Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Aplicación Web para la Gestión de Incidencias en el Ámbito Hotelero

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Juan Jordá Pérez

Tutor: Joan Fons i Cors

Cotutor: Vicente Pelechano Ferragud

Curso 2018 - 2019

Resumen

En este trabajo se crea una aplicación web para la gestión de incidencias en el ámbito hotelero. La estrategia que se usa es la de un servidor con la base de datos, otro que alberga toda la lógica y una interfaz de usuario basada en web para facilitar la creación de interfaces para escritorio y para móvil. Se utiliza una arquitectura basada en capas, en la que cada una de ellas separa físicamente la parte de persistencia, la lógica y la interfaz. A su vez, la segunda de ellas se distribuye de manera lógica en capa de servicios, de lógica de negocio y de acceso a datos. La solución se implementa a través de la tecnología de Microsoft .NET para la parte del servidor, la de Angular para la parte de la interfaz web y para MySql para la base de datos.

Palabras clave: Angular, .NET, Full Stack, aplicación web, gestión de incidencias.

Resum

En aquest treball es crea una aplicación web per a la gestió d'incidències en l'àmbit hoteler. L'estratègia que s'empra és la d'un servidor amb la base de dades, un altre que alberga tota la lògica i una interfície d'usuari basada en web per a facilitar la creació d'interfícies per a escriptori i per a móvil. S'utilitza una arquitectura basada en capes, en la que cadascuna d'elles separa físicament la part de persistència, la lògica i la interfície. Al seu torn, la segona d'aquestes es distribueix de manera lògica en capa de servicis, de lògica de negoci i d'accés a dades. La solució s'implementa a través de la tecnología de Microsoft .NET per a la part del servidor, la d'Angular per a la part de la interfície web i MySql per a la base de dades.

Paraules clau: Angular, .NET, Full Stack, aplicació web, gestió d'incidències.

Abstract

In this assignment, a web application is created to manage incidents in the hotel business. The strategy used is one server with the database, another that hosts all the logic and a web-based user interface to facilitate the creations of interfaces for desktop and mobile. A layer-based architecture is used, in which each of them are physically separated in the persistent part, the logic and the interface. In turn, the second of them is distributed logically in the service layer, business logic and data access. The solution is implemented through Microsoft .NET technology for the server, Angular for the part of the web interface and MySql for the database.

Keywords: Angular, .NET, Full Stack, web application, management of incidents.

Tabla de contenidos

Índice de ilustraciones

Índice de tablas

1. Introducción

La tecnología se encuentra en constante evolución y cada vez más al alcance de nuestras manos. Hoy en día, dependemos totalmente de ella en cualquier campo, comunicaciones, banca, medicina, etc. Los dispositivos tecnológicos son herramientas básicas e imprescindibles para realizar cualquier operación, por lo que todos disponemos siempre de algún utensilio al alcance de nuestra mano, motivo por el cual el uso del papel en el día a día está desapareciendo, ya sea en la toma apuntes en clase o en la preinscripción de una receta médica.

Debido a las razones mencionadas anteriormente, las empresas se están sumergiendo en la transformación digital [8]. Les aportan muchos beneficios, tales como una mejor organización, control, seguridad y rapidez en la consulta de información. Así que, ¿por qué no transformar la gestión a papel en un servicio digital?

1.1 Motivación

La motivación por el que se crea este proyecto el desarrollo de una aplicación web completamente adaptada para la gestión de incidencias en el ámbito hotelero, debido a que existe una gran variedad de aplicaciones para ello, pero son demasiado genéricas, ya que sirven desde la gestión de incidencias para comunidades de vecinos hasta las que puedan suceder en una casa propia. Ésta en concreto se centra para que cualquier hotel tenga la capacidad de poder gestionar sus incidencias de una forma rápida, clara y fácil para el usuario y, además, se pueda hacer desde cualquier dispositivo y lugar distintos.

1.2 Objetivos

El principal objetivo del proyecto es el desarrollo de un servicio informático que permita gestionar y realizar el seguimiento de incidencias en el ámbito hotelero. La aplicación debe ayudar a controlar el ciclo de vida de una incidencia, pudiéndosele asignar distintos usuarios a cada estado por el que pasa. De esta forma, se debe saber en todo momento cómo se encuentra una incidencia, quién se ha encargado o se está encargando de ella y en qué momento se ha realizado. La aplicación debe diferenciar entre los distintos usuarios que se conecten a ella, ya que cada uno de ellos debe realizar tareas específicas.

1.3 Metodología

Dada la carencia en conocimientos en ciertas tecnologías utilizadas que se mencionarán con posterioridad, se procedió a la realización de cursos online, tutoriales y documentación oficial de las propias tecnologías utilizadas, etc.

Una vez adquiridos los conocimientos necesarios para la realización de este proyecto, se buscó cuál sería el mejor ciclo de vida del desarrollo de software para este problema planteado. Finalmente, se optó por el modelo clásico o en cascada con prototipado [4], dado que, al ser un trabajo individual, no se pueden ir desarrollando varios procesos que son complementarios unos de otros al mismo tiempo.

Esta metodología ordena rigurosamente las etapas del ciclo de vida del software, de tal forma que cada etapa depende de la anterior, por lo que debe esperar a su finalización. Al añadirle la opción de prototipado, queda mucho más clara la funcionalidad y el diseño a la hora de pasar a la etapa de diseño e implementación.

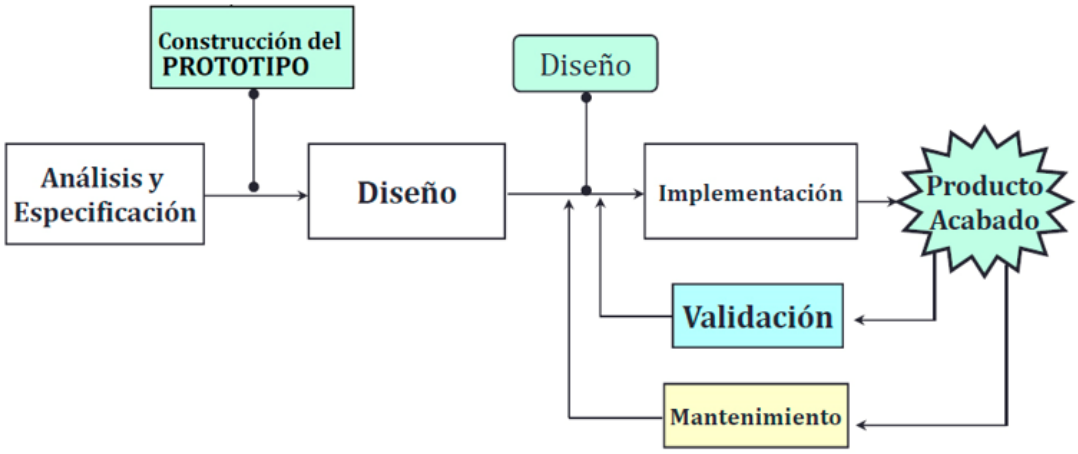

Ilustración 1: Ciclo de vida en cascada con prototipado.

Durante el proceso de la implementación del producto, se ha optado por la metodología Kanban [9]. Kanban, representada por una tarjeta, se irá desplazando a través de las distintas etapas de trabajo hasta su finalización. Es una forma visual de ver el flujo del trabajo, de visualizar lo que se está haciendo en cada momento y además es una manera de evitar mezclar unas tareas con otras. Está compuesta principalmente por tres estados:

- *To Do*: Se deben incluir las tarjetas Kanban con las tareas que están pendientes de realizar. En el supuesto de estar implementando otra tarea y encontrase con una nueva, se añadiría una más a este estado.

- *In Progress*: Se deben situar aquellas que se empiecen a implementar o se estén implementando. En el supuesto de encontrarse con algún problema o nueva tarea durante la realización de una en el estado actual, se procedería al añadido de una nueva tarjeta al estado anterior.

- *Done*: A esta etapa deben añadirse todas las tarjetas Kanban que estén finalizadas.

Ilustración 2: Metodología Kanban.

1.4 Estructura de la memoria

- Se empezará explicando el contexto tecnológico (Punto 2) en el que se desarrolla el presente trabajo.
- Se proseguirá (Punto 3) con el planteamiento de las características que debería tener la aplicación.
- A continuación, se analizará (Punto 4) y diseñará (Punto 5) el problema a resolver y se implementará (Punto 6) la solución de éste.
- Posteriormente, se presentarán varios casos de uso de la aplicación con el fin de verificar el uso de la aplicación (Punto 7).
- Para concluir, se analizarán las metas alcanzadas y las no alcanzadas, las mejoras sobre el trabajo futuro y la valoración personal sobre la realización de este proyecto.

2. Contexto tecnológico

El auge de las aplicaciones web está incrementando a un ritmo constante. La comodidad de no tener nada que descargar o instalar, está haciendo que la población se decante cada día más por éstas. Tanto los clientes como sus usuarios, no se percatan de las posibles actualizaciones que se hayan podido realizar, simplemente, acceden a la aplicación y está todo listo para su funcionamiento. Una ventaja muy a su favor es la versatilidad y compatibilidad de éstas, ya que no se necesitan desarrollar para cada tipo de sistema operativo, únicamente se desarrolla una interfaz que se pueda adaptar también a los pequeños dispositivos. Dicho esto, se puede deducir que la reducción de costes de producción y manutención es notable debido a que no se debe desarrollar para todo tipo de plataforma de manera individual, ya sea Windows, OS X o Linux.

Llegado el momento de la elección de las tecnologías a emplear, se tuvo un debate entre usar Java o .NET para el *back-end*. Después de analizar la ventajas e inconvenientes de ambas [10], la balanza se decantó por .NET, ya que, además, se disponía de cierta familiaridad con su entorno de desarrollo, que se mencionará más adelante.

Por parte del *front-end*, se optó por el *framework* de Angular, ya que se querían obtener conocimientos en nuevas tecnologías empleadas y demandadas para el desarrollo de aplicaciones web.

2.1 Herramientas

2.1.1 XAMPP

Ilustración 3: Logo de XAMPP.

XAMPP es un paquete de software libre que consiste, entre muchas cosas, en un servidor web Apache, un sistema de gestión de base de datos MySQL, intérpretes de PHP y Perl y servidores de FTP como FileZillas, ProFTP, etc.

El uso que se le ha otorgado en este proyecto es el de servidor y sistema de gestión de base de datos, debido a la fácil instalación y configuración proporcionada a través de su aplicación.

2.1.2 Sourcetree

Ilustración 4: Logo de Sourcetree

Sourcetree es un cliente gratuito de Git. Git es un sistema de control de versiones, gratuito y de código abierto, que está diseñado para almacenar las distintas versiones de nuestro producto. Puede crear distintas ramas del repositorio para trabajar de forma paralela con otros desarrolladores o con distintas características del mismo proyecto.

En nuestro caso, se ha hecho uso de GitLab para la creación y almacenaje del repositorio. Gracias a SourceTree, se entiende de una manera más visual el concepto de control de versiones de Git, ya que dispone de una interfaz gráfica que nos permite crear ramas, ver nuestro código modificado, ver el registro de cambios y mantenerlos o descartarlos, recuperar versiones antiguas y muchas más opciones, y todo esto con la facilidad de ver de forma gráfica todo lo que se debería estar haciendo a través de comandos de Git.

2.1.3 Postman

Ilustración 5: Logo de Postman.

Postman es una herramienta que se caracteriza por la realización de distintas tareas dentro del mundo del API REST. Se ha utilizado en este proyecto para comprobar el correcto funcionamiento de la API a través de peticiones HTTP. Ha sido un gran aliado, ya que ha permitido verificar el adecuado comportamiento de nuestra API Web.

2.2 Entornos

Dada la elección de las tecnologías .NET y Angular, la primera desarrollada por Microsoft y la segunda que emplea TypeScript, también creado por Microsoft, se han elegido los entornos relacionados y más adecuados a las tecnologías empleadas en el desarrollo de la aplicación: Microsoft Visual Studio 2019 y Microsoft Visual Studio Code.

2.2.1 Microsoft Visual Studio 2019

Ilustración 6: Logo de Microsoft Visual Studio 2019.

Microsoft Visual Studio es un entorno de desarrollo integrado compatible con los sistemas operativos más comunes. Se usa para el desarrollo de aplicaciones de escritorio, aplicaciones y servicios web y aplicaciones móviles. Es capaz de soportar 36 tipos distintos de lenguajes de programación, pero para nuestro proyecto, se ha utilizado el lenguaje C# para todo el desarrollo del *back-end.*

2.2.2 Microsoft Visual Studio Code

Ilustración 7: Logo de Microsoft Visual Studio Code.

Microsoft Visual Studio Code es un editor de código fuente compatible con distintos sistemas operativos y lenguajes de programación. Se ha usado para este proyecto en la parte del *front-end* ya que, debido a su adaptabilidad con el f*ramework* de Angular, puede llegar a convertirse en un pequeño entorno de desarrollo en el cual se permite la instalación de componentes y módulos, compilación y ejecución de la aplicación por parte del *front-end.* Con este entorno y para este proyecto, se han utilizado los lenguajes TypeScript, HTML y CSS.

2.3 Frameworks

Un *framework* es una plataforma para desarrollar aplicaciones software. Provee una base en la que los desarrolladores pueden construir programas para una específica plataforma a través de clases y funciones predefinidas, que éstas, a su vez, pueden ser modificadas a gusto del programador. Esto hace la vida mucho más fácil a quien debe desarrollar, ya que no es necesario reinventar la rueda cada vez que se quiere crear una nueva aplicación.

2.3.1 Microsoft .NET

Ilustración 8: Logo de Microsoft .NET.

Consiste en una plataforma para el desarrollo de software creada por Microsoft [11] con el objetivo de poder desarrollar aplicaciones y sistemas que fueran independientes de la arquitectura física y del sistema operativo en el que se ejecutaran. Nuestra aplicación está construida sobre .NET usando uno de sus lenguajes de programación, el de C#.

2.3.2 Angular

Ilustración 9: Logo de Angular.

Angular es una plataforma y *framework* para construir aplicaciones en HTML y TypeScript. Se utiliza para crear aplicaciones web de una sola página y usa el Modelo Vista Controlador. Básicamente, la biblioteca lee el HTML con notaciones Angular. Éstas contienen propiedades que se relacionan bidireccionalmente con el componente desarrollado en TypeScript, haciendo que el HTML pueda cambiar y actualizarse

constantemente sin necesidad de refrescar el navegador. Además, posee servicios para poder compartir información con otros componentes o para traer información de peticiones HTTP. Se ha utilizado la versión 7.

2.3.3 Entity Framework

Ilustración 10: Logo de Entity Framework.

Entity Framework [6] es un *framework* de mapeo objeto-relacional que permite a los desarrolladores .NET trabajar con la base de datos usando objetos .NET. Elimina la mayor necesidad de crear código para acceder a los datos. Se encarga de crear la conexión con la base de datos y ejecutar sus comandos, así como de la transformación de los datos consultados a los objetos propios de la aplicación.

2.3.4 StructureMap

StructureMap es un *framework* encargado de la inyección de dependencias para .NET. La inyección de dependencias es la encargada de instanciar las dependencias que necesita una clase en lugar de ser dicha clase quien se encargue de crear explícitamente dichas dependencias necesarias.

2.3.5 Bootstrap

Ilustración 11: Logo de Bootstrap.

Bootstrap es un *framework* de *front-end* gratuito que permite un rápido y fácil desarrollo web. Incluye plantillas de diseño para tipografías, formularios, botones, tablas, modales, etc. Permite una gran capacidad de diseño web adaptable a todo tipo de pantallas y dispositivos.

2.4 Librerías

2.4.1 AutoMapper

Ilustración 12: Logo de AutoMapper

AutoMapper es una pequeña y simple librería construida para solventar el problema de mapear un objeto a otro. Utiliza un algoritmo para hacer coincidir los valores de origen con los de destino. Desarrollado para el Microsoft .NET.

2.5 Componentes

2.5.1 Ag-Grid

Ag-Grid es un componente compatible con *frameworks* más populares de JavaScript. Consiste en la creación de tablas completamente personalizables y con una gran documentación oficial que la soporta. Posee versiones tanto de pago como gratuitas. En este proyecto, se ha utilizado para mostrar prácticamente toda la información de cada vista.

2.5.2 Ngx-toastr

Ngx-toastr es un componente que muestra alertas en pantalla para el usuario. Posee unos estilos propios para casos de funcionamiento correcto, erróneo, de alerta y de información. En la aplicación, se ha hecho uso para los casos en que se muestra cualquier tipo de notificación al usuario.

2.5.3 Ng-select

Ng-select es otro de los componentes utilizados. Tiene la misma funcionalidad que la conocida etiqueta <select> del lenguaje HTML, pero éste posee una gran configurabilidad, como el tener él mismo un buscador por teclado y no ser necesario recorrer toda la lista para encontrar un elemento.

2.6 Soluciones existentes

En el mercado, existe una aplicación móvil sobre la misma temática, la gestión de incidencias, que se engloba entre una de sus opciones. Ésta se llama UuupsApp [15] y sirve tanto para Android como para iOS. La aplicación permite la gestión de incidencias, quejas, reclamaciones, etc. para cualquier ámbito, ya que puedes formar grupos con tu familia, empresa, vecinos, etc. Éstas tienen un corto ciclo de vida, pasan de abiertas a en curso y finalmente a cerradas, además del estado de canceladas.

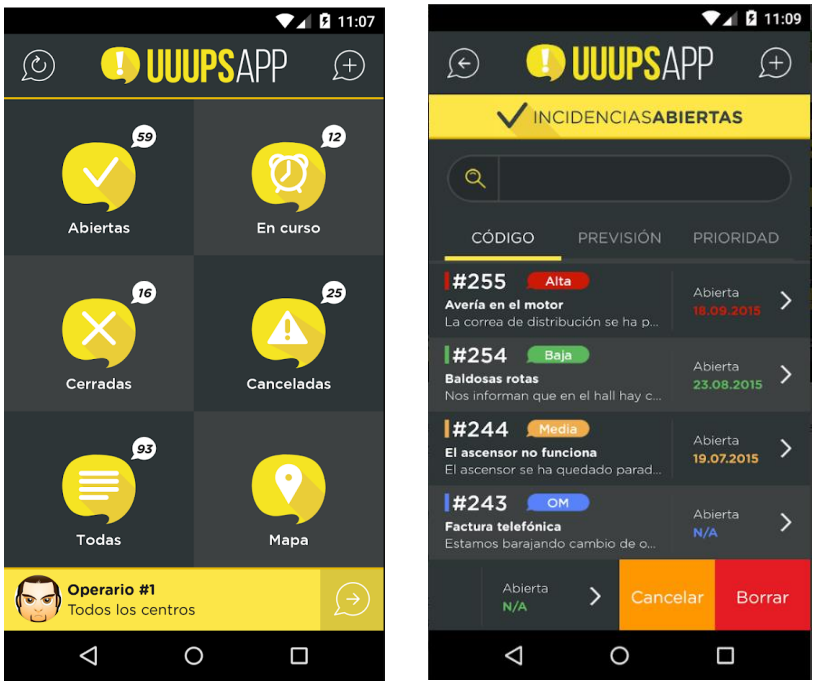

Ilustración 13: Interfaz de UuupsApp.

En el caso de nuestra aplicación, se está buscando únicamente para las incidencias en el ámbito hotelero, ya que es un sector que está expandido mundialmente y al que le podría ser de gran utilidad al ser diseñada especialmente para él. Además de poderse emplear en un teléfono móvil como la aplicación anterior, se debe emplear también en equipos de escritorio, ampliando de esta forma su compatibilidad y adaptabilidad. Además, UuupsApp posee una serie de estados de incidencias muy genéricos, en cambio, nuestra aplicación dispondrá de más estados de avería y más concretos para el ámbito descrito.

3. Planteamiento inicial

Se va a proceder a la creación de una aplicación web con la finalidad de agilizar, automatizar e informatizar la gestión de incidencias en el ámbito hotelero. Se desea registrar cualquier tipo de avería, desde una localizada en una habitación hasta otras localizadas en zonas comunes del hotel. Será una aplicación genérica para todos los hoteles, por lo que se accederá a la aplicación vía identificación de usuarios y se mostrarán los datos pertenecientes a la empresa de dicho usuario.

Se dispondrá de tres posibles tipos de usuario:

- Administrador: será la persona encargada de la gestión completa de la aplicación web.
- Recepcionista: su función será la de registrar, visualizar y controlar cuál es el estado de las incidencias en todo momento.
- Servicio técnico: es la persona con funcionalidades más restringidas y con el uso de la aplicación desde un terminal móvil. Su función será la de cambiar el estado de las incidencias o añadir comentarios sobre éstas.

Cada usuario tendrá su propio acceso a la aplicación, donde el DNI será el identificador del inicio de sesión y la contraseña será proporcionada por el administrador, aunque deberá ser cambiada una vez se acceda a la aplicación.

Desde recepción se recibirán las quejas o sugerencias de cualquier persona sobre una avería, ya sea cliente o empleado. El usuario con rol de recepcionista que se encuentre atendiendo la queja, deberá registrarla en el sistema introduciendo los datos necesarios para la creación de ésta.

