitar la labor de dicho servicio técnico automatizando tareas y registrando cada uno de los movimientos de cada una de las entidades implicadas (clientes, técnicos, dispositivos, servicios, etc.) en el desarrollo diario del trabajo del servicio técnico.

Abstract (in English, 250 words or less):

The aim of this project is to bring together the studies and skills acquired during the Computer Engineering Degree and its specialization in software engineering. Being necessary knowledge in databases, programming, analysis and design of software, design patterns, frameworks.

The methodology used has been an adaptation of the agile technology, specifically an adaptation of SCRUM, applying the JavaEE technologies: Java Server Faces (JSF) in the presentation layer, ManagedBean in the business layer and JPA in the integration layer. The web application has been enhanced with the use of JQuery and JavaScript.

The web application implements the daily management routine of a computer technical service for both external and internal services. This includes user registrations, both technical and customer, device registrations, service registrations as main functionalities.

This project has managed to control and facilitate the work of this technical service by automating tasks and recording each of the movements of each of the entities involved (customers, technicians, devices, services, etc.) in the daily development of the technical service's work.

Índice

A Isabel, mi amor, por tu infinita paciencia.

1. Introducción

1.1 Contexto y justificación del Trabajo

En este TFG la necesidad a cubrir es el desarrollo de una aplicación que facilite el trabajo diario a un servicio técnico de informática, incluyendo, la creación de clientes, creación de técnicos, la creación de dispositivos y la creación de servicios o incidencias.

Es un tema relevante ya que se pretende desarrollar la aplicación mediante tecnologías JavaEE, logrando de esta manera aplicar los objetivos del TFG en cuanto a tecnología se refiere y proporcionando un entorno adecuado para aplicaciones en un entorno empresarial. En lo referente a las aplicaciones de las competencias adquiridas en el Grado de Ingeniería Informática constituye un proyecto que permite aplicar dichas competencias.

Actualmente el servicio técnico toma los servicios a mano y eso es inadmisible en la época actual con el desarrollo de las tecnologías de información que tenemos a nuestro alcance, ya que la afirmación que más se les escucha a los clientes cuando están tomando nota de su avería es: “ya estoy dado de alta en su base de datos, ya debe tener mis datos ahí”, esto es del todo punto inverosímil que en una empresa del siglo XXI de informática se sigan haciendo las cosas así, de ahí la motivación de mi proyecto TFG.

Como objetivo final se pretende obtener una aplicación en la que solamente se deba dar de alta una sola vez a un cliente y de ahí agregarle cada uno de los dispositivos que pasen por el servicio técnico para así llevar un control de averías de los mismos, un historial de reparaciones de equipos y otros cálculos económicos como rentabilidades por cliente y rentabilidades por equipos, y algunos otros estudios estadísticos y económicos sobre la información que seremos capaces de manejar con dicha base de datos.

1.2 Objetivos del Trabajo

1.2.1 Objetivos generales

- Permitir a los técnicos de la empresa registrar clientes, dispositivos y servicios mediante la aplicación web sin tener que depender de infraestructura hardware o software en cada ordenador disponiendo de una web en un servidor con persistencia en base de datos.
- Permitir a los clientes consultar el estado de su reparación o servicio a través de la aplicación web sin necesidad de contactar por teléfono con la empresa o sin necesidad de desplazamientos innecesarios.

- Permitir a la empresa llevar un historial de los equipos y servicios reparados para fines estadísticos.

1.2.2 Objetivos específicos

- Desarrollar una aplicación web basada en tecnologías JavaEE mediante JSF (Java Server Faces), *Managed Bean* y JPA, adornado con el *framework* JQuery para proporcionar una interfaz gráfica sencilla, intuitiva y amigable.

1.2.3 Objetivos didácticos

- Lenguaje de programación Java, versión 1.7.
- JavaEE
    - JSF (Java Server Faces): es un lenguaje de marcado de la capa de presentación que interactúa con los *Managed Bean* que son objetos que recogen la interacción del usuario con la interfaz y la transmiten a la capa de negocio.
    - *Managed Bean*: son objetos (clases Java) de la capa presentación que recogen la interacción de los usuarios con las JSF y la transmiten a la capa de negocio (EJB), a través de anotaciones son accesibles desde las JSFs.
    - EJB: son objetos Java (clases e interfaces) de la capa de negocio que reciben desde los *Managed Bean* peticiones, tratando los datos recibidos, logrando su persistencia en base de datos y retornando una respuesta satisfactoria o no, al usuario a través de la capa de presentación.
    - JPA: es un *framework* de la capa de negocio que permite representar entidades mediante clases Java, dichas clases Java se corresponden con las tablas de la base de datos sobre la que se conseguirá la persistencia de los datos usados en la aplicación y mediante el lenguaje de consultas JPQL
- JQuery: es un *framework* Javascript que permite simplificar la interacción con los elementos HTML para dotar a las aplicaciones web de sencillez, intuitividad y amigabilidad.
- PostgreSQL: sistema gestor de bases de datos relacionales que se utiliza para dotar de persistencia a los datos que maneja la aplicación.
- Aplicar conocimientos adquiridos en el Grado de Ingeniería Informática como gestión de proyectos, ingeniería de requisitos, diseño de base de datos, análisis y diseño de software, patrones de diseño, etc.

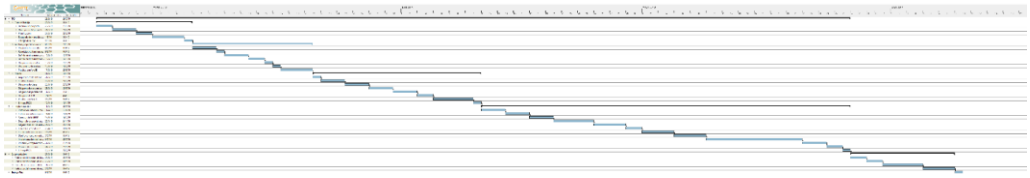
1.3 Enfoque y método seguido

Debido a la especificación de los requisitos del cliente no se puede adaptar un producto existente en el mercado por lo que se opta por

desarrollar un producto nuevo ya que será mucho más económico, sencillo y específico para nuestro cliente que adaptar la empresa al funcionamiento de cualquier producto ya existente. Por otro lado, al desarrollar un producto nuevo se consigue profundizar mejor en el uso de las tecnologías mencionadas en el apartado anterior.

1.4 Planificación del Trabajo

Diagrama de planificación temporal del TFG:



Los recursos utilizados en el desarrollo del TFG son:

RECURSO	VERSION	LOGO
PC portátil	Asus K551L – Intel core I7 8GB RAM 240 GB SSD	
Sistema operativo	Windows 10 Home edition	
Entorno de desarrollo (IDE)	Eclipse Photon	
JavaEE	JSF, Managed Bean y JPA	
Sistema gestor de bases de datos	PostGreSQL versión 10.6	
Aplicación de gestión de bases de datos para el SGDB PostGreSQL	pgAdmin 4 versión 4.16	

Framework JavaScript	JQUERY 1.7.1	
Servidor de aplicaciones	WildFly 13.0.0.Final	
Automatización de procesos de compilación	Apache Ant 1.10.5	
Herramienta de diagramas de Gantt	GanttProject version 2.8.10	
Herramienta diseño de software CASE	MagicDraw 18.5	
Herramienta de control de versiones	Git (versión 2.19.1)	
Herramienta de control de versiones	GitLab (versión 12.5.3)	

1.5 Breve sumario de productos obtenidos

Los productos obtenidos durante el desarrollo del TFG son:

- **Plan de trabajo**: objetivos del proyecto, requisitos funcionales de la aplicación, planificación temporal del proyecto, tecnologías propuestas, y metodologías de desarrollo de software.
- **Documentación de análisis y diseño**: descripción textual y diagramas de casos de uso, diseño de la interfaz de la aplicación (prototipo), diseño relacional de la base de datos, diagrama de clases, y diagrama de arquitectura.
- **Proyecto Java (Eclipse):** implementación de la aplicación, o sea, el código fuente.

- **Documento de pruebas**: Documenta las pruebas funcionales de la aplicación web.
- **Manual de despliegue**: documenta como implementar el entorno de desarrollo.
- **Presentación virtual**: video explicativo y demostrativo de la aplicación.
- **Memoria**: el presente documento.

1.6 Breve descripción de los otros capítulos de la memoria

- **Análisis**: en este capítulo se describen las necesidades del proyecto, los requisitos no funcionales y los requisitos funcionales, así como el diseño conceptual.

  Su relación con el trabajo global a realizar en el TFG es que este capítulo constituye la base para poder desarrollar el contenido del resto del trabajo final de grado, es lo que se espera que la aplicación web resuelva.

- **Diseño**: esta es la fase previa a la implementación, en esta fase se diseña la aplicación, comenzando por la arquitectura de la aplicación en tres capas (MVC).

  Seguimos por el diseño estructurado en componentes que aún es independiente de la tecnología que usemos en concreto, y finalizando por el diseño en componentes adaptado a la tecnología concreta (JavaEE).

  En esta fase también quedaran definidos los casos de uso, el diseño relacional de la base de datos que posteriormente implementaremos al definir los JPA.

  Como último subcapítulo dentro de esta misma fase de diseño diseñaremos el prototipo de la aplicación en cuanto a interfaz gráfica de usuario.

  Su relación con el trabajo global es que partiendo del análisis definido en la etapa anterior, se documenta la estructura de la aplicación, desde la arquitectura más global hasta el diseño más detallado y dependiente de la arquitectura y tecnologías elegidas permitiendo continuar con la implementación.

- **Implementación**: en esta fase se describe la implementación del diseño comentando las diferentes capas de la aplicación y los elementos no previstos en el diseño inicial.

  Este apartado está relacionado con el trabajo global en que partiendo del producto de la fase anterior, el diseño, se obtiene el producto de esta fase, la implementación de la aplicación o código fuente.

- **Objetivos conseguidos**: en esta apartado se analizan los objetivos previstos en la planificación original (PEC1), y el objetivo global del TFG, determinando qué objetivos se alcanzaron y cuales no se alcanzaron.

  Su relación con el trabajo global se basa en evaluar cuales de los objetivos propuestos se han alcanzado y su consecución.

- **Trabajo futuro y potencial mejora**: este apartado define qué características se pueden agregar a la aplicación en un futuro como mejora, tanto funcional como no funcional (automatización de prueba, evaluación de la calidad del código, seguridad, etc.)

  Su relación con el trabajo global radica en analizar cómo se podría mejorar el diseño del producto, tanto internamente como de cara al cliente.

- **Conclusiones**: este capítulo describe las reflexiones del autor del TFG a partir del trabajo realizado a lo largo del semestre, siendo su relación con el trabajo global analizar las ideas obtenidas tras desarrollar el TFG, enmarcándolo en el contexto de los estudios de Grado de Ingeniería Informática.

2. Análisis

2.1 Análisis de necesidades del proyecto

Para desarrollar el Trabajo de Fin de Grado, dentro del área de JavaEE, voy a desarrollar una aplicación de Gestión de Reparaciones que funciona basada en dicha arquitectura. Las principales funcionalidades de la aplicación son las siguientes:

- El cliente o el técnico pueden crear y consultar las incidencias mediante un portal web.
- El técnico puede dar de alta equipos o incidencias indicando todas las características de los mismos así como los fallos que se encuentran en dichas incidencias o equipos.
- El administrador puede asignar las incidencias al técnico que considere.
- Los técnicos registran las intervenciones que realizan sobre una incidencia y pueden cerrar esa incidencia.
- El sistema envía un mensaje de correo electrónico por cada uno de los hitos de las incidencias que ocurran como puede ser el alta, modificación y cierre de cada incidencia.

La aplicación debe ser fácil de modificar y extensible para poder añadir nuevas funcionalidades de forma fácil y económica.

2.2 Análisis de requisitos

2.2.1 Requisitos no funcionales

RNF1: La aplicación será ejecutable en cualquier navegador web de cualquier dispositivo ya que se trata de una aplicación web que genera un servidor de aplicaciones (Wildfly).
RNF2: La aplicación web deberá ser intuitiva, simple y amigable para cualquier usuario.
RNF3: La aplicación será diseñada usando el patrón MVC, dicha arquitectura de tres capas nos permitirá separar la presentación de la lógica de negocio y del acceso a los datos usando J2EE facilitando así el posterior mantenimiento y ampliaciones de dicha aplicación.
RNF4: La aplicación se desplegara en un servidor de aplicaciones Wildfly alojado en un servidor provisto de un procesador Intel Core I7, 8 GB de RAM y sistema operativo Windows 10.
RNF5: La aplicación usará una BBDD PostgreSQL alojada en el servidor anteriormente descrito.
RNF6: El acceso a la aplicación será controlado mediante el correo electrónico y contraseña para cada usuario.

2.2.2 Requisitos funcionales

Administrador:

Aunque la aplicación web contará inicialmente con un solo tipo de usuario (administrador) que podrá realizar todas las acciones necesarias para creación y modificación del resto de usuarios de la aplicación como son los técnicos o los clientes.
El proceso de alta de los nuevos técnicos o clientes se llevara a cabo mediante un nombre de usuario (email) y una contraseña. No se solicitaran más datos en las altas ya que se orienta a la robustez y la seguridad. Posteriormente el técnico podrá rellenar el resto de datos.

Técnico:

En caso de no recordar los datos de acceso los técnicos podrán solicitar la recuperación de la contraseña mediante el correo electrónico indicado en el alta de dicho usuario.

El proceso de acceso a la aplicación consiste en la introducción en un formulario de la cuenta de correo electrónico y la contraseña correspondiente a dicho usuario.

Los técnicos habilitados en la aplicación podrán crear, modificar, consultar y buscar incidencias.

Creación de incidencias:

> En cada incidencia se indicaran un código automático autoincrementado de numero de incidencia, la fecha de entrada de la incidencia, el nombre del cliente, el código de seguimiento, el teléfono del cliente, el email del cliente, el nombre del dispositivo a reparar, un desplegable con un tipo de dispositivo (ordenador clónico, ordenador portátil, Tablet, Smartphone, etc.), el número de serie, un campo de texto con la descripción del problema, un desplegable con la actuación a realizar, un campo de texto con los comentarios internos no visibles para el cliente, un campo de texto con los detalles de la reparación, el PVP, un desplegable con el precio, el nombre del técnico asignado, un desplegable con los técnicos, el estado de la incidencia, el desplegable con el estado de la incidencia, la fecha de salida y un botón con la fecha de hoy para introducirla fácilmente en el campo anterior.

Modificación de incidencias:

> Se debe permitir modificar toda la información perteneciente a cualquier incidencia siempre que esta incidencia no se encuentre cerrada.

Consulta de incidencias:

> Se debe permitir consultar cualquier incidencia desde el listado inicial de incidencias.

Se debe permitir la búsqueda por nombre, apellido, teléfono o número de serie del equipo.

Cliente:

El usuario de tipo cliente solamente podrá consultar las reparaciones que le correspondan mediante un código que el sistema genera automáticamente en cada alta de una incidencia y que se comunicara por correo electrónico además de físicamente cuando el cliente abra la incidencia en nuestra tienda.

El usuario cliente recibirá una notificación cada vez que la incidencia cambie de estado.

La desconexión de la aplicación puede ser automática cuando la realice el sistema o mediante solicitud de cada usuario.

La aplicación deberá soportar un sistema de desconexión automática de la sesión transcurrido un tiempo prudencial de inactividad.

2.2.3 Definición de subsistemas.

Subsistema de Conexión: Este subsistema se encarga del acceso y validación de los distintos usuarios en la aplicación. El objetivo principal es impedir que usuarios no registrados accedan al sistema. Se encargará del alta, baja o modificación de los distintos usuarios. Estos usuarios del sistema serán. Administrador, Técnicos, clientes.

Subsistema de Gestión: Este subsistema se encarga de todas las acciones que tengan que ver con los equipos y las incidencias que se generan en el sistema para darlos de alta, baja, modificación etc.

Subsistema de comunicación: Este subsistema se encarga de la gestión de las comunicaciones. Aunque ahora será un subsistema muy simple debido a que albergará pocas funcionalidades se crea por la necesidades en futuras versiones ya que actualmente el foco no se centra en la comunicación con el cliente sino en obtener un sistema de gestión de incidencias donde la única funcionalidad en este subsistema es comunicar cada alta, cambio o finalización de incidencias al cliente.

2.2.4 Definición de actores

La herramienta web contará con tres tipos de usuario desde el inicio que según sus funciones podrán crear al resto de usuarios de esta aplicación, como es el caso del administrador que puede crear clientes o técnicos, el técnico que puede crear incidencias, y el cliente que solamente puede consultar mensajes o incidencias que le correspondan.

Cliente: El usuario de tipo cliente solamente puede consultar las incidencias que le atañen y no puede realizar ninguna otra acción. Las consultas de las incidencias se hacen mediante un código que se autogenera al crear la incidencia, dicho código se entrega al cliente cuando entrega un equipo o solicita una asistencia por teléfono. Recibirá notificaciones en su cuenta de correo electrónico por cada cambio de estado de las incidencias con él asociadas.

Técnico: El usuario de tipo técnico puede generar una incidencia por petición de un cliente y modificar dicha incidencia. No puede eliminar incidencias, aunque realmente no son eliminadas sino que pasan a un registro de incidencias eliminadas, pero si pueden cerrar incidencias. Registran cada intervención en cada una de las incidencias.

Administrador: El usuario tipo administrador puede crear a otros administradores u otros tipos de usuarios (técnico o cliente).

Sistema: Aunque no es un actor puede enviar mensajes a los implicados en cada incidencia por cada intervención que se realice o cada cambio de estado que se produzca en la incidencia en cuestión.

2.2.5 Casos de uso

2.2.5.1 Casos de uso de sistema de conexión

CASO DE USO	CU_01: alta de usuario
Actores	Administrador, Técnico
Precondición	El usuario dispone de una cuenta de correo electrónico y no está registrado ni identificado en el sistema
Postcondición	El usuario dispone de una cuenta para acceder al sistema y debe estar registrado e identificado en el sistema.
Escenario principal	1. Introduce el email, una contraseña y repite la contraseña
	2. El usuario pulsa sobre el botón registrar y accede a la aplicación
	3. El sistema validad que la cuenta no existe en el sistema y permite continuar con el registro.
	4. El usuario tiene acceso a las funcionalidades especificadas en su tipo.
Escenario alternativo	2.1 Los datos están mal formateados, el usuario ya existe, o la contraseña es incorrecta. A continuación, el sistema le notifica el error y lo devuelve a pantalla de creación de usuario.

CASO DE USO	CU_02: identificación en el sistema (login)
Actores	Administrador, Técnico
Precondición	El usuario debe disponer de una cuenta de acceso al sistema
Postcondición	El sistema valida los datos de acceso, si existe, se permite acceder a las funcionalidades correspondientes al rol correspondiente. Si no existe muestra un error.
Escenario principal	1. Introduce el email y contraseña.
	2. El sistema valida los datos y permite el acceso.
	3. El usuario tiene acceso a las funcionalidades especificadas en su tipo.
Escenario alternativo	2.1 El usuario no existe, o la contraseña es incorrecta. A continuación, el sistema le notifica el error y lo devuelve a pantalla de identificación de usuario.

CASO DE USO	CU_03: recuperar contraseña

Actores	Administrador, Técnico
Precondición	El usuario dispone de una cuenta de correo electrónico y está registrado en el sistema
Postcondición	El usuario recupera la contraseña para acceder al sistema.
Escenario principal	1. El usuario introduce el email.
	2. El sistema comprueba que existe ese email registrado en el sistema y envía un enlace para recuperar la contraseña.
	3. El usuario tiene acceso a las funcionalidades especificadas en su tipo.
Escenario alternativo	2.1 El sistema comprueba que el correo electrónico no existe, notifica el error y devuelve a la pantalla de inicio.

CASO DE USO	CU_04: baja de usuario
Actores	Administrador
Precondición	El usuario dispone de una cuenta de correo electrónico y está registrado en el sistema
Postcondición	El usuario se marca con inactivo en el sistema.
Escenario principal	1. Introduce el email del usuario a eliminar.
	2. El sistema comprueba que el usuario esta dado de alta.
	3. El sistema pide confirmación de la solicitud.
	4. El usuario confirma la operación.
	5. Se marca al usuario como inactivo.
Escenario alternativo	2.1 El usuario no existe.
	2.2 Se cancela la operación.

CASO DE USO	CU_05: desconectar sesión usuario (logout)
Actores	Administrador, Técnico
Precondición	El usuario está registrado e identificado en el sistema.
Postcondición	La sesión actual del usuario queda desconectada del sistema.
Escenario principal	1. El usuario selecciona la opción “Salir” del sistema
	2. El sistema destruye los objetos de la sesión.
	3. El sistema presenta la página de Login.
Escenario alternativo	No existe flujo alternativo.

2.2.5.2 Casos de uso subsistema gestión

CASO DE USO	CU_10: gestionar los equipos (alta, consulta, modificación)
Actores	Administrador, técnico.
Precondición	El usuario debe haberse identificado en el sistema.
Postcondición	Ninguna.
Escenario principal	1. El usuario accede al listado de dispositivos por medio de la opción apropiada del menú principal
	2. El usuario solicita la acción adecuada presente en cada línea del listado.
	3. El usuario puede modificar, dar de alta o consultar un equipo.
Escenario alternativo	El usuario puede evitar la operación en cualquier momento pulsando cancelar.

CASO DE USO	CU_11: gestionar incidencia (alta, consulta, modificación)
Actores	Administrador, técnico.
Precondición	El usuario debe haberse identificado en el sistema.
Postcondición	Ninguna.
Escenario principal	1. El usuario accede al listado de servicios por medio de la opción apropiada del menú principal.
	2. El usuario solicita la acción adecuada presente en cada línea del listado.
	3. El usuario puede modificar, dar de alta o consultar una incidencia o servicio.
Escenario alternativo	El usuario puede impedir la operación en cualquier momento pulsando cancelar.

CASO DE USO	CU_12: buscar una incidencia por cliente
Actores	Administrador, técnico.
Precondición	El usuario debe haberse identificado en el sistema.
Postcondición	Ninguna.
Escenario principal	1. El usuario introduce en el lugar apropiado el nombre del cliente y pulsa sobre buscar.
	2. El sistema encuentra las incidencias asociadas a la búsqueda.
	3. El sistema muestra las incidencias asociadas a la búsqueda.
Escenario alternativo	2.a El sistema no encuentra incidencia alguna asociada a la búsqueda y muestra un mensaje como que no existe la incidencia en el sistema.

CASO DE USO	CU_14: buscar incidencia por numero serie
Actores	Administrador, técnico.

Precondición	El usuario debe haberse identificado en el sistema.
Postcondición	Ninguna.
Escenario principal	1. El usuario introduce un número de serie en el lugar apropiado y pulsa sobre buscar
	2. El sistema encuentra la incidencia asociada al número de serie del equipo del cliente.
	3. El sistema muestra la incidencia asociada a la búsqueda.
Escenario alternativo	2.a El sistema no encuentra la incidencia asociada a la búsqueda e indica que no existe esa incidencia en el sistema.

