



BACKOFFICE FOR REAL USERS

2021

Proyecto Fin de Ciclo

Desarrollo de Aplicaciones Web

Francisco Rodríguez Serrano
Didáctica Ágil Centros

Comencé esta investigación por el conocimiento que tengo directa e indirectamente con el sector del mantenimiento y he buscado y perseguido simplificar el trabajo al usuario final, los técnicos y encargados de mantenimiento. Los resultados obtenidos han sido muy satisfactorios y a pesar de haberme encontrado con numerosos contratiempos, los he resuelto a base de investigación y “hardworking”.

M-tile es un ERP desarrollado para ser distribuido como un *SaaS (Software as a Service)*.

El objetivo de esta aplicación es facilitar el control y la gestión del mantenimiento de una fábrica, principalmente del sector cerámico, aunque permitiría adaptarse con ligeras modificaciones a otro ámbito industrial.

Gracias a **M-tile**, cada técnico de mantenimiento puede gestionar desde un ordenador o dispositivo móvil sus operaciones, fichajes, ubicación de la incidencia, materiales usados en las intervenciones y observaciones, entre otras funciones.

Estas funcionalidades permiten que se registren todas las intervenciones de los técnicos de una forma ordenada, simplificando las tareas de gestión y control a las personas encargadas del mantenimiento.

Además de las funcionalidades mencionadas, **M-tile** genera estadísticas automáticamente, alertas cuando algún material está cerca de agotarse, permite almacenar documentaciones de maquinaria, avisos para el resto de técnicos y organizar mantenimientos, reparaciones y tareas con antelación asignados a un técnico en particular, de forma que el mismo sólo tiene que consultar el calendario para conocer sus tareas asignadas.

Cabe destacar que la aplicación **M-tile** ha sido programada con lenguajes de programación y frameworks actuales, lo que hace que sea una aplicación, ágil, veloz y sencilla de mantener.

Abstract

I started this research because I have plenty of experience in the maintenance area, and I was looking for ways to improve the tasks of the final customer, the technicians and chiefs of maintenance. In my opinion, the results are excellent, and in spite of the troubles, I have solved it with investigation and hard working.

M-tile is an application ERP developed to be used as a *SaaS (Software as a Service)*. The application is focused on make easy the management of a factory, mainly tile factories, but with slight changes the application could fit in another areas.

Thanks to **M-tile**, every maintenance technician will be able to manage from a mobile device or a computer, his/her clock in and clock out, interventions, place of the breakdown, the items used in the interventions and set commentaries as notes.

This attributes allow to register every single operation of a technician with an orderly manner, doing very easy the management of the maintenance to chiefs and supervisors.

In addition to the mentioned functionalities, **M-tile** generates automated statistics, alerts when items warehouse are under the minimum, creates notices for the rest of technicians, organizes in advance maintenances, fixings and tasks assigned to a single technician and that makes tasks easier to the technicians.

Finally, emphasis must be placed on the programming languages that have been used to make the application **M-tile**, making it safe, agile and very fast.

Resumen	1
Abstract	2
Índice de capítulos	3
Índice de tablas	5
Índice de imágenes	6
1. Introducción	8
2. Objetivos	9
2.1. Objetivo general	9
2.2. Objetivos específicos	9
3. Estado actual del problema	10
4. Solución propuesta	11
4.1. Análisis de requisitos	11
4.1.1. Actores del sistema	11
4.1.2. Requisitos funcionales	12
4.1.3. Requisitos no funcionales	13
4.2. Tecnologías que se usan	14
4.2.1. Frontend	14
4.2.2. Backend	15
4.2.3. Software	17
4.3. Modelos	18
4.3.1. Diagrama de casos de uso	18
4.3.2. Diagrama entidad-relación	19
4.3.3. Esquema base de datos	20
4.3.4. Modelo de arquitectura	21
4.4. Implementación	22
4.4.1. Login Screen	24
4.4.2. Dashboard Screen	25
4.4.3. Admin Screen	26
4.4.4. Order Screen	28
4.4.5. Crew Screen	30
4.4.6. Calendar Screen	31
4.4.7. Docs Screen	34
4.4.8. Warehouse Screen	36
4.4.9. Statistics Screen	38

4.4.10. Historical Screen	39
4.4.11. Implementación del Backend	41
4.5. Experimentación	45
4.6. Requisitos de funcionamiento	46
4.7. Conclusiones	46
5. Estudio de mercado	47
5.1. Idea de negocio	47
5.2. Análisis DAFO	48
5.3. Fuentes de financiación	48
5.4. Marketing	49
6. Trabajos futuros	51
7. Bibliografía	52
8. Anexos	53
8.1. Manual de instalación	53
8.1.1. Instalación en servidor	53
8.1.2. Instalación con docker-compose	54
8.2. Manual de uso	55
8.2.1. Formulario Login	55
8.2.2. Formulario Creación/Actualizar orden	55
8.2.3. Sección Administración	55
8.2.4. Sección Técnicos	56
8.2.5. Sección Calendario	56
8.2.6. Sección Documentaciones	56
8.2.7. Sección Almacén	57
8.2.8. Sección Estadísticas	57
8.2.9. Sección Histórico	57
8.3. Solución de posibles errores	58

Índice de tablas

Tabla 4.1.1. Listado de actores	11
Tabla 4.1.2.. Listado de requisitos funcionales	12
Tabla 4.1.3. Listado de requisitos no funcionales	11
Tabla 4.4.1. Listado endpoints de la API	42
Tabla 5.2.1. Análisis DAFO	48
Tabla 5.4.1. Marketing mix o 4ps	49
Tabla 8.3.1. Posibles errores y soluciones	58

Índice de imágenes

Imagen 4.2.1 Logotipo de React	14
Imagen 4.2.2 Logotipo de Redux	15
Imagen 4.2.3 Captura de los estados con Redux	15
Imagen 4.2.4 Logotipo de SASS	15
Imagen 4.2.5 Logotipo de NodeJS.....	16
Imagen 4.2.5 Logotipo de MySQL.....	16
Imagen 4.2.6 Logotipo de phpMyAdmin	16
Imagen 4.2.7 Logotipo de VSCode	17
Imagen 4.2.8 Logotipo de XAMMP	17
Imagen 4.2.9 Logotipo de Chrome, Firefox y Safari	17
Imagen 4.2.10 Extensión Redux Devtools para Chrome	17
Imagen 4.3.1 Diagrama de casos de uso	18
Imagen 4.3.2 Diagrama de entidad-relación	19
Imagen 4.3.3 Esquema de la base de datos	20
Imagen 4.3.4 Ejemplo de funcionamiento patrón React + Redux	21
Imagen 4.4.1 Nav normal, nav extendido, nav versión móvil	23
Imagen 4.4.2 Login Screen	24
Imagen 4.4.3 Control de errores en formularios	24
Imagen 4.4.4 Captura de Dashboard Screen	25
Imagen 4.4.5 Ejemplo consulta a la BBDD	26
Imagen 4.4.6 Modal para agregar avisos	26
Imagen 4.4.7 Captura de Admin Screen	27
Imagen 4.4.8 Confirmación de borrados de órdenes y usuarios	27
Imagen 4.4.9 Captura de Order Screen	28
Imagen 4.4.10 Mensaje resaltado para guiar al usuario	29
Imagen 4.4.11 Modales para la adición de operaciones, fichajes y materiales	29
Imagen 4.4.12 Captura de Crew Screen	30
Imagen 4.4.13 Modal para la creación y edición técnico	31
Imagen 4.4.14 Ejemplo de colores al agregar nuevas órdenes	31
Imagen 4.4.15 Captura del calendario – mes	32
Imagen 4.4.16 Captura del calendario – semana	33
Imagen 4.4.17 Captura del calendario – día	33
Imagen 4.4.18 Captura de Doc Screen	34
Imagen 4.4.19 Mostrando cuando no hay documentos en la selección	34

Imagen 4.4.20 Mostrando cuando hay documentos existente en la selección	35
Imagen 4.4.21 Vista de un documento	35
Imagen 4.4.22 Modal para la subida de ficheros	36
Imagen 4.4.23 Acceso al buscador del almacén	36
Imagen 4.4.24 Tabla al buscar un item en el almacén	37
Imagen 4.4.25 Modales para editar y agregar items	38
Imagen 4.4.26 Captura del sistema de alertas de stock	38
Imagen 4.4.27 Captura de Statistics Screen	38
Imagen 4.4.28 Fragmento de código CSS	39
Imagen 4.4.29 Historical Screen	39
Imagen 4.4.30 Modal ver orden usada en varios componentes	40
Imagen 4.4.31 Directorios del Backend	41

1. Introducción

¿Cuántas empresas continúan en 2021 haciendo uso de hojas de cálculo para gran multitud de tareas? ¿Existen alternativas reales para evitar esta excesiva dependencia? ¿Puede el mantenimiento de una empresa ser completamente gestionado en hojas de cálculo? Estas preguntas y alguna más son las que me han llevado a considerar y crear una aplicación desde la que se pueda gestionar el mantenimiento mecánico y eléctrico de una empresa, con el fin de unificar todas las tareas en una sola aplicación y que además permita ser usada desde cualquier lugar y dispositivo.

Las hojas de cálculo es un tipo de software con una interfaz bastante amigable y que permiten con un poco de formación e investigación lograr muy buenos resultados para diversas tareas. Y la realidad es que con programas como Excel se pueden crear aplicaciones suficientemente robustas para llevar a cabo labores relativamente complejas, como puede ser un cuadro de mando para generar órdenes de trabajo o creación de gráficos de estadísticas que cambian dinámicamente. Por otro lado también permiten visualizar y analizar datos de una forma rápida, importar datos, trabajar en equipo con una misma hoja de cálculo, llevar la contabilidad, resolver fórmulas matemáticas y almacenar datos como si de una base de datos se tratase, además de muchas otras funciones.

Lo cierto es que hay todavía muchas compañías que usan programas como *Excel*, *Calc* o *Spreadsheets de Google* para muchas de sus tareas, y en cierto modo es lógico y comprensible que empresas de pequeño tamaño y/o con un bajo presupuesto, hagan uso de ellos, pero no lo es tanto para empresas de un tamaño considerablemente, que a día de hoy todavía usan software de este tipo para su día a día, bien a veces por desconocimiento, y otras muchas para ahorrar en gastos que más tarde indirectamente se convertirán en pérdidas, debido al tiempo consumido en busca de información, por no estar bien organizada e incluso incompleta.

Gracias a que durante muchos años he trabajado como electromecánico en varias empresas, tanto del sector cerámico, como del sector químico, he conocido de cerca estas carencias que tienen algunas de ellas en lo relativo al mantenimiento. En algunos casos no existe ni si quiera un control mediante hojas de cálculo, si no que amontonan partes de papel, que más tarde se pierden por algún archivador acumulando polvo. El problema ocurre cuando se necesita acceder a esa información de nuevo, porque se invierte tiempo y dinero para la empresa en la búsqueda de un simple papel, y que en multitud de ocasiones termina la búsqueda en nada. Esa es la motivación principal por la que decidí comenzar el desarrollo de esta aplicación: *facilitar las tareas de control y gestión de mantenimiento* a las personas encargas de ello.

2. Objetivos

2.1. Objetivo General

El objetivo general es proporcionar a las empresas y a las personas encargadas del control y gestión del mantenimiento, un software que les permita llevarlo a cabo de una forma intuitiva, simple, rápida, y que contemple todas las características necesarias que requiere el sector.

2.2. Objetivos Específicos

- ▶ Proporcionar acceso desde cualquier lugar y dispositivo.
- ▶ Establecer una interfaz intuitiva y fácil de usar.
- ▶ Obtener datos de una forma rápida y coherente.
- ▶ Almacenar datos de una forma consistente.
- ▶ Proporcionar estadísticas.
- ▶ Controlar el inventario que será usado por el personal de mantenimiento.
- ▶ Clasificar documentación de la maquinaria y facilitar su acceso.
- ▶ Controlar movimientos y operaciones de los técnicos.
- ▶ Proporcionar un histórico de las intervenciones.

3. Estado actual del problema

¿Cuántas empresas continúan en 2021 haciendo uso de hojas de cálculo para gran multitud de tareas? ¿Existen alternativas reales para evitar esta excesiva dependencia? ¿Puede el mantenimiento de una empresa ser completamente gestionado en hojas de cálculo?

La situación actual es que en muchísimas empresas, los departamentos de mantenimiento dependen excesivamente de hojas de cálculo para el registro de sus actividades, por no hablar de que, otro porcentaje todavía muy alto, siguen haciendo uso de papel para estos registros. Esto acaba generando numerosos inconvenientes que se traducen en pérdidas de tiempo y dinero para la empresa y frustración para los técnicos y responsables.

Existe una gran cantidad de software para la gestión del mantenimiento de una empresa, e incluso podemos encontrar muchas aplicaciones en su versión “*free*”, pero apenas ofrecen características de uso libre, incluyen anuncios y el número de usuarios es muy limitado. En cambio si se decide pagar una licencia, descubriremos software mucho más completo, el problema es que al ser aplicaciones bastante genéricas, no siempre se adaptan a las necesidades reales de la empresa y gran parte de las características de la aplicación no le serán de utilidad. Además hay que tener en cuenta en que los precios no suelen ser bajos, y un software a medida es inviable para muchas compañías.

Estos inconvenientes podrían solventarse con **M-tile**, un software desarrollado completamente para usarlo en la nube, adaptado al sector cerámico, sin necesidad de instalaciones ni equipos adicionales y con un bajo coste.

4. Solución propuesta

4.1. Análisis de requisitos

Para que la aplicación sea una buena elección por las empresas del sector cerámico, debe de cumplir unos requisitos para cumplir con una necesidades muy específicas. Estas necesidades van a ser representadas como *requisitos funcionales*, *requisitos no funcionales* y *los actores que intervienen en el sistema*, y para obtener estas necesidades, he trabajado junto con el coordinador de mantenimiento de un importante grupo del sector cerámico para identificar los problemas y definir un sistema que los solucione.

4.1.1. Actores del sistema:

Listado de Actores	
Nombre	Descripción
Técnico de Mantenimiento	Es la persona que más información introduce en la aplicación. Sus funciones son la de ir creando órdenes de trabajo y cumplimentarlas con la información necesaria durante su jornada laboral. También tendrá acceso a estadísticas, histórico de órdenes, podrá hacer consultas de lectura del stock actual del almacén, generar avisos visibles para el resto de usuarios y visualización de documentación de la maquinaria.
Responsable Mantenimiento	Es la persona encargada de controlar toda la información introducida por los técnicos en las órdenes de trabajo y cerrar las órdenes cuando hayan sido resueltas las incidencias. Adicionalmente, puede ver y actualizar el stock del almacén, ver y actualizar la información personal de los técnicos a su cargo, gestión de usuarios, gestión de documentación de la maquinaria, generar avisos e impresión de estadísticas.
Desarrollador	Es la persona dedicada a la creación de la aplicación y de realizar todas las funciones necesarias para que los otros actores puedan llevarlas a cabo. Además debe de realizar las pruebas unitarias y todas las necesarias para que el comportamiento de la aplicación sea el deseado.

Tabla 4.1.1. Listado de Actores de la aplicación

4.1.2. Requisitos funcionales:

Listado de requisitos funcionales		
Acción	Actor	Descripción
Consultar, crear, editar y borrar órdenes de trabajo.	Técnico de Mantenimiento/Responsable Mantenimiento	La aplicación permitirá que los usuarios puedan realizar gestiones con órdenes de trabajo
Consultar calendario de órdenes.	Técnico de Mantenimiento/Responsable Mantenimiento	La aplicación permitirá que los usuarios puedan consultar el calendario en el que aparecerán las órdenes.
Consultar, insertar y borrar documentación de maquinaria.	Técnico de Mantenimiento/Responsable Mantenimiento	La aplicación permitirá que los usuarios puedan gestionar los archivos de documentaciones
Consultar, crear y borrado de avisos.	Técnico de Mantenimiento/Responsable Mantenimiento	La aplicación permitirá que los usuarios puedan realizar realizar inserciones y borrados de avisos.
Gestión de usuarios que pueden acceder a la aplicación.	Responsable Mantenimiento	La aplicación permitirá que los usuarios con poderes de administrador puedan realizar gestiones sobre los usuarios.
Acceso a la aplicación desde cualquier ubicación y dispositivo.	Técnico de Mantenimiento/Responsable Mantenimiento	La aplicación debe estar alojada en la nube y que permita el acceso a usuarios registrados.
Consultar, actualizar y eliminar productos del stock del almacén.	Técnico de Mantenimiento/Responsable Mantenimiento	La aplicación debe incluir una sección para la gestión de un almacén.
Visualización global de un histórico de órdenes de trabajo.	Técnico de Mantenimiento/Responsable Mantenimiento	La aplicación presentará una sección desde la que se puedan ver el histórico de las órdenes de trabajo.
Consultar gráficos de la productividad y de las actividades de los técnicos.	Técnico de Mantenimiento/Responsable Mantenimiento	La aplicación debe de contemplar una sección desde la que se puedan gestionar estadísticas

Navegar con un menú lateral expansible y con versión responsiva.	Técnico de Mantenimiento/Responsable Mantenimiento	La aplicación debe de poder cambiar entre secciones por un menú lateral y el mismo debe de tener una versión para dispositivos móviles.
Gestionar la sección de los técnicos.	Responsable Mantenimiento	La aplicación debe de tener una sección para poder gestionar toda la información de los técnicos, así como su horario y anotaciones.

Tabla 4.1.2. Listado de requisitos funcionales

Hay que destacar que estos requisitos funcionales han sido generalizados en relación al rol del usuario, es decir, habrá algunas de las funciones mencionadas en las que el responsable de mantenimiento tendrá privilegios de crear, editar y borrar respecto a los técnicos, que sólo podrán hacer operaciones de lectura.

4.1.3. Requisitos no funcionales:

Listado de requisitos no funcionales	
Tipo	Descripción
De hardware	La aplicación debe de poder ejecutarse en dispositivos móviles de gama media y en ordenadores con recursos limitados.
De software	En cuanto a los navegadores web, debe de ser 100% compatible con Safari, Firefox y Chrome, y también en sus versiones para móvil.
De rendimiento	El sistema debe de estar funcionando las 24 horas del día durante todo el año, debido a que la actividad del personal de mantenimiento es continua.
De rendimiento	La aplicación debe de ofrecer tiempos de respuesta ágiles, si bien es cierto, el número de usuarios al mismo tiempo será muy pequeño, por lo que los requisitos del servidor serán bajos.
De soporte	A causa de que el área de mantenimiento siempre está funcionando, necesitan tener un servicio de ayuda al para posibles incidencias del sistema, al menos de lunes a viernes.
De seguridad	La aplicación debe de ser robusta en cuanto a seguridad, y sólo podrán acceder a ella usuarios con autenticación.

Tabla 4.1.3. Listado de requisitos no funcionales

4.2. Tecnologías que se usan

A continuación un análisis de las tecnologías usadas para la aplicación web, divididas entre *Frontend*, *Backend* y *Software*. Aunque se podría haber realizado el proyecto con un framework enfocado a dispositivos móviles, finalmente he decidido hacerlo en reversa, una aplicación web pensada para ordenador, pero desarrollada como *First-Mobile* para que se puedan realizar gestiones desde un dispositivo móvil con total normalidad.

4.2.1. Frontend

Este proyecto se va a realizar como una *SPA* por sus siglas (*Single Page Application*), para reducir tiempos de carga y ganar en agilidad. Para llevarlo a cabo, hubiera sido una muy buena opción hacer uso de *Angular*, ya que este framework se apoya de *Typescript*, que gracias a su fuerte tipado, hace que las aplicaciones desarrolladas con él, sean de una gran fiabilidad. Sin embargo, lo que la librería *React* propone, la creación de componentes muy pequeños e integrarlos en otros más grandes, el ser tan liviano y el poder realizar cambios en el *DOM* de una forma muy eficiente gracias al *Virtual DOM*, hizo inclinarme a desarrollar la aplicación con esta librería, eso sí, siempre hay contras, y en este caso es que requiere de bastantes dependencias.



Imagen 4.2.1 Logotipo de React

React, como he mencionado anteriormente, es una librería y no un framework, que hace uso de *JSX*, que el resultado es similar a como meter en una trituradora *JavaScript* y *HTML* juntos durante unos segundos. Esta extensión fue creada por *Facebook* para usarla con *React* y sirve para transformar el código a *JavaScript*. Tiene sus peculiaridades como por ejemplo que el atributo de *HTML* class pasa a ser className, porque esto confundiría al *pre-compilador*, ya que una clase en *JavaScript*, al igual que en otros muchos lenguajes, se define como class.

Como puntos negativos de *React*, es que al no ser un framework como tal, requiere de algunas librerías adicionales para parecerse más a un framework, como *react-router-dom*, para habilitar las rutas en la aplicación.

Una de estas librerías que he incorporado con *React*, y que se va a convertir casi en un estándar, es *Redux*, que a grandes rasgos, establece un store global en el que se almacenan los estados de la



Redux

Imagen 4.2.2 Logotipo de Redux.

aplicación, y sustituyen al *Provider* que se suele usar en *React* para hacer uso de *Context*, pero con *Redux* se evita estar haciendo herencia en cascada a los componentes hijos para que tengan acceso a los estados de la aplicación, de forma que cualquier componente en la aplicación, puede acceder a estos estados, lo cuál es sublime para el desarrollador, porque cuando necesita algún estado de la aplicación, sólo tiene que consultarlo gracias al *Hook useSelector*.

```
▶ auth (pin): { checking: false, loadingLogin: false, uid: "AKLJNCKLUJ..." }
▶ nav (pin): { navExtended: false, showResponsive: false }
▶ calendar (pin): { events: [...], types: [...], breakdowns: [...], ... }
▶ ui (pin): { modalOpen: false }
▶ crew (pin): { technicians: [...], activeTechnician: null }
▶ warehouse (pin): { items: [], activeItem: null }
▶ factory (pin): { factories: [...], sections: [...], machines: [...], ... }
▶ warning (pin): { warnings: [...] }
```

Imagen 4.2.3 Ejemplo de los estados con Redux.

Por otro lado en cuanto a estilos se refiere, *CSS* habría sido una buena opción, e incorporar una librería como *Bootstrap* o *Tailwind* también habría sido de gran ayuda, al evitar que



Imagen 4.2.4 Logotipo SASS.

definir muchas clases y estilos, pero como la aplicación ya iba a requerir de muchas dependencias, he tomado la decisión de no incorporarlas y realizar todos los estilos íntegramente. Además, en lugar de *CSS*, he decido usar *SASS* por los grandes beneficios que aporta, como poder crear variables de una forma sencilla, funciones, y por lo parecido que se escribe respecto a la programación, de forma anidada.

4.2.2. Backend

Para el lado del servidor, la elección era clara, *NodeJS* junto con *MongoDB*, pero debido a la complejidad de algunas consultas y de algunos requisitos de la aplicación, finalmente di un paso atrás para implementar una base de datos relacional, en este caso, *MySQL*.

Y por qué *NodeJS*? Es una respuesta muy clara. En mi caso, que ya poseo bastante experiencia y conocimientos de *JavaScript* y *EcmaScript* 6, programar en *NodeJS* es una tarea

relativamente sencilla, aunque también he tenido que investigar y aprender cómo resolver múltiples problemas.



Imagen 4.2.4 Logotipo de NodeJS.

Para este *Backend*, la idea inicial era hacerlo como una *FULL API Rest*, de forma que el *Frontend* pudiera estar alojado en un sitio, y el *Backend* en otro sitio distinto, para que sean independientes, pero cómo algunos requisitos de la aplicación incluían poder subir fotos de los técnicos, y documentación técnica interna de la maquinaria de la empresa, era un poco arriesgado subir esos ficheros a otros lugares y al final la decisión ha sido de que sea el *Backend* quien sirva una carpeta pública, que es dónde se encuentra el *Frontend*, y dónde se suben los archivos que el cliente necesita. De cualquier forma, modificando que al subir esos ficheros, se alojen en cualquier otro lugar y obteniendo su URL para almacenarla como dato en la base de datos, el *Backend* y *Frontend* pasarían a ser independientes, y desde cualquier lugar con un *JWT* (*Json Web Token*) válido, y con los parámetros que necesita el *endpoint*, podrían obtenerse la información desde la API.

Respecto a la base de datos, *MongoDB* habría aportado mucha rapidez, pero este proyecto requiere bastantes relaciones entre tablas, que en una base de datos no relacional, hubiera sido complicado de tratar, y hacer consultas complejas hubiera requerido de mucha ciencia y “carpintería” para llevarla a cabo, y esto es un trabajo muy laborioso que no hubiera compensado lo suficiente.



Imagen 4.2.5 Logotipo de MySQL.



Imagen 4.2.6 Logotipo de phpMyAdmin.

Finalmente para la gestión de esta base de datos, he seleccionado *phpMyAdmin*, por su sencilla instalación y por la forma intuitiva en que te muestra la base de datos. Este servicio será accesible desde cualquier lugar, de forma que permita a la persona que tenga acceso, acceder desde cualquier lugar, aunque la idea inicial es que la *BBDD*, no la manipule nadie.

4.2.3. Software

Por la parte de software y como entorno de desarrollo he utilizado exclusivamente VSCode (Visual Studio Code), el cuál gracias a la gran cantidad de agregando, se puede cualquier lenguaje o que lo desarrolla y lo mantiene en un funcionamiento muy bueno y compatibilidad con prácticamente todo.



Imagen 4.2.7 Logotipo de VSCode.

es un editor muy polivalente y extensiones que se le pueden ir programar en prácticamente tecnología. Además la empresa es Microsoft, lo que se traduce



Imagen 4.2.8 Logotipo de de XAMPP

Acompañando a VSCode, he utilizado el entorno XAMPP, para montar un servidor web en local que me permitiera tener en funcionamiento: un servidor apache con phpMyAdmin y MySQL para la BBDD. Este entorno es imprescindible si no

quieres estar instalando todo desde la línea de comandos mediante una terminal, y ahorra mucho tiempo.

Como navegadores web, he utilizado en un 90%, y el otro 10% ha sido sobre Mozilla Firefox y Safari. Estos últimos los uso para comprobar que todas las funcionalidades que he ido implementando en la aplicación, funciones debidamente. Respecto a Chrome



Imagen 4.2.9 Logotipo Chrome, Firefox y Safari.

junto con React y Redux, son absolutamente imprescindible dos extensiones: React Developer Tools y Redux Developer Tools. Desde ellas puede consultar los estados de la aplicación y ver el Store, que muchas veces, se hace estrictamente necesario para ver si las funcionalidades desarrolladas cumplen su propósito al crear los estados.

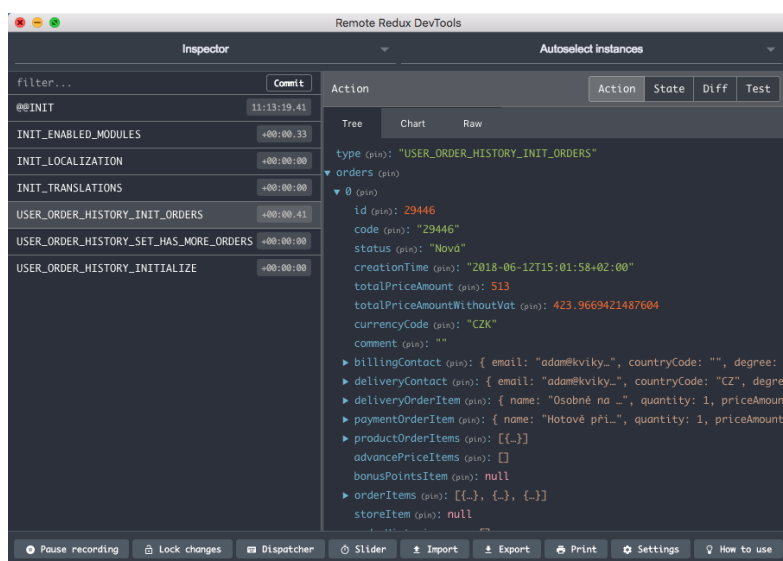


Imagen 4.2.10 Extensión Redux Devtools para Chrome.

4.3. Modelos

En este apartado vamos a hablar del diagrama de casos de uso y del diagrama de entidad-relación. También vamos a ver el esquema de tablas de la base de datos, para ver todas sus tablas, campos y relaciones y finalmente pasaré a explicar brevemente el modelo de arquitectura que usa React junto con Redux y cómo es su funcionamiento.

4.3.1 Diagrama de Casos de uso

El diagrama de casos de uso es una de los diagramas más usados de los diagramas *UML* (*Unified Modelling Language*), pertenece al grupo de los diagramas de comportamiento y se utilizan para definir el comportamiento que se espera que cumpla un software o un sistema.

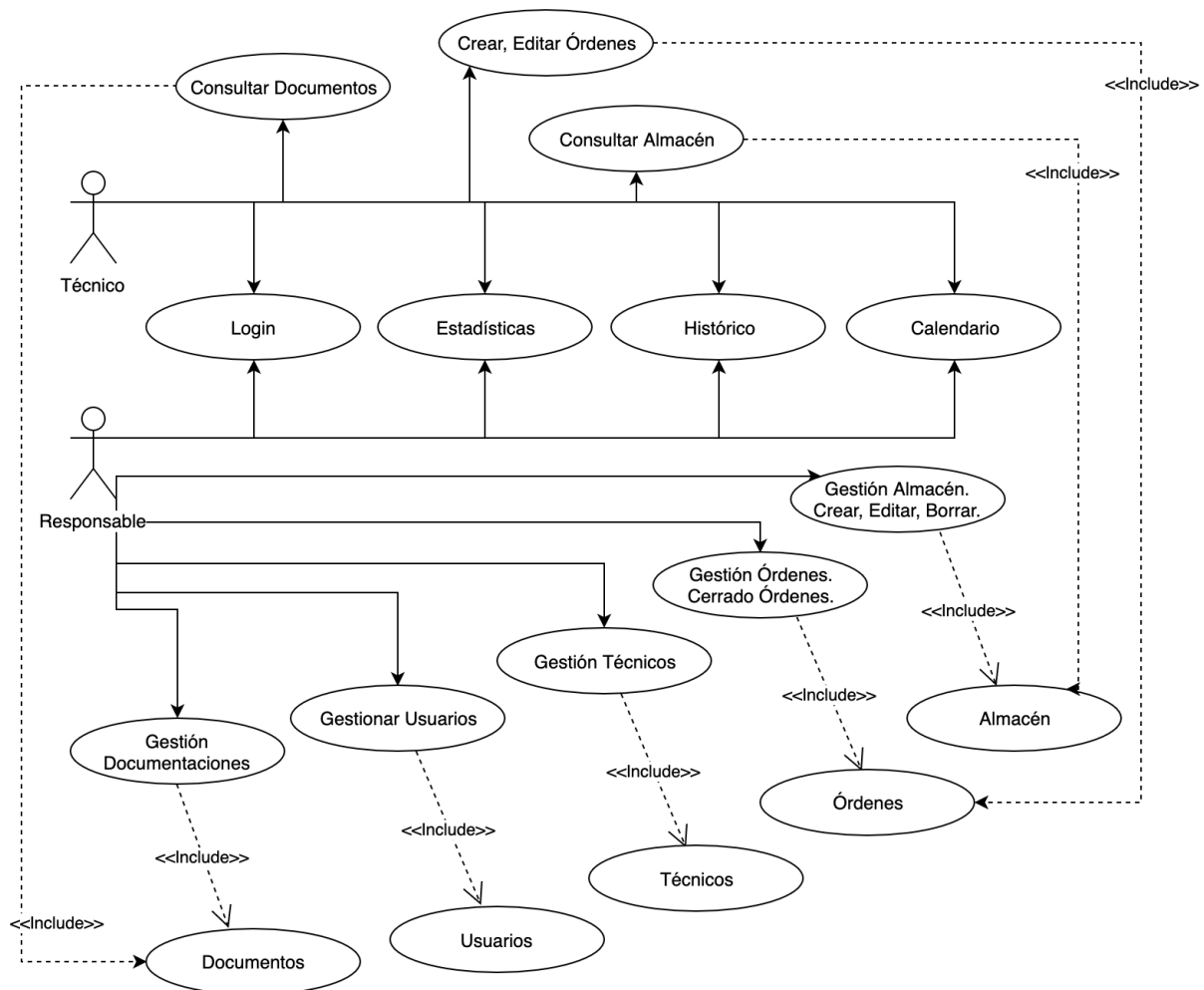


Imagen 4.3.1 Diagrama de casos de uso.

4.3.2. Diagrama entidad-relación

El diagrama entidad-relación también conocido como *ERD* es un tipo de los diagrama de flujo y su función es la de representar entidades que se relacionan dentro de un sistema. Estos diagramas se usan sobretodo para diseñar bases de datos e incluso para optimizarlas.

En el esquema siguiente podremos observar: entidades (representadas por un rectángulo), relaciones, (representadas por un diamante) y atributos (representados como un óvalo).

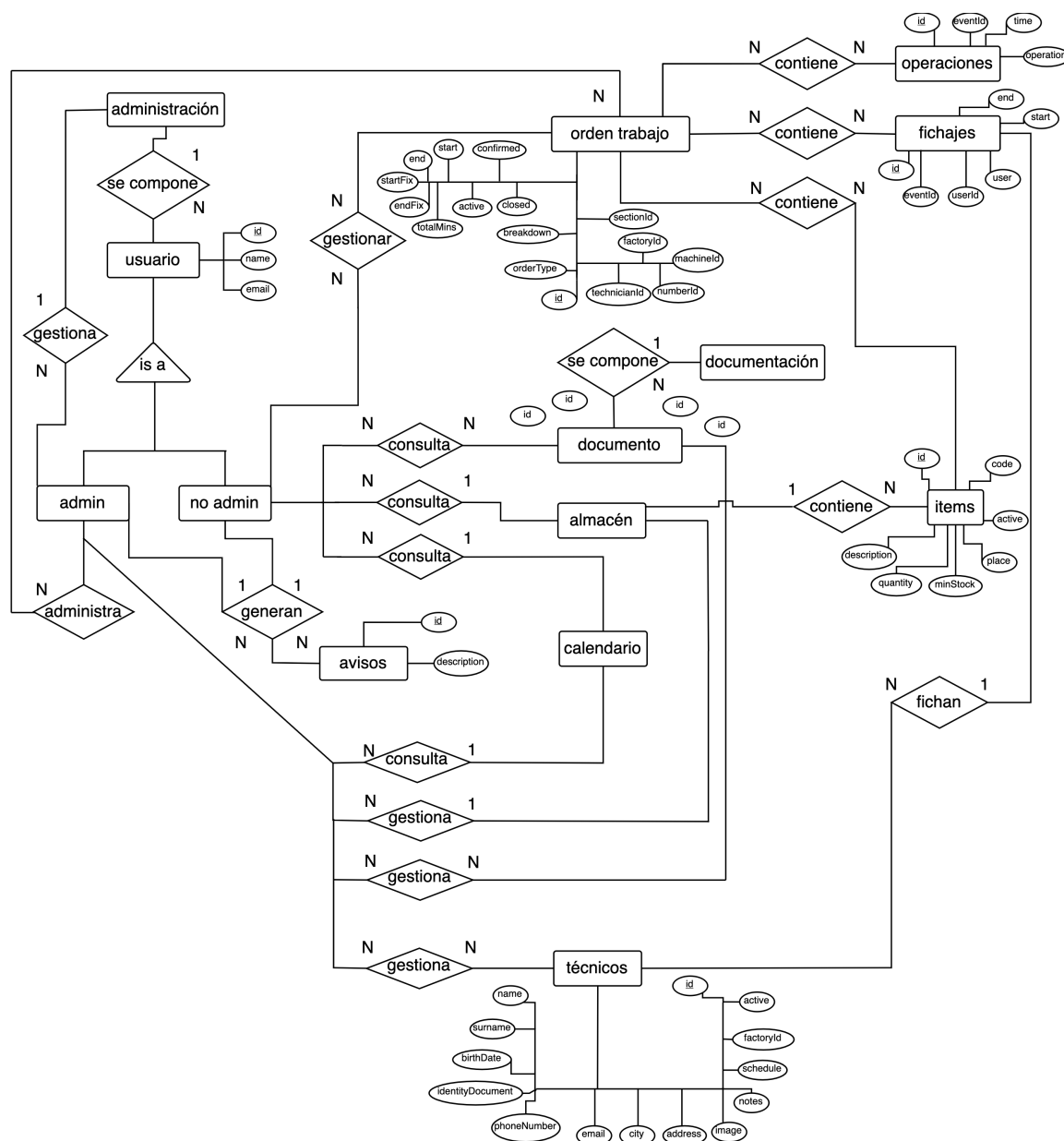


Imagen 4.3.2 Diagrama de entidad-relación.

4.3.3 Esquema Base de Datos

Este tipo de esquema, representa de forma visual la base de datos, y es generado por el SGDBD, es decir, el sistema gestor de la base de datos. En el esquema podemos observar cada tabla con su nombre, y dentro de las tablas cada columna con su respectivo nombre, del tipo que es la columna y su longitud máxima si procede, y por último podemos ver su clave primaria (normalmente representada por un ID único), y las llaves secundarias.

Las claves primarias suelen utilizarse como claves secundarias o compuestas en otras tablas para establecer relaciones y esto es lo que le da sentido de funcionamiento a una base de datos relacional.

Por otro lado tenemos que se recomienda que las bases de datos relacionales estén en la tercera forma normal, 3FN. Esta base de datos se encuentra en 2FN porque hay alguna tabla que repite algún dato constantemente, y esto no se considera buena práctica. Lo correcto hubiera sido generar una nueva tabla con los valores que se repiten y con su id único establecer una relación a la tabla en la que se está repitiendo el mismo dato, pero no lo he hecho por no complicar más el credo, y como las veces que se repites es un número muy pequeño y fijo, no es de suma importancia.

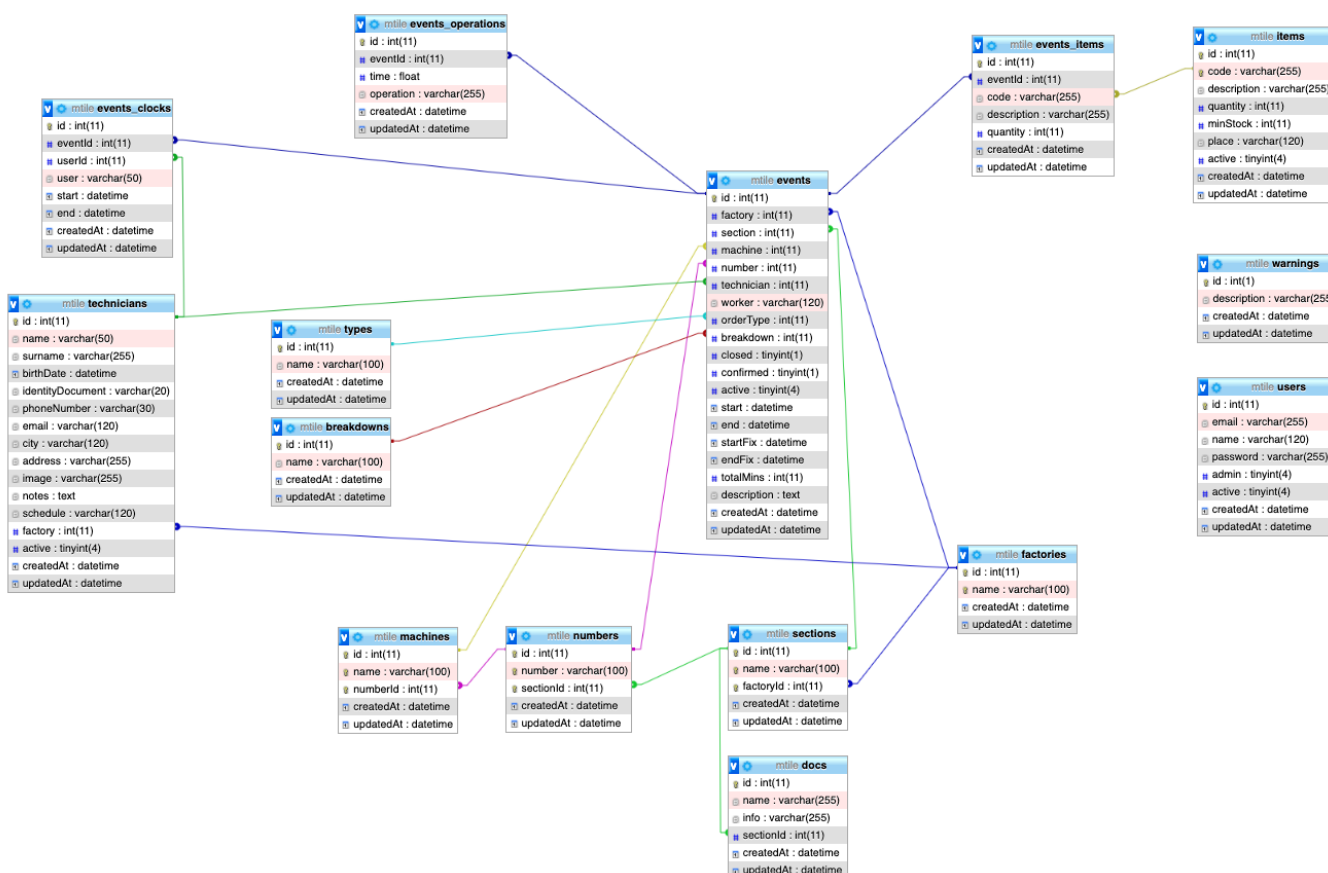


Imagen 4.3.3 Esquema de la base de datos.

4.3.4. Modelo de arquitectura

En este proyecto he utilizado *React* junto con *Redux* (Además de *NodeJS* y *MySQL*), con lo que el modelo de arquitectura usado es muy similar al que creó *Facebook* con *Flux*, es decir, el flujo de los datos de la aplicación van en una sola dirección, y no en ambas direcciones como ocurre en la mayoría de frameworks que hacen uso de *MVC* o *MVVM*.

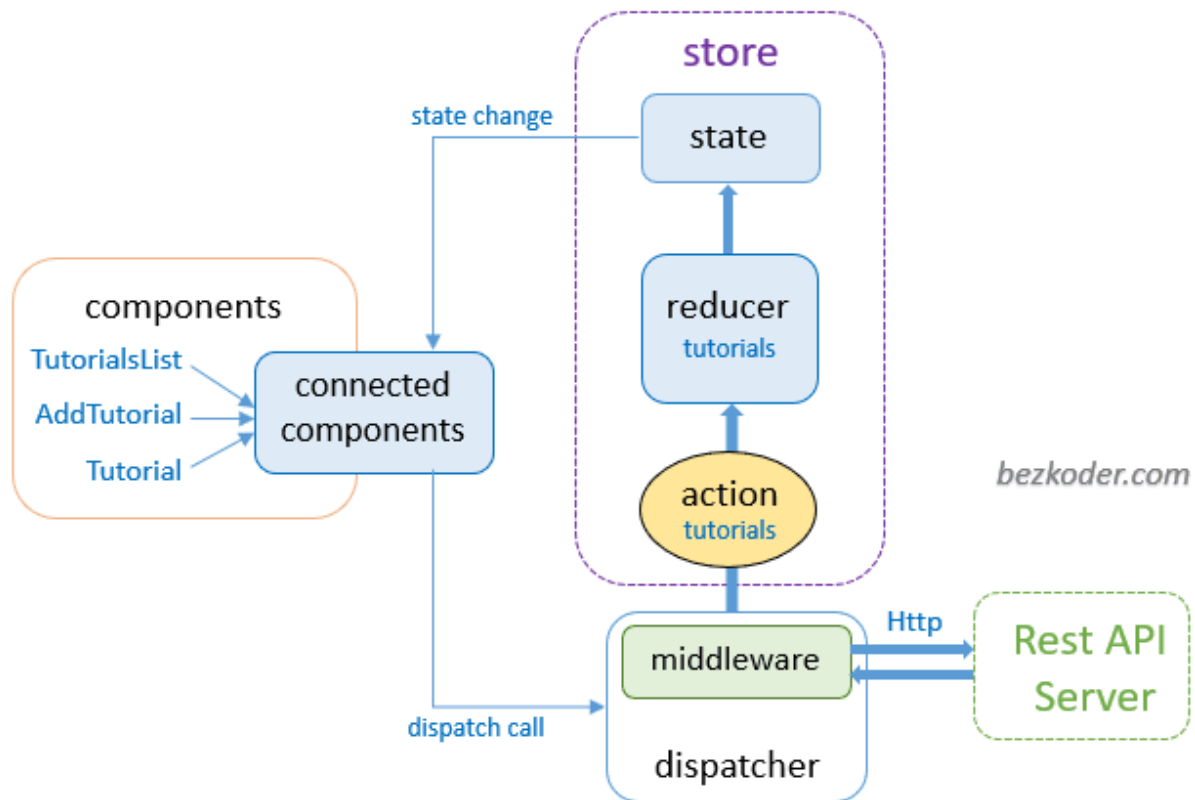


Imagen 4.3.4 Ejemplo de como funciona el patrón React + Redux.

Según *Facebook*, *React* es la *V* de *MVC*, con lo que con *React* sólo tenemos la vista. Al agregar *Redux*, *NodeJS* y *MySQL* da como resultado una arquitectura similar a *MVC* como en otros frameworks. Como dato, algunas fuentes mencionan esta arquitectura como *VVM*, porque es *React*, la vista, quién hace de vista y controlador y media con el Modelo (*NodeJS* + *MySQL*).

La forma de trabajar de *Redux* es que la vista dispara unas acciones que hemos definido previamente y son lanzadas mediante un dispatcher. Estas acciones una vez resueltas, bien de forma síncrona o asíncrona, se envían a unos reducers, que no son más que funciones puras, que en pocas palabras, se limitan a modificar los estados. Algo peculiar de estos reducers es que no hacen peticiones, no cambian las vistas, no llaman a otras funciones, ni ejecutan nada asíncrono.

4.4. Implementación

Mi idea inicial era centralizar toda la información y datos en una sola aplicación desplegada en la nube, de forma que cualquier usuario con los permisos necesarios pueda acceder a ella, consultar información, grabar, editar y borrar. Al estar la aplicación en un servidor en la nube, el cliente no debe preocuparse de cómo y dónde distribuirla. Por otro lado también podría suponerle un ahorro en materiales debido a que no necesitaba un servidor para servir la aplicación, y otro de los beneficios es que los backups son continuos y están almacenados también en la nube, lo que aporta un extra en seguridad.

Antes de comenzar a desarrollar la aplicación, lo primero que me puse a trabajar, fue la formación e investigación para programar en *React*, más aún porque *React* había cambiado la forma de trabajar. Ahora se usaba la programación funcional y el uso de los nuevos *Hooks*. Desde *React* y *Facebook* recomendaban programar de esta nueva forma, por lo que comencé el proyecto haciendo un curso sobre React en actualizado en 2021, de más de 40 horas de vídeo, y una vez completado, estaba mucho más preparado para construir la aplicación, y de paso aprendí muchas características nuevas de EcmaScript 6.

Una vez finalizada la formación, el siguiente paso fue dibujar en papel todo el *layout* de la aplicación en su diseño de ordenador y en forma adaptada para dispositivo móvil, con el fin de evitar fallos en el diseño, e intentar programar las cosas una sola vez.

Respecto a los colores desde un primer momento los tenía bastante claros, azul, blanco y algún gris, para más adelante agregar un naranja que en mi opinión le da un toque distintivo.

Después de hacer todo este I+D, llegué a la conclusión de como iba a quedar la estructura de la web con su *routing*, y el resultado es el siguiente a falta de la página Login:

- ▶ Sección Dashboard (Home de la aplicación).
- ▶ Sección de Administración (Gestión de Usuarios y Eliminación de Órdenes).
- ▶ Sección Creación de Órdenes (Formulario para creación de Órdenes).
- ▶ Sección de Técnicos (Gestión de los Técnicos).
- ▶ Sección Calendario (Visualización global de las Órdenes).
- ▶ Sección Documentación Técnica (Visualización de documentos).
- ▶ Sección Almacén (Control del Almacén de Mantenimiento).
- ▶ Sección Estadísticas (Visualización de Estadísticas globales).
- ▶ Sección Histórico (Ver Órdenes de forma ordenada).

Estas rutas son para todos los usuarios registrados, a excepción de la sección de Administración, a la que sólo pueden acceder usuarios con el rol de Admin activado. Una vez definidas estas rutas el nav quedó de la forma siguiente en sus diferentes versiones: normal, extendido, y versión móvil.



Imagen 4.4.1 Nav normal, nav extendido y nav versión móvil.

El siguiente paso fue traducir cada página de mis dibujos, a HTML, SASS y JavaScript, tarea laboriosa pero muy satisfactoria cuando ves la interfaz terminada.

A continuación voy a ilustrar cada una de las páginas ya terminadas. No he llevado una cuenta exacta del tiempo que me ha llevado cada cosa pero más o menos el tiempo de desarrollo completo de estas páginas, layout y la lógica para mostrar datos estáticos, me ha llevado cerca de 1 mes dedicando de media unas 10 horas al día, incluyendo fines de semana.

4.4.1 Login Screen

Esta es la zona de acceso visible a cualquiera que conozca la dirección de la aplicación.

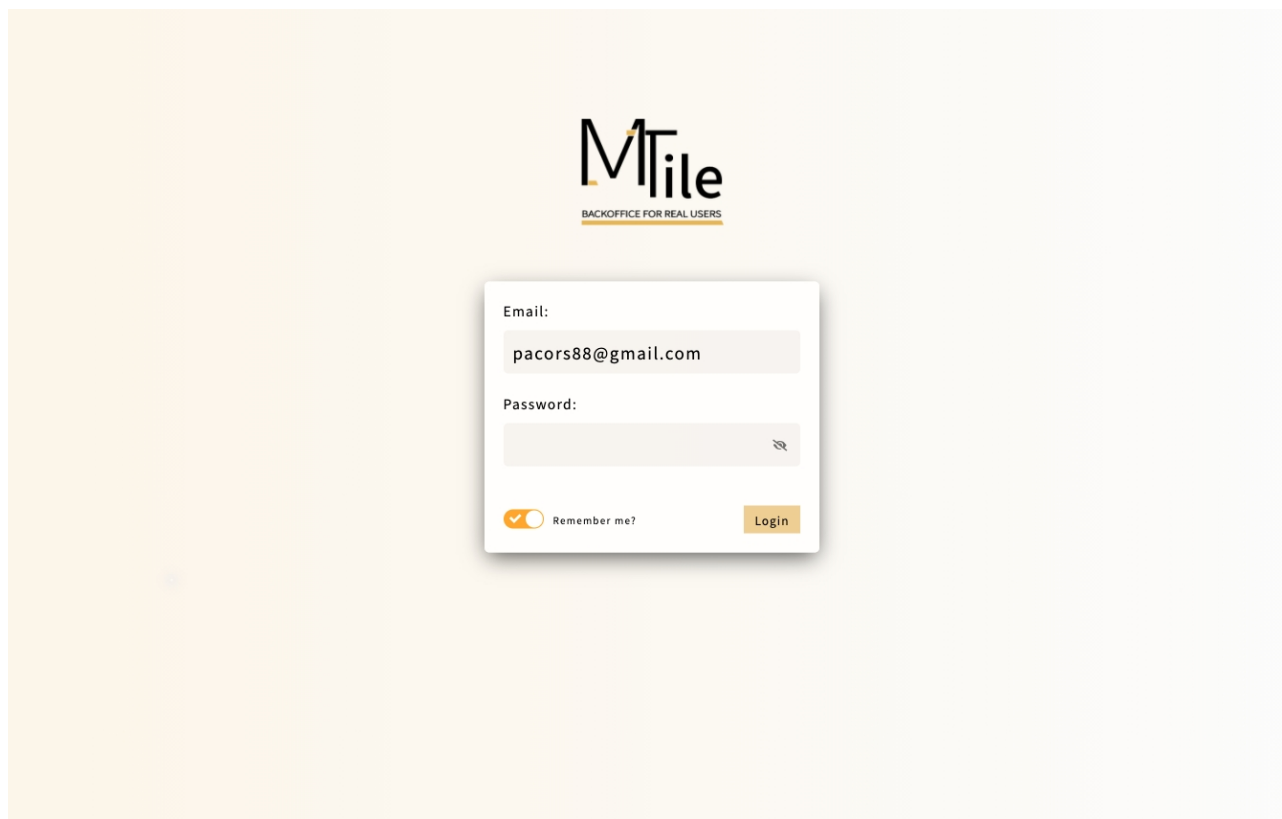


Imagen 4.4.2 Login Screen.

Todos los formularios de la aplicación tienen control de errores tanto en Frontend como en Backend y en caso de error en la conexión o que los datos sean erróneos, nos avisará con una alerta. En cuanto a los estilos, todos siguen el mismo cauce, por lo que no voy a ilustrar más debido a que son muy similares, para ello me he servido de el paquete `npm react-toast`.

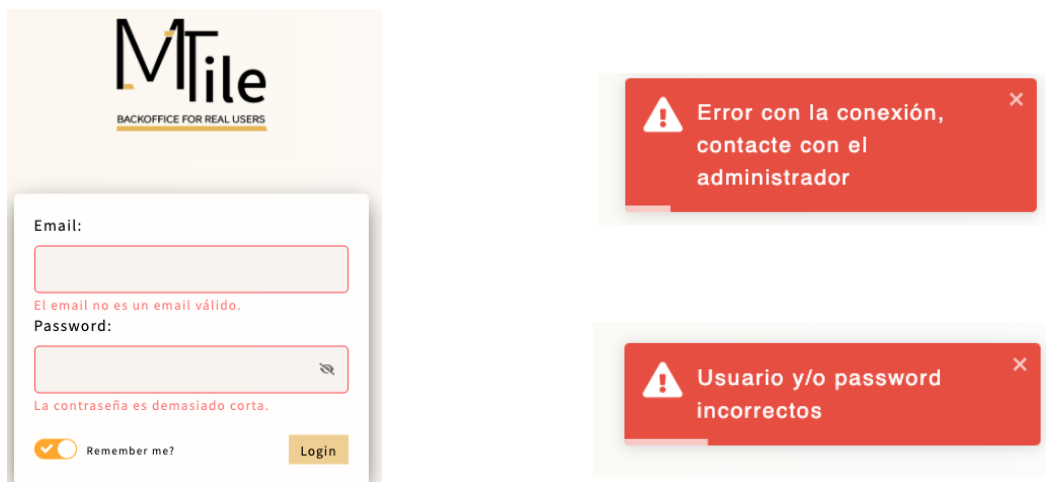


Imagen 4.4.3 Control de errores de los formularios.

4.4.2. Dashboard Screen

Esta es la primera pantalla que aparece al hacer login en la aplicación y dentro podemos ver una barra superior (presente en toda la aplicación). Esta barra incluye el logotipo de **M-tile**, el nombre de usuario, un icono desde el que se muestran alertas, y un icono para el logout. Más abajo no encontramos con dos gráficas de estadísticas, una tabla con las órdenes existentes que siguen abiertas y la sección de avisos, para que los usuarios puedan dejar anotaciones de una forma rápida y claramente visibles para el resto del equipo.

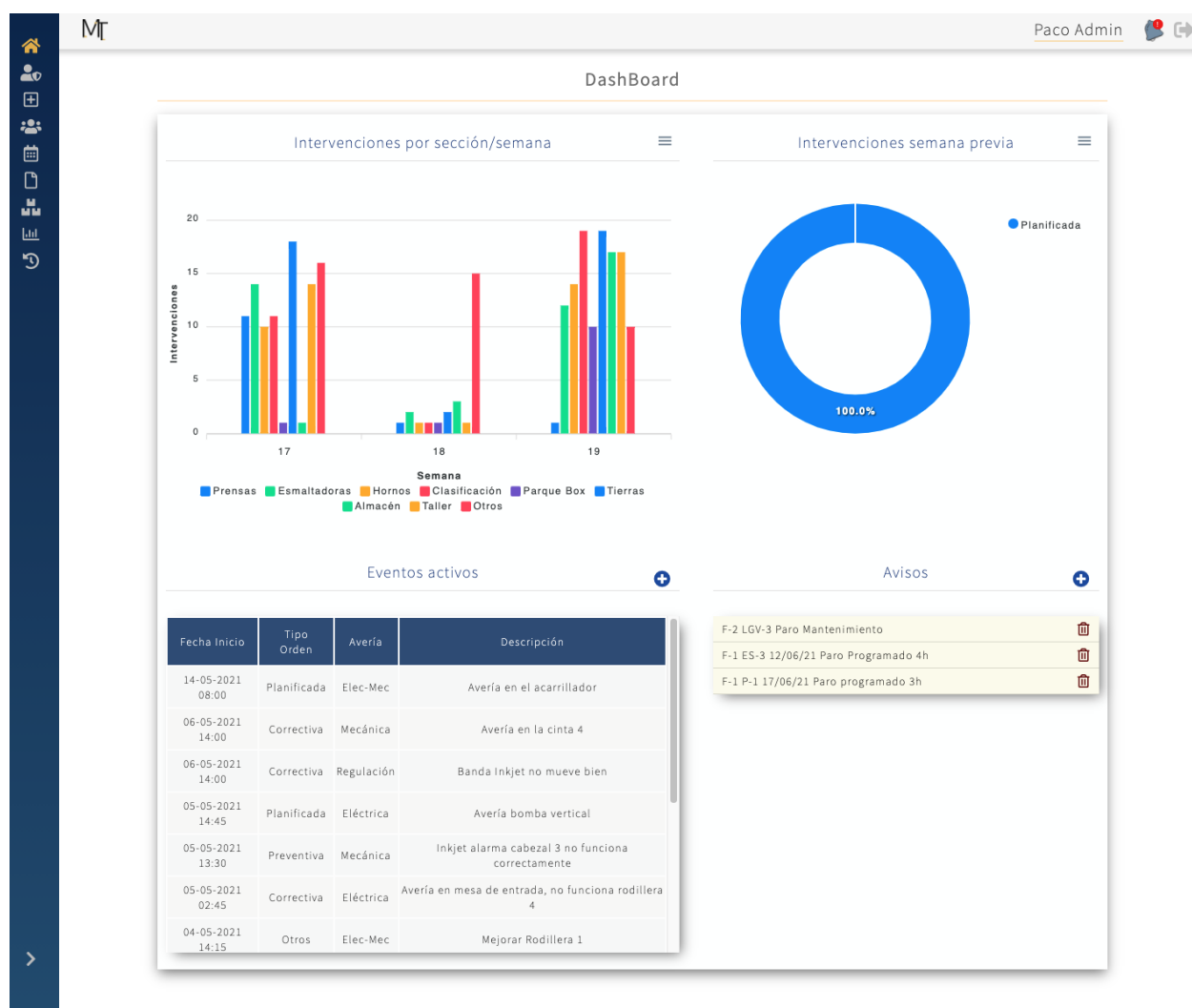


Imagen 4.4.4 Captura de Dashboard Screen.

Para llevar a cabo estas funcionalidades he tenido que hacer uso de algunos plugins, porque programarlo todo desde cero, no era viable, y más existiendo librerías altamente probadas y testeadas.

Los *charts* o gráficos pertenecen a una librería llamada *ApexCharts*, y ofrece una gran multitud de gráficas con decenas de opciones, incluso permite descargar un .PNG de la gráfica en cuestión. También tiene animaciones en cada gráfica y permite poder ocultar opciones, lo que le

aporta un dinamismo y apariencia excelente. Aquí me he encontrado con una gran dificultad y me ha llevado unas cuantas horas encontrarle una solución efectiva. El inconveniente fue que la librería necesita que se le proporcionen los datos de la forma en que la han programado, y la base de datos de la aplicación no está diseñada para guardar los datos de la forma en que la librería los necesita.

```
/* AGRUPAR TOTAL, TIPO ORDEN, SEMANA */  
SELECT count(types.name) as 'Total', types.name, WEEKOFYEAR(events.start) AS 'SEMANA'  
FROM events INNER JOIN types ON types.id = events.orderType  
WHERE events.start >= DATE_SUB(NOW(), INTERVAL 22 DAY) AND events.start <= NOW()  
GROUP BY types.name, WEEKOFYEAR(events.start)  
ORDER BY WEEKOFYEAR(events.start) ASC
```

Imagen 4.4.5 Ejemplo de una consulta a la BBDD.

Para mostrar las órdenes abiertas o eventos activos, he usado un paquete llamado *react-table*, porque me aportaba funcionalidades como poder ordenar haciendo click al <th> correspondiente sin tener que programar esa funcionalidad.



Imagen 4.4.6 Modal para agregar avisos.

Para el apartado de avisos, me he centrado en que se puedan crear o borrar, aunque en el futuro habilitaré el poder editar. Al pulsar el botón de crear aviso, aparecerá una modal desde la que poder introducir el aviso. Para esta modal he utilizado *react-modal*.

4.4.3. Admin Screen

Esta sección sólo está accesible a usuarios que posean el rol de admin y en ella se muestran los usuarios registrados en la aplicación, los cuáles podemos “eliminar” o editar. Eliminar no ha sido una opción interesante, ya que he preferido agregar una columna a los usuarios llamada *active*, y cuando eliminamos un usuario, realmente lo que hacemos es desactivarlo. Esto lo he aplicado como regla general en la aplicación, para que haya integridad referencial y no perder información al borrar datos.

Por otro lado también nos permite agregar nuevos usuarios y eliminar órdenes de trabajo, aunque realmente se desactivan, ya que con la eliminación podrían perderse datos innecesariamente.

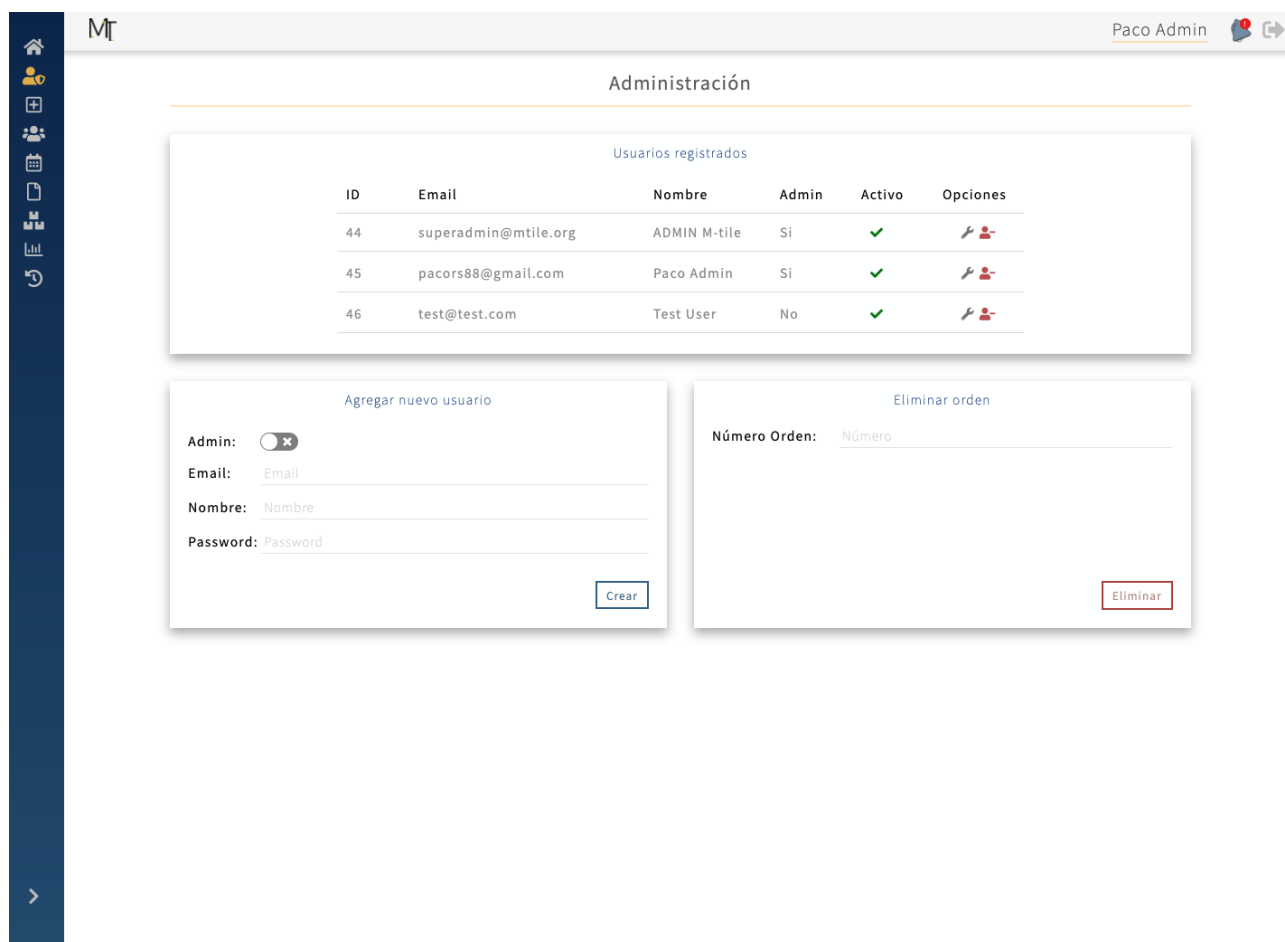


Imagen 4.4.7 Captura de Admin Screen.

Para el desactivado de usuarios y de eliminación de eliminación de órdenes de trabajo, hago uso de una ventana emergente de confirmación, para evitar borrados accidentales. Cabe destacar que el usuario una vez desactivado puede volver a activarse, acción que con las órdenes no se permite por el momento.

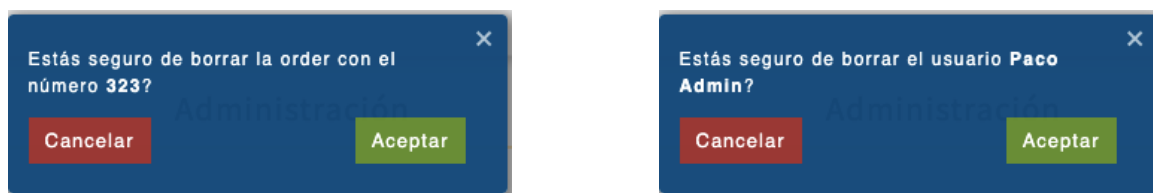


Imagen 4.4.8 Confirmación de borrados en órdenes y usuarios.

4.4.4. Order Screen

Esta sección es sin duda de las más importantes de la aplicación, porque en ella es dónde se van a introducir todos los datos de la aplicación, que son los que le van a dar sentido a casi todo el resto de secciones. En ella hay un formulario bastante completo con todos los datos requeridos para generar una orden nueva. En la parte de abajo se encuentra una componente de *Tabs* en el que podremos agregar las observaciones de la orden, las operaciones realizadas por los técnicos, sus fichajes, y los materiales si es que se han gastado.

The screenshot shows a web application interface for creating a new order. The page has a dark blue sidebar on the left with navigation icons. The top header bar is light gray with the 'MT' logo and the user name 'Paco Admin'. The main content area is white and titled 'Nueva Orden'. It contains three main sections: 'Datos Orden' with dropdown menus for 'Factoría', 'Sección', 'Número', 'Máquina', 'Tipo orden', 'Tipo avería', and 'Técnico', and a text input for 'Avisado por:'. Below this is the 'Fechas Orden' section with text inputs for 'Fecha aviso', 'Fecha fin', 'Inicio trabajo', and 'Fin trabajo'. The 'Información Adicional' section has tabs for 'Notas', 'Operaciones', 'Fichajes', and 'Materiales'. The 'Notas' tab is active, showing a text area for 'Observaciones:'. At the bottom right, there are two buttons: 'Volver' (red) and 'Crear Orden' (blue).

Imagen 4.4.9 Captura de Order Screen.

En este formulario y en especial todos los que tratan con fechas y horarios he tenido muchísimos problemas, porque cuando la fecha venía como un *String* en lugar de un tipo *Date*, que es lo habitual al recibir datos de una *BBDD*, recibía errores continuamente, y finalmente he tenido que controlar esa situación modificando cada propiedad que viene con una fecha, generando un tipo *Date* de ese fecha de tipo *String*.

También ha requerido de ciencia el asignar cada máquina a su número de sección, a la sección y a su factoría, para que en cada desplegable se muestren las opciones existentes reales. En una prueba real, se seleccionas la Factoría 1, la *UI* mostrará sólo las secciones existentes en esa Factoría y lo mismo ocurre con el número y con la máquina, así se evita al máximo que el usuario pueda cometer errores. Además también aparece un mensaje indicativo de lo que el usuario debe hacer para desbloquear los siguientes campos.

Datos Orden

*Selecciona una sección para desbloquear el siguiente campo

Factoría: 3 ▼ Sección: Elige Sección ▼

Imagen 4.4.10 Mensaje resaltado para guiar al usuario.

El candado que se puede observar en la parte superior derecha, es para finalizar una orden, pero que en el momento de la creación está desactivado, porque el objetivo es ir creando órdenes y más adelante el responsable dedicado a ello, las vaya cerrando una vez revisadas.

Respecto a las *Tabs* cada una muestra la información sobre operaciones, fichajes y materiales, y cada *Tab* tiene su propio modal para el agregado de esta información. El modal que más trabajo me ha llevado, ha sido el modal para agregar materiales, porque debe de filtrar lo que el usuario introduce en un input e ir mostrando en tiempo real en un *select* los ítems que corresponden con la búsqueda, filtrándolos por su descripción.

Modal titled "Agregar Operación" with a close button (X). It contains two input fields: "Operación:" and "Tiempo:". A green "Guardar" button is at the bottom right.

Modal titled "Agregar Material" with a close button (X). It contains a search input field labeled "Buscar:" with the placeholder text "Buscar Material". A green "Agregar" button is at the bottom right.

Modal titled "Nuevo Fichaje" with a close button (X). It contains three input fields: "Técnico:" with a dropdown menu showing "Elige Técnico", "Hora Inicio:" with the value "17/05/2021 13:22", and "Hora Fin:" with the value "17/05/2021 14:22". A green "Aceptar" button is at the bottom right.

Imagen 4.4.11 Modales para la adición de operaciones, fichajes y materiales.

4.4.5. Crew Screen

En esta sección se muestran los técnicos de mantenimiento, y en el caso de que el usuario sea del rol admin, podrá también ver detalles de los técnicos, actualizarlos, y agregar nuevos técnicos.

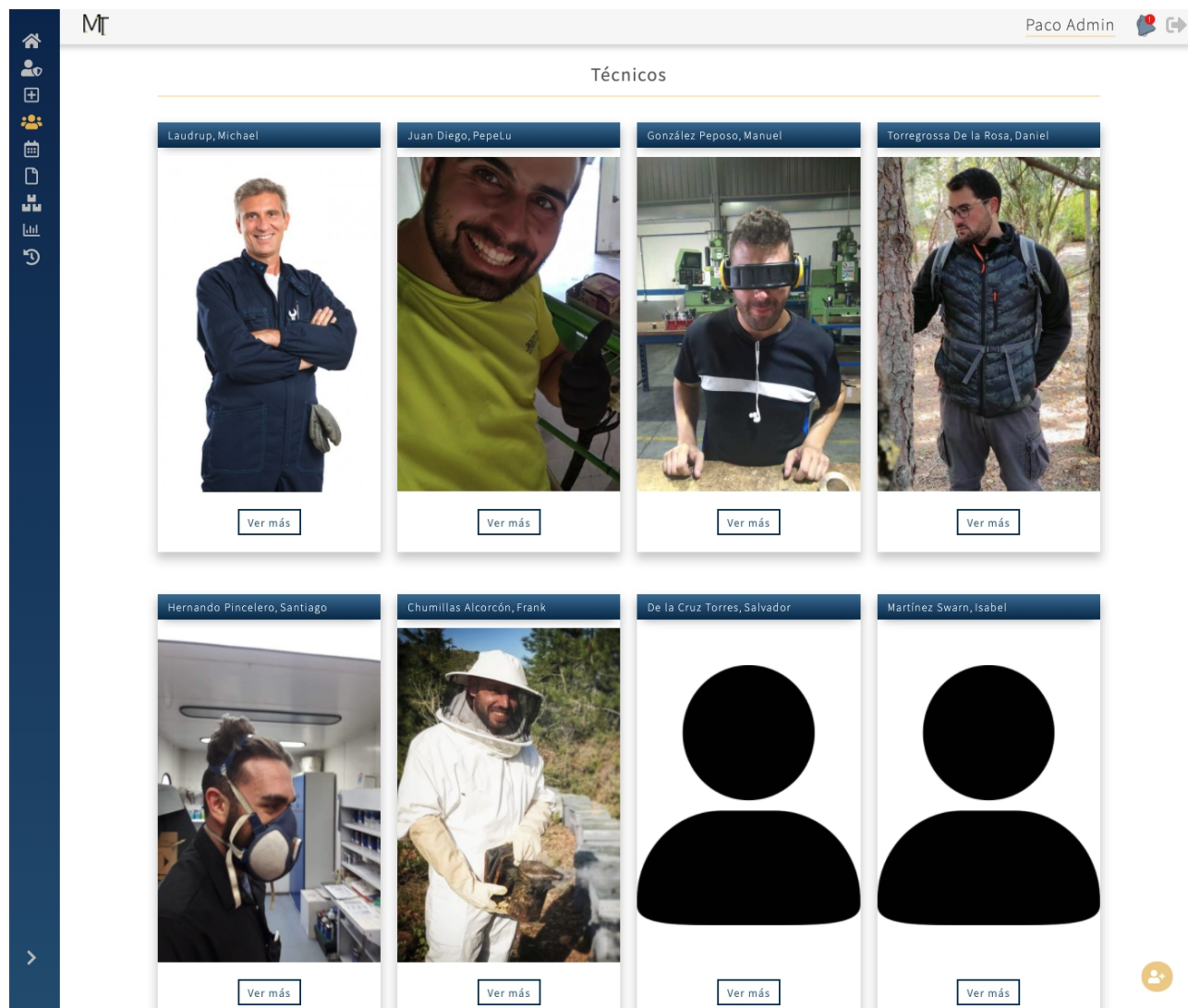


Imagen 4.4.12 Captura de Crew Screen.

Siempre existe la posibilidad de que al crear un nuevo técnico, no se introduzca una foto, con lo que se mostrará una por defecto.

En cada técnico se puede pulsar el botón de *ver más* y aparecerá una modal que muestra toda la información de un técnico, incluida la imagen, desde donde podremos editarlo y eliminarlo. Esta modal es exactamente la misma tanto para el agregado de nuevos técnicos, como para ver un técnico existente. Este aprovechamiento de componentes es gracias a React junto con Redux, porque cuando pulsas el botón de *ver más* de un técnico, coge los datos de ese técnico y los introduce en un estado, al cual he llamado, *activeTechnician*, y cuando la modal está abierta

comprueba si este estado existe, mostrando un *form* vacío en el caso de que *activeTechnician* esté vacío, y uno relleno con la información del técnico, en el caso de que *activeTechnician* exista.

Imagen 4.4.13 Modal de creación y editar técnico.

4.4.6. Calendar Screen

Esta es la sección desde la que se pueden ver todas las órdenes de trabajo, abiertas o cerradas, de una forma simple. Las órdenes están clasificadas por colores según el cliente ha solicitado, para que de una forma visual, sea sencillo ver como está funcionando la fábrica:

- ▶ Rojo: representa un mantenimiento correctivo.
- ▶ Azul: representa un mantenimiento planificado.
- ▶ Amarillo: representa un mantenimiento generado por Dirección.
- ▶ Verde: representa un mantenimiento preventivo.
- ▶ Turquesa: representa otros mantenimientos.

03	04	05	06	07	08	09
F1 - Parque Box - Box	F1 - Almacén - Rodilleros	F3 - Hornos - Mesa Entr	F2 - Clasificación - Rod	F1 - Tierras - Elevador		
F1 - Almacén - Muelle c.		F1 - Esmaltadoras - Ink	F1 - Tierras - Bandas			
		F3 - Taller - Bomba vert	F1 - Esmaltadoras - Ink			
		F1 - Almacén - Muestras				

Imagen 4.4.14 Ejemplo de colores al agregar nuevas órdenes .

Este calendario, es una librería instalada llamada *react-calendar*, la cual proporciona muchas de las funciones que necesitaba, aunque hay alguna que no he implementado, debido a que su comportamiento no era el deseado. Lo importante para hacer funcionar esta librería es proporcionarle la fecha de inicio y final de cada orden, y para el *badge* de color, he creado un componente con la información que quería que fuera visible, de forma que se mostrarán todos colocados en su fecha y horas correspondientes.

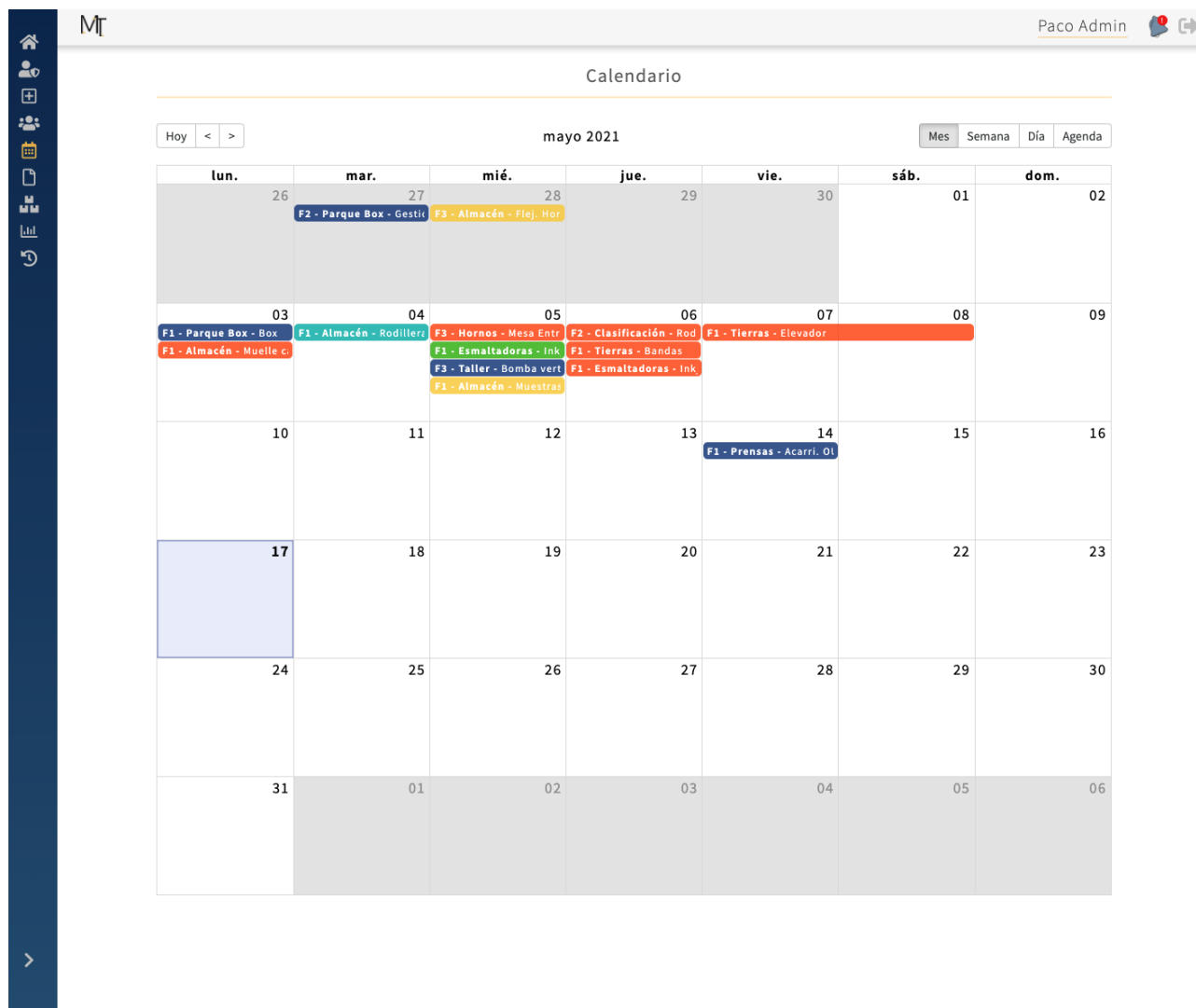


Imagen 4.4.15 Captura del calendario - mes.

Como funcionalidades importantes, cuando el usuario hace click en cada *badge*, se abrirá una modal con los datos de la orden pulsada, de sólo lectura, para ver la orden con más detalle. Además esta modal incorpora un botón para dirigirnos a la pantalla de la orden rellenada, para poder editarla entre otras funciones. Por otra parte el calendario también ofrece la posibilidad de ver por semana y día aparte de mes, lo cuál es sumamente útil para ver de una forma más en detalle como ha ido un día o semana en concreto.

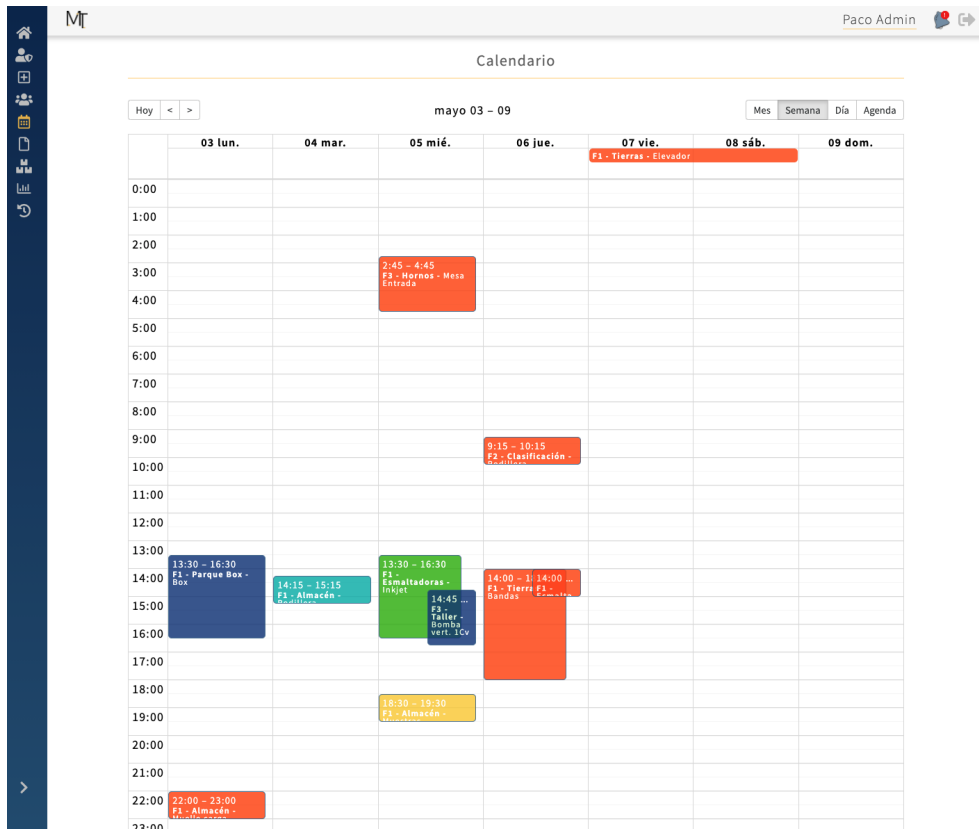


Imagen 4.4.16 Captura del calendario - semana.

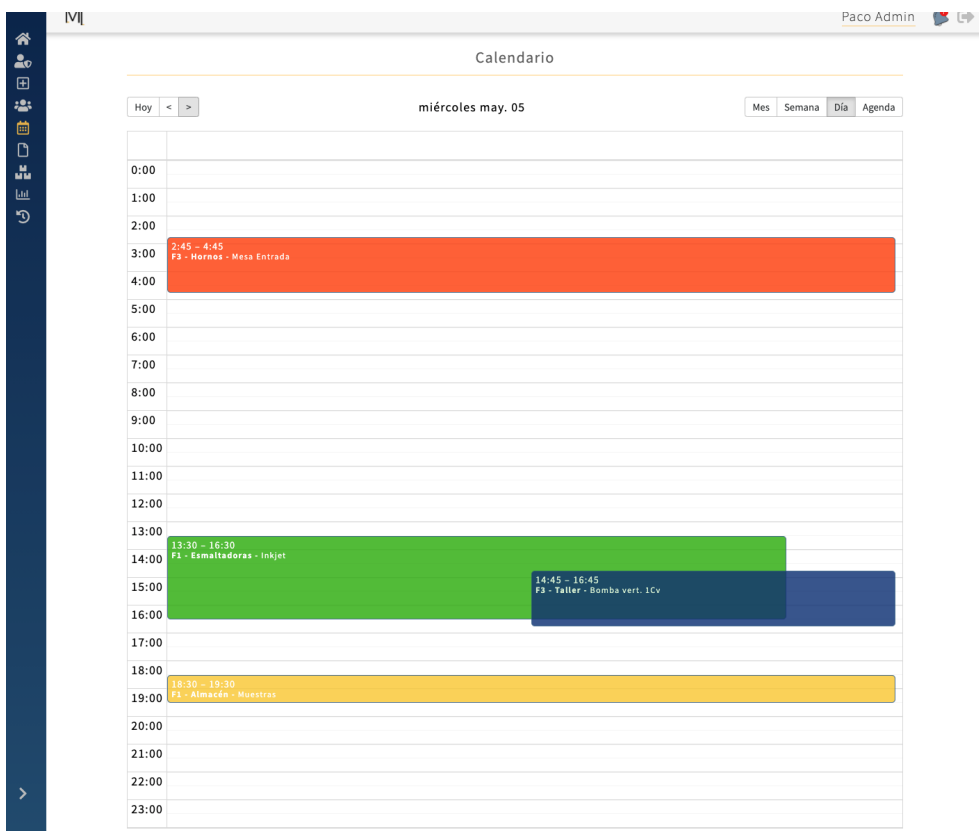


Imagen 4.4.17 Captura del calendario - día.

4.4.7. Docs Screen

Esta sección es una parte muy útil de la aplicación (pensando como electromecánico del sector), porque permite a los técnicos tener acceso a toda la documentación digitalizada de la fábrica, clasificado por secciones.

En la actualidad, dentro de los cuadros eléctricos de cada máquina, se puede encontrar su documentación respectiva para que cuando haya una avería, esté accesible en el mismo lugar. No obstante con el tiempo esta documentación se deteriora debido al tiempo y la suciedad, y en el peor de los casos faltan páginas e incluso llega a desaparecer. Y este apartado se ha creado para resolver todos estos contratiempos.

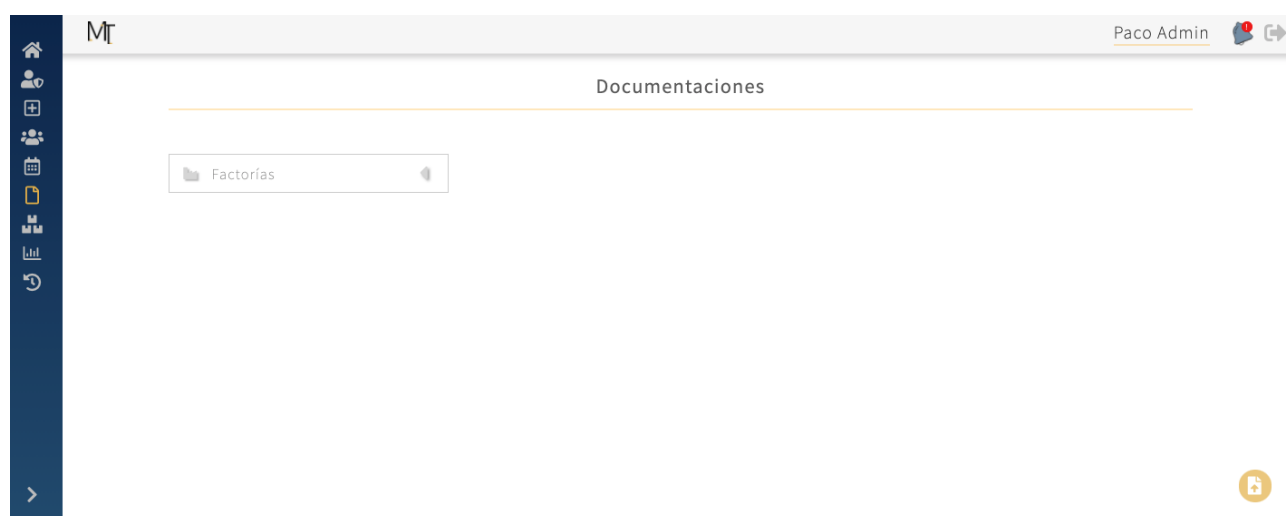


Imagen 4.4.18 Captura de Doc Screen.

Este componente que permite seleccionar una factoría y otro idéntico que aparece después de seleccionar una factoría, es un componente similar a un *select* de HTML, y tuve que programarlo a conciencia para poder agregar algunas animaciones y estilos adicionales, como son los iconos y sus movimientos, y las animaciones de la lista al desplegarse y recogerse.

Factoría 1 ▶ Otros



No se ha encontrado documentación

Imagen 4.4.19 Mostrando cuando no hay documentos en la selección.

Una vez seleccionada una factoría y/o una sección, en la parte superior irán apareciendo unas *breadcrumbs* para guiar al usuario que opciones ha escogido. Si con las opciones escogidas,

hay documentación agregada, irán apareciendo los documentos con una animación de *loading* previa, en caso contrario aparecerá un mensaje avisando al usuario que no hay documentos.

Factoría 1 ▶ Prensas

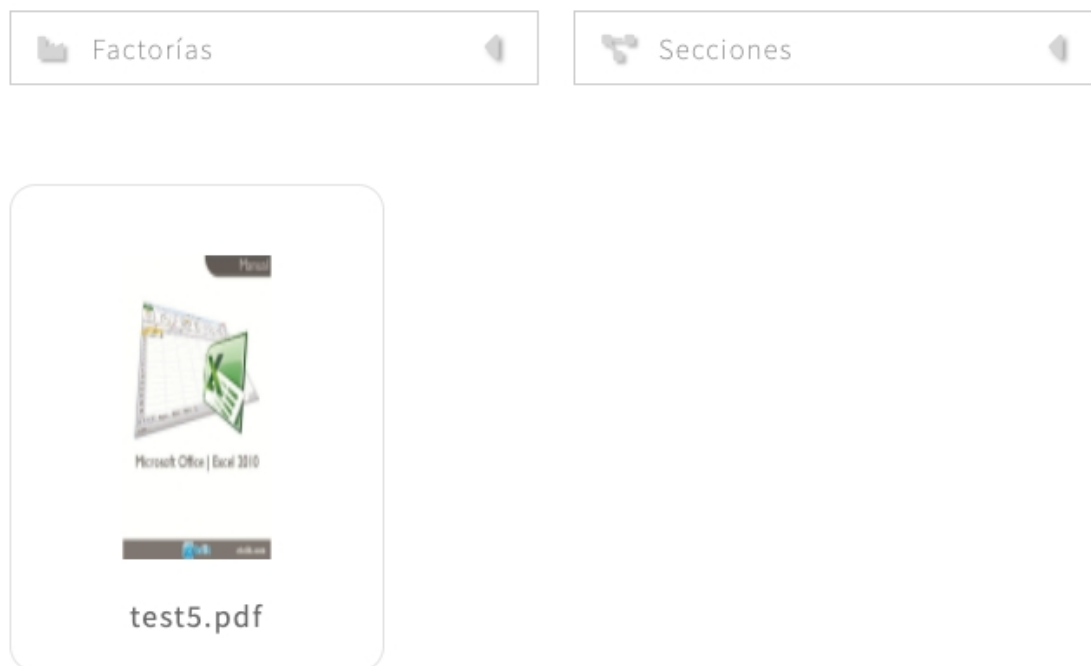


Imagen 4.4.20 Mostrando cuando hay documentos existentes en la selección.

Al hacer click en el documento, nos abrirá en una modal el archivo en cuestión, al estilo de *Google Drive*. Para lograr esto, he utilizado una librería llamada *react-pdf* para poder abrir documentos pdf en la misma aplicación, sin necesidad de abrir una nueva pestaña del navegador.



Imagen 4.4.21 Vista de un documento.

Una vez pasa el ratón por encima del documento, aparece un control para pasar páginas y mostrar al usuario la página actual. También se puede eliminar el documento en cuestión haciendo click en el icono situado en la parte superior derecha.

Por último, en la pantalla principal de ver documentos, hay un icono en la parte inferior derecha para el agregado de nueva documentación. Para ello se abrirá una modal para habilitar la subida de un fichero (sólo se permiten .pdf), y adicionalmente agregarle una descripción para su posterior identificación, porque el nombre del archivo actual se borra en el *Backend* para darle después un nombre único, con el fin de evitar problemas futuros al subir ficheros.

A modal window titled "Agregar Documentación" with a close button (X) in the top right corner. It contains four form fields: "Factoría:" with a dropdown menu showing "Seleccionar", "Sección:" with a dropdown menu showing "Seleccionar", "Descripción:" with a text input field containing "Descripción", and "Archivo:" with a text input field containing "certificadoAP_14031394.pdf". At the bottom, there are two buttons: "Cargar PDF" with a cloud upload icon and "Guardar" in a green box.

Imagen 4.4.22 Modal para la subida de ficheros.

4.4.8. Warehouse Screen

Esta sección es el lugar donde se puede consultar el almacén, actualizarlo, y borrado, aunque este último es ficticio, porque se desactivan para mantener la integridad referencial.

La forma de su buscador es simple, se muestra una lupa al usuario y se le incita a pulsarla mediante un mensaje. Además esta lupa cambia de color y tamaño para captar su atención. Al pulsarla aparece un input para introducir caracteres y se van mostrando las coincidencias de una forma dinámica. El componente filtra tanto por referencia del *item*, como por descripción, lo cuál es muy útil para cuando el usuario quiere agregar material existente y conoce su referencia exacta.

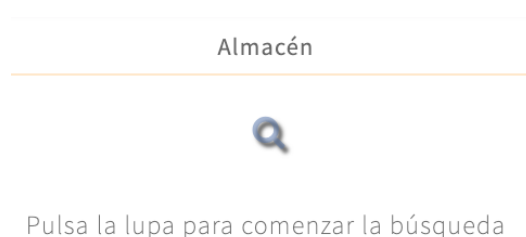


Imagen 4.4.23 Acceso al buscador del almacén.

Una vez existan coincidencias, se mostrara una tabla con ellas, y en el caso de que no las haya, se mostrará un mensaje en pantalla avisando al usuario de ello.

S

Código	Descripción	Cantidad	Stock Mínimo	Ubicación
45GRT	Sensor inductivo PNP 3mm	15	10	Estantería 10, Piso 3.
45657	Motor 0.75Kw Con patas. Eje 25	3	1	Almacén Factoría 1
23OLK	Correa Termosoldable Tipo B con núcleo	50	20	Estantería Taller
232LK	Correa Termosoldable Tipo B con núcleo	50	20	Estantería Taller
PLK23	Reductor Ejes 13/24 1/12	3	1	Estantería 4Z
SM763	Electrodos 3mm	50	10	Estantería 3A
JM123	Correa Termosoldable Cordón Sin Núcleo	15	20	Centro Logístico
JR763	Fotocélula Emiso Barrera IFM 5-500mm	10	5	Estantería 2J Taller
JR769	Fotocélula Emisora/Receptora Barrera IFM 5-500mm	5	2	Estantería 2J Taller
JR768	Fotocélula Emisora Barrera IFM 5-500mm	10	2	Estantería 2J Taller



Página 1 de 2

Imagen 4.4.24 Tabla al buscar un ítem en el almacén.

La tabla esta limitada a 10 líneas con el fin de evitar sobrecargar al usuario con exceso de información. Si hay un mayor de 10 coincidencias, se activarán unos botones de paginación, que permiten al usuario moverse entre páginas cómodamente. Además cada línea permite que al hacer click sobre ella, nos muestre una modal con los datos de ese ítem en concreto con el fin de poder editar el stock mínimo (para generar alertas), su descripción, su ubicación o su stock actual.

Por otro lado nos permite agregar ítems nuevos a la base de datos, gracias a un botón situado en la parte inferior derecha, de la misma forma que el agregado de nuevos técnicos y documentos. Al seguir siempre el mismo patrón permite al usuario saber con antelación lo que va a hacer ese botón sin necesidad de pulsarlo.

No obstante, otra funcionalidad interesante implementada en esta sección, es la de control del stock. Las órdenes de trabajo no restan en la *BBDD* los ítems hasta que no se cierra la orden por completo. Una vez se cierra la orden, busca los ítems que se han usado en la orden y los resta del stock para después retornar los nuevos valores de la *BBDD*, con el fin de que el Frontend tenga en sus estados el stock actual. Seguidamente el Frontend, busca de entre todos los ítems si

hay alguno por debajo del stock mínimo indicado. Finalmente, si encuentra alguno por debajo del mínimo, mostrará una alerta en el icono de la campana situado en la barra superior.

Editar Item

Código: SM763

Descripción: Electrodo 3mm

Cantidad: 50

Stock Mínimo: 10

Ubicación: Estantería 3A

Eliminar Actualizar

Agregar Item

Código:

Descripción:

Cantidad:

Stock Mínimo:

Ubicación:

Guardar

Imagen 4.4.25 Modales para editar y agregar items.

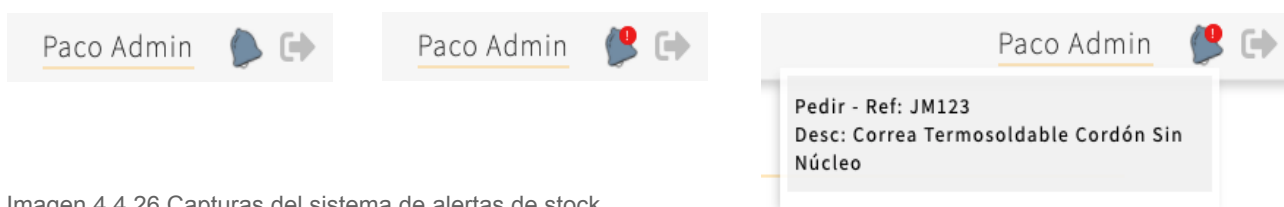


Imagen 4.4.26 Capturas del sistema de alertas de stock.

4.4.9. Statistics Screen

En esta sección se le muestran por el momento al usuario 6 gráficas. Estas son las que el cliente más utiliza y la información representada es del conjunto de todas las factorías.

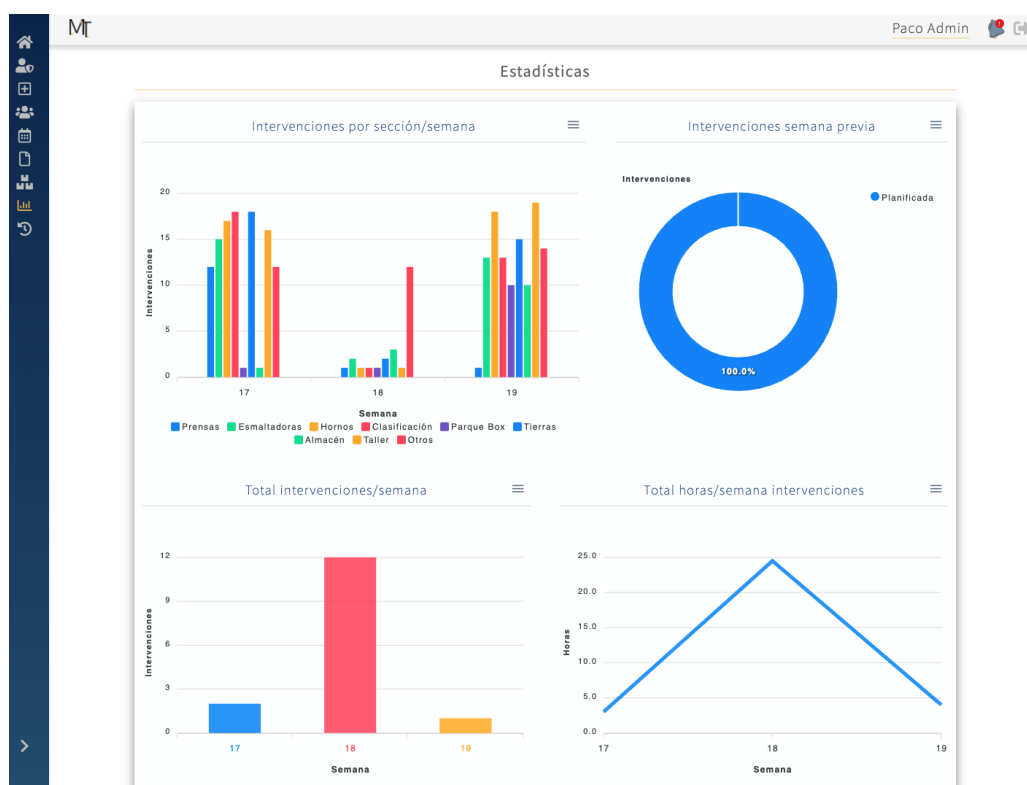


Imagen 4.4.27 Captura de Statistics Screen.

Aquí las dificultades han sido adaptar los datos obtenidos por la BBDD a la librería pero también he tenido problemas en que los gráficos se mostraran bien en todas las resoluciones. Para ello he tenido que usar unos parámetros de grid que desconocía:

```
grid-template-columns: minmax(0, 1fr) minmax(0, 0.75fr);
```

Imagen 4.4.28 Fragmento de código CSS.

De esta forma las columnas de una rejilla *GRID*, siempre ocuparan el máximo espacio posible y así solventar el problema. Esta solución es similar a realizar un **display: flex**, pero por alguna razón los gráficos no respondían adecuadamente con *flex*.

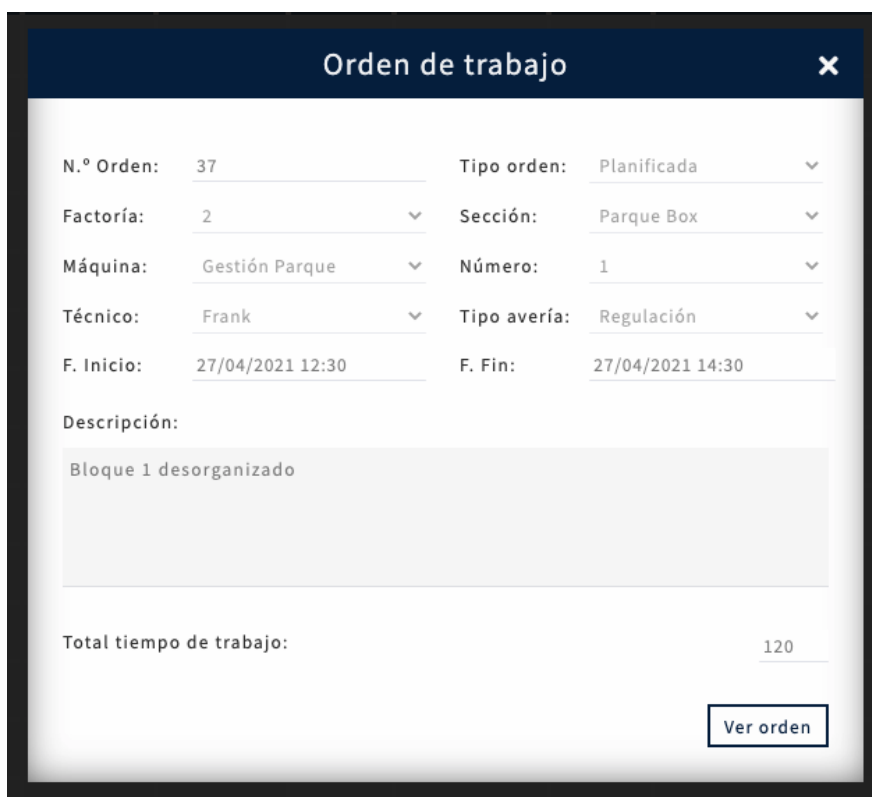
4.4.10. Historial Screen

Esta es la última sección de la aplicación en la que se muestra el histórico total de órdenes. Se muestran un máximo de 15 por pantalla, y después entra en funcionamiento unos botones para la paginación. Al hacer click en una orden se abrirá una modal idéntica a las que se abren tanto en Calendar Screen, como en la tabla de órdenes activas en Dashboard Screen. De esta forma permite al usuario ver un poco más de detalles de la orden sin tener que ir a la página completa de la orden.

Fecha Inicio	Tipo Orden	Avería	Factoría	Sección	Número	Máquina	Tiempo Trabajo	Estado
14-05-2021 08:00	Planificada	Elec-Mec	1	Prensas	3	Acarri. OUT PH	240 min.	🔒
07-05-2021 23:00	Correctiva	Mecánica	1	Tierras	1	Elevador	240 min.	🔒
06-05-2021 14:00	Correctiva	Mecánica	1	Tierras	1	Bandas	240 min.	🔒
06-05-2021 14:00	Correctiva	Regulación	1	Esmaltadoras	7	Inkjet	60 min.	🔒
06-05-2021 09:15	Correctiva	Eléctrica	2	Clasificación	22	Rodillera emergencia	60 min.	🔒
05-05-2021 18:30	Directiva	Mecánica	1	Almacén	1	Muestras	60 min.	🔒
05-05-2021 14:45	Planificada	Eléctrica	3	Taller	1	Bomba vert. 1Cv	120 min.	🔒
05-05-2021 13:30	Preventiva	Mecánica	1	Esmaltadoras	2	Inkjet	135 min.	🔒
05-05-2021 02:45	Correctiva	Eléctrica	3	Hornos	31	Mesa Entrada	240 min.	🔒
04-05-2021 14:15	Otros	Elec-Mec	1	Almacén	1	Rodillera	60 min.	🔒
03-05-2021 22:00	Correctiva	Mecánica	1	Almacén	1	Muelle carga	60 min.	🔒
03-05-2021 13:30	Planificada	Mecánica	1	Parque Box	1	Box	120 min.	🔒
28-04-2021 17:30	Directiva	Eléctrica	3	Almacén	1	Flej. Horizontal	60 min.	🔒
27-04-2021 12:30	Planificada	Regulación	2	Parque Box	1	Gestión Parque	120 min.	🔒
22-04-2021 14:15	Correctiva	Mecánica	2	Esmaltadoras	22	Zona Esmalt.	180 min.	🔒

Imagen 4.4.29 Historical Screen.

Por los momentos no existe un límite de total de órdenes, buscador, ni un filtrado por fecha exacta o máquina, no obstante está planeado hacerlo en siguientes versiones para optimizar búsquedas.



The image shows a modal window titled "Orden de trabajo" with a close button (X) in the top right corner. The modal contains a form with the following fields:

N.º Orden:	37	Tipo orden:	Planificada
Factoría:	2	Sección:	Parque Box
Máquina:	Gestión Parque	Número:	1
Técnico:	Frank	Tipo avería:	Regulación
F. Inicio:	27/04/2021 12:30	F. Fin:	27/04/2021 14:30

Descripción:

Bloque 1 desorganizado

Total tiempo de trabajo: 120

Ver orden

Imagen 4.4.30 Modal ver orden usada en varios componentes.

Para terminar con el Frontend, a continuación un listado de las dependencias usadas para el desarrollo de esta aplicación:

- ▶ React-redux: paquete que habilita el poder trabajar con un store global mediante acciones.
- ▶ Redux-thunk: paquete para poder lanzar acciones dentro de otras acciones asíncronas y tener acceso al estado global desde las acciones.
- ▶ React-router: paquete para poder trabajar con rutas en *React*.
- ▶ Node-sass: paquete para trabajar con SASS en lugar de CSS.
- ▶ Validator: validador polivalente para emails, números, etc.
- ▶ React-apexcharts paquete para generar gráficos.
- ▶ React-big-calendar: paquete para generar calendarios.
- ▶ React-click-away-listener: helper para detectar click's fuera de un contenedor.
- ▶ React-datepicker: input customizado para la selección de fechas.
- ▶ React-datetime: input customizado para la selección de fechas y hora.

- ▶ React-modal: paquete para crear modals.
- ▶ React-pdf: paquete que permite ejecutar archivos .pdf en tiempo real.
- ▶ React-switch: helper para agregar un checkbox tipo switch.
- ▶ React-table: componente para generar tablas con múltiples opciones.
- ▶ React-tabs: componente para crear *Tabs* en un container.

4.4.11. Implementación del Backend

El Backend de este proyecto, a pesar de ser una cuarta parte de tamaño respecto al Frontend, era una parte esencial para afrontar este proyecto y llevarlo a cabo. Además no había programado ni recibido formación en *NodeJS*, por lo que el primer paso fue comenzar un curso del mismo para poder programar una API de la mejor forma posible, que sea escalable y código reutilizable. Para ello completé un curso de *NodeJS* de 29 horas de vídeo, con muchos ejercicios y tareas para reforzar conocimientos.

Programar en *NodeJS* es relativamente sencillo teniendo experiencia con *JavaScript*, por lo que la curva de aprendizaje es menos pronunciada para quien ya conoce el lenguaje. Hay una cosa reseñable en *NodeJS* y es que necesitas algunas librerías adicionales para simplificar algunas tareas, como servir una *API*.

Para crear este Backend, he intentado seguir una estructura básica de directorios, que permita tener agrupados los archivos con funciones similares y el resultado ha sido bastante bueno, porque permite localizar rápidamente cada archivo.

La estructura de directorios del Backend, ha quedado ordenada de la siguiente forma:

- ▶ Controllers: se encargan de interactuar con la BBDD y de crear una respuesta al Frontend.
- ▶ Db: La conexión con MySQL.
- ▶ Helpers: Funciones de ayuda para diferentes propósitos .
- ▶ Middlewares: Funciones creadas para interactuar entre la petición y el controlador.
- ▶ Models: Clases para trabajar con cada tabla de la BBDD.
- ▶ Routes: Archivos donde se generan las rutas que usualmente terminan en un controlador o función que genera una respuesta al cliente.

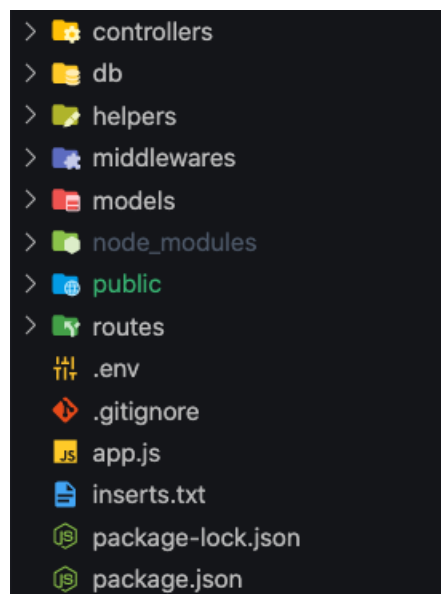


Imagen 4.4.31 Directorios del Backend.

- Public: Directorio en el que se introduce el proyecto de Frontend para servirlo como una sola aplicación.

Para crear la *API* de una forma sencilla es indispensable usar *Express*, porque te permite, con muy pocas líneas de código, crear múltiples endpoints de una forma fácil. Como nota, un *endpoint* es una URL exacta a la que accede un usuario, por ejemplo: www.google.es/myendpoint.

Esta aplicación ha requerido numerosos endpoints para llevarla a cabo, un total de 45 y son los siguientes:

Listado de endpoints API M-tile				
Petición	Controlador	URL	Headers	Body
GET	User - getUsers	{URL}/api/users/	x-token: tokenJWT	
GET	User - getUserById	{URL}/api/users/:id	x-token: tokenJWT	
POST	User - addUser	{URL}/api/users/new	x-token: tokenJWT	{ "name": "test", "email": "myemail@myemail.com", "password": "123456" }
GET	User - renewToken	{URL}/api/users/renew	x-token: tokenJWT	
POST	User - loginUser	{URL}/api/users/	x-token: tokenJWT	{ "name": "test", "password": "123456" }
PUT	User - updateUser	{URL}/api/users/:id	x-token: tokenJWT	{ "name": "test", "email": "myemail@myemail.com", "password": "123456" }
DEL	User - deleteUser	{URL}/api/users/:id	x-token: tokenJWT	
GET	Warning - getWarnings	{URL}/api/warnings/	x-token: tokenJWT	
POST	Warning - addWarning	{URL}/api/warnings/	x-token: tokenJWT	{ "description": "warning to add" }
DEL	Warning - deleteWarning	{URL}/api/warnings/:id	x-token: tokenJWT	
GET	Warehouse - getItems	{URL}/api/warehouse/	x-token: tokenJWT	
POST	Warehouse - addItem	{URL}/api/warehouse/	x-token: tokenJWT	{ "code": "75HRG", "description": "Correa B87", "quantity": "5", "minStock": "3", "place": "Almacén Factoría 1", }
PUT	Warehouse - updateItem	{URL}/api/warehouse/:id	x-token: tokenJWT	{ "code": "75HRG", "description": "Correa B87", "quantity": "5", "minStock": "3", "place": "Almacén Factoría 1", }
PUT	Warehouse - subtractItem	{URL}/api/warehouse/subtract/:id	x-token: tokenJWT	{ "code": "75HRG", "description": "Correa B87", "quantity": "5", "minStock": "3", "place": "Almacén Factoría 1", }

DEL	Warehouse - deleteItem	{URL}/api/warehouse/:id	x-token: tokenJWT	
POST	Doc - uploadDoc	{URL}/uploads/doc	x-token: tokenJWT	{ "file": "myfile.pdf", "section": "2", "info": "info about the file", }
DEL	Doc - deleteDoc	{URL}/uploads/doc/:id	x-token: tokenJWT	
GET	Technician - getTechnicians	{URL}/api/crew/	x-token: tokenJWT	
POST	Technician - addTechnician	{URL}/api/crew/	x-token: tokenJWT	{ "name": "Pepe", "surname": "María", "birthDate": "11-12-1988", "identityDcoument": "55555555K", "phoneNumber": "678543209", "email": "pepe.maria@gmail.com", "city": "Vila-real", "address": "C\ Andalucía 5", "image": "my-image.jpg", "notes": "notes about him/her", "schedule": "L-V M-T-N", "factory": "1", }
PUT	Technician - updateTechnician	{URL}/api/crew/:id	x-token: tokenJWT	{ "name": "Pepe", "surname": "María", "birthDate": "11-12-1988", "identityDcoument": "55555555K", "phoneNumber": "678543209", "email": "pepe.maria@gmail.com", "city": "Vila-real", "address": "C\ Andalucía 5", "image": "my-image.jpg", "notes": "notes about him/her", "schedule": "L-V M-T-N", "factory": "1", }
DEL	Technician - deleteTechnician	{URL}/api/crew/:id	x-token: tokenJWT	
POST	Technician - addImage	{URL}/api/uploads/technician	x-token: tokenJWT	{ "file": "my-image.jpg", }
GET	Events - getEvents	{URL}/api/events/events	x-token: tokenJWT	
POST	Events - addEvent	{URL}/api/events/events	x-token: tokenJWT	{ "factory": "1", "section": "1", "machine": "4", "number": "1", "technician": "2", "worker": "Juanito", "orderType": "2", "breakdown": "3", "start": "06-07-2021 20:00", "end": "06-07-2021 22:00", "startFix": "06-07-2021 20:00", "endFix": "06-07-2021 22:00", "totalMins": "2", "description": "avería en el horno", }
PUT	Events - updateEvent	{URL}/api/events/events/3	x-token: tokenJWT	{ "factory": "1", "section": "1", "machine": "4", "number": "1", "technician": "2", "worker": "Juanito", "orderType": "2", "breakdown": "3", "start": "06-07-2021 20:00", }

				"end": "06-07-2021 22:00", "startFix": "06-07-2021 20:00", "endFix": "06-07-2021 22:00", "totalMins": "2", "description": "avería en el horno", }
DEL	Events - removeEvent	{URL}/api/events/events/3	x-token: tokenJWT	
POST	Events- addEventOperation	{URL}/api/events/operation/3	x-token: tokenJWT	{ "time": "1.75", "operation": "Cambiar correas", }
DEL	Events- deleteEventOperation	{URL}/api/events/operation/3	x-token: tokenJWT	
POST	Events - addEventClock	{URL}/api/events/clock/3	x-token: tokenJWT	{ "userId": "1", "start": "06-07-2021 20:00", "end": "06-07-2021 22:00", "user": "pepe", }
DEL	Events - deleteEventClock	{URL}/api/events/clock/3	x-token: tokenJWT	
POST	Events - addEventItem	{URL}/api/events/item/3	x-token: tokenJWT	{ "code": "45GRT", "quantity": "2", "description": "Correa tipo C 124", }
DEL	Events - deleteEventItem	{URL}/api/events/item/3	x-token: tokenJWT	
GET	Factory - getBreakdowns	{URL}/api/events/breakdowns	x-token: tokenJWT	
GET	Factory - getTypes	{URL}/api/events/types	x-token: tokenJWT	
GET	Factory - getFactories	{URL}/api/events/factories	x-token: tokenJWT	
GET	Factory - getSections	{URL}/api/events/sections	x-token: tokenJWT	
GET	Factory - getNumbers	{URL}/api/events/numbers	x-token: tokenJWT	
GET	Factory - getMachines	{URL}/api/events/machines	x-token: tokenJWT	
GET	Factory - getDocs	{URL}/api/events/docs	x-token: tokenJWT	
GET	Statistics - getLastWeekByOrderType	{URL}/api/statistics/lastweekbyordertype	x-token: tokenJWT	
GET	Statistics- getWeeks	{URL}/api/statistics/lastweekbyordertype	x-token: tokenJWT	
POST	Statistics - getStatisticsBySectionWeek	{URL}/api/statistics/section/:nweek	x-token: tokenJWT	{ "section": "Almacén", }
GET	Statistics- getInterventionsWeeks	{URL}/api/statistics/interventionsweek	x-token: tokenJWT	
GET	Statistics- getTotalTimeByWeek	{URL}/api/statistics/totaltimebyweek	x-token: tokenJWT	
GET	Statistics- getLastWeekByBreakdown	{URL}/api/statistics/lastweekbybreakdown	x-token: tokenJWT	

Tabla 4.4.1. Listado de endpoints de la API de M-tile.

Hay algunos endpoints que no han sido programados, como el de agregar una factoría, sección, número o máquina, o poder eliminarnos, pero algunos movimientos los he considerado arriesgados, por lo que dejo esas funciones a la persona que controle la BBDD y sepa lo que está haciendo, en este caso el desarrollador.

Como he mencionado anteriormente, esta API es relativamente sencilla en comparación con el Frontend, y para llevarla a cabo no han hecho falta muchas dependencias, no obstante, todas han sido absolutamente necesarias.

- ▶ Express: para crear API's de una forma más simple.
- ▶ Cors: middleware para realizar respuestas desde la API.
- ▶ Bcrypt: helper para el encriptado de passwords.
- ▶ Sequelize: paquete para conectar NodeJS con una BBDD MySQL.
- ▶ Express-validator: validador polivalente para emails, números, etc.
- ▶ Jsonwebtoken: paquete para tratar con JWT de autenticación de usuarios.
- ▶ Express-fileupload: paquete para habilitar la subida de ficheros a través de NodeJS.
- ▶ Uuid: helper para asignar id's únicos.
- ▶ Dotenv: paquete para habilitar el uso de variables de entorno de una forma más simple.

4.5. Experimentación

En la actualidad realizar pruebas o *tests* a una aplicación es de gran importancia debido a que aporta una seguridad de que el *software* va a fallar lo mínimo posible. De hecho en la gran mayoría de vacantes de desarrolladores, solicitan que se esté familiarizado con librerías de *testing*, de ahí su importancia.

En esta aplicación, debido a su gran envergadura para una sola persona, no se han completado las pruebas de test unitarias. Solamente programar los tests para esta aplicación (*Frontend* y *Backend*), llevaría más tiempo que programar la aplicación entera. Por esta razón no se han llevado a cabo, sin embargo, sí se ha completado un *debugging* bastante intenso para depurar muchos de los errores.

Una de las formas más comunes usadas en esta aplicación para corregir errores ha sido haciendo uso de *console.log()* (con el propósito de comprobar entradas y salidas), y comprobando que en el *DOM* el efecto era el esperado. Muchas de estas pruebas también han sido a nivel de navegador, tanto con *Chrome*, *Firefox* y *Safari*, debido a que en ocasiones cada navegador interpreta unas instrucciones de una manera u otra. Además otra forma muy útil es que otra persona que no ha programado, testee el software, en mi caso he solicitado ayuda a un familiar por si se me “escapaba algo”.

En cuanto a experimentación, el siguiente paso va a ser realizar todas estas pruebas unitarias tanto de caja blanca como de caja negra, con el fin de entregar al cliente un software altamente fiable y 100% testado.

4.6. Requisitos de funcionamiento

Para ejecutar esta aplicación no se requiere ningún tipo de hardware exigente, ni especial. Ha sido probado en diferentes ordenadores y en todos ellos se ha comportado con normalidad. Quizá alguna vez se puede apreciar una pequeñísima ralentización cuando se cargan las tablas de estadísticas, debido a que consumen más recursos. No obstante el resto de la aplicación no es para nada un proceso pesado.

En cuanto a dispositivos móviles, se ha trabajado para que sea compatible en la mayor cantidad posible de resoluciones, incluso en los móviles con resoluciones más bajas, aunque el performance no es el mismo.

Por el lado del software, ha sido testeado con múltiples navegadores, *Chrome*, *Safari*, *Firefox*, *Opera* y algunos navegadores nativos de dispositivos móviles, y el feeling ha sido muy positivo en respuesta, fiabilidad y diseño.

4.7. Conclusiones

Este proyecto ha supuesto un reto a nivel personal y profesional bastante grande. Cuando decidí hacerlo, no me esperaba la gran cantidad de código que requería, de lógica, paquetes, componentes. Esto se ha traducido en una gran cantidad de hora empleadas, que no he contado con minuciosidad, pero debe rondar las 400 con facilidad.

Ha sido un todo un reto tratar con tantas fechas en los formularios (y un quebradero de cabeza). A pesar de que *React* gestiona los formularios de una forma más sencilla que *Angular* (en mi opinión), comprobar tantos campos, números, fechas, teléfonos, horas, cantidades, ha sido también un reto muy importante.

Sin embargo todos estos retos me han servido para aprender, de aciertos y de errores. Uno de estos errores fue el pensar que esta aplicación podía realizarse con *MongoDB*, y quizá lo hubiera conseguido, pero habría tenido que escribir mucho código difícil de mantener para tratar algunas relaciones complejas, y sobre todo, las consultas a la *BBDD* habrían sido un trabajo super laborioso. Por suerte modifiqué el proceso para trabajar con *MySQL* y pude completar las funciones buscadas.

5. Estudio de mercado

5.1. Idea de negocio

Inicialmente no pensaba que M-tile podría servir como un negocio, pero conforme he ido desarrollándolo, he visto las posibilidades que puede ofrecer para aquellas empresas que todavía no están muy informatizadas, sobre todo empresas pequeñas.

La idea de negocio en este caso es clara: vender la aplicación como un *SaaS (Software as a Service)*, mientras compagino trabajo de desarrollador en una empresa privada. Los beneficios de distribuir el software en cloud son los siguientes:

- ▶ El cliente no posee el código de la aplicación.
- ▶ El cliente evita tener que hacer instalaciones y configuraciones.
- ▶ El cliente no tiene que comprar hardware adicional para servir o utilizar la aplicación.
- ▶ No tiene que ir personal especializado a realizar su instalación.

Esto influye en el tiempo de instalación, porque se reduce al mínimo, mostrándose como único contratiempo la adaptación del cliente a la aplicación.

La aplicación va claramente dirigida a empresas del sector cerámico, no obstante se podría adaptar a otro sector con ligeras modificaciones y el objetivo son empresas de pequeño y mediano tamaño que no pueden permitirse mantener un software demasiado costoso.

El coste de la aplicación sería de 50€ mensuales todo incluido y se ofrecerían adicionalmente paquetes de horas de desarrollo y de asistencia técnica.

Castellón y sus alrededores es el lugar perfecto para distribuir este tipo de software debido a la gran cantidad de empresas dedicadas al sector cerámico, por lo que facilitaría su distribución y crecimiento. Además cubriría gran parte de las necesidades de los clientes:

- ▶ Control de cada avería detallada.
- ▶ Control del stock de recambios.
- ▶ Estadísticas actualizadas sobre averías, tipo de averías, lugar de la avería...
- ▶ Acceso a documentación de la maquinaria en la nube.
- ▶ Control de los técnicos de mantenimiento y sus movimientos.
- ▶ Histórico de averías en el tiempo.
- ▶ Acceso a la aplicación desde cualquier lugar y dispositivo.
- ▶ Servicio de asistencia técnica.

5.2. Análisis DAFO

Debilidades
<ul style="list-style-type: none">▶ Realizar ambas actividades simultáneamente.▶ Poca experiencia en el sector IT.
Amenazas
<ul style="list-style-type: none">▶ Competencia muy alta.▶ Poco presupuesto.▶ Software desconocido para los clientes.▶ Poco perfil comercial.
Fortalezas
<ul style="list-style-type: none">▶ No necesito contratar a nadie.▶ Software adaptable.▶ Precio del producto bajo.▶ Localización para comercializar el producto muy buena.▶ Conocimiento del sector.
Oportunidades
<ul style="list-style-type: none">▶ Transición tecnológica de la industria.▶ Distribución del software de forma remota.

Tabla 5.2.1. Análisis DAFO.

5.3. Fuentes de financiación

Respecto a los ingresos/gastos, como la aplicación la mantengo y desarrollo desde casa, y tengo que pagar luz, agua, gas e internet de igual forma, no tengo que hacer ningún gasto extra para llevara a cabo estas tareas, quizá podría suponer 30€ mensuales de gasto adicional. A esos 30€ habría que sumar la cuota de autónomos, que durante los primeros 12 meses serían de 60€, por lo que me encontraría en unos 90 €/mes de gastos fijos sin haber distribuido todavía la aplicación.

La idea inicial es distribuir la aplicación por un precio de 50 €/mes. A esa cuota habría que restar el gasto de un dominio, y del servidor, que sería de 1€ y 10€ mensuales respectivamente, por cada aplicación distribuida.

Por otro lado, ofrecería gratuitamente un paquete de 5 horas de desarrollo y 3 horas de asistencia técnica. Todas las horas que excedan de esos paquetes se cobrarán a 20 €/hora. Ahí

entrarían en juego los paquetes de horas extras, si el cliente quiere, se le ofrece la posibilidad de contratar paquetes adicionales de horas, con lo que le supondría un ahorro. Los paquetes de 25 horas se ofrecerían por 300€, y los de 50 horas, por 500€, aplicable a ambos tipos de horas, de desarrollo, y de asistencia técnica.

En resumidas cuentas, inicialmente tendría unos 90 €/mes de gastos fijos, y mínimo tendría que tener 3 clientes usándola mensualmente para cubrir gastos y en el caso de vender algún paquete extra, ya comenzaría a rentar un poco el proceso.

En principio, como la aplicación ya está desarrollada, no voy a necesitar de una fuente de financiación. Sí necesitaría algo de mantenimiento y con el tiempo ir desarrollando mejoras y novedades, lo que de momento no implica la contratación de personal extra, ya que yo mismo puedo hacerlo.

En el caso de que la aplicación creciera lo suficiente en éxito y en demanda, podría plantearme la búsqueda de un socio, pero por el momento la idea esta descartada.

5.4. Marketing

Producto
El producto que se va a vender no es algo único en el mercado, pero puede porque está muy enfocado en el área del mantenimiento y aporta las características necesarias para gestionarlo. Esto puede resultar interesante a empresas que busquen mejorar en este área, porque cuando se lleva el control del mantenimiento, mejora considerablemente la productividad de las máquinas y los imprevistos.
Precio
El precio fijado es competitivo para lo que realmente ofrece la aplicación. En este área no será fácil hacerse un hueco en el mercado, pero con el tiempo cabe la posibilidad de que gracias al bajo precio y las características que ofrece, se de a conocer y gane en popularidad.
Distribución
La forma de distribuirlo es sencilla, como un software en <i>cloud</i> al que el cliente podrá acceder con unas credenciales que se le facilitarán. Esto reduce costes, tiempo y problemas, por lo que puede ser un punto a favor a la hora de ofrecer el producto. Si el cliente desea tener el software en su propia red, podría estudiarse la posibilidad, pero sería con una licencia distinta y con un precio superior que habría que valorar.

Comunicación

Por el momento las formas de darse a conocer van a ser:

- ▶ Ofrecer el producto directamente en las instalaciones del cliente.
- ▶ A través de redes sociales, especialmente Linked In.
- ▶ Por el boca a boca.

Tabla 5.4.1. Marketing mix o 4ps.

6. Trabajos Futuros

- Una de la primeras tareas a realizar, es completar todos los test unitarios de la aplicación, para detectar posibles errores y/o fallos.
- Sustituir *timepickers* del formulario de crear y editar orden, porque en ocasiones hacen un efecto extraño de scroll infinito, es muy lento, pero puede resultar molesto al usuario. Habría que valorar la posibilidad de utilizar otro componente que cumpla las mismas funciones.
- En el caso de que este software creciera de forma exponencial, habría que cambiar la forma de servir la API, para que cambie de *BBDD* dinámicamente y poder servir a varios clientes en un mismo servidor y URL. Esto ayudaría también a la implementación de un generador de avisos, para que avise a los clientes en caso de actualización, interrupción o de incidencias.
- Habilitar un sistema de tickets por prioridades, para que el cliente no tenga que enviar un email o llamar por teléfono para solventar posibles incidencias.
- Crear un calendario adicional para gestionar las vacaciones de los técnicos y que el mismo calendario sirva para mostrar los horarios de cada técnico, de forma que se puede saber qué técnico debe trabajar en una fecha concreta.
- Mejorar la sección de estadísticas, ofrecer métodos de filtrado dinámicamente y más tipos de gráficas.
- Crear un sistema para el agregado de nuevas factorías, secciones, números y máquinas, de forma que sea seguro, porque no pueden quedar secciones sin números, números de sección sin máquinas, o lo que es peor, factorías vacías.
- Mejorar la forma en la que se visualizan los documentos .pdf y habilitar un salto a la página deseada.
- Hacer alguna mejora estética, entre ellas habilitar un modo de colores distinto (para personas con daltonismo, o un modo oscuro por ejemplo), permitir que la empresa suba su logo y posicionarlo en algún lugar visible, etc.
- Al sustituir una imagen de un técnico la imagen antigua en local no se borra y cuando se borra un documento .pdf, no se elimina de forma local.

7. Bibliografía

- Jose Mejía. (24 de Junio de 2016). *Por qué tener un sistema centralizado de datos*.
<https://www.yunbitsoftware.com/blog/2016/06/24/sistema-centralizado-datos/>
- Just exw. *¿Qué es Excel y para qué sirve?*
<https://es.justexw.com/que-es-excel-y-para-que-sirve.html>
- Carlos Azaustre. (25 de Mayo de 2018). *¿Qué es Flux?*
<https://carlosazaustre.es/como-funciona-flux>
- Mariano Vázquez. (28 de Febrero de 2019). *Como elegir entre Angular y React para tu próxima aplicación*. <https://medium.com/@nanovazquez/c%C3%B3mo-elegir-entre-angular-y-react-para-tu-pr%C3%B3xima-aplicaci%C3%B3n-217f99f624b8>
- Lucidchart. (s.f.). *Qué es un esquema de base de datos*.
<https://www.lucidchart.com/pages/es/que-es-un-esquema-de-base-de-datos>
- Lucidchart. (s.f.). *Qué es un diagrama entidad-relación*.
<https://www.lucidchart.com/pages/es/que-es-un-diagrama-entidad-relacion>
- Ionos. (24 de Julio de 2020). *El diagrama de casos de uso en UML*.
<https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/diagrama-de-casos-de-uso/>
- James Kolce, Nilson Jacques. (23 de Marzo de 2020). *How to build and Structure a Node.js MVC Application*. <https://www.sitepoint.com/node-js-mvc-application/>
- Kuworking. (5 de Septiembre de 2020). *¿Cómo funciona el minmax() en el css GRID?*
<https://desarrolloweb.com/articulos/que-es-redux.html>

8. Anexos

8.1. Manual de Instalación

8.1.1. Instalación en servidor

La instalación se va a realizar desde un servidor con *Ubuntu 20.04*. De la siguiente forma es como la he desplegado en un *droplet* en *DigitalOcean* para poder trabajar con ella en *cloud*.

- Actualizamos el gestor de paquetes *apt*.

```
$ sudo apt update
```

- Instalamos *NodeJS*.

```
$ sudo apt install nodejs
```

- Instalamos el gestor de paquetes de *Node npm*.

```
$ sudo apt install npm
```

- Comprobamos la versiones instaladas, si recibimos un número por respuesta es que todo ha ido bien.

```
$ node -v
$ 16.1.0
$ npm -v
$ 7.11.2
```

- Instalamos el gestor de procesos *pm2* de *NodeJS*, para tener funcionando la app como un proceso en segundo plano.

```
$ sudo npm install pm2 -g
```

- Instalamos *mariaDB* y la secuencia de comandos de seguridad. Se nos solicitará establecer una contraseña para “root” y pulsamos a todo “Y” para establecer los valores por defecto.

```
$ sudo apt install mariadb-server
$ sudo mysql_secure_installation
```

- Ejecutamos *mariadb* y creamos la *BBDD*.

```
$ mariadb
MariaDB [(none)]> CREATE DATABASE mtil;
```

- Salimos de *mariadb*.

```
MariaDB [(none)]> exit
Bye
```

- Ahora insertaremos en la *BBDD* del servidor el archivo de la *BBDD* que tenemos de mtil. Para ello usaremos un gestor *FTP* e introduciremos el archivo en un directorio cualquiera y nos posicionaremos en él desde la terminal para cargar el backup de la *BBDD*.

```
$ mariadb -u username -p mtil < backup_mtile.sql
```

- Ahora ya tenemos todo listo para iniciar el servidor, el siguiente paso sería introducir la carpeta del proyecto dentro de un directorio del servidor. Una vez dentro instalaremos los *node_modules* y ejecutaremos el proceso con *pm2*.

```
$ npm install
$ pm2 start app.js
```

- Si no ha habido ningún error, accediendo a la *IP* que nos ofrece nuestro proveedor al puerto 8088 estará funcionando la *API*. Además de la *API* se está sirviendo la carpeta *public*, por lo que estará sirviendo también en esa dirección el *Frontend*.

8.1.2. Instalación con docker-compose

Una vez descargado el proyecto, lo descomprimos en un directorio y ejecutamos *docker*.

```
$ docker-compose up
```

Automáticamente se ejecutarán los tres contenedores:

1. Phpmyadmin: para la gestión de la *BBDD*. Acceso en <http://localhost:8090>.
2. MariaDB: para poder trabajar con *BBDD*.
3. NodeJS: contenedor que sirve la *API* y el *Frontend*. Acceso en <http://localhost:8088>.

8.2. Manual de Uso

8.2.1. Formulario Login

Para pasar el control del login se debe de insertar un email válido y una contraseña de al menos 6 caracteres de longitud.

En el caso de marcarse los campos con un borde de color rojo, prestar atención al mensaje que se muestra bajo el campo.

En el caso de recibir un mensaje de error al conectar con la *BBDD* es que la conexión no se ha establecido correctamente, contactar con su administrador.

En el caso de recibir el error “*usuarios/passsword incorrectos*”, inténtelo el acceso de nuevo pasados unos instantes.

8.2.2. Formulario Creación/Actualizar orden

Todos los siguientes campos del formulario son obligatorios: *Factoría, Sección, Número, Máquina, Tipo Orden, Tipo avería, Técnico, Avisado por, Observaciones*, y la *Fecha Aviso, Fecha Fin, Inicio trabajo y Fin trabajo*. Tenga en cuenta que ninguna fecha puede ser previa a la *Fecha aviso*, sólo puede ser igual la fecha *Inicio trabajo*. Lo mismo ocurre con las *Fechas de Fin*, ambas pueden ser iguales, pero nunca podrán ser previas o iguales a las *Fecha aviso* e *Inicio trabajo*. La fecha de *Fin Trabajo* no puede ser posterior a la *Fecha Fin*.

Para agregar operaciones son obligatorios los campos Operación y Tiempo. Operación requiere de al menos 2 caracteres y Tiempo sólo acepta valores enteros (1, 2, 3, 8, 15, 24...), o decimales de 0.25, 0.5 y 0.75.

Para agregar fichajes se debe elegir obligatoriamente el técnico y las fechas. Por defecto las fechas es la hora actual y la hora actual más 1 hora.

Para agregar materiales debe de buscar por descripción del producto, una vez lo encuentre en la lista haga click en él y seleccione una cantidad. La cantidad mínima es 1, la máxima 100 y no se aceptan decimales.

Todos los datos introducidos sólo se guardan una vez se pulsa el botón de Crear Orden o el de Guardar, de lo contrario se perderán.

8.2.3. Sección Administración (Acceso exclusivo administradores).

Tenga en cuenta que las credenciales de superadmin no pueden ser cambiadas. El resto de usuarios pueden editarse (haciendo click en la llave inglesa), y desactivarse o activarse pulsado al icono de agregar o eliminar, según el estado en el que se encuentre el usuario. Preste

especial atención al editar un usuario, una vez editado se perderá su antigua contraseña y deberá ingresar una nueva por seguridad.

Para el agregado de nuevos usuarios todos los campos son requeridos. El campo de email debe contener un email válido, y los campos de *Nombre* y *Password* deben de tener al menos 4 caracteres. Por defecto los usuarios son creados sin permisos de administrador, para crear administradores, debe pulsar el switch y debe de mostrarse iluminado.

Para eliminar órdenes debe de introducir un número de orden válido, de lo contrario devolverá un error. Las órdenes borradas no se pueden recuperar. En caso de borrado accidental, contacte con su administrador.

8.2.4. Sección Técnicos.

Sólo administradores podrán ver más detalles de los técnicos.

Una vez se pulsa “ver más” se nos abrirá una ventana con los datos del técnico. El único campo no-obligatorio es el de Observaciones. Los valores de *DNI*, *Tel. Móvil* y *Email* sólo aceptan valores válidos. Para la imagen de un técnico se recomienda la extensión .jpg, aunque son soportadas .png, .jpeg, y .gif. Las dimensiones recomendadas son 600px de altura x 400px de anchura. El tamaño máximo soportado del archivo son 2 Megabytes.

Para crear un nuevo técnico pulse el botón situado en el borde inferior derecha de la pantalla y siga las indicaciones anteriores.

En caso de borrar un técnico, no se puede recuperar. En caso de borrado accidental, contacte con el administrador.

8.2.5. Sección Calendario.

Aquí se muestran todas las órdenes según el día que han sido creadas y se alargarán hasta la fecha de fin. Si se pulsa sobre una orden pueden verse más detalles de la misma o bien acceder al formulario principal mediante el botón *Ver orden* para editarla.

El calendario ofrece la posibilidad de filtrar por Mes, Semana, Día y Agenda.

En el caso de que las órdenes no quepan físicamente por tamaño en la caja del día, se mostrará un enlace para ver las órdenes en su día correspondiente.

8.2.6. Sección Documentaciones.

En esta sección el usuario debe escoger una de las factorías y una sección obligatoriamente. Después se le mostrarán los documentos disponibles en el caso de que los hubiera, en caso contrario, se mostrará el mensaje “No se ha encontrado documentación”.

Para ver un documento en detalle haga click en él y después de un instante se abrirá el documento. Para cambiar entre páginas, sitúe el cursor por encima del documento y aparecerán unos botones emergentes que permiten moverse entre las páginas. Para volver atrás, pulse la flecha situada en la parte superior izquierda o bien fuera del documento. Si quiere eliminar el documento, pulse el botón de la papelera. En caso de borrar un documento, no se puede recuperar.

8.2.7. Sección Almacén.

Por defecto en el Almacén no se muestran resultados, para buscar un ítem en concreto, debe pulsar la lupa e introducir en el campo de texto lo que desea buscar. El sistema busca por referencia y por descripción del ítem, por lo que con conocer una de las dos opciones será suficiente.

Al pulsar un ítem se abrirá una nueva ventana desde la que se puede editar el mismo. Todos los campos son obligatorios. Al pulsar “*Actualizar*” se guardará el ítem con los valores del formulario. Si pulsa “*Eliminar*” se borrará el ítem del sistema y no se puede recuperar. En caso de borrado accidental, contacte con su administrador.

Para el agregado de un nuevo ítem pulse el botón situado en la parte inferior derecha y rellene el formulario correctamente. Tenga en cuenta que todos los campos son obligatorios y el *Código* no se puede cambiar una vez introducido el ítem en el sistema, por lo que se recomienda prestar atención en el momento del creado. *Cantidad* y *Stock Mínimo* deben de ser números enteros.

Si en el sistema se detecta que algún ítem es igual o menor al *Stock Mínimo* indicado, aparecerá una alarma en la parte superior derecha indicando que productos deben de pedirse.

8.2.8. Sección Estadísticas.

En esta sección el usuario sólo puede visualizar gráficos y descargar una captura del gráfico que lo permita. Esta página puede demorar algo más que el resto de páginas, debido a la carga gráfica que supone para los dispositivos.

8.2.9. Sección Histórico.

Aquí puede consultar todas las órdenes actuales que están almacenadas en la BBDD ordenadas por la fecha más reciente. Si pulsa en los identificadores de las columnas, la tabla se reordenará dinámicamente según la columna pulsada, por lo general de forma alfabética o de menor a mayor. Si se pulsa sobre una orden se puede ver más detalles acerca de la misma, y si se vuelve a pulsar “Ver Orden”, la página redirigirá a la orden seleccionada.

8.3. Soluciones a posibles errores

Algunos de los errores más comunes están contemplados en el Manual del Usuario en el apartado 8.2, no obstante pueden ocurrir otros errores. La siguiente lista incluye algunos de ellos:

Posible errores	
Errores	Soluciones
Pantalla en blanco después de realizar una operación.	Puede deberse a un error interno de la aplicación, se recomienda reiniciar el navegador y reintentar la operación. En caso de persistir el problema, contactar con su administrador.
Aparece aviso “ <i>error al realizar la conexión con la BBDD</i> ”.	Este error ocurre porque el servidor no está funcionando correctamente y la aplicación no puede conectarse. La recomendación es esperar unos minutos. En caso de persistir el problema, contactar con su administrador.
La aplicación “no se actualiza” al moverse entre secciones.	La autenticación de usuarios funciona con un token y este token está limitado a 2 horas, pasado ese tiempo caducará y deberá renovarse. La aplicación lo hace automáticamente, pero existe la posibilidad de que no lo haga. Se recomienda hacer logout en la aplicación y volver a autenticarse. En caso de persistir el problema, contactar con su administrador.
Hay gráficas de estadísticas que no se muestran.	Para que las gráficas se muestren, debe de haber datos que mostrar, por el contrario se mostrará vacía. Si está seguro de que existe información y no se muestra, contacte con su administrador.
El documento que he subido “no se muestra”.	Puede ocurrir que el documento este corrupto, pruebe a borrarlo y subirlo de nuevo. En caso de no mostrarse, se recomienda probar con otro documento y si persiste el problema contacte con su administrador.
Error: El tamaño máximo de las imágenes es de 2Mb.	El tamaño máximo soportado para las imágenes es de 2Mb, debe de probar con una imagen menos pesada.
Error: El tamaño máximo de las fichero es de 150Mb.	Debe de subir ficheros con menor tamaño. Si desea subir archivos más pesados, contacto con su administrador.

Tabla 8.3.1. Posibles errores y soluciones.