UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

GRADO EN INGENIERÍA DEL SOFTWARE

PLATAFORMA DE CONTROL Y PREVENCIÓN DE EMERGENCIAS

EMERGENCY PREVENTION AND CONTROL PLATFORM

Realizado por
JAVIER PARADA TALLANTE

Tutorizado por
EDUARDO GUZMÁN DE LOS RISCOS

Departamento
LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

MÁLAGA, OCTUBRE 2020

RESUMEN

Plataforma concebida como un centro de control que permita a los servicios de emergencias (Protección Civil, Fuerzas de Seguridad del Estado, etc.) gestionar la información que se envía a través de sus cuentas en medios de comunicación como redes sociales y proporcionar de forma ordenada y categorizada información.

La información podrá ser monitorizada y permitirá aportar información pública como protocolos (en diversos formatos multimedia tales como PDF, imágenes, etc.) ante situaciones de emergencia, que faciliten el acceso a esta y la comunicación de forma directa con la población.

Además, la información de las emergencias estará geolocalizada en función de su ámbito geográfico de aplicación y/o generalizada para otros casos de emergencia posibles, disponibles en la plataforma. Se facilitará la generación de códigos QR con esta información para facilitar y agilizar el acceso a todos los datos requeridos en cada posible situación de cada posible lugar, siendo fácilmente ubicables en cualquier sitio.

Se permitirá interactuar al usuario normal (civil) con la plataforma, pudiendo generar sus propios QR, agrupando información útil si conoce el terreno o alguna zona crítica.

La plataforma contará también, para abarcar el máximo número de personas posibles, incluidas las que no utilicen las redes sociales, una sección con las últimas emergencias sucedidas y un navegador para encontrar mediante criterios de búsqueda y palabras clave la información deseada.

PALABRAS CLAVE

Aplicación para Prevención y Control de Emergencias, MongoDB, Node.js, Vue, Electron, Códigos QR

ABSTRACT

Platform conceived as a control center that allows to emergency services (Civil Protection, State Security Forces, etc.) to manage the information sent through their accounts in media such as social networks and provide in an orderly and categorized manner all information.

The information may be monitored and will allow the provision of public information such as protocols (in various multimedia formats such as PDF, images, etc.) in emergency situations, which facilitate access to it and direct communication with population.

In addition, the information on emergencies will be geolocated according to its geographical scope of application and/or generalized for other possible emergency cases, available on the platform. The generation of QR codes with this information will facilitate and speed up access to all the required data in each possible situation of each possible place, being easily located anywhere.

The normal (civil) user will be allowed to interact with the platform, being able to generate their own QR, grouping useful information if they know the terrain or any critical area.

The platform will also have to cover the maximum number of people possible, including those who do not use social networks, a section with the latest emergencies that have occurred and a browser to find the desired information using search criteria and keywords.

KEYWORDS

Emergency Prevention and Control Application, MongoDB, Node.js, Vue, Electron, QR Code

Tabla de contenido

1- Introducción

1.1- Motivación y objetivos del TFG

La necesidad del proyecto surge de la actual escasez de medios para informar a la población ante emergencias tales como desastres naturales, atentados terroristas o emergencias particulares asociadas a terrenos peligrosos. Asimismo, la información relativa a protocolos de emergencias de las diferentes provincias españolas está dispersa y es, por tanto, difícil de localizar perdiendo por esta razón su utilidad. Surge por tanto la necesidad de un medio para la prevención y control de estos mediante canales de comunicación directos con la población.

El objetivo es desarrollar una plataforma concebida como un centro de control que permita a los servicios de emergencias (Protección Civil, Fuerzas de Seguridad del Estado, etc.) gestionar la información que se envía a través de sus cuentas en medios de comunicación por redes sociales, monitorizarla y aportar información pública como protocolos (en diversos formatos multimedia tales como PDF, imágenes, etc.) ante situaciones de emergencia, que faciliten la comunicación de forma directa con la población. La información de protocolos estará geolocalizada en función de su ámbito geográfico de aplicación y se facilitará además la generación de códigos QR con accesos directos a esta información fácilmente ubicables en cualquier sitio.

1.2- Tecnologías y herramientas principales utilizadas

Las tecnologías utilizadas para el proyecto se han elegido en función de la adaptabilidad de estas a cada requisito de cada parte del software. Por lo tanto, se exponen a continuación ordenadas por componente y con una breve descripción de la tecnología, explicación de su papel en el proyecto y el motivo de su elección en caso de ser relevante.

1.2.1- Tecnologías y herramientas utilizadas en el sistema de persistencia

- MongoDB[1]: Base de datos de tipo No Relacional, de propósito general y basada en documentos. Es el sistema de persistencia general del proyecto, en el que se almacenan todos los datos de este. El motivo de su elección es su adaptabilidad a servicios de computación en la nube, su velocidad frente a otros servicios de

[1] MongoDB, Inc 2019. MongoDB La base de datos para aplicaciones modernas. https://www.mongodb.com/es

bases de datos ante grandes volúmenes de datos y la adaptabilidad de los propios datos ante futuros cambios debido a que este está basado en documentos.

- Robo3T[2]: Interfaz gráfica para interactuar de forma sencilla con bases de datos MongoDB. Su papel en el proyecto se reduce únicamente al proceso de desarrollo para realizar comprobaciones y modificaciones rápidas en la base de datos. El motivo de su elección es su sencillez y tamaño.

- Studio3T[3]: Interfaz gráfica para interactuar de forma sencilla con bases de datos MongoDB. Su papel en el proyecto se reduce únicamente al proceso de desarrollo para realizar consultas complejas en las que intervienen múltiples entidades y/o atributos y probarlas antes de añadirlas al proyecto. El motivo de su elección es su sencillez que proporciona esta interfaz para realizar dichas consultas, aunque permitiendo consultas más complejas que la anterior herramienta.

1.2.2- Tecnologías y herramientas utilizadas en el lado del servidor

- Java[4]: Lenguaje de programación de propósito general, orientado a objetos. Su papel en el proyecto es la de lenguaje de programación de la aplicación web del lado del servidor. El motivo de su elección es principalmente su robustez y a la posibilidad de ejecutarse en diferentes plataformas.

- Spring Boot[5]: Framework para el desarrollo de aplicaciones web en el lenguaje JAVA. En el proyecto, la aplicación web del lado del servidor es un proyecto en este framework. Es empleado para generar un API basado en el estilo de arquitectura REST. Su elección radica en la cantidad de soporte por parte de la comunidad de esta tecnología, su robustez y la cantidad de elementos o componentes proporcionados en esta tecnología (incluidas también en esta sección).

- Spring Initializr[6]: Sitio web generadora de proyectos Spring Boot. Con este generador se ha creado inicialmente el proyecto del lado del servidor. Se ha optado por este generador gracias a la facilidad que aporta a la hora de escoger las librerías y elementos básicos que el proyecto dispondrá desde el principio.

---

[2] 3T Software Labs, 2017. Robo3T. https://robomongo.org/
[3] 3T Software Labs, 2017. Studio3T. https://robomongo.org/
[4] Oracle, ¿Qué es Java? https://www.java.com/es/about/whatis_java.jsp?bucket_value=desktop-firefox68-osx1014&in_query=no
[5] Pivotal Software Inc 2019, Spring By Pivotal, Spring Boot https://spring.io/projects/spring-boot
[6] Pivotal Software 2013 – 2019, Spring initializr, arranca tu aplicación, https://start.spring.io/

- Gradle[7]: Gestor de paquetes y dependencias para proyectos Java. Gradle es el gestor de dependencias utilizado en el proyecto del lado del servidor. Se ha escogido por simple preferencia personal frente a otras opciones como Maven.

- Dependencias de Spring Framework[8]: Aunque existe gran variedad de librerías con las que gestionar las diferentes funcionalidades del proyecto, se ha intentado dar preferencia a las predefinidas por Spring para reducir el número de dependencias de terceros incluidas en este componente, ya que es el componente mas crítico del sistema y en el que prima facilitar la resolución de futuras vulnerabilidades en las dependencias del sistema. Además del gran soporte de la comunidad de Spring que reciben todas estas librerías frente a otros proyectos.

    - Spring Boot Starter Data MongoDB[9]: Librería de Spring Boot para gestionar el acceso a bases de datos MongoDB.

    - Spring Boot starter Security[10]: Librería de Spring Boot para gestionar la seguridad y autenticación en la aplicación del lado del servidor.

    - Spring Boot Starter Web[11]: Librería de Spring Boot con múltiples recursos para construir un componente o aplicación web. Entre los que se incluyen recursos para construir un API REST.

    - Spring Boot Starter Test[12]: Librería de Spring Boot para el Desarrollo de pruebas de software para la aplicación web.

    - Spring Security Test[13]: Sublibrería de Spring Security para realizar pruebas de Software relacionadas con la seguridad de la aplicación web.

[7] Gradle Inc, 2019, Gradle Build Tool, https://gradle.org/

[8] Todas las librerías mencionadas son visibles junto con su versión, organización y nombre completo en el fichero "build.gradle" en el directorio raíz del módulo "Back-end".

[9] MvnRepository 2006-2019, Spring Boot Data MongoDB Starter, https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-data-mongodb

[10] MvnRepository 2006-2019, Spring Boot Starter Security, https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-security

[11] MvnRepository 2006-2019, Spring Boot Web Starter, https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-web

[12] MvnRepository 2006-2019, Spring Boot Starter Test, https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-test

[13] MvnRepository 2006-2019, Spring Boot Security Test, https://mvnrepository.com/artifact/org.springframework.security/spring-security-test/4.0.0.RELEASE

- Dependencias Externas[14]: Pese a lo citado anteriormente, se han incluido algunas librerías de terceros debido a que, o bien su funcionalidad no estaba incluida en ninguna librería predefinida de Spring o bien tras un balance se ha optado por utilizar una librería adicional por motivos que se adjuntarán en cada una en caso de así serlo.

    - IO JSON Web Token jjwt[15]: Librería para gestionar tokens de sesión bajo el RFC 7519[16]. Generarlos, encriptarlos, firmarlos etc. El motivo por el que se ha optado por esta tecnología es su popularidad, soporte de la comunidad y facilidad de uso.

    - Apache Org commons-collections4[17]: Librería para manejar listas e iteradores con más facilidad. Destacada por utilizar las librerías predefinidas de Spring muchos de los tipos y estructuras a los que esta librería va dirigida.

    - Apache HTTPComponents[18]: Librería para realizar peticiones bajo el protocolo HTTP.

    - Oauth SignPost[19]: Librería compatible con Apache Http para definir con facilidad en sus cabeceras, parámetros para el protocolo Oauth 1.0.

    - Twitter4j[20] y Facebook4j[21]: Librerías de código libre para el fácil uso y simplificación de las APIs de Facebook y Twitter, principales redes sociales de este proyecto.

- Intellj IDEA CE[22]: Entorno de desarrollo utilizado para desarrollar la aplicación web del lado del servidor. El motivo por el que se ha escogido esta opción es una preferencia personal.

[14] Todas las librerías mencionadas son visibles junto con su versión, organización y nombre completo en el fichero "build.gradle" en el directorio raíz del módulo "Back-end".
[15] Auth0, JWT, https://jwt.io/
[16] RFC7519, JSON Web token (JWT), https://tools.ietf.org/html/rfc7519
[17] The Apache Software Foundation, 2001-2019, Apache Commons Collections, https://commons.apache.org/proper/commons-collections/
[18] The Apache Software Foundation, 2005-2019, Apache HttpComponent, https://hc.apache.org/
[19] Matthias Käppler, Signpost, https://github.com/mttkay/signpost
[20] Yusuke Yamamoto 2017, twitter4j, http://twitter4j.org/ja/
[21] Ryuji Yamashita, 2012, facebook4j, https://facebook4j.github.io/ja/index.html
[22] Jetbrains s.r.o, 2000-2019, IntellJ Idea, https://www.jetbrains.com/idea/download/

- Postman[23]: Software con interfaz gráfica de usuario para realizar llamadas basadas en el protocolo HTTP. Su uso principal es el de hacer pruebas y consultas sobre la aplicación del lado del servidor durante la fase de desarrollo. Se ha escogido este software por su facilidad de uso y la posibilidad de generar documentos con todos los tipos de llamadas HTTP de un servidor.

1.2.3- Tecnologías y herramientas utilizadas en el lado del cliente

- JavaScript: Lenguaje de programación orientado a objetos y a eventos. Su papel en el proyecto es de lenguaje de programación de la aplicación en el lado del cliente. Dentro de JavaScript se ha optado por utilizar el dialecto del lenguaje ECMAScript 6. Respecto a las diferentes versiones de ECMAScript se ha optado por esta por su simplicidad, la capacidad de modularización y todas las nuevas funcionalidades que ofrece frente a las versiones más antiguas soportadas por navegadores. Por otro lado, respecto a otros lenguajes de programación precompilados como TypeScript o Coffee-Script, se ha optado por usar simplemente JavaScript por su integración con el framework utilizado (y citado posteriormente en esta sección) así como por la diferencia de soporte frente a esta.

- Node js[24]: Entorno de ejecución de código JavaScript. Es el entorno con el que ejecutar la aplicación web del lado del cliente durante las fases de desarrollo, ya que los navegadores convencionales no soportan JavaScript bajo ECMAScript 6[25].

- Vue[26]: Framework JavaScript progresivo para la construcción de interfaces dinámicas y reactivas. Es la tecnología principal de la aplicación web del lado del cliente. El motivo de su elección frente a otras opciones como React o Angular es por preferencia personal. Además de Vue se han utilizado algunas librerías propias de este framework tales como:

    - Vue Router[27]: Librería para el framework Vue, con el que gestionar de forma dinámica las rutas de la aplicación web.

[23] Postman Inc. 2019, Postman, https://www.getpostman.com/products
[24] Linux Foundation, Node.js, https://nodejs.org/es/
[25] Linux Foundation, ECMASript6 and beyond, https://nodejs.org/es/docs/es6/
[26] Evan You, 2014-2019, Vue.js, The Progressive JavaScript Framework, https://vuejs.org/
[27] Evan You, 2013-2019, Vue Router, https://router.vuejs.org/

    - Vuex[28]: Librería para el framework Vue para facilitar la gestión de memoria y el acceso a API REST.

- HTML5: Lenguaje de marcado para el desarrollo de páginas web.

- Sass[29]: Lenguaje de hoja de estilos precompilada para proyectos web. En el proyecto para definir los estilos de la aplicación en el lado del cliente. Su uso frente al convencional CSS3 se debe principalmente a la facilidad para estructurar el código que concede, mediante definición de variables etc.

- Bulma[30]: Framework CSS basado en Flexbox compatible con Sass. Utilizado al igual que Sass para definir los estilos de la aplicación web del lado del cliente. El principal motivo de su uso frente a otras opciones como Bootstrap es que Bulma utiliza únicamente CSS y Sass para sus componentes, por lo que tiene una mayor integración con proyectos Node.js y no depender además de librerías adicionales de terceros como podría ser jQuery o usar código JavaScript.

- Babel[31]: Empaquetador de código JavaScript (ECMAScript 6) a JavaScript utilizable por un navegador web convencional. El motivo de su uso es de eficiencia, ampliar las posibilidades de uso y la compatibilidad con navegadores convencionales.

- Visual Studio Code[32]: Editor de código multiplataforma y multilenguaje. Utilizado para el desarrollo de la aplicación web del lado del cliente.

- npm[33]: Gestor de paquetes JavaScript (a continuación describen las principales dependencias).

- Axios[34]: Librería de código abierto para hacer llamadas bajo el protocolo HTTP. Esta librería se ha escogido frente a otras posibilidades como Vue-Resources para hacer una separación diferenciada entre el API, diseñada empleando el patrón de diseño fachada (facade), de los componentes y gestores de memoria de Vue.

[28] Evan You, 2015-2019, Vuex, https://router.vuejs.org/
[29] Sass, 2006-2019, Sass, https://sass-lang.com/
[30] Jeremy Thomas, 2019, Bulma, https://bulma.io/
[31] Sebastian McKenzie, 2014-2019, Babel, https://babeljs.io/
[32] Microsoft, 2019, Visual Studio Code, https://code.visualstudio.com/docs
[33] Npm Inc., npm, https://docs.npmjs.com/
[34] Matt Zabriskie, 2014-2019, Axios, https://github.com/axios/axios

- Github-Markdown-CSS: Librería con diferentes estilos predefinidos para componentes y textos con formato "Markdown"

- Leaflet[35]: Librería para JavaScript para la generación de mapas dinámicos. El motivo por el que esta librería se ha escogido frente a otras opciones como "Google Maps", es el de la independencia en la mayor medida posible de servicios externos de pago o limitados en sus versiones gratuitas de software de terceros. Especialmente si, como en este caso, no es indispensable o no conlleva una perdida de calidad significativa del resultado final.

- Vue2-Leaflet[36]: Adaptador para Vue de la librería LeafLet.

- Vue qrcode component[37]: librería para la generación dinámica de códigos QR.

- Vue simple markdown[38]: Librería para la conversión dinámica de textos en formato "Markdown" a formato "HTML".

- Vue simplemde[39]: Librería que proporciona un editor completo y dinámico en formato markdown.

- PDFmake[40]: Librería para la generación de PDFs con diferentes estilos y múltiples posibles configuraciones.

- FontAwesome[41]: Librería de iconos e imágenes vectoriales de código abierto adaptada para aplicaciones con Vue.js.

1.2.4- Otras Tecnologías y herramientas utilizadas

- git[42]: Controlador de versiones de proyecto.

[35] Vladimir Agafonkin, 2010-2019, Leaflet, https://leafletjs.com/
[36] Mickael Bouchand, 2016-2017, Vue2Leaflet, https://korigan.github.io/Vue2Leaflet/#/
[37] Gerard Reches, 2016, Vue QRComponent, https://gerardreches.github.io/vue-qrcode-component/
[38] Milan Backonja, 2017, Vue Simple Markdown, https://gerardreches.github.io/vue-qrcode-component/
[39] F-loat, 2019, Vue Simple MdE, https://f-loat.github.io/vue-simplemde/
[40] Bpampuch, 2014, PDFMake, http://pdfmake.org/#/
[41] Fonticons Inc, FontAwesome, https://fontawesome.com/start
[42] Git, https://git-scm.com/

- GitHub[43]: Servicio de almacenamiento en la nube de repositorios Git. Es el servicio de almacenamiento de todo el proyecto, incluyendo su documentación y memoria.

- CodeFactor[44]: Software evaluador de código en base a los estilos y la limpieza de este. Utilizado en cada "commit" del proyecto.

- Es-lint[45]: Librería para marcar errores estilísticos conforme a los estándares de ECMAScript.

- Github Pages[46]: Servicio de Hosting de Github para alojar páginas estáticas.

- Heroku[47]: Servicio de computación en la nube.

- mLab[48]: Servicio en la nube de almacenamiento de bases de datos tipo MongoDB.

- Docker[49]: Servicio de virtualización en contenedores.

1.3- Estructura de la memoria

En el resto de la memoria se abordarán diferentes temas desde la parte más general o conceptual, como pueden ser normas de estilo, consideraciones o diagramas, a la parte más concreta, como pueden ser implementaciones, algoritmos utilizados, secciones de librerías etc.

Para ello también se seguirá un orden lógico junto al proceso de desarrollo seguido, y profundizando en cada etapa tomando en consideración partes concretas de la ingeniería del software e implementaciones. Por ejemplo, para abordar la aplicación del lado del cliente, se hablará de la arquitectura seguida, los estilos comunes y de la interacción que los diferentes componentes tienen entre sí, para posteriormente analizar los componentes y los detalles mas particulares.

[43] Github Inc. 2019, Github, https://github.com/
[44] Codefactor 2019, Codefactor, https://www.codefactor.io/
[45] JS Foundation, eslint, https://eslint.org/
[46] Github Inc, 2019, Github Pages, https://pages.github.com/
[47] Salesforces 2019, Heroku, https://www.heroku.com/
[48] ObjectLabs Corporation 2018, mLab Database as a service, https://mlab.com/
[49] 2020 Docker Inc, Docker, https://www.docker.com

2- Ingeniería del Software

2.1- Ingeniería de Requisitos

Previo a todas las demás etapas del proyecto, se realizó una fase intensiva de toma de los requisitos principales, a partir de la cual se elaboraron una serie de diagramas UML en los que se describen todos los detalles del software a realizar.

