



captainer

Deployers Service

SOFTWARE DE GESTIÓN DE IMPLANTACIONES

Proyecto integrado de carácter práctico

2019

Descripción breve

Aplicación Web para la gestión de implantaciones de la empresa y las incidencias que ocurren con las implantaciones llevadas a cabo

Juan José González Fernández

Ciclo Formativo Grado Superior Desarrollo de Aplicaciones Multiplataforma



captainer

Deployers Service

Proyecto Integrado de Carácter práctico.

Titulo:
Captainer: Software de gestión de implantaciones
Autor:
Juan José González Fernández
Centro:
I.E.S. La Marisma (Huelva)
Titulación:
Ciclo formativo grado superior Desarrollo Aplicaciones Multiplataforma
Departamento:
Informática
Curso académico:
2018 – 2019

Resumen

La empresa onubense afincada en Bellavista, Prisma Marketing SG, requiere de un software empresarial interno que se ocupe de la gestión de las implantaciones que lleva a cabo la compañía. Desean que todas las incidencias que surjan alrededor de las implantaciones existentes queden registradas en la aplicación, de modo que puedan ser consultadas en cualquier momento, con el objetivo de mejorar el soporte y atención al cliente, haciendo que éste sea más dinámico y personalizado. La aplicación realizará búsquedas de anteriores incidencias, al mismo tiempo que permitirá añadir nuevas incidencias, editar y borrar incidencias existentes. Se desea que la aplicación pueda administrar las implantaciones y muestre información al respecto, como los contactos de dicha implantación, información del proyecto al que pertenecen e información acerca de los equipos de los clientes donde se realizan las implantaciones.

La aplicación se ha desarrollado durante las horas activas de la empresa, con la idea de ser incorporada a las aplicaciones propietarias de la propia empresa, siendo desplegada en los servidores una vez sea completamente funcional.

Abstract

The Huelva company based in Bellavista, Prisma Marketing SG, requires an internal business software that deals with the management of the implementations carried out by the company. They want all the incidents that arise around the existing implementations to be registered in the application, so that they can be consulted at any time, with the aim of improving support and customer service, making it more dynamic and personalized. The application will perform searches of previous incidents, at the same time that it will allow to add new incidents, edit and delete existing incidents. It is desired that the application can manage the implementations and show information about it, such as the contacts of said implementation, information of the project to which they belong and information about the equipment of the clients where the implementations are made.

The application was developed during the active hours of the company, with the idea of being incorporated into the proprietary applications of the company itself, being deployed in the servers once it is fully functional.

ÍNDICE:

Resumen	3
Abstract.....	3
Tabla de Ilustraciones:	7
1. Introducción.....	9
1.1. Introducción a la memoria	9
1.2. Descripción	9
1.3. Objetivos generales	11
1.4. Beneficios.....	11
1.5. Motivaciones Personales:.....	11
2. Estudio de viabilidad.....	12
2.1. Introducción:.....	12
2.1.1. Tipología y palabras claves:	12
2.1.2. Descripción:	15
2.1.3. Objetivos del proyecto:.....	16
2.1.4. Clasificación de los objetivos:.....	16
2.1.5. Partes interesadas	17
2.1.6. Referencias	18
2.2. Estudio de la situación actual	18
2.2.1. Contexto:	18
2.2.2. Normativa y legislación	20
2.3 Requisitos del sistema	26
2.3.1 Requisitos	26
2.3.2 Restricciones del sistema.....	27
2.4 Planificación del proyecto.....	27
2.4.1 Recursos del proyecto	27
2.4.2 Tareas del proyecto.....	28
2.4.3 Planificación temporal	29
2.5 Evaluación de riesgos.....	29
2.5.1 Lista de riesgos.....	29
2.5.2 Catalogación de riesgos	30
2.5.3. Plan de contingencia.....	31
2.6. Presupuesto	32

2.6.1 Estimación de coste material.....	32
2.6.2 Estimación de coste personal	32
2.6.3 Resumen y análisis coste beneficio	33
2.7. Conclusiones.....	34
2.7.1 Beneficios.....	34
2.7.2. Inconvenientes.....	34
3. Análisis	35
3.1.Introducción.....	35
3.2.Requisitos funcionales de usuarios	36
3.3. Requisitos no funcionales	38
3.4Diagramas de casos de uso / Casos de uso	39
3.5. Diagrama de actividad	40
3.6. Conclusión del análisis.....	42
4. Diseño	43
4.1 Introducción.....	43
4.1.1. Selección del entorno de desarrollo.....	43
4.2 Configuración de la plataforma	52
4.3 Arquitectura de la aplicación	57
4.4 Estructura de la base de datos	59
4.5. Estructura del ORM	62
4.5.1. Middleware y Bearer Token:.....	65
4.6 Estructura de la aplicación.....	69
4.6.1. Los componentes.....	71
4.7. Diseño de la interfaz	73
4.7.1. Diseño Responsive	78
4.8. Lógica de la aplicación	81
5. Implementación	84
5.1 Introducción.....	84
5.2 Codificación de las diferentes capas.....	85
5.2.1. Fase Inicial: Configuración y despliegue del Back-End (base de datos y ORM en el servidor):.....	85
5.2.2. Primera fase de la aplicación: Creación del proyecto Angular y sus componentes Básicos	93
5.2.3. Segunda fase de la aplicación. Creación de botones, componentes y servicios para añadir, editar y eliminar incidencias e implantaciones	100
5.2.4. Tercera fase de la aplicación: Control de sesión y vista detalles	103
5.2.5. Cuarta fase de la aplicación y fase actual: Implementación de animaciones de carga, filtrado de objetos, pantalla de error y otros detalles	107

6. Pruebas	112
6.1 Introducción	112
6.2 Pruebas.....	113
6.2.1. Pruebas de Integración.....	113
6.2.2. Pruebas de Sistema	114
6.2.3. Pruebas de Usuario.....	117
6.2.4. Pruebas de aceptación	117
6.3 Resultados Obtenidos.	118
7. Conclusiones.....	119
7.1 Conclusiones finales.....	119
7.2 Desviaciones temporales	119
7.3 Posibles ampliaciones y modificaciones	120
7.4 Valoración personal.....	121
7.5. Agradecimientos	122
8. Bibliografía	123

Anexo:

- Manual de Usuario
- Manual de Administrador

Tabla de Ilustraciones:

Ilustración 1: Diseño genérico de una ventana principal	9
Ilustración 2: Ejemplo de diseño usable y atractivo	17
Ilustración 3: Calendario de tareas para el proyecto	27
Ilustración 4: Pautas del diseño Responsive	36
Ilustración 5: Diagrama de casos de uso	37
Ilustración 6: Diagrama de actividad	39
Ilustración 7: Ciclo de vida de una SPA	44
Ilustración 8: Uso de la herramienta Postman	47
Ilustración 9: uso de Git	47
Ilustración 10: Petición en el navegador	51
Ilustración 11: Peticiones en el navegador formato JSON	52
Ilustración 12: Jerarquía de capas	53
Ilustración 13: Jerarquía extendida de la aplicación	54
Ilustración 14: Esquema de la base de datos	55
Ilustración 15: Esquema de la fase actual de la aplicación	57
Ilustración 16: Genealogía de Lumen	58
Ilustración 17: Petición GET en Postman de las incidencias	61
Ilustración 18: Esquema Back-End y Middleware	65
Ilustración 19: Esquema Front-End Angular	66
Ilustración 20: Esquema de un componente Angular	67
Ilustración 21: Pantalla bienvenida de nuestra aplicación	69
Ilustración 22: Menú de inicio de sesión	70
Ilustración 23: Pantalla de administración de incidencias	71
Ilustración 24: Vista de detalles	71
Ilustración 25: Modal de edición	72
Ilustración 26: Modal de borrado	72
Ilustración 27: Formulario de inserción de nuevas incidencias	73
Ilustración 28: Diálogo de confirmación de cerrar sesión	73
Ilustración 29: Pantalla bienvenida primera versión	74
Ilustración 30: Vista móvil	75
Ilustración 31: Vista Portátil 768p	75
Ilustración 32: Móvil Responsive	75
Ilustración 33: Portátil Responsive	75
Ilustración 34: Ventana de administración desde móvil	76
Ilustración 35: Formulario de incidencias desde móvil	76
Ilustración 36: MySQL	80
Ilustración 37: Ficheros de migración de Lumen	83
Ilustración 38: Tablas de la BBDD	84
Ilustración 39: Filas de la tabla 'user'	85
Ilustración 40: Pagina de error de conexión	104
Ilustración 41: esquema de calidad del software	111
Ilustración 42: Pruebas de rendimiento en Chrome	115
Ilustración 43: Consumo de recursos en Windows 10	116

1. Introducción

1.1. Introducción a la memoria

Tenemos como finalidad el desarrollo e implementación de una aplicación que gestione el soporte de la empresa a sus clientes. El objetivo es que la aplicación muestre el número de incidencias surgidas de las diferentes implementaciones de los proyectos llevados a cabo por la empresa. Las incidencias serán mostradas en la interfaz de usuario, siendo posible su ordenación y filtrado y, cada una de ellas, permitirá un CRUD al usuario de la aplicación; es decir, permitirá editar dichas incidencias y eliminarlas, así como la propia acción de listarlas como ya hemos narrado, y añadir nuevas incidencias a la aplicación.

1.2. Descripción

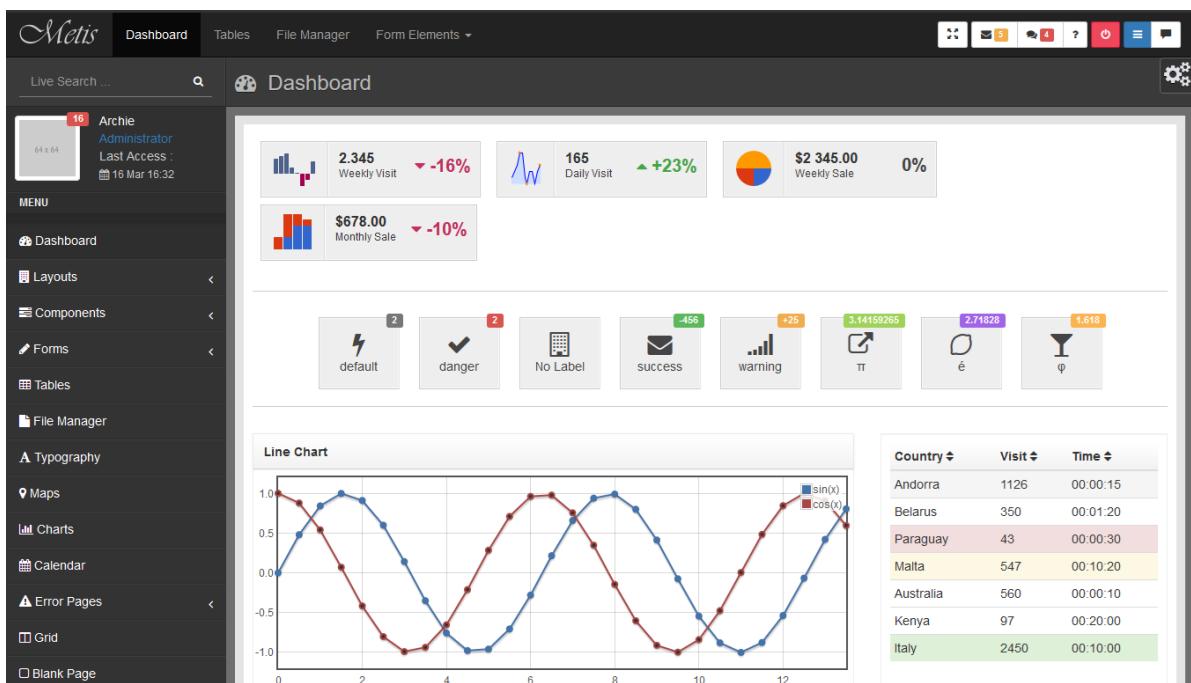


Ilustración 1: Diseño genérico de una ventana principal

Para llevar a cabo esta tarea, y atendiendo a nuestras necesidades y propiedades que debe tener la aplicación, hemos optado por desarrollar nuestra aplicación en Angular 7, puesto que se adapta a los cánones de programación de la empresa y el paradigma que sigue a la hora de desarrollar e implementar aplicaciones. Además, resulta más ligera y escalable, y nos permite crear una interfaz muy completa utilizando las herramientas bien conocidas usadas en el entorno Web y que sigue los dictámenes de la W3C. Para interactuar con los datos, desarrollaremos la aplicación en Angular y seguiremos un modelo de diseño maestro-detalle, y volcaremos la información básica en una pantalla principal, conocida como 'Dashboard', pudiendo movernos por las diferentes acciones de la aplicación por medio de elementos en la interfaz, como menús laterales o la barra superior, creando un acceso sencillo a todas las acciones y propiedades del proyecto.

Para almacenar los datos de uso de la aplicación, como clientes, proyectos, implementaciones, información de los equipos, incidencias, etc. Usaremos un modelo de bases de datos relacional, concretamente MySQL, debido a que la empresa trabaja con dicha plataforma y gestiona los datos desde un servidor que trabaja con ella. Para comunicar la aplicación con la base de datos, crearemos una capa de abstracción ORM para poder manejar y estructurar los datos de una manera más sofisticada y práctica. Emplearemos Eloquent, una herramienta del framework Lumen (Microframework basado en Laravel) que nos permitirá la creación, adición, modificación, etc. De tablas y consultas SQL. Desde un controlador generado en el Back-End, podremos gestionar el CRUD de usuario en las diferentes vistas de la aplicación.

Por convenio de la empresa, la aplicación será desplegada como aplicación web inicialmente, y no como aplicación de escritorio. Esto tiene como ventaja principal el no depender de ningún medio de instalación en los equipos y facilita el control de versiones, puesto que no es necesaria una reinstalación para poner en vigor en la empresa futuras versiones de la aplicación. Además, la conexión de la empresa permite trabajar sin ningún problema con aplicaciones orientadas a la web y gracias a la potencia que ofrece el framework Angular, es posible generar aplicaciones muy completas utilizando los patrones de diseño tradicionales, combinados con la versatilidad de TypeScript.

La autenticación se producirá por medio de JWT (JSON Web Token). Esto nos permitirá generar un token único por sesión de usuario que podrá ser almacenado de manera temporal en el navegador que usemos para trabajar con la aplicación. Se trata de un estándar de seguridad cuyo token es firmado por una firma digital, con clave simétrica o asimétrica. Algunas de las ventajas de los JWT son su reducido tamaño, ya que al ser JSON tan solo añaden unos pocos bytes a nuestras peticiones contra el servidor, y otra de las mayores ventajas es que el payload del JWT contiene toda la información que necesitemos sobre el usuario, de forma que se evita la necesidad de repetir consultas a base de datos para obtener estos datos.

Por cuestiones de diseño y atendiendo a las funcionalidades del framework encargado de desarrollar el Front-End, la aplicación constará de una SPA que gestionará el soporte de la empresa. SPA son las siglas de Single Page Application. Es un tipo de aplicación web donde todas las pantallas las muestra en la misma página, sin recargar el navegador. Las páginas de gestión, o administración de cualquier tipo de servicio, paneles de control y cosas así son muy adecuadas para las SPA. El resultado es una aplicación web se comporta muy parecido a una aplicación de escritorio; por lo que al final lo que tendremos será una aplicación que recarga muy rápido y es, al mismo tiempo, una alternativa muy potente a la clásica aplicación de escritorio. A su vez, nos permitirá la implantación de formularios reactivos, donde las acciones sobre los datos y los elementos html (el DOM) ocurren principalmente a nivel del componente (código typescript), favoreciendo la velocidad de desarrollo, y amplía la posibilidad de test con pruebas unitarias en la aplicación, ya que hablamos de formularios diseñados a nivel de programación, no a nivel de lenguaje de marcado.

En definitiva, debemos planificarnos para desarrollar las siguientes pautas:

1. Front-End en Angular con SPA, formularios reactivos, diseño Dashboard, tratado de la información de forma responsive, binding y routing de componentes visuales.
2. Back-End en Lumen para la organización e interacción de la información y uso de JWT para la autenticación y seguridad.
3. Modelo de datos relacional en MySQL.

1.3. Objetivos generales

Se pretende que la aplicación realice las funciones esperadas de un CRUD de usuario (Crear, Listar, Modificar y Eliminar), y que la información dada sea accesible de forma sencilla, rápida e intuitiva, a través de menús de navegación. La aplicación debe encontrarse alojada en un servidor y ser Responsive. Manejará formularios reactivos y girará en torno al paradigma de diseño SPA. El CRUD dependerá de los roles de los usuarios, siendo los roles controlados desde el inicio de sesión, el cual integrará un mecanismo JWT para mejorar la autenticación y facilitar el inicio de sesión a los usuarios de la aplicación.

1.4. Beneficios

Los beneficios que se esperan que aporte la aplicación serán los siguientes:

- Control de todas las incidencias de las implantaciones de la empresa en una sola aplicación
- Posibilidad de consultar los motivos de la incidencia, fecha de emisión y posibles soluciones al respecto
- Posibilidad de consultar el hardware e infraestructura de conexiones de los equipos y sistemas informáticos de las diferentes implantaciones de las empresas de los clientes
- Gestión más dinámica y elaborada de la información: contar con un repositorio de documentos facilitados por los clientes que ayuden a detectar y comprender el motivo de la incidencia (capturas de pantalla, documentos con extensión .pdf, etc.)
- Favorece la comunicación acerca de las incidencias en la empresa de forma inmediata y sin pérdida de información. Todas las incidencias quedan registradas en la aplicación, siendo posible consultarlas al momento de su creación o modificación de forma rápida.

1.5. Motivaciones Personales:

Uno de los objetivos de la aplicación es alcanzar un nivel de conocimientos acerca de los Frameworks a utilizar. Aprender a desenvolvernos en los entornos web con dichas herramientas supone un incentivo al proyecto, de cara al aprendizaje y las metas propuestas a largo plazo: comprender como funciona una aplicación web, consumir un servicio RESTFul, atender a aplicaciones y páginas web asíncronas, etc.

2. Estudio de viabilidad

2.1. Introducción:

2.1.1. Tipología y palabras claves:

- **AJAX:** Técnica para el desarrollo de páginas (sitios) web que implementan aplicaciones interactivas.
- **API:** Procesos, las funciones y los métodos que brinda una determinada biblioteca de programación a modo de capa de abstracción para que sea empleada por otro programa informático.
- **Angular:** Framework para aplicaciones web desarrollado en Typescript, de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página.
- **Artisan:** interfaz de línea de comandos de Lumen. Ofrece un conjunto de comandos que nos pueden ayudar a realizar diferentes tareas durante el desarrollo.
- **Asíncrono/a:** Capacidad de “diferir” la ejecución de una función a la espera de que se complete una operación, normalmente de I/O (red, disco duro, etc.) y así evitar bloquear la ejecución hasta que se haya completado la tarea en cuestión.
- **Back-End:** Es la parte del desarrollo web que se encarga de que toda la lógica de una página web funcione como, por ejemplo, la comunicación con el servidor.
- **Base de Datos relacional:** recopilación de elementos de datos con relaciones predefinidas entre ellos. Estos elementos se organizan como un conjunto de tablas con columnas y filas.
- **Bootstrap:** biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web. Contiene elementos de diseño basado en HTML y CSS, así como extensiones de JavaScript adicionales.
- **Borrado lógico:** Metodología que permite recuperar datos eliminado, evitando que estos se borren de forma definitiva de la base de datos.
- **Bug:** Es un error o un defecto en el software o hardware que hace que un programa funcione incorrectamente.
- **CRUD:** Acrónimo de "Crear, Leer, Actualizar y Borrar", que se usa para referirse a las funciones básicas en bases de datos o la capa de persistencia en un software.
- **CSS:** lenguaje que describe la presentación de los documentos estructurados en hojas de estilo para diferentes métodos de interpretación.
- **Clase (Programación):** Plantilla para la creación de objetos de datos según un modelo predefinido. Las clases se utilizan para representar entidades o conceptos, como los sustantivos en el lenguaje.

- **Clave foránea:** columna o grupo de columnas de una tabla que contiene valores que coinciden con la clave primaria de otra tabla.
- **Componente (Angular):** Elemento de Angular que controla un trozo de pantalla o de vista. Todo lo que se puede ver en pantalla está controlado y gestionado por este tipo de elementos
- **Composer:** Herramienta para la gestión de dependencias en PHP que permite declarar y administrar las bibliotecas de las que depende el proyecto.
- **Constructor (programación):** Subrutina cuya misión es inicializar un objeto de una clase. En el constructor se asignan los valores iniciales del nuevo objeto.
- **Cosechador (Lumen):** También conocido como Seed o Seeder. Fichero que se encarga de introducir de forma autónoma, registros en una tabla de la base de datos.
- **Decorador (Angular):** es un modo de añadir metadatos a declaraciones de clase para ser usados por inyecciones de dependencia o compilaciones de directivas.
- **DELETE:** Método HTTP encargado de eliminar un recurso.
- **Directiva (Angular):** elementos en el HTML que permiten añadir, manipular o eliminar elementos del DOM
- **Framework:** Estructura en capas que indica qué tipo de programas pueden o deben ser construidos y cómo se interrelacionan.
- **Front-End:** todas aquellas tecnologías que corren del lado del cliente, es decir, todas aquellas tecnologías que corren del lado del navegador web
- **GET:** Método HTTP encargado de *obtener información* del servidor
- **HTML:** Lenguaje de marcado que se utiliza para el desarrollo de páginas y aplicaciones web.
- **HTTP:** protocolo de transferencia donde se utiliza un sistema mediante el cual se permite la transferencia de información entre diferentes servicios y los clientes que utilizan páginas web.
- **IDE:** Es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.
- **Implantación:** Puesta en marcha de una aplicación en el entorno de trabajo del cliente.
- **Incidencia:** Aviso de un contratiempo surgido a raíz de un fallo personal o informático, y que no permite el correcto funcionamiento del programa en el entorno corporativo
- **Instancia (programación):** todo objeto que derive de algún otro. De esta forma, todos los objetos son instancias de algún otro.
- **JWT:** Estándar abierto (RFC-7519) basado en JSON para crear un Token que sirva para enviar datos entre aplicaciones o servicios y garantizar que sean válidos y seguros
- **Javascript:** lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

- **Laravel:** es un framework código abierto para desarrollar aplicaciones y servicios web con PHP 5.
- **Login:** nombre dado al momento de autentificación al ingresar a un servicio o sistema mediante una sesión, usualmente mediante un nombre de usuario y contraseña.
- **Lumen:** versión más liviana de Laravel y orientado más a la creación de APIs y **microservicios**
- **MVC:** Estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.
- **Microservicio:** enfoque para desarrollar una aplicación software como una serie de pequeños servicios, cada uno ejecutándose de forma autónoma y comunicándose entre sí, por ejemplo, a través de peticiones HTTP a sus API.
- **Middleware:** software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, o paquetes de programas, redes, hardware o sistemas operativos.
- **Migración (Lumen):** Fichero que permite crear tablas de forma autónoma en la base de datos, así como definir las propiedades de ella y sus columnas, y las relaciones entre otras tablas.
- **Modulo (Angular):** Organiza las partes de nuestra aplicación. Ordenando la misma en bloques. Permitiendo extender nuestra aplicación con funcionalidades de librerías externas.
- **MySQL:** sistema de gestión de base de datos relacional (RDBMS) de código abierto, basado en lenguaje de consulta estructurado (SQL).
- **ORM:** Permite convertir los datos de los objetos en un formato correcto para poder guardar la información en una base de datos (**mapeo**).
- **Objeto (programación):** Es la entidad en torno a la cual gira la POO. Un objeto es un ejemplar concreto de una clase.
- **Payload:** Sección de un Token. Contiene los datos que identifican al usuario, como puede ser su ID, nombre de usuario, etc.
- **PHP:** Es un lenguaje interpretado de código abierto muy popular especialmente adecuado para el desarrollo web.
- **PHPStorm:** IDE de programación desarrollado por JetBrains.
- **Pipe:** Cadena de procesos conectados de forma tal que la salida de cada elemento de la cadena es la entrada del próximo.
- **POST:** Método HTTP que permite *enviar información* desde el cliente
- **PUT:** Método HTTP que coloca un recurso en la dirección especificada en la URL.
- **Postman:** extensión gratuita para el navegador Google Chrome que permite probar servicios web fácilmente.
- **Powershell:** Es una consola de sistema bastante más avanzado y completo que MS-DOS o CMD

- **REST:** es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON.
- **RESTFul:** Hace referencia a un servicio web que implementa la arquitectura REST.
- **Responsive:** técnica de diseño web que busca la correcta visualización de una misma página en distintos dispositivos.
- **Rol:** Función que una persona desempeña en un lugar o en una situación.
- **Router (Angular):** Concepto. contiene toda la lógica necesaria para enrutar en el navegador.
- **SPA:** Sitio web que cabe en una sola página con el propósito de dar una experiencia más fluida a los usuarios como una aplicación de escritorio.
- **SQL Workbench:** herramienta visual de diseño de bases de datos que integra desarrollo de software, administración de bases de datos, diseño de bases de datos, gestión y mantenimiento para el sistema de base de datos MySQL.
- **Servicio (Angular):** mecanismo para compartir funcionalidad entre componentes
- **Token:** cadena de caracteres con un significado coherente.
- **TypeScript:** Lenguaje de programación libre y de código abierto desarrollado y mantenido por Microsoft. Es un superconjunto de JavaScript, que esencialmente añade tipado estático y objetos basados en clases.
- **VSCode:** Visual Studio Code es un editor de código fuente. Es compatible con varios lenguajes de programación

2.1.2. Descripción:

1. Alcance del proyecto: El proyecto tiene como finalidad la creación de un CRUD de usuarios (listar, crear, modificar y eliminar) de incidencias en las implantaciones de los clientes. Dicha gestión se encontrará limitada por los roles de los usuarios, siendo el administrador quien gestione de forma total el tratamiento de la información desde la interfaz. Se controlará el acceso e inicio de sesión mediante JWT y la aplicación contará con formularios reactivos que permitan el filtrado y ordenación de incidencias por los distintos valores.

2. Análisis de situación: Fortalezas y debilidades: Contamos con la documentación necesaria para superar la curva de aprendizaje y reunir los conocimientos necesarios para llevar a cabo el proyecto. El desarrollo es asequible y los componentes desarrollados para según qué finalidad son escalables y se pueden adaptar a los siguientes desarrollos, haciendo que desarrollar un nuevo componente no requiera comenzar desde cero. Como punto en contra, contamos con un tiempo limitado para familiarizarnos con el entorno de trabajo y los lenguajes de programación establecidos, por lo que no debemos descuidar la organización del proyecto.

3. Definición de requisitos: Para la realización del proyecto, debemos dominar los conocimientos de las herramientas de programación que vamos a utilizar, como son Angular y Lumen, así como la gestión de bases de datos en MySQL. Además, debemos contar con un servicio de alojamiento que permita el despliegue de la aplicación para su posterior utilización. En nuestro caso, la empresa cuenta con servidores dedicados que ya incluyen el servicio de Angular y se encuentra configurado para el despliegue de aplicaciones web.

4. Determinación del enfoque: La aplicación debe ser capaz de listar las incidencias producidas en las diferentes implantaciones de los clientes de la empresa. Además, debe ser capaz de registrar, editar registros y eliminarlos en caso de ser necesario, aquellos registros que ya no sean necesarios. La aplicación debe contar con una interfaz amigable y que permita acceder de forma sencilla a los datos y filtrarlos u ordenarlos por sus atributos. Debemos, a su vez, diseñar un inicio de sesión que nos permita acceder a la aplicación con nuestras credenciales y que facilite un recordatorio de inicio de sesión de forma temporal, para que no sea necesario ingresar cada vez que abrimos el navegador.

5. Evaluación de la viabilidad del proyecto: El proyecto es viable, ya que contamos con los medios físicos para llevarlos a cabo. Tanto Angular como Lumen son frameworks de código abierto cuyo uso no suponen un desembolso económico, así como el uso de MySQL, que tampoco requiere una licencia de pago para poder ser usado. El servidor y el equipo de trabajo son proporcionados por la empresa, así como el mantenimiento de los mismo. El coste humano se reduce a un solo programador responsable de su desarrollo. La documentación es suficiente para abarcar los conocimientos necesarios para cumplir las expectativas del proyecto.

2.1.3. Objetivos del proyecto:

El proyecto tiene como objetivo todo lo anteriormente mencionado en el apartado de la descripción: 'Alcance del proyecto' en el tiempo establecido, que en nuestro caso es el día 3 de junio, que es la fecha límite impuesta para finalizar el desarrollo y preparar la documentación necesaria, así como el despliegue de la aplicación y las pruebas necesarias para comprobar su correcto funcionamiento. Será necesario, en el tiempo restante hasta la presentación del proyecto, subsanar bugs o errores de programación y preparar la exposición oral, todo ello documentado debidamente hasta la fecha de entrega y exposición.

2.1.4. Clasificación de los objetivos:

Podemos clasificar los objetivos de nuestro proyecto en dos grandes grupos:

1. Esenciales: La aplicación debe realizar un CRUD de incidencias en la que se listan las incidencias y muestren información de las diferentes implantaciones, estado de la incidencia, nombre del proyecto de la incidencia, etc. Para acceder a la aplicación, debemos desarrollar un Login o Inicio de sesión que cuente con una funcionalidad JWT que permita registrar la sesión del usuario y delimitar el acceso a distintas rutas de la aplicación mediante dicho Token. Dicho front-End

debe ser Responsive (adaptado a distintas resoluciones y periféricos u formas de interactuar con el sistema) y desarrollarse en el Framework Angular 7. El Back-End debe estar desarrollado con el Microframework Lumen 5.5 (ORM) y debe utilizarse MySQL como servicio de alojamiento de datos. Debemos estructurar la base de datos de la aplicación tal y como está contemplado en el análisis y diseño de la aplicación (Más adelante). Todas las interacciones con la base de datos se deben hacer mediante los controladores y modelos del framework, y no desde la lógica de la interfaz de usuario, entendiéndose por interacciones con la base de datos con las acciones directas sobre ella (Consultas SQL. La interacción del front-end será mediante peticiones Get/Post/Put desde la directiva HttpClient, por lo que interactuaremos con el Servicio Restful, no directamente con la base de datos). La aplicación debe permitir desloguearse y entrar con otro usuario registrado en la aplicación, haciendo posible que el Token generado se elimine y por tanto la sesión del anterior usuario expire.

2. A largo plazo: Estas funcionalidades serán extensiones de las funcionalidades esenciales y pueden no encontrarse estrictamente desarrolladas en su totalidad en la versión final del proyecto. Se trata de mejoras que la aplicación sufrirá a medida que su uso se vaya consolidando y el desarrollo permita la adición de mejores características o soluciones a errores o limitaciones de la propia aplicación. Entre estas, están:
 - Distinción de roles de usuarios: haciendo que el CRUD dependa del rol de usuario que ingrese en la aplicación
 - Filtrado de datos mediante una barra de búsqueda que permita encontrar una implantación data.
 - Organización de las incidencias en distintas pestañas de la aplicación, en función de las implantaciones seleccionadas.
 - Posibilidad de que sea multiplataforma de forma híbrida, mediante Ionic o Cordova.

2.1.5. Partes interesadas

El proyecto presenta una motivación particular, ya que se trata de un proyecto propuesto por la empresa FCT y llevada a cabo allí, con el objetivo de ser utilizado en el entorno de trabajo como herramienta corporativa. Esto hace que el desarrollo, además de estar sujeto a las exigencias esperables por parte del centro educativo, sea también motivo de interés para la empresa en pos de contar con una nueva herramienta de gestión empresarial. Por la parte del desarrollador, es interesante, puesto que permite aprender nuevos lenguajes y técnicas de programación que hasta ahora no habían sido planteados en su carrera. Es, por tanto, un proyecto que cuenta con la aprobación de diversas partes para su desarrollo final.

2.1.6. Referencias

A la hora de diseñar las distintas partes de la aplicación, es necesario tener un punto de inicio y conocer el mercado y las necesidades reales de quienes van a requerir el uso de ésta. A modo orientativo, se nos ha facilitado la estructura del Back-End de otra aplicación de la empresa para poder entender que pautas hemos de seguir a la hora de desarrollarlo. Esto ha ayudado a responder las preguntas de: que necesito, hacia donde va dirigida la información y que elementos deben conformarla.

Para el diseño del Front-end, se ha buscado diversas referencias visuales en la web, para poder gestar una idea de diseño que satisfaga las necesidades de los usuarios de la aplicación, tenga un grado de usabilidad alto y al mismo tiempo permita al programador ciertas dosis de creatividad.

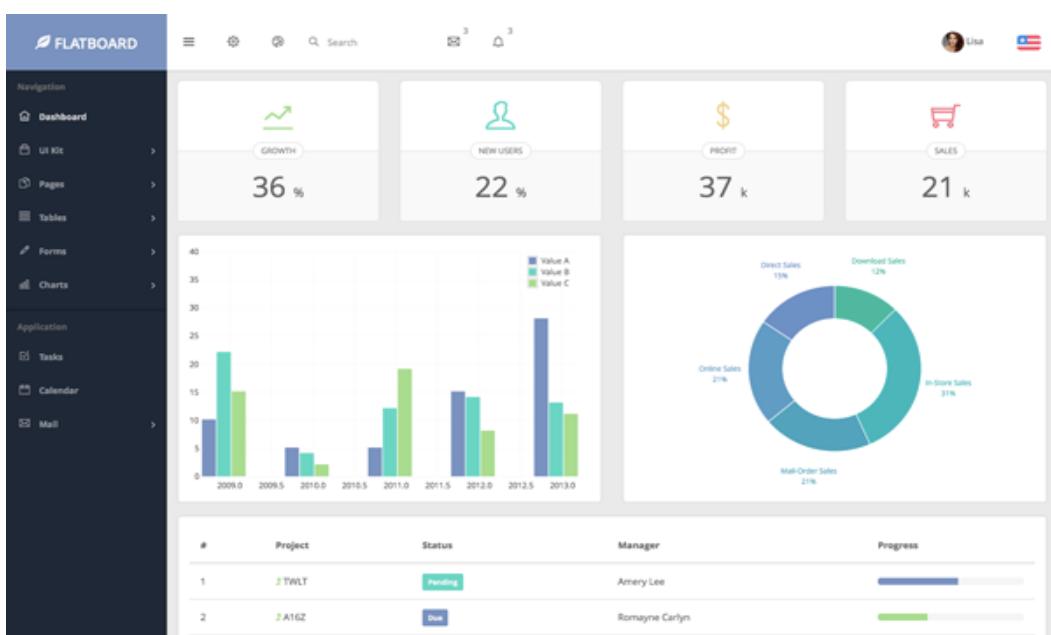


Ilustración 2: Ejemplo de diseño usable y atractivo

2.2. Estudio de la situación actual

2.2.1. Contexto:

Actualmente, el sistema empleado para atender incidencias de los clientes es tradicional y básico: el cliente llama por teléfono, es atendido y se le intenta solucionar la incidencia. En caso positivo, la incidencia queda resuelta pero no archivada. En caso de recibir en días próximos incidencias similares, se intentan solucionar en base a la experiencia previa. En caso de no ser resuelta al momento, la incidencia queda apuntada como objetivo de la empresa a realizar y en mayor o menor medida acaba

resuelta en un plazo de horas o días.

Con nuestra solución, lo que tratamos de conseguir es que las incidencias se registren en el sistema en el momento de aparecer y que este sistema sea multiusuario. Dichas incidencias quedan registradas en el sistema y pueden ser consultadas en cualquier momento y, en caso de surgir incidencias similares en un espacio de tiempo posterior, contar con la documentación necesaria para hacerle frente. Además, el sistema combatirá cualquier probabilidad de olvido de la incidencia o la solución hallada, así como los datos de contacto del cliente que la reportó. Finalmente, de entre las posibles implementaciones posteriores al proyecto, está la de gestionar las implantaciones futuras, a nivel de producción, justo antes de ser implantadas, con el objetivo de almacenar previamente todo tipo de información útil acerca de las distintas características de dicha implantación. Por tanto, nuestra aplicación, además de atender a las distintas incidencias, también permitiría la gestión de implantaciones desarrolladas en la fase anterior al despliegue.

Por otra parte, resulta práctico que nos paremos a analizar qué tipo de soluciones software existen ya en el mercado, cuáles son sus características, potencial y limitaciones, y de qué manera podemos justificar el uso de nuestra propia solución.

A continuación, vamos a citar 3 posibles productos que, por características, podríamos considerar competencia directa del nuestro.

- FreshService: <https://freshservice.com/es/lp/herramienta-gestion-incidencias-itil/>
- ActivaLink: <https://www.activalink.com/software-la-gestion-incidencias>
- ServiceTonic: <https://www.servicetonic.es/software-atencion-al-cliente-helpdesk/>

Estos tres productos son plenamente ejemplares, ya que existen múltiples soluciones informáticas, amén de estas, que desempeñan la misma labor que el resultado de nuestra idea; o abarca, entre otras funciones relacionadas con el entorno, el terreno que acapara nuestra solución. Estamos ante aplicaciones que se encuentran en un escalafón del ciclo de vida más alto que nuestro desarrollo, y gozan de un estado de madurez que no posee nuestro software. Los controles y las interfaces tienen buen acopio de cualidades y resulta palpable el seguimiento de los criterios de usabilidad y diseño. En resumen, nos enfrentamos de primera mano a productos de gestión y soporte con un bagaje más extenso y que se han consolidado con el paso del tiempo. Productos que cuentan con más reconocimiento y menos barreras de entrada. Sin embargo, a la hora de utilizarlos, observamos que estas soluciones empresariales tan sofisticadas cuentan con una política de uso que se aleja de los intereses actuales de nuestra empresa. En los tres casos nos ofrecen un periodo de prueba en el que contaremos, presumiblemente durante un mes, con todas o casi todas las funcionalidades que nos brinda este software. A partir del trigésimo primer día, la licencia gratuita expiraría y nos veríamos forzados a desembolsar la cantidad monetaria correspondiente y periódica para poder seguir utilizando el servicio. A priori, nuestra empresa no está interesada en ninguna de estas soluciones "de pago", por lo que nos vemos instados al desarrollo de un software similar que reúna las mismas funcionalidades y sea interna y a medida para nuestra empresa.

Existen alternativas gratuitas como <https://www.uuupsapp.com/> que permite descargar una aplicación adaptada a dispositivos móviles y al mismo tiempo, es gratuita. No obstante, soluciones móviles no son interesantes para el entorno de la empresa, ya que esta nos pide que la aplicación este orientada a escritorio y sea fácilmente escalable según vaya necesitando nuevas funcionalidades, o desecharando funcionalidades que ya no resultan atractivas. Por tanto, con estos dos últimos argumentos dejamos claro que si bien existen soluciones muy atractivas que en general podrían resolver la necesidad de la empresa, es mucho más eficiente tanto en el terreno económico como en el funcional, contar con un software a medida que pueda adaptarse día a día a aquello requerido por la empresa.

2.2.2. Normativa y legislación

Actualmente, si queremos que nuestra aplicación cumpla con las normativas y legislaciones vigentes, de cara a ser un producto legal, veraz, honesto y leal, debemos consultar diversas normativas referentes al **uso y distribución** de software empresarial, así como la actual **Ley Orgánica de Protección de Datos**. Puesto que nuestro software va a ser un producto interno de la empresa, por lo que su distribución será interna también, y dicha empresa trabaja en ámbito nacional, atenderemos a aquellas legislaciones que se apliquen al término nacional, pero no a aquellas fuera del mismo:

- **Normativa corporativa de software legal (cita):** "La normativa de uso de software legal es una obligación de la organización y de cada una de las personas que la forman. En la presente normativa, de obligado cumplimiento para los empleados de la organización, se garantiza el cumplimiento de las restricciones legales existentes en material jurídica.

La legislación que regula la propiedad del software es fundamentalmente el Texto Refundido de la Ley de Propiedad Intelectual, aprobada por el Real Decreto Legislativo 1/1996, de 12 de abril.

La norma ISO 27002 que establece un código de buenas prácticas para la gestión de la seguridad de la información en su dominio 15, concretamente el control 15.1.2 Derechos de propiedad intelectual detalla las directrices que se deben tener en cuenta para proteger cualquier material que pueda ser considerado propiedad intelectual.

Desarrollo: En esta normativa corporativa afecta a todo el software desarrollado por la propia organización o a aquel software desarrollado por terceros proveedores y que la organización ha adquirido. Normas generales de uso software legal A continuación se especifican los aspectos que todo empleado debe conocer respecto al uso del software dentro de la organización. • Todos los productos software que sean utilizados por los empleados en los entornos corporativos deberán encontrarse debidamente autorizados y se deberá disponer de la correspondiente licencia:

- El empleado en ningún caso podrá hacer copia de los productos software puestos a su disposición para el correcto desempeño de sus tareas.
- Ningún miembro de la empresa instalará o utilizará programas sobre los

sistemas de información de la organización para los que no esté autorizado.

- En caso de que un empleado utilice productos ilegales o productos sobre los que no esté autorizado podrá dar lugar a una sanción disciplinaria, así como acarrear responsabilidades civiles y/o penales derivadas de la legislación vigente.
 - Únicamente el área técnica competente tiene la potestad de instalar productos software en los equipos.
 - Se dispondrá de un registro de licencias de software donde se indique la fecha de adquisición, vigencia de la licencia (si la hubiese), número de usuarios permitidos (si existiese limitación), pruebas de la adquisición, y el producto al que hace referencia. Cualquier cambio sobre el software debe indicarse en dicho registro. De esta forma se garantiza el correcto inventariado de las licencias y por tanto la legalidad de los productos software utilizados por la organización. " fuente
- **Licencia de Software o de nuevas Tecnologías en España:** Encontramos como referencia la "Ley de la Propiedad Intelectual", que expone lo siguiente:
 - Cuando un programador/empresa tecnológica crea un software para un cliente, puede transmitir los derechos de explotación del mismo. Para ello, es muy importante ser muy específico en los derechos que se transmiten, y que según la LPI son cuatro:
 1. Distribución
 2. Transformación
 3. Reproducción
 4. Comunicación pública

El art. 8 de la LPI es el único caso que contempla que una persona jurídica pueda ostentar derechos morales de autor (obra colectiva), y al mismo tiempo reconoce que los derechos de explotación corresponderán al coordinador y el que se encarga de la divulgación del software. Así mismo. Reza:

"Se considera obra colectiva la creada por la iniciativa y bajo la coordinación de una persona natural o jurídica que la edita y divulga bajo su nombre y está constituida por la reunión de aportaciones de diferentes autores cuya contribución personal se funde en una creación única y autónoma, para la cual haya sido concebida sin que sea posible atribuir separadamente a cualquiera de ellos un derecho sobre el conjunto de la obra realizada.

