Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

CityOnContact: Aplicación web para la comunicación ciudadana

CityOnContact: Web Application for citizen communication

Borja Carmelo García González

La Laguna, 6 de julio de 2020

D. **Vicente José Blanco Pérez**, con N.I.F. 42.171.808-C profesor Titular de Universidad adscrito al Departamento de Estadística, Investigación Operativa y Computación de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

"CityOnContact: Apicación web para la comunicación ciudadana"

ha sido realizada bajo su dirección por D. **Borja Carmelo García González**, con N.I.F. 43.485.532-F.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 6 de julio de 2020

// Agradecimientos

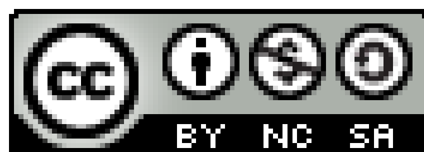
En plena crisis sanitaria a nivel mundial, prolongada durante meses de inestabilidad, incertidumbre, miedo y reflexión, creo que es apropiado comenzar este apartado de la memoria dando mi agradecimiento a todas las trabajadoras, sea cual sea su sector, que han hecho posible que podamos haber sobrellevado esta situación enfrentándose a ella en primera línea.

Aunque desde un punto de vista más personal, a quien debo dar las gracias ante todo es a mi familia, con quienes he convivido esta pasada cuarentena y quienes han tenido que soportar mis malos humores y fallos; junto a las personas que han querido escucharme. Es algo que suelo olvidar y que debería tener más presente. Gracias.

Por supuesto quiero agradecer también a Vicente su labor como tutor, siempre disponible y dispuesto a resolver cualquier duda y sugerir las herramientas adecuadas.

Por último, creo que también es justo agradecer a Lean Mind y a mis compañeros aprendices, con quienes cursé mis Prácticas Externas, todo lo que he mejorado como desarrollador durante estos meses. Este proyecto habría sido distinto sin esa experiencia previa.

Licencia



© Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional.

Resumen

La idea en la que se basa la aplicación desarrollada en este trabajo es la de mejorar las comunicaciones entre los organismos gubernamentales a nivel local y la ciudadanía, ofreciendo información relevante que afecte directamente a ambas partes de forma bidireccional.

Para ello se divide en un apartado Avisos, donde el ayuntamiento se encarga de aportar información al ciudadano recogida en distintas categorías (Eventos, Información, Avisos y Emergencias); y un apartado Comunica, donde son los ciudadanos quienes informan sobre el estado de la localidad en forma de Incidencias y Sugerencias.

Dado que se ha dispuesto de un tiempo limitado para desarrollar una aplicación completa, especialmente teniendo en cuenta la inexperiencia del alumno, este trabajo se ha planteado como un pet project en el que se prima el ejercitar el uso de buenas prácticas en el desarrollo software sobre la implementación rápida y descuidada de todas las funcionalidades.

La aplicación está basada en el stack MEVN (MongoDB - Express - Vue.js - Node). En lugar de usar una instancia local de MongoDB como base de datos, se ha optado por usar Atlas, su equivalente en la nube, por la flexibilidad que supone. El back end se ha estructurado con una arquitectura básica de tres capas sobre un servidor Express, y se ha hecho uso del framework Quasar para el desarrollo del front end, que provee la estructura básica para implementar una Single Page Application (SPA) basada en Vue.js y facilita el prototipado de la interfaz de usuario con su propia librería de componentes.

Se ha tratado de respetar en la medida de lo posible el ciclo de Desarrollo Guiado por Pruebas (TDD - Test-Driven Development), así como la aplicación de principios y patrones de diseño para un código limpio que resulte mantenible e intuitivo, destacando la refactorización constante del código.

Palabras clave: Ciudad, Comunicación, Aplicación Web, Full Stack, Quasar, Vue.js, Node, Express, MongoDB Atlas, JavaScript, Travis CI, Codecov, Leaflet, Heroku

Abstract

The idea on which the application developed for this project is based is to improve communications between local government entities and citizens, offering relevant information that directly affects both sides in a bidirectional way.

To achieve this, the application is divided into a Notices section, where the city council is responsible for providing information to the citizen collected in different categories (Events, Information, Notices and Emergencies); and a Communicate section, where it is the citizens who report on the state of the town in form of Incidents and Suggestions.

Given that time has not been enough to develop a complete application, especially taking into account the inexperience of the student, this project has been considered a pet project where the use of good practices in software development is more important than a fast and careless implementation of all features.

The application is based on the MEVN Stack (MongoDB - Express - Vue.js - Node). Instead of using a local instance of MongoDB as the database, Atlas, its cloud equivalent, has been used due to the flexibility it offers. The back end has been structured with a basic three-layer architecture on an Express server, and the Quasar framework has been used for the development of the front end, which provides the basic structure to implement a Single Page Application (SPA) based on Vue.js and makes it easy to prototype the user interface with its own component library.

The Test-Driven Development (TDD) cycle has been applied as much as possible, as well as the application of design principles and patterns for a clean code that is maintainable and intuitive, with a continuous refactoring of the code.

Keywords: *City, Communitacion, Web Application, Full Stack, Quasar, Vue.js, Node, Express, MongoDB Atlas, JavaScript, Travis CI, Codecov, Leaflet, Heroku*

Índice general

1. **Introducción** 1
   - 1.1. Antecedentes y estado actual del tema 1
   - 1.2. Tecnologías usadas en el Proyecto 2
   - 1.3. Planificación del Desarrollo 3
2. **Tecnologías y Herramientas** 5
   - 2.1. Herramientas de Desarrollo 5
     - 2.1.1. Control de versiones: Git y GitHub 5
     - 2.1.2. Cubrimiento de código: Codecov y ESLint 5
     - 2.1.3. Integración continua: Travis CI 6
     - 2.1.4. Despliegue en la nube: Heroku 6
   - 2.2. Back end 6
     - 2.2.1. MongoDB Atlas 6
     - 2.2.2. Servidor Express 6
   - 2.3. Front end 7
     - 2.3.1. Quasar Framework 7
     - 2.3.2. Enrutamiento de una SPA 7
     - 2.3.3. Control de estado con Vuex 8
     - 2.3.4. Leaflet 9
   - 2.4. Gestión de la sesión: JWT 9
   - 2.5. Test-Driven Development 11
3. **Desarrollo de la Aplicación** 12
   - 3.1. Prototipado 12
   - 3.2. Entorno de desarrollo 15
   - 3.3. Configuración de Travis CI y Codecov 16
   - 3.4. Back end 17
     - 3.4.1. Introducción: Arquitectura de 3 Capas 17
     - 3.4.2. Estructura 19
     - 3.4.3. Testing 21
     - 3.4.4. Dependencias Usadas 21
   - 3.5. Front end 22
     - 3.5.1. Configuración inicial de Quasar 22
     - 3.5.2. Estructura 23

3.5.3. Enrutamiento 24
3.6. Implementación de la sesión con JWT 26
3.6.1. Back end 26
3.6.2. Front end 28
3.7. Inclusión de Mapas Interactivos con Leaflet 29

4. Descripción de la Aplicación **31**
4.1. Introducción 31
4.2. Página de Inicio: Sección de Avisos 31
4.3. Contenido de una Categoría 35
4.4. Autenticación 36
4.5. Sección Comunica 37

5. Conclusiones y líneas futuras **39**

6. Summary and Conclusions **41**

7. Presupuesto **43**

Índice de Figuras

2.1. Flujo de una SPA en comparación a una arquitectura MVC extraído de Excellent Webworld [36] . . . 8
2.2. Flujo de Vuex extraído de un artículo de Lauri Hiltunen [16] . . . 9
2.3. Ciclo del Patrón Refresh Token extraído de un artículo en Auth0 [1] . 11

3.1. Prototipo incial de la pantalla principal . . . 13
3.2. Prototipo inicial de las pantallas de categorías . . . 14
3.3. Prototipo inicial de la pantalla de Sugerencias y Perfil de usuario . . . 15
3.4. Ejemplo de una Arquitectura Hexagonal completa extraído de un artículo de Francisco Ugalde [33] . . . 18
3.5. Ejecución de los tests del back end . . . 21
3.6. Configuración inicial de Quasar . . . 22

4.1. Pantalla de inicio en la versión móvil . . . 32
4.2. Pantalla de inicio en la versión de escritorio . . . 32
4.3. Categoría de Eventos en la versión de escritorio . . . 33
4.4. Calendario y Municipios en la versión móvil . . . 34
4.5. Calendario en la versión de escritorio . . . 34
4.6. Municipios en la versión de escritorio . . . 35
4.7. Evento en la versión móvil . . . 35
4.8. Evento en la versión de escritorio . . . 36
4.9. Formulario de Login en la versión móvil . . . 36
4.10Formulario de Registro en la versión de escritorio . . . 37
4.11Sección Comunica en la versión móvil . . . 38
4.12Sección Comunica en la versión de escritorio . . . 38

7.1. Pricing para el alojamiento en Heroku . . . 43
7.2. Pricing de MongoDB Atlas . . . 44
7.3. Pricing de Google Cloud . . . 44

Índice de Tablas

1.1. Tareas planteadas 4
7.1. Estimación del presupuesto para el desarrolo del proyecto en 6 meses 45

Capítulo 1

Introducción

Este capítulo sirve de introducción al planteamiento del proyecto, recogiendo un resumen de las motivaciones y el estado del tema en el que está centrado, las características de la aplicación y la planificación del desarrollo.

1.1. Antecedentes y estado actual del tema

Al consultar el portal web de un ayuntamiento español en la actualidad, aunque los recursos disponibles e incluso lo cuidada que esté la interfaz de usuario en muchos casos dependa de la magnitud de la localidad, en general suelen aparecer las mismas secciones comunes: transparencia del gobierno, gestión de trámites, noticias, agenda, información sobre servicios culturales, deportivos, educativos, atención social, medio ambiente, igualdad, etc.

De forma adicional, las ciudades relativamente grandes suelen disponer de aplicaciones móviles municipales, cada una cubriendo una necesidad específica. Como ejemplo, el Ayuntamiento de Santa Cruz de Tenerife dispone de aplicaciones como SC Mejora [7], donde se permite avisar de incidencias al personal de mantenimiento; SC Farmacias [6], para ubicar farmacias abiertas cercanas; SC Viva [8], que ofrece información cultural municipal; entre otras.

Al consultar estas aplicaciones en Google Play se puede apreciar como se encuentran atendidas de una forma desigual. Mientras algunas parecen recibir un feedback mayoritariamente positivo por parte de los usuarios, hay otras que parecen totalmente descuidadas por los desarrolladores e inutilizadas por los usuarios. Incluso la aplicación de incidencias de una ciudad tan importante como Madrid parece tener bastantes problemas relacionados con la experiencia de usuario.

Teniendo esto en cuenta, la presente propuesta de aplicación trata de unificar la información de mayor relevancia práctica para los ciudadanos, que en los portales web municipales suele estar incompleta y no cubre aspectos como los avisos o emergencias; además de centrarse en la sencillez y comodidad de uso.

Se debe considerar también la tendencia actual al desarrollo de ciudades inteligentes, de hecho la implementación de servicios telemáticos o las aplicaciones municipales mencionadas, podrían considerarse una primera aproximación a esta nueva forma de organización. Evidentemente, para alcanzar este siguiente nivel no basta con el desarrollo de software, se necesita una inversión en infraestructuras tecnológicas sobre las que apoyarse, pero ya existen proyectos que tratan de ofrecer un acercamiento más trabajado con vistas a largo plazo.

Un ejemplo interesante de esto es SmartAppCity [4], una aplicación móvil multiplataforma que trata de abarcar todos los servicios que ofrece una ciudad de forma modular, ajustándose a los requisitos de cada administración local. Actualmente se encuentra implementada incluso en localidades españolas como Logroño o Toledo.

CityOnContact, la aplicación propuesta, no pretende llegar a cubrir todos los servicios ni promocionar la cultura, comercio o turismo municipales. En su lugar, apuesta por la simplicidad concretando su utilidad en ofrecer información necesaria actualizada y mejorar la comunicación ciudadano-administración.

1.2. Tecnologías usadas en el Proyecto

La aplicación está basada en el stack MEVN (MongoDB - Express - Vue.js - Node). Se ha optado por usar Atlas, la versión de MongoDB en la nube, por la flexibilidad que supone. El back end se ha estructurado con una arquitectura básica de tres capas sobre un servidor Express, y se ha hecho uso del framework Quasar, basado en Vue.js, para el desarrollo del front end.

A continuación se recoge un listado de las tecnologías y herramientas que han sido usadas durante el desarrollo del proyecto, en las que se profundizará en los siguientes apartados de la memoria.

- **Back end**
  - MongoDB Atlas [22]
  - Mongoose [23]
  - Node [24]
  - Express [10]
- **Front end**
  - Quasar [11]
  - Vue.js [34]
  - Vuex [35]
  - Leaflet [19]

- **Recursos y Herramientas**
  - Git [13]
  - GitHub [14]
  - Chai [2]
  - Sinon [32]
  - Jest [17]
  - JWT [18]
  - Travis CI [3]
  - Codecov [5]
  - Heroku [15]

1.3. Planificación del Desarrollo

Para el desarrollo del proyecto se han seguido los principios de las *Metodologías Ágiles* del desarrollo software, es decir, en lugar de definir una planificación rígida y estricta desde un primer momento, se ha apostado por ajustar el desarrollo a las necesidades específicas que surgen con cada nueva funcionalidad que se va añadiendo.

Esto hace que se respete además el principio *YAGNI (You Are not Gonna Need It)*, considerado una buena práctica al evitar dar complejidad al desarrollo tratando de abarcar demasiado, centrándose en implementar lo mínimo necesario.

En un inicio se planteó un seguimiento del desarrollo haciendo uso de la herramienta Pivotal Tracker [27], lo que acabó siendo desechado debido principalmente a que las tareas que se establecieron inicialmente estaban mal planteadas. Al tratarse del desarrollo no solo de una aplicación, sino también del alumno como programador, el nivel de concreción de las tareas y la forma de plantearlas se reestructuraba continuamente, por lo que finalmente se fue llevando a cabo de forma manual.

Aún así, a grandes rasgos las tareas desarrolladas podrían resumirse en:

Tarea	Descripción
Prototipado	Diseño de la Interfaz de Usuario
Setup Infraestructura	Preparación del repositorio y herramientas a usar
Setup Backend	Preparación de la estructura del backend
Setup Frontend	Preparación de la estructura del frontend
Desarrollo	Cíclico entre backend, testing y frontend
Despliegue	Despliegue de la aplicación en la nube

Tabla 1.1: Tareas planteadas

Capítulo 2

Tecnologías y Herramientas

Este capítulo dará contexto al desarrollo del proyecto enumerando y explicando brevemente las tecnologías en las que se basa la aplicación y el por qué de su uso.

2.1. Herramientas de Desarrollo

2.1.1. Control de versiones: Git y GitHub

Para el control de versiones de la aplicación se ha usado Git [13], la herramienta más extendida para ello actualmente, junto a un repositorio remoto alojado en GitHub [14] perteneciente a la organización de la que dispone el tutor para organizar los trabajos que tutoriza, por lo que permanece como repositorio privado.

Se ha seguido la estrategia de *branching* típica para este tipo de proyectos: disponer de dos ramas principales, desarrollo y producción (en este caso se ha usado la rama master como rama de producción), para guardar los cambios en la rama desarrollo mientras se esté trabajando en el proyecto y fusionarlos a la rama de producción cuando se tenga lista una *release*. Al ser un único desarrollador en lugar de un equipo de trabajo, no se ha considerado relevante crear ramas *feature* para cada funcionalidad que se desarrolle puesto que se tiene control total sobre el código.

2.1.2. Cubrimiento de código: Codecov y ESLint

Codecov [5] es una herramienta de cubrimiento estático de código que proporciona estadísticas sobre la cantidad de código que es ejecutada en los tests de una aplicación, siendo lo recomendable mantener el índice de cubrimiento sobre un 80 %.

Resulta de utilidad para tener una visión general del estado del desarrollo, encontrar fragmentos de código que queden sin testear y comparar la relación entre

cantidad de código añadida y cantidad de código testeada con cada nuevo commit que se hace al repositorio.

Cabe también mencionar el uso de ESLint [9] como plugin de Quasar para el análisis del código en el front end, que va un paso más allá del cubrimiento y comprueba de una forma estricta incluso la identación del código para mantenerlo limpio y legible.

2.1.3. Integración continua: Travis CI

Travis CI [3] es un servicio de integración continua en la nube al que se ha asociado el repositorio del proyecto.

Cada vez que se suben cambios al repositorio se dispara la batería de tests indicada en el script de Travis e informa sobre sus resultados, lo que da una visión sobre el estado de salud del proyecto. Este tipo de herramientas puede configurarse incluso para no permitir que se suban los cambios al repositorio si no se pasan todos los tests, pero al ser un proyecto pequeño en el que un solo desarrollador tiene control total sobre la aplicación no ha sido necesario.

2.1.4. Despliegue en la nube: Heroku

Heroku [15] ha sido la elección para el despliegue en la nube de la aplicación. Se trata de una plataforma diseñada para ello con soporte para varios lenguajes y un plan gratuito que resulta de utilidad para pequeños proyectos.

2.2. Back end

2.2.1. MongoDB Atlas

MongoDB es una base de datos open source no relacional, de las más populares de este tipo. La gran ventaja de orientar su modelado a documentos en lugar de tablas interrelacionadas es la flexibilidad que provee, ofreciendo escalabilidad y sencillez para el procesado de los datos al no tener que preocuparse de mantener la estructura de las bases de datos relacionales.

Atlas [22] es su vertiente en la nube. Ofrece distintos planes de alojamiento, incluyendo uno gratuito que ha sido usado en este proyecto, y resulta muy sencilla de configurar y conectar a un servidor.

2.2.2. Servidor Express

El back end de la aplicación se ha desarrollado sobre Express [10], el framework considerado un estándar para el desarrollo de aplicaciones servidor en Node [24].

Se configura y desarrolla de una forma relativamente sencilla y bastante flexible, organizando sus *endpoints* para dar servicios en forma de rutas gestionadas por su propio *router*, en el que se pueden introducir *middlewares* para procesar las peticiones entrantes y simplificar la abstracción de responsabilidades de cada componente del servidor.

2.3. Front end

2.3.1. Quasar Framework

Dado que el front end del proyecto se iba a desarrollar inicialmente usando Vue.js [34], el tutor sugirió probar Quasar [11], un framework open source basado en el mismo que permite generar aplicaciones para varias plataformas (Web SPA, PWA o SSR, Móvil o de Escritorio) partiendo del mismo código. Para el proyecto se ha usado su vertiente SPA (Single Page Application), que es la opción por defecto.

Quasar es de gran utilidad para agilizar el prototipado de una aplicación, puesto que provee una estructura lista para usar desde que se genera el proyecto, y su propia librería de componentes para facilitar el diseño de la interfaz de usuario sin tener que implementar los estilos CSS desde cero o recurrir a librerías externas.

2.3.2. Enrutamiento de una SPA

Las SPA (Single Page Application - Aplicaciones de una Sola Página) son una evolución, teóricamente más eficiente (si una SPA pura crece demasiado en complejidad el rendimiento puede verse afectado), del funcionamiento tradicional de las aplicaciones web.

Antes el planteamiento era enviar al cliente una nueva página cuando este navegaba y hacía una petición para acceder a su ruta correspondiente. Con las SPA se envía la aplicación en la primera petición al cliente, y el enrutamiento entre páginas se gestiona completamente desde el front end, cambiando el contenido que compone la página de forma dinámica y usando el servidor solamente para llamadas a APIs que proporcionen los datos que necesita la página en formato JSON.

En Vue esto se consigue con Vue Router [31], el enrutador oficial del framework. Con él se pueden organizar las rutas que componen la interfaz de la aplicación y asociar cada una a un componente, anidarlas e incluso añadir funciones que se ejecutan antes de acceder a una ruta, por ejemplo para comprobar las credenciales del usuario.

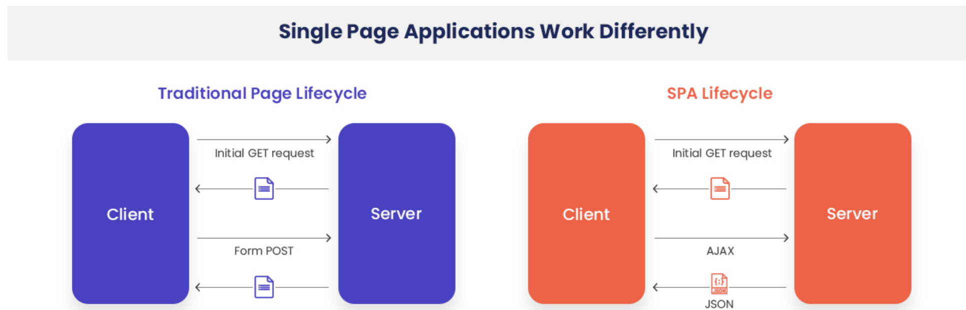


Figura 2.1: Flujo de una SPA en comparación a una arquitectura MVC extraído de Excellent Webworld [36]

2.3.3. Control de estado con Vuex

El control de estado en una aplicación es un concepto que se ha popularizado gracias al Patrón Flux, un patrón de diseño implementado por Facebook y popularizado por Redux [30] integrándose en el framework React [29].

El clásico Modelo Vista-Controlador (MVC) plantea el problema de disponer de múltiples fuentes de información que pueden entrar en conflicto. Flux trata de solucinar esto almacenando todos los datos relevantes para el estado de la aplicación en un único punto que consultan todos sus componentes.

La implementación oficial de este patrón en Vue, usada para este proyecto, es Vuex [35]. Su funcionamiento se basa en disponer de una *Store* como fuente única de información. La Store se divide a su vez en los siguientes componentes:

- *State* - Estado actual de los datos relevantes para varios componentes de la aplicación.
- *Mutations* - Las únicas funciones que pueden modificar los datos del State de forma directa. Deben ser síncronas, puesto que los cambios de estado no deben depender de cómo se secuencian eventos impredecibles.
- *Actions* - Funciones encargadas de realizar operaciones y hacer una llamada a las Mutations para guardar los cambios. Estas sí pueden ser asíncronas.
- *Getters* - Funciones que devuelven valores del State para hacerlos accesibles desde cualquier componente de la aplicación.

Vuex permite además descomponer la Store en distintos módulos según la funcionalidad con la que están relacionados sus estados. Cada módulo dispone de sus propios State, Mutations, Actions y Getters, funcionando de la misma manera que una única Store global, solo que mejorando la gestión del código al encapsular responsabilidades distintas.

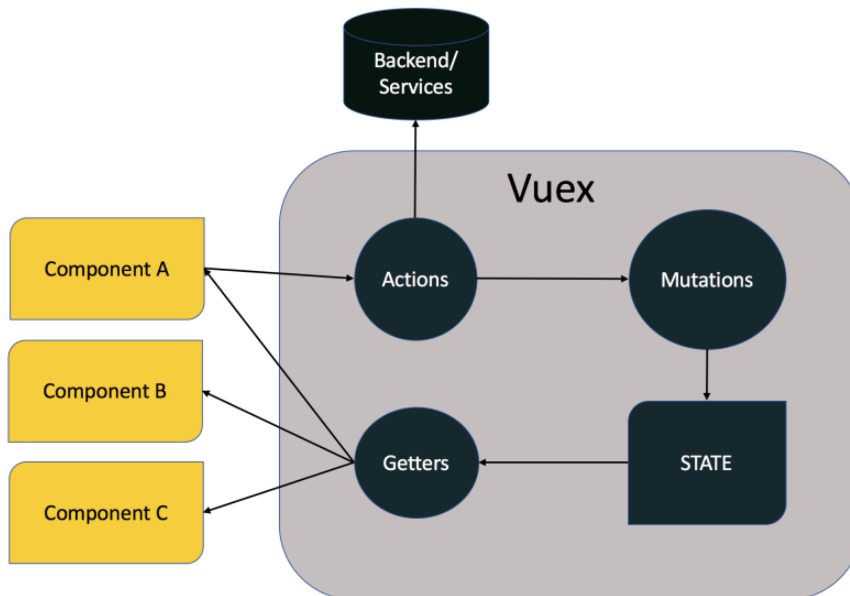


Figura 2.2: Flujo de Vuex extraído de un artículo de Lauri Hiltunen [16]

2.3.4. Leaflet

Dado que parte de la funcionalidad de la aplicación requiere el uso de mapas interactivos, como se verá en siguientes apartados, se ha decidido usar Leaflet [19] para ello.

Leaflet es una de las alternativas más populares como librería de mapas interactivos open source. Permite múltiples funcionalidades útiles como la adición de marcadores, pop-ups, botones de zoom, etc, todo diseñado de una forma simple y usable.

2.4. Gestión de la sesión: JWT

Para la gestión de la sesión de los usuarios en la aplicación se ha decidido usar JSON Web Tokens (JWT) [18]. JWT es un estándar que se basa en garantizar el intercambio de información de una forma segura a través de objetos JSON firmados digitalmente que contienen las *claims* para identificar a los usuarios, conocidos como tokens.

Los tokens codificados tienen la forma de una cadena de caracteres alfanuméricos compuesta de tres campos separados por puntos. Estos campos son:

- *Header* - La cabecera del token contiene información sobre su tipo (JWT) y el algoritmo de cifrado que se ha usado para firmarlo.
- *Payload* - La carga útil contiene las *claims* del usuario e información como el tiempo de expiración del token.

- *Signature* - La firma generada con el algoritmo especificado en la cabecera y una clave secreta que sirve para garantizar la validez del token.

Este token se añade como campo a la cabecera de las peticiones que realiza el usuario, permitiendo al servidor identificar de dónde viene. La gran ventaja de este enfoque es que es mucho más rápido y tiene menor coste el verificar la información del usuario parseando un objeto que con accesos a la base de datos como se ha hecho tradicionalmente. Aunque requiere tomar ciertas precauciones en cuanto a seguridad, como no introducir información sensible en el token (por ejemplo, contraseñas) y preferiblemente fijar tiempos de expiración cortos para cada token.

Dado que la expiración de los tokens suele fijarse en torno a unos pocos minutos para evitar ataques, la forma de mantener la sesión del usuario que se ha elegido en el proyecto es el *Patrón de Refresh Tokens*.

Este patrón consiste en devolver al usuario dos tokens cuando se autentica:

- *Access Token* - El token JWT que contiene las credenciales del usuario para garantizarle acceso a recursos. Este token tiene un tiempo de vida muy corto y se añade a la cabecera de cada petición que envía el usuario para identificarlo.

- *Refresh Token* - Este segundo token tiene un tiempo de vida mucho más extenso que el Access Token. Una vez recibido este token se almacena en el cliente y no se envía con cada petición.

Al caducar el Access Token inicial, el cliente se encargará de hacer una petición con el Refresh Token al endpoint expuesto para ello en el servidor para generar de nuevo un Acces Token válido con un tiempo de expiración nuevo. Una vez se dispone del nuevo Access Token, el cliente puede reenviar la petición que había hecho previamente al caducar el anterior Access Token, manteniendo así la sesión.

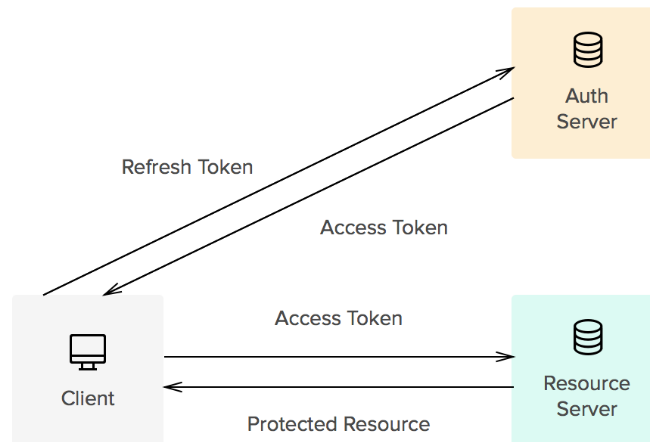


Figura 2.3: Ciclo del Patrón Refresh Token extraído de un artículo en Auth0 [1]

2.5. Test-Driven Development

Durante el desarrollo se ha tratado de respetar lo máximo posible el uso de *Desarrollo Guiado por Pruebas (TDD - Test-Driven Development)*. Este consiste en un ciclo compuesto por tres fases:

1. Escribir un test sobre una funcionalidad concreta. Ejecutarlo y comprobar que falla como se espera.
2. Escribir el *mínimo* código posible que permita pasar el test planteado.
3. Refactorizar el código generalizando su funcionalidad y haciéndolo más limpio y eficiente, de forma que el test planteado se siga pasando. Reiniciar el ciclo.

El TDD no se ha aplicado de forma estricta en el proyecto, de hecho queda patente en la desigualdad de cubrimiento de código entre el back end y el front end. Esto es debido a que la mayor parte de conocimientos del alumno sobre el funcionamiento de una aplicación full stack estaban concentrados en el back end. Al tener menos experiencia en el desarrollo de front end, se decidió implementar el código en esta parte de la aplicación antes de implementar sus tests, puesto que sin tener claro cómo se estructura la funcionalidad es imposible plantear un test donde se plantea qué es lo que se espera que pase.

El framework que se han usado para el testing en el back end ha sido Chai [2], mientras que en el front end se ha usado la opción recomendada de Quasar, Jest [17]. Para poder hacer mocks de distintas funciones en tests unitarios se ha usado la librería Sinon [32].

Capítulo 3

Desarrollo de la Aplicación

Este capítulo tratará de explicar de forma resumida cómo se han aplicado las tecnologías citadas en el capítulo anterior durante el desarrollo del proyecto.

3.1. Prototipado

Como primer paso previo al desarrollo se ha llevado a cabo una fase de diseño de la interfaz de usuario de la aplicación, centrando así el desarrollo en la experiencia de usuario como base de las funcionalidades.

Para ello se realizaron una serie de mockups a mano sobre las pantallas principales de la aplicación, centrados en un diseño *mobile first*.

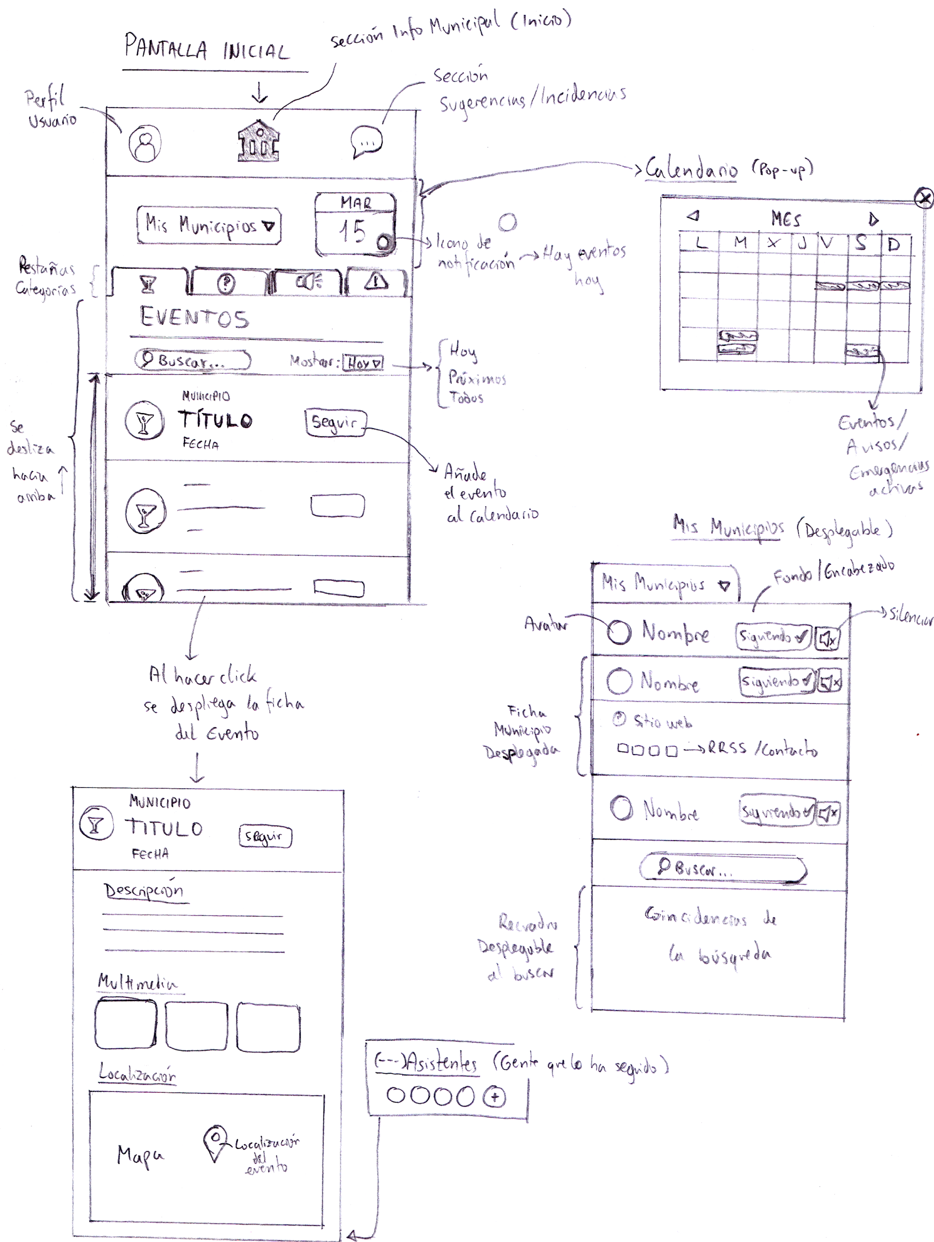
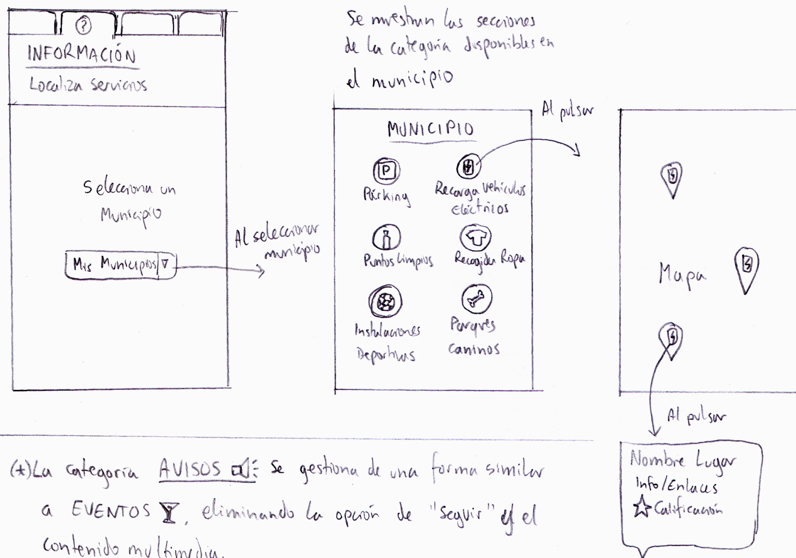


Figura 3.1: Prototipo incial de la pantalla principal



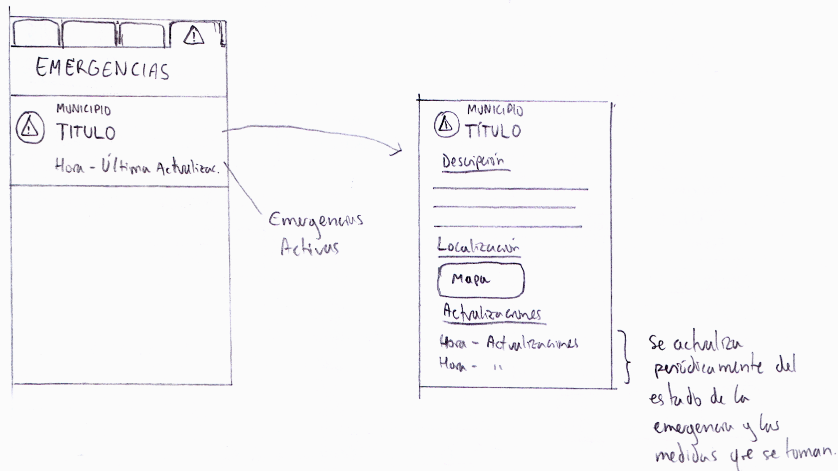


Figura 3.2: Prototipo inicial de las pantallas de categorías

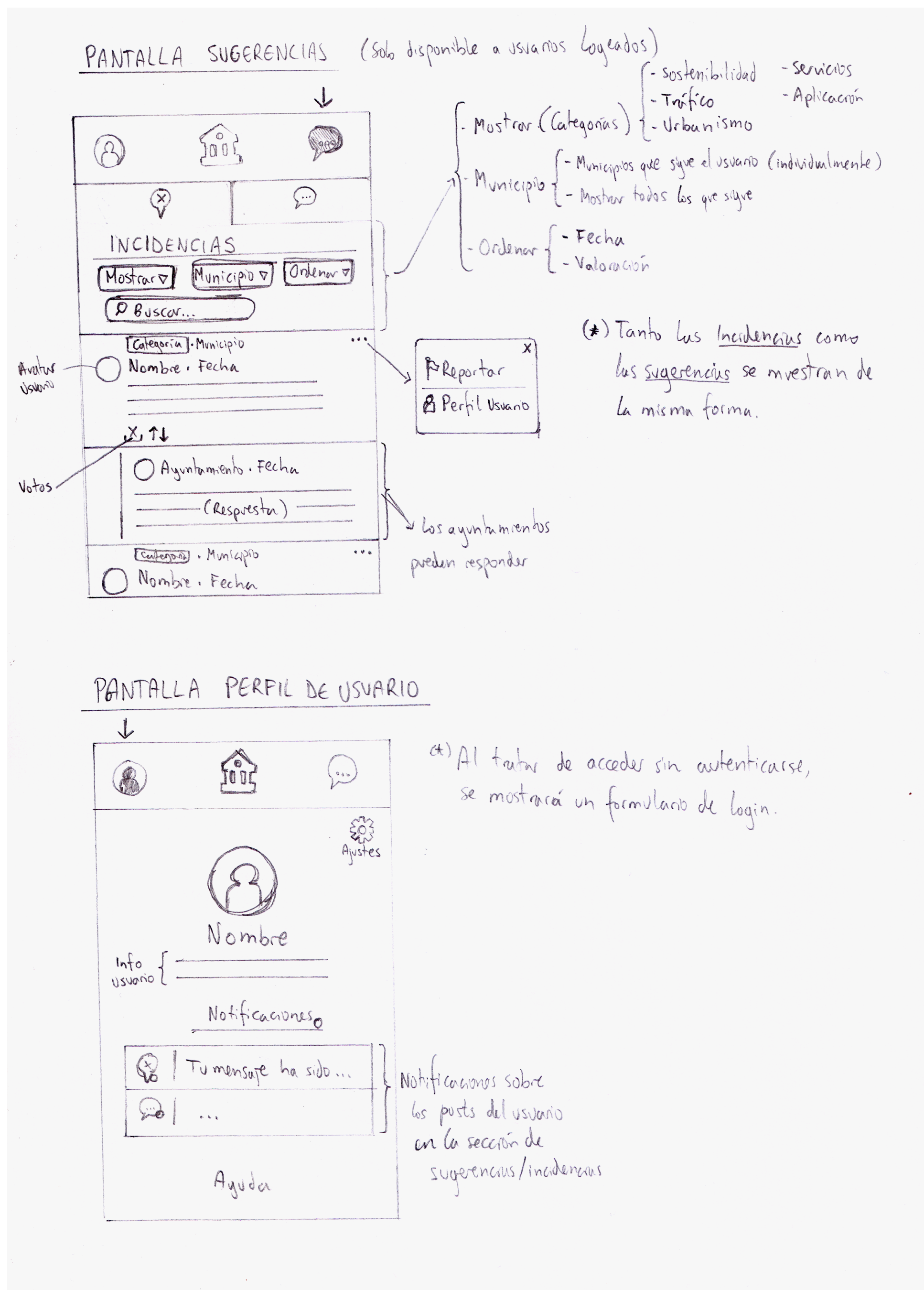


Figura 3.3: Prototipo inicial de la pantalla de Sugerencias y Perfil de usuario

3.2. Entorno de desarrollo

Durante el desarrollo del proyecto se ha trabajado usando Ubuntu como sistema operativo, por lo que las operaciones necesarias para la configuración de las herramientas se han llevado a cabo con comandos de Linux. Además se ha usado Visual Studio como IDE, que aún sin ser muy potente en cuanto a opciones de refactoring automático es cómodo para trabajar en JavaScript.

Partiendo de que se dispone previamente de Git instalado en el equipo, el

primer paso es iniciar el control de versiones en el directorio de trabajo y añadir como repositorio remoto el repositorio creado en GitHub por el tutor.

Partiendo también de disponer de Node en el equipo, todas las instalaciones requeridas se realizarán haciendo uso de NPM [26], el gestor de paquetes del framework. Se indicarán los paquetes necesarios en cada sección correspondiente de este capítulo.

Para separar las responsabilidades de cada parte de la aplicación full stack, se ha decidido dividir el proyecto en dos directorios situados en la raíz del repositorio: `server` y `client`, para alojar el back y front end de la aplicación respectivamente.

3.3. Configuración de Travis CI y Codecov

La configuración de la integración continua del proyecto con Travis es sencilla. Cabe destacar que al ser un repositorio privado, el dominio que debe usarse es `https://travis-ci.com/` en lugar de `https://travis-ci.org/`.

Primero se debe dar permisos a Travis como aplicación dentro de GitHub para que tenga acceso al repositorio que aloja la aplicación. Una vez hecho esto, basta con añadir un fichero `.travis.yml` en el directorio raíz del proyecto. Este fichero es simplemente un script donde se configuran los parámetros necesarios y se indica a Travis qué acciones ejecutar, tal como se muestra en el Listing 3.1.

```
1  languaje: node_js
2  node_js:
3  - stable
4  services:
5  - mongodb
6  install:
7  - cd server && npm install && cd ..
8  script:
9  - cd server/scripts && ./createConfigLocal.sh && cd ../..
10 - cd server && DB_ENV='travis' mocha --recursive --exit && cd ..
11 - ...
```

Listing 3.1: Contenido del fichero `.travis.yml`

La configuración de Codecov se hace de una forma similar. Tras dar permisos a Codecov en el repositorio, se usa el propio `.travis.yml` para añadir el cubrimiento a la integración continua.

Como se muestra en el Listing 3.2, se debe añadir el token que enlaza con Codecov como variable de entorno, en la sección de instalación se indica que se debe instalar el paquete `coverage` para ejecutar el análisis de la cobertura de código, se añade `coverage` como argumento al ejecutar los tests y finalmente se añade una

sección `after-success` que envía los resultados del análisis a Codecov.

```
...
env:
- CODECOV_TOKEN=<Token de Codecov>
install:
- ...
- cd server && npm install coverage && cd ..
script:
- ...
- cd server && DB_ENV='travis' coverage mocha --recursive --exit && cd ..
- ...
after-success:
- bash <(curl -s https://codecov.io/bash)
```

Listing 3.2: Configuración de Codecov en el fichero `.travis.yml`

De esta forma, cada vez que se haga un *push* al repositorio remoto enlazado a ambas herramientas, se ejecutará el script en Travis y se enviarán a su vez los resultados a Codecov, todo automáticamente.

3.4. Back end

3.4.1. Introducción: Arquitectura de 3 Capas

Inicialmente se comenzó a trabajar en el proyecto sin pensar ni tener claro un modelo de arquitectura software a seguir en el servidor. A medida que se avanzaba, simplemente con el registro, autenticación y gestión de la sesión de los usuarios, ya el código comenzaba a hacerse confuso y se empezó a ver la necesidad de separar las responsabilidades de los distintos componentes del back end.

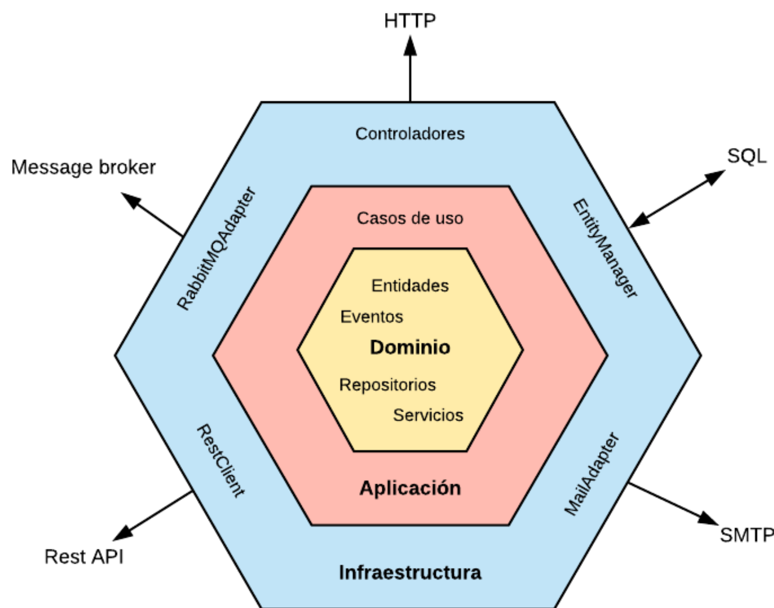


Figura 3.4: Ejemplo de una Arquitectura Hexagonal completa extraído de un artículo de Francisco Ugalde [33]

Gracias a la experiencia desarrollando en un proyecto con Arquitectura Hexagonal durante la asignatura de Prácticas Externas, al retomar este proyecto se tenía más claro el flujo de una aplicación y se decidió refactorizar todo el back end implementando la arquitectura más simple, la Arquitectura de 3 Capas. Las tres capas básicas del servidor son:

- *Controller* - La capa Controller es la más externa, la que expone los endpoints de las APIs y recibe las peticiones del front end. Es importante extraer toda lógica de negocio de esta capa, puesto que su responsabilidad se basa únicamente en procesar peticiones y respuestas. Es la que se ocupa de gestionar los estados HTTP.

- *Service* - La capa Service es la que concentra las funcionalidades y la lógica de la aplicación. Desde ella se accede a la capa Repository y sus servicios son invocados desde la capa Controller.

- *Repository* - La capa Repository es la representación de los modelos de datos. Concentra toda la lógica de acceso a la base de datos, permitiendo al resto de la aplicación abstraerse de esta.

Además de permitir gestionar el código de una forma más simple y clara, el dar modularidad al back end en forma de capas permite también portabilidad entre distintas tecnologías, por ejemplo, al poder reutilizar el resto de capas si se cambia de base de datos.

3.4.2. Estructura

La estructura de directorios del back end es la que se muestra en el Listing 3.3.

```
/server
  - /config
      - databaseConfig.js
      - local.js
  - /scripts
      - createConfigLocal.sh
  - /src
      - /api
          - /middlewares
          - /routes
      - /models
      - /services
      - /utils
          - /constants
          - /errors
          - /httpResponses
  - /test
      - /models
      - /resources
      - /unit
          - /api
          - /services
  - server.js
```

Listing 3.3: Estructura de directorios del back end

- `/config` - Ficheros de configuración del back end.
  - `databaseConfig.js` almacena las URIs de conexión a las distintas instancias de bases de datos creadas en Atlas (una para testing, otra para desarrollo, otra para producción y otra especificando la conexión por defecto para los tests en Travis).
  - `local.js` almacena las variables de configuración del back end que deben quedarse en local por ser secretas, como la clave para firmar los tokens JWT.
- `/scripts` - Scripts para utilidades.
  - `createConfigLocal.sh` es un script en bash que simplemente genera el fichero `/config/local.js` con la estructura necesaria. Este script fue

creado debido a que al incluir `local.js` entre los ficheros a ignorar para el repositorio remoto, generaba conflictos de dependencias para la ejecución de los tests en Travis por haber ficheros que dependían de él.

- `/src` - Directorio donde almacenar el código fuente.
  - `/api` - Directorio que almacena el código referente a la Capa Controller del servidor.
    - `/middlewares` - Los middlewares son funciones intermedias que se añaden antes de entrar a un endpoint. Son útiles para hacer un preprocesado de las peticiones y validar los datos entrantes.
    - `/routes` - Rutas que se añaden al router de Express para exponer los endpoints de las distintas APIs.
  - `/models` - Directorio que almacena el código referente a la Capa Repository del servidor. En este caso, al ceder la gestión de la capa a Mongoose [23], simplemente se incluyen las definiciones de los modelos de datos.
  - `/services` - Directorio que almacena el código referente a la Capa Service del servidor. Cada fichero que almacena representa un servicio y contiene funciones relacionadas con el mismo.
  - `/utils` - Directorio de utilidades.
    - `/constants` - Variables constantes como los códigos de estado HTTP o los roles de los usuarios.
    - `/errors` - Tipos de Error (equivalente a las excepciones en JavaScript) propios para dar semántica a la aplicación.
    - `/httpResponses` - Se han creado dos funciones auxiliares para gestionar dar formato las respuestas HTTP desde los controllers. En caso de error, se llama a la función con el tipo de error (por esto se definen previamente tipos de Error propios) y este se mapea a su código de estado HTTP correspondiente.
- `/test` - Directorio de tests.
  - `/models` - Tests de integración que comprueban los modelos de la capa Repository.
  - `/resources` - Recursos auxiliares para los tests, como declaraciones reutilizables.
  - `/unit` - Directorio contenedor de tests unitarios de las capas Controller y Service.
- `/server.js` - Fichero *main* de la aplicación back end, donde se configura Express, se cargan las rutas, se conecta a la base de datos y se pone a la escucha el servidor en un puerto de la máquina.

3.4.3. Testing

El back end cuenta con una amplia batería de tests que cubren casi la totalidad de la implementación. Se han testeado los componentes de forma unitaria, haciendo mocks de sus dependencias excepto en casos como el modelo de datos de los usuarios, en los que se permite testear haciendo uso de una base de datos real.

El haber desarrollado tests acabó siendo realmente útil en la labor de refactor del código, por ejemplo al separar responsabilidades entre capas, para saber que la funcionalidad se seguía cumpliendo.

```
> server@1.0.0 test /home/uno/Escritorio/TFG/CityOnContact/server
> DB_ENV="testing" mocha --recursive --exit

Server started on port 3000


  User Model:
    ✓ create user successfully (256ms)
    ✓ create user without required field should fail
    ✓ create user with invalid field should fail
    ✓ create user with already saved email should fail (289ms)
    ✓ compare user password correctly (305ms)

  JWT Authorization Middleware:
    - Admin role needed route
      ✓ return 200 for valid token (189ms)
      ✓ return 401 for no given token
      ✓ return 401 for malformed token
      ✓ return 401 for invalid role token
      ✓ return 401 for expired token
    - logged user needed route
      ✓ return 200 for valid admin token
      ✓ return 200 for valid user token
      ✓ return 401 for no given token
      ✓ return 401 for malformed token
      ✓ return 401 for expired token

  Refresh Token Validator Middleware:
    ✓ return 401 for no present refresh token
    ✓ return 401 for malformed refresh token (121ms)
    ✓ return 401 for expired refresh token

  User Controller Validator Middleware:
    ✓ return 400 for invalid input to /register endpoint
    ✓ return 400 for invalid input to /login endpoint

  User Controller: /register
    ✓ return 200 for valid refresh token
```

Figura 3.5: Ejecución de los tests del back end

3.4.4. Dependencias Usadas

Los paquetes de NPM que se han usado para la implementación del back end, además de Express, han sido:

- *bcrypt* - Cifrado de contraseñas.
- *body-parser* - `body-parser.json()` Permite al servidor parsear las peticiones entrantes y leerlas como objetos JSON.
- *chai* - Librería de asserts para el testing.

- *chai-http* - Permite simular peticiones HTTP para el testing.
- *cors* - Al incluirlo en Express, por defecto se permiten las peticiones CORS entrantes para cualquier origen.
- *express-jwt* - Módulo para automatizar el parseo de tokens JWT en Express.
- *express-validator* - Módulo de Express para facilitar la validación de campos de una petición entrante.
- *jsonwebtoken* - Gestión de JSON Web Tokens.
- *mongoose* - Módulo gestor de bases de datos MongoDB que provee todas las funcionalidades necesarias.
- *morgan* - Módulo auxiliar para llevar un log de las peticiones entrantes al servidor (habilitado solo en modo desarrollo).
- *nodemon* - *Watcher* que vigila los cambios en los ficheros y reinicia automáticamente el servidor en ejecución.
- *sinon* - Librería de mocks para testing.

3.5. Front end

3.5.1. Configuración inicial de Quasar

Una vez instalado Quasar CLI siguiendo su documentación [28], basta con ejecutar el comando [quasar create <nombre del proyecto>] para generar el directorio de la aplicación front end con toda su estructura creada lista para usar. La configuración inicial seleccionada en el asistente de creación de Quasar para este proyecto ha sido la mostrada en la Figura 3.6.

Figura 3.6: Configuración inicial de Quasar

Con el proyecto generado, se puede pasar al desarrollo con el comando [quasar dev], que inicia la aplicación en modo desarrollo ejecutándola sobre un servidor local en el puerto 8080.

Al ejecutar el comando [quasar build], se compila el código y se genera la aplicación para la plataforma indicada según el modo que se especifique con el flag `-m` (por defecto, y el usado en este proyecto, es el modo spa). El directorio donde se genera por defecto es `/client/dist/spa`, que es la ruta estática a la que accede el back end del proyecto para servir la aplicación.

3.5.2. Estructura

La estructura de directorios principal del front end, gran parte generada por Quasar junto a alguna adición realizada para este proyecto, es la que se muestra en el Listing 3.4.

```
1  /client
2    - /src
3        - /assets
4        - /boot
5        - /components
6        - /constants
7        - /css
8        - /layouts
9        - /pages
10       - /router
11       - /services
12       - /statics
13       - /store
14       - App.vue
15   - /test
16     ...
17   - Ficheros de configuración, destaca quasar.conf.js como núcleo
18     ...
```

Listing 3.4: Estructura de directorios del front end

- `/src` - Código fuente del front end.
    - `/assets` - Directorio que contiene ficheros extra como imágenes .svg.
    - `/boot` - Directorio que contiene los plugins que se quieran definir para Quasar. Es aquí donde se instancian servicios necesarios para la aplicación para que sean inicializados desde el arranque de la aplicación. Un ejemplo de ello es Axios, contenido en su fichero `axios.js`.

- `/components` - Directorio que contiene la implementación de componentes de la interfaz de usuario.
- `/constants` - Este directorio es uno de los que no ha generado Quasar por defecto. Contiene constantes relevantes para la aplicación como las URLs de los endpoints del servidor.
- `/css` - Directorio que almacena estilos CSS globales predefinidos por Quasar que abarcan cuestiones como los colores del tema principal de la aplicación.
- `/layouts` - Directorio que almacena los Layouts, componentes invariables de la interfaz de usuario como la barra de herramientas de la cabecera.
- `/pages` - Directorio que almacena los ficheros con el contenido de las distintas páginas de la aplicación.
- `/router` - Directorio donde se almacenan los ficheros para gestionar el Router de Vue.
- `/services` - Este directorio es otro de los que no genera Quasar de forma automática. Se ha creado para extraer responsabilidades de servicios de la aplicación, de forma similar a como se hizo en el back end.
- `/statics` - Directorio para almacenar ficheros estáticos como imágenes.
- `/store` - Directorio que contiene las definiciones e implementación de los componentes de Vuex de la aplicación.
- `App.vue` - Componente .vue raíz de toda la aplicación.

- `/test` - Directorio de tests de la aplicación. Dentro se crea un subdirectorio para cada framework añadido con Quasar para el testing, en este caso existe solo un subdirectorio `jest`.

3.5.3. Enrutamiento

Tal como se ha mencionado en apartados previos, para gestionar la aplicación como una SPA se hace uso de Vue Router. Las rutas que se han definido para la aplicación cliente se organizan como puede verse en el Listing 3.5.

```
1 const routes = [
2   {
3     path: '/',
4     component: () => import('layouts/AnonUserLayout.vue'),
5     children: [
6       { path: '', component: () => import('pages/Index.vue') },
7       { path: 'register', component: () => import('pages/Register.vue') }
  ,
```

```
8        { path: 'login', component: () => import('pages/Login.vue') }
9      ],
10     beforeEnter: function userMustNotBeAuthenticated (to, from, next) {
11       if (!store.getters['user/isAuthenticated']) {
12         return next()
13       }
14       return next('/home')
15     }
16   },
17   {
18     path: '/home',
19     component: () => import('layouts/AuthenticatedLayout.vue'),
20     children: [
21       { path: '', component: () => import('pages/Index.vue') },
22       { path: '/profile', component: () => import('pages/Profile.vue') },
23       { path: 'comunica', component: () => import('pages/Comunica.vue') }
24     ],
25     beforeEnter: function userMustBeAuthenticated (to, from, next) {
26       if (store.getters['user/isAuthenticated']) {
27         return next()
28       }
29       return next('/login')
30     }
31   }
32 ]
```

Listing 3.5: Enrutamiento de la aplicación cliente - `/src/router/routes.js`

Como se puede ver se han añadido dos funciones de comprobación antes de entrar a las dos rutas principales o cualquiera de sus hijas.

La función `userMustNotBeAuthenticated()` de la línea 10, comprueba que cuando se intenta acceder a las rutas destinadas a usuarios anónimos, como el formulario de registro o login, el usuario no esté autenticado. De lo contrario se redirecciona a la página principal.

La función `userMustBeAuthenticated()` de la línea 25, por el contrario, comprueba que cuando se intenta acceder a las rutas protegidas, como el perfil del usuario, el usuario esté autenticado. de lo contrario se redirecciona al formulario de login.

3.6. Implementación de la sesión con JWT

3.6.1. Back end

Generación y envío de los tokens

Existen tres puntos implicados en la generación y envío de los tokens:

- `/services/JWTService.js` - JWTService es un servicio que contiene funciones para la creación de JSON Web Tokens basándose en su paquete de Node homónimo, `jsonwebtoken`. Para generar un token, se introducen en su payload el Id del usuario asignado en la base de datos y sus roles como credenciales, se le asigna un tiempo de expiración (5 minutos para el Access Token, 1 hora para el Refresh Token) y se firma con la clave secreta almacenada en el directorio de configuraciguraciones. Además, se le añade el prefijo al string del token resultante como parte del estándar.

```
1 const generateToken = (user, expirationTime) => {
2     const payload = {
3         sub: user._id,
4         role: user.role
5     };
6     return 'Bearer ' + jwt.sign(payload, jwtSecret, { expiresIn:
    expirationTime });
7 }
```

Listing 3.6: Creación de un token en el fichero `JWTService.js`

- `/services/userService.js` - userService es el servicio destinado a gestionar las acciones del usuario. Tanto la función de login como la de registro, si sus validaciones previas son correctas, llaman a JWT service para generar los tokens Access y Refresh y los devuelven como resultado.

- `/api/routes/userController.js` - userController es el controlador que contiene los endpoints relacionados con los usuarios. En los endpoints de registro y login, llaman a sus funciones respectivas de userService y devuelven los tokens como parte del cuerpo de la repuesta al front end.

Validación de peticiones que requieren autorización

Para la validación de las credenciales al intentar acceder a una ruta protegida se ha creado el middleware `/api/middlewares/JWTAuthorization.js`.

Este consiste en dos funciones anidadas, la primera comprueba la validez del token haciendo uso del módulo NPM `express-jwt` y la segunda comprueba que

los roles extraídos del token codificado se encuentren entre los requeridos, indicados por el endpoint pasándolos como parámetro al middleware.

```
const authorize = (requiredRoles) => {
    return [
        expressJWT({ secret: jwtSecret }),
        (err, req, res, next) => {
            if (err) {
                return httpErrorResponse(res, new InvalidAccessError('Error parsing token: ' + err.message));
            }
            next();
        },
        (req, res, next) => {
            if (typeof requiredRoles === 'string'){
                requiredRoles = [requiredRoles];
            }
            if (requiredRoles && !requiredRoles.includes(req.user.role)) {
                return httpErrorResponse(res, new InvalidAccessError('Invalid credentials'));
            }
            next();
        }
    ];
}
```

Listing 3.7: Middleware para la validación de la sesión en `/api/middlewares/JWTAuthorization.js`

Para usar el middleware, basta con añadirlo como argumento previo a la callback del endpoint que se quiera proteger. Si no se le pasa ningún argumento indicando los roles requeridos, el middleware comprobará que la petición esté autenticada independientemente del rol.

Refresh endpoint

Se ha añadido el controller `/api/routes/tokenController.js` para exponer el endpoint de refresh al que el front end enviará una petición cuando caduque su Access Token. Se comporta de forma similar a lo visto en las dos secciones anteriores: Dispone de un middleware para validar el Refresh Token que debe encontrar en el cuerpo de la petición, y llama a jwtService para regenerar ambos tokens, prolongando la sesión del usuario.

3.6.2. Front end

Gestión de los tokens

En el front end la sesión es una parte fundamental que afecta al comportamiento de toda la aplicación, por lo que para gestionarla se ha creado un módulo en la Store de Vuex: `/src/store/userModule`.

Para separar responsabilidades, se ha extraído un servicio `/src/services/authService.js` que contiene las funciones que se encargan de gestionar las peticiones con Axios al back end relacionadas con la sesión, como el registro y el login.

Cuando se reciben los tokens como respuesta del backend, se deben realizar tres acciones:

1. Almacenar ambos tokens en el almacenamiento local del navegador. Para ello se ha creado un servicio `localStorageService.js`.
2. Añadir a la instancia de Axios usada por la aplicación la cabecera Authorization con el valor del Access Token para cada petición que se envíe. Este punto es clave, pues es el formato estándar para la autenticación con JWT y las comprobaciones desde el back end siempre comprobarán que se cumpla.
3. Hacer un *Commit* de los tokens al State de Vuex en la aplicación.

Gestión de la caducidad

Para detectar cuándo caduca un token desde el front end, se hace uso de los *Interceptors* de Axios. Estos no son más que middlewares que pueden añadirse a las peticiones o respuestas que gestiona Axios.

En este caso, el interceptor se añade a las respuestas del servidor. En caso de que la respuesta contenga un código HTTP 401 (Unauthorized), se llama a una función auxiliar que, siempre que el origen de la respuesta sea distinto al endpoint de refresh (ya que esto significaría que se ha fallado la renovación de los tokens) comprueba que el mensaje de error indique que JWT ha expirado.

Si es así, se recupera el Refresh Token del almacenamiento local, se envía en el cuerpo de la petición al endpoint de refresh del servidor y en caso de que se reciban los nuevos tokens, se vuelve a reenviar la petición original que había fallado al caducar la sesión.

```
1 axiosInstance.interceptors.response
2   .use(function HTTP2xxResponse (response) {
3     return response
4   }, function HTTPNot2xxResponse (error) {
5     if (error.response.status === 401) {
```

```
      return handleJWTExpiration(error)
    }
    return Promise.reject(error)
  })
```

Listing 3.8: Interceptor de las respuestas en `/src/boot/axios.js`

3.7. Inclusión de Mapas Interactivos con Leaflet

En el momento de la redacción de esta memoria aún no se ha añadido la implementación de mapas interactivos, aunque ya se encuentra prototipada en el contenido de las categorías de Avisos, como podrá verse en su sección correspondiente del Capítulo Cuatro [4]. Se espera que esté ya implementado en el momento de la defensa del proyecto.

Trabajar con Leaflet en Vue es sencillo, además de instalar su propio paquete, existe el paquete `vue2-leaflet` [20] para integrarlo en el framework.

Siguiendo la documentación de Vue Leaflet [21] se describen paso a paso las opciones para instalarlo en la aplicación, siendo la más adecuada la instalación de los componentes de Leaflet como componentes locales de la aplicación, usables desde cualquier punto, como se ilustra en el Listing 3.9. Usando Quasar, probablemente lo más adecuado sea implementar esta opción como un plugin del framework, añadiendo el fichero con la implementación en el directorio `/src/boot`.

```
import Vue from 'vue';
import { LMap, LTileLayer, LMarker } from 'vue2-leaflet';
import 'leaflet/dist/leaflet.css';

Vue.component('l-map', LMap);
Vue.component('l-tile-layer', LTileLayer);
Vue.component('l-marker', LMarker);
```

Listing 3.9: Uso de componentes de Leaflet globales a la aplicación Vue

El resto es simplemente consultar la documentación y guías oficiales de Leaflet para adaptar el mapa a las páginas en las que se use.

Aunque lo interesante del proyecto no es simplemente añadir un mapa interactivo para los usuarios. La idea es proporcionar a los ayuntamientos la posibilidad de fijar puntos en el mapa desde su cuenta para avisar de los eventos que quieran publicar.

Para ello es necesario lo que se conoce como *Geocoding*, es decir, la capacidad de traducir direcciones entendibles en lenguaje humano a coordenadas geográficas para situar los puntos en el mapa.

Existe un paquete publicado recientemente en NPM que constituye una opción interesante para esta funcionalidad: Leaflet GeoSearch [12].

Este paquete funciona como un plugin de Leaflet que integra en los mapas la posibilidad de añadir un buscador de direcciones, sirviendo como intermediario entre la aplicación y las APIs de proveedores de motores de búsqueda geográficos.

El proveedor escogido para el proyecto será Nominatim, el buscador open source de Open Street Maps. Aunque tal como se advierte en la documentación de Leaflet GeoSearch, Nominatim no es gratuito de forma ilimitada solo por ser open source, tiene sus limitaciones citadas en su política de uso [25].

Capítulo 4

Descripción de la Aplicación

Tras hablar de las tecnologías y desarrollo del proyecto, este capítulo está destinado a proporcionar una visión de la interfaz y uso de la aplicación que encontrará el usuario.

4.1. Introducción

Toda la interfaz de usuario ha sido íntegramente desarrollada haciendo uso de los componentes que ofrece Quasar. Aunque cercano al diseño real que tendría la aplicación no deja de ser un prototipo, por lo que las pantallas mostradas en los siguientes apartados representan la idea que se pretende conseguir, a falta de algunos retoques a nivel visual.

El diseño de las páginas se ha hecho aprovechando las propiedades responsivas del *grid* que usa la librería, por lo que no ha sido necesario un rediseño para pasar de la versión móvil a la versión de escritorio.

El único cambio que se decidió hacer entre ambas versiones fue la barra de herramientas de la cabecera, diseñada como un *layout* de Vue. Esto quiere decir que se comporta como un componente estático que se renderiza siempre mientras cambia el contenido.

4.2. Página de Inicio: Sección de Avisos

La página de principal de la aplicación es la Sección de Avisos. El acceso a esta página es público, no hace falta que el usuario esté autenticado para poder verla.

En ella se ofrece la opción de buscar las ciudades registradas para consultar la información que se ofrece en las distintas categorías, entre las que se puede buscar y aplicar filtros de búsqueda como el ordenar por fecha o las ciudades que se están mostrando. Cada categoría tiene su propio color temático característico y se puede navegar entre ellas haciendo click en su pestaña correspondiente.

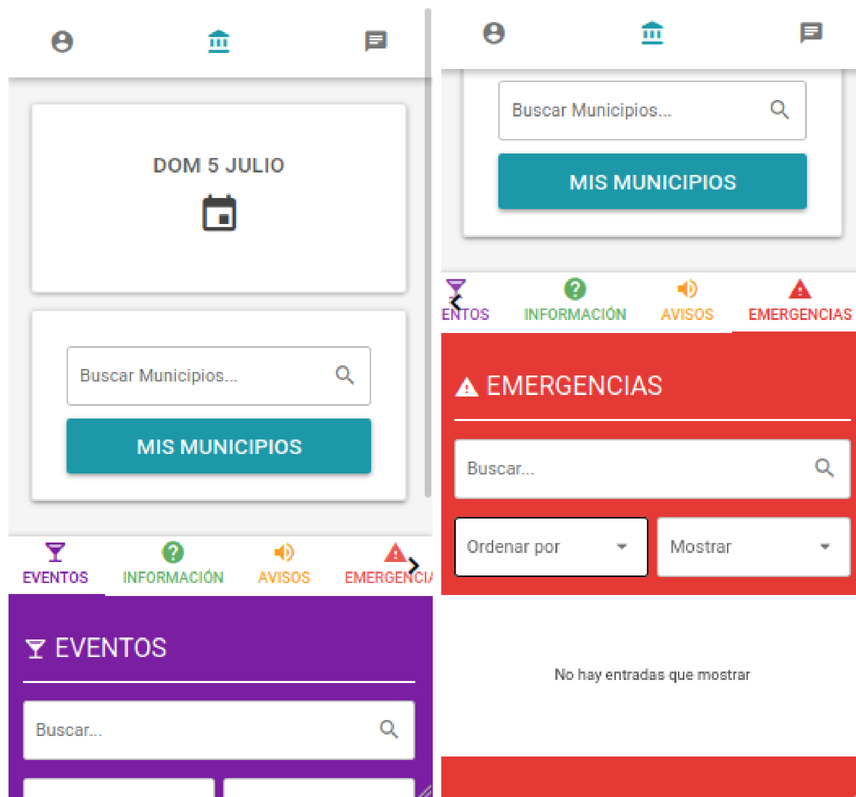


Figura 4.1: Pantalla de inicio en la versión móvil

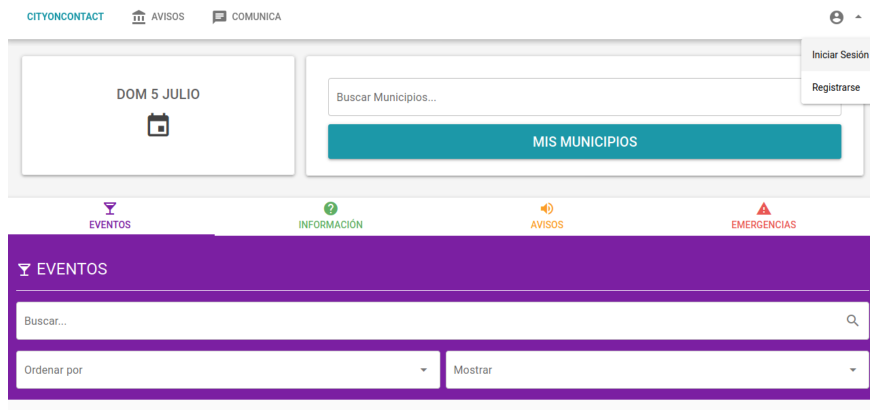


Figura 4.2: Pantalla de inicio en la versión de escritorio

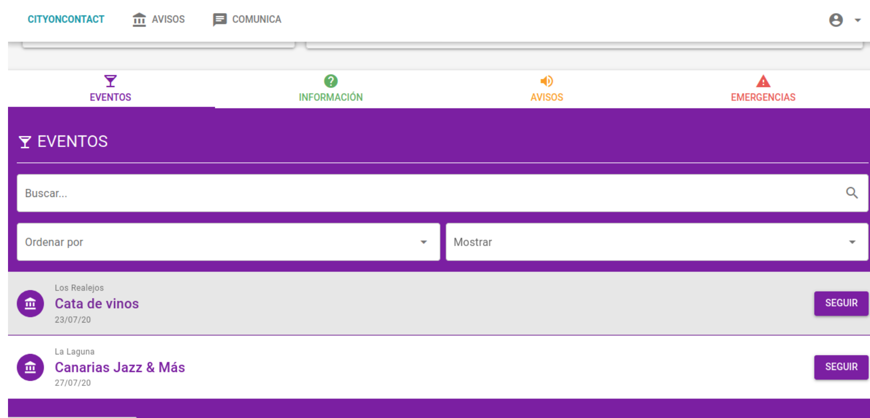


Figura 4.3: Categoría de Eventos en la versión de escritorio

La pantalla de inicio cuenta además con un calendario en el que se señalan los días que contienen eventos en las ciudades que está siguiendo el usuario.

Bajo el buscador de ciudades, se puede gestionar también las ciudades que se están siguiendo, mostrando información sobre ellas como la dirección a sus portales web o redes sociales, y dando la opción de dejar de seguirlas o silenciar sus avisos.

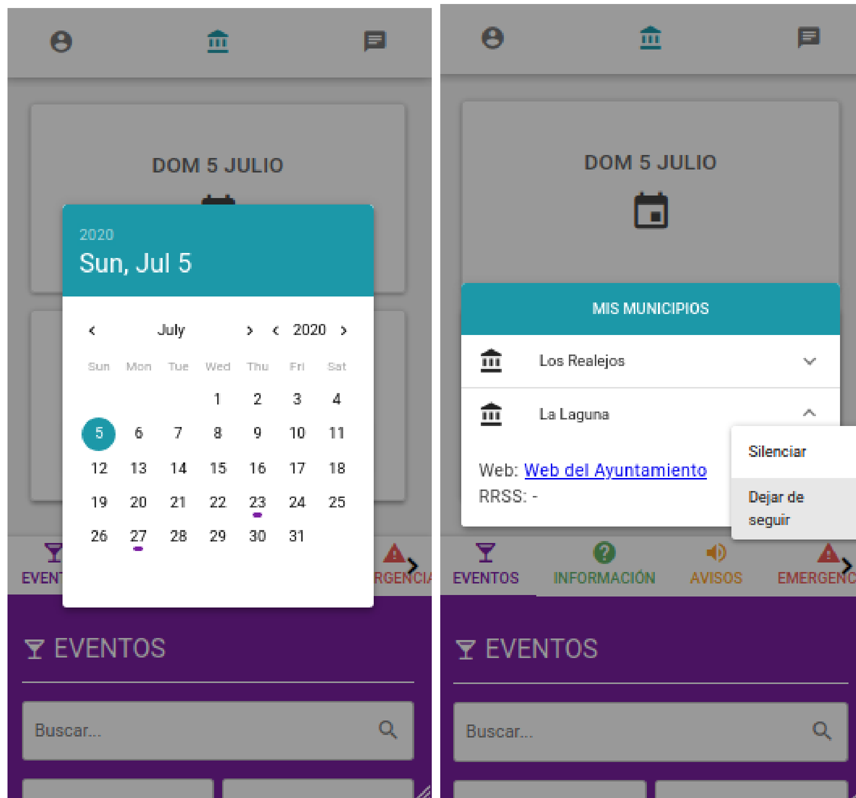


Figura 4.4: Calendario y Municipios en la versión móvil

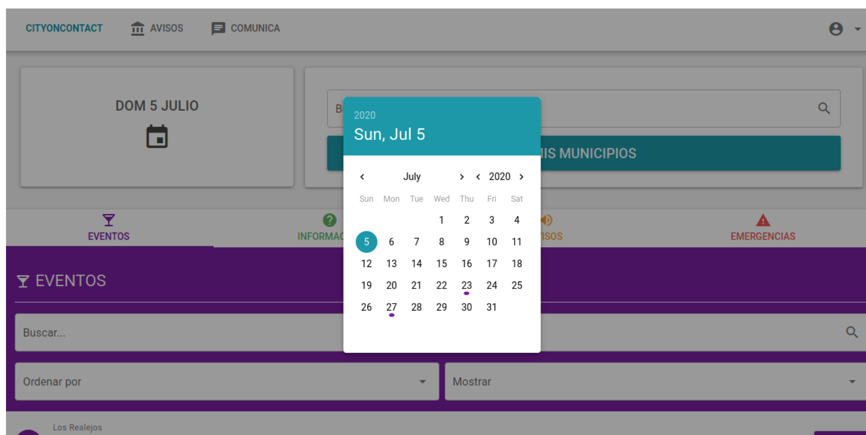


Figura 4.5: Calendario en la versión de escritorio

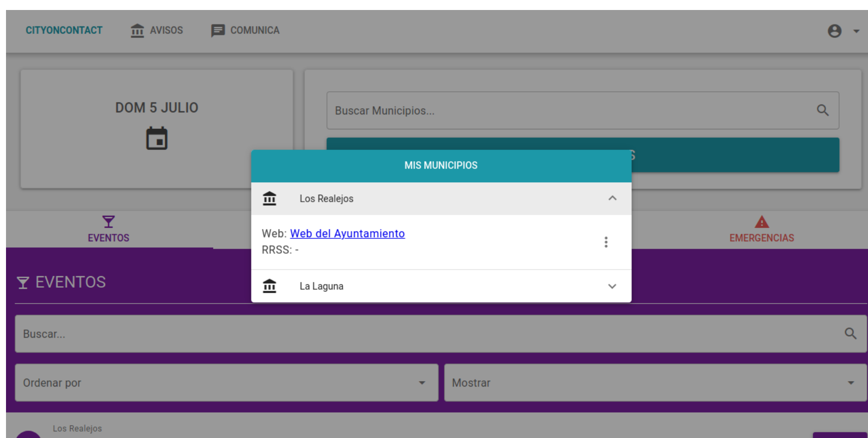


Figura 4.6: Municipios en la versión de escritorio

4.3. Contenido de una Categoría

El listado de contenidos que aparece en cada pestaña de la Sección de Avisos es interactivo. Al hacer click en el botón "Seguir", se añadirá el evento al calendario del usuario.

Al hacer click en el propio contenido, se abrirá su propia página que incluye una descripción del mismo, su localización (es aquí donde se añadirán los mapas interactivos de Leaflet) y opcionalmente contenido multimedia.

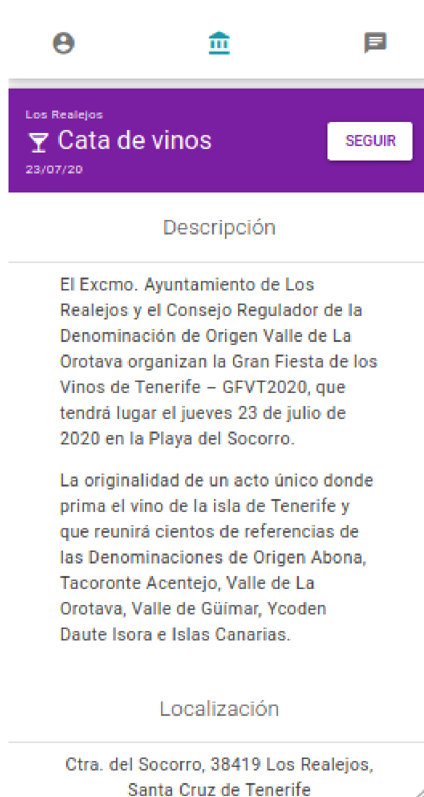


Figura 4.7: Evento en la versión móvil

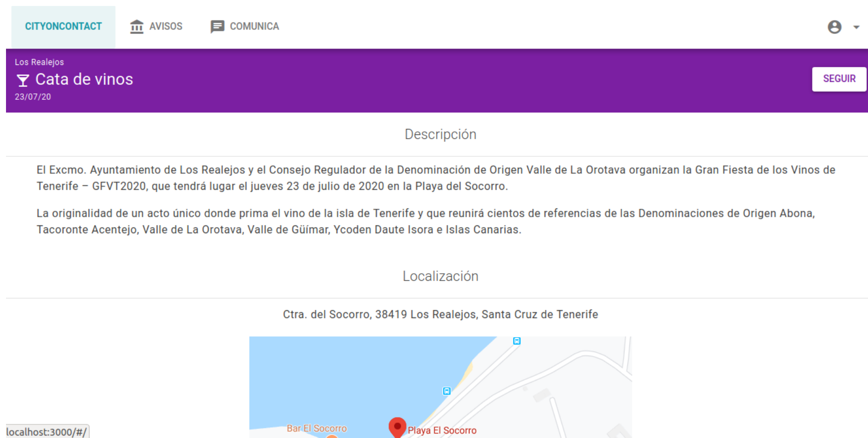


Figura 4.8: Evento en la versión de escritorio

4.4. Autenticación

Para la autenticación de los usuarios se dispone de los clásicos formularios de Registro y Login. Se han incluído reglas de validación para cada campo, incluida una evaluación del formato del email basada en una expresión regular, una comprobación de que el email no está en uso al registrarse (se ha expuesto un endpoint en el Controller de usuarios del back end para ello) y un mínimo de 6 caracteres para la contraseña.

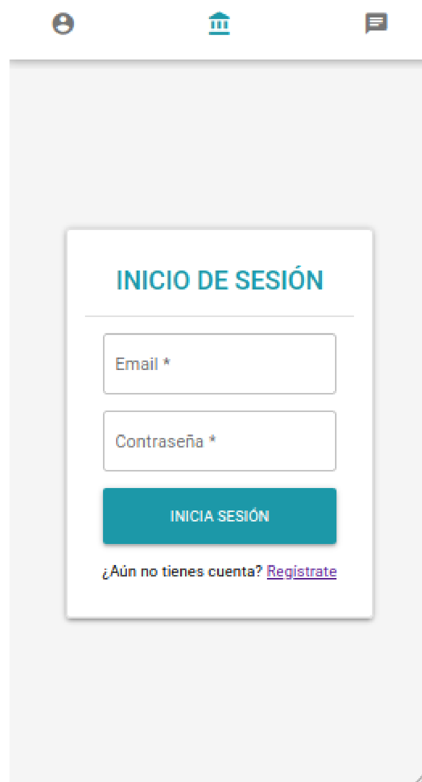


Figura 4.9: Formulario de Login en la versión móvil

Figura 4.10: Formulario de Registro en la versión de escritorio

4.5. Sección Comunica

La Sección Comunica, donde son los usuarios quienes proporcionan información a los ayuntamientos, se organiza visualmente de una forma parecida a la Sección de Avisos, solo que esta se divide en dos categorías: Incidencias, donde se informa de desperfectos, y Sugerencias, donde se ofrecen observaciones a mejorar de la localidad.

A diferencia del contenido de la Sección de Avisos, en Comunica el contenido se organiza de una forma parecida a un foro. Cada item del listado es un comentario que se puede votar positiva y negativamente, con un contador de votos. De esta forma los propios usuarios pueden moderar el contenido que se publica, además de poder reportar un uso incorrecto de la aplicación.

Para mayor facilidad organizativa de los comentarios, se etiqueta cada uno con una categoría de entre las existentes: Sostenibilidad, Tráfico, Urbanismo y Servicios. Cada una tiene un color asignado y se muestra como una etiqueta sobre el comentario.

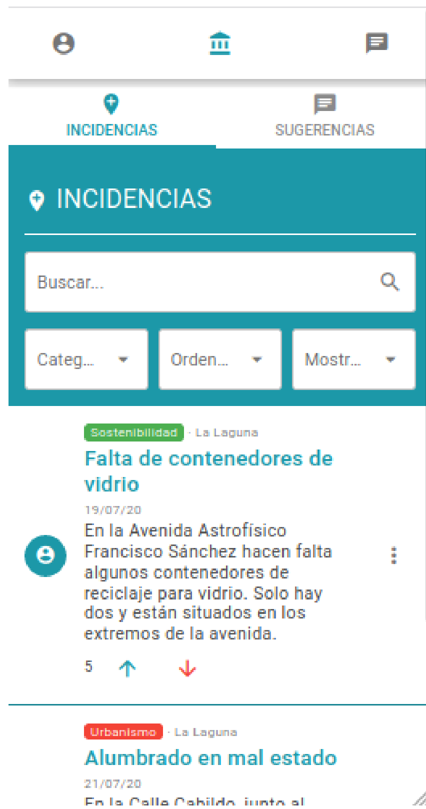


Figura 4.11: Sección Comunica en la versión móvil

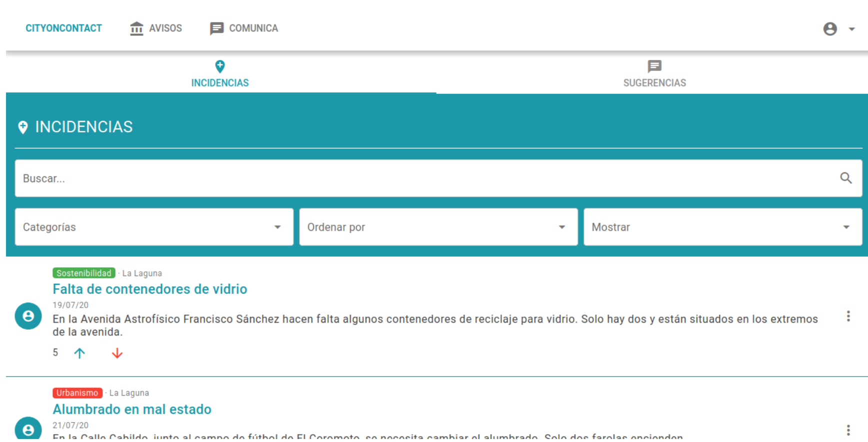


Figura 4.12: Sección Comunica en la versión de escritorio

Capítulo 5

Conclusiones y líneas futuras

A priori, la primera conclusión a la que resulta más evidente llegar es que este proyecto ha servido para conocer mejor la envergadura del desarrollo de una aplicación full stack.

En un principio se inició el proyecto con unos conocimientos limitados y básicos, y lo que se planteó como idea acabó siendo demasiado ambicioso. Al no conocer muchas de las tecnologías usadas, además del tiempo normal de desarrollo se ha tenido que invertir un tiempo extra de estudio y aprendizaje sobre las mismas, junto a los errores de programación que se dan por la inexperiencia y que pueden volverse un quebradero de cabeza al hacer debug si no se han seguido buenas prácticas, como el testing unitario de los componentes de la aplicación.

Se han distinguido dos periodos claros en el desarrollo del proyecto. Por un lado el inicio del desarrollo, a principios del segundo cuatrimestre académico, fue un periodo de toma de contacto con las herramientas marcado por el desconocimiento y un código poco pulido. Dado que este se solapó con la cuarentena y la asignatura de Prácticas Externas, en una empresa dedicada también al desarrollo web pero con otros frameworks, el alumno se vio sobrepasado y decidió pausar el desarrollo para retomarlo más adelante.

Es así como se inicia el segundo periodo, ya con unas bases interiorizadas sobre desarrollo web tras meses practicando en una empresa junto a más desarrolladores junior. Gracias a esto, se llegó a la conclusión de que se necesitaba un refactor general del proyecto y se avanzó dando forma al estado actual del mismo.

Tras haber comprendido que era irreal tratar de proporcionar funcionalidad completa a la aplicación, se decidió tomarse este proyecto como un *pet project* personal para practicar las habilidades y buenas prácticas como desarrollador y conocer frameworks, y eso es algo que se ha conseguido. De hecho, si se pulen un poco más los detalles de implementación obviados para simplificar, como temas relacionados con la seguridad de la sesión o un servicio de envío de emails para la validación de usuarios, el proyecto podría servir como *boilerplate* para futuros proyectos en el stack MEVN.

En cuanto a la temática de la aplicación en sí, personalmente se considera interesante y útil, en especial el tema de avisos y emergencias, que es algo muy poco contemplado en herramientas de este tipo. Pero evidentemente es solo un prototipo, para una implementación real habría que replantear la forma de estructurarla.

Por ejemplo, en la forma de tratar el contenido de las localidades se ha considerado por simplificación reunir las notificaciones de toda una ciudad. Esto en lugares pequeños como la isla de Tenerife podría ser válido, pero en grandes ciudades probablemente se volvería demasiado incómodo de usar por la cantidad de contenido irrelevante de otras zonas lejanas que vería el usuario.

También podría ofrecerse el servicio de la aplicación de forma modular a los ayuntamientos, adaptando la aplicación según sus necesidades, aunque esto limitaría la experiencia de usuario al tener que usar una aplicación web para cada municipio que frecuente.

En líneas generales se considera este proyecto como una experiencia positiva en el desarrollo del alumno. Se ha aprendido sobre varias cuestiones, tanto técnicas de frameworks como de flujo de una aplicación, y es posible que se siga desarrollando como *side project* para practicar y poder incluirlo en la marca personal de la carrera profesional del alumno.

Capítulo 6

Summary and Conclusions

The first and most obvious conclusion is that this project has served to gain a better understanding of the scope of the development of a full stack application.

Initially, the project was started with limited and basic knowledge, and what was proposed as an idea ended up being too ambitious. Since many of the technologies were unknown for the student, in addition to the normal development time it has been necessary to invest an extra time of study and learning about them, along with the programming errors that occur due to inexperience and that can become a problem for debug when good practices have not been followed, such as unit testing of the application components.

Two clear periods in the development of the project have been distinguished. On the one hand, the start of development, at the beginning of the second academic term, was a period of making contact with the tools marked by ignorance and a poorly polished code. Since this overlapped with the quarantine and the External Internship subject, in a company also dedicated to web development but with other frameworks, the student was overwhelmed and decided to pause development to continue it later.

This is how the second period begins, with internalized bases on web development after months practicing in a company with more junior developers. Thanks to this, it was concluded that a general refactor of the project was needed and progress was made in shaping its current state.

Having understood that it was an unreal expectation to provide full functionality to the application, it was decided to take this project as a personal *pet project* to practice the skills and best practices as a developer and learn about frameworks, and that is something that has been achieved . In fact, if all the implementation details, ommited for simplicity, are polished up a bit more, like session security related issues or an email sending service for user validation, the project could serve as *boilerplate* for future projects on the MEVN stack.

As for the theme of the application itself, it is personally considered interesting

and useful, especially the topic of warnings and emergencies, which is something very little considered in this kind of tools. But obviously it is only a prototype, for a real implementation it would be necessary to rethink the way to structure it.

For example, in the way of dealing with the content of the places, it has been considered for simplicity to gather notifications from an entire city. This could be valid in small places like the island of Tenerife, but in big cities it would probably become too inconvenient to use due to the amount of irrelevant content from other distant areas that the user would see.

The service of the application could also be offered in a modular way to the cities, adapting the application according to their needs, although this would limit the user experience by having to use a web application for each city that they frequent.

In general terms, this project is considered a positive experience in the development of the student. Various things have been learnt, both framework techniques and application flow, and it is possible that it will be further developed as a *side project* to practice and be able to include it in the personal mark of the student's professional career.

Capítulo 7

Presupuesto

Dado que la aplicación que propone el proyecto puede plantearse de distintas formas, ya sea tal como está el prototipo o de forma modular adaptándose a las entidades que quieran contratarla, no existe una forma concreta de presupuestarla. A continuación se hará una estimación suponiendo el caso en el que la aplicación funcionará siguiendo el prototipo actual.

Tal como se plantea la aplicación actualmente, existen tres servicios indispensables a contratar: El alojamiento, el almacenamiento de datos y la sucripción a la API para la gestión de mapas interactivos. Para los dos primeros puntos se optará por las versiones de pago de las plataformas que ya se están usando, Heroku y MongoDB Atlas respectivamente. En cuanto a los mapas, se considerará contratar Google Maps por su potencia y fiabilidad.

Para Heroku se considerará su plan estándar Production, de 25$ 22€/mes.

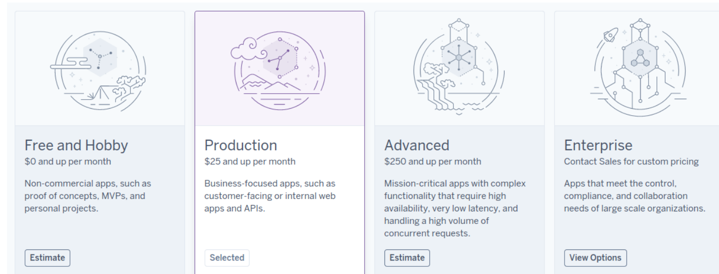


Figura 7.1: Pricing para el alojamiento en Heroku

Para MongoDB Atlas, se seleccinará el plan estándar de 57$ 50€/mes.

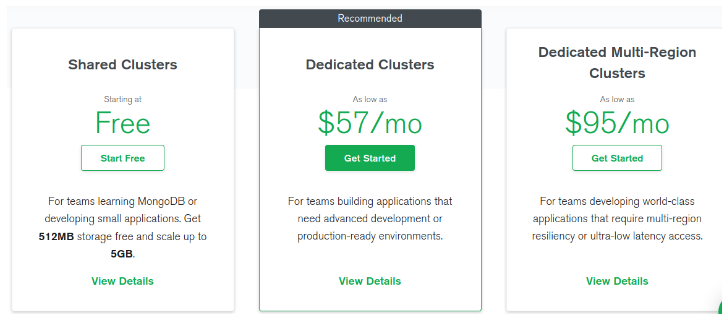


Figura 7.2: Pricing de MongoDB Atlas

En cuanto al servicio de Google Cloud, la estimación resulta más complicada dado que la mayoría de sus servicios se pagan según el flujo de tráfico recibido y la aplicación podría ser escalable desde municipios de zonas concretas a nivel nacional incluso. Se considerará como estimación un precio de 200$ 178€/mes.

MAPS	ROUTES	PLACES
Static Maps Muestra mapas como imágenes	GRATIS PARA MÓVILES	2 USD POR 1000 SOLICITUDES
Dynamic Maps Muestra mapas interactivos y personalizables	GRATIS PARA MÓVILES	7 USD POR 1000 SOLICITUDES
Local Context Map (beta) Ofrece información detallada sobre sitios cercanos, como valoraciones de los usuarios, reseñas, fotos, precios, tiempo estimado de llegada y rutas a pie directamente en tus mapas.	GRATIS DURANTE LA FASE BETA	Es obligatorio habilitar Dynamic Maps (7 USD/mes)
Static Street View Muestra una miniatura o una imagen panorámica no interactiva de Google Street View en 360°.		7 USD POR 1000 SOLICITUDES
Dynamic Street View Muestra una miniatura o una imagen panorámica interactiva de Google Street View en 360°.		14 USD POR 1000 SOLICITUDES

Figura 7.3: Pricing de Google Cloud

Suponiendo además un sueldo para un desarrollador no-senior de unos 1200€/mes, durante unos 6 meses de desarrollo, la estimación del coste total sería la siguiente:

Concepto	Coste mensual	Coste total
Desarrollo de la aplicación	1200€	7200€
Alojamiento	25€	150€
Almacenamiento	57€	342€
Servicios de geolocalización	200€	1200€
Total	1482€	**8892€**

Tabla 7.1: Estimación del presupuesto para el desarrolo del proyecto en 6 meses

Bibliografía

[1] Auth0. Refresh tokens: When to use them and how they interact with JWTs. https://auth0.com/blog/refresh-tokens-what-are-they-and-when-to-use-them/. Accessed: 2020-07-04.

[2] Chai. BDD/TDD assertion library for node.js. https://www.chaijs.com/. Accessed: 2020-07-03.

[3] Travis CI. The simplest way to test and deploy your projects. https://travis-ci.com/. Accessed: 2020-07-03.

[4] Smart App City. La app de las ciudades inteligentes. http://www.smartappcity.com/es/. Accessed: 2020-07-02.

[5] Codecov. The leading code coverage solution. https://codecov.io/. Accessed: 2020-07-03.

[6] Ayuntamiento de Santa Cruz de Tenerife. App SC Farmacias. https://www.santacruzdetenerife.es/opendata/showcase/sc-farmacias. Accessed: 2020-07-05.

[7] Ayuntamiento de Santa Cruz de Tenerife. App SC Mejora. https://www.santacruzdetenerife.es/web/servicios-municipales/servicios-publicos/app-sc-mejora. Accessed: 2020-07-05.

[8] Ayuntamiento de Santa Cruz de Tenerife. App SC Viva. https://www.santacruzdetenerife.es/opendata/showcase/sc-viva. Accessed: 2020-07-05.

[9] ESLint. Pluggeable JavaScript linter. https://eslint.org/. Accessed: 2020-07-03.

[10] Express. Infraestructura web rápida, minimalista y flexible para Node.js. https://expressjs.com/es/. Accessed: 2020-07-03.

[11] Quasar Framework. Build high-performance VueJS user interfaces in record time. https://quasar.dev/. Accessed: 2020-07-03.

[12] Leaflet GeoSearch. Introduction. https://smeijer.github.io/leaflet-geosearch/. Accessed: 2020-07-05.

[13] Git. Open-source version control system. https://git-scm.com/. Accessed: 2020-07-03.

[14] GitHub. The world's leading software development platform. https://github.com/. Accessed: 2020-07-03.

[15] Heroku. Cloud application platform. https://www.heroku.com/. Accessed: 2020-07-03.

[16] Lauri Hiltunen. Managing state with vuex - the guide i wish i'd had. https://dev.to/decoeur_/managing-state-with-vuex---the-guide-i-wish-id-had-28h. Accessed: 2020-07-03.

[17] Jest. JavaScript testing framework with a focus on simplicity. https://jestjs.io/. Accessed: 2020-07-03.

[18] JWT. Open, industry standard RFC 7519 method for representing claims securely between two parties. https://jwt.io/. Accessed: 2020-07-03.

[19] Leaflet. Open-source JavaScript library for mobile-friendly interactive maps. https://leafletjs.com/. Accessed: 2020-07-03.

[20] Vue Leaflet. Harness the power of Leaflet in VueJs. https://vue2-leaflet.netlify.app/. Accessed: 2020-07-05.

[21] Vue Leaflet. Quick start. https://vue2-leaflet.netlify.app/quickstart/#installation. Accessed: 2020-07-05.

[22] MongoDB Atlas. Managed MongoDB hosting. https://www.mongodb.com/cloud/atlas. Accessed: 2020-07-03.

[23] Mongoose. Elegant MongoDB object modeling for Node.js. https://mongoosejs.com/. Accessed: 2020-07-03.

[24] Node.js. Entorno de ejecución para JavaScript. https://nodejs.org/es/. Accessed: 2020-07-03.

[25] Nominatim. Nominatim usage policy. https://operations.osmfoundation.org/policies/nominatim/. Accessed: 2020-07-05.

[26] NPM. Node package manager. https://www.npmjs.com/. Accessed: 2020-07-04.

[27] PivotalTracker. Agile project management. https://www.pivotaltracker.com/. Accessed: 2020-07-03.

[28] Quasar. Quasar cli installation. https://quasar.dev/quasar-cli/installation. Accessed: 2020-07-04.

[29] React. Una librería de javascript para construir interfaces de usuario. https://es.reactjs.org/. Accessed: 2020-07-03.

[30] Redux. Redux docummentation. https://redux.js.org/introduction/getting-started. Accessed: 2020-07-03.

[31] Vue Router. Vue router documentation. https://router.vuejs.org/. Accessed: 2020-07-03.

[32] Sinon.js. Standalone test spies, stubs and mocks for JavaScript. https://sinonjs.org/. Accessed: 2020-07-03.

[33] Francisco Ugalde. Introducción a la arquitectura hexagonal. https://www.franciscougalde.com/2019/09/17/introduccion-a-la-arquitectura-hexagonal/. Accessed: 2020-07-05.

[34] Vue.js. The progressive javascript framework. https://vuejs.org/. Accessed: 2020-07-03.

[35] Vuex. State management pattern + library for Vue.js applications. https://vuex.vuejs.org/. Accessed: 2020-07-03.

[36] Excellent Webworld. What is a single page application? meaning, pitfalls and benefits. https://www.excellentwebworld.com/what-is-a-single-page-application/. Accessed: 2020-07-05.