Los requisitos principales giran en torno a la eficiencia y eficacia de cara al usuario, es decir, fácil y rápido acceso por parte del usuario medio (población civil) que desea conocer los detalles de protocolos y diversa información de utilidad. Y, por otro lado, un sistema o plataforma centrado en la seguridad y simplicidad para el envio y/o puesta a disposición de la población la ya mencionada información.

Los primeros requisitos se toman mediante la elaboración de casos de uso junto a breves descripciones, que se adjuntan a continuación.

2.2- Registro e inicio de sesión

La aplicación constará de una parte privada, desde la cual el personal cualificado de las instituciones a las que va dirigido este trabajo podrá acceder y realizar cambios, información etc. Para ello debemos separar esta área y mantenerla seguro mediante un inicio de sesión con usuario y contraseña.

Por otro lado, también se necesita de un registro, para añadir los nuevos usuarios que se harán cargo y operarán a través de la plataforma.

En la *Ilustración 1* se observa cómo, el usuario normal (sin estar registrado en la base de datos) accede al registro, pero necesita de la aprobación de este. Un usuario que se haya registrado previamente y que posea a su vez los privilegios necesarios podrá aprobar y asignar los privilegios correspondientes a un nuevo usuario o rechazar el registro del mismo.

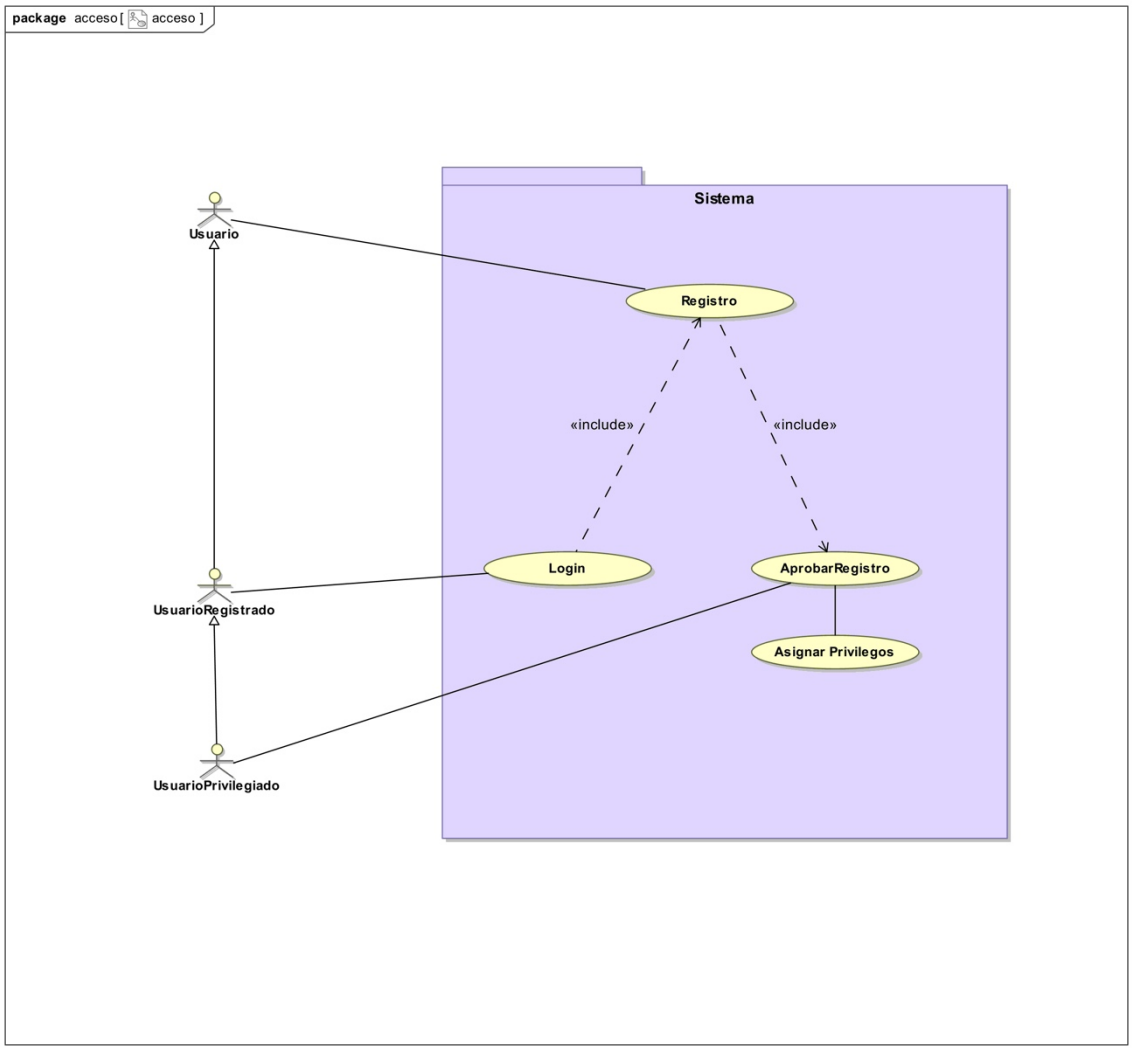

Ilustración 1: Proceso de Inicio de sesión y registro

2.3- Administración de roles y privilegios

Una de las características más importantes que debe poseer la plataforma es una gran versatilidad a la hora de gestionar los roles y privilegios dentro de cada institución, ya que, al ser utilizada por diferentes entidades, las necesidades de cada una son muy diferentes. Para ello, se crean una serie de privilegios básicos, en función de cada una de las funcionalidades que posee la aplicación. Sin embargo, los usuarios no dispondrán de estos privilegios directamente, sino que estos tendrán un "rol". Siendo un rol un conjunto de privilegios asociado a los usuarios de la institución que crea este rol.

De esta forma y como se muestra en la *Ilustración 2,* un usuario con el rol y privilegios adecuados puede crear nuevos roles, asignarlos, modificarlos o eliminarlos. También es destacable que todas las operaciones que realiza el administrador de roles pasan por los mismos filtros de privilegios que cualquier otra operación. De ahí que estas hereden del mismo caso de uso.

Además, la creación de los roles está relacionada con una institución, algo que será comprobado de forma implícita con la información de sesión del usuario.

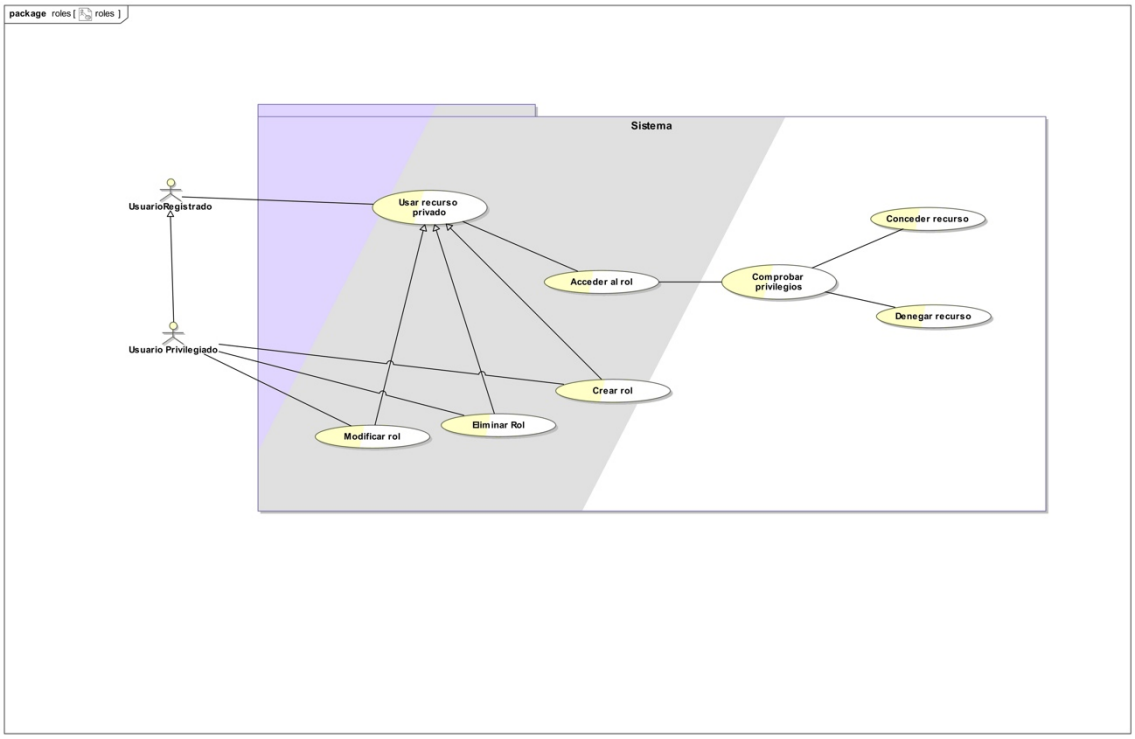

Ilustración 2 Proceso de administración de roles y privilegios

2.4- Generación de contenido

Dentro de la plataforma hay diferente contenido, dependiendo de la funcionalidad de este. Encontramos:

- Protocolos: información útil ante una situación peligrosa.
- Emergencias: conjuntos de protocolos útiles ante determinada situación de emergencia o situación peligrosa.
- Alertas: Instancias de emergencias que han sucedido o van a suceder próximamente y que, por supuesto, llevan enlazados la información correspondiente a dicha emergencia.

Esta información traza una estructura tipo árbol, en la que se puede navegar

Para la generación de contenido, se necesitan unos privilegios concretos.

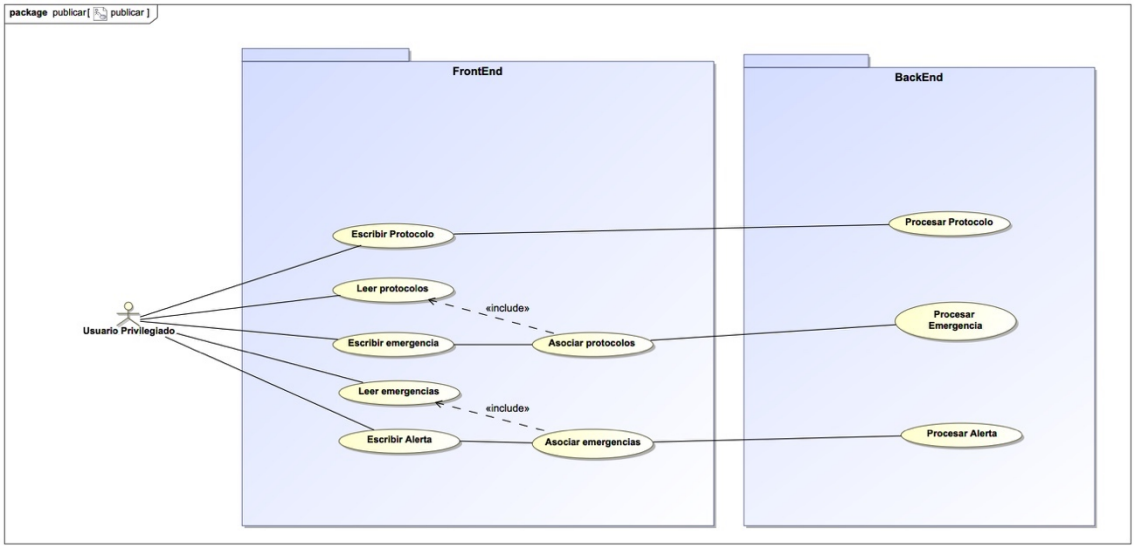

Ilustración 3: Proceso de publicación de contenido

Como se observa en la imagen (Ilustración 3), el orden lógico necesario para llegar a publicar una alerta es: Publicar uno o más protocolos, publicar una emergencia asociada a uno o más protocolos y, por último, publicar una alerta, asociada a una emergencia.

2.5- Obtener información

Los casos de uso en relación con la obtención de información son el eje principal de la aplicación, ya que es el objetivo de esta: informar por los medios y de las formas más accesibles posibles a la población. Estas son:

- Filtros y criterios de Búsqueda: Introducción de palabras clave en un buscador que nos permita obtener los protocolos, emergencias y alertas deseadas.

- Códigos QR localizados en cualquier parte: Mediante la lectura y enrutamiento delos códigos QR que nos muestren exactamente la información que necesitamos.

- Redes sociales: Mediante enlaces y mensajes a través de las redes sociales. Ya sean haciendo referencias directas a la información de la plataforma, o incluyendo esta en los propios mensajes.

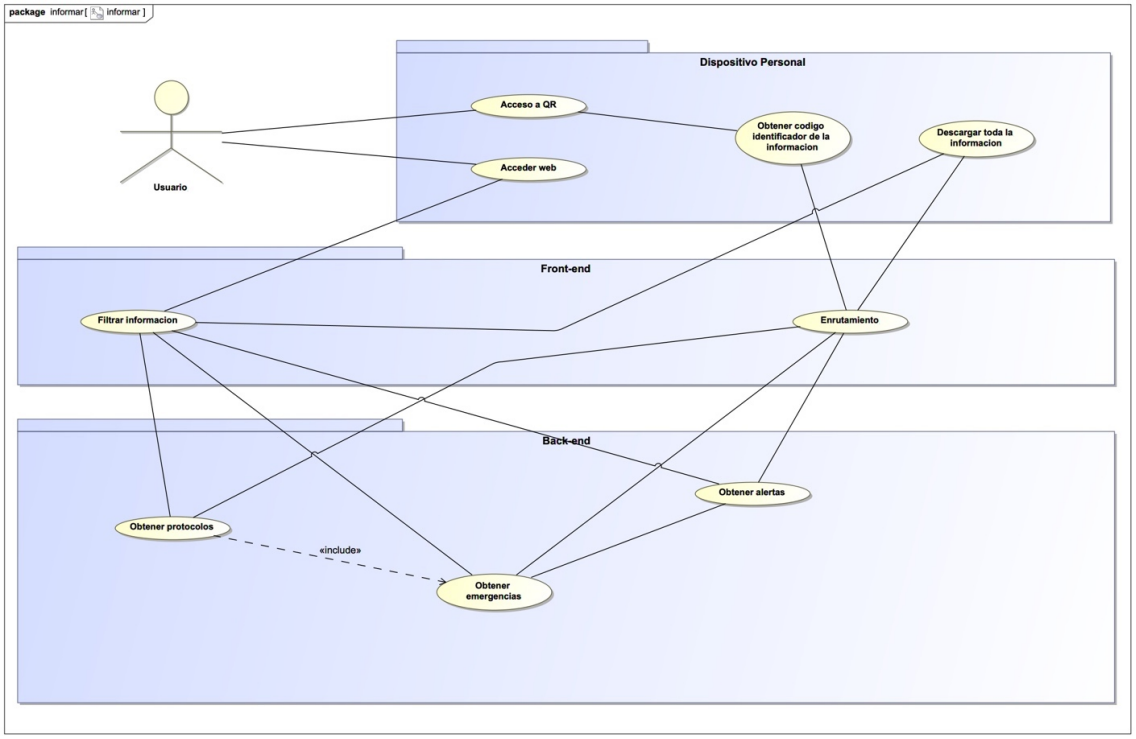

Ilustración 4: Proceso de acceso al contenido

2.6- Arquitectura y distribución del sistema

El sistema formará parte de un entorno distribuido con diferentes componentes software, ya que este requiere de diferentes partes con diferentes características y necesidades. En la siguiente imagen (Ilustración 5) se muestra un esquema simplificado del sistema distribuido.

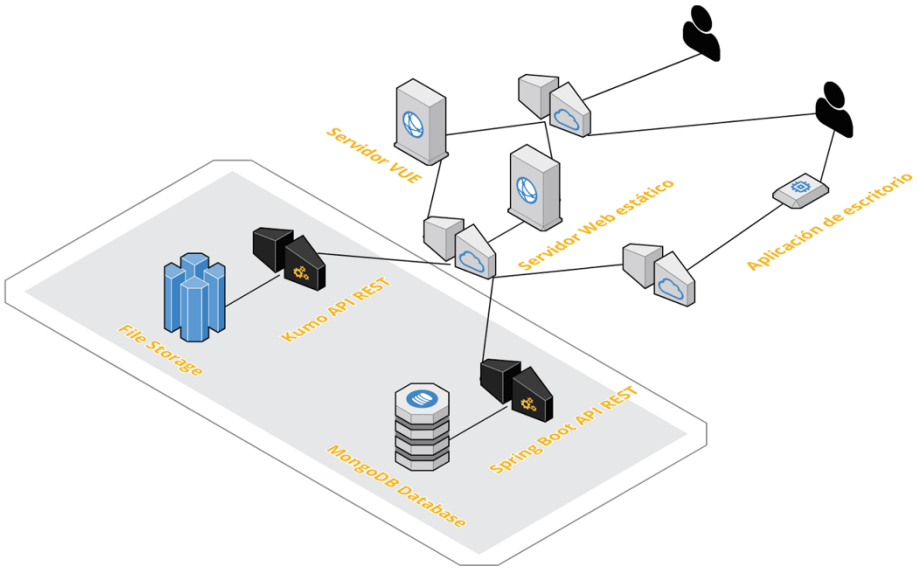

Ilustración 5: Diagrama del sistema distribuido[50]

En la zona con la base más oscura se encuentra la aplicación del lado del servidor. Los principales requisitos de esta parte son la seguridad y un buen tiempo de respuesta de la información requerida.

A un lado, podemos observar una base de datos de tipo MongoDB, que almacenará toda la información introducida por las instituciones gestoras del software (Protocolos, emergencias etc.) y un API REST para acceder a esta.

Para mejorar el rendimiento de la aplicación, las imágenes y archivos multimedia no solo no se almacenarán en la misma base de datos/sistema de persistencia, sino que se utilizará una aplicación diferente para almacenarlas. La aplicación/software que se encarga del procesamiento de imágenes se llama "Kumo[51]", un software realizado por mí mismo y adaptado para el proyecto.

De esta forma, utilizando un servidor Kumo, la aplicación principal no necesita procesar imágenes o videos que tienen un gran tamaño y que pueden ralentizar el servicio principal. También cabe destacar, que los servidores o aplicaciones de Kumo y la principal no se comunican de forma directa en ningún momento, ya que si las imágenes tienen que pasar por la aplicación principal se presentaría el mismo problema. La solución para este caso es utilizar un protocolo basado en tokens, generados por la aplicación principal que permitirá al usuario utilizar recursos de ambas aplicaciones.

Este protocolo esta inspirado en una simplificación del protocolo Oauth 2.0 que utiliza Google. En este caso, es el utilizado para el inicio de sesión de usuarios mediante Gmail en aplicaciones externas.

[50] Imagen generada mediante CloudCraft, Cloudcraft Inc. https://www.cloudcraft.co
[51] Javier Parada Tallante, 2019, Kumo, https://www.javierparada.dev/kumo/

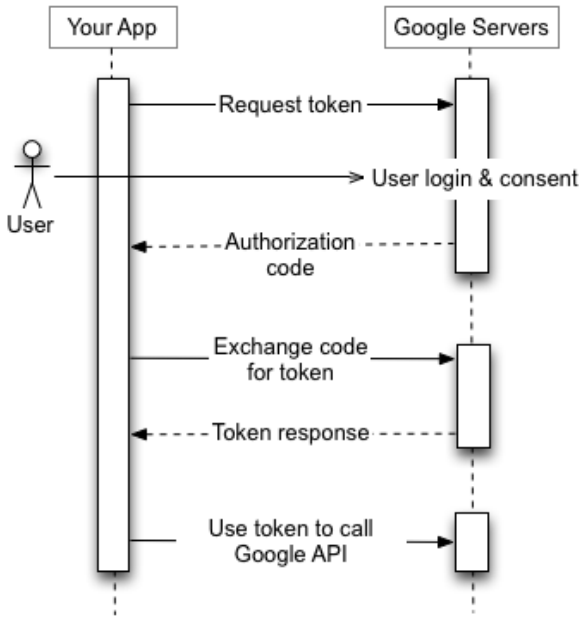

Ilustración 6: Diagrama OAuth de Google[52]

Saliendo del lado del servidor nos encontramos con el lado del cliente, donde destacan:

- Servidor VUE: Esto es un servidor dedicado a aplicaciones del framework Vue. Es un servidor dinámico de renderizado en el lado del servidor. Además, es un servidor muy eficiente para el cliente, ya que procesa todas las páginas y componentes en lugar de su dispositivo, sin alterar el SEO de la aplicación. El SEO (*Search Engine Optimization*) son una serie de técnicas para optimizar el posicionamiento de páginas web en buscadores (u otro tipo de productos en otros contextos, como aplicaciones móviles).