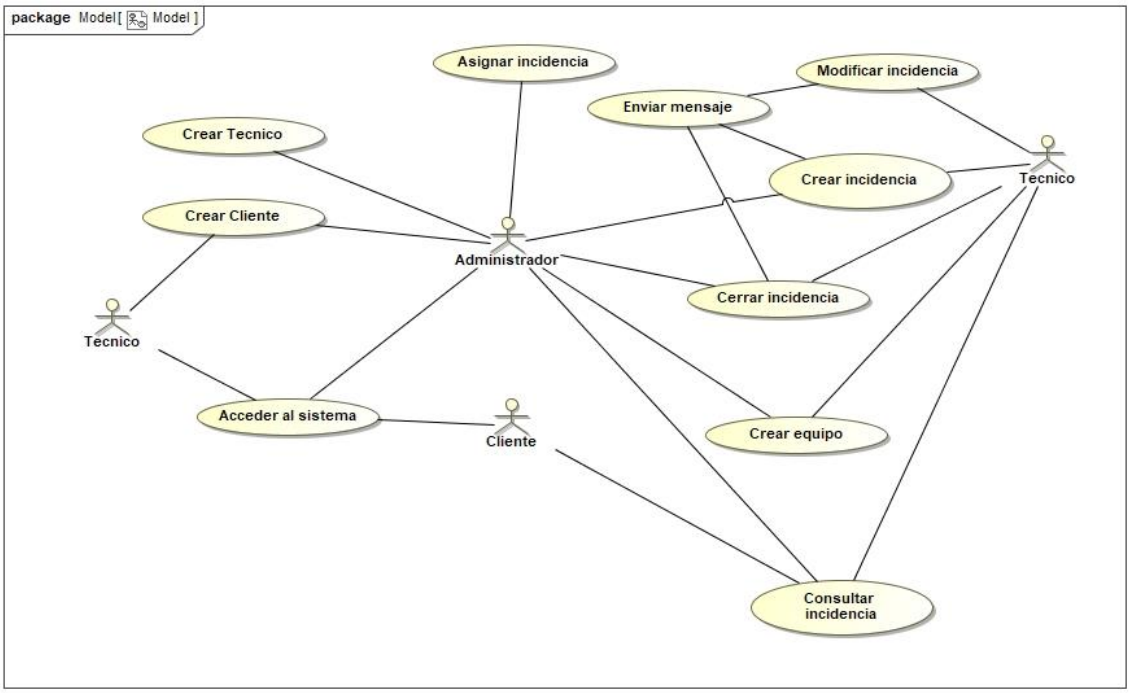
CASO DE USO	CU_15: ver listado de incidencias
Actores	Administrador, técnico.
Precondición	El usuario debe haberse identificado en el sistema.
Postcondición	Ninguna.
Escenario principal	1. Es la pantalla de inicio cuando un técnico inicio sesión correctamente
	2. El usuario puede ver el listado de incidencias o servicios pulsando sobre la opción adecuada del menú principal.
	3.
Escenario alternativo	

CASO DE USO	CU_16: ver listado de usuarios
Actores	Administrador.
Precondición	El usuario debe haberse identificado en el sistema.
Postcondición	Ninguna.
Escenario principal	1. El administrador puede ver el listado de usuarios pulsando sobre la opción adecuada situada en su pantalla inicial junto a las estadísticas.
	2.
	3.
Escenario alternativo	

2.2.5.3 Casos de uso subsistema comunicación

CASO DE USO	CU_20: mensaje modificación estado incidencia
Actores	Administrador, técnico.
Precondición	El usuario debe haberse identificado en el sistema y haber modificado el estado de una incidencia o incluido alguna novedad en una incidencia.
Postcondición	Ninguna.
Escenario principal	1. El sistema envía una notificación con los datos necesarios.
	2. El usuario recibe el email con las modificaciones.
Escenario alternativo	No existe flujo alternativo.

2.2.5.4 Diagrama de casos de uso



2.3 Diseño conceptual

Aquí se describe el diseño conceptual del sistema identificando aquellas entidades del mundo real, con sus atributos correspondientes, que se aplican al dominio de caso que nos ocupa: el servicio técnico.

Diagrama de clases

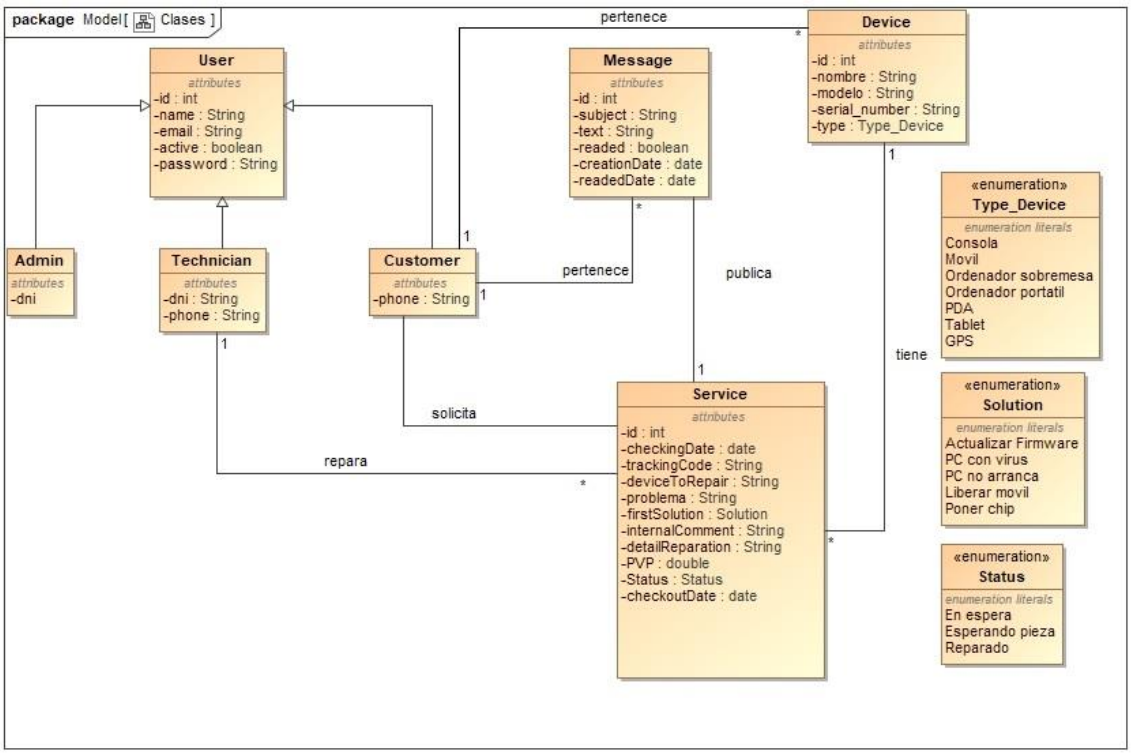
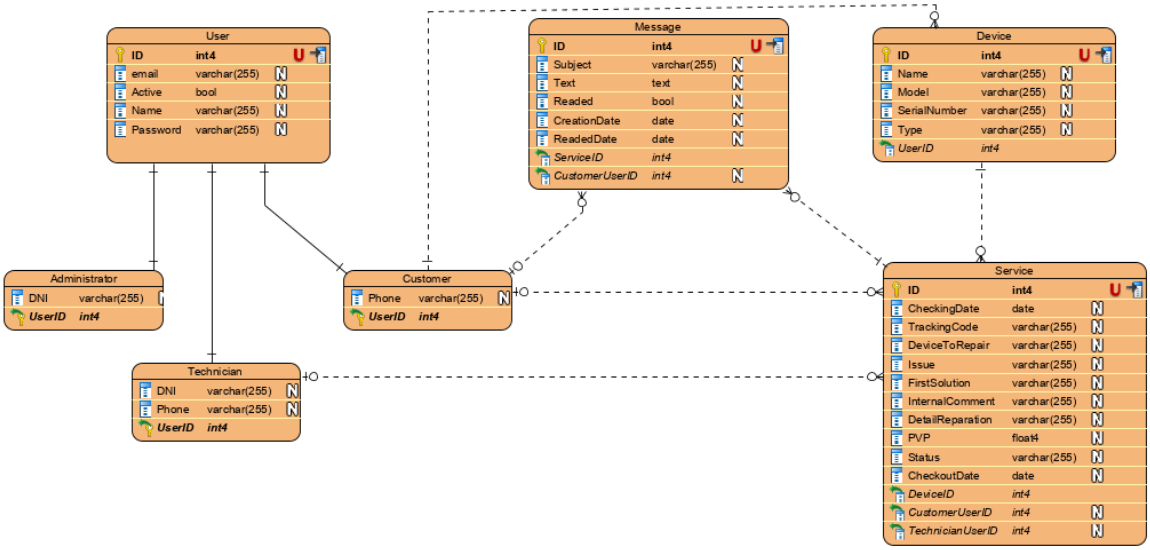


Diagrama relacional de base de datos:



3. Diseño

En este apartado se describe la arquitectura del sistema decidiendo la estructura de la aplicación web UOCSAT.

3.1 Arquitectura de la aplicación

Al decantarme por la arquitectura de la aplicación como arquitectura de tres capas o MVC tenemos:

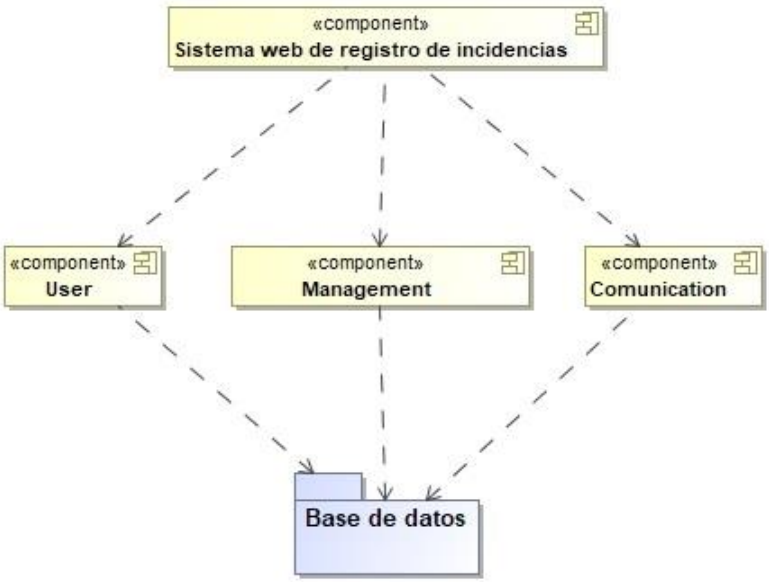
Capa de presentación: capa de la aplicación que se encarga de mostrar la información (vistas) que deben tener un mínimo de proceso para realizar un filtro previo para que no existan errores de formato. También se conoce como interfaz de usuario y debe ser amigable para el usuario. Se comunica con la capa de negocio. Muestran las funcionalidades exclusivas de un tipo de usuario, por ejemplo, en nuestro caso el administrador puede ver una opción del menú principal que solamente puede ver el que es la opción Técnicos para poder crear, actualizar o buscar técnicos dentro de la BBDD propia.

Capa de negocio: se reciben las peticiones del usuario desde la capa de presentación y se procesan para solicitar a la capa de integración su almacenamiento o lectura de datos.

Capa de integración: capa donde residen los datos, se encarga de acceder a los datos y de comunicarse con la capa de negocio para atender a sus solicitudes de almacenamiento o lectura de datos.

3.2 División de la aplicación en componentes

La aplicación se divide en tres componentes (usuario, gestión y comunicación) que están estrictamente relacionados con los paquetes que ya definimos en los casos de uso. Los componentes se diseñan mediante una interfaz y un componente en cada una de las capas de la aplicación: presentación, negocio e integración.

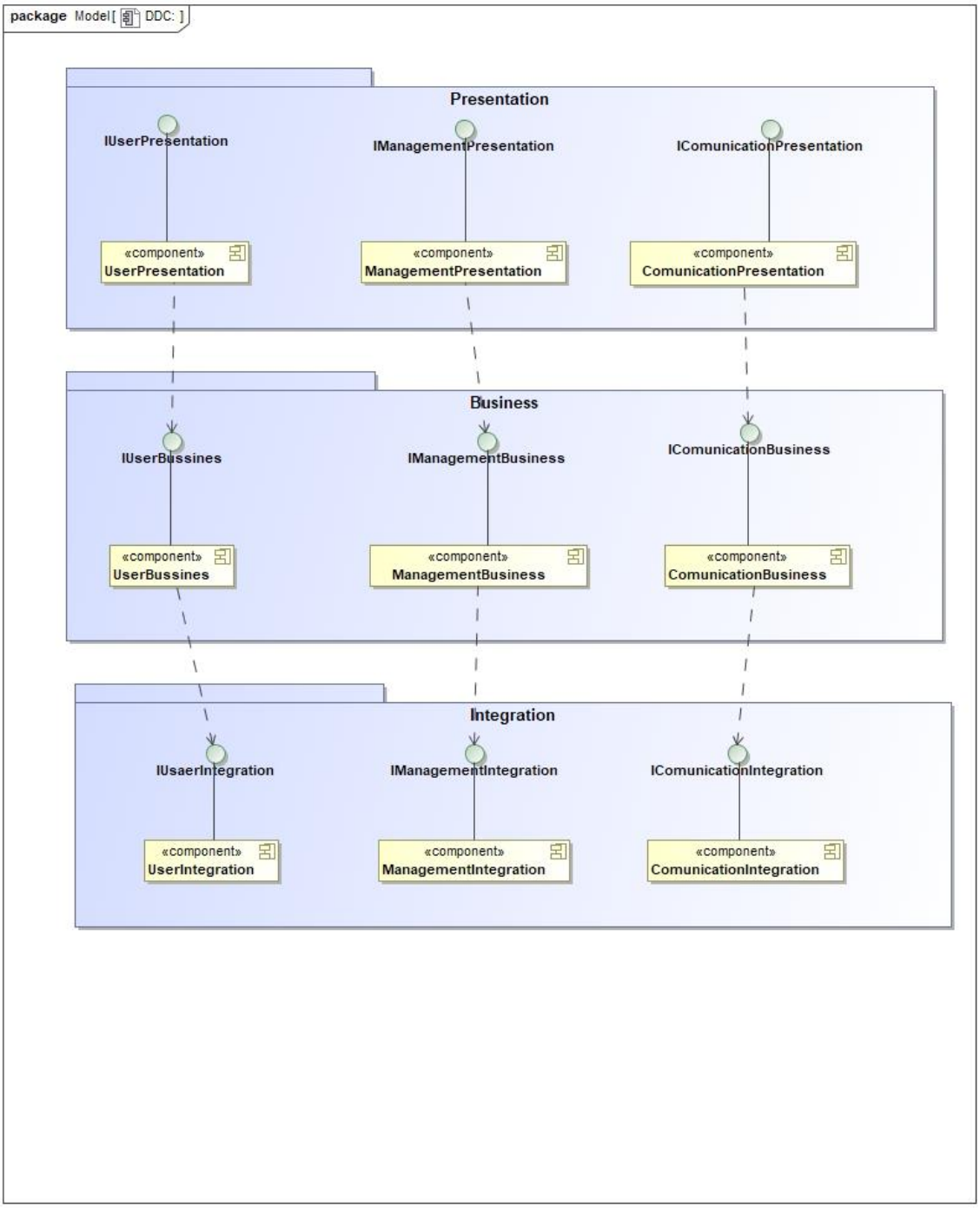


En la capa de presentación se haya la interfaz de usuario mediante la cual interactúa la aplicación con el usuario (vista) y se encarga de recoger toda interacción del usuario con la interfaz.

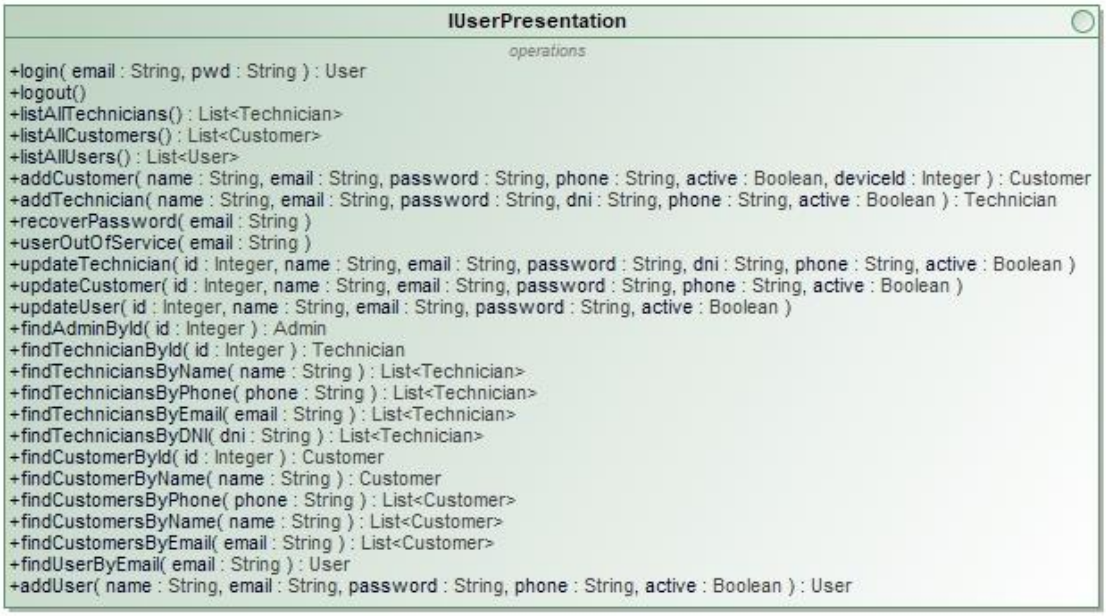
En la capa de negocio se encuentra la lógica de negocio de la aplicación y en ella se tratan los datos introducidos por el usuario en la capa de presentación para retornar una respuesta al usuario y transmitirlos a la capa de integración si corresponde.

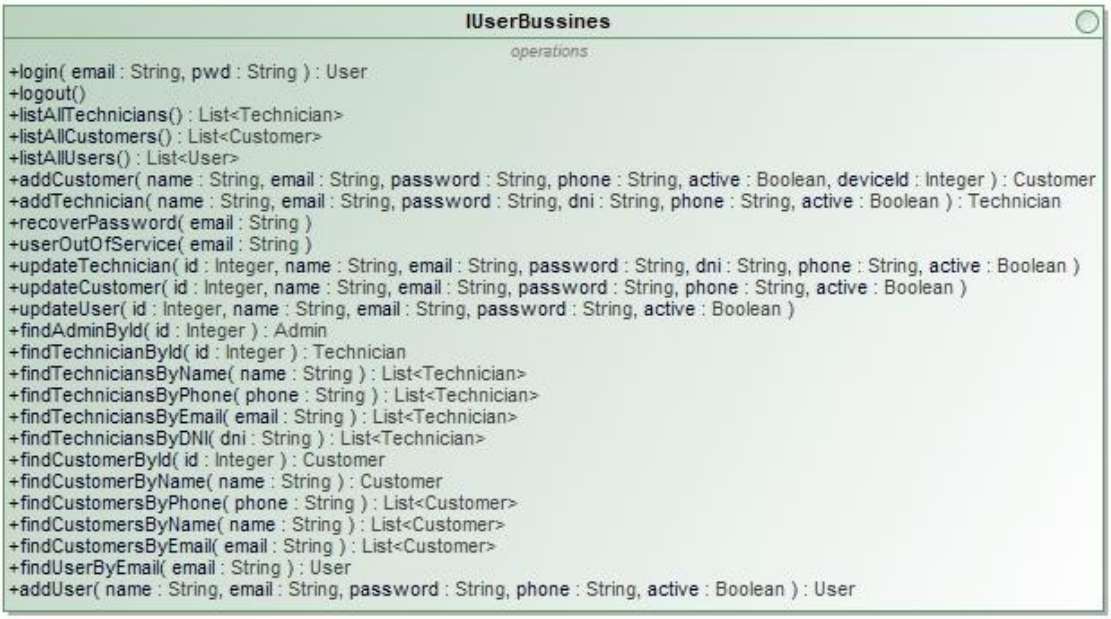
En la capa de integración podemos encontrar los datos que maneja la aplicación o también llamada persistencia.

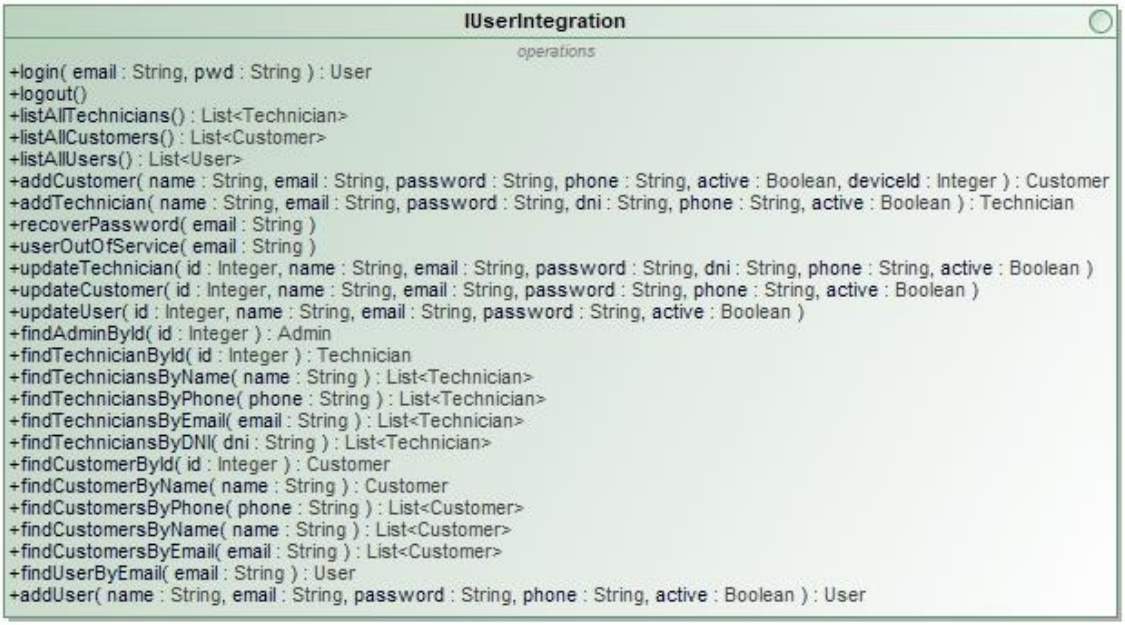
En el diagrama del punto de vista de la computación no se incluyen los detalles de cada interfaz para aumentar la legibilidad del diagrama.