Cada incidencia se deberá registrar con los siguientes datos:

- Localización donde se ha producido.
- Tipo de avería.
- Descripción donde poder especificar mejor dónde ubicar la avería en caso de no ser localizada a simple vista o datos útiles para la resolución de ésta por parte del servicio técnico.
- Tendrá otros datos ocultos al usuario, como la fecha actual de registro, el primer estado de creación y el usuario que la registra.

Una vez creada una incidencia, únicamente podrá ser eliminada por el usuario Administrador, mientras que los demás usuarios podrán añadir información a cualquiera de sus datos o modificarlos.

Cada incidencia dispondrá de una serie de estados que clarificarán el momento en el que se encuentra:

- Nueva: es el estado en el que se encuentra una incidencia en el momento de su creación.

- En verificación: indica que está siendo revisada por alguien del servicio técnico para ver que existe y para obtener una descripción más específica para su futura reparación.
- Verificada: estado por el que el servicio técnico verificará que la información proporcionada a la incidencia es correcta y está lista para su reparación.
- En reparación: existe algún usuario que se está encargando del proceso de reparación de la avería.
- Reparada: estado que indica que la avería ha sido reparada con éxito.
- Cerrada: el usuario con rol recepcionista deberá cerrarla y avisar al cliente del estado en caso de que alguien haya sido perjudicado debido a ésta.

Las incidencias podrán ser buscadas y ordenadas por cualquiera de las opciones que se presenten en la tabla de visualización. Además de los filtros generales, se dispondrá de un filtro de estados para mostrar los que sean más adecuados para cada tipo de usuario.

Todos los usuarios disponen de un sistema de notificaciones en el cual serán avisados de los nuevos registros o modificaciones sobre cualquier incidencia y con posibilidad de acceso directo a ésta.

Por parte del usuario Administrador, tendrá la función de crear, borrar y editar tanto empleados como incidencias y los tipos de éstas. El resto de los usuarios tendrá el acceso bloqueado a todas las vistas de gestión de la aplicación a excepción de la de incidencias.

4. Análisis del problema

4.1 Requisitos funcionales

Los requisitos funcionales son aquellos que especifican cualquier actividad que se pueda realizar en el sistema, es decir, describen un comportamiento o una particular función éste cuando se dan ciertas condiciones.

Para el desarrollo de este proyecto, se han tomado en cuenta los siguientes requisitos funcionales:

- Cualquier usuario con credenciales podrá conectarse a la aplicación.
- Cualquier usuario que haya iniciado sesión será capaz de desconectarse.
- El administrador de una nueva empresa podrá registrarla en la aplicación.
- Únicamente el usuario con rol de administrador será el encargado de crear, modificar o borrar los usuarios.
- Cualquier usuario podrá modificar sus datos personales y contraseña.
- Solamente el administrador podrá crear, modificar o borrar los tipos de incidencias y tipos de instalaciones.
- El usuario administrador podrá crear, borrar o modificar incidencias.
- El usuario recepcionista podrá crear o modificar incidencias.
- El usuario con rol de servicio técnico únicamente podrá modificar incidencias.
- Todos los usuarios recibirán notificaciones de aviso cuando se cree o modifique alguna incidencia y podrán acceder directamente a ella, sin necesidad de buscarla.
- Cualquier usuario será capaz de filtrar por el estado en que se encuentren las incidencias y de ordenar y buscar incidencias por cualquiera de sus datos.

4.2 Casos de uso

Una vez se han estudiado todos los requisitos funcionales de la aplicación, se procede al análisis de cada uno de ellos. Estos análisis son conocidos como casos de uso. Son una forma de representar todos los requisitos expuestos anteriormente desde el punto de vista del usuario, mostrando la interacción entre él mismo y el sistema. Se expondrán diagramas para facilitar la identificación de todos los casos de uso.

4.2.1 Requisitos de sesión o registro de usuario

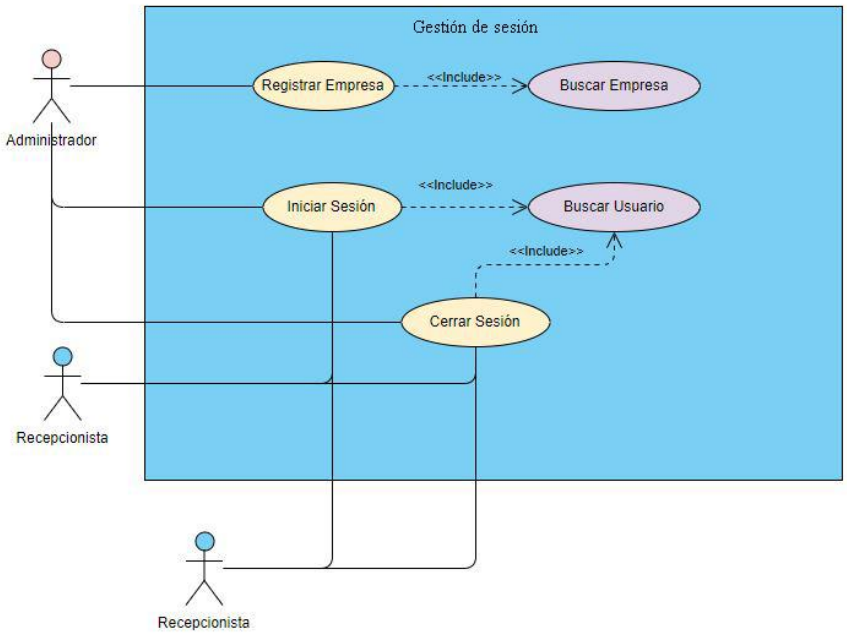

Ilustración 14: Diagrama de caso de uso de gestión de sesión.

ID	1
Caso de uso	**Acceso a la aplicación**
Actores	Administrador, servicio técnico, recepcionista
Propósito	Acceso a la aplicación web
Resumen	1. Se introducirá la URL de la aplicación web. 2. Se introducirán los campos requeridos para el inicio de sesión del usuario. 3. El sistema da acceso a la funcionalidad de la aplicación web.
Precondición	El actor debe conocer tanto el usuario como la contraseña para poder acceder a la aplicación.
Postcondición	Se accede a la aplicación
Extensión síncrona	En 3, si los datos introducidos para el inicio de sesión no son correctos, el sistema mostrará un error y el empleado podrá volver a introducir los datos.

Tabla 1: Requisito de acceso a la aplicación.

ID	2
Caso de uso	**Registro de una nueva empresa**
Actores	Administrador
Propósito	Crear un nuevo registro de una empresa
Resumen	1. El usuario clicará en el botón de nuevo registro. 2. El usuario introducirá los datos necesarios para la creación de la nueva empresa (CIF y nombre). 3. El usuario recibirá una alerta con un usuario por defecto para el inicio de sesión.

Precondición	El usuario debe encontrarse en la vista de inicio de sesión
Postcondición	El usuario recibirá una cuenta con la que iniciar sesión
Extensión síncrona	En 2, si los datos introducidos no cumplen los requisitos mínimos (longitud) o están vacíos, el sistema mostrará un aviso para la corrección de los datos.

Tabla 2: Requisito de registro de una nueva empresa.

ID	3
Caso de uso	**Desconectarse de la sesión**
Actores	Administrador, recepcionista, servicio técnico
Propósito	Desconectar tu cuenta de usuario de la sesión
Resumen	1. El usuario clicará sobre el logotipo de usuario que aparece junto con su nombre y seleccionará “Cerrar Sesión”. 2. Aparecerá un cuadro de diálogo para la aceptación del cerrado de sesión. 3. El sistema desconectará al usuario.
Precondición	El usuario debe encontrarse conectado a la aplicación.
Postcondición	El sistema regresará a la página de inicio de sesión.
Extensión síncrona	

Tabla 3: Requisito de descontarse de la sesión.

4.2.2 Requisitos de gestión de usuarios

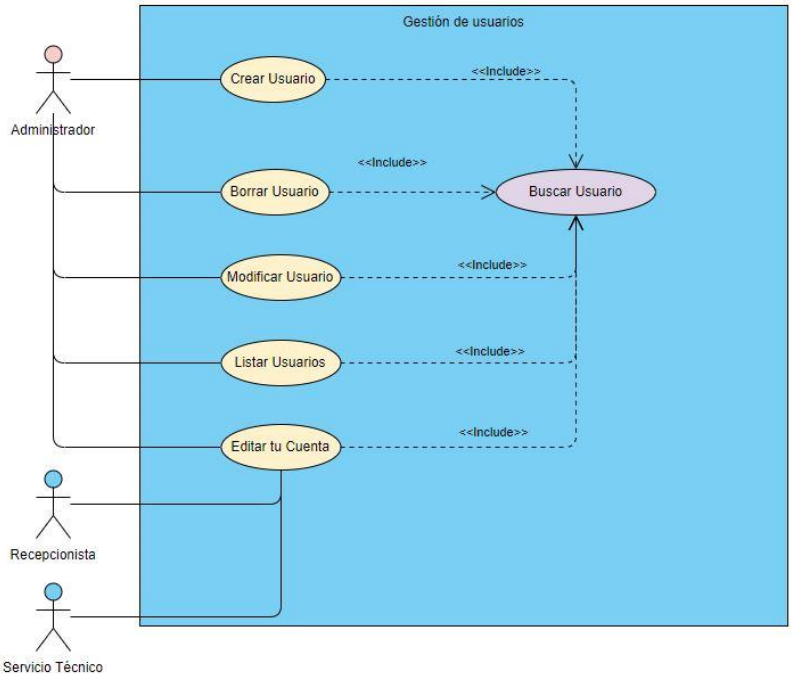

Ilustración 15: Diagrama de caso de uso de gestión de usuarios.

ID	4
Caso de uso	**Creación de un usuario**
Actores	Administrador
Propósito	Añadir un usuario
Resumen	1. El usuario introduce la información específica del usuario que va a crear (nombre, apellidos, contraseña, DNI y tipo de usuario). 2. El sistema crea el nuevo usuario.
Precondición	El usuario debe encontrarse en la vista de gestión de usuarios
Postcondición	El usuario es creado y almacenado
Extensión síncrona	En 2, si los datos introducidos no cumplen los requisitos mínimos (longitud o caracteres), están vacíos o el usuario ya existe, el sistema mostrará un aviso para la corrección de éstos.

Tabla 4: Requisito de creación de un usuario.

ID	5
Caso de uso	**Modificación de un usuario**
Actores	Administrador
Propósito	Modificar los datos de un usuario
Resumen	1. El sistema mostrará todos los usuarios. 2. El actor seleccionará el usuario a modificar. 3. El sistema mostrará la información de ese usuario. 4. El actor modificará los datos necesarios. 5. El sistema actualizará los datos del usuario.
Precondición	El usuario debe encontrarse en la vista de gestión de usuarios
Postcondición	El usuario es actualizado y almacenado
Extensión síncrona	En 4, si los datos introducidos no cumplen los requisitos mínimos (longitud, caracteres), están vacíos o el usuario contiene datos de otro ya existente, el sistema mostrará un aviso para la corrección de éstos. En 4, si el administrador se ha modificado los datos a sí mismo, se pedirá un nuevo inicio de sesión en la aplicación.

Tabla 5: Requisito de modificación de un usuario.

ID	6
Caso de uso	**Eliminación de un usuario**
Actores	Administrador
Propósito	Supresión de un usuario
Resumen	1. El sistema mostrará todos los usuarios. 2. El actor seleccionará el usuario a eliminar. 3. El sistema mostrará un diálogo de confirmación. 4. El actor aceptará el diálogo. 5. El sistema borrará el usuario.
Precondición	El usuario debe encontrarse en la vista de gestión de usuarios
Postcondición	El usuario es eliminado
Extensión síncrona	En 4, si el usuario no acepta el diálogo de confirmación, el usuario no será eliminado.

Tabla 6: Requisito de eliminación de un usuario.

ID	7
Caso de uso	**Modificación del perfil de usuario**
Actores	Administrador, recepcionista, servicio técnico
Propósito	Modificar datos del usuario con la sesión activa
Resumen	1. El usuario clicará sobre el logotipo de usuario que aparece junto con su nombre y seleccionará “Mi Cuenta”. 2. Aparecerá un cuadro de diálogo con los datos a modificar. 3. El usuario modificará los campos que desee (nombre, apellidos o contraseña). 4. El sistema actualizará los datos del usuario.
Precondición	El usuario debe encontrarse conectado a la aplicación.
Postcondición	El usuario dispondrá de sus nuevos datos.
Extensión síncrona	En 3, si se ha modificado la contraseña, se pedirá de nuevo un inicio de sesión. En 3, si los datos modificados no son correctos, se mostrará una alerta para su corrección.

Tabla 7: Requisito de modificación del perfil de usuario.

4.2.3 Requisitos de gestión de parámetros

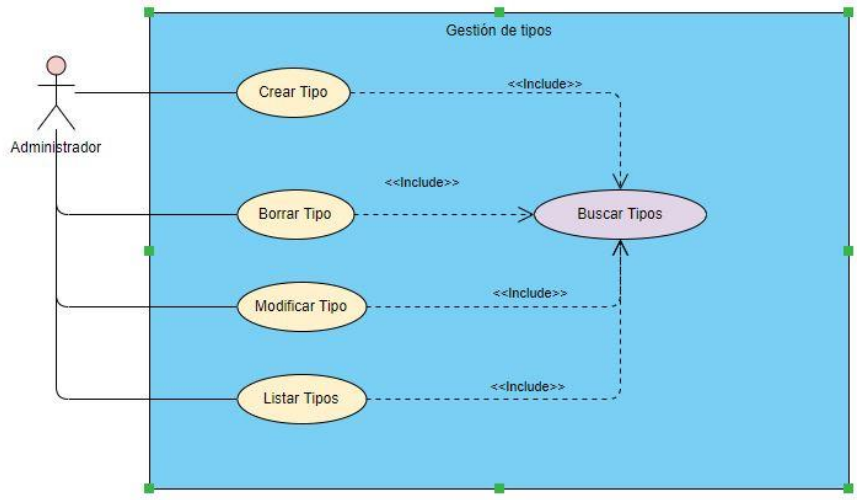

Ilustración 16: Diagrama de caso de uso de gestión de parámetros.

ID	8
Caso de uso	**Creación de un tipo de incidencia**
Actores	Administrador
Propósito	Añadir un tipo de incidencia
Resumen	1. El usuario introduce la información específica del tipo de incidencia que va a crear (descripción). 2. El sistema crea el nuevo tipo de incidencia.
Precondición	El usuario debe encontrarse en la vista de gestión de parámetros
Postcondición	El tipo de incidencia es creado y almacenado
Extensión síncrona	En 1, si los datos introducidos no cumplen los requisitos mínimos (longitud), están vacíos o la descripción ya existe, el sistema mostrará un aviso para la corrección del dato.

Tabla 8: Requisito de creación de tipo de incidencia.

ID	9
Caso de uso	**Modificación de un tipo de incidencia**
Actores	Administrador
Propósito	Modificar los datos de un tipo de incidencia
Resumen	1. El sistema mostrará todos los tipos de incidencia. 2. El usuario seleccionará el tipo de incidencia a modificar. 3. El sistema mostrará la información de ese tipo de incidencia. 4. El usuario modificará los datos necesarios. 5. El sistema actualizará los datos del tipo de incidencia.
Precondición	El usuario debe encontrarse en la vista de gestión de parámetros
Postcondición	El tipo de incidencia es actualizado y almacenado
Extensión síncrona	En 4, si los datos introducidos no cumplen los requisitos mínimos (longitud), están vacíos o la descripción modificada ya existe, el sistema mostrará un aviso para la corrección del dato.

Tabla 9: Requisito de modificación de un tipo de incidencia.

ID	10
Caso de uso	**Eliminación de un tipo de incidencia**
Actores	Administrador
Propósito	Supresión de un tipo de incidencia
Resumen	1. El sistema mostrará todos los tipos de incidencia. 2. El actor seleccionará el tipo de incidencia a eliminar. 3. El sistema mostrará un diálogo de confirmación. 4. El actor aceptará el diálogo. 5. El sistema borrará el tipo de incidencia.
Precondición	El usuario debe encontrarse en la vista de gestión de tipo de incidencia
Postcondición	El tipo de incidencia es eliminado
Extensión síncrona	En 4, si el usuario no acepta el diálogo de confirmación, el tipo de incidencia no será eliminado.

Tabla 10: Requisito de eliminación de un tipo de incidencia.

ID	11
Caso de uso	**Creación de un tipo de instalación**
Actores	Administrador
Propósito	Añadir un tipo de instalación
Resumen	1. El usuario introduce la información específica del tipo de instalación que va a crear (descripción). 2. El sistema crea el nuevo tipo de instalación.
Precondición	El usuario debe encontrarse en la vista de gestión de parámetros
Postcondición	El tipo de instalación es creado y almacenado
Extensión síncrona	En 1, si los datos introducidos no cumplen los requisitos mínimos (longitud), están vacíos o la descripción ya existe, el sistema mostrará un aviso para la corrección del dato.

Tabla 11: Requisito de creación de un tipo de instalación.

ID	12
Caso de uso	**Modificación de un tipo de instalación**
Actores	Administrador
Propósito	Modificar los datos de un tipo de instalación
Resumen	1. El sistema mostrará todos los tipos de instalación. 2. El usuario seleccionará el tipo de instalación a modificar. 3. El sistema mostrará la información de ese tipo de instalación. 4. El usuario modificará los datos necesarios. 5. El sistema actualizará los datos del tipo de instalación.
Precondición	El usuario debe encontrarse en la vista de gestión de parámetros
Postcondición	El tipo de instalación es actualizado y almacenado
Extensión síncrona	En 4, si los datos introducidos no cumplen los requisitos mínimos (longitud), están vacíos o la descripción ya existe, el sistema mostrará un aviso para la corrección del dato.

Tabla 12: Requisito de modificación de un tipo de instalación.

ID	13
Caso de uso	**Eliminación de un tipo de instalación**
Actores	Administrador
Propósito	Supresión de un tipo de instalación
Resumen	1. El sistema mostrará todos los tipos de instalación. 2. El actor seleccionará el tipo de instalación a eliminar. 3. El sistema mostrará un diálogo de confirmación. 4. El actor aceptará el diálogo. 5. El sistema borrará el tipo de instalación.
Precondición	El usuario debe encontrarse en la vista de gestión de parámetros
Postcondición	El tipo de instalación es eliminado
Extensión síncrona	En 4, si el usuario no acepta el diálogo de confirmación, el tipo de instalación no será eliminado.

Tabla 13: Requisito de eliminación de un tipo de instalación.

4.2.4 Requisitos de gestión de incidencias

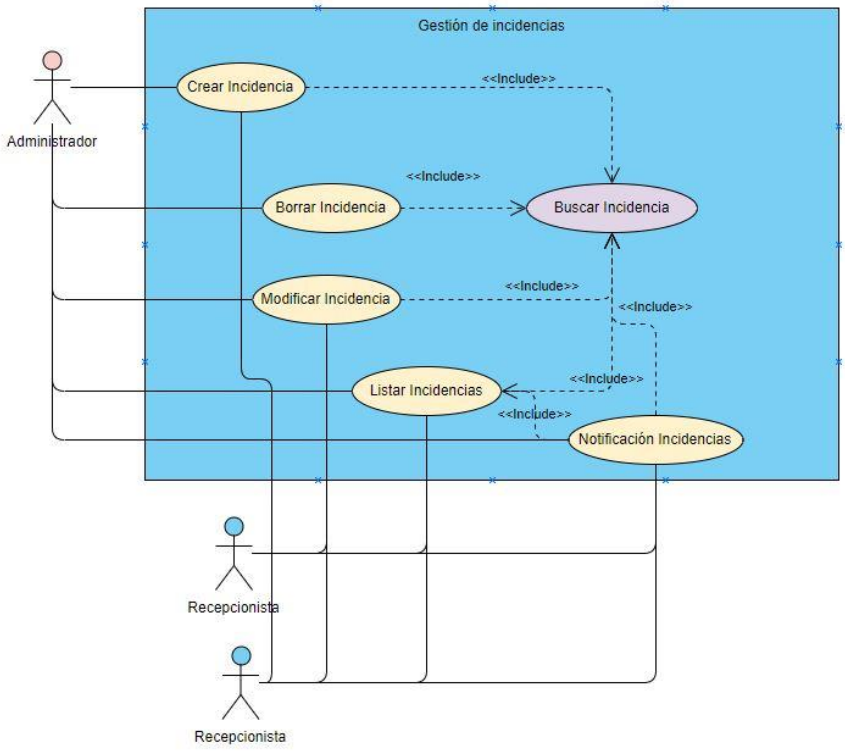

Ilustración 17: Diagrama de caso de uso de gestión de incidencias.

ID	14
Caso de uso	**Creación de una incidencia**
Actores	Administrador, recepcionista
Propósito	Añadir una incidencia
Resumen	1. El usuario introduce la información específica de la incidencia que va a crear (descripción, tipo de instalación, tipo de incidencia). 2. Los datos restantes (fecha, usuario, estado) son añadidos automáticamente por el sistema. 3. El sistema crea el nuevo tipo de instalación.
Precondición	El usuario debe encontrarse en la vista de gestión incidencias
Postcondición	La incidencia es creada y almacenada
Extensión síncrona	En 1, si los datos introducidos no cumplen los requisitos mínimos (longitud) o están vacíos, el sistema mostrará un aviso para la corrección de los datos.