- Servidor estático: Servidor de páginas estáticas (formatos HTML, CSS y JS). Este es el servidor que se recomienda usar, ya que es el más eficiente para el proveedor de la infraestructura. Aunque de cara al cliente no es tan eficiente como el servidor dedicado Vue, este se ejecuta completamente en el lado del cliente, actuando como un simple servidor FTP de lectura para el código. Aunque el proyecto no está realizado en estas tecnologías, este se compilará y optimizará en ellas. Una vez compilado, la carga de componentes y código se realiza mediante "carga perezosa", es decir, solo se cargarán del servidor los componentes que se necesitan en cada momento.

- Aplicación en Electron: Para los administradores de la aplicación, se proporciona además una aplicación de escritorio con exactamente las mismas

[52] Google LLC, Google Identity Platform, https://developers.google.com/identity/protocols/OAuth2

características que el panel Web. El motivo principal de esta aplicación es aligerar la carga de peticiones al servidor del lado del cliente, teniendo una conexión directa y sin intermediarios con el lado del servidor. Además, en comparación con el área pública, la sección privada es significativamente más pesada y requiere de mayores recursos y librerías a cargar. Por lo que, si estos se encuentran almacenados localmente y no en internet, aligerarán el proceso.

2.7- Modelado y Diseño de Software

Una vez definidos la arquitectura y los casos de uso principales de la aplicación, se desarrolla el diagrama de clases principal. Este diagrama representa la interconexión de los datos en la base de datos y las formas de acceso a los mismos. Se destaca que este diagrama no representa al modelo de la base de datos, ya que esta es no relacional, pero sí una forma simplificada del sistema.

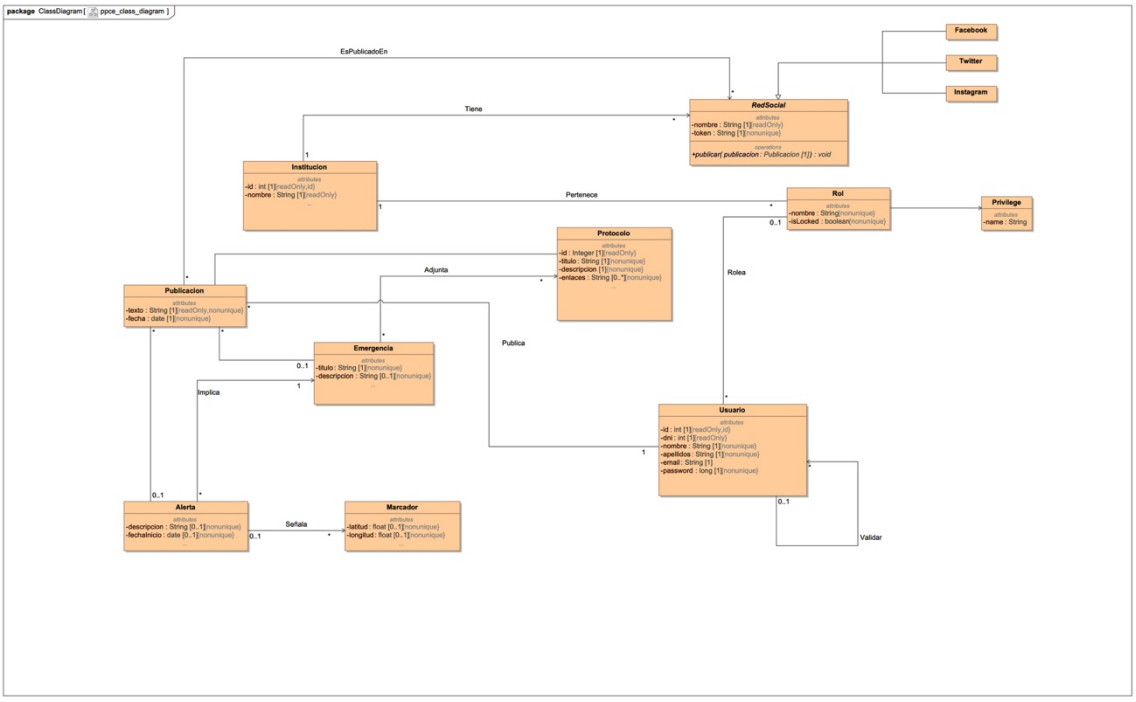

Ilustración 7: Diagrama de clases general

A continuación, se analizarán cada una de las clases y relaciones.

2.7.1- Usuario

El objetivo de la clase usuario es la de almacenar los datos identificadores de los usuarios registrados de la aplicación. La información básica serán un correo electrónico, como identificador del resto de usuarios y una contraseña para garantizar el acceso.

El registro a la aplicación será libre, cualquier persona puede realizar un registro, aunque una vez hecho, el registro debe ser aprobado por otro usuario con los permisos necesarios.

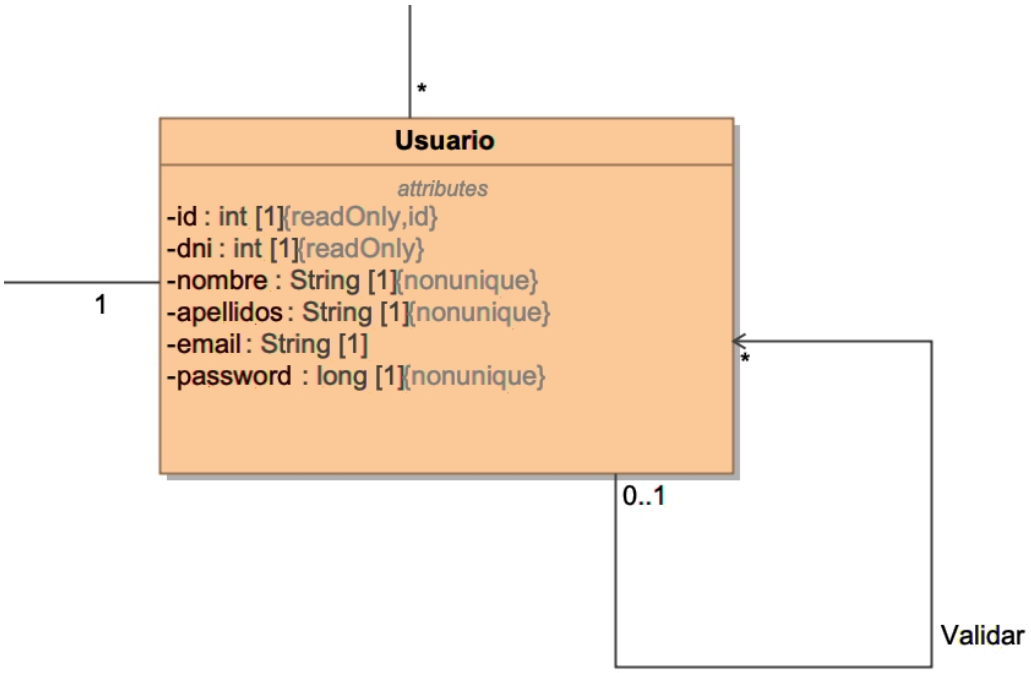

Ilustración 8: Usuario

Atributos:

- ID: Identificador del usuario. Este identificador se usa para indexar todos los usuarios y acceder rápidamente a ellos sí se posee esta referencia.

- DNI: Puesto que este sistema esta diseñado para instituciones públicas, será necesario añadir un identificador de la persona física. Además, junto con los dos siguientes atributos (nombre y apellidos) podrá diferenciarse a cada persona que realiza una petición de registro.

- Nombre: Nombre del usuario.

- Apellidos: Apellidos del usuario.

- Email: Identificador único de cada usuario. Debe ser un correo electrónico.

- Password: Contraseña del usuario para identificarse en la aplicación. Las contraseñas además no deben guardarse, si no que se guardarán los resultados de aplicar una función "Hash" a estas, para evitar que estas sean robadas. También se añadirá a todas las contraseñas un texto de 64 bits o "SALT" para evitar ataques de diccionario a las contraseñas-hash guardadas.

Relaciones:

- Validación: Un usuario no tiene ningún tipo de privilegio al registrarse en la aplicación. Para poder interactuar con la sección privada del sistema, este deberá ser validado por otro usuario diferente que contenga los privilegios necesarios para aceptarlo, además de pertenecer a la misma institución a la que el solicitante desea unirse. Un "validador" no puede validar ni ver peticiones dirigidas a instituciones diferentes a la suya.

- Rol: Un usuario además tendrá un rol. En el momento del registro, el usuario tendrá un rol de tipo "EMPTY" bloqueante (este rol no puede eliminarse ni modificarse, es un rol por defecto para todos los nuevos usuarios). Una vez un usuario con privilegios asigne al nuevo usuario un nuevo rol, de este rol se podrá acceder a los nuevos privilegios. Un usuario no podrá tener un rol de una institución diferente a la suya.

- Publicación: Son las publicaciones en redes sociales que realiza el usuario. Aunque se almacenen en las redes sociales, se guardará un registro de los envíos, para discernir qué usuario lo ha realizado, algo imposible desde las propias redes sociales, que solo apuntan a la aplicación desde la que se ha realizado.

2.7.2- Rol y Privilegio

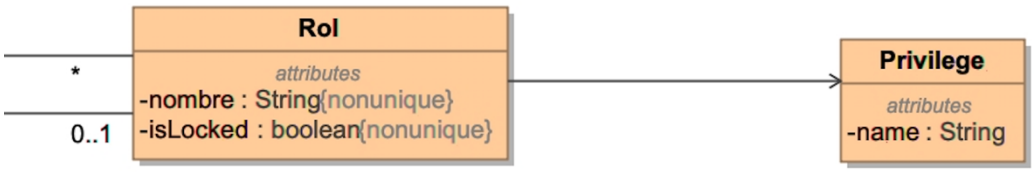

Ilustración 9: Rol y privilegio

Un rol estará constituido un conjunto de privilegios creado por una institución, con el fin de que esta puede mantener una organización dentro de la aplicación, lo más parecida posible a su organización real.

Atributos:

- Nombre: Nombre del rol designado por la institución.

- Bloqueo: Valor booleano que indica si el rol está bloqueado o no. En caso de que un rol esté bloqueado, este no se podrá borrar ni modificar por ningún usuario de la base de datos. Los roles bloqueados tampoco pueden crearse, para que una institución pueda tener un rol bloqueado, debe comunicarse al administrador de la base de datos. Esto se debe a que no está recomendado crear este tipo roles en la aplicación por motivos de seguridad. El único rol bloqueado existente en la aplicación es el rol "Empty" autogenerado al iniciar por primera vez la aplicación. Esto se debe a que es un rol sin ningún tipo de privilegio, que debe estar siempre presente y debe ser asignable a un usuario en cualquier momento. A este rol se le ha dotado de esta característica por motivos de seguridad. Ningún usuario podrá nunca tener ningún permiso al registrarse en la aplicación.

Relaciones:

- Usuario: Todos los usuarios tienen un rol. Un rol define los privilegios del usuario. El rol de un usuario debe ser de la misma institución que este.

- Institución: Institución que ha creado el Rol.

- Privilegios: Privilegios asociados al rol. Cada privilegio posee un nombre que actúa como identificador único y a su vez, tiene asociado un recurso o conjunto de recursos sobre el que permite interactuar.

2.7.3- Contenido

Protocolo

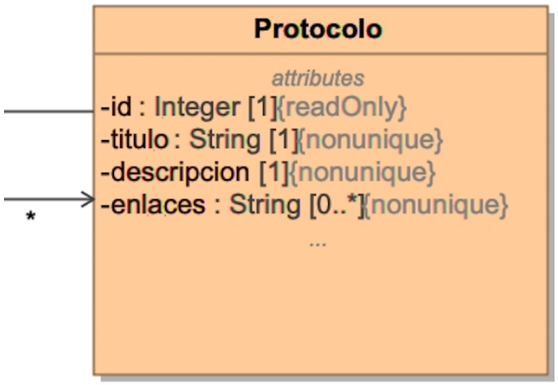

Ilustración 10: Protocolo

Un protocolo es la unidad básica de información. Contiene información sobre un procedimiento concreto a una situación de emergencia. Toda la información se guardará en formato "Markdown" por su sencillez y flexibilidad frente a otros formatos. Además, Markdown permite el uso de formatos adicionales como LaTeX. Cabe destacar que la inserción de código HTML, permitida en markdown en este caso esta bloqueada, con el fin de evitar ataques de tipo "Cross-syte scripting".

Atributos:

- Id: Identificador único del protocolo. Se utiliza además para los enrutamientos desde códigos QR, por lo que en la base de datos debe indexarse respecto a este atributo para mejorar la eficiencia.

- Título: Título o resumen del protocolo. Identificador visual para el usuario.

- Descripción: Contenido del protocolo en formato "markdown". Es la base de la información del protocolo.

- Enlaces: Enlaces de interés añadidos.

- Media: Contenido multimedia añadido. Imágenes, videos etc.

Emergencia

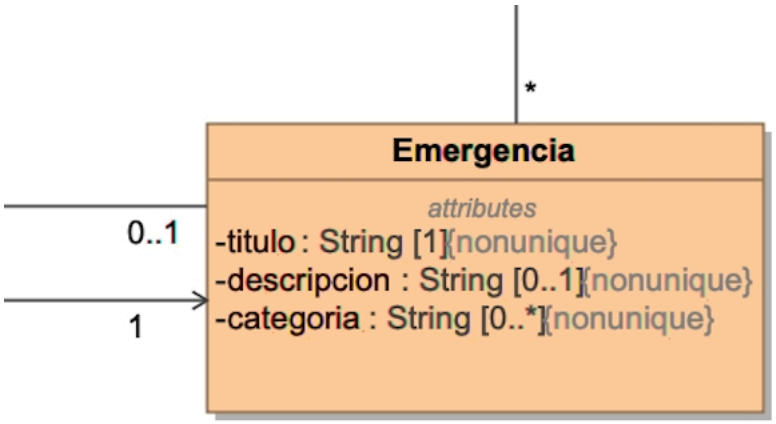

Ilustración 11: Emergencia

Una emergencia son un conjunto de protocolos, frente a los cuales es útil poseer cierta información de actuación. Cabe destacar que una emergencia no es un suceso concreto, sino general. Su objetivo es almacenar toda la información de este tipo de suceso para no tener que ser redactado cada vez que esta ocurra.

Al igual que los protocolos, las emergencias estarán escritas utilizando el formato "Markdown" por los motivos descritos anteriormente.

Atributos:

- Id: Identificador único de la emergencia. Se utiliza para los enrutamientos desde códigos QR, por lo que en la base de datos debe indexarse respecto a este atributo para mejorar la eficiencia.

- Título: Título o resumen de la emergencia. Identificador visual para el usuario.

- Descripción: Contenido de la emergencia. En formato "Markdown". Es una descripción de la emergencia o posible contenido adicional necesario que no esté descrito en los protocolos por su carácter general.

Relaciones:

- Protocolo: Protocolos útiles para la determinada emergencia.

Alerta

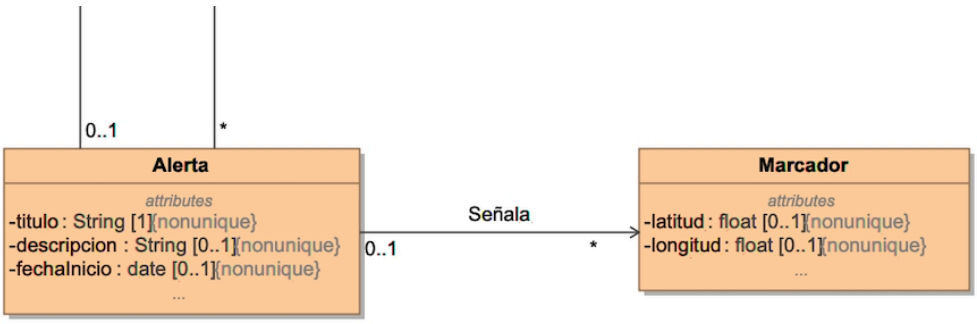

Ilustración 12: Alerta y marcador

Una alerta es el suceso o instancia de una emergencia en un momento concreto y con la posibilidad también de suceder en un lugar concreto. Las alertas irán acompañadas de marcadores, en caso de así requerirlo, para informar de los lugares vulnerables.

Una alerta únicamente tiene una descripción que aporta detalles adicionales sobre la misma y una única emergencia asociada, a través de la cual se podrá navegar para obtener la información necesaria para afrontar la situación. Una alerta además puede

ser referenciada por una publicación, al igual que las emergencias y protocolos, Aunque en este caso, la publicación no se utilizará como aporte de información casual sino como prevención inmediata y de alarma.

2.7.4- Redes Sociales

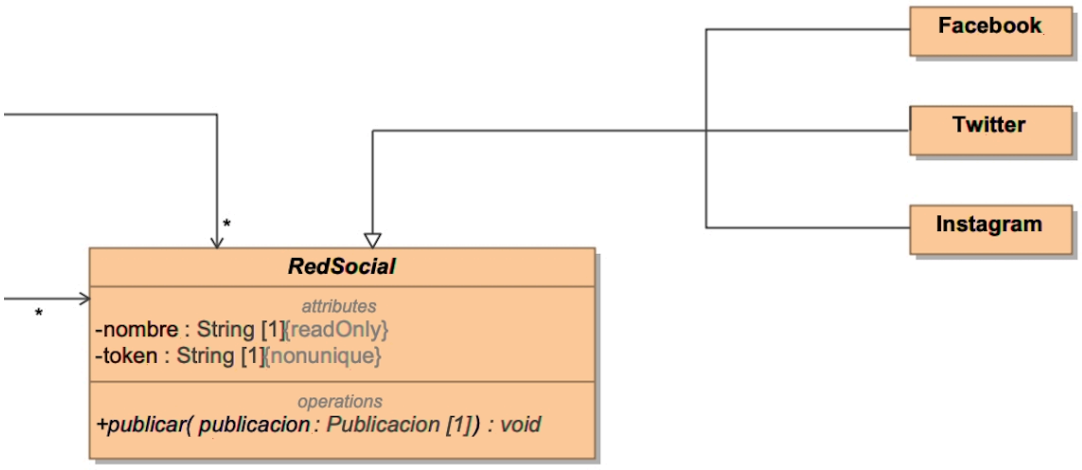

Ilustración 13: Redes sociales

El principal obstáculo para las redes sociales es que estas tienen APIs mutables y actualizables además de que pueden cambiar a lo largo del tiempo. Es por esto por lo que la información de las redes sociales seguirá un patrón ”Método de la plantilla” en la que se definen los algoritmos necesarios en la subclase y las subclases que lo implementan. De esta forma, tendremos un método de interacción con el API de cada red social fácilmente actualizable y adaptado a las necesidades de cada uno.

Cada objeto que contiene información de una red social estará asignado a una única institución, es decir, cada institución tendrá que solicitar su propio token de acceso a las redes sociales. De esta forma aumentamos la seguridad entre las diferentes instituciones.

3- Implementación del servidor

Puesto que esta aplicación es un software hecho para instituciones públicas, con información sensible y crítica, el principal aspecto a tener en cuenta a la hora del desarrollo del lado del servidor es la seguridad e integridad de la información del sistema.

Por otro lado, es de especial importancia la eficiencia en todo a lo que se refiere la información pública, por lo que en función del ámbito (público o privado) la aplicación se centra más en al menos uno de los dos aspectos.

En el lado del servidor destacan dos aplicaciones y dos sistemas de persistencia independientes, aunque pueden alojarse en el mismo servidor, aunque esto último no esta aconsejado, ya que como se ha dicho anteriormente, estas aplicaciones se han separado para agilizar los procesos que realizan cada uno.