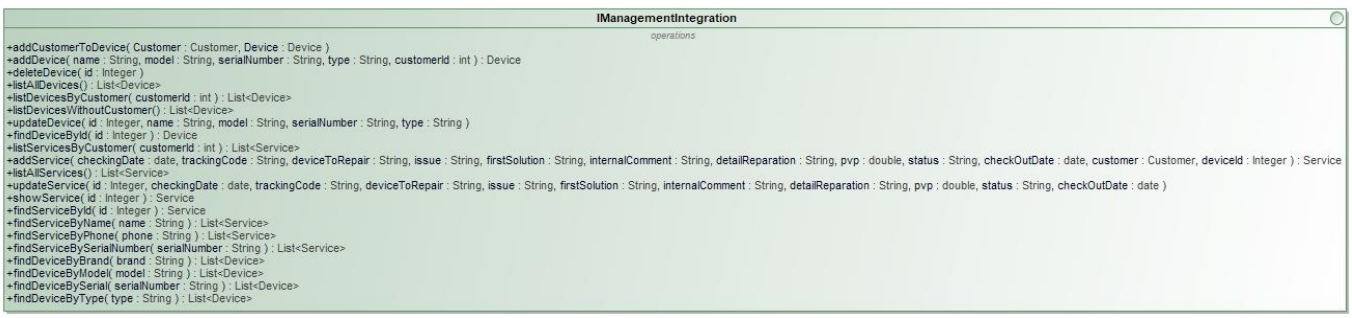
Detalle del componente Usuario:


IUserPresentation
operations
+login(email : String, pwd : String) : User
+logout()
+listAllTechnicians() : List<Technician>
+listAllCustomers() : List<Customer>
+listAllUsers() : List<User>
+addCustomer(name : String, email : String, password : String, phone : String, active : Boolean, deviceId : Integer) : Customer
+addTechnician(name : String, email : String, password : String, dni : String, phone : String, active : Boolean) : Technician
+recoverPassword(email : String)
+userOutOfService(email : String)
+updateTechnician(id : Integer, name : String, email : String, password : String, dni : String, phone : String, active : Boolean)
+updateCustomer(id : Integer, name : String, email : String, password : String, phone : String, active : Boolean)
+updateUser(id : Integer, name : String, email : String, password : String, active : Boolean)
+findAdminById(id : Integer) : Admin
+findTechnicianById(id : Integer) : Technician
+findTechniciansByName(name : String) : List<Technician>
+findTechniciansByPhone(phone : String) : List<Technician>
+findTechniciansByEmail(email : String) : List<Technician>
+findTechniciansByDNI(dni : String) : List<Technician>
+findCustomerById(id : Integer) : Customer
+findCustomerByName(name : String) : Customer
+findCustomersByPhone(phone : String) : List<Customer>
+findCustomersByName(name : String) : List<Customer>
+findCustomersByEmail(email : String) : List<Customer>
+findUserByEmail(email : String) : User
+addUser(name : String, email : String, password : String, phone : String, active : Boolean) : User

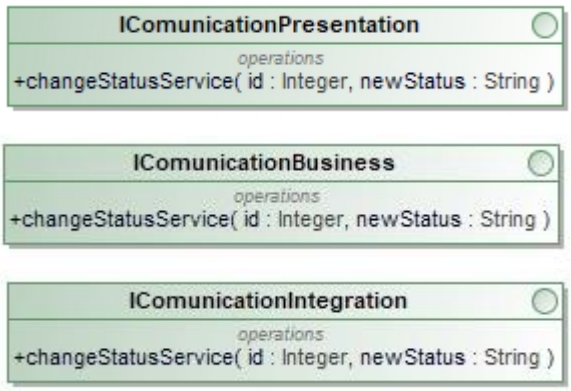

IUserBussines
operations
+login(email : String, pwd : String) : User
+logout()
+listAllTechnicians() : List<Technician>
+listAllCustomers() : List<Customer>
+listAllUsers() : List<User>
+addCustomer(name : String, email : String, password : String, phone : String, active : Boolean, deviceId : Integer) : Customer
+addTechnician(name : String, email : String, password : String, dni : String, phone : String, active : Boolean) : Technician
+recoverPassword(email : String)
+userOutOfService(email : String)
+updateTechnician(id : Integer, name : String, email : String, password : String, dni : String, phone : String, active : Boolean)
+updateCustomer(id : Integer, name : String, email : String, password : String, phone : String, active : Boolean)
+updateUser(id : Integer, name : String, email : String, password : String, active : Boolean)
+findAdminById(id : Integer) : Admin
+findTechnicianById(id : Integer) : Technician
+findTechniciansByName(name : String) : List<Technician>
+findTechniciansByPhone(phone : String) : List<Technician>
+findTechniciansByEmail(email : String) : List<Technician>
+findTechniciansByDNI(dni : String) : List<Technician>
+findCustomerById(id : Integer) : Customer
+findCustomerByName(name : String) : Customer
+findCustomersByPhone(phone : String) : List<Customer>
+findCustomersByName(name : String) : List<Customer>
+findCustomersByEmail(email : String) : List<Customer>
+findUserByEmail(email : String) : User
+addUser(name : String, email : String, password : String, phone : String, active : Boolean) : User



Detalle del componente Gestión:



Detalle del componente Comunicación:



3.3 Relación de la arquitectura de la aplicación con la tecnología JavaEE

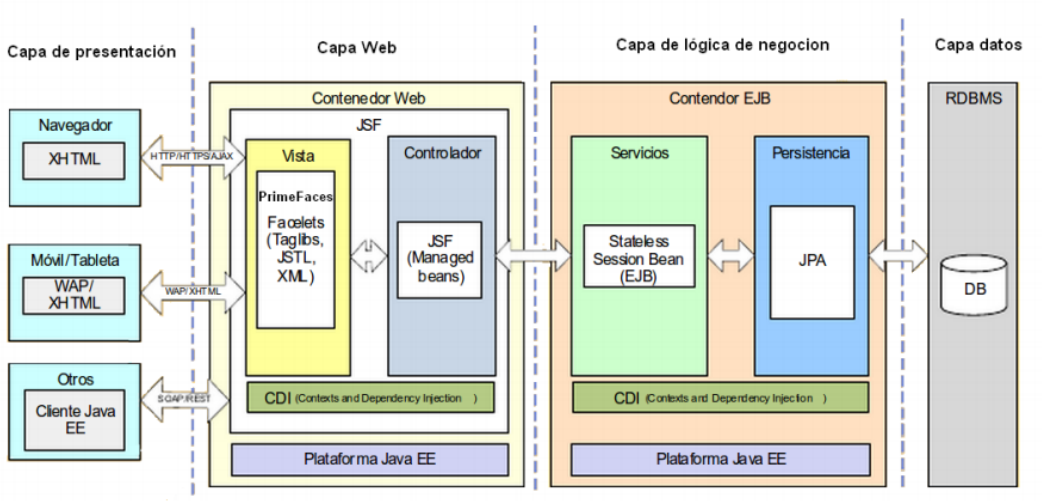
En las distintas capas de la aplicación la tecnología JavaEE se aplica debido a los siguientes elementos:

Capa presentación: Se usa el patrón de diseño MVC, las vistas se implementan mediante *Facelets* (JSF), el controlador, que es el encargado de gestionar el ciclo de vida de las aplicaciones web que implementan la interfaz mediante JSF está implícito en la tecnología JavaEE (*Faces Servlet*) y el modelo se implementa mediante *Managed Bean*, cuya función es recoger la interacción del usuario con la interfaz (*facelets*). En esta capa se integra la tecnología de las vistas JavaEE (JSF) con la librería JavaScript JQuery para lograr funcionalidades que no ofrece Java, y así conseguir una interfaz más atractiva y agradable al usuario.

Capa de negocio: en esta capa se implementa la lógica de la aplicación encapsulada en componentes EJB. O sea, componentes de servidor pertenecientes a la tecnología JavaEE.

Capa de integración: en esta capa se implementa la persistencia de los datos usados por la aplicación almacenando la información mediante uno o más gestores de bases de datos. Se utilizan los JPA para realizar un mapeo objetos / relacional para gestionar datos relacionales en aplicaciones JavaEE, esta tecnología conlleva el uso del lenguaje de consultas propio JPQL.

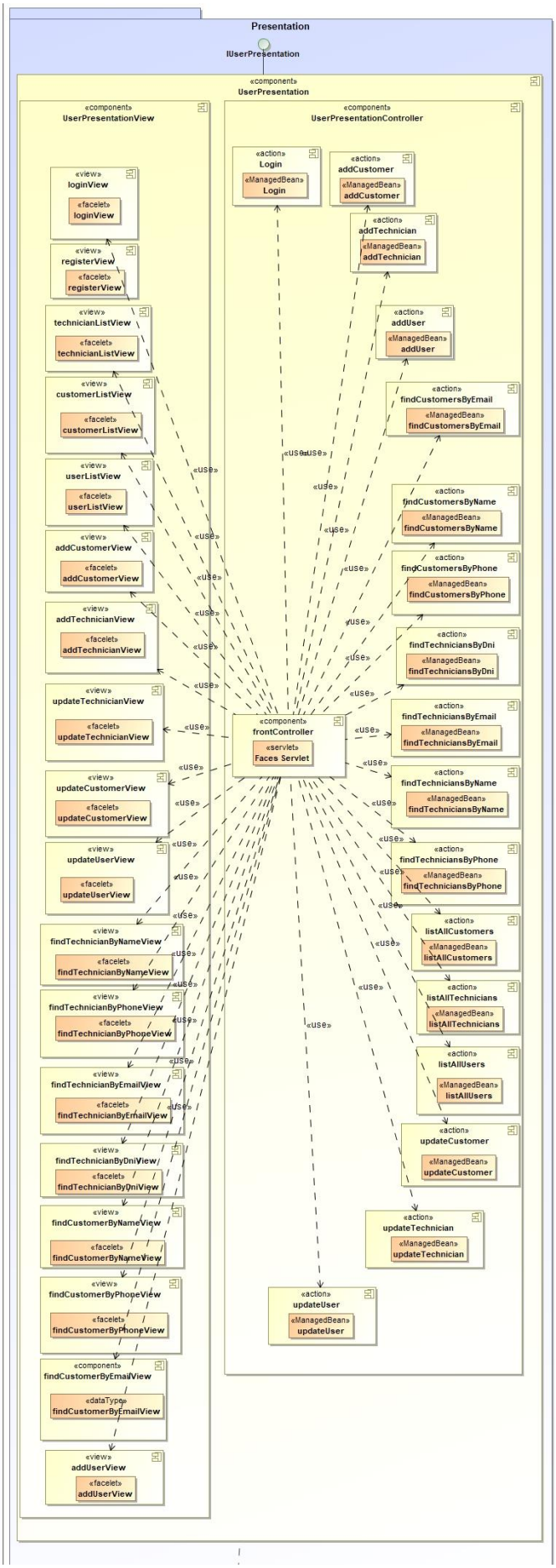
ESTRUCTURA EN CAPAS MVC PARA JAVAEE

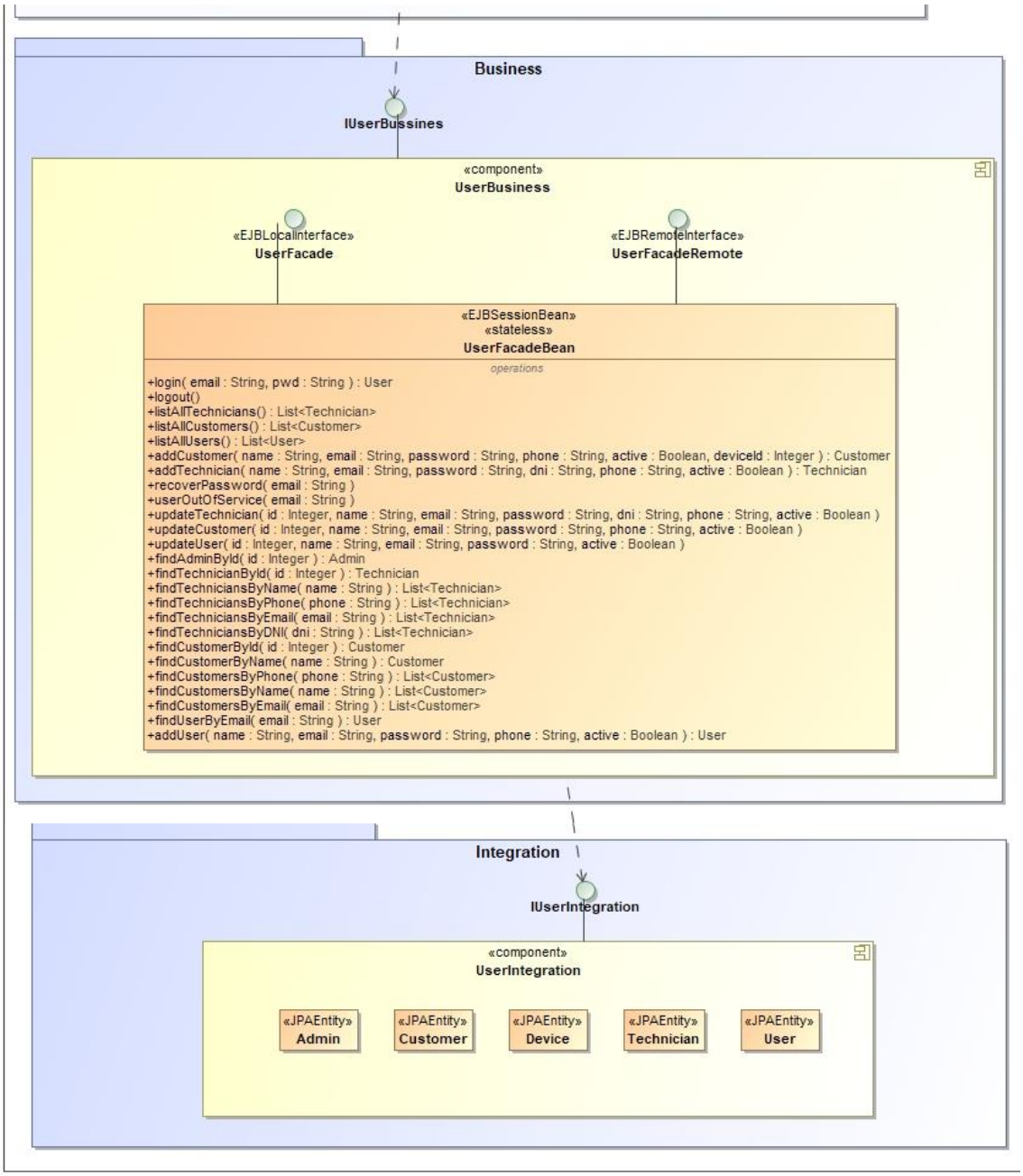


Aunque tenía previsto usar varios *Frameworks* como *Primefaces* o Spring en el desarrollo del TFG debido a la escasez de tiempo y el esfuerzo necesario para ello, así como el poco uso que iba a hacer de estos *frameworks* he decidido hacer uso de JQuery para las pocas funcionalidades que no cubría la tecnología JavaEE y así no poner en riesgo el desarrollo en las fechas programadas al inicio del TFG.

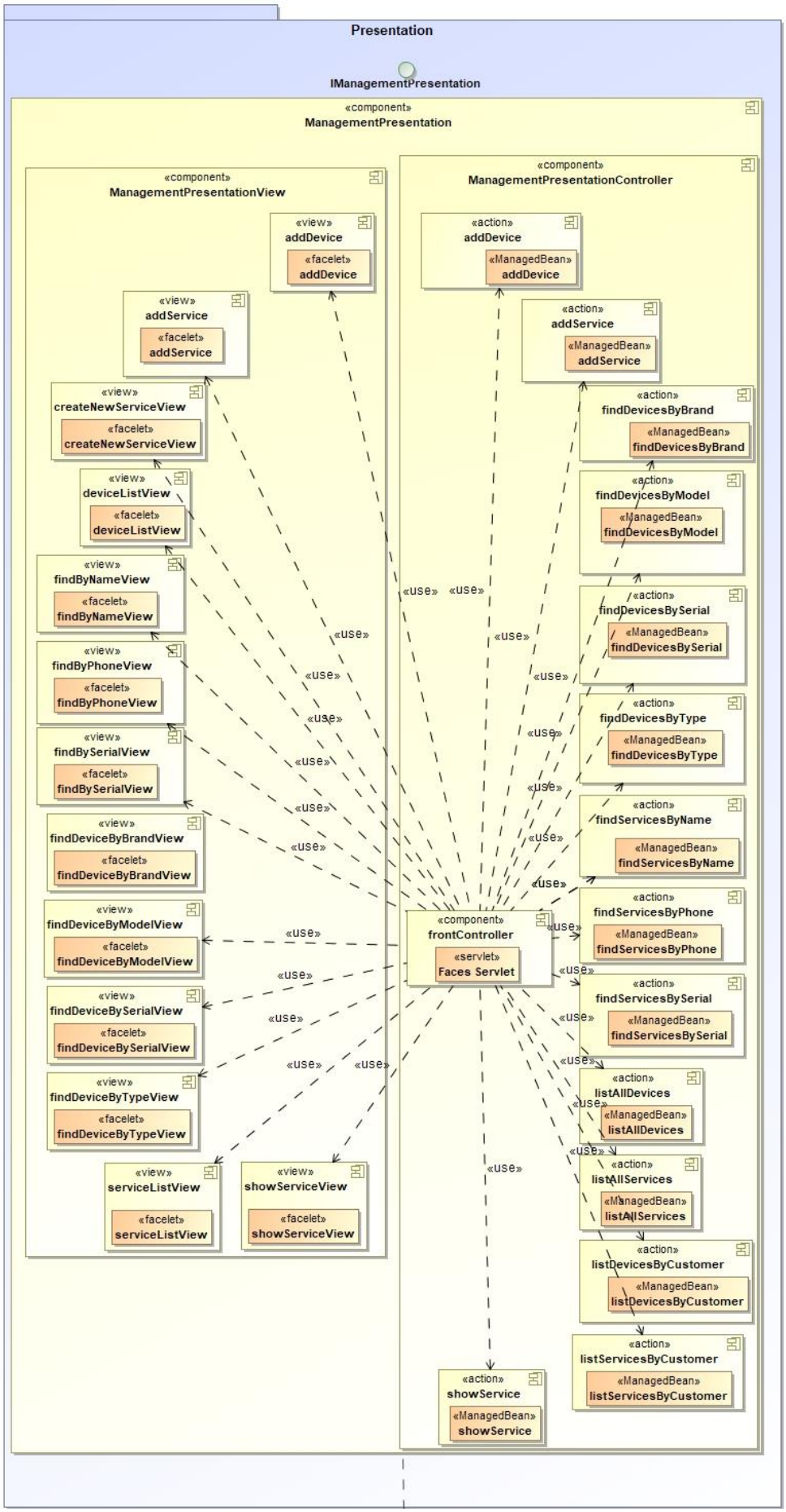
Seguimos mostrando los diagramas de componentes para cada uno de los componentes en los que he dividido la aplicación.

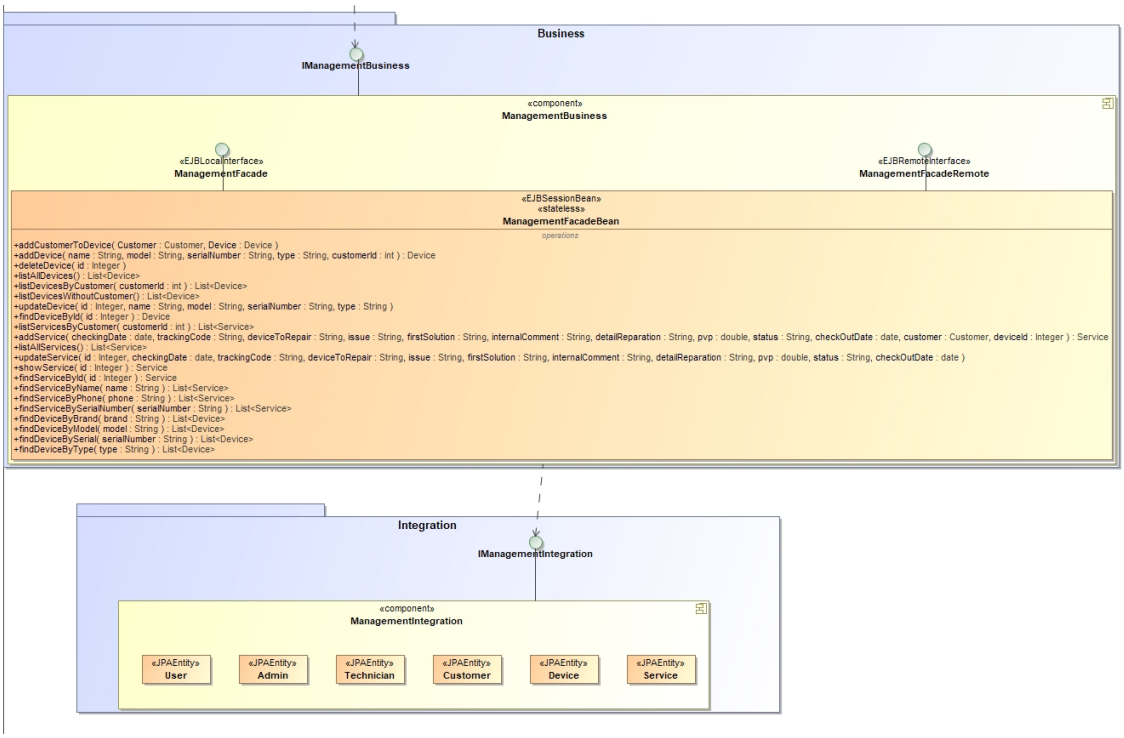
PUNTO DE VISTA DE LA COMPUTACION – COMPONENTE USUARIO:


Presentation
IUserPresentation
«component»
UserPresentation
«component»
UserPresentationView
«component»
UserPresentationController
«view»
loginView
«facelet»
loginView
«view»
registerView
«facelet»
registerView
«view»
technicianListView
«facelet»
technicianListView
«view»
customerListView
«facelet»
customerListView
«view»
userListView
«facelet»
userListView
«view»
addCustomerView
«facelet»
addCustomerView
«view»
addTechnicianView
«facelet»
addTechnicianView
«view»
updateTechnicianView
«facelet»
updateTechnicianView
«view»
updateCustomerView
«facelet»
updateCustomerView
«view»
updateUserView
«facelet»
updateUserView
«view»
findTechnicianByNameView
«facelet»
findTechnicianByNameView
«view»
findTechnicianByPhoneView
«facelet»
findTechnicianByPhoneView
«view»
findTechnicianByEmailView
«facelet»
findTechnicianByEmailView
«view»
findTechnicianByDniView
«facelet»
findTechnicianByDniView
«view»
findCustomerByNameView
«facelet»
findCustomerByNameView
«view»
findCustomerByPhoneView
«facelet»
findCustomerByPhoneView
«component»
findCustomerByEmailView
«dataType»
findCustomerByEmailView
«view»
addUserView
«facelet»
addUserView
«action»
Login
«ManagedBean»
Login
«action»
addCustomer
«ManagedBean»
addCustomer
«action»
addTechnician
«ManagedBean»
addTechnician
«action»
addUser
«ManagedBean»
addUser
«action»
findCustomersByEmail
«ManagedBean»
findCustomersByEmail
«action»
findCustomersByName
«ManagedBean»
findCustomersByName
«action»
findCustomersByPhone
«ManagedBean»
findCustomersByPhone
«action»
findTechniciansByDni
«ManagedBean»
findTechniciansByDni
«action»
findTechniciansByEmail
«ManagedBean»
findTechniciansByEmail
«action»
findTechniciansByName
«ManagedBean»
findTechniciansByName
«action»
findTechniciansByPhone
«ManagedBean»
findTechniciansByPhone
«action»
listAllCustomers
«ManagedBean»
listAllCustomers
«action»
listAllTechnicians
«ManagedBean»
listAllTechnicians
«action»
listAllUsers
«ManagedBean»
listAllUsers
«action»
updateCustomer
«ManagedBean»
updateCustomer
«action»
updateTechnician
«ManagedBean»
updateTechnician
«action»
updateUser
«ManagedBean»
updateUser
«component»
frontController
«servlet»
Faces Servlet
«use»

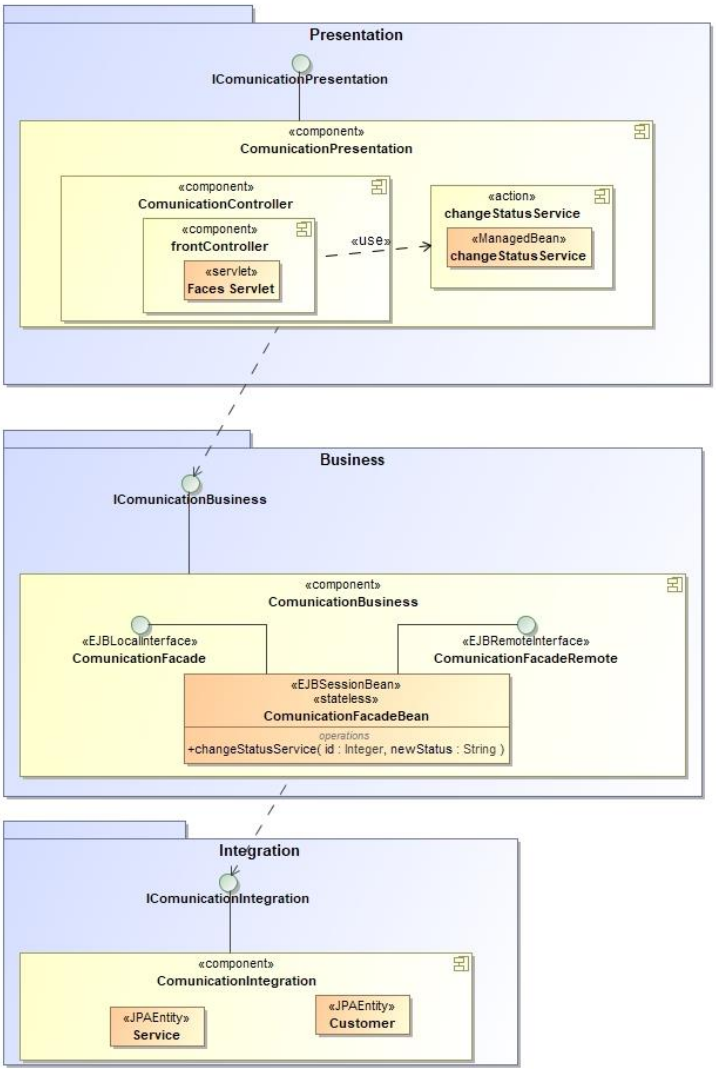


PUNTO VISTA DE LA COMPUTACION – COMPONENTE GESTION:

