Universidad de La Laguna
Escuela Superior de Ingeniería y Tecnología

Trabajo de Fin de Grado

Aplicación para ayudar en la búsqueda de personas y mascotas desaparecidas

Application to help in the search for missing people and pets

Alex Jimmy Montaño Fuentes

La Laguna, 4 de septiembre de 2018

D. Pino Caballero Gil, con N.I.F. 45.534.310-Z Catedrática de Universidad adscrita al Departamento de Ingeniería Informática y Sistemas de la Universidad de La Laguna, como tutora.

D. Alexandra Rivero García, con N.I.F. 78.646.309-V Investigadora FPI adscrita al Departamento de Ingeniería Informática y Sistemas de la Universidad de La Laguna, como cotutora.

C E R T I F I C A (N)

Que la presente memoria titulada:

"Aplicación para ayudar en la búsqueda de personas y mascotas desaparecidas"

ha sido realizada bajo su dirección por D. Alex Jimmy Montaño Fuentes, con N.I.F 42293090-T.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 15 de mayo de 2018.

Agradecimientos

A mi familia, por estar siempre ahí y darme su apoyo pese a todo.

A mis profesoras Pino Caballero y Alexandra Rivero, por haber haberme permitido realizar este proyecto, por vuestra paciencia conmigo y por la ayuda recibida.

A todos los profesores del grado de los que he tenido la oportunidad de aprender.

Licencia

Resumen

Desde que en 2010, entrara en funcionamiento el Sistema de Personas Desaparecidas y Restos Humanos Sin Identificar (PDyRH) hasta el 31 de diciembre de 2017, se registraron en España un total de 146.042 denuncias de desaparición de personas, de las cuales 6.053 siguen activas.

El presente trabajo de fin de grado plantea el desarrollo de una aplicación que ayude a disminuir el número de desapariciones activas que cada año se suman a las ya presentes, facilitando para ello, la difusión de las desapariciones, mediante su compartición en las redes sociales por los usuarios, y el envío de notificaciones. Se trata de una aplicación compuesta por tres componentes principales; API y las aplicaciones web y móvil, implementadas mediante la tecnología MEAN; MongoDB para la base de datos, Node.js junto a Express para la implementación de la API, Angular para el desarrollo de la aplicación Web y, adicionalmente, Ionic para la aplicación móvil.

Palabras clave: Desapariciones, MEAN, Ionic, API, Nginx

Abstract

Since 2010, the System of Missing Persons and Unidentified Human Remains (PDYRH) became operational until December 31, 2017, a total of 146,042 reports of missing persons were registered in Spain, of which 6,053 are still active.

The present end-of-degree project proposes the development of an application that helps reduce the number of active disappearances that are added each year to those already present, facilitating the dissemination of the disappearances through their sharing on social networks by users, and sending notifications. The application is composed of three main components; API and web and mobile applications, implemented through MEAN technology; MongoDB for the database, Node.js with Express for the implementation of the API, Angular for the development of the Web application and, additionally, Ionic for the mobile application.

Keywords: Disappearances, MEAN, Ionic, API, Nginx

Índice general

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

1.1 Motivación

En 2017 se registraron un total de 24.794 denuncias, en España, de las cuales 2.928 (11,81%) siguen estando activas al no haberse resuelto la desaparición, según datos emitidos por el Sistema de Personas Desaparecidas y Restos Humanos Sin Identificar (PDyRH), dependiente del Ministerio del Interior [1].

El PDyRH se puso en funcionamiento en el año 2010 con el objetivo de contabilizar los datos registrados por las Fuerzas y Cuerpos de Seguridad del Estado en cuanto a desapariciones se refiere. Hasta el 31 de diciembre de 2017 se registraron en España un total de 146.042 denuncias por pérdida de personas desde la creación de este organismo. Del total de estas denuncias, en la actualidad se encuentran 6.053 casos sin resolver, lo que supone un 4,14% del total de las registradas.

Pese a la alarmante cifra de desapariciones registradas, solo contadas de ellas son difundidas mediante los medios de comunicación. Sin embargo mediante estas difusiones se ha podido comprobar la solidaridad de las personas para estos casos, como se señala en el artículo [2].

Por otro lado, con respecto a los datos de animales desaparecidos, no se han encontrado cifras oficiales, sin embargo, las aplicaciones existentes para tal fin, junto con los carteles de desaparición que sin duda cualquiera puede ver, hacen suponer que se tratan de cifras aún mayores.

A todo esto, ¿cómo se debería actuar ante una desaparición?. La respuesta reflejada en todos los artículos que responden a esta pregunta se puede resumir en la siguiente: Una rápida actuación es importante para acelerar la búsqueda, al igual que una amplia difusión de la noticia; dicho de otra forma, hay que actuar rápido y además, haciendo un escándalo.

Con todo ello, el presente proyecto busca servir de nexo entre dos componentes fundamentales para estos casos: tecnologías para difundir información, las cuales están más que nunca presentes en nuestra sociedad; la solidaridad de las personas, que éstas han demostrado tener en estos casos. La principal motivación para su desarrollo es que la aplicación resultante de este proyecto, pueda servir de ayuda, para que el número de las desapariciones activas que cada año se suman a las ya presentes sean nulas; para que toda persona o mascota desaparecida pueda ser encontrada.

1.2 Objetivo

El principal objetivo de este proyecto es desarrollar una aplicación que sirva como herramienta de difusión de desapariciones, ya se traten de personas o mascotas, mediante el uso de las redes sociales y envió de notificaciones. Para ello el proyecto estará compuesto por tres componentes principales: API, Aplicación web y aplicación móvil. Dichas componentes tienen sus objetivos específicos.

API

El principal objetivo de la API, es garantizar la seguridad e integridad de la información almacenada en la base de datos, así como la correcta ejecución de los procesos de la aplicación, fundamentalmente de aquellas relacionadas con el registro y difusión de desapariciones.

Aplicaciones web y móvil

El principal objetivo de estos componentes se basa en ofrecer un registro de desapariciones sencillo e intuitivo. De igual forma, es muy importante que el diseño de las desapariciones sea un diseño enfocado en la información a mostrar, y que permita su fácil compartición.

En general, ambas componentes deben ofrecer una interfaz minimalista, usable y accesible.

1.3 Fases del desarrollo

El proyecto ha sido desarrollado en las siguientes seis fases.

En la primera etapa se realizó un análisis de los requisitos de la aplicación, para lo cual se definió el comportamiento y funcionalidades que ésta tendría según el tipo de usuario.

A partir de los requisitos obtenidos, en la segunda etapa, se creó una serie diagramas de actividades con el fin de facilitar la implementación de aquellas funcionalidades de la API que se preveían más complejas. Además de ello, se creó el diseño de la base de datos de datos y el diseño de la interfaz de usuario en dos resoluciones diferentes (móvil y ordenador).

En la tercera etapa, se configuró e instaló el gestor de base de datos MongoDB, en un servidor virtual privado (VPS), realizando las configuraciones necesarias para acceder a ella mediante autenticación y de forma remota. Posteriormente se realizó la implementación de la API, para lo cual se usó Node.js junto a Express. Con el propósito de que la API fuese fácilmente mantenible, ante mejoras en el código e incorporación de nuevas funcionalidades, se implementaron tests que validarán la correcta ejecución de las funcionalidades.

En cuarta etapa, una vez la API fue implementada, se procedió a implementar la aplicación web, utilizando para ello Angular 6. Durante esta etapa fue necesario leer mucha documentación, debido a los conocimientos limitados que se tenía del desarrollo con Angular. Además, se modificó varias veces el diseño de la interfaz de usuario, con el fin de ofrecer un diseño minimalista y usable. Como resultado se obtuvo una aplicación web progresiva, un micro framework CSS y se inició el desarrollo de un módulo para Angular.

Durante la quinta etapa se procedió a desplegar en producción los componentes desarrollados. Para el caso de la API, su despliegue se realizó directamente en heroku; sin embargo, para el caso de la aplicación web supuso probar múltiples plataformas de despliegue (Surge.sh, Heroku, Gitlab Pages…), como se verá posteriormente. El último y definitivo, implicó contratar y configurar otro VPS con Nginx. Gracias a Nginx, ambos componentes fueron fácilmente desplegados en el mismo servidor, y además, accesibles desde dominios diferentes; quedemi.com para la aplicación web, y api.quedemi.com para la API.

La sexta etapa fue causada por problemas en el funcionamiento de la aplicación web en dispositivos móviles, para lo cual se decidió crear una aplicación móvil con el uso de Ionic, lo que supuso aprender sobre este framework.

1.4 Estructura de la memoria

El contenido de la memoria está dividido en los siguientes 9 capítulos.

1. Introducción: Se realiza una introducción al proyecto, en donde se exponen los motivos que impulsaron el desarrollo del proyecto, los objetivos a cumplir y las distintas fases del proyecto.
2. Introducción a la aplicación: Se explica el funcionamiento de la aplicación desarrollada, las partes que la componen y cómo éstas se conectan entre sí. Igualmente se realizará una breve descripción de aplicaciones encontradas, similares a la de este proyecto.
3. Tecnologías y herramientas: En este capítulo se describe las diferentes tecnologías y herramientas utilizadas en la desarrollo del presente proyecto.
4. Análisis funcional y diseño técnico: Se exponen las tareas que se realizaron antes de empezar con la implementación de los componentes del proyecto, con el fin de facilitar dicha implementación.
5. Servidor: En este capítulo se describe el proceso de instalación y configuración de la base de datos; al igual que el proceso de implementación de la API, y los problemas encontrados durante este último.

6. Aplicaciones web y móvil: Se describe las correspondientes implementaciones de cada aplicación, y adicionalmente para la aplicación web, los problemas encontrados dura dicho proceso.
7. Despliegue: En este capítulo se hablará sobre el proceso de despliegue de la API y aplicación web, así como de las configuraciones que fueron requeridas para éste.
8. Presupuesto. Se analiza el presupuesto total del proyecto; desde los costes de desarrollo, hasta los costes, mensual y anual, que supondría su mantenimiento.
9. Conclusiones y trabajo futuro. Se exponen las conclusiones, mejoras y nuevas funcionalidades que serán añadidas a la aplicación.

Capítulo 2. Introducción a la aplicación

2.1 Definición

El presente proyecto pretende ayudar en la difusión de desapariciones mediante el uso de las redes sociales y envío de notificaciones.

Está formado por tres componentes principales; una API REST, una aplicación web y otra móvil.

Para poder realizar la difusión de una desaparición, debe ser creada desde la aplicación web o móvil por un usuario registrado. La desaparición puede crearse para una persona o mascota desaparecida y para ello será necesario especificar una serie de datos como:

- El nombre de la persona o mascota desaparecida.
- La fecha de desaparición.
- Posibles ubicaciones.
- Imágenes.
- Datos de contacto.

Desde el momento de su creación, la desaparición puede ser compartida en la redes sociales Facebook, Whatsapp y Twitter. Cualquier persona que vea la desaparición podrá realizar dicha acción.

Los datos de la desaparición no podrán ser modificados salvo los datos de contacto; como es lógico, dicha modificación podrá ser llevada a cabo sólo por el usuario que la creó.

Con respecto a los usuarios, pueden registrarse e iniciar sesión en la aplicación mediante su email y contraseña, o mediante las aplicaciones externas google o facebook; los que se registren mediante correo electrónico deberán realizar la verificación del mismo para poder acceder a su cuenta. Para ello, al momento de su registro, se les enviará un enlace de verificación al correo electrónico indicado. En caso que el usuario intente iniciar sesión sin haber realizado dicha verificación, se le avisará sobre ello y ofrecerá la oportunidad de reenviar el enlace de verificación.

Los usuarios una vez registrados y autenticados, tendrán que especificar una ubicación a partir de la cual se buscarán desapariciones cercanas. Al momento que se especifique la ubicación, si el navegador lo soporta, se realizará una petición de suscripción a notificaciones push; aparte de estas notificaciones el usuario podrá suscribirse a recibir

correos electrónicos. Entonces, cuando se cree una desaparición con alguna ubicación cercana a la del usuario, se le enviará la respectiva notificación y correo electrónico con información de dicha desaparición.

Adicionalmente, los usuarios registrados podrán:

- Editar sus datos: nombre, email, contraseña y ubicación. En el caso de cambiar su correo electrónico y si el usuario puede autenticarse mediante contraseña, deberá verificar el nuevo correo electrónico.
- En caso de tratarse de un usuario que sólo puede autenticarse mediante aplicaciones externas, podrá habilitar la autenticación mediante contraseña. Para ello simplemente deberá establecerla.

En el caso que un usuario no recuerde su contraseña, podrá restablecerla; para ello debe especificar su correo electrónico a la que se enviará un enlace, la cual le llevará a la página donde establecerá su nueva contraseña.

2.2 Conceptualización

En este apartado se describirán las diferentes partes que componen el sistema propuesto, y cómo estas están conectadas entre sí.

Figura 1: Arquitectura del sistema

Como se puede ver en la Figura 1, el proyecto consta de de cuatro cuatro componentes: La aplicación web, la aplicación móvil, la API REST y la base de datos. Estos se comunican de la siguiente forma: La aplicación web y móvil realizan las peticiones HTTP a la API REST, donde ésta, interactuando con la base de datos, procesa dichas peticiones y resuelve la respectiva respuesta de éxito o error.

Inicialmente, la aplicación web iba a ser el único componente implementado para trabajar en el lado del cliente, pero debido a problemas encontrados durante su desarrollo en dispositivos móviles, las cuales se comentarán posteriormente, se decidió desarrollar la aplicación móvil. Sin embargo, por falta de tiempo, no se pudo realizar la implementación de todas las funcionalidades previstas. Ambas aplicaciones proveen una interfaz minimalista y, en el caso de la aplicación web, diseñada desde sus inicios para dispositivos móviles, siguiendo la técnica mobile first.

La API, de sus siglas en inglés Application Programming Interfaces, constituye un servicio web RESTful, implementado con Node.js y Express, que procesa las peticiones HTTP recibidas desde las aplicaciones web y móvil mediante los métodos HTTP GET, POST, DELETE Y PUT y protocolo SSL, aceptando también HTTP/2. Ambos componentes, API y base de datos, se encuentran en la nube, ejecutándose en servidores diferentes. Cada petición HTTP que procese, debe contener toda la información necesaria para ello.

Para el caso de peticiones que necesiten autenticación, se proveerá un Bearer Token, con el token de acceso del usuario, en el header Authorization de la petición. Este token de acceso se trata de información cifrada que identifica al usuario de forma unívoca en la base de datos y que expira al cabo de un tiempo. Para generar dicho token se utilizará JSON Web Token (JWT).

2.3 Propuestas similares

A pesar que existen numerosas propuestas similares a la de este proyecto, todas las estudiadas se orientan al registro de desapariciones de mascotas o de personas, de forma exclusiva.

2.3.1 Orientados a mascotas

Wizapet

Wizapet [3] es una aplicación móvil disponible para IOS y Android, en la que los usuarios registrados pueden crear tanto anuncios de desapariciones de mascotas, como anuncios sobre animales encontrados o para su adopción. A la creación de estos anuncios se alertará a los usuarios cercanos a la ubicación indicada mediante notificaciones push. Dichas desapariciones pueden ser compartidas por los medios que el dispositivo móvil provee. La aplicación tambien provee un chat por el cual es posible establecer una comunicación directa con el creador del anuncio, además permite el pago de recompensas.

Zoocan

Zoocan [4] es la aplicación informática del Registro Canario de Identificación Animal que permite a los veterinarios de las islas la correcta identificación de los animales. Ésta está

disponible como aplicación web y móvil. Los datos se registran a través de un microchip implantado por un veterinario en el animal, y la información relativa a esta identificación (datos del animal, del propietario y del veterinario) se introducen en Zoocan para permitir que el propietario la pueda comprobar en todo momento. Además, Zoocan ofrece información detallada de todas las consultas, clínicas y hospitales veterinarios de Canarias y de forma adicional, en su plataforma web, la posibilidad de notificar la pérdida de una mascota y un listado de animales perdidos.

Animales-Perdidos.org

Animales-Perdidos.org [5] es una plataforma web que permite registrar la desaparición de mascotas así como alertar que se ha encontrado un animal, acceder a un listado de los anuncios y realizar búsquedas sobre éstos.

perrosperdidos.es

perrosperdidos.es [6] es una aplicación web muy similar a la aplicación anterior, sin embargo se diferencia en que sólo permite el registro de anuncios de perros, ya sea perdidos o encontrados.

2.3.2 Orientados a personas

sosdesaparecidos.es

sosdesaparecidos [7] es una asociación, sin ánimo de lucro, de ayuda y difusión de casos de personas desaparecidas. Para ello, es necesario ponerse en contacto con la asociación y haber realizado la denuncia de la desaparición. En su plataforma web se puede acceder a un listado de las desapariciones de cada comunidad autónoma de España.

Capítulo 3. Tecnologías y herramientas

En el presente capítulo se describen las tecnologías y herramientas utilizadas durante el desarrollo del proyecto.

3.1 De forma general

Javascript ES7

JavaScript [8] es un lenguaje de programación interpretado, considerado la más famosa implementación del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Puede ser utilizado tanto en el lado del cliente, como del servidor, gracias a Node.js.