La aplicación principal es el API REST con toda la información del sistema, la otra aplicación es una variante de una aplicación "Kumo" adaptada a este sistema.

Por otro lado, cada aplicación utiliza un sistema de persistencia diferente: el API principal utilizará una base de datos de tipo MongoDB, mientras que el servidor Kumo únicamente necesitará un entorno Node.js sobre el que ejecutarse y del sistema de persistencia de ficheros básicos proporcionado por el sistema operativo. Aún así, puede utilizar también su propia base de datos de Mongo DB almacenando en "chunks" o bloques de información las imágenes y archivos.

3.1- El API REST o aplicación principal en el lado del servidor

3.1.1- Estilos de programación y arquitectura

La programación del servidor se ha seguido conforme a algunos estilos básicos y patrones principalmente inspirados en los principios del libro "Clean Code[53]" como el uso de variables descriptivas en lugar de comentarios o el uso de "Setters" y "Getters" en lugar de accesos directos a los atributos de los modelos o clases.

El API se ha realizado de la forma más adaptable y escalable posible para la combinación o acoplamiento a otros sistemas de las instituciones públicas a las que va dirigida. Además, se tiene en cuenta que es posible que se haga una migración inicial de datos a la plataforma desde otro sistema o que incluso se utilice esta información para otros sistemas o aplicaciones.

[53] Clean Code (2008). Robert C. Martin.

La arquitectura seguida es una Hexagonal o DDD, en la que se define un micro-servicio por cada recurso en función de su tipo de acceso.

Dentro de cada servicio se observan tres niveles (y en el caso del proyecto tres directorios), los niveles de infraestructura, aplicación y dominio.

Cada nivel tiene acceso a su nivel inferior y superior directamente, aunque un nivel no puede acceder a un nivel no adyacente.

El nivel de dominio pertenece principalmente a las acciones más concretas. Esto puede ser como “registrar usuario”. Puede verse cada una de las rutinas de este nivel como una acción de un caso de uso o una acción atómica. Este es el nivel más alto de la estructura.

El nivel directamente inferior es el de aplicación, son los servicios a los que se accede directamente desde el API. Puede verse por tanto este nivel como el conjunto u orquestación del nivel de dominio, en el que una serie de acciones son llevadas a cabo consecutivamente. Un ejemplo podría ser la creación de un nuevo usuario con la previa comprobación de que el usuario no esta ya registrado en la base de datos.

El nivel inferior es el de infraestructura. Este nivel es el más acoplado a la tecnología que se está utilizando, en este caso el framework "Spring Boot”. El nivel de infraestructura tiene clases a utilizar por el nivel de aplicación como repositorios para acceder a bases de datos o configuraciones.

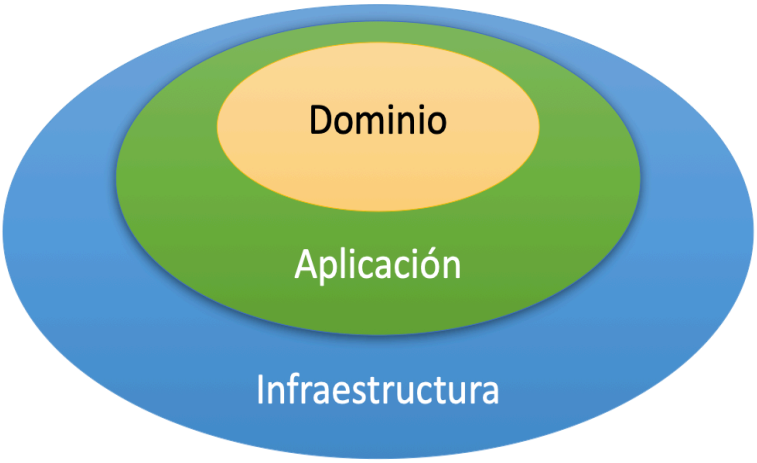

Ilustración 14: Arquitectura hexagonal

El principal motivo de la elección de esta arquitectura es la posible escalabilidad del sistema. En otras arquitecturas o modelos mas convencionales, se realiza una agrupación de directorios en función del tipo de ficheros (directorios para modelos, controladores, repositorios, etc.). El problema de estas arquitecturas es que al haber un incremento de estos ficheros, la navegabilidad y el mantenimiento de los recursos se hace más difícil.

Algunos patrones destacables dentro del proyecto son el constructor o patrón "builder", utilizado para generar las respuestas del API. Por defecto se ha seguido la especificación "JSEND[54]" por su simplicidad. Tambien se aporta un constructor para la especificación de "{JSON:API} [55]. Posibilitando con esto, añadir de forma sencilla nuevos constructores para adaptar fácilmente el formato de las respuestas del API.

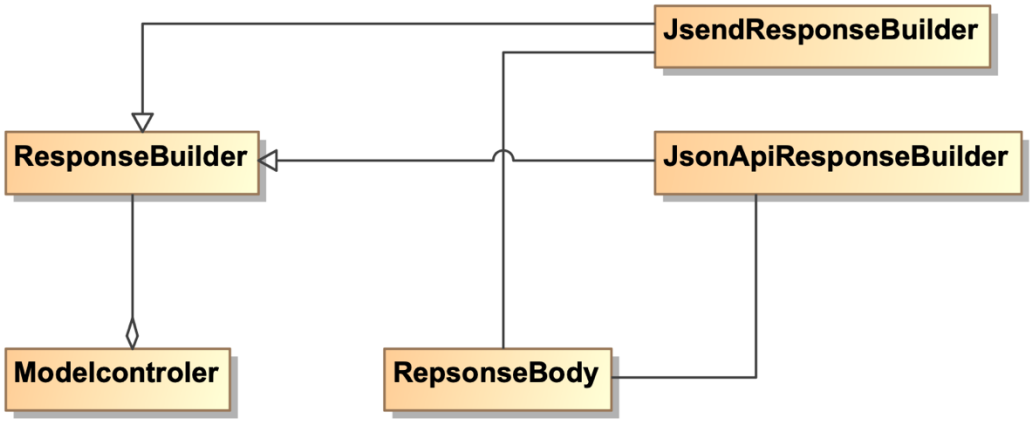

Ilustración 15: Constructor de respuestas del API

Otros componentes y patrón importante son los repositorios, que tienen acceso a la base de datos directamente y están presentes en ellos todas las consultas posibles a cada una de las clases. Todos los accesos a la base de datos se harán a través de estos. Como puede observarse, la función de estos repositorios no es mas que el de actuar de factoría para los modelos en el sistema, con una interfaz básica de consultas a la base de datos (crear entidades, eliminarlas, editarlas, etc.)

Los modelos son los productos generados por las factorías o repositorios. Representan los documentos de la base de datos no relacional y proporcionan métodos útiles como validación de atributos. Para mejorar la eficiencia de la aplicación, aunque las características de los atributos están presentes en forma de etiquetas interpretables para la base de datos (atributos únicos, requeridos, referencias a otros elementos, etc.) estos se comprobarán antes de ser ingresados a la base de datos con el fin de evitar cualquier tipo de transacción innecesaria.

De hecho, todos los modelos heredan de la clase "Model" que les obligará a ejecutar un método "isValid()" (Ilustración 16) para comprobar que todas las etiquetas de los atributos se cumplen.

[54] Jsend Documentation, https://github.com/omniti-labs/jsend
[55] Json:api Specification, https://jsonapi.org

```java
package dev.kabi404.ppce.models.mongo;

public abstract class Model {

    public static final boolean CHECK_ID = true;
    public static final boolean IGNORE_ID = false;

    public boolean isValid(boolean checkId) {
        return !checkId || getId() != null;
    }

    public abstract void update(Model model);

    public abstract void patch(Model model);

    public abstract String getId();

}
```

Ilustración 16: Superclase modelo[56]

Por último, están los controladores, de los que se generan los microservicios del API.

Estos controladores estarán ordenados en función del recurso al que gestionan y tendrán unos parámetros de entrada, una ruta y una serie de repositorios con los que interactuar.

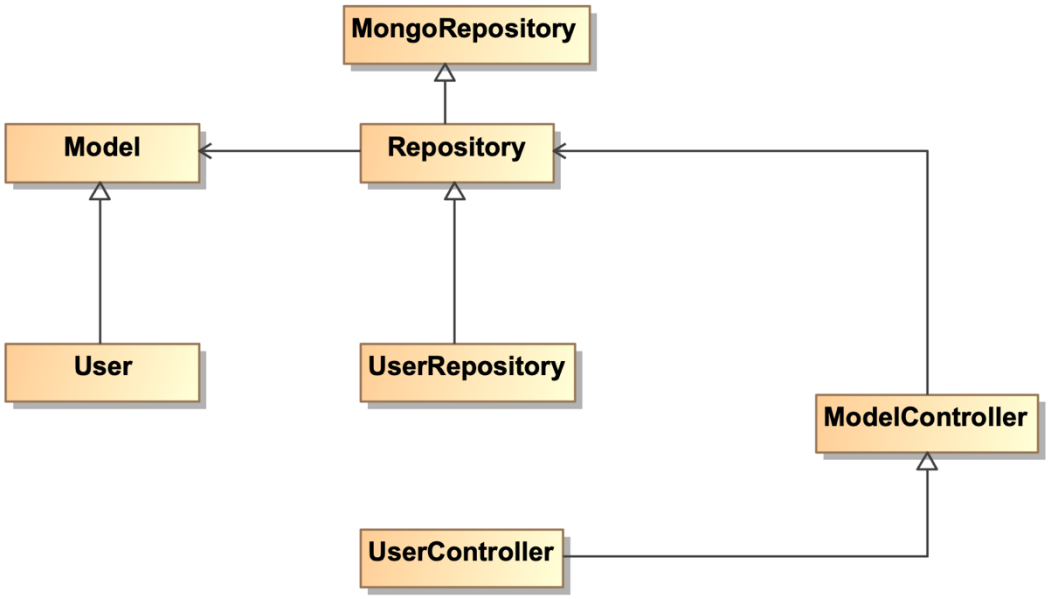

Ilustración 17: Esquema aplicación del lado del servidor

[56] Imagen generada mediante Carbon, https://carbon.now.sh

3.1.2- Seguridad y Sesiones con JSON Web Token

La seguridad del API se construye sobre un sistema de JWT (JSON Web Token), en los que no se almacenan sesiones de usuario en memoria, sino que todos los datos necesarios para poder mantener la sesión se almacenarán en un Token de sesión, enviado al usuario continuamente y devuelto por este en cada operación que se desee realizar.

El contenido principal de los tokens de sesión de la aplicación es: El usuario registrado, una fecha de expiración y una fecha de emisión.

Por motivos de seguridad, del usuario se utilizará únicamente su ID. Cada interacción con el área privada requiere cargar de nuevo el usuario completo junto a sus roles para comprobar sus privilegios y el estado de usuario en cada acción. De esta forma, su rol y estado no esta ligado a la sesión, sino que se modifica y actualiza en tiempo real.

Para asegurar la integridad del token de sesión se utilizará criptografía asimétrica, concretamente se utilizará el algoritmo RSA con claves de 4096 bits (las claves más seguras dentro de este algoritmo).
Aunque es cierto que hay otros algoritmos más eficientes o incluso la posibilidad de utilizar criptografía simétrica, en este caso prima la seguridad de utilizar claves que no sean sujetas a ataques. Además, si se hace un balance, el verdadero grueso del tráfico de datos se encuentra en el área pública disponible para la población, mientras que las responsabilidades delegadas en los administradores de la plataforma pueden cubrirse con unos pocos usuarios e interacciones.

Por defecto la aplicación genera unas nuevas claves asimétricas cada vez que se inicializa, esta configuración es la mas segura ya que pueden actualizarse las claves reiniciando simplemente el servidor. Pese a esto, pueden introducirse claves propias de las instituciones.

La clave asimétrica se utilizará para firmar el token siguiendo el RFC 7515[57], por lo que no se incluyen datos sensibles del usuario por si pudiese ser robado de alguna manera. Además, también se permite la posibilidad de cambiar el tiempo de caducidad de la sesión.

[57] RFC7515, https://tools.ietf.org/html/rfc7515

3.1.3- Seguridad y Validación de Privilegios

La validación de privilegios se realiza previamente a ejecutar el código del controlador. Por defecto, cada método está definido en los controladores como un servicio del API y tiene asociado una ruta. Cada ruta es única e identificadora de ese servicio, definiendo además de privilegio necesario para poder ser ejecutada.

La primera parte de la ruta define el privilegio del recurso al que se intenta acceder.

Si se requiriese validar dos privilegios de una ruta, simplemente se añadirían los dos directorios (sin importar el orden en que estos se pongan en la ruta). Lo privilegios principales son:

- USER_ADMIN: Es el privilegio que permite a un usuario gestionar a otros usuarios. Le permite concretamente, aceptar nuevos registros, suspender o bloquear cuentas temporalmente, reanudar cuentas suspendidas y cambiar los roles de los usuarios. Este privilegio es el mas crítico pues es el único que tiene acceso al resto de privilegios. El directorio global asociado a este privilegio es “user”.

- SOCIAL_MEDIA: Es el privilegio enfocado al control de las redes sociales. Este privilegio permite publicar contenido en las redes sociales y publicar en estas referencias de alertas, emergencias y/o protocolos. El directorio global asociado a este privilegio es “social”.

- CONTENT_MANAGER: Es el privilegio asociado a la redacción de contenido en la plataforma, de emergencias y protocolos. Además, el rol con este privilegio puede solicitar un token para la interacción con el servidor Kumo y añadir nuevas imágenes al sistema. El directorio global asociado a este privilegio es “content”.

- ALERT_MANAGER: Aunque las alertas son consideradas como contenido dentro de la aplicación, se ha decidido separar por la especial criticidad de este tipo de publicaciones. Las alertas son especialmente críticas por poder causar un fuerte impacto sobre la población, por lo que poseen su propio privilegio. Cabe destacar que además, aunque se puedan lanzar alertas, si no se posee el privilegio de gestión de redes sociales, estas alertas tampoco pueden publicarse fuera de la plataforma. De esta forma, se deja aún más capacidad de autogestión sobre las instituciones, ya que la gestión de las alertas podrá pasar por uno o dos filtros. El directorio global de este privilegio es “alert”.

- ROLE_ADMIN: Por motivos también de seguridad se ha decidido separar la gestión de roles de la creación de usuarios, ya que así podemos evitar la

creación de un “superusuario” por parte del administrador de usuarios. Se recomienda que el mismo usuario no posea la capacidad de crear nuevos roles y poder asignarlos. Este privilegio consiste únicamente en la elaboración de nuevos roles como agrupaciones de privilegios. El directorio global del privilegio es “role”.

- ANY: Este privilegio no está presente en la base de datos. Consiste en un privilegio únicamente para aquellos usuarios que están validados y activos dentro de la plataforma. Dentro de las configuraciones de seguridad, el privilegio “ANY” se añade en cada comprobación de los privilegios, siempre y cuando el usuario tenga al menos un privilegio. De esta forma tenemos un privilegio común para todos los usuarios registrados con un estado activo dentro de la plataforma. El motivo de este privilegio no es más que el de simplificar y acortar las URLs de servicios comunes a todos los privilegios.

- Métodos sin validación: Finalmente, los métodos públicos de la aplicación que no requieren de un campo de autenticación en ninguna cabecera de la llamada HTTP pertenecen al directorio global “pub”. Este campo omite la fase de validación completa para mejorar en la mayor medida posible la eficiencia.

Ni los roles, ni el usuario, ni la comprobación de estos forma parte del código presente en los controladores. Todas las validaciones se realizan previo a la ejecución de estos, con el fin de simplificar en la medida de lo posible el código. En caso de que esta información fuera necesaria, simplemente se puede acceder a ella a través de la información proporcionada y previamente validada en la llamada HTTP.

Por último, los privilegios se generan automáticamente al iniciar la aplicación por primera vez en la base de datos. Los privilegios, a diferencia de los roles, no se pueden crear ni eliminar. En caso de ser borrados manualmente de la base de datos se volverán a generar al iniciar la aplicación. Si en el futuro se añaden nuevos privilegios para nuevas funcionalidades no contempladas en esta versión deben añadirse al código fuente en la función de inicialización y en el proveedor de seguridad en el directorio de configuraciones con sus correspondientes directorios globales asociados.

3.1.4- Seguridad e Integración con Kumo

Como ya se indicaba anteriormente, la integración con el servidor Kumo se realizará mediante una versión simplificada del protocolo “Oauth 2.0” de Google para implementar el inicio de sesión en aplicaciones de terceros. Este protocolo consiste en la generación de un token de sesión, con un email del usuario que intenta acceder al

servicio, sus roles, una fecha de expiración, una fecha de emisión y la firma de la plataforma[58].

Las claves a partir de las cuales se origina este token son compartidas inicialmente tanto por el servidor de la plataforma como por el servidor Kumo. De esta forma, podremos validar al usuario en el servidor Kumo, de manera independiente a nuestra aplicación.

La generación del token, debido a la baja criticidad en comparación con la información textual de la aplicación principal, se hará mediante algoritmos basados en criptografía más eficiente respecto de la aplicación Spring, específicamente con el algoritmo HS256, un HMAC sobre el algoritmo SHA-256 con claves de 256 bits.

El motivo de utilizar una aplicación aparte del servidor principal es la adaptabilidad a esta función. Al usar el formato Markdown el número de imágenes puede ser ilimitado así que se hace de gran importancia tener un buen rendimiento a la hora de la carga de imagenes.

A diferencia de aplicaciones REST convencionales, Node.js es una tecnología no bloqueante, es decir, no contiene bloqueos a la hora de acceder a la base de datos o sistema de persistencia en la entrada y salida. La ventaja de esto es que, se hacen lecturas múltiples a la base de datos simultáneamente, sin tener bloqueados procesos por cada acceso al API, es decir, por cada imagen.

Además Node.js se ejecuta en una sola hebra de ejecución, es decir, no se abren nuevos procesos por cada llamada al API como en el servidor principal. Esto ahorra un tiempo que en grandes escalas puede volverse significativo, a la hora de hacer cambios de contexto por cada proceso que esta accediendo a la plataforma[59].

Este sistema es, por tanto, el encargado de resolver estas llamadas que únicamente requieren de entrada y salida y un rendimiento notable, y no tanto de requisitos como integridad o seguridad.

3.1.5- Seguridad y confidencialidad de la información

Respecto a la confidencialidad de información crítica tales como contraseñas y tokens utilizados para las redes sociales, estos no se guardarán en el sistema de persistencia en texto plano, sino que, en el caso de las contraseñas se les aplicará una función Hash, concretamente con el algoritmo “bcrypt” para evitar posibles robos de información. Además, se le añadirá a todas las contraseñas una “SALT” no almacenado en el sistema

[58] Google LLC, Google Identity Platform, Oauth de google, https://developers.google.com/identity/protocols/oauth2

[59] OpenJS Foundation, Sobre el proyecto Node.js, https://nodejs.org/en/about/

de persistencia para evitar posibles ataques de diccionario a estas. Una SALT es un conjunto de bits aleatorios añadidos al texto a cifrar con el fin de evitar ataques de diccionario sobre los datos.

Para validar a un usuario simplemente, se añade la "SALT" a la contraseña enviada por el usuario y se aplica la correspondiente función Hash. Una vez hecho esto se comparan las contraseñas filtradas para comprobar la autenticación.

3.1.6- Integración con Redes Sociales

La interacción con las redes sociales se hará desde el lado del servidor, esto es así por motivos de seguridad, para evitar principalmente que el token pueda ser robado. Se crea una clase con su controlador correspondiente para cada red social. Cada red social además tiene su propio documento en la base de datos, ya que los requisitos necesarios para cada uno pueden ser diferente. Por ejemplo, para el caso de Twitter se necesitan hasta cuatro claves diferentes, ya que este API funciona bajo el protocolo Oauth 1.0, mientras que el API de Facebook sencillamente utiliza un token de aplicación y otro de identificación del usuario, página o grupo.

Inicialmente la comunicación de las redes sociales añadidas para la primera versión, se utilizan clases Java predefinidas únicas y exclusivas para estas redes sociales. Sin embargo, también se proporciona una librería para realizar peticiones HTTP genéricas con la que posteriormente pueden incluirse nuevos APIs de redes sociales no presentes en la actualidad.

