Desarrollo de un prototipo de aplicación web y móvil para la gestión de obras de instalación de fibra óptica - Parte 1.

Grado en Ingeniería Informática

Trabajo de fin de grado

Julio de 2017, Las Palmas de Gran Canaria

Autor: Alejandro Pérez Martín y Tutores: Dr. Alexis Quesada Arencibia y D. Jonathan Alemán Alemán

Resumen

Este proyecto surge bajo la necesidad de resolución de un problema real, en un contexto real, donde se pretende dar soporte a la gestión de obras de instalación de fibra óptica, permitiendo así comunicación bidireccional eficiente entre los diferentes intervinientes de la misma.

El objetivo de este proyecto es el análisis, diseño e implementación de un panel de administración que permita la creación y edición de las obras, así como la asignación de empleados a la misma que lleven a cabo las tareas pertenecientes a ella.

Como objetivo secundario se nos presenta la posibilidad de experimentar en el campo del desarrollo híbrido de aplicaciones para móviles, así como el estudio de diferentes tecnologías totalmente novedosas para el autor de este trabajo.

Abstract

This project raises under the necessity of a real problem resolution, in a real context, to give support to optical fiber installation works, thus allowing an efficient and bidirectional communication between all participants.

The main objective of this project is an administration panel analysis, design and implementation that allows the works creation and editing, as well as the employees allocation to it that perform it tasks.

The secondary project objective is the experimenting possibility inside the hybrid mobile applications development, as well as the study of different technologies totally novel for the author of this work.

Índice

Índice de ilustraciones

Índice de tablas

Estructura del documento

Durante el desarrollo de la memoria, hemos decidido dividirla en cinco grandes bloques que abarcan la totalidad del proyecto.

1. **Bloque 1. Introducción y Contextualización:** En este bloque presentaremos el problema que pretende resolver el proyecto y su contexto. Además, se presentan las soluciones actuales que utilizan los usuarios, qué aporta este proyecto y qué competencias cubre.
2. **Bloque 2. Desarrollo del proyecto:** En este bloque se recogen las diferentes fases por las que hemos pasado durante el desarrollo del proyecto. Los problemas con los que nos hemos ido encontrando durante su desarrollo y qué decisiones hemos tomado.
3. **Bloque 3. Conclusiones y Trabajos Futuros:** A lo largo de este bloque concluiremos la memoria reflexionando acerca de qué nos ha aportado este proyecto y sugeriremos qué podría incluir futuras versiones del desarrollo.
4. **Bloque 4. Fuentes de Información:** Aquí recogeremos las fuentes de información utilizadas durante el desarrollo del proyecto.
5. **Bloque 5. Anexos:** Completaremos la memoria a través de los anexos que aportarán información adicional a consultar si el lector lo desea.

Como comentaremos en la introducción, debemos adelantar que este proyecto ha sido dividido desde el principio en tres grandes partes, una conjunta y dos desarrolladas por diferentes alumnos. Por lo que ambas memorias siguen una estructura muy similar y comparten grandes similitudes en su desarrollo a excepción del apartado de implementación, trabajos futuros y aquellos intrínsecamente personales como las conclusiones.

1. Bloque 1. Introducción y Contextualización

1.1. Introducción

Es indudable la expansión actual de las tecnologías de la información, esto hace necesaria una mejor cobertura del servicio que las compañías proveedoras de internet brindan a sus clientes. Para ello en muchas ocasiones se requiere de la instalación de nuevas líneas de fibra óptica que sean accesibles para el cliente.

Para hacernos una idea del número de instalaciones de fibra óptica en España, la operadora Telefónica cerro el año 2015 en torno a los 14 millones de inmuebles cableados con fibra óptica FTTH.

Estas instalaciones/obras tienen una documentación asociada de vital importancia para el desarrollo de la misma. Muchas veces se transmite esta información entre los diferentes empleados de la empresa instaladora de formas poco ortodoxas, dando lugar a malentendidos y pérdidas de información. Estas pérdidas tienen un impacto directo en su capacidad de facturación, ya que requieren de esta documentación para acreditar su actividad.

Por lo tanto, se hace necesario un medio que permita a las empresas instaladoras gestionar este flujo de información sin pérdidas y de manera más accesible y rápida.

La motivación principal de este trabajo es la búsqueda de una mejora de la organización y gestión de trabajos en empresas instaladoras de fibra óptica mediante el desarrollo de un portal de administración que permita gestionar y asignar tareas a empleados de una empresa, así como el desarrollo de una aplicación móvil personal para que los empleados puedan a su vez enviar la información necesaria para la cumplimentación de la misma.

Desde el punto de vista académico, este trabajo nos ha permitido el aprendizaje y puesta en práctica del *MEAN Stack* que está formado por el conjunto de frameworks y tecnologías: "*MongoDB*", "*Express.js*", "*AngularJS/Angular*", "*Node.js*".

Además, escogimos este trabajo porque nos permitía abordar y resolver un problema real.

Creemos oportuno resaltar que este proyecto ha sido dividido en tres grandes bloques, uno de ellos desarrollado de forma conjunta por dos alumnos y luego cada uno habiendo desarrollado un bloque de manera independiente.

Este proyecto se centra en el desarrollo conjunto de una *API REST* en *Node.js* y *MongoDB* y el análisis, diseño e implementación individual de una aplicación móvil, desarrollada con el framework *Ionic*, necesaria para visualización e interacción con los datos generados a partir de un panel de administración creado con *AngularJS*, siendo este último, desarrollado por mi compañero Álvaro Romero Perdomo *Desarrollo de aplicación web y móvil para la gestión de las obras de instalación de fibra óptica – Parte 2*.

1.2. Estado actual

Como hemos citado anteriormente, las instalaciones/obras tienen una documentación asociada de vital importancia para el desarrollo o facturación de la misma.

En este intercambio de información participan tres agentes principalmente:

- En primer lugar, la operadora que desea realizar la obra.
- En segundo lugar, la empresa instaladora.
- En tercer lugar, los empleados de la instaladora asignados a dicha obra.

Por lo tanto, cuando hablamos de esta documentación nos referimos a dos tipos fundamentalmente: Una documentación necesaria para el comienzo y desarrollo de la obra (generada por la operadora, donde se indican las tareas a realizar y el material a utilizar dentro de la instalación), y otra documentación necesaria para la facturación de la misma (generada por los empleados de la instaladora que acreditan la realización de las tareas pertenecientes a la obra).

Actualmente la documentación necesaria para el comienzo y desarrollo de la obra es enviada por parte de la operadora a la instaladora a través del correo electrónico. Esta documentación está formada por:

- Una hoja de cálculo *Excel* donde están descritas las diferentes tareas a realizar en la obra y el material asignado a la misma.
- Una o varias fotos de la zona donde se va realizar la obra.
- Uno o varios planos en *PDF* de la zona donde se va realizar la obra.
- Y, opcionalmente, una presentación en *PowerPoint* donde se muestran diferentes localizaciones de la obra y permite la desambiguación en la realización de algunas tareas.

La documentación necesaria para la clausura y certificación/facturación de la obra es generada por los empleados de la instaladora a medida que van finalizando las tareas pertenecientes a la misma. Esta documentación está formada por:

- Fotos donde se muestran las tareas pertenecientes a la obra terminadas.
- Datos numéricos, como por ejemplo longitud real de cable colocado en la instalación.
- Relación tareas – estado de las tareas.

Actualmente esta documentación es transmitida por los empleados al jefe/encargado de la obra a través de aplicaciones móviles de mensajería instantánea como *WhatsApp* o por correo electrónico en algunos casos, debiendo éste almacenar la información recibida en hojas de cálculo y carpetas creadas en el Sistema Operativo de su ordenador personal. Esto genera, en muchos casos, confusión y pérdida de información, ya que no existe ningún tipo de sincronización entre la documentación que pueda tener el encargado y el empleado.

1.3. Objetivos

1.3.1. Objetivos generales

Como hemos citado anteriormente, el flujo bidireccional de información relacionada con una obra, se transmite entre los diferentes actores de manera poco ortodoxa, dando lugar a, en algunos casos, pérdida de la misma y una gestión por parte del encargado de la obra que se podría automatizar.

Éste es precisamente el objetivo de este trabajo fin de título, automatizar tareas que hasta ahora realizaba el encargado de la obra, proporcionar un medio de transmisión de la información mucho más fiable y que además se garantice el almacenamiento seguro de la misma mediante una base de datos.

Para conseguir estos objetivos se pretende:

- Desarrollar una *API REST* en *Node.js* y *MongoDB* que almacene tanto la información de la obra como tareas asociadas a la misma, empleados que trabajan en la obra, material necesario para llevarla a cabo, ficheros adjuntos y el estado de la obra entre otros.
- La creación de un portal de administración en *AngularJS* que permita al encargado de obras y responsable de la empresa instaladora crear obras a las que asignarle trabajadores de su empresa, crear tareas y subtareas para la misma, adjuntarle archivos necesarios para desempeñarla y controlar el estado de la obra en todo momento.
- Analizar, diseñar e implementar una aplicación móvil con *Ionic* y *Angular* que conecte a los empleados con la API REST de manera directa y sencilla, y que les permita gestionar de manera fácil y eficaz sus tareas, actualizando el estado en que se encuentran, cumplimentando información adicional e/o incorporando archivos multimedia, como por ejemplo fotografías, que permitirán a los responsables tener una visión más profunda y objetiva de una obra.

1.3.2. Objetivos académicos

Con respecto a los objetivos académicos que nos hemos planteado en la realización de este trabajo fin de título intervienen además de los descritos en el "Reglamento General para la realización y evaluación de trabajos de fin de título" nuestro interés en aprender nuevas tecnologías como el *MEAN Stack* en lugar de optar por otras más comunes.

Además, nos ha servido para aprender desarrollo web desde un punto de vista más autónomo y en un contexto real y poner en práctica conocimientos adquiridos en asignaturas cursadas durante la carrera.

1.4. Normativa y legislación

1.4.1. Ley de protección de Datos de Carácter Personal

La Ley Orgánica 15/1999 de 13 de diciembre de Protección de Datos de Carácter Personal, (LOPD), es una ley orgánica española que tiene por objeto garantizar y proteger, en lo que concierne al tratamiento de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas, y especialmente de su honor, intimidad y privacidad personal y familiar. Fue aprobada por las Cortes Generales el 13 de diciembre de 1999. Esta ley se desarrolla fundamentándose en el artículo 18 de la constitución española de 1978, sobre el derecho a la intimidad familiar y personal y el secreto de las comunicaciones.

Su objetivo principal es regular el tratamiento de los datos y ficheros, de carácter personal, independientemente del soporte en el cual sean tratados, los derechos de los ciudadanos sobre ellos y las obligaciones de aquellos que los crean o tratan.

Esta ley afecta a todos los datos que hacen referencia a personas físicas registradas sobre cualquier soporte, informático o no. Quedan excluidas de esta normativa aquellos datos recogidos para uso doméstico, las materias clasificadas del estado y aquellos ficheros que recogen datos sobre Terrorismo y otras formas de delincuencia organizada (no simple delincuencia).

A partir de esta ley se creó la Agencia Española de Protección de Datos, de ámbito estatal que vela por el cumplimiento de esta Ley.

El órgano de control del cumplimiento de la normativa de protección de datos dentro del territorio español, con carácter general es la Agencia Española de Protección de Datos (AEPD), existiendo otras Agencias de Protección de Datos de carácter autonómico, en las Comunidades Autónomas de Cataluña y en el País Vasco.

Las sanciones se dividen en tres grupos dependiendo de la gravedad del hecho cometido, siendo España el país de la Unión Europea que tiene las sanciones más altas en materia de protección de datos. Dichas sanciones dependen de la infracción cometida.

Se dividen en:

- Las sanciones leves van desde 900 a 40.000 €.
- Las sanciones graves van desde 40.001 a 300.000 €.
- Las sanciones muy graves van desde 300.001 a 600.000 €.

Pese al importe de las sanciones, existen muchas empresas en España que todavía no se han adecuado a la misma, o lo han hecho de forma parcial o no revisan de forma periódica su adecuación; por lo que resulta esencial el mantenimiento y revisión de la adecuación realizada.

En el sector público, la citada Ley regula igualmente el uso y manejo de la información y los ficheros con datos de carácter personal utilizados por todas las administraciones públicas.

La Agencia Española de Protección de Datos (AEPD) fue creada en 1994 conforme a lo establecido en la derogada LORTAD. Su sede se encuentra en Madrid, si bien las Comunidades Autónomas de Madrid, País Vasco y Cataluña han creado sus propias Agencias de carácter autonómico.

Nuestras aplicaciones hacen uso de datos de carácter personal de sus usuarios, tales como su nombre, dirección, fecha de nacimiento, puesto de trabajo y actividad en el mismo, etc. Es por ello que debemos tomarnos muy en serio la LOPD garantizando su cumplimiento en todo momento mediante el uso de una base de datos robusta y que mantiene estos datos sensibles encriptados.

Además, los usuarios de dicha aplicación deben conocer en todo momento que sus datos personales están siendo almacenados como requisito para el uso de la misma.

1.5. Justificación de las competencias específicas cubiertas

1.5.1. Competencias comunes a la Ingeniería Informática

CII01

Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente.

Esta competencia queda cubierta por parte del alumno porque en este proyecto:

- Se ha realizado una etapa de análisis dónde se recogieron un gran número de requisitos expresados por el "cliente" que garantizan que la aplicación en desarrollo cumple las expectativas del mismo.
- Se realizó un proceso de diseño de la arquitectura del proyecto y de una interfaz asociada que fue validada por el "cliente".
- La información guardada por la aplicación se encuentra cifrada protegiendo así la integridad de las personas o de la información que en ella participan atendiendo a la Ley de protección de datos.

CII05

Conocimiento, administración y mantenimiento sistemas, servicios y aplicaciones informáticas.

Esta competencia queda cubierta por parte del alumno porque en este proyecto:

- La aplicación está desplegada sobre un entorno robusto y testeado proporcionado por el *MEAN Stack* que asegura el correcto funcionamiento de la aplicación en todo momento.

CII08

Capacidad para analizar, diseñar, construir y mantener aplicaciones de forma robusta, segura y eficiente, eligiendo el paradigma y los lenguajes de programación más adecuados.

Esta competencia queda cubierta por parte del alumno porque en este proyecto:

- Se ha realizado una etapa de análisis dónde se recogieron un gran número de requisitos expresados por el "cliente" que garantizan que la aplicación en desarrollo cumple las expectativas del mismo.
- Se realizó un proceso de diseño de la arquitectura del proyecto y de una interfaz asociada que fue validada por el "cliente".
- Se eligió el estándar de desarrollo web en el lenguaje de programación JavaScript (*MEAN Stack*) garantizando así la utilización de un lenguaje con gran cantidad de documentación accesible y una comunidad de usuarios creciente.

CII13

Conocimiento y aplicación de las herramientas necesarias para el almacenamiento, procesamiento y acceso a los Sistemas de información, incluidos los basados en web.

Esta competencia queda cubierta por parte del alumno porque en este proyecto:

- Se ha garantizado la persistencia de los datos introducidos en la misma mediante la utilización de una base de datos no relacional MongoDB, que permite el acceso a los datos de manera rápida, sencilla y segura.

CII16

Conocimiento y aplicación de los principios, metodologías y ciclos de vida de la ingeniería de software.

Esta competencia queda cubierta por parte del alumno porque en este proyecto:

- Se han aplicado principios de la ingeniería del software tales como la modularidad en la arquitectura y el código que permiten la ampliación de la herramienta en futuras iteraciones y la refactorización del código para facilitar su mantenimiento.

CII17

Capacidad para diseñar y evaluar interfaces persona computador que garanticen la accesibilidad y usabilidad a los sistemas, servicios y aplicaciones informáticas.

Esta competencia queda cubierta por parte del alumno porque en este proyecto:

- Se ha hecho uso de librerías como *AngularJS Material* basadas en *Google Material,* cuyo objetivo es la accesibilidad de la interfaz y su usabilidad.

TFG01

Ejercicio original a realizar individualmente y presentar y defender ante un tribunal universitario, consistente en un proyecto en el ámbito de las tecnologías específicas de la Ingeniería en Informática de naturaleza profesional en el que se sinteticen integren las competencias adquiridas en las enseñanzas.

1.6. Aportaciones

Este proyecto trata de resolver un problema real a través del desarrollo de un portal web de administración, una *API REST* y una aplicación móvil funcionales, intentando además realizar una aportación tecnológica, económica y social.

En el plano tecnológico intentamos dar un enfoque más innovador y actual a la gestión de obras por parte de la empresa instaladora aportando un software capaz de, prácticamente, abstraer tanto al empleado como al administrador de complicados y poco ortodoxos métodos de traspaso de información, convirtiendo este proceso en algo más simple y a la misma vez más robusto. Para conseguir este fin se ha hecho uso de tecnologías actuales en el campo del desarrollo web tales como el *Stack MEAN*. Además, la interfaz de usuario tanto del portal de administración como de la aplicación móvil está basada en estándares actuales como el *Material Design*.

Desde el punto de vista económico, este proyecto permite a las empresas instaladoras de fibra óptica facturar a las proveedoras de internet de manera eficiente e intachable evitando así pérdidas reales de dinero para ellas derivadas de la pérdida de información necesaria para la certificación y facturación de una obra debida a la forma en que ésta era gestionada.

En el ámbito social la aplicación trata de facilitar el trabajo a los empleados que, a la hora de generar información necesaria para la certificación, debían utilizar varias aplicaciones en su dispositivo para intercambiarla con el encargado de la obra. Ahora disponen de una aplicación centralizada y específica para el control de la información que les permite ahorrar tiempo y realizar las mismas tareas de una manera mucho más sencilla.

Desde el punto de vista del encargado de la obra se ha facilitado en gran medida el trabajo organizativo que éste debía hacer tanto a la hora de comenzar una nueva obra, como a la hora de certificar la misma.

1.7. Recursos

1.7.1. Recursos software

A continuación, se exponen las herramientas software utilizadas para el desarrollo del proyecto.

Sistemas operativos

El proyecto ha sido realizado en tres entornos diferentes como son Windows 10, Ubuntu 16.04 y Mac OS.

En un principio se decidió trabajar en Windows 10 y en Mac OS paralelamente, pero la instalación de *MongoDB* en Linux era mucho más directa y presentaba menos problemas en comparación con su versión en Windows.

Ante esta diversidad de entornos de desarrollo, se hizo uso de la tecnología de contenedores de *Docker* cuyas características se describen a continuación.

Docker

Docker es una tecnología de código abierto que le permite crear, ejecutar, probar e implementar aplicaciones distribuidas dentro de contenedores de software. Permite empaquetar una aplicación en una unidad estandarizada para el desarrollo de software, que contiene todo lo que necesita para funcionar: código, tiempo de ejecución, herramientas y bibliotecas del sistema, etc. *Docker* permite implementar las aplicaciones de forma rápida, fiable y sistemática, en cualquier entorno.

Sus principales características son:

- Consume pocos recursos, lo que permite desplegar multitud de contenedores en un mismo equipo físico.
- Se elimina el problema de dependencias de las aplicaciones.
- Los contenedores son livianos y facilitan su almacenaje, transporte y despliegue.
- Repositorio de imágenes públicos y privados de imágenes.
- Multiplataforma, podremos desplegar nuestros contenedores en multitud de entornos.

Sistema de control de versiones del código, GIT

Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente.

Git se ha convertido en un sistema de control de versiones con funcionalidad plena. Hay algunos proyectos de mucha relevancia que ya usan *Git*, en particular, el grupo de programación del núcleo Linux.

Dentro de *Git* hemos utilizado la metodología de trabajo, *git-flow*. Sus características principales son las siguientes.

El trabajo se organiza en dos ramas principales:

- Rama **master**: cualquier *commit* que pongamos en esta rama debe estar preparado para subir a producción
- Rama **develop**: rama en la que está el código que conformará la siguiente versión planificada del proyecto

Además de estas dos ramas, se proponen las siguientes ramas auxiliares: *Feature*, *Release* y *Hotfix*.

También se ha hecho uso de las *Issues* de *GitHub* para llevar un control de los errores/bugs a solucionar y mejoras a implementar en posteriores versiones.

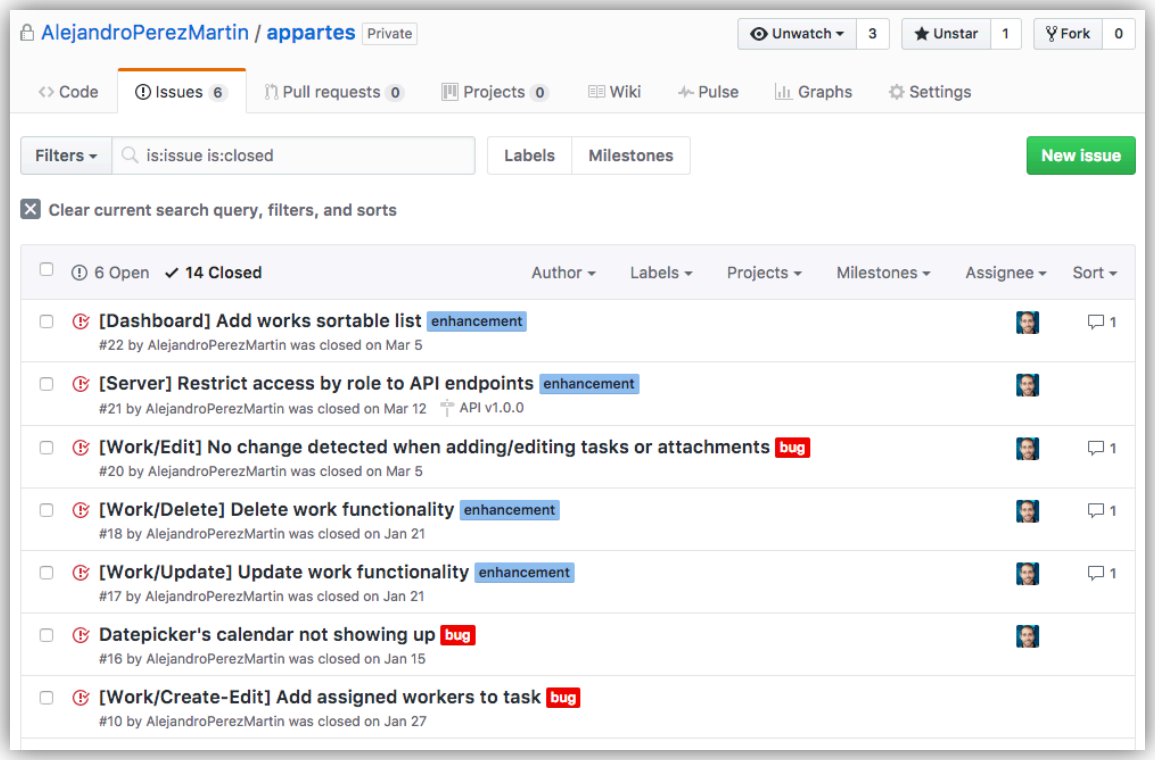

Ilustración 1. Captura de la página de incidencias de GitHub

Editor de texto/código, Visual Studio Code

Visual Studio Code es un editor de código open-source de Microsoft disponible para Mac OS, Linux, y Windows con soporte integrado para Git, auto-detección y auto-completado inteligente para decenas de lenguajes, herramientas de depuración y un amplio abanico de extensiones que convierten a este editor de código en todo un IDE (entorno de desarrollo integrado).

Visual Studio Code está desarrollado íntegramente empleando tecnologías web como Node.js, TypeScript, JavaScript y CSS y da soporte a lenguajes como CSS, HTML, JavaScript, JSON, Less, Sass, TypeScript, PHP, Python, Ruby, Perl, Objective-C, Java, C#, C++, etc.

Lenguaje de programación, JavaScript

JavaScript (abreviado comúnmente JS) es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas, aunque existe una forma de JavaScript del lado del servidor (Server-side JavaScript o SSJS). Su uso en aplicaciones externas a la web, por ejemplo, en documentos PDF, aplicaciones de escritorio (mayoritariamente widgets) es también significativo.

Las características principales de JavaScript son las siguientes:

- JavaScript es un lenguaje de secuencias de comandos basado en objetos e interpretado.
- Aunque tiene menos capacidades que los lenguajes orientados a objetos de altas prestaciones como C++ y Java, JavaScript es más que suficientemente eficiente para los propósitos para los que está creado.
- JavaScript no es una versión reducida de cualquier otro lenguaje (sólo está relacionado, distante e indirectamente, con Java, por ejemplo), ni es una simplificación de ningún lenguaje.
- JavaScript es un lenguaje limitado. Por ejemplo, no es posible escribir aplicaciones independientes en JavaScript y la capacidad de lectura y escritura de archivos es mínima.
- Las secuencias de comandos de JavaScript sólo pueden ejecutarse con un intérprete, que bien puede estar en un servidor Web o en un explorador de Web.
- JavaScript es un lenguaje en el que no necesita declarar los tipos de datos. Esto significa que no es necesario declarar explícitamente los tipos de datos de las variables. De hecho, no es posible declarar explícitamente los tipos de datos en JavaScript. Más aún, en muchos casos JavaScript realiza conversiones, automáticamente, cuando son necesarias. Por ejemplo, si intenta agregar un número a un elemento que contiene texto (una cadena), el número se convierte en texto.

Node.js

Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación ECMAScript, asíncrono, con I/O de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google.

Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables, como, por ejemplo, servidores web. Fue creado por Ryan Dahl en 2009 y su evolución está apadrinada por la empresa *Joyent*, que además tiene contratado a Dahl en plantilla.

Node.js es una forma de ejecutar JavaScript en el servidor, además de mucho más. Además de la alta velocidad de ejecución de *JavaScript*, la verdadera magia detrás de Node.js es algo que se llama Bucle de Eventos (*Event Loop*).

Para escalar grandes volúmenes de clientes, todas las operaciones intensivas I/O en Node.js se llevan a cabo de forma asíncrona. El enfoque tradicional para generar código asíncrono es engorroso y crea un espacio en memoria no trivial para un gran número de clientes (cada cliente genera un hilo, y el uso de memoria de cada uno se suma).

Para evitar esta ineficiencia, así como la dificultad conocida de las aplicaciones basadas en hilos, (*programming threaded applications*), Node.js mantiene un *event loop* que gestiona todas las operaciones asíncronas.

AngularJS

AngularJS (comunalmente llamado "*Angular.js*" o en sus últimas versiones, “Angular”), es un *framework* de JavaScript de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (*MVC*), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles.

La biblioteca lee el *HTML* que contiene atributos de las etiquetas personalizadas adicionales, entonces obedece a las directivas de los atributos personalizados, y une las piezas de entrada o salida de la página a un modelo representado por las variables estándar de *JavaScript*. Los valores de las variables de JavaScript se pueden configurar manualmente, o recuperados de los recursos *JSON* estáticos o dinámicos.

AngularJS se puede combinar con el entorno en tiempo de ejecución Node.js, el *framework* para servidor Express.js y la base de datos *MongoDB* para formar el conjunto *MEAN*.

MongoDB

MongoDB es un sistema de base de datos *No-SQL* orientado a documentos, desarrollado bajo el concepto de código abierto.

Forma parte de la nueva familia de sistemas de base de datos *No-SQL*. En lugar de guardar los datos en tablas como se hace en las bases de datos relacionales, *MongoDB* guarda estructuras de datos en documentos similares a JSON con un esquema dinámico (*MongoDB* utiliza una especificación llamada BSON), haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

Las características principales de *MongoDB* son:

- De propósito general, casi tan rápida como las bases de datos *No-SQL* de tipo *clave:valor*, y con casi todas las funcionalidades de las bases de datos relacionales.
- Alta disponibilidad.
- Escalabilidad, desde un servidor aislado a arquitecturas distribuidas de grandes *clusters*.
- *Aggregation Framework*, procesamiento *batch* de datos para cálculos agrupados utilizando operaciones nativas de *MongoDB*.
- Auto balanceado de carga, a través de distintos *shards*.
- Replicación nativa, sincronización de datos entre servidores.
- Seguridad, autenticación, autorización, etc.
- Gestión avanzada de usuarios.
- Recuperación automática ante errores.
- Se actualiza sin dejar de dar servicio.
- No tiene los cuellos de botella que se producen en las bases de datos relacionales (RDBMS).
- Utiliza objetos *JSON* para guardar y transmitir la información.

En el desarrollo de la aplicación utilizamos Mongoose como driver de MongoDB, lo que facilitó la generación de modelos y esquemas que luego se verían reflejados en la base de datos de manera mucho más clara y directa, así como proporcionar algunas funciones adicionales como el embebido de documentos en las referencias de los esquemas.

Express.js

Express.js o simplemente *Express*, es un *framework* para el desarrollo de aplicaciones web para *Node.js* distribuido de manera libre y gratuita bajo licencia *MIT*. Está diseñado para construir aplicaciones web y *APIs*. Se le considera el *framework* del lado del servidor estándar para *Node.js*. Por lo tanto, podríamos decir que *Express.js* es una infraestructura de aplicaciones web Node.js mínima y flexible que proporciona un conjunto sólido de características para las aplicaciones web y móviles.

JSON

JSON, acrónimo de *JavaScript Object Notation*, es un formato de texto ligero para el intercambio de datos. *JSON* es un subconjunto de la notación literal de objetos de *JavaScript* aunque hoy, debido a su amplia adopción como alternativa a *XML*, se considera un formato de lenguaje independiente.

Una de las ventajas de *JSON* sobre *XML* como formato de intercambio de datos es que es mucho más sencillo escribir un analizador sintáctico (*parser*) de *JSON*. En *JavaScript*, un texto *JSON* se puede analizar fácilmente usando la función *eval*, lo cual ha sido fundamental para que *JSON* haya sido aceptado por parte de la comunidad de desarrolladores *AJAX*, debido a la ubicuidad de *JavaScript* en casi cualquier navegador web.

TypeScript

TypeScript es un lenguaje de programación de código abierto desarrollado y presentado por Microsoft hace unos tres años. Es un superset de JavaScript que esencialmente añade capacidades de POO como es el tipado estático y objetos basados en clases.

Extiende la sintaxis de JavaScript, por medio de un lenguaje propio que compila ficheros en lenguaje JavaScript original, asegurando la compatibilidad con todos los navegadores, servidores y sistemas operativos.

Ionic

Ionic es un framework open source para el desarrollo de aplicaciones híbridas que permite crear aplicaciones multiplataforma utilizando *HTML5* optimizado para móvil, *CSS3*, componentes *JavaScript*, gestos y herramientas para la construcción de aplicaciones altamente interactivas. Construido con *Sass* y optimizado para *Angular* con *TypeScript* permite asegurar aplicaciones robustas, rápidas y escalables.

Las aplicaciones son híbridas, ¿Qué quiere decir eso? Que puedes desarrollar una misma aplicación y ejecutarla en Android, iOS y Windows Phone sin tener que desarrollarla en el correspondiente lenguaje nativo de cada plataforma.

Una de las características de *Ionic* Framework es que está construido para ser rápido debido a la mínima manipulación del DOM, sin utilizar *jQuery* y con aceleraciones de transiciones por hardware.

Angular

Angular es otro framework, no simplemente una nueva versión de *AngularJS*. Todos esos problemas, difíciles de solucionar con la tecnología usada por *AngularJS* han sido los que han impulsado a sus creadores a desarrollar desde cero una nueva versión del framework. La nueva herramienta está pensada para dar cabida a todos los usos dados por los desarrolladores, llevar a *JavaScript* a un nuevo nivel comparable a lenguajes más tradicionales, siendo además capaz de resolver de una manera adecuada las necesidades y problemas de la programación del lado del cliente.

Entre las nuevas características podemos destacar:

- Mejor rendimiento que *AngularJS*.
- Inyección de dependencias.
- Uso de *TypeScript*.
- Desarrollo Móvil: El desarrollo de aplicaciones de escritorio es mucho más fácil cuando primero se manejan los problemas de rendimiento en el desarrollo móvil.
- Modularidad: Para desarrollar una nueva funcionalidad esta se empaqueta en un módulo, produciendo un núcleo más ligero y más rápido.
- Compilación AOT: la aplicación está pre-compilada, reduciendo así el tiempo de carga de la misma.

HTML

HTML, sigla en inglés de *HyperText Markup Language* (lenguaje de marcas de hipertexto), hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código *HTML*) para la definición de contenido de una página web, como texto, imágenes, videos, juegos, entre otros. Es un estándar a cargo del *World Wide Web Consortium* (*W3C*) o Consorcio *WWW*, organización dedicada a la estandarización de casi todas las tecnologías ligadas a la web, sobre todo en lo referente a su escritura e interpretación. Se considera el lenguaje web más importante siendo su invención crucial en la aparición, desarrollo y expansión de la *World Wide Web* (*WWW*). Es el estándar que se ha impuesto en la visualización de páginas web y es el que todos los navegadores actuales han adoptado.

El lenguaje *HTML* basa su filosofía de desarrollo en la diferenciación. Para añadir un elemento externo a la página (imagen, vídeo, *script*, entre otros.), este no se incrusta directamente en el código de la página, sino que se hace una referencia a la ubicación de dicho elemento mediante texto. De este modo, la página web contiene solamente texto mientras que recae en el navegador web (interpretador del código) la tarea de unir todos los elementos y visualizar la página final. Al ser un estándar, *HTML* busca ser un lenguaje que permita que cualquier página web escrita en una determinada versión, pueda ser interpretada de la misma forma (estándar) por cualquier navegador web actualizado.

RxJS

Reactive Extension for JavaScript (RxJS) es una librería que nos permite trabajar con flujos de datos asíncronos. Esta librería nos permite crear programas mediante la composición de flujos de datos asíncronos o basados en eventos.

Utilizando *RxJS* se puede representar flujos de datos asíncronos mediante Observables y manipular estos flujos con diferentes operadores como si fueran simples colecciones de datos.

GitHub

GitHub es una forja (plataforma de desarrollo colaborativo) para alojar proyectos utilizando el sistema de control de versiones *Git*. Utiliza el framework *Ruby on Rails* por *GitHub*, Inc. (anteriormente conocida como Logical Awesome). Desde enero de 2010, GitHub opera bajo el nombre de GitHub, Inc. El código se almacena de forma pública, aunque también se puede hacer de forma privada, creando una cuenta de pago.

CSS

Hojas de estilo en cascada (o *CSS*, siglas en inglés de *Cascading Stylesheets*) es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado. Es muy usado para establecer el diseño visual de las páginas web, e interfaces de usuario escritas en *HTML* o *XHTML*; el lenguaje puede ser aplicado a cualquier documento *XML*, incluyendo *XHTML*, *SVG*, *XUL*, *RSS*, etcétera. También permite aplicar estilos no visuales, como las hojas de estilo auditivas.

Junto con *HTML* y *JavaScript*, *CSS* es una tecnología usada por muchos sitios web para crear páginas visualmente atractivas, interfaces de usuario para aplicaciones web, y *GUIs* para muchas aplicaciones móviles (como *Firefox OS*).

CSS está diseñado principalmente para marcar la separación del contenido del documento y la forma de presentación de este, características tales como las capas, los colores y las fuentes. Esta separación busca mejorar la accesibilidad del documento, proveer más flexibilidad y control en la especificación de características, permitir que varios documentos *HTML* compartan un mismo estilo usando una sola hoja de estilos separada en un archivo *.css*, y reducir la complejidad y la repetición de código en la estructura del documento.

Sass

Sass es un pre-procesador de CSS que nos permiten escribir estilos con una sintaxis particular y mejorada que es luego transformada a código CSS natural.

La principal ventaja de SASS es la posibilidad de convertir los CSS en algo dinámico. Permite trabajar mucho más rápido en la creación de código con la posibilidad de crear funciones que realicen ciertas operaciones matemáticas y reutilizar código gracias a los mixins, variables que nos permiten guardar valores, etc.

Librerías utilizadas

En informática, una librería (del inglés *library*) es un conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca.

A diferencia de un programa ejecutable, el comportamiento que implementa una librería no espera ser utilizada de forma autónoma (un programa sí: tiene un punto de entrada principal), sino que su fin es ser utilizada por otros programas, independientes y de forma simultánea. Por otra parte, el comportamiento de una librería no tiene por qué diferenciarse en demasía del que pudiera especificarse en un programa. Es más, unas librerías pueden requerir de otras para funcionar, pues el comportamiento que definen refina, o altera, el comportamiento de la biblioteca original; o bien la hace disponible para otra tecnología o lenguaje de programación.

La librería de interfaces de usuario utilizada para el desarrollo *front-end* del panel de administración ha sido *AngularJS Material,* un *framework* de interfaces de usuario adaptado a *AngularJS* basado en *Material Design* de Google. Esta librería facilita en gran medida el trabajo de maquetación y colocación de elementos en *HTML* gracias a componentes que siguen el estándar *responsive.*

Pencil Project

Software de modelado bajo el estándar UML utilizado para el diseño de los diagramas de casos de uso.

StarUML

Software de modelado bajo el estándar *UML* utilizado para el diseño de los diagramas de casos de uso.

UML

El lenguaje unificado de modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el Object Management Group (OMG).

Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados.

Es importante remarcar que UML es un "lenguaje de modelado" para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo.

1.7.2. Recursos hardware

Durante el desarrollo del proyecto se utilizaron diferentes equipos (con diferentes sistemas operativos) para comprobar el comportamiento del portal de administración en diferentes navegadores.

Estos equipos utilizados fueron tres fundamentalmente:

- Portátil HP Pavilion G6 con sistema operativo Ubuntu 16.04 y navegadores Mozilla Firefox y Google Chrome.
- Ordenador de sobremesa con sistema operativo Windows 10 y navegadores Mozilla Firefox y Google Chrome.
- Portátil Macbook Pro 15” con sistema operativo Mac OS Sierra y navegadores Google Chrome y Vivaldi.

1.8. Planificación temporal

A continuación, se muestra la estimación temporal inicial que asignamos a los diferentes módulos del desarrollo del proyecto, así como la dedicación final y la variación entre ellas.

Fase	Duración (horas)
Formación y preparación del entorno de trabajo	20
Análisis y diseño	30
Desarrollo e implementación	180
Prueba	20
Documentación	30

Tabla 1. Planificación inicial

Como podemos observar en la primera tabla, estimamos que íbamos a dedicar un tiempo mayor al apartado del análisis del problema y diseño de la solución que a la formación y a la preparación del entorno de trabajo necesario para la realización del proyecto. Al final, como podemos observar en la segunda tabla, estos tiempos se invirtieron, llevándonos aproximadamente el doble de tiempo la preparación del entorno de trabajo que las etapas de análisis y diseño. Esto fue debido a que surgieron problemas de incompatibilidad con algunas herramientas que empleamos al principio y porque, en mi caso, no había tenido experiencia anteriormente en desarrollar un proyecto.

Fase	Duración (horas)
Formación y preparación del entorno de trabajo	40
Análisis y diseño	20
Desarrollo e implementación	200
Prueba	10
Documentación	60

Tabla 2. Dedicación final

En un principio también estimamos que la realización de la documentación relativa al proyecto iba a tener una duración aproximada de 30 horas y pudimos observar que al final este tiempo se extendió en gran medida.

También es interesante remarcar, aunque sea común, que la previsión temporal destinada a la implementación del proyecto fue quizás algo optimista.

2. Bloque 2. Desarrollo del Proyecto

2.1. Análisis

Esta etapa tiene por objeto realizar un análisis global del sistema, estableciendo los requisitos de todos los elementos del sistema y luego asignar al software la parte de los requisitos que le afectan. Este planteamiento del sistema es esencial cuando el software debe interrelacionarse con otros elementos, tales como personas, hardware y bases de datos.

2.1.1. Primera reunión y contextualización del problema

La etapa de análisis comienza cuando tenemos el primer contacto con el encargado de la empresa instaladora, cuyo problema resolvemos en este proyecto. En esa reunión conocemos el problema de una manera global y lo tomamos como una primera toma de contacto donde ya podemos capturar algunos requisitos y familiarizarnos con el contexto del mismo.

En esta primera reunión obtuvimos que:

- La empresa actualmente se comunica con sus empleados a través de llamadas telefónicas o a través de aplicaciones de mensajería instantánea como *WhatsApp*.
- La empresa recibe la información de la obra a través de hojas de cálculo (*Excel*) y presentaciones (*PowerPoint*).
- Cuando los empleados terminan una determinada tarea de la obra deben fotografiarla. Dicha foto es enviada en múltiples casos al encargado de la obra a través de correo electrónico o aplicaciones de mensajería instantánea como *WhatsApp*.
- Los empleados a veces se dejan atrás alguna foto o alguna tarea sin hacer, por lo que posteriormente deben volver a la obra, con la consecuente pérdida de tiempo.
- La factura de la obra es calculada y generada manualmente por el encargado de la obra.
- Las obras están formadas por tareas y además tienen asociadas materiales y empleados.
- Es tarea del encargado de la obra dibujar sobre plano, las conexiones que se deben realizar en la obra, así como por dónde deben pasar los cables, cuántas cajas de conexión y de qué tipo son necesarias en la obra.
- Por lo tanto, una posible estructura de la aplicación a priori podría ser la generación de un portal web de administración (donde el encargado puede generar las obras y asignar empleados a las mismas) y una aplicación móvil para los empleados de la obra donde podrían acceder a la obra que tienen asignada e ir notificando el proceso de la misma.

2.1.2. Glosario de lenguaje propio del contexto del problema

Palabra	Descripción
Obra	Eje central del proyecto, está formada por una o más tareas, material asociado a ella y en la que intervienen uno o más empleados, al final de la misma se lleva a cabo un proceso de certificación por parte de la empresa instaladora a la empresa contratante.
Tarea	Subconjunto perteneciente a una obra y que puede contener subtareas. Determinan el estado de finalización de una obra y pueden ser generada por los empleados.
Certificación	Al finalizar una obra, este es el proceso por el cual la empresa que la ha realizado, mediante la documentación generada de la finalización de las tareas, finaliza la obra y genera un certificado que le permite el cobro de la misma en función de un baremo.
Empleado	Persona que acude a la obra que ha sido asignada teniendo que realizar las tareas pertenecientes a ella y pudiendo crear nuevas tareas si fuera necesario.
Encargado de obra	Persona que se encarga de dar de alta las obras en el sistema y asignar empleados a ella, una vez terminada la obra le corresponde realizar el proceso de certificación.

Tabla 3. Glosario de lenguaje propio del contexto del problema

2.1.3. Generación del prototipo

Después de esa primera reunión, decidimos, en base a los requisitos capturados, trabajar en un prototipo no funcional, de la aplicación móvil para los empleados, el cual nos permitiría ajustar nuestra posible solución a la realidad planteada por él. Para ello utilizamos el software *Pencil*.

Los prototipos son una visión preliminar del sistema futuro que se implementará. La elaboración de prototipos de un sistema de información es una técnica valiosa para la recopilación rápida de información específica acerca de los requerimientos de información de los usuarios.

En nuestro caso, el prototipo recogía una posible interfaz gráfica para la aplicación destinada a los empleados de la obra. En él podíamos observar una pequeña pantalla de inicio de sesión y posteriormente el listado de obras asignado a dicho empleado.

Lo primero que se le mostraba al empleado en la aplicación era una pantalla de inicio de sesión donde debía introducir sus credenciales para acceder a la lista de obras que tenía asignada.

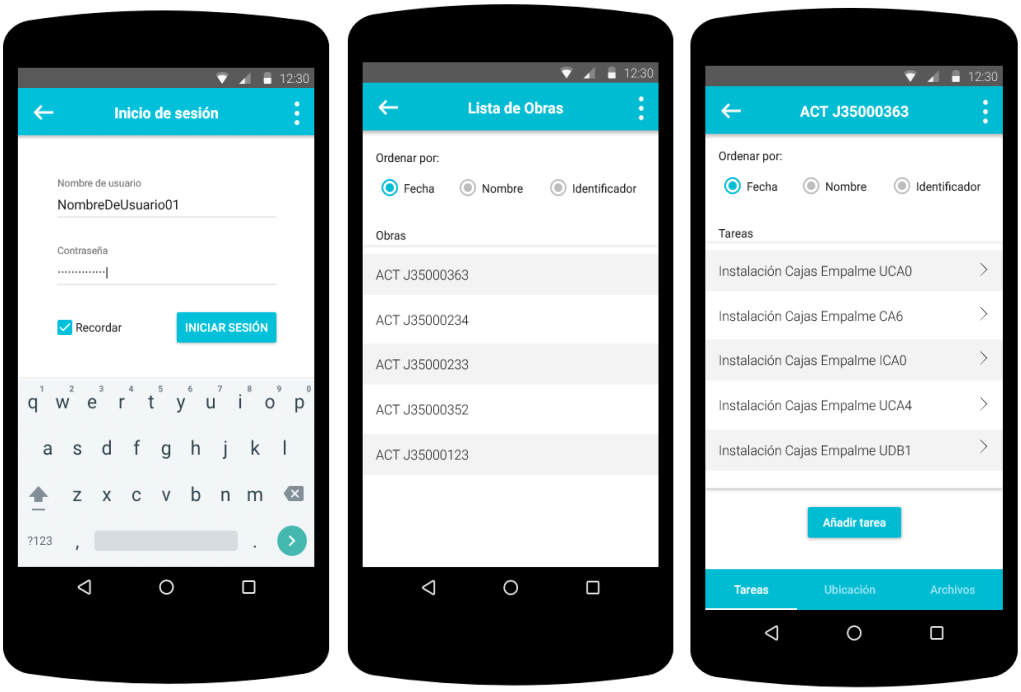

Ilustración 2. Pantallas principales del prototipo no funcional

Una vez autenticado se le mostraría una serie de obras que tendría asignadas, por parte del administrador, pudiendo así seleccionar la obra a la que deseara acceder.

Además, el usuario podía ordenar la aparición de las obras según sus campos, como la fecha, el nombre o el identificador de la misma.

Al seleccionar una obra el empleado tenía la posibilidad de ver toda la información relativa a la obra distribuida en la vista a través de pestañas (tareas, ubicación y archivos) totalmente navegables.

2.1.4. Segunda reunión, validación y acotación de requisitos

En la segunda reunión, ya conociendo el contexto del problema y habiendo trabajado en el prototipo no funcional pudimos concretar más los requisitos y perfilar nuestra visión del problema y una posible solución. También nos sirvió para validar los requisitos que surgieron durante la primera reunión.

En esta segunda reunión surgieron, además, nuevos requisitos:

- El encargado de la obra desea disponer de un panel de administración que refleje la actividad de cada empleado individualmente para así observar y evaluar su evolución dentro de la empresa.

- El portal de administración, encargado de la generación de las obras, debía poder auto importar las tareas a través de un *Excel* dado por la empresa
- El empleado puede añadir subtareas a una tarea creada por él.
- El empleado debe poder cambiar el estado de las tareas que vaya realizando.
- No todas las tareas tienen por qué tener subtareas.
- La existencia de un sistema de incidencias que permita la comunicación entre empleado y administrador de manera bidireccional.
- Sería interesante que la aplicación también funcionara en entornos sin acceso a internet.
- Se remarcó que en el diseño final se podría incluir, para tareas relacionadas con el tendido del cable, que la aplicación al marcar dos tramos de cable auto calculara la distancia real utilizada.
- El encargado de obras nos consultó si, para futuras versiones, sería posible que la aplicación auto dibujara sobre plano las conexiones a realizar en la obra y por dónde debía pasar el cable instalado.

2.1.5. Especificación final de requisitos y acotación del proyecto

En resumen y como conclusión a la etapa de recogida de requisitos, la solución al problema propuesto por el encargado de la empresa instaladora de fibra óptica pasa por el desarrollo de un portal de administración que permita:

- Generar obras a las que poder asignarles empleados, materiales, ficheros adjuntos necesarios para la realización de la misma y una lista de tareas que pueden contener subtareas. Así como observar el estado de realización de la misma.
- Importar tareas a una obra a través de un *Excel* dado.
- Importar material a una obra a través de un *Excel* dado.
- Dar de alta a nuevos empleados en la plataforma.
- Hacer el seguimiento del rendimiento de los empleados de manera individual.
- Una vez terminada la obra, y haciendo uso de la información aportada por los empleados a través de la aplicación, la herramienta deberá generar una certificación que permita a la empresa instaladora facturar.
- Gestión de incidencias relativas a la obra que permitan la comunicación bidireccional entre administrador y empleado.

Asimismo, se desea el desarrollo de una aplicación móvil para los empleados que debe darles la posibilidad de:

- Iniciar sesión en ella y tener acceso a las obras que tengan asignadas.
- Consultar el estado de realización de la misma mediante la lista de tareas pendientes.
- Cambiar el estado de una tarea perteneciente a la obra.
- Adjuntar información relativa a la realización de una tarea dentro de la obra que permita la certificación de la misma.
- Crear nuevas tareas.
- Gestión de incidencias relativas a la obra que permitan la comunicación bidireccional entre administrador y empleado.
- La aplicación debe ser multiplataforma.

Posteriormente a la segunda reunión y debido al gran número de requisitos ya especificados, decidimos acotar qué aspectos abarcaría nuestro proyecto. Para ello priorizamos en los requisitos y casos de uso más importantes, decidimos generar un diagrama de casos de uso, así como un documento asociado que nos permitiera visualizarlos de manera más sencilla y nos garantizara una visión global del proyecto.

2.1.6. Identificación de actores principales del sistema

Durante el proceso de análisis identificamos los siguientes actores que interactuarían con el sistema.

Actor	Descripción
Administrador	Usuario principal del sistema es capaz de crear obras y asignar empleados a ella, además puede certificar las obras. También puede dar de alta a empleados dentro del sistema a través del panel de administración.
Empleado	Usuario principal de la aplicación móvil del proyecto dónde podrá actualizar el estado de una obra mediante la realización de tareas que tenga asignadas.

Tabla 4. Actores principales del sistema

2.1.7. Especificación y diagramas de casos de uso

A continuación, mostramos la tabla de especificación de los casos de uso que, a través de la etapa de análisis creemos que deberían cumplir las aplicaciones del proyecto para resolver el problema en su totalidad.

Actor principal	Casos de uso
Administrador	1. Iniciar sesión en panel de administración
Administrador	2. Crear obra
Administrador	3. Gestionar obra 3.1. Gestionar tareas 3.1.1. Añadir tarea 3.1.2. Eliminar tarea 3.1.3. Editar tarea 3.1.3.1. Añadir subtarea 3.2. Importar tarea 3.3. Gestionar material 3.3.1. Añadir material 3.3.2. Eliminar material 3.4. Importar material 3.5. Gestionar adjuntos 3.5.1. Añadir adjunto 3.5.2. Eliminar adjunto 3.6. Cambiar estado (obra) 3.7. Certificar obra
Administrador	4. Registrar empleados
Administrador	5. Gestionar empleados 5.1. Editar empleado 5.2. Eliminar empleado 5.3. Notificar empleado 5.4. Visualizar empleado
Empleado	6. Iniciar sesión en la app
Empleado	7. Visualizar obra
Empleado	8. Gestionar archivos adjuntos a tarea 8.1. Adjuntar archivo adjunto a tarea 8.2. Eliminar archivo adjunto a tarea
Empleado	9. Añadir tarea a obra
Empleado	10. Cambiar estado (tarea)
Empleado	11. Notificar incidencia

Tabla 5. Tabla de especificación de los casos de uso

Como hemos dicho anteriormente, por cuestión de limitación en el número de horas del TFT, debimos acotar qué casos de uso no se iban a implementar durante el desarrollo del proyecto, estando estos subrayados en la tabla 2.1.3.

A modo de ilustración también hemos querido añadir el caso de uso del administrador, Gestionar empleados, aunque la especificación y diagrama completos puede encontrarse en el anexo I.

Nombre	Gestionar empleados	ID	5
Creado por	Álvaro Romero Perdomo	**Fecha**	22/10/2016
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	22/10/2016

Actor principal: ADMINISTRADOR
Personal involucrado o intereses: 1. ADMINISTRADOR: quiere poder editar/borrar empleados del sistema.
Descripción: El ADMINISTRADOR puede editar la información de un empleado ya existente, así como eliminar el mismo si fuera necesario.
Precondición: El ADMINISTRADOR debe estar autenticado y existir al menos un empleado a editar/borrar en el sistema.
Postcondición: 1. La información del empleado queda actualizada. 2. El empleado queda eliminado del sistema.
Flujo normal: 1. El ADMINISTRADOR accede al portal de administración. 2. El ADMINISTRADOR accede a la vista de los empleados. 3. El ADMINISTRADOR selecciona el empleado que desea gestionar. a. Editar empleado (5.1). b. Borrar empleado (5.2). c. Notificar empleado (5.3). d. Visualizar empleado (5.4).
Flujo alternativo:

Tabla 6. Tabla de especificación del caso de uso "Gestionar empleados"

2.2. Diseño

La etapa de diseño se describe como el proceso de aplicar distintas técnicas y principios con el propósito de definir un dispositivo, proceso o sistemas con los suficientes detalles como para permitir su desarrollo. El objetivo del diseñador es producir un modelo o representación de una entidad que será construida más adelante.

2.2.1. Componentes del sistema

Durante la fase de análisis identificamos dos grandes módulos software que comprenderían el proyecto, por un lado, el portal de administración y por otro lado una aplicación móvil. Estos dos módulos estarían alimentados de una única base de datos y ambos harían labores de lectura y escritura sobre la misma, por lo que decidimos que se hacía necesaria la creación de una *API REST* utilizando para ello el *stack MEAN*. De este modo también cubriríamos la necesidad de aprendizaje de nuevas tecnologías de desarrollo web que teníamos para este proyecto.

De esta manera definimos para el proyecto una arquitectura cliente-servidor.

Esta arquitectura se divide en dos partes claramente diferenciadas, la primera es la parte del servidor y la segunda la de un conjunto de clientes (siendo dos los clientes en este caso: portal de administración y aplicación para empleados).

Normalmente el servidor es una máquina bastante potente que actúa de depósito de datos y funciona como un sistema gestor de base de datos (SGBD).

Por otro lado, los clientes suelen ser estaciones de trabajo que solicitan varios servicios al servidor.

Ambas partes deben estar conectadas entre sí mediante una red.

En la ilustración 3 se muestra una representación gráfica de este tipo de arquitectura:

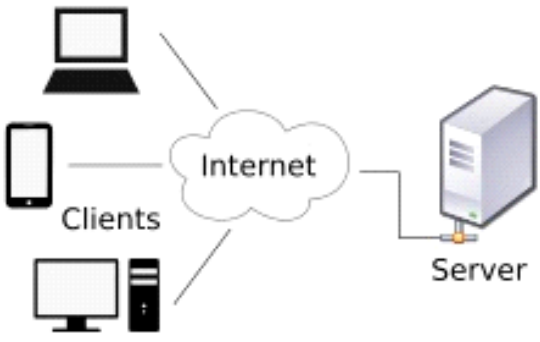

Ilustración 3. Representación gráfica de una arquitectura cliente-servidor

El cliente realiza peticiones a un servidor y éste le responde ofreciéndole un determinado servicio.

2.2.2. Utilización del patrón de diseño Modelo – Vista – Controlador (MVC)

Desde el principio de la etapa de diseño decidimos que íbamos a construir tanto el portal de administración como la aplicación destinada a los empleados. Para ello, se haría uso del stack *MEAN*, por lo que para el lado del cliente construiríamos las aplicaciones en *AngularJS* implementando éste el patrón de diseño Model View Controller (MVC) en el desarrollo de aplicaciones web.

En el siguiente diagrama podemos observar la interacción entre los diferentes componentes.

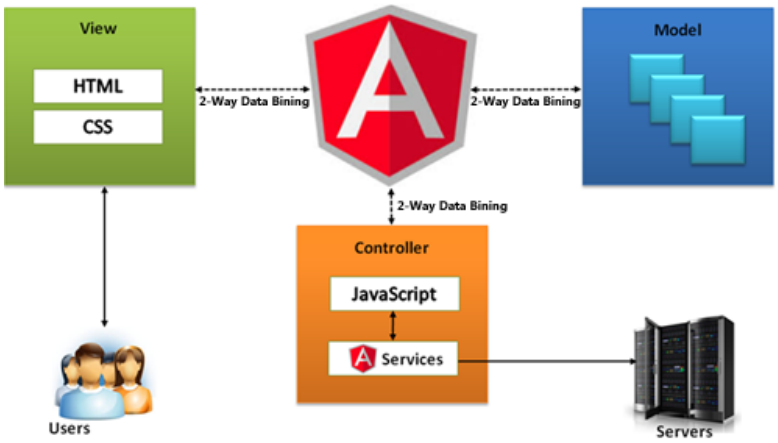

Ilustración 4. Interacción de componentes con AngularJS

2.2.3. Desarrollo híbrido con *Ionic*

Para el desarrollo de la aplicación móvil decidimos optar por un desarrollo híbrido porque, aparte de permitirnos cumplir los requisitos del proyecto (compatibilidad en varios dispositivos), cubríamos así la necesidad de aprendizaje del desarrollo de aplicaciones móviles.

Además, consideramos que el aprendizaje de las tecnologías web sería mucho más rápido que el aprendizaje de tecnologías necesarias para el desarrollo nativo de las aplicaciones, por lo que éste fue un motivo más que nos hizo decidirnos por un desarrollo híbrido para la aplicación móvil.

2.2.4. Estructura y entidades de la base de datos

Las bases de datos en *MongoDB* están formadas por esquemas en lugar de por tablas como en una base de datos relacional común como puede ser *MySQL.* Siguiendo esta nomenclatura y a través del siguiente diagrama, nuestra base de datos está formada por los siguientes esquemas.

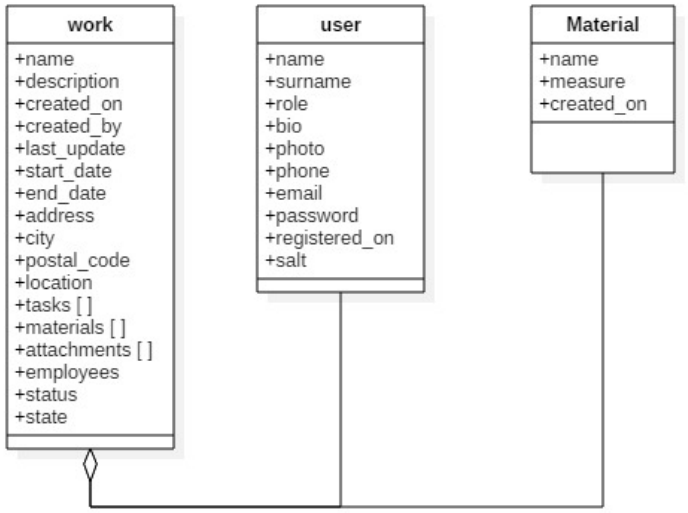

Ilustración 5. Diagrama de entidades de la base de datos

Aunque en un principio no nos lo habíamos planteado, decidimos incluir en la base de datos el esquema *Equipment*, de manera que el sistema pudiera almacenar el material que se iba añadiendo a las diferentes obras y éste podría ser reutilizado, ya que se repetía con bastante frecuencia entre las diferentes obras.

Debemos añadir que el driver *Mongoose* para Node.js facilitó en gran medida la labor de conexión con la base de datos, permitiéndonos trabajar en un entorno mucho más cercano a la orientación de objetos y haciendo que la traducción en la base de datos fuera instantánea.

2.2.5. Arquitectura del portal de administración en *AngularJS*

Como hemos comentado anteriormente, decidimos desarrollar el proyecto bajo *MEAN Stack* por lo que implementamos el panel de administración como una aplicación en *AngularJS* que se conectaba a una API construida en *Node.js*.

Antes de comenzar a trabajar con Angular, decidimos cómo íbamos a organizar la aplicación y qué responsabilidades iba a tener cada fichero. Para ello, consultamos las formas más comunes en las que se suelen dividir las aplicaciones en *AngularJS* y escogimos el *Patrón Específico*, que se puede observar a continuación y que se caracteriza por separar los servicios y los controladores de un mismo elemento en distintas carpetas, consiguiendo así una separación a su vez de las responsabilidades de cada fichero.

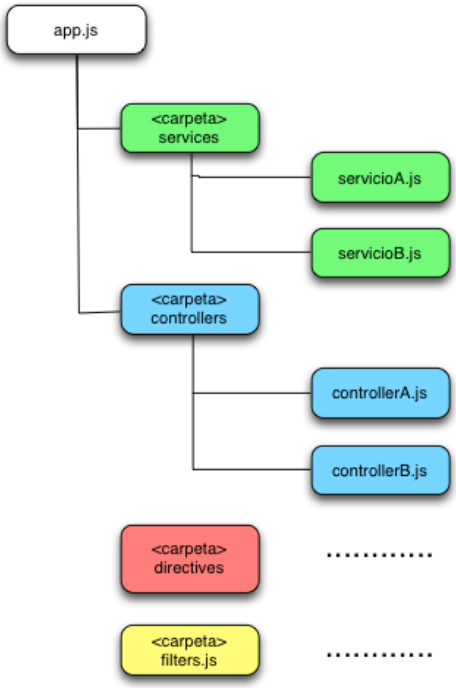

Ilustración 6. Jerarquía de ficheros de una aplicación en AngularJS

2.3. Implementación

Es el proceso de instalar equipos o software nuevo, como resultado de un análisis y diseño previo como resultado de la situación o mejoramiento de la forma de llevar a cabo un proceso automatizado. Al implementar un sistema lo primero que debemos hacer es asegurarnos qué el sistema sea operacional o que funcione de acuerdo a los requerimientos del análisis y permitir que los usuarios puedan operarlos.

Antes de comenzar la etapa de implementación decidimos dividirla en tres grandes módulos:

1. Desarrollo de *API REST* en *MongoDB* y *Node.js* que permita el almacenamiento de los datos y el intercambio de información entre las diferentes aplicaciones.
2. Desarrollo de un panel de administración que permita la generación de obras, tareas, empleados y material asociado.
3. Desarrollo de una aplicación móvil que permita a los empleados la consulta de la obra que tengan asignada, así como el envío de información relativa a las tareas.

En este módulo vamos a tratar, sobre todo, el desarrollo de la *API REST* y el panel de administración. Habiendo sido el primero de los dos desarrollado de manera conjunta y el segundo de manera individual.

En la primera etapa del desarrollo comenzamos con la configuración del servidor de *Node.js* para la *API REST* y la creación de la base de datos. Veremos cómo generamos las entidades y sus campos en *MongoDB* a través de su driver para *Node.js*, *Mongoose*. También mostraremos la respuesta de nuestra *API REST* siendo sometida a interrogaciones HTTP.

En la segunda etapa del desarrollo y ya con la *API REST* operativa, pasamos a la implementación de la aplicación móvil, dividida, fundamentalmente, en tres partes:

1. Sistema de control de usuarios (*login* y *logout*).
2. Visualización del contenido de la base de datos (listado de obras asignadas, directorio de empleados, detalle de la obra).
3. Gestión del contenido en la base de datos (edición de obras asignadas).

2.3.1. Desarrollo de la *API REST* en *Node.js* y *MongoDB*

La primera etapa de la implementación del proyecto consistió en el desarrollo de una *API REST* que ofreciera un servicio web a ser consumido tanto por un panel de administración como por una aplicación móvil. A continuación, explicaremos en qué consistió dicho proceso de desarrollo.

Entorno de desarrollo con Docker

Como solución a la necesidad de que cada módulo del proyecto fuera independiente decidimos hacer uso de la tecnología de contenedores que ofrece Docker, la cual facilita al desarrollador trabajar en el entorno necesario independientemente del sistema operativo y evitando la configuración manual de servidores, bases de datos, etc.

Para ello se generó un archivo de configuración de Docker *(docker-compose.yml)* con el fin de configurar cada uno de los servicios necesarios:

- Un servidor Node.js para la API REST
- Servidor Apache para el panel de administración
- Servidor para la base de datos de MongoDB

A continuación, se muestra una captura del archivo de configuración que compone cada uno de los entornos individuales definidos:

docker-compose.yml

```
version: '2'

services:

    db:
      image: mongo
      ports:
        - "27017:27017"
      command: "--smallfiles --logpath=/dev/null"
      container_name: appartes_mongo_db

    server:
      build:
        context: ./server
      container_name: appartes_nodejs_server
      command: nodemon server/server.js --ignore 'client/' --ignore 'mobile/'
      env_file:
        - ./vars.env
      volumes:
        - .:/home/app
      ports:
        - "3000:3000"
      links:
        - db
      depends_on:
        - db
      ...
```

Ilustración 7. Captua del fichero de configuración de Docker

Primeros pasos del desarrollo de la *API REST* (instalación de dependencias y configuración del servidor *Node.js*)

Como todas estas tecnologías eran nuevas para nosotros, decidimos empezar desde abajo hacia arriba, por lo que una vez generado el directorio del proyecto y habiendo iniciado el repositorio Git, nos dispusimos a instalar Node.js. Ya finalizada la instalación de Node.js en nuestro ordenador procedimos a generar el fichero *package.json*, que recoge la meta-información relacionada con el proyecto, así como las dependencias que irán siendo necesarias a lo largo del desarrollo. A continuación podemos ver la versión inicial del fichero *package.json*.

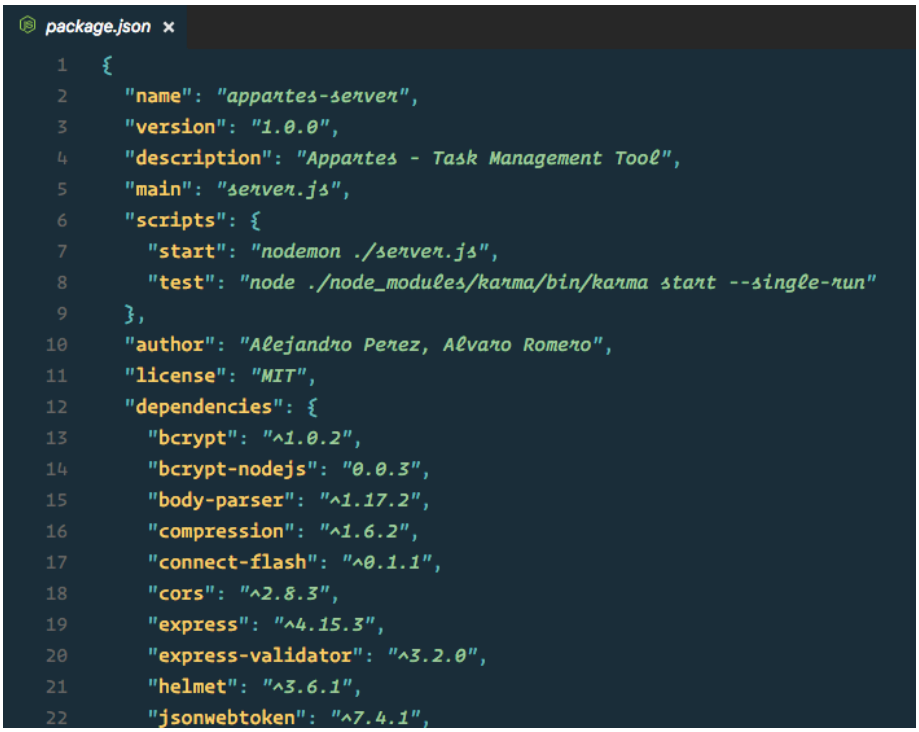

Ilustración 8. Versión inicial del fichero package.json

Una vez especificadas las dependencias a través del fichero *package.json* procedimos a ejecutar el gestor de paquetes de *Node.js, npm (node package manager).*

Ya instaladas las dependencias de la versión inicial de nuestro proyecto podíamos empezar a configurar el servidor *Node.js* a través del fichero *server.js* importando las dependencias necesarias para el arranque del mismo, definiendo las rutas de acceso HTTP que va a tener nuestro servidor e indicándole un puerto por el que escuchará la aplicación, en nuestro caso el tres mil.

Tras lanzar el servidor de *Node.js* podríamos observar el siguiente mensaje haciendo una petición *GET* al servidor a través del puerto *3000*.

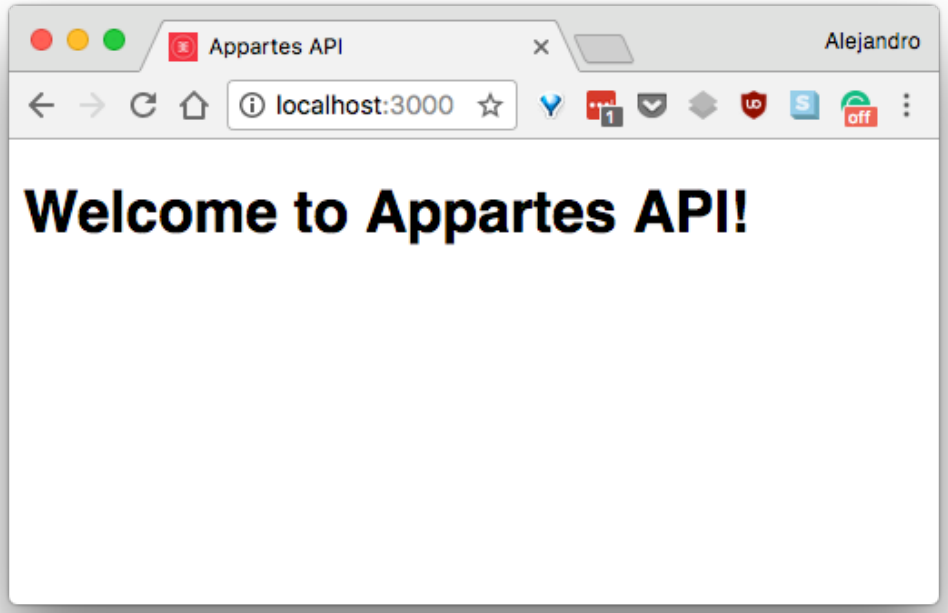

Ilustración 9. Petición GET al servidor en el puerto 3000

Creando los modelos de nuestra *API REST*

A la hora de realizar nuestro modelado de datos para la *API REST* hemos utilizando el driver de *MongoDB* para *Node.js, Mongoose.* Hemos decidido usarlo porque aparte de estructurar de mejor manera el acceso a la base de datos por parte del servidor, también nos permite una mejora en el código y una mayor modularización del mismo.

A continuación, podemos observar el fichero resumido del esquema de Mongoose propuesto para el modelado de las obras.

work.js

```
const mongoose = require('mongoose'),
    Schema = mongoose.Schema;

let workSchema = new Schema({
    name: { type: String, required: true },
    description: { type: String },
    created_on: { type: Date, default: Date.now },
    created_by: {
        type: Schema.Types.ObjectId,
        ref: 'User'
    },
    materials: [{
        amount: { type: Number, required: true },
        material: {
            type: Schema.Types.ObjectId,
            ref: 'Material',
            required: true
        }
    }],
    // ...
});

module.exports = mongoose.model('Work', workSchema);
```

Ilustración 10. Esquema resumido del modelo de obras de Mongoose

Estos ficheros asociados al modelo de la aplicación decidimos introducirlos en un nuevo directorio para el proyecto, *models*. Por lo que pronto nos dimos cuenta de que debíamos optar por un desarrollo orientado al patrón de diseño MVC donde las responsabilidades estuvieran claramente diferenciadas.

Utilización del patrón de diseño Modelo – Vista – Controlador (MVC)

Durante el proceso de aprendizaje inicial todas las responsabilidades de la API REST (acceso a la base de datos, gestionar peticiones HTTP, configuración del servidor Node.js, etc.) se encontraban en un único fichero, haciéndonos más engorrosa la labor de mantenimiento del código y modularidad del mismo por lo que durante la implementación final de la *API REST* decidimos seguir el patrón de diseño MVC como se puede observar en la siguiente jerarquía de ficheros.

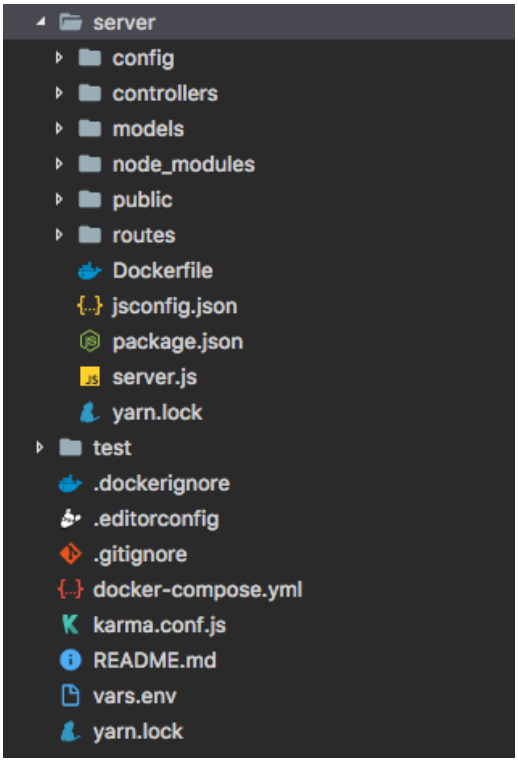

Ilustración 11. Jerarquía de ficheros de la API REST siguiendo patrón MVC

Sistema de autenticación por Token (JWT)

El funcionamiento es el siguiente: el usuario se autentica en la aplicación con un par usuario/contraseña. A partir de entonces, cada petición HTTP que haga el usuario va acompañada de un Token en la cabecera. Este Token no es más que una firma cifrada que permite a nuestro API identificar al usuario. Pero este Token no se almacena en el servidor, sino en el lado del cliente (en nuestro caso *LocalStorage*) y la API es el que se encarga de descifrar ese Token y redirigir el flujo de la aplicación en un sentido u otro.

Como los tokens son almacenados en el lado del cliente, no hay información de estado y la aplicación se vuelve totalmente escalable. Podemos usar la misma API para diferentes aplicaciones (web, mobile, Android, iOS...), tan sólo debemos preocuparnos de enviar los datos en formato JSON y generar y descifrar tokens en la autenticación y posteriores peticiones HTTP.

En la firma del token por parte del servidor, se incluye además información relativa al usuario, como por ejemplo el rol que tiene dentro de la aplicación, lo cual se emplea para restringir el acceso a diversos recursos de la API, evitando así, por ejemplo, que un usuario pueda eliminar obras o usuarios sin ser administrador.

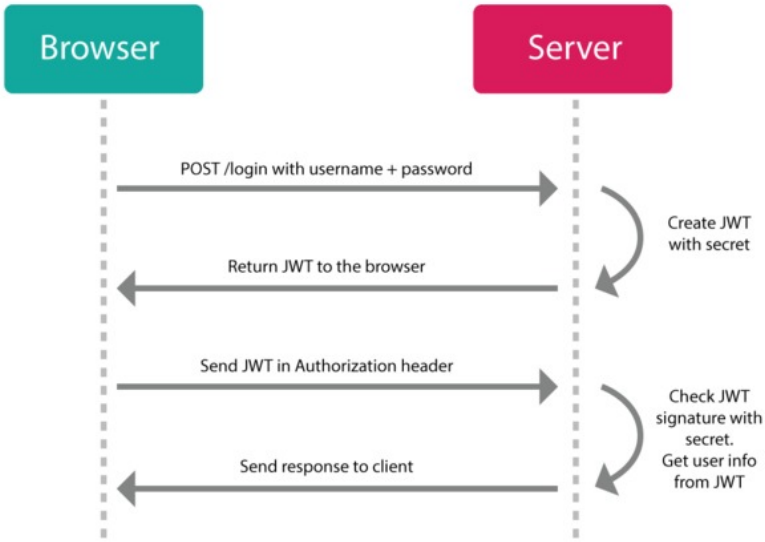

Ilustración 12. Diagrama de peticiones HTTP con token (JWT)

Pruebas y peticiones a la *API REST*

Posterior al proceso de desarrollo e implementación de la *API,* la sometimos a diferentes peticiones para comprobar su funcionamiento mediante el software *Postman,* el cual nos permitía generar peticiones GET, PUT, DELETE o UPDATE, entre otras, contra la *API* de manera que podíamos ver cuál era la respuesta de ésta frente a esas peticiones. En el anexo número cinco se detalla completamente la estructura de peticiones que acepta la *API.*

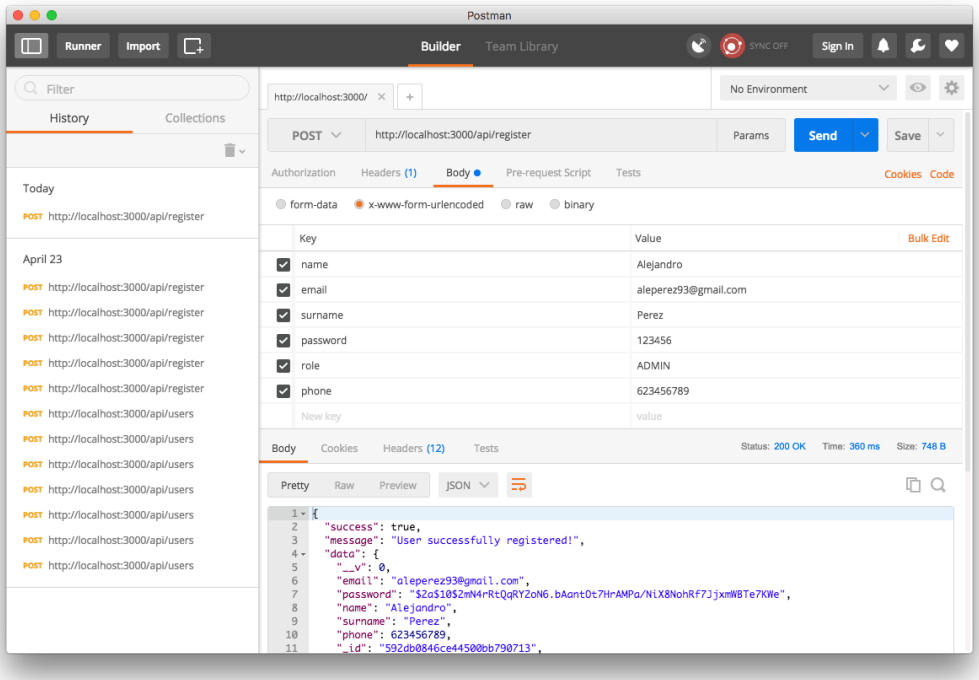

Ilustración 13. Ejemplo de petición HTTP utilizando Postman

2.3.2. Desarrollo de la aplicación móvil

Una vez creados los servicios de la API necesarios comenzamos con el desarrollo de la aplicación móvil de los trabajadores, que les permitirá acceder a la información de las obras que, previamente creadas a través del panel de administración, tengan asignadas, así como modificar su información en pro de cumplimentar la mayor cantidad de datos posible. Como hemos dicho anteriormente este trabajo se centra en específico en el desarrollo tanto de la *API REST*, ya descrito, como en el desarrollo de la aplicación móvil.

Estructura inicial y vista principal, el *Dashboard*

Tras una segunda reunión y gracias a la demo no funcional propuesta, identificamos que, para el usuario final de la aplicación, la información más relevante al acceder a la aplicación debía ser la relativa a las obras que el propio empleado tuviera asignadas, por lo que se estableció esta vista como la principal de la aplicación.

Para su desarrollo y siguiendo la estructura proporcionada por el cliente de terminal de *Ionic,* se comenzó generando las distintas páginas/pantallas que compondrían la aplicación. Siguiendo las directrices de la Ingeniería del Software, concretamente del desarrollo basado en componentes, y el código limpio, se crearon diversos directorios donde irán situados cada uno de los ficheros correspondientes a cada página. A continuación, mostramos la jerarquía de ficheros final resultante:

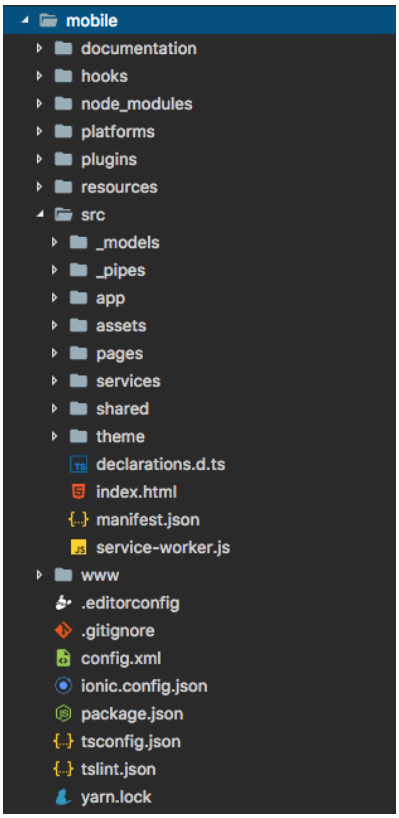

Ilustración 14. Jerarquía de ficheros de la aplicación móvil

Dentro de la carpeta src/pages/*dashboard* y al igual que en el resto de carpetas de las páginas, encontraremos un archivo TypeScript (*.ts)* que contendrá toda la lógica de página, así como sus estilos en Sass (archivo *.scss*) y la plantilla del componente en HTML (archivo *.html*).

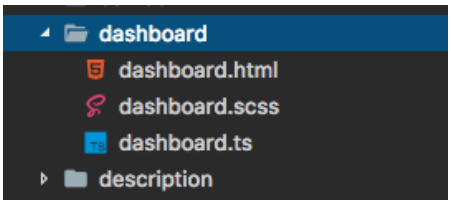

Ilustración 15. Jerarquía de ficheros de la página principal de la aplicación

Modelos de datos

Aprovechando una de las características de *TypeScript* como es el tipado de datos, se han creado distintos modelos de datos utilizados en distintos puntos de la aplicación con el fin de detectar errores de código en tiempo de compilación y así evitarlos durante la ejecución de la aplicación. A continuación, se muestra un ejemplo de uno de los modelos de datos definidos.

work.model.ts

```typescript
import { Attachment, Material, Task, User } from './index';

/**
 * Work model interface
 *
 * @export
 * @interface Work
 */
export interface Work {
    _id: string;
    name: string;
    description?: string;
    created_on: Date;
    created_by?: User;
    tasks?: Task[];
    materials: [{
        amount: number,
        material: Material
    }];
    attachments?: Attachment[];
    employees?: User[];
    status?: string;
    // ...
}
```

Ilustración 16. Modelo de datos del tipo 'Obra'

Servicios y conexión a la API

Los servicios utilizados por los distintos componentes de la aplicación albergan la responsabilidad de conexión a la API y por lo tanto el acceso a la base de datos. Estos servicios son inyectados haciendo uso del patrón de inyección de dependencias (DI) a cada uno de los

componentes. A continuación, podemos observar un fragmento del fichero del servicio encargado de la conexión con la información relativa a las obras en la *API, work.service.ts*.

work.service.ts

```
import { Injectable } from '@angular/core';

/**
 * Work service
 *
 * @export
 * @class WorkService
 */
@Injectable()
export class WorkService {

    /**
     * Creates an instance of WorkService
     * @param {HttpClient} httpClient
     * @param {Transfer} transfer
     *
     * @memberof WorkService
     */
    constructor(
        private httpClient: HttpClient,
        private transfer: Transfer
    ) { }

    /**
     * Returns the selected work information
     *
     * @param {string} workId Work ID
     *
     * @memberof WorkService
     */
    getWork(workId: string) {
        return this.httpClient.get(`works/${workId}`);
    }
```

Ilustración 17. Fichero work.service.ts

Como podemos observar el fichero guarda gran similitud con la interfaz descrita por la *API.*

Una vez creados los métodos de acceso a los servicios, sólo tendríamos que instanciarlos y suscribirnos a los mismos para obtener las respuestas enviadas por la API.

En nuestro caso, hacemos uso del servicio de las obras desde la página principal de la aplicación para mostrar el listado completo de obras asignadas al empleado.

A modo de ejemplo mostramos la instanciación y suscripción al servicio a través del método *fetchWorks().*

```
93        /**
94         * Event fired on init
95         *
96         * @memberof DashboardPage
97         */
98        ngOnInit() {
99            this.fetchWorks();
100       }
101
102       /**
103        * Returns the list of works
104        *
105        * @param {Refresher} [refresher] Refresher object passed as $event
106        *
107        * @memberof DashboardPage
108        */
109       fetchWorks(refresher?) {
110           return this.workService.getWorks(this.authService.user._id)
111               .subscribe(res => {
112                   this.works$ = res;
113                   this.worksOriginal = res;
114                   if (refresher) refresher.complete();
115               },
116               err => {
117                   console.log(err);
118               });
119       }
```

Ilustración 18. Fichero dashboard.ts

Proceso de *Login* y autenticación en el panel de administración

Una vez implementada la página principal, la tarea más relevante a acometer consistiría en desarrollar el control de acceso a la misma.

Para ello generamos una vista de nombre *login*, que se mostraría antes que la pantalla principal y sería la encargada de obtener los datos necesarios para autenticar al usuario en el sistema y, por ende, tener acceso a los servicios de la API.

Notificaciones Push

Una funcionalidad bastante útil es la incorporación de notificaciones Push, que no son más que mensajes directos a los dispositivos móviles que tengan instalada la aplicación. Para ello, se ha hecho uso del servicio de notificaciones *Firebase Cloud Messaging* de *Google*.

Algunos de los usos de este servicio podrían ser por ejemplo la notificación de cambios en una obra o enviar un aviso urgente a todos los empleados. En las siguientes capturas puede verse un ejemplo de este tipo de notificaciones:

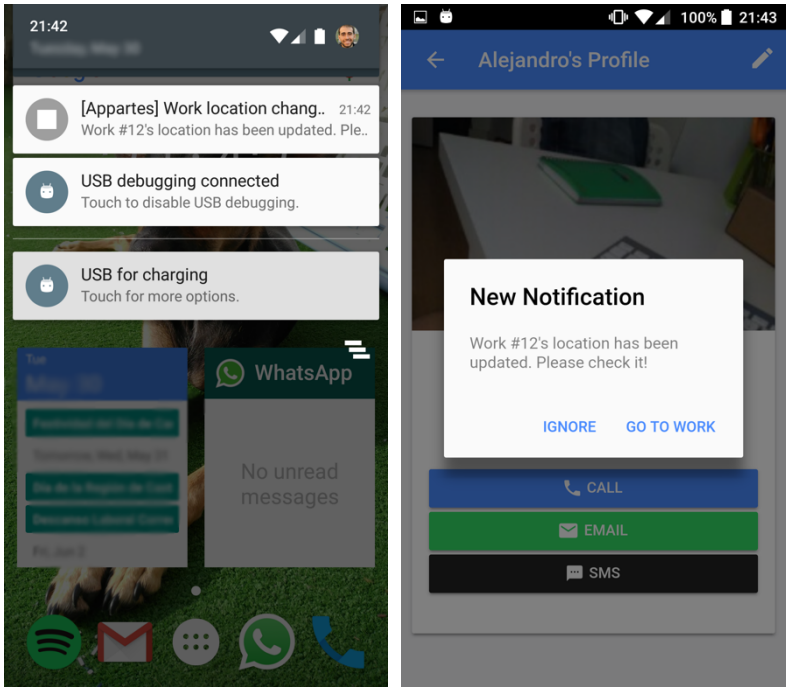

Ilustración 19. Ejemplo de recepción de notificaciones push

Documentación del código

Con el objetivo de mejorar la documentación del código y con la ventaja de usar Github como servicio de hospedaje de nuestro código, hemos hecho uso de Codacy, que ofrece análisis de código, documentación y estilo del proyecto. Los resultados obtenidos se muestran a continuación.

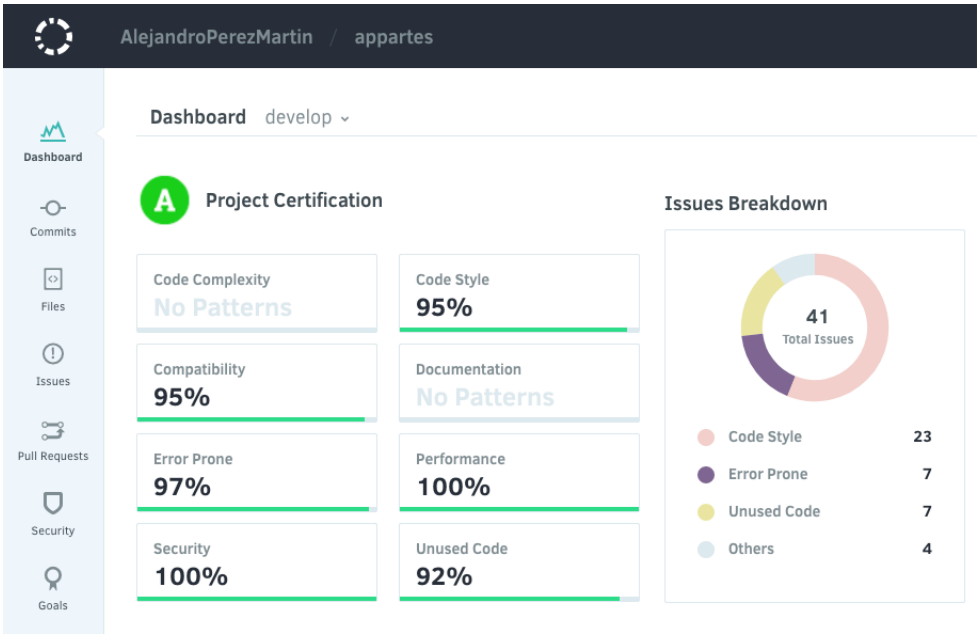

Ilustración 20. Captura del resumen de análisis de código hecho con Codacy

Además de Codacy, se ha incorporado documentación de la aplicación móvil del proyecto con la herramienta Compodoc, que genera de forma automática un resumen de los distintos componentes de la aplicación haciendo uso de los comentarios incluidos en el código durante el desarrollo. En nuestro caso se ha obtenido un porcentaje del 100% en cuanto al alcance de la documentación del código, el cual se ha realizado siguiendo las directrices del estándar *JSDoc*.

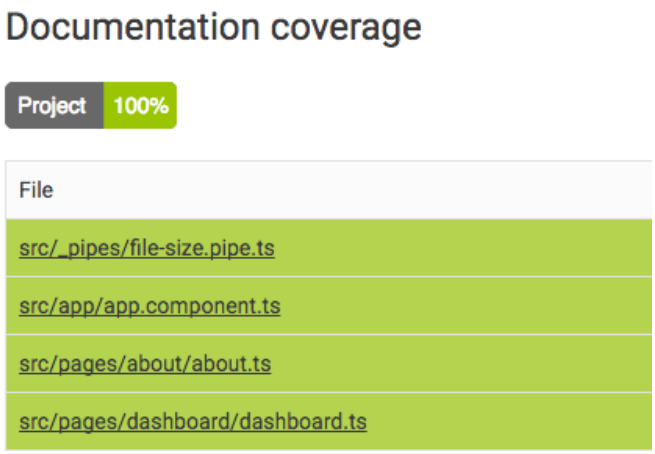

Ilustración 21. Captura del resumen del alcance de los comentarios hecho con Compodoc

2.4. Pruebas

En todo proceso de desarrollo de software es necesario una etapa consistente en el testeo o verificación del correcto funcionamiento de la aplicación que se esté desarrollando. En nuestro caso decidimos testear funcionamiento de la API en el acceso a la base de datos y operaciones sobre la misma (creación, actualización y eliminación), para ello utilizamos una librería llamada Mocha que nos facilita la ejecución de tests unitarios asíncronos.

Antes de poder utilizar la librería *Mocha* hizo falta la instalación del paquete *should*, que permite la utilización de directivas de tipo *assert* para la verificación de los datos obtenidos de la consulta a la base de datos.

A modo de ejemplo hemos decidido adjuntar el fichero encargado de comprobar el correcto funcionamiento de las operaciones de inserción, actualización y eliminación de usuarios dentro de la base de datos.

```
9   describe('#create()', function () {
10      it('should create a new User', function (done) {
11
12          let user = {
13              name: 'Alejandro',
14              surname: 'Perez',
15              role: 'ADMIN',
16              phone: 612345678,
17              email: 'aleta@gmail.com',
18              password: '1234567890'
19          };
20
21          User.create(user, function (err, createdUser) {
22              should.not.exist(err);
23              createdUser.name.should.equal('Alejandro');
24              createdUser.surname.should.equal('Perez');
25              done();
26          });
27      });
28  });
```

Ilustración 22. Ejemplo del test de creación de un usuario

Debido a que a la hora de ejecutar los tests se producen operaciones sobre la base de datos, decidimos hacer éstas sobre una base de datos destinada a los tests. De esta manera nos aseguramos de que no se pierden o modifican datos en el proceso de comprobación.

A través del terminal, ejecutamos los tests relativos a las operaciones con usuarios obteniendo la siguiente respuesta en consola.

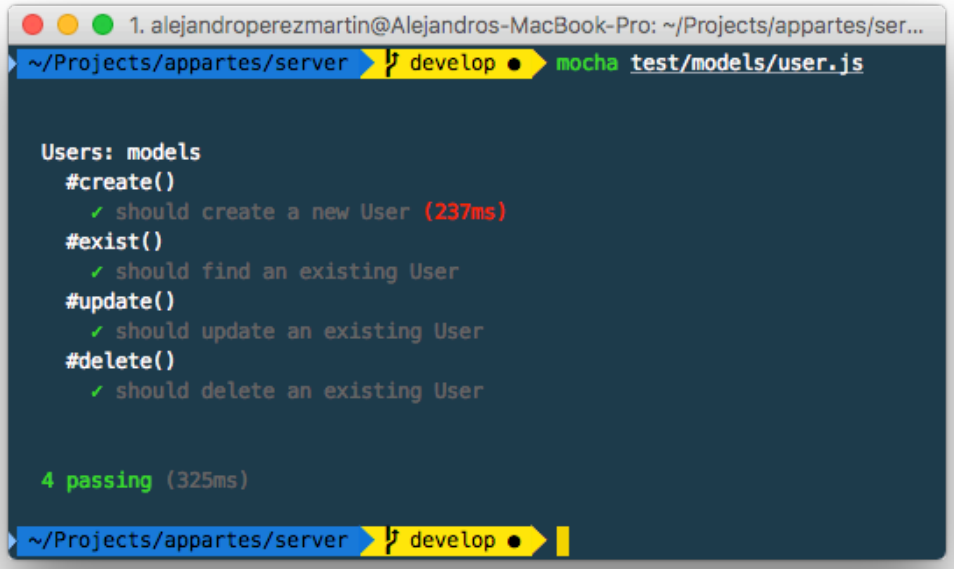

Ilustración 23. Captura de los resultados de las pruebas unitarias

3. Bloque 3. Conclusiones y Trabajos Futuros

3.1. Conclusiones y trabajos futuros

Como colofón, en este bloque comentaremos, qué ha supuesto para nosotros el desarrollo del proyecto, si hemos cumplido nuestras expectativas formativas, y qué conclusiones extraemos del trabajo en equipo derivado de la compartición del proyecto entre dos alumnos.

Finalmente, indicaremos qué funcionalidades han quedado pendientes de desarrollar a causa de la limitación temporal y una batería de propuestas de mejora a tener en cuenta en futuras iteraciones del desarrollo.

3.1.1. Conclusiones sobre el desarrollo del proyecto

En líneas generales, puedo decir que estoy satisfecho con el trabajo desarrollado. Inicialmente no contaba con experiencia previa en desarrollos similares, aunque sí partía con nociones básicas en algunas de las tecnologías empleadas, por lo que ha supuesto un gran reto el haber llegado a hasta este punto de consecución.

Además, me ha ayudado a reforzar los conocimientos adquiridos durante la carrera ya que he podido participar en todas las etapas del desarrollo de un producto software, iniciándome con el análisis, captación de requisitos, diseño e implementación (tanto back-end como front-end), hasta terminar con la documentación. Todo esto ha resultado más sencillo e interesante al contar con un problema real de un cliente con el que hemos tenido que interactuar durante las diversas etapas del desarrollo, lo cual ha proporcionado un primer acercamiento a lo que podría ser un futuro entorno laboral.

A todo esto hay que añadir el aprendizaje de nuevas tecnologías que a día de hoy tienen gran peso en el marco del desarrollo software, como por ejemplo el framework *Angular*, del cuál he podido ver su evolución ya que el desarrollo de la aplicación se inició con su versión inicial, *AngularJS/Angular 1*, y luego fue migrado a su versión más reciente, Angular 4. Esta migración, aunque exitosa, supuso un gran esfuerzo de implementación y aprendizaje debido a las más que notables diferencias entre ambas versiones.

En definitiva, estoy orgulloso del resultado final del trabajo y más aún de las nuevas habilidades que me ha otorgado el desarrollo del mismo.

Conclusiones derivadas de la compartición del trabajo fin de título

Respecto al trabajo en equipo, ha sido de gran ayuda contar con el apoyo de mi compañero, que durante las distintas fases del desarrollo ha sido resolutivo con las diferentes dudas que han surgido, además de aportar otro punto de vista ante la resolución de problemas a los que nos hemos enfrentado.

En cuanto a lo más interesante de esta experiencia de trabajo en equipo, a mi parecer, ha sido el manejo del control de versiones Git con *GitHub*, con el que no había tenido experiencia en trabajos colaborativos y del que hemos aprovechado la mayoría de funcionalidades ofrecidas de cara a la gestión de tareas, como es el caso de las '*Issues'*, que nos han sido de gran ayuda a la hora de gestionar las tareas a desarrollar, documentar errores y nuevas funcionalidades, así como llevar un control de la evolución de las mismas.

Además de esto y debido a la diferencia de los sistemas operativos que empleamos para el desarrollo, hemos hecho uso de una de las tecnologías que más suenan hoy en día como es Docker, lo cual nos ha permitido contar con un entorno de trabajo homogéneo e independiente del sistema utilizado.

Para finalizar, quiero agradecer tanto a mi compañero como a los tutores la ayuda prestada durante toda las fases de realización del proyecto, que sin ella, habría sido mucho más difícil haber podido finalizarlo.

Resultados obtenidos

En este apartado haremos un breve recorrido por los objetivos propuestos para este proyecto y veremos si estos se han visto cubiertos de manera satisfactoria.

Como citamos en el apartado de objetivos del proyecto, la comunicación entre los empleados de las obras y el administrador de las mismas se realizaba de manera poco ortodoxa, resultando, en muchos casos en errores en la transmisión de la información o en la pérdida de la misma. Con la creación de la aplicación para los empleados de las obras, este objetivo se ha visto cubierto ya que permite al empleado guardar información de las tareas que vaya realizando en una obra de manera inmediata y eficaz.

Además, mediante la creación de un panel de administración, el encargado de la obra puede visualizar en todo momento el estado actual de la obra y asignar de manera efectiva empleados a una determinada obra.

Todo esto ha supuesto un avance para la empresa encargada de la instalación de la fibra óptica, tanto en la manera de comunicarse internamente con sus empleados como en la capacidad de certificación de las obras una vez éstas han finalizado, no dando lugar a pérdidas de información que perjudicaran la certificación de la obra y, por lo tanto, su facturación por parte de la empresa instaladora.

3.1.2. Trabajos futuros

A pesar de que creemos que los requisitos principales para resolver el problema propuesto han quedado satisfechos a través de la realización del proyecto, es cierto que, con motivo de la limitación temporal que supone el trabajo fin de grado, han quedado algunos flecos sueltos a realizar en futuras iteraciones del proyecto. A continuación, hacemos referencia a ellos:

- Automatización del proceso de certificación a través del panel de administración.
- Importación de tareas a través de un *Excel* dado.
- Importación de material a través de un *Excel* dado.
- Auto acotación y trazado de planos dados.
- Chat o comunicación directa entre empleado y administrador a través de ambas aplicaciones.
- Cálculo de estadísticas relacionadas con el rendimiento del empleado.
- Sistema de almacenamiento temporal en la aplicación móvil para trabajar sin conexión y posterior sincronización de los datos.
- Mejora del diseño de algunas vistas.
- Publicar el servicio en un entorno web accesible externamente.

Asimismo, queríamos expresar una serie de propuestas de mejora que se podrían incluir también en sucesivas iteraciones del proyecto, pero que son accesorias a la resolución del problema principal propuesto para éste:

- Abstracción que permita la vinculación de la aplicación a varias empresas instaladoras simultáneamente.
- Actualizar el portal de administración a las versiones recientes de *Angular*.
- Implementación de estadísticas relacionadas con el volumen de datos manejado por la aplicación.
- Envío de emails de notificación a través del panel de administración a otras compañías para notificar incidencias.
- Implementación de estadísticas relacionadas al flujo económico derivado de la certificación de las obras.

4. Bloque 4. Fuentes de Información

4.1. Fuentes de información del Bloque 1. Introducción y Contextualización

- https://www.adslzone.net/2016/04/01/asi-queda-la-cobertura-despliegue-actual-la-fibra-optica-espana/
- https://www.genbeta.com/actualidad/asi-si-microsoft-visual-studio-code-gratuito-y-tambien-disponible-en-linux-y-os-x
- https://es.wikipedia.org/wiki/JavaScript
- http://edumatica.ing.ula.ve/edumatica/Teleclases/Tecniweb/Ingenieria%20Web/Teleclase/Ejecucion/Practicas/JavaScript/Paginas/CaracteristicasGenerales.htm
- https://es.wikipedia.org/wiki/Node.js
- https://es.wikipedia.org/wiki/AngularJS
- http://www.intelygenz.es/introduccion-a-reactive-programming-con-rxjs-por-gustavo-marin/
- https://es.wikipedia.org/wiki/Biblioteca_(inform%C3%A1tica)
- https://es.wikipedia.org/wiki/MongoDB
- https://en.wikipedia.org/wiki/Express.js
- http://expressjs.com/es/
- http://www.mongodbspain.com/es/2014/08/17/mongodb-characteristics-future/
- https://es.wikipedia.org/wiki/JSON
- https://es.wikipedia.org/wiki/Git
- http://aprendegit.com/que-es-git-flow/
- https://es.wikipedia.org/wiki/HTML
- https://es.wikipedia.org/wiki/Hoja_de_estilos_en_cascada
- https://es.wikipedia.org/wiki/GitHub
- https://openwebinars.net/blog/docker-que-es-sus-principales-caracteristicas/
- https://es.wikipedia.org/wiki/Ley_Org%C3%A1nica_de_Protecci%C3%B3n_de_Datos_de_Car%C3%A1cter_Personal_(Espa%C3%B1a)
- https://carlosazaustre.es/blog/que-es-la-autenticacion-con-token/

4.2. Fuentes de información del Bloque 2. Desarrollo del proyecto

- http://vilmarygalindez.blogspot.com.es/2011/02/fases-del-diseno.html
- https://silkeguabylopez20.wordpress.com/2013/05/01/desarrollo-de-prototipos/
- http://equipo.altran.es/desarrollo-aplicaciones-hibridas-moviles-ionic-framework/
- https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador
- https://carlosazaustre.es/blog/como-crear-una-api-rest-usando-node-js/
- https://desarrolloweb.com/articulos/uso-bower-gestor-dependencias.html
- http://blog.ionic.io/10-minutes-with-ionic-2-using-the-camera-with-ionic-native/
- https://ionicframework.com/docs/

5. Bloque 5. Anexos

5.1. Anexo 1. Casos de uso

A continuación, presentamos el diagrama completo de casos de uso del proyecto donde se muestran todas las interacciones que pueden realizar los diferentes actores con las aplicaciones, así como el documento de especificación de todos los casos de uso del proyecto.

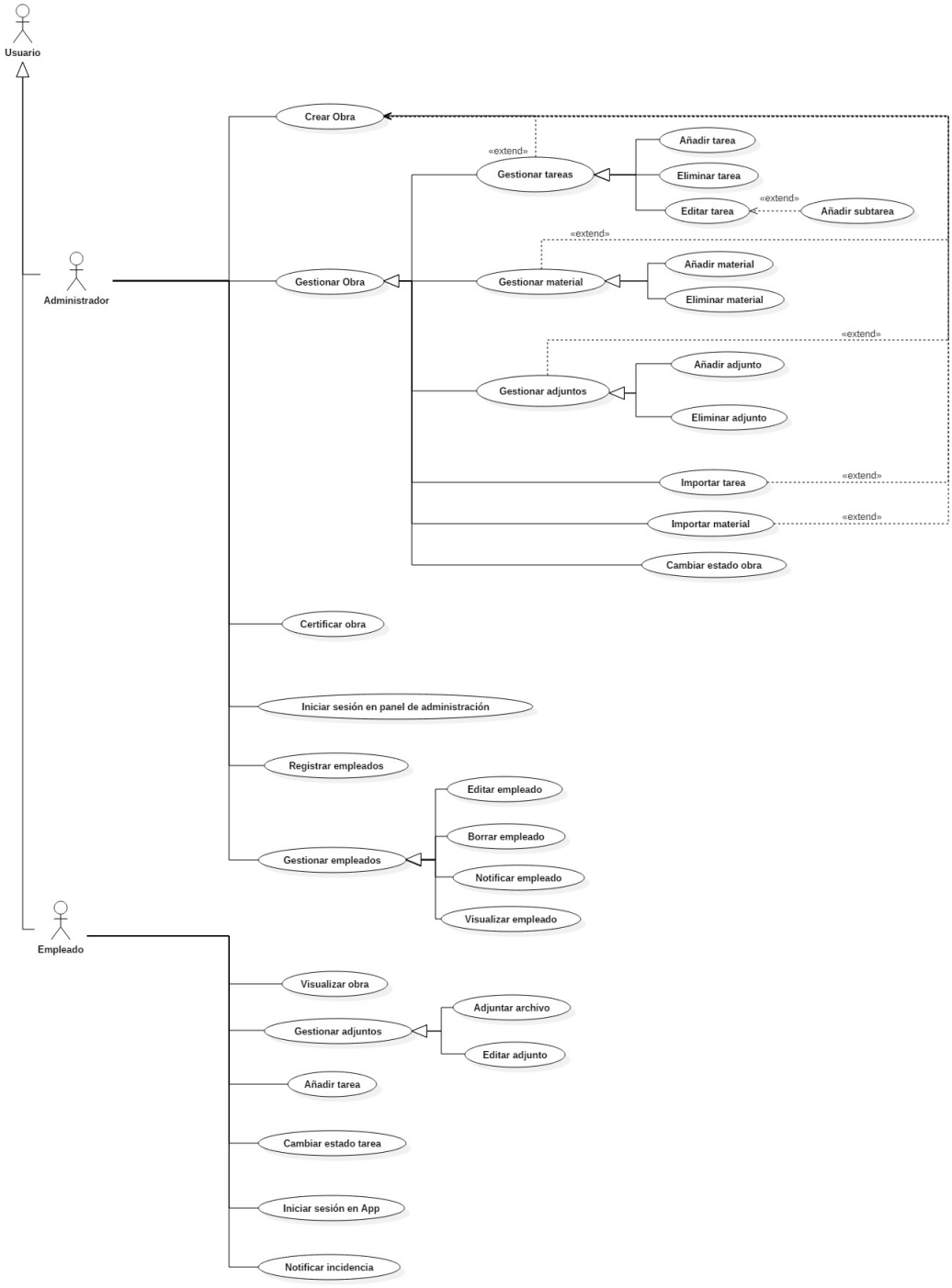

Ilustración 24. Diagrama de casos de uso completo

Nombre	Iniciar sesión en el panel de administración	ID	1
Creado por	Álvaro Romero Perdomo	Fecha	22/10/2016
Modif. por	Álvaro Romero Perdomo	Fecha modif.	27/01/2017

Actor principal: ADMINISTRADOR
Personal involucrado o intereses: 1. ADMINISTRADOR: quiere acceder al panel de administración.
Descripción: El ADMINISTRADOR, en el panel de administración, accede al sistema haciendo uso de sus credenciales.
Precondición: El ADMINISTRADOR debe estar registrado en el sistema previamente.
Postcondición: El ADMINISTRADOR es redirigido al panel de administración.
Flujo normal: 1. El ADMINISTRADOR abre el panel de administración web. 2. El ADMINISTRADOR introduce sus credenciales (usuario y contraseña). 3. El ADMINISTRADOR es conducido al panel principal de administración.
Flujo alternativo: 2.a. Credenciales no válidas: 2.a.1. El sistema muestra un mensaje de error.

Tabla 7. Caso de uso: Iniciar sesión en el panel de administración

Nombre	Crear obra	**ID**	2
Creado por	Alejandro Pérez Martín	**Fecha**	22/10/2016
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	27/01/2017

Actor principal:
ADMINISTRADOR

Personal involucrado o intereses:
1. ADMINISTRADOR: quiere poder crear obras.

Descripción:
El ADMINISTRADOR crea una obra para asignarla, al menos, a un empleado.

Precondición:
1. El ADMINISTRADOR debe estar autenticado, acceder al panel de administración y debe pulsar el botón *"Crear obra"*.

Postcondición:
La obra se guarda en la base de datos y es accesible para el EMPLEADO desde el panel de trabajo de la aplicación móvil y para el ADMINISTRADOR desde el panel de principal.

Extensiones:
1. Añadir tarea (3.1.1)
2. Importar tarea (3.2)
3. Añadir material (3.3.1)
4. Importar material (3.4)
5. Añadir adjunto (3.5.1)
6. Cambiar estado de la obra (3.6)

Flujo normal:
1. El ADMINISTRADOR accede al formulario de creación de obras a través de la interfaz.
2. El ADMINISTRADOR introduce los datos requeridos en el formulario básico de creación de una obra y al terminar pulsa en *"Continuar".*
3. El sistema guarda esa información básica en la base de datos.
4. El ADMINISTRADOR es redirigido a la gestión/edición de la obra. Esta acción involucra los siguientes casos de uso:

a. Añadir tarea (3.1.1)
b. Importar tareas (3.2)
c. Añadir material (3.3.1)
d. Importar material (3.4)
e. Añadir adjunto (3.5.1)
f. Cambiar estado de la obra (3.6)

5. El ADMINISTRADOR pulsa el botón *"Finalizar"*.

Flujo alternativo:

Tabla 8. Caso de uso: Crear obra

Nombre	Gestionar obra	**ID**	3
Creado por	Alejandro Pérez Martín	**Fecha**	22/10/2016
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	27/01/2017

Actor principal:
ADMINISTRADOR

Personal involucrado o intereses:
1. ADMINISTRADOR: quiere poder gestionar las obras.

Descripción:
El ADMINISTRADOR edita una obra ya creada.

Precondición:
El ADMINISTRADOR debe estar autenticado y la obra debe estar creada previamente.

Postcondición:
La información de la obra es modificada.

Flujo normal:
1. El ADMINISTRADOR accede al portal web y pulsa sobre una de las obras del listado.
2. El ADMINISTRADOR es conducido a la vista de los detalles de la obra.
3. El ADMINISTRADOR pulsa el botón *"Modificar obra"* que aparece en esta vista y accede al formulario de edición de obras a través de la interfaz.
4. El ADMINISTRADOR modifica/añade los datos requeridos en el formulario de edición de obras. Esta acción involucra los siguientes casos de uso:

a. Gestionar tareas (3.1)
b. Importar tareas (3.2)
c. Gestionar material (3.3)
d. Importar material (3.4)
e. Gestionar adjuntos (3.5)
f. Cambiar estado (3.6)
g. Añadir subtarea (3.7)

5. El ADMINISTRADOR pulsa el botón *"Guardar"*.

Flujo alternativo:
4.a. Errores de validación de campos:
4.a.1. Se muestra un mensaje de error al ADMINISTRADOR a través de la interfaz.

Tabla 9. Caso de uso: Gestionar obra

Nombre	Gestionar tareas	**ID**	3.1
Creado por	Álvaro Romero Perdomo	**Fecha**	27/01/2017
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	27/01/2017

Actor principal: ADMINISTRADOR
Personal involucrado o intereses: 1. ADMINISTRADOR: quiere poder gestionar las tareas de una obra.
Descripción: El ADMINISTRADOR puede crear, editar o eliminar tareas pertenecientes a una obra a través del diálogo de gestionar tarea o en el momento de creación de la obra.
Precondición: El ADMINISTRADOR debe estar autenticado y la obra debe estar creada previamente o estar el ADMINISTRADOR en el formulario de creación de obras.
Postcondición: La información de la obra relativa a la tarea se modifica en la base de datos.
Flujo normal: 1. El ADMINISTRADOR se encuentra en el formulario de creación/edición de obras. 2. El ADMINISTRADOR pulsa sobre el botón *"Añadir tarea"*. 3. El ADMINISTRADOR completa los campos del formulario requeridos. 4. El ADMINISTRADOR pulsa el botón *"Aceptar"* y la tarea es añadida a la lista de tareas. a. Los pasos 2-4 pueden repetirse tantas veces como tareas necesite añadir el ADMINISTRADOR. 5. El ADMINISTRADOR puede editar esa tarea si lo desea, eliminarla o continuar si lo prefiere. 6. El ADMINISTRADOR pulsa el botón *"Siguiente"* y es dirigido al siguiente paso del formulario de creación de obras.
Flujo alternativo: 3.a. Errores de validación de campos: 3.a.1. Se muestra un mensaje de error al ADMINISTRADOR a través de la interfaz.

Tabla 10. Gestionar tareas

Nombre	Añadir tarea	**ID**	3.1.1
Creado por	Alejandro Pérez Martín	**Fecha**	22/10/2016
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	27/01/2017

Actor principal: ADMINISTRADOR
Personal involucrado o intereses: 1. ADMINISTRADOR: quiere añadir tareas a una obra.
Descripción: El ADMINISTRADOR mediante el formulario de creación/edición de obras añade tareas a la misma.
Precondición: El ADMINISTRADOR debe estar autenticado y la obra debe estar creada previamente o estar el ADMINISTRADOR en el formulario de creación de obras.
Postcondición: La tarea se añade a la obra y se modifica en la base de datos.
Flujo normal: 1. El ADMINISTRADOR se encuentra en el formulario de creación de obras o en el de edición de obras. 2. El ADMINISTRADOR pulsa sobre el botón *"Añadir tarea"*. 3. El ADMINISTRADOR completa los campos del formulario requeridos. 4. El ADMINISTRADOR pulsa el botón *"Aceptar"* y la tarea es añadida a la lista de tareas. a. Los pasos 2-4 pueden repetirse tantas veces como tareas necesite añadir el ADMINISTRADOR. 5. El ADMINISTRADOR pulsa el botón *"Siguiente"* y es dirigido al siguiente paso del formulario de creación de obras.
Flujo alternativo: 3.a. Errores de validación de campos: 3.a.1. Se muestra un mensaje de error al ADMINISTRADOR a través de la interfaz.

Tabla 11. Caso de uso: Añadir tarea

Nombre	Eliminar tarea	**ID**	3.1.2
Creado por	Álvaro Romero Perdomo	**Fecha**	27/01/2017
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	09/02/2017

Actor principal: ADMINISTRADOR
Personal involucrado o intereses: 1. ADMINISTRADOR: quiere eliminar tareas de una obra.
Descripción: El ADMINISTRADOR mediante el formulario de creación o edición de obras elimina tareas de la misma.
Precondición: El ADMINISTRADOR debe estar autenticado y la obra debe estar creada previamente o estar el ADMINISTRADOR en el formulario de creación/edición de obras.
Postcondición: La tarea se elimina de la obra en la base de datos.
Flujo normal: 1. El ADMINISTRADOR se encuentra en el formulario de creación/edición de obras. 2. El ADMINISTRADOR pulsa sobre el botón *"Eliminar tarea"*. 3. El ADMINISTRADOR pulsa el botón *"Confirmar"* y la tarea es eliminada de la base de datos.
Flujo alternativo:

Tabla 12. Caso de uso: Eliminar tarea

Nombre	Editar tarea	**ID**	3.1.3
Creado por	Álvaro Romero Perdomo	**Fecha**	27/01/2017
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	27/01/2017

Actor principal: ADMINISTRADOR
Personal involucrado o intereses: 1. ADMINISTRADOR: quiere editar tareas de una obra.
Descripción: El ADMINISTRADOR mediante el formulario de edición de tareas edita tareas de una obra.
Precondición: El ADMINISTRADOR debe estar autenticado y la obra debe estar creada previamente o estar el ADMINISTRADOR en el formulario de creación/edición de obras.
Postcondición: La tarea asignada a una obra se elimina de la base de datos.
Flujo normal: 1. El ADMINISTRADOR se encuentra en el formulario de creación de obras o edición de obras. 2. El ADMINISTRADOR pulsa sobre el botón *"Editar tarea"*. 3. El ADMINISTRADOR edita los campos de la tarea en el formulario que se muestra. 4. El ADMINISTRADOR pulsa el botón *"Aceptar"* y la tarea es editada.
Flujo alternativo:

Tabla 13. Caso de uso: Editar tarea

Nombre	Añadir subtarea	**ID**	3.1.3.1
Creado por	Álvaro Romero Perdomo	**Fecha**	27/01/2017
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	09/02/2017

Actor principal: ADMINISTRADOR
Personal involucrado o intereses: 1. ADMINISTRADOR: quiere añadir subtareas a una tarea.
Descripción: El ADMINISTRADOR mediante el formulario de edición de tareas añade subtareas a la misma.
Precondición: El ADMINISTRADOR debe estar autenticado el sistema y encontrarse en el formulario de creación/edición de tareas.
Postcondición: La subtarea se añade a la tarea y se modifica en la base de datos.
Flujo normal: 1. El ADMINISTRADOR se encuentra en el formulario de creación de obras o en el de edición de obras. 2. El ADMINISTRADOR pulsa sobre el botón *"Editar tarea"*. 3. El ADMINISTRADOR pulsa sobre el botón *"Añadir subtarea"*. 4. El ADMINISTRADOR selecciona de un desplegable, entre los tipos de subtareas presentes en la base de datos, la subtarea a añadir. a. Los pasos 3 y 4 pueden repetirse tantas veces como subtareas necesite añadir el ADMINISTRADOR. 5. El ADMINISTRADOR pulsa el botón *"Finalizar"* y termina la edición de la tarea.
Flujo alternativo:

Tabla 14. Caso de uso: Añadir subtarea

Nombre	Importar tareas	**ID**	3.2
Creado por	Alejandro Pérez Martín	**Fecha**	22/10/2016
Modif. por	Alejandro Pérez Martín	**Fecha modif.**	22/10/2016

Actor principal: ADMINISTRADOR
Personal involucrado o intereses: 1. ADMINISTRADOR: quiere importar tareas a una obra.
Descripción: El ADMINISTRADOR mediante el formulario de creación o edición de obras importa tareas a la misma.
Precondición: El ADMINISTRADOR debe estar autenticado y la obra debe estar creada previamente o estar el ADMINISTRADOR en el formulario de creación de obras.
Postcondición: La información de la obra se modifica en la base de datos.
Flujo normal: 1. El ADMINISTRADOR se encuentra en el formulario de creación de obras. 2. El ADMINISTRADOR pulsa sobre el botón *"Importar tareas"*. 3. El ADMINISTRADOR selecciona un archivo *Excel (.xls* o *<.csv)* de su equipo donde se encuentran las tareas que quiere importar. 4. El sistema analiza el archivo y automáticamente añade las tareas encontradas a la lista de tareas. 5. El ADMINISTRADOR puede seguir añadiendo tareas como se describe en el caso de uso "*3.1.1. Añadir tareas"*. 6. El ADMINISTRADOR pulsa el botón *"Siguiente"* y es dirigido al siguiente paso del formulario de creación/edición de obras.
Flujo alternativo: 1.a. El ADMINISTRADOR accede al portal web y pulsa sobre una de las obras para editar sus tareas. 3.a. Errores de validación de campos: 3.a.1. Se muestra un mensaje de error al ADMINISTRADOR a través de la interfaz.

Tabla 15. Caso de uso: Importar tareas

Nombre	Gestionar material	**ID**	3.3
Creado por	Álvaro Romero Perdomo	**Fecha**	27/01/2017
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	09/02/2017

Actor principal: ADMINISTRADOR
Personal involucrado o intereses: 1. ADMINISTRADOR: quiere gestionar el material de una obra.
Descripción: El ADMINISTRADOR mediante el formulario de creación o edición de obras edita el material de la misma.
Precondición: El ADMINISTRADOR debe estar autenticado y la obra debe estar creada previamente o estar el ADMINISTRADOR en el formulario de creación de obras.
Postcondición: La información del material de la obra se modifica en la base de datos.
Flujo normal: 1. El ADMINISTRADOR se encuentra en el formulario de creación o edición de obras. 2. El ADMINISTRADOR accede al módulo de gestión de materiales dónde puede eliminar material existente o añadir material nuevo. 3. El ADMINISTRADOR pulsa el botón *"Finalizar"* y la información de la obra se almacena en la base de datos.
Flujo alternativo:

Tabla 16. Caso de uso: Gestionar material

Nombre	Añadir material	**ID**	3.3.1
Creado por	Alejandro Pérez Martín	**Fecha**	22/10/2016
Modif. por	Alejandro Pérez Martín	**Fecha modif.**	22/10/2016

Actor principal: ADMINISTRADOR
Personal involucrado o intereses: 1. ADMINISTRADOR: quiere añadir material a una obra.
Descripción: El ADMINISTRADOR mediante el formulario de creación o edición de obras añade material a la misma.
Precondición: El ADMINISTRADOR debe estar autenticado y la obra debe estar creada previamente o estar el ADMINISTRADOR en el formulario de creación de obras.
Postcondición: La información de la obra se modifica en la base de datos.
Flujo normal: 1. El ADMINISTRADOR se encuentra en el formulario de creación o edición de obras. 2. El ADMINISTRADOR pulsa sobre el botón *"Añadir material"*. 3. El ADMINISTRADOR completa los campos del formulario requeridos. 4. El ADMINISTRADOR pulsa el botón *"Aceptar"* y el material es añadido a la lista de materiales. a. Los pasos 2-4 pueden repetirse tantas veces como material necesite añadir el ADMINISTRADOR. 5. El ADMINISTRADOR pulsa el botón *"Siguiente"* y es dirigido al siguiente paso del formulario de creación/edición de obras.
Flujo alternativo:

Tabla 17. Caso de uso: Añadir material

Nombre	Eliminar material	**ID**	3.3.2
Creado por	Álvaro Romero Perdomo	**Fecha**	27/01/2017
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	27/01/2017

Actor principal: ADMINISTRADOR
Personal involucrado o intereses: 1. ADMINISTRADOR: quiere eliminar material de una obra.
Descripción: El ADMINISTRADOR mediante el formulario de creación o edición de obras elimina material asociado a la misma.
Precondición: El ADMINISTRADOR debe estar autenticado y la obra debe estar creada previamente o estar el ADMINISTRADOR en el formulario de creación o edición de obras.
Postcondición: La información de la obra se modifica en la base de datos.
Flujo normal: 1. El ADMINISTRADOR se encuentra en el formulario de creación o edición de obras. 2. El ADMINISTRADOR pulsa sobre el botón *"Eliminar material"*. 3. El ADMINISTRADOR pulsa el botón *"Finalizar"* y el material asociado a la obra se elimina de la base de datos.
Flujo alternativo:

Tabla 18. Caso de uso: Editar material

Nombre	Importar material	**ID**	3.4
Creado por	Alejandro Pérez Martín	**Fecha**	22/10/2016
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	09/02/2017

Actor principal: ADMINISTRADOR
Personal involucrado o intereses: 1. ADMINISTRADOR: quiere importar material a una obra.
Descripción: El ADMINISTRADOR mediante el formulario de creación o edición de obras importa materiales a la misma.
Precondición: El ADMINISTRADOR debe estar autenticado y la obra debe estar creada previamente o estar el ADMINISTRADOR en el formulario de creación de obras.
Postcondición: La información de la obra se modifica en la base de datos.
Flujo normal: 1. El ADMINISTRADOR se encuentra en el formulario de creación de obras. 2. El ADMINISTRADOR pulsa sobre el botón *"Importar material"*. 3. El ADMINISTRADOR selecciona un archivo *Excel (.xls* o *.csv)* de su equipo donde se encuentran los materiales que quiere importar. 4. El sistema analiza el archivo y automáticamente añade los materiales encontrados a la lista de materiales. 5. El ADMINISTRADOR puede seguir añadiendo material como se describe en el caso de uso "*3.3.1. Añadir material".* 6. El ADMINISTRADOR pulsa el botón *"Siguiente"* y es dirigido al siguiente paso del formulario de creación/edición de obras.
Flujo alternativo:

Tabla 19. Caso de uso: Importar material

Nombre	Gestionar adjuntos	**ID**	3.5
Creado por	Alejandro Pérez Martín	**Fecha**	27/01/2017
Modif. por	Alejandro Pérez Martín	**Fecha modif.**	27/01/2017

Actor principal: ADMINISTRADOR
Personal involucrado o intereses: 1. ADMINISTRADOR: quiere gestionar archivos adjuntos a una obra.
Descripción: El ADMINISTRADOR mediante el formulario de creación o edición de obras gestiona los archivos adjuntos a la misma.
Precondición: El ADMINISTRADOR debe estar autenticado y la obra debe estar creada previamente o estar el ADMINISTRADOR en el formulario de creación o edición de obras.
Postcondición: La información de la obra se modifica en la base de datos.
Flujo normal: 1. El ADMINISTRADOR se encuentra en el formulario de creación o edición de obras. 2. El ADMINISTRADOR accede al módulo de edición de adjuntos de una obra donde puede añadir un nuevo archivo o eliminar los existentes. 3. El ADMINISTRADOR pulsa el botón *"Finalizar"* y la información de la obra se almacena en la base de datos.
Flujo alternativo:

Tabla 20. Caso de uso: Gestionar adjuntos

Nombre	Añadir adjunto	**ID**	3.5.1
Creado por	Alejandro Pérez Martín	**Fecha**	22/10/2016
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	27/01/2017

Actor principal: ADMINISTRADOR
Personal involucrado o intereses: 1. ADMINISTRADOR: quiere añadir archivos adjuntos a una obra.
Descripción: El ADMINISTRADOR mediante el formulario de creación o edición de obras añade archivos adjuntos a la misma.
Precondición: El ADMINISTRADOR debe estar autenticado y la obra debe estar creada previamente o estar el ADMINISTRADOR en el formulario de creación o edición de obras.
Postcondición: La información de la obra se modifica en la base de datos.
Flujo normal: 1. El ADMINISTRADOR se encuentra en el formulario de creación o edición de obras. 2. El ADMINISTRADOR pulsa sobre el botón *"Añadir adjunto"*. 3. El ADMINISTRADOR selecciona un archivo de su equipo que quiera adjuntar a la obra. 4. El sistema adjunta el archivo si es de un formato válido. a. Los pasos 2-4 pueden repetirse tantas veces como archivos adjuntos necesite añadir el ADMINISTRADOR. 5. El ADMINISTRADOR pulsa el botón *"Siguiente"* y es dirigido al siguiente paso del formulario de creación/edición de obras.
Flujo alternativo:

Tabla 21. Caso de uso: Añadir adjunto

Nombre	Eliminar adjunto	**ID**	3.5.2
Creado por	Alejandro Pérez Martín	**Fecha**	22/10/2016
Modif. por	Alejandro Pérez Martín	**Fecha modif.**	27/01/2017

Actor principal: ADMINISTRADOR
Personal involucrado o intereses: 1. ADMINISTRADOR: quiere eliminar archivos adjuntos de una obra.
Descripción: El ADMINISTRADOR mediante el formulario de creación o edición de obras elimina archivos adjuntos a la misma.
Precondición: El ADMINISTRADOR debe estar autenticado y la obra debe estar creada previamente o estar el ADMINISTRADOR en el formulario de creación de obras.
Postcondición: La información de la obra se modifica en la base de datos.
Flujo normal: 1. El ADMINISTRADOR se encuentra en el formulario de creación o edición de obras. 2. El ADMINISTRADOR accede a la pestaña de edición de adjuntos a esa obra. 3. El ADMINISTRADOR pulsa sobre el botón *"Eliminar adjunto"*. a. El paso 3 se puede repetir tantas veces como archivos adjuntos necesite eliminar el ADMINISTRADOR. 4. El ADMINISTRADOR pulsa el botón *"Finalizar"* y la información de la obra se actualiza en la base de datos.
Flujo alternativo:

Tabla 22. Caso de uso: Eliminar adjunto

Nombre	Cambiar estado de una obra	**ID**	3.6
Creado por	Alejandro Pérez Martín	**Fecha**	22/10/2016
Modif. por	Alejandro Pérez Martín	**Fecha modif.**	22/10/2016

Actor principal: ADMINISTRADOR
Personal involucrado o intereses: 1. ADMINISTRADOR: quiere cambiar el estado de una obra.
Descripción: El ADMINISTRADOR mediante un desplegable tendrá la opción de cambiar el estado en que se encuentra una obra (*Pendiente, En curso y Finalizada*).
Precondición: El ADMINISTRADOR debe estar autenticado y la obra debe estar creada previamente.
Postcondición: La información de la obra se modifica en la base de datos.
Flujo normal: 1. El ADMINISTRADOR se encuentra en el listado de obras. 2. El ADMINISTRADOR pulsa el campo desplegable *"Estado"* de una obra. 3. El ADMINISTRADOR selecciona uno de los estados de la obra. 4. El SISTEMA automáticamente cambia el valor del estado de la obra en la base de datos
Flujo alternativo: 3.a. Si el estado seleccionado es *"Finalizada"* la obra se moverá automáticamente al listado de obras terminadas.

Tabla 23. Caso de uso: Cambiar estado de una obra

Nombre	Certificar obra	**ID**	3.7
Creado por	Alejandro Pérez Martín	**Fecha**	13/12/2016
Modif. por	Alejandro Pérez Martín	**Fecha modif.**	13/12/2016

Actor principal: ADMINISTRADOR
Personal involucrado o intereses: ADMINISTRADOR: quiere poder certificar una obra.
Descripción: El ADMINISTRADOR a través de la vista de la obra tendrá acceso a la certificación por medio de un botón.
Precondición: El ADMINISTRADOR debe estar autenticado, la obra debe estar creada y finalizada previamente.
Postcondición: Se genera un fichero de certificación.
Flujo normal: 1. El ADMINISTRADOR se encuentra en el listado de obras. 2. El ADMINISTRADOR selecciona una obra que desee certificar. 3. El ADMINISTRADOR pulsa el botón de certificar obra. 4. El SISTEMA genera los ficheros asociados a la certificación.
Flujo alternativo: 3.a. Si el estado seleccionado no es *"Finalizada",* se le mostrará un mensaje al usuario indicando que no puede certificar la obra.

Tabla 24. Caso de uso: Certificar obra

Nombre	Registrar empleados	**ID**	4
Creado por	Álvaro Romero Perdomo	**Fecha**	22/10/2016
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	22/10/2016

Actor principal: ADMINISTRADOR
Personal involucrado o intereses: 1. ADMINISTRADOR: quiere poder registrar empleados en el sistema. 2. EMPLEADO: quiere que el ADMINISTRADOR pueda darle de alta en el sistema y así poder registrar su actividad como EMPLEADO.
Descripción: El ADMINISTRADOR puede registrar empleados de distintos perfiles de empleado que posteriormente podrá editar o eliminar. Estos empleados podrán ser asignados a obras por parte del administrador y éste podrá asimismo realizar un seguimiento profesional del empleado.
Precondición: El ADMINISTRADOR debe estar autenticado.
Postcondición: El usuario queda insertado en el sistema.
Flujo normal: 1. El ADMINISTRADOR accede al portal de administración. 2. El ADMINISTRADOR accede al formulario de registro de empleado a través de la interfaz. 3. El ADMINISTRADOR introduce los datos requeridos en el formulario de registro de empleado. 4. El ADMINISTRADOR introduce, entre los datos requeridos, el rol del empleado. 5. El empleado queda registrado en el sistema.
Flujo alternativo: 3.a. Errores de validación de campos: 3.a.1. Se muestra un mensaje de error al ADMINISTRADOR a través de la interfaz. 3.a.2. Se permite al ADMINISTRADOR volver a rellenar los campos erróneos.

Tabla 25. Caso de uso: Registrar empleados

Nombre	Gestionar empleados	**ID**	5
Creado por	Álvaro Romero Perdomo	**Fecha**	22/10/2016
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	22/10/2016

Actor principal: ADMINISTRADOR
Personal involucrado o intereses: 1. ADMINISTRADOR: quiere poder editar/borrar empleados del sistema.
Descripción: El ADMINISTRADOR puede editar la información de un empleado ya existente, así como eliminar el mismo si fuera necesario.
Precondición: El ADMINISTRADOR debe estar autenticado y existir al menos un empleado a editar/borrar en el sistema.
Postcondición: 1. La información del empleado queda actualizada. 2. El empleado queda eliminado del sistema.
Flujo normal: 1. El ADMINISTRADOR accede al portal de administración. 2. El ADMINISTRADOR accede a la vista de los empleados. 3. El ADMINISTRADOR selecciona el empleado que desea gestionar. a. Editar empleado (5.1). b. Borrar empleado (5.2). c. Notificar empleado (5.3). d. Visualizar empleado (5.4).
Flujo alternativo:

Tabla 26. Caso de uso: Gestionar empleados

Nombre	Editar empleado	**ID**	5.1
Creado por	Álvaro Romero Perdomo	**Fecha**	22/10/2016
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	22/10/2016

Actor principal: ADMINISTRADOR
Personal involucrado o intereses: ADMINISTRADOR: quiere poder editar empleados del sistema.
Descripción: El ADMINISTRADOR puede editar la información de un empleado ya existente.
Precondición: El ADMINISTRADOR debe estar autenticado y existir al menos un empleado en el sistema.
Postcondición: 1. La información del empleado queda actualizada.
Flujo normal: 1. El ADMINISTRADOR accede al portal de administración. 2. El ADMINISTRADOR accede a la vista de los empleados. 3. El ADMINISTRADOR selecciona el empleado que desea editar. 4. El ADMINISTRADOR accede al formulario de edición del empleado a través de la interfaz. 5. El ADMINISTRADOR introduce los campos que desea editar del empleado. 6. El ADMINISTRADOR finaliza la edición de la información del empleado. 7. La información queda actualizada en el sistema.
Flujo alternativo: 5.1. Errores de validación de campos: 5.1.a. Se muestra un mensaje de error al ADMINISTRADOR a través de la interfaz. 5.1.b. Se permite al ADMINISTRADOR volver a rellenar los campos erróneos.

Tabla 27. Caso de uso: Editar empleado

Nombre	Eliminar empleado	**ID**	5.2
Creado por	Álvaro Romero Perdomo	**Fecha**	22/10/2016
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	22/10/2016

Actor principal: ADMINISTRADOR
Personal involucrado o intereses: ADMINISTRADOR: quiere poder borrar empleados del sistema.
Descripción: El ADMINISTRADOR puede eliminar un empleado del sistema.
Precondición: El ADMINISTRADOR debe estar autenticado y existir al menos un empleado a eliminar en el sistema.
Postcondición: 1. El empleado queda eliminado del sistema.
Flujo normal: 1. El ADMINISTRADOR accede al portal de administración. 2. El ADMINISTRADOR accede a la vista de los empleados. 3. El ADMINISTRADOR selecciona el empleado que desea eliminar. 4. El ADMINISTRADOR elimina al empleado del sistema a través de la interfaz. 5. El ADMINISTRADOR confirma la eliminación del empleado. 6. El empleado queda eliminado del sistema.
Flujo alternativo: 5.1. El ADMINISTRADOR decide no confirmar la eliminación: 5.1.a. El sistema vuelve a la vista de los empleados.

Tabla 28. Caso de uso: Eliminar empleado

Nombre	Notificar empleado	**ID**	5.3
Creado por	Álvaro Romero Perdomo	**Fecha**	22/10/2016
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	22/10/2016

Actor principal: ADMINISTRADOR
Personal involucrado o intereses: 1. ADMINISTRADOR: quiere poder notificar a los empleados. 2. EMPLEADO: el empleado desea poder recibir notificaciones en tiempo real.
Descripción: El ADMINISTRADOR enviar un mensaje a un empleado en concreto a través del portal web. El empleado recibirá una notificación en tiempo real en la aplicación móvil.
Precondición: El ADMINISTRADOR debe estar autenticado y el sistema debe tener al menos un empleado a notificar.
Postcondición: El EMPLEADO recibe una notificación en tiempo real en la aplicación móvil.
Flujo normal: 1. El ADMINISTRADOR accede al portal de administración. 2. El ADMINISTRADOR accede a la vista de los empleados. 3. El ADMINISTRADOR selecciona el empleado que desea notificar. 4. El ADMINISTRADOR envía el mensaje al EMPLEADO. 5. El EMPLEADO recibe una notificación en la aplicación móvil.
Flujo alternativo: 5.1. Cuerpo del mensaje vacío: 5.1.a. Se muestra al ADMINISTRADOR un mensaje de error indicando que debe introducir el cuerpo del mensaje.

Tabla 29. Caso de uso: Notificar empleado

Nombre	Visualizar empleado	**ID**	5.4
Creado por	Álvaro Romero Perdomo	**Fecha**	22/12/2016
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	22/12/2016

Actor principal: ADMINISTRADOR
Personal involucrado o intereses: 1. ADMINISTRADOR: quiere poder visualizar el rendimiento de los empleados.
Descripción: El ADMINISTRADOR visualiza el rendimiento de los empleados, donde observa la actividad de éste como trabajador, las obras en las que ha participado, etc...
Precondición: El ADMINISTRADOR debe estar autenticado y el sistema debe tener al menos un empleado a visualizar.
Postcondición:
Flujo normal: 1. El ADMINISTRADOR accede al portal de administración. 2. El ADMINISTRADOR accede a la vista de los empleados. 3. El ADMINISTRADOR selecciona el empleado que desea visualizar.
Flujo alternativo:

Tabla 30. Caso de uso: Visualizar empleado

Nombre	Iniciar sesión en la APP	**ID**	6
Creado por	Álvaro Romero Perdomo	**Fecha**	27/01/2017
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	27/01/2017

Actor principal: EMPLEADO
Personal involucrado o intereses: 1. EMPLEADO: quiere acceder al panel principal de la aplicación móvil.
Descripción: El EMPLEADO ejecuta la aplicación móvil y accede al sistema haciendo uso de sus credenciales.
Precondición: El EMPLEADO debe estar registrado previamente.
Postcondición: El EMPLEADO es redirigido al panel principal de la aplicación móvil.
Flujo normal: 1. El EMPLEADO abre la aplicación móvil. 2. El EMPLEADO introduce sus credenciales (usuario y contraseña). 3. El EMPLEADO es conducido al panel principal donde verá sus obras asignadas.
Flujo alternativo: 2.a. Credenciales no válidas: 2.a.1. El sistema muestra un mensaje de error.

Tabla 31. Caso de uso: Iniciar sesión en la APP

Nombre	Visualizar obra	**ID**	7
Creado por	Álvaro Romero Perdomo	**Fecha**	22/10/2016
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	22/10/2016

Actor principal: EMPLEADO
Personal involucrado o intereses: 1. EMPLEADO: quiere poder visualizar las obras que tiene asignadas en la aplicación móvil.
Descripción: El EMPLEADO puede visualizar las obras que tenga asignadas, así como toda la documentación asociada (datos tales como la dirección, el identificador, estado actual de la obra …) y la lista de tareas a realizar.
Precondición: El ADMINISTRADOR debe estar autenticado y el sistema debe tener un empleado, una obra y ésta debe estar asignada a dicho empleado.
Postcondición:
Flujo normal: 1. El EMPLEADO accede a la aplicación móvil. 2. El EMPLEADO accede a la pantalla principal de la aplicación (*dashboard*). 3. Al EMPLEADO se le presentan las obras que tiene asignadas. 4. El EMPLEADO accede a la obra que elija a través de la interfaz. 5. El EMPLEADO visualiza la información relativa a la obra (datos tales como la dirección, el identificador, estado actual de la obra…).
Flujo alternativo:

Tabla 32. Caso de uso: Visualizar obra

Nombre	Gestionar archivos adjuntos a tarea	**ID**	8
Creado por	Álvaro Romero Perdomo	**Fecha**	22/10/2016
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	22/10/2016

Actor principal: EMPLEADO
Personal involucrado o intereses: 1. EMPLEADO: quiere poder adjuntar o editar información de respaldo de la realización de sus tareas.
Descripción: El EMPLEADO puede adjuntar o editar información de respaldo de la realización de sus tareas a través de la aplicación.
Precondición: El sistema debe tener un empleado, una obra y ésta debe estar asignada a dicho empleado. Además, la obra debe tener al menos una tarea de la que se quiera adjuntar o editar información de respaldo.
Postcondición: 1. La información queda registrada en el sistema. 2. La información queda borrada del sistema.
Flujo normal: 1. El EMPLEADO accede a la aplicación móvil. 2. El EMPLEADO accede a la pantalla principal de la aplicación (*dashboard*). 3. Al EMPLEADO se le presentan las obras que tiene asignadas. 4. El EMPLEADO accede a la obra que elija a través de la interfaz. 5. El EMPLEADO accede al módulo correspondiente a las tareas de dicha obra. 6. El EMPLEADO accede a la tarea que desea. 7. El EMPLEADO selecciona: a. Adjuntar archivo a tarea. b. Editar archivo de tarea.
Flujo alternativo:

Tabla 33. Caso de uso: Gestionar archivos adjuntos a tarea

Nombre	Adjuntar archivo	**ID**	8.1
Creado por	Álvaro Romero Perdomo	**Fecha**	22/10/2016
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	22/10/2016

Actor principal: EMPLEADO
Personal involucrado o intereses: 1. ADMINISTRADOR: quiere poder disponer de información de respaldo de la realización de las tareas por parte de los empleados. 2. EMPLEADO: quiere poder adjuntar información de respaldo de la realización de sus tareas.
Descripción: El EMPLEADO puede adjuntar información de respaldo de la realización de sus tareas a través de la aplicación.
Precondición: El sistema debe tener un empleado, una obra y ésta debe estar asignada a dicho empleado. Además, la obra debe tener al menos una tarea de la que se quiera adjuntar información de respaldo.
Postcondición: 1. La información queda registrada en el sistema.
Flujo normal: 1. El EMPLEADO accede a la aplicación móvil. 2. El EMPLEADO accede a la pantalla principal de la aplicación (*dashboard*). 3. Al EMPLEADO se le presentan las obras que tiene asignadas. 4. El EMPLEADO accede a la obra que elija a través de la interfaz. 5. El EMPLEADO accede al módulo correspondiente a las tareas de dicha obra. 6. El EMPLEADO accede a la tarea que desea. 7. El EMPLEADO selecciona adjuntar archivo a tarea. 8. El EMPLEADO selecciona el archivo a adjuntar de su sistema operativo desde el diálogo de adjuntar archivo. 9. El archivo adjunto asociado a la tarea queda registrado en el sistema.
Flujo alternativo:

Tabla 34. Caso de uso: Adjuntar archivo

Nombre	Eliminar archivo de una tarea	**ID**	8.2
Creado por	Álvaro Romero Perdomo	**Fecha**	22/10/2016
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	22/10/2016

Actor principal:
EMPLEADO

Personal involucrado o intereses:
1. EMPLEADO: quiere poder eliminar archivos adjuntos a una tarea.

Descripción:
El EMPLEADO puede eliminar archivos adjuntos a una tarea.

Precondición:
El EMPLEADO debe estar autenticado y existir una obra asignada a dicho empleado. Además, la obra debe tener al menos una tarea con información adjunta que eliminar.

Postcondición:
1. La información queda eliminada.

Flujo normal:
1. El EMPLEADO accede a la aplicación móvil.
2. El EMPLEADO accede a la pantalla principal de la aplicación (*dashboard*).
3. Al EMPLEADO se le presentan las obras que tiene asignadas.
4. El EMPLEADO accede a la obra que elija a través de la interfaz.
5. El EMPLEADO accede al módulo correspondiente a las tareas de dicha obra.
6. El EMPLEADO accede a la tarea que desea.
7. El EMPLEADO selecciona ver archivos adjuntos.
8. El EMPLEADO selecciona el archivo a eliminar.
9. El archivo adjunto asociado a la tarea va a ser eliminado.
10. El EMPLEADO confirma la acción de eliminar el archivo adjunto.
11. El archivo se elimina del sistema.

Flujo alternativo:
10.1. El EMPLEADO no confirma la acción de eliminar el archivo adjunto.
10.1.a. El sistema devuelve al EMPLEADO a la pantalla de visualización de archivos adjuntos.

Tabla 35. Caso de uso: Eliminar archivos de una tarea

Nombre	Añadir tareas a obra (empleado)	**ID**	9
Creado por	Álvaro Romero Perdomo	**Fecha**	22/10/2016
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	22/10/2016

Actor principal: EMPLEADO
Personal involucrado o intereses: 1. EMPLEADO: quiere añadir tareas a una obra.
Descripción: El EMPLEADO puede añadir tareas desde la aplicación móvil a la obra que tiene asignada si fuera necesario.
Precondición: El EMPLEADO debe estar autenticado y tener una obra asignada a la que añadirle una nueva tarea.
Postcondición: La información de la obra se modifica en la base de datos con la tarea añadida.
Flujo normal: 1. El EMPLEADO accede a la aplicación móvil. 2. El EMPLEADO accede a la pantalla principal de la aplicación (*dashboard*). 3. Al EMPLEADO se le presentan las obras que tiene asignadas. 4. El EMPLEADO accede a la obra que elija a través de la interfaz. 5. El EMPLEADO accede al formulario de añadir una nueva tarea a través de la interfaz. 6. El EMPLEADO especifica información relativa a la tarea a través de los campos indicados. 7. El EMPLEADO finaliza la creación de la tarea mediante el botón "*finalizar*". 8. La tarea queda registrada en el sistema.
Flujo alternativo: 6.1. Errores de validación de campos: 6.1.a. Se muestra un mensaje de error al EMPLEADO a través de la interfaz. 6.1.b. Se permite al EMPLEADO volver a rellenar los campos erróneos.

Tabla 36. Caso de uso: Añadir tareas a obra (empleado)

Nombre	Notificar incidencia	**ID**	11
Creado por	Álvaro Romero Perdomo	**Fecha**	22/10/2016
Modif. por	Álvaro Romero Perdomo	**Fecha modif.**	22/10/2016

Actor principal: EMPLEADO
Personal involucrado o intereses: 1. EMPLEADO: quiere poder notificar incidencias de la obra al administrador.
Descripción: El EMPLEADO puede notificar incidencias de una obra al administrador.
Precondición: El sistema debe contener una obra asignada a dicho empleado.
Postcondición: La notificación queda registrada por el EMPLEADO en la base de datos.
Flujo normal: 1. El EMPLEADO accede a la aplicación móvil. 2. El EMPLEADO accede a la pantalla principal de la aplicación móvil (*dashboard*). 3. Al EMPLEADO se le presentan las obras que tiene asignadas. 4. El EMPLEADO accede a la obra que elija a través de la interfaz. 5. El EMPLEADO accede a la vista de la lista de tareas. 6. El EMPLEADO accede al diálogo de cambio de estado de la tarea a través de la interfaz. 7. El EMPLEADO especifica el estado que desea asignar a la tarea. 8. El EMPLEADO finaliza la edición del estado mediante el botón "*guardar tarea*". 9. La tarea queda registrada en el sistema.
Flujo alternativo:

Tabla 37. Caso de uso: Notificar incidencia

5.2. Anexo 2. Manual de usuario del panel de administración

En este manual se pretende dar al usuario una visión general acerca de cómo utilizar el panel de administración y mostrarle sus principales funcionalidades.

A continuación, se muestra la pantalla de inicio o presentación de la aplicación al acceder a ella.

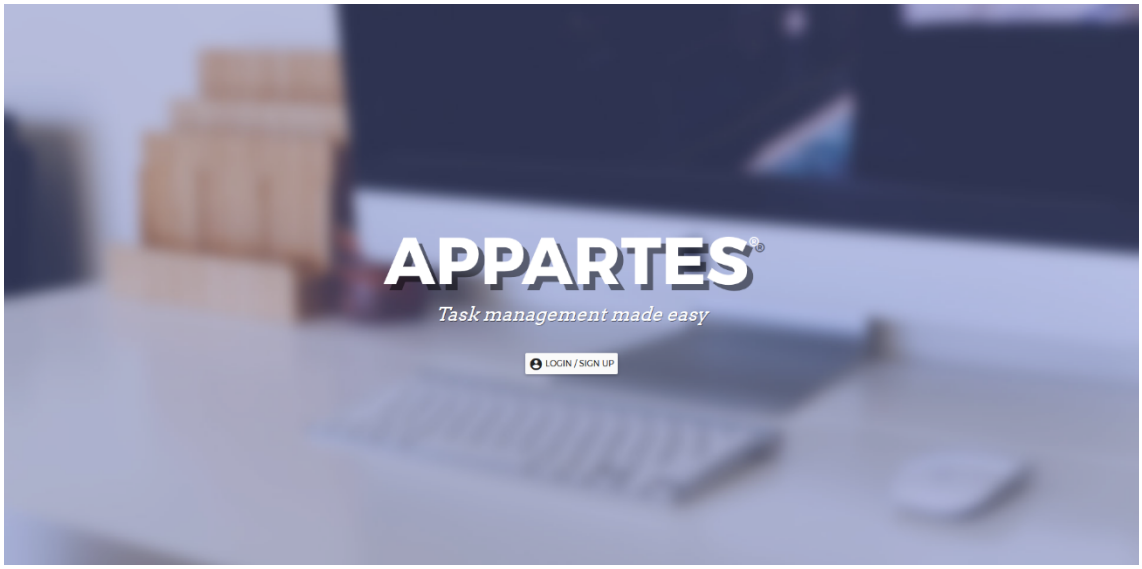

Ilustración 25. Pantalla de inicio del panel de administración

El siguiente paso consistiría en hacer login en la aplicación siempre que tuviéramos una cuenta como administrador en ella.

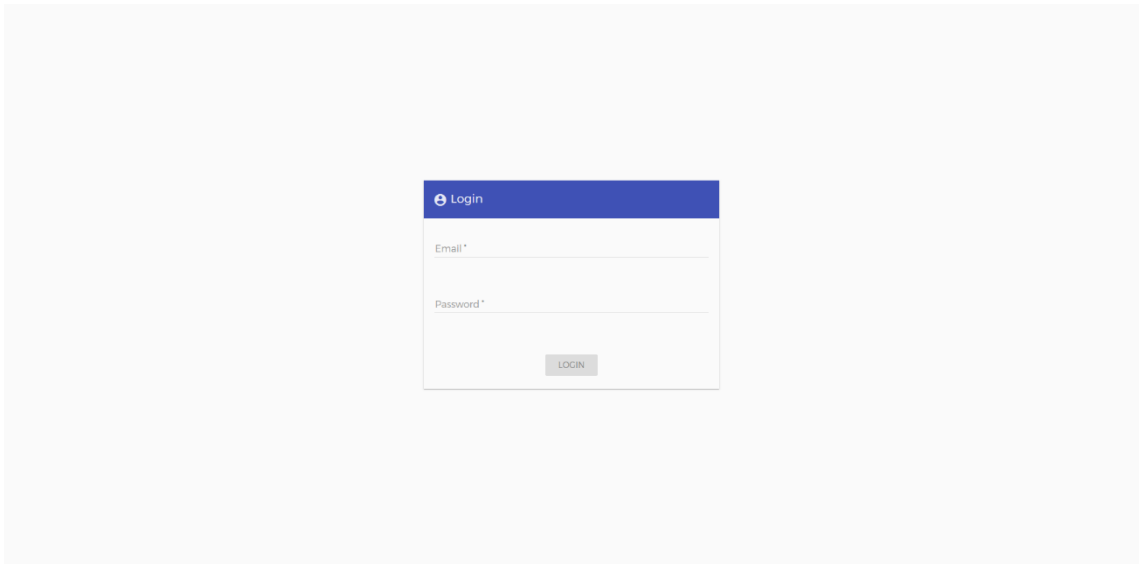

Ilustración 26. Pantalla de login del panel de administración

A continuación, se le presenta al usuario la pantalla principal del panel de administración, el *dashboard* donde se muestran las obras presentes en el sistema, pudiendo editarlas o crear una nueva.

También en esta pantalla el usuario puede ver los usuarios y materiales del sistema.

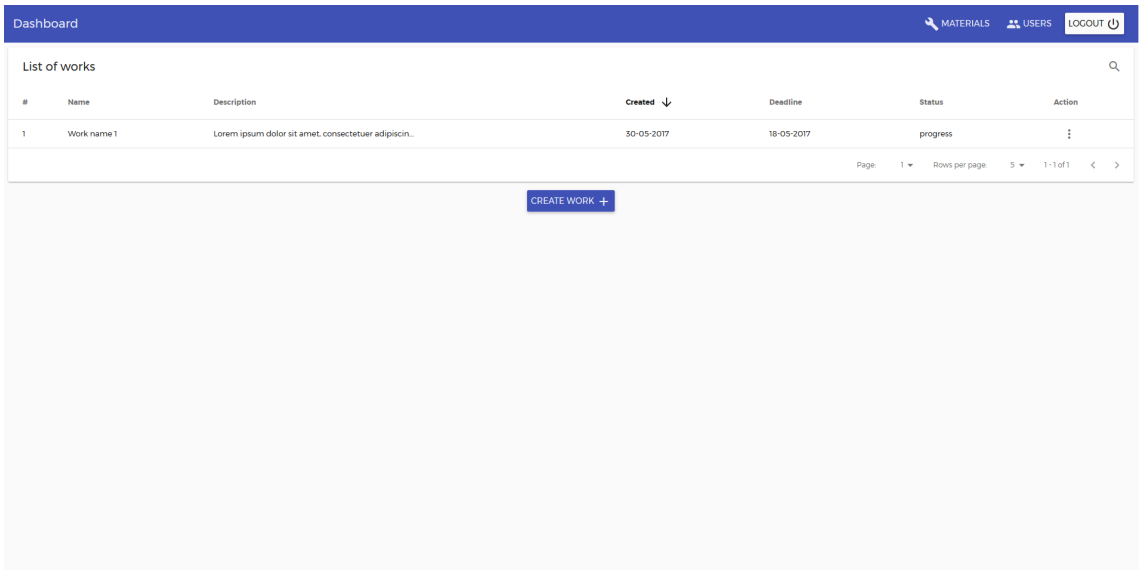

Ilustración 27. Dashboard del panel de administración

A través del botón CREATE WORK de la anterior pantalla, accederíamos a la vista de creación de obra. En este punto deberíamos rellenar los campos que se nos presentan a través de las pestañas podríamos añadir también los materiales, las tareas y los archivos adjuntos de relevancia para dicha obra.

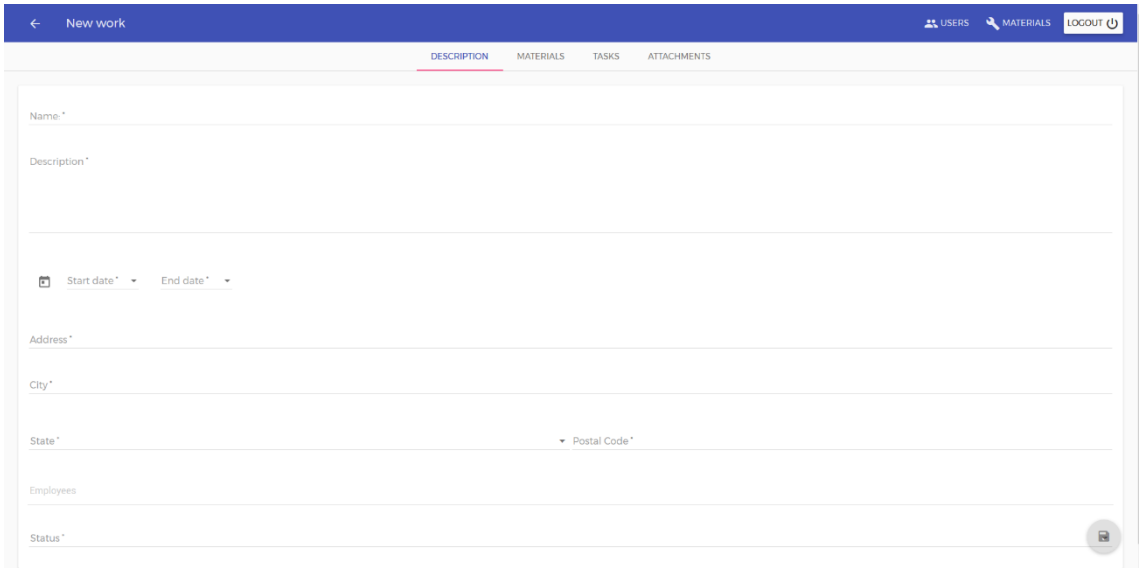

Ilustración 28. Creación de una obra en el panel de administración

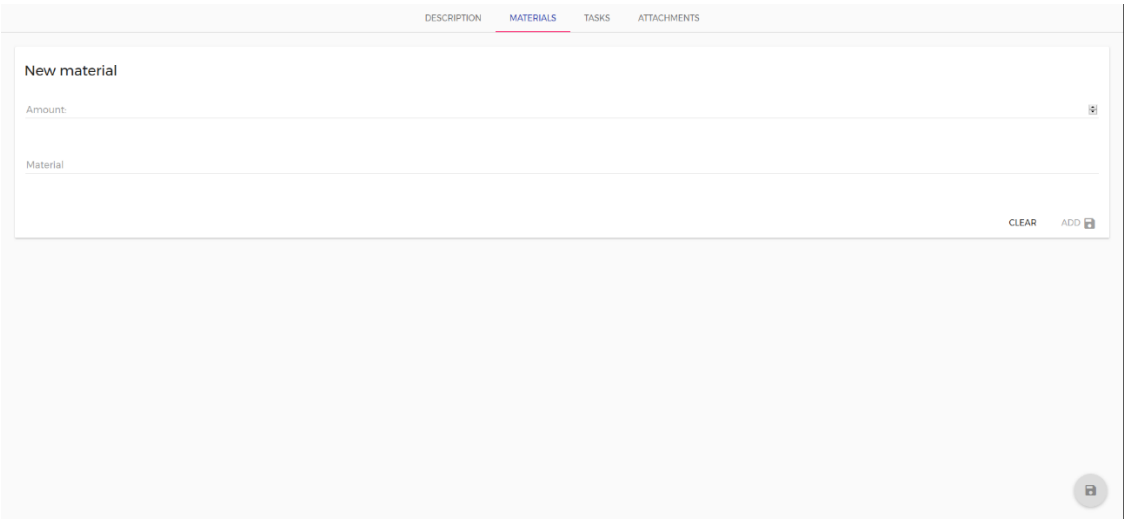

Ilustración 29. Creación de una obra en el panel de administración (materiales)

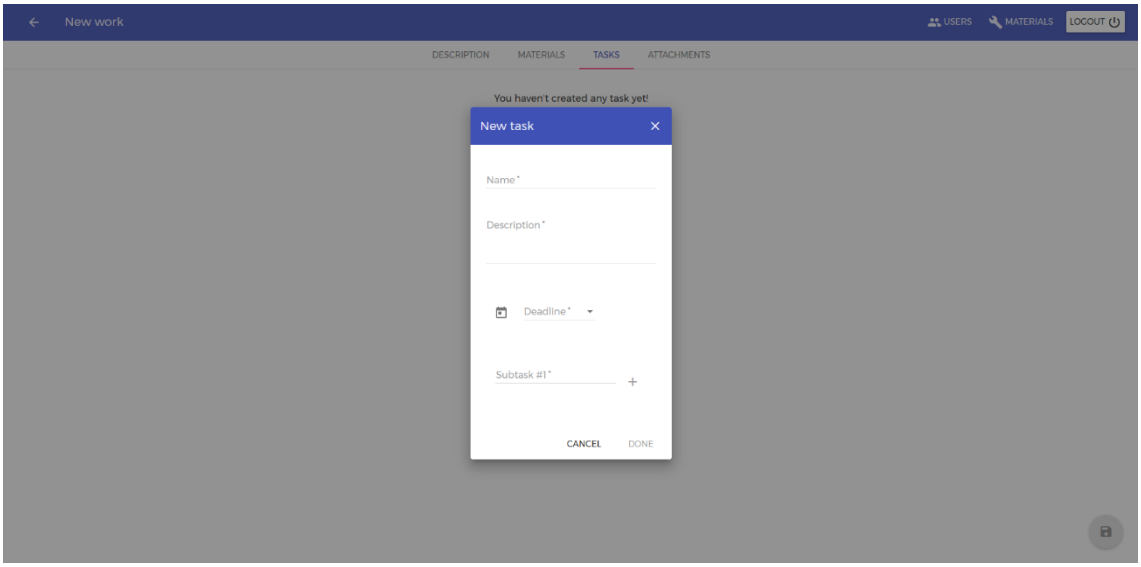

Ilustración 30. Creación de una obra en el panel de administración (añadiendo una tarea)

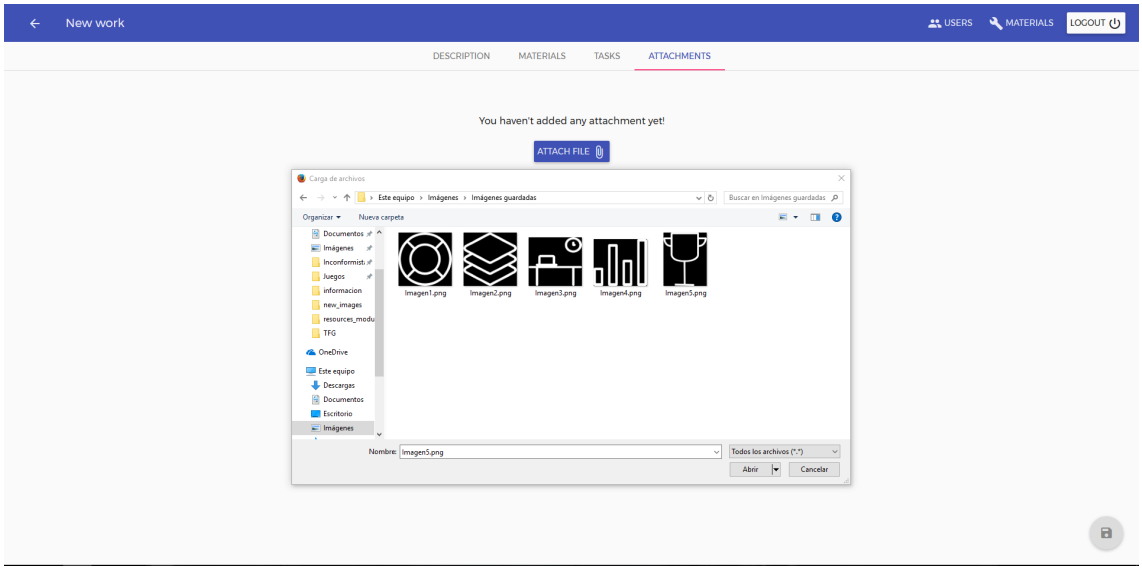

Ilustración 31. Creación de una obra en el panel de administración (añadiendo archivos adjuntos)

En la pantalla *users* podemos consultar los usuarios actuales del sistema y hacer búsquedas a través del buscador de un usuario en concreto. A través de esta pantalla también podemos editar los usuarios actuales del sistema.

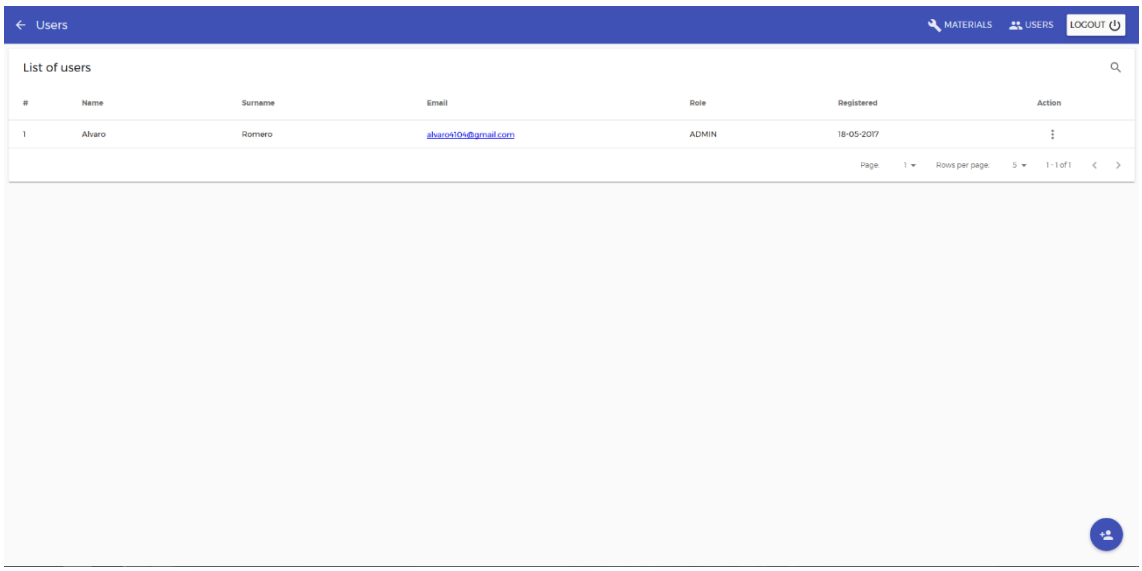

Ilustración 32. Vista de la pantalla users

En la pantalla *materials* podemos consultar los materiales actuales del sistema. A través de esta pantalla también podemos editar los usuarios materiales del sistema y añadir nuevos al sistema.

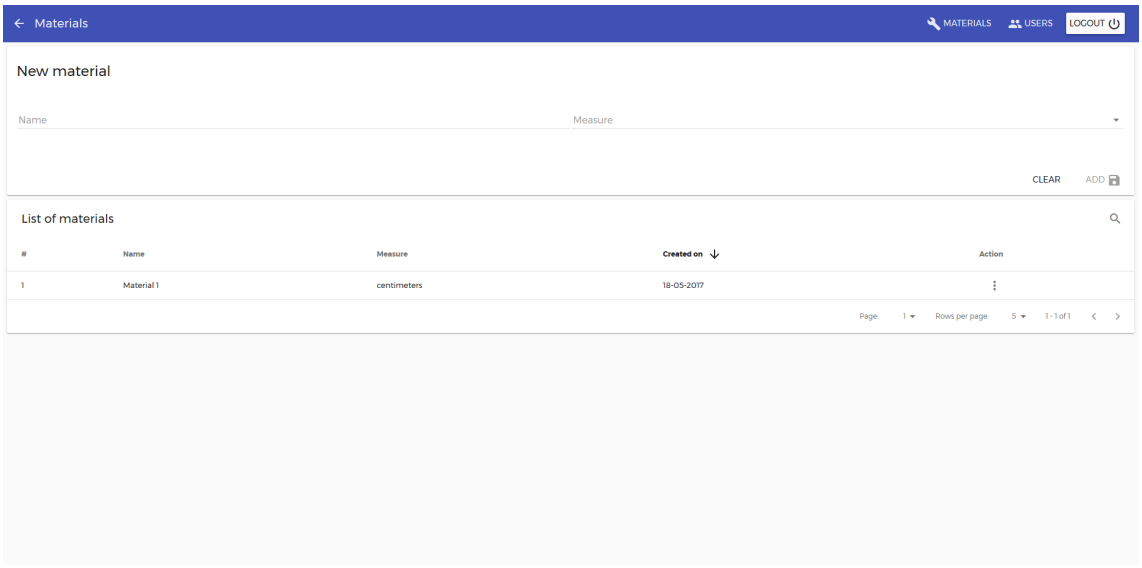

Ilustración 33. Vista de la pantalla materials

A través del *dashboard* podemos acceder a la pantalla de edición de una obra seleccionándola a través del toolbar a la derecha de cada una. En esta vista podemos volver a rellenar los campos que quedaron incompletos en el momento de creación de la obra o cambiarlos a nuestro antojo, para ello sólo tenemos que recordar guardar la obra antes de salir mediante el icono flotante de abajo derecha.

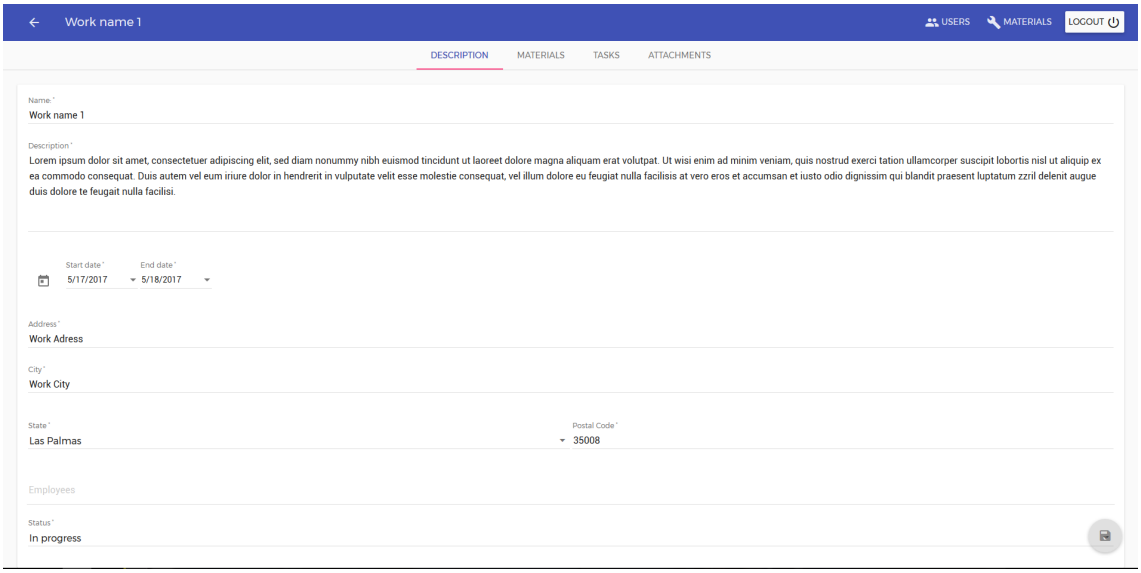

Ilustración 34. Vista de edición de una obra

5.3. Anexo 3. Manual de usuario de la aplicación móvil

Este manual detalla el uso general así como las distintas funcionalidades de la aplicación móvil.

A continuación se muestra la pantalla inicial que se muestra al abrir la aplicación en la cual se piden las credenciales de acceso a la misma.

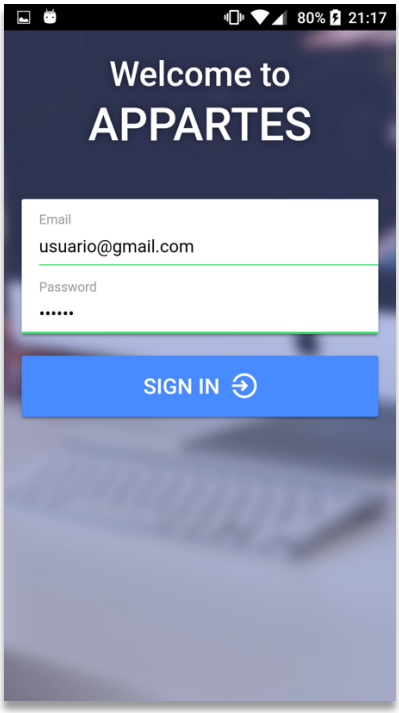

Ilustración 35. Inicio de sesión en de la aplicación

Tras acceder con un usuario previamente creado en el panel de administración se presenta la pantalla principal (*dashboard*) de la aplicación en la que se muestra un listado de las distintas obras en las que un usuario tiene tareas u obras directamente asignadas. Este listado además, está clasificado en pestañas en función del estado de la obra e incluso permite filtrar las obras a través buscador.

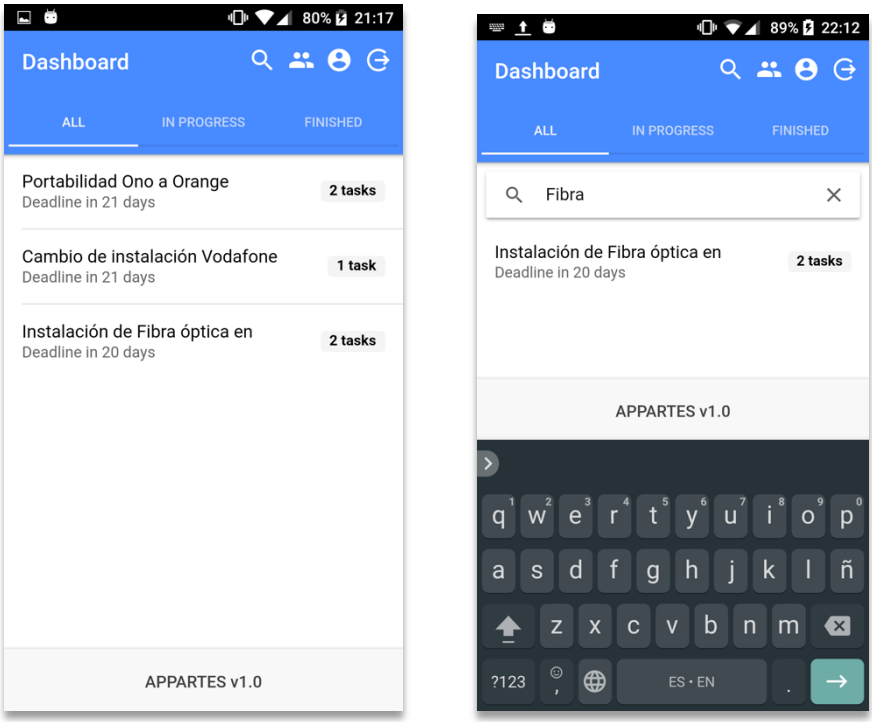

Ilustración 36. Vistas de la pantalla de principal de la aplicación (dashboard)

Al pulsar sobra una de las obras del listado se abrirá la pantalla del detalle de la obra correspondiente.

En esta pantalla se presentan además otras dos pestañas además de la descripción general de la obra, como son las pestañas de Tareas (*Tasks*) y Adjuntos (*Attachments*),

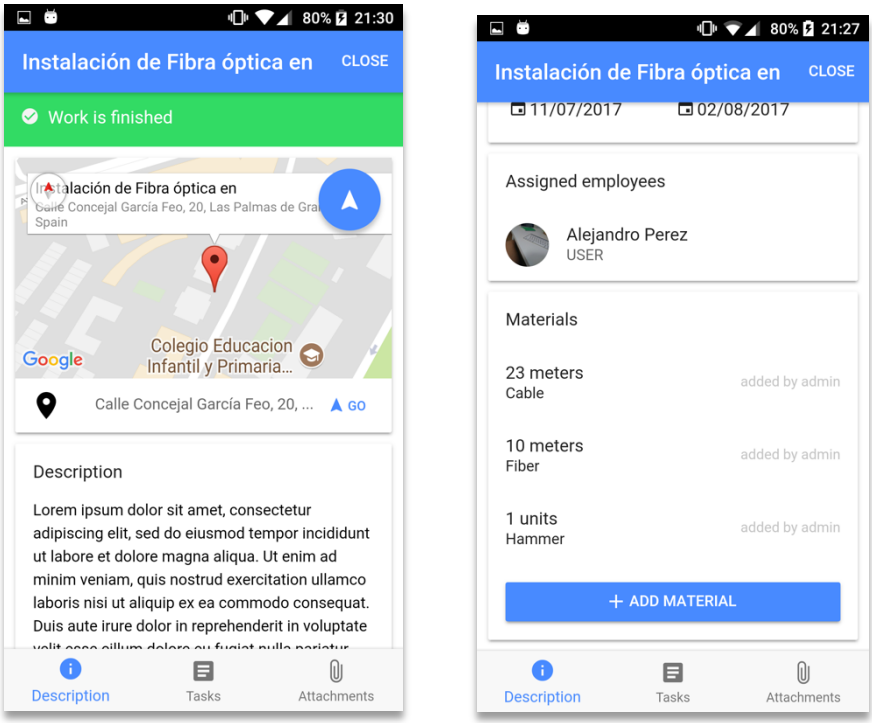

Ilustración 37. Pantalla del detalle de una obra

Para añadir materiales a una obra, basta con pulsar el botón “Add Material”, que abrirá una ventana de edición de materiales que una vez hechas las modificaciones pueden guardarse los cambios pulsando en el botón de guardar en la parte superior derecha.

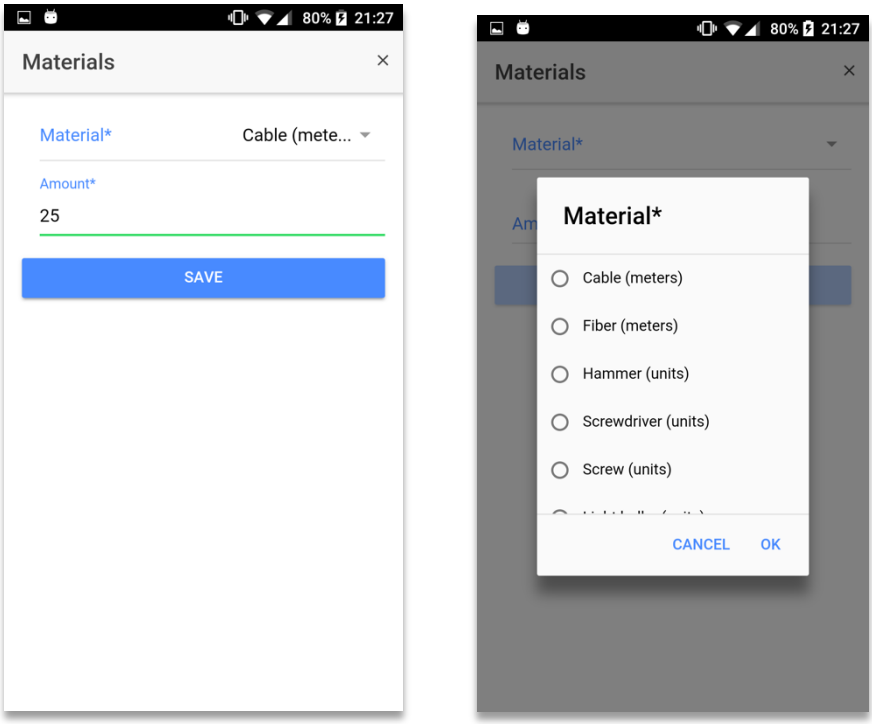

Ilustración 38. Añadir materiales a una obra desde la app

Los materiales pueden ser editados/eliminados únicamente por el usuario que los crea, para ello, hay que deslizar el dedo de derecha a izquierda sobre la tarea a editar como se muestra a continuación para que aparezcan estas opciones.

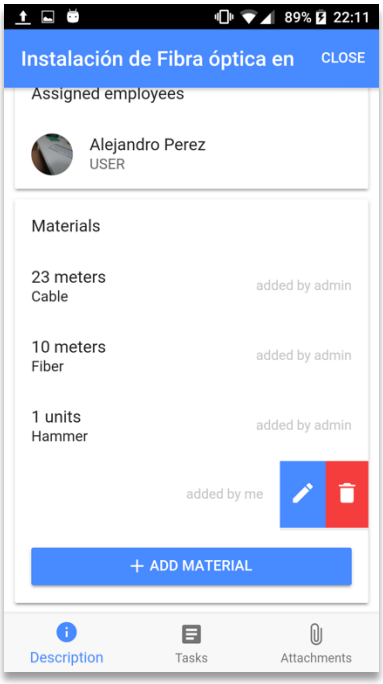

Ilustración 39. Editar materiales de la obra material desde la app

Al pulsar sobre la pestaña tareas en el detalle de una obra, se mostrará el listado de las distintas tareas que hay creadas en la obra. Al pulsar sobre ellas se abrirá otra pantalla con información sobre la misma.

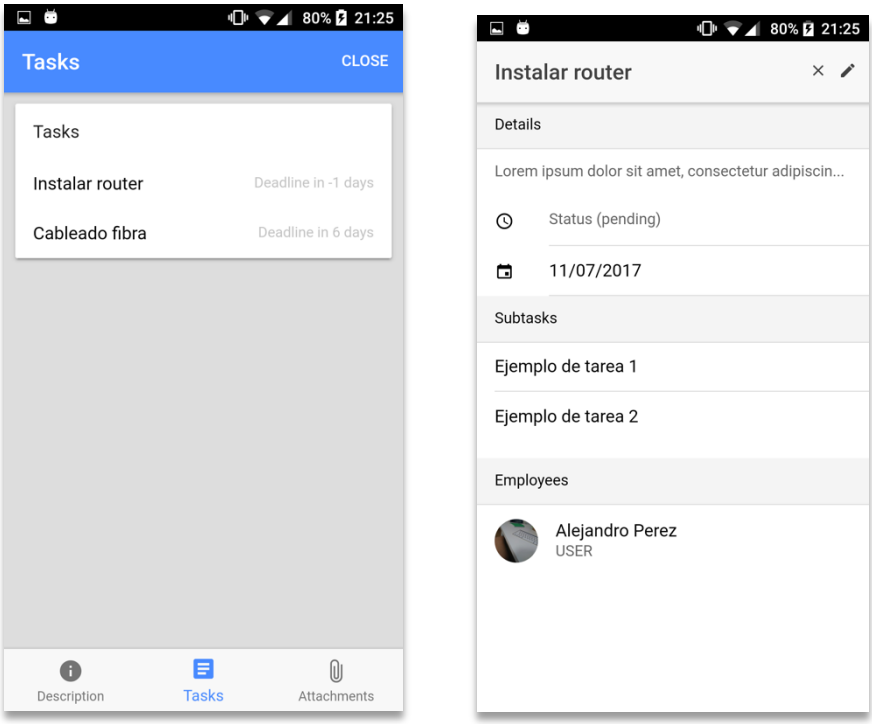

Ilustración 40. Vistas del las pantallas de listado de tareas y detalle de una tarea

Las tareas sólo pueden ser editadas por los usuarios que estén asignados a la misma, para hacerlo, hay que pulsar sobre el icono de edición junto al título de la tarea.

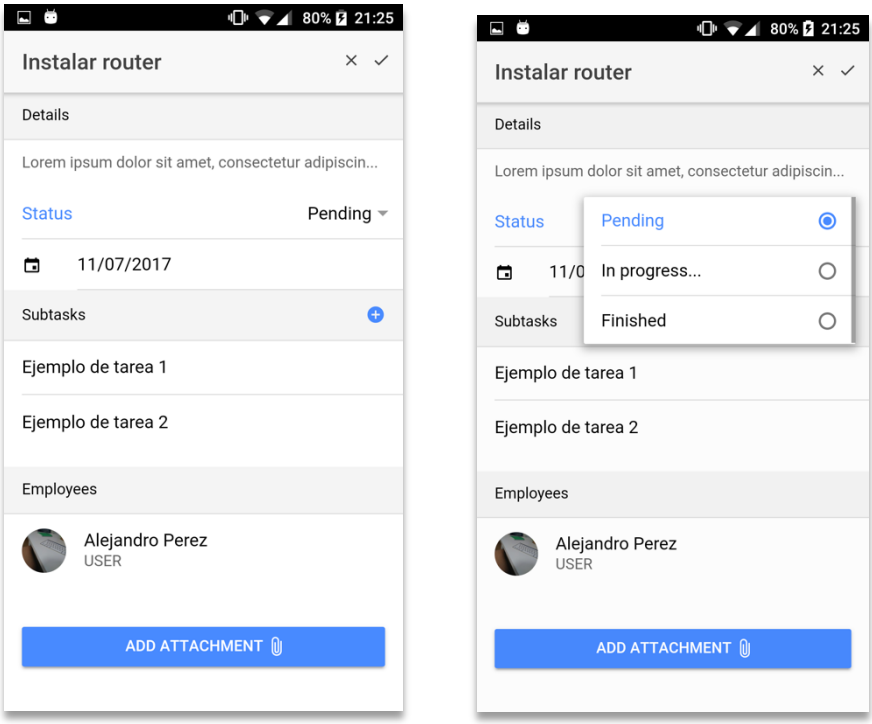

Ilustración 41. Edición de una tarea en la app

A la tarea se pueden adjuntar imágenes y archivos presentes en el dispositivo del empleado y pueden añadirse pulsando el botón “Add attachment”.

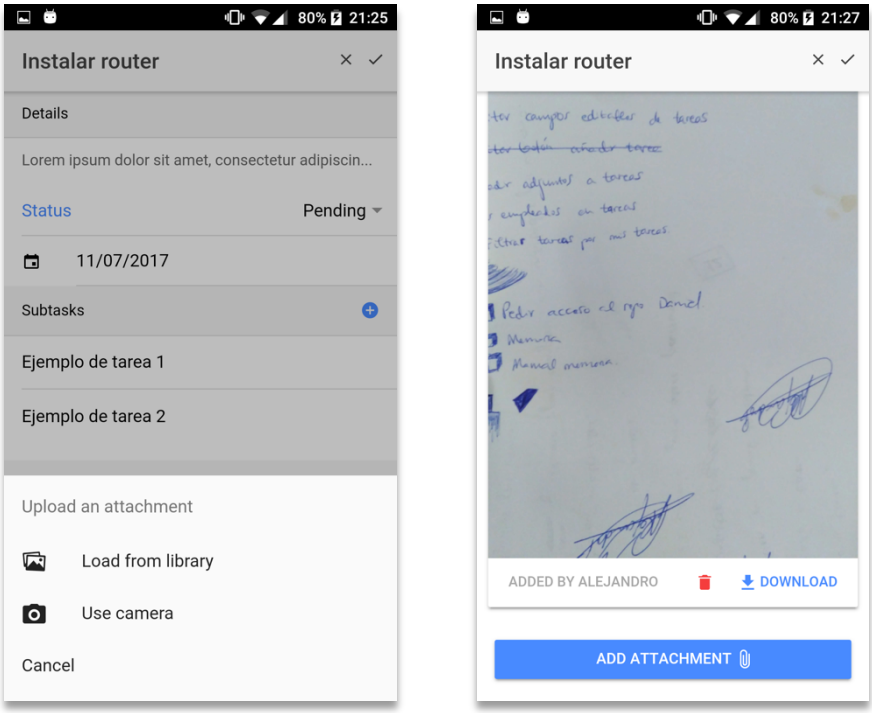

Ilustración 42. Añadir adjuntos a una tarea

La última pestaña del detalle de una obra es la de “Attachments”, en la que se encuentran los documentos adjuntos agregados por el usuario administrador. Estos archivos pueden descargarse pulsando el botón “Download” que se encuentra bajo cada adjunto.

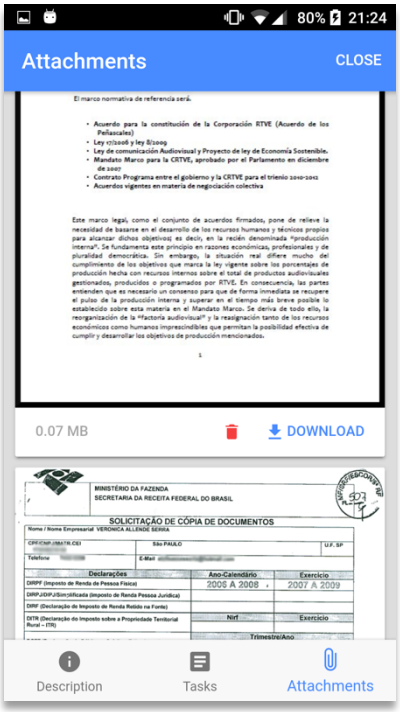

Ilustración 43. Captura de la pantalla de “Attachments”

Otra de las opciones disponibles es el listado de usuarios de la aplicación, muy útil para el empleado en caso de que necesite el teléfono u otra información de contacto de algún compañero. Se puede acceder pulsando el icono visible en la pantalla “dashboard”.

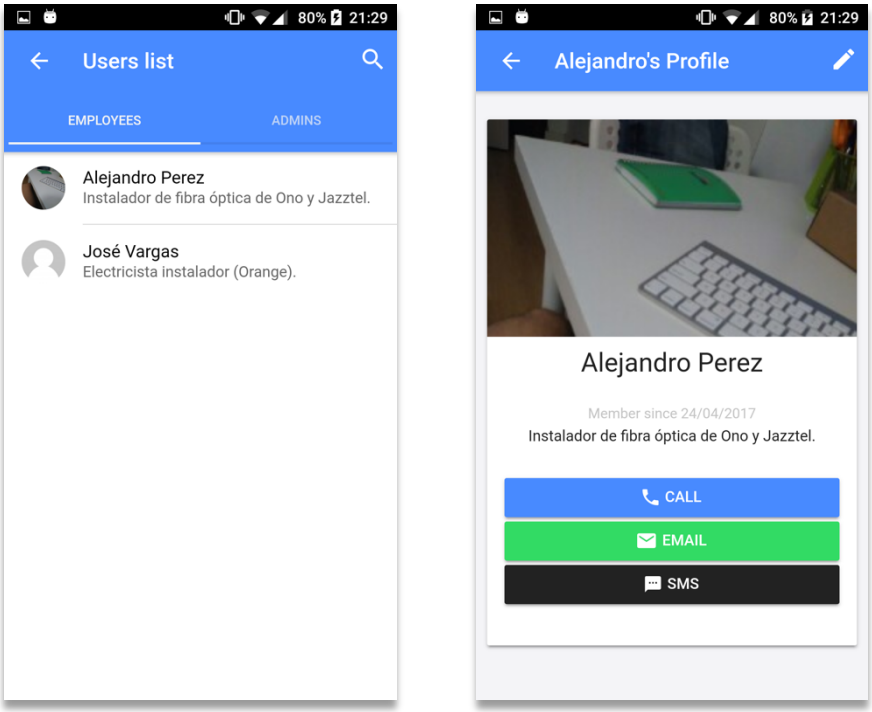

Ilustración 44. Listado de usuarios y vista del perfil de un usuario

Un usuario también puede acceder a su perfil pulsando el icono , donde será posible que edite sus datos personales e imagen de perfil.

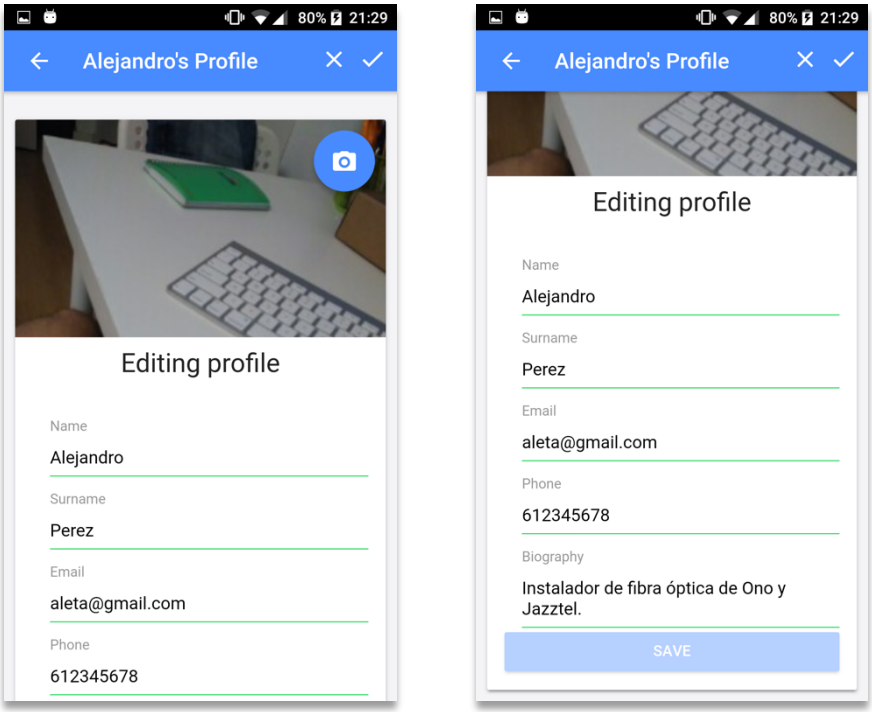

Ilustración 45. Edición del perfil del usuario

Por último, para salir de la aplicación es necesario pulsar el botón de salida que se encuentra en pantalla principal “dashboard”. Si se confirma la salida, el usuario será redirigido a la pantalla de inicio de sesión.

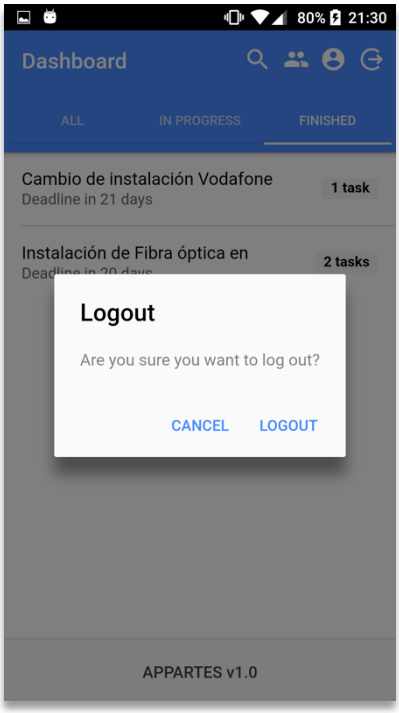

Ilustración 46. Vistas de la pantalla la pantalla de cierre de sesión

El envío de notificaciones push a la app se realiza desde la sección ‘Notificaciones’ de la consola de Firebase de Google, a la que se puede acceder desde este enlace: https://console.firebase.google.com/

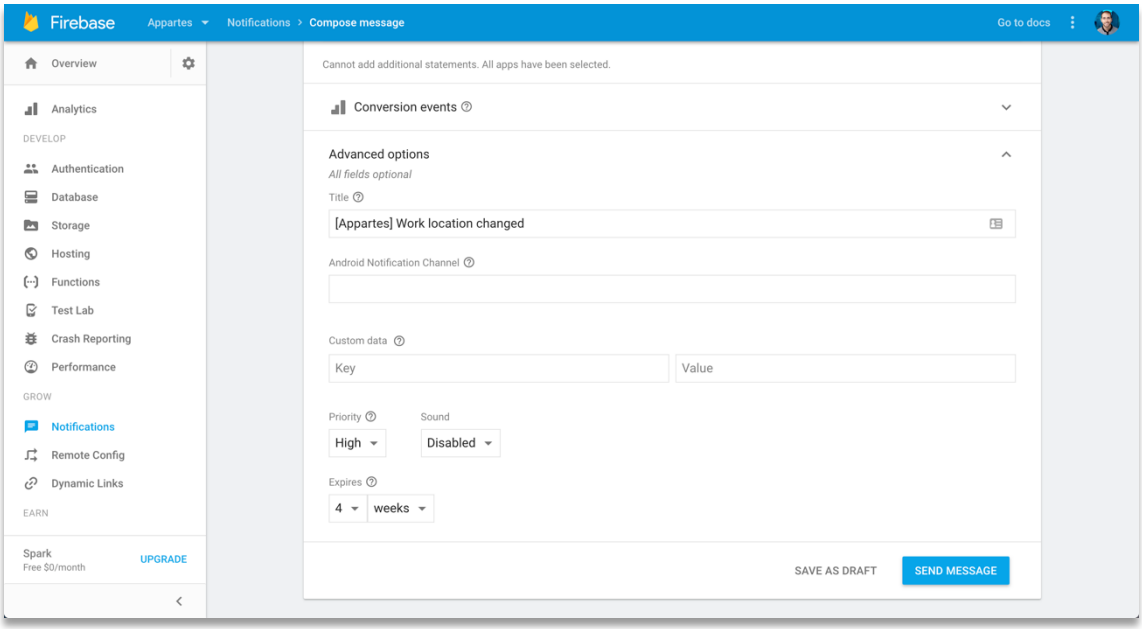

Ilustración 47. Envío de notificaciones Push desde la consola de Google Firebase

La forma en que dichas notificaciones se mostrarán en el dispositivo del empleado puede apreciarse en la siguiente ilustración:

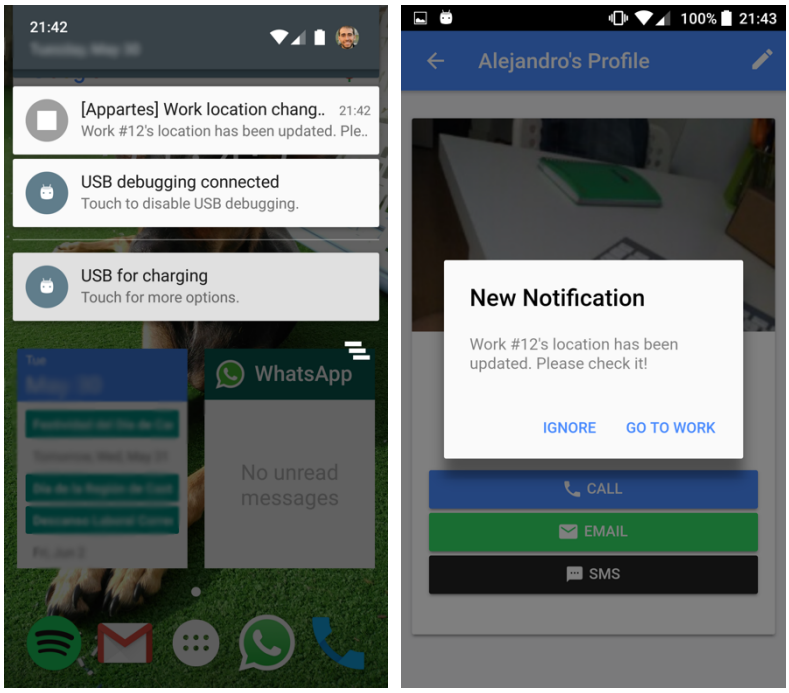

Ilustración 48. Vista de notificaciones push en el dispositivo del empleado

5.4. Anexo 4. Pantallazos completos de la demo no funciona

A continuación, mostramos capturas de pantallas que muestran la demo no funcional completa presentada en la segunda reunión durante la etapa de análisis.

Las primeras pantallas corresponden al inicio de sesión por parte del empleado dentro de la aplicación un desglose de las obras que tiene asignadas ordenadas por fecha.

Una vez ha seleccionado una pasaríamos a la tercera pantalla, donde puede obtener información relativa a las tareas de la obra (consideramos las tareas como aquello que desea ver primero un empleado al acceder a una obra).

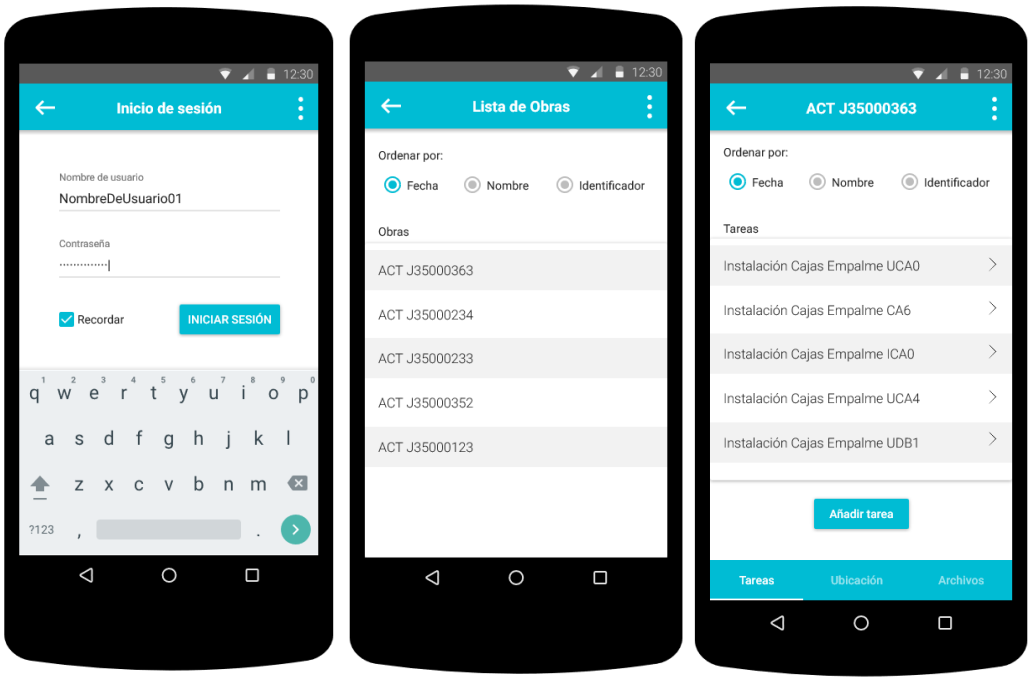

Ilustración 49. Primeras pantallas de la demo no funcional

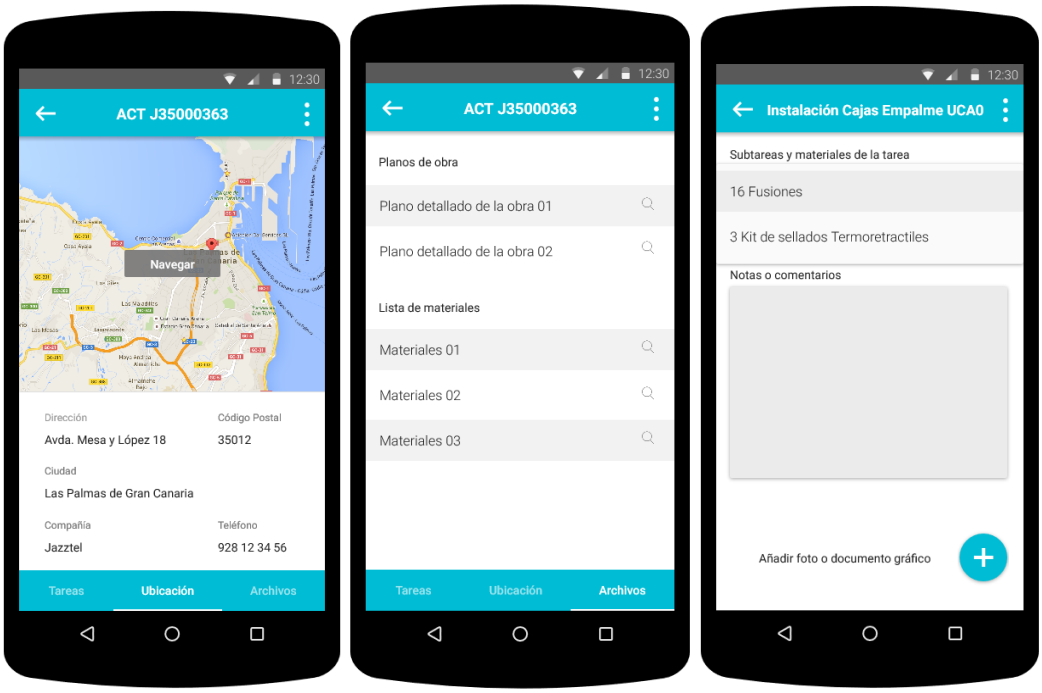

Ilustración 50. Información de la obra en la demo no funcional

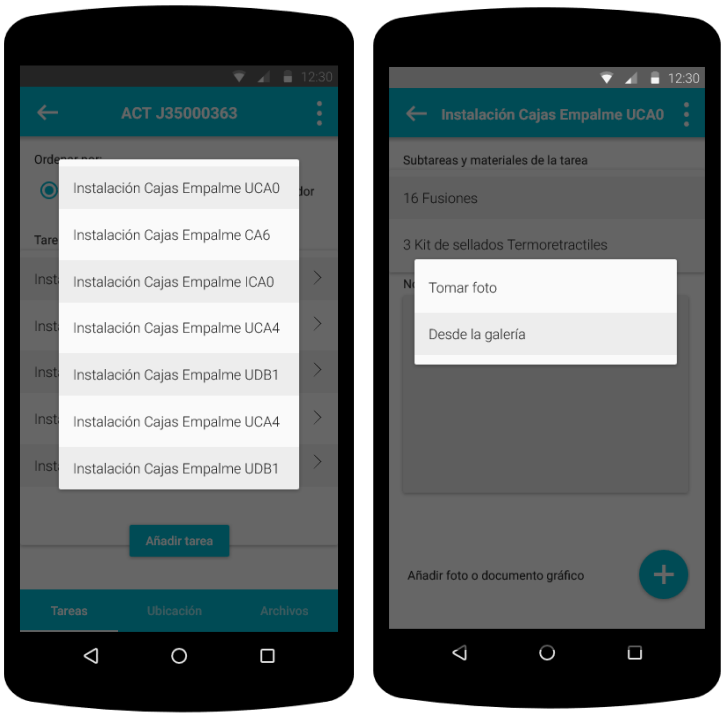

Ilustración 51. Acciones disponibles en una tarea en la demo no funcional

5.5. Anexo 5. Esquemas completos *MongoDB*

A continuación, se muestra la estructura que concebimos en un primer momento para la base de datos donde podemos ver las diferentes entidades que la conforman y los campos por los que están compuestos.

Obras (Works)

Field	Type	Description
id	String	Work identifier (autogenerated)
work_id	String	Work identifier
name	String	Work name
description	String	Work description
created_on	Date	Work creation date and time
created_by	Object<User>	Work creator identifier
last_update	Date	Work latest update
deadline	Date	Work deadline
address	String	Work address
city	String	Work city
postal_code	Integer	Work postal code
location	Object<String - *Longitude, Latitude*>	Work location
tasks	Array<Tasks>	Work tasks list
equipment	Array<Equipment>	Work equipment list
attachments	Object<String - *Filepath*>	Work attachments
employees	Array<Employees>	Work assigned employees
state	String	Finished/In progress/Pending/Cancelled

Tabla 38. Especificación de la entidad Obras en la base de datos

Tareas (Tasks)

Field	Type	Description
id	String	Task identifier
name	String	Get all current task
description	String	Get info from a certain task
last_update	Date	Task latest update
deadline	Date	Task deadline date
state	String	Finished/In progress/Pending/Cancelled

Tabla 39. Especificación de la entidad Tareas en la base de datos

Material (Equipment)

Field	Type	Description
id	String	Task identifier
name	String	Get all current task
description	String	Get info from a certain task

Tabla 40. Especificación de la entidad Material en la base de datos

Usuarios (Users)

Field	Type	Description
id	String	User identifier
name	String	User name
full_name	String	User fullname
role	String	Employer/Employee/Admin
bio	String	User biography
phone	Integer	User phone number
email	String	User email (for login)
password	String	Encoded password
salt	String	Password salt

Tabla 41. Especificación de la entidad Usuarios en la base de datos

5.6. Anexo 5. Documentación de la *API REST*

A continuación, se describen las diferentes operaciones *pull* y *push* a la *API REST* y la respuesta dada por ella en cada caso.

Autenticación (Authentication)

Ruta	Método	Descripción
/api/login	POST	Autentica un usuario en el sistema
/api/register	POST	Registra un nuevo usuario en el sistema

Tabla 42. Documentación de acceso a la entidad Autenticación

Obras (Works)

Ruta	Método	Descripción
/api/works	GET	Obtiene todas las obras
/api/works	POST	Crea una obra en la base de datos
/api/works	DELETE	Elimina todas las obras de la base de datos
/api/works/:id	GET	Obtiene la obra con el ID indicado
/api/works/:id	PUT	Actualiza los datos de una obra
/api/works/:id	DELETE	Elimina la obra con el ID indicado
/api/works/user/:id	GET	Obtiene las obras asociadas a un empleado

Tabla 43. Documentación de acceso a la entidad Obras

Materiales (Materials)

Ruta	Método	Descripción
/api/materials	GET	Obtiene todos los materiales existentes
/api/materials	POST	Crea un nuevo material en la base de datos
/api/materials/:id	GET	Obtiene el material con el ID indicado
/api/materials/:id	PUT	Actualiza los datos de un material
/api/materials/:id	DELETE	Elimina el material con el ID indicado
/api/materials/search	POST	Obtiene los materiales filtrados por el nombre indicado

Tabla 44. Documentación de acceso a la entidad Materiales

Usuarios (Users)

Ruta	Método	Descripción
/api/users	GET	Obtiene el listado de usuarios
/api/users	POST	Crea un nuevo usuario en la base de datos
/api/users/:id	GET	Obtiene el usuario con el ID indicado
/api/users/:id	PUT	Actualiza los datos de un usuario
/api/users/:id	DELETE	Elimina el usuario con el ID indicado
/api/users/search	POST	Obtiene los usuarios filtrados por el nombre indicado

Tabla 45. Documentación de acceso a la entidad Usuarios

Subidas (Uploads)

Ruta	Método	Descripción
/api/upload	POST	Sube un archivo al sistema
/api/upload	DELETE	Elimina un archivo de las subidas del sistema

Tabla 46. Documentación de acceso a la entidad Subidas en la API REST