En el proyecto se trabajará con la versión 7 de ECMAScript, cuyo nombre oficial es ECMAScript 2016. Ésta provee una serie de funcionalidades nuevas a javascript de las cuales destacan las funciones Async/Await; funciones que trabajan combinando las Promesas y los Generadores de ES6, y cuya finalidad es simplificar la implementación de código asíncrono de forma que se puede realizar como síncrono.

En los siguientes códigos de ejemplo, se puede ver las diferencias entre el uso de callback, una promesa y una función async, para implementar la misma funcionalidad.

CALLBACK

```
getUser(function(err, user){
  getProfile(user, function(){
    getAccount(profile,
function(err, acc){
      getReport(acc, function(err,
report){
        ...
      })
    })
  })
})
```

PROMESA

```
getUser()
  .then(getProfile)
  .then(getAccount)
  .then(getReport)
  ...
  .catch(function(err){
    console.error(err);
  });
```

ASYNC/AWAIT

```
async function asynFunction(){
  try {
    let user = await getUser();
    let profile = await getProfile(user);
    let account = await getAccount(profile);
    ...
  }
  catch(err){
    console.error(err);
  }
}
```

Gitlab

Gitlab [16] es un servicio web de control de versiones y desarrollo de software colaborativo basado en Git, similar a otros servicios como Github y Bitbucket.

Sketch App

Es una aplicación de escritorio disponible para macOs que sirve como herramienta de diseño vectorial y prototipado, especializada en el diseño de interfaces de aplicaciones web y móviles.

3.2 API

Node.js

Entorno JavaScript multiplataforma y de código abierto, que nos permite ejecutar en el servidor, de manera asíncrona, con una arquitectura orientada a eventos y basado en el motor V8 de Google. Dicho motor V8 compila Javascript en código máquina nativo consiguiendo así una alta velocidad.

Una de las principales características de Node es que para manejar las conexiones trabaja con eventos, evitando así un gran uso de la memoria y permitiendo que la escalabilidad de la herramienta no sea un problema, a diferencia de algunos servidores web, como por ejemplo Apache el cual crea un nuevo hilo por cada conexión.

Otra gran característica de Node es su gestor de paquetes NPM (Node Package Manager). Éste nos permite acceder a una enorme cantidad de librerías Open Source desarrolladas por su comunidad. Para ello NPM provee el comando npm con el cual podremos instalar los módulos, denominados dependencias, que nuestro proyecto requerirá.

Express

Express [9] es un framework de desarrollo de aplicaciones web minimalista y flexible para Node.js. Proporciona una fina capa de características de aplicaciones web básicas, además de varios métodos, utilidades HTTP, y middlewares que permiten crear una API sólida y perfectamente funcional de forma rápida.

Express es el módulo npm principal que usaremos para la implementacion de la API REST.

Postman

Postman [10] es una herramienta que permite realizar peticiones HTTP a cualquier API.

Surgió originariamente como una extensión para el navegador Google Chrome. Sin embargo, actualmente dispone de aplicaciones nativas para MAC, Windows y Linux.

Está compuesto por diferentes herramientas y utilidades gratuitas que permiten realizar diferentes tareas dentro del mundo API REST: creación de peticiones a APIs internas o de terceros, elaboración de tests para validar el comportamiento de APIs, posibilidad de crear entornos de trabajo diferentes, con variables globales y locales; y todo ello con la posibilidad de ser compartido con otros compañeros del equipo de manera gratuita, mediante la exportación de toda esta información mediante URL en formato JSON.

3.3 Base de datos

MongoDB

MongoDB [11] es una base de datos NoSQL, basada en documentos. Dichos documentos son agrupados en colecciones y almacenados en formato BSON (Binary JSON), el cual es una versión modificada de JSON con información adicional de tipo y que permite búsquedas rápidas de datos. Gracias a que la información se almacena de esta forma soporta todos los tipos de datos de JavaScript, incluyendo variables compuestas.

Es de esquema libre, lo que significa que los objetos de una colección pueden tener esquemas diferentes. Entre las principales características de MongoDB se encuentran la escalabilidad, el rendimiento y una gran disponibilidad.

En el proyecto no se usará MongoDB directamente, sino mediante su más famoso ODM, Mongoose, el cual simplifica el uso de MongoDB mediante la utilización de esquemas para modelar los datos que se desea almacenar y manipular en MongoDB. Mongoose además incluye características tales como conversión de tipo, validación, creación de consultas, posibilidad de añadir plugins para extender su funcionalidad, y más, dando como resultado un código más legible y mantenible. Por otro lado la principal desventaja de su uso es que la abstracción se produce a costa del rendimiento en comparación con el uso de MongoDB

nativo. Sin embargo, desde mi punto de vista, la serie de facilidades que ofrece supera con creces sus desventajas.

Amazon EC2

Amazon Elastic Compute Cloud [12], más conocido como Amazon EC2, es la solución de Amazon Web Services para ofrecer entornos informáticos virtuales en la nube, de capacidad y tamaño modificable. Se trata de máquinas virtuales, denominados instancias, en los que poder desarrollar, testear, desplegar y gestionar aplicaciones.

Para el proyecto se usará una de estas instancias para la instalación y configuración de la base de datos en MongoDB.

Amazon S3

Amazon S3 [13] es un servicio de almacenamiento simple de objetos creado para almacenar y recuperar cualquier volumen de datos con un nivel extremadamente alto de durabilidad, disponibilidad y escalabilidad.

Amazon S3 será utilizado en el proyecto como sistema de almacenamiento de imágenes, con lo cual en la base de datos se almacenarán sólo la correspondiente URL de cada imagen.

Robo 3T

Robo 3T [14], anteriormente conocido como RoboMongo, es una herramienta de administración gráfica para para bases de datos NoSQL, principalmente para MongoDB, opensource y multiplataforma, con el cual se puede acceder y manipular todas las colecciones y documentos de la base de datos.

3.4 Aplicación web

Angular

Angular es un framework de desarrollo de aplicación web en typescript de código abierto creado y mantenido por Google. La finalidad de Angular es facilitarnos el desarrollo de aplicaciones SPA (Single Page Application) ya sean web, móviles o de escritorio, y además darnos herramientas para trabajar con los elementos de una web de una manera más sencilla y óptima. Otro propósito que tiene Angular es la separación completa entre el front-end y el back-end en una aplicación web.

Una aplicación SPA creada con Angular es una aplicación de una sola página, en la cual la navegación entre secciones y páginas de la propia aplicación, así como la carga de datos, se realiza de manera dinámica y asíncrona, haciendo llamadas al servidor y, sobre todo, sin

refrescar la página en ningún momento. Es decir, las aplicaciones que podemos hacer con Angular son reactivas y no recargan el navegador.

La primera versión de Angular, fue AngularJS, lanzada en 2010, esta versión resultó con varios problemas para proyectos grandes, como la carga diferida (en inglés lazy loading), la cual consiste en retrasar la carga o inicialización de un objeto hasta el mismo momento de su utilización. Estos problemas originaron que en 2016 se publicara una nueva versión totalmente distinta a la primera, Angular 2. Actualmente, la última versión disponible es la versión 6, que es la que se usó en el desarrollo de este proyecto.

PWA

De sus siglas en inglés, Progressive Web App [15], una aplicación web progresiva es una evolución de las aplicaciones web que utiliza las últimas tecnologías disponibles en los navegadores para:

- Tener el mayor rendimiento posible en dispositivos móviles y lograr que se carguen más rápido.
- Ofrecer una experiencia en móviles lo más parecida a la de una aplicación nativa.
- Poder trabajar sin conexión.
- Poder enviar notificaciones a los usuarios, como una app nativa.

Para convertir una aplicación web a una aplicación web progresiva es necesario cumplir una serie de requisitos:

- Conexión HTTPS segura válida.
- Un archivo de manifiesto: Archivo que describe la información de la aplicación web progresiva.
- Un icono de App que será utilizada para ser el icono en los dispositivos móviles al ser instalada.
- Service Workers: Es un archivo Javascript que es registrado en el navegador y es el que va a permitir hacer las tareas de almacenamiento en caché, notificaciones push, actualización de contenido y más.

3.5 Aplicación móvil

Ionic

Ionic es un framework gratuito y Open Source para desarrollar aplicaciones híbridas, basado en angular y que integra Apache Cordova como base.

Una aplicación móvil híbrida es una combinación de tecnologías web, que actúa como una aplicación nativa, pero no lo es, ya que la aplicación consiste en un navegador interno del dispositivo móvil, llamado WebView, ejecutado dentro de un contenedor nativo; ni tampoco está basada en Web, porque se empaqueta como una aplicación para distribución y tienen acceso a las APIs nativas del dispositivo. Para ello existe Apache Cordova que provee una serie de plugins con las cuales poder acceder a dichas APIs nativas. La gran ventaja que ofrecen este tipo de aplicaciones es que podemos crear aplicaciones compatibles con todos los dispositivos móviles (IOS, Android, Windows Phone) con una sola implementación.

3.6 Seguridad

JSON Web Token

JSON Web Token [18] (JWT) es un estándar abierto basado en la generación de tokens, objetos JSON, que permite el intercambio de datos entre aplicaciones o servicios, garantizando que sean válidos y seguros.

Los JWT tienen una estructura definida y estándar basada en tres partes unidos por un punto:

```
Header.Payload.Signature
```

- Header: El header de un JWT tiene la siguiente forma:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

  donde La propiedad alg indica el algoritmo usado para en la firma y la propiedad typ define el tipo de token, en nuestro caso JWT.

- Payload: String en base64 de un objeto JSON en la que se especifica la información a transmitir. Puede tener cualquier propiedad, sin embargo hay campos estándar que tienen cierto significado asociados. De estos campos estándar, los más comunes son:
    - sub: indica el sujeto del token como el identificador de usuario.

    - iat: identifica la fecha de creación del token para establecer una fecha de caducidad.
    - exp: establece la fecha de expiración del token a partir de la introducida en el atributo iat.
- Signature: La firma del JWT está compuesta por los elementos anteriores (Header y Payload) en Base64 cifrados con una clave secreta.

El caso más común de uso de los JWT es para manejar la autenticación en aplicaciones móviles o web. Para esto cuando el usuario se quiere autenticar manda sus datos de inicio del sesión al servidor, éste genera el JWT y se lo manda a la aplicación cliente, luego en cada petición el cliente envía este token que el servidor usa para verificar que el usuario esté correctamente autenticado, por ejemplo.

En la Figura 2 se puede ver las partes que componen un JSON Web Token.

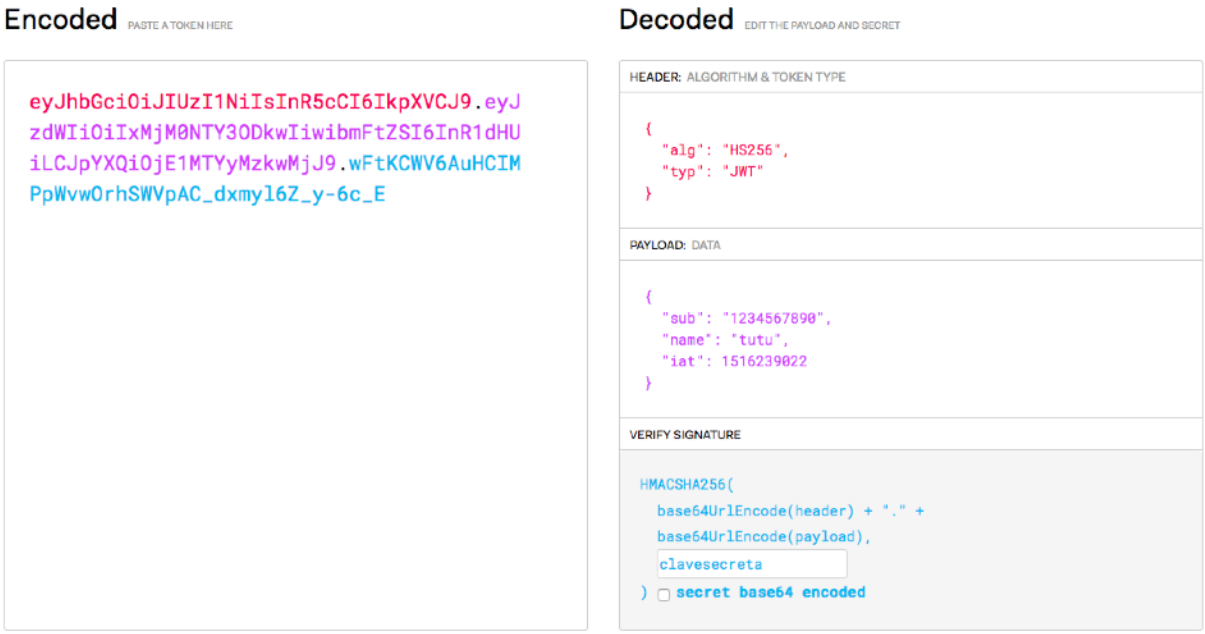

Figura 2: Estructura JSON Web Token

Certbot

Certbot [17] es un cliente ACME (Automated Certificate Management Environment) que obtiene, instala y actualiza certificados SSL/TLS, provistos por la Autoridad Certificadora (CA) Let's Encrypt o cualquier CA que soporte el protocolo ACME, de forma gratuita y automatizada.

3.7 Despliegue del proyecto

DigitalOcean

DigitalOcean es un proveedor de servidores virtuales privados, a los cuales llama droplets, que ofrece también servicio de hosting de DNS. Para el proyecto se contrató y configuró un servidor virtual de DigitalOcean, con el fin de poder desplegar en él los componentes desarrollados en este proyecto. Se decidió utilizar un servidor virtual de DigitalOcean por motivos que se comentarán posteriormente.

Nginx

Nginx [19] Es un servidor web asíncrono de código abierto usado como servidor web, proxy inverso, caché de HTTP, y balanceador de carga.

Está diseñado para ofrecer un bajo uso de memoria y alta concurrencia. En lugar de crear nuevos procesos para cada solicitud web, usa un enfoque asincrónico basado en eventos donde las solicitudes se manejan en un sólo hilo (single-thread).
Debido a su asincronía, cada solicitud se ejecuta de forma concurrente sin bloquear otras solicitudes.

HTTP/2

Es un protocolo de red utilizado por la World Wide Web, actualización del protocolo HTTP/1.1, con el que es compatible.

HTTP/2 no modifica la semántica de aplicación de HTTP, es decir, todos los conceptos básicos, tales como los métodos HTTP, códigos de estado, URI, y campos de cabecera se mantienen sin cambios, sin embargo introduce innumerables mejoras como:

- Cabeceras en formato binario en lugar de formato texto.
- Cabeceras comprimidas: Cuando se establece una conexión HTTP/2, todas las cabeceras se empaquetan en un solo bloque comprimido para ser enviados como una unidad.
- Tecnología server push: Permite enviar recursos a la caché del navegador sin que éste los solicite enviándole recursos que probablemente vaya a necesitar.

GZIP

Herramienta de compresión que se encarga de comprimir páginas en un servidor web antes de que las página se envíe al visitante, lo que aumenta significativamente la velocidad de carga de la página web.

Gitlab Pages

Servicio de alojamiento de páginas web provisto por Gitlab, que permite el uso de un dominios propios y certificados SSL/TLS provistos por Let's encrypt.

Surge.sh

Servicio web que ofrece alojamiento de páginas donde el despliegue por línea de comandos.

Heroku

Plataforma como servicio de computación en la Nube que permite alojar distintos tipos de aplicaciones, según el lenguaje de programación utilizado para su desarrollo, de forma gratuita.

Digital Ocean

Es un proveedor de servidores virtuales privados que además ofrece un servicio de hosting de DNS.

Capítulo 4. Análisis funcional y diseño técnico

4.1 Especificación de requisitos

Los requisitos variarán según el tipo de usuario. Estos pueden ser usuarios autenticados o usuarios no autenticados. A continuación se detallan cuales son los requisitos para usuarios autenticados, no autenticados y los que son comunes.

Requisitos comunes

- Ver anuncio de desaparición.
- Compartir anuncio.

Usuarios no autenticados

- Podrán registrarse/iniciar sesión mediante email y contraseña, Google o Facebook.
- Recuperar contraseña.
- Tendrá que confirmar su correo electrónico para poder iniciar sesión.
- Reenviar correo de confirmación de correo electrónico.

Usuarios autenticados

- Cambiar contraseña.
- Editar sus datos de usuario.
- Crear anuncios de desaparición.
- Ver/editar/eliminar sus anuncios de desaparición. La edición se podrá realizar sólo sobre los datos de contacto.
- Ver listado de anuncios cercanos, para poder verlo es necesario que el usuario especifique su ubicación.

4.2 Diseño técnico

4.2.1 Diseño de la base de datos

Para el diseño de la base de datos se planteó el siguiente diseño de la Figura 3:

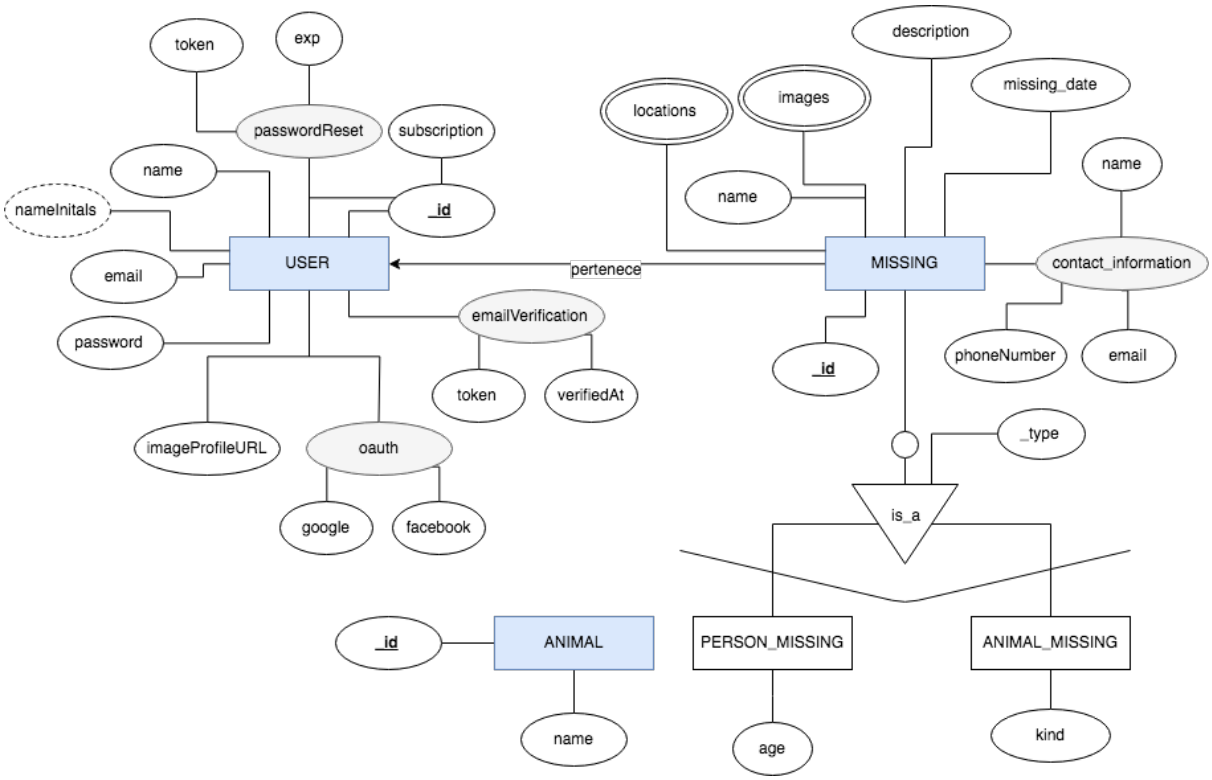

Figura 3: Diseño de base de datos

4.2.2 Diagramas de actividades

Con el propósito de facilitar la implementación y control de errores de las funcionalidades de la API, se crearon diagramas de actividades para aquellas, cuya ejecución suponía realizar varias operaciones y tener en cuenta múltiples errores. A continuación se muestran algunas de ellas.

Registro

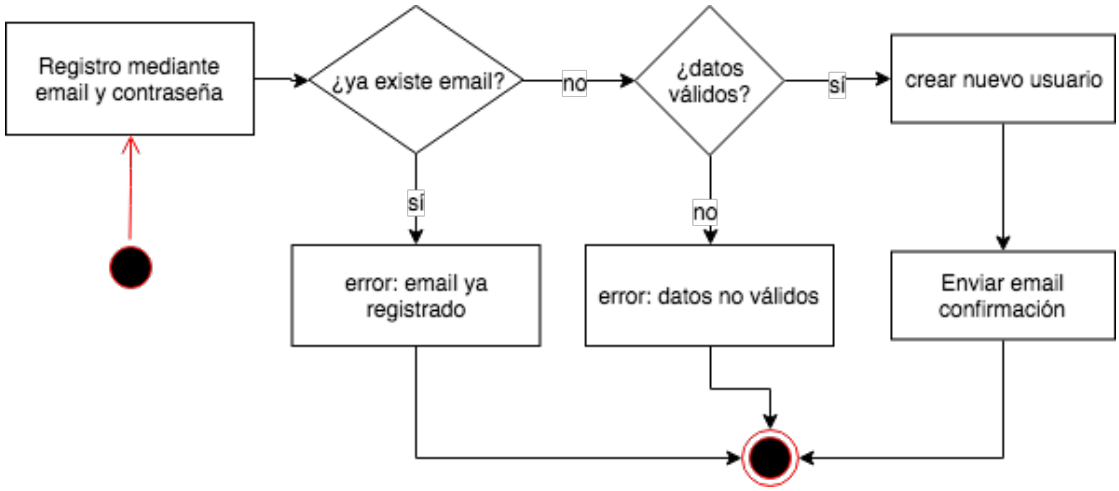

Figura 4: Diagrama de actividades de registro

Inicio sesión

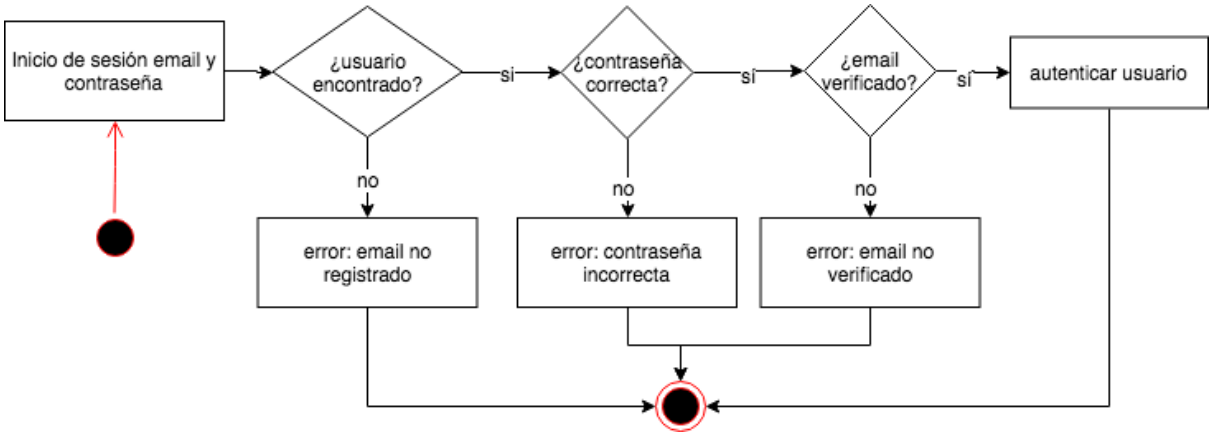

Figura 5: Diagrama de actividades de inicio de sesión

Autenticación mediante aplicación externa

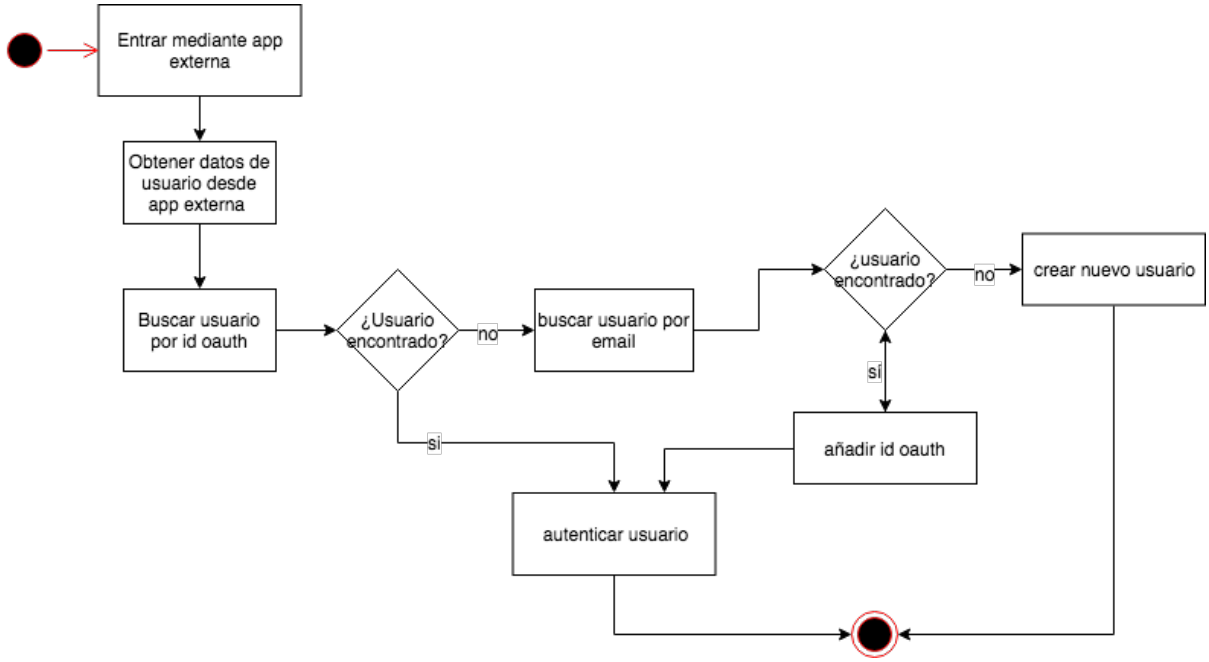

Figura 6: Diagrama de actividades de autenticación oauth

Crear desaparición

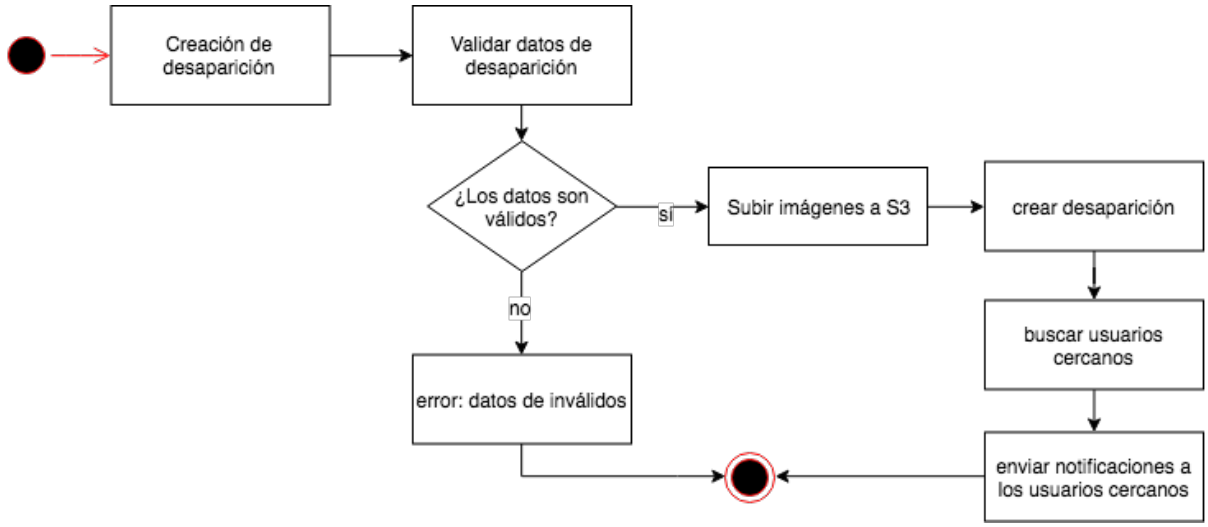

Figura 7: Diagrama de actividades de crear desaparición

4.2.3 Diseño de la interfaz de usuario

Para realizar el diseño de la interfaz de usuario de la aplicación web se utilizó la aplicación Sketch App. La forma de proceder fue:

1. Realizar el diseño en Sketch.
2. Implementar el diseño.

Repitiendo el mismo proceso para las modificaciones.

Si bien la creación o modificación del diseño en sketch no era inmediato, puesto que se realizaban varias alternativas de diseño hasta quedarse con el deseado, el uso de Sketch supuso un ahorro de tiempo considerable, ya que no se requiere el mismo tiempo para generar un diseño con Sketch que con HTML y CSS.

A continuación se muestran algunos diseños realizados con sketch, finales (a la izquierda) y alternativas.

Logo de la aplicación

Figura 8 Diseños de logo

Información completa de desaparición

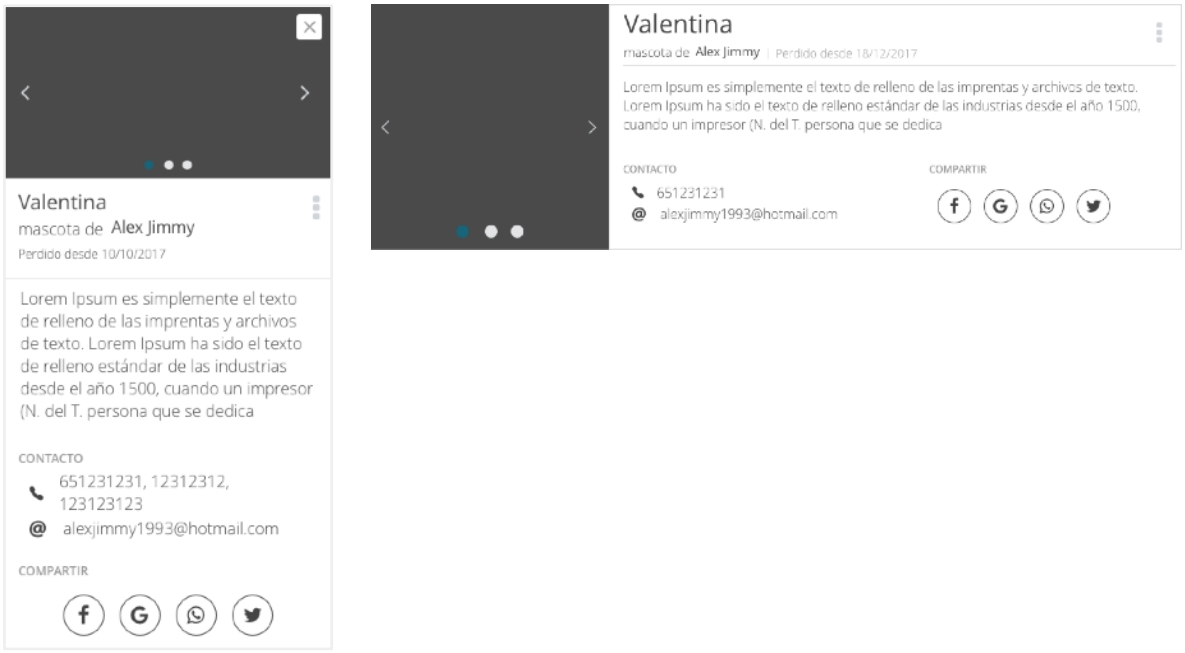

Figura 9: Diseños de información de desaparición

Vista de autenticación

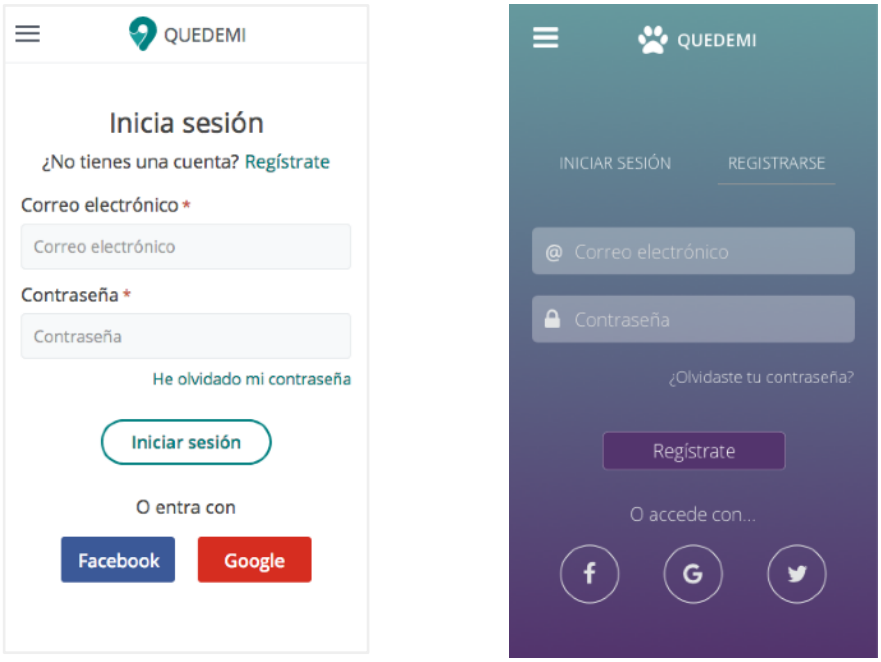

Figura 10: Diseños de vista de autenticación

Desaparición de listado

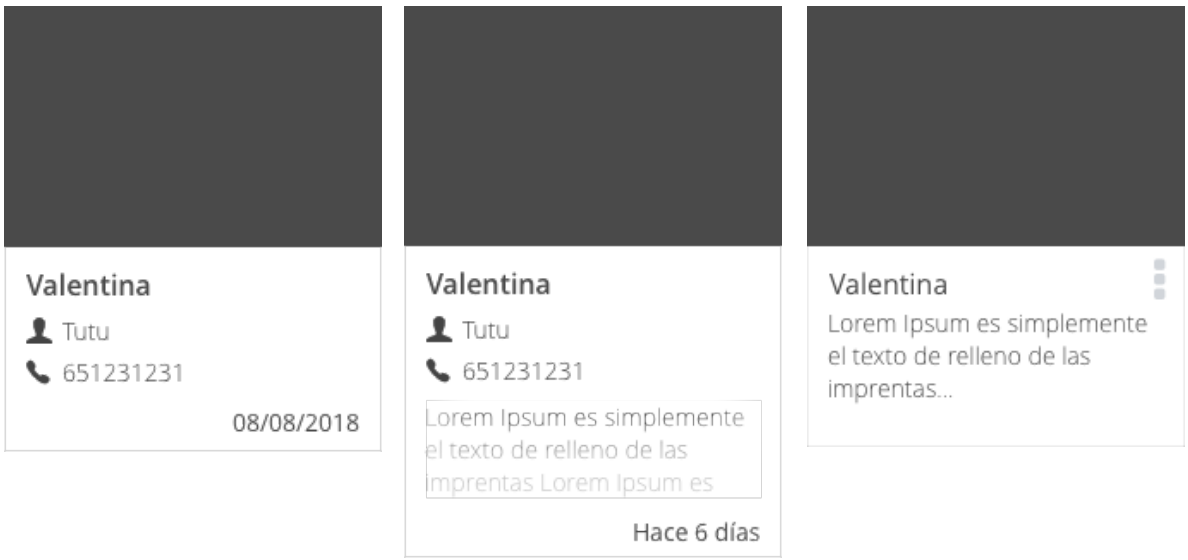

Figura 11: Diseños de desaparición en listado

Capítulo 5 Servidor

5.1 Base de datos

El sistema gestor de base de datos escogido fue MongoDB debido a su simplicidad, gran documentación y por su fácil integración y desarrollo con Node.js, mediante mongoose en este caso. De igual forma, MongoDB es la mejor opción para los requisitos del proyecto según los siguientes factores:

- Variabilidad: Según el diseño realizado de la base de datos, visto en el capítulo anterior, existen esquemas diferentes para un mismo modelo, como es el caso del modelo **Missing**. Además con la implementación de trabajos futuros, se prevé que existan modelos aún más cambiantes.
- Velocidad: Para ofrecer un listado de las desapariciones cercanas a la ubicación del usuario, es necesario que en la respectiva petición a la base de datos se calcule la distancia entre las localizaciones de cada desaparición y la del usuario, por lo que la velocidad de la base de datos es un factor importante si se quiere ofrecer una rápida respuesta.

5.1.1 Instalación

La instalación de la base de datos de MongoDB se realizó sobre una instancia Amazon EC2, para lo cual se creó una instancia con el sistema operativo Ubuntu Server 16.04 LTS con la siguiente configuración de grupo de seguridad asociada.

Type	Protocol	Port Range	Source	Description
HTTP	TCP	80	0.0.0.0/0	
HTTP	TCP	80	::/0	
Custom TCP	TCP	8080	0.0.0.0/0	MongoDB
Custom TCP	TCP	8080	::/0	MongoDB
SSH	TCP	22	0.0.0.0/0	
SSH	TCP	22	::/0	
Custom TCP	TCP	4000	0.0.0.0/0	quedemi dev
Custom TCP	TCP	4000	::/0	quedemi dev
HTTPS	TCP	443	0.0.0.0/0	
HTTPS	TCP	443	::/0	

Figura 12: Grupo de seguridad de instancia EC2

Estos grupos de seguridad actúan como un firewall para las instancias asociadas de Amazon EC2, al controlar el tráfico entrante y saliente en el ámbito de la instancia. La configuración de la Figura 12 se refiere al tráfico de entrada y como se puede observar, el puerto asociado a la base de datos será el 8080 aceptando peticiones desde cualquier origen.

5.1.2 Conexión a la base de datos

Durante el desarrollo del proyecto se accedió a la base de datos por dos medios:

- Mediante la propia API utilizando mongoose
- Mediante la aplicación Robo 3T, el cual, como se comentó anteriormente, provee una interfaz gráfica para trabajar con bases de datos NoSQL.

El motivo del uso de Robo 3T fue para poder manipular los documentos de la base de datos de forma sencilla y rápida, además de controlar que la información guardada fuese la esperada.Para que Robot 3T puede conectarse a la base de datos creada, se creó una conexión con la configuración de la Figura 13.

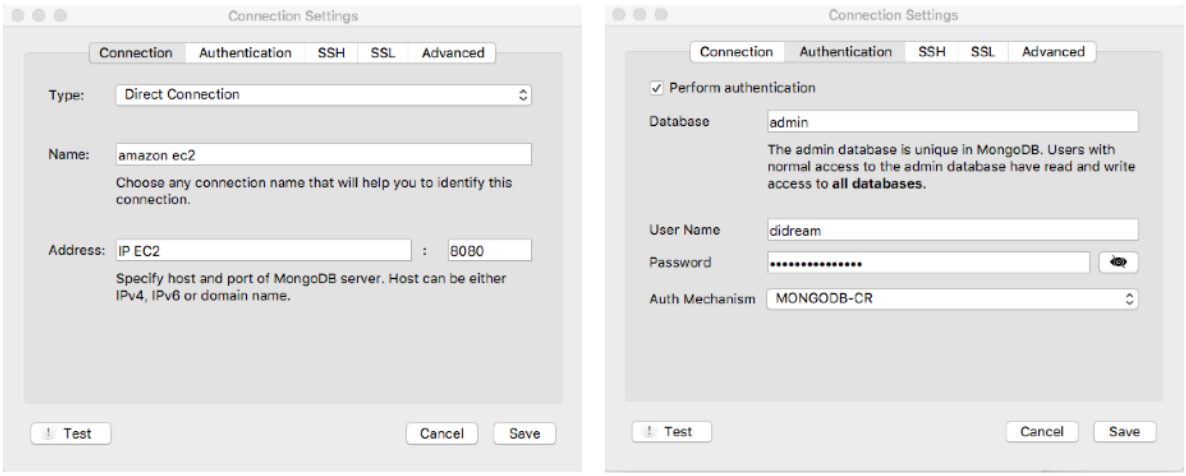

Figura 13: Configuración de conexión en Robo 3T

5.2 API REST

En este apartado se describirá la implementación realizada de la API. No se pretende explicar todo el código desarrollado sino aquellas partes que sirvan de ejemplo para alguna explicación dada.

Su implementación, como ya se comentó anteriormente, fue realizada con Node.js junto al uso del módulo Express. Su completa implementación puede ser consultada desde la rama **api** del repositorio del proyecto.

5.2.1 Configuración de la API

En este punto se describirá la implementación realizada para la configuración de la API. Los ficheros que se tratarán se encuentran en el directorio config/ de proyecto (Figura 14):

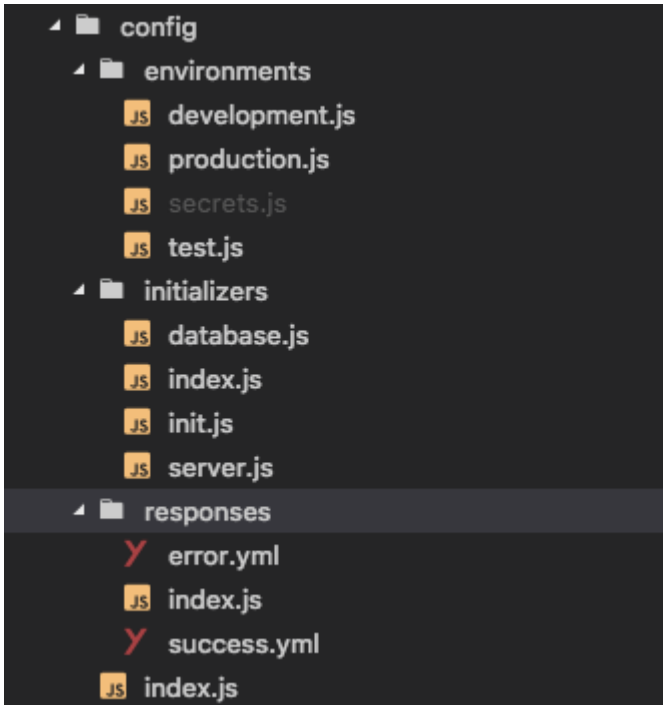

Figura 14. Contenido config/

Al ejecutar un programa con Node se crea una variable que nos da información sobre el proceso que está ejecutando, llamada process. process a su vez tiene un campo donde almacena las variables de entorno, llamado env.

Las variables de entorno sirven para definir parámetros sencillos de configuración de un programa, de modo que éstos puedan ejecutarse en diferentes ambientes. De estas variables de entorno, una muy popular y fundamental para la configuración de la API es process.env.NODE_ENV, puesto de que su valor depende qué fichero de ambiente se cargará.

En la API se especifican tres ambientes (development, test, production), con sus respectivos ficheros, presentes en el directorio config/environments/. En estos ficheros se declaran constantes que servirán para distintos fines, como indicar la dirección de los servicios externos que se usará, establecer claves públicas y privadas, o cambiar el comportamiento de la API para una funcionalidad determinada. Dichos valores pueden cambiar de un ambiente a otro, como el nombre de la base de datos, la cual será diferente para cada ambiente.

Junto a los ficheros de ambiente se encuentra secrets.js, el cual, por motivos de seguridad, se encuentra ignorado del sistema de control de versiones ya que almacena datos sensibles del proyecto. Su incorporación en las variables de entorno se realiza en el fichero de producción con la siguiente condición:

```
config/environments/production.js
```

```
...
if (process.env.VARIABLES_DECLARED != 'true') {
 Object.assign(process.env, require('./secrets'));
}
...
```

Si dicha condición se cumple se carga el fichero secrets.js, el caso contrario significa que las variables sensibles ya están presentes en las variables de entorno. Esto es útil para servicios como heroku, donde dichas variables sensibles deben especificarse manualmente (Figura 15), o al menos es lo recomendable.

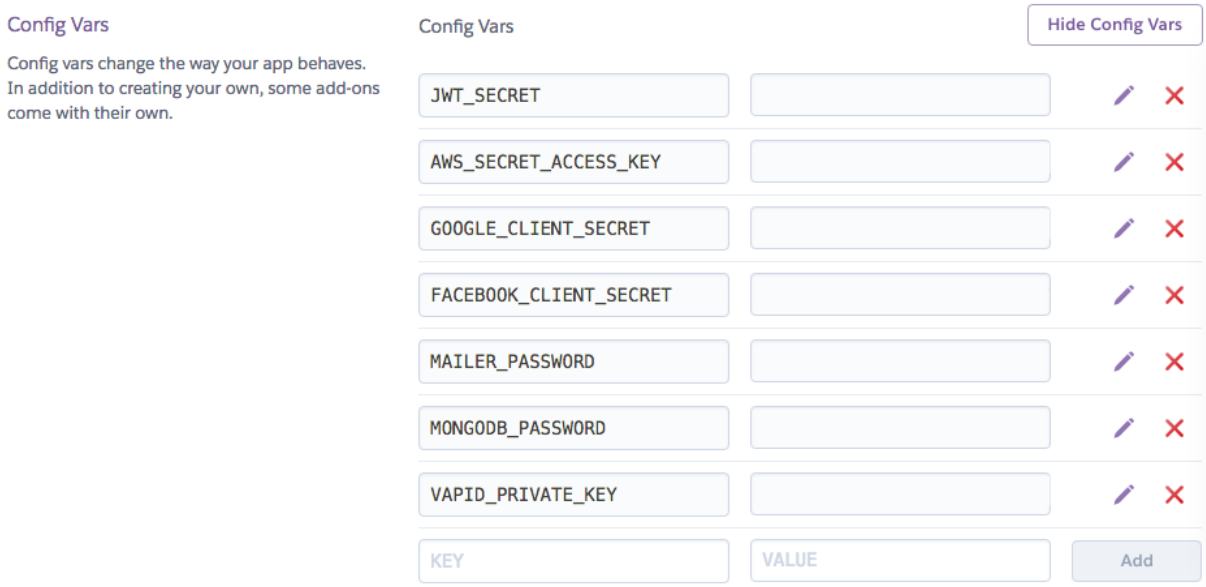

Figura 15. Variables de configuración en Heroku

Para generar las variables de configuración para la API, se parten de las variables del fichero de ambiente de producción, algunas de las cuales se modificarán si el ambiente es de desarrollo o test. Para ello se implementó la función merge, la cual sobrescribe los valores respetando la estructura existente.

config/index.js

```
'use strict';

const production =  require('./environments/production');

const environment = (process.env.NODE_ENV != 'production') &&
require(`./environments/${process.env.NODE_ENV}`);


function merge(container, part){
 for(let p in part){
   if (part[p] != null) {
     if (container[p] && typeof container[p] == 'object')
       merge(container[p], part[p])
     else
       container[p] = part[p];
   }
 }
 return container;
}
const join = (...parts) => parts.reduce(merge, {});
module.exports = join(production, environment);
```

De la misma forma, en el directorio de configuración se encuentra el directorio initializers/, que almacena las respectivas implementaciones de la conexión a la base de datos y configuración del servidor; a parte del fichero init.js el cual establece el entorno development por defecto, e implementa la generación de mensajes de éxito y de error previstos que puede devolver la API. Para dicha generación, se parte de los datos de respuestas consideradas, de error y de éxito, que se encuentran en responses/error.yml y responses/success.yml, respectivamente; los cuales servirán como parámetros para los constructores de las clases ErrorResponse y SuccessResponse. Esto se realizó con el fin de proveer respuestas que sigan un determinado esquema (Figura 16), tanto para todos los errores que puedan ocurrir en la api, ya sean errores de cliente, autorización, de la propia API, etc; como para las respuestas satisfactorias.

Figura 16. Respuestas de error de la API

5.2.2 Modelos

De acuerdo al diseño de la base de datos realizada, los modelos implementados fueron los modelos Missing, User y Animal. Los esquemas de cada modelo fueron realizados con el uso de mongoose. En ellos se especifica los datos que cada documento almacenará. Si bien en MongoDB el esquema de los documentos es libre, con mongoose se respeta, ignorando los atributos adicionales.

Para crear las variantes de desapariciones AnimalMissing y PersonMissing se hace uso de una clave discriminatoria (discriminator key). Dicho proceso funciona como si se tratase de herencia de clases: Los modelos discriminatorios que se creen heredarán la configuración del esquema del modelo padre (atributos, métodos, hooks…). De igual forma dicha configuración podrá ser sobreescrita en el esquema de cada modelo discriminatorio. Esta implementación dará como resultado que en la base de datos se guarde una sola colección, sin embargo, gracias a mongoose, se podrá trabajar con estos modelos como si se tratasen de colecciones independientes.

Además del uso de la clave discriminatoria, se usaron las siguientes funcionalidades de mongoose.

Métodos virtuales

Volviendo al símil con las clases, estos representarían métodos getter y setter.

app/models/user.js

```
...
userSchema.virtual('payload').get(function(){
 return {
   access_token: jwt.encode({id: this._id}, 1209600),  //
expiration time: Se suma 14 dias (en segundos 14*24*60*60);
   ...this.publicProfile,
   missings: this.missings || []
 };
});
...
```

Dentro de la función especificada, this referencia al documento.

Virtual Populate

La función populate de mongoose, "pobla" las relaciones (referencias) de un documento con las respectivas instancias. Mediante virtual, estas referencias se especifican de forma virtual.

En los modelos de la API, el esquema del modelo User "referencia virtualmente" a Missing.

app/models/user.js

```
...
userSchema.virtual('missings', {
  ref: 'Missing',
  localField: '_id',
  foreignField: 'createdBy'
});
...
```

Para ello el esquema de Missing tiene una referencia al modelo User mediante su id (_id).

app/models/missing/missing.js

```
...
let missingSchema = new Schema({
 ...
 createdBy: {
   type: Schema.Types.ObjectId,
   ref: 'User',
   required: true,
   protected: true
 }
 ...
});
...
```

De esta forma cuando se quiera obtener las desapariciones creadas por un usuario se añadirá .populate('missings') a la correspondiente petición.

app/controllers/user.js

```
...
 login(req, res, next){
   const {email, password} = req.body;
   User.findOne({email})
     .populate('missings')
     .then(user => {
       ...
     })
     .catch(next);
 },
...
```

Document middlewares

Los middlewares de documento de mongoose (también llamado pre y post hooks) son funciones a las que se les pasa el control durante la ejecución de una operación sobre un documento. Las operaciones a las que se puede asociar middlewares son: validate, save, remove e init. Los middlewares pueden asociarse antes (.pre) o después (.post) de dichas operaciones.

app/models/user.js

```
...
userSchema.post('validate', async function(user, next) {
 try {
   if (user.isModified('password')) {
     const salt = await bcrypt.genSalt(10);
     const hash = await bcrypt.hash(user.password, salt, null);
     user.password = hash;
   }
   ...
 }
 catch(err) { next(err) }
});
...
```

Métodos estáticos del modelo

Métodos que pueden ser ejecutadas desde el modelo.

app/models/user.js

```
...
userSchema.statics.findByPasswordResetToken = async
function(token){
 if (!token) throw error.token.invalid;


 const user = await this.findOne({'passwordReset.token':
token});
 if (!user || Date.now() > user.passwordReset.exp) throw
error.token.invalid;
 return user;
}
...
```

Métodos de documento

Métodos que pueden ser ejecutados desde la instancia de un documento.

app/models/user.js

```
...
userSchema.methods.verifyPassword = function(pass){
 return bcrypt.compareSync(pass, this.password);
};
...
```

5.2.3 Rutas

Las rutas especifican cómo responderá la API a una petición del cliente en una determinada URI y con un determinado método de acceso HTTP (GET POST, PUT o DELETE), mediante manejadores.

La implementación de las rutas se realiza a la través de la clase Router de Express. Gracias a ello podemos modularizar las rutas en diferentes ficheros, los cuales están ubicados en app/routes (Figura 17).

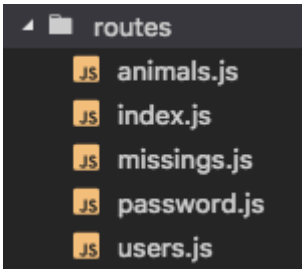

Figura 17. Contenido de app/routes

Cada ruta puede tener una o varias funciones de manejador, las cuales se ejecutan cuando el path y método de acceso de la petición corresponden con los especificados en la ruta.

La definición de una ruta tiene la siguiente estructura:

```
router.METHOD(PATH, MANEJADORES)
```

donde **router** corresponde a una instancia de express.Router.

METHOD puede ser un método de solicitud HTTP (GET, POST, PUT, DELETE, …) o un método provisto por Express con el fin de que la ruta invoque más peticiones. Algunos de estos métodos son:

```
router.use([path], [function, ...] function)
```

El manejador o manejadores se ejecutan para cualquier petición recibida y si el path es especificado, las que correspondan con el path, sin importar el método HTTP.

```
router.use([path], [function, ...] function)
```

El manejador se ejecutará para aquellas rutas que pasen un parámetro que corresponda con el de name.

PATH es un string o expresión regular que representa una vía de acceso en el servidor.

Y, por último los **MANEJADORES** representa los middlewares y controladores, los cuales será comentados en el siguiente apartado.

5.2.4 Controladores y middlewares

Los controladores y middlewares están ubicados en el directorio app/controllers/ (Figura 18) implementados en los mismos ficheros.

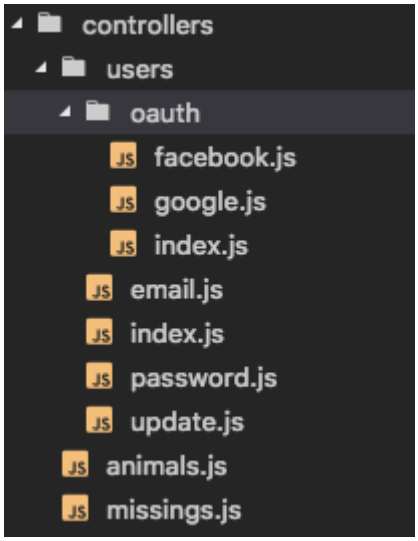

Figura 18. Contenido de app/controllers

El motivo por el cual, controladores y middlewares están en los mismos ficheros es para evitar los requerimientos, tanto en los ficheros de rutas, como el de los nuevos ficheros que supondría crear, además que la presencia de middlewares en el proyecto es escasa, de momento. Sin embargo a medida que el proyecto crezca será conveniente crear un nuevo directorio "app/middlewares/" donde éstos serán ubicados.

Controladores y middlewares constituyen los manejadores de las rutas. Son los encargados de implementar las operaciones que las funcionalidades de la API requieren; como el acceso o modificación de los documentos de la base de datos, a través de los modelos; envío de correos electrónicos, etc.

La diferencia entre ambos es que el middleware no actúa como punto final (endpoint) al procesar la petición, como sí lo hacen los controladores, si no como métodos reutilizables, que revisan que la petición cumpla ciertos requisitos; como por ejemplo que el token de acceso esté presente en la petición y sea válido, para el caso de la autenticación.

Un manejador puede recibe tres parámetros:

1. **request**: Representa la petición HTTP.
2. **response**: Representa la respuesta HTTP.
3. **next**: Función que indica el siguiente manejador; si recibe algún parámetro, éste corresponderá al manejador de errores.

Por lo que si durante su ejecución se necesita indicar un error, éste debe pasarse como parámetro a next, tras lo cual la petición será procesada por el manejador de errores global donde se elaborará la respectiva respuesta de error.

El controlador, si al final de su ejecución no indicó ni obtuvo ningún error, elaborará la respectiva respuesta de éxito. Sin embargo el middleware llamará al siguiente manejador mediante next.

Para describir el proceso que sigue la petición a través de middlewares y controladores, se tomará como ejemplo la operación de editar una desaparición mediante la ruta missings/:missingId/edit.

En los siguientes ficheros se puede ver el "camino" que seguirá la petición.

app/routes/index.js

```
...

router.use('/missings', require('./missings'));

...
```

app/routes/missings.js

```
...

 router.use(users.authenticate);

   router.post('/', missings.create);

 router.all(['/:missingId', '/:missingId/edit'], missings.find);

   router.use(missings.checkOwner);

     router.get('/:missingId/edit', missings.edit);

...
```

Si dicha petición es correcta, se ejecutará las siguientes funciones en el orden establecido:

1. users.authenticate

Middleware que se encarga de verificar la correcta autenticación del usuario.

app/controllers/users/index.js

```
...
 authenticate(req, res, next){
   if (!req.headers.authorization) throw error.authorization.required
   const payload = jwt.parseAuthorization(req.headers.authorization);
   User.findById(payload.id)
     .then(user => {
       if (!user || user.verified === false) throw
error.authorization.denied;
       req.user = user;

       next();
     })
     .catch(next);
 },
...
```

2. missings.find

Middleware que busca la desaparición en la base de datos según el parámetro :missingId especificado en la ruta de la petición.

app/controllers/missings.js

```
...
 async find(req, res, next){
   const missingId = req.params.missingId;
   try {
     if (!mongoose.Types.ObjectId.isValid(missingId)) throw
error.missing.notFound;
     const missing = await Missing.findById(missingId);
     if (!missing) throw error.missing.notFound;
     req.missing = missing;
     next();
   }
   catch(err) {
     next(err);
   }
 }
...
```

3. missings.checkOwner

Middleware que se encarga de comprobar si la desaparición pertenece al usuario autenticado.

```
app/controllers/missings.js
```

```
...
 checkOwner(req, res, next){
   if (req.missing.createdBy != req.user.id)
     next(error.missing.notFound);
   else
     next();
 },
...
```

4. missings.edit

Método de controlador que responde con los datos de desaparición.

app/controllers/missings.js

```
...
 edit(req, res, next){

res.build( success.missing.allow.data(req.missing.toObject()));
 },
...
```

5.2.5 Emails y notificaciones push

Con el propósito de enviar mensajes al correo electrónico del usuario, ya sea sobre una nueva desaparición registrada cerca de su ubicación, para restablecer la contraseña o para confirmar el correo electrónico, se ha implementado el servicio de correo mediante el módulo nodemailer, el cual nos permite crear el servicio de mensajería de forma sencilla. Para ello es necesario especificar el servidor SMTP que usaremos para enviar los correos electrónicos.

Junto a nodemailer, para la implementación del diseño de los correos electrónicos, se ha utilizado el módulo email-templates, el cual nos permite:

- Trabajar con el motor de plantilla pug, entre otros.
- Pasar variables a la plantilla.

- Separar los estilos css de la implementación, ya que para el diseño de emails los estilos deben especificarse en línea, para cada elemento HTML.

Estos diseño están almacenados en el directorio resources/ junto a sus assets (Figura 19).

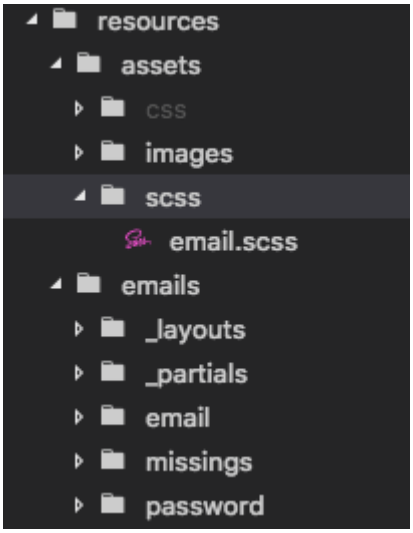

Figura 19. Contenido de resources

Para simplificar el diseño de las plantillas y aprovechando el uso de pug, se ha creado un layout (primary.pug), el cual les servirá de base. En este layout se ha implementado un header, con el logo de la aplicación. Gracias a esto sólo ha sido necesario implementar el cuerpo del mensaje en cada plantilla de email.

Para poder trabajar con los estilos css de forma ordenada y que resulte mantenible, se ha optado por utilizar el preprocesador de estilos scss, cuya compilación a css se realiza al inicio del servidor, mediante el comando npm start o npm run dev.

package.json

```
...
 "scripts": {
   "start": "gulp sass && node index.js",
   "dev": "gulp",
...
```

Para ello se ha instalado los módulos gulpfile, gulp-sass y gulp-nodemon; e implementado el fichero gulpfile.js con la tarea 'sass' encargada de dicha conversión, y adicionalmente una tarea que trabaja con nodemon. En la tarea por defecto se ejecutan ambas tareas, añadiendo además, el observador de cambios sobre los ficheros scss.

De igual forma que los correos electrónicos, las notificaciones push también comunicarán el registro de una desaparición con alguna localización cercana a la del usuario. Para poder

implementar este tipo de notificaciones se ha utilizado el módulo web-push. Su uso consiste en especificar una suscripción push y los datos de la notificación. Esta suscripción push se genera tras aceptar recibir notificaciones en el cliente, la cual se almacena en los datos del usuario.

5.2.6 Tests

Los correspondientes ficheros de tests así como toda la implementación realizada para esta labor se encuentra en el directorio test/ (Figura 20)

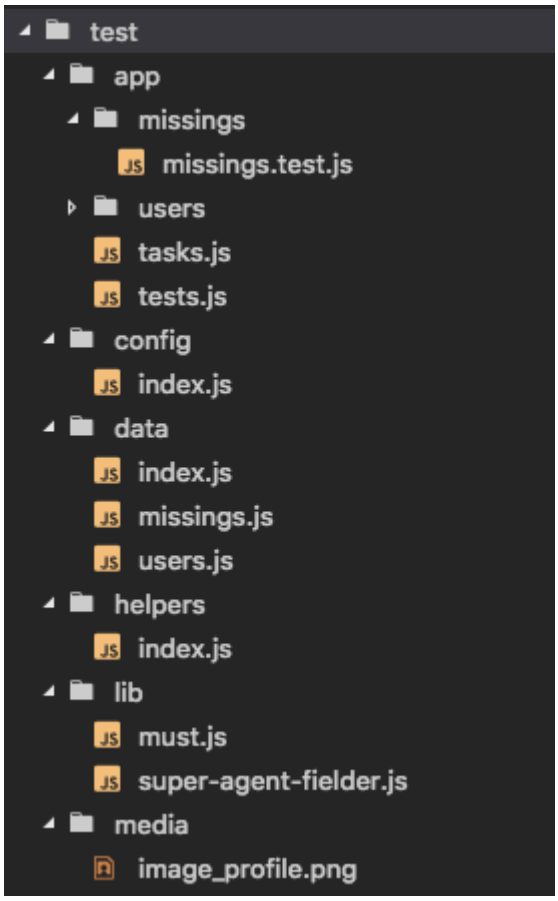

Figura 20. Contenido de test/

Los tests se crearon con el fin de que la API sea fácilmente mantenible y para garantizar que las funcionalidades existentes sigan funcionando correctamente ante modificaciones o incorporación de nuevas. Para ello, cuando las funcionalidades a testear fueron implementadas, se procedió a crear los respectivos tests que comprobaran su correcto funcionamiento, tanto para los casos en los que las operaciones debieran finalizar correctamente, como para comprobar que los errores ocurridos fueran los esperados.

Para facilitar esta labor, se implementó funciones que comparan las respuestas obtenidas, con las respuestas establecidas en el directorio de configuración, más concretamente, se comparan los campos domain y reason (Figura 21).

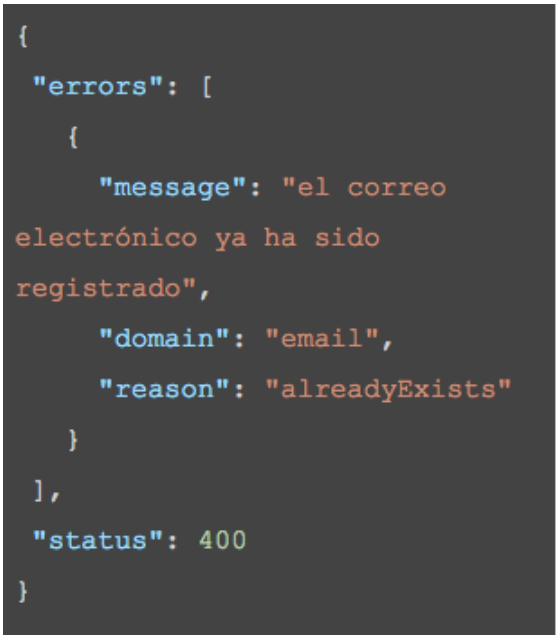

Figura 21. Respuesta de error de la API

Estas funciones se encuentran en el fichero test/lib/must.js.

La ejecución de los tests se realizará de forma ordenada, dicho orden se especifica en el fichero test/app/tests.js, el cual será el único fichero que ejecute mocha. Antes del inicio de los tests, se ejecutará el fichero tasks.js, en el que la tarea configurada se encargará de borrar la base de datos y de poblarla, mediante el uso del módulo populater.

Los tests se ejecutan mediante el comando npm test. En la Figura 22 se puede ver el resultado.

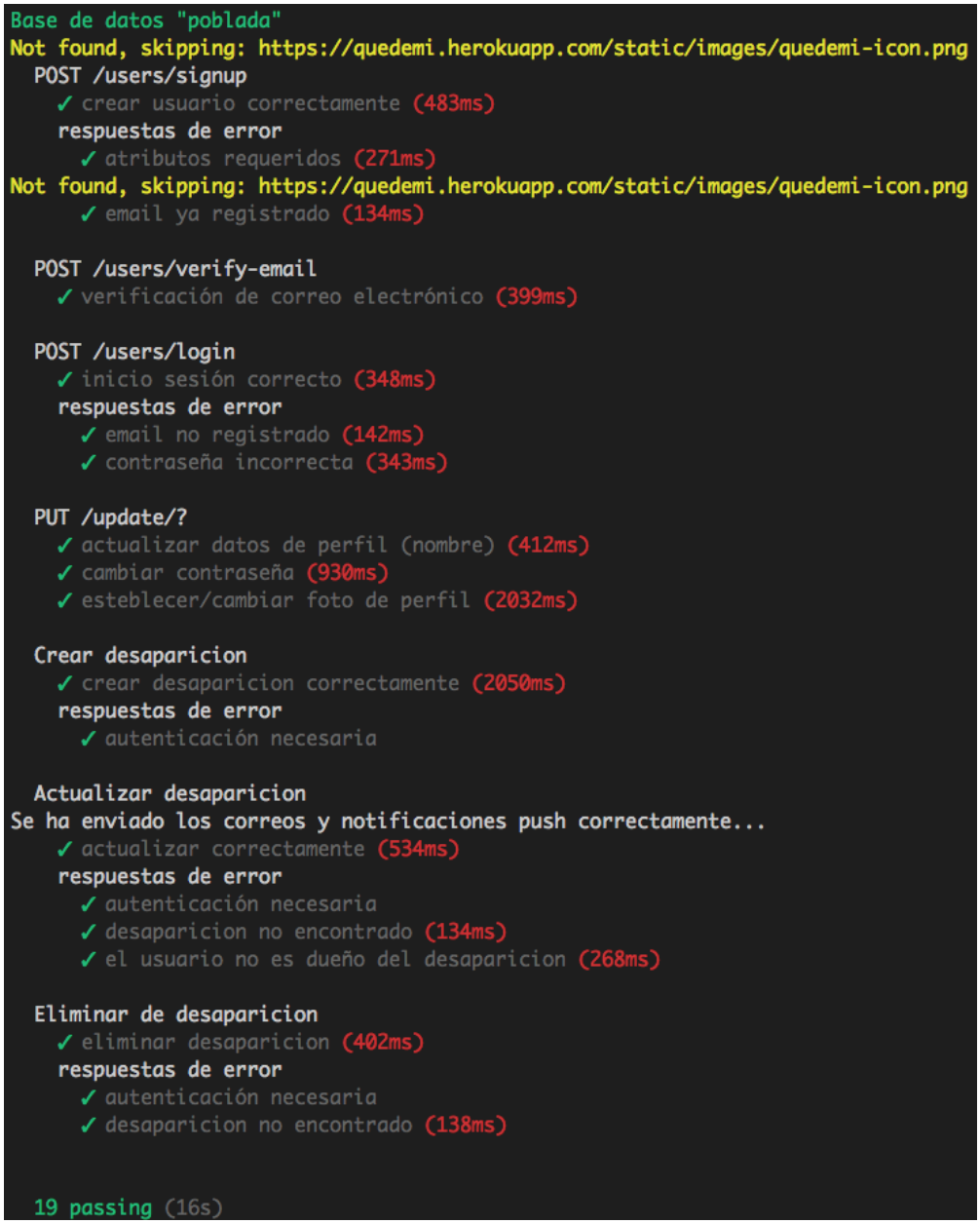

Figura 22. Resultado ejecución de tests.

5.2.7 Seguridad

Como es común en aplicaciones que permitan la autenticación basada en contraseña, es necesario que dicha contraseña sea guardada cifrada en la base de datos. Para ello se ha usado el módulo bcrypt el cual nos provee métodos para el cifrado y comprobación de las contraseña.

De igual para el proceso de autenticación, se ha usado jwt-simple para crear el token de acceso de un usuario. Para la generación del token se ha usado como payload el identificador (_id) del usuario, junto a los campos estándar iat y exp los cuales, como se comentó anteriormente, sirven para controlar el tiempo de validez del token.

5.2.8 Módulos desarrollados

Junto con la implementación de la API, de han desarrollado dos módulos, que si bien aún no se encuentran en npm, cuando se implemente sus respectivos test y README, lo estarán.

populater

Módulo que trabaja con mongoose para poblar la base de datos. Los datos pueden ser procesados desde un fichero .yml o .js y deben seguir la siguiente sintaxis. Su ejecución puede ser realizada desde línea de comandos a través del comando populate o mediante el módulo requerido, tras lo cual se devuelve un array con los documentos creados.

mongoose-mass-assignment

Este módulo nace de la necesidad de proteger campo de un esquema, ante modificaciones, para asignaciones en masa, puesto que mongoose no provee ningún método para este fin.

La solución planteada en el plugin, quita los paths especificados en el esquema con protected:true, del objeto de datos a asignar. Esto se realiza en el método massAssignment, estático y de documento que éste provee.

5.2.9 Problemas encontrados

El primer problema que se presentó durante el desarrollo de la API estuvo relacionado con el nombre que inicialmente se dio al modelo Missing, la cual fue Adverts, en español Anuncios; dicho nombre también era usado en las rutas (adverts). Si bien las rutas funcionaban como se esperaba con Postman; en los navegadores eran bloqueados, sin especificar ningún error ni aviso. El problema era ocasionado por el bloqueador de anuncios instalado en estos navegadores, que bloqueaba dichas peticiones por el nombre comentado. La solución fue cambiar el nombre de adverts en las rutas por el actual, missings, y para mantener la relación de nombre de rutas, controladores, modelos… el cambio se hizo en todo el proyecto.

El segundo problema, para el cual aún no se ha encontrado una solución, está relacionado con el envío de notificaciones push a la aplicaciones web progresiva. La funcionalidad que se desea ofrecer, mediante estas notificaciones, es que el usuario pueda recibirlas en múltiples dispositivos suscritos; sin embargo, para ello habría que identificar cada dispositivo, desde el cual el usuario se conecte, para determinar si está suscrito o no. Para este propósito, inicialmente se usaron los datos de la propia suscripción push, sin embargo ésta cambia cada vez que el service worker es actualizado.

Capítulo 6. Aplicaciones web y móvil

En el presente capítulo se describirá la implementación realizada de los dos componentes que trabajan en el lado del cliente: la aplicación web y la aplicación móvil.

De la misma forma que en el capítulo anterior, no se explicará todo el código implementado, sino aquellas partes que sirvan de ejemplo para alguna funcionalidad explicada.

6.1 Aplicación web

Se trata de una aplicación de página única, denominada SPA (single page application), que ha sido implementada con el framework Angular; el proyecto fue creado inicialmente con Angular 5, pero durante su desarrollo se ha actualizado a la última versión actualmente disponible, la versión 6.

Al tratarse de un SPA, todas las vistas que se muestre se harán a través de la misma página, sin recargar el navegador.

En este apartado se describirá, brevemente, las partes que componen Angular y cómo han sido usadas para la implementación de la aplicación web.

6.1.1 Componentes

En Angular IO, el desarrollo de la aplicación se realiza en base a componentes web mediante una jerarquía en árbol, en el que se podría considerar los siguientes niveles:

- Nivel de raíz: corresponde al componente principal del proyecto. Este por defecto está implementada en app.component.ts.
- Nivel Troncal: Generalmente está compuesto por dos o tres componentes para la estructura de la página. Los componentes troncales pueden ser el header, footer y el contenedor del cuerpo de la página, por ejemplo.
- Nivel de Ramas: Las ramas equivalen a las vistas de la aplicación. Los componentes de este nivel corresponden a los que se cargan dentro del componente troncal contenedor.
- Nivel de Hojas: Cada una de las vistas puede estar formada a su vez por múltiples componentes.

Un componente web se encarga de controlar una zona de espacio de la pantalla y su uso se realiza como si se tratase de una nueva etiqueta html mediante su selector. Su implementación se realiza de la siguiente forma:

src/app/modules/nousers/components/carusel/carusel.component.ts

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

@Component({

 selector: 'app-carusel',

 templateUrl: './carusel.component.html',

 styleUrls: ['./carusel.component.scss']

})

export class CaruselComponent{

...

}
```

En ella se puede ver los campos básicos que un componente tiene asociados:

- **selector**: Se utiliza en la plantilla para instanciar el componente.
- **templateUrl**: El fichero con la plantilla html. Alternativamente, la plantilla puede ser especificada en el propio fichero mediante el campo template.
- **styleUrls**: Estilos css declarados en uno o más ficheros. Estos estilos también pueden ser declarados en el propio fichero mediante el campo styles. Los estilos no son obligatorios para el componente.

Finalmente, cabe destacar que en la propia clase se definirá su funcionalidad.

Para definir algunos componentes desarrollados en la aplicación web, se parte de la vista de inicio de sesión. En ella se pueden encontrar 4 componentes distintos, que, tomando la clasificación por niveles, serían:

- Nível de raíz: AppComponent.
- Nivel troncal: HeaderComponent y SigninComponent.
- Nivel rama: LoginComponent

En la Figura 23 se puede apreciar la ubicación de cada componente en la vista.

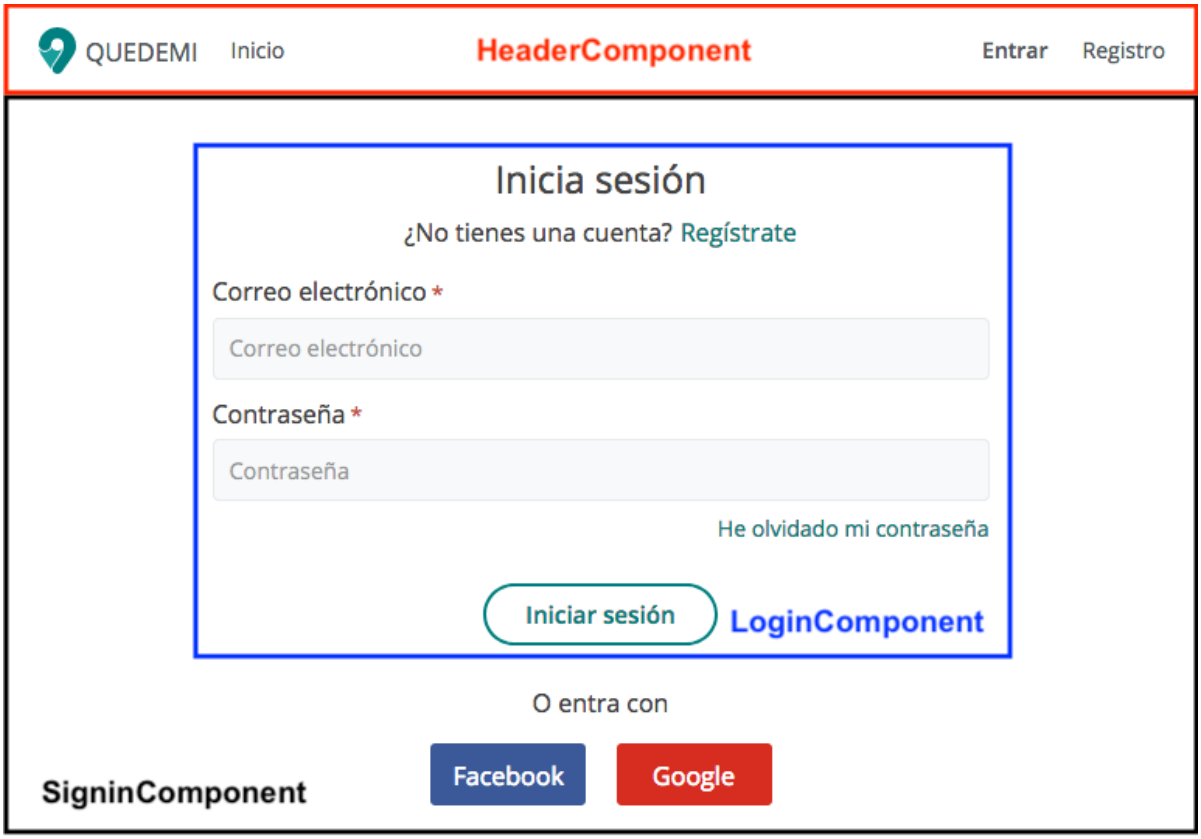

Figura 23. Ejemplo de componentes

6.1.2 Módulos

Los módulos, en Angular, agrupan elementos (componentes, directivas, servicios…), con el fin de tener un proyecto estructurado en partes y encapsular funcionalidad para crear aplicaciones desacopladas con bloques reutilizables.

Se podrían considerar como una fábrica de funcionalidad que:

- Importan elementos que otros módulos exportan.
- Declara los elementos que él mismo fabrica.
- Exporta algunos de estos elementos, para que los consuman otros módulos.

Además dichos módulos pueden ser cargados de forma diferida, método de carga conocido como lazy load; es decir, la carga del módulo no se realiza hasta el momento de su uso. De esta forma se mejora el tiempo de la carga inicial de la aplicación.Los principales módulos implementados en la aplicación fueron los que se observan en la Figura 24. Estos se encuentran en el directorio src/app/modules.

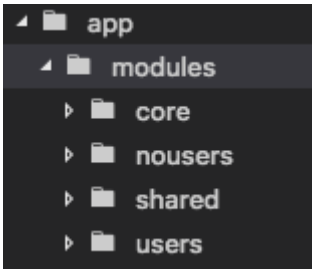

Figura 24. Módulos principales

- **core.module**: Corresponde al módulo principal de la aplicación (Figura 25). Ésta sólo debe ser importada una vez.

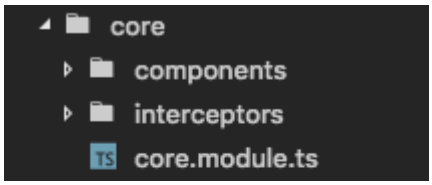

Figura 25. Módulo core

- **shared.module**: Es el modulo mas importado puesto que agrupa las componentes que se utilizaran en múltiples módulos (Figura 26).

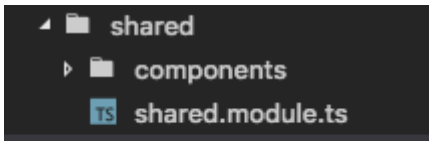

Figura 26. Módulo shared

- **users.module**: Como su nombre lo indica, representa el módulo de los usuarios. No es importado por ningún módulo, pues en este se aplica lazy load y su carga se realizará cuando el usuario acceda a los recursos de la plataforma, siempre después de haberse llevado a cabo una correcta autenticación. A través de este módulo se pueden acceder a otros módulos del usuario, estos representan las distintas funcionalidades que tendrá el usuario, como crear una desaparición o editar sus datos (Figura 27).

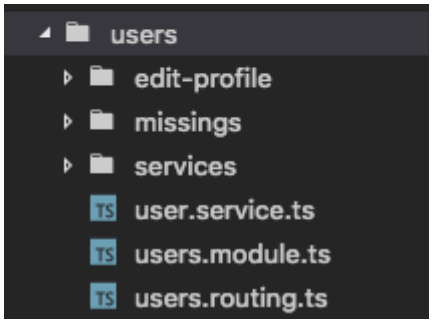

Figura 27. Módulo users

- **nousers.module**: Módulo para los usuarios que no están autenticados. Si bien su carga se realiza desde un inicio, contiene módulos que serán cargados en diferido, estos módulos tienen que ver con la autenticación y con el restablecimiento de contraseña (Figura 28).

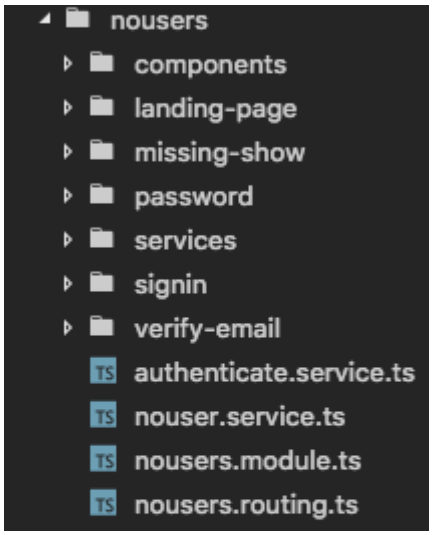

Figura 28. Ejemplo de componentes

6.1.3 Servicios

Los servicios son clases que servirán para realizar acciones concretas, como proveer datos, implementar las peticiones a la API, o realizar una lógica compleja. Pueden ser usadas (inyectadas) en componentes o directivas, e incluso en otros servicios; en las cuales habrá que añadir el decorador inyectable().

Los servicios son usados a menudo para compartir datos entre distintos elementos.Un claro ejemplo de esto es el servicio MissingCreateService, el cual comparte una variable object entre los dos componentes que conforman el proceso de creación de desapariciones; DataMissingComponent, y ContactDataComponent (Figura 29).

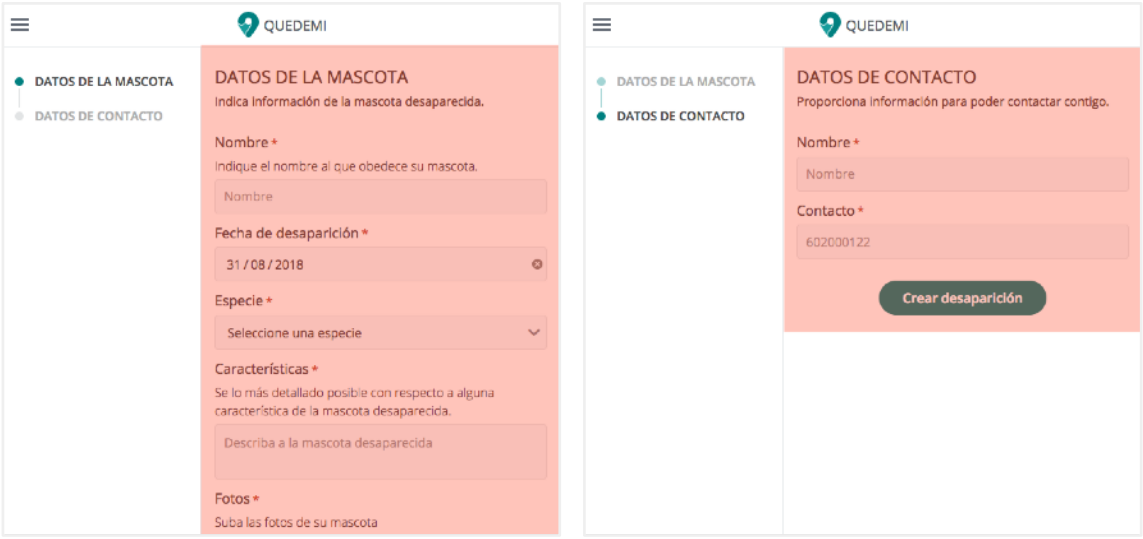

Figura 29. Ejemplo de componentes

Cada uno de ellos se encarga de recoger una parte de los datos necesarios para este proceso, a través de un formulario, para posteriormente, añadirlos a la variable compartida. Como resultado, la variable contendrá todos los datos necesarios para crear la desaparición en la API.

6.1.4 Rutas

La rutas sirven para cargar un componente determinado en la página, trabajando en el nivel de ramas de la jerarquía en árbol comentado en el punto 5.1.1 Componentes. De igual forma, permiten implementar la carga diferida de módulos, la cual consiste en que la carga del módulo no se realizará hasta el momento de uso, como se comentó en el punto 6.1.2 Módulos.

Para que la navegación mediante rutas funcione en la aplicación, es necesario importar el módulo RouterModule desde @angular/router y especificar dónde se va a cargar el componente asociado a la ruta. Esto se realiza mediante el tag router-outlet en la plantillas de los componentes contenedores, los cuales corresponden con el componente principal de la aplicación y los componentes asociados a una ruta padre.

Para describir las principales propiedades que se pueden especificar en una ruta, se tomará como ejemplo las rutas ‘/’, ‘/entrar’ de la aplicación web.

Ruta /

src/app/modules/nousers/nousers.routing.ts

```
...
{ path: '', pathMatch: 'full', component: LandingPageComponent,
canActivate: [ NouserService ]},

...
```

- path: Especifica la ruta que resuelve. En este caso la ruta vacía o '/'.

- **pathMatch**: Con el valor 'full' significa que la ruta a resolver debe coincidir exactamente con la especificada en path.

- **component**: Establece que componente se cargará. En este caso LandingPageComponent.

- **canActivate**: Establece funciones, conocidas como guards, que realizarán comprobaciones, retornando al final un bolean, ya sea de forma síncrona o asíncrona; mediante una promesa o un observable. Existen 4 tipos de guards, los cuales son:

- CanActivate: Decide si una ruta puede puede ser activada
- CanActivateChild: Decide si una de las rutas hijas puede ser activada.
- CanDeactivate: Decide si una ruta puede ser desactivada.
- CanLoad: Decide si un módulo puede ser cargado. Dicha carga se realizaría de forma diferida.

Como es lógico, los guards que pueden ser especificados para esta propiedad son del tipo CanActivate.

Para definir un guard es necesario que la clase implemente el método asociado al guard donde se implemente el proceso de comprobación.

En este caso, el guard especificado comprueba si el usuario está autenticado; en tal caso permite acceder a la ruta y en caso contrario lo redirige a /desapariciones.

src/app/modules/nousers/nousers.service.ts

```
...
export class NouserService implements CanActivate, CanLoad {
 ...
 canActivate():boolean {
   if (!this._userService.isAuthenticated) return true;

   this._router.navigate(['/desapariciones']);
   return false;
 }
}
...
```

Ruta /entrar

src/app/modules/nousers/nousers.routing.ts

```
...
{ path: 'entrar', loadChildren: './signin/
signin.module#SigninModule', canLoad: [ NouserService ]}
...
```

- **loadChildren**: Permite especificar el módulo que será cargado mediante carga diferida, como se puede ver éste módulo deberá ser indicado relativamente al fichero de rutas.

- **canLoad**: Permite especificar guards del tipo CanLoad, el cual, como se comentó anteriormente, decide si un módulo puede ser cargado. Dicho módulo será el especificado en loadChildren.

Adicionalmente a estas propiedades, también está presente children, el cual permite especificar rutas hijas.

6.1.5 Implementación de diseño

Para el diseño de la aplicación se ha seguido la técnica mobile first, la cual consiste en realizar el diseño desde un principio para terminales móviles y posteriormente adaptarlo para pantallas más grandes (Figura 30).

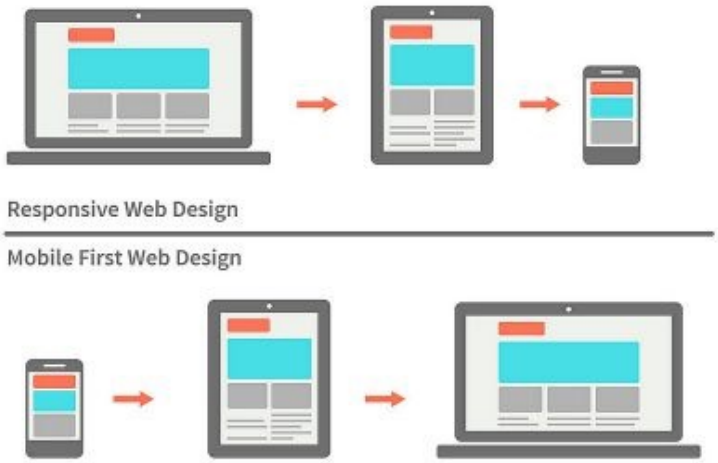

Figura 30. Técnicas de diseño

Para la implementación de los estilos, se ha usado el preprocesador SASS la cual está integrada en el desarrollo con Angular.

Con el fin de mejorar con el uso de CSS, se ha desarrollado un pequeño framework, llamado framywork, el cual se usa como base para el diseño de esta aplicación, evitando así el uso de frameworks externos.

6.1.6 PWA

Convertir la aplicación a un aplicación web progresiva, era un objetivo muy importante para el proyecto, debido a las ventajas y nuevas funcionalidad que una PWA ofrece, de las cuales, la funcionalidad de enviar notificaciones push es ideal para este proyecto.

La conversión a PWA fue inmediata, gracias, nuevamente, al uso de Angular 6, quién la integra ya en su desarrollo a partir de la versión 5. Para ello se ejecuta:

```
> ng add @angular/pwa
```

El cual modifica ciertos parámetros del proyecto y genera los ficheros ngsw-config.json y manifest.json en src/, las cuales, simplemente, hay que configurar.

Para comprobar que la PWA fue correctamente configurada, se puede usar la ventana Application de las herramientas de desarrollo de Google Chrome. En ella se puede ver si hay un service worker asociado a la aplicación y en tal caso mostrará algo similar a la Figura 31.

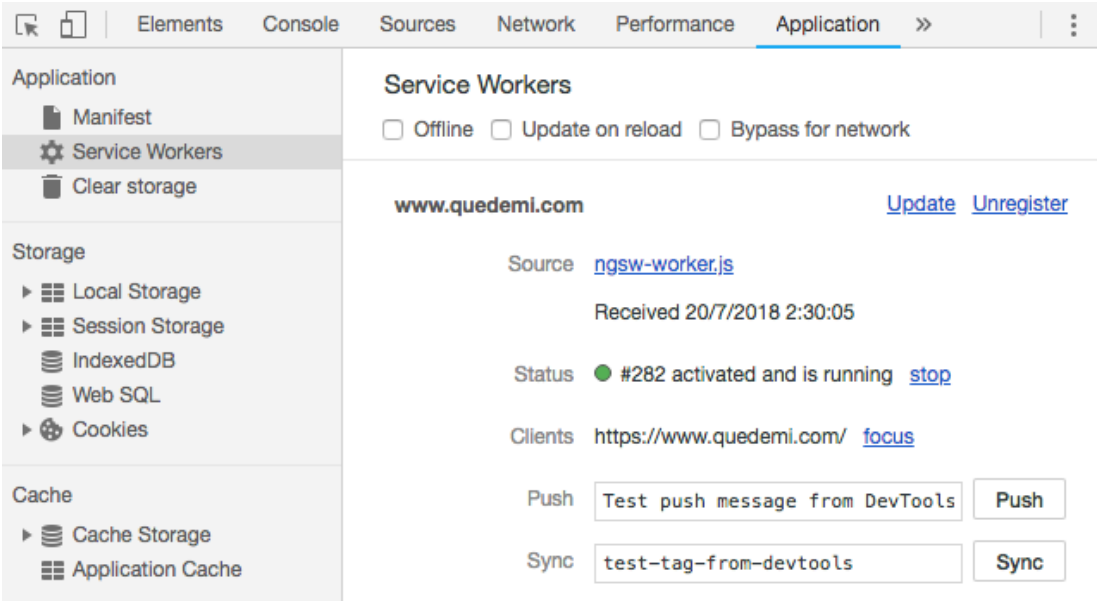

Figura 31. Service Worker en Google Chrome

En la misma ventana también se puede ver la configuración del archivo manifest.json (Figura 32).

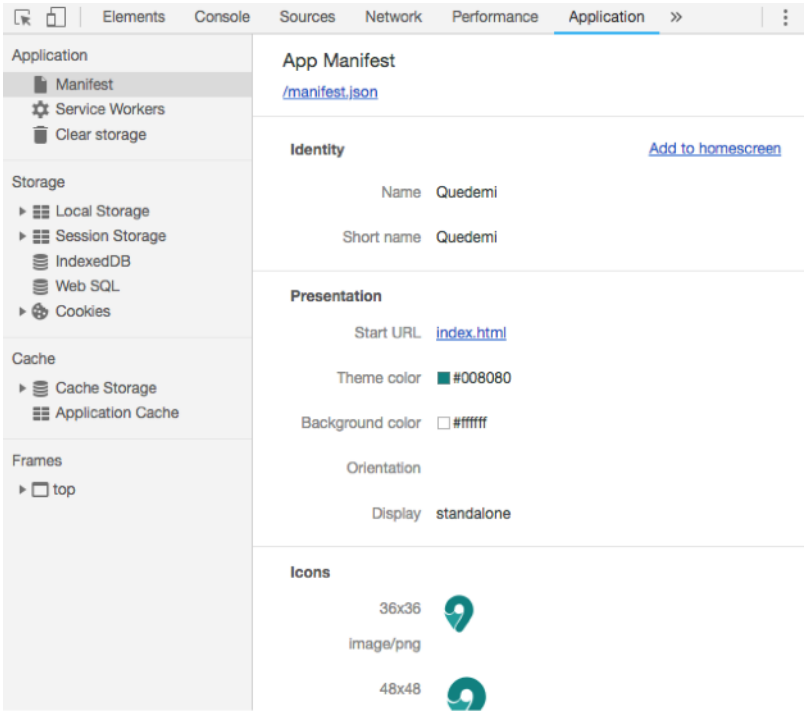

Figura 32. Manifest.json en Google Chrome

Tras la configuración de la PWA, se implementó la funcionalidad para suscribirse a recibir notificaciones push.

```
/src/app/modules/users/services/push-notification.service.ts
```

```
...
subscribe(): Promise<any> {
  return this._swPush.requestSubscription({
     serverPublicKey: environment.vapidPublicKey
   })
   .then(sub => {
     this.subscription = sub;

this._userService.updatePushSubscription({ pushSubscription: sub
}).toPromise()
   })
   .then(data => {
     this.subscribed = true;
     return data;
   })
   .catch(err => console.error("NO SE HA PODIDO SUSCRIBIR:",
err));
}
...
```

Dicha petición de suscripción se realiza cuando el usuario especifica su localización.

6.1.7 Módulos desarrollados

Durante el desarrollo de la aplicación web se ha desarrollado un módulo para Angular llamado submittable. Este módulo sirve para mostrar una animación de carga en los formularios en los que se indique la directiva submittable y realicen una petición HTTP mediante HTTPClient. Adicionalmente, junto a la animación, se puede especificar un texto de la siguiente forma:

```
/src/app/modules/users/missings/missing-edit/missing-edit.component.html
```

```
...
<form [formGroup]="missingEditForm" (ngSubmit)="onSubmit(s)"
submittable #s="submittable">
  ...
  <div class="form-actions center">
    <button type="submit" class="btn btn-primary btn-submit">
      <span class="loading-button-normal-text">Guardar</span>
      <app-loader-submit>Guardando</app-loader-submit>
    </button>
   </div>
</form>
...
```

La animación se mostrará cuando se ejecute el método submit, la cual recibe como primer parámetro el observable que retorna la petición HTTP; y se ocultará cuando dicha petición haya sido resuelta. Opcionalmente, cuando la petición haya sido resuelta exitosamente, se puede mostrar un aviso con el mensaje que se indique como segundo parámetro del método submit. En la Figura 33 se puede ver su funcionamiento.

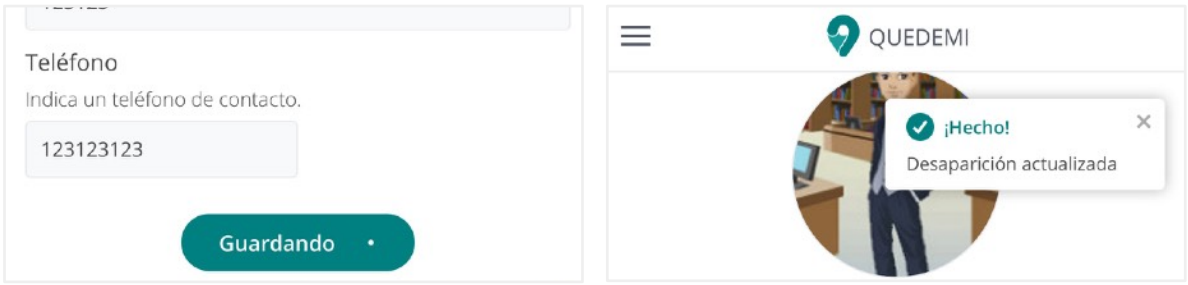

Figura 33. Animación submittable

6.1.8 Problemas encontrados

Los principales problemas encontrados durante el desarrollo de la aplicación web, ocurrieron tras su conversión a PWA. Dichos problemas tenían que ver con el funcionamiento de la PWA en dispositivos móviles:

- Las aplicaciones web progresivas no ofrecen la funcionalidad de envío de notificaciones push en los dispositivos móviles ios.

- No es posible autenticar usuarios mediante aplicaciones externas, puesto que después de autenticarse en alguna de ellas, no se vuelve a la PWA para completar el proceso.

En consecuencia a los problemas comentados, se decidió crear la aplicación móvil.

6.2 Aplicación móvil

En este apartado se describirá el desarrollo de la aplicación móvil del proyecto.

La aplicación móvil consiste en una aplicación híbrida desarrollada con Ionic 3, el cual está basado en Angular. La versión de Angular que corresponde con Ionic 3 es la 5, sin embargo los módulos de Angular han sido actualizados para usar su última versión disponible, actualmente, la 6.

La aplicación móvil, si bien no implementa todas las funcionalidades previstas, sí implementa aquellas que representan el producto mínimo viable; las funcionalidades no implementadas tienen que ver con la edición de datos del usuario.

6.2.1 Implementación

Como ya se explicó, en Angular el diseño está basado en componentes, sin embargo en Ionic lo está en páginas. Una página corresponde con una vista que ocupará toda la pantalla del dispositivo, por lo que sólo una puede estar activa a la vez.

De igual forma, en Ionic a diferencia que Angular, la navegación está menos orientada a usar URLs, y más orientada al tipo de navegación que se hace desde un móvil, donde es habitual moverse adelante y atrás, o pasar de repente a una sección completamente distinta.

El concepto básico de la navegación con Ionic es la pila de páginas. Esto es, básicamente, que las páginas se van apilando una encima de otra; cada vez que pasamos a la siguiente página, se realiza push de dicha página, y cuando la quitamos, pop.

Para implementar las funcionalidades de la aplicación móvil, se han reutilizado componentes de la aplicación web, adaptándolas al desarrollo basado en páginas.

De igual forma, para implementar su diseño se ha incorporado el framework CSS desarrollado y si bien, se ha seguido sin utilizar un framework CSS externo, Ionic implementa una serie de componentes con un diseño predefinido que se asemejan, en gran medida, a los componentes visuales nativos; por lo que con el objetivo que la aplicación resulte lo más parecida a una aplicación nativa, utilizaremos dichos componentes.

6.2.2 Cordova plugins

Los plugins de Cordova nos permitirán acceder a las APIs nativas del dispositivo. De cuales en la aplicación móvil usamos:

- **cordova-plugin-facebook4**: Añade SDK de Facebook al proyecto. Se usa para permitir la autenticación mediante Facebook.
- **cordova-plugin-googleplus**: Permite la autenticación mediante Google.
- **cordova-plugin-statusbar**: Proporciona algunas funciones para personalizar el StatusBar de iOS y Android.
- **ionic-plugin-deeplinks**: Permite especificar una esquema de URL, con la cual se abrirá la aplicación directamente.

Capítulo 7. Despliegue

En este capítulo se describe el proceso de despliegue de los componentes del proyecto en una máquina virtual de digitalocean, con las siguientes características (Figura 34).

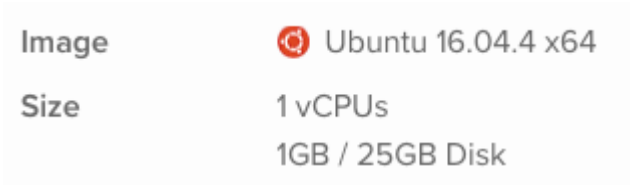

Figura 34. Características droplet Digital Ocean

También se ha usado el servicio de hosting de DNS de digitalocean para configurar el dominio quedemi.com y los subdominios que se usarán para acceder a dichos componentes. Dicha configuración se puede ver en la Figura 35.

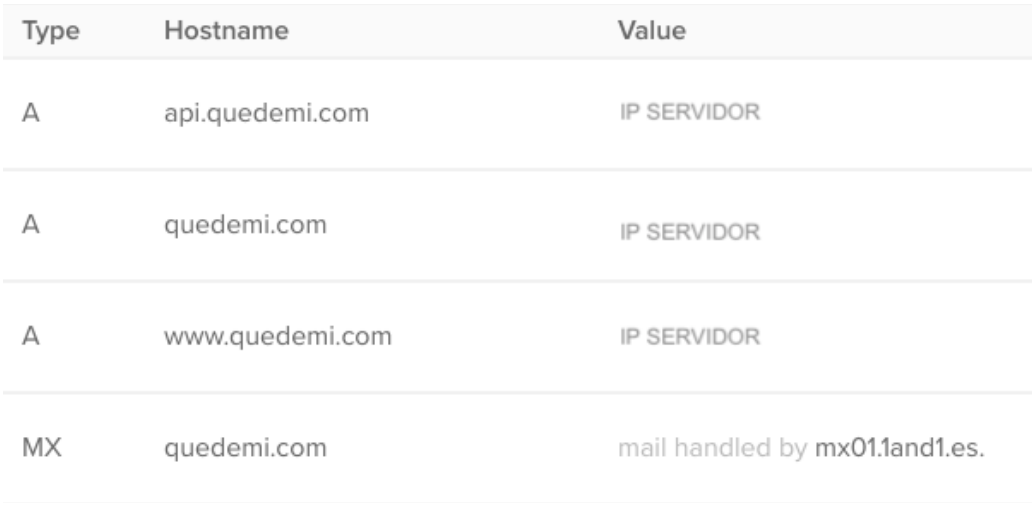

Figura 35. Registros DNS de dominio

Para el despliegue de los componentes se usa Nginx.

Nginx es un servidor web orientado a eventos (como Node) que puede actuar como un servidor proxy y es quien nos permite usar los múltiples dominios para acceder a los componentes desplegados, gracias a bloques de servidor, como se describirá posteriormente.

7.1 Configuración de Nginx

Nginx nos permitirá configurar los múltiples dominios que queremos usar para acceder a los componentes del proyecto a través de bloques de servidor. Adicionalmente, para el caso de la API, utilizaremos a Nginx como servidor proxy el cual nos permitirá redireccionar el tráfico entrante desde el dominio api.quedemi.com, hacia el puerto donde la API se esté ejecutando.

La forma de crear un bloque de servidor en Nginx para servir una página web estática, la cual será accesible mediante el dominio ejemplo.com es la siguiente:

1. Crear el directorio ejemplo.com en /var/www/ejemplo.com con el contenido del proyecto.
2. Crear un archivo de configuración en /etc/nginx/sites-available con el nombre ejemplo.com.conf con la siguiente implementación:

```
/etc/nginx/sites-available/api.quedemi.com
server {
     listen 80;
     listen [::]:80;

     server_name ejemplo.com;

     root /var/www/ejemplo.com;
     index index.html;

     location / {
          try_files $uri $uri/ =404;
     }
}
```

3. Habilitar el bloque de servidor enlazando el archivo de configuración en /etc/nginx/sites-enabled.
4. Reiniciar el servicio de nginx

En los siguientes apartados se describe el proceso y configuración llevada a cabo para los distintos componentes del proyecto.

7.2 API

Para su despliegue se clonó la rama del proyecto en /var/www/api.quedemi.com y se creo el fichero de configuración del bloque de servidor de la siguiente forma:

```
/etc/nginx/sites-available/api.quedemi.com
```

```
server {
  server_name api.quedemi.com;

  location / {
    proxy_pass http://localhost:8080;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
    proxy_cache_bypass $http_upgrade;
  }

    listen [::]:443 ssl http2; # managed by Certbot
    listen 443 ssl http2; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/api.quedemi.com/
fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/
api.quedemi.com/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; #
managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed
by Certbot

}
server {
  listen 80;
  listen [::]:80;
  server_name api.quedemi.com;
  return 301 https://api.quedemi.com$request_uri;
}
```

Para la ejecución del servidor se usó el módulo pm2, el cual nos permite gestionar el proceso de ejecución de la API, así como establecer una configuración para el entorno de producción mediante el fichero ecosystem.config.js, situada en el directorio raíz del proyecto. Además, gracias a pm2 se garantiza que la api esté siempre corriendo, puesto que ante un fallo que pare su ejecución, éste se encargará de reiniciarlo.

7.3 Aplicación web

El despliegue de la aplicación web supuso solucionar una serie de problemas:

- Poder enrutar todas las rutas al fichero index.html.
- Añadir un certificado SSL con el dominio personalizado de forma gratuita.

Las cuales, gracias a Nginx, se solucionaron fácilmente; sin embargo antes de usar el VPS con Nginx, ya se había probado múltiples plataformas de despliegue, los cuales se comentarán posteriormente.

De la misma forma que con la API, se creó el archivo de configuración del servidor del bloque en /var/www/api.quedemi.com, con la siguiente implementación:

```
/etc/nginx/sites-available/api.quedemi.com
```

```
server {
  listen [::]:443 ssl http2 ipv6only=on;
  listen 443 ssl http2;
  server_name quedemi.com www.quedemi.com;
  if ($host = quedemi.com){
    return 301 https://www.quedemi.com$request_uri;
  }
  root /var/www/quedemi.com/html;
  index index.html;
  location / {
    try_files $uri $uri/ /index.html;
  }
  ssl_certificate /etc/letsencrypt/live/quedemi.com/
fullchain.pem; # managed by Certbot
  ssl_certificate_key /etc/letsencrypt/live/quedemi.com/
privkey.pem; # managed by Certbot
  include /etc/letsencrypt/options-ssl-nginx.conf; # managed
by Certbot
  ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by
Certbot
}
server {
  listen 80;
  listen [::]:80;
  server_name quedemi.com www.quedemi.com;
  return 301 https://www.quedemi.com$request_uri;
}
```

Como se puede ver en la configuración el contenido del proyecto debe estar ubicado en /var/www/quedemi.com/html. El proceso de subir dicho contenido al servidor se automatizó mediante una tarea de integración continua, cuya configuración está en el fichero gitlab-ci.yml. Dicha tarea realiza las siguientes acciones:

1. Generar los archivos de producción
2. Subir los archivos generados al servidor con el uso de los comandos rsync, sshpass y con el usuario uploader del servidor.

Capítulo 8. Presupuesto

A continuación se presenta el desglose del presupuesto necesario para la puesta en producción, mantenimiento y desarrollo de las los componentes que se han expuesto en el presente documento.

8.1 Personal

Para determinar los costes de personal se calculará el coste asociado a cada fase según sus jornadas requeridas. Tomando como base el precio de 15€ por hora, y teniendo en cuenta que una jornada está compuesta por 5 horas, el precio por jornada equivale a 75€. En la Tabla 1 se puede ver el calculo de los costes de cada fase.

Fase	Jornadas	Coste
Análisis funcional y diseño técnico de la aplicación	5	375€
Implementación API REST	30	2.250€
Implementación aplicación web	25	1.875€
Implementación aplicación móvil	22	1.650€
Despliegue de los componentes	2	150€

Tabla 1. Costes de fases del proyecto

8.2 Servicios y licencias

En Tabla 2 se especifican los costes de los servicios utilizados para el despliegue de los componentes del proyecto, así como las licencias requeridas. El coste total es aproximado puesto que los servicios de Amazon variarán dependiendo de factores como la transferencia de datos, peticiones realizadas, entre otros.

Servicio	Coste
Amazon EC2	15€ / mes
Amazon S3	1€ / mes
Droplet Digital Ocean	5€ / mes
Dominio	12€ / año
Licencia desarrollador Android	25€
Licencia desarrollador Apple	99€ / año

Tabla 2. Costes de servicios y licencias

8.3 Costes totales

En la tabla 3, se muestra el precio total del desarrollo, junto a los precios totales por mes y año, que conllevan mantener los servicios y licencias contratados.

Parte	**Coste**
Personal	6.300€
Servicios y licencias	157€
Total	6.457€
Total / mes	21€
Total / año	111€

Tabla 3. Costes totales

Capítulo 9. Conclusiones y trabajo futuro

El desarrollo de este proyecto ha dado lugar a la implementación de la API, aplicación web y aplicación móvil que componen Quedemi.

Quedemi es una herramienta que ayuda en la difusión de desapariciones mediante el uso de redes sociales y envío de notificaciones. Su desarrollo me ha permitido seguir aprendiendo javascript, mi lenguaje de programación favorito, y utilizar múltiples herramientas por primera vez, sobre todo, las relacionadas con el despliegue del proyecto; aparte de Ionic, el cual estoy seguro, utilizaré en proyectos posteriores.

Ha sido una gran experiencia que por supuesto no acaba aquí, los componentes implementados seguirán siendo desarrollados con el propósito de mejorar las funcionalidades ofrecidas e incorporar nuevas que puedan servir de ayuda en este difícil proceso que conlleva la pérdida de un ser querido.

Las nuevas funcionalidades que se implementarán serán:

- Creación de anuncios de animales encontrados, según se sospeche que el animal está perdido o para su adopción.
- Uso de geolocalización en tiempo real para el envío de notificaciones.

Sin embargo antes de la implementación de nuevas funcionalidades se mejorarán:

- Implementación completa de la aplicación móvil, puesto que por falta de tiempo no se pudieron implementar todas las funcionalidades
- Unificación de los proyectos de la aplicación web y móvil en uno solo, mediante Ionic 4.

Adicionalmente se documentará las funcionalidades de la API, para que pueda ser libremente usada y estudiará la posibilidad de integrar APIs externas.

Capítulo 10. Conclusions and future works

The development of this project has led to the implementation of the API, web application and mobile application that make up Quedemi.

Quedemi is a tool that helps in the dissemination of disappearances through the use of social networks and sending notifications. Its development has allowed me to continue learning javascript, my favorite programming language, and to use multiple tools for the first time, especially those related to the deployment of the project; Apart from Ionic, which I am sure, I will use in later projects.

It has been a great experience that of course does not end here, the components implemented will continue to be developed with the purpose of improving the functionalities offered and incorporating new ones that can help in this difficult process that entails the loss of a loved one.

The new features that will be implemented will be:

Creation of announcements of animals found, depending on whether the animal is suspected of being lost or for its adoption.

Use of geolocation in real time to send notifications.

However, before the implementation of new functionalities, the following will be improved:

Complete implementation of the mobile application, since due to lack of time it was not possible to implement all the functionalities

Unification of the projects of the web and mobile application into one, through Ionic 4.

Additionally, the functionalities of the API will be documented, so that it can be freely used and will study the possibility of integrating external APIs.

Bibliografía

[1] En 2017 se registraron un total de 29.794 denuncias de desaparición [online]. Disponible en: http://www.expansion.com/sociedad/2018/05/06/5aef1bee268e3e432f8b4597.html

[2] En España cada día desaparecen tres personas que nunca son encontradas [online]. Disponible en: https://www.cope.es/actualidad/sociedad/noticias/espana-cada-dia-desaparecen-tres-personas-que-nunca-son-encontradas-20180305_173826

[3] Wizapet, no lo des por perdido [online]. Disponible en: https://www.wizapet.com

[4] Registro Canario de Identificación Animal [online]. Disponible en: http://www.zoocan.net/

[5] Anuncios para encontrar a tu perros [online]. Disponible en: http://www.perrosperdidos.es/

[6] Anuncios de perros perdidos actualizados [online]. Disponible en: http://animales-perdidos.org/

[7] Asociación sosdesaparecidos [online]. Disponible en: http://sosdesaparecidos.es/

[8] ¿Qué es JavaScript? [online]. Disponible en: https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/Qu%C3%A9_es_JavaScript

[9] Express, página oficial [online]. Disponible en: http://expressjs.com/es/

[10] Postman, página oficial [online]. Disponible en: https://www.getpostman.com/

[11] MongoDB, página oficial en español [online]. Disponible en: https://www.mongodb.com/es

[12] Amazon EC2, página oficial en español [online]. Disponible en: https://aws.amazon.com/es/ec2/

[13] Amazon S3, página oficial en español [online]. Disponible en: https://aws.amazon.com/es/s3/

[14] Robot 3T, página oficial [online]. Disponible en: https://robomongo.org/

[15] Progressive Web Apps [online]. Disponible en:

https://developers.google.com/web/progressive-web-apps/

[16] Gitlab, página oficial[online]. Disponible en: https://about.gitlab.com/

[17] How To Secure Nginx with Let's Encrypt on Ubuntu 16.04 [online]. Disponible en: https://www.digitalocean.com/community/tutorials/how-to-secure-nginx-with-let-s-encrypt-on-ubuntu-16-04

[18] JSON Web Tokens [online]. Disponible en: https://jwt.io/

[19] ¿Qué es Nginx y cómo funciona? [online]. Disponible en: https://kinsta.com/es/base-de-conocimiento/que-es-nginx/

Apéndice A. Instalación y configuración de MongoDB.

Posteriormente se realizó la instalación de MongoDB mediante los siguientes comandos:

```
> sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80
--recv 0C49F3730359A14518585931BC711F9BA15703C6 # importar la
llave para el repositorio oficial de MongoDB

> echo "deb [ arch=amd64,arm64 ] http://repo.mongodb.org/apt/
ubuntuxenial/mongodb-org/3.4 multiverse" | sudo tee /etc/apt/
sources.list.d/mongodb-org-3.4.list # crear la lista para
MongoDB

> sudo apt-get update # actualizar la lista de paquetes

> sudo apt-get install -y mongodb-org # instalar mongodb
```

Una vez instalado MongoDB, los siguientes comandos nos permiten controlar el servicio de MongoDB.

```
> sudo systemctl mongod start # iniciar MongoDB

> sudo systemctl mongod stop # parar MongoDB

> sudo systemctl mongod restart # reiniciar MongoDB

> sudo systemctl mongod status # ver el estado del servicio
de MongoDB (activo, inactivo)

> sudo systemctl enable mongod # iniciar MongoDB al inicio
del sistema
```

Tras la instalación de MongoDB, con la finalidad de activar la autenticación y permitir las conexiones remotas, se editó su archivo de configuración ubicado en /etc/mongod.conf modificando los siguientes campos.

```
/etc/mongod.conf
```

```
...
security:
  authorization: enabled
...
net:
  port: 8080
  bindIp: 0.0.0.0
...
```

Para que se apliquen los cambios se debe reiniciar el servicio de MongoDB, en caso de estar activo.

Posteriormente se creó el usuario administrador de la base de datos mediante los siguientes comandos, teniendo MongoDB activo:

```
> mongo
mongo > use admin
mongo > db.createUser({user: "didream", pwd: "***", roles:
[{role: "root", db: "admin"}]})
mongo > quit()
```

Para comprobar la correcta creación del administrador se ejecuta:

```
> mongo -u didream -p --authenticationDatabase admin
```

lo que solicita posteriormente la contraseña establecida.

Apéndice B. Configuración del servidor de despliegue

Nginx

Para instalar Nginx en el servidor, se ejecutó:

```
> sudo apt-get update
> sudo apt-get install nginx
```

En la instalación, Nginx se registra como servicio en ufw; el software de firewall instalado en el servidor, con los siguientes perfiles:

- Nginx Full: Abre tanto el puerto 80 (tráfico web normal, sin cifrar) como el puerto 443 (tráfico cifrado TLS / SSL)
- Nginx HTTP: Abre sólo el puerto 80 (normal, tráfico web no cifrado)
- Nginx HTTPS: Abre sólo el puerto 443 (tráfico cifrado TLS / SSL)

Por lo que tras la instalación, se activó el perfil Nginx Full en ufw, mediante:

```
> sudo ufw allow 'Nginx Full
```

Estos perfiles pueden listarse mediante el comando:

```
> sudo ufw app list
```

Posteriormente, con el fin de automatizar la tarea de despliegue, se creó un usuario "uploader" quien se cargará de subir el contenido actualizado de los proyectos, accediendo al servidor mediante ssh y contraseña. Para ello hubo que habilitar la autenticación mediante contraseña en ssh y cambiar los permisos del directorio /var/www, en el cual se almacenarán los proyectos.

Certificado SSL

Con el fin de obtener e instalar un certificado SSL de forma gratuita para cada componente desplegado, se usó la herramienta certbot que automatiza los pasos requeridos para este proceso.

Para ello se ha seguido la siguiente el artículo [ ]

HTTP/2

Para habilitar el protocolo HTTP/2 simplemente se añadió http2 en la directiva listen de la configuración del bloque de servidor

/etc/nginx/sites-available/api.quedemi.com

```
server {
  listen [::]:443 ssl http2 ipv6only=on;
  listen 443 ssl http2;
  ...
}
```

gzip

Para habilitar fginx/nginx.conf, modificando los siguientes campos:

/etc/nginx/sites-available/api.quedemi.com

```
...
gzip on;
gzip_disable "msie6";

gzip_vary on;
gzip_proxied any;
gzip_comp_level 6;
gzip_buffers 16 8k;
gzip_http_version 1.1;
gzip_min_length 256; #no comprimir ficheros que ocupen menos
de 256 bytes
gzip_types text/plain text/css application/json application/
x-javascript text/xml application/xml application/xml+rss
text/javascript application/vnd.ms-fontobject application/x-
font-ttf font/opentype image/svg+xml image/x-icon;
...
```

Apéndice C. Despliegues previos de la Aplicación web.

En este punto, se describirán, brevemente, los problemas encontrados durante el despliegue en anteriores plataformas a VPS de Digital Ocean con Nginx. Estos problemas tienen que ver con la adición gratuita de un certificado SSL a la aplicación, y para enrutar todas las rutas a index.html, que es dónde se carga toda la aplicación.

Surge.sh

En primer lugar, la aplicación se desplegó en la plataforma Surge.sh, la cual provee un sencillo despliegue de una página web estática por línea de comandos.

Para solucionar el problema de las rutas fue necesario renombrar el fichero index.html a 200.html, puesto que surge.sh devuelve, por defecto, este fichero para todas las peticiones exitosas.

El problema con este servicio apareció cuando se intentó añadir un certificado SSL con el dominio personalizado quedemi.com, ya que esta operación no se puede llevar a cabo con una suscripción gratuita.

Amazon S3

Este servicio de almacenamiento puede ser configurado para servir como alojamiento de un sitio web estático accesible a través de una url proporcionada o mediante un dominio personalizado. Los problemas con este servicio fueron:

- El coste, aunque poco, del uso del dominio personalizado.
- El problema de las rutas antes comentada.
- El incómodo despliegue de la aplicación. Puesto que no se supo automatizar esta tarea.

Heroku

Para su despliegue en Heroku, fue necesario crear un servidor en Node.js, que simplemente sirviese el fichero index.html de la aplicación para cualquier ruta, solucionando así el problema de ruteo. El problema con este servicio fue el mismo que el comentado con Surge.sh: no es posible añadir un certificado SSL con una suscripción gratuita.

Gitlab Pages

Gitlab al igual que Github, ofrecen un servicio de alojamiento de contenido web estático, sin embargo con Gitlab es necesario configurar el fichero de integración continua gitlab-ci.yml, con la respectiva **tarea** que realice el despliegue.

Con Gitlab fue posible solucionar los problemas planteados, sin embargo la solución al problema de navegación no es del todo ideal. Ésta consistió en renombrar index.html a 404.html, que es el fichero que se devuelve en consecuencia de un código HTTP 404, cuando no se encuentra el recurso solicitado mediante la ruta