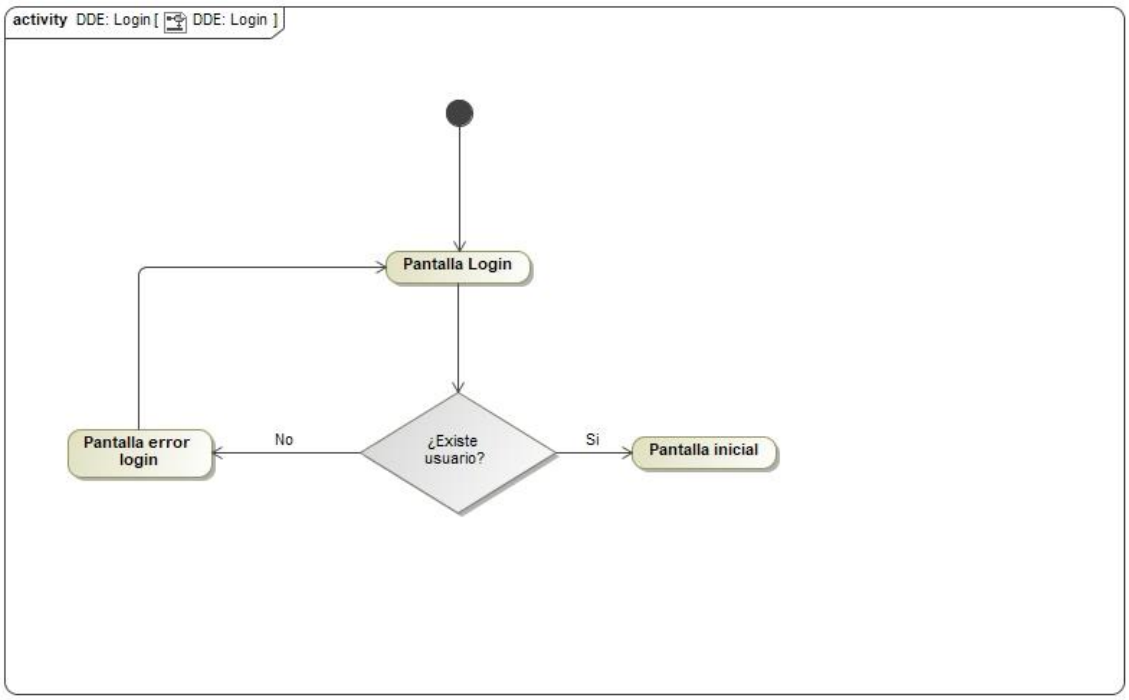

Presentation
IManagementPresentation
«component»
ManagementPresentation
«component»
ManagementPresentationView
«view»
addDevice
«facelet»
addDevice
«view»
addService
«facelet»
addService
«view»
createNewServiceView
«facelet»
createNewServiceView
«view»
deviceListView
«facelet»
deviceListView
«view»
findByNameView
«facelet»
findByNameView
«view»
findByPhoneView
«facelet»
findByPhoneView
«view»
findBySerialView
«facelet»
findBySerialView
«view»
findDeviceByBrandView
«facelet»
findDeviceByBrandView
«view»
findDeviceByModelView
«facelet»
findDeviceByModelView
«view»
findDeviceBySerialView
«facelet»
findDeviceBySerialView
«view»
findDeviceByTypeView
«facelet»
findDeviceByTypeView
«view»
serviceListView
«facelet»
serviceListView
«view»
showServiceView
«facelet»
showServiceView
«component»
ManagementPresentationController
«action»
addDevice
«ManagedBean»
addDevice
«action»
addService
«ManagedBean»
addService
«action»
findDevicesByBrand
«ManagedBean»
findDevicesByBrand
«action»
findDevicesByModel
«ManagedBean»
findDevicesByModel
«action»
findDevicesBySerial
«ManagedBean»
findDevicesBySerial
«action»
findDevicesByType
«ManagedBean»
findDevicesByType
«action»
findServicesByName
«ManagedBean»
findServicesByName
«action»
findServicesByPhone
«ManagedBean»
findServicesByPhone
«action»
findServicesBySerial
«ManagedBean»
findServicesBySerial
«action»
listAllDevices
«ManagedBean»
listAllDevices
«action»
listAllServices
«ManagedBean»
listAllServices
«action»
listDevicesByCustomer
«ManagedBean»
listDevicesByCustomer
«action»
listServicesByCustomer
«ManagedBean»
listServicesByCustomer
«action»
showService
«ManagedBean»
showService
«component»
frontController
«servlet»
Faces Servlet
«use»

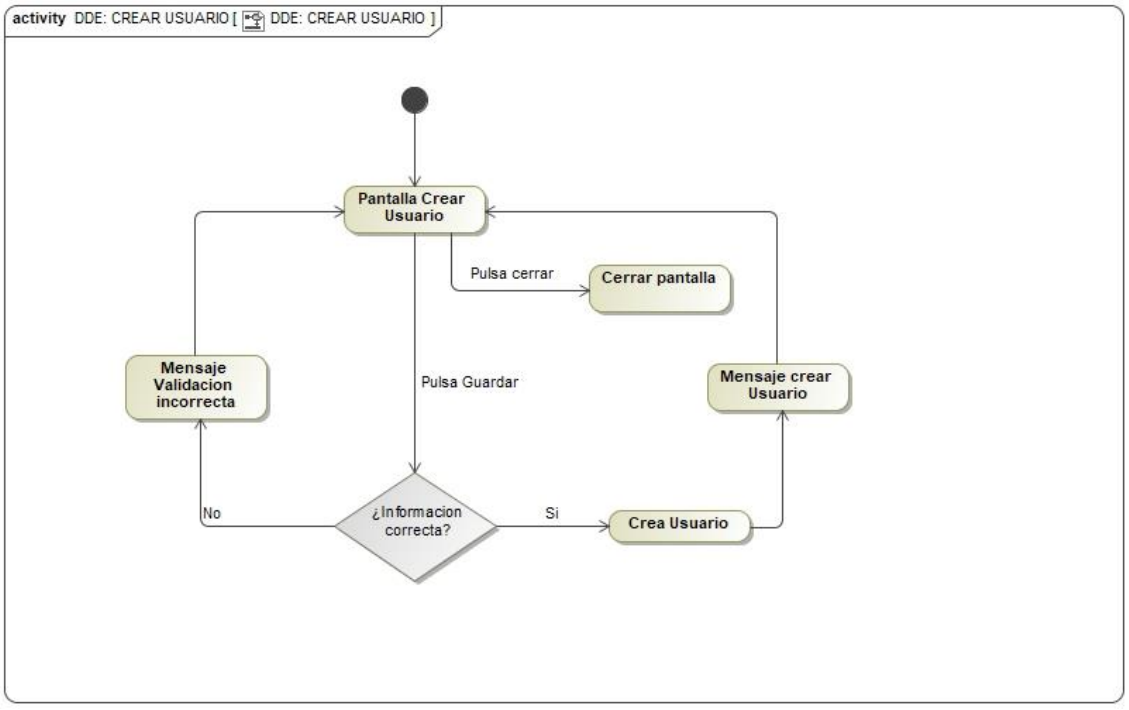

Business
IManagementBusiness
«component»
ManagementBusiness
«EJBLocalInterface»
ManagementFacade
«EJBRemoteInterface»
ManagementFacadeRemote
«EJBSessionBean»
«stateless»
ManagementFacadeBean
operations
+addCustomerToDevice(Customer : Customer, Device : Device)
+addDevice(name : String, model : String, serialNumber : String, type : String, customerId : int) : Device
+deleteDevice(id : Integer)
+listAllDevices() : List<Device>
+listDevicesByCustomer(customerId : int) : List<Device>
+listDevicesWithoutCustomer() : List<Device>
+updateDevice(id : Integer, name : String, model : String, serialNumber : String, type : String)
+findDeviceById(id : Integer) : Device
+listServicesByCustomer(customerId : int) : List<Service>
+addService(checkingDate : date, trackingCode : String, deviceToRepair : String, issue : String, firstSolution : String, internalComment : String, detailReparation : String, pvp : double, status : String, checkOutDate : date, customer : Customer, deviceId : Integer) : Service
+listAllServices() : List<Service>
+updateService(id : Integer, checkingDate : date, trackingCode : String, deviceToRepair : String, issue : String, firstSolution : String, internalComment : String, detailReparation : String, pvp : double, status : String, checkOutDate : date)
+showService(id : Integer) : Service
+findServiceById(id : Integer) : Service
+findServiceByName(name : String) : List<Service>
+findServiceByPhone(phone : String) : List<Service>
+findServiceBySerialNumber(serialNumber : String) : List<Service>
+findDeviceByBrand(brand : String) : List<Device>
+findDeviceByModel(model : String) : List<Device>
+findDeviceBySerial(serialNumber : String) : List<Device>
+findDeviceByType(type : String) : List<Device>
Integration
IManagementIntegration
«component»
ManagementIntegration
«JPAEntity»
User
«JPAEntity»
Admin
«JPAEntity»
Technician
«JPAEntity»
Customer
«JPAEntity»
Device
«JPAEntity»
Service

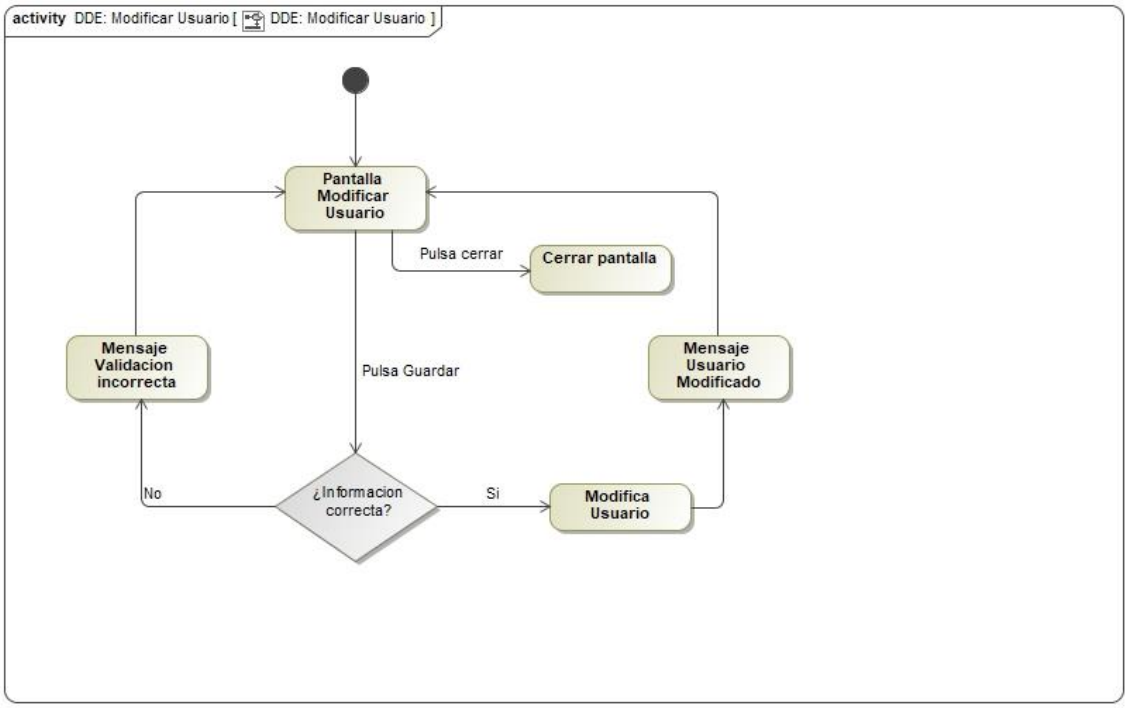
PUNTO DE VISTA DE LA COMPUTACION – COMPONENTE COMUNICACIÓN:

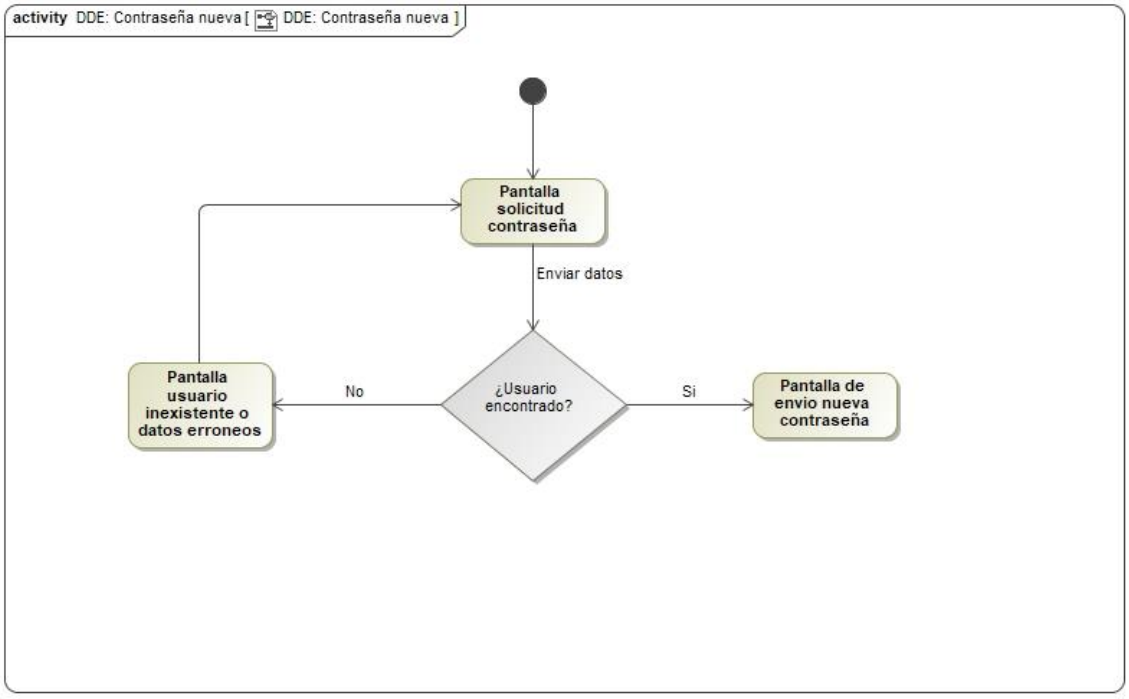


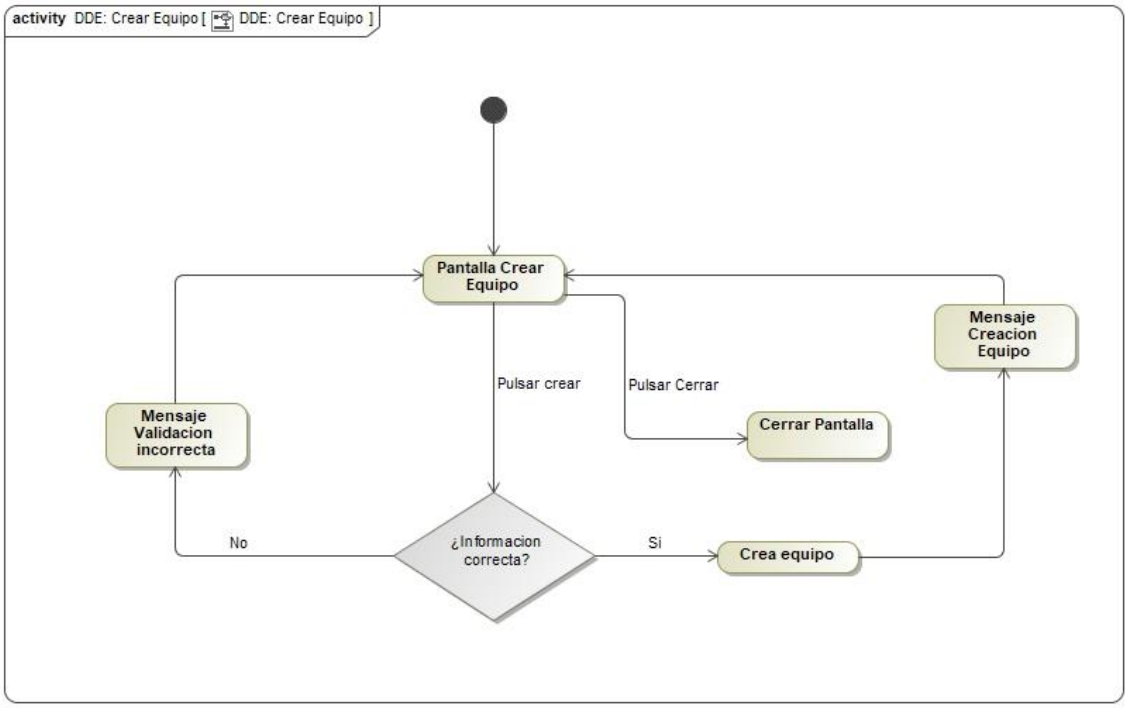
3.4 Diagramas de estado

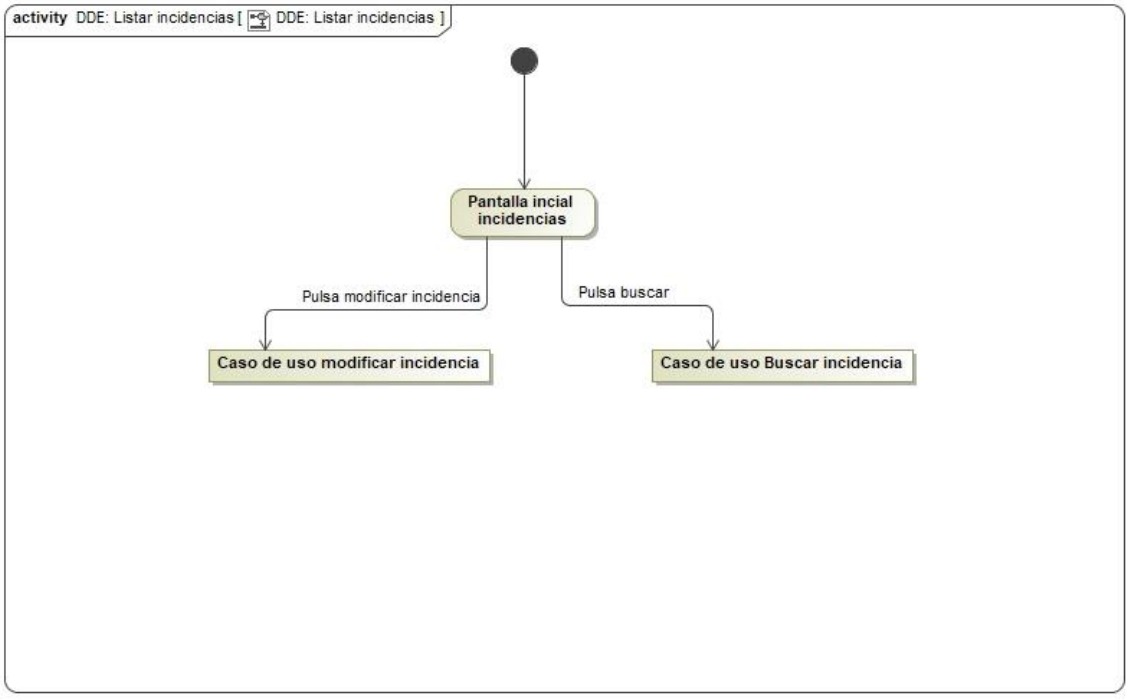

activity DDE: Login [DDE: Login]
Pantalla Login
¿Existe usuario?
No
Si
Pantalla error login
Pantalla inicial

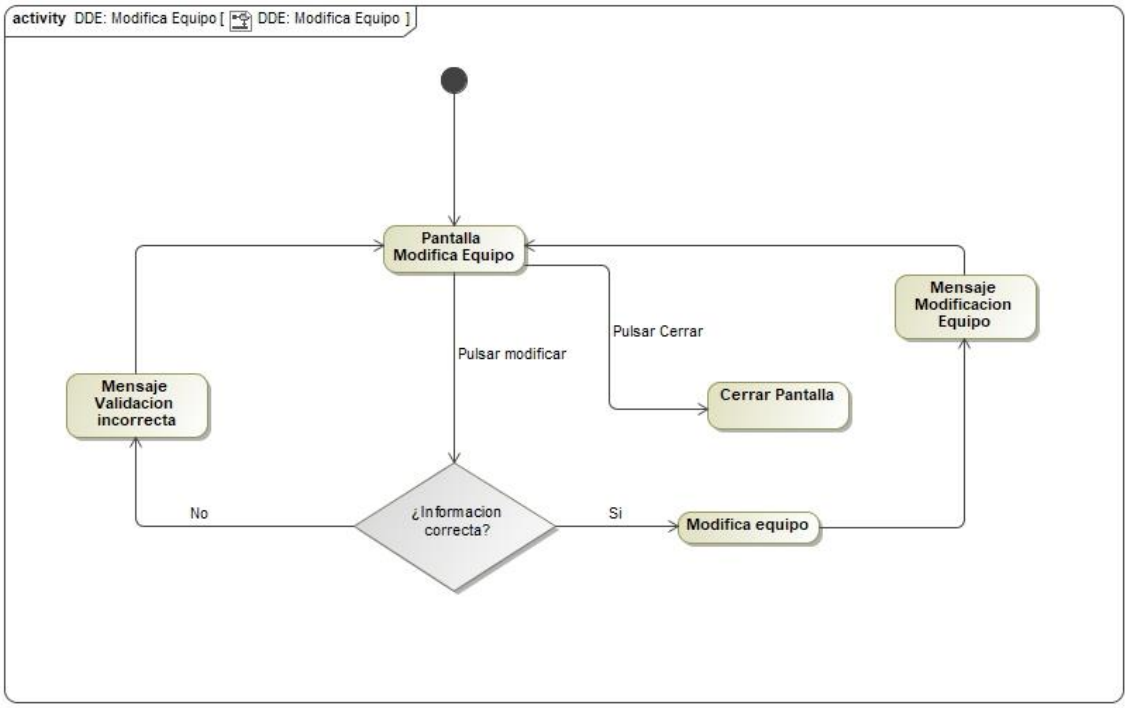

activity DDE: CREAR USUARIO [DDE: CREAR USUARIO]
Pantalla Crear Usuario
Pulsa cerrar
Cerrar pantalla
Mensaje Validacion incorrecta
Pulsa Guardar
Mensaje crear Usuario
No
¿Informacion correcta?
Si
Crea Usuario

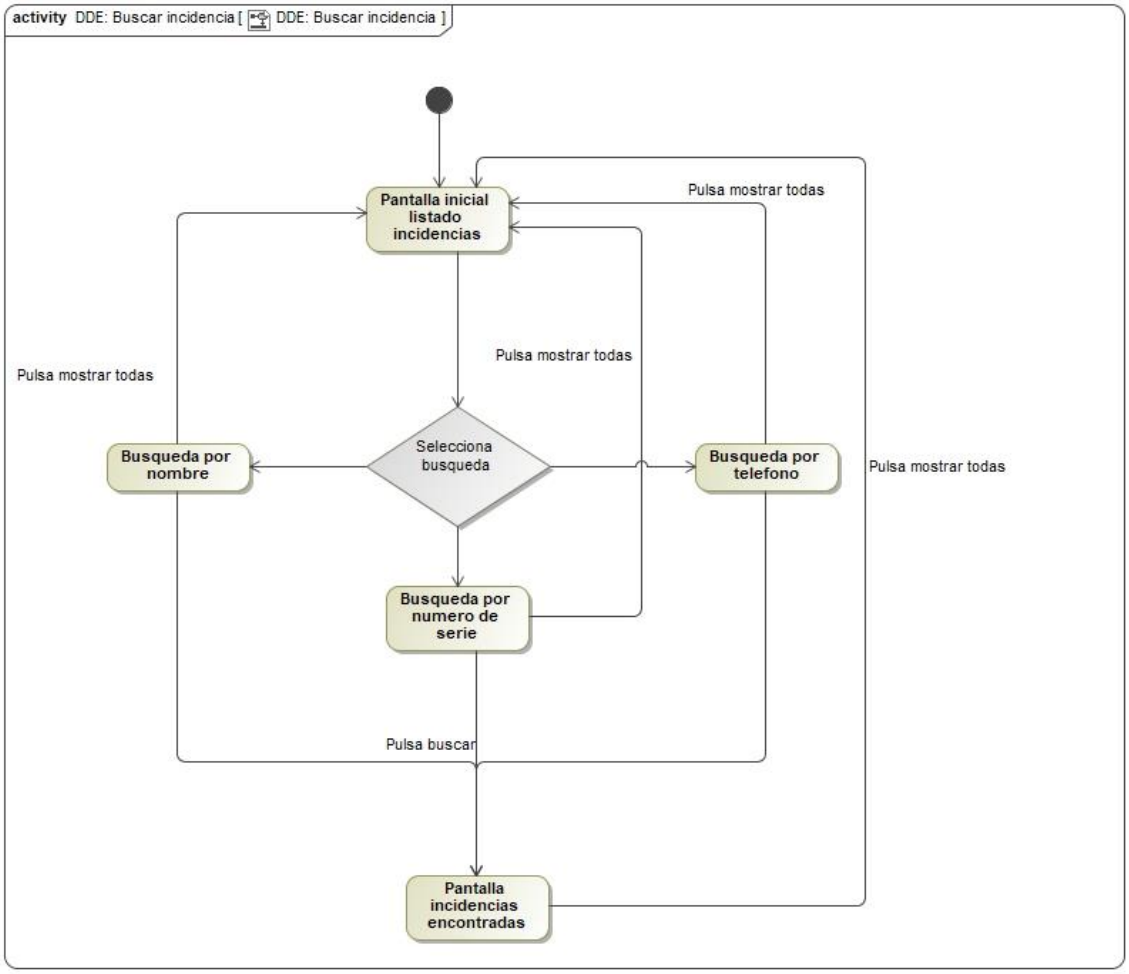

activity DDE: Modificar Usuario [DDE: Modificar Usuario]
Pantalla Modificar Usuario
Pulsa cerrar
Cerrar pantalla
Mensaje Validacion incorrecta
Pulsa Guardar
Mensaje Usuario Modificado
No
¿Informacion correcta?
Si
Modifica Usuario