Por último, no se utiliza un único token, sino que cada institución tendrá que solicitar el suyo propio. Esto se debe principalmente a que muchas redes sociales (en este caso Twitter, por ejemplo) tienen asociado un token a un usuario concreto con el que permiten interactuar. Además, por motivos de seguridad y configuración, es más sencillo para las instituciones tener sus propios tokens con sus métricas, permisos y estadísticas propias.

3.2- La aplicación de almacenamiento multimedia: Kumo

Kumo es una aplicación de código abierto realizada por el autor de este TFG, en JavaScript sobre Node.js como sistema de almacenamiento de archivos multimedia. Esta aplicación consiste en una sencilla API-REST para mandar archivos multimedia en formatos como como .png, .gif etc. Mediante un proceso de autenticación por JSON Web Token como ya se ha explicado anteriormente.

La aplicación Kumo además dispone de un área pública en el API-REST, para la lectura de imágenes. La lectura de imágenes se realiza únicamente sobre un directorio público. De esta forma se simplifica la lectura de estos archivos, haciendo referencias por URL mediante etiquetas convencionales de imágenes en HTML.

Como se ha dicho antes, el servidor Kumo puede alojarse en otro servidor diferente de la aplicación principal para aumentar la eficiencia, ya que las imágenes se cargan de forma asíncrona, respecto del resto de elementos de la aplicación, por lo que es independiente de las comunicaciones realizadas con la aplicación principal, evitando así cuellos de botella y saturación de la red.

También, como ya se ha indicado antes, la conexión con el servidor Kumo se realiza desde la aplicación en el lado del cliente, cediendo únicamente al API principal la tarea de la generación del token para la validación de usuarios en caso de requerir de permisos de escritura.

Para la integración con el proyecto únicamente se han añadido las claves compartidas con el API principal y un proceso de validación con clases y documentos utilizados para los usuarios.

4- Implementación del Cliente – Aplicación Vue

4.1- Arquitectura y estilos de programación

4.1.1- Vue Router y la gestión de rutas

Al igual que la aplicación del lado del servidor, esta aplicación constará de dos partes principales: La parte privada y la parte pública. La arquitectura de ambas es diferente debido a que la parte privada debe estar adaptada para que su código pueda ser reutilizado para la aplicación de escritorio en "Electron.js".

La arquitectura del apartado público se há diseñado con diferentes rutas. Para facilitar esto se utiliza la librería "Vue Router", que nos permite gestionar los componentes de vistas de cada ruta de forma dinámica y perezosa. Las vistas formarán una rejilla o "Grid" sobre la cual se ubicarán los componentes; la lógica real de la aplicación estará, por lo tanto, en los componentes que forman las vistas. Inicialmente en el directorio raíz del proyecto encontramos el componente de renderizado inicial, "main.js". En este componente se realizan las importaciones de librerías globales como "Vue Router" y otras muchas utilizadas. Dentro del fichero de este mismo directorio "Router.js" se encuentran las rutas y parámetros utilizados en la aplicación junto a su correspondiente componente.

Por otro lado, la gestión de las rutas en la sección privada se realiza mediante una variable controladora, que renderiza al igual que Vue Router, de forma dinámica y perezosa, la correspondiente vista. Esta vista estará a su vez encapsulada en la vista real de la ruta "/private" donde estarán el resto de vistas. Esto es debido a que la aplicación de Electron será un SPA (*Single Page Application*) o aplicación de una única página. Por lo que el cambio de rutas podría dar problemas en la aplicación en local.

4.1.2- Vuex y la gestión de memoria

Se puede observar cómo, además, se importa una librería de mayor extensión llamada Vuex, que se encargará de separar la información contenido (protocolos, emergencias, usuarios, etc.) de las interfaces (como variables encargadas de gestionar la visibilidad de componentes, tamaños, etc.).

Vuex simplifica la gestión de memoria y el control de interfaces con un patrón "Redux". De esta forma, toda la información de contenido esta centralizada y accesible por todos los componentes.

Finalmente, la información de interfaces queda guardada en las variables del propio componente que las utiliza, mientras que en caso de necesitar información o contenido de la plataforma, se debe pasar por Vuex. Tanto la información local como las llamadas al API mediante HTTP son gestionadas por Vuex; no se realizan directamente por el componente.

Para acceder al contenido guardado en Vuex se llamará a los "getters" que utilizan funciones síncronas en JavaScript. Sin embargo, para cargar o enviar datos del API, se utilizan siempre las "actions", que son promesas asíncronas que actualizarán el estado de la aplicación.

El flujo, por tanto, del acceso a la información será el siguiente:

1- Mediante un "getter" a Vuex (acceso directo al estado de la aplicación) se comprueba si el protocolo deseado ya está en memoria. En caso de estarlo, se renderiza automáticamente, si no se muestra un campo vacío. Simultáneamente a este paso, el componente llama al "Action" para obtener el protocolo (se encuentre este en el estado de la aplicación o no).
2- El "Action" de Vuex comprobará de forma síncrona si el protocolo está en el estado de la aplicación. Si está presente, aquí termina el flujo. Si no lo está, realizará una petición al API para obtenerlo de forma asíncrona.
3- Una vez el "action" obtiene el protocolo, cambia el estado de la aplicación mediante una mutación.
4- El componente detecta automáticamente un cambio en el estado de la aplicación, dentro de las variables que este utiliza. Detecta el protocolo antes ausente, y lo renderiza automáticamente.

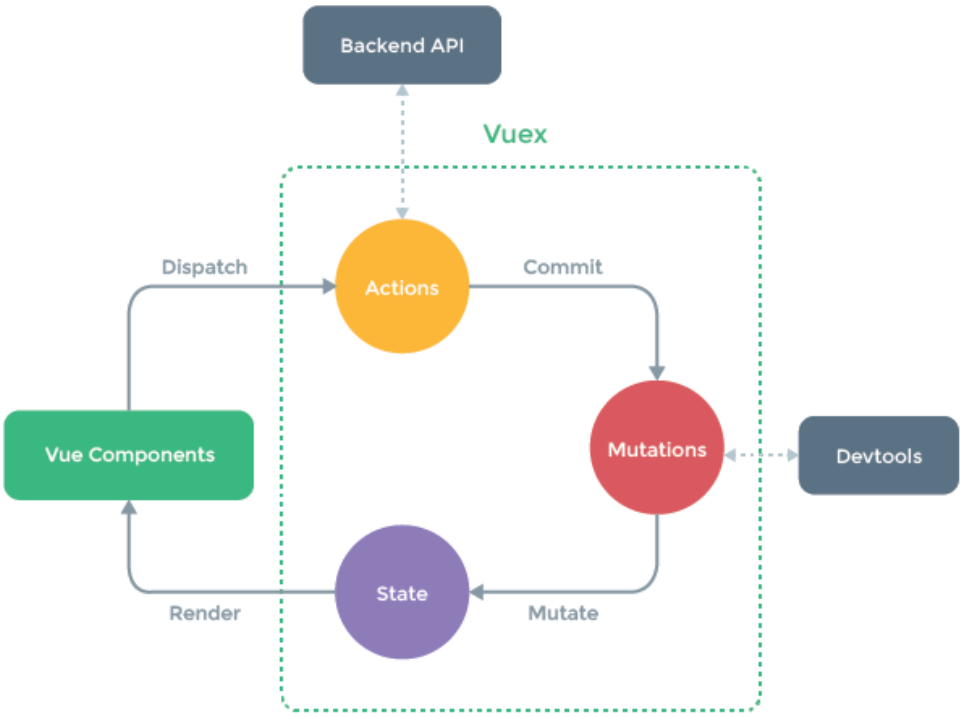

Ilustración 18: Esquema gestión de memoria en Vuex[60]

4.1.3- Gestión de sesiones con JSON Web Token

También se utilizan algunas funciones asíncronas en diversas partes de la aplicación hasta el final de su ejecución; es el caso de la actualización del Token de sesión, el cual, al iniciar sesión se actualizará en el estado de la aplicación.

Aunque es habitual que el token de sesión esté presente en el almacén de memoria del proceso de la aplicación, este no será el caso. En caso de recargar la ventana de nuevo la sesión se eliminará, borrando toda la memoria del sistema. Con esto evitamos posibles problemas en que usuarios dejen abierta la sesión saliendo de la página, pero no cerrando el navegador, por ejemplo.

Hay que tener en cuenta que esta parte de la aplicación esta hecha para ser utilizada (entre otras formas) por una aplicación de escritorio. En la aplicación de escritorio no se dispone de la sección publica ni de cambios de URL, por lo que la decisión de utilizar el token únicamente en la memoria dinámica no supone ningún problema.

[60] Vuex, Gestión de memoria en Vuex, https://vuex.vuejs.org/

4.1.2- Llamadas al API y factoría

Aunque la gestión de memoria y las llamadas asíncronas se encuentran en los componentes Vuex, se han separado los métodos de llamadas HTTP directas al API en forma de una "Facade" o patrón factoría.

Cada componente Vuex se encuentra agrupado en función del recurso que se controla (protocolos, alertas, etc.) y de igual forma se estructura en los módulos de llamadas al API. Estas funciones son siempre promesas, ya que las llamadas al API mediante la librería "Axios" son llamadas asíncronas, sin embargo, en las funciones que utilizan estas promesas, estas se solucionan mediante la sintaxis ECMAScript 6 "async/await", que permite utilizar y esperar estas llamadas como si se tratase de una llamada síncrona visualmente.

De esta forma el código queda más limpio y no se anidan las funciones de resolución. Además, si se cambiase el formato de los objetos o de las respuestas del API sería muy fácil de implementar la nueva configuración, cambiando la forma de los métodos de estas clases o configurando la variable global de la librería "Axios".

4.2- Modelado de componentes

4.2.1- Sección general y pública

Bajo la interacción de los componentes Vue y Vuex se encuentra un nivel de abstracción inferior en el que se ordenan los diferentes componentes por funcionalidad e interfaz.
Inicialmente, el manejador de rutas es el encargado de mostrar las vistas correspondientes. Un componente "VueRouter" es el que importa y maneja la vista renderizada de cada momento. Estos componentes se encuentran en el directorio "views" del proyecto.

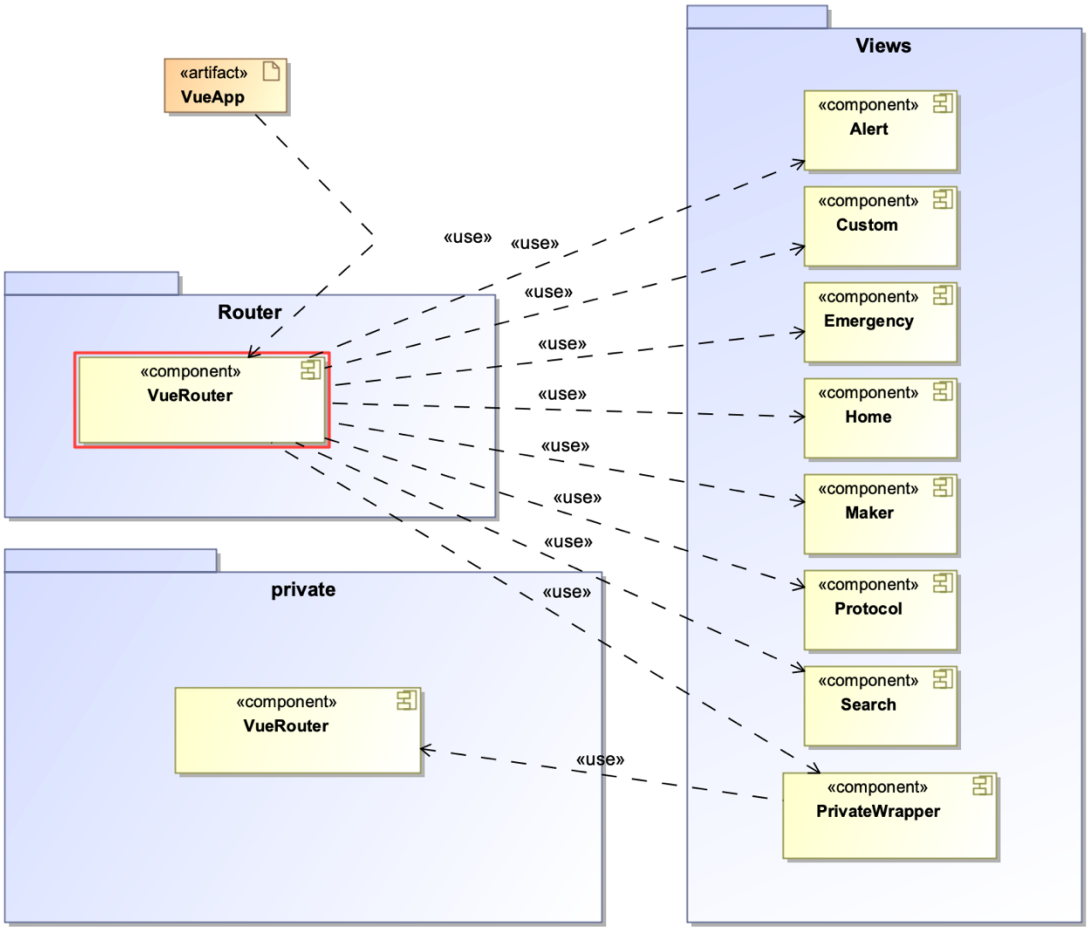

Ilustración 19: Vistas de la aplicación del cliente

Cada componente del paquete "views" representa una vista. Las diferentes vistas son:

- Alert: Muestra una alerta concreta con su información (emergencia) junto con un generador de QR, que se podrá ver en la siguiente ilustración. Almacenará

además la información de la alerta en la memoria dinámica de la aplicación para evitar hacer llamadas al API innecesarias.

- Protocol y Emergency: Al igual que la vista "Alert" muestra la información del protocolo o la emergencia actual. En caso de no estar cargada en memoria, esta realiza una carga al API. Si estuviese, simplemente se carga de manera dinámica y reactiva a través de Vuex. Las cargas se mantienen en memoria para poder navegar a través de la información sin realizar nuevas cargas al API.

- Maker: Es una vista con un buscador de contenido para añadirlos a un nuevo QR. Este componente construirá una URL con los identificadores del contenido seleccionado. Una vez generado esta URL y el correspondiente código QR permite a la población localizar físicamente la referencia a esta información y agruparla en función de las necesidades particulares de la zona. Este componente necesita de la vista "Custom" para visualizar los datos generados.

- Custom: Es la vista para los conjuntos de emergencias y protocolos agrupados por el generador de la vista "Maker". Analizará la información de identificadores presentes del URL y hará una consulta al API para obtener toda la información necesaria. Al igual que otros componentes centrados en la información, esta se guardará para poder ser consultada en cualquier momento en la memoria sin necesidad de hacer una nueva llamada al API al navegar a través de los distintos elementos.

- Home: Es una vista inicial con una presentación de la plataforma. La información se carga automáticamente de los ficheros de la aplicación (sin llamadas al API). Esta información es estática para mejorar en la medida de lo posible el posicionamiento de la página web.

- Search: Es una de las vistas principales. Consiste en un simple buscador para consultar la información de la plataforma a través de criterios de búsqueda. También permite consultar si se ha lanzado alguna alerta o las últimas enviadas.

- PrivateWrapper: Es un adaptador para la zona privada, ya que la arquitectura de es diferente para adaptarse a la aplicación de escritorio. Los componentes y vistas de la zona privada se rigen bajo la misma URL de este componente.

Cada una de las vistas utilizan componentes, alojados en la carpeta "components" del proyecto. Estos componentes proporcionan una utilidad modularizada, para poder ser empleados por otras vistas e incluso componentes anidados.

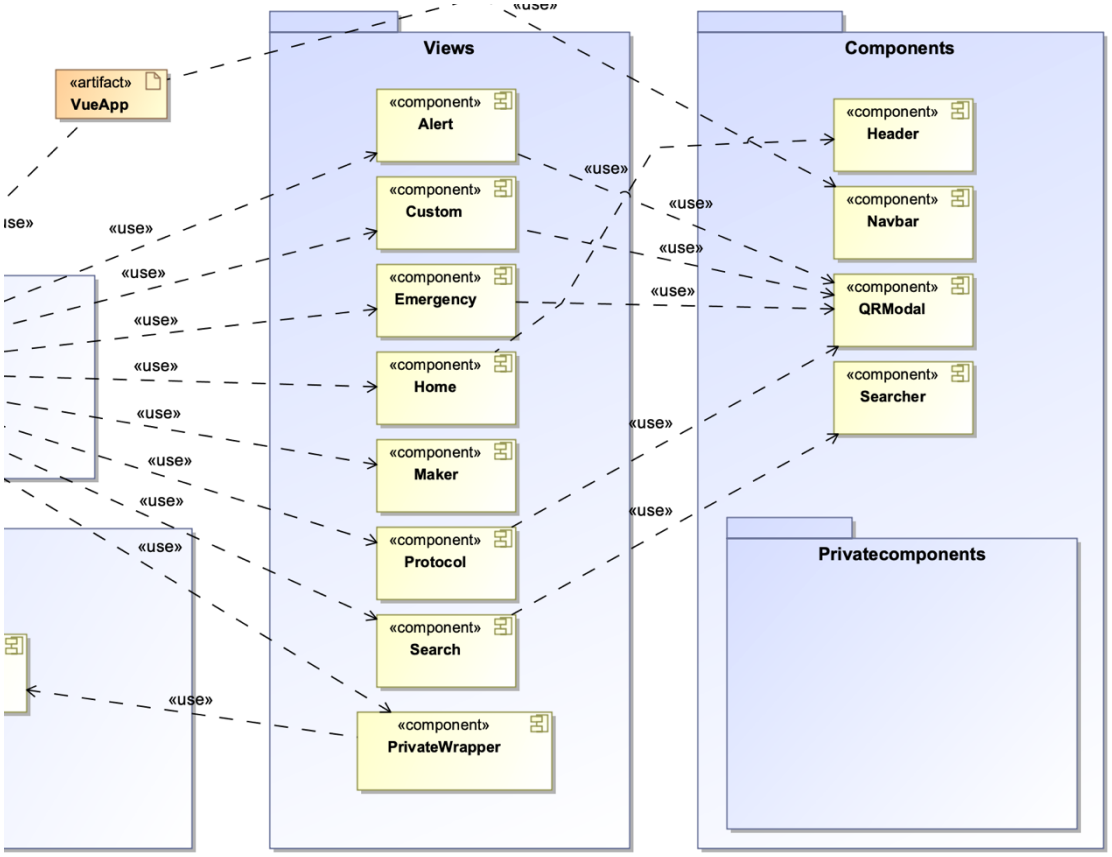

Ilustración 20: Componentes del lado del cliente

Los componentes utilizados por las vistas son:

- Header: Proporciona una cabecera (etiqueta en HTML) que se minimiza para aumentar la visibilidad de la información, al cambiar de sección desde el inicio a cualquier otra vista de la zona pública. Este componente solo cumple una función estética.

- Navbar: Barra de navegación, el principal objetivo de esta es la de cambiar la ruta fácilmente al seleccionar diferentes pestañas de la aplicación. Además reacciona automáticamente ante cualquier cambio de la ruta, mostrando siempre la pestaña actual sin necesidad de recargar la página. Incluye también botones para la generación de un enlace en formato QR a la página actual y la generación de archivos en formato PDF. Este componente no realiza la generación, sino que cada componente es el encargado de generar (en función de las necesidades) cada enlace y pdf. Por lo tanto, este componente solo lanzará un evento que indicará al componente renderizado qué debe generar el archivo indicado.

- QRModal: Este componente se encarga de generar un QR dado un enlace que debe pasarse por las propiedades de los componentes “Vue”. Este componente solo genera el QR a partir del enlace, no hace ningún tipo de comprobación de formato ni de los elementos incluidos. Este tipo de comprobaciones se hacen en cada componente que utiliza la modal.

- Searcher: Componente de búsqueda modularizado que permite buscar contenido en la plataforma. Es el más reutilizado de la plataforma, ya que tanto en la sección pública como privada se hacen búsquedas, ya sea para ver información, como para usarla, modificar o crear nueva. El buscador lanzará un evento con una referencia a cada selección del API. De esta forma, el componente que lo encapsula puede gestionar el o los componentes seleccionados de múltiples formas.