Salvo pacto en contrario, los derechos sobre la obra colectiva corresponderán a la persona que la edite y divulgue bajo su nombre."

Por otra parte, dicha ley también recoge lo siguiente:

- " En el contrato de licencia de software **no podrán prohibirse** las siguientes actividades:
 1. Realizar copias de seguridad, en la medida en que resulte necesario para la utilización del programa por parte del licenciatario. La copia de seguridad está, por tanto, exenta de autorización, sin posibilidad de pacto en contrario, en cuanto resulte necesaria para la utilización del

programa.

2. Observar, estudiar o verificar el funcionamiento del programa, con el fin de determinar las ideas y principios implícitos en cualquier elemento del programa, siempre que lo haga durante cualquiera de las operaciones de carga, visualización, ejecución, transmisión o almacenamiento del programa que tiene derecho a realizar.
 3. Reproducir y traducir el código, cuando sea indispensable para obtener la información necesaria para la interoperabilidad de un programa creado de forma independiente con otros programas.
 4. Asimismo, la Ley de Propiedad Intelectual recoge una serie de actividades en las que el licenciatario no necesitará la autorización del licenciante, salvo pacto en contrario: La reproducción o transformación de un programa de ordenador, incluida la traducción de errores, cuando dichos actos sean necesarios para la utilización del mismo por parte del usuario legítimo, con arreglo a su finalidad propuesta.
 5. La realización o autorización de versiones sucesivas del programa o de programas derivados del mismo."
- En el **Real Decreto Legislativo**, referente a los "programas de ordenador", encontramos las siguientes referencias:
 - "Disposiciones generales: La propiedad intelectual de una obra literaria, artística o científica corresponde al autor por el solo hecho de su creación. La propiedad intelectual está integrada por derechos de carácter personal y patrimonial, que atribuyen al autor la plena disposición y el derecho exclusivo a la explotación de la obra, sin más limitaciones que las establecidas en la Ley."
 - "Sujetos: Se considera autor a la persona natural que crea alguna obra literaria, artística o científica. No obstante, de la protección que esta Ley concede al autor se podrán beneficiar personas jurídicas en los casos expresamente previstos en ella."
 - "Obra colectiva: Se considera obra colectiva la creada por la iniciativa y bajo la coordinación de una persona natural o jurídica que la edita y divulga bajo su nombre y está constituida por la reunión de aportaciones de diferentes autores cuya contribución personal se funde en una creación única y autónoma, para la cual haya sido concebida sin que sea posible atribuir separadamente a cualquiera de ellos un derecho sobre el conjunto de la obra realizada."
 - "Duración y cómputo: Los derechos de explotación de la obra durarán toda la vida del autor y setenta años después de su muerte o declaración de fallecimiento."
 - "Contenido y características del derecho moral: Corresponden al autor los siguientes derechos irrenunciables e inalienables:
 1. Decidir si su obra ha de ser divulgada y en qué forma.
 2. Determinar si tal divulgación ha de hacerse con su nombre, bajo

seudónimo o signo, o anónimamente.

3. Exigir el reconocimiento de su condición de autor de la obra.
4. Exigir el respeto a la integridad de la obra e impedir cualquier deformación, modificación, alteración o atentado contra ella que suponga perjuicio a sus legítimos intereses o menoscabo a su reputación.
5. Modificar la obra respetando los derechos adquiridos por terceros y las exigencias de protección de bienes de interés cultural.
6. Retirar la obra del comercio, por cambio de sus convicciones intelectuales o morales, previa indemnización de daños y perjuicios a los titulares de derechos de explotación. Si, posteriormente, el autor decide reemprender la explotación de su obra deberá ofrecer preferentemente los correspondientes derechos al anterior titular de los mismos y en condiciones razonablemente similares a las originarias.
7. Acceder al ejemplar único o raro de la obra, cuando se halle en poder de otro, a fin de ejercitar el derecho de divulgación o cualquier otro que le corresponda."

Dicho decreto recoge más puntos, aparte de los aquí citados, y se pueden consultar **mediante el siguiente enlace:**

https://www.boe.es/diario_boe/txt.php?id=BOE-A-1996-8930

Por último, en la **Ley Orgánica de Protección de Datos** (LOPD), se mencionan los siguientes casos:

- Reglamento de desarrollo de la LOPD: "Productos de software: Los productos de software destinados al tratamiento automatizado de datos personales deberán incluir en su descripción técnica el nivel de seguridad, básico, medio o alto, que permitan alcanzar de acuerdo con lo establecido en el título VIII de este reglamento."

El Reglamento General de Protección de Datos (RGPD) europeo considera las siguientes cuestiones:

- Principios: Los principios de protección de datos se establecen del siguiente modo:
 - Los datos tienen que ser exactos, por tanto, si fuera necesario, habrá que actualizarlos.
 - Se recoge el deber de confidencialidad tanto para el responsable del tratamiento de los datos, como para todo aquel que intervenga en el proceso.
 - Es necesario el consentimiento expreso del titular de los datos para poder recabarlos y usarlos.
 - El titular de los datos personales tiene derecho a saber quién es el responsable del tratamiento de su información, y debe tener acceso, de forma sencilla e inmediata, a él, que, además, debe informar sobre los medios disponibles para ejercer sus derechos.

- Para evitar situaciones discriminatorias no se podrá tratar con datos cuya finalidad sea la de identificar la ideología, afiliación sindical, religión, orientación sexual, creencias u origen racial o étnico, tan solo porque el afectado de su consentimiento.
- Derechos: En el RGPD se establecen:
 - Derecho a rectificar tus datos inexactos o incompletos.
 - Derecho de oposición al tratamiento de los datos.
 - Derecho a suprimir tus datos si se usan para fines ilícitos o llega a término la finalidad para la que fueron recabados.
 - Derecho a conocer para qué van a ser usados y el plazo de uso de los mismos.
 - Derecho a solicitar la suspensión del tratamiento de tus datos, la conservación y la portabilidad de los mismos.
- Obligación de la empresa a realizar formación LOPD a sus trabajadores:
 - Conocer la privacidad de todos los datos a los que acceden y, por lo tanto, su **obligación de guardar secreto** sobre dicha información.
 - Usar los datos únicamente para los fines para los cuales han sido recabados.
 - **No divulgar las contraseñas** que tengan para acceder tanto a los sistemas informáticos como a los ficheros que contengan datos de carácter personal.
 - **Solicitar las autorizaciones necesarias** para el tratamiento de dichos datos siempre que se refieran a salidas o entradas de soportes informáticos.
 - **No utilizar con fines privados** los sistemas informáticos de la empresa, sin autorización del empresario.
- Régimen sancionador en la LOPD GDD 2019:
 - **Muy graves:** las que supongan una vulneración sustancial del tratamiento y tengan que ver con el uso de los datos para una finalidad diferente de la anunciada, la omisión del deber de informar al afectado, la exigencia de un pago para poder acceder a los datos propios almacenados, transferencia internacional de información sin garantías, etc. Este tipo de infracción prescribe a los 3 años.
 - **Graves:** las que supongan una vulneración sustancial del tratamiento y tengan que ver con datos de un menor recabados sin consentimiento, falta de adopción de medidas técnicas y organizativas necesarias para la efectiva protección de datos o, por ejemplo, el incumplimiento de la obligación de nombrar responsable o encargado de tratamiento de datos. Este tipo de infracción prescribe a los 2 años.
 - **Leves:** las restantes que no queden contempladas en los grupos anteriores. Prescribirán al año y se refieren a casos como la no transparencia de la información, el incumplimiento de no informar al afectado cuando lo haya solicitado o, por ejemplo, el incumplimiento por parte del encargado de sus obligaciones.

Cabe destacar que el código penal contempla en el artículo 197 aquellas acciones malintencionadas, relacionadas con el uso indebido de software: "Será castigado con una pena de prisión de seis meses a dos años o multa de tres a dieciocho meses el que, sin estar debidamente autorizado, produzca, adquiera para su uso, importe o, de cualquier modo, facilite a terceros, con la intención de facilitar la comisión de alguno de los delitos a que se refieren los apartados 1 y 2 del artículo 197 o el artículo 197 bis:

- a) Un **programa informático**, concebido o adaptado principalmente para cometer dichos delitos.
- b) Una **contraseña de ordenador**, un código de acceso o datos similares que permitan acceder a la totalidad o a una parte de un sistema de información.
- explotar económicamente de cualquier otro modo una obra o prestación protegida sin la autorización de los titulares de los derechos de la propiedad intelectual, sustituyéndose, además, el elemento subjetivo 'ánimo de lucro' por el de 'ánimo de obtener un beneficio económico directo o indirecto' "

Fuentes:

1. https://protecciondatos-lopd.com/empresas/obligacion-informar-trabajadores/#Contrasenas_de_los_equipos
2. <https://infoautonomos.eleconomista.es/tecnologia-pymes-autonomos/ley-de-proteccion-de-datos-lopd/#derechos>
3. <https://www.boe.es/buscar/act.php?id=BOE-A-2008-979>
4. <https://protecciondatos-lopd.com/empresas/cifrado-datos/>

2.3 Requisitos del sistema

2.3.1 Requisitos

Para poder trabajar correctamente de cara al proyecto, necesitamos un equipo hardware, así como algunas cuestiones software, que atiendan a las siguientes necesidades:

- El equipo debe contar con un espacio en disco de, al menos 240 GB, puesto que no conocemos el volumen que podría llegar a alcanzar la aplicación y, por tanto, preferimos no tener problemas de espacio a la hora de implementar y desplegar la aplicación una vez terminada.
- Necesitamos que el equipo cuente con un procesador lo suficientemente potente como para trabajar con las herramientas de desarrollo y que, en caso de tener todos los procesos activos, el equipo no quede desamparado en el sentido de no contar con un hardware que le otorgue el rendimiento que requiere.
- Requerimos, al menos, 8 GB de memoria RAM, ya que vamos a emplear algunas herramientas y características del sistema que consumen gran cantidad de recursos en el ordenador, y es algo a tener en cuenta. Concretamente, hablamos de entornos de desarrollo integrado, donde la aplicación será compilada y levantada en un servidor virtualizado, mientras que esta es visualizada en un navegador web moderno, con el consumo que esto conlleva.
- Necesitamos contar con herramientas de desarrollo adecuadas para nuestro fin, como un IDE, un navegador, un servicio Apache, una base de datos MySQL, etc. Que nos permita un desarrollo avanzado y sin restricciones y, posteriormente, emular con la mayor veracidad posible, un entorno real de trabajo, donde la aplicación se encuentre instalada y totalmente funcional.
- Por todo lo anterior, requerimos un sistema operativo de 64 Bits (x64) que permita la instalación y uso de herramientas modernas, al mismo tiempo que resuelva con soltura las interacciones con el hardware del equipo, proporcionando fluidez y estabilidad al sistema, y por ende, al entorno de trabajo.

2.3.2 Restricciones del sistema

Dado el punto anterior, repasaremos cuales son las características de nuestro equipo adjudicado en la empresa, ya que es dicho equipo aquel donde vamos a trabajar mayormente en la aplicación:

- **Procesador:** Intel Core i5-3570k a 3.40 GHz.
- **RAM:** Kingston Fury 2x4 GB DDR3
- **Almacenamiento:** Samsung EVO 240 GB SSD Sata III.
- **Sistema Operativo:** Windows 10® Pro 64 Bits.

Como vemos, el equipo es lo suficientemente solvente, al menos trabajando sobre números, como para llevar a cabo todas y cada una de las tareas requeridas para el desarrollo del proyecto. Podríamos decir, por tanto, que de la mano del sistema **no contamos con ninguna restricción**. La empresa, por otra parte, a la hora de desplegar la aplicación, cuenta con **servidores dedicados** para aplicaciones, por lo que **no vamos a tener problemas de almacenamiento ni de despliegue**.

2.4 Planificación del proyecto

2.4.1 Recursos del proyecto

A la hora de evaluar los recursos del proyecto, se han atendido a las siguientes cuestiones: *¿Que necesitamos para el desarrollo del proyecto? ¿Cuánto nos supone la inversión total a la hora de querer llevarlo a cabo?* Estas y otras cuestiones quedan resueltas en los distintos apartados de este capítulo. No obstante, debemos ser concisos y repasar cuales son los recursos con los que contamos y que serán indispensables para que el proyecto esté finalizado dentro de los plazos establecidos:

- **Recursos de bienes inmateriales:** Desconocemos muchas de las herramientas empleadas para llevar a cabo el proyecto. No obstante, nos constituyimos como un programador y por tanto, contamos con la inquietud y conocimiento base acerca de los lenguajes de programación, en especial aquellos catalogados como “Orientados a objetos”. Por tanto, no resultará una tarea ardua acabar dominando dichas materias en pos del desarrollo final. Por otra parte, encontramos el factor ‘tiempo’. Contamos con varios meses para la documentación, desarrollo y pruebas finales del software. Por tanto, no tememos por el hecho de quedar desabastecidos a la, valga la redundancia, hora de hablar del tiempo de desarrollo.

- **Recursos materiales y de entorno:** Contamos con una serie de equipos capaces de hacer posible que la aplicación se desarrolle con éxito. Por tanto. Por otra parte, la empresa posee herramientas, así como el propio entorno, que propician el correcto funcionamiento estructural y logístico del procedimiento que debe llevarse a cabo.
- **Recursos económicos:** Como veremos en siguientes apartados, el factor económico es esencial a la hora de evaluar un proceso. Gracias al uso de herramientas Freeware, y que la empresa cuenta con las plataformas necesarias para desplegar la aplicación, solo queda estudiar el factor económico personal, en cuyo caso resulta favorable tal y como veremos con más detalle en futuros apartados.

2.4.2 Tareas del proyecto

Las tareas del proyecto se pueden **clasificar** mediante la siguiente estructura:

- **Análisis de requisitos:** Requerimos un tiempo para analizar la viabilidad del proyecto, las herramientas requeridas para su desarrollo, así como realizar un estudio más profundo acerca de otros factores, como el factor económico, el factor de la duración de las tareas, etc. El tiempo estimado es de **una semana laboral**.
- **Diseño de la aplicación:** Una vez realizado el análisis del proyecto, es necesario comenzar con el diseño de la aplicación. Teniendo ya en cuenta cuales son las herramientas necesarias, el entorno de trabajo, los lenguajes empleados para tal finalidad, etc. Comenzaremos especificando las partes diferenciadas de la aplicación: interfaz, controladores lógicos, lógica de negocios, acceso a la base de datos, etc. Así como la forma de diseñar y desarrollar las partes de la aplicación en todas sus fases. El tiempo estimado es de **3 a 4 semanas laborales**.
- **Implementación de la aplicación:** Una vez diseñados todos los elementos de la aplicación, es necesario llevarlo a cabo y poner en funcionamiento la aplicación, solucionando los errores surgidos e intentando mejorar a la par que seguimos con el desarrollo final de la aplicación. El tiempo estimado para la implementación es de **2 a 3 semanas laborales**.
- **Despliegue y pruebas:** Una vez desarrollada la versión final de la aplicación (entendiéndose como final, la versión presentada en el proyecto), es necesario un intervalo de tiempo para efectuar pruebas en los distintas partes de la aplicación, para así pulir aquellos detalles que requieren ser mejorados. El tiempo estimado de duración de esta fase es de **3 días a una semana laboral**.
- **Documentación y manual del proyecto:** Tras concluir la aplicación, necesitamos un periodo para organizar la documentación y generar un manual de usuario. Si bien, esta documentación ha sido en gran parte realizada a lo largo del proceso de desarrollo, es en este punto cuando requerimos la organización de la totalidad de ésta, con el fin de preservar los conocimientos adquiridos y postular las distintas pautas para la comprensión del trabajo realizado. Se estima que dicha fase puede durar en torno a **una semana laboral**.

2.4.3 Planificación temporal

Sobre los subsiguientes apartados, toca hablar de como organizamos las tareas a lo largo del periodo de prácticas. Para ello, debemos de tener en cuenta el comienzo de las prácticas (*18 de marzo*), así como el día límite de entrega del proyecto (*10 de junio de 2019*). Contamos, por tanto, con un total de 12 semanas para el desarrollo íntegro del proyecto, con todo lo que eso conlleva. En definitiva, lo que se refleja en el siguiente calendario es el esquema de la planificación llevada a cabo:



Ilustración 3: Calendario de tareas para el proyecto

Nota: Como vemos, el tiempo de diseño de la aplicación se ha visto muy incrementado respecto a la previsión inicial, ya que ha sido necesario familiarizarse con el entorno de desarrollo y los lenguajes de programación (Frameworks) propuestos para tal fin. A esto, sumamos el desarrollo modular de Angular, que hace que el diseño sea más complejo a la hora de diseñar, pero más fácil e intuitivo al momento de implementar los distintos componentes.

2.5 Evaluación de riesgos

2.5.1 Lista de riesgos

No hay que descuidar el factor riesgo que tiene nuestro proyecto. Si bien no es algo sumamente ambicioso o que dependa de muchos factores externos, **toda actividad entraña sus riesgos**, tarde o temprano. Conviene analizar que riesgos podemos encontrarnos a lo largo de nuestro desarrollo:

- **Riesgos relacionados con el tiempo disponible:** El factor de riesgo más acusado es sin lugar a dudas el **incumplimiento de los plazos de entrega** predefinidos. Debemos llevar una organización del trabajo muy eficaz para evitar la aparición de este riesgo potencial.
- **Riesgos personales:** Entre los riesgos posibles, encontramos el riesgo relacionado con la salud e integridad personal del programador
- **Incumplimiento con la normativa:** Un riesgo muy a tener en cuenta es que nuestro software pueda vulnerar los derechos de uso y protección de datos en la empresa o fuera de ella. Debemos guardar cautela con este tema e informarnos debidamente antes de desplegar la aplicación para su uso correspondiente
- **Otros riesgos materiales:** También conocidos como *imprevistos de suma gravedad*. Estos pueden generar un atraso en los plazos de desarrollo de la aplicación, o la **pérdida grave de desarrollo o información**. Estos pueden ser debidos a un fallo hardware o de sistema operativo, o causado por un agente externo, como por ejemplo un *malware*. Para evitar o más bien, minimizar los daños relacionados con este riesgo potencial, debemos procurar siempre trabajar bajo **Control de Versiones**, como el caso de ‘Git’. Y emplear, si es posible, otras formas de generar y almacenar **copias de seguridad** de la aplicación, como por ejemplo comprimir el fichero de la aplicación y almacenarlo en un dispositivo externo (USB) o en *la nube* (Google Drive).

2.5.2 Catalogación de riesgos

Podemos catalogar los riesgos en tres tipos: leves, graves y muy graves.

- **Leves:** Son aquellos riesgos que, en caso de surgir, plantean una solución poco o medianamente costosa. Pueden ser solucionados de inmediato, o al cabo de un corto periodo de tiempo. Por ejemplo, que la empresa quede sin acceso a internet durante una jornada laboral, perdiendo una herramienta de trabajo, como es internet.
- **Graves:** Constituyen una fuerte amenaza para el proyecto. Son fuentes de riesgo, aquellas relacionadas con el **equipo de trabajo** o el **macroentorno** que engloba a la empresa o a las herramientas de trabajo. Por ejemplo, que el **control de versiones haya dejado de funcionar** sin previo aviso durante un par de días, haciendo que, en caso de querer revertir el proyecto, podamos incluso **perder el desarrollo avanzado** de muchas de las funciones, acarreando un gasto de tiempo y recursos de la empresa.
- **Muy graves:** Se entiende como riesgos muy graves aquellos que son sumamente peligrosas para el desarrollo del proyecto y, en muchos de los casos, irreversibles. Son las menos probables pero su causa podría provocar un efecto devastador en el tiempo de desarrollo. Un ejemplo de este riesgo sería la **pérdida humana o material y económica de los recursos que hacen sostenibles el acceso a las prácticas**.

2.5.3. Plan de contingencia

Para solventar algunos de los riesgos planteados anteriormente, hemos elaborado un plan de contingencia, cuyo objetivo es tratar de **minimizar lo máximo posible** los riesgos ocasionados por los distintos factores, tanto internos como externos.

1. **Plan de respaldo:** Actuaremos sobre las amenazas antes de que estas puedan aparecer. Para ello, se tomarán las siguientes medidas:
 - I. Crearemos una copia de seguridad semanalmente con todos los progresos realizados en la aplicación, con el fin de que, en caso de fallar algún proceso, tengamos la copia anterior para que sean restaurados todos los archivos.
 - II. Organizaremos un control de versiones que diariamente se irá actualizando. A su vez, contaremos con un repositorio remoto donde todos los cambios quedarán registrados una vez comprobado que el comportamiento de la aplicación es correcto
 - III. Tendremos un segundo equipo preparado en casa, con todos las herramientas y recursos instalados y configurados, en caso de que el equipo del entorno de trabajo falle en según qué circunstancias.
2. **El plan de emergencia:** Debemos contar con un plan de actuación en caso de la posible aparición de una amenaza. Dicho plan se materializa en las siguientes medidas:
 - I. Contaremos con las herramientas de descompresión y recepción de datos desde remoto, para poder reponer en el menor tiempo posible cualquier incidencia relacionada con el desarrollo de la aplicación.
 - II. La empresa dispondrá de un equipo y un servidor de apoyo, así como un **SAI** que nos permita anteponernos a la pérdida de información, almacenando la información rápidamente siguiendo un protocolo de actuación.
3. **El plan de recuperación:** Contempla las medidas necesarias después de controlar la amenaza. Su finalidad es restaurar el estado de las cosas tal y como se encontraban antes de surgir la amenaza. Para ello, lo queharemos será disponer de un **protocolo de recuperación**, el cual constará de los siguientes pasos:
 - I. Analizar la magnitud de los daños
 - II. Reponer los equipos hardware afectados
 - III. Reponer las herramientas software afectadas
 - IV. Recuperar en medida de lo posible, la información
 - V. Reestructurar el proyecto, procurando la máxima similitud con el último estado conocido.

2.6. Presupuesto

2.6.1 Estimación de coste material

La aplicación desde el punto de vista del software no requiere ninguna inversión económica. Esto es debido a que todo el software empleado para el desarrollo de la misma posee licencias ‘freeware’: *Visual Studio Code, licencia de prueba de PhpStorm, Xampp, Postman, etc.* A la hora de poner en marcha el servidor del proyecto, hemos empleado uno de los servidores de la empresa para el software nuevo en pruebas. Por tanto, no hemos necesitado contratar un servicio de *hosting* web. Tampoco hemos invertido en equipos hardware, ya que todo el desarrollo se realizó con el equipo de la empresa, empleando el equipo de casa como apoyo circunstancial. En definitiva, si tenemos que hablar de costes materiales, debemos atender a criterios de segundo orden, como son **los gastos de mantenimiento de los equipos**. No podemos saber con exactitud cuáles han sido los costes totales en el tiempo de desarrollo. No obstante, podemos hacer una estimación aproximada: En el año 2018, la media anual de consumo eléctrico fue de 57,17€ mensuales. Teniendo en cuenta que nuestro equipo en la empresa constituye un bajo porcentaje de consumo respecto al total, digamos un 5% del consumo total, podríamos estimar que el gasto de mantenimiento eléctrico del equipo constituiría, en estos 3 meses de desarrollo, un total de $171,5/20 = 8,575\text{€}$, por dilucidar un coste material.

Por otra parte, el equipo con el que desarrollamos requería una licencia por volumen de Windows 10® Pro. El precio de la licencia ascendió a **2,10€**, ya que se trataba de una licencia por volumen bastante económica.

2.6.2 Estimación de coste personal

Al tratarse de un trabajo realizado durante el contrato en prácticas, el desarrollo se ha realizado durante unas horas de trabajo no remuneradas, por lo que no podremos tratar conceptos como *sueldo o nómina*. Sin embargo, podemos concretar, como desarrolladores que somos del producto, un precio a nuestro producto, ya que, como autores de la obra, tenemos pleno derecho a decidir qué coste tiene en el mercado. Siendo un desarrollo interno de la empresa, el coste para la empresa del producto y mantenimiento va a ser cero.

Por otra parte, debemos considerar cuales han sido los gastos meramente personales durante el periodo de desarrollo. La empresa se encuentra, a día de hoy, localizada en Bellavista. Esto quiere decir que sus oficinas se encuentran aproximadamente a **19 kilómetros** de distancia del domicilio actual. Si calculamos a partir de la distancia, el gasto por desplazamiento en función del consumo medio del vehículo empleado, resulta la siguiente solución:

13 semanas de prácticas resultan en 65 días laborables en la empresa, los cuales resultan en **62 días** si suprimimos 3 días no laborables por festividad. En total, 19 km de ida y 19 km de vuelta suman un total de **38 km diarios**. El producto de 38 km y 62 días arrojan un resultado de **2.356km totales realizados en el transporte desde el comienzo de las prácticas hasta el fin**. Si realizamos un estudio acerca de la media del consumo, comprobamos que el vehículo posee un **consumo medio aproximado de 4.5 L/100km**. Realizando una simple regla de tres, comprobamos que, si en 100 km hemos realizado un consumo de 4,5 litros, entonces, a los 2.356 km hemos invertido un total de **106,02 litros** en combustible. Teniendo en cuenta el 3% de incremento en el precio de los carburantes desde el inicio de las prácticas, hemos decidido emplear una media, resultado de 1,23 y 1,26 €/L. En total, el precio medio en estos meses del **carburante diésel** es de **1,245€/L**. En definitiva, si nuestro consumo es de 106,02 litros de carburante diésel, la inversión personal **aproximada** a la hora de desempeñar las prácticas y, además, desarrollar este software, ha sido de **131,99€**.

Además de todos los costes anteriores, existen una serie de costes intrínsecos relacionados con la actividad laboral, como la adquisición de calzado y ropa, que no son procedentes para dicho análisis y, por tanto, **no van a ser objeto de valoración**.

2.6.3 Resumen y análisis coste beneficio

Resulta sencillo afirmar que el costo beneficio resulta negativo, en el sentido de que la aplicación ha requerido de un tiempo de desarrollo y una inversión económica por parte del programador y la empresa que, a la hora de la verdad, no aporta ningún beneficio monetario. Sin embargo, hay que tener en cuenta que el objetivo era crear un software especializado en atender incidencias y gestionar implantaciones para la propia empresa y que, de hecho, el objetivo ha quedado resuelto, proveyendo a la empresa de un software de soporte a medida sin costo alguno. Esto, a la empresa, si le ha proporcionado un costo beneficio considerable, tiendo en cuenta que el precio por desarrollo de una aplicación es en muchas ocasiones un proceso costoso a nivel económico y humano. Por nuestra parte, queremos recordar que cuando hablamos de beneficios y de coste, no solamente debemos tener en cuenta el ya nombrado económico. Aunque este parámetro ha sido tomado como referencia en este apartado, cabe destacar la presencia de otros valores que acostumbran ser más valiosos, como el conocimiento. Podríamos hablar de un mal costo beneficio por la parte económica. Sin embargo, el tiempo de desarrollo ha permitido alcanzar unas cuotas de conocimiento muy altas en lo relacionado a la materia que se trabaja en la empresa. Vale por tanto la pena el coste económico, ya que a cambio hemos recibido un beneficio académico muy importante.

2.7. Conclusiones

Podemos concluir que, tras el exhaustivo análisis de viabilidad, el producto resulta viable y que, por tanto, vamos a llevar a cabo el desarrollo. Contamos con el tiempo material y las herramientas necesarias para ello, y por tanto resulta viable y asequible para el equipo de programación.

2.7.1 Beneficios

- El software empleado para su desarrollo posee licencias de uso gratuito, por lo que el objetivo de completar el desarrollo no se verá frustrado por los bienes económicos del programador ni de la empresa
- El equipo de desarrollo de la empresa conoce bien las herramientas y Frameworks que se van a emplear para desarrollar la aplicación. Por tanto, contamos con apoyo ante posibles imprevistos a nivel de programación y de tiempo.
- La empresa nos da la libertad de utilizar todos sus recursos, ya sean a nivel de equipo, soporte, herramientas de desarrollo, material didáctico, etc. Así como el tiempo que requiramos para completar el trabajo

2.7.2. Inconvenientes

- Partimos de Frameworks y lenguajes de programación poco empleados o, directamente, desconocidos en algunos casos por lo que, durante la primera fase de desarrollo, vamos a emplear bastante tiempo aprendiendo cómo funcionan las distintas herramientas y lenguajes a utilizar. Esto provoca que tengamos al principio una visión distorsionada acerca del tiempo total que pueda requerir el proyecto, ya que no sabemos con certeza la cantidad de tiempo que vamos a necesitar.

3. Análisis

3.1. Introducción

Como ya hemos visto, el enfoque de nuestra aplicación orientada a la web será gestionar implantaciones software y hardware en las sedes de los distintos clientes que requieran algún servicio que suministra la empresa para la que desarrollaremos esta herramienta. En dicha aplicación deberán verse reflejadas las incidencias acontecidas posteriormente al despliegue de estas implantaciones, con información de estas incidencias, los equipos afectados, solución y observaciones, etc. Para controlar las incidencias, aparte de su identificador único, ID, utilizaremos un 'estatus': un campo numérico en la base de datos en el que especificaremos la situación actual de la incidencia, siendo '0' las incidencias por resolver, '1' las incidencias resueltas y '2' aquellas incidencias que hayan sido borradas de forma lógica del Front-End.

Por tanto, se pide que nuestra aplicación reúna los siguientes requisitos iniciales para que cumplan las expectativas del desarrollo:

- Listar implantaciones e incidencias: La aplicación debe, por una parte, listar las implantaciones actuales de la empresa, tanto aquellas que ya hayan sido desplegadas en la sede o entorno de trabajo de un cliente, como aquellas que se encuentran en fase de despliegue o en la fase final de su desarrollo. Por otra parte, se pide que dicha aplicación muestre todas las incidencias registradas en la empresa. Dichas incidencias serán visualizadas en la ventana principal de la aplicación (Dashboard) una vez que el usuario haya iniciado sesión y, además del título y la descripción de la incidencia, también se mostrarán datos relacionados con la incidencia en cuestión, como el nombre de la implantación a la que pertenece, el proyecto al que se encuentra ligado, el nombre del contacto, entre otras cuestiones. Desarrollaremos la posibilidad de filtrar las incidencias listadas por proyecto, haciendo que podamos filtrar por pestañas aquellas incidencias pertenecientes a un proyecto concreto (Actualmente, la empresa con la que trabajamos cuenta con dos proyectos, que son Encolate® y Logistic Dock®), y también filtrar por implantaciones. Esta última, dado la cantidad de implantaciones distintas que pueden existir dentro de un proyecto determinado, se va a realizar alrededor de una barra de búsqueda que se encontrará integrada en la interfaz gráfica.
- Añadir nuevas implantaciones e incidencias: Será necesario que la aplicación permita agregar nuevas implantaciones en nuestro sistema. Esto se realizará mediante un formulario que se encuentre conectado con la base de datos y permita añadir la nueva implantación para que pueda ser utilizada en la misma aplicación, por ejemplo, para añadir incidencias relacionadas con ella. Al mismo tiempo, la aplicación debe permitir la inclusión de nuevas incidencias que puedan aparecer y que necesiten ser consultadas en un futuro. Para ello, se usará también una ventana de formulario, donde el usuario debe añadir información acerca de la incidencia a registrar.

- Editar las incidencias e implantaciones creadas: Además de añadir implantaciones e incidencias, la aplicación debe permitir editar campos de dichas incidencias creadas, por ejemplo, en caso de haber cometido un error al momento de su creación
- Eliminar incidencias e implantaciones mostradas en la aplicación: En nuestro caso, la aplicación no debe borrar los registros de la base de datos, sino que se procederá a emplear un 'borrado lógico' de dicho registro. Este borrado asegura la preservación información en la base de datos en caso de querer recuperar dicha información en un caso específico.

3.2. Requisitos funcionales de usuarios

El usuario común de la aplicación serán los desarrolladores web de la empresa, que a menudo son el contacto directo con los clientes de la empresa que reportan las incidencias. Dichos usuarios no estarán sujetos a un solo rol, sino que en nuestra aplicación contemplamos su distinta utilización en función del rol del usuario que la maneja. En la primera fase de desarrollo (Actual), se definen cuatro roles de usuario:

- Administrador global (Admin): Puede hacer uso de todas las funcionalidades que proporciona la aplicación.
- Técnico de implantaciones: Entre sus competencias, éste puede:
 - Crear implantaciones para los proyectos y clientes asignados
 - Editar implantaciones para clientes asignados
 - Asignar implantación/es a los técnicos de soporte
- Técnico de soporte: Éste puede:
 - Listar implantaciones asignadas
 - Añadir y/o editar incidencias para las implantaciones que tiene asignada
- Técnico de facturación: Los usuarios ligados a este rol pueden:
 - Listar todas las implantaciones
 - Añadir y/o editar soporte de cara a la facturación.

La aplicación debe controlar el acceso a las diferentes funcionalidades dependiendo del rol del usuario que inicie sesión. Dicho control entrará en funcionamiento al momento de ingresar el usuario y la contraseña y autenticar si el usuario es un usuario registrado en la aplicación. La aplicación registrará la sesión en el navegador, en el cual quedará constancia del usuario y su rol durante el tiempo que la sesión quede activa y disponible. Los usuarios que puedan ingresar de forma satisfactoria en nuestra aplicación deben encontrarse previamente creados en la base de datos. Se dispondrá de una tabla en la que registremos los usuarios totales que puedan ingresar en la aplicación. Dichos usuarios podrán ser visualizados en la base de datos, pero la contraseña de ingreso a la aplicación se encontrará encriptada mediante Bcrypt. Bcrypt es una función de hashing de contraseñas diseñado por Niels Provos y

David Maxieres, basado en el cifrado de Blowfish. Se usa por defecto en sistemas OpenBSD y algunas distribuciones Linux y SUSE. Lleva incorporado un valor llamado **Salt**, que es un fragmento aleatorio que se usará para generar el **hash** asociado a la contraseña, y se guardará junto con ella en la base de datos. Así se evita que dos passwords iguales generen el mismo hash y los problemas que ello conlleva, por ejemplo, ataque por fuerza bruta a todas las passwords del sistema a la vez. Otro ataque relacionado es el de **Rainbow table** (tabla arcoíris), que son tablas de asociaciones entre textos y su hash asociado, para evitar su cálculo y acelerar la búsqueda de la contraseña. Con el Salt, se añade un grado de complejidad que evita que el hash asociado a una contraseña sea único. Mediante este sistema, el usuario contará con una seguridad mucho más alta que con cualquier otro algoritmo de encriptación, o directamente no contar con ninguno de ellos, ya que una de las ventajas de Bcrypt es que el tiempo que podría emplear una unidad de computación en aplicar fuerza bruta y resolver el algoritmo es mayor cuanto mayor sea la potencia de dicho ordenador. ¿Como? introduciendo un factor de trabajo, el cual te permite determinar qué tan costoso sería el hecho de generar un hash. En otras palabras, puedes limitar el tiempo en el que tarda en generar un hash un ordenador y utilizarlo en su contra.

El hecho de que el usuario se registre con la contraseña encriptada, además de ser debido a un tema de seguridad, se produce para que la aplicación cumpla la normativa de la LOPD que deja claro que la información comprometida de un usuario debe estar lo máxima protegida posible. Por tanto, para trabajar con el inicio de sesión en nuestra aplicación debemos contar con herramientas de cifrado y descifrado de algoritmos empleados en Bcrypt, que será la utilidad que vamos a emplear.

En la interfaz de usuario, debe encontrarse con facilidad la función para iniciar sesión, así como el formulario para llevarla a cabo y el acceso posterior a las funcionalidades que otorga la sesión activa. De igual modo, debemos contar con una función para cerrar la sesión, en pos de aumentar la seguridad de nuestra aplicación y permitir que la aplicación pueda ser utilizada al momento por otro usuario diferente. Los usuarios de la aplicación, como ya hemos mencionado, al tratarse de una aplicación interna de la empresa, serán los propios trabajadores que estén autorizados al uso correcto de dicha aplicación, y serán dados de alta en la base de dato por fuentes externas a esta aplicación. Por tanto, no contaremos con un menú de registro de usuarios, sino que los usuarios autorizados previamente tendrán que ser dados de alta de forma externa a la aplicación. Así mismo, la aplicación no contará con un CRUD de usuarios, por lo que no tendremos acceso a los métodos de listar, insertar, editar y eliminar usuarios en ella. Al menos en la primera fase (versión actual) de la misma.

3.3. Requisitos no funcionales

Una parte importante que debemos abarcar en el desarrollo de la aplicación, son los aspectos que citaremos a continuación:

- La aplicación atenderá a los requisitos iniciales que demanda el ciclo, como es el hecho de que la aplicación sea multiplataforma. Es decir, debemos procurar que dicha aplicación pueda ser consumida por la mayoría de dispositivos que componen el parque tecnológico actual, indiferentemente del sistema operativo que utilice o el hardware que lo componga. Así mismo, hay que tener en cuenta que todos los dispositivos que van a consumir la aplicación no cuenta con las mismas propiedades en cuanto a reproducción por pantalla. Si queremos que la aplicación pueda ser utilizada en un equipo portátil o un dispositivo móvil, además de un equipo sobremesa, debemos tener en cuenta la resolución de pantalla de los distintos dispositivos, y hacer que el desarrollo de las distintas interfaces siga un estándar de diseño que se acoja al diseño Responsive. En nuestro caso, vamos a desarrollar una aplicación y va a ser orientada a la web, por lo que es muy importante hacer hincapié en un diseño Responsive, ya que siendo una aplicación alojada en un servidor y consumida por un navegador, no vamos a tener un control directo de qué tipo de dispositivos van a consumir la aplicación, ni de qué forma (otras alternativas, como una aplicación empaquetada, un '.apk' en caso de Android, permite que las distintas versiones de la aplicación sean alojadas en la tienda de aplicaciones y, en función del dispositivo que tengamos, se descargue o no la aplicación. Hay casos en el que la tienda da el siguiente error: "Tu dispositivo no es compatible con esta versión de la aplicación". Nosotros no vamos a tener esa opción, por lo que es importante controlar ciertas cuestiones como estas). El diseño web Responsive o adaptativo es una técnica de diseño web que busca la correcta visualización de una misma página en distintos dispositivos. Se trata de redimensionar y colocar los elementos de la web de forma que se adapten al ancho de cada dispositivo permitiendo una correcta visualización y una mejor experiencia de usuario. Se caracteriza porque los layouts (contenidos) e imágenes son fluidos y se usa código media-queries de CSS3. El diseño Responsive permite **reducir el tiempo de desarrollo, evita los contenidos duplicados, y aumenta la viralidad** de los contenidos ya que permite compartirlos de una forma mucho más rápida y natural.

Fuente: <https://www.40defiebre.com/que-es/diseno-responsive>



Ilustración 4: Pautas del diseño Responsive

- Debemos proporcionar seguridad al usuario a la hora de utilizar la aplicación que impidan que puedan atacar a los datos almacenados, poniendo en juego la consistencia y fiabilidad de estos. Para mejorar la seguridad de nuestro sistema, utilizaremos el método de autenticación ya mencionado, que nos permita controlar en tiempo real quien se encuentra utilizando la aplicación; de modo que podemos controlar si el acceso es autorizado o no. En caso afirmativo, la aplicación mostrará los datos y las acciones requeridas para su funcionamiento. En caso contrario, la aplicación no facilitará el acceso. Para reforzar todo lo anterior, usaremos JWT (Json Web Token). Los JSON Web Tokens son un método abierto, estándar de la industria RFC 7519 para representar peticiones de forma segura entre dos partes. Esta herramienta, de la cual profundizaremos más adelante en el apartado Diseño, hará posible la navegación entre menús en la aplicación siempre y cuando, el objeto generado en el Back-End por un correcto acceso: 1. Exista, 2. Sea correcto y 3. No haya expirado. Nos permitirá, por tanto, un grado de seguridad mayor al que pueda existir con una simple validación de usuario, haciendo que la información de este sea inaccesible, insuplantable, y que el sistema sea invulnerable ante posibles intentos de *Phising*.

3.4 Diagramas de casos de uso / Casos de uso

Descripción de los casos de uso: Nuestra aplicación debe validar la sesión creada por el usuario para continuar siendo utilizada. Un cliente posee un rol concreto. Dependiendo del rol, un usuario puede (o no) hacer un determinado tipo de acciones del CRUD de implantaciones y/o incidencias

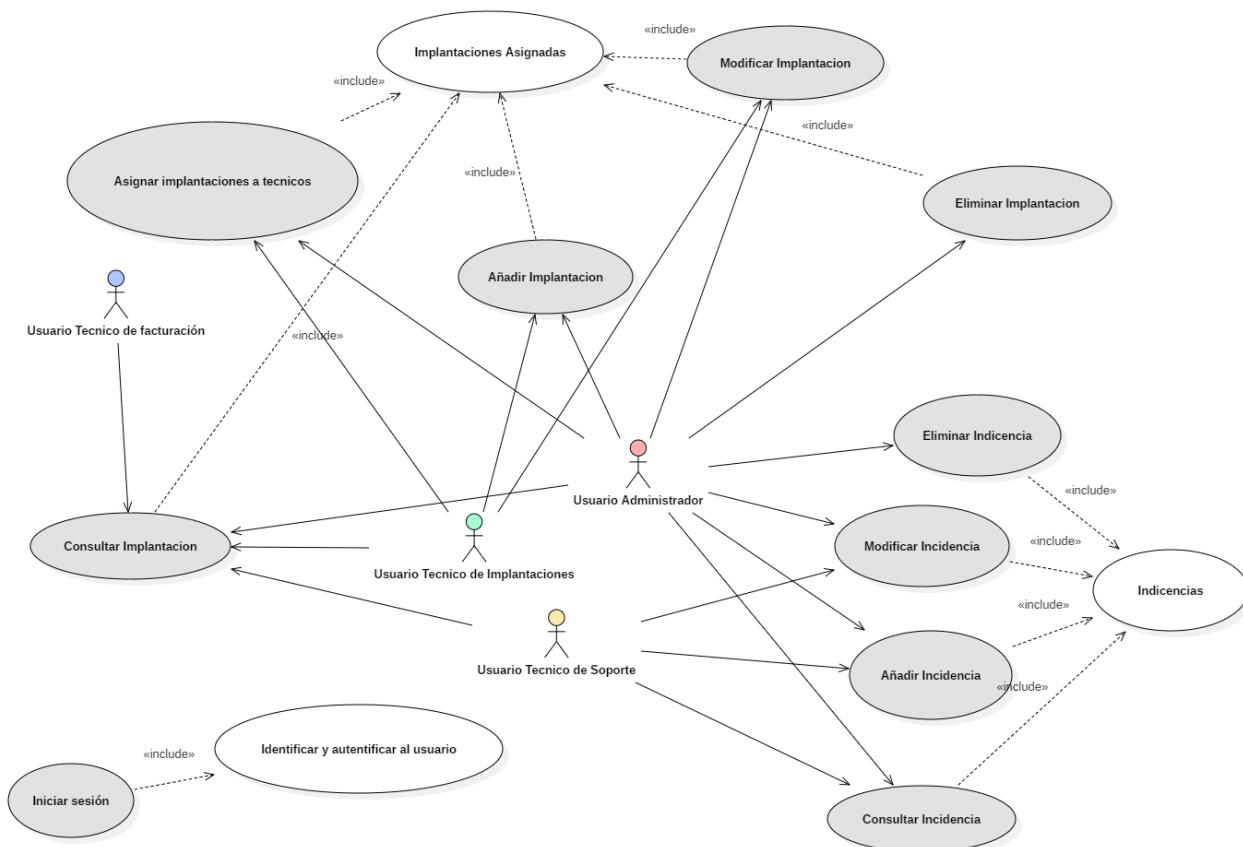


Ilustración 5: Diagrama de casos de uso

Como hemos tratado en el apartado 3.2, el usuario administrador será aquel que pueda gestionar todo. Posteriormente, tendremos técnicos de implantación, soporte y facturación, que podrán (en función del rol que se les otorgue) gestionar distintas acciones dentro de la aplicación.