Tabla 14: Requisito de creación de una incidencia.

ID	15
Caso de uso	**Modificación de una incidencia**
Actores	Administrador, recepcionista, servicio técnico

Propósito	Modificar los datos de una incidencia
Resumen	1. El sistema mostrará todas las incidencias. 2. El usuario seleccionará la incidencia a modificar. 3. El sistema mostrará la información de esa incidencia. 4. El usuario modificará los datos necesarios, ya sea descripción, tipo de incidencia, tipo de instalación o el cambio al siguiente estado. 5. El sistema se encargará de la introducción automática de datos restantes (fecha, usuario, estado) en caso de modificar el estado. 6. El sistema actualizará los datos de la incidencia.
Precondición	El usuario debe encontrarse en la vista de gestión incidencias
Postcondición	La incidencia es actualizada y almacenada
Extensión síncrona	En 4, si los datos introducidos no cumplen los requisitos mínimos (longitud) o están vacíos, el sistema mostrará un aviso para la corrección de los datos.

Tabla 15: Requisito de modificación de una incidencia.

ID	16
Caso de uso	**Eliminación de una incidencia**
Actores	Administrador
Propósito	Supresión de una incidencia
Resumen	1. El sistema mostrará todas las incidencias. 2. El usuario seleccionará la incidencia a eliminar. 3. El sistema mostrará un diálogo de confirmación. 4. El usuario aceptará el diálogo. 5. El sistema borrará la incidencia.
Precondición	El usuario debe encontrarse en la vista de gestión incidencias
Postcondición	La incidencia es eliminada
Extensión síncrona	En 4, si el usuario no acepta el diálogo de confirmación, el tipo de instalación no será eliminado.

Tabla 16: Requisito de eliminación de una incidencia.

ID	17
Caso de uso	**Alerta de notificación de un estado de incidencia modificado**
Actores	Servicio técnico, recepcionista, administrador
Propósito	Alerta de notificación de la incidencia
Resumen	1. El usuario recibirá una alerta o verá el icono de notificaciones con revisiones pendientes de nuevos estados de incidencias. 2. El usuario verá una lista con las incidencias con un nuevo estado a las que puede acceder para su modificación.
Precondición	El usuario debe estar dentro de la aplicación web.
Postcondición	
Extensión síncrona	En 1, si el usuario es de servicio técnico, recibirá notificación en los estados: nueva, verificada. En 1, si el usuario es recepcionista, recibirá notificación en el estado ‘reparada’.

Tabla 17: Requisito de alerta de una notificación.

ID	18
Caso de uso	**Filtrado de incidencias según su estado**
Actores	Administrador, recepcionista, servicio técnico
Propósito	Filtrar las incidencias según su estado actual
Resumen	1. El sistema mostrará todas las incidencias. 2. El usuario seleccionará el estado deseado a filtrar. 3. El sistema mostrará todas las incidencias que coincidan con el estado seleccionado.
Precondición	El usuario debe encontrarse en la vista de gestión incidencias
Postcondición	
Extensión síncrona	

Tabla 18: Requisito de filtrado según estado.

ID	19
Caso de uso	**Búsqueda de una incidencia por sus datos**
Actores	Administrador, recepcionista, servicio técnico
Propósito	Obtener ciertas incidencias con una condición dada
Resumen	1. El usuario visualizará la tabla de incidencias. 2. Introducirá en cualquiera de las columnas el dato sobre una incidencia que desee mostrar. 3. El sistema mostrará las incidencias que coincidan con el criterio de búsqueda.
Precondición	El usuario debe encontrarse en la vista de incidencias
Postcondición	
Extensión síncrona	

Tabla 19: Requisito de búsqueda de una incidencia por datos.

ID	20
Caso de uso	**Ordenación de incidencias por sus datos**
Actores	Administrador, recepcionista, servicio técnico
Propósito	Obtener ciertas incidencias con un orden específico
Resumen	1. El usuario visualizará la tabla de incidencias. 2. Clicará en cualquiera de las columnas que desee ordenar. 3. El sistema mostrará las incidencias en el orden proporcionado.
Precondición	El usuario debe encontrarse en la vista de incidencias
Postcondición	
Extensión síncrona	

Tabla 20: Requisito de ordenación de incidencias por sus datos.

4.3 Diagrama de clases

Una vez analizados los requisitos y sus respectivos casos de uso, se puede proceder a la realización del modelado de datos de la aplicación. Un diagrama de clases describe la estructura de un sistema mostrando sus clases, operaciones, atributos y relaciones entre objetos.

Como se verá en el punto 5 de esta memoria, se ha realizado una arquitectura por capas con inversión de dependencias. Éstas forman interfaces, pero no se muestran en el diagrama de clases debido a que se haría imposible su legibilidad, al igual que con las operaciones y los tipos de los atributos. Además, éste muestra los modelos creados a partir de las entidades (se explicará en el punto 5), pero debido a que las entidades mantienen los mismos campos que los de la base de datos, se ha realizado el diagrama completo de los modelos.

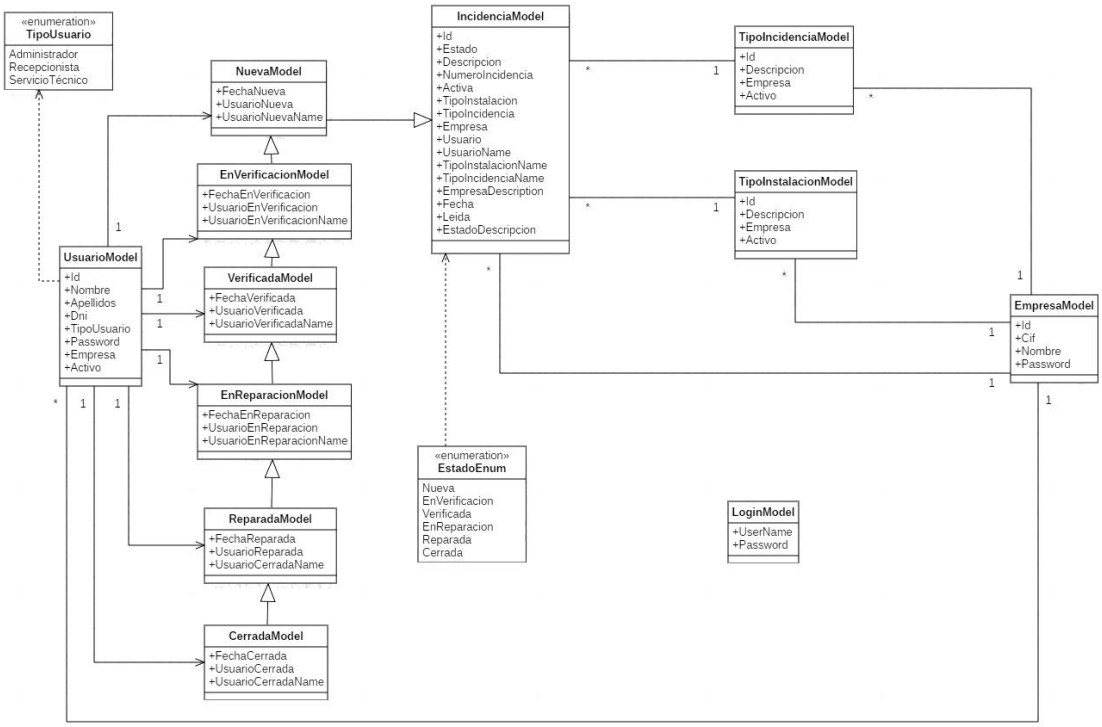

Ilustración 18: Diagrama de clases.

Como se puede observar, se dispone de cinco clases principales: IncidenciaModel, EmpresaModel, TipoIncidenciaModel, TipoInstalacionModel y UsuarioModel. EmpresaModel dispone de relaciones con todas las principales ya que, se utiliza para diferenciar las incidencias, tipos y usuarios de unas empresas con otras. Los tipos a su vez se relacionan con IncidenciaModel para ayudar a describir y filtrar mejor las incidencias. IncidenciaModel y UsuarioModel no se encuentran relacionados de forma directa, ya que una incidencia podrá pasar por distintos estados. Esto se ha

solucionado de tal forma que UsuarioModel se relaciona con cada clase estado de una incidencia. Además, existe una clase LoginModel que es la encargada del inicio de sesión y hay dos enumeraciones que contienen los tipos de usuarios y los estados por los que pasa una incidencia.

Como se puede observar, los distintos estados se han solucionado de forma que hereden unas clases de otras, con el objetivo de mantener los atributos de incidencia iniciales y agregar únicamente los nuevos que se hayan modificado o creado al cambiar de estado. De esta forma, no hay necesidad de crear nuevos campos en la base de datos ni tablas adicionales a las que guardar objetos con los nuevos estados.

4.5 Bocetos

Una vez analizado todo el problema planteado, hay que seguir con el diseño de la interacción del usuario con la interfaz. Los prototipos son representaciones concretas que simulan la interacción con la interfaz de un sistema. Tienen el objetivo de explorar aspectos interactivos del sistema, incluyendo su funcionalidad, usabilidad y accesibilidad.

El prototipado posee varias técnicas para su realización, pero para este proyecto, se ha utilizado la de bocetos. Esta técnica es una forma de representar las primeras impresiones del espacio de trabajo de la interacción. Para ello, se ha utilizado la herramienta Balsamiq.

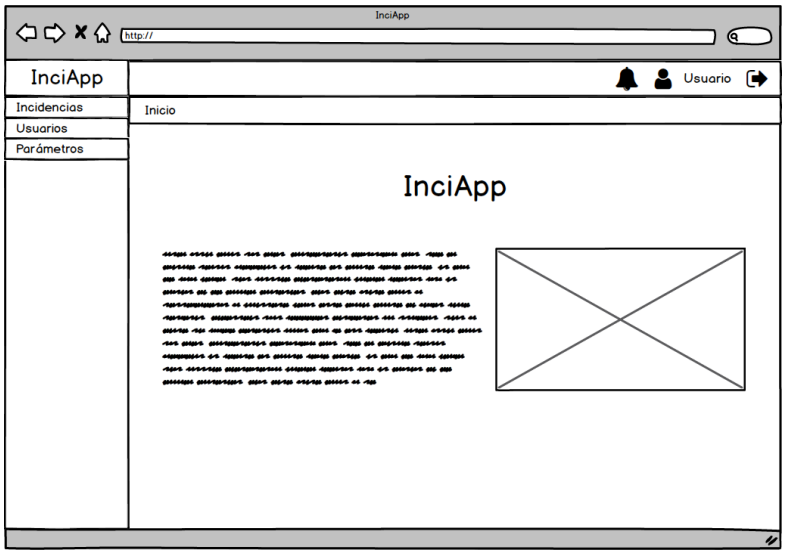

Ilustración 19: Boceto de la pantalla de inicio.

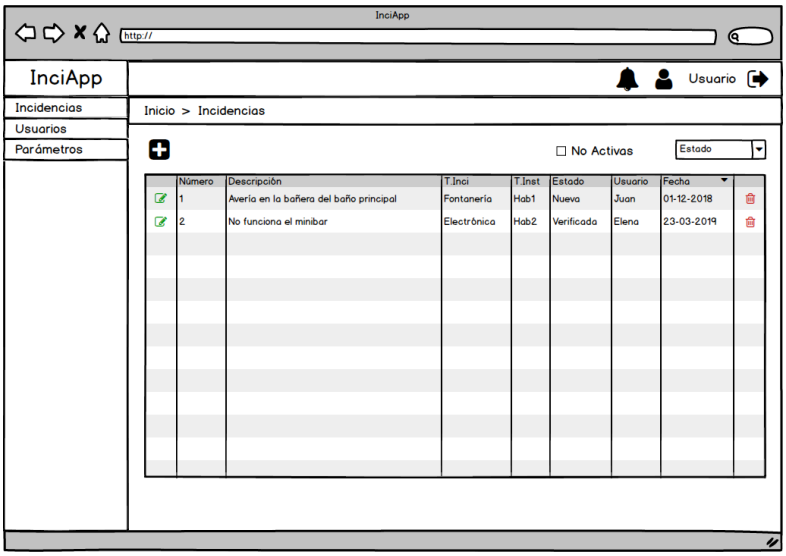

Ilustración 20: Boceto de la vista de incidencias.

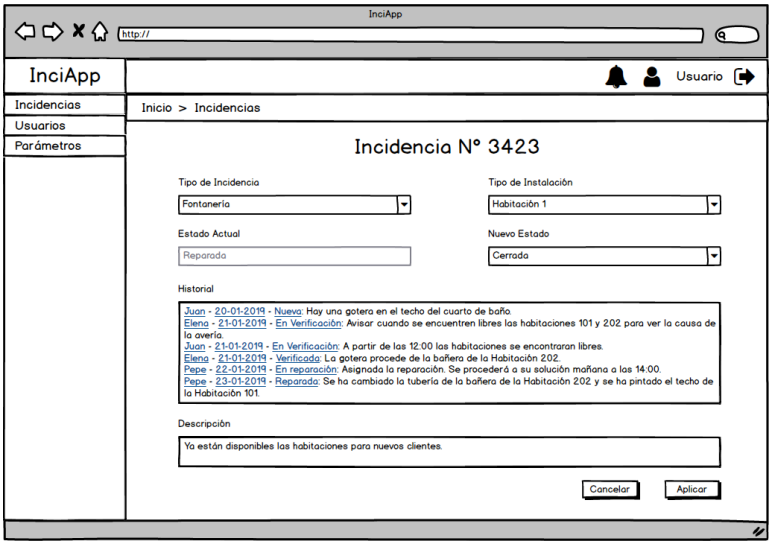

Ilustración 21: Boceto de la vista de editar incidencia.

Como se puede observar, en las ilustraciones anteriores, se ha intentado buscar un diseño que facilite las funciones del usuario a través de estándares de diseño, iconos reconocibles para el usuario, etc.

Durante el proceso de desarrollo, se observó que la interacción que podría tener el usuario para mostrar el estado actual y para cambiar de estado, éste último con un desplegable que únicamente daba opción a seleccionar el siguiente estado, no era la más adecuada. Por lo que buscó simplicidad, la facilidad para el uso de nuestra interfaz de cara al usuario, así que se rediseñó la vista como se puede observar en la Ilustración 22, eliminando los dos campos de estado y añadiendo unas migas de pan donde el usuario puede ver cuál es el estado actual y un botón que directamente pasa al siguiente.

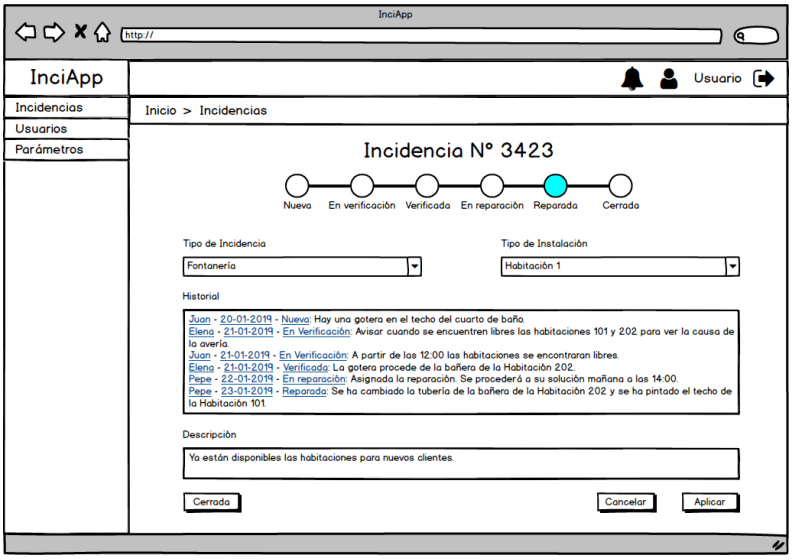

Ilustración 22: Prototipo de la vista de editar incidencias simplificada.

5. Diseño de la solución

Ya se ha analizado el problema, se han obtenido los requisitos, casos de uso, diagrama de clases y los bocetos. Por lo que ahora ya se puede diseñar la solución.

5.1 Arquitectura

Según Gerald Weinberg [7], *si los constructores construyeran los edificios como los programadores escriben los programas, el primer pájaro carpintero que apareciera destruiría la civilización*. (G. Weinberg, 2010).

Todo lo que conocemos debe tener una organización y una estructura para que funcione como se desee. Toda solución software se debe diseñar y crear con el mantenimiento en mente. En este proyecto, se ha elegido una arquitectura de “N capas” [16], en la que se hace una segmentación lógica de componentes, además de la separación física existente: cliente, servidor web y servidor de base de datos.

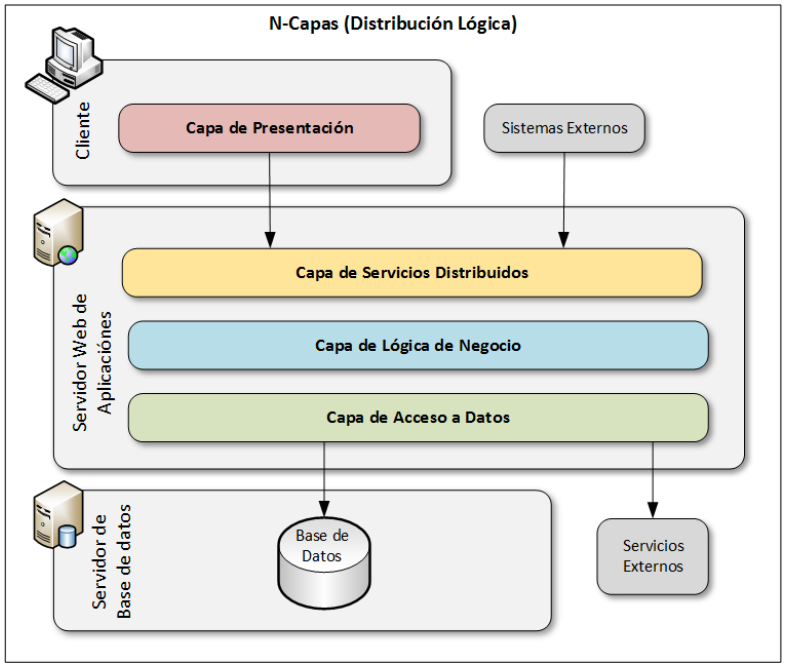

Ilustración 23: Arquitectura de "N capas".

Como se puede observar en la Ilustración 23, existe una clara de distinción de cada capa, en la que cada una posee una función que se detallará a continuación.

- La capa más inferior que podemos ver, la de acceso a datos, es la encargada de proporcionar acceso a los datos en la base de datos a través de una entidad-relación.
- La siguiente capa, la de lógica de negocio, es la que proporciona toda la funcionalidad de la aplicación.
- La capa de servicios distribuidos es la encargada de administrar los servicios REST [1] y contiene los puntos finales de comunicación (*endpoints*), que proporcionan información necesaria para hacer frente a un servicio.
- La última capa es la de presentación, que es la responsabilizada de presentar los resultados al usuario y de la recogida de sus datos.

Esta arquitectura posee diversas ventajas como:

- Desacople: permite la posibilidad de reutilizar o variar la lógica de forma cómoda y rápida y sin tener que modificar la aplicación completa, únicamente la zona a reformar.
- Mejora en pruebas: incrementa la capacidad de implementar pruebas de una forma apropiada.
- Alta cohesión: cada capa se responsabiliza de una tarea determinada.
- Mejora del mantenimiento: debido al desarrollo estructurado y localizado dentro de la aplicación.

Esta arquitectura permite el uso del Principio de Inversión de Dependencias. El objetivo es desacoplar los componentes de alto nivel de los de bajo nivel de forma que sea posible llegar a utilizar los mismos componentes de alto nivel con diferentes implementaciones de componentes de bajo nivel.

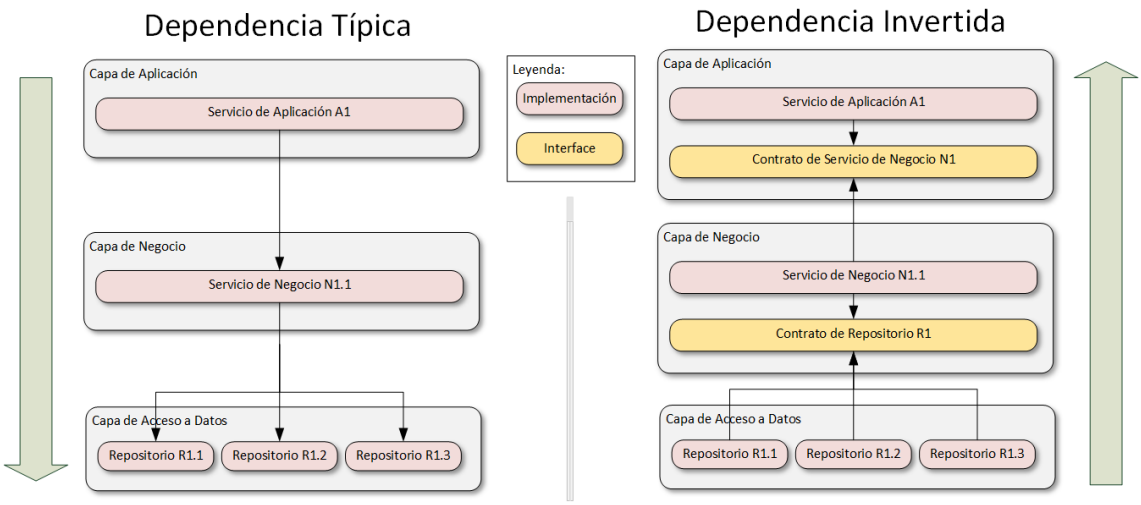

Ilustración 24: Principio de Inversión de Dependencias.