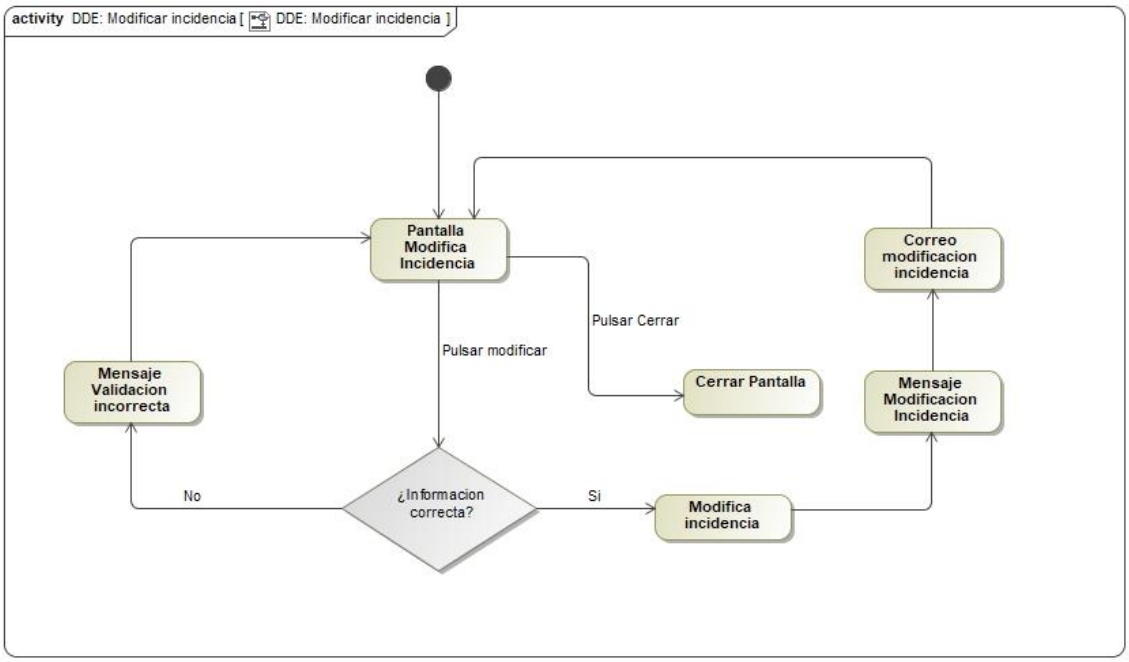

activity DDE: Contraseña nueva [DDE: Contraseña nueva]
Pantalla solicitud contraseña
Enviar datos
Pantalla usuario inexistente o datos erroneos
No
¿Usuario encontrado?
Si
Pantalla de envio nueva contraseña


activity DDE: Crear Equipo [DDE: Crear Equipo]
Pantalla Crear Equipo
Pulsar crear
Pulsar Cerrar
Cerrar Pantalla
Mensaje Creacion Equipo
Mensaje Validacion incorrecta
¿Informacion correcta?
No
Si
Crea equipo


activity DDE: Listar incidencias [DDE: Listar incidencias]
Pantalla incial incidencias
Pulsa modificar incidencia
Pulsa buscar
Caso de uso modificar incidencia
Caso de uso Buscar incidencia


activity DDE: Modifica Equipo [DDE: Modifica Equipo]
Pantalla Modifica Equipo
Pulsar modificar
Pulsar Cerrar
Mensaje Modificacion Equipo
Cerrar Pantalla
Mensaje Validacion incorrecta
No
¿Informacion correcta?
Si
Modifica equipo


activity DDE: Buscar incidencia [DDE: Buscar incidencia]
Pantalla inicial listado incidencias
Pulsa mostrar todas
Pulsa mostrar todas
Pulsa mostrar todas
Busqueda por nombre
Selecciona busqueda
Busqueda por telefono
Pulsa mostrar todas
Busqueda por numero de serie
Pulsa buscar
Pantalla incidencias encontradas



3.5 Diseño relacional de la base de datos

Partiendo del diseño conceptual de la base de datos se aplica la transformación del diseño conceptual al diseño relacional, verificando el cumplimiento de las formas normales para bases de datos:

*User (id, **name, email, active, password**)*

*Administrator (id, **name, email, active, password, dni**)*

*Technician (id, **name, email, active, password, dni, phone**)*
*Customer (id, **name, email, active, password, phone**)*

*Message (id, **subject, text, readed, creationDate, readedDate, serviceId, customerUserId**)*

{*serviceId*} es clave ajena de *service*.
{*customerUserId*} es clave ajena de *customer*.

Device (id, **name, model, serialNumber, type, userId**)

{*userId*} es clave ajena de *user*.

Service (id, **checkingDate, trackingCode, deviceToRepair, issue, firstSolution, internalComment, detailReparation, pvp, status, checkoutDate, deviceId, customerUserId, technicianUserId**)

{*deviceId*} es clave ajena de *device*.

{*customerUserId*} es clave ajena de *customer*.
{*technicianUserId*} es clave ajena de *technician*.

MessageUser (user, message)

{*user*} es clave ajena de *user*.
{*message*} es clave ajena de *message*.

DeviceCustomer(device, customer)

{*device*} es clave ajena de *device*.
{*customer*} es clave ajena de *customer*.

DeviceService(device, service)

{*device*} es clave ajena de *device*.
{*service*} es clave ajena de *service*.

TechnicianService (technician, service)

{*technician*} es clave ajena de *technician*.
{*service*} es clave ajena de *service*.

CustomerService (customer, service)

{*customer*} es clave ajena de *customer*.
{*service*} es clave ajena de *service*.

Ahora se comprueba que el diseño relacional cumple las formas normales.

La primera forma normal (1FN) indica que todos los atributos de una relación son atómicos, es decir, no son otra relación, ni son descomponibles, ni tienen multiplicidad de valores.

Al analizar todas las relaciones se concluye que todos los atributos son atómicos, por tanto, cumple la 1FN.

La segunda forma normal (2FN) concluye que se cumple esta forma normal, si y solo si, se cumple la 1FN y además todo atributo que no forma parte de una clave candidata depende completamente de todas las claves candidatas de la relación.

Al analizar todas las relaciones se concluye que todos los atributos no pertenecientes a claves candidatas dependen de todas las claves candidatas de la relación, por tanto, cumple la 2FN.

La tercera forma normal (3FN) concluye que cumple esta forma normal si, y solo si, cumple la 2FN y además y ningún atributo que no forma parte de una clave candidata depende de un conjunto de atributos que contiene alguno que forma parte de una clave candidata.

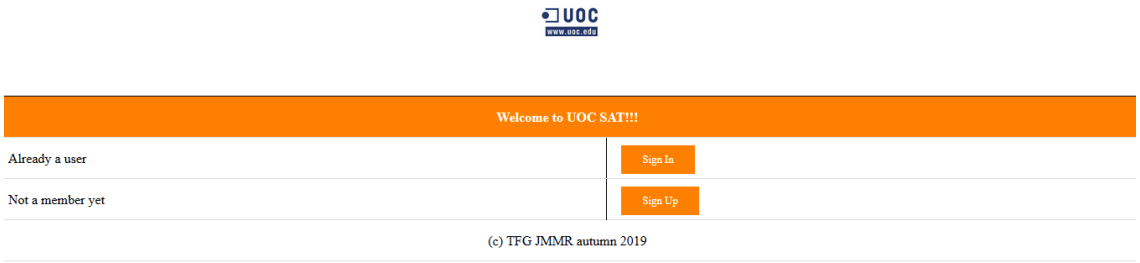
Al analizar todas las relaciones se concluye que todos los atributos que no forman parte de una clave candidata depende de un conjunto de atributos que contiene alguno que forma parte de una clave candidata, por tanto, se cumple la 3FN.

Por tanto, el diseño relacional alcanzado cumple las tres primeras formas normales.

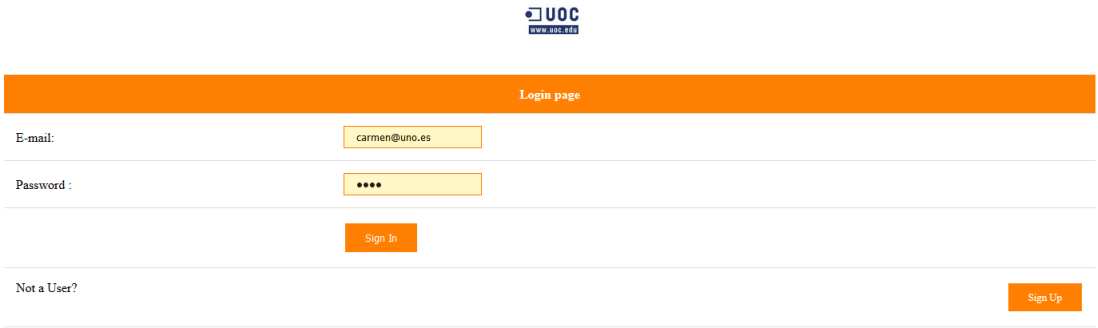
3.6 Interfaz gráfica de usuario

Pantallas de la aplicación.

Pantalla de bienvenida:

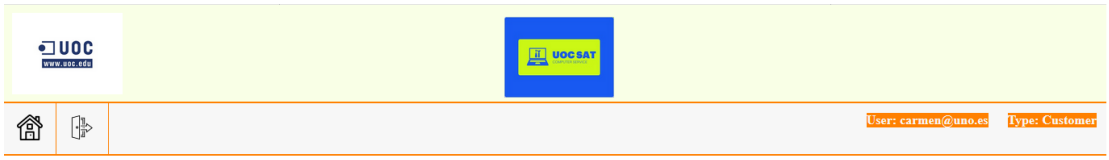


Pantalla de inicio de sesión:



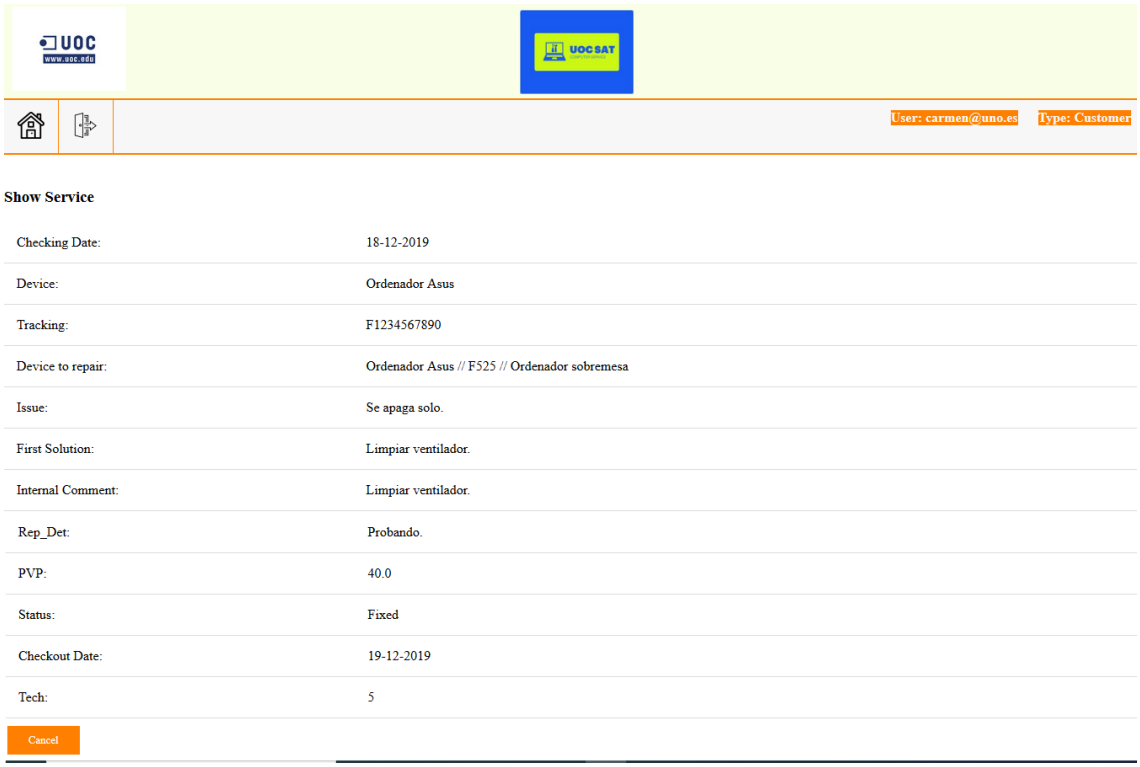
Pantalla de registro de un usuario nuevo:

Pantalla principal de un usuario tipo cliente:

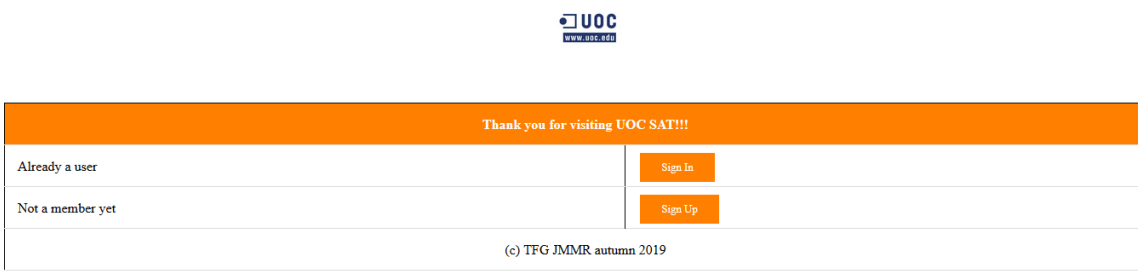


Checking Date	Device	Issue	Serial	Tracking Code	PVP	
18/12/2019	Ordenador Asus // F525 // Ordenador sobremesa	Se apaga solo.	F11111	F1234567890	40.0	
15/12/2019	Ordenador Asus // F525 // Ordenador sobremesa	No enciende.	F11111	F1234567890	40.0	
09/09/2018	Ordenador sobremesa negro Asus	NO ENCIENDE	F11111	SD122244KD	50.0	

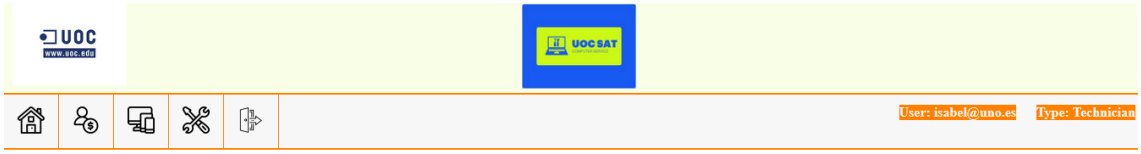
Pantalla detalle de un servicio desde el cliente afectado:



Pantalla de cierre de sesión de un cliente tipo cliente:

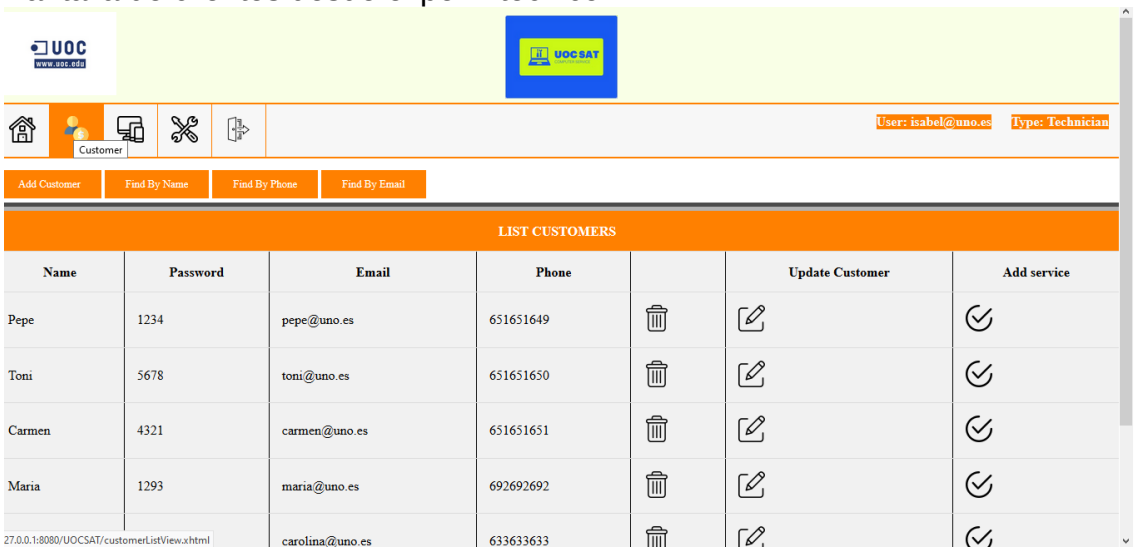


Pantalla inicial de un cliente de tipo técnico:

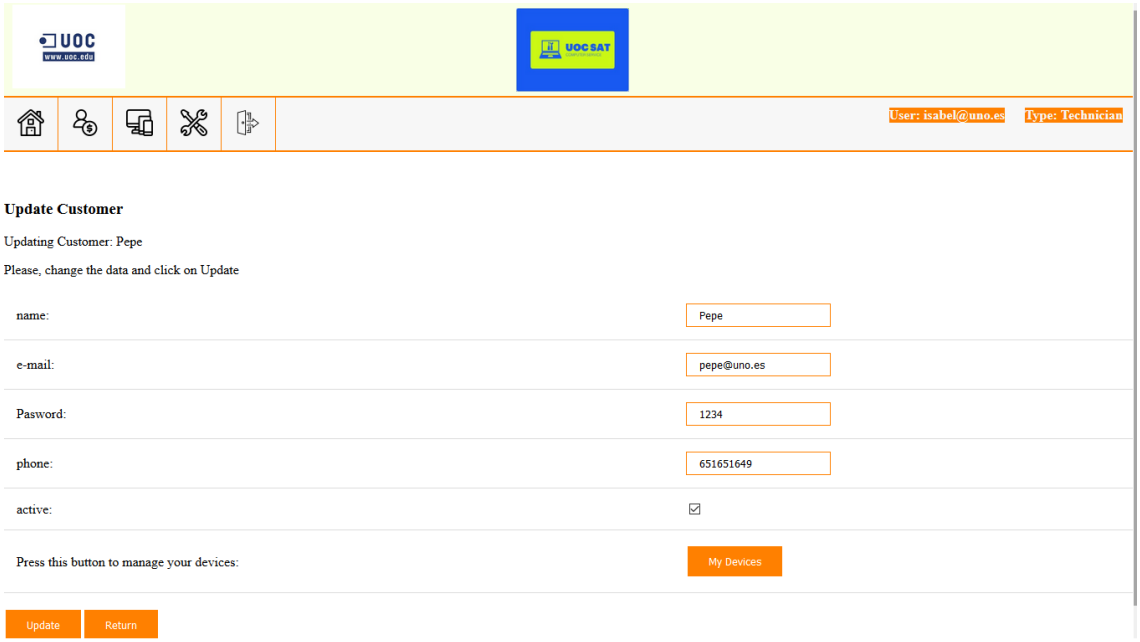


ID	Entrada	Cliente	Phone	Device	Serial	Seguimiento	PVP
5	19/12/2019	roberto	965710497	MSI // Dimension 1450 // Ordenador portatil	FR350	F1234567890	90.0
6	18/12/2019	Carmen	651651651	Ordenador Asus // F525 // Ordenador sobremesa	F11111	F1234567890	40.0
4	15/12/2019	Carmen	651651651	Ordenador Asus // F525 // Ordenador sobremesa	F11111	F1234567890	40.0
3	11/10/2018	Carolina	633633633	Ordenador portatil negro Asus	P3333333	929k949ad4	50.0
2	09/10/2018	Maria	692692692	Ordenador portatil negro Asus	FG2222	93848jf9k	50.0

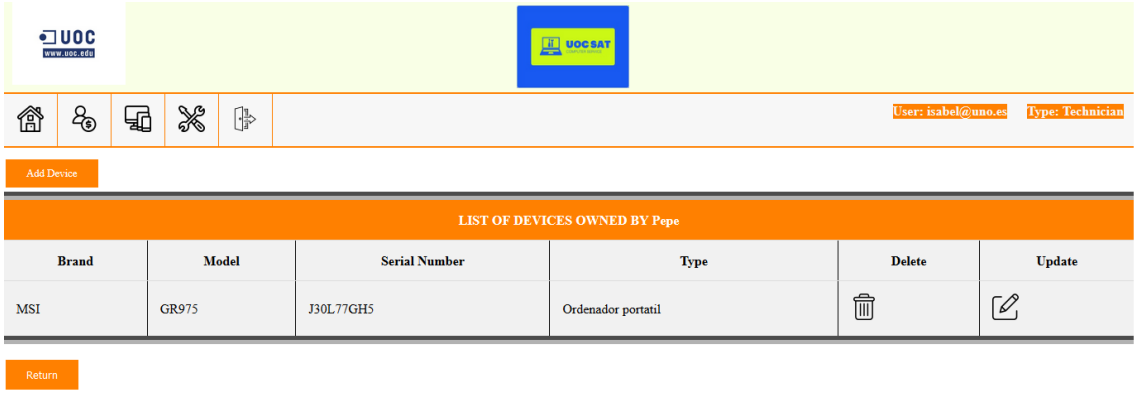
Pantalla de clientes desde el perfil técnico:



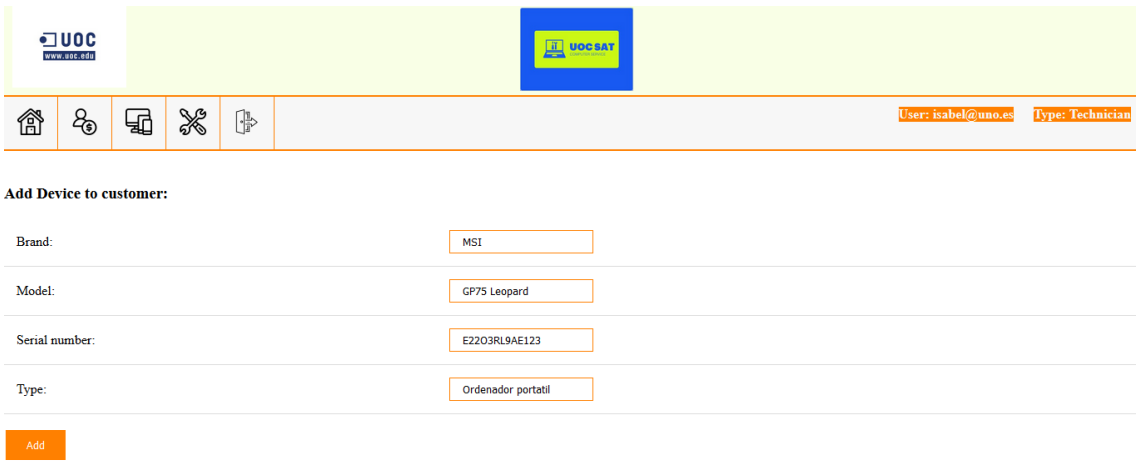
Pantalla de actualización de clientes:



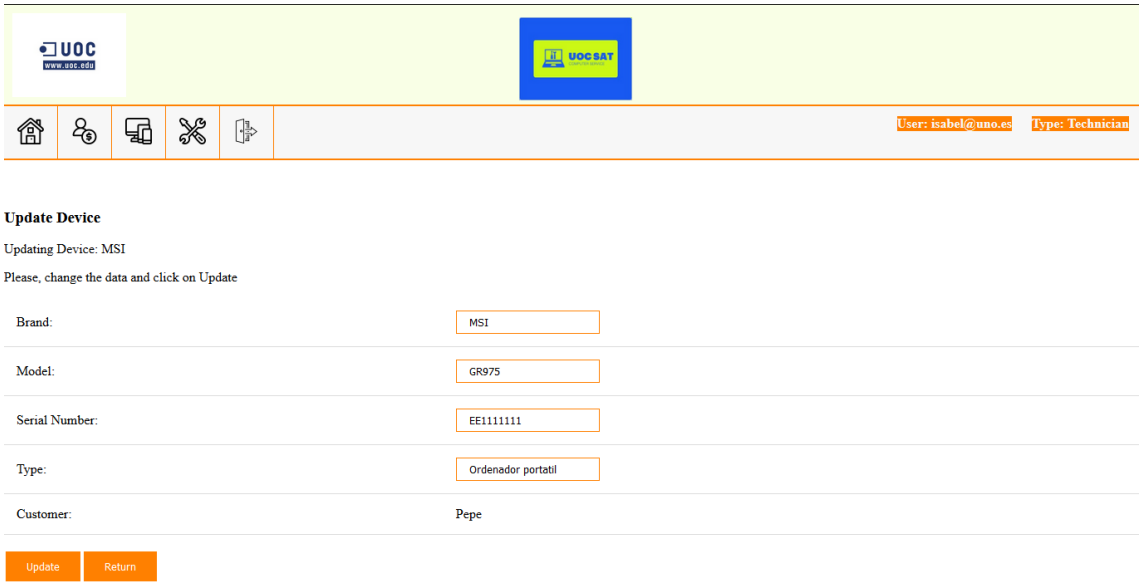
Pantalla de dispositivos de un cliente en concreto:



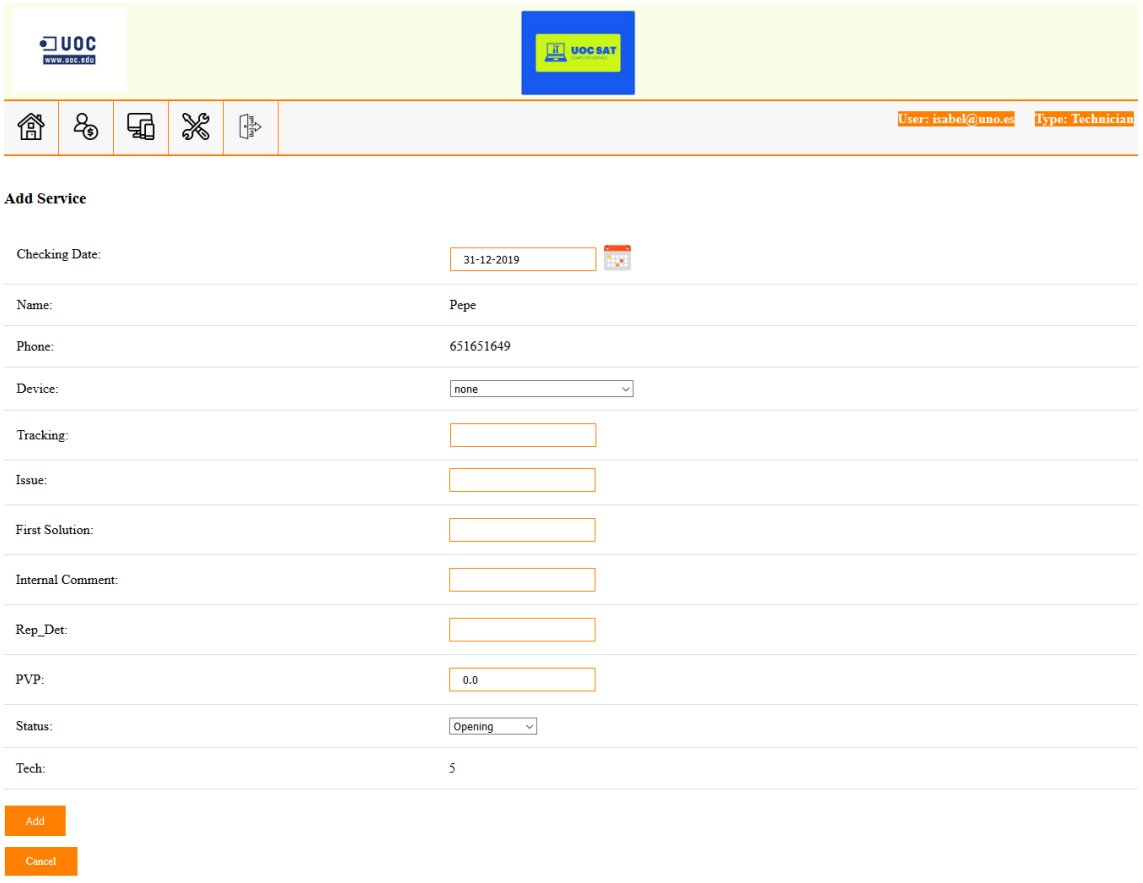
Pantalla de agregar un dispositivo a un cliente concreto:



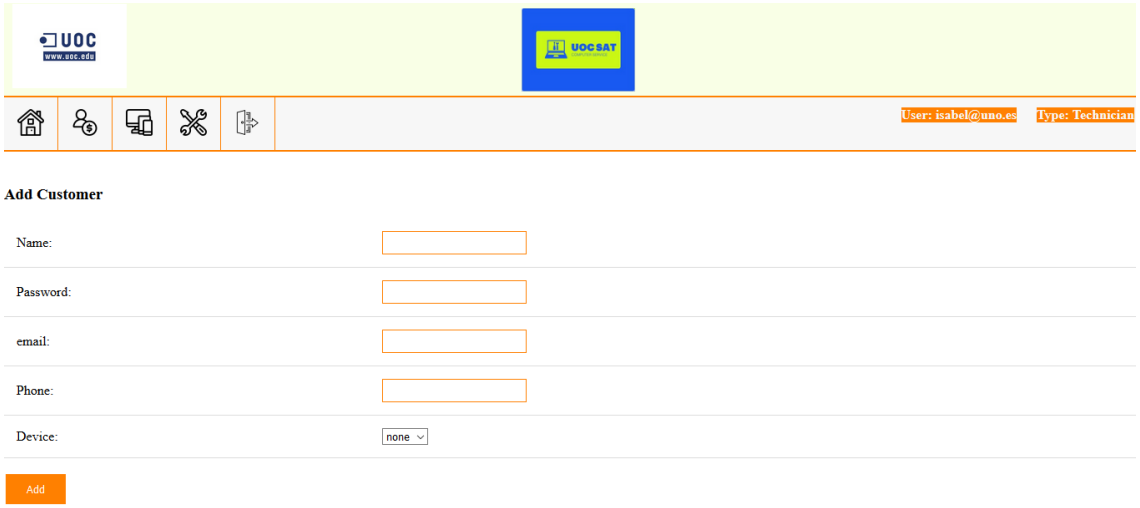
Pantalla para actualizar / modificar un dispositivo de un cliente:



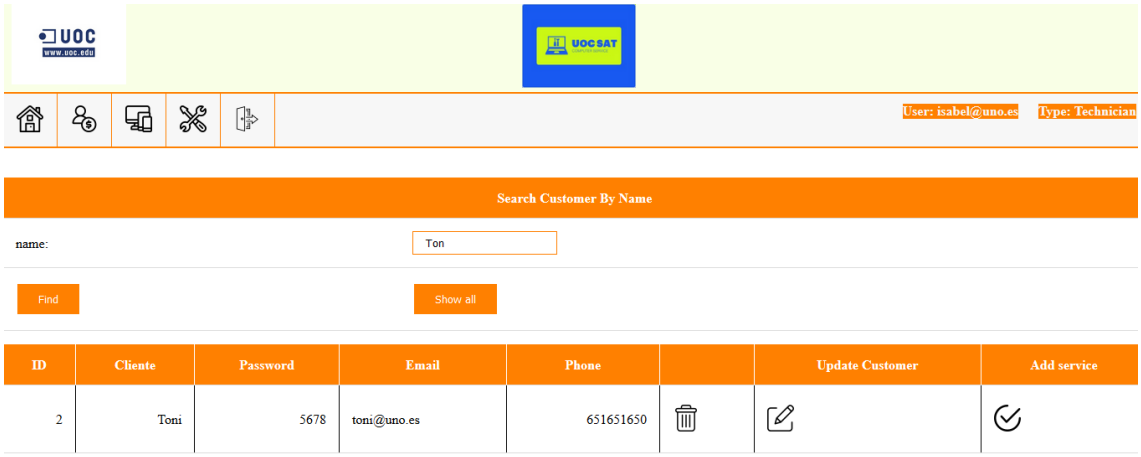
Pantalla para agregar un servicio a un cliente concreto:



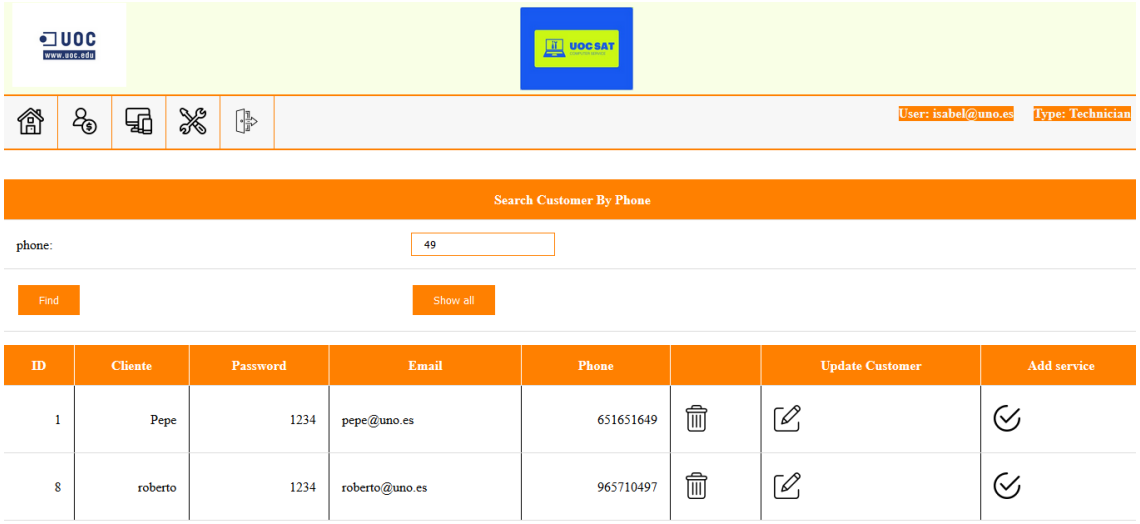
Pantalla para registro de un cliente nuevo:



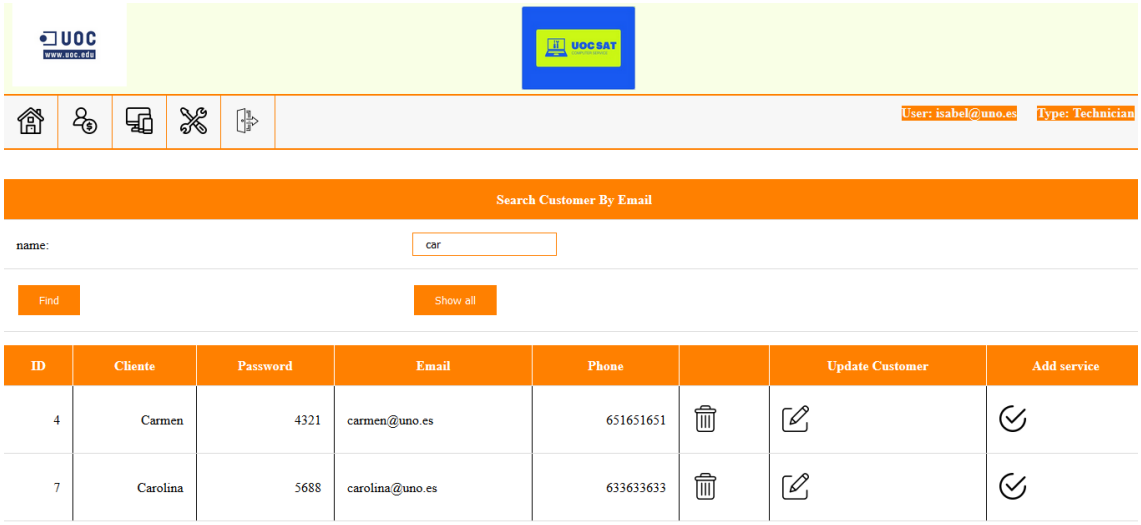
Pantalla de búsqueda de un cliente por nombre:



Pantalla para búsqueda de un cliente por teléfono:

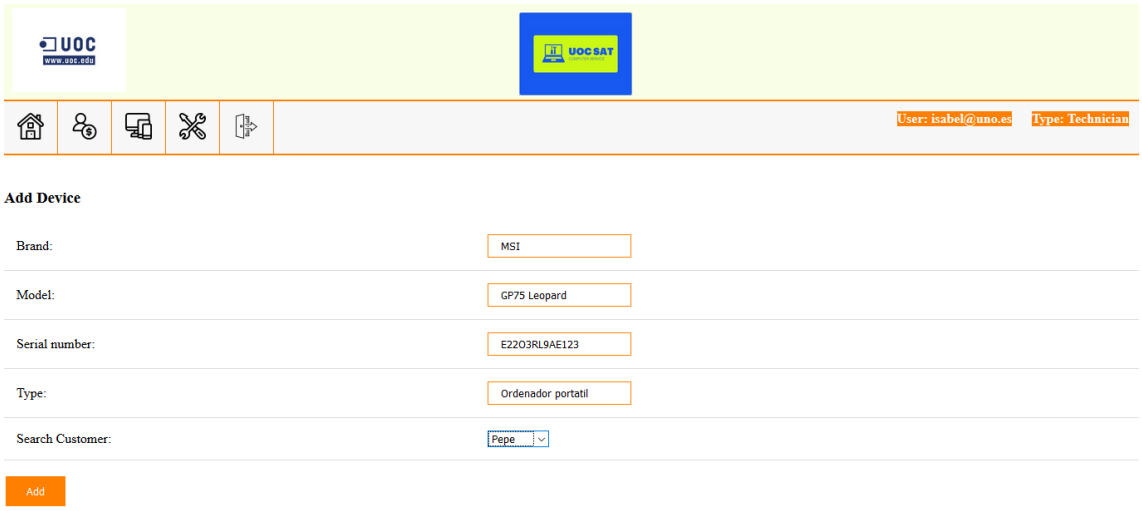


Pantalla de búsqueda de un cliente por e-mail:

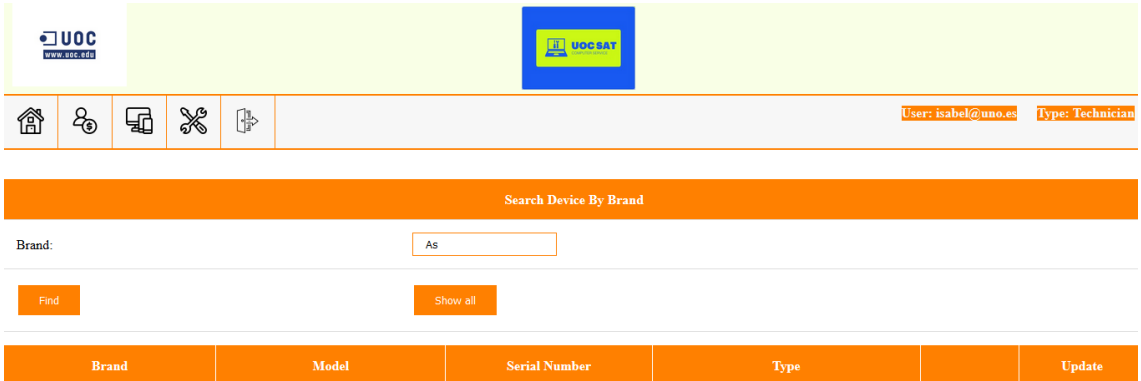


Pantalla de dispositivos para un técnico:

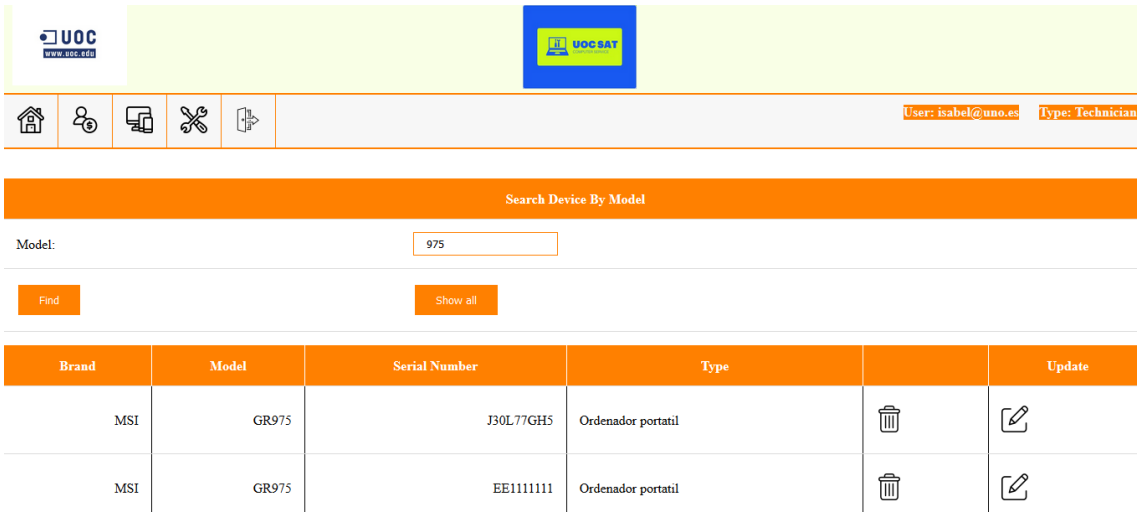
Pantalla de agregar un dispositivo a un cliente del listado de clientes:



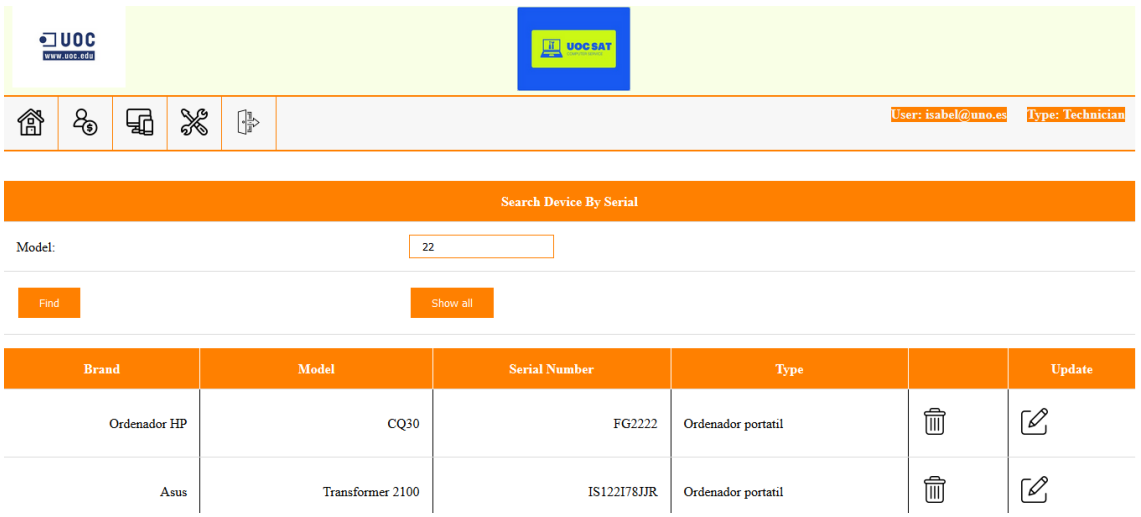
Pantalla de búsqueda de dispositivos por marca:



Pantalla de búsqueda de un dispositivo por modelo:

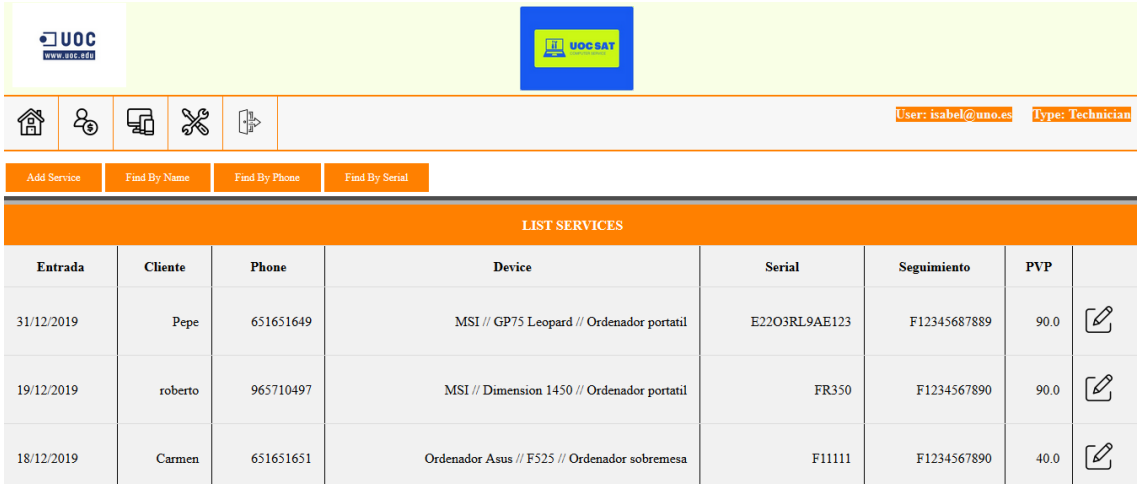


Pantalla de búsqueda de un dispositivo por número de serie:

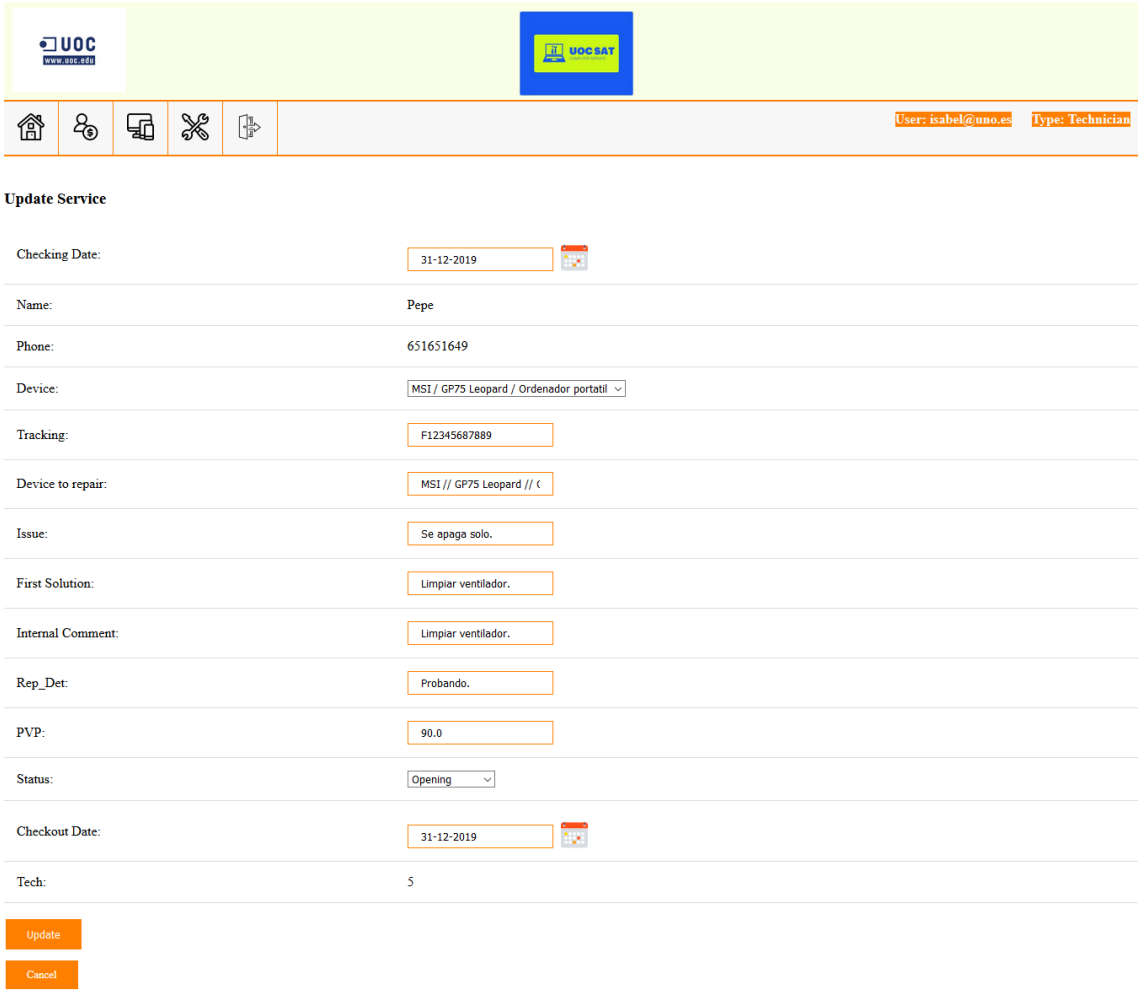


Pantalla de búsqueda de dispositivos por tipo:

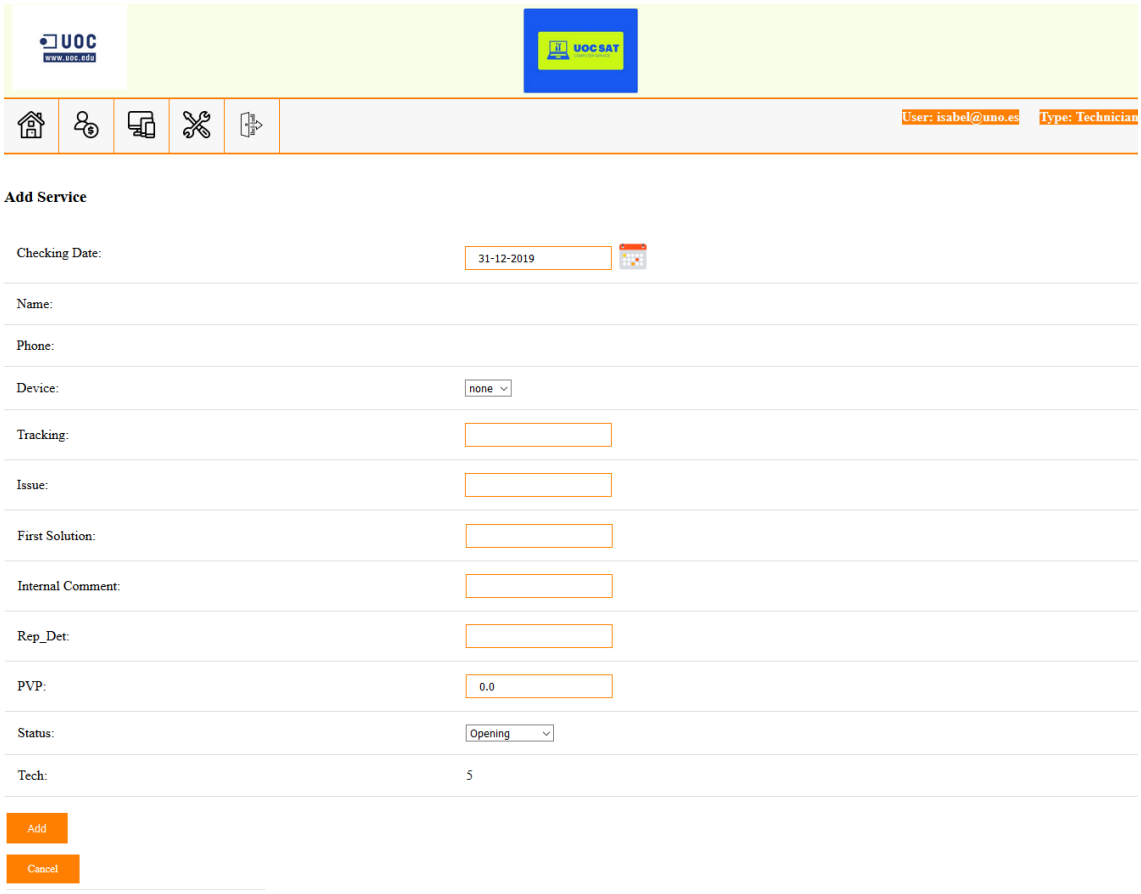
Pantalla de servicios:



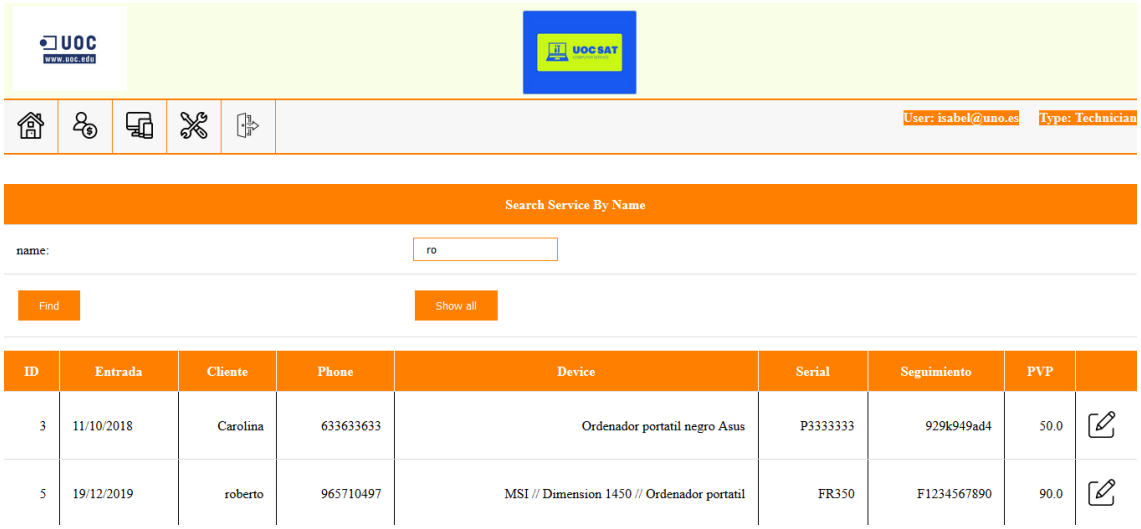
Pantalla de actualización de un servicio:



Pantalla de agregar un servicio a un cliente del listado de clientes:

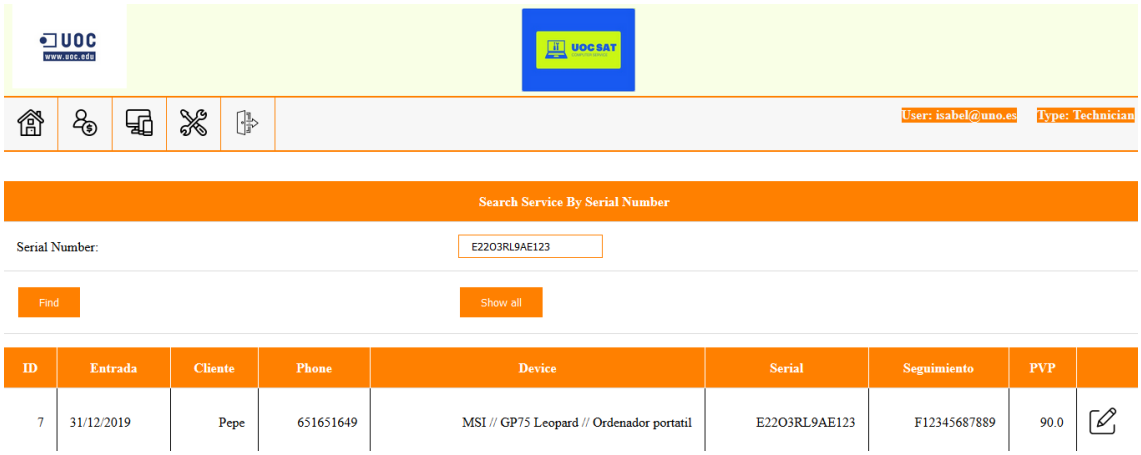


Pantalla de búsqueda de un servicio por nombre de cliente:

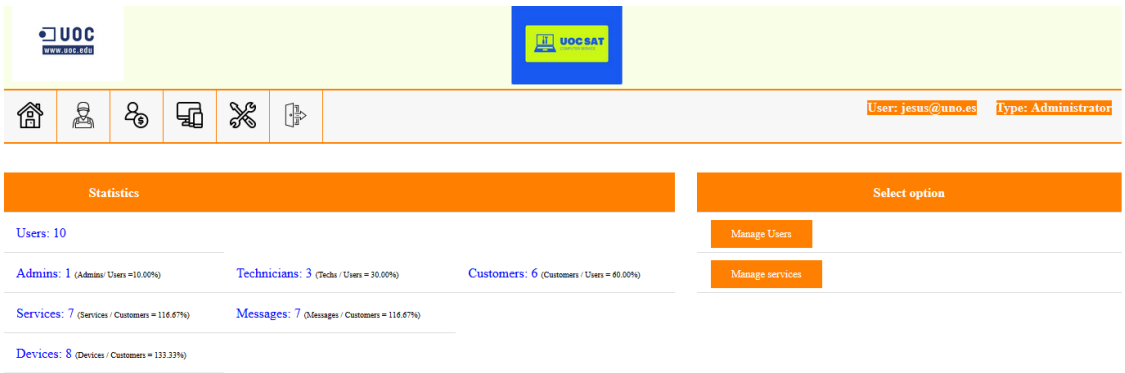


Pantalla de búsqueda de un servicio por teléfono:

Pantalla de búsqueda de un servicio por número de serie:



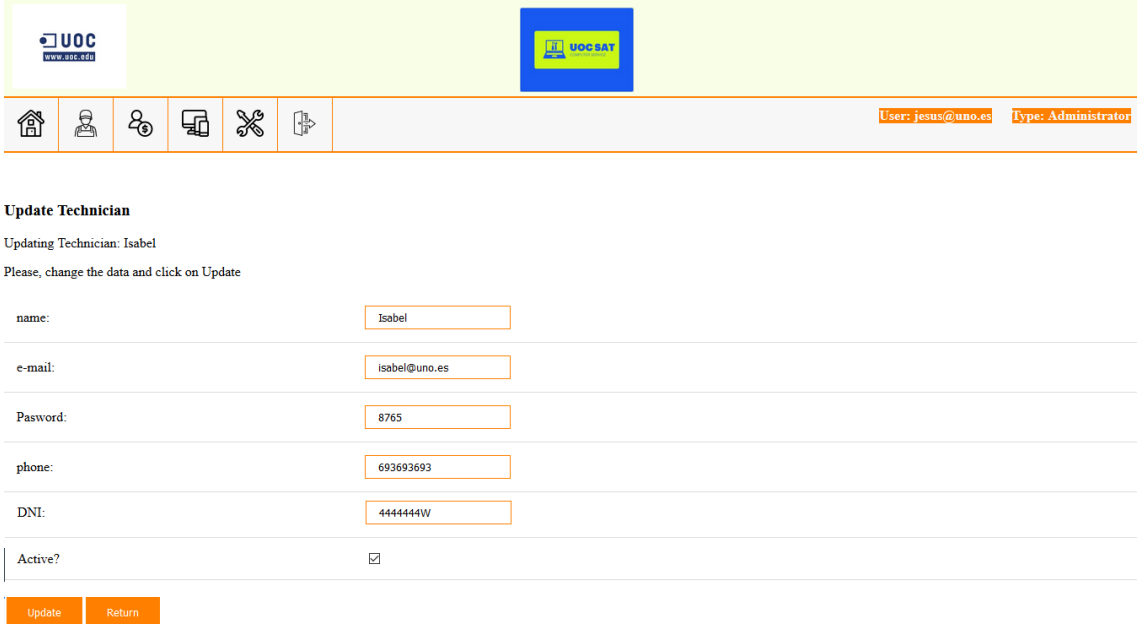
Pantalla inicial de un usuario de tipo administrador:



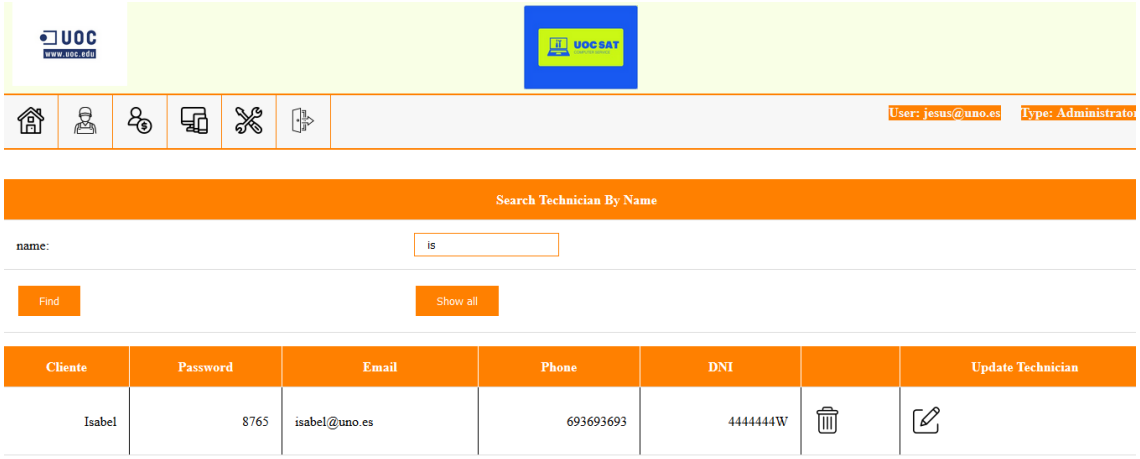
Pantalla de técnicos desde un perfil administrador:

Pantalla para agregar un técnico:

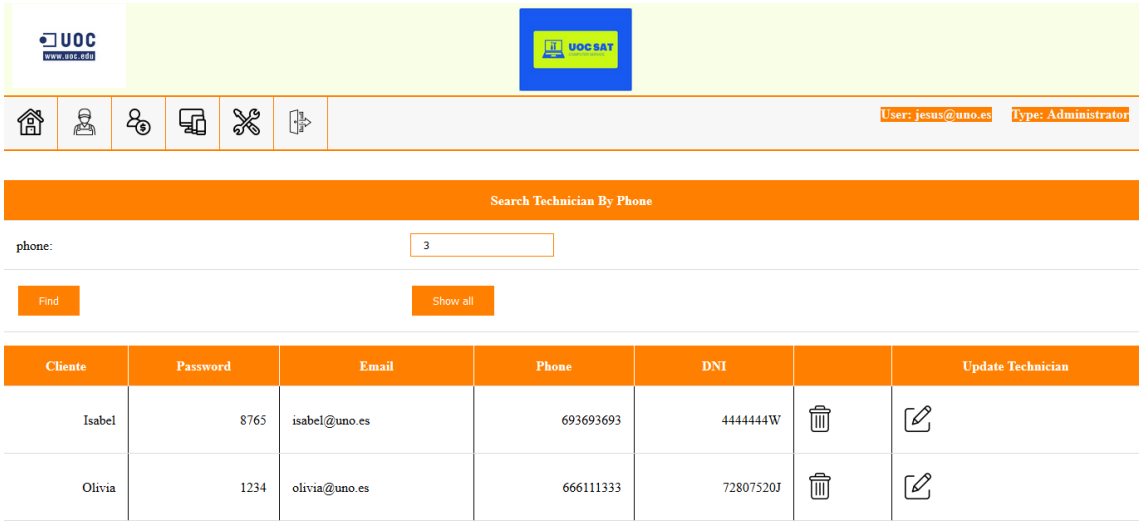
Pantalla para actualizar datos de un técnico:



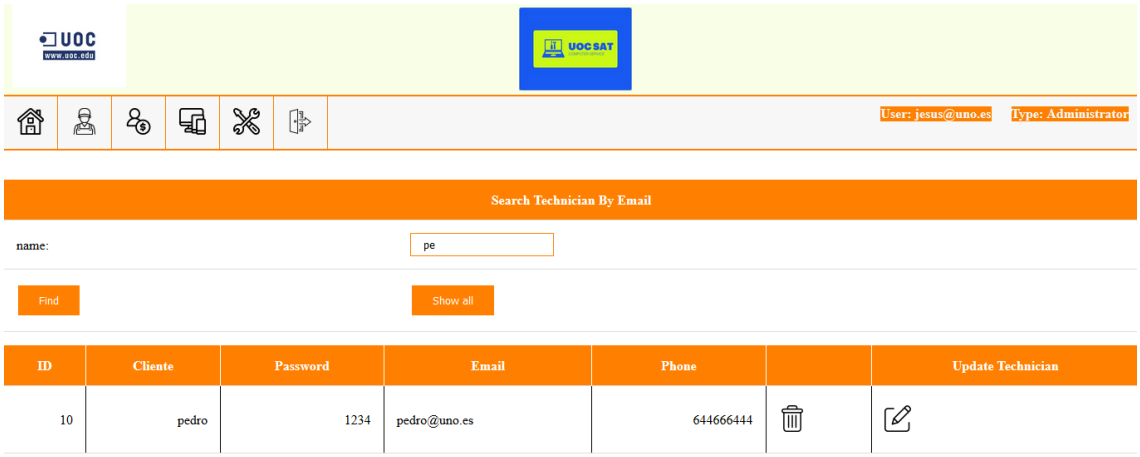
Pantalla para búsqueda de un técnico por nombre:



Pantalla de búsqueda de un técnico por teléfono:



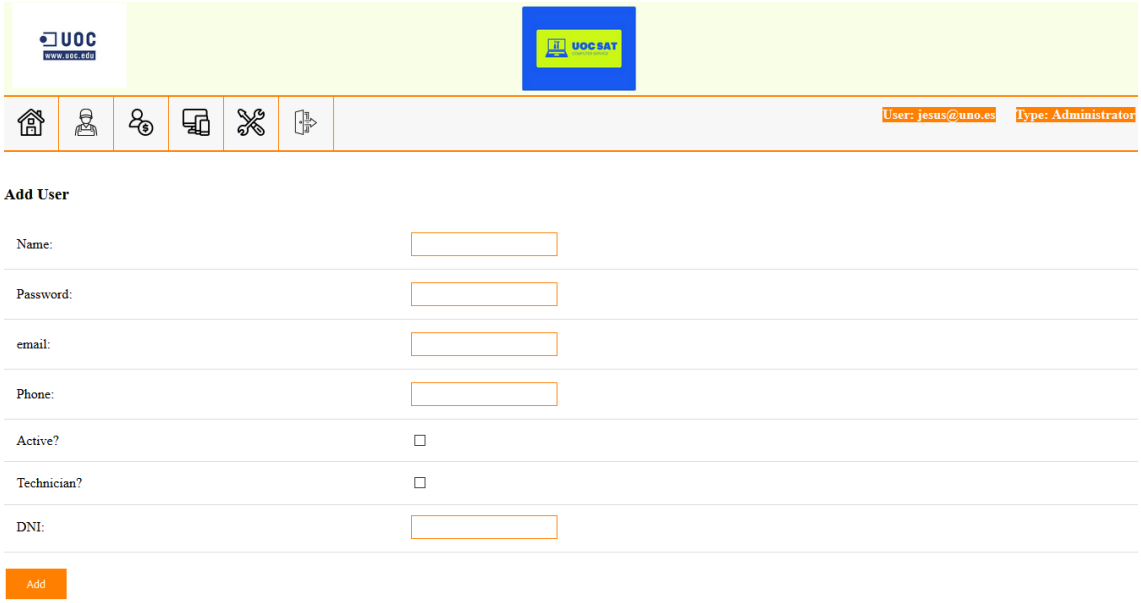
Pantalla de búsqueda de un técnico por email:



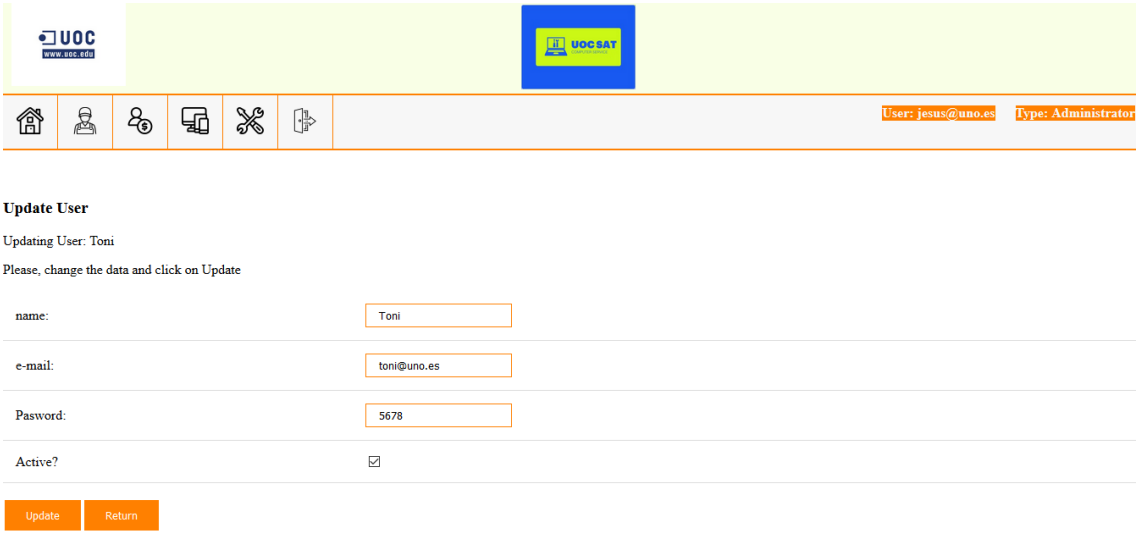
Pantalla de búsqueda de un técnico por dni:

Pantalla principal de usuarios desde el perfil administrador:

Pantalla para agregar un usuario desde el perfil administrador:



Pantalla de actualización de datos de un usuario:



4. Implementación

Todo proyecto JavaEE, incluido el que nos ocupa, deben poseer los siguientes componentes en su estructura:

- src: carpeta en la que se alojan los códigos fuente de las clases Java, interfaces, etc., que conforman la aplicación. Las subcarpetas que contienen son:
  - ejb: paquete que contiene las clases EJB que conforman la capa de negocio de la aplicación.
  - jpa: paquete que contiene las clases que representan las entidades JPA y que constituyen la capa de integración.
  - *managedBean*: paquete que contiene las clases Java que recogen la interacción del usuario con la interfaz de la aplicación, forman parte de la capa de presentación.
  - META-INF: carpeta que contiene los ficheros application.xml y persistence.xml, el primero nos indica cómo se estructura la aplicación al compilarse para el servidor de aplicaciones y el segundo configura las opciones de persistencia para el gestor de la base de datos creando tablas o validando el esquema de la base de datos.
- Docroot: carpeta en la que podemos encontrar los recursos necesarios para la interfaz de usuario de la aplicación. Contiene las imágenes, hojas de estilo y JavaScript necesario para conformar la interfaz, así como los ficheros xhtml que forman las vistas o páginas que el usuario puede visitar en la aplicación. Las subcarpetas que contienen son:
  - *Resources*: la forman las carpetas siguientes en las que se contienen recursos para la interfaz:
    - css: contiene las hojas de estilo
    - *images*: imágenes de la aplicación.
    - js: JavaScript necesario para la correcta funcionalidad de la interfaz.

- WEB-INF
  - Faces-config.xml: contiene la configuración necesaria para JSF.
  - Web.xml: fichero con las opciones de configuración para el despliegue de la capa de presentación de la aplicación, donde por ejemplo se indica el controlador *Faces Servlet*, o cual es la página de inicio de la aplicación.

4.1 Vistas (Capa de presentación)

Para las vistas se ha definido como punto de entrada a la aplicación web la vista home.xhtml.

Las vistas se han definido en base a las funcionalidades de la aplicación. Se ha procurado seguir fielmente el diseño inicial de la aplicación al definir las vistas, aunque en algunos casos se ha necesitado de vistas adicionales o se ha considerado más adecuado otra disposición o nombre de vista al definido originalmente.

En este punto cabe destacar que se ha definido una plantilla (headerView.xhtml) para los elementos comunes a todas las vistas, como por ejemplo el menú principal, las imágenes tanto de la aplicación como de la UOC, etc. Esto se puede considerar como reutilización de código salvando las distancias con la herencia de clases. Esta estrategia ha resultado muy práctica porque al necesitar cambiar la imagen de la aplicación en la plantilla automáticamente se ha modificado en todas las vistas que la usaban.

Había planeado la aplicación de un *framework* para la capa de presentación, pero debido al escaso tiempo y el poco uso que le iba a dar he decidido usar JQuery para un par de funcionalidades como es el selector de fechas de las vistas para agregar y actualizar servicios (*dateTimePicker*) y el cálculo de estadísticas en la vista principal del perfil administrador. Sé que es un elemento muy valioso a tener en cuenta en proyectos más grandes y que requieran de una mejor presentación pero por la curva de aprendizaje y la referida escasez de tiempo se ha optado por no usar ningún *framework* y realizar esas funcionalidades no incluidas en JSF con las anteriormente nombradas estrategias.

4.2 *Managed Bean* (Capa de presentación)

Las clases Java *Managed Bean* representan el modelo del patrón MVC, se comunican con el usuario a través del controlador implícito en la tecnología JSF (*Faces Servlet*) recogiendo la interacción del usuario con la interfaz de la aplicación y respondiendo al mismo por las acciones realizadas.

Al implementar estas clases he logrado comprender y profundizar en una mejor interacción entre las vistas y el modelo, haciendo más eficiente y permitiendo la reutilización de vistas entre los distintos perfiles de la aplicación.

En esta capa se han definido las clases Java previstas y que se exponen a continuación:

- *addCustomerMBean*: clase Java que define la funcionalidad de añadir un usuario de tipo cliente a la aplicación.
- *addDeviceMBean*: clase Java que define la funcionalidad de agregar un dispositivo a la aplicación.
- *addServiceMBean*: clase Java que define la funcionalidad de agregar un servicio a la aplicación.
- *addTechnicianMBean*: clase Java que define la funcionalidad de agregar un usuario de tipo técnico a la aplicación.
- *addUserMBean*: clase Java que define la funcionalidad de agregar un usuario de tipo usuario a la aplicación.
- *findCustomersByEmailMBean*: clase Java que se encarga de la búsqueda de usuarios de tipo cliente filtrando por su atributo email.
- *findCustomersByNameMBean*: clase Java que se encarga de la búsqueda de usuarios de tipo cliente filtrando por su atributo nombre.
- *findCustomersByPhoneMBean*: clase Java que se encarga de la búsqueda de usuarios de tipo cliente filtrando por su atributo teléfono.
- *findDevicesByBrandMBean*: clase Java que se encarga de la búsqueda de dispositivos filtrando por su atributo marca.
- *findDevicesByModelMBean*: clase Java que se encarga de la búsqueda de dispositivos filtrando por su atributo modelo.
- *findDevicesBySerialMBean*: clase Java que se encarga de la búsqueda de dispositivos filtrando por su atributo número de serie.
- *findDevicesByTypeMBean*: clase Java que se encarga de la búsqueda de dispositivos filtrando por su atributo tipo.
- *findServicesByNameMBean*: clase Java que se encarga de la búsqueda de servicios filtrando por su atributo nombre del cliente derivado del dispositivo sobre el que se realiza el servicio.
- *findServicesByPhoneMBean*: clase Java que se encarga de la búsqueda de servicios filtrando por su atributo teléfono del cliente derivado del cliente sobre el que se realiza el servicio.
- *findServicesBySerialMBean*: clase Java que se encarga de la búsqueda de servicios filtrando por su atributo número de serie derivado del dispositivo sobre el que se realiza el servicio.
- *findTechniciansByDniMBean*: clase Java que se encarga de la búsqueda de técnicos filtrando por su atributo dni.
- *findTechniciansByEmailMBean*: clase Java que se encarga de la búsqueda de técnicos filtrando por su atributo email.
- *findTechniciansByNameMBean*: clase Java que se encarga de la búsqueda de técnicos filtrando por su atributo nombre.
- *findTechniciansByPhoneMBean*: clase Java que se encarga de la búsqueda de técnicos filtrando por su atributo teléfono.
- *listAllCustomersMBean*: clase Java que se encarga de listar todos los usuarios de tipo cliente.
- *listAllDevicesMBean*: clase Java que se encarga de listar todos los dispositivos.

- *listAllServicesMBean*: clase Java que se encarga de listar todos los servicios.
- *listAllTechniciansMBean*: clase Java que se encarga de listar todos los técnicos.
- *listAllUsersMBean*: clase Java que se encarga de listar todos los usuarios.
- *listDevicesByCustomerMBean*: clase Java que se encarga de listar todos los dispositivos de un cliente.
- *listServicesByCustomerMBean*: clase Java que se encarga de listar todos los servicios de un cliente.
- *loginMBean*: clase Java que se encarga del inicio de sesión de un usuario en la aplicación.
- *logoutMBean*: clase Java que se encarga del cierre de sesión de un usuario en la aplicación.
- *showServiceMBean*: clase Java que se encarga de mostrar un servicio a un usuario de tipo cliente.
- *updateCustomerMBean*: clase Java que se encarga de actualizar los datos de un usuario de tipo cliente.
- *updateDeviceMBean*: clase Java que se encarga de actualizar los datos de un dispositivo.
- *updateServiceMBean*: clase Java que se encarga de actualizar los datos de un servicio.
- *updateTechnicianMBean*: clase Java que se encarga de actualizar los datos de un usuario de tipo técnico.
- *updateUserMBean*: clase Java que se encarga de actualizar los datos de un usuario.

4.3 EJB (Capa de negocio)

Al implementar esta capa me ha permitido profundizar en el desarrollo de EJB y su interacción con la capa de integración, al mismo tiempo se ha conseguido que la persistencia en la base de datos sea consistente e integra.

En esta capa se han definido los EJBs previstos y aunque algunos de ellos se han modificado para definir métodos no previstos se enumeran a continuación agrupados por EJB:

- *ComunicationFacadeBean, ComunicationFacade* y *ComunicationFacadeRemote*:
    - *changeStatusService*: método que avisa al cliente afectado del cambio de estado de un servicio.
    - *sendMail*: método que envía un email a un usuario de tipo cliente. Aunque este método no estaba previsto inicialmente se ha decidido implementarlo por calidad de software, buenas prácticas en la construcción de software y encapsulación adecuada. Es un método de uso interno para el componente *comunication* de ahí

que no aparezca en el resto de componentes del componente *Comunication*.

- *ManagementFacade, ManagementFacadeBean* y *ManagementFacadeRemote*:
  - *addCustomerToDevice*: método que agrega un usuario de tipo cliente a un dispositivo.
  - *addDevice*: método que agrega un dispositivo a la aplicación.
  - *addService*: método que agrega un servicio a la aplicación.
  - *deleteDevice*: método que borra un dispositivo, no se ha implementado este método puesto que no queremos borrar ningún dispositivo y no es primordial para las funcionalidades inicialmente exigidas.
  - *findDeviceByBrand*: método que se encarga de retornar una colección de dispositivos mediante la búsqueda de un dispositivo filtrando por su atributo marca.
  - *findDeviceById*: método que se encarga de retornar un único dispositivo mediante la búsqueda de un dispositivo filtrando por su atributo id.
  - *findDeviceByModel*: método que se encarga de retornar una colección de dispositivos mediante la búsqueda de un dispositivo filtrando por su atributo modelo.
  - *findDeviceBySerial*: método que se encarga de retornar una colección de dispositivos mediante la búsqueda de un dispositivo filtrando por su atributo número de serie.
  - *findDeviceByType*: método que se encarga de retornar una colección de dispositivos mediante la búsqueda de un dispositivo filtrando por su atributo tipo.
  - *findServiceById*: método que se encarga de retornar un único servicio mediante la búsqueda de un servicio filtrando por su atributo id.
  - *findServiceByName*: método que se encarga de retornar una colección de servicios mediante la búsqueda de un servicio filtrando por su atributo nombre de cliente.
  - *findServiceByPhone*: método que se encarga de retornar una colección de servicios mediante la búsqueda de un servicio filtrando por su atributo teléfono de cliente.
  - *findServiceBySerialNumber*: método que se encarga de retornar una colección de servicios mediante la búsqueda de un servicio filtrando por su atributo número de serie del dispositivo sobre el que se realiza el servicio.
  - *listAllDevices*: método que retorna todos los dispositivos.
  - *listAllServices*: método que retorna todos los servicios.
  - *listDevicesByCustomer*: método que retorna todos los dispositivos de un usuario de tipo cliente.
  - *listDevicesWithoutCustomer*: método que retorna todos los dispositivos que no pertenecen a ningún cliente.
  - *listServicesByCustomer*: método que retorna todos los servicios de un usuario de tipo cliente.
  - *showService*: método que retorna un servicio mediante la búsqueda de servicios filtrando por el atributo id.

    - updateDevice: método que actualiza los datos de un dispositivo.
    - *updateService*: método que actualiza los datos de un servicio.
- *UserFacadeBean*, *UserFacade* y *UserFacadeRemote*:
    - addCustomer: método que añade un usuario de tipo cliente a la aplicación.
    - *addTechnician*: método que agrega un usuario de tipo técnico a la aplicación.
    - *addUser*: método que agrega un usuario a la aplicación.
    - *findAdminById*: método que se encarga de retornar un usuario de tipo administrador mediante la búsqueda de un usuario de tipo administrador filtrando por su atributo id.
    - *findCustomerById*: método que se encarga de retornar un usuario de tipo cliente mediante la búsqueda de un usuario de tipo cliente filtrando por su atributo id.
    - *findCustomerByName*: método que se encarga de la búsqueda de un usuario de tipo cliente por su atributo nombre. Parece repetido pero este solamente devuelve un cliente.
    - *findCustomersByEmail*: método que se encarga de retornar una colección de usuarios de tipo cliente mediante la búsqueda de un usuario de tipo cliente filtrando por su atributo email.
    - *findCustomersByName*: método que se encarga de retornar una colección de usuarios de tipo cliente mediante la búsqueda de un usuario de tipo cliente filtrando por su atributo nombre.
    - *findCustomersByPhone*: método que se encarga de retornar una colección de usuarios de tipo cliente mediante la búsqueda de un usuario de tipo cliente filtrando por su atributo teléfono.
    - *findTechnicianById*: método que se encarga de retornar una colección de usuarios de tipo técnico mediante la búsqueda de un usuario de tipo técnico filtrando por su atributo id.
    - *findTechniciansByDNI*: método que se encarga de retornar una colección de usuarios de tipo técnico mediante la búsqueda de un usuario de tipo técnico filtrando por su atributo dni.
    - *findTechniciansByEmail*: método que se encarga de retornar una colección de usuarios de tipo técnico mediante la búsqueda de un usuario de tipo técnico filtrando por su atributo email.
    - *findTechniciansByName*: método que se encarga de retornar una colección de usuarios de tipo técnico mediante la búsqueda de un usuario de tipo técnico filtrando por su atributo nombre.
    - *findTechniciansByPhone*: método que se encarga de retornar una colección de usuarios de tipo técnico mediante la búsqueda de un usuario de tipo técnico filtrando por su atributo teléfono.
    - *findUserByEmail*: método que se encarga de retornar un cliente mediante la búsqueda de un usuario de tipo usuario filtrando por su atributo email.
    - *listAllCustomers*: método que devuelve todos los usuarios de tipo cliente.
    - *listAllTechnicians*: método que devuelve todos los usuarios de tipo técnico.
    - *listAllUsers*: método que devuelve todos los usuarios de la aplicación.

- *Login*: método que se encarga de comprobar si el usuario que intenta iniciar sesión en la aplicación esta dado de alta en la base de datos.
- *Logout*: método que no necesita hacer nada, puesto que toda la funcionalidad recae sobre el *Managed Bean* de este mismo nombre, por ejemplo, no necesitamos comprobar que el usuario que cierra sesión en la aplicación está en la base de datos ya que simplemente se cierra la sesión actualmente abierta y no se necesita ninguna otra funcionalidad.
- *recoverPassword*: no implementada.
- *updateCustomer*: método que actualiza los datos de un usuario de tipo cliente.
- *updateTechnician*: método que actualiza los datos de un usuario de tipo técnico.
- *updateUser*: método que actualiza los datos de un usuario de la aplicación.
- *userOutOfService*: no implementada.

4.4 JPA (Capa de integración)

La capa de integración se ha implementado según lo previsto en el diseño inicial.

Se mencionan hechos destacables como que se ha usado la herencia de clases para una mejor reutilización de código siendo las clases CustomerJPA, TechnicianJPA y AdminJPA herederas de la super clase UserJPA. Esto nos ahorra tener que duplicar un excesivo número de líneas, por un lado, y tener preparada la aplicación para cambios futuros, por otro lado, así que cuando se decida diferenciar aún más a los distintos perfiles de la aplicación ya que ahora es una diferencia meramente académica ya que no es lógico que tengamos el dni de un técnico y sin embargo no lo tengamos de un cliente, pero sabemos por experiencia que más adelante las diferencias entre ambos perfiles van mucho más allá de un simple DNI o de un simple atributo.

5. Objetivos conseguidos.

5.1 Objetivos de desarrollo de la aplicación

Se ha logrado desarrollar el núcleo principal de una aplicación web para gestionar un servicio técnico de soporte TIC. Es posible mejorarla añadiendo funcionalidades adicionales con un coste de tiempo no excesivamente duro debido a la estructuración en componentes independientes pero conectados entre sí de los componentes que conforman la aplicación. El perfil administrador puede gestionar los técnicos, pero también los dispositivos y los clientes. El perfil técnico puede gestionar los dispositivos y los clientes. Y el perfil cliente puede solamente comprobar el estado de sus servicios y recibe un

mensaje cada vez que un servicio cambia de estado enviado automáticamente por el sistema sin intervención de cualquier otro usuario del mismo.

No se ha incidido en el desarrollo de borrado de dispositivos, servicios o cualquier otra entidad de la aplicación por que no es la finalidad del desarrollo ya que el cliente que nos encargó la aplicación quería desactivar los elementos a borrar, no eliminar ninguna entidad agregada a la aplicación, si no macarlas como desactivadas pero en ningún caso borrarlas.

Tampoco se ha desarrollado la funcionalidad de recuperar la contraseña por falta de tiempo ya que implica muchas nociones de seguridad que no han sido abordadas hasta el momento pero que tendrán una pronta solución.

5.2 Objetivos de aprendizaje tecnológico.

El trabajo desarrollado durante todo el TFG ha permitido profundizar en el conocimiento de la tecnología JavaEE en un ambiente de trabajo solitario, sin equipo alguno en el que apoyarme, debiendo resolver todas las complicaciones y problemas sin ayuda de nadie, y desarrollando la aplicación desde cero hasta el final sin ayudas externas cosa que nunca había realizado hasta hoy.

He logrado progresar en la reutilización de código, en la implementación de patrones de diseño como MVC o *Facade*, he logrado profundizar también en la interacción de las vistas con el modelo y este último a su vez con las entidades JPA y el lenguaje JPQL.

También he logrado comprender mejor el *framework* JSF y sus características implícitas, sobretodo valoro la propiedad *rendered* para mostrar partes de las vistas o ocultarlas dependiendo del perfil que acceda a ellas. También los mensajes de JSF han sido útiles para informar al usuario de los errores que cometía o los fallos de validación de los campos en los que ingresaba algún dato erróneo.

He intentado usar css3 para lograr una interfaz más agradable y atractiva al usuario pero debido a mis escasos conocimientos en diseño, me considero un programador y no un diseñador, me ha sido muy complicado establecer algún tipo de interfaz medianamente regular, cálido y atractivo para mejorar la experiencia del usuario.

5.3 Reflexión propia.

Como reflexión final quiero dejar constancia que la tecnología JavaEE permite desarrollar aplicaciones que proporcionan un marco de trabajo fácil de comprender, sencillo de usar para desarrollar aplicaciones, enormemente estructurado para no perderse por laberintos y al mismo tiempo implementa patrones de diseño y se ajusta perfectamente a diversos entornos con requisitos funcionales y no funcionales muy diversos, como entornos distribuidos, por lo prometedora de la tecnología y lo fácil de comprender quiero seguir avanzando en esta tecnología.

6. Trabajo futuro y posibles mejoras

6.1 Mejoras en la aplicación

6.1.1 Perfil cliente

- Una funcionalidad que permitiese a los clientes enviar y recibir mensajes entre los usuarios de tipo cliente y los usuarios de tipo técnicos. Esta prevista esta funcionalidad hasta el punto que está definido el JPA para los mensajes pero no se ha implementado todavía.
- Una funcionalidad que permita a un usuario de tipo cliente agregar imágenes a su dispositivo o perfil para una mejor captación por parte del servicio técnico del problema a resolver.
- Una funcionalidad que permita ver los servicios relativos a un usuario tipo cliente sin necesidad de iniciar sesión. También está medianamente contemplada esta funcionalidad hasta el punto de existir un *tracking code* dentro del JPA servicios, pero aún no se ha implementado.

6.1.2 Perfil técnico

- Una funcionalidad que marque los servicios por orden de urgencia y que sea posible verlos por colores en el listado de servicios, por ejemplo, color de fondo rojo para los urgentes, fondo amarillo para los menos urgentes, y fondo verde para los problemas sin urgencia, los cerrados estarían con el fondo de color azul y los servicios abiertos con cualquier tipo de situación acatarían un color de esa proposición.

6.1.3 Perfil administrador

- Una funcionalidad que permita a un administrador…

6.1.4 Mejora de la interfaz de usuario

Se considera que una interfaz es correcta, amigable y accesible cuando se puede consultar en distintos dispositivos (pantallas de PC de sobremesa o portátiles, dispositivos móviles tipo IOS, Android, AndroidTV, etc.) conduce al usuario sin que lo note por el camino correcto sin contradicciones, sin puntos de ruptura y sin estridencias. Además que sea lo más atractiva posible y agradable a la vista del usuario. Conseguir esto sería un objetivo bastante importante y deseable para la aplicación web.

6.2 Manuales del usuario.

Esta funcionalidad permitiría a un usuario cualquiera consultar el manual de usuario de la aplicación para tener el manual a mano y disponible en línea.

6.3 Evaluación de la calidad del código.

Cuando tratáramos una implementación real de un proyecto se debería incluir en su planificación desde que dispusiéramos de una versión entregable aunque no esté completa, de herramientas de evaluación de la calidad del código, como SonarQube.

6.4 Pruebas automatizadas.

En una implantación real de un proyecto se contaría desde un principio en la planificación del proyecto con un producto entregable aunque no completo de herramientas de pruebas automatizadas para detectar fallos incluso sin la participación de los desarrolladores de la aplicación y sin necesidad de que el usuario final hiciese pruebas con la aplicación.

Esto último no implica que se lleven a cabo pruebas de integración o unitarias por parte de usuarios finales, solo es una herramienta de ayuda adicional a estas pruebas con usuarios finales.

6.5 Seguridad de la aplicación.

La aplicación debe ser segura para asegurar los datos sensibles de los usuarios que manejan todas las aplicaciones actualmente. Por lo tanto, la información debería viajar cifrada por Internet, se deben verificar los datos introducidos, para evitar ataques de inyección SQL, entre otros muchos ataques posibles.

Los servidores que alojen la aplicación y la base de datos deberían estar protegidos y pueden comunicarse por túneles VPN. Se deben mantener actualizados y protegidos con software de seguridad como antivirus y firewalls.

Se debe fortalecer la configuración del servidor web (apache) y el servidor de aplicaciones (Wildfly) para evitar agujeros de seguridad ambos deben mantenerse actualizados y por su puesto deben implementar el protocolo https.

7. Conclusiones

Para todo estudiante universitario sabe que el TFG es el final de su trayectoria como estudiante pero también es el trabajo donde se aglutina y se reúnen los conocimientos adquiridos durante todo el itinerario en la ingeniería informática en nuestro caso. Sirve de recopilación, de visión de conjunto, de recuerdo de asignaturas, de renovación de conocimientos y de aprendizaje de nuevas metodologías, tecnologías o de mejora del desarrollo de aplicaciones.

El TFG, bajo mi humilde opinión, aúna conocimientos de las siguientes asignaturas:

- Gestión de proyectos: se aplican en la planificación inicial del proyecto, en la gestión a través de todo el proyecto, respondiendo a las dificultades y evaluando el proyecto en intervalos de tiempos acotados y breves por el denominado sprint de la metodología Scrum.
- Análisis y diseño con patrones: se aplican en el diseño de la aplicación analizando y determinando que patrones se pueden aplicar en el diseño de la aplicación.
- Diseño y arquitectura de bases de datos: se aplican los conocimientos en el diseño conceptual de la base de datos y en el diseño relacional de la misma.
- Ingeniería de requisitos: se aplican los conocimientos en el diseño, en la obtención de requisitos y en la documentación de estos requisitos.
- Diseño y programación orientada a objetos: se aplican estos conocimientos en este proyecto puesto que JavaEE está basado en orientación a objetos.
- Ingeniería del software de componentes y sistemas distribuidos: se aplican en este proyecto puesto que JavaEE está basado en entornos distribuidos.

Por falta de tiempo no se han logrado implementar el cien por cien de la aplicación, pero aun así, con el porcentaje conseguido el resultado es satisfactorio ya que disponemos de una herramienta totalmente funcional que cumple las funcionalidades básicas que nos había propuesto el cliente desde un principio. Algunos elementos como el borrado de entidades (clientes, equipos, servicios, etc.) no están disponibles pero son funcionalidades accesorias.

La planificación se ha desarrollado estrictamente como se había planeado desde el principio recayendo las penalizaciones de tiempo en los huecos de implementación anteriormente comentados resultando en la falta de implementación de los borrados de entidades.

La planificación de las entregas parciales se ha cumplido en todos los casos, PEC1, PEC2 y PEC3, sin desviaciones, el diagrama de Gantt para la planificación de tiempos ha resultado realmente útil en este caso, al mismo tiempo que se lograban los objetivos he procurado profundizar en los

conocimientos de cada una de las tecnologías usadas nutriéndome a su vez con nuevas funcionalidades y conociendo en detalle cada uno de los pormenores que guardan tecnologías como JavaEE, JSF, EJB, etc.

La metodología Ágil es la más adecuada para este tipo de proyecto, aunque no debemos cerrarnos y usar esta tecnología por defecto, ya que puede que en otros proyectos sea mejor usar otra tecnología. Nos ha permitido solventar dificultados grandes subdividiendo problemas complejos en pequeñas tareas más fáciles de solventar por separado teniendo en cuenta, eso sí, el acoplamiento global en el proyecto superando así numerosas dificultades que de otra manera hubieran sido inabordables.

No he tenido que abordar cambios drásticos en el proyecto, además del mencionado descarte del *framework* Spring para la implementación de la capa de presentación, no se han introducido cambios significativos para asegurar el éxito del trabajo excepto por algún rediseño mínimo de los bocetos iniciales de las pantallas.

El *framework* Spring se descartó por duplicidad de funcionalidades con las ya ofrecidas por defecto por la tecnología JavaEE y por qué no daba tiempo a profundizar en ese *framework* en el espacio de tiempo tan corto marcado para todo el proyecto.

Debido a que el proyecto era reducido y no excesivamente vinculado a los sistemas distribuidos no se ha ahondado este aspecto tan interesante de esta tecnología usada en JavaEE pero si en un futuro se necesitase por necesidades del cliente el proyecto ya estaría cimentado sobre una tecnología que permite las ubicaciones múltiples de talleres de servicio con sistemas distribuidos.

8. Glosario

EJB: *Enterprise Java Beans*: Clases Java diseñadas para ejecutarse en un contenedor específico.

HTTP: protocolo de comunicación por Internet, del inglés, *Hypertext Transfer Protocol*.

HTTPS: versión securizada extremo a extremo del protocolo de HTTP.

IDE: *Integrated Development Environment*. Software de desarrollo integrado o que soporta todo el ciclo de desarrollo.

JavaEE: *Java Enterprise Edition*. Plataforma de desarrollo con funcionalidades específicas para desarrollo de aplicaciones empresariales.

JPA: *Java Persistence API*. Especificación Java para la persistencia de entidades Java en bases de datos relacionales.

JPQL: *Java Persistence Query Language*. Lenguaje de consultas SQL orientado a objetos para el entorno de persistencia JavaEE.

JSF: *Java Server Faces*. Especificación Java para el desarrollo de entornos web de la plataforma JavaEE.

Scrum: metodología ágil de desarrollo de software.

TIC: tecnologías de la información.

9. Bibliografía

10. Anexos

Diagrama de Gantt en formato gráfico.