4.2.2- Sección Privada

La sección privada se encuentra limitada respecto a la pública en lo que a gestión de componentes se refiere. Para poder gestionar los componentes, estos se renderizan selectivamente por un adaptador. Este adaptador contiene una variable identificadora del componente actual a mostrar, sin embargo, esto no se refleja en las rutas que se mantienen sobre “/private”.

La sección privada no es accesible mediante interfaz a través de navegación simple. La dirección a esta debe hacerse de forma directa al URL. Así puede evitarse que el usuario medio se encuentre con esta sección accidentalmente, evitando aportar información y secciones innecesarias.

Como ya se ha dicho anteriormente, estas decisiones también se deben al deseo de compatibilizar el código de esta sección con la aplicación de escritorio.

Aunque la gestión de memoria se lleva a cabo en el adaptador de los componentes, la selección de pestañas o componentes se realiza en un sub-componente siempre visible que actúa de Barra de navegación (PrivateNavbar). Sin embargo, al no tener la posibilidad de gestionar rutas mediante Vue Router, estas se llevan a cabo mediante eventos, donde cada vez que se selecciona una pestaña, la barra de navegación lanza un evento a su componente padre (el adaptador) con la ruta (artificial) deseada. Este evento es tratado únicamente por el adaptador, que cambiará en función de este, el componente a mostrar.

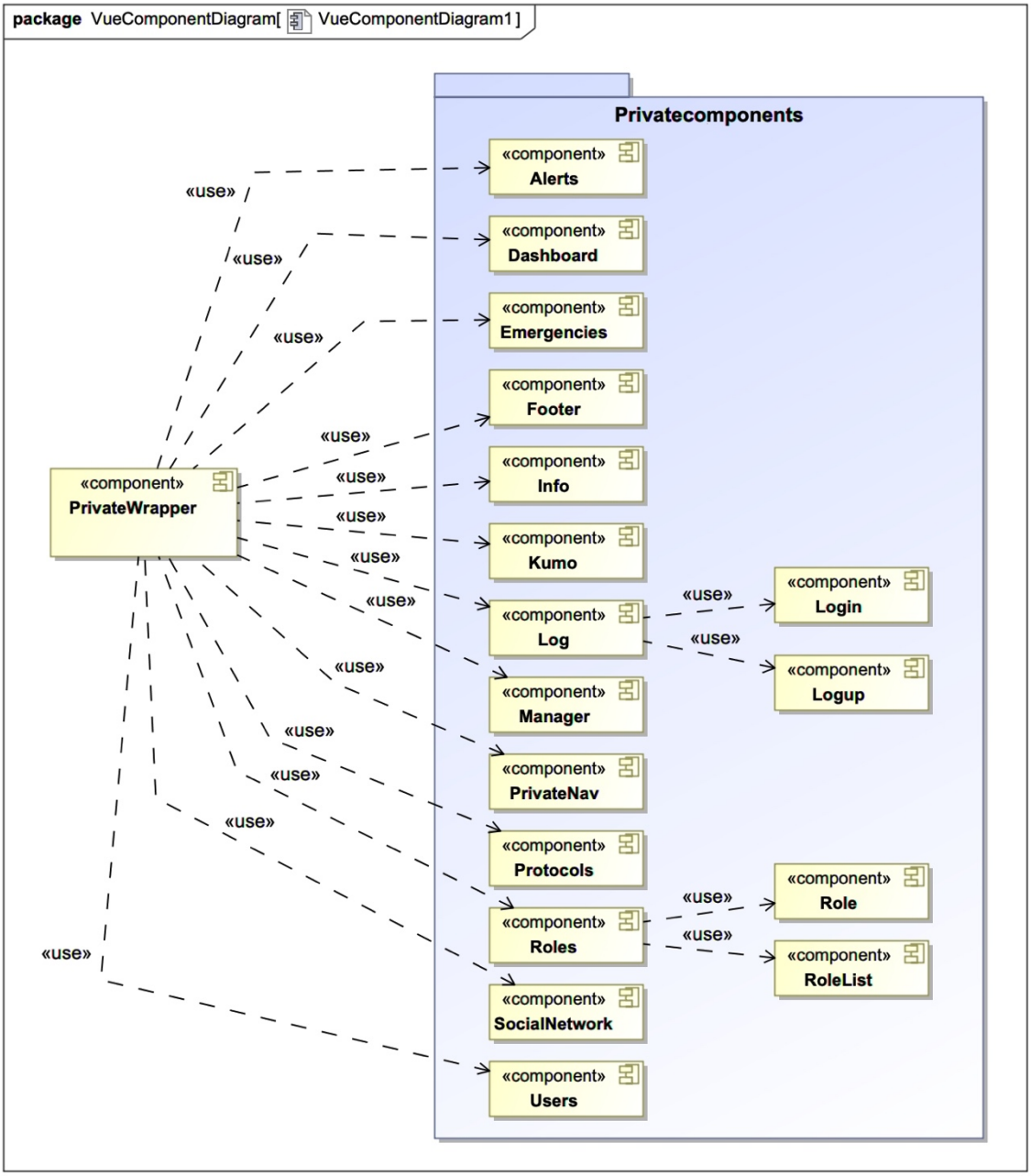

Ilustración 21: Componentes de la sección privada

En la imagen puede observarse como los componentes directamente utilizados por el adaptador son vistas y estos a su vez pueden estar conformados por sus propios componentes. Los componentes que forman la sección privada son:

- Alerts, Emergencies y Protocols: Son los gestores de contenido. Permiten la creación y actualización del contenido. Utilizan "Markdown" como formato principal y el componente "Searcher" para realizar búsquedas internas.

- PrivateNav y Footer: Son vistas que componen parcialmente la interfaz. Su objetivo es únicamente modularizar y añadir simplicidad a la navegación y mostrado de componentes.

- Kumo: Este componente tiene como único objetivo interactuar con el sistema Kumo para añadir nuevas imágenes que puedan ser añadidas a las publicaciones. Para estas operaciones se hará una nueva petición de un token de interacción con Kumo cada vez que se renderice o se haga una petición.

- Log: Es el componente que se encarga del inicio de sesión. Este componente se renderizará siempre que el token de sesión esté vacío o su actualización lance un error (que se maneja eliminando el último token de sesión almacenado). Dispone de dos subcomponentes para el inicio de sesión y el registro, controlados por una variable booleana.

- Roles: Es el componente encargado de gestionar los privilegios y roles de usuarios. Se divide en dos subcomponentes: uno para el mostrado de listas de roles de la institución a la que pertenece el usuario y otro para la creación o lectura de roles particulares.

- Social Network: Componente reutilizable asignado a una red social concreta. Este componente necesita un controlador que se le añade desde las propiedades del componente Vue. De esta forma, no utiliza un controlador para cada red social, puesto que la información es la misma. Se utiliza únicamente un controlador de envío que corrige el formato de cada red social y hace una petición al servicio del API correspondiente.

- DashBoard: Al igual que SocialNetwork, es un componente para el envío de información a las redes sociales. Este componente, sin embargo, admite más de un controlador, permitiendo el envío simultaneo de información a las redes sociales concretas seleccionada por el usuario. La ventaja frente al anterior componente es que permite una interfaz concreta y específica de cada red social, no limitando así el uso de cada una particularmente.

- Users: Es el componente encargado de la validación, eliminación y suspensión o bloqueo de usuarios.

4.3- Idiomas

La aplicación en la versión de entrega del proyecto se encuentra traducida a tres idiomas: español, inglés y japonés. La traducción se realiza al comenzar la aplicación, detectando el idioma del navegador y del usuario. La traducción se realiza a través un filtro definido globalmente en la aplicación Vue que, tomando un parámetro identificador de los datos a traducir, realiza una búsqueda por clave en un fichero que contiene todas las traducciones del lado del cliente.

Esta búsqueda no se hace en un diccionario como tal, sino en un objeto JavaScript con múltiples atributos. Cada atributo representa, mediante su nombre, el identificador de la traducción, y su contenido no es más que otro objeto, con atributos identificadores del idioma. Esta estructura se ha tomado por tres motivos principales:

1. Sencillez a la hora de introducir nuevos idiomas o traducciones: A diferencia de otras estructuras, cada elemento consta de sus traducciones en el mismo objeto, por lo que no requerimos de mas de un fichero para cada idioma, todas las cadenas de caracteres están situadas de forma consecutiva en el objeto.

Ilustración 22: Grupo de traducciones en JSON

2. Limpieza en los componentes: Los objetos en JavaScript internamente se implementan como listas desordenadas. Por lo que la búsqueda por parámetro, aunque menos eficiente que en una lista ordenada, funciona igual que una búsqueda en una estructura clave/valor y el código resultante en cada componente es de unos pocos caracteres por variable. De esta forma, se crean claves identificativas del propio contenido, sabiendo el contenido de este sin necesidad de acceder al propio fichero de idiomas.

Ilustración 23: Aplicación de las traducciones

3. Eficiencia: Puesto que la carga mediante el operador "require" se utiliza de forma perezosa y la asignación de cadenas se realiza una sola vez en la carga inicial. Únicamente se utilizarán los elementos que se requieren en el idioma que se ha seleccionado. Además, la mayor parte de las cadenas utilizadas se reutilizan en la mayor parte de los componentes o son sustituidas por iconos, por lo que el número de traducciones no es elevado.

4.4- Implementación de la Aplicación en Electron

Una vez compilada y optimizada la aplicación del lado del cliente, se añadirá el código fuente generado a una aplicación en Electron. Esta aplicación actuará a modo de servidor de código estático genérico para alojar una copia de la aplicación como si fuese un servidor más. Mientras se ejecuta este servidor en Node.js, la interfaz de la aplicación principal se conectará al servidor con el código en la ruta de la sección privada.

La aplicación en Electron no muestra (en su versión de producción) la posibilidad de cambiar la ruta renderizada, por lo que se reducirá únicamente a esta sección de la aplicación del lado del cliente. Puesto que el renderizado de componentes se realiza mediante importaciones perezosas, el código fuente de la sección pública no se cargará en ningún momento.

4.5- Despliegue en producción

Una de las cosas mas complicadas del proyecto, es el despliegue y configuración de un entorno de producción. Esto se debe al número de módulos, configuraciones, dependencias y diferentes tecnologías presentes en todo el proyecto.

Para solucionar esto se utilizará la tecnología Docker. Docker es una capa de virtualización de servicios.

Cada servicio del sistema estará presente en un contenedor Docker (en un entorno virtualizado) con el componente en cuestión cómo único servicio.

El principal problema que soluciona el uso de Docker es la compatibilidad en diferentes sistemas. Aunque las tecnologías que se usan en el proyecto son todas multi-plataforma, usar un entorno Docker evita problemas en futuras versiones de sistemas operativos o de las propias librerías del proyecto.

Además, para solucionar la complejidad de configuración y puesta en marcha de producción, se añade la automatización del proceso de despliegue, descargando todas las dependencias y recursos necesarios, como la maquina virtual de Java, Node.js, etc.

5- Conclusiones y líneas futuras

El proyecto resultante se define por sus dos principales funcionalidades.

La primera, como un intermediario entre la población y los sistemas de emergencias, ya sea a través de las redes sociales como en las publicaciones de alertas de la propia plataforma.

La segunda, como una biblioteca fácilmente accesible por la población a información oficial, ya sea mediante buscadores o mediante códigos QR.

El principal problema encontrado es el de mantener una mínima compatibilidad con posibles sistemas externos de las propias instituciones. Para ello se han añadido diferentes formateadores en las respuestas y filtros flexibles para realizar búsquedas.

Esto también ha ocasionado un problema con la falta de flexibilidad al utilizar un ODM (Object Document Mapper) que genera las consultas automáticamente sin tener que realizarlas manualmente. La solución ha sido utilizar una solución intermedia, donde la consulta se genera automáticamente en función de los parámetros de la llamada al API.

Respecto a las líneas futuras, al igual que se ha desarrollado una solución como aplicación de escritorios reutilizando el código de la plataforma, se podría también realizar una solución como aplicación móvil, reutilizando el código, pero haciendo una integración, esta vez mayor, debido a la complejidad de estos dispositivos.

Otra posible línea futura para realizar podría ser el desarrollo inverso del proyecto, en esta plataforma, el origen de los datos siempre es la propia plataforma. Podría implementarse un proyecto en que sea el propio software el que recoja los datos de las redes sociales o pueda recibir avisos y/o información de la población, para analizarla, exponerla, mejorarla etc.

Bibliografía

- Documentación de Spring – https://docs.spring.io/spring/docs/current/spring-framework-reference/
- Documentación de Vue.js – https://vuejs.org/
- Documentación de Vue Router – https://router.vuejs.org/
- Documentación de Vuex – https://vuex.vuejs.org/
- Documentación de Bulma – https://bulma.io/
- Documentación de Node.js – https://nodejs.org/en/
- Docuemtnación de Sass – https://sass-lang.com/
- Documentación de Electron.js – https://www.electronjs.org/
- Securing RESTful API with Spring Boot, Security and Data MongoDB – https://morioh.com/p/edd00fda48b0/securing-restful-api-with-spring-boot-security-and-data-mongodb
- Github – https://github.com
- Bulma templates – https://bulmatemplates.github.io/bulma-templates/
- Bulma vertical helpers – https://github.com/jgthms/bulma/issues/451#issuecomment-331758839
- Stack Overflow – https://stackoverflow.com

Apéndice

Manual de uso

Para acceder al área privada hay que insertar manualmente a la URL del host propio más "/private". Una vez se accede a esta vista, podrá verse una interfaz de inicio y/o registro de usuario.

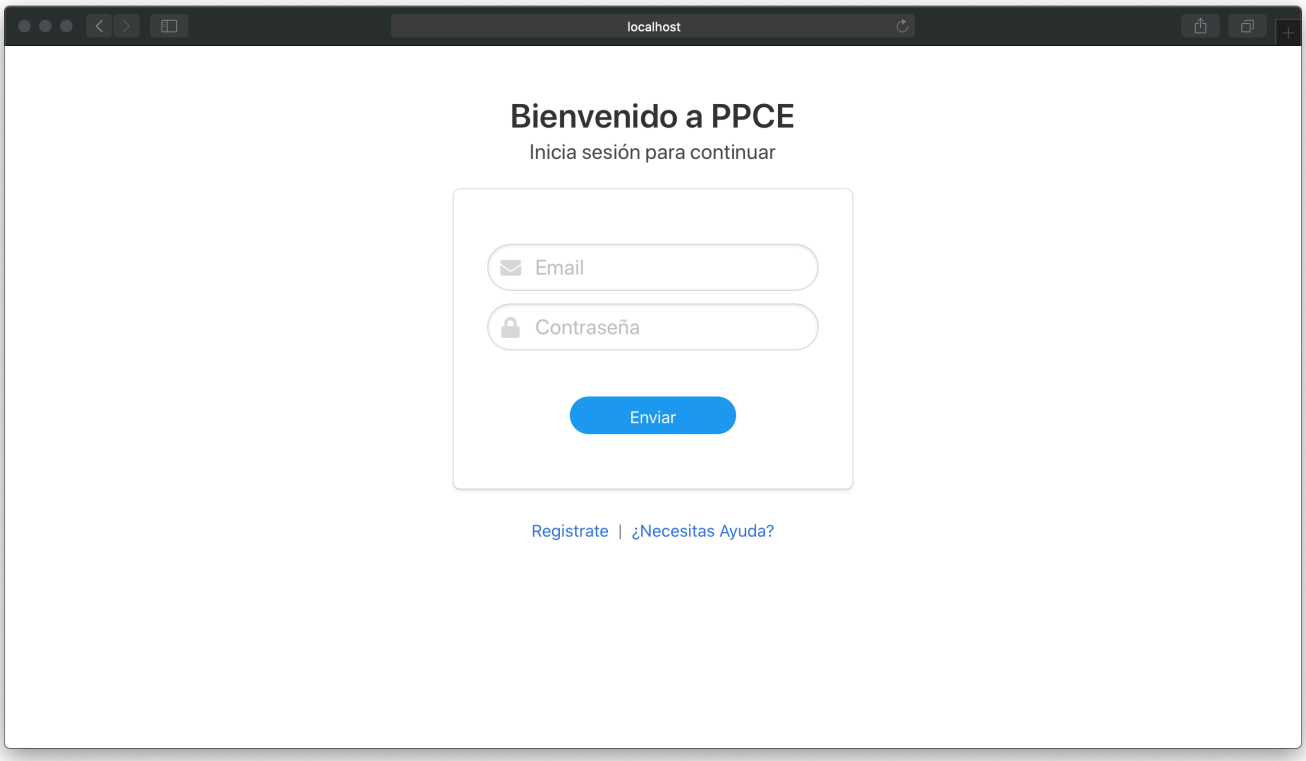

Ilustración 22.1: Interfaz de inicio de sesión

Esta interfaz requiere específicamente el email de un usuario registrado y validado y su correspondiente contraseña.

Debajo de esta interfaz se encuentra un botón que permite la muestra de información extra de contacto de la plataforma (vacía por defecto) y un botón para navegar hasta el registro.

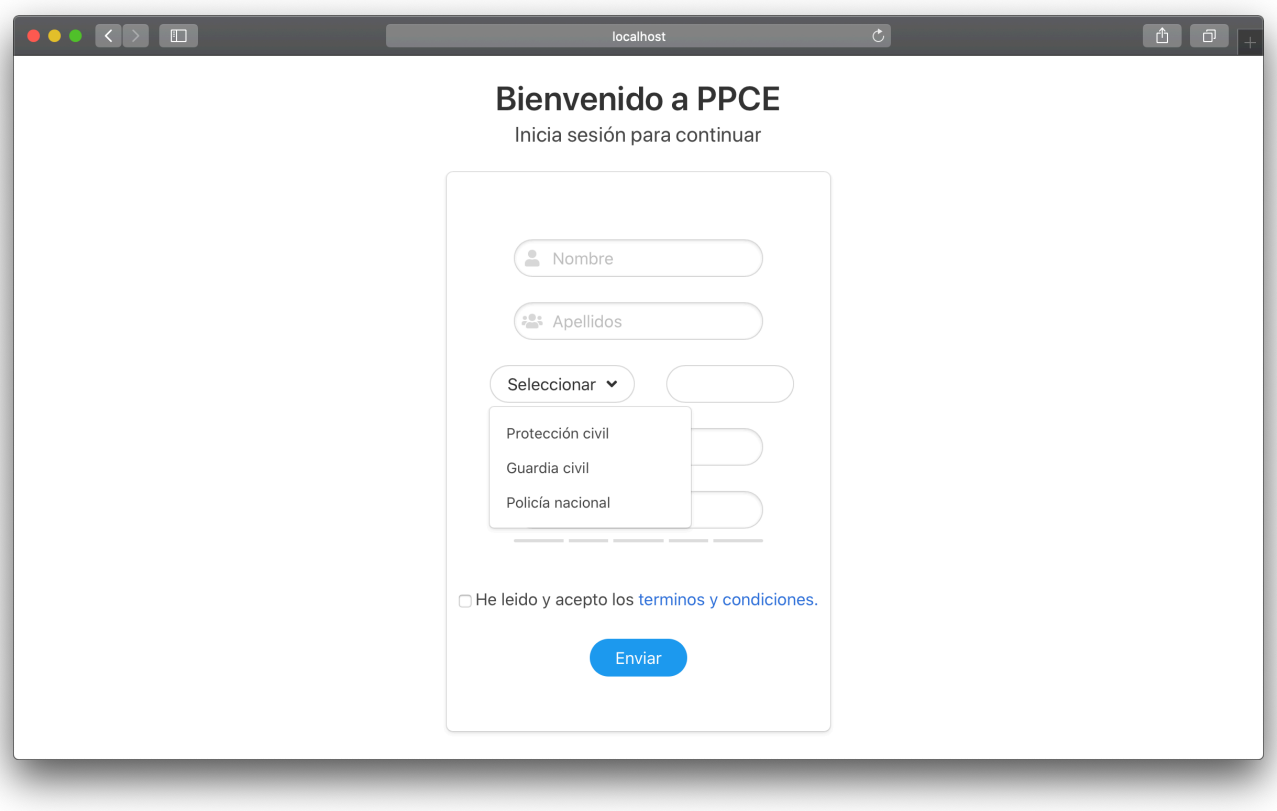

Ilustración 22.2: Interfaz de registro

Esta es la interfaz de registro de usuario, que requiere rellenar todos los campos.