[1] REST es cualquier interfaz entre sistemas que utilice HTTP para obtener datos o generar operaciones sobre esos datos.

Por su parte, la capa de presentación, al usar Angular, posee una arquitectura propia del *framework* [5] compuesta por distintas partes:

- Módulos: es una agrupación de código dedicado a una funcionalidad o ámbito concreto de la aplicación. Se define mediante una clase decorada con *@NgModule.* Este decorador recibe un objeto con metadatos que definen al módulo. Éstos pueden ser vistas que pertenecen al módulo, declaraciones accesibles o importadas para otros módulos, servicios que se necesitan, etc.
- Componentes: controla una zona de espacio de la pantalla. Contiene propiedades y métodos que están disponibles en su *template*.
- *Templates*: es lo que permite definir la vista de un componente en HTML, pero es mucho más enriquecido que éste.
- Metadatos: se incorpora al componente para indicar cuál es su *template*, estilos, selector[2], etc.
- *Data Binding*: es una forma de actualizar valores y convertir respuestas de los usuarios en acciones específicas. Se puede manifestar de distintas formas: hacia o desde el HTML o bidireccionalmente.
- Directiva: transforman el *template* de manera dinámica a medida que se encuentran estas directivas en el HTML, ya sea para realizar bucles, comprobaciones, estilos, etc.
- Servicios: todo lo que la aplicación necesita, ya sean constantes, lógica de negocio, peticiones al servidor, etc. se encapsula dentro de ellos.
- Inyección de dependencias: se utiliza para proporcionar instancias de una clase a otras sin necesidad de crear dichos objectos. Se suele utilizar para inyectar servicios en los constructores de los componentes.

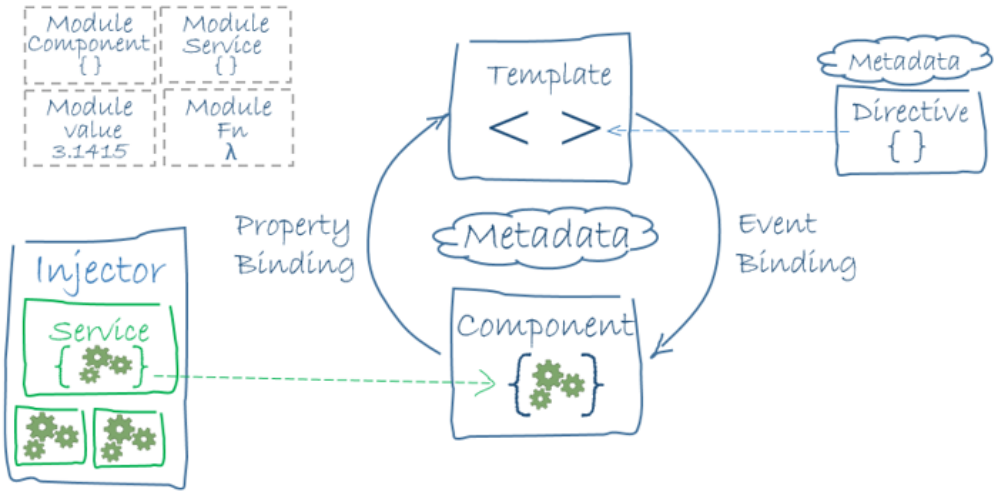

Ilustración 25: Arquitectura de Angular.

[2] Es un selector CSS que indica que se debe instanciar un componente cuando se encuentra una etiqueta con este nombre en el HTML.

5.2 Modularización

La modularización es la división de un problema en partes funcionales independientes, que encapsule operaciones y datos. Se debe separar en funciones lógicas con datos propios y de comunicación especificados a la perfección.

Un módulo es un conjunto de instrucciones que efectúan una tarea específica bien definida, cooperan para conseguir un objetivo común y se comunican entre sí adecuadamente.

Para este proyecto, se ha modularizado de la siguiente forma en la parte del *framework* de .NET:

- Entidades: incluyen clases con los objetos equivalentes a su tabla en la base de datos.
- Configuraciones: contienen las relaciones entre entidades equivalentes a las de la base de datos.
- Modelos: incluyen clases similares a las entidades, pero con peculiaridades para el uso más fácil de los objetos en la capa de presentación. Se crearán a partir del diagrama de clases de la Ilustración 18.
- Perfiles: son los encargados de relacionar las entidades con los modelos, ya sea equivalente la relación entidad-modelo o personalizada.
- Repositorios: son los encargados de realizar las consultas a la base de datos con las condiciones y filtros que se deseen.
- Interfaces: son las utilizadas para mantener el Principio de Inversión de Dependencias.
- Controladores: son las clases que interactúan con las peticiones HTTP.

Por parte del *framework* de Angular, siguiendo su estructura propia, la modularización se ha realizado de la siguiente forma:

- Modelos: son equivalentes a los mencionados anteriormente, ya que deben comunicarse entre ellos y deben entenderse.
- Servicios: son los encargados de comunicarse con los controladores o de contener información para comunicarse entre componentes de la aplicación.
- *Routing*: ficheros que contienen los módulos a los que se debe acceder cuando se declara una nueva ruta.
- *Guard*: encargados de controlar los permisos de acceso en la aplicación, ya sean por rol de usuario o porque no se ha iniciado sesión.
- Módulos o componentes: grupo de ficheros de tipo HTML, CSS y TypeScript que en su conjunto forman o módulos o componentes.

5.3 Diseño de las comunicaciones

Una vez diseñada la arquitectura y la modularización, se ha procedido a estructurar cómo sería la comunicación entre los distintos servicios: la base de datos, los servicios web y la parte del cliente.

Una de las bibliotecas proporcionadas por .NET Framework es ADO.NET. Ofrece una serie de servicios para acceder a datos que, en nuestro caso, ha sido utilizado para conectar la base de datos con la capa de acceso a datos.

Otra biblioteca facilitada por el mismo *framework* es ASP.NET. Ésta nos proporciona la capacidad de crear servicios web a través del uso del protocolo HTTP. Gracias a ella, nos ha permitido crear una Web API con la que comunicar la capa de presentación con la de servicios.

Para finalizar, se ha utilizado la biblioteca HttpClient proporcionada por el *framework* de Angular. Ésta nos permite realizar peticiones HTTP, por lo éstas pueden ejecutar las peticiones a nuestra Web API, por lo que ya se tendría la comunicación de la capa de presentación con la de servicios.

Definidas estas conexiones, la aplicación ya estaría dotada de una comunicación completa entre los distintos niveles físicos, en nuestro caso, el servidor de la base de datos, el servidor web y el cliente.

5.4 Diseño de las interfaces de usuario

El diseño de la interfaz se ha basado durante todo su proceso de implementación en la usabilidad [12], que es la facilidad con la que las personas interactúan con una herramienta con el fin de alcanzar un objetivo concreto.

Jakob Nielsen, considerado una de las personas más respetadas en el ámbito de la usabilidad web a nivel mundial, formuló en 1995 diez principios de usabilidad que se siguen siendo la base hoy en día para cualquier web. Debido a esto, para el diseño de las interfaces de esta aplicación también se han tenido en cuenta uno de los patrones de diseño más comunes de cualquier aplicación web, ya que facilita la aplicación de los principios de usabilidad de Nielsen.

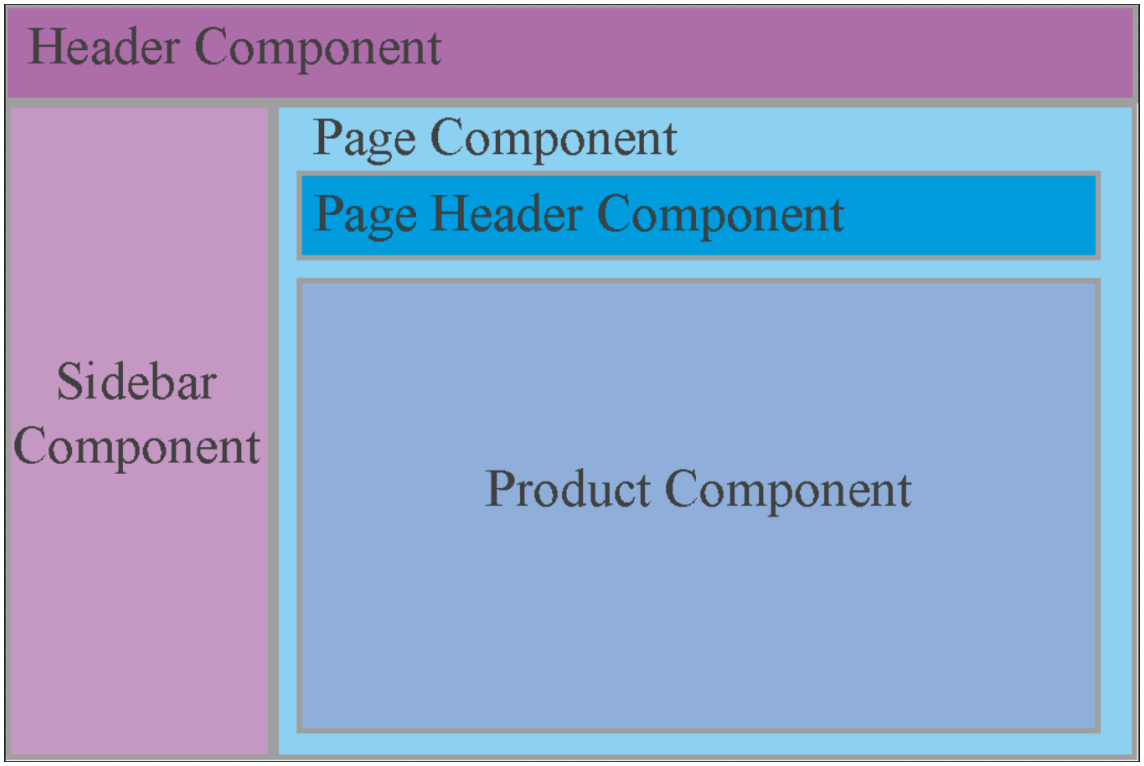

Ilustración 26: Patrón de diseño de la interfaz de usuario.

Como se puede observar en la Ilustración 26, el diseño de la interfaz se separa en distintos componentes. El componente de la cabecera y la barra lateral serán estáticos, en cambio, el resto de los componentes variarán según la vista en la que esté el usuario.

5.5 Diseño de la base de datos

Para el diseño de la base de datos, se barajó la opción de crear una tabla para cada estado de una incidencia, pero debido al coste computacional de tener que realizar más consultas, inserciones, etc. a la base de datos, se ha optado por unificarlo en una tabla, quedando el siguiente diseño:

- Tabla empresa: como se puede observar en la Ilustración 27, esta tabla contiene relaciones con todas las demás. Esto es debido a que la aplicación no tendrá una base de datos propia para cada empresa registrada, sino que será la misma para todas, así que será un tipo de filtrado de datos de cada una. El CIF se empleará para el registro de una empresa, por lo que debe ser único.
- Tabla usuario: contiene varias relaciones con la tabla incidencias, una para cada estado y una última para registrar quién realizó la última modificación. El DNI se utilizará para el inicio de sesión, por lo que debe ser único.
- Tabla tipoinstalacion: esta tabla se relaciona con la de incidencias para indicar el tipo instalación.
- Tabla tipoincidencia: ídem de lo anterior, pero con el tipo de incidencia.
- Tabla incidencias: esta tabla es la que soporta el punto fuerte de la aplicación. Contiene, además de todas las relaciones comentadas en las demás tablas, fechas para cada estado y la última fecha de modificación.

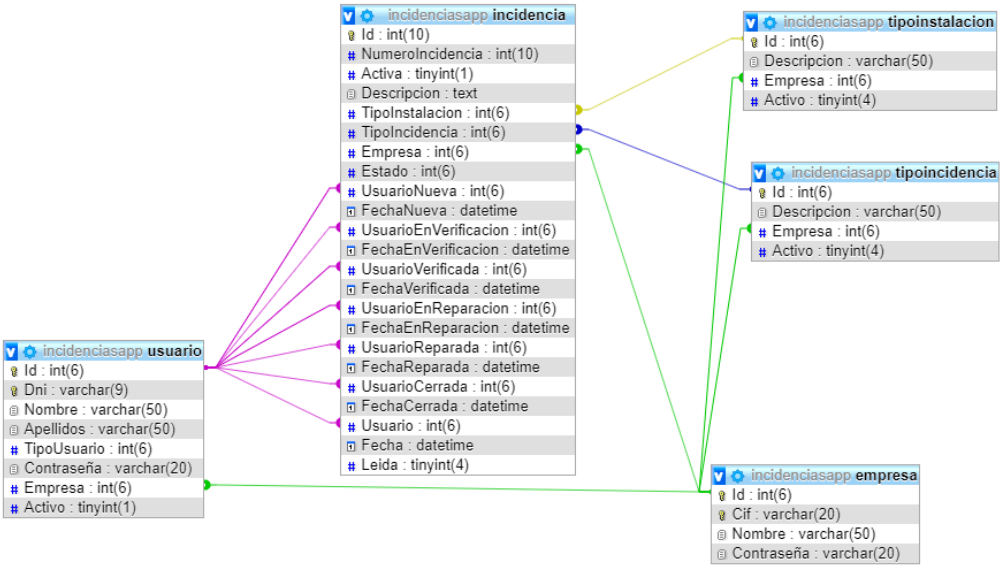

Ilustración 27: Diseño de la base de datos.

Hay campos en las tablas (Activo) que se utilizan para hacer un borrado lógico, pero esto ya se especificará con más detalle en puntos posteriores.

6. Implementación

En esta etapa de implementación se muestra todo el proceso de creación y evolución de la aplicación. Se deberá aplicar y usar todo lo comentado, analizado y diseñado en los apartados anteriores hasta llegar a la solución deseada, que será aquella que cumpla con los objetivos planteados de la aplicación. Durante el apartado 6.1, se va a introducir una imagen indicando qué capa de la arquitectura se está implementando en cada momento.

6.1 Estrategias

6.1.1 Base de datos

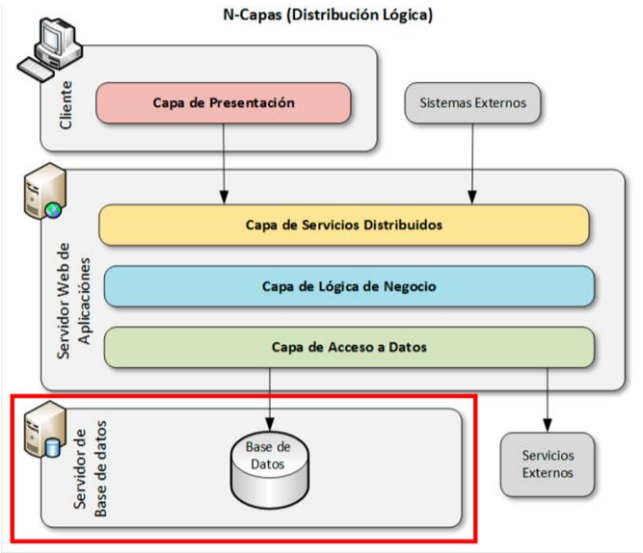

Ilustración 28: Implementación de la capa de base de datos.

Para empezar, se debe crear la base de datos MySql a través de la herramienta XAMPP. Se deben añadir columnas en las tablas para permitir el borrado lógico de información, que consiste en no eliminar una fila de la base de datos, simplemente se cambia su bit de la columna perteneciente a esta eliminación y se filtran los datos al usuario para que no pueda visualizar los borrados. De esta forma, para él, el dato se habrá eliminado, pero en caso de borrado accidental o de dependencias del dato con otras tablas, éste sigue ahí para su uso en las posibles relaciones o para una futura recuperación en caso de necesidad, sin necesidad de hacer un borrado en cascada.

6.1.2 Control de versiones

Una vez creada la base de datos, se debe crear un repositorio en GitLab y se debe clonar con SourceTree [13] para empezar con la creación de la aplicación y el control de versión de código. Durante todo el desarrollo, se ha creado una rama *feature*[3] para cada funcionalidad o proyecto desarrollado.

6.1.3 Creación de proyectos y capa de acceso a datos

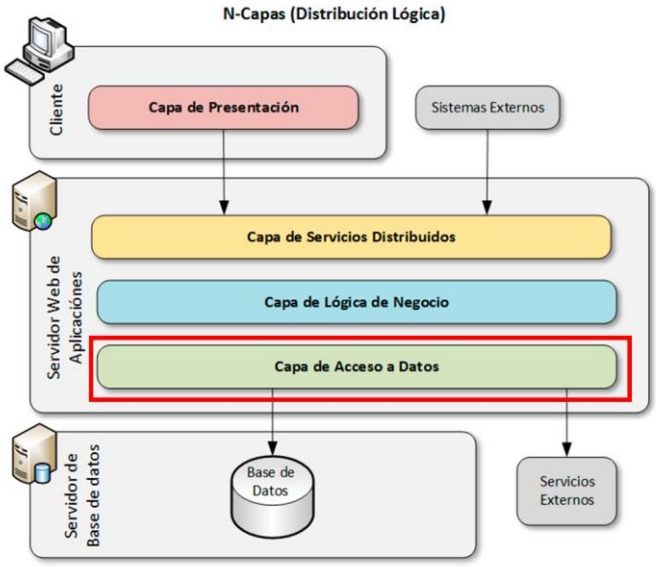

Ilustración 29: Implementación de la capa de acceso a datos.

Ya con el control de versiones en funcionamiento, se procede a la creación de los proyectos de la parte del servidor en Visual Studio 2019.

Se crean tres proyectos, uno por cada capa lógica, y se añaden los respectivos frameworks y librerías, excepto StructureMap, necesarias a través del Administrador de paquetes NuGet que ofrece el entorno, el funcionamiento de los cuales ya aparece explicado en puntos anteriores.

Dado que MySql no es compatible por defecto con Visual Studio, se siguieron los siguientes pasos [4] para poder hacerlo compatible. Una vez realizados, ya se debe utilizar ADO.NET para crear de forma automática las entidades, configuraciones, cadena de conexión [5] y el contexto. Ya con todos los nuevos ficheros, se debe comprobar que son correctos y se procede a la extracción de las configuraciones

[3] Rama de Git que indica que se está desarrollando alguna nueva funcionalidad a la aplicación. Una vez finalizada, se une a *develop,* que es la rama principal de desarrollo.
[4] https://stackoverflow.com/questions/42748396/mysql-is-not-appearing-in-choose-data-source-for-visual-studio-2017 Comentario explicativo de Alfredo Rodríguez.
[5] Indica los datos completos de la base de datos (nombre, usuario, contraseña) para que se permita la conexión a ella.

creadas en el contexto a ficheros individuales que lo relacione con su entidad, de esta forma se empieza a garantizar la modularización de la aplicación. Hecho esto, al contexto se le deben añadir referencias de los nuevos ficheros, para que se puedan realizar las relaciones correctamente.