3.5. Diagrama de actividad

Hemos desarrollado un diagrama de actividad en base a las exigencias funcionales de nuestra aplicación. El diagrama sería lo siguiente:

- 1 – El usuario intenta validar una sesión.
- 2 – Si la validación se produce de forma exitosa, el usuario inicia sesión y accede a la página principal. En caso contrario el usuario puede volver a intentar la validación (1), o bien terminar la actividad.
- 3 – En la página principal, podemos listar incidencias (4), añadir una nueva incidencia (5), entrar en la vista "detalles" de una de ellas (6) o acceder al menú de implantaciones (9).
- 4 – En la propia página principal consultamos las incidencias, pudiendo filtrar los resultados obtenidos por incidencias e implantaciones.
- 5 – En la página de añadir incidencias, podremos, mediante un formulario, añadir una nueva incidencia, que quedará registrada en la base de datos y se mostrará en la lista.

6 – En la vista de detalles, se mostrará más información de la incidencia. Desde esa misma interfaz nosotros podremos Editar (7) o Eliminar de forma lógica (8) una incidencia. También podremos cerrar sesión (14).

7 – Al hacer clic sobre editar, se debe abrir un formulario donde podamos editar la incidencia

8- Al hacer clic sobre Eliminar, nos debe aparecer un mensaje de confirmación. En caso afirmativo, la incidencia desaparecerá de la lista de incidencias, pero no de la base de datos (borrado lógico). En caso omiso, no ocurrirá nada.

9 – Desde la ventana principal, podremos acceder a un menú de implantaciones donde podemos, al igual que con las incidencias, añadir (10), listar (11), editar (12) y eliminar (13).

10 – Desde dicha ventana, podremos añadir una nueva implantación, que quedará registrada en la base de datos.

11 – Desde la ventana de listar, podremos consultar el número total de implantaciones de cada uno de los proyectos de la empresa. Desde la misma página, podríamos editar (12) o eliminar (13).

12 – Al hacer clic sobre editar, se debe abrir un formulario, donde podemos editar el contenido de la implantación seleccionada.

13 – Al hacer clic sobre eliminar, se debe abrir un mensaje de confirmación. En caso afirmativo, la implantación quedará fuera de la lista de implantaciones (pero no de la base de datos). En caso contrario, no ocurrirá nada.

14 – Desde la ventana principal, podemos cerrar la sesión correspondiente y volver a la pantalla de inicio.

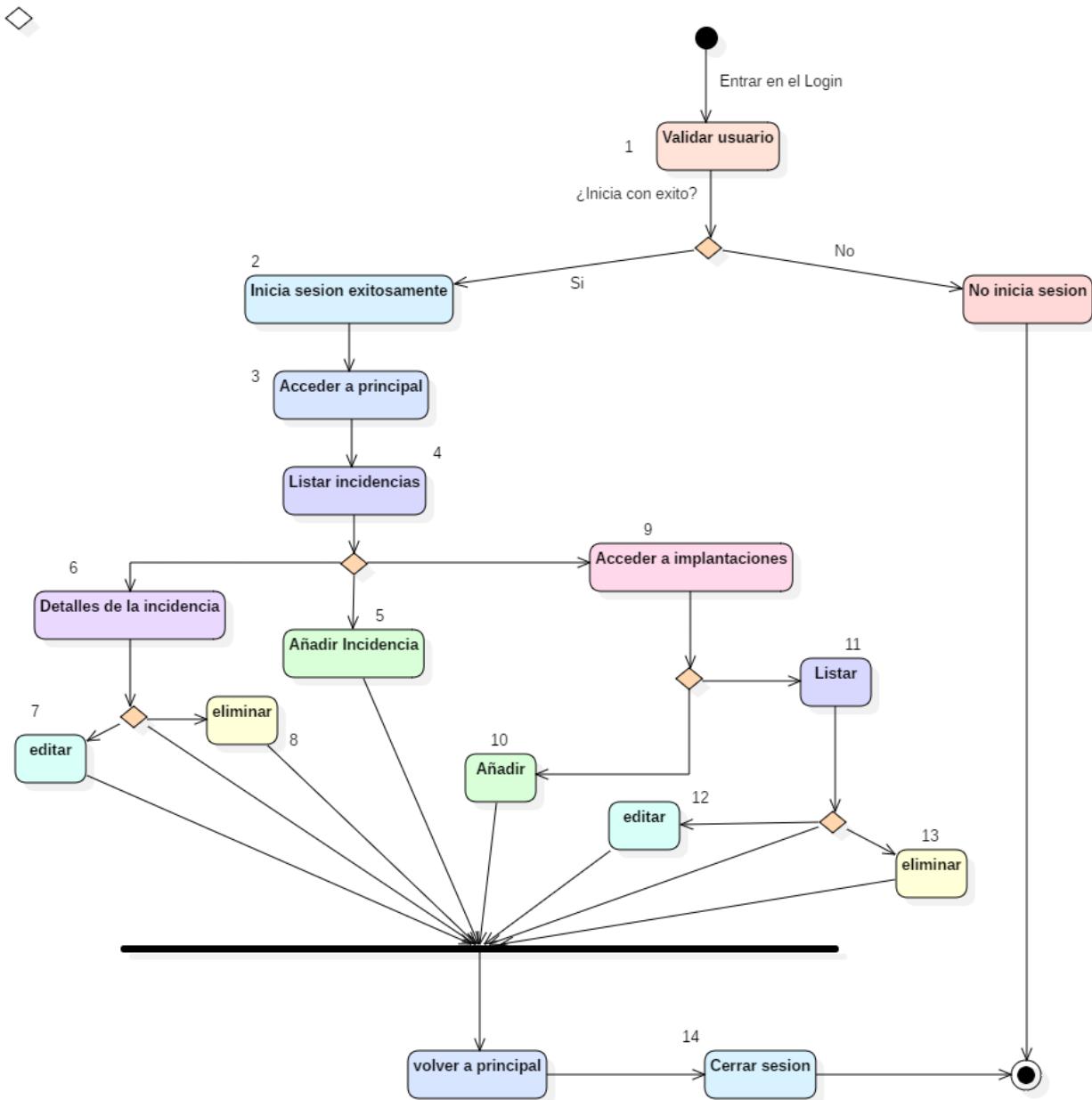


Ilustración 6: Diagrama de actividad

3.6. Conclusión del análisis

Tras realizar un profundo análisis de la situación del desarrollo, se ha llegado a la conclusión de que la aplicación es viable respecto a las herramientas y Frameworks (lenguajes de programación) a emplear, respecto al tiempo de desarrollo requerido y los conocimientos innatos. Por ello, vamos a emplear la lógica expuesta en este apartado, atendiendo a que funcionalidades y limitaciones tiene cada plataforma (Angular, Lumen), así como los criterios de evaluación de la empresa y los criterios de corrección del proyecto aquí expuesto.

4. Diseño

4.1 Introducción

4.1.1. Selección del entorno de desarrollo

Después de sopesar las posibilidades de desarrollo de la aplicación. En la empresa se ha decidido, en conjunto con el equipo de programación y el desarrollador de la aplicación, en este caso nosotros, que tecnologías de desarrollo vamos a emplear para construir esta aplicación. En orden ascendente de menor a mayor nivel de abstracción, iremos indicando cada una de estas y explicando la casuística y la razón de ello, así como posibles alternativas (en caso de que las hubiera):

1. **La Base de Datos:** La BBDD a utilizar será MySQL. Principalmente, porque la empresa trabaja con este tipo de bases de datos, y por tanto resulta necesario que la base de datos del proyecto se encuentre estructurada dentro de este tipo. Utilizaremos una tabla de la base de datos para cada elemento notable de nuestra aplicación. Véase los siguientes ejemplos de tablas: usuario, implantación, incidencia, proyecto, contacto. En dichas tablas se representarán las propiedades de cada uno de estos elementos, como pueden ser: nombre, descripción, estatus, fecha, etc. Además, será necesario que organicemos un sistema de relaciones entre tablas, mediante claves foráneas, que permita la comunicación entre ellas, y por tanto, faciliten las consultas SQL a dicha base de datos. Por ejemplo: id_project en las tablas 'proyecto' y 'implantación'.
2. **El Back-End:** Se realizará un API Restful con protocolo JSON en PHP 7.* , haciendo uso del Microframework Lumen 5.*. Dicho framework se trata de un micro-framework para PHP que comparte muchos de los componentes de Laravel. A diferencia de Laravel PHP, hay archivos y componentes que no vienen por defecto en Lumen (como por ejemplo la integración con Bootstrap y los módulos preestablecidos de autenticación y registro), pero sí podemos usar ORM Eloquent y otros componentes importantes para nuestro desarrollo. Usaremos esta capa de abstracción para las operaciones necesarias y la comunicación entre la base de datos y la aplicación. El ORM (Eloquent) trabajará con los modelos definidos. El controlador recibirá y responderá a las peticiones de los recursos solicitadas por el front-end. La lógica de negocio se encargará de realizar las operaciones necesarias para obtener los recursos del controlador, así como del uso del ORM. La razón de utilizar este framework para la capa de abstracción entre la base de datos y la aplicación es porque al tratarse de una aplicación no demasiado grande, podemos prescindir de según qué tipos de funcionalidades que nos ofrecen frameworks como Laravel. Al mismo tiempo, contamos con las ventajas más representativas de éste, otorgándonos la posibilidad de crear distintas clases dentro del entorno, que desempeñan una labor muy específica. Dentro del Microframework Lumen, hallamos los siguientes tipos de clases:

A. Migraciones (Migrations): Lo primero que debemos programar son las migraciones. Dentro de Lumen, encontramos una serie de ficheros de configuración. El más representativo es el fichero. env, que es aquel en el que vamos a indicar la conexión con la base de datos: el nombre de ésta, el nombre del usuario mysql y su contraseña, su dirección IP y el puerto por donde escucha. Pues bien. Una vez tengamos configurado este fichero, el Back-End conocerá el origen y destino de la información que va a almacenar. De modo que, a la hora de ejecutar el comando que pone en marcha una migración, Lumen sabe, en función de que servidor y base de datos hayamos especificado, en qué base de datos va a volcar la información inicial. ¿Pero, qué es una migración? Una migración es un fichero que almacena, en lenguaje php, las propiedades de una tabla mysql en forma de clase. Dicha clase contiene como propiedades de la clase las columnas de dicha tabla, con el tipo de valor que van a representar, el tamaño máximo, si puede o no ser nula, si se trata de una clave primaria o foránea, etc. Además, dicha clase php además de las propiedades y el método que define su estructura, puede tener, en caso de que lo necesite, un método que defina las relaciones entre tablas, las cuales deben tratarse de tablas existentes, lo que nos lleva a la siguiente cuestión. Para que las relaciones puedan ser interpretadas, anteriormente, las tablas con las que nuestra tabla creada a partir de ese fichero de migración debe haber sido creadas previamente. Esto quiere decir que no podemos ejecutar un fichero de migración, el cual contenga una relación con otra tabla, 'que pensamos crear, pero que todavía no hemos hecho'. A la hora de volcar la información de la clase, 'Artisan' creará la tabla en la base de datos, pero no podrá efectuar las relaciones. Para solucionar esto, lo primero que debemos hacer es, o bien crear las tablas en ficheros separados y, posteriormente, ejecutar un fichero de migración con las relaciones, o bien directamente crear un único fichero de migraciones que contenga, de forma lineal y descendente, los pasos a seguir por 'Artisan' y que tablas o métodos debe atender primero, de forma que el código más cercano al inicio se interprete antes que las últimas líneas de código, donde debemos insertar el método que nos crea las relaciones entre tablas. Hemos hablado de 'Artisan', pero no hemos definido de qué se trata. Llamamos Artisan a la interfaz de línea de comandos (CLI, de *Command-line interface*), donde los usuarios (en este caso los desarrolladores) dan instrucciones en forma de línea de texto simple o línea de comando. Artisan está basado en el componente Console de Symfony y nos ofrece un conjunto de comandos que nos pueden ayudar a realizar diferentes tareas durante el desarrollo e incluso cuando la aplicación se encuentra en producción. Con Artisan, la cual se encuentra integrada en Lumen y Laravel, podemos ahorrar tiempo, ya que sus comandos atienden a órdenes predefinidas, como es el caso de "php migrate", que nos permite hacer la migración a la base de datos en la que estemos trabajando (recordemos que esta ha sido indicada en el fichero. env). Una vez que la migración se haya producido de forma exitosa, desde un cliente de MySQL, como MySQL Workbench, podemos realizar una conexión y consultar la base de datos para comprobar que las tablas y

relaciones migradas se encuentra en perfecto estado de creación. La principal ventaja de las migraciones es que nos permite crear las tablas que luego va a utilizar la aplicación sin necesidad de un cliente MySQL y sin necesidad de manejar el lenguaje SQL empleado en los gestores de bases de datos, ya que todo el proceso de creación se realiza en Php.

- B. Cosechadores (Seeders): El segundo paso a la hora de crear las migraciones, es "llenar" las tablas creadas, o algunas de ellas, con datos genéricos, necesarios para hacer consultas de prueba con la base de datos y comprobar la consistencia de la información. Para ello, contamos con los 'seeders', ficheros php que realizan la función de introducir datos en una tabla definida en la clase. Cabe destacar que, para emplear un 'seeder', previamente debe existir la tabla a la que vamos a atacar, puesto que, en caso de no existir, la operación concluirá con un error. Para poner en marcha este mecanismo de Lumen y también Laravel, utilizamos otro comando php orquestado por la herramienta 'Artisan'. Para comprobar que la operación se ha realizado correctamente, utilizaremos un cliente MySQL y haremos una consulta a varias tablas inyectadas con información, para comprobar que, efectivamente, se han introducido los valores especificados en dicha tabla.
- C. Modelos (Models): Este concepto es algo más abstracto, puesto que no desarrolla ninguna acción que se pueda observar a simple vista. Un modelo es una clase php que contiene las propiedades de una tabla, de modo similar a como lo hacía el fichero de migraciones. El objetivo de las clases que extienden de 'Model' (modelo) es facilitar las operaciones del Back-End con una tabla concreta, al hacer que las operaciones de inserción, creación, modificación, borrado, listado, etc. Sean tratadas como métodos php y no como consultas como tal. Para entender esto, supongamos que deseamos actualizar todos los campos de una tabla. En SQL, nosotros debemos atender a la sintaxis tradicional, que tiene un aspecto tal que así: "SELECT name from clientes WHERE id_cliente = 4". Este ejemplo es una consulta muy sencilla, pero se puede complicar la situación con otros ejemplos. La cuestión es solventar una consulta que puede resultar densa para el programador, tratando la tabla como si fuese una clase php y estuviésemos trabajando con variables. De modo que, en lugar de un select, estaríamos trabajando con un array en el que, para listar un contenido concreto, utilizamos un id y una sintaxis tal que así: "return \$resultado[\$id]", siendo más sencillo trabajar con tablas y relaciones con php gracias al framework. Esto se entiende mejor en la práctica que en la teoría donde, a la hora de hacer un método que inserte valores en la tabla, resulta muchísimo más fácil trabajar con modelos que con consultas más tradicionales.
- D. Controladores (Controllers): Al igual que ocurre con los modelos, los controladores son un tipo de clase en php que tratan de agilizar y facilitar el acceso a la información de la base de datos. Es en el controlador donde definimos los métodos CRUD y en donde vamos a desarrollar toda la lógica que maneja el Back-End. Estos métodos son los que van a interactuar directamente con el Front-End, en forma de peticiones a URL para consumir un recurso único. No obstante, no es en

este fichero donde definimos la URL, sino que este se encuentra en otro fichero del que hablaremos más adelante.

- E. Middleware: Cuando hablamos de Middleware, hablamos de dos conceptos. Por una parte, es un tipo de fichero que encontramos en nuestro framework. Por otra parte, es la labor que desempeñan dichos ficheros en cuestión. Un Middleware no es más que un mecanismo empleado para filtrar las peticiones HTTP. Permite agregar capas adicionales a la lógica de la aplicación. El ejemplo más común es el Middleware de autenticación, el cual se ocupa de restringir el acceso al Back-End un sitio en donde es necesario iniciar sesión y tener ciertos privilegios para poder ver el contenido de determinadas páginas. Digamos, en resumen, que es una capa más de abstracción entre el controlador y el Front-End, puesto que antes de acceder al controlador, la petición debe viajar por el Middleware y pasar el control adecuadamente.
- F. Routes: En nuestro framework, contamos con un fichero llamado web.php que se encarga de organizar las rutas (URL) de los métodos del Back-End. Por una parte, contamos con la ruta a la que vamos a atender. Por otra, definimos el nombre del método y la dirección del archivo en la que se encuentra. La sintaxis empleada en este fichero es la siguiente:

```
$router->GET('check', 'UserController@check');
```

Donde, '\$router' es la variable de la clase que va a recibir la ruta, 'GET' es la petición que se va a realizar, 'check' es el sufijo de la URL (ya veremos más adelante) y 'UserController@check' hace referencia al método check(), que se encarga de comprobar que la conexión con el servidor se produce de forma satisfactoria; situada en el fichero 'UserController.php'. Todo esto, en conjunto, definen el enlace entre un método y una dirección URL, que es la dirección a utilizar en caso de poner en marcha una determinada acción, como comprobar la conexión en este caso, o insertar un valor en una tabla especificada.

- G. Public: Podríamos definir este espacio como la frontera entre el Back-End y el nivel más bajo del Front-End. Contamos con el fichero index.php, donde se van a construir las rutas para las peticiones y en la que vamos a definir los valores de las variables '\$app' y '\$router', esta última citada anteriormente. De modo que, cuando hagamos una petición, por ejemplo, la que vimos en el punto anterior (comprobar la conexión con el servidor), la dirección URL de ésta partirá de 'public', y se hará uso de las variables '\$app' y '\$router' para completarlas. Veamos el ejemplo anterior. Para consultar si el servidor está conectado, desde el front-end, escribimos:

```
http://dirección_servidor:puerto/ruta_del_backend/..../public/check
```

En consultas donde es necesario que la petición viaje por el 'Middleware', ya que éste controla si la petición ha sido realizada tras estar validados y autenticados, la capa de abstracción de las consultas sería mayor. Por ejemplo, la petición para insertar incidencias, donde previamente debemos estar validados como usuarios de la aplicación para poder añadir contenido a la base de datos:

http://dirección_servidor:puerto/ruta_del_backend/..../public/auth/issue

Una alternativa a esta solución es haber hecho el Back-End en .NET, puesto que fue otra de las opciones que se dieron en la empresa. .NET cuenta también con muchas opciones de Back-End y es tan válida como la empleada finalmente en el proyecto. Al final, nos decantamos por una solución basada en Php, ya que consideré que se trata de un entorno más acertado para nuestra aplicación, en temas de seguridad, rendimiento y comunidad de desarrollo. En el siguiente enlace se facilita una comparativa, donde salen a relucir las principales ventajas de Php respecto a .NET:

https://guiadev.com/php-vs-asp-net/#ComparativanbspPHP_vs_ASPNET

3. **El Front-End:** La realizará una SPA en Angular 7, ya que es la versión más reciente de la tecnología con la que se trabaja en la empresa. Cabe destacar que es una SPA y que es Angular, brevemente.

Citamos textualmente: "Un single-page application (SPA), o aplicación de página única, es una aplicación web o es un sitio web que cabe en una sola página con el propósito de dar una experiencia más fluida a los usuarios, como una aplicación de escritorio. En un SPA, todos los códigos de HTML, JavaScript, y CSS se carga de una vez, o los recursos necesarios se cargan dinámicamente como lo requiera la página y se van agregando, normalmente como respuesta de las acciones del usuario". Dicho de otro modo. Una SPA permite tener una sola página en la que el resto de direcciones son lógicas, y se van cargando en el visor web a través de componentes de esa misma aplicación. No necesitamos tener un fichero .html por cada página que vemos por pantalla, sino que la propia aplicación contempla como deben cargarse los distintos resultados en función de cómo definamos la navegación entre ventanas. Entre sus principales ventajas, están:

1. Eliminación del refresco de la página. Las páginas son dinámicas y su contenido interactúan con el usuario en tiempo real.
2. Carga de página más rápida. Al encontrarse todo en una sola página, se carga el contenido una sola vez.
3. Mejores interacciones y más fluidas.
4. Menor transferencia de datos.
5. Desarrollo más rápido y reutilización de elementos.

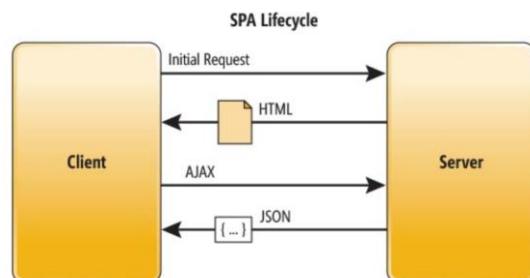


Ilustración 7: Ciclo de vida de una SPA

Acerca de Angular, se trata de un Frameworks para el diseño de páginas SPA (Front-End), mantenido por Google, y cuya primera versión actual de dicho Frameworks fue la versión 2.0, en la que se separó del ya conocido AngularJS para pasar a ser simplemente Angular, a secas. Aunque es conocido por ser un Frameworks Javascript, en la actualidad utiliza Typescript como lenguaje de programación, **un súper-conjunto de JavaScript/ECMAScript** que facilita mucho el desarrollo. Typescript hereda mucha de la sintaxis de Javascript, solo que organiza el contenido de una forma más ordenada y exigente. Esto, lejos de un inconveniente, resulta una ventaja, puesto que obliga al desarrollador a seguir unas pautas de diseño y estructura del programa, haciendo cumplir

unos estándares que aumentan la coherencia del código, haciéndolo más entendible tanto para el programador que lo emplea como para el resto del equipo. Volviendo a Angular, a las ventajas que ya hemos visto que posee una SPA, añadimos las siguientes:

- A. **Uso de componentes que trabajan con HTML, CSS y JS.** Esto permite compatibilidad con los navegadores actuales y nos permiten una multitud de herramientas muy conocidas que, además, cuentan con una inmensa comunidad detrás.
- B. Debido a lo anterior, contamos con una **amplia variedad de herramientas de edición de código avanzado**, IDEs, etc. Como pueden ser WebStorm, Sublime Text, Visual Studio o DreamViewer, entre otros. Esto nos lleva al último punto a tratar en este apartado, que veremos más adelante: El entorno de desarrollo elegido.

Una vez tratadas las definiciones de SPA y Angular, podemos proseguir con los objetivos de nuestro Front-End: Se desarrollarán componentes nativos para homogeneizar y agilizar la creación de la interfaz de usuario. A saber:

- Rejilla reactiva con filtros y ordenables.
- Formularios basados en formularios reactivos, con validaciones y binding al modelo.
- Modales.
- Modales y secciones con rejilla y CRUD
- Se realizará una capa de abstracción para realizar las peticiones a la API, de manera que los servicios de cada página hereden de una base común que implemente el CRUD. Esa base a su vez heredará de una base inicial HTTP para el manejo de peticiones con la autenticación.

Para hacer peticiones HTTP en Angular que apunten a nuestro Back-End, debemos crear un servicio inyectable. Un **servicio** es una clase typescript que crearemos en Angular. Su propósito es contener lógica de negocio, clases para acceso a datos o utilidades de infraestructura. Estas clases son perfectamente instanciables desde cualquier otro fichero que las importe. Pero Angular nos sugiere y facilita que usemos su sistema de inyección de dependencias. En nuestro caso, nos comunicaremos mediante un servicio, utilizando el módulo **HttpClient**. Dicho módulo contiene los verbos GET, POST, PUT, PATCH, DELETE, etc. Por lo que podremos consumir los servicios del Back-End si conocemos la URL que accede al método que ejecuta una operación concreta de Inserción, adición, edición o eliminación, entre otras. Más adelante veremos de forma más exhaustiva como se produce la comunicación entre el Front y el Back.

Conocidas las tecnologías abarcadas en el proyecto, así como la justificación de su uso, toca hablar de **los entornos de desarrollo** que empleamos para cada una de las partes diferenciadas de la aplicación:

1. **MySQL Workbench para la gestión de la base de datos:** Se trata de una herramienta visual de diseño de bases de datos que integra desarrollo de software, administración de bases de datos, diseño de bases de datos, gestión y mantenimiento para el sistema de base de datos MySQL. Usaremos su versión actual, la versión 8.0, con el propósito de consultar las tablas creadas por el Back-End, y comprobar que el programa responde correctamente a las peticiones CRUD. Así mismo, corregiremos cualquier dato de forma inmediata que pueda ocasionar un fallo en la aplicación durante el desarrollo de la aplicación. Por ejemplo. Durante nuestro desarrollo, vimos conveniente cambiar los valores del campo 'Role' de los usuarios. Pasaron de denominarse como 'A', 'TI', etc. A 'admin', 'tec_imp', etc. Esta labor se llevó a cabo desde el gestor de bases de datos, MySQL Workbench, por su inmediatez y simplicidad para corregir o modificar valores creados.
2. **De PHPStorm a VSCode para el Diseño del Back-End:** A la hora de desarrollar nuestro proyecto, el primer paso sobre la que comenzar la aplicación fue el desarrollo del Back-End, ya que se trata de los cimientos de la aplicación y sobre donde descansa las peticiones del Front-End. Como ya hemos comentado. Una vez conocida la base de datos sobre la que íbamos a trabajar, configuramos el fichero. env, las migraciones, los 'seeders', los modelos, controladores y rutas de los métodos (aunque mucho de estos se desarrollaron posteriormente, durante la fase final de desarrollo de la aplicación). Todo eso fue desarrollado con JetBrains PHPStorm, un IDE (entorno de desarrollo integrado) multiplataforma para PHP, construido sobre la plataforma IntelliJ IDEA de JetBrains. PHPStorm proporciona un editor para PHP, HTML y JavaScript con análisis de código, prevención de errores y refactorizaciones automatizadas para código PHP y JavaScript. Es compatible con PHP 5.3, 5.4, 5.5, 5.6, 7.0, 7.1 y 7.2, e incluye un editor de SQL completo con resultados de consulta editables. Durante la primera fase de desarrollo, estuvimos utilizando este IDE en su **versión de evaluación**. No obstante, una vez que el desarrollo básico y funcional del Back-End estuvo terminado y la licencia del software expiró, optamos por utilizar **Visual Studio Code** para desarrollar el Back-End en las posteriores fases de la aplicación, ya que es de uso gratuito, es decir, sin licencias de pago (Freeware) y nos permitía trabajar, mediante una extensión, con el lenguaje PHP y SQL; y ya nos encontrábamos trabajando con él, por lo que conocíamos el funcionamiento del entorno.
3. **Visual Studio Code para el desarrollo del Front-End (Y algo del Back-End):** Para el desarrollo de la aplicación Angular, entre las distintas alternativas con la que contamos en el panorama actual, nos decantamos por Visual Studio Code, por el ya mencionado hecho de ser un entorno con licencia Freeware, entre otras ventajas. A pesar de tener muchísimas funcionalidades heredadas de los entornos de desarrollo integrado, gracias a sus extensiones, se encuentra clasificado como editor de código fuente, y no como IDE, como si lo es su versión vitamina, **Visual Studio**. No obstante, visual studio code nos permite corregir errores sintácticos en el código, importar módulos de forma automatizada, sincronizar con GIT, e incluso incorpora Powershell, por lo que podremos levantar el servidor de pruebas de Angular por CLI mediante comandos 'npm'.
4. **POSTMAN para el testeo de los verbos HTTP:** Como mención especial, debemos comentar el uso de la herramienta Postman para probar las peticiones de nuestro servicio API Restful, haciendo uso de las URL (rutas) del Back-End para

atacar a sus métodos del CRUD. Postman originalmente era una extensión para navegadores web, pero actualmente posee aplicaciones de escritorio. Esta herramienta, ademas de permitirnos probar los métodos del Back-End, a través de las respuestas de las peticiones, también nos facilita almacenar los métodos utilizados en colecciones. De modo que podemos tener a la mano todos los métodos funcionales de nuestro servicio REST, para que, en caso de alterar o insertar una nueva funcionalidad, podamos testear de forma rápida si el cambio ha producido o no alguna consecuencia en nuestro servicio.

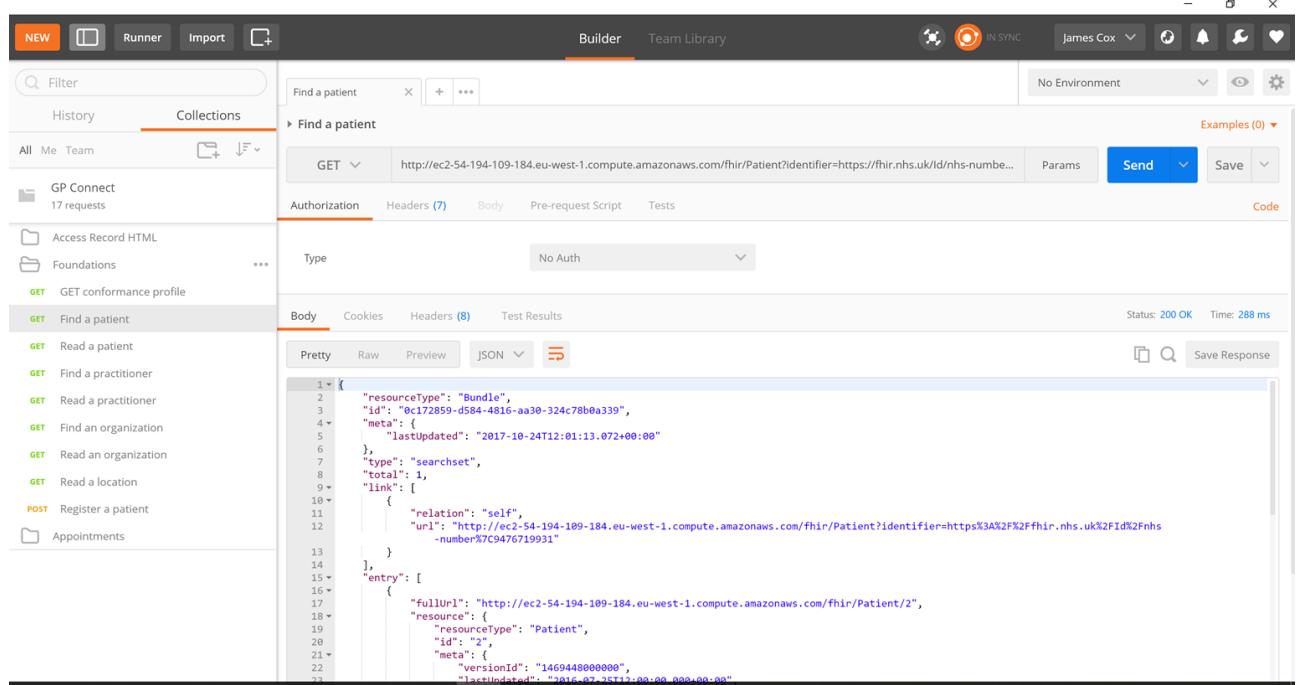


Ilustración 8: Uso de la herramienta Postman

5. **GitHub Desktop:** Por último, es interesante comentar las herramientas que nos ofrece GitHub. Como bien sabemos, GitHub es un repositorio remoto donde los desarrolladores podemos subir nuestro trabajo, de forma pública o privada, principalmente para compartir nuestro trabajo con la comunidad de desarrollo. Pero, además, una de las ventajas de tener nuestro repositorio en GitHub es la posibilidad de utilizar las **herramientas de Git**, como pueden ser 'push', 'pull', 'commit' o 'merge', entre otras. Gracias a subir nuestro proyecto a GitHub, hemos podido trabajar con el proyecto desde varios equipos distintos, haciendo posible que la aplicación pudiese ser desarrollada desde la empresa y desde casa, sin comprometer el código ni la información, y haciendo uso de los comandos ya mencionados. De esta forma, el repositorio remoto se encuentra siempre actualizado a la versión más reciente de la aplicación, y para trabajar con ella no hay más que bajar el repositorio al directorio de trabajo de ese momento para, posteriormente, subir los cambios producidos al directorio remoto. Para ayudarnos con esta tarea, además de la extensión GIT de Visual Studio Code, que nos permite hacer 'commit' de los cambios de una manera muy sencilla (repositorio local), hemos utilizado **GitHub Desktop**. Se trata de una **aplicación de escritorio**, como su nombre nos indica, y facilita enormemente la interacción entre el directorio de trabajo y el repositorio remoto. Desde dicha aplicación, podemos iniciar sesión con nuestra cuenta de GitHub y añadir repositorios existentes, crear nuevos o crear ramas de repositorios existentes, entre otras funciones. Una vez que tenemos Visual Studio Code y GitHub Desktop conectados con el repositorio remoto, será tremadamente sencillo trabajar con los cambios, puesto que solo necesitaremos encontrarnos en el mismo repositorio y rama para que el proceso sea a tiempo real, pudiendo, desde la aplicación, hacer 'commit' y 'push' cada vez que hacemos un cambio, por mínimo que sea. Para traernos al directorio de trabajo la versión más actual del repositorio remoto, no utilizaremos GitHub Desktop, sino que lo haremos directamente desde la consola Powershell incrustada directamente en VSCode con un 'pull'. Además de poder trabajar en distintos lugares con distintos equipos de forma cómoda y así evitar la tediosa operación de comprimir el proyecto para posteriormente descomprimirlo en otro equipo y desplegarlo, se trata de una copia de seguridad muy efectiva; por lo que es algo muy a tener en cuenta.

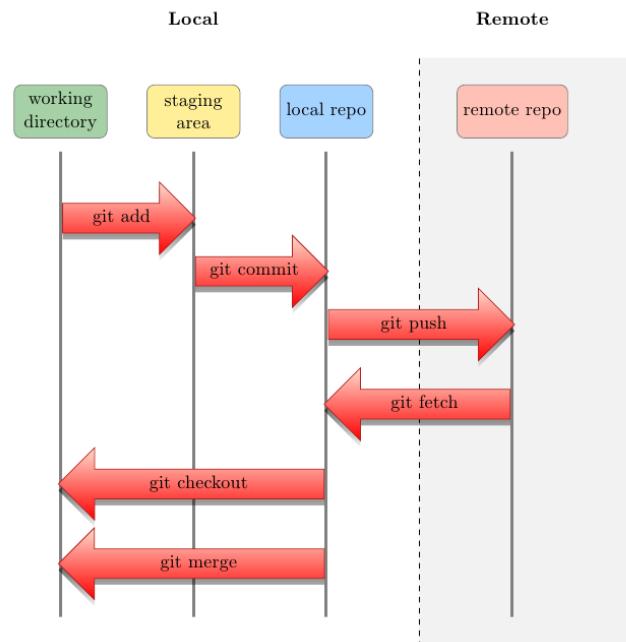


Ilustración 9: uso de Git

4.2 Configuración de la plataforma

A la hora de configurar nuestro entorno de trabajo, es importante que primero contemos con las herramientas de Angular, necesarias para poder llevar a cabo el desarrollo del proyecto. Lo primero es instalar Node.js y el manejador de paquetes. npm. Posteriormente, instalamos Angular CLI, que es la herramienta de línea de comandos estándar para **crear, depurar y publicar aplicaciones Angular**. Una vez hecho este proceso, crearemos desde la ventana de comandos una nueva aplicación Angular en un directorio elegido. Es importante que nos acostumbremos a manejar la consola de comandos, ya que será necesario para crear componentes, servicios, directivas, tuberías, etc. Comenzaremos trabajando en nuestra aplicación desde cero, primero desarrollando la estructura interna y la navegación entre componentes, y posteriormente le daremos sentido a dichos componentes creados.

Para instalar Node.js y npm, lo recomendable es descargarlo desde su página oficial. Una vez descargado, instalaremos el paquete como cualquier software de terceros, mediante su asistente de instalación. Una vez instalados, npm se comportará como una variable de entorno, por lo que será posible utilizarla desde cualquier ruta del sistema.

Para instalar Angular CLI, hacemos uso del comando:

```
npm install -g @angular/cli
```

De este modo, ya tenemos preparado nuestro entorno para crear un nuevo proyecto Angular.

Lo siguiente es instalar los programas necesarios para trabajar con el entorno de trabajo. Anteriormente, hablamos de varias herramientas como Visual Studio Code y PHPStorm. Vamos a omitir este último y solo vamos a hablar de Visual Studio Code, ya que, al tratarse de un programa con una licencia comercial y haber sido utilizado durante la primera fase del desarrollo, no es estrictamente necesario para llevar a cabo del desarrollo total de la aplicación, puesto que Visual Studio Code es perfectamente usable para el mismo cometido. Por tanto, contamos con 4 herramientas esenciales para trabajar con el proyecto:

- 1. Visual Studio Code:** Como ya hemos comentado, no se trata de un entorno de desarrollo integrado, sino que está catalogado como editor de texto avanzado. No obstante, y como ya hemos dicho, resulta que este editor, mediante sus extensiones, cuenta con una amplia variedad de funcionalidades que lo colocan al mismo nivel de un IDE. En nuestro caso, no vamos a echar en falta ninguna de las ventajas que nos ofrece un entorno de desarrollo con respecto a él. Para instalarlo en nuestro espacio de trabajo, no tenemos más que dirigirnos a la web oficial <https://aka.ms/win32-x64-user-stable> (en caso de contar con un equipo de 64 bits, lo cual es recomendable para el desarrollo del equipo) y descargar el paquete. Una vez descargado, lo instalaremos en nuestro equipo y ya podremos utilizar este editor. Para disfrutar de una experiencia más interesante, deberíamos instalar "PHP Intelephense", una extensión que permite utilizar la inteligencia artificial de VSCode con PHP, además de la extensión "SQL Server (mssql)" que, de la misma forma, nos permite utilizar el lenguaje SQL dentro de la aplicación. Una vez creado el proyecto en Angular, el cual lo creamos desde la consola de comandos (se puede emplear la propia consola integrada de VSCode) mediante el comando CLI:

```
'ng new nombre_aplicacion'
```

Nos situamos en la ubicación del proyecto creado (cuya dirección debemos tener claro cuál queremos que sea, antes de crear el proyecto), utilizando el explorador de windows desde el Visual Studio Code desde la pestaña "File" con la opción "Open Folder". Visual Studio Code abrirá en el lado izquierdo del programa un directorio ramificado con todos los ficheros del proyecto, desde el cual podremos navegar por los distintos ficheros para su edición. Algo a tener en cuenta es que VSCode posee una consola de comandos integrada. Por tanto, al abrir esta ubicación, también estaremos situando la consola en la dirección del proyecto. Esto resulta muy útil, ya que no necesitaremos abrir una instancia de la consola en Windows, sino que desde VSCode podemos abrir varias consolas, una para cada cometido. En nuestro caso, dejaremos una consola abierta para la instalación de componentes, servicios y directivas, entre otras cosas, y abriremos una nueva consola (Powershell) cuando queramos levantar un servidor virtual para visualizar nuestra aplicación en un navegador. Esto se consigue con el comando:

```
ng serve <nombre_proyecto> --port xxxx
```

Donde 'xxxx' es el puerto donde queremos que escuche (por defecto, 4200) O bien:

```
npm start [-- <args>]
```

Algo más avanzado que el anterior, y que en la mayoría de casos también engloba el comando 'ng serve'. Una vez que tengamos el proyecto desarrollado y queramos compilarlo para ser instalado en un servidor, ejecutaremos el siguiente comando:

```
ng build <project>
```

Los comandos para crear componentes, módulos, servicios o directivas siguen el mismo esquema:

```
ng create component <nombre_componente>
```

Por último, Git se encuentra integrado en VSCode, de modo que, si iniciamos un proyecto el cual está conectado a un repositorio Git, automáticamente, VSCode se conecta con Git para poder ejecutar varios comandos, como 'commit'.

2. Chrome Dev: Utilizaremos como navegador la versión para desarrolladores de Chrome. Necesitaremos utilizar un navegador como Chrome o Firefox, ya que ambos cuentan con una herramienta de inspección, la cual se activa, normalmente, pulsando F12 en el teclado. Mediante esta herramienta del navegador, además de poder visualizar a tiempo real como avanza el desarrollo de nuestra aplicación, poseeremos varias ventajas:

- I. Desde la pestaña "**elementos**", podemos visualizar las etiquetas que definen la estructura y diseño de la página en la que nos encontramos. No solo visualizaremos el Html de la página, sino que, además, podremos ver el estilo de cada uno de los elementos y clases. Mientras la aplicación se encuentre levantada, nosotros podremos hacer cambios en las etiquetas y los estilos, de modo que podremos visualizar a tiempo real el efecto de cambiar ciertas propiedades, con el afán de mejorar el diseño y la usabilidad, con la tranquilidad que da el saber que al refrescar la página, esos cambios no se guardarán y, por tanto, un mal diseño por nuestra parte no repercutirá en lo más mínimo en caso de hacer un cambio poco conveniente (aunque, nos interesa apuntar los cambios po-

sitivos que hagamos en pos de un mejor diseño, ya que al refrescar también se perderán). Esto, como ya hemos comentado, resulta útil a la hora de reorganizar la disposición de los elementos de la pantalla, en el caso de que queramos re-dimensionar la barra de navegación, el tamaño de un 'input' del formulario, el margen de una tabla, etc.