Como puede verse, contiene un desplegable con todas las instituciones disponibles en la plataforma hasta el momento. Además, también se muestra una evaluación de la seguridad de la contraseña.

El registro no genera un usuario completo; posteriormente al registro aún no se podrá acceder al área privada a través de la interfaz anterior. Para que pueda acceder un usuario con los privilegios suficientes y de la misma institución que se ha aplicado en el registro, debe validar la solicitud.

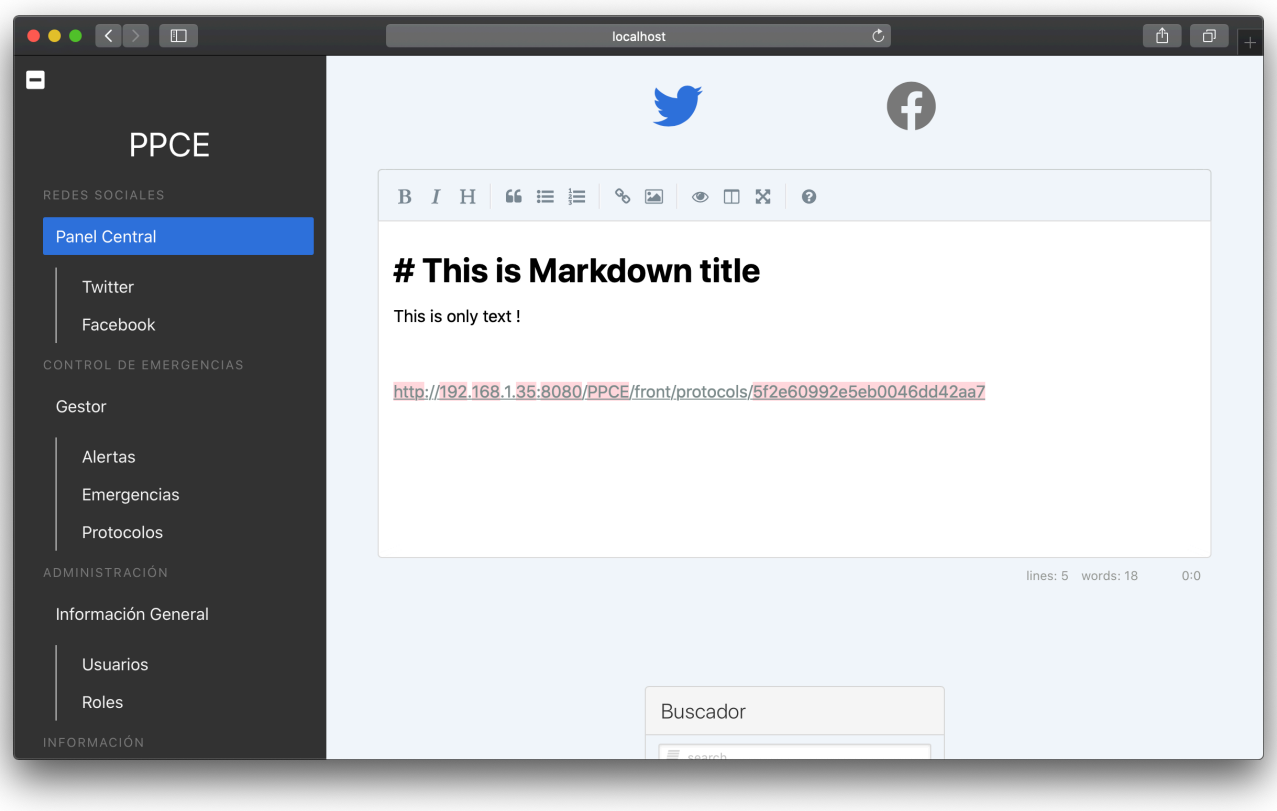

Ilustración 22.3: Panel central de redes sociales

En la ilustración 22.3 puede verse la interfaz gestión de las redes sociales. En el editor de texto que puede verse en la interfaz es donde se adjunta la información a añadir. Justo arriba se observan los logotipos de Facebook y Twitter, redes sociales a los que se enviará el mensaje de así marcarlas. En este caso únicamente se mandaría el texto a la red social Twitter.

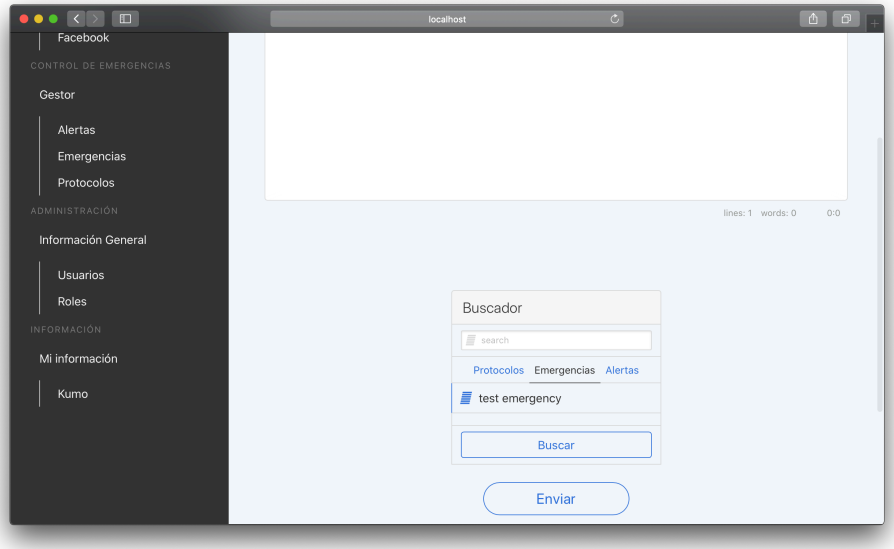

Ilustración 22.4: Buscador

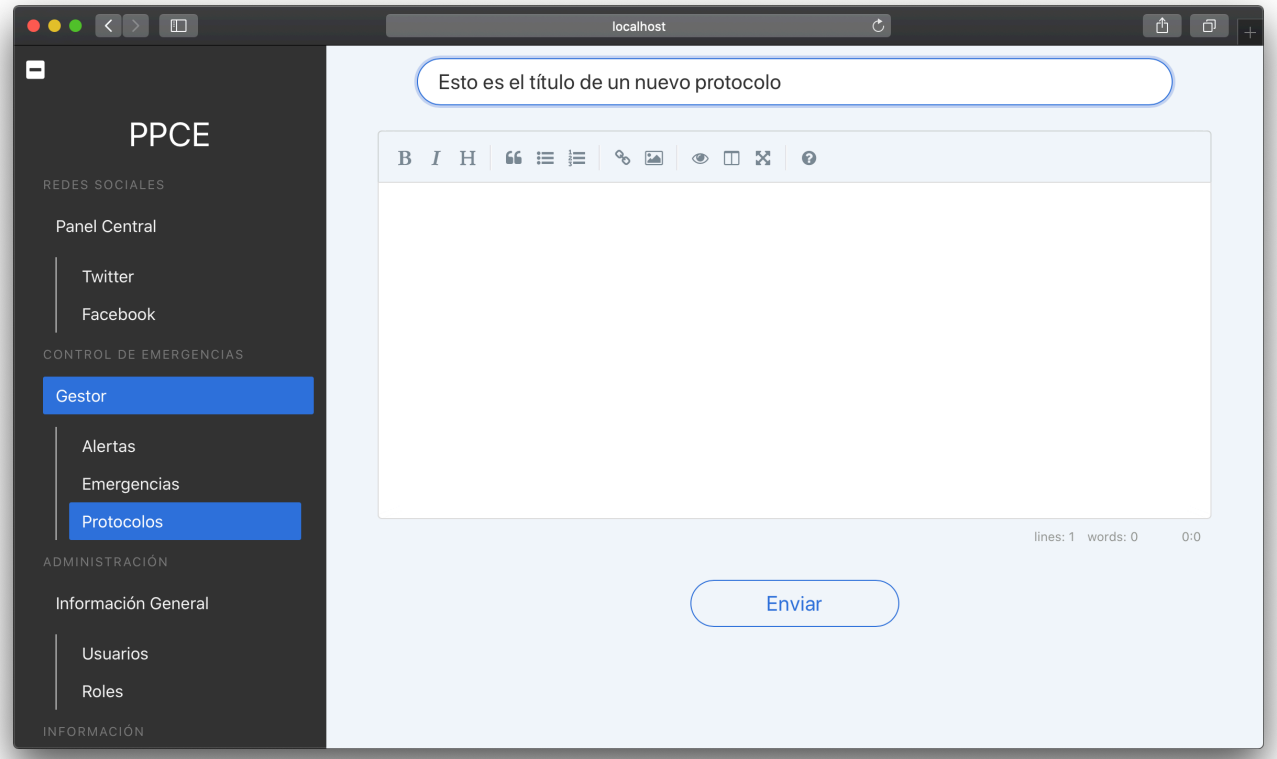

Ilustración 22.5: Gestor de contenido (interfaz de creación y edición)

Como se muestra en la ilustración 22.4, debajo del editor se encuentra el botón de enviar y un buscador que permite adjuntar al texto los enlaces a los protocolos, emergencias o alertas que se deseen.

En esta la ilustración 22.5 puede observarse el gestor de contenido. Aquí puede crearse el tipo de contenido ya visto a lo largo de las especificaciones. Según el tipo de contenido que se quiera editar o crear se deberán rellenar una serie de campos, como títulos, coordenadas, descripciones, etc. Todas las descripciones y textos se encuentran en formato Markdown, permitiendo añadir archivos multimedia, diferentes tipos de texto, diferentes fuentes, código, etc. Exceptuando la escritura mediante código HTML por motivos de seguridad ya descritos.

En el caso de las emergencias y las alertas, también se añade un buscador para poder enlazar con los correspondientes protocolos o entidades que se requieran.

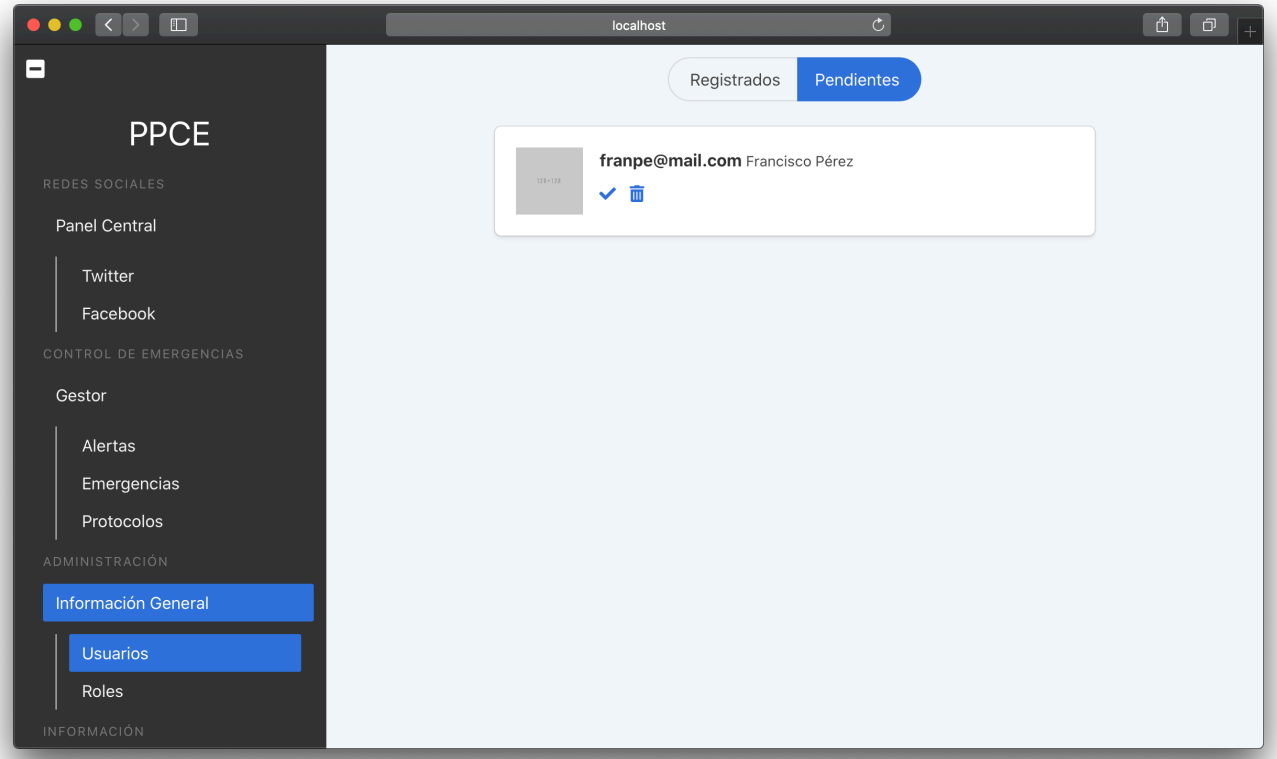

Ilustración 22.6: Registros pendientes de validación

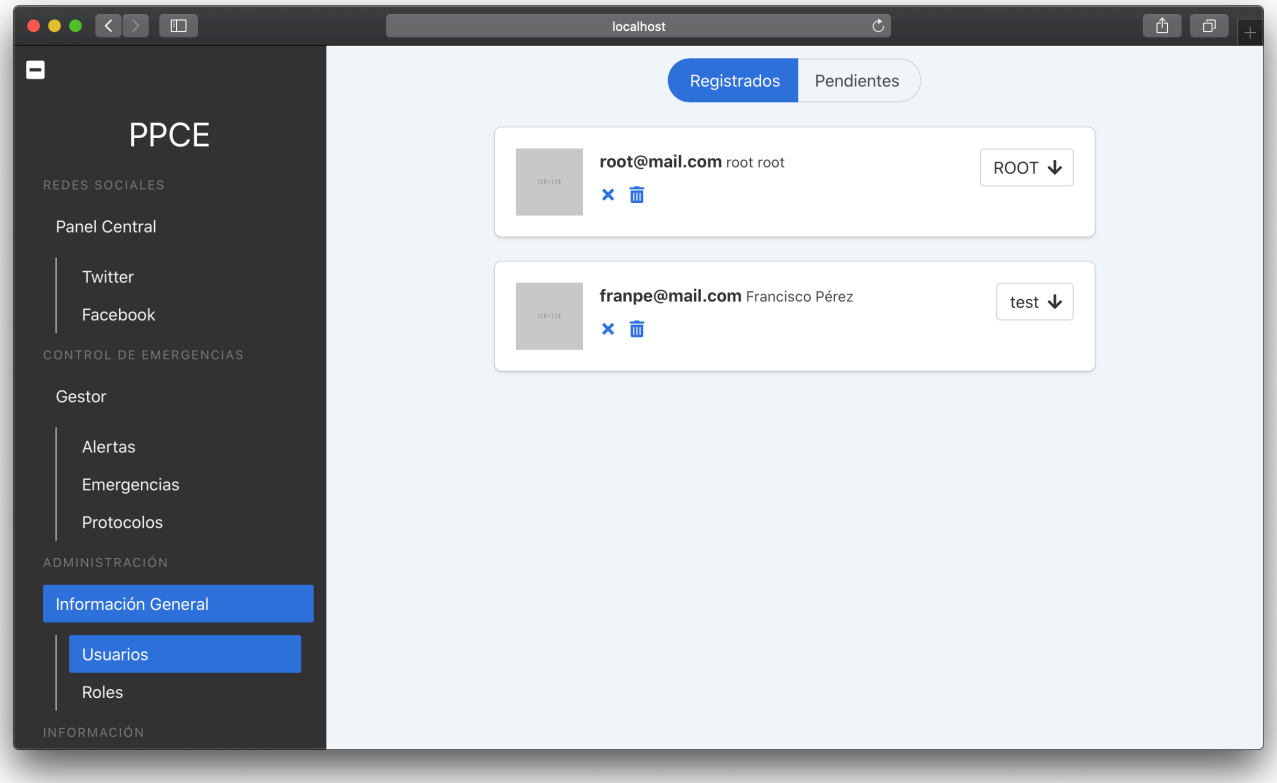

Ilustración 22.7: Usuarios registrados

Esta interfaz es para la validación de usuarios pendientes (ilustración 22.6). Una vez validados pasarán a la pestaña de "registrados" (Ilustración 22.7), donde el usuario con los permisos suficientes podrá borrarlos, cambiar su rol o ambas. Por defecto, todos los usuarios validados tienen el rol "Empty", que carece de privilegios.

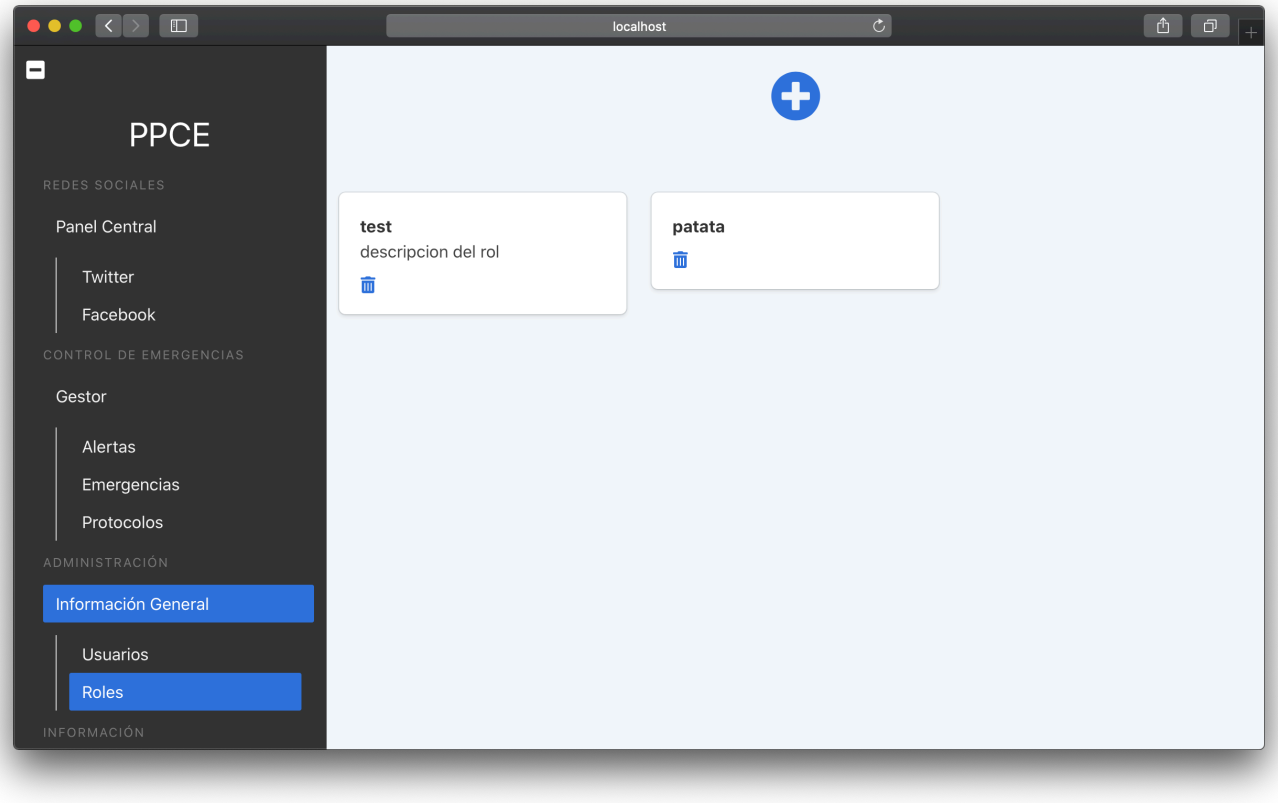

Ilustración 22.8: Gestión de roles

En la ilustración 22.8 puede verse el panel de roles. Cada usuario únicamente podrá ver los roles presentes en su institución. Los roles pueden eliminarse o crearse, pero no modificarse.