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
        modelBuilder.Configurations.Add(new EmpresaEntityConfiguration());
        modelBuilder.Configurations.Add(new IncidenciaEntityConfiguration());
        modelBuilder.Configurations.Add(new TipoIncidenciaEntityConfiguration());
        modelBuilder.Configurations.Add(new TipoInstalacionEntityConfiguration());
        modelBuilder.Configurations.Add(new UsuarioEntityConfiguration());
}
```

Además, se inhabilita la funcionalidad por defecto activa de *Lazy Loading*. Ésta consiste en que, si se realiza una petición de consulta a una tabla de la base de datos, cargue por defecto el objeto relacionado de otra tabla. En este proyecto se ha decidido no hacerlo de forma predeterminada ya que, se consigue una mejora de rendimiento porque únicamente devuelve una fila y no varios objetos que requieren más consultas y, además, tiene la ventaja de que se puede elegir concretamente en qué peticiones hacer esa carga de relaciones.

Con todo esto anterior realizado con éxito y comprobado que en el fichero App.config existe la relación con la base de datos y los datos generados son los correctos, se debe dar por terminada la capa de acceso a datos.

6.1.4 Capa de lógica de negocio

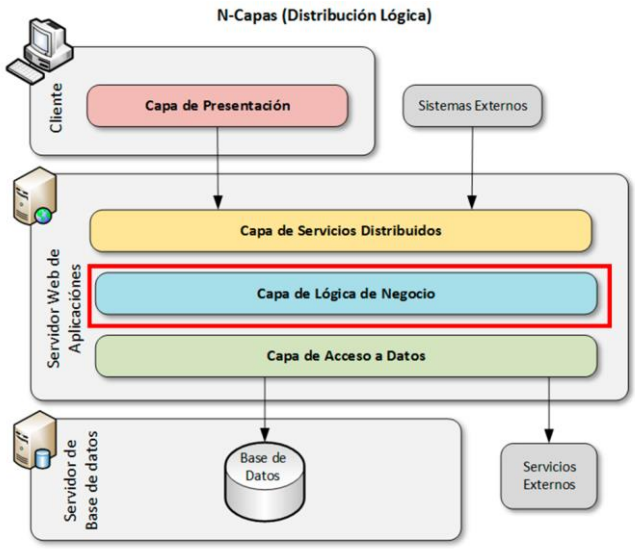

Ilustración 30: Implementación de la capa de lógica de negocio.

Ahora es el momento de pasar a la siguiente capa, la capa lógica de negocio. Para empezar, se debe añadir la referencia del proyecto anterior, ya que deberá hacer uso de sus entidades. Se deben crear los modelos según el diagrama de clases que se mostró con anterioridad. Para ello, hay que crear una clase por cada una que aparece en el diagrama. Las clases principales, mencionadas en el apartado 4.3, se crean como clases sin herencias, en cambio, las que pertenecen a los estados, deberán heredar cada una de su predecesora.

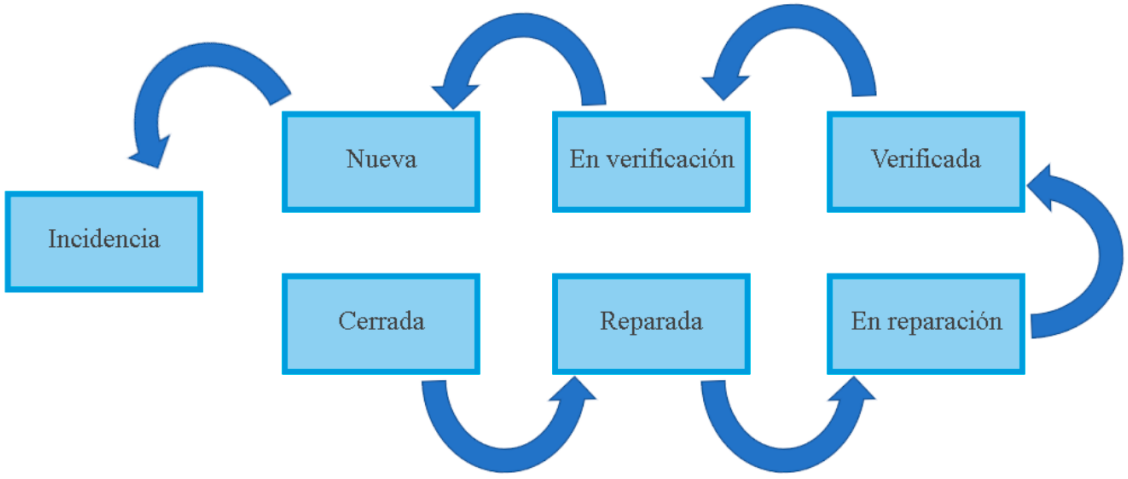

Ilustración 31: Flujo de herencia de una incidencia.

Como se puede observar en la Ilustración 31, una incidencia no puede pasar de nueva a cerrada, ya que antes de ello, debe contener todos los estados intermedios, ya

que es una herencia en cadena, no son individuales de cada estado directamente con la clase incidencia. De esta forma, se puede garantizar que el ciclo de vida por el que transcurre una incidencia es el correcto.

En algunos de los modelos, no se han añadido las colecciones u objetos que corresponderían a las relaciones entre modelos ya que, las entidades ya contienen estas relaciones. Como se verá a continuación, a pesar de no añadirlas, sí que aparecen de manera implícita, ya que se han creado propiedades únicamente para los datos que se consideraban necesarios para la capa de presentación.

Una vez creados los modelos, se deben relacionar con las entidades ya que, la base de datos no aceptará objetos que no sean compatibles con sus tablas. Para ello se usa AutoMapper [3]. Se debe crear un perfil específico para cada relación entidad-modelo, en el que se deben indicar qué propiedades de cada entidad deben relacionarse con cada propiedad del modelo, y viceversa. En la mayoría de los casos, se ha usado para relacionar los objetos de las entidades con las propiedades de los modelos, para, como se ha mencionado anteriormente, facilitar el trabajo en la capa de presentación.

```
public IncidenciaModelProfile()
{
        CreateMap<IncidenciaModel, Incidencia>()
                .ForMember(x => x.Estado, opt => opt.MapFrom(e =>
                        e.Estado.GetHashCode()));


        CreateMap<Incidencia, IncidenciaModel>()
                 .ForMember(i => i.EmpresaDesciption, opt => opt.MapFrom(e =>
                        e.EmpresaObject.Nombre))
                  .ForMember(i => i.TipoIncidenciaDescription, opt => opt.MapFrom(e =>
                        e.TipoIncidenciaObject.Descripcion))
                  .ForMember(i => i.TipoInstalacionDescription, opt => opt.MapFrom(e =>
                        e.TipoInstalacionObject.Descripcion))
                  .ForMember(x => x.Estado, opt => opt.MapFrom(e =>
                        (EstadoEnum)e.Estado))
                  .ForMember(x => x.EstadoDescription, opt => opt.MapFrom(e =>
                        Enum.GetName(typeof(EstadoEnum), e.Estado)))
                  .ForMember(i => i.UsuarioName, opt => opt.MapFrom(e =>
                        e.UsuObject.Nombre));
}
```

Además, se debe crear un fichero en el que se deben añadir todas las instancias a perfiles debido a que AutoMapper necesita ser cargado, como se mostrará más adelante, en el fichero de inicio de la aplicación para realizar su correcto funcionamiento y, de esta forma, únicamente debe instanciarse ese fichero una vez y, aunque luego sufra modificaciones, se estará garantizando la modularización ya que, esa instancia no se verá modificada por los cambios que se le hagan internamente.

```
public static void Initialize()
{
	Mapper.Initialize((config) =>
	{
		config.AddProfile<EmpresaModelProfile>();
		config.AddProfile<IncidenciaModelProfile>();
		config.AddProfile<TipoIncidenciaModelProfile>();
		config.AddProfile<TipoInstalacionModelProfile>();
		config.AddProfile<UsuarioModelProfile>();
		config.AddProfile<NuevaModelProfile>();
		config.AddProfile<EnVerificacionModelProfile>();
		config.AddProfile<VerificadaModelProfile>();
		config.AddProfile<EnReparacionModelProfile>();
		config.AddProfile<ReparadaModelProfile>();
		config.AddProfile<CerradaModelProfile>();
	});
}
```

Con las relaciones de entidad-modelo cubiertas, es el momento de crear los repositorios. En ellos se deben añadir las funciones encargadas de consultar, modificar o añadir datos y guardar cambios en la base de datos. En algunas de estas funciones, en las de consulta que necesitásemos la relación con el objeto de otra tabla, es donde se debe añadir la funcionalidad de *Include* para contrarrestar la desactivación de *Lazy Loading*. En los repositorios, es donde se hace uso del AutoMapper cada vez que se utiliza una función de ellos ya que, los modelos son clases distintas a las entidades y por lo que, si se intentase hacer una consulta, en nuestro caso se ha utilizado LINQ[6], con un tipo distinto al que debe recibir la base de datos, no funcionaría la aplicación.

```
public IncidenciaModel GetById(int id)
{
	var repo = context.Incidencias
		.Include(i => i.TipoIncidenciaObject)
		.Include(i => i.TipoInstalacionObject)
		.Include(i => i.EmpresaObject)
		.Include(i => i.UsuObject)
		.Where(i=> i.Id == id)
		.FirstOrDefault();

	return AutoMapper.Mapper.Map<Incidencia, IncidenciaModel>(repo);
}
```

Finalizados los repositorios, se debe crear una interfaz para cada uno de ellos, en los que añadir los métodos públicos de los repositorios que sean necesarios para realizar consultas a la base de datos. Al realizar este proceso de creación, se estará garantizando el Principio de Inversión de Dependencias y finalizando la implementación de la capa de lógica de negocio.

[6] LINQ contiene métodos para agilizar el proceso de redacción de consultas SQL.

6.1.5 Capa de servicios

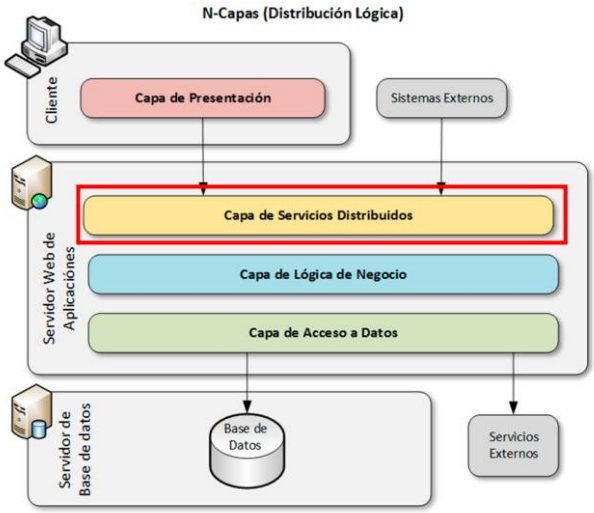

Ilustración 32: Implementación de la capa de servicios.

Llega el turno de la última capa por parte del servidor, la de servicios. El tipo de proyecto creado para esta capa es distinto a los demás. En este caso, se ha utilizado la biblioteca ASP.NET para crear un proyecto del tipo Web API. Llegados a este punto, ya se debe proceder a la instalación de StructureMap [14] únicamente en este proyecto para poder realizar la inyección de dependencias. En el fichero DefaultRegistry.cs, creado automáticamente por el paquete, se deben registrar todas las inyecciones, indicando que para X interfaz del repositorio, se use X repositorio.

```
For<IEmpresaRepository>().Use<EmpresaRepository>();
For<ITipoIncidenciaRepository>().Use<TipoIncidenciaRepository>();
For<ITipoInstalacionRepository>().Use<TipoInstalacionRepository>();
For<IUsuarioRepository>().Use<UsuarioRepository>();

. . .
```

Ahora se debe indicar dónde debe realizarse la inyección de dependencias, por lo que, en el fichero de configuración del Web API (WebApiConfig.cs), inicializamos los registros. Además, como se mencionó con anterioridad, se debe registrar en el fichero Global.asax la inicialización de los perfiles creados para el uso de Automapper, añadidos todos ellos previamente en éste.

Para finalizar la programación de esta capa, faltaría crear los controladores para cada modelo. Para ello se toma como base el que aparece por defecto en la creación del proyecto. En cada uno de ellos, se debe indicar cuál es el prefijo de la ruta que da acceso a ese controlador, en nuestro caso se ha decidido “api/” junto con el nombre del controlador. El constructor de cada controlador será el encargado de realizar la inyección de dependencias de su repositorio correspondiente o de los necesarios para cada uno de ellos.

```
private readonly IIncidenciaRepository repository;
private readonly IUsuarioRepository usuarioRepository;

public IncidenciaController(IIncidenciaRepository repository,
       IUsuarioRepository usuarioRepository)
{
       this.repository = repository;
       this.usuarioRepository = usuarioRepository;
}
```

Una vez realizada la inyección, se deben crear los métodos encargados de recibir las peticiones HTTP, indicando a cada uno de ellos su tipo (GET, POST, PUT o DELETE) y su ruta, que será añadida a la del prefijo, ya sea vacía, por id, etc. Dentro de cada función, se han controlado sus posibles casos de error, desde acceso prohibido, no encontrado, duplicado, etc. Importante destacar que, cada vez que se realiza una modificación en la base de datos, se debe llamar al repositorio utilizado para guardar los cambios ya que, si se actualiza, añade o elimina y no se realiza dicha llamada, éstos se perderían.

Finalizados los controladores, únicamente restaría la comprobación de su correcto funcionamiento, pero antes de eso, para evitar futuros errores de conexión (CORS[7]) con la capa de presentación, debemos permitir el acceso del futuro cliente al servidor. Para ello se debe indicar en el fichero Web.config, que se permita las conexiones de cualquier origen. Además, a este fichero, también se le debe incluir la cadena de conexión con la base de datos.

```
<httpProtocol>
   <customHeaders>
      <add name="Access-Control-Allow-Origin" value="*" />
   </customHeaders>
</httpProtocol>
```

Ahora se procede al uso de Postman para la comprobación de que los controladores funcionan correctamente. Para ello se debe indicar la ruta con el método HTTP que se corresponda, y en los casos necesarios, se debe añadir al cuerpo del método el objeto del tipo del modelo a introducir, actualizar o borrar. En este proyecto, se ha utilizado el PUT para actualizar y el POST para añadir.

[7] *Access-Control-Allow-Origin*

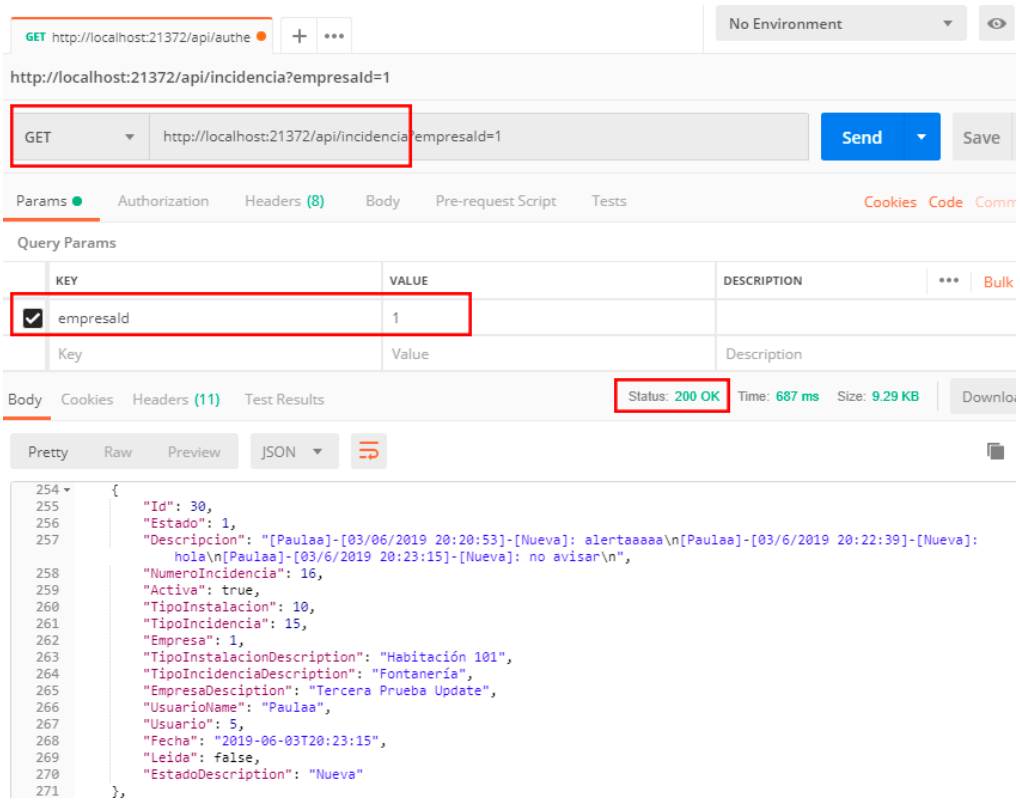

Ilustración 33: Ejemplo de petición HTTP en Postman.

En el ejemplo propuesto en la Ilustración 33, se puede observar que se realiza una petición GET con la ruta que accede al controlador de incidencias, pero como ésta, además de la ruta, necesita un parámetro indicando el id de la empresa con la que buscar las incidencias, se le añade indicando el nombre y el valor. Finalmente, se observa que la petición resulta satisfactoria (estado 200 OK) y muestra todas las incidencias que coinciden con el criterio de búsqueda.

6.1.6 Capa de presentación

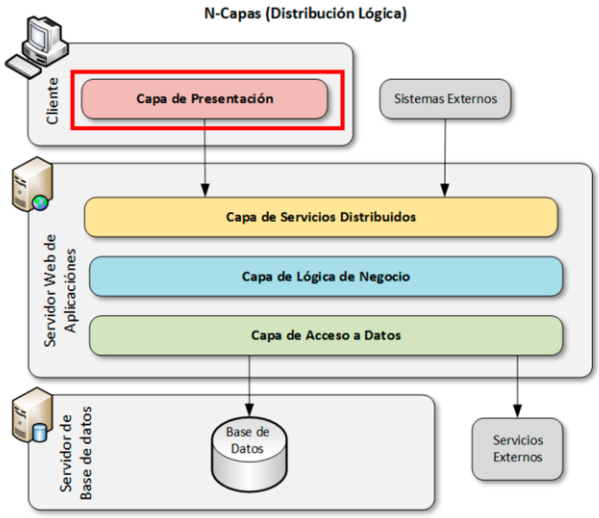

Ilustración 34: Implementación de la capa de presentación.

Para finalizar la implementación de la aplicación, faltaría crear un proyecto de Angular. Para ello, Angular ofrece una extensa documentación [2] y un tutorial con los pasos previos para la creación de éste. Se debe crear en Visual Studio Code y se debe proceder a la implementación de la parte ligada al servidor, que serían los modelos y los servicios. Los modelos deben ser equivalentes a los del servidor ya que, en caso contrario, no podrían realizarse operaciones debido a errores de tipado de objetos. Los servicios deben contener las funciones para realizar las peticiones HTTP a la Web API, indicando la ruta, parámetros, objetos en el cuerpo, etc. necesarios para que ésta pueda reconocerlas. Los parámetros se han utilizado para informar sobre el usuario conectado y su empresa.

Una vez se tiene la parte correspondiente de conexión y relación del proyecto con el servidor, se procedió a crear la implementación a partir de los bocetos, pero debido a la complejidad y al esfuerzo de crear sin ninguna base una interfaz completamente adaptable y que no se llegase al funcionamiento deseado, se optó por buscar una plantilla [8] que agilizase este proceso de creación.

Ya adaptada la plantilla para este proyecto, se debe proceder a la creación de cada vista (la de incidencias, usuarios y parámetros), la instalación de los componentes y su importación en los módulos de la vista en la que se utilicen. Para todos los *templates* de la aplicación, se ha hecho uso de Bootstrap, para evitar la creación de estilos para cada elemento, aunque se han hecho modificaciones personalizadas para alguno de ellos.

Como es propio del *framework* de Angular, se ha hecho uso de la inyección de dependencias, que ésta se introduce en los constructores de los componentes para

[8] https://github.com/start-angular/SB-Admin-BS4-Angular-8

hacer uso de los servicios creados o de los modelos importados que se usen en la aplicación, ya sean ventanas modales, mensajes de error, etc.

Empecemos por el inicio de sesión y el registro. El primero de ellos, debe realizar una petición HTTP para recibir el permiso de acceso a la aplicación. En caso de no lograrlo, se deben reciben mensajes de error indicando el motivo de éste. En caso contrario, el usuario accede a la aplicación. Para el control de la sesión, se ha usado Local Storage, guardando en un elemento el rol, el id de la empresa y el id y el nombre de usuario, y en otro elemento, como que el usuario se ha conectado con éxito. Con estos datos, ya se puede bloquear acceso por roles o por inicio de sesión a las rutas. El proceso de registro se ha implementado de tal forma para que el usuario introduzca los datos de su nueva empresa y se proceda a la creación de ésta, y el sistema debe proporcionar un usuario administrador por defecto y debe informar de éste para que se pueda acceder por primera vez a la aplicación.

Posteriormente, se debe continuar con la creación y asignación de las rutas para cada vista de la aplicación. Se va a mostrar el código en el que aparece el control del acceso por el rol de usuario, pero no aparece el de si el usuario está conectado o no. Esto es debido a que estas rutas son hijas de las rutas principales del proyecto, donde ya se controla este aspecto, por lo que sería redundante incluirlo en éstas también.

```
const routes: Routes = [
    {
        path: '',
        component: LayoutComponent,
        children: [
            { path: '', redirectTo: 'inicio', pathMatch: 'prefix' },
            { path: 'inicio', loadChildren:
                      './dashboard/dashboard.module#DashboardModule' },
            { path: 'incidencias', loadChildren:
                      './../incidencias/incidencias.module#IncidenciasModule' },
            { path: 'usuarios', loadChildren:
                      './../usuarios/usuarios.module#UsuariosModule',
                      canActivate: [RoleGuard]},
            { path: 'parametros', loadChildren:
                      './../parametros/parametros.module#ParametrosModule',
                      canActivate: [RoleGuard] }
        ]
    }
];
```

Comprobado el funcionamiento del bloqueo de rutas y permisos de acceso, se debe proceder al desarrollo de cada vista o componente con el que tenga que interactuar el usuario. La vista de usuarios y parámetros es similar en cuanto a su funcionamiento, ya que ambas utilizan ventanas modales tanto para crear, editar o eliminar datos de sus respectivas tablas de Ag-Grid [1].

Dado que este componente no incorpora botones por defecto para la funcionalidad de eliminar o editar un elemento, se procedió a la creación de dos componentes para suplir dichas funciones. Éstos contienen un icono identificativo para su fácil reconocimiento visual. Su funcionalidad es la de identificar qué tabla se ha pulsado y qué fila. Una vez reconocidos estos datos, se tratan de la forma que nos guste, en este caso, abriendo un modal con los datos a modificar o a borrar.

El desarrollo de la vista de incidencias es la más compleja, ya que contiene un filtro para cada estado cuando se muestra el listado y una vista personalizada para la edición de ellas ya que, a través de un modal, no sería la forma adecuada de editar una incidencia, por lo que se accede a otra vista distinta. Ésta se debe cargar con los datos que adquiere a partir del id de la incidencia que aparece en la ruta de acceso y la vista se va personalizando dependiendo del estado en que se encuentre la incidencia seleccionada. En las vistas de incidencias, tanto en ésta como en la general, queda registrada toda modificación que realice el usuario, desde la lectura de nuevas modificaciones en incidencias hasta de la creación o actualización de éstas.

En la vista de edición, los usuarios podrán diferenciar dos funcionalidades, en la parte derecha verán unos botones para guardar o cancelar cambios, mientras que en la izquierda un botón con el nombre del estado siguiente al que cambiar una incidencia. Éste contendrá el siguiente estado del actual, que se podrá visualizar en las migas de pan en la parte superior del componente de edición, permitiendo una rápida visual del estado, reconociendo e identificando fácilmente la tarea de cambiar de estado.

Para la creación o modificación de incidencias, se puede aprovechar el componente de ng-select de tal forma en la que un usuario no debe recorrer todo el listado de tipos de incidencia o parámetros ya que, a diferencia del select de HTML, éste permite la introducción de texto para buscar en la lista el elemento deseado.

Ahora se procede a la creación de un elemento que se relaciona, en parte, con la vista de edición de incidencias, el de alertas de notificaciones de nuevos estados de incidencias. Éste muestra un número con la cantidad de notificaciones que están por leer entre todos los usuarios. Se recibe un aviso y se actualiza el número cada vez que modifica o añade alguna incidencia cualquier usuario. Al clicar el elemento, se despliega una lista que muestra las incidencias pendientes de lectura. Si se pulsa en alguna de ellas, te redirecciona directamente a su vista de edición, quedando ésta marcada como leída y así se evita que otro usuario acceda para hacer la misma tarea que podría estar realizando ese usuario en ese mismo momento.

El elemento que indica el usuario conectado, al clicar, debe mostrar un desplegable con dos botones, el de editar tu perfil y el de cerrar sesión. El primero de ellos debe permitir modificar cualquiera de tus datos excepto el rol y el DNI. El elemento de cerrar sesión, una vez confirmado el modal para realizar esa acción, debe regresar a la pantalla de inicio de sesión y debe borrar del Local Storage todo lo que esté relacionado con el inicio de sesión anterior. Aparte de eso, se debe hacer un refresco de la página debido a que, al no recargar los componentes desde cero, mantenía datos anteriores en caso de conectarse un usuario de otra empresa distinta al anterior.

Durante el desarrollo de todo el proyecto, se deben mostrar mensajes de error personalizados dependiendo del código de estado que devuelva el servidor en la

respuesta de las peticiones a su Web API. Estos mensajes se muestran a través del componente ngx-toastr. Además, se han creado estilos para los elementos requeridos en los formularios y mensajes de error en caso de no estar introduciendo correctamente los datos.

La interfaz tanto para el usuario administrador como para el recepcionista es exactamente la misma, sólo que este último no debe tener acceso a ciertas vistas o funciones, como el eliminado de una incidencia. La interfaz para el usuario con rol de servicio técnico se ha personalizado completamente. Desaparece la barra superior de navegación y se colocan sus elementos el menú desplegable de la parte izquierda. Las vistas sí que poseen *scroll* debido a que se visualizará en un dispositivo móvil y la tabla que muestra la información de las incidencias no será igual de completa ya que no se necesitaría visualizar tantos datos sobre las incidencias, aunque al editar una incidencia la información es exactamente la misma, pero con la interfaz adaptada.

Para la implementación de la interfaz, se ha tenido en cuenta el diseño del apartado 5.4, por lo que ahora se explicará cómo se ha resuelto. A continuación, se especifican y explican los principios de Nielsen aplicados en la interfaz.

- Visibilidad del estado del sistema: este principio se basa en que el usuario debe saber en todo momento que está pasando en la web, por lo que se han usado migas de pan para indicar la vista en la que se encuentra el usuario en todo momento y además, como se mencionó en el punto 4.5, se simplificó la idea del estado actual y pasar al siguiente creando un indicador de proceso en el que se encuentra en cada momento una incidencia.

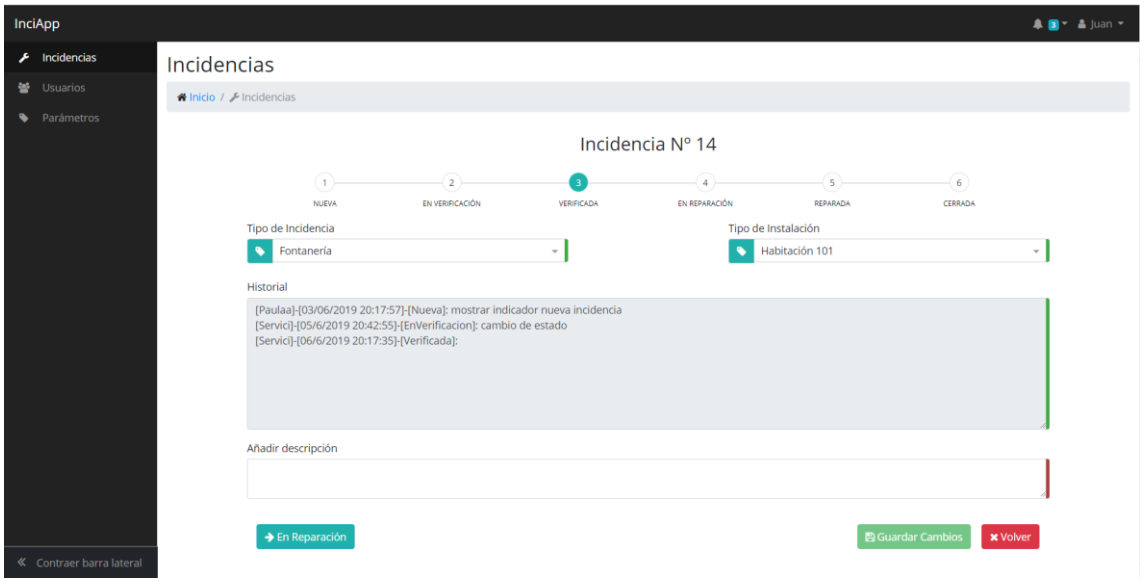

Ilustración 35: Principio de visibilidad del estado del sistema.

- Relación entre el sistema y el mundo real: los elementos de la interfaz deben ser consistentes con los del mundo real. Se han aplicado una serie de iconos que facilitan la identificación y entendimiento de los elementos en la aplicación.

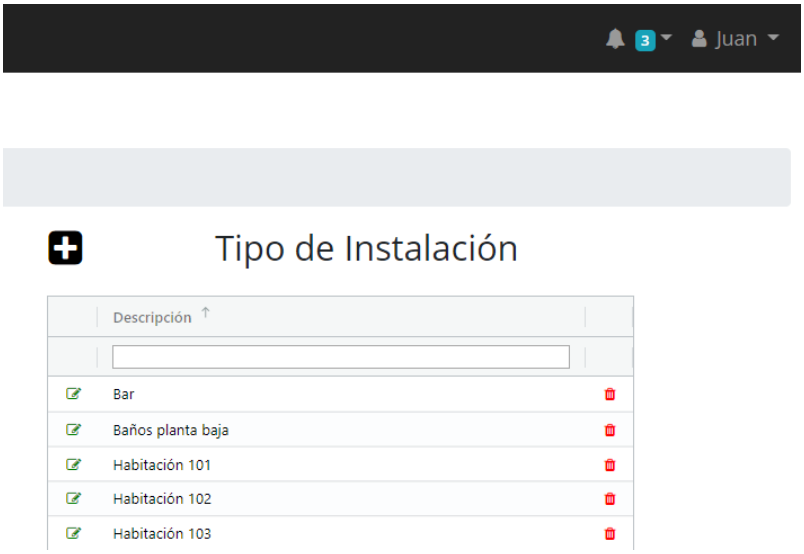

Ilustración 36: Principio de relación entre el sistema y el mundo real.

- Control y libertad del usuario: debe cederse al usuario una posibilidad de subsanar un error, es por esto por lo que la aplicación permite corregir su perfil de usuario o borrar cualquier elemento que haya creado de forma errónea.
- Consistencia y estándares: se deben seguir los convenios establecido para ciertos iconos o tareas. En nuestro caso, se ha utilizado el típico icono de menú desplegable en caso de reducir la resolución, colores en los botones para aceptar o cancelar (verde o rojo), menú de navegación a la izquierda, icono de la aplicación para volver al inicio, etc.

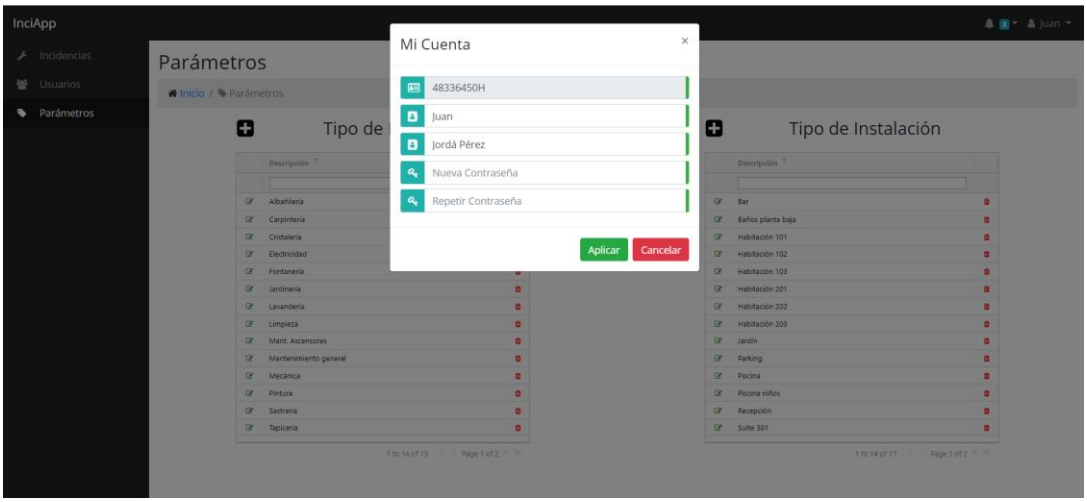

Ilustración 37:Principio de control del usuario y de consistencia y estándares.

- Prevención de errores: hay que prevenir al usuario de cometer cualquier error. Por eso se han bloqueado los botones para aplicar cualquier cambio si no se han rellenado los campos correspondientes, estos campos se encuentran marcados visualmente indicando su rellenado, mensajes de error en tiempo real para no tener que realizar las operaciones de nuevo, etc.

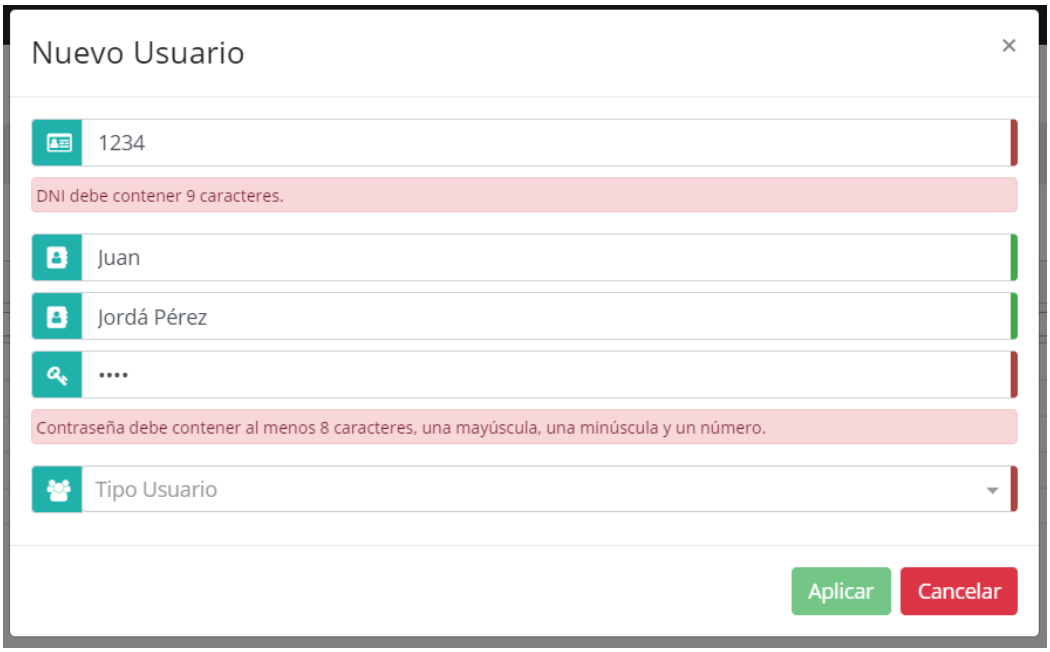

Ilustración 38: Principio de prevención de errores.

- Reconocer frente a memorizar: es mejor reconocer que obligar recordar objetos o acciones para llegar a realizar un objetivo. Se ha hecho uso de un menú simplificado para que el usuario a partir del icono reconozca cada elemento del menú lateral que puede visitar.

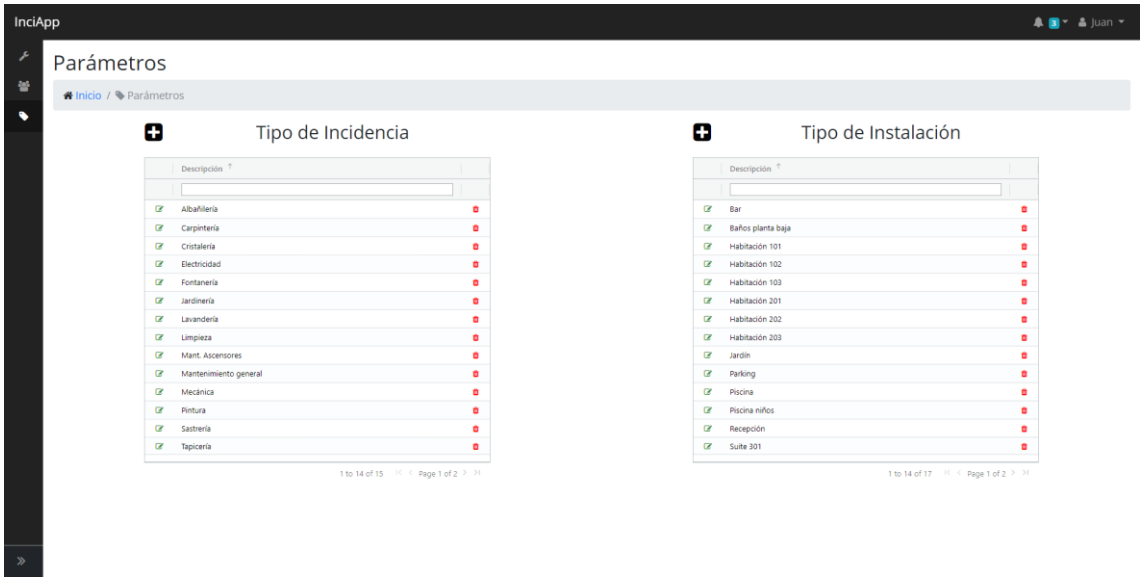

Ilustración 39: Principio de reconocer frente a memorizar.

- Flexibilidad y eficiencia de uso: a menudo hay que acelerar o crear atajos a los usuarios más expertos para mejorar su usabilidad, sin impedir que los más noveles puedan lograr el mismo objetivo. Debido a esto, se ha creado el símbolo de alerta donde aparecerán las nuevas incidencias, en las que el usuario accederá directamente a la seleccionada sin necesidad de ir al menú y buscar una incidencia que pudiese tener un nuevo estado.

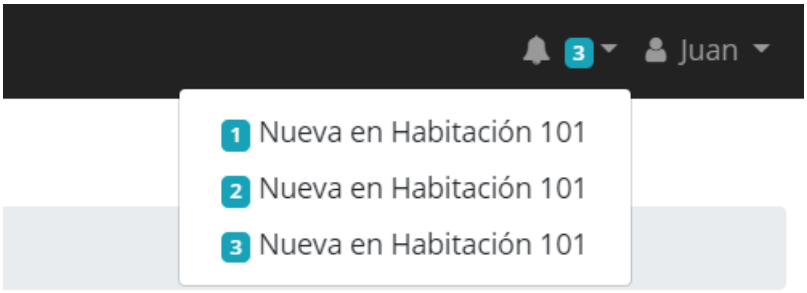

Ilustración 40: Principio de flexibilidad y eficiencia de uso.

- Diseño estético y minimalista: las páginas no deben mostrar información innecesaria, ya que eso distrae al usuario y puede dificultar su navegación. Debido a este principio, únicamente se han añadido elementos funcionales para agilizar y asegurar el correcto uso de la aplicación.

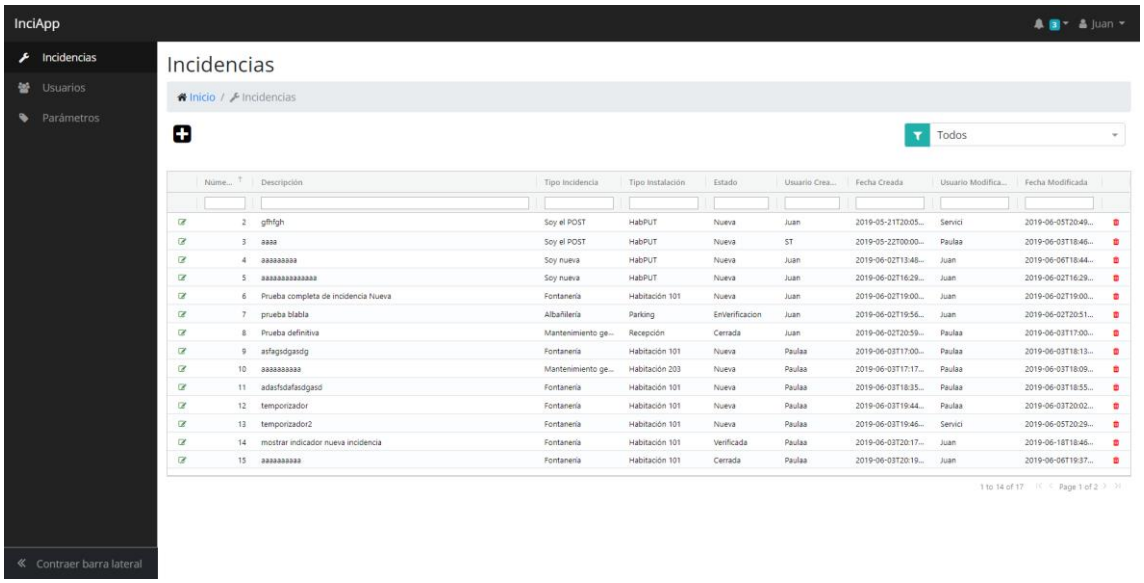

Ilustración 41: Principio de diseño estético y minimalista.

- Ayuda a los usuarios a reconocer, diagnosticar y corregir los errores: hay que intentar que todos los errores que aparecen en la aplicación estén en un lenguaje entendible para cualquier posible usuario, así que se han personalizado los mensajes dependiendo del código de error que llegue por parte del servidor. Se muestra un ejemplo del error al crear un nuevo usuario con un DNI ya existente.

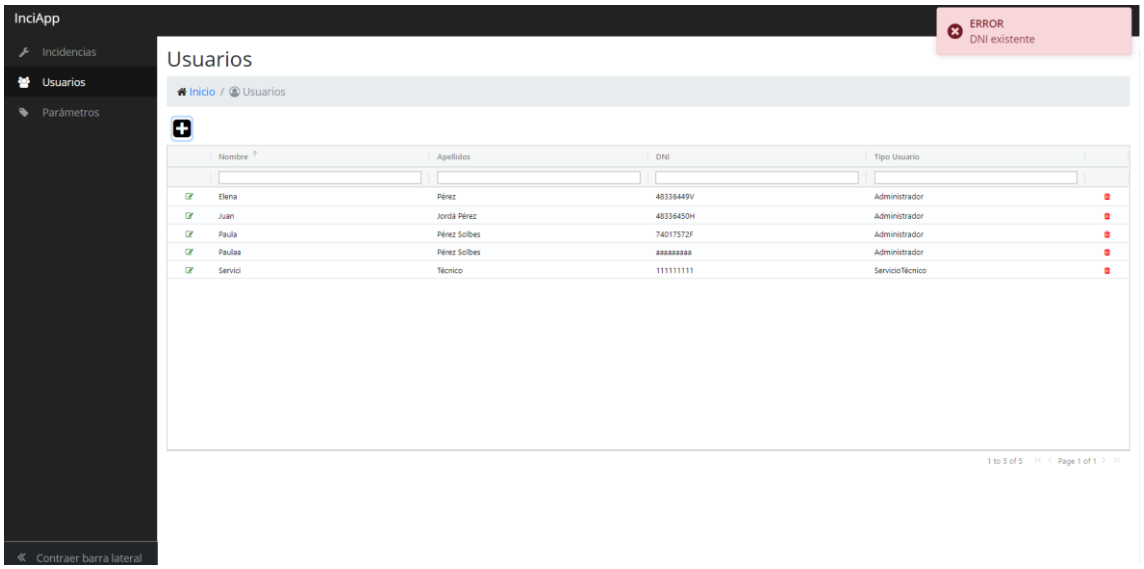

Ilustración 42: Principio de ayuda a reconocer, diagnosticar y corregir los errores.

Además de estos principios, se han adoptado una serie de guías para contribuir más con la mejora de la usabilidad del usuario.

- La estructura de la interfaz se ha trabajado con bloques rectangulares y siguiendo los estándares con menús de navegación en los márgenes y una página principal.
- La tipografía se ha utilizado correctamente en cada significado del elemento que muestre en la interfaz, aumentando o disminuyendo el tamaño dependiendo del nivel de importancia, ya sea un título, subtítulo, etc.
- Los enlaces aparecen diferenciados de otros elementos porque el puntero cambia de la forma de flecha a mano cuando se pasa por encima.
- Por parte del espaciado, se ha optado por el uso de líneas o espacios blancos para separar y diferencias contenidos.
- La aplicación se adapta a distintos entornos, ya que es una interfaz adaptable que funciona en distintas resoluciones (interfaz móvil para el rol de servicio técnico) y navegadores.
- Se han utilizado fondos lisos para mejorar la legibilidad de los textos.
- Se ha intentado evitar el uso del *scroll* debido a que, al trabajar con tablas con paginación y opciones de búsqueda, no se ha considerado necesario su uso.

6.2 Directorios

En este apartado, se va a explicar cómo se ha organizado cada capa y qué contiene cada directorio.

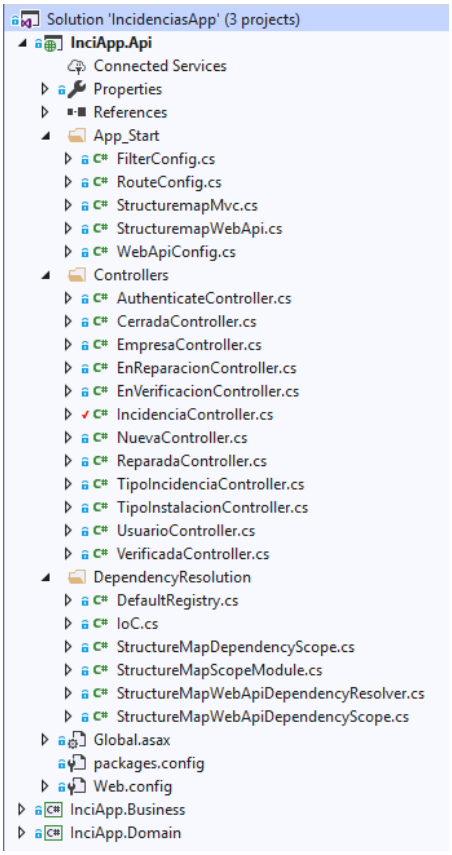

Ilustración 43: Directorio de InciApp.Api.

InciApp.Api es un proyecto de tipo ASP.NET, con el que se ha creado la Web API y es el que pertenece a la capa de servicios.

El primer directorio que se puede observar es el de App_Start. Éste contiene ficheros de configuración para que, en el arranque del proyecto, cargue con ciertas características, como hemos mencionado con anterioridad. El siguiente es Controllers, que como su nombre indica, contiene los controladores de la aplicación. El último es el DependencyResolution, que se creó por defecto al realizar la instalación de StructureMap.

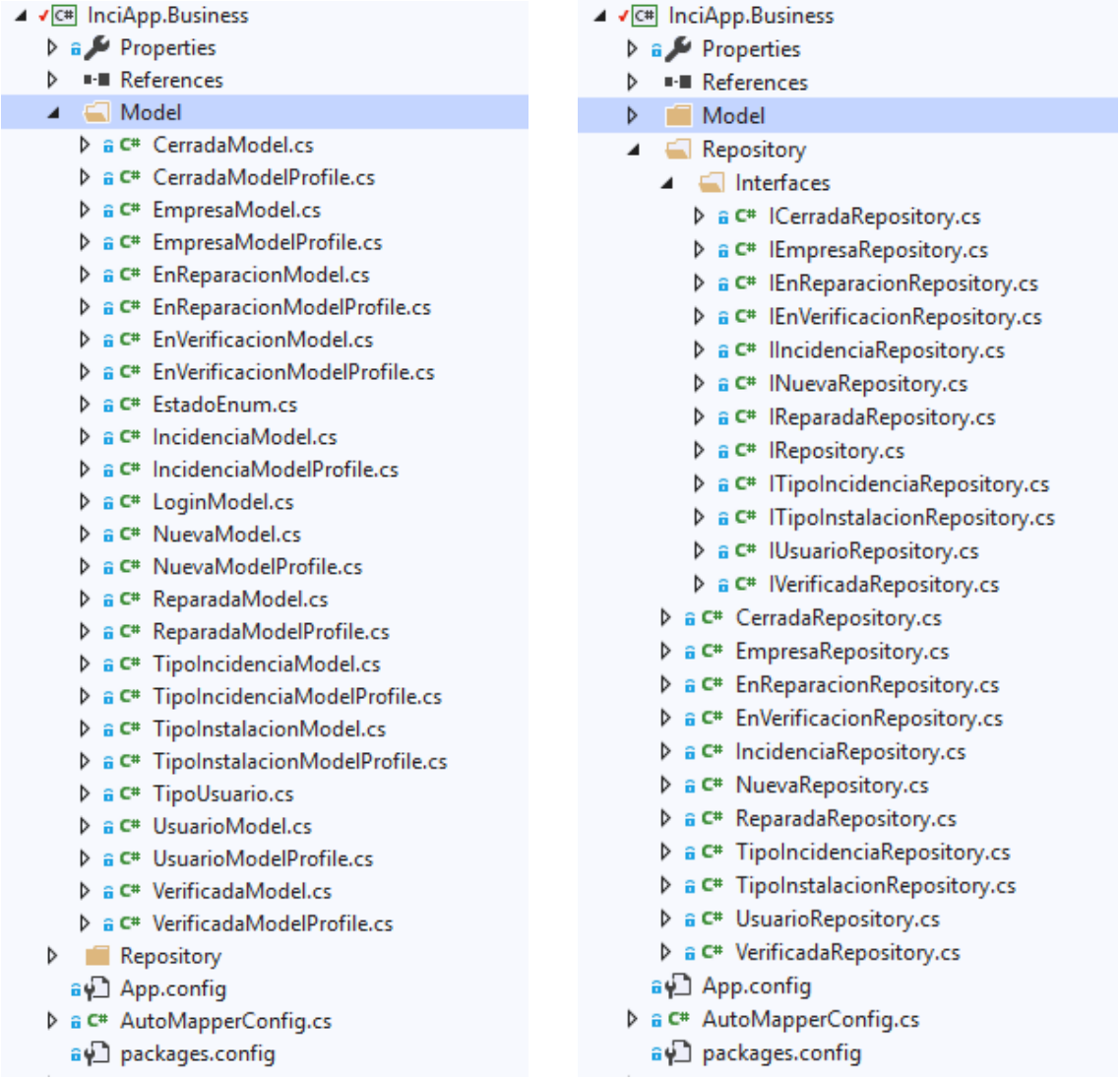

Ilustración 44: Directorio de InciApp.Business.

InciApp.Business es un proyecto de tipo biblioteca de clases, en el que se ha creado la parte lógica de la aplicación y es el que pertenece a la capa lógica de negocio.

El primer directorio que se puede observar es el de Model. Éste contiene todos los modelos y los perfiles de los modelos que utiliza la aplicación. También se han incluido las enumeraciones, como las que muestran los estados o los tipos de usuario. El siguiente directorio es Repository, que contiene todos los repositorios y un subdirectorio Interfaces con todas las interfaces de éstos. El último fichero que podemos ver es el de AutoMapperConfig, del que ya hemos comentado su funcionalidad anteriormente.

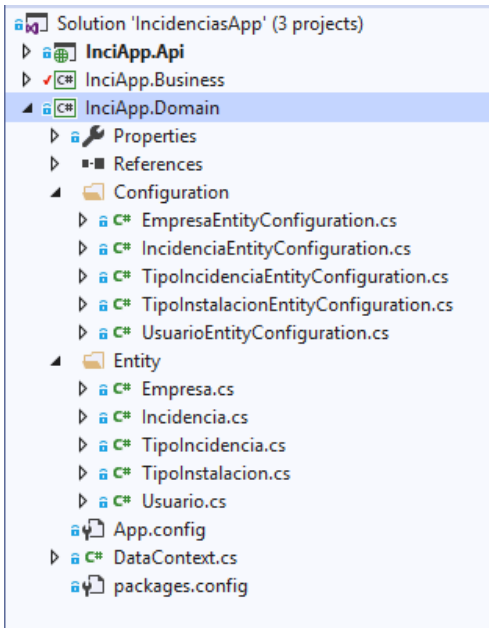

Ilustración 45: Directorio de InciApp.Domain.

InciApp.Domain es un proyecto de tipo biblioteca de clases y el cual pertenece a la capa de acceso a datos. En este se pueden observar dos directorios, el de Configuration con las configuraciones de cada entidad y el directorio de Entity que las contiene. Además, contiene el fichero DataContext que se encarga de cargar las configuraciones, conectar con la base de datos, etc.

Aquí finalizarían los directorios de la parte del proyecto del servidor en el que se crearon tres capas. Ahora se va a proceder a la disección de los de la capa de presentación.

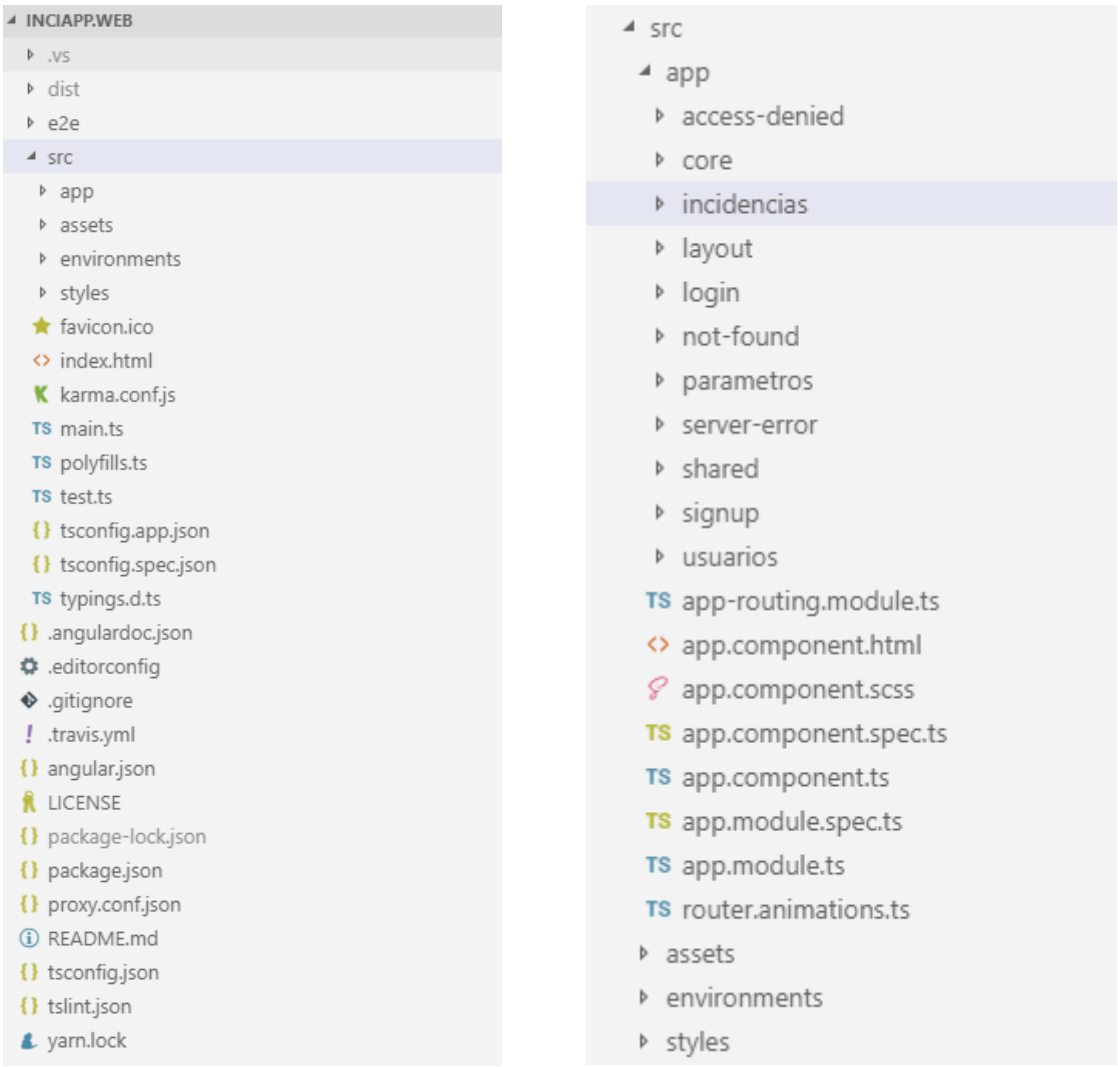

Ilustración 46: Directorios de InciApp.Web.

InciApp.Web es un proyecto de Angular, en el que se ha creado la capa de presentación y la que será usada por el cliente.

El directorio src contiene todo lo programable de la aplicación. Dentro de éste, se han creado o añadido los estilos (styles) y se han creado los modelos, servicios, componentes, etc. en app.

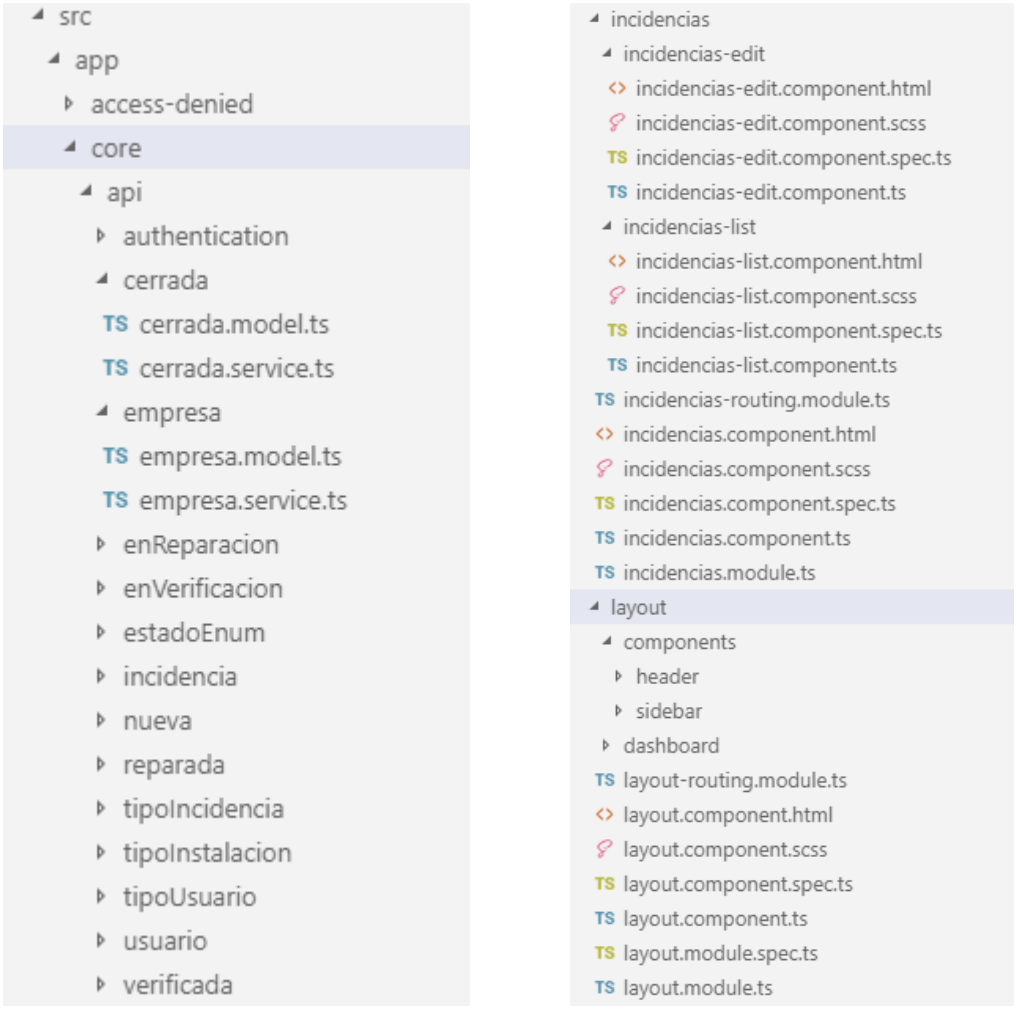

Ilustración 47: Directorios de servicios, modelos y componentes.

En estas ilustraciones se pueden especificar con más detalles los directorios. Dentro del directorio core, se ha creado otro llamado api para crear todos los ficheros relacionados con ésta (modelos y servicios). Los directorios de incidencias y layout son ejemplos de cómo se ha construido la estructura de las vistas o componentes, añadiendo los ficheros de ruta o módulos donde fuesen necesarios.

7. Manual de uso

En este punto se va a mostrar un sencillo manual de uso en el que se aplicarán dos casos de uso, el de creación de un usuario (Tabla 4) y el de la modificación de una incidencia (Tabla 15). Éste último se realizará con un usuario con rol de servicio técnico, para poder mostrar la interfaz móvil.

- Caso de uso 4: creación de un usuario.

El usuario conectado, en este caso con rol de administrador, se encontrará en la vista de usuarios y clicará el botón de crear un nuevo usuario para proceder a la creación de éste.

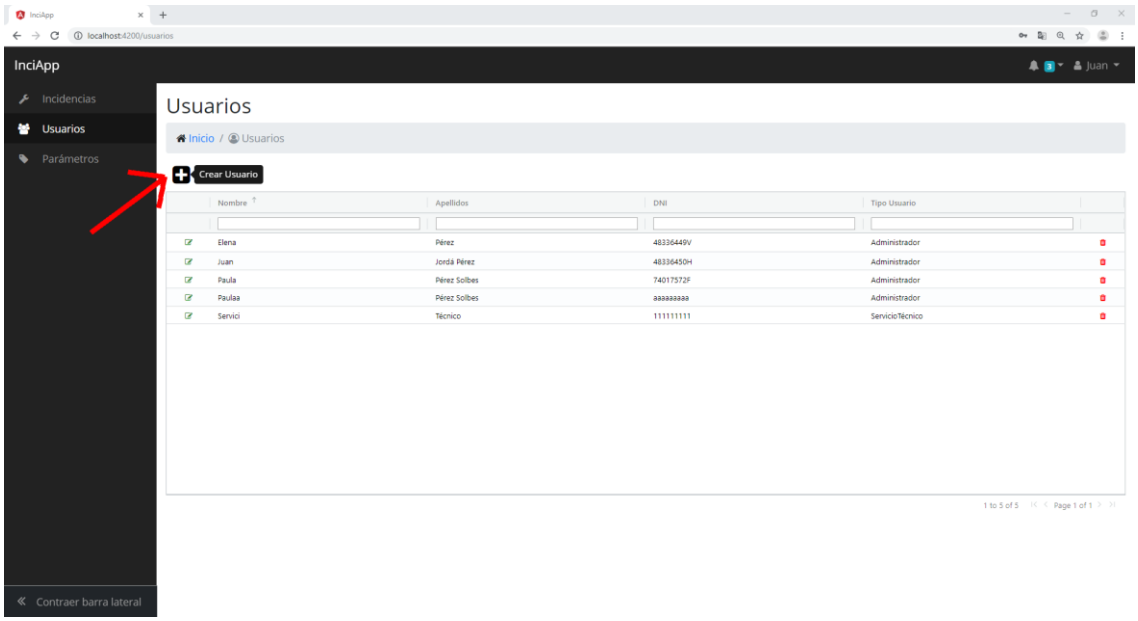

Ilustración 48: Vista de usuarios.

El usuario en cuestión completará correctamente todos los campos requeridos marcados con una barra lateral en color rojo y mostrará errores en tiempo real mientras que se están introduciendo.

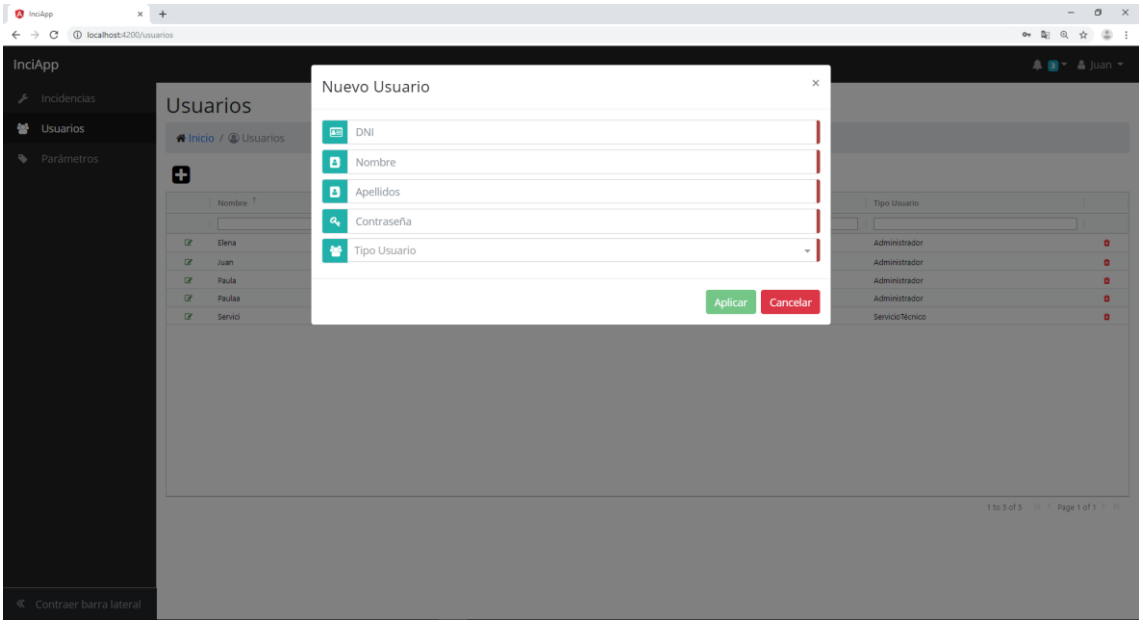

Ilustración 49: Modal vacío de creación de usuario.

En el momento en que sean correctos, el botón de aplicar se habilitará y se creará el usuario.

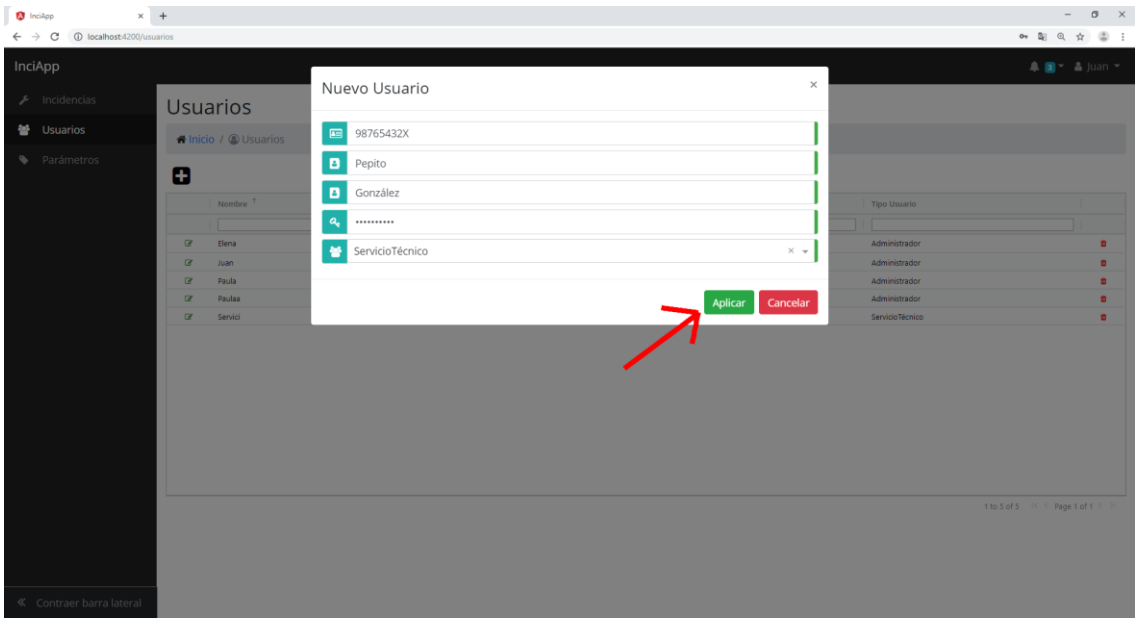

Ilustración 50: Modal de creación de datos correctos.

Una vez el usuario pulsa el botón de crear, el usuario recibe un aviso como que el usuario ha sido creado con éxito y éste se puede observar en la lista.

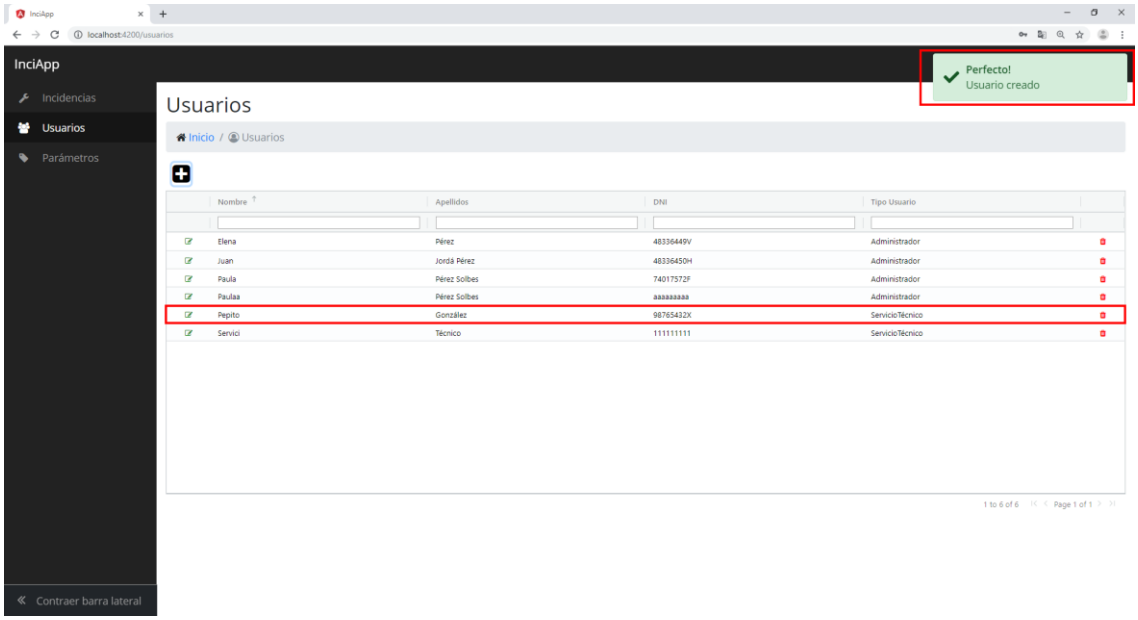

Ilustración 51: Vista de usuarios con aviso de creación.

- Caso de uso 15: modificación de una incidencia.

El usuario conectado, en este caso con rol de servicio técnico y con interfaz móvil, se encontrará en la vista de incidencias y clicará el botón de editar de la incidencia que desee modificar.

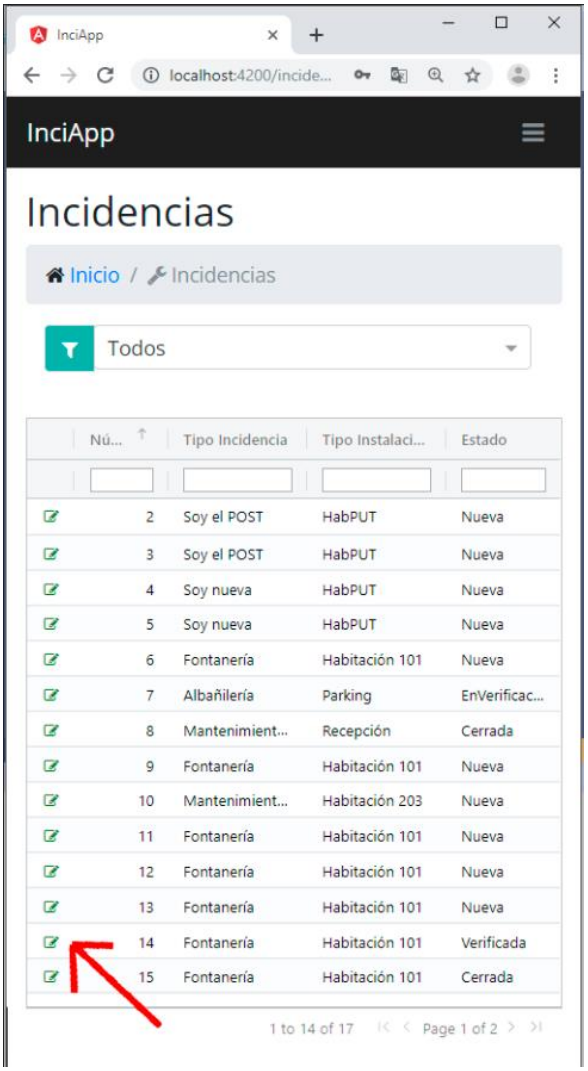

Ilustración 52: Vista de incidencias.

El usuario observará la vista donde se ve el número de incidencia, el estado actual, tipo de incidencia y de instalación, el historial de descripciones y un campo para añadir una nueva descripción. Si sólo se desea cambiar a un estado nuevo, el usuario clicará en el botón que muestra el estado siguiente de la incidencia.

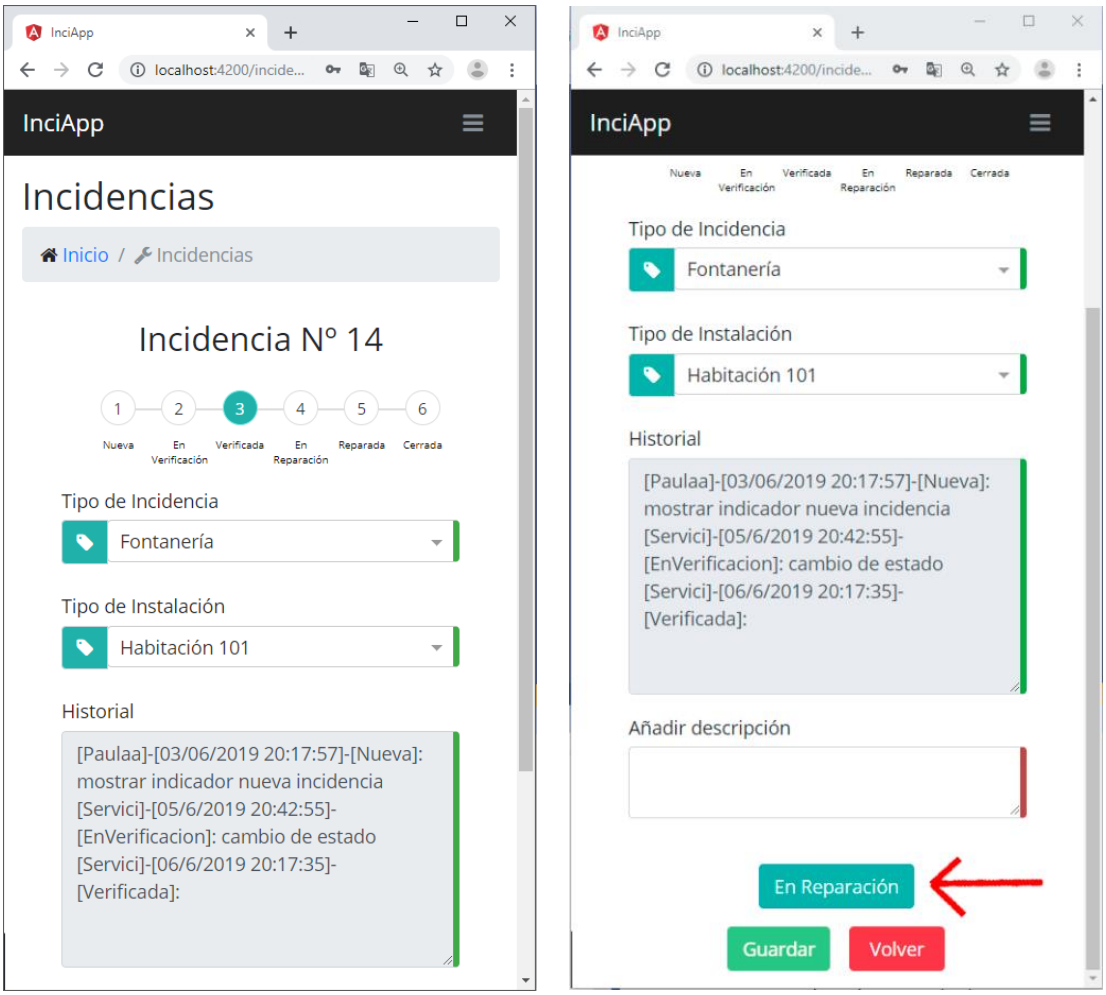

Ilustración 53: Vista de edición de usuario.

Una vez el usuario pulsa el botón para cambiar de estado, el usuario recibe un aviso como que la incidencia ha sido modificada con éxito y ésta se puede observar en la lista con los nuevos datos.

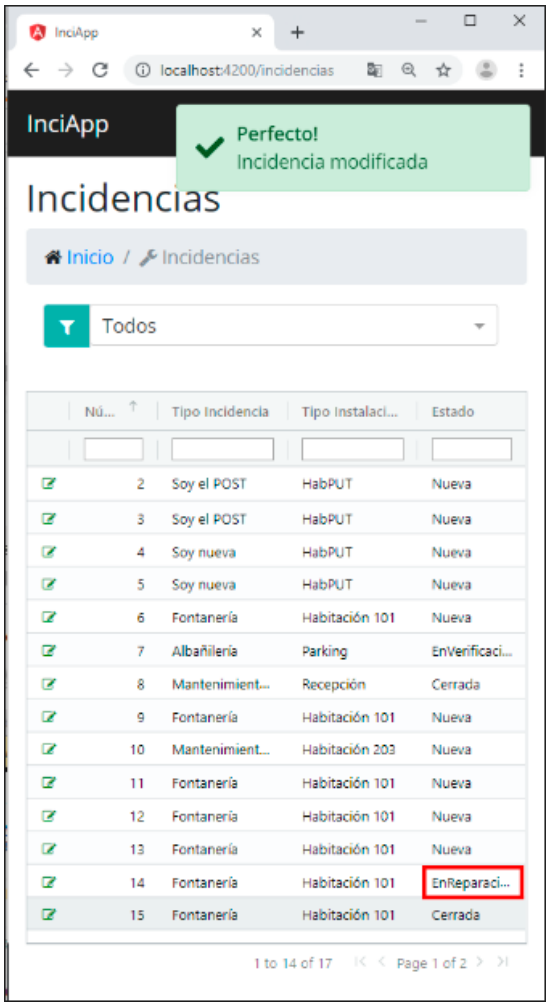

Ilustración 54: Vista incidencias con modificación.

8. Conclusiones

Dados los conocimientos, prácticamente nulos, que poseía en la creación de una aplicación web, fue lo que me obligó a tener que buscar y elegir tecnologías con las que formarme. Como debía aprender varias de ellas, me decanté por utilizar las que están a la orden del día y en continuo auge, Angular y Microsoft .NET. Teniendo la idea en mente y comenzado el proceso de lectura de documentación y formación en las tecnologías, surgió la casualidad de que se me contrató como empleado en prácticas en una empresa que utilizaba las mismas tecnologías, por lo que se agilizó el proceso de adquisición de conocimientos.

El proceso inicial de la parte de implementación fue bastante frustrante, debido a que los tutoriales y cursos eran muy simples y lo que se tenía planteado era bastante más complejo de desarrollar de lo que parecía desde un inicio. La complejidad de comprensión de ciertos aspectos a desarrollar, como la inyección de dependencias y la separación por capas, causo que la curva de aprendizaje fuese exponencial. Gracias a ello, cualquier pequeña modificación o nueva función que se ocurría en algún momento, se solucionaba o creaba con extrema rapidez.

Hubo momentos en los que el proyecto se estancó, como cuando siguiendo varios tutoriales de instalación para la compatibilidad de MySql con Visual Studio, surgían errores que no estaban contemplados en éstos, pero finalmente, como se indicó en un punto anterior, se encontró la solución a este problema, que resultó ser el orden de instalación de ciertos programas.

En relación con los conocimientos adquiridos durante el Grado, se podría decir que los principales se han adquirido fuera de éste, pero no quiere decir que no hayan sido de gran ayuda. Por ejemplo, no se han sido necesarios adquirir conocimientos en el lenguaje SQL o HTML en lo que refería al desarrollo de este proyecto, se conocía el lenguaje de C# y las arquitecturas gracias a la asignatura de Ingeniería del Software, etc. También la elección de mi especialización en el Grado, la de Tecnologías de la Información, creo que ha sido de gran ayuda para este proyecto, debido a que asignaturas como Desarrollo Centrado en el Usuario, Desarrollo Web o Integración de Aplicaciones, han aportado grandes conocimientos. Las dos primeras me ofrecieron una base de programación web a la vez que un gran entendimiento de la importancia de la usabilidad en las aplicaciones. Por parte de la última de éstas, me proporcionó el descubrimiento de las API REST, que para mí era un mundo desconocido y casualmente, me sirvió de gran ayuda para el desarrollo de la parte del servidor.

En conclusión, estoy muy satisfecho con el trabajo realizado y con los conocimientos adquiridos durante su desarrollo, ya que me han aportado una gran mejora a nivel profesional. Únicamente me gustaría encontrar algún compañero que me ayudase a continuar con el desarrollo del proyecto y poder llevarlo a su comercialización.

8.1 Metas alcanzadas

Se han alcanzado todos los objetivos de este proyecto. Se ha completado el funcionamiento planteado, que era el de realizar una aplicación web con la que gestionar las incidencias en el ámbito hotelero, permitiendo el acceso para distintos tipos de usuario y mostrando interfaces adaptadas para cada uno de ellos.

8.2 Metas no alcanzadas

Personalmente no creo que haya alguna meta no alcanzada, pero si hay puntos que no se han podido desarrollar debido a la escasez de tiempo y que me hubiera gustado hacerlo, como el caso del desarrollo propio de las barras laterales y superiores, aunque al final el resultado desarrollado a partir de la plantilla ha sido satisfactorio.

8.3 Trabajos futuros

En la aplicación todavía pueden añadirse muchísimas funcionalidades, muchas de ellas necesarias en caso de publicar la aplicación para su uso comercial:

- Añadir seguridad mediante el uso de *tokens*.
- Cifrado de las contraseñas en la base de datos.
- Migrar la base de datos a SQL Server.
- Personalizar los filtros y las notificaciones por defecto para cada rol.
- Mejorar la configuración de CORS.
- Añadir opciones para exportar las tablas.
- Gráficos y estadísticas para supervisar dónde hay más problemas.
- Posible control de stock y gastos de materiales.
- Creación de tests unitarios debido a que la arquitectura proporciona facilidad para ello.

Bibliografía

[1] AG-GRID. Documentación de Ag-Grid. <https://www.ag-grid.com/documentation-main/documentation.php> [Fecha de consulta: 28 de mayo del 2019].

[2] ANGULAR V7. Documentación de Angula versión 7. <https://v7.angular.io/docs> [Fecha de consulta: 5 de marzo del 2019].

[3] CHRIS PRATT. Using AutoMapper: Getting Started. <https://cpratt.co/using-automapper-getting-started/> [Fecha de consulta: 7 de mayo del 2019].

[4] DSIC - UPV. "El proceso del software" en Ingeniería del Software, 2017, Tema 2, ETSINF. [Fecha de consulta: 12 de junio del 2019].

[5] ENRIQUE ORIOL. Intro a Angular (parte I): Módulo, Componente, Template y Metadatos. <http://blog.enriqueoriol.com/2017/03/introduccion-angular-modulo-y-componente.html> [Fecha de consulta: 10 de marzo del 2019].

[6] ENTITY FRAMEWORK 6. Documentación de Entity Framework 6. <https://docs.microsoft.com/es-es/ef/ef6/> [Fecha de consulta: 26 de febrero del 2019].

[7] GERALS WEINBERG. Principios de la arquitectura. <https://docs.microsoft.com/es-es/dotnet/standard/modern-web-apps-azure-architecture/architectural-principles> [Fecha de consulta: 10 de junio del 2019].

[8] GONZALO RIVAS. La revolución digital: el poder de estar en las nubes. <https://elpais.com/elpais/2018/04/06/planeta_futuro/1523023364_337768.html> [Fecha de consulta: 7 de junio del 2019].

[9] KANBAN. ¿Por qué utilizar la metodología Kanban?. <https://kanbantool.com/es/metodologia-kanban> [Fecha de consulta: 23 de abril del 2019].

[10] MARINA MARININA. Java vs .NET: Factors to Consider. <https://codeburst.io/java-vs-net-factors-to-consider-cde1d22b06a7> [Fecha de consulta: 10 de febrero de 2019].

[11] MICROSOFT .NET. Documentación de .NET. <https://docs.microsoft.com/es-es/dotnet/> [Fecha de consulta: 25 de febrero del 2019].

[12] PEDRO VALDERAS Y MANOLI ALBERT. "Aspectos clave en el diseño para la usabilidad" en Desarrollo Centrado en el Usuario, 2018, Tema 4, ETSINF. [Fecha de consulta: 8 de junio del 2019].

[13] SOURCETREE. Documentacion de SourceTree. <https://confluence.atlassian.com/get-started-with-sourcetree> [Fecha de consulta: 15 de febrero del 2019].

[14] STRUCTUREMAP. Documentación de StructureMap. <https://structuremap.github.io/documentation/> [Fecha de consulta: 16 de mayo del 2019].

[15] UUUAPSAPP. App de gestión de incidencias. <https://www.uuupsapp.com/> [Fecha de consulta: 9 de junio del 2019].

[16] WUSHENG HU. N-Tier Architecture and Tips. <https://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips> [Fecha de consulta: 20 de abril del 2019].