- II. Desde la pestaña "**consola**", podremos consultar los 'logs' que nos arroja la sesión activa. No solo podremos ver los errores de la página (en caso de que una mala praxis o el fallo de un servicio se hagan presentes), sino que, además, desde el Visual Studio Code (o el editor/IDE con el que estamos trabajando), podremos crear distintas funciones en typescript que arrojen resultados a dicha consola. Por ejemplo:

```
console.log("El usuario ya esta logueado");
```

Hará que, colocado en un punto concreto de nuestro código, por ejemplo, en el método de validación de usuario en el 'login', mostrará por consola el mensaje en su interior. Esto resulta útil para identificar el momento en el que se ejecuta un método o bien, la parte del programa que se ejecuta en función de una acción que toma el usuario. En caso de no encontrarse ya logueado, el mensaje no se mostraría. Tampoco se mostraría si no ingresamos en el login. Si en otra parte del código se encuentra otro mensaje, éste se mostrará también en la consola, antes o después del otro, dependiendo de en qué parte del código (que parte del código se ejecuta antes). Por ejemplo, un log en el constructor de un componente se mostrará antes que un log escrito en el método 'onInit()'. Además de cadenas, dentro de un 'console.log' podemos incluir variables de tipo entero, booleano, métodos, elementos html (por id), respuestas de peticiones (array[]), entre otras cosas.

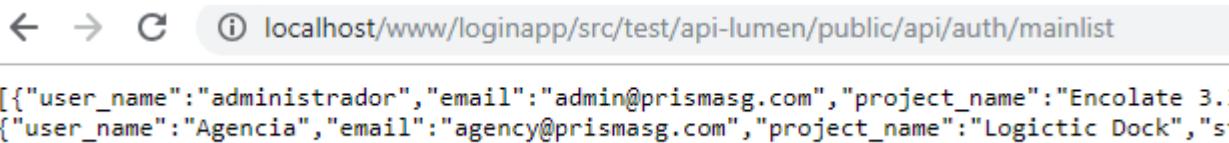
- III. En la pestaña "**fuentes**", encontraremos los ficheros .css y .js que utiliza nuestro sitio, así como las imágenes que hayamos importado en nuestro proyecto.
- IV. Resulta muy útil el contenido de la pestaña "**red**", ya que en ella encontraremos las peticiones que efectúa y resuelve nuestra aplicación, ordenadas por nombre, estatus, tipo, etc. Dentro de ellas, podremos ver sus detalles, como las cabeceras de la petición, la respuesta o respuestas a la petición, etc.
- V. En la pestaña "**aplicación**" se encuentra el lugar donde el navegador almacena los objetos, bien sean tipo 'Cookie' o bien sea tipo 'LocalStorage'. En nuestro caso, utilizaremos el 'LocalStorage' para guardar la sesión validada del usuario, por lo que será útil que tengamos localizada esta pestaña, sobre todo durante las primeras fases del diseño, para comprobar que estamos haciendo el procedimiento correctamente. En nuestro caso, además, lo que guardaremos en 'LocalStorage' será un objeto cuyo valor será un 'Token', ya que utilizaremos la metodología JWT (JSON Web Token) para la validación de usuario.

Aparte de las herramientas para desarrolladores, nos valdremos de algunas extensiones de Chrome que nos ayudarán durante el desarrollo del proyecto. Estas extensiones son las siguientes:

- I. **Allow-Control-Allow-Origin:** El Intercambio de Recursos de Origen Cruzado (CORS) es un mecanismo que utiliza cabeceras HTTP adicionales para permitir que un user agent obtenga permiso para acceder a recursos seleccionados desde un servidor, en un origen distinto (dominio) al

que pertenece. Un agente crea una petición HTTP de origen cruzado cuando solicita un recurso desde un dominio distinto, un protocolo o un puerto diferente al del documento que lo generó. Con esta extensión, podremos habilitar o inhabilitar las cabeceras generadas por el navegador, de forma que nadie ajeno a nuestro dominio pueda acceder a ciertas peticiones. Esto puede ser una hoja de doble filo, dependiendo de qué tipo de aplicaciones queramos utilizar, ya que aquellas de origen público, como por ejemplo youtube, no permitirán reproducir contenido con esta herramienta habilitada, puesto que intentamos acceder desde un dominio catalogado para el servidor como '*'; ajeno al dominio conocido. A cambio, nuestra aplicación será más segura, ya que ningún dominio ajeno al nuestro podrá acceder a los recursos de la aplicación, como aquellos frutos de las invocaciones XMLHttpRequest a las API.

- VI. **JSON Viewer:** Esta sencilla extensión nos permite visualizar en el navegador cualquier respuesta JSON en su formato estándar. Cualquier navegador moderno puede visualizar el contenido de un JSON. No obstante, la forma en la que se presenta la información resulta la siguiente:



A screenshot of a web browser window. The address bar shows the URL: localhost/www/loginapp/src/test/api-lumen/public/api/auth/mainlist. The main content area displays a JSON array with two elements:

```
[{"user_name": "administrador", "email": "admin@prismasg.com", "project_name": "Encolate 3."}, {"user_name": "Agencia", "email": "agency@prismasg.com", "project_name": "Logictic Dock", "s"}]
```

Ilustración 10: Petición en el navegador

Si queremos que la respuesta a la petición, o cualquier respuesta en JSON se muestre en un formato más estilizado, debemos instalar y poner en funcionamiento esta extensión. Una vez hecho, la misma petición se mostrará en el navegador de la siguiente forma:

```
[{"user_name": "administrador", "email": "admin@prismasg.com", "project_name": "Encolate 3.3", "status": 1}, {"user_name": "Responsable CL", "email": "admin@prismasg.com", "project_name": "Encolate 3.3", "status": 1}, {"user_name": "Agencia", "email": "agency@prismasg.com", "project_name": "Logistic Dock", "status": 1}, {"user_name": "Subcontratado", "email": "sub@prismasg.com", "project_name": "Encolate 3.3", "status": 1}]
```

Ilustración 11: Peticiones en el navegador formato JSON

VII. **Colorzilla:** Esta herramienta es puramente de diseño. Permite captar el color, tanto en 'rgb' como en hexadecimal, de un elemento html. Su utilidad reside en "clonar" un color de otro sitio web, incluso de nuestra misma aplicación, en caso de desconocerlo. Su empleo ha sido hallar un color concreto, con el afán de que los colores del apartado gráfico sean concretos y no haya incoherencia visual.

Una vez que levantemos el servidor Angular, debemos ingresar, por defecto, en la dirección "localhost:4200". En dicha localización, en caso de trabajar en local, será donde trabajemos con nuestro proyecto.

- 3. XAMPP:** Xampp es una herramienta de uso gratuito que permite simular un servidor en tu equipo. Incorpora Apache y MySQL, por lo que podremos crear una base de datos MySQL, que es lo que necesitamos para el proyecto, y levantarla en el servidor. Nuestro Back-End se debe encontrar dentro de la carpeta '/htdocs', dentro del directorio raíz de Xampp. Para que nuestro proyecto tenga orígenes de datos, es necesario que arranquemos los servicios de Apache y MySQL desde el panel de control o desde comandos.
- 4. SQL WorkBench:** Esta herramienta no es estrictamente necesaria para nuestro proyecto, aunque si es recomendable usarla, debido a su versatilidad. Esto se debe a que Xampp ya incorpora un gestor de bases de datos MySQL (MariaDB), al cual accedemos desde la dirección '/phpmyadmin'. No obstante, dicha herramienta no tiene el mismo alcance que SQL WorkBench, el cual es más funcional y permite acciones más avanzadas, como la creación de diagramas de modelos de bases de datos.

4.3 Arquitectura de la aplicación

Tal y como comentamos en el origen de este documento, la distribución de la aplicación es **cliente-servidor** y se ha de realizar de la siguiente manera: En primer lugar, en el nivel más bajo de abstracción, se encuentra nuestra base de datos, que en nuestro caso será MySQL. Le procede el ORM, que se tratará de nuestro Back-End en Lumen 5.5. Dicha capa se encargará del **enlace entre la base de datos y las peticiones a esta**, convirtiendo las consultas en un servicio API Rest. Dicho servicio utilizará un modelo de datos diferente y se servirá de direcciones URL para que las consultas a la base de datos se consuman, de modo que al final de este apartado se mostrará un esquema conceptual con las peticiones en orden ascendente, de menor a mayor nivel de abstracción. En nuestro ORM contamos con varias capas, como son los **modelos**, por una parte, que definen las clases que representan las diferentes tablas de nuestro sistema de datos, y por otra parte las **rutas y controladores**, más cercanos al nivel alto, que definen los métodos y direcciones que atenderán las peticiones del servicio REST-Ful, y que utilizarán la estructura de los modelos. Más arriba tenemos nuestro Front-end en Angular, compuesto por **módulos, componentes, directivas y servicios**, entre otras cosas. Dentro del Front-End, nuestra petición pasará, primero por un servicio, ya que se halla en un nivel inmediatamente inferior que el componente que la consume, y posteriormente pasa al componente, que hace uso de ella. De este modo, las peticiones se harán mediante una función de 'Callback': una función en el componente efectuará una petición, usando un método de un servicio, y quedará esperando respuesta. En dicho servicio, dicho método recibe por parámetro las variables de la petición, como pueden ser la URL del servicio REST y el cuerpo de la consulta que debe hacer. En dicho servicio, que es un observable, la petición AJAX será mandada usando el módulo HttpClient que viene incluido en Angular 7, junto con las cabeceras, que usan el módulo HttpHeaders. La operación es devuelta junto con un código de petición (200 ok, 400 error, etc.) y es regresada a la función en el componente en forma de JSON. Dicho resultado, dependiendo de cuál sea la petición, será la desencadenante de un evento u otro. En el caso de listar un número x de clientes, la **petición es enviada usando la URL** al método 'listarClientes()' del Back-End, el cual hace un select * a la base de datos y se trae el resultado. La función de Callback utiliza un servicio que efectuá **una petición** http.get(URL) y se trae la respuesta a la petición, la cual es recogida en una 'suscripción' ('-> clientes[]') y almacenada en un array. Posteriormente, la información será mostrada en una tabla en la plantilla gráfica (Html) mediante una directiva '*ngFor' que recorre dicho array.

Por tanto, nuestra aplicación funcionará con **3 partes muy diferenciadas**, en la que encontraremos consultas, peticiones, funciones y métodos:

```
Select * from clientes --> listClients() --> URL --> this.http.get(url) --> this.requestService.getCli()
```

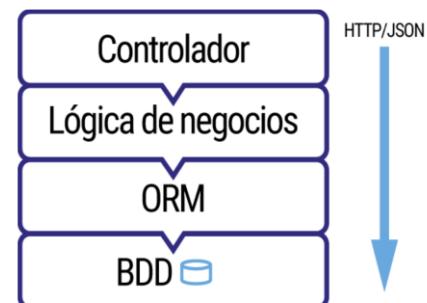


Ilustración 12: Jerarquía de capas

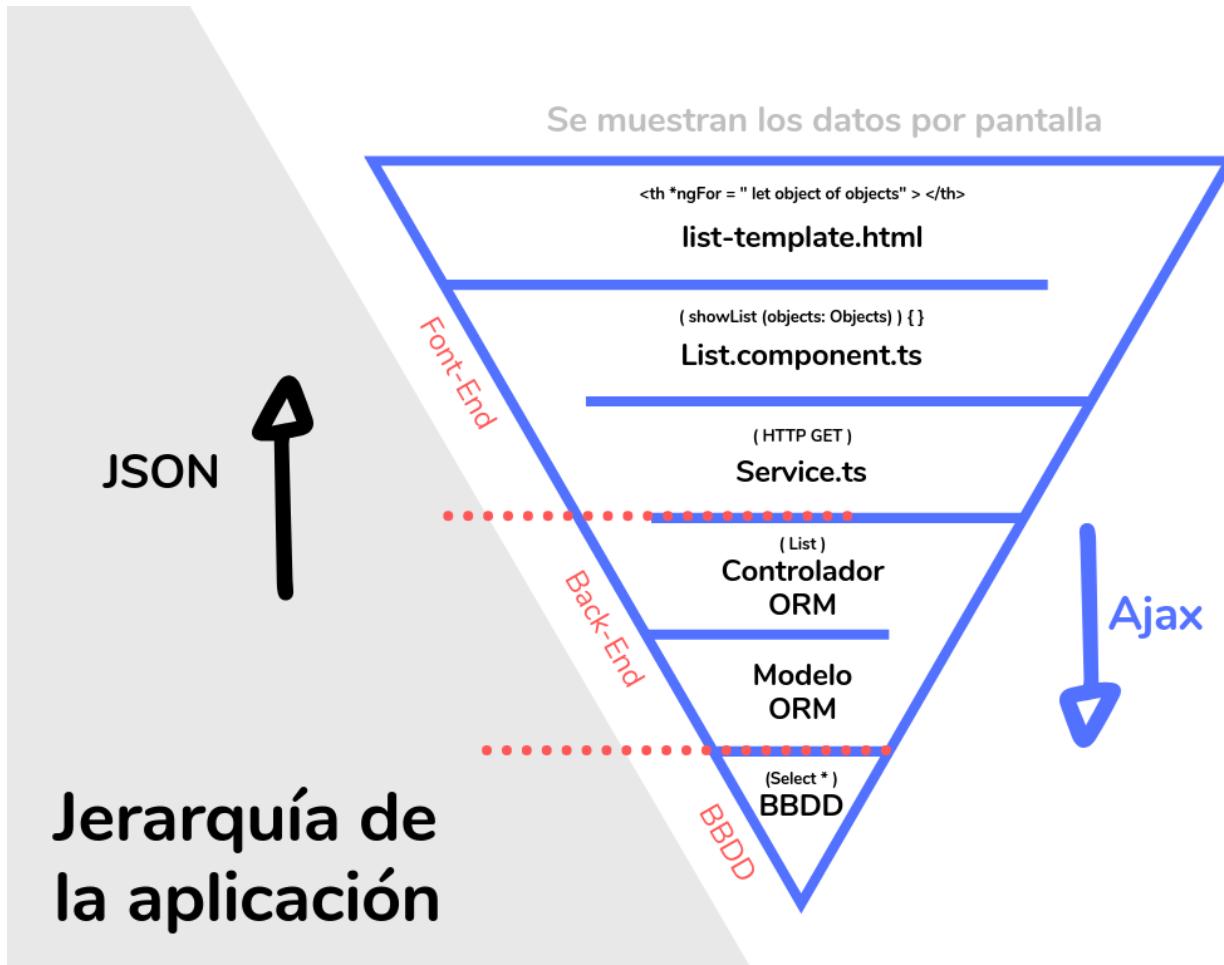


Ilustración 13: Jerarquía extendida de la aplicación

4.4 Estructura de la base de datos

Como hemos comentado en anteriores apartados, nuestra base de datos sigue el modelo MySQL. En total, nuestra aplicación requiere **16 tablas para su funcionamiento**, además de una tabla adicional creada por Lumen, donde quedan registradas las migraciones y la fecha de estas. No obstante, **en esta fase de desarrollo** (versión actual de la aplicación), **aun no son utilizadas todas las tablas de la aplicación**. En futuros desarrollos, la aplicación podrá gestionar toda la información aquí descrita en este diagrama:

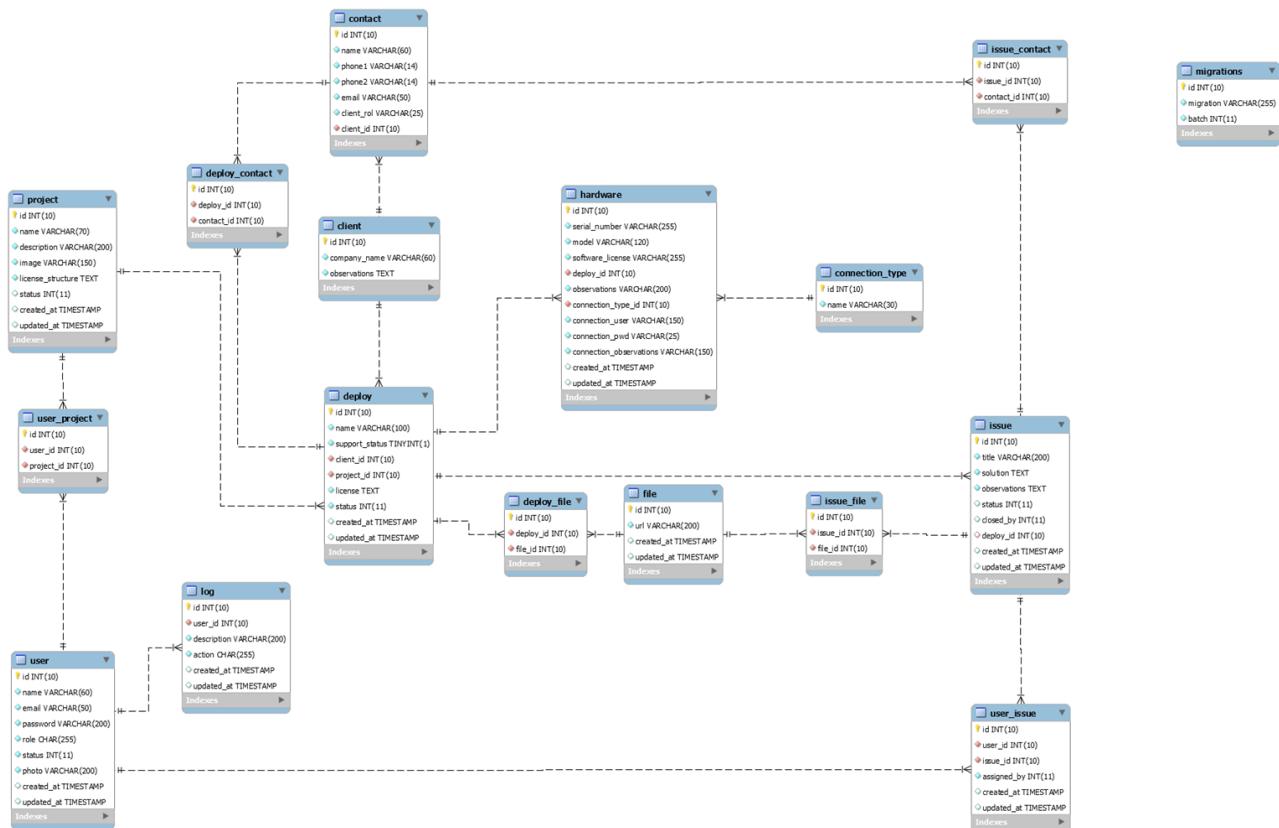


Ilustración 14: Esquema de la base de datos

Cabe destacar cuáles son las tablas empleadas, sobre todo las más importantes, y cuáles son sus propiedades y valores (nombraremos todas aquellas que si están siendo utilizadas en la presente versión de la aplicación):

- **user**: Tabla donde se guardan los datos de los usuarios de la aplicación, como el nick de usuario, la contraseña o el estatus.
- **user project**: Tabla intermedia de enlace entre usuarios y proyectos. En dicha tabla almacenamos el id de ambos para que estén relacionados. ¿Por qué utilizamos este tipo de tablas? (véase más abajo: uso de tablas intermedias)
- **project**: Tabla donde se registrarán los proyectos de la empresa, con su nombre, descripción, licencia, entre otras. Actualmente, la empresa solo tiene dos productos, que son Encolate® y Logistic Dock®.

- **deploy:** Tabla donde quedan registradas las implantaciones de la empresa. Contiene el nombre de la empresa, el estatus de soporte, el id de cliente y proyecto, etc. Es una tabla importante en el proyecto, ya que se realiza sobre ella uno de los CRUD diseñados.
- **deploy contact:** Tabla intermedia de los contactos de las implantaciones. En ella se registran los id de implantaciones y de contactos, para que puedan ser relacionados.
- **contact:** Contactos de la empresa. Quedan guardados en esta tabla aquellas personas, físicas o jurídicas, que contactan con nuestra empresa en representación de un cliente, con el fin de tratar una incidencia o implantación.
- **client:** Tabla donde se guardan los datos de los clientes de la empresa. Suelen ser otras empresas o entidades, por lo que en dicha tabla almacenamos el nombre de la compañía.
- **issue contact:** Tabla intermedia de contactos e incidencias. Al igual que ocurría con la tabla 'deploy contact', guardamos los id de contactos, pero con el id de incidencias.
- **issue:** Tabla donde se almacenan las incidencias acontecidas a lo largo del ciclo de vida de una implantación. Es una tabla importante del proyecto, ya que éste gira alrededor de las incidencias y el CRUD principal se realiza en torno a ellas. En dicha tabla encontramos del título de la incidencia, la solución llevada a cabo para solucionarla, las observaciones al respecto, junto con el id de implantación, el estatus de la incidencia y por quien ha sido cerrada (donde contamos con un tipo numérico que correspondería al id de usuario de la aplicación). El estado de las incidencias se define como:
 - cero (0): Inactiva. Se ha solucionado la incidencia y por tanto no aparece en el listado.
 - Uno (1): Activa. La incidencia se encuentra sin solucionar y aparece en el listado
 - dos (2): Borrado lógico. La incidencia ha sido eliminada manualmente del listado.
- **user issue:** Tabla intermedia entre usuarios e incidencias. Almacenan las id de incidencias y usuarios, ya que los usuarios están ligados a las incidencias que puedan atender y cerrar.

Uso de tablas intermedias: Para homogeneizar la base de datos, en lugar de enlazar dos tablas directamente, utilizamos una tabla intermedia que actuá como tabla auxiliar y que contiene **las claves primarias de ambas tablas**. Esto sucede porque estamos trabajando con dos tablas con una relación muchos a muchos. Un **usuario** o **varios usuarios** pueden tener asignado uno o varios **proyectos**; y, a su vez, uno o más **proyectos** pueden pertenecer a uno o varios **usuarios**. En una base de datos con un esquema relacional en tercera o cuarta forma normal, se rompe esta redundancia, con una tabla intermedia, que está compuesta por las claves primarias de las tablas que se relacionan con ella. Así se logra que la relación sea de uno a muchos, en los extremos. De esta forma, evitamos que existan usuarios con un id de proyecto de un proyecto que no existe, o viceversa, siendo obligatorio a la hora de crear una relación, que ambos elementos coexistan en la base de datos.

Relaciones en la fase de la aplicación: Como ya hemos comentado, la base de datos del proyecto no es usada en su totalidad. Eso es debido a que dicha aplicación, aunque

ya es altamente funcional (cumple su cometido básico), **aun no se encuentra totalmente terminada**. Las relaciones en esta fase actual es la siguiente:

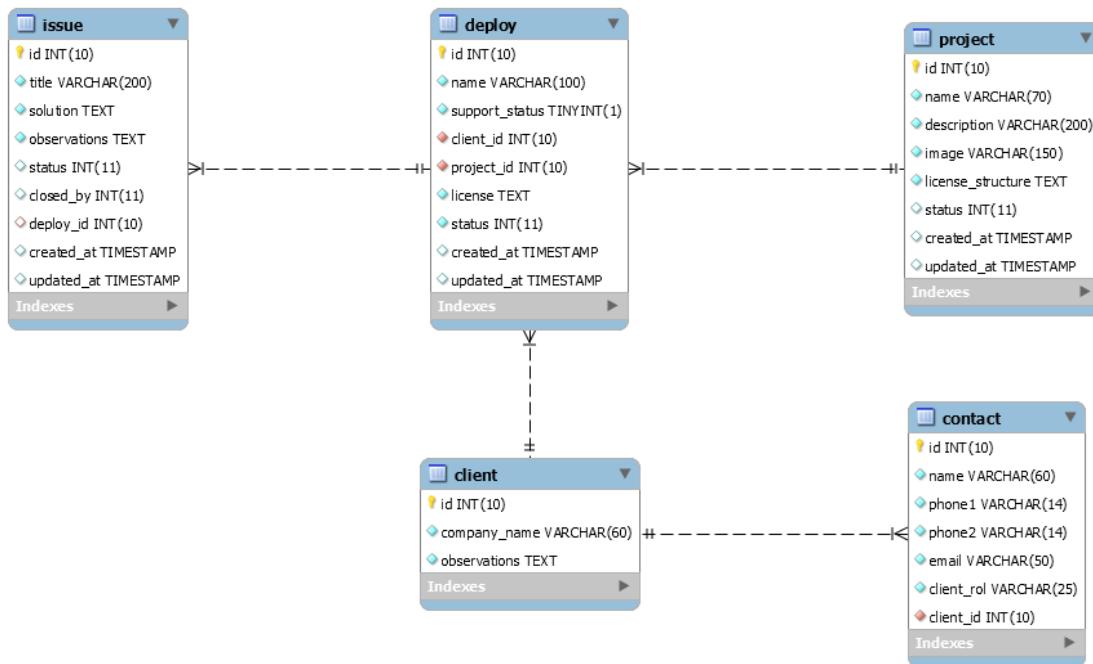


Ilustración 15: Esquema de la fase actual de la aplicación

La aplicación se centra especialmente en incidencias e implantaciones. Ambas son necesarias para el funcionamiento de la aplicación, por lo que dichas tablas se encuentran creadas e infladas con información desde el comienzo. Atendiendo al diagrama anterior, lo que se ha hecho es desarrollar la aplicación alrededor de estas dos tablas, añadiendo las tablas que eran necesarias para una información básica, de modo que la aplicación en su estado actual emplea datos de las tablas aquí señaladas para llevar a cabo su funcionamiento. Posteriormente, veremos cómo se desarrollan las consultas a dichas tablas

4.5. Estructura del ORM

Hasta ahora, hemos visto cómo se estructura nuestra base de datos del proyecto. A la hora de consultar datos de una incidencia, por ejemplo, resulta interesante que, además de los datos recogidos en la tabla 'issue' (incidencias), se muestren otros datos de interés que se encuentran en otras tablas, como puede ser el caso del nombre de la implantación a la que pertenece, el nombre del proyecto al que pertenece la implantación, el contacto, etc. En nuestra base de datos, esto se consigue gracias a las relaciones entre las tablas del proyecto. Si quisieramos consultar desde un gestor de bases de datos, como es el caso de SQL Workbench, la forma de conseguir los datos que requerimos es haciendo una consulta utilizando la orden 'JOIN'. Mediante ellos, podemos enlazar contenido de diferentes tablas empleando los identificadores únicos (claves primarias y foráneas). Un ejemplo de esto es la siguiente sintaxis:

```
SELECT <nombre_tabla1.nombre_columna, nombre_tabla2.nombre_columna ... >
FROM <nombre_tabla>
INNER JOIN <nombre_tabla2>
    ON <nombre_tabla.nombre_columna = nombre_tabla2.nombre_columna>;
```

Y el resultado obtenido es una serie de filas que corresponden a la consulta deseada:

columna1	columna2	columna3	columna4
dato1	dato2	dato3	dato4
dato1	dato2	dato3	dato4
dato1	dato2	dato3	dato4
....

Ahora bien. Si queremos que nuestro Back-End basado en el framework Lumen ejerza este tipo de acciones, debemos partir de esta base. La sintaxis no será exactamente la misma, pero todo lo aplicado hasta ahora nos servirá de manera orientativa. Las consultas a realizar serán las ya vistas en el informe; solo tenemos que adaptarlas al framework.

Lo primero que tenemos que tener en cuenta es la estructura del framework, y como se distribuyen las funciones y la metodología por cada uno de sus ficheros. Contamos con diversos directorios, cada uno de ellos alberga un tipo de fichero .php específico que se encarga de una función determinada. En nuestro caso, si queremos efectuar una consulta SQL a la base de datos a través de nuestro fra-

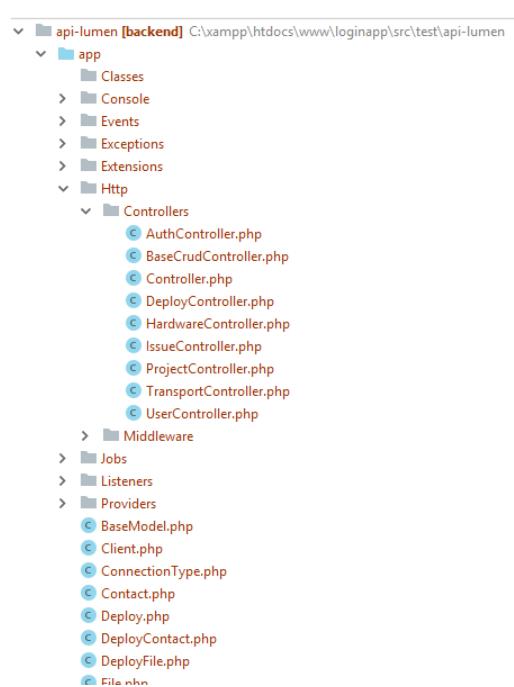


Ilustración 16: Genealogía de Lumen

mework, tal y como está estructurado, necesitamos considerar tres cuestiones: El método, la ruta del método y la ruta relativa de la petición final.

Para crear un método que efectúe las consultas y liste el resultado obtenido, nos dirigimos a **App → Http → Controller**. En dicho directorio se almacenan los controladores: la parte del programa que **ejecuta la lógica** y en el que decidiremos las acciones que queremos que puedan ser tomadas por el framework. El controlador se encarga de recoger toda la lógica **referente a una o varias clases** que referencia a **una o varias tablas** a la base de datos. No son estrictamente necesarios, pero su uso nos permite estructurar mejor la aplicación. Se trata de una clase que extiende (hereda) de 'Controller' y en la que definiremos los métodos que posteriormente serán usados de manera implícita e indirectamente por el Front-End a la hora de hacer un CRUD. En nuestro caso, **queremos que se nos muestren resultados de varias tablas en una sola consulta**, de modo que elaboraremos un método que veremos a continuación y explicaremos sus particularidades:

Lo primero es tener creados los modelos de las diferentes tablas en Lumen. Veamos un rápido ejemplo de cómo se estructura un modelo. Usaremos como tal al modelo de la clase 'Issue' (incidencia):

```
class Issue extends BaseModel
{
protected $table = 'issue';
public $timestamps = true;

public static $validationArray = [
'title' => 'required|string|max:200',
'solution' => 'required|text',
'observations' => 'required|text',
'estatus' => 'required|integer',
'closed_by' => 'required|integer',
'deploy_id' => 'required|integer',
];

protected $fillable = [
'title', 'solution', 'observations', 'estatus', 'closed_by', 'deploy_id'
];

public static $searchArray = [
'title', 'solution', 'observations', 'estatus', 'closed_by', 'deploy_id'
];
}
```

En él, especificamos los campos de la tabla, que tipo de datos emplea, sus restricciones, etc.

Pues bien, a la hora de crear un método para generar una lista de incidencias (consultas a la base de datos) mediante una petición GET, usaremos un controlador que, a su vez, hará uso de tantos modelos como tablas necesitemos para llevar a cabo la operación.

Veamos en la práctica como desarrollamos un método que recoja las incidencias, junto con la información del proyecto, el contacto y la implantación:

```

namespace App\Http\Controllers;

use App\Issue;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

class IssueController extends BaseCrudController
{
    public function __construct()
    {
        parent::__construct(Issue::class);
    }

/* GET */

    public function getMainAdvanced()
    {
        $lista_Dashboard = DB::table('issue')
            ->join('deploy', 'deploy.id', '=', 'issue.deploy_id')
            ->join('project', 'project.id', '=', 'deploy.project_id')
            ->join('client', 'client.id', '=', 'deploy.client_id')
            ->join('contact', 'contact.client_id', '=', 'client.id')
            ->select('issue.id', 'client.company_name', 'deploy.name AS
deploy_name', 'project.name AS project_name', 'contact.name AS
contact_name', 'issue.title AS issue_title', 'project.status')
            ->where('issue.status', '=', 1)
            ->get();
        return $lista_Dashboard;
    }
}

```

Para usar este método, debemos dirigirnos al fichero 'routes -> web.php' y, desde ahí, colocaremos nuestro método en una ruta hacia la petición que queremos publicar:

```

use App\Http\Controllers\Controller;
$router->GET('api/auth/mainadvlist', 'IssueController@getMainAdvanced');

```

Donde especificamos el método a emplear ('GET'), la URL de la API que hace la petición al método ('api/auth/mainadvlist'), el nombre del método ('@getMainAdvanced') y el controlador donde se encuentra ('IssueController').

Si hacemos uso de la aplicación 'Postman', y realizamos la petición a dicha consulta, comprobaremos que ésta se realiza correctamente, tal y como deseamos que ocurra:

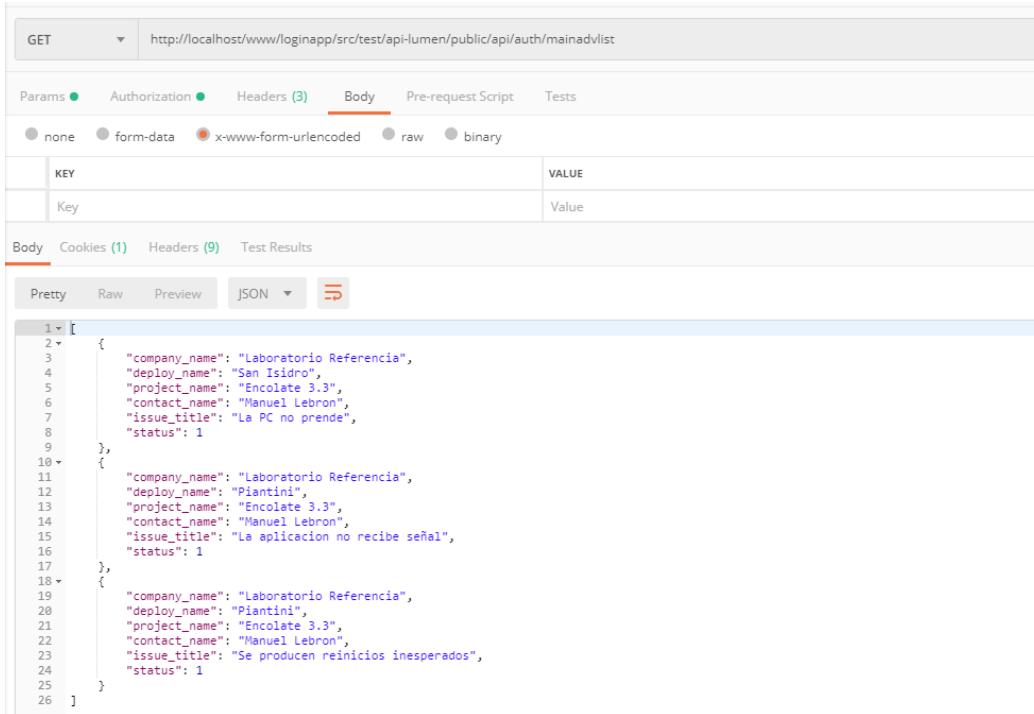


Ilustración 17: Petición GET en Postman de las incidencias

Todo el CRUD de implantaciones e incidencias del API Rest ha sido probado previamente en Postman antes de ser utilizado en el Front-End de nuestra aplicación.

4.5.1. Middleware y Bearer Token:

Sin embargo, lo ideal no es que las consultas se hallen tan fácilmente mediante la URL de la petición, sin ningún tipo de mecanismo de seguridad. Ya que, en caso de que nuestro programa deje expuesta dicha dirección, podríamos sufrir de **inyección de código**. Para evitar esto, lo recomendable es utilizar el mecanismo **JWT**. Lo que conseguimos con esto es que las peticiones no se realicen mientras las cabeceras de estas no contengan dicho mecanismo de seguridad en ellos. Esto igual no resulta tan interesante para la consulta anterior, en la que, mediante un GET, extraímos una serie de datos de distintas bases de datos, ya que, con este tipo de métodos, la información no se ve comprometida. Sin embargo, la cosa cambia cuando se trata de métodos POST/PUT/DELETE, donde la información puede ser modificada y puede surgir la inyección de código.

Para poner en marcha este recurso, lo primero que tenemos que hacer es definir donde vamos a otorgar al usuario la posibilidad de utilizar las consultas en cuestión. Y no hay mejor lugar que en la propia validación de usuario. La lógica es la siguiente: El usuario procederá a validar su sesión, con el fin de poder utilizar los servicios de la aplicación. El usuario que no se encuentra validado:

1. No puede acceder al contenido de las pantallas de la aplicación
2. **No puede realizar las peticiones correspondientes**, Siendo este último punto el que más nos interesa. Para que esto ocurra, debemos controlar el método del Back-End donde hacemos la validación de usuario. Veamos como ocurre:

```

namespace App\Http\Controllers;

use App\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;
use Tymon\JWTAuth\Exceptions\JWTException;
use Tymon\JWTAuth\Exceptions\TokenExpiredException;
use Tymon\JWTAuth\Exceptions\TokenInvalidException;
use Tymon\JWTAuth\Facades\JWTFactory;
use Tymon\JWTAuth\JWTAuth;

class AuthController extends Controller
{
    public function __construct(JWTAuth $jwt)
    {
        $this->jwt = $jwt;
    }

    public function loginPost(Request $request)
    {
        $this->validate($request, [
            'name' => 'required|max:255',
            'password' => 'required',
        ]);

        // Get role
        $user = User::where('user.name', $request->name)
            ->first();
        if(!$user)
            return response()->json(['error' => 'User dont found'], 404);

        if(!Hash::check($request->password, $user->password))
            return response()->json(['error' => 'Bad password'], 404);

        $customClaims['role'] = $user->role;
        $payload = JWTFactory::addClaims($customClaims);

        try {
            if (!$token = $this->jwt->fromUser($user)){
                return response()->json(['user_not_found'], 404);
            }
            } catch (TokenExpiredException $e) {
                return response()->json(['token_expired'],
$e->getStatusCode());
            } catch (TokenInvalidException $e) {
                return response()->json(['token_invalid'],
$e->getStatusCode());
            } catch (JWTException $e) {
                return response()->json(['token_absent' => $e->getMessage()],
$e->getStatusCode());
            }
        }

        $response["token"] = $token;
        $response["id"] = $user->id;
        $response["role"] = $user->role;
        return response()->json($response);
    }
}

```

En resumen, el método recibirá una petición por parámetros. En ella, recibirá el nombre de usuario y la contraseña. Puesto que este controlador tiene cargado el modelo de la tabla 'user', es anecdótico hacer una consulta a dicha tabla y **comprobar que ambos valores son correctos**. Recordemos que, en la base de datos, **la tabla de usuarios contiene el campo de contraseña cifrado mediante Bcrypt**, por lo que, a la hora de comprobar la contraseña, el Back-End debe ser capaz de **descifrar la contraseña cifrada** para compararla con la que le llega por parámetros, de ahí que usemos la función 'Hash' para descifrar el resultado.

Si los parámetros recogidos coinciden con ambos campos de la base de datos, el sistema dará por correcta la validación y, entonces, mediante la función 'JWTFactory', el sistema creará un 'token' que no es más que una **clave pública** que permite ser autenticado constantemente por la aplicación y, al mismo tiempo, contiene varios datos referentes a la validación, como son el **propio token** (cadena de caracteres), el **id del usuario** que ha iniciado sesión y **el rol que desempeña** (esto último muy interesante para el sistema de gestión multi-rol de la aplicación).

A partir de aquí, ya tenemos como podemos proteger más nuestra aplicación, ya que, a la hora de enviar una petición al Back-End, éste **solicitará que la petición que llega desde el Front-End incluya en sus cabeceras el token** que el propio Back-End ha otorgado al usuario y que éste ha debido de ser guardado en el **LocalStorage** (recordemos que LocalStorage es una propiedad que accede al objeto 'Storage' y tiene la función de almacenar datos de manera local, de forma indefinida o hasta que se decida limpiar los datos del navegador). Lo siguiente es indicar en el Back-End que peticiones están protegidas por dicho Token y cuáles no. Aquí es donde empleamos el **Middleware**. Recordemos que el Middleware es una capa de abstracción adicional que se encuentra justo antes de los controladores de la API.

Lumen incluye por defecto (por herencia de Laravel) un fichero Middleware, donde podemos especificar de qué forma queremos que salte el sistema de seguridad. Por ejemplo:

```
namespace App\Http\Middleware;
use Closure;
use Illuminate\Contracts\Auth\Factory as Auth;

class Authenticate
{
    public function __construct(Auth $auth)
    {
        $this->auth = $auth;
    }

    public function handle($request, Closure $next, $guard = null)
    {
        if ($this->auth->guard($guard)->guest()) {
            return response('Unauthorized.', 401);
        }

        return $next($request);
    }
}
```

En dicho fichero, especificamos que se haga la petición y que, en el caso en el que se incumplan las condiciones para que se lleve a cabo satisfactoriamente (en nuestro caso, que no exista el token en la cabecera, o que éste sea incorrecto o esté expirado), la consulta devuelva un mensaje del tipo: "Unauthorized, 401".

¿Cómo protegemos las rutas de las peticiones con el Middleware? Lo primero es utilizar "auth" como prefijo para las consultas o grupos de consultas. Echemos un vistazo al fichero de rutas:

```
$router->group(['Middleware' => 'auth:api', 'prefix' => 'api'],
function($app) {
    $app->group(['prefix' => 'user'], function($app){
        $app->get('{id}', 'UserController@find');
        $app->get('', 'UserController@filter');
        $app->post('', 'UserController@store');
        $app->put('{id}', 'UserController@update');
        $app->delete('{id}', 'UserController@delete');
    });
})
```

Como vemos, todos los métodos que respondan a '/user', se encuentran dentro del Middleware que, a su vez, tiene un prefijo 'auth: "api'. Por lo que **todas las consultas /api/user deben contener el token en sus cabeceras** que, normalmente, suele tener esta forma: "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9eyJhZGRDbGFpbXMiOnsicm9sZSI6ImFkbWluIn0sImlzcyI6Imh0dHA6Ly9sb2NhbGhvc3Qvd3d3L2NhcHRhaW5Ici9zcmMvdGVzdC9hcGktbHVtZW4vcHVibGljL2FwaS9hdXR0L2xvZ2luliwiaWF0Ijox-NTU5MDQzMDC4LCJleHAIoJE1NTkwNzkwNzgslm5iZii6MTU1OTA0MzA3OCwianRpIjoiZ-DhTZ0JMYnprZThwS2FBcIsInN1Yil6MSwichHJ2IjoiO-DdIMGFmMWVmOWZkMTU4MTJmZGVjOTcxNTNhMTRIMGIwNDc1NDZhYSJ9.xDoYav9qmLGhNbAJODuVBbGJhMj6W581bQoaoMf9uig"

A este tipo de autenticación se le conoce como "**Bearer Token**". Pero, ¿qué es exactamente?

OAuth2 es uno de los protocolos de autorización más utilizados en la web. OAuth2 introduce la utilización de **bearer tokens** (o tokens al portador), los cuales:

- Son un tipo de token de acceso, mediante el cual, el simple hecho de poseerlo, proveerá acceso a los recursos protegidos
- Una vez obtenido, éste puede ser enviado en las solicitudes a las APIs de varias formas ('Query parameter', 'Form Encoded body parameter', 'HTTP Authorization header').

En nuestro caso concreto, será enviado a través de 'HTTP Authorization header', que utilizará nuestro Front-End.

En resumen, con este mecanismo obligamos a incluir en las cabeceras de la petición el token generado para el usuario una vez que ha sido validada su sesión. El token se encontrará en todo momento almacenado (en caso de que esté) en el 'LocalStorage'. Las peticiones enviadas al Back-End deberán recoger del 'LocalStorage' el token generado y emplearlo en las cabeceras de la petición. En caso correcto, la petición se realizará. La petición que no venga acompañada de dicho Token, o esté este considerado como incorrecto o expirado, dará un error. De esta forma evitamos que la información pueda ser capturada por cualquiera, conocida la URL de la API y, a su vez, evitamos la inyección de código desde usuarios no autorizados.

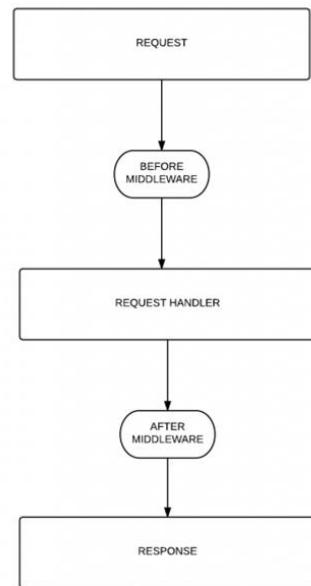


Ilustración 18: Esquema Back-End y Middleware

4.6 Estructura de la aplicación

Hemos visto la arquitectura y capas de la aplicación. Y hemos visto que se trata de una aplicación Cliente-Servidor. En el lado del servidor, hemos estudiado la estructura de la base de datos del proyecto. Ahora estudiaremos la estructura de la aplicación, de cara al Front-End y el cliente.

El paradigma de programación que hemos seguido es el de Vista – Modelo – Controlador. Esto es Angular se encuentra bastante diferenciado, ya que la vista son las plantillas Html, los modelos son las clases typescript del módulo donde se definen los objetos a utilizar (incidencias, implantaciones, etc.) y los controladores son los componentes y servicios que se encargan de ejecutar la lógica del programa. Así pues, veamos cómo funciona la estructura de nuestra aplicación de una forma más extensa:

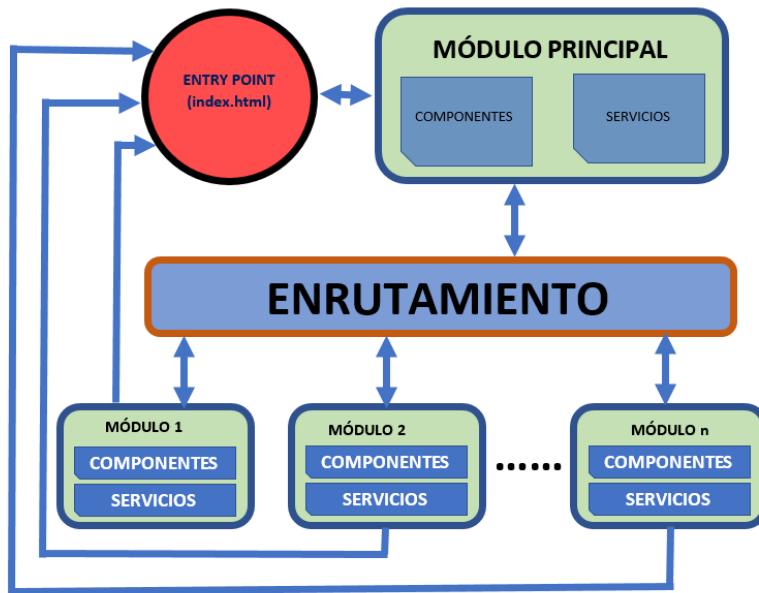


Ilustración 19: Esquema Front-End Angular

1. **index.html:** Es el fichero de la interfaz principal de la aplicación. Es la puerta de entrada a la aplicación y se suele comunicar con el módulo principal mediante las etiquetas que llaman a dicho módulo:

```
<body> <app-root></app-root> </body>
```

En él, debemos importar los script de javascript que afecten a toda la programación, así como las librerías css o bootstrap que queramos utilizar.

2. **app.module.ts:** Se trata de la configuración del módulo principal. En él, debemos listar todos los componentes que vamos a utilizar, así como los servicios ('providers') y módulos de Angular ('imports') que vayamos a usar. En dicha clase podemos colocar la lógica que queramos que sea común para todo el módulo (por ejemplo, crear un temporizador para tareas dentro del módulo). En mi caso, además de los servicios y métodos, he importado el módulo de enrulado en este fichero, con la siguiente función:

```
RouterModule.forRoot([{...}, {...}])
```

En el que escribiremos todas las rutas de la aplicación, tal y como veremos más adelante.

3. **app.component.ts:** Fichero del componente principal del módulo. En él, escribiremos la lógica que queremos que se repita en el resto de componentes hijos que se comuniquen con él. Se encuentra enlazado con el fichero app.component.html, en el cual desarrollaremos la interfaz del componente. Al usar navegación por rutas, dicho fichero contiene lo siguiente:

```
<router-outlet></router-outlet>
```

4. **app.service.ts:** Se trata de un servicio en Angular cualquiera. En nuestra aplicación contamos con varios servicios: para la autenticación, para el filtrado de datos, para las peticiones CRUD, etc. Un servicio es una clase typescript empleada de forma peculiar para que cualquier componente pueda incorporarla de forma sencilla y gozar de todas sus funcionalidades. Para que un servicio sea usado como tal, debe contener la directiva: '@Injectable()' .
5. **app-routing.module.ts :** Se trata de un módulo para el enrutado de componentes, aunque podemos usarlo de forma nativa como un módulo dentro del módulo principal de nuestra aplicación, tal y como hemos visto en la página anterior. De usarlo de forma independiente, nuestro fichero tendría el siguiente aspecto:

```
const appRoutes: Routes = [
  { path: 'crisis-center', component: CrisisListComponent },
  { path: 'heroes', component: HeroListComponent },
  { path: '', redirectTo: '/heroes', pathMatch: 'full' },
  { path: '**', component: PageNotFoundComponent }
];
```

Visto cómo se estructuran los programas en Angular, solo queda hacer hincapié en el elemento más complejo de todos: El componente.

4.6.1. Los componentes

Un componente en Angular es una combinación de un archivo html con un fichero typescript y una hoja de estilos, que pueden ser css, scss o sass, para crear un elemento con características propias, tanto de comportamiento como de apariencia, que se puede mostrar en un navegador.

La interacción entre los tres ficheros es curiosa, pudiendo de esta manera combinar diseño con lógica de programación y estilos, siguiendo el paradigma MVC. En todo componente, existe un fichero typescript con el siguiente **Decorador** justo antes de exportar la clase:

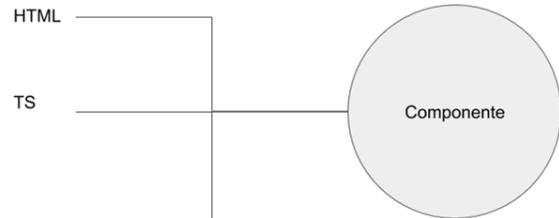


Ilustración 20: Esquema de un componente Angular

```
@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
```

Donde se especifican los ficheros que forman parte del tridente del componente. Para interactuar entre el diseño y la lógica, se usan las **Directivas y los bindings**, entre otras herramientas.

Una **directiva** en Angular se representa como **un atributo** en una etiqueta HTML. Este atributo está dotando al elemento del **DOM** que la contiene de un comportamiento. Ese comportamiento lo definimos nosotros y se puede utilizar en todos los elementos que queramos. Un **Data binding** o enlace de datos, es una **técnica general que une una fuente de datos entre el origen de datos y la vista**; y se encarga de sincronizarlos. Esto provoca que cada cambio en los datos se refleje automáticamente en los elementos que están sincronizados.

Por ejemplo, si queremos guardar los datos de un formulario, como es el caso del componente de 'Inicio de Sesión', donde necesitamos que el usuario introduzca un usuario y una contraseña, previamente debemos tener declaradas dos variables en nuestro typescript que representen dichos valores:

```
user = new User();
name: string = "";
password: string = "";
```

Ahora, en nuestra plantilla .html, contamos con la siguiente forma de recoger los datos:

```
<form #formLogin="ngForm" id="login-form" (ngSubmit)="onSubmit()">
<input type="text" name="name" placeholder="Usuario" id="name"
[(ngModel)]="name" class="form-control" />
<input type="password" name="password" placeholder="Clave" id="password"
[(ngModel)]="password"
class="form-control" />
<input type="submit" value="Iniciar sesión" class="btn">
</form>
```

En la plantilla, mediante la directiva '**ngModel**', hacemos que el valor de las variables del fichero typescript sean las introducidas en el las etiquetas 'input' que hacen referencia a cada una de ellas. A su vez, el formulario se encuentra suscrito a una función '**onSubmit()**', la cual activará un método en la lógica del componente:

```
onSubmit() {
this.authenticationService.login(this.name,
this.password).pipe(first()).subscribe(
data => {
this.router.navigate(['admin']);
},
error => {
console.log("Se ha producido un error al ingresar");
});
}
```

4.7. Diseño de la interfaz

El estilo de diseño elegido para la aplicación web es "Flat Design". La aplicación se verá visualmente agradable, al mismo tiempo que atendemos a criterios de usabilidad. Es importante que el usuario encuentre el diseño homogéneo y cuya disposición de elementos sea la acertada, es decir, se adapte a las cotas de diseño actual (Barra de navegación con botones y menús en cascada, diseño de tablas y formularios, etc.). Cabe destacar que todas las ilustraciones en pantalla son del diseño final y que, por tanto, explicaremos en base a ellas todas las consideraciones acerca del mismo.

El framework empleado para el diseño es 'Bootstrap'. Concretamente, en su versión 4, aunque también se han empleado elementos de Bootstrap 3. Bootstrap nos brinda diversas facilidades a la hora de diseñar la interfaz de la aplicación, ya que facilita la construcción de elementos visuales y hace que colocar elementos como un botón y otorgarle estilo sea infinitamente más simple. Basta con importar las librerías de Bootstrap y definir un elemento con una clase ya definida en Bootstrap (como en este caso, 'btn-primary'):

- **Pantalla Inicial:** En la pantalla Inicial encontraremos una pantalla de bienvenida, en la cual se nos muestra el eslogan de la aplicación, un contenedor central con imágenes orientativas a modo de esquema representativo visual de la aplicación; y ya se dejan ver los distintos elementos que conforma la aplicación, como lo son la barra superior de navegación, el 'footer' y la paleta de colores utilizada para el diseño final, así como el logo de la aplicación, situado en el centro de la barra superior. El único elemento interactivo que nos encontramos en esta página es el botón de 'Iniciar sesión', el cual producirá los siguientes cambios al ser pulsado:
 - a) Que al pulsarlo se abra la ventana de inicio de sesión, puesto que la sesión de usuario se encuentra inactiva (o bien, el Token se encuentra invalidado por expiración o identificación incorrecta)
 - b) Que al pulsarlo se abra directamente el panel de administración de incidencias. Esto se debe a que la sesión del usuario en el navegador web se encuentra ya validada y activa. Por lo que no es necesario que volvamos a iniciar sesión.



Ilustración 21: Pantalla bienvenida de nuestra aplicación

- **Inicio de Sesión:** La ventana de inicio de sesión no alardea de hallarse saturada de elementos visuales y funcionalidades. Simplemente, se nos desplegará un formulario de inicio de sesión, con el cual interactuaremos para acceder a la aplicación. Podemos volver atrás mediante un botón "volver", situado en la parte superior izquierda, o bien continuar con la validación de usuario. En este último caso, ingresaremos únicamente el nombre de usuario y la contraseña. En caso de hallarse el usuario y la contraseña en la base de datos y ejecutarse correctamente la validación (véase el apartado anterior "Estructura del ORM"), el sistema accederá a la pantalla de administración. En caso contrario, el formulario mostrará un error señalizado en color rojo, que indica que la validación no es correcta. Para llevar a cabo dicha acción, se empleará el botón de "Iniciar sesión", situado al final de dicho formulario.

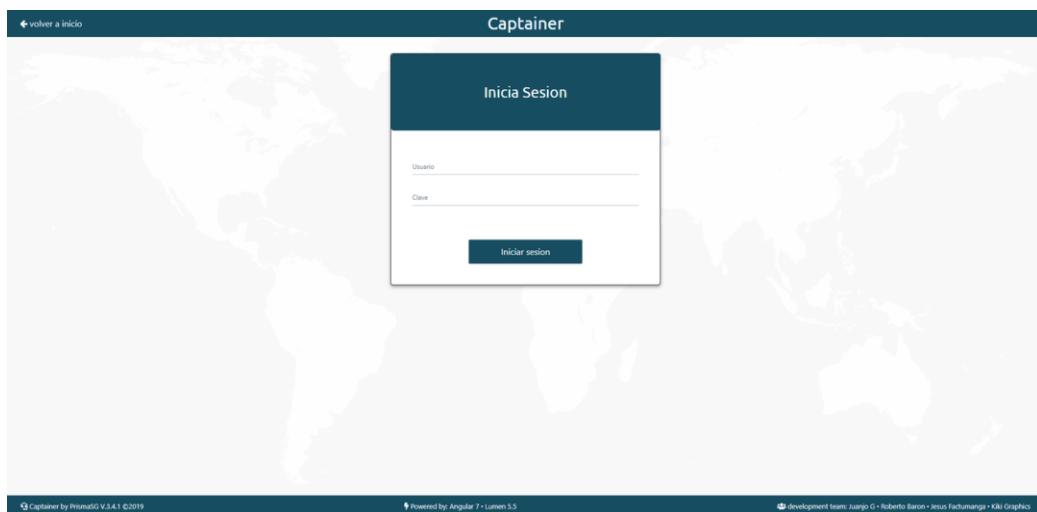


Ilustración 22: Menu de inicio de sesión

- **Menús de administración y listado de incidencias e implantaciones:** Justo al acceder por primera vez, nos encontramos con la ventana de administración de incidencias, donde se recogen las incidencias registradas, junto con el contacto, la implantación a la que pertenece y el proyecto al que pertenece dicha implantación. En estas ventanas, lo normal es encontrar la barra de navegación lateral, la cual nos permite movernos por diferentes categorías, además de una barra de filtrado que nos permite encontrar de una manera más fácil el elemento que estamos buscando. Más adelante, hablaremos de como implementamos esta barra de filtrado y volveremos a repasar conceptos anteriores.

ULTIMAS INCIDENCIAS					
Company Name	Deploy Name	Project Name	Contact Name	Issue Title	Status
Laboratorio Referencia	San Isidro	Encolate 3.3	Manuel Lebron	La PC no prende	1
Laboratorio Referencia	Plantini	Encolate 3.3	Manuel Lebron	La aplicacion no recibe señal	1
Laboratorio Referencia	Plantini	Encolate 3.3	Manuel Lebron	Se producen reñidos inesperados	1
Carrefour Huelva S.A.	Carrefour Huelva	Logicistic Dock	jesus javier Cocacolo	No se realiza un cambio de muelle	1
Carrefour Huelva S.A.	Carrefour Huelva	Logicistic Dock	jesus javier Cocacolo	Los camiones se desconectan de la central al entrar en el punto de carga	1
Carrefour Huelva S.A.	Carrefour Huelva	Logicistic Dock	jesus javier Cocacolo	Los camiones se desconectan en el punto logistico	1
Salvesen Logistica	Salvense 1	Logicistic Dock	Manolo Rodriguez Leon	la aplicacion se desconecta a los pocos minutos de iniciar	1
Salvesen Logistica	Salvense 1	Logicistic Dock	Manolo Rodriguez Leon	la aplicacion se bloquea al cerrar y querer volver a abrirla	1

Ilustración 23: Pantalla de administración de incidencias

Desde la tabla de incidencias, nosotros podemos acceder a los detalles de cada incidencia de forma independiente, es decir, que cada elemento generará un enlace a un componente que mostrará los detalles en función del elemento al que intentamos acceder. Desde dicha ventana, nosotros podremos editar o archivar (borrado lógico) dicha incidencia. Véase que, para las tablas de resultados, hemos empleado la clase "table-striped" de Bootstrap para este diseño de la aplicación. Los botones siguen también un diseño homogéneo en Bootstrap y, por lo general, el resto de componentes gráficos también incorpora este tipo de patrones de diseño.

Nombre de la compañía:	Laboratorio Referencia
Nombre de la implantación:	Plantini
Nombre del proyecto:	Encolate 3.3
Contacto:	Manuel Lebron
Título de la incidencia:	La aplicacion no recibe señal
Status:	1
Solución:	El dispositivo no interactuaba con la aplicación porque no estaba configurado a la red correctamente
Observaciones:	Debemos señalizar en la aplicación al cliente que no se encuentra debidamente conectado
Cerrado por:	1

Ilustración 24: Vista de detalles

Si pulsamos sobre el botón "Editar", se desplegará un modal con un formulario que, por medio de una petición PUT, alterará el registro existente en la base de datos. La utilidad de esto es, principalmente, dar por resuelta una incidencia, o simplemente corregir alguna errata de redacción.

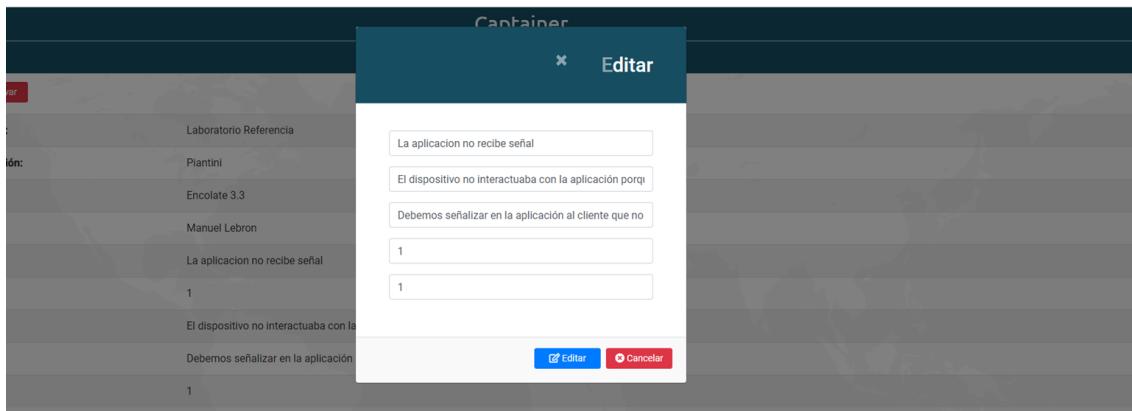


Ilustración 25: Modal de edición

Si pulsamos sobre el botón de "Archivar", se debe abrir un mensaje de confirmación que nos obligue a hacer énfasis en la acción que debemos tomar. Esto se hace, básicamente, para que el usuario no haga clic por error y por ende, archive sin querer una incidencia en la base de datos (recordemos que un borrado lógico es un proceso reversible y que de todas formas, podemos consultar la incidencia en la base de datos). Este mensaje de confirmación se encuentra incrustado en un modal, igual que ocurría con el formulario de edición. Y, en ambos casos, se ha empleado un componente para su diseño y funcionalidad. En caso de aceptar archivar la incidencia, dicha incidencia ya no se mostrará en el listado, ya que este solo muestra las incidencias activas.

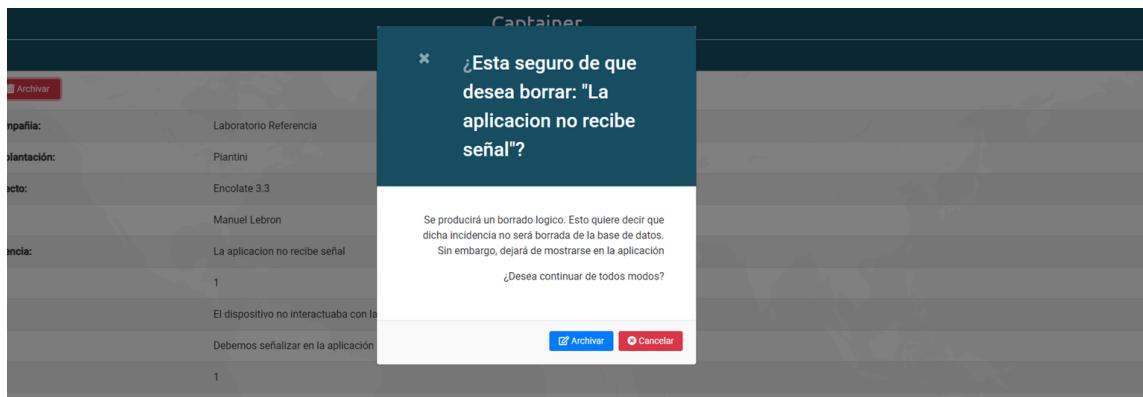


Ilustración 26: Modal de borrado

- **Formularios de inserción y edición de elementos:** Contamos con dos componentes dedicados a insertar elementos en la base de datos, los cuales insertan incidencias e implantaciones, respectivamente. Para acceder al formulario de inserción de incidencias, debemos hacerlo desde la ventana principal, pulsando sobre "Añadir incidencia", o bien escribiendo la dirección en la barra del navegador. En el caso de querer añadir una implantación, debemos hacerlo desde

Opciones avanzadas. En esta fase de desarrollo de la aplicación, el diseño es simple y minimalista. Se han suprimido la decoración visual de los elementos en pantalla y se muestra en su lugar un diseño muy plano, sin que ello afecte a la funcionalidad de la aplicación. Una vez relleno cualquiera de estos formularios, el sistema enviará una petición POST y la información, en caso de hallarse correctamente, quedará registrada en el sistema.

The screenshot shows a web-based application interface titled "Captainer". At the top, there's a navigation bar with a "volver atrás" button and the "Captainer" logo. Below the header, a section titled "REPORTAR NUEVA INCIDENCIA:" contains several input fields: "Título de la incidencia", "Solución al problema", "Observaciones", "Cerrado por:", and "ID de implantación". At the bottom of this section are two buttons: "Añadir" (in blue) and "Limpiar" (in red). To the right of the form, a large red triangle with a white exclamation mark is overlaid on the page. The footer of the page includes copyright information: "Captainer by PrismaSO V3.4.1 ©2019", "Powered by: Angular 7 + Lumen 5.5", and "development team: Juanjo G · Roberto Barón · Jesus Factumanga · Kiki Graphics".

Ilustración 27: Formulario de inserción de nuevas incidencias

- Cerrar sesión:** En el menú principal, así como en cualquiera de las ventanas de categorías por proyecto, encontraremos un botón de cerrar sesión que, tal y como vemos en la siguiente imagen, nos obliga a confirmar el cierre de sesión, el cual provocará que el sistema elimine el token de sesión y, por ende, nos devuelva a la pantalla de inicio.

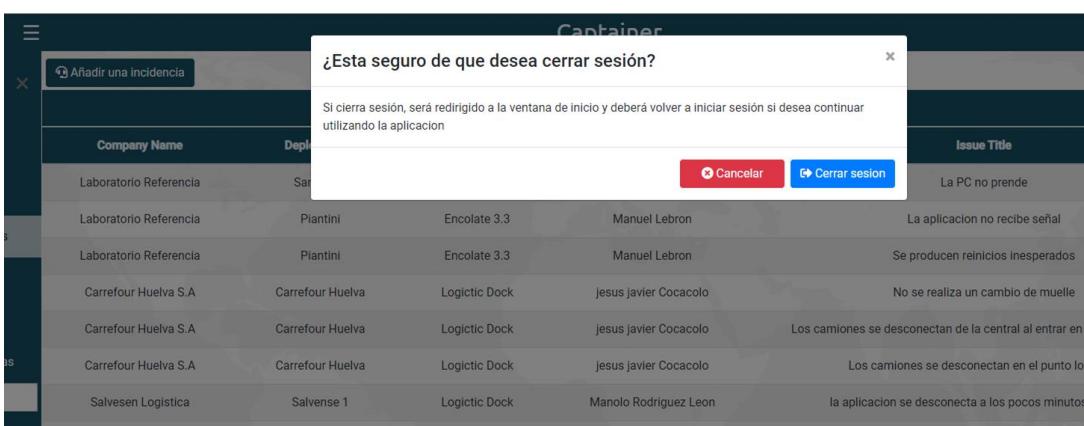


Ilustración 28: Diálogo de confirmación de cerrar sesión

4.7.1. Diseño Responsive

Una de las mejoras sustanciales que hemos tenido que ir solventando al mismo tiempo que desarrollábamos la aplicación ha sido el diseño 'Responsive' ya que, por cada componente diseñado, era necesario atender a la usabilidad desde distintos dispositivos. El diseño web Responsive o adaptativo **es una técnica de diseño web** que busca la correcta visualización de una misma página **en distintos dispositivos**, desde ordenadores de escritorio a tablets o móviles. Se trata de redimensionar y colocar los elementos de la web, de forma que se adapten al ancho de cada dispositivo, permitiendo una correcta visualización y una mejor experiencia de usuario. El diseño Responsive **permite reducir el tiempo de desarrollo y evita los contenidos duplicados**. Una de las ventajas de usar Angular y Bootstrap es que ambas plataformas contemplan el diseño Responsive como algo que debe ser nativo e integrado en sus características, luego por este lado hemos ahorrado mucho tiempo y costes, debido a que muchos de nuestros componentes visuales se ayudan de la tecnología de Bootstrap. De forma paralela, para aquellos componentes visuales que han necesitado un rediseño adicional, con el fin de adaptarse a distintas pantallas del dispositivo, se han empleado códigos 'media-queries' de CSS3. Esta funcionalidad de CSS nos permite cambiar el estilo y las características de un componente visual en función de, por ejemplo, el ancho de pantalla que en ese momento este visualizando nuestra aplicación, redimensionando elementos, ocultándolos o añadiendo otros, de modo que el diseño no comprometa la usabilidad en ningún momento.

A continuación, vamos a hacer un viaje a lo largo de las primeras fases de desarrollo para que podamos ver cómo ha evolucionado el diseño de nuestra aplicación, y en qué momento comenzó a surgir la necesidad de que nuestro diseño fuese Responsive:

- 1. Primera versión de la aplicación:** En nuestra primera versión de la aplicación (Alpha, v1.0), el diseño no era Responsive y, por tanto, a la hora de visualizar el contenido, si bien en una resolución igual a la del entorno de desarrollo (1920x1080) se visualizaba correctamente, la aplicación sufría las consecuencias de cambiar la resolución de pantalla a la hora de ser consumida desde otro dispositivo



Ilustración 29: Pantalla bienvenida primera versión

Al momento de utilizar un dispositivo con una resolución inferior, como un ordenador portátil (1360x768) o un dispositivo móvil estándar (720x1270), cuyo uso común se hace empleando un ratio de pantalla invertido al de un ordenador (9:16 frente a los 16:9 de los ordenadores convencionales, sin tener en cuenta los nuevos formatos ultra-panorámicos: 18:9, 21:9, 9:18, etc.), vemos como la información en pantalla hace que los elementos adquieran unas dimensiones rocambolescas, haciendo difícil su correcto consumo y, lo que es peor, haciendo que ciertos elementos en pantalla como los botones (en este caso, el de inicio de sesión) queden inaccesibles, haciendo que la usabilidad de la aplicación caiga en picado.



Ilustración 31: Vista Portatil 768p

Ilustración 30: Vista móvil

Para solucionar este problema, tuvimos que adaptar la aplicación, de forma que el diseño adoptara las directrices de la filosofía del diseño Responsive.

- Versión rediseñada de la aplicación:** Para el rediseño de la aplicación, se emplearon las distintas clases de 'Bootstrap', como 'mr-auto' o la clase 'collapse', que permite crear un menú "dropdown", y los 'Media-queries' de CSS donde, en función del tamaño en anchura de la pantalla, hemos re-dimensionado algunos elementos a través de sus clases e identificadores únicos, definiendo de distinta forma el tamaño de las fuentes, empleando la tipología 'vw' en lugar de 'px'. El resultado de rediseñar la aplicación en su siguiente versión fue el siguiente:



Ilustración 33: Diseño portátil Responsive

Ilustración 32: Móvil Responsive

3. **Versión actual:** El desarrollo póstumo desde la primera versión rediseñada ha permitido pulir los detalles del diseño Responsive. Actualmente, la aplicación goza de diseño adaptativo en todas sus ventanas y la estética ha evolucionado, de modo que dicho diseño se acerca mucho más a una futura versión definitiva:

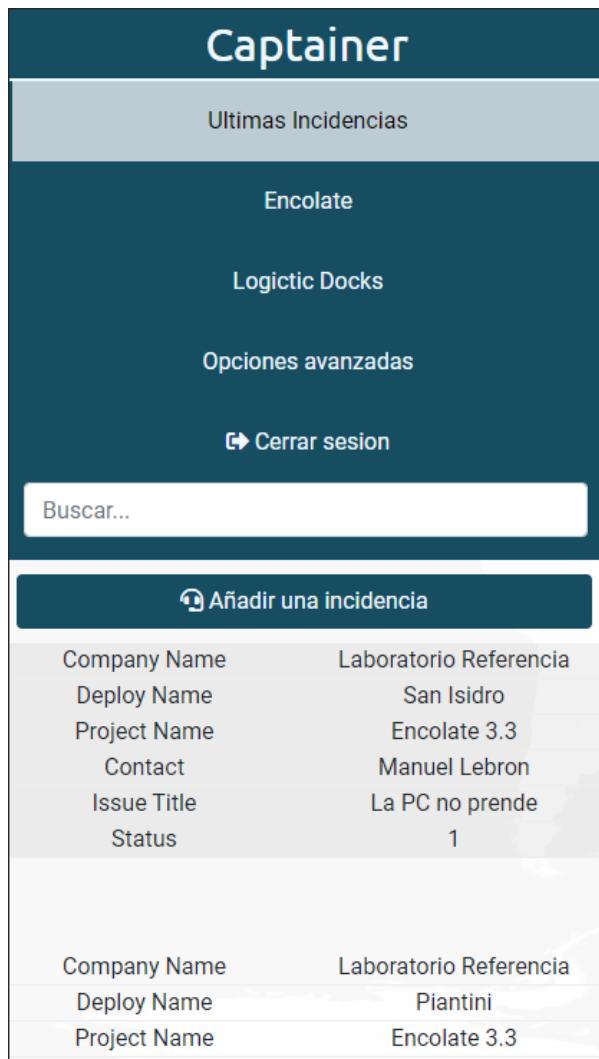


Ilustración 34: Ventana de administración desde móvil

The screenshot shows the Captainer mobile application's incident reporting form. At the top is a dark blue header with the word "Captainer" in white. Below it is a light grey form area with the title "REPORTAR NUEVA INCIDENCIA:" in bold. The form consists of several input fields: "Titulo de la incidencia" (Title of the incident), "Solucion al problema" (Solution to the problem), "Solucion a la incidencia" (Solution to the incident), "Observaciones" (Observations), and another "Observaciones" field. Below these is a "Cerrado por:" (Closed by:) field, followed by a "Cerrado por:" field for the closure reason. Further down are fields for "ID de implantacion:" (ID of the implementation) and "ID Implantacion" (ID of the implementation). At the bottom are two buttons: a blue "Añadir" (Add) button with a plus sign icon, and a red "Limpiar" (Clear) button with a trash can icon.

Ilustración 35: Formulario de incidencias desde móvil

4.8. Lógica de la aplicación

Finalmente, concluiremos el apartado "Diseño" hablando de la lógica de nuestra aplicación, refiriéndonos a la lógica del Front-End, exclusivamente (ya que hemos hablado anteriormente de cómo se construye la lógica en el Back-End). Como ya sabemos, la lógica en Angular es llevada a cabo por los ficheros 'TypeScript' el cual, como ya hemos mencionado, se trata de un 'superset' de 'Javascript'. Estos ficheros desarrollan los diferentes roles del programa, como pueden ser la navegación entre ventanas, las peticiones HTTP, las operaciones de filtrado, etc. No todas las clases TypeScript se comportan igual. Existen varias clases que comparten sintaxis, pero que forman parte de estructuras muy diferenciadas dentro de la aplicación. Estas pueden ser: componentes, directivas, servicios, tuberías ('pipes'), módulos, etc.

Para diseñar la lógica de la aplicación, debemos comenzar por el principio. Las pautas del desarrollo las trataremos en el siguiente apartado: "Implementación". No obstante, es necesario que entendamos como hemos diseñado los distintos tipos de ventanas y que hay detrás de la funcionalidad de cada una de estas:
Lo primero es que, a la hora de crear un proyecto en Angular, vamos a contar al principio y de primera mano con el fichero app.module.ts, con la siguiente estructura:

```
//1. Importación de módulos, componentes y servicios
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { CommonModule } from '@angular/common';
import { RouterModule } from '@angular/router';
import { HttpClientModule } from '@angular/common/http';
import { AppComponent } from './app.component';
import { HomeComponent } from './home/home.component';
import { LoginComponent } from './login/login.component';
import { AdminComponent } from './admin/admin.component';
import { AuthenticationService } from
'./servicios/autenticacion.service';

@NgModule({
declarations: [
  AppComponent,
  HomeComponent,
  LoginComponent,
  AdminComponent,
],
imports: [
  BrowserModule,
  FormsModule,
  CommonModule,
  HttpClientModule,
  RouterModule.forRoot([
{
  path: 'login',
  component: LoginComponent
}],
{
  path: '**',
  component: HomeComponent
}
])
export class AppModule { }
```

```

        path: 'admin',
        component: AdminComponent,

        canActivate: [RoleGuard],
        data: {
            role: ['admin', 'tec_imp', 'tec_sop', 'tec_fac']
        }
    },
    {
        path: '',
        component: HomeComponent
    },
    {
        path: '**',
        component: HomeComponent
    }
),
],
providers: [AuthenticationService],
bootstrap: [AppComponent]
})
export class AppModule { }

```

Diferenciamos, por una parte, la importación de las clases utilizadas por la aplicación, junto a la ubicación de estas. Por otra, la implementación de dichas clases en cada una de las categorías que entiende el programa: 'import', 'declarations', 'providers', 'bootstrap'. Por último, la clase AppModule, que usaremos en caso de buscar una funcionalidad específica que queramos que sea usada y compartida por todos los elementos incrustados en el módulo.

En nuestros componentes Angular colocaremos en primer lugar, las importaciones de módulos y servicios que vamos a utilizar. Requerimos conocer su ubicación dentro de las librerías de Angular o de nuestro proyecto. Posteriormente, emplearemos el decorador '@Component()' para especificar nuestra plantilla, selector y hoja de estilo del componente. Justo abajo crearemos las variables y objetos globales. Posteriormente, exportaremos la clase del fichero Typescript, donde se encontrarán las funciones y métodos de la lógica del componente. Dos de estos métodos son el método 'constructor()' y el método 'onInit()'. La principal diferencia entre ambos, es que en el constructor instanciamos los objetos que usará el componente y ejecutará la lógica que queremos que se lleve a cabo antes de cargar la página, mientras que todo aquello que se encuentre en el método onInit() se ejecutará una vez que la página haya cargado. Por ello inyectamos los servicios en dicho método, para evitar que el código que infla las directivas en la plantilla cargue antes que la propia plantilla.

Por tanto, atendiendo a lo ya mencionado, estructuraremos nuestros componentes de la siguiente manera, separando importaciones del separador, de las var globales y la clase:

```
import { Component, OnInit, TemplateRef } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';
import { AuthenticationService } from
'./servicios/autenticacion.service';
import { AuthGuardService } from '../servicios/auth-guard.service'
import { User } from '../modelos/user';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent {

  user = new User();
  name: string = "";
  password: string = "";

  constructor(
    private router: Router,
    private authenticationService: AuthenticationService,
    private authGuard: AuthGuardService
  ) {}

  onSubmit() {
    this.loading = true;
    console.log(
      "name (" + this.name + ")"
      + "password (" + this.password + ")"
    );
    this.authenticationService.login(this.name,
    this.password).subscribe(
      data => {
        this.router.navigate(['admin']);
      },
      error => {
        console.log("Se ha producido un error al ingresar");
      });
  }
}
```

5. Implementación

5.1 Introducción

Hasta ahora, hemos visto el análisis y diseño de la aplicación. Hemos viajado por el proceso de investigar cuales son las reglas y paradigmas a seguir, así como las tecnologías a emplear, y como se estructuran estas para su correcto uso. Es hora de mostrar el proceso de creación de la aplicación siguiendo las pautas de diseño llevadas a cabo en el espacio de trabajo. Abarcaremos todo el proceso, desde como hemos creado la base de datos, hasta como hemos desplegado la aplicación, siguiendo como ya hemos comentado, el orden lógico de desarrollo. Vamos a partir de la base de que el entorno de trabajo está preparado para el desarrollo, con todos los Frameworks, editores y entornos de desarrollo integrado ya instalados y correctamente configurados. En dicho entorno, se han realizado pruebas de diversos tipos: levantar un servidor local con Xampp, crear un API para pruebas con Lumen y lanzar peticiones desde Postman a la base de datos, desarrollar una aplicación de prueba con Angular que muestra una lista estática de objetos, etc. Todo ello antes de comenzar el proyecto, con el fin de aprender los conceptos básicos de ambas tecnologías y dejar el entorno preparado para la aplicación final. Es importante dejar todo preparado, ya que desarrollar el producto final con un entorno que no ha sido preparado correctamente puede provocar que, en una fase concreta del desarrollo, podamos echar en falta alguna herramienta o configuración, siendo irreversible el proceso en algunos casos. Es decir, existen ocasiones en las que no podemos dar marcha atrás, ya que la inclusión de ciertos componentes en una fase avanzada puede crear inestabilidad o incoherencia de datos. Por ello es imprescindible que hagamos un análisis de todo lo que debemos preparar y, en función de ello, seguir los pasos oportunos.

Otra cuestión a tener en cuenta es que, a pesar de tratar este apartado con el debido orden y siempre atendiendo a que sea lo máximo didáctico posible, la aplicación final no ha sido desarrollada de una sola vez, sino que parte de las pruebas del entorno que hemos comentado, así como otras pruebas posteriores de cómo funcionan las herramientas del Front-End, han sido empleadas para el desarrollo final; ya que resultaba más cómodo reutilizar fragmentos de código que volver a desarrollar una funcionalidad de cero. Así ha ocurrido con el listado de objetos, el componente que realiza peticiones GET/POST y la ventana de inicio de sesión y validación de usuario. Con esto queremos decir que, en realidad, la construcción de Front-End no es necesariamente lineal, ya que Angular es un sistema modular y, por tanto, los componentes pueden desarrollarse por separado sin ningún problema, siempre y cuando sepamos como indexar los nuevos componentes debidamente. No obstante, y como recomendación, lo ideal es desarrollar la aplicación de una vez, creando los componentes, directivas y servicios oportunos al momento de surgir la necesidad de contar con alguno de ellos.

5.2 Codificación de las diferentes capas

Vamos a tratar de hacer que este apartado sea lo más explicativo posible, procurando que todos los pasos queden bien resueltos, con la finalidad de que el lector comprenda los pasos esenciales necesarios para poner en marcha un proyecto de tales dimensiones. Lo primero en este caso, será dividir el diseño de la aplicación y puesta en marcha en varios apartados, de modo que sea más fácil diferenciar las siguientes fases del desarrollo de dicha aplicación. Cabe señalar nuevamente que **este proyecto sigue aún en fase de desarrollo**. Es decir, el proyecto presentado en este documento es una versión beta, funcional en algunos casos, pero que no se encuentra en su versión final, debido al tiempo limitado del desarrollo. En cualquier caso, el desarrollo del proyecto se encuentra en una fase muy avanzada y, por tanto, podemos enumerar todos los pasos seguidos hasta la aplicación, aventurándonos incluso, a predecir cuales son los pasos restantes hasta la versión estable.

5.2.1. Fase Inicial: Configuración y despliegue del Back-End (base de datos y ORM en el servidor):



Ilustración 36: MySQL

Lo primero es crear la base de datos en el servidor. Dicha base de datos es una base de datos MySQL. Empleamos la herramienta SQL Workbench, con la opción ‘create new Schema in the connected server’. Nos aseguramos previamente que tenemos conexión con nuestro servidor (el usuario y contraseña por defecto serán ‘root’ y ‘’, respectivamente). Una vez creada y nombrada, tendremos que tener muy en cuenta los siguientes datos:

- Nombre de la base de datos
- Dirección y puerto del servidor
- Usuario y contraseña de acceso

A continuación, crearemos nuestro Back-End, utilizando nuestro Microframework Lumen en su versión 5.5. Previamente, lo instalamos siguiendo los pasos que marcan la API de Lumen en su web oficial:

Previamente. Nuestro equipo debe contar con la herramienta ‘Composer’, descargada desde aquí, e instalada en nuestro equipo. Composer es un administrador de dependencias para PHP, que nos permite descargar paquetes desde un repositorio para agregarlo a nuestro proyecto. Por defecto, se agregan a una carpeta llamada /vendor. De esta manera, evitamos hacer las búsquedas manualmente, siendo el mismo Composer el que se encarga de actualizar las dependencias que hayamos descargado, por una nueva versión existente. Composer, al igual que Node.JS, hará que podamos emplear órdenes y comandos que por defecto no contamos en nuestro equipo.

Una vez instalada la herramienta Composer, hacemos lo siguiente:

En el directorio en el que queramos desplegar el Back-End: Ejecutamos una ventana de comandos, y ordenamos mediante Composer lo siguiente:

```
composer create-project --prefer-dist laravel/lumen nombre_elegido
```

Se creará una carpeta de proyecto con el nombre que hallamos elegido, en la que se encuentra todos los ficheros de Lumen, listos para ser usados. Si queremos levantar el servicio de Lumen para comprobar que la creación ha sido exitosa, utilizamos el siguiente comando:

```
php -S localhost:8000 -t public
```

En caso de estar trabajando en local. Normalmente, en las pruebas iniciales, se suele trabajar en local. Posteriormente, la aplicación va a un servidor dedicado, donde la dirección es la dirección IP de la máquina y el puerto no necesariamente es el puerto 8080.

Dentro de los ficheros de la carpeta, se encuentra la carpeta '/vendor'. En dicha carpeta se encuentran las funcionalidades de Lumen, entre las que se encuentran las más importantes: '**Artisan**'. Si nos movemos al directorio raíz del proyecto Lumen, podemos utilizar los comandos necesarios para emplear este gestor de comandos.

Antes de nada, es muy importante este paso. En el interior de la carpeta creada, existe en la raíz un archivo, cuyo nombre acaba con ".env.example". Dicho fichero contiene la **configuración de comunicación con nuestro servidor**. Por tanto, es altamente necesario y modificarlo para que se adapte a nuestro servicio. El fichero contiene, ya modificado, la siguiente estructura:

```
APP_ENV=local
APP_DEBUG=true
APP_KEY=
APP_TIMEZONE=Europe/Madrid

DB_CONNECTION=mysql ← Conexión con MySQL
DB_HOST=127.0.0.1 ← Dirección del servidor de la base de datos
DB_PORT=3306 ← Puerto del servidor de la base de datos
DB_DATABASE=captainerDB ← Nombre de la base de datos
DB_USERNAME=root ← Usuario de la base de datos
DB_PASSWORD= ← Contraseña de la base de datos (vacía)

CACHE_DRIVER=file
QUEUE_DRIVER=sync

JWT_SECRET=i3XS6vI2LiQjExRKR3CbsK5W9KagbKvC
JWT_TTL=600
```

Por defecto vendrán otros valores. Nosotros hemos adaptado las líneas señaladas en función de nuestras necesidades. Es importante que, tras hacer los cambios, **guardemos el fichero como .env**, y no como .env.example.

Una vez estructurado el ORM, lo siguiente será crear los 'seeders' y las migraciones. En anteriores apartados, definimos cuales eran sus funciones y características. Pues bien. Dichos ficheros podemos crearlos manualmente con Visual Studio Code u otro editor de texto, pero lo ideal es hacerlo con las herramientas que nos ofrece Artisan, aunque luego tengamos que modificarlos y adaptarlos a nuestras necesidades. Veamos como hacemos el procedimiento:

Desde la ventana de comandos, y en el fichero raíz, empleamos los siguientes comandos:

```
php artisan make:migration create_users_table --create=user
```

Con este comando, crearemos el fichero de migración “[fecha_actual]_create_users_table”, el cual tendrá la siguiente estructura:

```
use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Migrations\Migration;

class CreateUsersTable extends Migration {
    public function up() {
        $schema = \Illuminate\Support\Facades\DB::connection()
            ->getSchemaBuilder();
        $schema->blueprintResolver(function ($table, $Callback) {
            return new \App\Extensions\BlueprintExtension($table, $Callback);
        });
        $schema->create('user', function (\App\Extensions\BlueprintExtension $table) {
            $table->increments('id');
            $table->string('name', 60)->nonnullable();
            $table->string('email', 50)->nonnullable();
            $table->string('password', 200)->nonnullable();
            $table->char('role')->nonnullable();
            $table->integer('estatus')->nonnullable();
            $table->string('photo', 200);
            $table->timestamps();
        });
    }
    public function down() {
        Schema::dropIfExists('user');
    }
}
```

En dicho fichero, especificamos el nombre de la tabla ('user'), las columnas de la tabla (id, nombre, email, contraseña, rol, estado, foto, y 'Timestamp' que creará dos columnas: fecha de creación y fecha de actualización) con sus respectivas propiedades (tipo de datos a almacenar, si es nulo, tamaño máximo) y, además, crearemos una función donde, si la tabla de usuarios existe, la borre y vuelva a crearla, con la finalidad de no crear redundancia de datos.

Haremos el mismo procedimiento para las tablas restantes. Recordemos que contamos con un total de 16 tablas y que, por ende, necesitamos hacer un fichero para cada una de ellas.

En algunos ficheros que contienen las relaciones entre varias tablas, como por ejemplo en el de la tabla 'user_project', encontraremos, adicionalmente, las siguientes líneas:

```
//Fks
$table->foreign('user_id')
    ->references('id')
    ->on('user')
    ->onDelete('cascade');
```

```
$table->foreign('project_id')
->references('id')
->on('project')
->onDelete('cascade');
```

Donde especificamos dos relaciones en dicha tabla: una con la tabla ‘user’ y otra con la tabla ‘project’. Hay varias formas de actuar. La primera y más rápida es crear todas las tablas y relaciones en el mismo fichero. No obstante, este método es más susceptible a errores, por lo que nosotros seremos más metódicos y crearemos un fichero por cada tabla. Concretamente, crearemos primero aquellos ficheros cuyas tablas no posean relaciones. Posteriormente, crearemos aquellas tablas con una clave foránea y una relación y, por último, crearemos aquellas tablas como esta última, con dos claves foráneas y dos relaciones. La cuestión es que, a la hora de desplegar las tablas, no podemos crear una relación entre dos tablas, si una de las dos aún no ha sido creada. Por ello, es importante que antes de crear los ficheros de migraciones, decidamos cuál de ellos es necesario crear antes, debido a que Artisan creará las tablas por orden ascendente de creación del fichero de migración.

Una vez creados todos los ficheros de migraciones, lo que debemos hacer es despegarlos en el servidor. Si hemos configurado correctamente dichos ficheros de migración, así como el fichero que comunica la conexión con el servidor, deberíamos ser capaz de observar que se han cargado las tablas en la base de datos, mediante el siguiente comando:

```
php artisan migrate
```

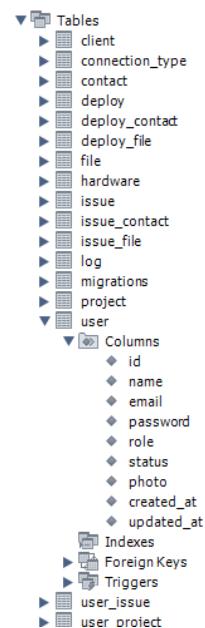
```
migrations
├── .gitkeep
└── 2017_01_30_180420_create_user_table.php
    ├── 2019_03_21_074708_create_file_table.php
    ├── 2019_03_21_074714_create_client_table.php
    ├── 2019_03_21_134638_create_project_table.php
    ├── 2019_03_21_134641_create_connection_type_table.php
    ├── 2019_03_21_134758_create_deploy_table.php
    ├── 2019_03_21_135009_create_hardware_table.php
    ├── 2019_03_21_135042_create_issue_table.php
    ├── 2019_03_21_135311_create_contact_table.php
    ├── 2019_03_22_074511_create_log_table.php
    ├── 2019_03_22_074551_create_user_project_table.php
    ├── 2019_03_22_074740_create_issue_contact_table.php
    ├── 2019_03_22_074826_create_user_issue_table.php
    ├── 2019_03_22_074858_create_issue_file_table.php
    ├── 2019_03_22_074953_create_deploy_file_table.php
    └── 2019_03_22_075013_create_deploy_contact_table.php
```

Ilustración 37: Ficheros de migración de Lumen

Si accedemos desde un gestor de bases de datos, comprobaremos que nuestras tablas se encuentran correctamente creadas y con todas las columnas correctamente creadas y funcionales. De hecho, podemos comprobarlo, accediendo a cada una de ellas.

La cuestión es que, con el fin de desarrollar la aplicación, algunas tablas no pueden estar vacías. Si deseamos comprobar la validación de usuario, necesitamos que la tabla ‘user’ al menos cuente con un registro de usuarios. Y casos similares ocurren con tablas como ‘issue’ o ‘deploy’, donde para mostrar información, necesitamos que las tablas tengan registros. Aquí es donde entra en juego los ‘seeders’.

Se trata de ficheros cuya función es injectar registros predefinidos en las tablas existentes, con el fin de que no estén vacías. Al igual que ocurre con las migraciones, existe un ‘seeds’ para cada tabla. Veamos un ejemplo, con el ‘seeder’ de la tabla ‘user’:



*Ilustración 38:
Tablas de la BBDD*

```

class UserTableSeeder extends Seeder {

    public function run() {

        DB::table('user')->insert([
            'name' => 'administrador',
            'email' => 'admin@prismasg.com',
            'password' => app('hash')->make('123456'),
            'role' => 'admin',
            'estatus' => '1',
            'photo' => 'sin_foto'
        ]);

        DB::table('user')->insert([
            'name' => 'Responsable CL',
            'email' => 'admin@prismasg.com',
            'password' => app('hash')->make('123456'),
            'role' => 'admin',
            'estatus' => 1,
            'photo' => 'sin_foto'
        ]);
    }
}
  
```

Para crear un ‘seeder’, lo hacemos de forma similar a como lo haríamos con un fichero de migración:

```
php artisan make: seeder UsersTableSeeder
```

Donde creamos el fichero de nombre “UsersTableSeeder”. Si queremos ejecutarlo, tenemos que ejecutar el siguiente código:

```
php artisan db:seed
```

Por cada ‘seed’ creado, tendremos una tabla con filas creadas.

id	name	email	password	role	status
1	administrador	admin@prismsg.com	\$2y\$10\$EIdv8UuqemXSLUctUsm1uyxWpR0Ube7/krpTdxMCVjBXxAa7SyYW	admin	1
2	Responsable CL	admin@prismsg.com	\$2y\$10\$96oREhqeyNCGktvIH0sbOC62aAQzy1jpyRymw4E73v7arWrwzP0	admin	1

Ilustración 39: Filas de la tabla ‘user’

Por último, pero no menos importante, queda la ardua tarea de diseñar los **modelos y controladores** de las tablas. Debemos de crear un modelo para cada tabla existente en la base de datos. Para ello, nos dirigimos al directorio raíz de Lumen y accedemos a la carpeta ‘/app’. No vamos a generar los modelos mediante comandos con Artisan, como hemos hecho con anteriores archivos, por lo que lo recomendable es hacer estos pasos con un editor de código avanzado, como PHPStorm o Visual Studio Code, que permiten abrir un directorio y movernos en torno a él desde la propia aplicación, haciendo muy fácil abrir múltiples ficheros, crear nuevos ficheros, etc. Desde la propia interfaz.

Crearemos, como ya hemos dicho, un modelo para cada tabla. Usando de referencia la tabla ‘issue’, su modelo debe tener la siguiente estructura:

```
namespace App;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Support\Facades\DB;

class Issue extends BaseModel {
    protected $table = 'issue';
    public $timestamps = true;

    public static $validationArray = [
        'title' => 'required|string|max:200',
        'solution' => 'required|text',
        'observations' => 'required|text',
        'estatus' => 'required|integer',
        'closed_by' => 'required|integer',
        'deploy_id' => 'required|integer',
    ];
    protected $fillable = [
        'title', 'solution', 'observations', 'estatus', 'closed_by',
        'deploy_id'
    ];

    public static $searchArray = [
        'title', 'solution', 'observations', 'estatus', 'closed_by',
        'deploy_id'
    ];
}
```

Donde definimos las columnas y tipos de valores que tienen en ellas, así como funciones que luego servirán para insertar o modificar valores en dichas tablas.

BaseModel es una clase de la que extiende el modelo que hemos predefinido anteriormente, y que posee propiedades comunes con todos los modelos:

```
class BaseModel extends Model {
    public static $validationArray = [];
    public static $searchArray = [];
    public static function getTableName() {
        return with(new static)->getTable();
    }
    public static function getSearchArray() {
        dd((new self())->fillable);
        return (BaseModel::$searchArray == [])
            ? (new self())->fillable
            : BaseModel::$searchArray;
    }
}
```

La importancia de los **modelos**, es que, a la hora de operar sobre cada una de las tablas, los modelos nos facilitarán enormemente las tareas del CRUD, ya que en lugar de utilizar consultas SQL, dichas operaciones serán llevadas a cabo por el framework, mientras que nosotros solo nos tenemos que limitar a insertar métodos que vienen predefinidos en Lumen y que utilizan dichas consultas. Es, digamos, una capa de abstracción que convierten una consulta más elaborada en una o dos sencillas líneas de código, más entendible y eficiente, ya que no requerimos escribir los nombres de la tabla y sus columnas cada vez que queramos hacer una consulta, sino que el framework usará la clase de la tabla para acceder a dichos nombres para hacer las consultas “por detrás”. Es una de las mayores ventajas de este framework.

Hecho esto, solo nos queda crear los controladores, antes de comenzar con nuestra aplicación en el Front-End. Al contrario que los modelos, no es necesario que creamos un controlador por cada tabla de la base de datos. De hecho, podemos tener un solo controlador para todo, un controlador para cada una, un controlador para varias tablas, etc. Lo que haremos será crear controladores para aquellas tablas que vamos a utilizar de forma activa, como lo son ‘user’, ‘issue’ y ‘deploy’.

Un **controlador** es un fichero .php que extiende de la clase ‘Controller’ y en el que desarrollamos la lógica de programación del ORM. Un ejemplo es el controlador de usuario:

```
namespace App\Http\Controllers;

use App\User;
use Illuminate\Support\Facades\DB;

class UserController extends BaseCrudController {
    public function __construct(){
        parent::__construct(User::class);
    }

    public function check() {
        $lista_Dashboard = DB::table('user')
            ->select('user.id')
            ->get();
        if (!empty($lista_Dashboard)) {
            return response()->json([
                'response' => [
                    'code' => 200,
                    'message' => "Se ha conectado",
                ],
            ]);
        }
    }
}
```

Donde encontramos el método ‘check()’, que valida si el usuario se encuentra conectado, mediante un ‘Select’ al id de usuario de la tabla ‘user’. Dicho id lo consigue mediante el modelo de usuario que creamos anteriormente, por lo que, de un vistazo, vemos por un lado como se emplean los modelos en los controladores y, por otro lado, la estructura final de dicho controlador.

Dicho controlador extiende de otro controlador, llamado ‘BaseCrudController’, en el cual contamos con todos los métodos utilizados en los controladores para los CRUD de la aplicación.

Veamos un ejemplo de su estructura (no puedo colocar el código íntegro, ya que posee un total de 256 líneas de código, que se traducirían en 5 páginas de código en este documento).

Nota: Hay que aclarar una cuestión: El código generado en esta clase en concreto, aunque también ocurre en otros ficheros del proyecto, **no se encuentran desarrollado por mí**, sino que el grupo de programadores del entorno **me han facilitado** este fragmento del código para que pueda utilizarlo en el proyecto. No se trata de que dicho proyecto sea íntegramente desarrollado por mi persona en todos los sentidos. No intentamos reinventar la rueda. Tampoco quiero que sirva de precedente ni de lugar a pensar que el proyecto se encuentra desarrollado por más de una persona. Se trata de dejar claro que **este tipo de funciones son complejas para ser desarrolladas por una sola persona**, de una sola vez, y que en este sentido he contado con ayuda y material para continuar con el proyecto.

```

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Validation\ValidationException;

class BaseCrudController extends Controller {
    public $model;
    private $table;
    public $crudID = 'id';

    public function __construct($model) {
        $this->model = $model;
        $this->table = $model::getTableName();
    }

    public function selectData($query){
        return $query; // función de listar
    }

    public function store(Request $request){
        // Acción para insertar
    }

    public function update(Request $request, $id){
        // Función para modificar
    }

    public function delete(Request $request, $id){
        // Función para eliminar
    }
}

```

5.2.2. Primera fase de la aplicación: Creación del proyecto Angular y sus componentes Básicos

Una vez que ya tenemos los cimientos de la aplicación desarrollados, toca llevar a cabo el desarrollo del Front-End. Comenzaremos con lo principal, que será **crear la aplicación Angular**. Para ello y como comentamos al principio de apartados anteriores, debemos instalar Node.JS y AngularCLI. Tal y como comentamos en la introducción, vamos a obviar este paso y a considerar que estas herramientas ya se encuentran instaladas e integradas a la perfección en el sistema. Como ya comentamos, además, ya hemos hecho varias pruebas con aplicaciones Angular en el equipo, por lo que partimos de la base que **todo se encuentra instalado y configurado correctamente**.

Creamos un nuevo proyecto en el directorio en el que queremos desplegar la aplicación, mediante el comando:

```
ng new captainer
```

Se creará un proyecto Angular con el nombre “captainer” (nombre corporativo de la aplicación), en el que debemos especificar si queremos que nos cree el **fichero de rutas**, al cual le decimos ‘No’, y nos pregunta qué **tipo de hojas de estilos** queremos emplear. En nuestro caso, **seleccionamos ‘CSS’** y damos a continuar. Solo queda esperar a que las carpetas del proyecto terminen de crearse. Una vez creado, tendremos un proyecto por defecto de Angular. Por ello, comenzaremos con la lógica inicial del módulo principal y su componente principal.

Antes de comenzar con la estructura final del programa, nos detenemos a pensar que va a ser lo primero que debemos dejar preparado. **En nuestro caso ha sido la ventana de inicio de sesión**, ya que consideramos que la validación era un tema esencial para la aplicación, y al mismo tiempo nos permitiría abarcar un poco de todo, desde servicios y ‘binding’, hasta diseño CSS y disposición de los métodos y empleo de las variables.

Pensamos en desarrollar dicha ventana en el componente principal (app.component.ts) de la aplicación, pero luego pensamos que **lo más conveniente era crear un componente hijo específico para dicha función**, y un servicio de autenticación que se encargara de todo lo relacionado con los verbos HTTP.

Por tanto, **creamos el componente ‘login’ y el servicio ‘authentication.service.ts’**.

Para crear un componente, empleamos el siguiente comando:

```
ng generate component nombre_componente
```

se generará una carpeta con el nombre del componente dentro de ‘/app’, que contiene un fichero typescript (‘.ts’), un fichero HTML y una hoja de estilos CSS.

Para crear un servicio, empleamos un comando similar al anterior:

```
ng generate service nombre_servicio
```

Y se creará un servicio (‘.ts’). Es importante que tanto el componente como el servicio estén bien localizados dentro del proyecto ya que, a la hora de importar un componente o servicio en el módulo, debemos conocer su ubicación en el directorio de Angular.

Si creamos estos ficheros de forma manual **serán iguales de funcionales que si empleamos los comandos para ello**. Sin embargo, no solo es un trámite de mera comodidad. Cuando creamos un tipo de funcionalidad **mediante comandos, el propio sistema añade las clases creadas** (tanto los servicios como el typescript del componente contienen una clase que se exporta al módulo) **al módulo** en cuyo seno se hallen estos elementos. Si hacemos la operación manualmente, seremos responsables de tener que añadir estos ficheros a las importaciones del ‘app.module.ts’, puesto que en caso contrario, el sistema dará error, ya que manejamos componentes y servicios **no declarados** en la aplicación.

Ya con el componente creado, lo primero que hacemos para desarrollar un componente en su totalidad, y que esto sirva para el resto de componentes, es ir probando las funcionalidades progresivamente. **Un error** que cometía, y que todo el

mando acaba haciendo al inicio de la programación en Angular, **es querer probar que funciona todo lo que ha desarrollado al mismo tiempo.** Y esto suele desembocar en fracaso, ya que el mínimo error, como una etiqueta mal cerrada, puede producir que la aplicación no pueda compilar correctamente y, por tanto, no pueda ser visualizada, por lo que no podremos ver el error de primera mano y tendremos que recurrir a la consola de errores, con todo lo que ello conlleva. Si hemos cometido 5 errores de programación o diseño al mismo tiempo, la consola de errores arrojará una cantidad de líneas tal que **no resultará tan fácil encontrarlos y corregirlos todos.** Resulta mucho más práctico seguir un guion para proceder. En mi modesta opinión, **recomiendo seguir estos pasos** para el desarrollo de todos los componentes de la aplicación, de forma rigurosa al menos durante las primeras semanas de desarrollo, aunque cuando cojamos soltura nos saltemos un par de pasos o combinemos pasos:

Puntos para el desarrollo de un componente:

- **Desarrollar el diseño de la plantilla HTML, sin estilos y sin directivas Angular.** Compilar y probar solo como se visualiza la plantilla y que no se produzcan errores (pulsar sobre F12 o clic derecho e inspeccionar elemento, y seleccionar ‘consola’).
- **Desarrollar los estilos de la plantilla en la hoja de estilos CSS** hasta que el diseño quede tal y como deseamos que se vea. Probamos que todo se vea a nuestro gusto.
- Una vez el diseño desarrollado, **comenzar con el diseño de los métodos de la lógica del componente en Typescript** que interactúan con la interfaz de la aplicación, pero **no interactuaremos aun** con la interfaz. Es decir. Si queremos implementar un método que recoja de una entrada por teclado el nombre de usuario y la contraseña (mediante una etiqueta <input>, por ejemplo), diseñaremos el método tal y como deseamos que se estructure, pero no llegaremos a producir un resultado, entendiéndose como resultado aquella parte del código que **produce un cambio o un evento**, como un ‘return’, un ‘navigate’ o arrancar un método. Dejamos el desarrollo aquí y pasamos al siguiente punto.
- **Introducimos las directivas ‘ngModel’ en la plantilla HTML del componente.** Probamos nuevamente que el sistema no arroje ningún error. Como ya hemos mencionado en anteriores capítulos, esta directiva es una de las herramientas de Angular y, concretamente, **nos permite recoger valores introducidos por dispositivos de entrada, y almacenarlos en una variable.** Previamente, antes de introducir las directivas en la plantilla, habremos **creado una serie de variables, tantas como elementos queremos recoger de la interfaz.**
- **Terminamos con el desarrollo del método o los métodos en el typescript, pero no desarrollamos aun el resultado o evento.** Esto quiere decir que si tenemos ya desarrollado el método que recoge los valores por teclado del usuario y la contraseña, no hagamos que intente validarse en el sistema (aparte que aún queda desarrollar el servicio) sino que, en lugar de ello, **tratemos de verificar antes que en efecto dicho método desempeña su función correctamente.** Esto se consigue **colocando un ‘console.log()’ que muestre por consola aquello que deseemos visualizar.** En este caso, el método recogería los valores del usuario y

contraseña y haría la petición POST. Para hacer dicha petición, necesitamos **recoger estos valores** y enviarlos como un objeto al método observable. Lo que haremos será colocar en el código: ‘console.log([usuario + contraseña])’ para que el programa muestre si la directiva está recogiendo los valores correctamente:

```
export class LoginComponent {
  user = new User();
  name: string = "";
  password: string = "";

  onSubmit() {
    console.log("name (" + this.name + ")" + "password (" +
      +this.password+ ")");
  }
}
```

- **Desarrollar el servicio de autenticación de usuario:** Una vez desarrollado el método, el siguiente paso es crear el servicio de autenticación. En él, vamos a **crear el método POST** que lance la petición al Back-End y se traiga consigo el **Token de validación de usuario**, en caso de que la validación sea correcta. Veamos dicho ejemplo para estudiar cómo funcionan nuestros servicios en Angular, ya que el resto de servicios funcionan de manera muy similar.

Nota: El objetivo de esta documentación es meramente didáctico. No vamos a exponer cada uno de los códigos de la aplicación. Simplemente mostramos estos fragmentos como ejemplo, para que sea más sencillo entender nuestras aclaraciones acerca de cómo efectuamos la implementación de la aplicación.

```

import { User } from '../modelos/user';

@Injectable({ providedIn: 'root' })

export class AuthenticationService {

    jwtHelper: JwtHelperService = new JwtHelperService();
    private currentUserSubject: BehaviorSubject<User>;
    public currentUser: Observable<User>;

    constructor(private http: HttpClient) {
        this.currentUserSubject = new
        BehaviorSubject<User>(JSON.parse(localStorage.getItem('currentUser')));
        this.currentUser = this.currentUserSubject.asObservable();
    }

    login(name: string, password: string) {
        return
        this.http.post<any>('http://localhost:80/www/captainer/src/test/api-
        lumen/public/api/auth/login', { name, password })
        .pipe(map(user => {
            if (user && user.token) {
                localStorage.setItem('currentUser', JSON.stringify(user));
                this.currentUserSubject.next(user);
            }
            console.log(user.token);
            console.log(JSON.stringify(user))
            return user;
        }));
    }
}

```

Lo que tenemos en nuestro servicio no es más que un método, llamado ‘**login()**’, el cual emplea **un usuario y una contraseña que recibe por parámetros y, mediante una URL conocida**, la cual es la que recibe la petición POST de inicio de sesión en el Back-End, **y mediante dichos usuario y contraseña recibidos, realiza la petición POST (véase el apartado 4.5.1)**.

- **Terminar la lógica del componente para que realice la función esperada.**
Comprobar los resultados: Si la petición es correcta (y lo es, puesto que los métodos del Back-End están testeados en Postman), ya podemos terminar el desarrollo de la lógica de nuestro componente:

```

onSubmit() {

    this.authenticationService.login(this.name,
    this.password).subscribe(
    data => {
        console.log("Usuario logueado correctamente");
}

```

```

        console.log( "Usuario correcto" )
    },
error => {
    console.log("Se ha producido un error al ingresar");
});
}
}

```

El siguiente paso es continuar el desarrollo, **creando la página de administración de incidencias**, que será la pantalla principal de navegación de la aplicación. Para ello, repetimos los pasos anteriores de creación de un componente: primero diseñamos la plantilla, luego aplicamos el estilo y, una vez finalizado estos pasos, lo siguiente será que la aplicación, una vez comprobado que el inicio de sesión funciona correctamente, pueda **redirigir a la ventana de administración**. Para ello, debemos implementar un **servicio de enrutado**, lo cual quiere decir que la aplicación debe contar con una serie de direcciones definidas y, mediante un método, **especificar que nos cambie de componente al llevar a cabo una determinada acción**. En nuestro caso, dicho servicio se encontrará incrustado en el fichero del módulo, ‘app.module.ts’. Para cada uno de los componentes de la aplicación (**véase el apartado 4.8**), contaremos con una dirección fija. Por lo que, al indicar que queremos movernos a un componente en específico, el programa cambiará de componente, simplemente haciendo uso de la acción siguiente:

```

import { Router, ActivatedRoute } from '@angular/router';

export class LoginComponent {

    constructor( private router: Router ) {}

    onSubmit() {

        this.authenticationService.login(this.name,
            this.password).subscribe(
                data => {
                    this.router.navigate(['admin']);
    }
}

```

Se importará el módulo ‘Router’, lo **instanciaremos en el constructor** y emplearemos su método ‘**navigate()**’ para acceder, en este caso, al componente ‘**admin**’, que no es otro que el componente de administración de incidencias que hemos creado. Ahora, cuando el usuario ingrese correctamente en el sistema, la función de ‘**CallBack**’ ejecuta el método ‘**navigate()**’ con el parámetro “admin”, por lo que **seremos redirigidos** al componente creado.

En el componente “admin”, crearemos una tabla donde mostraremos las incidencias registradas en la base de datos, junto con otros datos como el nombre de la implantación a la que pertenece, el nombre del proyecto, el nombre del contacto, etc. Una vez comprobado que el método ‘**navigate()**’ nos redirige al componente una vez que el usuario y contraseña introducidos son correctos, el siguiente paso es crear la tabla antes mencionada.

Para ello, usamos la directiva ‘**ngFor**’ (**véase los apartados 4.3 y 4.6.1 acerca de esta directiva**). Si hemos seguido los “puntos para el desarrollo de un componente”, nos habremos quedado justo en el desarrollo de la plantilla y el estilo. Ahora debemos **generar la lógica del componente, el servicio que envía las peticiones GET e incrustar las directivas correspondientes en el diseño**.

En este caso, vamos a probar como ahorrarnos un fichero de servicio, haciendo que el propio método del componente sea un ‘observable’. Resultará un método más complejo, pero nos permitirá, como ya hemos comentado, prescindir de un fichero más de un componente que solo tiene una determinada acción.

En la lógica del componente, ‘admin.component.ts’, desarrollamos el siguiente método:

```
sites: DiveSite[] = [];

ngOnInit(): void {
  this.http.get('http://localhost/www/captainer/src/test/api-
lumen/public/api/auth/mainadvlist').subscribe((response: DiveSite[]) => {
  headers:
    new HttpHeaders(
    {
      'Content-Type': 'application/json',
      'X-Requested-With': 'XMLHttpRequest',
      'MyClientCert': '', // This is empty
      'MyToken': '' // This is empty
    })
  this.sites = response;
});
}
```

donde:

- ngOnInit() es un método que se ejecuta al iniciar el componente
- this.http.get es un método del módulo HttpClient que envía una petición GET
- response es la respuesta a la petición que recibe con el CallBack
- DiveSite[] es un array de la clase DiveSite, la cual contiene elementos de una consulta personalizada (**véase la función ‘getMainAdvanced()’ del apartado 4.5**), la cual contiene:
 - company_name
 - deploy_name
 - project_name
 - contact
 - issue_title
 - status
- Headers son las cabeceras que necesita la consulta
- Sites es una variable que recibe la respuesta a la petición

Ahora, en la plantilla HTML, utilizamos el valor de la variable ‘sites’ y la directiva ‘ngFor’ para ejecutar la siguiente acción:

```
<tr role="row" *ngFor="let site of sites">
<td role="cell">{{site.company_name}}</td>
<td role="cell">{{site.deploy_name}}</td>
<td role="cell">{{site.project_name}}</td>
<td role="cell">{{site.contact_name}}</td>
<td role="cell">{{site.issue_title}}</td>
<td role="cell">{{site.status}}</td>
</tr>
```

Donde instanciamos una nueva variable, ‘site’, que recibirá el valor de cada elemento de ‘sites’, el cual es un array de objetos ‘DiveSite’. Concretamente, este procedimiento no es más que un **For-Each** que **recorrerá la variable ‘sites’ y por cada posición del array**, guardará dicho valor en ‘site’ para que pueda ser utilizado. En bucle se repetirá una vez finalice **la etiqueta donde se ha iniciado la directiva, y tantas veces como elementos tenga** la variable ‘sites’.

Este mismo desarrollo lo emplearemos para cada componente que **contenga una tabla de información y una petición GET** tras de ella.

5.2.3. Segunda fase de la aplicación. Creación de botones, componentes y servicios para añadir, editar y eliminar incidencias e implantaciones

Una vez que tenemos desarrollado el inicio de sesión, validación y muestra de contenidos, el siguiente paso es continuar con el CRUD de incidencias e implantaciones. Para cada una de estas acciones, vamos a desarrollar un componente, el cual va a recibir el modelo de incidencias o implantaciones y, en función de su cometido, realizará una inserción, modificación o borrado lógico. En el caso de la inserción, **ambos componentes serán formularios**, por lo que seguiremos la lógica de desarrollo que empleamos en el componente de ‘Inicio de sesión’. En el caso de modificar, crearemos un contenido similar, solo que este **debe recoger un parámetro que contenga el objeto actual a modificar**. En el caso del borrado lógico, simplemente **necesitamos el id de la incidencia o implantación**, sobre la cual vamos a modificar el valor del estado en el que se encuentra. Recordemos que los valores son 0, 1 y 2, los cuales corresponden, respectivamente, a inactiva, activa y archivada.

Para acceder directamente a dichos componentes (recordemos que vamos a acceder a ellos pulsando un botón, cómo por ejemplo, el botón: “añadir incidencia”), no necesitamos un método que contenga el módulo ‘Routes’ ni emplear el método ‘navigate()’. En su lugar, utilizaremos la función ‘RouterLink’ que incorpora angular y que nos **permite acceder a otro componente mediante su dirección en el servicio de enrutado**.

```
<button id="add-deploy" routerLink="/add-deploy" class="btn btn-primary">Añadir</button>
```

La lógica y diseño para estos dos casos, inserción de incidencia y de implantación, serán prácticamente los mismos. Para ambos vamos a crear un servicio,

al cual llamaremos “request.service.ts” y que contendrá los métodos para el CRUD, con el fin de simplificar las operaciones con el Back-End a partir de este momento:

```
@Injectable()
export class RequestService {
constructor(
private http: HttpClient, private auth: AuthenticationService) { }
tok = this.auth.currentUserValue.token;

//GET
getAll(url: string): Observable<any> {
let header = new HttpHeaders();
header = header.set('Authorization', 'Bearer ' +
this.tok).set('Content-Type', 'application/json; charset=utf-8');
return this.http.get(url, { headers: header });
}
get(url: string): Observable<any> {
let header = new HttpHeaders();
header = header.set('Authorization', 'Bearer ' +
this.tok).set('Content-Type', 'application/json; charset=utf-8');
return this.http.get(url, { headers: header });
}
//POST
post(url: string, item: any): Observable<any> {
let header = new HttpHeaders();
header = header.set('Authorization', 'Bearer ' +
this.tok).set('Content-Type', 'application/json; charset=utf-8');
return this.http.post(url, item, { headers: header });
}
//PUT
put(url: string, item: Issue): Observable<any> {
let header = new HttpHeaders();
header = header.set('Authorization', 'Bearer ' +
this.tok).set('Content-Type', 'application/json; charset=utf-8');
return this.http.put(url, item, { headers: header });
}
// Método DELETE
del(url: string): Observable<any> {
let header = new HttpHeaders();
console.log(this.tok);
header = header.set('Authorization', 'Bearer ' +
this.tok).set('Content-Type', 'application/json; charset=utf-8');
return this.http.put(url, null, { headers: header });
}
}
```

Con dicho servicio, vamos a atender el resto de peticiones que quedan para la APP. Cabe señalar que, aunque el último método se denomina ‘delete’, **no es una función tipo DELETE como tal, sino que es una petición PUT que emplea el id del elemento para cambiar el ‘status’ de 0 ó 1 a 2 (archivado)**, por lo que no se mostrará.

Para editar las incidencias e implantaciones vamos a emplear componentes muy particulares. Se tratan de modales que surgirán en la misma ventana donde se muestran los detalles de las incidencias e implantaciones. Hemos elegido este sistema, ya que era mucho más cómodo desarrollar la interfaz de un modal que tener que

desarrollar el diseño de otra página adicional. Para eliminar elementos no emplearemos componentes tampoco, sino que crearemos **diálogos de confirmación** los cuales, en caso afirmativo, archivarán las incidencias e implantaciones. En caso negativo, saldremos del diálogo y no se producirá acción alguna.

Tanto los modales como la confirmación de borrado serán componentes creados con el método de crear componentes que ya conocemos. En el caso de los modales de edición, accederemos a ellos desde el componente padre, que en este caso es el componente donde se encuentra el botón donde los llaman, de la siguiente manera:

```
<button class="btn btn-primary btn-sm" data-toggle="modal" data-target="#myModal"> <i class='far fa-edit'></i> Editar </button>

<app-edit-modal *ngIf="enableModal" [sites]="sites"
[issues]="issues"></app-edit-modal>
```

En el caso del componente de borrado del elemento, accedemos a él de la siguiente forma:

```
<button data-toggle="modal" data-target="#myRemoveModal" class="btn btn-danger btn-sm"><i class='fas fa-trash-alt'></i> Archivar</button>

<app-delete-modal *ngIf="removeConfirmation" [sites]="sites"
[issues]="issues"></app-delete-modal>
```

Como vemos, son formas muy similares de acceder a ellos. La diferencia radica en la etiqueta del componente empleado para cada caso.

En cada componente de adición, modificación o borrado, vamos a encontrar un método al que accedemos mediante un botón en el diseño. En el caso de editar un contenido, accedemos a él de la siguiente forma:

```
<button class="btn btn-primary btn-sm" (click)="edit()"> Editar </button>
```

Esto activará el método ‘edit()’, que se encuentra en el fichero typescript del componente. Veamos que ocurre en el modal de **editar incidencias**:

```
edit() {
    let uri = 'http://localhost/www/captainer/src/test/api-lumen/public/api/issue/update/' + this.id;

    let issue = new Issue(this.issueTitle, this.solution, this.observations,
    this.status, this.closed_by, this.deploy_id);

    this.requesting.put(uri, issue).subscribe(
        data => {
```

```
    this.refresh();
  },
  error => {
    console.log(error);
  }
})}
```

Como vemos, estamos empleando la sentencia ‘**this.requesting.put**’, siendo ‘requesting’ una instancia del servicio ‘request.service.ts’. En caso de inserción correcto, el programa usará el método ‘refresh()’ refrescando la pantalla con la lista actualizada de incidencias, donde aparecerá la modificación.

5.2.4. Tercera fase de la aplicación: Control de sesión y vista detalles

Durante el desarrollo, hemos avanzado la aplicación. Ya contamos con los métodos del CRUD y la validación de usuario. En definitiva: **la aplicación es capaz de consumir el servicio API REST e interactuar con la base de datos**, entre otras cosas. Es el momento de poner en marcha la tercera fase.

El gran problema de la aplicación en este punto es que, a pesar de que el usuario puede ingresar (y debe hacerlo en el sistema), la aplicación **no contempla la posibilidad de que un usuario no validado pueda entrar en la dirección '/admin', simplemente escribiendo la dirección en la barra del navegador**. Y, por contra, tampoco contempla la posibilidad de que un usuario ya validado pulse "atrás" y vuelva a ingresar en el sistema una y otra vez. A groso modo, podríamos decir que **la validación funciona, pero el usuario, o mejor dicho, la sesión; no es capaz de ser validada**. Hasta ahora, la validación solo sirve como enlace a un nuevo componente, pero poco más. Resultaría igual de útil hacer una pregunta de seguridad y que esta fuese acertada por el usuario de la empresa. Así que, en este apartado, vamos a contemplar la posibilidad de validar una sesión correctamente.

A su misma vez, los métodos del CRUD que hemos desarrollado a partir de la validación de **usuario no son seguros**. Cualquier persona que acceda a la aplicación sin estar validado podría insertar, eliminar o editar una incidencia o implantación. Lo ideal es que solo un usuario validado de la aplicación pueda lanzar las peticiones. Esto se consigue mediante el correcto uso de las cabeceras de las peticiones.

De hecho, **ya contamos con las herramientas necesarias**, puesto que hemos estado allanando el terreno para cuando este momento llegase. En otros apartados hemos hablado de **Token**, de **JWT** y de **encriptación** en el Back-End. Es momento de poner en práctica lo aprendido. Al momento de validar la sesión, una de las acciones es la siguiente:

```
if (user && user.token) {
    localStorage.setItem('currentUser', JSON.stringify(user));
    this.currentUserSubject.next(user);
}
```

Esto guarda un objeto en el ‘**LocalStorage**’ del navegador, el cual contiene el **Token** que genera el Back-End. Para controlar el acceso de la sesión, tenemos que comprobar el estado del Token en todo momento: si se encuentra o no en el ‘**LocalStorage**’, y si éste se encuentra expirado o no.

Crearemos para ello un nuevo servicio, el cual llamaremos ‘**auth-guard.service.ts**’ y que se encargará de verificar que el usuario se encuentra manejando una sesión activa.

```
@Injectable()
export class AuthGuardService implements CanActivate {
  constructor(public auth: AuthenticationService, public router: Router) {}

  canActivate(route: ActivatedRouteSnapshot): boolean {
    try {
      const rol = route.data.role;
      const token = localStorage.getItem('currentUser');
      const tokenPayload = decode(token);
    } catch (error) {
      console.log("El token no existe o es invalido")
      this.router.navigate(['login']);
    }
  }
  public getTokenRole() {
    const token = localStorage.getItem('currentUser');
    const tokenPayload = decode(token);
  }
}
```

Con este servicio, lo que haremos será **capturar el Rol del usuario en el Token almacenado en el ‘LocalStorage’**. Si el rol existe y/o coincide con los roles indicados en el servicio de enrutado del módulo, se procederá a ingresar en el sistema. En caso contrario, **el sistema nos devolverá a la pantalla de ‘/login’**. Veamos como se controla esto en el enrutado:

```
RouterModule.forRoot([
{
  path: 'login',
  component: LoginComponent
},
{
  path: 'admin',
  component: AdminComponent,
  canActivate: [RoleGuard],
  data: {
    role: ['admin', 'tec_imp', 'tec_sop', 'tec_fac']
  }
}],
```

inyectamos el servicio en la ruta de cada uno de los componentes que queramos que cumplan esta condición (rol), tal y como hacemos en ‘admin’. Ahora, vamos a controlar lo opuesto, es decir, que un usuario ya activo quiera ingresar en la dirección ‘/login’.

En el constructor del componente ‘login.component.ts’, hacemos lo siguiente:

```
constructor(
  private router: Router,
  private authenticationService: AuthenticationService,
  private authGuard: AuthGuardService
) {
  if (this.authenticationService.isAuthenticated() == true) {
    this.router.navigate(['admin']);
    this.authGuard.getTokenRole();
  }
  else {
    this.authenticationService.logout();
  }
}
```

De esta forma, cuando ingresemos en dicha dirección, **si el usuario se encuentra validado, será redirigido a la ventana ‘/admin’**, evitando que vuelva nuevamente a autenticarse. En caso contrario, **ejecutaremos el método ‘logout()’** del servicio ‘authentication.service.ts’.

Dicho método funciona de la siguiente forma:

```
logout() {
  localStorage.removeItem('currentUser');
  this.currentUserSubject.next(null);
}
```

El objeto del ‘LocalStorage’ será eliminado. Esto es útil cuando el **Token existe, pero ha expirado**. Será eliminado en pos de que el usuario se valide y se vuelva a generar.

Otra forma de cerrar sesión **de forma manual y consentida**, será creando un botón de “cerrar sesión”. Dicho botón, situado en la ventana de administración, tendrá la siguiente estructura:

```
<a (click)="openConfirmationDialog()">Cerrar sesion</a>
```

El cual ejecutará el método ‘**openConfirmationDialog()**’ del controlador, el cual lo que hace es abrir un mensaje de confirmación que se halla en otro componente que hemos creado para ello. En dicho componente, encontramos el siguiente método (en caso de que hayamos pulsado que ‘Si’):

```
public accept() {
  this.activeModal.close(true);
  this.router.navigate(['logout']);
}
```

Donde ‘/logout’ es un componente que hemos creado, en cuyo controlador typescript, al iniciar, efectuá la siguiente acción:

```
ngOnInit() {
  this.auth.logout();
  this.router.navigate([''])
}
```

Eso producirá que, al pulsar sobre el botón de ‘cerrar sesión’, el objeto del ‘LocalStorage’ sea eliminado, y nosotros seamos redirigidos a ”, es decir, a la dirección raíz.

Una vez solventado este problema, estudiemos la posibilidad de crear una vista ‘detalles’.

Para crear la vista **detalles**, vamos a emplear una herramienta de Angular hasta ahora no vista: **las rutas activas o rutas dinámicas**. Para ello, nos vamos al fichero donde hemos implementado el módulo de rutas, bien en app.module.ts como en nuestro caso, o bien en un servicio independiente.

```
path: 'admin/:id',
component: IssueDetailsComponent,
canActivate: [RoleGuard],
data: {
  role: ['admin', 'tec_imp', 'tec_sop', 'tec_fac']
}
```

mediante la sintaxis ‘/:variable’, lo quearemos será redirigir a una clase creada para mostrar los detalles de las incidencias.

Ahora, desde la plantilla donde mostramos todas las incidencias, vamos a crear un método que nos permita acceder al Id de una incidencia seleccionada:

```
<tr role="row" (click)="onSelect(site)">
```

Si pulsas sobre cualquier etiqueta ‘<tr>’ efectuaremos el método ‘onSelect()’, cuyo parámetro es el objeto ‘site’ seleccionado (el cual contiene el Id). En el typescript, elaboramos la lógica de dicho elemento:

```
selectedIssue;
onSelect(site: DiveSite) {
  this.selectedIssue = site;
  this.router.navigate(['admin', this.selectedIssue.id])
}
```

De esta forma, indicamos que vamos a navegar a la dirección, cuyo sufijo es el Id del objeto seleccionado.

Ahora, debemos indicar como vamos a mostrar los datos de dicho objeto. Una vez que hemos sido redirigidos a dicho componente, supongamos que con Id = 3; en la lógica del componente añadimos lo siguiente en el constructor:

```
this.router.paramMap.subscribe((params: ParamMap) => {
  this.id = +params.get('id');
```

Lo que hacemos con esto es capturar el Id de la dirección de navegación actual ('/admin/3'). Digamos que este procedimiento es algo así como ingeniería inversa. Primero, seleccionamos un objeto de la lista de objetos, y extraemos su Id. Luego, empleamos esa Id para viajar a una dirección, cuya Id es el sufijo de dicha dirección. Y por último, de esta dirección extraemos la Id en otro componente. Esto lo hacemos para que, a la hora de hacer una petición GET del objeto en cuestión, tengamos de primera mano el parámetro necesario.

```
issueSelected(ide: number) {
  let uri = '192.168.1.20/captainer/backend/public/api/issue/' + ide;
  this.requesting.get(uri).subscribe( response => {
    this.issues = response;
  },
  error => {
    console.log(error);
  })
}
```

En la plantilla, mostraremos los detalles empleando las técnicas ya vistas hasta ahora.

```
<table class="table table-responsive" *ngFor="let issue of issues">
```

En este caso, como vamos a mostrar más detalles, consultaremos más datos de otras tablas mediante otras peticiones GET en la clase. Recordemos que 'this.requesting' es la instancia al servicio 'request.service.ts'.

5.2.5. Cuarta fase de la aplicación y fase actual: Implementación de animaciones de carga, filtrado de objetos, pantalla de error y otros detalles

Por último, demos un repaso a los últimos avances de la aplicación. Una vez finalizado los requisitos básicos: CRUD de incidencias, CRUD de implantaciones, rol de usuarios, etc. Vamos a pulir ciertos detalles de la aplicación, atendiendo a los criterios de la empresa y de la correcta programación orientada a eventos. Puesto que las peticiones requieren conexión con el servidor y la base de datos, existen ocasiones en las que dichas peticiones requieren un tiempo de carga mayor al tiempo que tarda la aplicación en ejecutarse, sobre todo en equipos menos potentes. Esto ocurre en parte porque la conexión tiene micro-retardos o simplemente porque la conexión no se encuentra disponible. En caso de ser así, debemos, en primer lugar, controlar que la aplicación no se encuentre funcional hasta que los datos sean cargados. Si lo dejamos al azar, como hemos hecho hasta ahora, la aplicación cargará su interfaz totalmente antes de que los datos hayan sido recogidos por la función de 'CallBack'. Esto provocará

que la ventana se haya cargado indebidamente y se muestre vacía, ya que una vez que la carga de la interfaz se ha completado, esta se muestra con o sin datos, y no hay posibilidad de que estos se carguen. Para evitar esto, debemos poner en marcha el siguiente planteamiento:

La App se inicia → La carga de la interfaz se congela → se cargan los datos → Se inicia la carga de la interfaz con los datos

Ahora bien. ¿Cómo se consigue? Pues, en primer lugar, debemos saber que en Angular podemos utilizar mediante el uso de ‘Async’ lo cual permitirá “dormir” el hilo durante un tiempo, mientras que el hilo de la petición se produce.

```
private delay(ms: number) {
  return new Promise(resolve => setTimeout(resolve, ms));
}

cargarLista() {
  (async () => {
    this.loading = true;
    //Ejecutamos la acción
    this.loading = false;
  })();
}
```

Donde: ‘loading’ es una variable de tipo booleana que actuará como interruptor y que permitirá, o no, la animación de carga mientras se ejecuta el código y se obtiene la respuesta (Observable). Esta animación se trata de una importación aparte de Angular base, y se consigue instalándola de forma externa mediante el comando:

```
npm install --save ngx-loading
```

Ahora, con la animación instalada, podremos importarla en nuestras clases y utilizarla a voluntad:

```
import { ngxLoadingAnimationTypes, NgxLoadingComponent } from 'ngx-loading';

export class AdminComponent implements OnInit {

  public ngxLoadingAnimationTypes = ngxLoadingAnimationTypes;
  public loading = true;
  public primaryColour = '#F08080';
  public secondaryColour = '#87CEFA';
  public coloursEnabled = false;
  public loadingTemplate: TemplateRef<any>;
  public config = { animationType: ngxLoadingAnimationTypes.none,
    primaryColour: this.primaryColour, secondaryColour: this.secondaryColour,
    tertiaryColour: this.primaryColour, backdropBorderRadius: '3px' };
}
```

Y, en la plantilla:

```
<ngx-loading [show]="loading"
[config]="{{animationType: ngxLoadingAnimationTypes.chasingDots,
primaryColour: primaryColour, secondaryColour: secondaryColour,
backdropBorderRadius: '3px'}}"
[template]="loadingTemplate"></ngx-loading>
```

Con este procedimiento, conseguiremos crear una animación que durará x segundos, dependiendo de lo que tarde en cargar la lista. A veces será inmediato y en otras ocasiones notaremos como se demora en cargar la aplicación. Otra ventaja de este procedimiento es que, mientras se encuentra activa la animación, el control de la pantalla queda inactivo, por lo que no podremos efectuar ninguna acción sobre los elementos de la pantalla. Gracias a ello, el usuario no podrá manipular información mientras ésta se está procesando, pudiendo provocar inconsistencia o corrupción de datos.

Ahora, vamos a tratar de buscar una alternativa, en caso de que la conexión con el servidor se demore demasiado o éste no se encuentre activo en ese momento. Deberemos hacer una comprobación continua en la que, en caso de que el servidor no se encuentre disponible, ser redirigidos a una ventana informativa. Esto lo hacemos en el componente padre del módulo, indicándole que cada vez que cargue (recordemos que un componente padre también se carga cada vez que un componente hijo lo hace), compruebe mediante una petición si hay conexión y que, en caso desfavorable, seamos redirigidos a una página alternativa. Esto lo hacemos para que el usuario no pueda interactuar con la aplicación mientras que esta se encuentra inactiva.

```
Uri = '.../public/check';

constructor(public route: Router, private check: CheckConnService) { }

ngOnInit(): void {
    this.checking();
}

checking() {
    this.check.get(this.uri).subscribe(
        error => {
            this.route.navigate(['error']);
        }
    )
}
```

Donde 'error' es un componente creado para este fin, y se encuentra declarado en 'app.module.ts'.



Ilustración 40: Pagina de error de conexión

En dicho módulo encontramos la misma consulta, pero con lógica inversa: si la conexión genera una respuesta, entonces seremos redirigidos de nuevo a la aplicación. Si queremos que la consulta se produzca cada x segundos, de modo que la página intente refrescar automáticamente, podemos usar el módulo ‘interval’ que se encuentra dentro de la librería ‘rxjs’ y emplearlo de la siguiente forma:

```
secondCounter = interval(4000);
this.secondCounter.subscribe(n => {
    //Aqui hacemos la petición
});
```

En este caso, la consulta se produciría cada 4 segundos mientras que la aplicación estuviese activa.

Para finalizar este apartado de desarrollo, desarrollaremos una funcionalidad que permita, mediante una barra de búsqueda, hacer un filtrado de objetos en una lista. Para esto, emplearemos lo que hasta ahora desconocíamos: las tuberías, o ‘pipes’. Este tipo de scripts se utilizan para injectar una funcionalidad pequeña en cualquier parte del código.

Para elaborar el filtrado de objetos, hacemos lo siguiente: en primer lugar, debemos desarrollar el diseño de la barra de búsqueda. Nosotros la vamos a colocar en una barra lateral que tiene la pestaña de administración:

```
<div id="busqueda" class="md-form active-cyan-2 mb-3">
<input [(ngModel)]="searchText" class="form-control" type="text"
placeholder="Buscar..." aria-label="Search">
</div>
```

Echamos mano de la directiva ‘ngModel’, la cual como ya sabemos, permite la interacción entre la plantilla HTML y el código Typescript. ‘searchText’ es una variable que permanecerá vacía en el fichero Typescript del componente hasta que escribamos sobre la barra.

Para que esto resulte, creamos un ‘pipe’, el cual tiene la siguiente forma:

```
import { Pipe, PipeTransform } from '@angular/core';
import { DiveSite } from './resources/dive.site';

@Pipe({
    name: 'filter'
})

export class FilterPipe implements PipeTransform {
    transform(sites: DiveSite[], searchText: string): DiveSite[] {
        if (!sites || !searchText) {
            return sites;
        }
        return sites.filter(sites =>
            sites.deploy_name.toLowerCase().indexOf(searchText.toLowerCase()) !=
            -1);
    }
}
```

Una vez creada, para injectarla solo tendremos que acceder desde la plantilla a esta funcionalidad:

```
<tr role="row" *ngFor="let site of sites |filter: searchText">
```

Angular es un servicio dinámico y es capaz de encontrar la ‘pipe’ por su nombre, sin necesidad de que esta sea importada. Como vemos en el código de ésta, requiere que sea de tipo ‘string’ y, como tal, devolverá una cadena. Recogerá, por tanto, el valor que introduzcamos por pantalla en la barra de filtrado y ejecutará su lógica, la cual es eliminar espacios, mayúsculas, etc. Con esta cadena ya procesada, mostraremos en la plantilla de la lista de implantaciones aquellas cuyo ‘deploy_name’ coincida con aquello escrito en la barra.

6. Pruebas

6.1 Introducción

Una vez completado el desarrollo de la aplicación, no podremos dar por concluido su fase final y lanzarlo al uso sin antes haber realizado las pruebas oportunas en él. Las pruebas de software nos ayudan a acercar el producto final al ideal de software funcional y libre de errores. Si bien, ningún software está exento de errores, las pruebas nos permiten detectar y corregir la gran mayoría de estos. Si queremos asegurar la calidad de nuestra aplicación, deberemos efectuar la mayor cantidad y diversidad posible de pruebas, con el fin de detectar las anomalías existentes. Es imposible acertar 100 por 100 con la idea del cliente a la hora de desarrollar un producto. Ocurre que a veces no interpretamos los requisitos de éste de manera exacta, y es necesario que los clientes finales utilicen la aplicación en la fase final de pruebas, para que sean ellos los que encuentren aquellas partes que requieren una modificación.

Hablemos nuevamente de la calidad de la aplicación. No es solo una garantía de correcto funcionamiento. Implica, además, que satisfaga las necesidades de los clientes, que sea usable, eficiente, cumpla los estándares, etc.

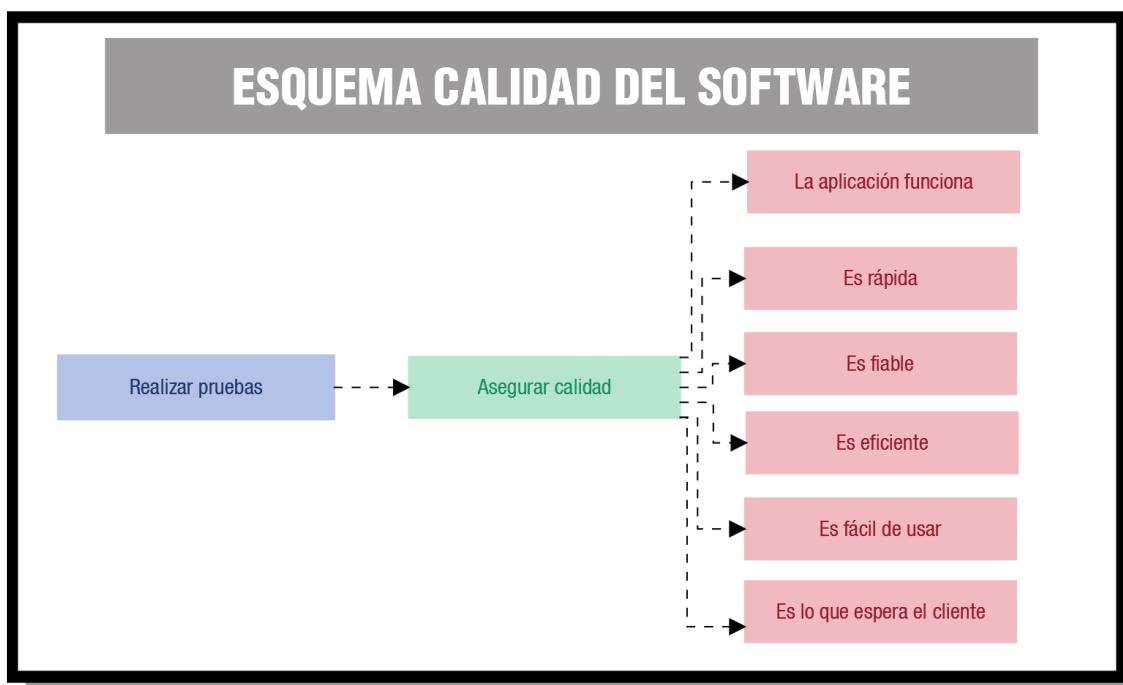


Ilustración 41: esquema de calidad del software

No contar con una estrategia de pruebas puede acarrear una serie de inconvenientes:

- Falta de satisfacción de los clientes.
- Fallos en la explotación por defectos no detectados y eliminados a tiempo
- Altos costes finales, por el tiempo empleado en corregir defectos de diseño.

En definitiva, tener una buena estrategia de pruebas se traduce en las siguientes ventajas:

- Mayor satisfacción de los clientes
- Mayor productividad
- Menor riesgo de negocio
- Facilidad de crecimiento del negocio
- Gestión correcta de pruebas y de calidad.

6.2 Pruebas

Descubrir un error en la aplicación se considera un éxito en una prueba, ya que, como hemos mencionado, ningún software está exento de ellos. Identificarlos y solucionarlos constituye en este apartado los dos mayores objetivos a realizar. Por ello, es fundamental que los defectos no se propaguen de unas fases a otras hasta llegar a los sistemas de producción: lugar donde realizar una modificación puede resultar algo costoso e incluso trágico. Es por ello que vamos a realizar distintas pruebas sobre nuestro software, intentando profundizar en ellas lo máximo posible y procurando obtener los mejores resultados, dentro de los esperados:

6.2.1. Pruebas de Integración

Se realizan una vez probado el funcionamiento de las partes o módulos de la aplicación por separado. Consiste en verificar que el software, en conjunto, cumple con su cometido y se prueba que el programa, una vez unido, no da lugar a error o defecto. Lo recomendable es hacer una integración **Incremental**, en la cual se van añadiendo nuevos módulos cada vez en la realización de las pruebas; y el módulo incorporado se prueba con el conjunto de módulos ya testeados. En nuestro caso, y como hemos hablado durante todo el documento, nuestro desarrollo ha sido totalmente **modular**, ya que Angular se constituye de módulos y componentes. Y, por otra parte, el Back-End se encuentra prácticamente independiente del Front-End. Por tanto, las pruebas de integración **se han ido realizando gradualmente y por mera necesidad durante el desarrollo del producto** en las fases de diseño e implantación. En nuestro caso, primero hemos desarrollado el componente de Inicio de sesión, haciendo posible la validación de usuario y la navegación entre ventanas. Posteriormente, hemos

desarrollado por separado el componente que realizaba una petición GET al servidor y recogía un listado de incidencias. Luego, hemos probado ambos módulos en conjunto, al fusionarlos en un solo módulo, una vez que hemos comprobado que por separado funcionaban correctamente. Lo mismo hemos hecho con el módulo que realizaba la inserción de elementos en la base de datos, utilizándolo posteriormente en otro componente para la aplicación final y comprobando que éste, junto con los demás, funcionaba correctamente. En algunos casos, gracias a las pruebas de integración se han detectado diversos errores, como pueden ser el caso de que los distintos desarrollos seguían directrices distintas, ya que empleaban lógicas muy dispares y, a la hora de realizar el diseño final, hemos tenido que optar por una u otra, ya que en conjunto esto no funcionaba como era debido. Por ejemplo, el módulo que realizaba la recogida de información mediante un GET, empleaba directivas '@input' Y '@output', mientras que, en módulo desarrollado para la validación de usuario, la interacción con la plantilla HTML se realizaba mediante la directiva '**ngModel**'. Mediante las pruebas de integración, se llegó a la conclusión que la segunda alternativa era más eficiente, ya que ahorraba un proceso en el programa y permitía la modificación con más facilidad.

6.2.2. Pruebas de Sistema

Las pruebas de sistema se realizan para comprobar que la aplicación no solo funciona en un equipo, **que suele ser en el equipo empleado para el desarrollo de la aplicación**, sino que, además, cumple sus funciones al mismo nivel en equipos diferentes. Dichas pruebas verifican el comportamiento del sistema en su conjunto, y se comprueba que se cumplen los requisitos siguientes:

- Seguridad
- Velocidad
- Exactitud
- Fiabilidad

Para llevar a cabo este tipo de pruebas, primero debemos dividirlas en las siguientes subcategorías para que puedan ser estudiadas con mayor detenimiento, atendiendo a las características de cada prueba. Estas son:

- A. **Pruebas de configuración:** Verifican la instalación del software en el entorno de destino y comprueban el comportamiento del sistema frente a los requisitos de configuración. Verifican que se han integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas.
- B. **Pruebas de recuperación:** Fuerza el fallo del software de múltiples formas, con el fin de verificar que la recuperación ante el fallo se lleva a cabo de forma satisfactoria
- C. **Pruebas funcionales o de caja negra:** validan si el comportamiento observado del software es conforme con sus especificaciones (análisis de los valores límite). En nuestro caso **si ha sido así**, ya que el proceso de desarrollo ha sido **muy gradual** y ha permitido ir probando los resultados con mucho detenimiento durante cada fase de desarrollo. Gracias a esto, contamos con una aplicación con diferentes funcionalidades que, al realizar dichas pruebas

sobre ella, los resultados son más que favorables en este sentido.

Concretamente, hemos probado esto en la **validación de usuario, inserción de datos, filtrado de información y cierre de sesión**. Todas las pruebas han sido exitosas.

- D. De capacidad y rendimiento:** Una de las cuestiones que nos abordaban era que capacidad de producción tiene nuestra aplicación en el entorno de trabajo, si esta es utilizada por múltiples procesos, provocando este tipo de cuestiones que el rendimiento del equipo descienda gravemente. En nuestro caso particular **no ha sido así**. Para una de las pruebas de rendimiento y capacidad, hemos abierto **20 pestañas** con la aplicación, de manera simultánea, y hemos intentado que el rendimiento caiga. Cabe destacar que los ordenadores de la empresa son equipos potentes, por lo que, tratándose de los encargados de utilizar este software, no tendrán ningún tipo de problema de cara a futuras mejoras y actualizaciones del producto, con el incremento de consumo que eso conlleva.

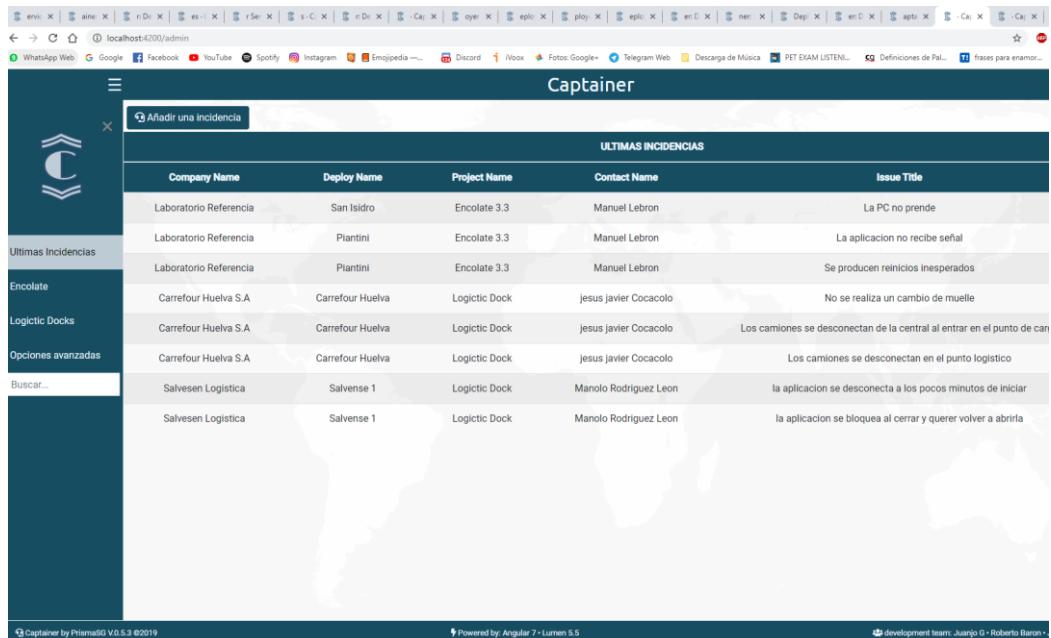


Ilustración 1: Pruebas de rendimiento en Chrome

E. De uso de recursos:

Otra de las pruebas interesantes que podemos hacer a nuestro software una vez desarrolladas sus funciones, son las pruebas de recursos. En ella, vamos a comprobar que el equipo de trabajo **no alcance una situación de estrés** tras usar durante un rato nuestra aplicación. Pues bien. Hemos estado utilizando la aplicación aproximadamente 5 minutos, mientras otras muchas instancias de la aplicación en el navegador continuaban su uso, refrescando cada cierto tiempo su actividad. El resultado, en uso de recursos en el equipo, es el siguiente:

Nombre	Estado	8% CPU	60% Memoria	0% Disco	0% Red	Consumo de energía	Tendencia de consumo
Aplicaciones (8)							
> Administrador de tareas		0,7%	19,0 MB	0 MB/s	0 Mbps	Muy baja	Muy baja
> Explorador de Windows		0%	28,0 MB	0 MB/s	0 Mbps	Muy baja	Muy baja
▼ Google Chrome (31)		4,3%	1.471,1 MB	0,1 MB/s	0,1 Mbps	Bajo	Moderado
Google Chrome		0%	45,3 MB	0 MB/s	0 Mbps	Muy baja	Muy baja
Google Chrome		0%	16,1 MB	0 MB/s	0 Mbps	Muy baja	Muy baja
Google Chrome		0,4%	95,4 MB	0 MB/s	0 Mbps	Muy baja	Bajo
Google Chrome		0,9%	18,4 MB	0 MB/s	0 Mbps	Muy baja	Muy baja
Google Chrome		0%	6,2 MB	0 MB/s	0 Mbps	Muy baja	Muy baja
Google Chrome		0%	0,6 MB	0 MB/s	0 Mbps	Muy baja	Muy baja
Google Chrome		0%	0,5 MB	0 MB/s	0 Mbps	Muy baja	Muy baja
Google Chrome		0%	0,6 MB	0 MB/s	0 Mbps	Muy baja	Muy baja
Google Chrome		0,7%	14,5 MB	0 MB/s	0 Mbps	Muy baja	Muy baja
Google Chrome		0%	192,7 MB	0 MB/s	0 Mbps	Muy baja	Muy baja
Google Chrome		0,2%	11,8 MB	0 MB/s	0,1 Mbps	Muy baja	Muy baja
Google Chrome		0%	47,2 MB	0 MB/s	0 Mbps	Muy baja	Muy baja
...							
Menos detalles							

Ilustración 2: Consumo de recursos en Windows 10

F. De seguridad:

Por último, haremos un repaso a algunas pruebas de seguridad que se deben hacer a la aplicación. En concreto, con dos pruebas de autenticación es suficiente para corroborar que el sistema es seguro (o no).

- La primera prueba consiste en **entrar en las diferentes páginas protegidas sin validación** de usuario. El sistema detectará que no existe el token de validación y que, por tanto, no es posible acceder a ella, ya que nuestro servicio de autenticación no podrá validar el usuario y, por tanto, **seremos redirigidos a la ventana de iniciar sesión**, en todos los casos.
- La segunda prueba ha consistido en **borrar el token de forma manual**, con la finalidad de alterar el funcionamiento de la aplicación. Y, una vez borrado, probar a lanzar una petición POST al servidor. Nuevamente, el sistema tratará de comprobar mediante el servicio la validación si existe o no un token de validación. Al encontrarse borrado del registro del ‘LocalStorage’, será imposible recibir la respuesta a la petición, puesto que las cabeceras **se envían sin el token**. Y, en el Back-End, al recibir la petición, la cual pasa por el Middleware, **arrojará un mensaje de error** del tipo: “*401 Unauthorized*”.

6.2.3. Pruebas de Usuario

Las pruebas de usuario (también conocida como de usabilidad) consiste en asegurar que la interfaz de usuario (GUI) de la aplicación sea amigable, intuitiva y que funcione correctamente. Se determina la calidad del software en la forma en la que el usuario interactúa con el sistema, considerando la facilidad de uso y satisfacción del cliente.

Para atender a esta necesidad de usabilidad y diseño, hemos recurrido al equipo de desarrollo de la empresa, los cuales han realizado las pruebas y han determinado cuales son las posibles mejoras de cara a futuras versiones; así como correcciones que deberían efectuarse de forma inmediata. Algunas de estas pruebas, son:

- “La ventana inicial, ‘Home’, debería ser más dinámica puesto que, al iniciar, no tiene sentido una ventana inicial si no es para aportar algún valor añadido a la aplicación”. Ante esta necesidad, se rediseñó la interfaz de dicha ventana para que, al momento de ingresar, diese esa sensación de dinamismo, carente en anteriores versiones
- “El botón de añadir incidencias, así como otro tipo de controles, deberían estar situados de forma fija en la parte superior de la pantalla, y no justo debajo de la tabla donde se muestra información, ya que, al haber múltiples registros, el control quedaría subordinado a la altura de la tabla y, por tanto, sería cada vez más difícil acceder a dichos controles”. Ante esta disyuntiva, hemos recolocado los controles en la aplicación. De modo que ahora siempre se muestran en un lugar fijo, sin probabilidad de que vayan a quedar inaccesibles.
- “El botón de cerrar sesión se encuentra en el lado superior izquierdo de la pantalla, y eso puede confundir al usuario, ya que, lo normal, es que se sitúe, de forma estandarizada, en la parte superior derecha, y no en la izquierda”. Tras esta valoración, se ha colocado el botón de cerrar sesión en el lado ideal, es decir, el lado derecho de la barra superior.

6.2.4. Pruebas de aceptación

Por último, pero no por ello menos importante, se han realizado diversas pruebas de aceptación de cara a evaluar el uso normal de la aplicación de un usuario de ésta. El objetivo es comprobar que el software cumple con los requerimientos iniciales y que funciona de acuerdo con lo que el usuario final espera de él, es decir, que sus expectativas queden colmadas. Por eso, y para que la prueba sea objetiva, es el usuario final quien las ha realizado. Concretamente, la aplicación ha sido usada por un voluntario del equipo de desarrollo de la empresa, Jesús Pérez Manga quién, tras el uso continuado de la aplicación, ha elaborado el siguiente informe:

Uso de la aplicación tras cinco minutos de uso:

En lo general la aplicación cumple con las expectativas esperadas. Los botones cumplen su función y los elementos de la pantalla son accesibles y no comprometen la visibilidad tras redimensionar la pantalla. De cara a futuras versiones, se advierten las siguientes mejoras:

- *La dirección de las incidencias debería llamarse '/proyectos/general', por ejemplo, y no '/admin', ya que el objetivo es mostrar los proyectos por distintos filtros, entre los que se encuentran las incidencias y los filtros que incorporen las incidencias.*
- *La aplicación debe contar con mayor cantidad de filtros, para que sea más sencillo encontrar una incidencia, pudiendo filtrar estas por fecha, implantación, estado, etc.*
- *En el menú de formulario de inserción de incidencias, el campo "observaciones" debería ser un 'textarea', o contar con varias filas para que, a la hora de escribir comentarios largos, se pueda visualizar el contenido escrito.*
- *El menú lateral debería contar con "categorías", que mostrase las incidencias según los estados: 'inactivo', 'activo' y 'archivado'.*
 - *En futuras versiones, deberían mostrarse los botones de edición y borrado como un menú flotante en cada incidencia listada.*

6.3 Resultados Obtenidos.

En vista a todo lo anterior, los resultados obtenidos durante el conjunto de pruebas han sido **favorables**, puesto que, a pesar de algunas recomendaciones y resultados que no eran los esperados, la aplicación ha salido airosa de la mayoría de las pruebas realizadas. Por tanto, **se cumplen la mayoría de los aspectos** relacionados con la usabilidad, la integridad, la robustez, la legibilidad y la facilidad de uso. Concluimos, por tanto, que la aplicación se encuentra preparada para entrar en fase de producción y para el despliegue en el servidor. Recordemos que la aplicación se encuentra actualmente en una fase aun en desarrollo y que, por tanto, podríamos considerarla como una "**versión beta**" respecto a la versión final esperada. Sin embargo, se trata de una beta abierta, que va a ser utilizada en su versión actual.

7. Conclusiones

7.1 Conclusiones finales

Como conclusión final, podemos afirmar que **se han cumplido, en gran medida, toda la serie de objetivos propuestos** al alumno al respecto del desarrollo de la aplicación y el aprendizaje que había de lograr. Se ha conseguido diseñar y desarrollar un producto usable, íntegro, seguro, confiable y escalable que, esperemos, en futuras versiones, ya con los conocimientos adquiridos y con el respaldo del equipo de programación de la empresa, alcance su punto más álgido de desarrollo y funcionalidades. Hemos logrado diseñar una aplicación ligera, muy potente gracias a los Frameworks utilizados, y que cuenta con el valor añadido de ser una aplicación real, usada por los miembros de la organización. Podemos catalogar, por tanto, de éxito esta fase del aprendizaje.

7.2 Desviaciones temporales

Durante todas las fases de desarrollo, hemos sufrido **ligeros contratiempos**, relacionados con distintos espectros, surgidos en cada una de las fases. Estos contratiempos han sido debidos a varias cuestiones, como **el desconocimiento en la materia** en aspectos como los Frameworks con los que hemos trabajado, la logística con la que trabaja la empresa, y la falta de conocimientos acerca del correcto uso del control de versiones. Debido a estas carencias, y al incorrecto cumplimiento de algunos patrones de diseño, **han sido necesarias más horas de las previstas para alcanzar cada uno de los puntos requeridos**, dentro del orden racional impuesto por el propio alumno y el equipo de programación, en representación de los intereses de la empresa. Estos problemas temporales han sido, en algunos casos, evitables. Afortunadamente y, a pesar de todo esto, hemos conseguido cumplir con el plazo establecido y tener la aplicación lista para producción. En futuros proyectos, intentaremos aplicar los conocimientos adquiridos por esta experiencia, tanto por las cosas buenas como por las malas.

7.3 Posibles ampliaciones y modificaciones

La aplicación **seguirá en desarrollo** tras la finalización del proyecto integrado de carácter práctico. La empresa requiere el mantenimiento de la aplicación, y la mejora de algunos aspectos, ya que son conscientes de que el desarrollo de la misma requiere una ampliación razonable en el plazo de entrega final y, por tanto, debemos emplear los conocimientos adquiridos para completar los objetivos marcados en un inicio y que, por razones de tiempo y logística, fueron desechados del proyecto, pero siguen pendientes de desarrollo en los puntos de los objetivos impuestos por la empresa.

Algunos de estos, son:

- Sistema Multi-Rol: La aplicación debe limitar el uso de la aplicación (añadir incidencias, asignar implantaciones a técnicos, etc.) a según qué tipos de usuarios, filtrados por rol (administrador, técnico de implantaciones, técnico de soporte, etc.). Actualmente, todos los usuarios de la aplicación tienen plenos derechos sobre la misma, pudiendo realizar todas las acciones posibles
- Formularios reactivos: La aplicación empleará formularios avanzados, programados desde el controlador del componente, y serán más versátiles que la alternativa propuesta en este proyecto

A su vez, durante el desarrollo de la aplicación, el equipo de desarrollo de la empresa **ha propuesto una serie de mejoras y ampliaciones** para la aplicación, las cuales han quedado archivadas, con el fin de llevarlas a cabo en futuras versiones. Algunas de estas son:

- La aplicación debe tener un apartado en el que queden registradas aquellas implantaciones **que aún no han sido puestas en marcha**, con el fin de ir añadiendo sugerencias e información al respecto, a priori de ser desplegadas (para contar con los puntos de vista de los programadores, en vista de las experiencias previas)
- La aplicación debe contar con un apartado para **sugerencias de desarrollo**, es decir, para nuevas ideas para implantaciones, visores, etc.

Por último, me gustaría aclarar nuevamente que **esta aplicación como tal no es una aplicación multiplataforma al uso**. Si, es posible ejecutarla en múltiples dispositivos que funcionan con diferente arquitectura. Pero es gracias al navegador (agente externo). Esto es debido a las necesidades de la empresa, la cual trabaja con este tipo de tecnologías. No obstante, si queremos que nuestra aplicación sea multiplataforma de la forma que nosotros entendemos dicha definición, podríamos conseguirlo gracias a tecnologías como **Ionic o Electron**, que nos permita convertir **esta aplicación en una aplicación híbrida**.

7.4 Valoración personal

Inicialmente, el proyecto iba enfocado a la **gestión de usuarios de una autoescuela**: *test de preparación de contenidos teóricos, número de prácticas de conducir realizadas, etc.* Sin embargo, la empresa propuso esta idea de proyecto, la cual me permitía trabajar en ella, elaborando la aplicación en su seno día a día. De otra forma, hubiera tenido que continuar con la idea inicial, mientras que por la mañana hubiera estado desempeñando otras tareas de distinta índole. Gracias a esta propuesta, he alcanzado unas cotas de conocimiento que antes no poseía, y me ha permitido conocer cómo funciona la simbiosis programador- empresa.

Personalmente, pienso que he defendido una propuesta de trabajo que, a priori, resultaba incluso imposible, dados algunos precedentes, como que la aplicación está inicialmente orientada a web, que emplean Frameworks y herramientas que desconocía, y que contaba para todo ello con un plazo límite, el cual se veía desbordado ante diversas situaciones, como el tiempo adicional que suponía documentar todo el proceso de trabajar en el proyecto, la distancia comprendida entre la sede de la empresa y el domicilio familiar, la inversión de tiempo que debía emplear para conocer las entrañas de la programación que se utilizaba en la empresa, así como la organización en la misma y sus peculiaridades, etc. A pesar de todos los inconvenientes, he logrado sacar el proyecto adelante, y éste, pienso, se ha convertido en una propuesta interesante, y del cual podríamos sacar buenas directrices acerca del correcto desarrollo de una aplicación Angular que **puedan servir a un principiante como yo, o un alumno de cursos posteriores** que quieran emplear este proyecto como base para la elaboración de su proyecto, ya sea para alumnos del Ciclo superior de Desarrollo de Aplicaciones Multiplataformas, como para aquellos alumnos que cursan otros grados superiores, como el grado de Desarrollo de Aplicaciones Web, o también aquellos que cursan el grado de Administración de Sistemas informáticos en Red, entendiendo que este tipo de proyectos no se encuentran dentro de las competencias de estos últimos, al igual que al inicio tampoco entraba dentro de las competencias de nuestro ciclo. Sin embargo, me gustaría romper una lanza a favor de esta propuesta, ya que, gracias a esta iniciativa, no solo he conseguido dominar los conocimientos acerca de todo lo estudiado **en el ciclo superior de DAM** (Desarrollo de interfaces de escritorio, desarrollo de aplicaciones nativas en Android, desarrollo de videojuegos y animaciones con motores gráficos, etc.). Sino que, además, me ha permitido dominar ciertas cuestiones que pertenecen al **ciclo superior de DAW** (comunicación y despliegue con el servidor, interfaces web, gestión de procesos y sesiones en el navegador, etc.), **complementando así mi educación y formación como analista-programador de software**.

7.5. Agradecimientos

En primer lugar, quiero dirigir mis agradecimientos a las personas más importantes de mi vida: Mi familia, mi pareja y mi grupo de amigos, que son los responsables de que jamás haya tirado la toalla en las distintas etapas de mi vida, fueron las que fuesen las adversidades y la dificultad de los caminos que debía tomar. A mis padres les debo la inquietud, la educación, el respeto y la paciencia que me han hecho pelear por mis ambiciones. Gran parte de los frutos que hoy recojo les corresponden a ellos. A mi pareja, le debo mi autoestima, mis ganas de luchar y no rendirme. Gracias a ella, puedo decir que hoy me encuentro defendiendo un proyecto que parecía imposible. Espero que por muchos años más, sigas alimentando mis sueños. A mis amigos, quiero agradecer la picardía, el lado más humano de este proyecto, ya que gracias a ellos soy quien soy, y les debo mucho más de lo que ellos creen o quieren presumir. Gracias a todos.

También me gustaría dar las gracias a los profesores y, en especial, a mi tutor de prácticas, Juan, por confiar en mí a la hora de presentar este proyecto como una apuesta sólida, la cual tendría que defender mejor que nadie. Gracias por ser artífices de la enseñanza, haciendo equilibrio con las lecciones a impartir mientras os reinventabais como docentes y como personas. Si en un futuro considerara seguir vuestra estela como profesional docente, e impartir clases a posibles diamantes en bruto, quisiera tener muchas de las virtudes que tenéis con nosotros, como la paciencia, la perseverancia y esa aura de seguridad que os envuelve. Gracias por enseñarme a ser un buen profesional, con el riesgo de haber podido morir en el intento.

No puedo dejar por detrás, evidentemente, a mis compañeros de ciclo. Esos truhanes, por llamarlos de un modo educado, que me han acompañado a lo largo y ancho de esta travesía. Compañeros de lágrimas y confidentes, con los cuales he llegado a pasar más tiempo que con mi propia familia, convirtiéndose algunos de ellos, en una segunda familia con la que compartir sentimientos, alegrías, y algún que otro código para las prácticas de Santiago... ejem. Por todo esto, y por mucho más, ya que no quiero convertir este documento en mis memorias personales, quisiera agradecerlos estos dos años compartiendo mesa conmigo. Siempre os llevaré en el corazón.

8. Bibliografía

Durante todo el proyecto, nos hemos valido de varios manuales para aprender y comprender las herramientas que nos facilitan los Frameworks Angular y Lumen.

- Angular: “*El gran libro de Angular*” – Editorial Marcombo
- Lumen: “*Lumen programming Guide*”– Editorial Apress

Para el estudio de las distintas aplicaciones de soporte, hemos empleado los siguientes enlaces web:

- FreshService: <https://freshservice.com/es/tp/herramienta-gestion-incidencias-itil/>
- ActivaLink: <https://www.activalink.com/software-la-gestion-incidencias>
- ServiceTonic: <https://www.servicetonic.es/software-atencion-al-cliente-helpdesk/>

Para el estudio de la legislación y normativa vigente en torno a las aplicaciones en el término nacional, se han consultado los siguientes enlaces de interés:

[https://protecciondatos-lopd.com/empresas/obligacion-informar-trabajadores/#Contrasenas de los equipos](https://protecciondatos-lopd.com/empresas/obligacion-informar-trabajadores/#Contrasenas_de_los_equipos)

<https://infoautonomos.eleconomista.es/tecnologia-pymes-autonomos/ley-de-proteccion-de-datos-lopd/#derechos>

<https://www.boe.es/buscar/act.php?id=BOE-A-2008-979>

<https://protecciondatos-lopd.com/empresas/cifrado-datos/>

https://www.boe.es/diario_boe/txt.php?

Para el desarrollo durante las distintas fases del proyecto, como apoyo a los conocimientos adquiridos mediante otras fuentes de información, así como la terminología de ciertas herramientas o lenguajes, hemos empleado los siguientes enlaces de interés (entre otros muchos):

<https://angular.io/>

<https://lumen.laravel.com/docs/5.5/configuration>

https://developer.mozilla.org/es/docs/Web/HTTP/Access_control_CORS

<https://www.w3schools.com/>

<https://stackoverflow.com>

<https://www.udemy.com/curso-de-angular-4-desde-cero-hasta-profesional/>

<https://jasonwatmore.com>

<https://www.youtube.com>

Captainer v.1.4.5:

Manual de usuario para el uso
diario de la aplicación de soporte

2019

Juan José González Fernández
Prisma Virtual S.G

Introducción

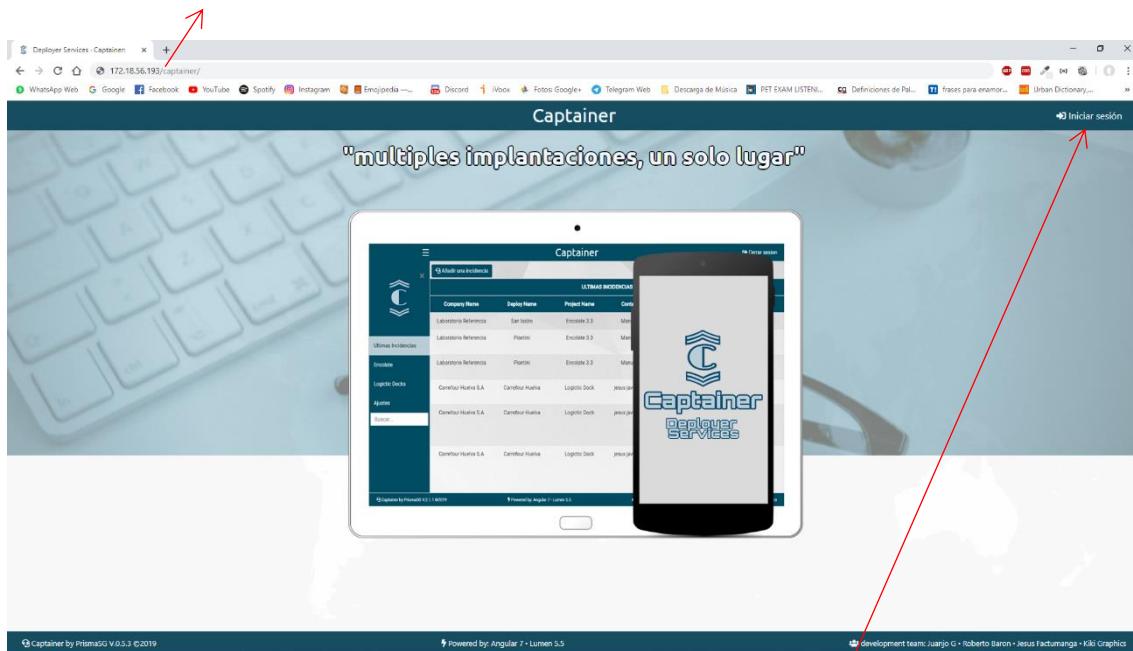
Este manual ha sido realizado con la intención de que el usuario de la aplicación sea capaz de controlar y manejar todas y cada una de las operaciones aquí citadas con la aplicación, de forma autónoma y atendiendo a las pautas de correcto uso de la aplicación.

Recordemos que la aplicación se encuentra aún en una fase de desarrollo, y que dicho manual contiene instrucciones del uso que podemos llegar a realizar en su fase actual. La aplicación cuenta con la capacidad de interactuar con las implantaciones e incidencias de la empresa, las cuales se encuentran registradas en la base de datos del proyecto. En este documento quedarán reflejados las posibles interacciones con la aplicación.

Cabe destacar que, actualmente, la aplicación no es capaz de distinguir el rol de usuario, por lo que este manual será válido tanto para administradores como para técnicos de implantación, técnicos de soporte y técnicos de facturación.

1. Ventana de bienvenida

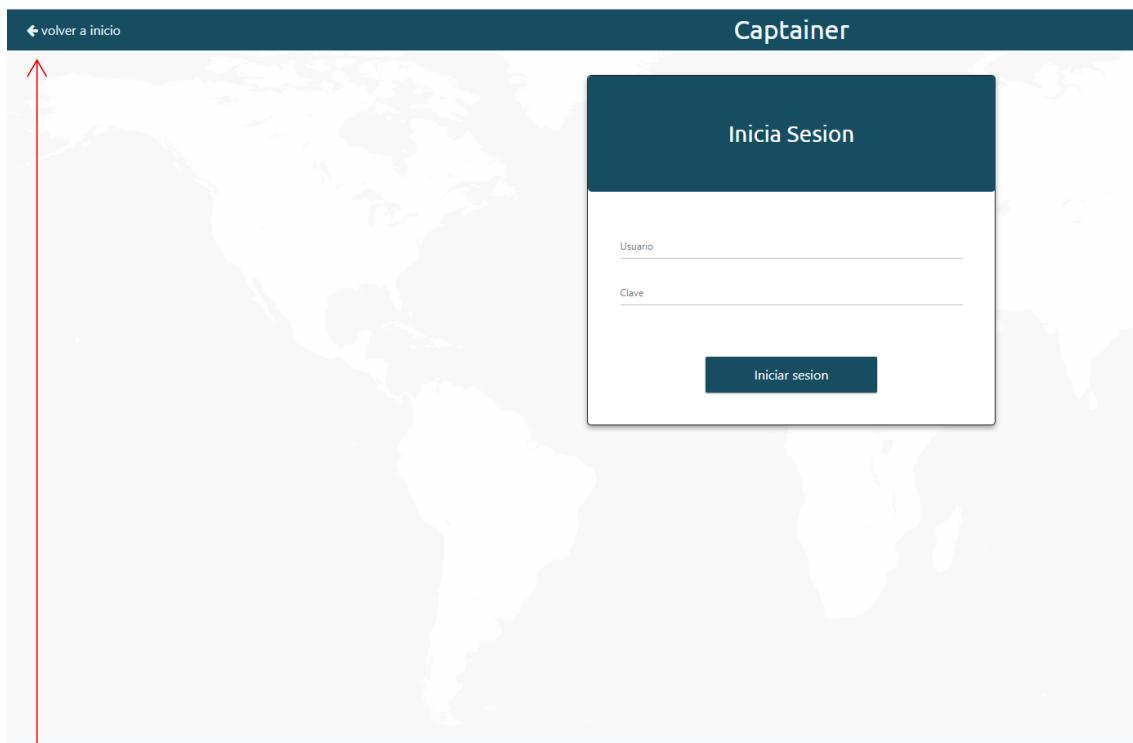
Para acceder al **menú inicial**, no tenemos más que poner la **dirección del servidor**, seguido de la **carpeta donde se ubica** la aplicación:



Para **iniciar sesión**, debemos hacer clic sobre el botón '**Iniciar Sesión**'

2. Ventana de inicio de sesión

Para **iniciar sesión**, debemos **escribir** nuestras credenciales en los campos '**usuario**' y '**clave**'. Pulsaremos el botón '**Iniciar sesión**' para validar la sesión y ser redirigidos al **panel administrativo**



Si deseamos **volver a la pantalla inicial**, bastará con hacer clic sobre el botón '**volver a inicio**'

3. Ventana principal

Una vez iniciemos sesión, se nos mostrará la **pantalla principal** de la aplicación, donde tendremos acceso a todas las funciones de la misma. Dentro de esta ventana, tendremos la opción de realizar las siguientes acciones:

1. Se nos **muestra** un panel con información acerca de las incidencias registradas en la base de datos. Si queremos **acceder a los detalles** de una incidencia, no tenemos más que **hacer clic sobre la incidencia deseada**.
2. Si queremos **registrar una incidencia**, no tenemos más que pulsar sobre el botón '**Añadir incidencia**'.
3. En la parte izquierda de la pantalla, encontramos un **menú lateral**, donde podremos **filtrar los resultados referentes a la incidencia por proyecto** al que pertenecen. Por otra parte, podemos acceder a la ventana de '**Opciones avanzadas**'.
4. En la parte superior izquierda de la pantalla encontramos una serie de botones cuya función es **ocultar la barra lateral** para que la información principal sea más accesible.
5. Si deseamos eliminar nuestra sesión, no tenemos más que pulsar sobre el botón '**Cerrar sesión**'.
6. Podemos **filtrar los contenidos de la pantalla** mediante la barra de búsqueda. Para ello, pinchamos sobre ella y **escribimos el nombre de la implantación** de la incidencia que deseamos buscar. Los resultados se filtrarán de forma dinámica según vamos acercándonos a la palabra completa.

The screenshot shows the Captainer application interface. The main area displays a table of 'ULTIMAS INCIDENCIAS' (Recent Incidents) with columns: Company Name, Deploy Name, Project Name, Contact Name, Issue Title, and Status. The sidebar on the left contains a navigation menu with items: Últimas Incidencias (3), Encolate, Logistic Docks, Opciones avanzadas, and Buscar... (6). A red number '4' is placed above the sidebar. A red number '2' is placed above the 'Añadir una incidencia' button. A red number '5' is placed above the 'Cerrar sesión' button. A red number '1' is placed at the bottom center of the main content area.

ULTIMAS INCIDENCIAS					
Company Name	Deploy Name	Project Name	Contact Name	Issue Title	Status
Laboratorio Referencia	San Isidro	Encolate 3.3	Manuel Lebron	La PC no prende	1
Laboratorio Referencia	Plantini	Encolate 3.3	Manuel Lebron	La aplicacion no recibe señal	1
Laboratorio Referencia	Plantini	Encolate 3.3	Manuel Lebron	Se producen reñidos inesperados	1
Carrefour Huelva S.A.	Carrefour Huelva	Logistic Dock	jesus javier Cocacolo	No se realiza un cambio de muelle	1
Carrefour Huelva S.A.	Carrefour Huelva	Logistic Dock	jesus javier Cocacolo	Los camiones se desconectan de la central al entrar en el punto de carga	1
Carrefour Huelva S.A.	Carrefour Huelva	Logistic Dock	jesus javier Cocacolo	Los camiones se desconectan en el punto logistico	1
Salvesen Logistica	Salvicense 1	Logistic Dock	Manolo Rodriguez Leon	la aplicacion se desconecta a los pocos minutos de iniciar	1
Salvesen Logistica	Salvicense 1	Logistic Dock	Manolo Rodriguez Leon	la aplicacion se bloquea al cerrar y querer volver a abrirla	1

4. Ventana de detalles

Al pulsar sobre cualquier incidencia, se nos mostrará una ventana con los **detalles de la incidencia** seleccionada. Dentro de la ventana, encontramos las siguientes opciones:

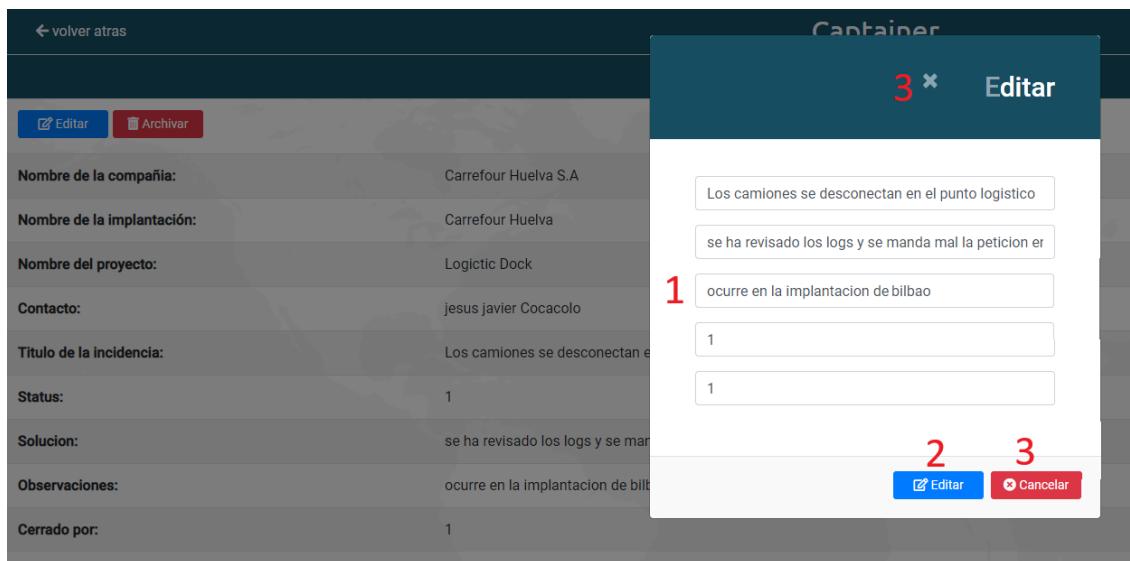
1. Si deseamos **volver a la ventana principal**, debemos pulsar sobre el botón '**Volver atrás**'
2. Podremos **editar** la información de la incidencia en cuestión, pulsando el botón '**Editar**'. Se nos mostrará el **modal de edición**.
3. Si lo deseamos, podemos **archivar la incidencia**, evitando que se muestre en la **lista de incidencias**, pulsando sobre el botón '**Archivar**'. Tenga en cuenta que una vez confirmado, la incidencia **ya no se mostrará en la lista**.

Nombre de la compañía:	Laboratorio Referencia
Nombre de la implantación:	San Isidro
Nombre del proyecto:	Encolate 3.3
Contacto:	Manuel Lebron
Título de la incidencia:	La PC no prende
Status:	1
Solución:	Nos desplazamos al lugar del cliente y comprobamos el hardware. El problema lo producía una memoria RAM en mal estado.
Observaciones:	El cliente no provee de mantenimiento al equipo correctamente
Cerrado por:	1

5. Formulario de edición

En caso de pulsar el botón **'Editar'**, se nos abrirá el **modal de edición**, donde se nos mostrará el formulario de edición de contenidos de la incidencia.

- 1.** Por defecto, los campos que se pueden editar **vienen rellenos con la información actual**. Podemos editar el contenido, simplemente seleccionando el campo que deseamos editar y **sobrescribiendo** el contenido.
- 2.** Si pulsamos sobre el botón '**Editar**', confirmaremos la edición del contenido de la incidencia, por lo que la información cambiará tras la pulsación del mismo.
- 3.** Si pulsamos sobre '**Cancelar**', o bien sobre la **cruceta** de la cabecera del modal, saldremos del menú de edición, **descartando** los posibles cambios.



6. Mensaje de confirmación de borrar incidencia

En caso de pulsar ‘Archivar’ en los detalles de una incidencia, la aplicación mostrará un modal con **un mensaje de confirmación**, en el cual se nos preguntará si estamos seguros del archivado de la incidencia. Tenga en cuenta que una vez archivada, esta **no se mostrará en la tabla de incidencias**.

- 1.** Si pulsamos sobre ‘Archivar’, la incidencia quedará archivada y **no será visualizada en la tabla de información**.
- 2.** Si pulsamos sobre **cancelar**, o bien, en la **cruceta** situada en la parte superior de la cabecera del modal, cerraremos el menú, **cancelando** el borrado de la incidencia.



7. Pantalla de añadir incidencia

Si en el menú principal, pulsamos sobre el botón ‘Añadir incidencia’, abriremos la ventana que contiene el formulario para **registrar una nueva incidencia** en la base de datos. En ella:

1. Podremos **rellenar** los campos que definen la información de la **incidencia** a registrar
2. Podremos **añadir** la **incidencia** una vez rellenos los campos, mediante el botón ‘**Añadir**’.
3. Podremos **limpiar** los **campos del formulario**, revirtiendo los cambios, pulsando sobre el botón ‘**Limpiar**’.

4

Captainer

REPORTAR NUEVA INCIDENCIA:

1

2 3

4

← volver atrás

Título de la incidencia

Solución al problema

Observaciones

Cerrado por:

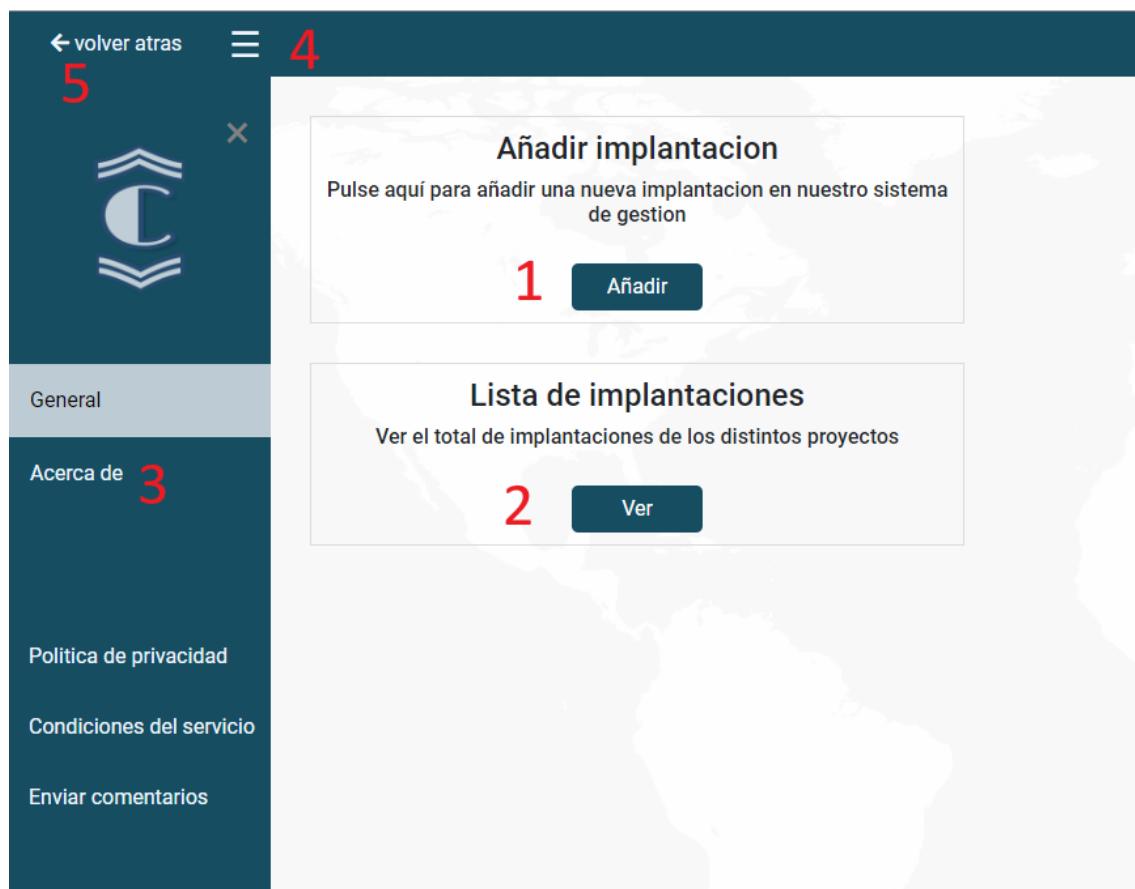
ID de implantacion:

Añadir **Limpiar**

8. Pantalla de opciones avanzadas

Desde el menú principal, podremos acceder a esta ventana pulsando sobre el elemento '**Opciones avanzadas**', situada en la barra lateral de navegación. Ya en esta ventana, podremos realizar las siguientes funciones:

1. Podremos **añadir una nueva implantación** en la base de datos, pulsando sobre el botón '**Añadir**'
2. Podremos **listar las actuales implantaciones registradas**, pulsando sobre el botón '**Ver**'.
3. En el menú lateral, podremos encontrar información adicional de la aplicación: versión e información del autor, pulsando sobre el elemento '**Acerca de**'.
4. Desde la parte superior izquierda de la ventana, podremos **ocultar la barra lateral de navegación** empleando los botones aquí situados.
5. Podremos **regresar** a la ventana principal, pulsado sobre '**Volver atrás**'



9. Ventana de listar implantaciones

Dentro de la ventana de **listar implantaciones**, podremos:

- 1.** **Visualizar** las implantaciones, las cuales se nos muestran en la tabla señalada en la imagen
- 2.** Podremos **editar** la implantación señalada, modificando los datos de la misma (En construcción)
- 3.** Podremos **archivar** una implantación registrada, siempre y cuando esta no contenga incidencias relacionadas con ella (En construcción)
- 4.** Podremos **regresar** a la ventana Opciones avanzadas, pulsando sobre el botón '**Volver atrás**',

LISTA DE IMPLANTACIONES						
Nombre	Status de soporte	ID de cliente	ID de proyecto	Licencia	Status	
San Isidro	1	1	1	L-540uAAAAAAAAAA	1	2 3 Editar Archivar
Plantini	0	1	1	L-562dAAAAAAAAAA	0	Editar Archivar
Ozama	1	1	1	L-5839AAAAAAAAAA	2	Editar Archivar
Carrefour Huelva	1	4	2	L-582dAAAAAAAAAA	1	Editar Archivar
Salvense 1	0	5	2	FHGE GHTJ WKDI IH01	0	Editar Archivar

10. Ventana de añadir implantaciones

Accederemos a esta ventana desde la ventana de **opciones avanzadas**, pulsando sobre el botón ‘**Añadir**’. En esta ventana, podremos **registrar una nueva implantación** en la base de datos, tras definir la información requerida para ello.

1. Podremos **rellenar** los campos que definen la información de la implantación a registrar
2. Podremos **añadir** la implantación una vez llenados los campos, mediante el botón ‘**Añadir**’.
3. Podremos **limpiar** los **campos del formulario**, revirtiendo los cambios, pulsando sobre el botón ‘**Limpiar**’.

← volver atrás 4

AÑADIR NUEVA IMPLANTACION:

Nombre de la implantacion	Introduce nombre de la implantacion
Licencia	Introduce licencia
Status de soporte	
Id Cliente	
Id Proyecto	
Status	

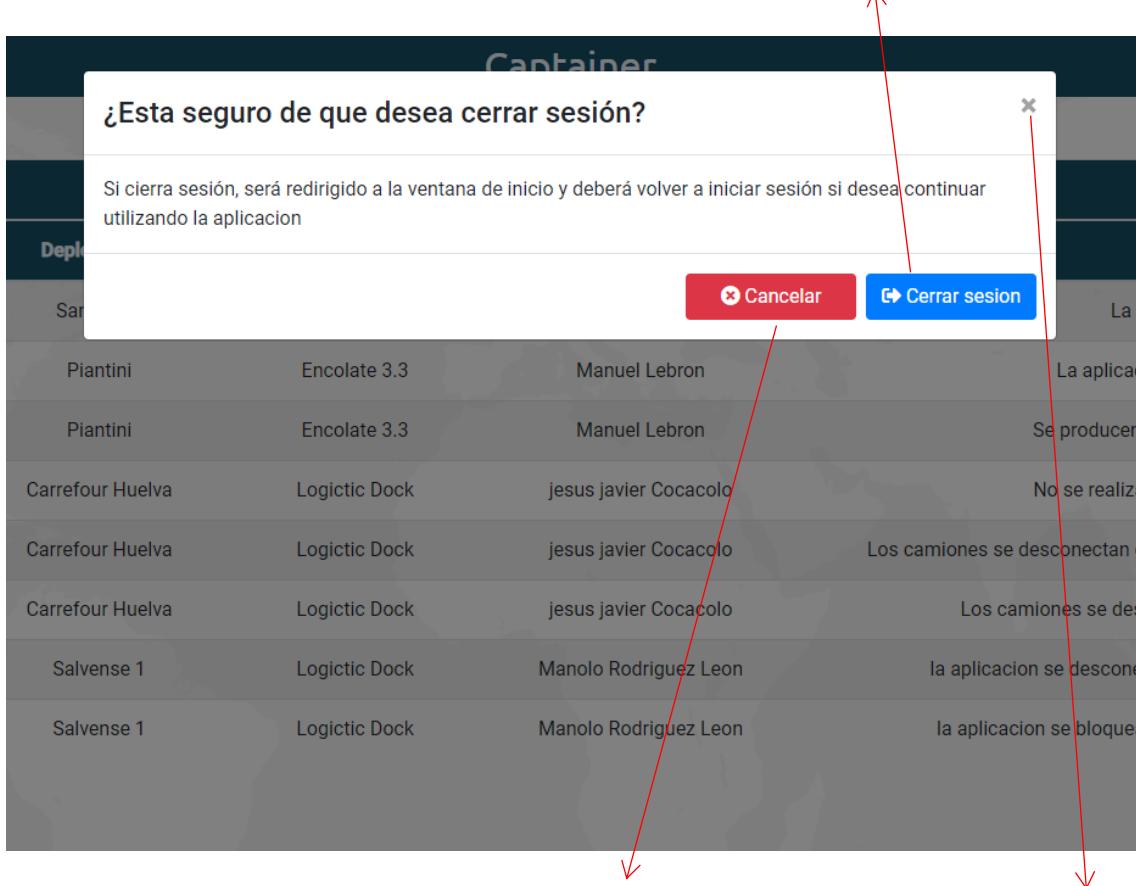
Añadir **Limpiar**

1
2 3

11. Mensaje de confirmación de cerrar sesión

Si queremos **abandonar la sesión actual**, desde el menú principal, pulsaremos sobre el botón ‘**Cerrar sesión**’. Se nos abrirá un **mensaje de confirmación**, advirtiéndonos que estamos a punto de cerrar una sesión activa.

Si queremos **confirmar** el cierre de sesión, pulsaremos sobre el botón ‘**cerrar sesión**’, situado dentro del **mensaje de confirmación**.



Si deseamos **rechazar el cierre de sesión**, debemos pulsar sobre el botón ‘**Cancelar**’, o bien, hacer clic sobre la **cruceta** situada en la parte superior del mensaje de confirmación.

12. Ventana de error de conexión

En caso de que la conexión con el servidor **se pierda** durante un periodo de tiempo mientras utilizamos la aplicación, ésta nos mostrará un mensaje de error a través de la siguiente ventana.

Es posible **recargar la aplicación**, haciendo clic sobre el botón '**Refrescar**' donde, en caso de **haber recuperado la conexión** con el servidor, nos permitirá regresar al **menú principal** de la aplicación.



Captainer

Manual de Administrador

para la puesta en marcha de la aplicación

Juan José González Fernández
Prisma Virtual S.G

2019

Introducción:

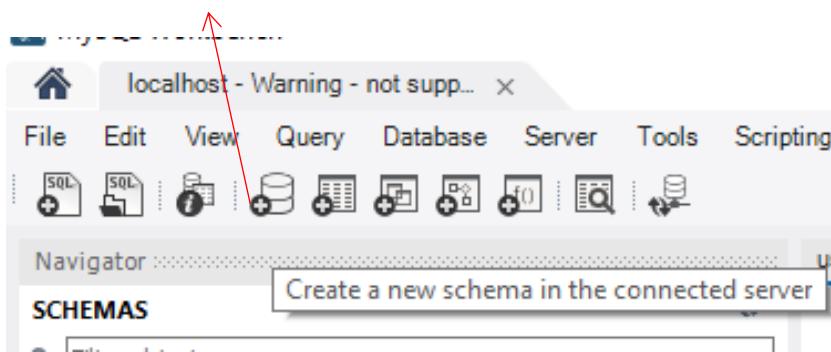
En este documento que expongo a continuación explicaremos los pasos necesarios para desplegar la aplicación web basada en Angular 7, dentro de un servidor. Donde, además, deberemos desplegar el Back-End del proyecto: bases de datos, modelo ORM, etc. Que sean necesarios para que la aplicación pueda funcionar correctamente dentro del servidor.

Dicho material va dirigido a los administradores de la empresa, así como los técnicos de implantación, responsables de desplegar la información de cara al uso empresarial, dentro del microentorno laboral.

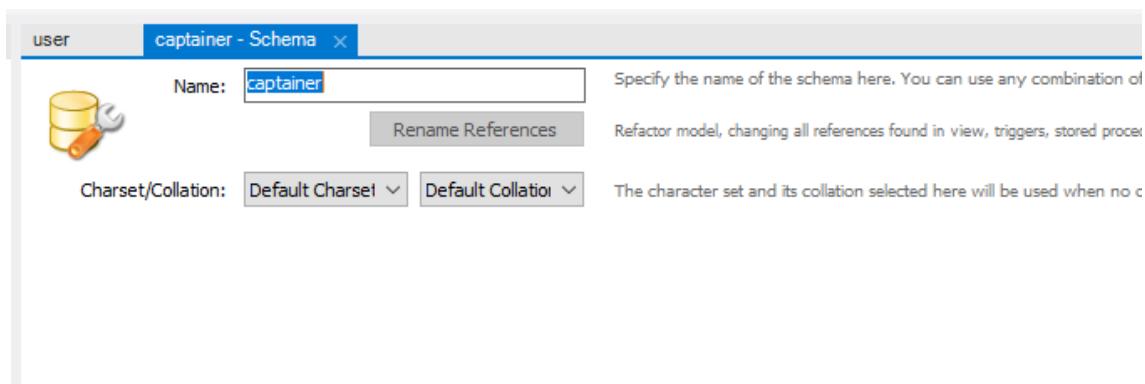
1. Configurar el servidor y crear la base de datos

Lo primero que debemos hacer es **levantar un servicio MySQL**, y acceder mediante una conexión al servidor MySQL, empleando **un gestor de Bases de datos**, como puede ser '**MySQL Workbench**'.

Debemos crear una **nueva base de datos** antes de comenzar el despliegue. Recomendamos hacerlo mediante el gestor de bases de datos, puesto que es más rápido e intuitivo.

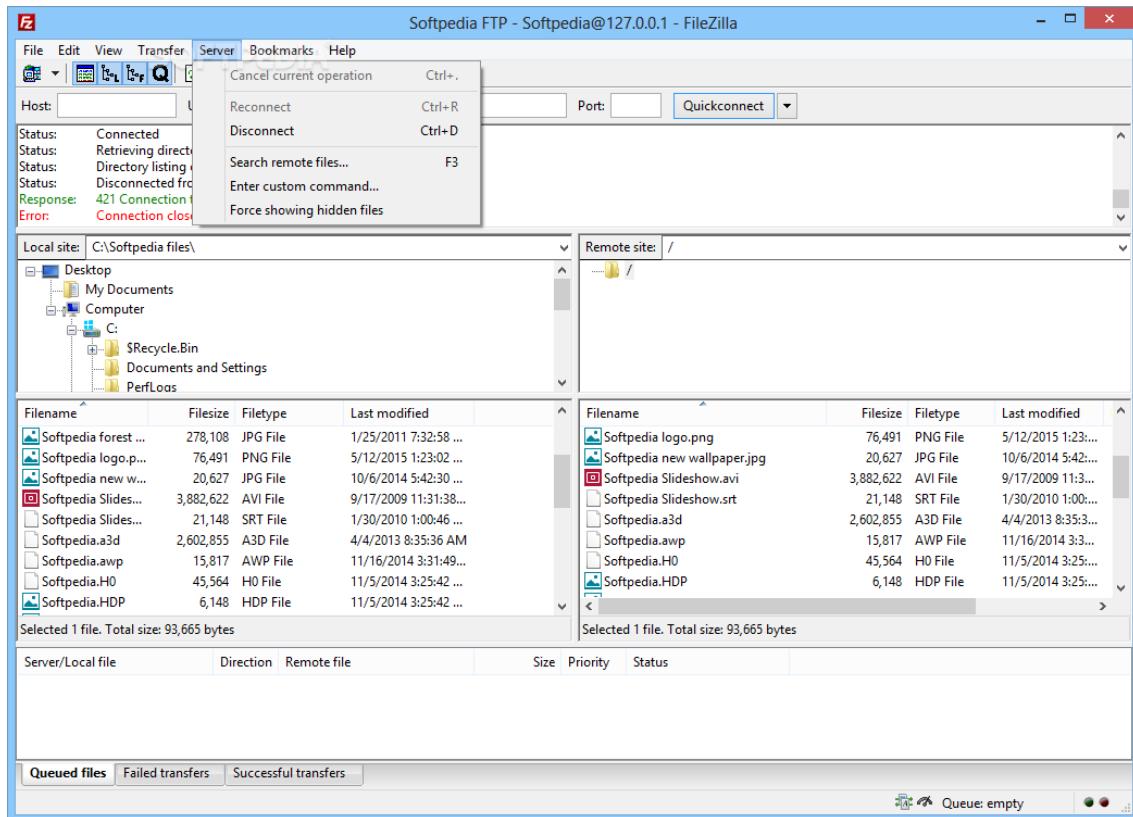


Ahora, solo tenemos que **definir el nombre de la base de datos** que deseamos crear. En nuestro caso, la llamaremos '**Captainer**'



Tras especificar el nombre de la base de datos a la que nos conectaremos, hacemos clic en 'Aplicar' y ya tendremos nuestra base de datos creada.

Ahora, vamos a **subir la carpeta** con el Back-End (Lumen), al **servidor**. Lo podremos hacer **mediante FTP**, empleando un cliente FTP como **Filezilla Client**.



Una vez subida la carpeta que contiene el proyecto de Lumen, en un directorio bien conocido, debemos especificar la **dirección del servidor**, así como la base de datos creada en él, **en el fichero ‘.env’** que contiene nuestro proyecto lumen en su directorio raíz:

```
DB_CONNECTION=mysql → "Tipo de conexión (ODBC empleado)"
DB_HOST=172.18.56.193 → "Dirección del servidor"
DB_PORT=3306 → "puerto por el que escucha"
DB_DATABASE=Captainer → "nombre de la base de datos"
DB_USERNAME=root → "usuario del administrador"
DB_PASSWORD= → "contraseña del administrador"
```

2. Desplegar las tablas en la base de datos mediante las *Migraciones*:

Si la configuración de nuestro fichero ‘.env’ es correcta, no deberíamos tener problemas para desplegar las **migraciones**.

Emplearemos el **gestor de comandos ‘Artisan’** para que realice todo el trabajo. Desde la raíz del proyecto Lumen, ejecutamos una **consola de comandos**. Lanzamos a la línea de comandos el siguiente comando:

```
php artisan migrate --force
```

Con este comando, ya tendremos desplegadas las tablas en la base de datos del servidor. Ahora, lanzaremos los ‘**Seeders**’ en aquellas tablas que deseamos inflar de contenido, como es el caso de la tabla ‘**user**’, donde insertaremos en un inicio los datos de los usuarios **administradores de la aplicación**:

```
php artisan db:seed
```

Ahora que ya tenemos las tablas creadas en el servidor, podremos consultarlas mediante un gestor de bases de datos, o continuar con el siguiente paso:

	id	name	email	password	role	status	photo
▶	1	administrador	admin@prismasmg.com	\$2y\$10\$EtIdV8UuqemXSLUctUsm1uyxWpR0Ube7/krpTdxMCVjBxxAa7SyYW	admin	1	sin_foto
	2	Responsable CL	admin@prismasmg.com	\$2y\$10\$960REhqeYNCgktrJlHOsbOCo62aAQzy1jpyRymw4E73v7arWrwzPO	admin	1	sin_foto
	3	Agencia	agency@prismasmg.com	\$2y\$10\$YM/Ht8CIVsr3vroxGmJG.PBcrzY94bONLAnNg27bwLYn99tWWUAS.	tec_imp	1	sin_foto
	4	Subcontratado	sub@prismasmg.com	\$2y\$10\$NrrZMTAgtkUO16.71kh/OrXGKOjMDdcMHxEkhIRamPXXZ7WO/xF6m	tec_fac	0	sin_foto

Es importante que nos aseguremos de las rutas de la aplicación en el servidor, ya que será indispensable para el correcto uso de las rutas que vamos a emplear.

3. Compilar nuestra aplicación para producción:

Ahora que nuestra aplicación se encuentra desarrollada, podremos **compilarla** para que esta sea usada de forma pública en nuestro **servidor**. Para ello, empleamos el siguiente comando en la **carpeta raíz de nuestro proyecto Angular**:

```
ng build --prod --aot --build-optimizer --base-href=/captainer /
```

En nuestro caso, la **dirección y nombre de la carpeta** es ‘/Captainer’. Puesto que hemos especificado esta ruta en el nombre del comando, a la hora de colocar nuestra aplicación en nuestro servidor Apache, **debemos respetar esta dirección**, ubicando la carpeta generada tras la compilación en la raíz de ‘htdocs’. También es muy importante que en nuestra aplicación Angular **hayamos definido rutas relativas**. De otra manera, las peticiones **no se realizarán correctamente**, puesto que la ubicación de la aplicación respecto a la dirección del Back-End va a cambiar.

Aunque hayamos optimizado el código y eliminado todos los errores que han ido surgiendo durante la fase de desarrollo, es posible que, a la hora de lanzar la aplicación en **modo de producción**, **aparezcan nuevos errores** que, hasta ahora, no se habían manifestado; a pesar de haber tenido abierto en todo momento el servidor virtual (‘ng serve’) durante el desarrollo.

```

chunk {1} es2015-polyfills.c5dd28b362270c767b34c.js (es2015-polyfills) 50.4 kB [initial] [rendered]
chunk {2} main.fa4681da67b0e16d34dc.js (main) 128 bytes [initial] [rendered]
chunk {3} polyfills.4159ea504b9f00b6dea.js (polyfills) 130 bytes [initial] [rendered]
chunk {4} styles.97aa336b28d847f3b99e.css (styles) 61.7 kB [initial] [rendered]

ERROR in src\app\add-site\add-site.component.html(28,94): : Expected 1 arguments, but got 0.
src\app\add-site\add-site.component.html(38,15): : Expected 1 arguments, but got 0.
src\app\add-site\add-site.component.html(50,145): : Expected 1 arguments, but got 0.
src\app\add-site\add-site.component.html(64,15): : Expected 1 arguments, but got 0.
src\app\add-site\add-site.component.html(74,15): : Expected 1 arguments, but got 0.
src\app\add-deploy\add-deploy.component.html(26,40): : Expected 1 arguments, but got 0.
src\app\add-deploy\add-deploy.component.html(35,46): : Expected 1 arguments, but got 0.
src\app\add-deploy\add-deploy.component.html(78,11): : Expected 1 arguments, but got 0.

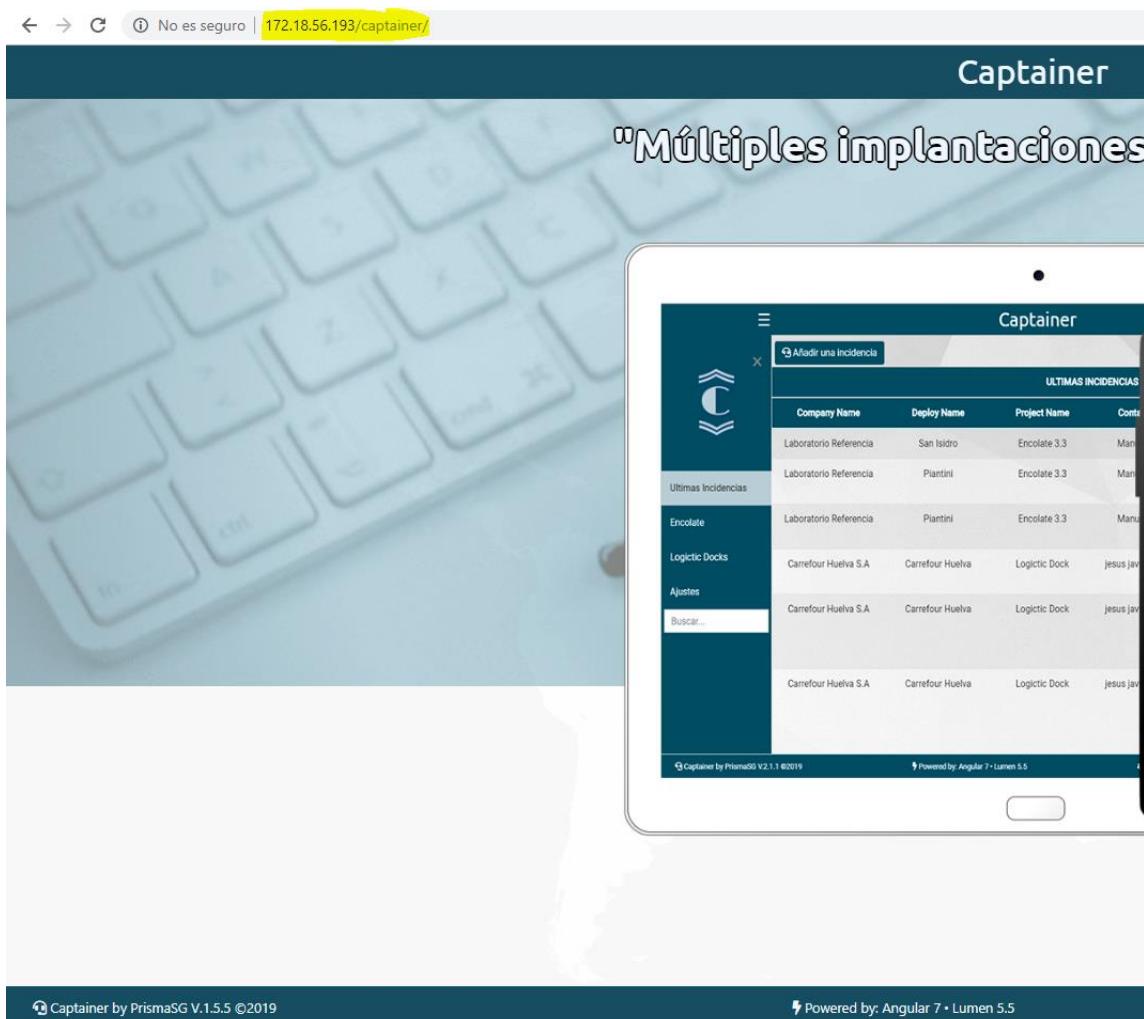
```

Para que nuestra aplicación pueda ser compilada para producción, **debemos solucionar todos estos errores e intentar nuevamente la compilación**. Todo esto será repetido hasta que la aplicación no sea víctima de ningún error y **acabe compilada correctamente**. La aplicación compilada será generada en la carpeta ‘**dist**’ de nuestro proyecto Angular.

	Nombre	Fecha de modificación	Tipo	Tamaño
ido	assets	07/06/2019 12:46	Carpeta de archivos	
s	3rdpartylicenses	07/06/2019 12:46	Documento de te...	33 KB
ntos	bocet.bb964dd6d4fe1069e1ec	07/06/2019 12:46	Archivo PNG	525 KB
;	es2015-polyfills.c5dd28b362270c767b34	07/06/2019 12:46	JetBrains WebStorm	57 KB
úblico	favicon	07/06/2019 12:46	Icono	67 KB
	index	07/06/2019 12:46	Archivo HTML	4 KB
	main.69552c1df05b16828cff	07/06/2019 12:46	JetBrains WebStorm	750 KB
	polyfills.8bbb231b43165d65d357	07/06/2019 12:46	JetBrains WebStorm	42 KB
	runtime.26209474bfa8dc87a77c	07/06/2019 12:46	JetBrains WebStorm	2 KB

4. Desplegar la aplicación en el servidor

Ahora que ya tenemos preparado el entorno para el despliegue, no tenemos más que **repetir la acción de transferir mediante FTP**. Esta vez, la carpeta ‘**captainer**’, que contiene nuestra aplicación ya compilada. Haciendo uso de una aplicación FTP, como **Filezilla Client**, **vamos a desplegar la aplicación en una carpeta situada en la raíz del servidor, cuyo nombre debe ser ‘captainer’ por definición**. Una vez desplegada, vamos a **ingresar en la dirección del servidor** y a comprobar que la aplicación se encuentra desplegada:



Nuestra aplicación se encuentra **desplegada** y lista para ser usada por los equipos que comparten red local con el servidor de la empresa. Si queremos hacer que nuestra dirección sea más profesional, no tenemos más que definir un nombre para la dirección en el fichero '**.htaccess**'.