Tampoco se muestran listados los roles bloqueados y que, por lo tanto, no pueden eliminarse, como el rol "Empty" o "Root".

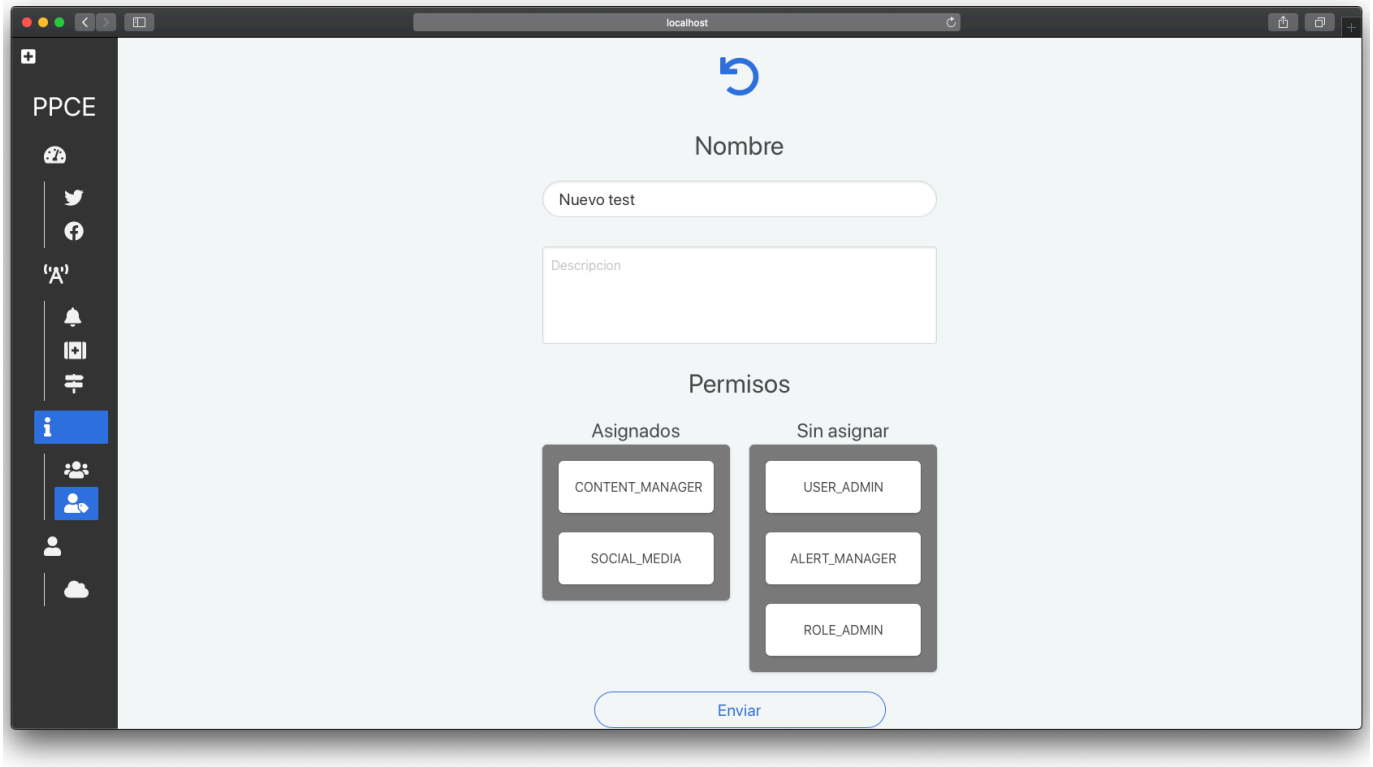

Ilustración 22.9: Creación de nuevos roles

En la ilustración 22.9 puede verse el panel de creación de roles. Los roles por defecto se crearán para la institución a la que pertenece el usuario que creó el rol.

Un rol además de un título, contendrá una descripción y una serie de privilegios.

En la interfaz se observan dos columnas; para añadir privilegios al nuevo rol únicamente se tiene que arrastrar las etiquetas a la columna de “Asignados”.

Ilustración 22.10: Archivos y multimedia en Kumo

En la ilustración 22.10 se puede ver la interfaz para añadir archivos al servidor "Kumo". Este servidor admite cualquier formato de archivo, aunque esta pensado principalmente para imágenes, permitiendo su carga asíncrona desde páginas web.

Al momento de subir una imagen, el servidor devuelve la URL de la imagen, que podrá añadirse directamente sobre los editores Markdown de cualquier tipo de contenido.

La lectura de la imagen se realizará a través del navegador como cualquier imagen desde HTML.

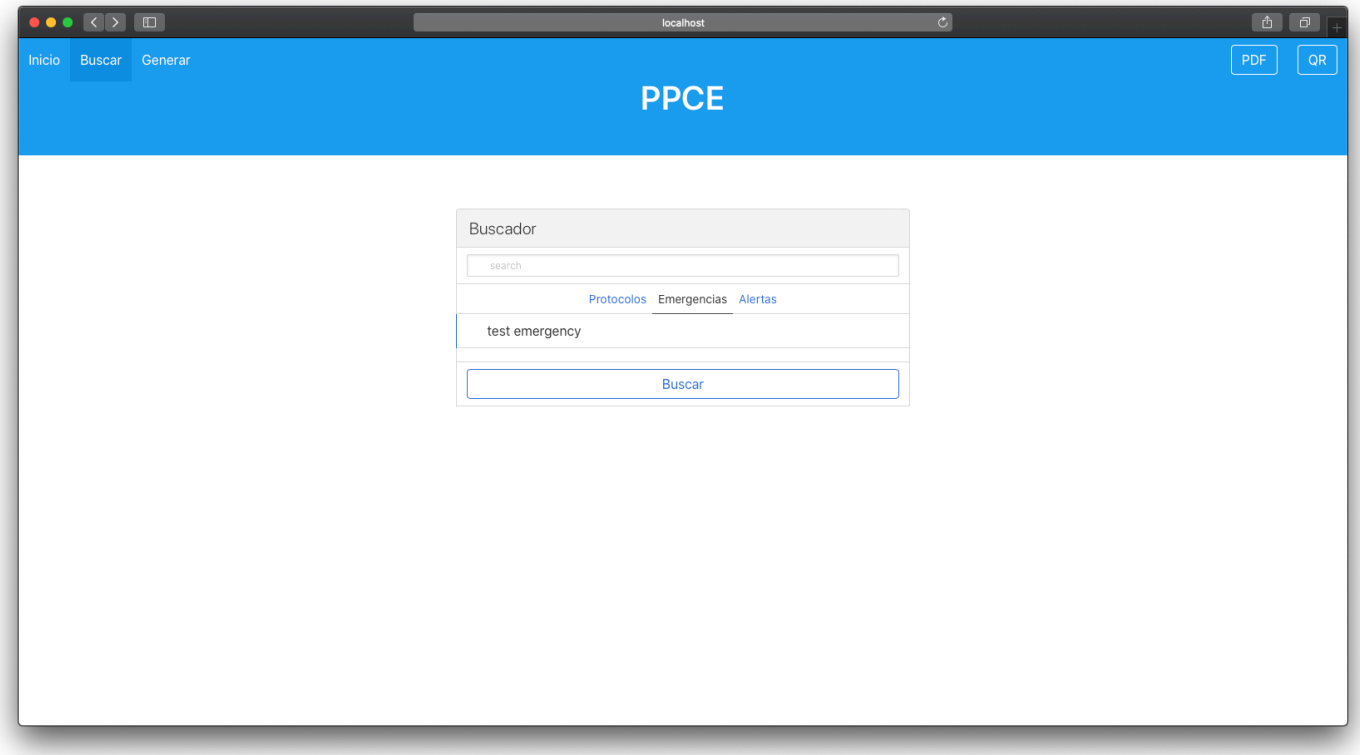

Ilustración 22.11: Buscador

El buscador es la principal herramienta de la sección pública, ubicada en el directorio raíz de la aplicación.

En el buscador pueden buscarse mediante palabras clave, protocolos, emergencias o alertas. En caso de no definir un criterio de búsqueda específico, la búsqueda se realizará mediante la fecha de creación de los elementos, en orden decreciente.

Ilustración 22.12: Visualización de un protocolo

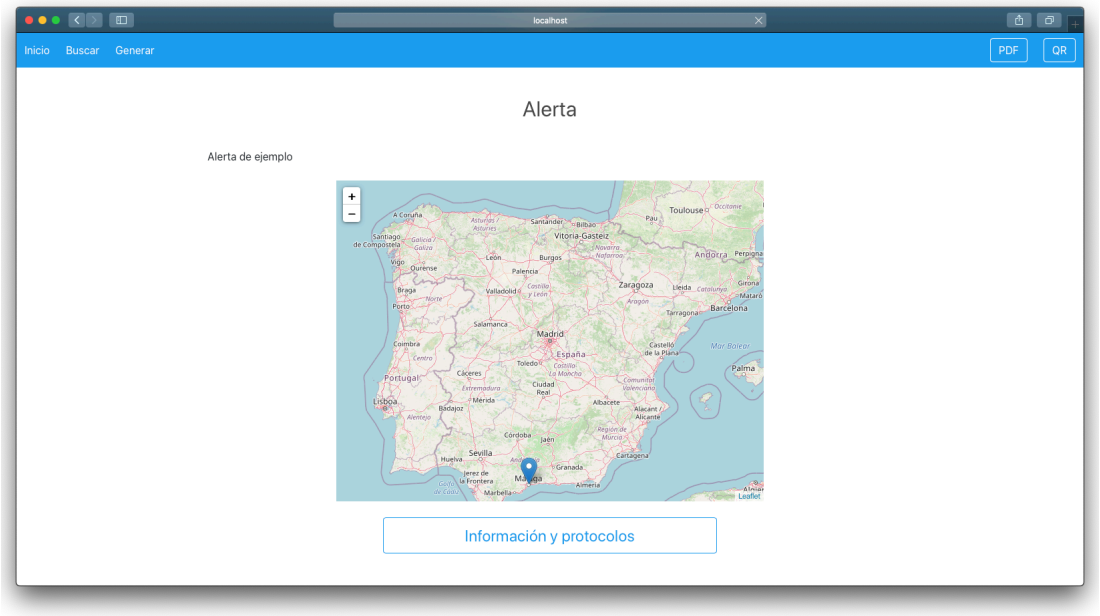

Ilustración 22.13: Visualización de una alerta

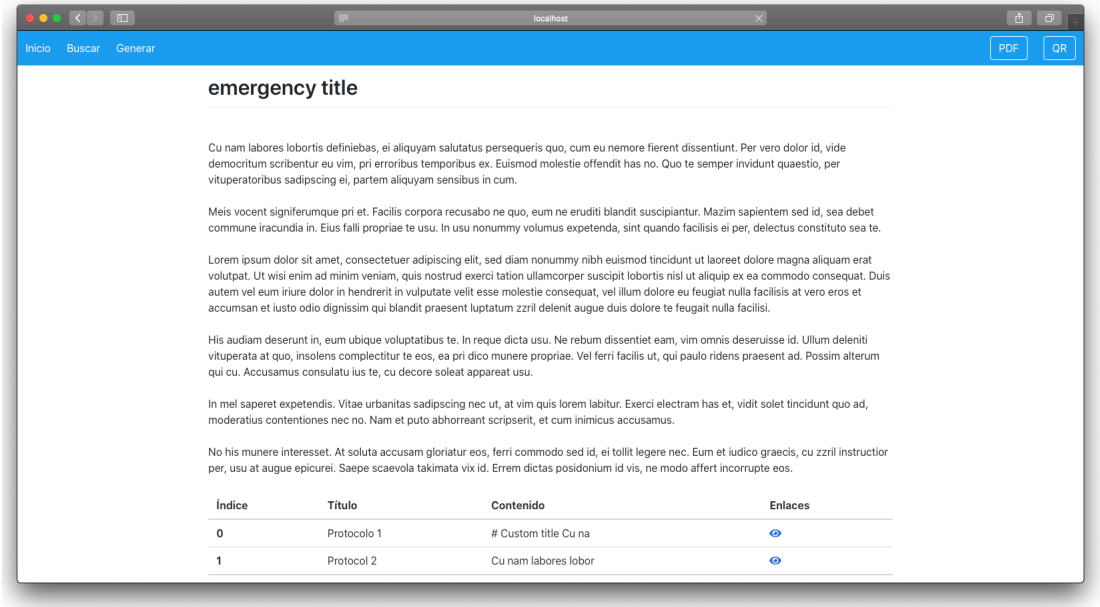

Ilustración 22.14: Visualización de una emergencia

En las visualizaciones de cada uno de los elementos, habrá disponible un texto generado del texto Markdown introducido.

En el caso además de la ilustración 22.13, las alertas dispondrán de un mapa con una serie de coordenadas y un enlace para visualizar sus correspondientes emergencias y/o protocolos asociados.

En el caso de la ilustración 22.14, las emergencias tendrán una lista de protocolos asociadas (parte inferior).

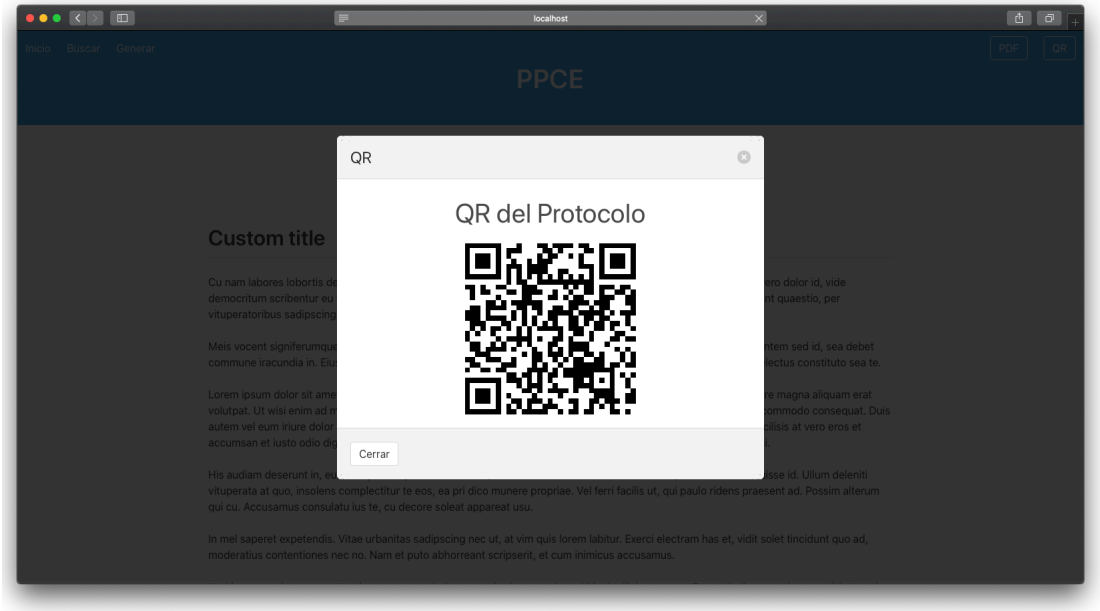

Ilustración 22.15: Generación de código QR

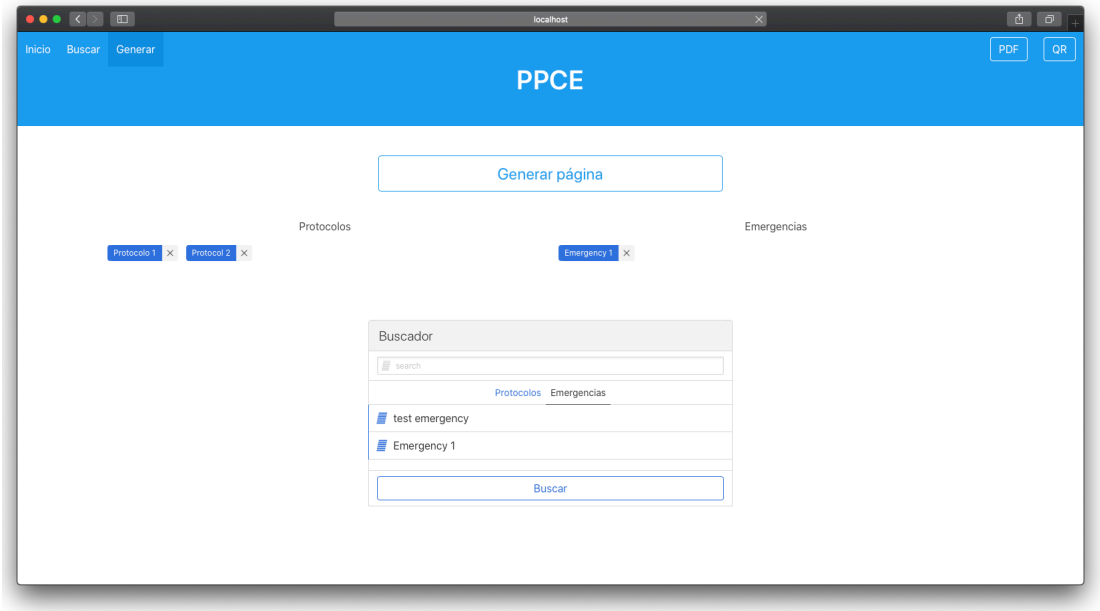

Ilustración 22.16: Generacdor de conjuntos

Para cualquiera de las interfaces anteriores, pueden generarse códigos QR que enlazan con el elemento presente. Este QR es una imagen descargable y localizable en cualquier lugar, enlazando con la información deseada.

Por último, se pueden generar conjuntos de emergencias y protocolos, de los cuales a su vez se pueden generar códigos QR. Estos conjuntos pueden realizarse sin necesidad de iniciar sesión. Estos conjuntos no se guardan en la base de datos ni ocupan espacio físico en la plataforma; únicamente se trata de una serie de referencias a los recursos.

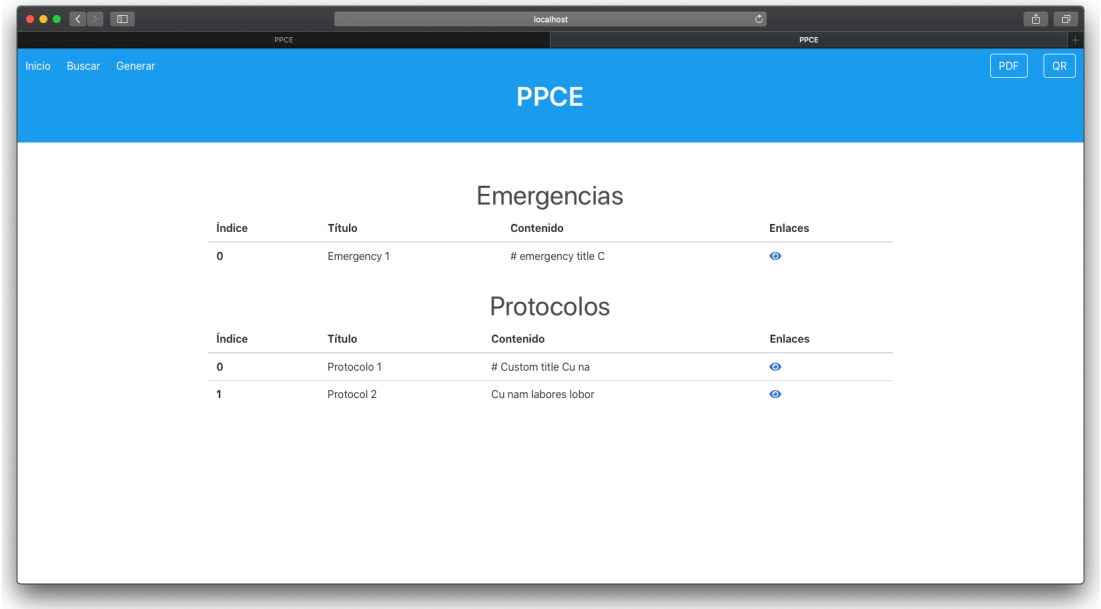

Ilustración 22.17: Conjunto de protocolos y emergencias

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA