ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
INGENIERÍA DEL SOFTWARE

# ENTORNO SOFTWARE PARA LA PREVENCIÓN DE EMERGENCIAS

# SOFTWARE ENVIROMENT FOR EMERGENY PREVENTION

Realizado por
**SERGIO RODRÍGUEZ MOYANO**
Tutorizado por
**EDUARDO GUZMÁN DE LOS RISCOS**
Departamento
**LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN**

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO 2017

Fecha defensa:
El Secretario del Tribunal

**RESUMEN:**

Este trabajo viene a resolver el problema de la sobreinformación que existe en la actualidad, sobre todo, a través de las redes sociales, en las cuales es fácil encontrar bulos que pueden dar lugar a confusión. Además, también resuelve la desinformación que puede llegar a tener la población ante casos de emergencia habituales, dando pautas a seguir cuando ocurran estas.

Para esto, se ha desarrollado una plataforma web y una aplicación móvil, además de una API REST para el tratamiento de datos. Para el almacenamiento de imágenes, se ha hecho uso de la plataforma Dropbox, utilizando su API.

En la plataforma web, se podrá tratar todos los datos del sistema, añadiendo emergencias, información oficial veraz, y añadiendo protocolos a seguir ante cualquier emergencia, editando aquellos que ya existan en el sistema. Además, se podrá realizar toda la gestión de usuarios del sistema.

En la aplicación móvil, los usuarios podrán consultar las emergencias actuales, la información oficial asociada a estas y enviar información acerca de su estado, adjuntando imágenes si lo ven necesario. Además, podrá consultar las pautas a seguir en cada emergencia.

**PALABRAS CLAVES**:

Emergencia, emergencia oficial, emergencia no oficial, Ionic, API Rest, protocolos

**ABSTRACT:**

In this Project, we want to resolve overinformation that exist nowadays, especially in social media, where we can find not verified news. Also, the project resolve disinformation about how to act when an emergency happen.

We have developed web platform and app mobile. Also, we have developed an API Rest to manage data. We use Dropbox to store images, with their API.

In the web platform, users will can manage all data in the system, adding emergencies, adding verified official information, and adding action protocols about any emergency, editing those that exist in the system. Also some users will can make user management.

In the app mobile, users will can consult current emergencies, official information associated to them and send information about their state, append images if they want. Also, they will can consult action protocols.

**KEYWORDS:**

Emergency, official emergency, no official emergency, Ionic, API Rest, protocols

# ÍNDICE

# 1 INTRODUCCIÓN

Esta introducción servirá para definir un poco más en qué consiste este proyecto.

## 1.1 Problemática

En la actualidad, Internet ha supuesto una gran fuente de información a la que acceder recurrentemente para todo. El mayor problema viene a la hora de juzgar si esa información es verídica o no. Habitualmente, se da como veraz cualquier noticia que los usuarios se encuentran, mayoritariamente en las redes sociales, y esto es un gran problema para la sociedad. Más si cabe cuando acontecen hechos como los que están ocurriendo desgraciadamente en la actualidad, como son los atentados terroristas. Esto es solo un ejemplo, porque ocurre con cualquier tipo de emergencia que se nos ocurra. Esta es la problemática que viene a resolver este proyecto.

## 1.2 Objetivos

El proyecto va a ser un trabajo de fin de grado en grupo, junto a mi compañero Ignacio Olivares Subías. El objetivo principal del proyecto es el de crear un entorno software centralizado en el que poder dar una fuente de información verificada y en tiempo real a la ciudadanía acerca de emergencias que ocurran. Además, se pretenda que sea colaborativa, que los ciudadanos encuentren en la aplicación un sitio donde poder aportar todo tipo de información acerca de dichas emergencias.

Mi parte del proyecto, la aplicación móvil, viene a cubrir la necesidad de la ciudadanía de consumir las diferentes emergencias que se dan de alta, y consultar tanto la información verificada aportada por los diferentes equipos VOST como la información que vaya aportando la ciudadanía (previo filtrado para descartar aquella información que no sea apropiada), además de las pautas básicas de actuación ante casos de emergencia.

## 1.3 Tecnologías utilizadas

En este apartado se va a describir las diferentes tecnologías que se han utilizado en este sistema:

### 1.3.1 Java EE

Plataforma de programación web basada en Java. Ha sido utilizada para construir la API donde se gestiona toda la conexión con la base de datos y para la aplicación web.

### 1.3.2 MySQL

Sistema de gestión de base de datos relacional propiedad de Oracle. Permite la persistencia de los datos en el sistema.

### 1.3.3 Ionic

Framework para desarrollar aplicaciones móviles híbridas basado en Angular JS y Apache Cordova, además de tecnologías web como son TypeScript, CSS y HTML5. De esta forma, se ha desarrollado una aplicación móvil compatible con los sistemas operativos que usa la mayoría de los dispositivos que dispone la gente, iOS y Android. Incluso es posible hacer la aplicación para otros sistemas operativos en un futuro, ya que el framework se encuentra en continuo crecimiento y puede llegar a ser compatible con ellos.

Para poder acceder a todas las funcionalidades nativas del dispositivo, contiene una gran cantidad de plugins para poder, desde acceder a la cámara, hasta acceder a los servicios de notificaciones push.

### 1.3.4 NetBeans

Entorno de desarrollo, hecho principalmente para construir aplicaciones en Java. Tiene la ventaja de que es modular, por lo que se pueden incluir una gran cantidad de funcionalidades bajo demanda.

Utilizado para el desarrollo de la API REST.

### 1.3.5 Git

Sistema de control de versiones que permite la gestión de los cambios que se van produciendo dentro del proyecto.

### 1.3.6 jQuery

Librería basada en JavaScript que permite realizar funcionalidades con menos código y de forma más simple y rápida, pudiendo acceder y modificar elementos del DOM, propiedades de CSS, llamadas AJAX, etc.

### 1.3.7 Bootstrap

Framework de diseño de aplicaciones web. Hace uso de HTML, JavaScript y CSS. Gracias a este framework, se ha podido usar plantillas para el diseño del front-end del cliente web.

### 1.3.8 SweetAlert

Plugin para mejorar estéticamente las alertas realizadas a través de JavaScript

### 1.3.9 Bootstrap FileInput

Plugin basado en Bootstrap y JavaScript para mejorar estéticamente los campos de entrada de los ficheros.

### 1.3.10 TinyMce

Plugin de editor de texto con formato HTML que hace uso de JavaScript. Utilizado en la aplicación web.

### 1.3.8 Dropbox

Dropbox es un servicio de almacenamiento en la nube. A través de su API, se hace uso de su servicio para almacenar las imágenes que se suben tanto a través de la web como de la aplicación móvil.

# 2 ANÁLISIS DE REQUISITOS

Para hacer la captura de requisitos, se hizo una entrevista personal no estructurada con el cliente.

## 2.1 Requisitos funcionales

En este apartado, se va a listar todos los requisitos funcionales del sistema. (AM = Aplicación móvil, AWEB = Aplicación Web).

| ID | Nombre | Descripción |
|---|---|---|
| **AWEB1** | Iniciar Sesión | El usuario podrá iniciar sesión en el sistema mediante correo electrónico y contraseña |
| **AWEB2** | Cerrar Sesión | El usuario podrá salir del sistema |
| **AWEB3** | Ver perfil | El usuario podrá ver sus datos de usuario |
| **AWEB4** | Modificar perfil | El usuario podrá modificar sus datos personales salvo el correo electrónico |
| **AWEB5** | Cambiar contraseña | El usuario podrá cambiar su contraseña |
| **AWEB6** | Ver lista de protocolos | El usuario podrá ver una lista con todos los protocolos |
| **AWEB7** | Ver Protocolo | El usuario podrá ver la descripción de un protocolo |
| **AWEB8** | Añadir Protocolo | El usuario podrá añadir un nuevo protocolo |

| ID | Nombre | Descripción |
|---|---|---|
| **AWEB9** | Editar Protocolo | El usuario podrá editar la descripción de un protocolo existente |
| **AWEB10** | Ver lista de Emergencias recientes | El usuario podrá ver una lista con las emergencias más recientes. |
| **AWEB11** | Ver lista de Emergencias | El usuario podrá ver una lista con todas las emergencias |
| **AWEB12** | Filtrar Emergencias | El usuario podrá aplicar un filtro por Comunidades autónomas a la lista de las emergencias |
| **AWEB13** | Ver Emergencia | El usuario podrá ver toda la información de una emergencia, mostrando la información oficial y no oficial |
| **AWEB14** | Añadir Emergencia | El usuario podrá añadir una emergencia |
| **AWEB15** | Añadir información oficial | El usuario podrá añadir información oficial |
| **AWEB16** | Añadir información no oficial | El usuario podrá añadir información no oficial |
| **AWEB17** | Validar información | El usuario podrá aceptar/rechazar información no oficial |
| **AWEB18** | Solicitud voluntario | El usuario podrá enviar solicitud para darse de alta como voluntario |
| **AWEB19** | Alta de voluntarios | El usuario podrá aceptar solicitudes de voluntarios |
| **AWEB20** | Activar/desactivar voluntario | El usuario podrá activar o desactivar un voluntario. |

| ID | Nombre | Descripción |
|---|---|---|
| **AWEB21** | Añadir Usuario | El usuario podrá dar de alta un nuevo usuario. |
| **AWEB22** | Ver lista de usuarios | El usuario podrá ver la lista de todos usuarios pertenecientes al sistema. |
| **AWEB23** | Visualizar estado en línea | El usuario podrá ver la última conexión de cada usuario y su estado en línea. |
| **AWEB24** | Ver equipo | El usuario podrá ver la lista de usuarios pertenecientes a un equipo. |
| **AWEB25** | Crear equipo | El usuario podrá crear un equipo asignándole un coordinador ya creado. |
| **AWEB26** | Archivar emergencia | El usuario podrá archivar una emergencia. |
| **AWEB27** | Filtrar información no oficial | El usuario podrá aplicar un filtro sobre la lista de información no oficial. |
| **AWEB28** | Ver lista de equipos | El usuario podrá ver una lista de todos los equipos del sistema. |
| **AWEB29** | Ver lista de solicitudes | El usuario podrá ver la lista de las solicitudes de voluntarios pendientes de ser aceptadas. |
| **AWEB30** | Recuperar contraseña | El usuario podrá recuperar su contraseña |

| ID | Nombre | Descripción |
|---|---|---|
| **AM1** | Listado de emergencias | El usuario podrá ver el listado de emergencias |
| **AM2** | Listado de información oficial | El usuario podrá ver el listado de post oficiales |
| **AM3** | Listado de información no oficial | El usuario podrá ver el listado de post no oficiales |
| **AM4** | Recargar lista de emergencias | El usuario podrá recargar el listado de emergencias |
| **AM5** | Recargar lista de información oficial | El usuario podrá recargar el listado de información oficial |
| **AM6** | Recargar lista de información no oficial | El usuario podrá recargar el listado de información no oficial |
| **AM7** | Insertar información no oficial | El usuario podrá crear información no oficial |
| **AM8** | Adjuntar imagen en post no oficial | El usuario podrá adjuntar una imagen si lo desea al crear información no oficial |
| **AM9** | Filtrar información no oficial | El usuario podrá filtrar la información no oficiales |
| **AM10** | Listado de protocolos | El usuario podrá ver la lista de protocolos de actuación ante casos de emergencia |
| **AM11** | Filtrar protocolos | El usuario podrá aplicar un filtro la lista de protocolos |

| ID | Nombre | Descripción |
|---|---|---|
| **AM12** | Ver pautas de un protocolo | El usuario podrá visualizar el contenido de las pautas más comunes ante una cierta emergencia |
| **AM13** | Llamar al 112 | El usuario podrá llamar al teléfono 112 a través de un botón directo que aparecerá en todas las pantallas de la app |
| **AM14** | Notificaciones Push | El usuario será informado a través de una notificación push cuando se dé de alta una nueva emergencia |

## 2.2 Requisitos no funcionales

Estos requisitos determinan las condiciones sobre las que se tiene que ejecutar la aplicación diseñada.

| Categoría | ID | Descripción |
|---|---|---|
| **Hardware** | **RNF1** | La aplicación móvil debe estar disponible en los sistemas operativos iOS y Android |
| **Usabilidad** | **RNF2** | La aplicación móvil debe tener un botón de llamar al 112 siempre accesible desde cualquier pantalla |
| **Usabilidad** | **RNF3** | Algunas vistas tendrían una recarga automática, después de un periodo de tiempo establecido |
| **Seguridad** | **RNF4** | No se permitirá el acceso a funcionalidades con privilegios a ciertos usuarios |

## 2.3 Actores

Los actores dentro del sistema son:

- Usuarios no identificados: son todos aquellos que no estén registrados en el sistema, tanto en la aplicación web como en la móvil.
- Voluntarios: este rol tiene la única funcionalidad de validar información no oficial. Puede ser activado y desactivado una vez registrado
- Miembros: este rol pertenece a un equipo VOST y podrá modificar protocolos de actuación, además de validar información no oficial
- Coordinadores: este rol está destinado a la administración de un equipo, pudiendo añadir nuevos usuarios o monitorizar si están conectados, por ejemplo.
- Administrador: este rol puede abarcar todas las funcionalidades del sistema.

## 2.4 Casos de uso

En este apartado se presentará el diagrama de casos de uso planteado para este proyecto:

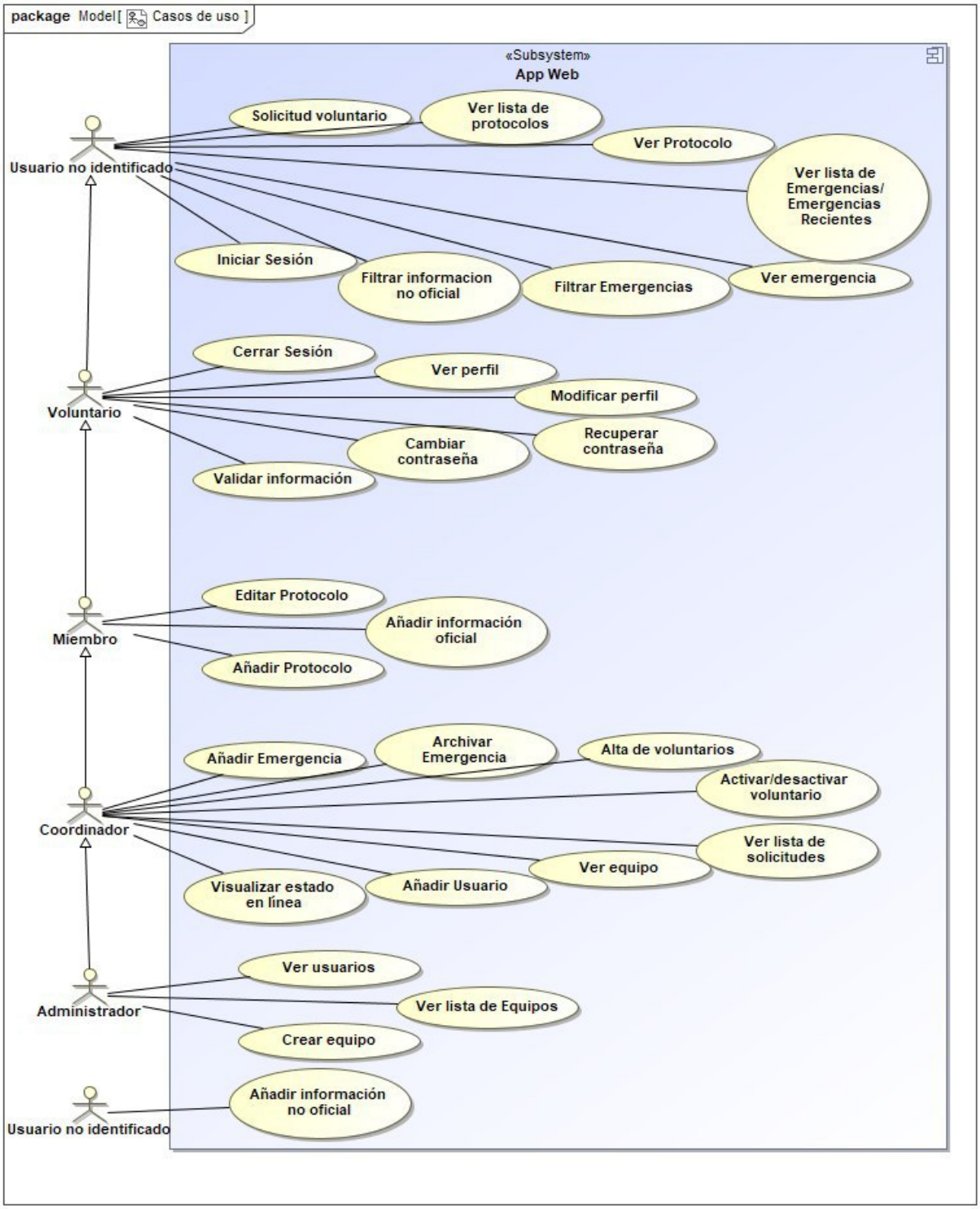


*Ilustración 1. Diagrama de casos de uso de la aplicación web*

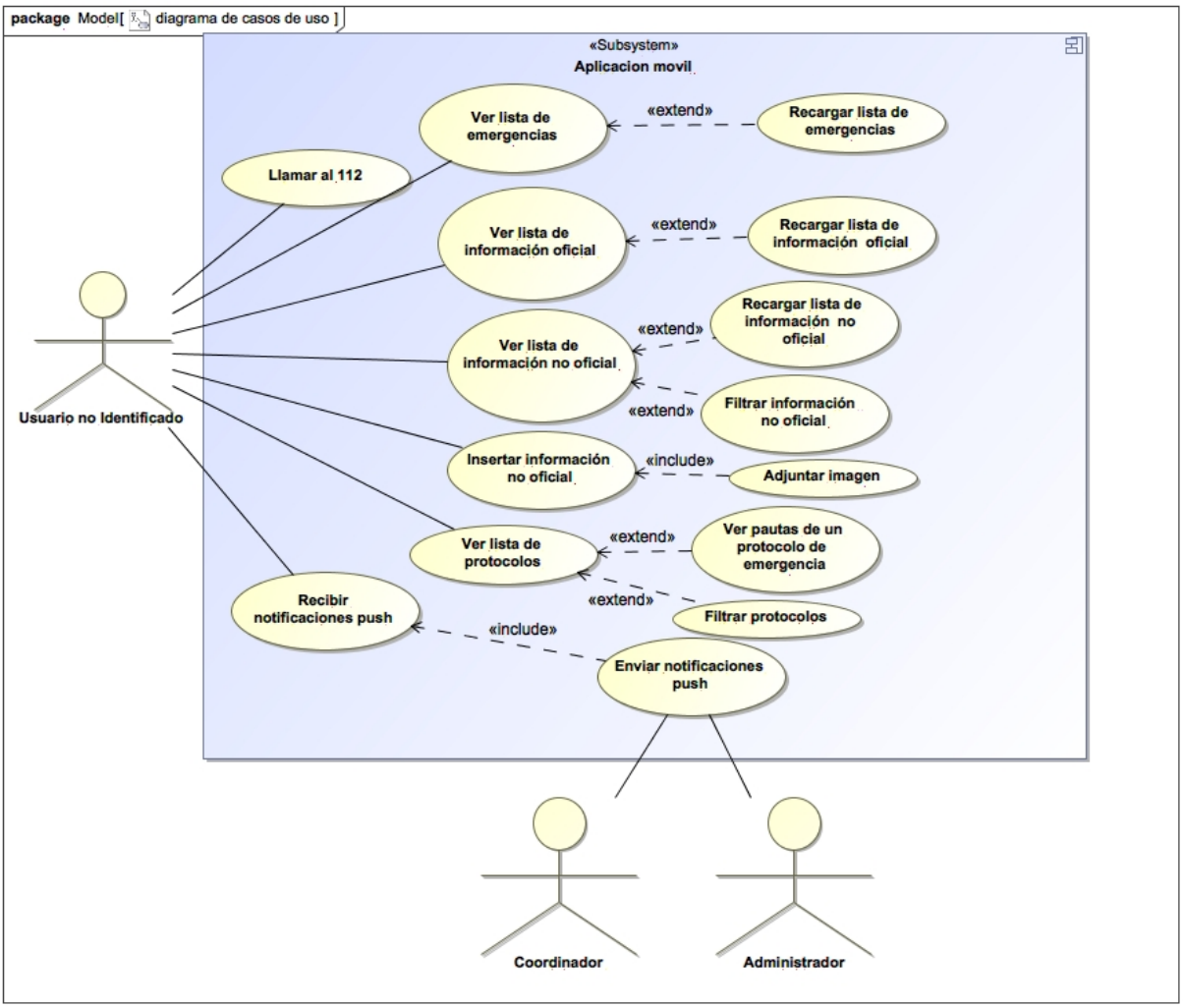


*Ilustración 2. Diagrama de casos de uso de la aplicación móvil*

## 2.5 Descripción de los casos de uso

A continuación, se detallarán los casos de uso que he desarrollado, indicando a qué requisito pertenece, e identificando cada uno de ellos.

| ID | CU1 |
|---|---|
| **Caso de uso** | Ver lista de emergencias |
| **Descripción** | El usuario podrá visualizar todas las emergencias que no estén archivadas |
| **Precondiciones** | Ninguna |
| **Escenario principal** | 1. El usuario accede a la aplicación.<br>2. El usuario observa en la pantalla principal todas las emergencias no archivadas. |
| **Escenario alternativo** | |
| **Requisito asociado** | AM1 |

| ID | CU2 |
|---|---|
| **Caso de uso** | Ver lista de información oficial |
| **Descripción** | El usuario podrá visualizar la lista de información oficial asociada a una emergencia |
| **Precondiciones** | El usuario tiene que seleccionar una emergencia en la pantalla principal |
| **Escenario principal** | 1. El usuario selecciona una de las emergencias.<br>2. El usuario accede a la página con la información asociada a la emergencia seleccionada. |
| **Escenario alternativo** | |
| **Requisito asociado** | AM2 |

| ID | CU3 |
|---|---|
| **Caso de uso** | Ver lista de información no oficial |
| **Descripción** | El usuario podrá visualizar toda la información no oficial asociada a una emergencia |
| **Precondiciones** | El usuario tiene que seleccionar una emergencia en la pantalla principal |
| **Escenario principal** | 1. El usuario selecciona una de las emergencias.<br>2. El usuario accede a la página con la información asociada a la emergencia seleccionada.<br>3. El usuario selecciona la pestaña de información no oficial. |
| **Escenario alternativo** | |
| **Requisito asociado** | AM3 |

| ID | CU4 |
|---|---|
| **Caso de uso** | Recargar lista de emergencias |
| **Descripción** | El usuario podrá recargar la lista de emergencias |
| **Precondiciones** | Ninguna |
| **Escenario principal** | 1. El usuario accede a la aplicación.<br>2. El usuario observará en la pantalla principal todas las emergencias no archivadas. |
| **Escenario alternativo** | |
| **Requisito asociado** | AM4 |

| ID | CU5 |
|---|---|
| **Caso de uso** | Recargar lista de información oficial |
| **Descripción** | El usuario podrá recargar la lista de información oficial |
| **Precondiciones** | El usuario tiene que seleccionar una emergencia en la pantalla principal |
| **Escenario principal** | 1. El usuario selecciona una de las emergencias.<br>2. El usuario accede a la página con la información asociada a la emergencia seleccionada. |
| **Escenario alternativo** | |
| **Requisito asociado** | AM5 |

| ID | CU6 |
|---|---|
| **Caso de uso** | Recargar lista de información no oficial |
| **Descripción** | El usuario podrá recargar la lista de información no oficial |
| **Precondiciones** | El usuario tiene que seleccionar una emergencia en la pantalla principal |
| **Escenario principal** | 1. El usuario selecciona una de las emergencias.<br>2. El usuario accede a la página con la información asociada a la emergencia seleccionada.<br>3. El usuario selecciona la pestaña de información no oficial. |
| **Escenario alternativo** | |
| **Requisito asociado** | AM6 |

| ID | CU7 |
|---|---|
| **Caso de uso** | Insertar información no oficial |
| **Descripción** | El usuario podrá crear nueva información no oficial |
| **Precondiciones** | El usuario tiene que seleccionar una emergencia en la pantalla principal |
| **Escenario principal** | 1. El usuario selecciona una de las emergencias.<br>2. El usuario accede a la página con la información asociada a la emergencia seleccionada.<br>3. El usuario pulsa el botón de nueva información no oficial.<br>4. El usuario rellena los datos que se le piden.<br>5. El usuario pulsa el botón enviar. |
| **Escenario alternativo** | 5.1.1. El usuario no introduce los datos necesarios para crear un post.<br>5.1.2. La aplicación manda una alerta al usuario informándole de este error, sin llegar a crear la información no oficial.<br>5.2.1. La conexión con Dropbox ha sido incorrecta.<br>5.2.2. Se crea la información no oficial, pero la imagen no se adjunta a esta. |
| **Requisito asociado** | AM7 |

| ID | CU8 |
|---|---|
| **Caso de uso** | Adjuntar imagen en información no oficial |
| **Descripción** | El usuario podrá adjuntar una imagen al crear una nueva información no oficial |
| **Precondiciones** | El usuario tiene que estar en la pantalla de crear nueva información no oficial |
| **Escenario principal** | 1. El usuario pulsa el botón de nueva información no oficial.<br>2. El usuario pulsa el botón de la cámara o de la galería.<br>3. El usuario hace una foto o selecciona una imagen de la galería. |
| **Escenario alternativo** | 3.1. El usuario pulsa el botón de la papelera y borra la imagen que tenía preparada para enviar. |
| **Requisito asociado** | AM8 |

| ID | CU9 |
|---|---|
| **Caso de uso** | Filtrar información no oficial |
| **Descripción** | El usuario podrá filtrar la lista de información no oficial |
| **Precondiciones** | El usuario tiene que seleccionar una emergencia en la pantalla principal y seleccionar la lista de información no oficial |
| **Escenario principal** | 1. El usuario selecciona una de las emergencias.<br>2. El usuario accede a la página con la información asociada a la emergencia seleccionada.<br>3. El usuario selecciona la pestaña de información no oficial.<br>4. El usuario pulsa filtrar.<br>5. El usuario rellena los datos que estime oportunos.<br>6. El usuario pulsa en el botón filtrar. |
| **Escenario alternativo** | |
| **Requisito asociado** | AM9 |

| ID | CU10 |
|---|---|
| **Caso de uso** | Listado de protocolos |
| **Descripción** | El usuario podrá visualizar la lista de protocolos de actuación ante casos de emergencia |
| **Precondiciones** | Ninguna |
| **Escenario principal** | 1. El usuario accede a la aplicación.<br>2. El usuario pulsa en la pestaña protocolos.<br>3. El usuario observa en la pestaña de protocolos todos los protocolos de actuación. |
| **Escenario alternativo** | |
| **Requisito asociado** | AM10 |

| ID | CU11 |
|---|---|
| **Caso de uso** | Filtrar protocolos |
| **Descripción** | El usuario podrá aplicar un filtro la lista de protocolos de actuación ante casos de emergencia |
| **Precondiciones** | Ninguna |
| **Escenario principal** | 1. El usuario accede a la aplicación.<br>2. El usuario pulsa en la pestaña protocolos.<br>3. El usuario escribe en el buscador.<br>4. El sistema carga la lista de protocolos. |
| **Escenario alternativo** | |
| **Requisito asociado** | AM11 |

| ID | CU12 |
|---|---|
| **Caso de uso** | Ver pautas de un protocolo |
| **Descripción** | El usuario podrá visualizar el contenido de un protocolo de actuación ante casos de emergencia |
| **Precondiciones** | El usuario tiene que seleccionar un protocolo en la pestaña |
| **Escenario principal** | 1. El usuario selecciona un protocolo de la lista.<br>2. El usuario accederá a la página con la información asociada al protocolo seleccionado. |
| **Escenario alternativo** | |
| **Requisito asociado** | AM12 |

| ID | CU13 |
|---|---|
| **Caso de uso** | Llamar al 112 |
| **Descripción** | El usuario podrá llamar al teléfono de emergencias 112 |
| **Precondiciones** | El usuario tiene que poder llamar desde cualquier pantalla de la aplicación |
| **Escenario principal** | 1. El usuario accede a la aplicación.<br>2. El usuario pulsa en el botón de llamar. |
| **Escenario alternativo** | 1.1. El usuario no tiene permisos para la función llamar del dispositivo. |
| **Requisito asociado** | AM13 |

| ID | CU14 |
|---|---|
| **Caso de uso** | Notificaciones push |
| **Actores** | Administrador, coordinador y usuario |
| **Descripción** | El usuario podrá recibir notificaciones push en su dispositivo |
| **Precondiciones** | El administrador da de alta una nueva emergencia en la aplicación web |
| **Escenario principal** | 1. El servicio de notificaciones envía el mensaje.<br>2. El dispositivo recibe el mensaje e informa al usuario. |
| **Escenario alternativo** | 2.1. El dispositivo no tiene permisos para lanzar notificaciones, por lo que no se informa al usuario. |
| **Requisito asociado** | AM14 |

| ID | CU15 |
|---|---|
| **Caso de uso** | Añadir emergencia |
| **Actores** | Administrador y coordinador |
| **Descripción** | El usuario podrá dar de alta una nueva emergencia |
| **Precondiciones** | Estar autentificado |
| **Escenario principal** | 1. El usuario pulsa sobre la barra de navegación en el botón “Nueva emergencia”.<br>2. El usuario accede a la página de crear emergencia.<br>3. El usuario rellena los datos que se le piden y pulsa sobre el botón “Nueva emergencia”.<br>4. El sistema registra la nueva emergencia. |
| **Escenario alternativo** | |
| **Requisito asociado** | AWEB14 |

| ID | CU16 |
|---|---|
| **Caso de uso** | Solicitud de voluntario |
| **Actores** | Usuario no identificado, coordinador y administrador |
| **Descripción** | El usuario podrá crear una solicitud para ser voluntario en el sistema. |
| **Precondiciones** | Ninguna |
| **Escenario principal** | 1. El usuario pulsa en el botón de la barra de navegación “Voluntario”.<br>2. El usuario accede al formulario de solicitud.<br>3. El usuario rellena los datos necesarios.<br>4. El sistema registra la nueva solicitud. |
| **Escenario alternativo** | 3.1. El usuario introduce un email que ya fue registrado como solicitud o como usuario.<br>3.2. El sistema reconoce dicho email e informa al solicitante que ese email existe en el sistema. |
| **Requisito asociado** | AWEB18 |

| ID | CU17 |
|---|---|
| **Caso de uso** | Ver lista de solicitudes |
| **Actores** | Administrador y coordinador |
| **Descripción** | El usuario podrá ver la lista de solicitudes de voluntario pendientes de ser aceptadas |
| **Precondiciones** | Estar autentificado |
| **Escenario principal** | 1. El usuario pulsa en el desplegable “voluntarios” de la barra de navegación, en el botón “Lista de solicitudes”.<br>2. El usuario accede a la pantalla de solicitudes pendientes.<br>3. El usuario puede ver la lista de solicitudes. |
| **Escenario alternativo** | |
| **Requisito asociado** | AWEB29 |

| ID | CU18 |
|---|---|
| **Caso de uso** | Alta de voluntarios |
| **Actores** | Administrador y coordinador |
| **Descripción** | El usuario podrá dar de alta como usuario del sistema a las personas que hubiesen mandado solicitud previamente |
| **Precondiciones** | Estar autentificado y ver la lista de solicitudes (CU16) |
| **Escenario principal** | 1. El usuario busca la solicitud que cree oportuna.<br>2. El usuario pulsa en el botón de dar de alta.<br>3. El sistema registra el nuevo usuario con el rol de voluntario. |
| **Escenario alternativo** | 1.1. El usuario filtra por email las solicitudes.<br>2.1. Si el usuario es administrador, seleccionará el equipo al que se asignará al voluntario. |
| **Requisito asociado** | AWEB19 |

| ID | CU19 |
|---|---|
| **Caso de uso** | Ver lista de equipos |
| **Actores** | Administrador |
| **Descripción** | El usuario podrá ver una lista de todos los equipos del sistema. |
| **Precondiciones** | Estar autentificado |
| **Escenario principal** | 1. El usuario pulsa en el desplegable “Equipos” de la barra de navegación, en el botón “Lista de equipos”.<br>2. El usuario accederá a la pantalla de lista de equipos<br>3. El usuario podrá ver la lista de equipos |
| **Escenario alternativo** | |
| **Requisito asociado** | AWEB28 |

| ID | CU20 |
|---|---|
| **Caso de uso** | Ver equipo |
| **Actores** | Administrador |
| **Descripción** | El usuario podrá ver la lista de usuarios pertenecientes a un equipo. |
| **Precondiciones** | Estar autentificado |
| **Escenario principal** | 1. El usuario pulsa en el desplegable “Equipos” de la barra de navegación, en el botón “Lista de equipos”.<br>2. El usuario accede a la pantalla de lista de equipos del sistema<br>3. El usuario selecciona un equipo de la lista.<br>4. El usuario puede ver la lista de usuarios asociada a ese equipo |
| **Escenario alternativo** | |
| **Requisito asociado** | AWEB24 |

| ID | CU21 |
|---|---|
| **Caso de uso** | Ver equipo |
| **Actores** | Coordinador |
| **Descripción** | El usuario podrá ver la lista de usuarios pertenecientes a un equipo. |
| **Precondiciones** | Estar autentificado |
| **Escenario principal** | 1. El usuario pulsa en el desplegable “Usuarios” de la barra de navegación, en el botón “Lista de usuarios”.<br>2. El usuario accede a la pantalla de lista de usuarios asociada a su equipo |
| **Escenario alternativo** | |
| **Requisito asociado** | AWEB24 |

| ID | CU22 |
|---|---|
| **Caso de uso** | Activar/desactivar voluntarios |
| **Actores** | Administrador y coordinador |
| **Descripción** | El usuario podrá activar o desactivar un voluntario. |
| **Precondiciones** | Estar autentificado y ver la lista de usuarios asociado a un equipo (CU19) |
| **Escenario principal** | 1. El usuario pulsa en el desplegable “Usuarios” de la barra de navegación, en el botón “Lista de usuarios”.<br>2. El usuario accede a la pantalla de lista de usuarios<br>3. El usuario activa o desactiva un checkbox, sólo para los voluntarios |
| **Escenario alternativo** | |
| **Requisito asociado** | AWEB20 |

| ID | CU23 |
|---|---|
| **Caso de uso** | Crear equipo |
| **Actores** | Administrador |
| **Descripción** | El usuario podrá crear un equipo en el sistema |
| **Precondiciones** | Tiene que haber coordinadores sin equipo asociado |
| **Escenario principal** | 1. El usuario pulsa en el desplegable “Equipos” de la barra de navegación, en el botón “Añadir equipo”.<br>2. El usuario accede a la pantalla de añadir equipo<br>3. El usuario aporta los datos necesarios.<br>4. El sistema registra el equipo. |
| **Escenario alternativo** | |
| **Requisito asociado** | AWEB25 |

| ID | CU24 |
|---|---|
| **Caso de uso** | Filtrar información no oficial |
| **Actores** | Todos |
| **Descripción** | El usuario podrá aplicar un filtro sobre la lista de información no oficial. |
| **Precondiciones** | Seleccionar una emergencia |
| **Escenario principal** | 1.El usuario pulsa en el botón de filtrar.<br>2.El usuario selecciona el campo que desea filtrar y la condición que debe cumplir.<br>3.El usuario pulsa el botón “Filtrar”.<br>4. El sistema devuelve la lista de información no oficial dadas las condiciones elegidas |
| **Escenario alternativo** | 2.1. El usuario quiere cancelar el filtrado, y pulsa el botón “Limpiar filtro”.<br>2.2. El sistema devuelve toda la lista de información no oficial |
| **Requisito asociado** | AWEB27 |

## 2.6 Matriz de trazabilidad

Aquí se va a mostrar la matriz de trazabilidad asociada a los requisitos y casos de uso que voy a desarrollar en el proyecto.

| | CU1 | CU2 | CU3 | CU4 | CU5 | CU6 | CU7 | CU8 | CU9 | CU10 | CU11 | CU12 | CU13 | CU14 | CU15 | CU16 | CU17 | CU18 | CU19 | CU20 | CU21 | CU22 | CU23 | CU24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AM1 | X | | | | | | | | | | | | | | | | | | | | | | | |
| AM2 | | X | | | | | | | | | | | | | | | | | | | | | | |
| AM3 | | | X | | | | | | | | | | | | | | | | | | | | | |
| AM4 | | | | X | | | | | | | | | | | | | | | | | | | | |
| AM5 | | | | | X | | | | | | | | | | | | | | | | | | | |
| AM6 | | | | | | X | | | | | | | | | | | | | | | | | | |
| AM7 | | | | | | | X | | | | | | | | | | | | | | | | | |
| AM8 | | | | | | | | X | | | | | | | | | | | | | | | | |
| AM9 | | | | | | | | | X | | | | | | | | | | | | | | | |
| AM10 | | | | | | | | | | X | | | | | | | | | | | | | | |
| AM11 | | | | | | | | | | | X | | | | | | | | | | | | | |
| AM12 | | | | | | | | | | | | X | | | | | | | | | | | | |
| AM13 | | | | | | | | | | | | | X | | | | | | | | | | | |
| AM14 | | | | | | | | | | | | | | X | | | | | | | | | | |
| AWEB14 | | | | | | | | | | | | | | | X | | | | | | | | | |
| AWEB18 | | | | | | | | | | | | | | | | X | | | | | | | | |
| AWEB19 | | | | | | | | | | | | | | | | | | X | | | | | | |
| AWEB20 | | | | | | | | | | | | | | | | | | | | | | X | | |
| AWEB24 | | | | | | | | | | | | | | | | | | | | X | X | | | |
| AWEB25 | | | | | | | | | | | | | | | | | | | | | | | X | |
| AWEB27 | | | | | | | | | | | | | | | | | | | | | | | | X |
| AWEB28 | | | | | | | | | | | | | | | | | | | X | | | | | |
| AWEB29 | | | | | | | | | | | | | | | | | X | | | | | | | |

*Ilustración 3. Matriz de trazabilidad*

# 3 DISEÑO Y MODELADO DEL SOFTWARE

## 3.1 Modelo de datos

En la siguiente imagen se puede observar el diagrama de base de datos planteado.

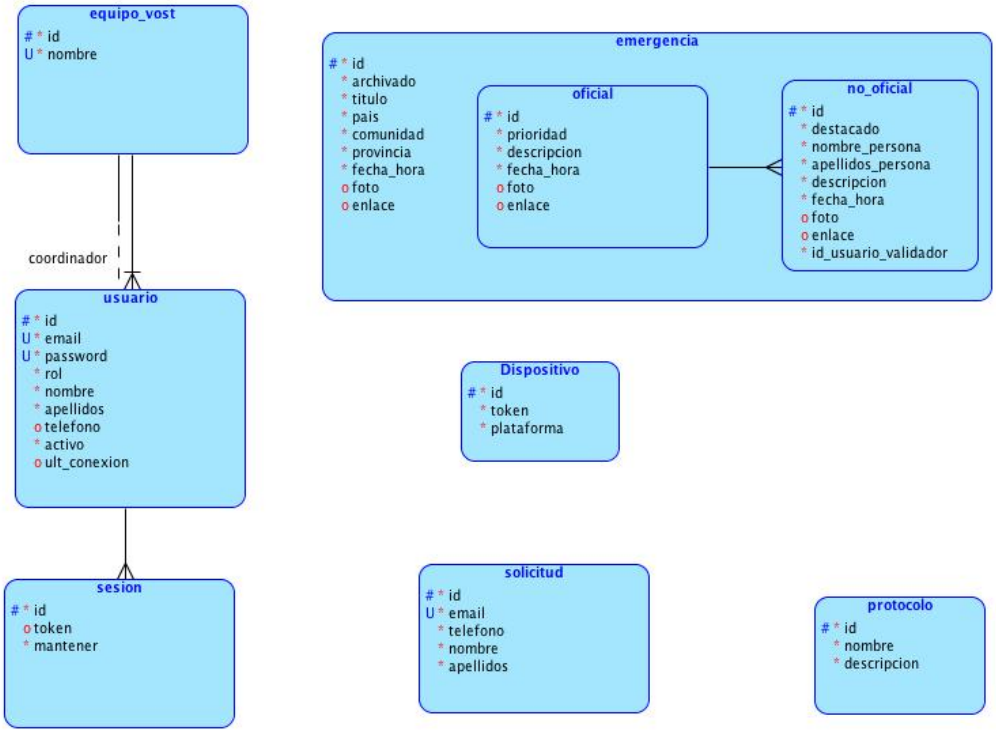


*Ilustración 4. Modelo de datos*

## 3.2 Diagrama de clases

A partir del modelo de datos mostrado anteriormente, se extrae el siguiente diagrama de clases:

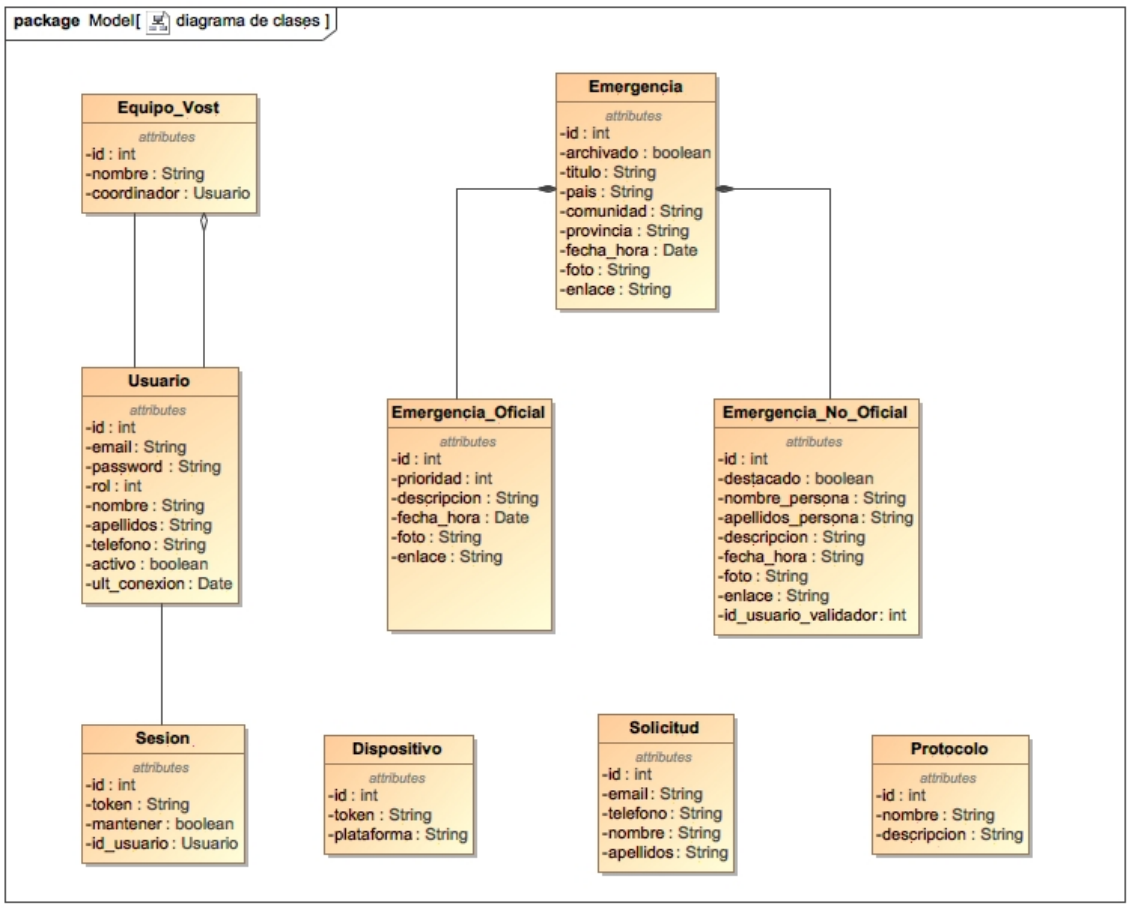


*Ilustración 5. Diagrama de clases*

## 3.3 Diagrama de despliegue

El sistema que se va desarrollar se va a basar en el siguiente diagrama de despliegue:

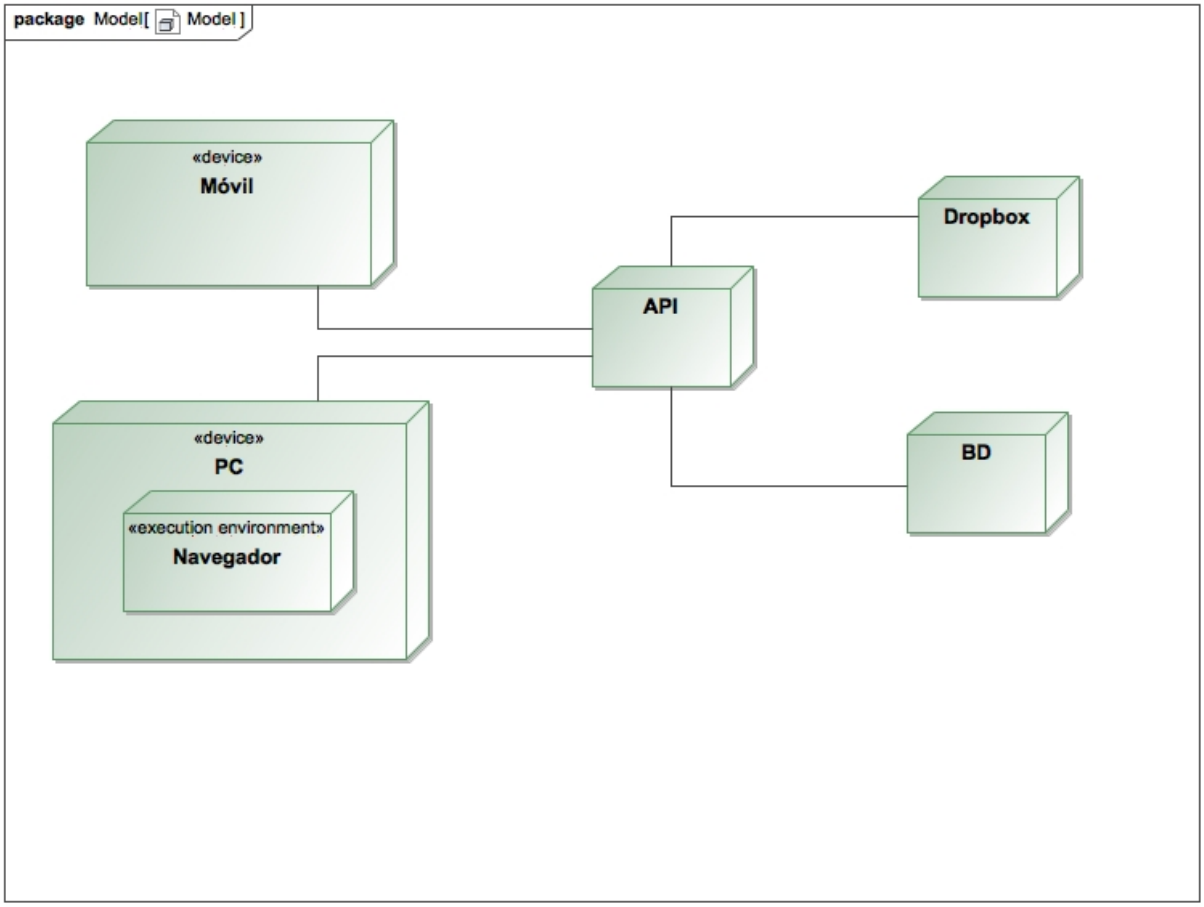


*Ilustración 6. Diagrama de despliegue*

## 3.4 Patrones de diseño utilizados

En este proyecto, se va a utilizar dos patrones de diseño: patrón MVC y patrón fachada.

El patrón Modelo-Vista-Controlador (MVC) es un patrón de diseño de arquitectura software que separa la interfaz de usuario, la lógica de la aplicación y los datos. Este patrón se va a usar tanto en la aplicación móvil como en la aplicación web.

El patrón fachada es un patrón estructural, orientado en la creación de objetos. Es el patrón bajo el que se desarrolla la API Rest.

# 4 IMPLEMENTACIÓN

En este apartado, se va a comentar la estructura del proyecto, así como las soluciones dadas para cada caso de uso, mostrando además qué se ha realizado en la API para llegar a esa solución. Se ha decidido utilizar una base de datos MySQL.

## 4.1 Estructura de la API

La API ha sido desarrollada como una Aplicación Java Web. La estructura de la misma es la siguiente:

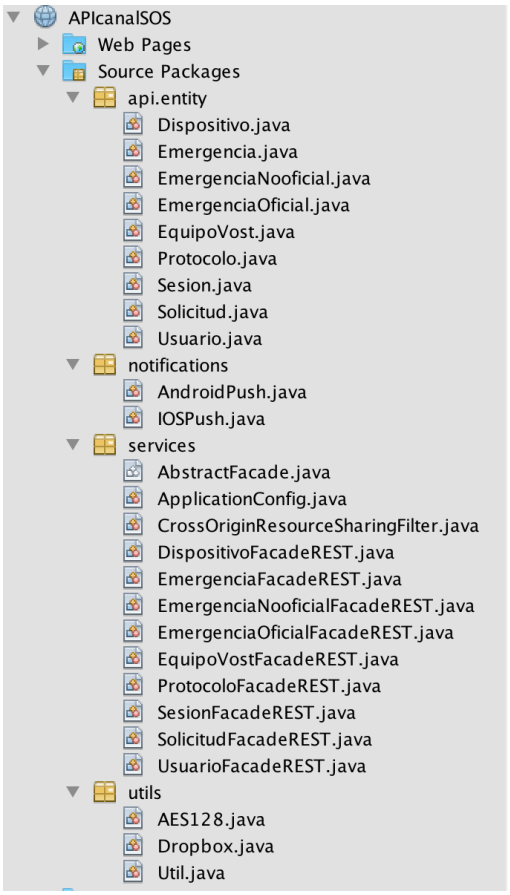


*Ilustración 7. Estructura de la API REST*

Se puede apreciar las entidades y los servicios, que son los que contienen los métodos que son llamados por las aplicaciones. Además, existen dos paquetes que se utilizan para diferentes fines. El paquete notifications contiene las clases usadas para mandar notificaciones push a los dispositivos que tengan instalada la aplicación. En el paquete utils, se encuentran las clases Util y AES128, que son utilizadas a la

hora de tratar la contraseña de los usuarios. Estas clases son reutilizadas de la asignatura “Seguridad en servicios y aplicaciones”. Además, se aprecia la clase Dropbox, la cual se encarga de realizar la comunicación con la API de Dropbox para ejecutar diferentes métodos.

Se va a destacar esta última clase. Lo primero que se debe mencionar de esta clase, es que se necesita un token generado a través de la consola de Dropbox. Sin ella, no se podrá establecer la conexión con la API.

La primera de las funcionalidades planteadas para el proyecto es la de subir ficheros. En realidad, para esta funcionalidad se han desarrollado dos métodos, uno para recibir por parámetro la imagen como array de bytes, y otro para recibir la imagen como String. Ambos reciben también como parámetro el path en el que se va a almacenar en Dropbox.

```
public FileMetadata upload(byte[] data, String fullPath) throws IOException, DbxException{
    FileMetadata response = null;

    ByteArrayInputStream in = new ByteArrayInputStream(data);

    response = client.files().upload(fullPath)
        .uploadAndFinish(in);


    return response;
}

public FileMetadata upload(String data, String fullPath) throws IOException, DbxException{
    FileMetadata response = null;

    InputStream in = new StringInputStream(data);

    response = client.files().upload(fullPath)
        .uploadAndFinish(in);


    return response;
}
```

*Ilustración 8. Métodos de subida de imagen de la clase Dropbox.java*

Posteriormente, en la clase se encuentra el método para cumplir la funcionalidad de borrar ficheros. Esta vez, recibe como parámetro únicamente el path que se debe borrar en Dropbox. Este path contiene también el nombre del fichero, pero se obvia para que se borre también la carpeta y no se vayan acumulando carpetas vacías.

```
public FolderMetadata delete(String fullPath) throws DbxException{
    FolderMetadata response = null;

    StringTokenizer st = new StringTokenizer(fullPath, "/");
    String root = st.nextToken();
    String folder = st.nextToken();

    String path = "/" + root + "/" + folder;

    response = (FolderMetadata) client.files().delete(path);


    return response;
}
```

*Ilustración 9. Método de borrar imágenes de la clase Dropbox.java*

Por último, se tenía que poder extraer enlaces temporales a esos ficheros almacenados en la nube. Para ello, se ha realizado un método al cual le pasamos el path exacto del cual se debe generar un enlace. Este método lo devuelve. Cabe destacar que, según la documentación de la API, este enlace funciona por 4 horas.

```
public String getLink(String path) throws DbxException{
    GetTemporaryLinkResult link = client.files().getTemporaryLink(path);
    return link.getLink();
}
```

*Ilustración 10. Método para obtener enlaces temporales en la clase Dropbox.java*

Mencionar también de la API que se apoya en varias librerías externas:

- Org-json: librería para leer y escribir json fácilmente
- Apns: librería para las notificaciones push de Apple para Java
- Dropbox-core: librería de Dropbox para Java
- Commons-codec: librería para codificar y descodificar en base 64

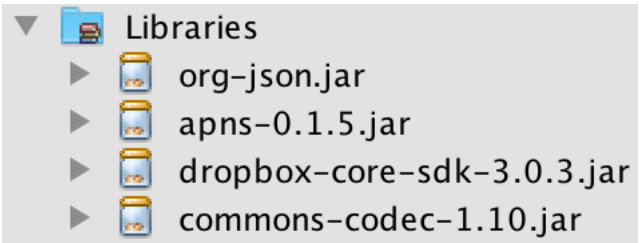


*Ilustración 11. Librerías utilizadas en la API REST*

## 4.2 Estructura de la aplicación móvil

Para empezar, decir que Ionic, a través de su interfaz de línea de comandos (CLI), te crea automáticamente un proyecto base, con o sin una plantilla preestablecida. Para ello, tendremos que ejecutar el comando:

```
ionic start canalSOS
```

*Ilustración 12. Comando para crear proyecto Ionic*

Cuando se ejecuta este comando (siempre en modo administrador), obtendremos un proyecto base como este:

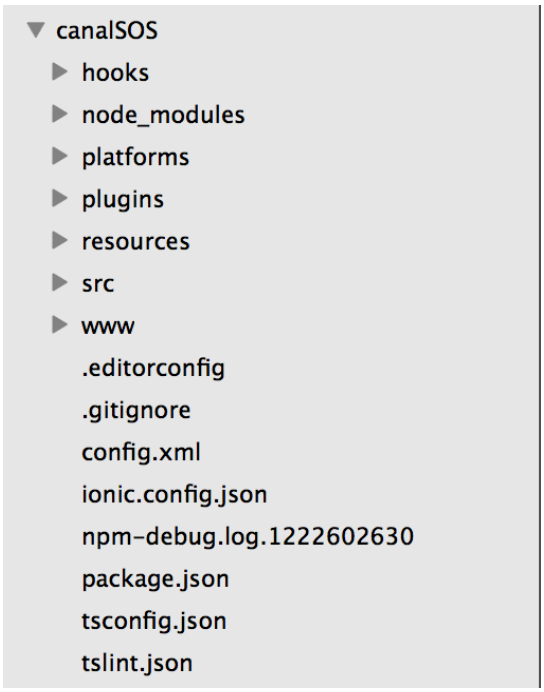


*Ilustración 13. Estructura de un proyecto Ionic*

Las carpetas más importantes son src, que es donde se genera todo el código necesario para correr la aplicación. Destacar también la carpeta platforms, que contiene el proyecto preparado para instalar la aplicación en las plataformas que se generen.


*Ilustración 14. Carpeta "src" de una aplicación Ionic*

Dentro de src, podemos diferenciar:

- app: contiene diferentes archivos que definen el contexto de la aplicación
- assets: contiene diferentes recursos, como la internacionalización o las imágenes
- models: contiene los modelos de datos de los diferentes objetos que se tratan en la aplicación
- pages: contiene las diferentes páginas que constituyen la aplicación, tanto la vista y el css como el controlador de las mismas
- providers: contiene los ficheros q llamarán a los servicios de la API. Habrá que introducir el dominio de la API a mano en la variable existente dentro de cada uno de los providers.

Destacar la facilidad que te ofrece Ionic para crear páginas o providers a través de su CLI, ya que te crea automáticamente los ficheros necesarios.

## 4.3 Casos de uso implementados

### 4.3.1 Caso de uso CU1

En este caso, nos encontramos que la finalidad es extraer de la base de datos la lista de emergencias que no estén archivadas. Para ellos, desde el código TypeScript del provider emergencia se hace una llamada a la API.

```
getEmergencias(){
    let response =
        this.http.get(this.base_url).map(res => res.json());
    return response;
}
```

*Ilustración 15. Método para obtener todas las emergencias en Ionic*

En la API, se accede a la base de datos y se devuelve la lista como un objeto JSON.

```
@GET
@Override
@Produces({"application/json"})
public List<Emergencia> findAll() {
    List<Emergencia> listaEmergencias = super.findAll();

    try {
        this.getLinks(listaEmergencias);
    } catch (DbxException ex) {
        Logger.getLogger(EmergenciaNooficialFacadeREST.class.getName()).log(Level.SEVERE, null, ex);
    }

    return listaEmergencias;
}
```

*Ilustración 16. Método para obtener todas las emergencias en API*

Al obtener la lista, posteriormente se accede a la API de Dropbox para obtener un enlace temporal asociado al path almacenado en la base de datos. Con este enlace, se podrá consumir la imagen previamente almacenada de esa emergencia. El enlace se almacenará en la base datos.

```
private void getLinks(List<Emergencia> listaEmergencias) throws DbxException{
    for(Emergencia emer: listaEmergencias){
        if(emer.getFoto() != null && !emer.getFoto().isEmpty()){
            String link = clientDropbox.getLink(emer.getFoto());
            emer.setEnlace(link);
        }
    }
}
```

*Ilustración 17. Método para obtener los enlaces temporales asociados a las imágenes de las emergencias*

Es importante saber que, si un enlace expira (como antes se mencionó en este documento), no habrá nunca problemas al estar renovándose el enlace temporal cada vez que se carga la lista.

Como resultado de este caso de uso, una vista de la aplicación sería el siguiente ejemplo:


*Ilustración 18. Ejemplo vista Inicio*

### 4.3.2 Casos de uso CU2 y CU3

Ambos casos de usos son muy similares. Para mostrar las listas, al seleccionar una emergencia, se crea una nueva página en la aplicación. Ionic trabaja este sistema de páginas como una pila, por lo que, si pulsamos hacia atrás, volveremos a la vista de lista de emergencias. Como la lista de información no oficial tiene más detalles a tener en cuenta, se va a detallar más en profundidad.

La finalidad de este caso de uso es la de obtener la lista de información no oficial ya validada. Para ello, desde el controlador de emergencia-detail, se llama al provider emergencia-no-oficial, donde se invoca al método de la API que obtiene de

la base de datos dicha lista, asociada al identificador de la emergencia seleccionada pasada por parámetro.

```
getPostByIdEmergencia(id_emergencia){
    let response =
      this.http.get(this.base_url + 'posts/' + id_emergencia).map(res => res.json());
    return response;
}
```

*Ilustración 19. Método para obtener la lista de información no oficial en Ionic*

En la API, se crea la sentencia SQL que pretende obtener la lista donde el id de la emergencia coincida con el parámetro, y que además no tenga almacenado ningún identificador de usuario del sistema en la columna id_validador, ya que esto indica que aún no ha sido valorado para admitir en el sistema esa información. Esta lista se devuelve ordenada según si la información es destacada y por fecha.

```
@GET
@Path("posts/{id_emergencia}")
@Produces({"application/json"})
public List<EmergenciaNooficial> posts(@PathParam("id_emergencia") Integer id_emergencia) {
    List<EmergenciaNooficial> listaPost;
    Emergencia e = emergenciaFacadeREST.find(id_emergencia);
    Query q;

    q = em.createQuery("SELECT e FROM EmergenciaNooficial e WHERE e.idEmergencia = :ID AND "
                       + "e.idUsuarioValidador != 0 ORDER BY e.destacado DESC, e.fechaHora DESC");
    q.setParameter("ID", e);

    listaPost = (List<EmergenciaNooficial>)q.getResultList();

    try {
        this.getLinks(listaPost);
    } catch (DbxException ex) {
        Logger.getLogger(EmergenciaNooficialFacadeREST.class.getName()).log(Level.SEVERE, null, ex);
    }

    return listaPost;
}
```

*Ilustración 20. Método para obtener la lista de información no oficial en la API*

Como ya pasó anteriormente en el caso de uso CU1, en esta lista pueden existir imágenes almacenadas en Dropbox de las cuales hay que obtener enlaces temporales para poder consumir esos recursos. Es por ello que se invoca al método getLinks, anteriormente descrito.

Mencionar que la lista de información oficial, el caso de uso CU1, consiste en obtener la lista de información oficial asociada al identificador de una cierta emergencia.

Un ejemplo de cada una de las listas obtenidas, ya mostradas en la aplicación, sería el siguiente:

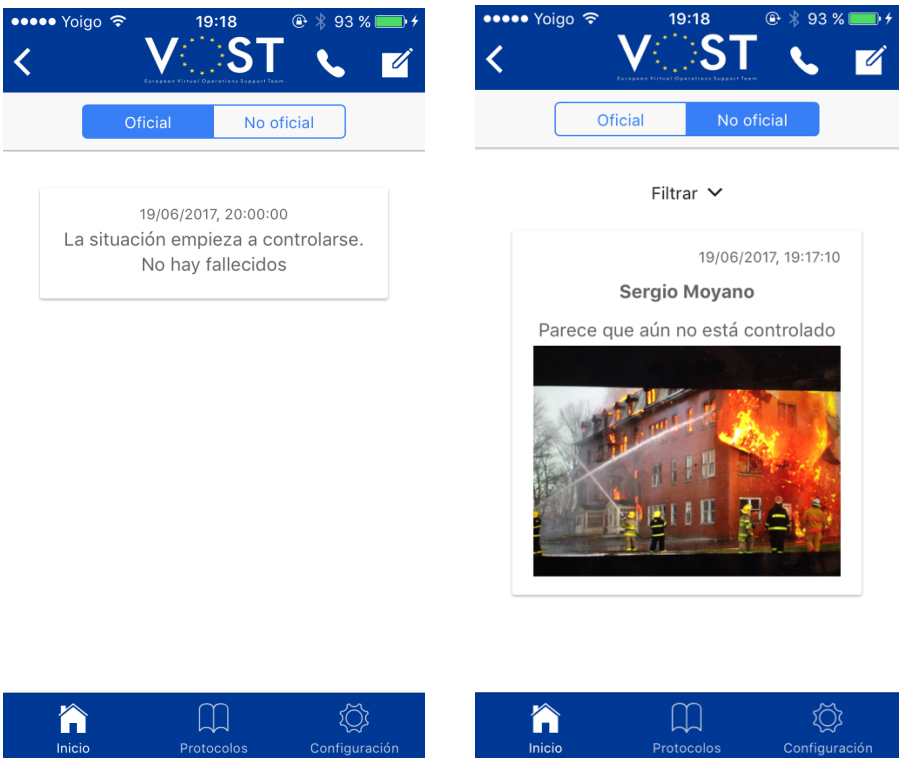


*Ilustración 21. Vistas de las listas de información oficial y no oficial, respectivamente*

### 4.3.3. Casos de uso CU4, CU5 y CU6

Estos tres casos de uso son muy similares a los que se han desarrollado anteriormente. La diferencia es el método del controlador que se invoca. En los anteriores, como lo que se pretendía era obtener la lista para mostrarla antes de cargar la vista HTML, se necesitaba obtener en el constructor. Esta vez, se necesita un método aparte que lo que haga sea recargar las listas. Como ejemplo, se adjunta imagen para recargar la lista de emergencias y la lista de información oficial:

```
doRefresh(refresher){
    this.emergenciaService.getEmergencias().subscribe(
        data => {
            if(data){
                this.emergencias = data;
            }
            refresher.complete();
        }
    );
}
```

```
doRefresh(refresher){
    this.emergenciaService.getEmergencias().subscribe(
        data => {
            if(data){
                this.emergencias = data;
            }
            refresher.complete();
        }
    );
}
```

*Ilustración 22. Método para refrescar la vista de las listas en Ionic*


*Ilustración 23. Vistas de las listas refrescándose*

### 4.3.4. Caso de uso CU7

Este caso de uso trata de insertar nueva información no oficial. Para hacerlo, se hace uso de un modal en el cual poder recoger la información necesaria para crearla. Se pide nombre, apellidos y una breve descripción.


*Ilustración 24. Vista de nueva información no oficial*

Cuando se recojan los datos, se llamará al provider emergencia-no-oficial para invocar al método de la API donde se inserta la nueva información, pasando un JSON con la información necesaria.

```
crearPost(nombre, apellidos, descripcion, id_emergencia){
    let body = JSON.stringify({"nombre": nombre, "apellidos": apellidos, "descripcion": descripcion, "id_emergencia": id_emergencia});
    let headers = new Headers();
    headers.append('Content-Type', 'application/json');
    let url = this.base_url + 'newpost';

    return this.http.post(url, body, {headers: headers}).map(res => res.json());
}
```

*Ilustración 25. Método para que llama a crear nueva información no oficial de la API*

En el caso de que al usuario se le olvide escribir algo en el formulario, saltará una alerta informándole de este error, y se volverá al modal para volver a intentar enviar información no oficial

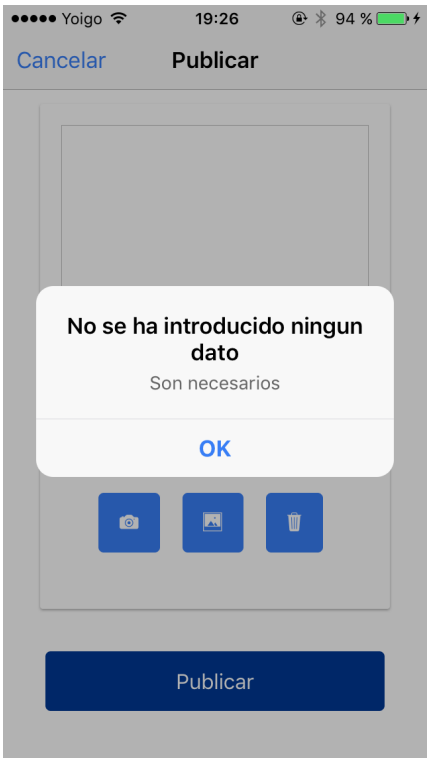


*Ilustración 26. Vista de formulario erróneo de nueva información no oficial*

En la API, se recoge el JSON para obtener los datos. Se crea un objeto EmergenciaNoOficial, insertándole los datos recogidos, y se registra el objeto en la base de datos. Es importante saber que esta información se almacenará como no validada, es decir, con el identificador de usuario validador igual a 0. Hasta que no se valide, permanecerá así y no se mostrará en la lista de información no oficial.

```
@POST
@Consumes({"application/json"})
@Produces({"application/json"})
@Path("newpost")
public EmergenciaNooficial newpost(String data){
    JSONObject json = new JSONObject(data);

    String nombre = (String)json.get("nombre");
    String apellidos = (String)json.get("apellidos");
    String descripcion = (String)json.get("descripcion");
    int id_emergencia = (int)json.getInt("id_emergencia");
    Emergencia e = emergenciaFacadeREST.find(id_emergencia);
    Date fecha = new Date();

    EmergenciaNooficial no_oficial = new EmergenciaNooficial();
    no_oficial.setDestacado(false);
    no_oficial.setNombrePersona(nombre);
    no_oficial.setApellidosPersona(apellidos);
    no_oficial.setDescripcion(descripcion);

    //Le ponemos 0 indicando de que aun no se ha validado. Cuando se valide, se almacenará el id del usuario que lo valide
    no_oficial.setIdUsuarioValidador(0);
    no_oficial.setFoto(null);
    no_oficial.setEnlace(null);
    no_oficial.setFechaHora(fecha);
    no_oficial.setIdEmergencia(e);

    create(no_oficial);

    return no_oficial;
}
```

*Ilustración 27. Método para crear nueva información no oficial en la API*

Cuando se inserte la información no oficial, se volverá a la vista de detalles de la emergencia anteriormente seleccionada, siendo informado con un toast (breve notificación) de que se ha insertado correctamente la información.


*Ilustración 28. Vista de confirmación de creación de la nueva información no oficial*

### 4.3.5. Caso de uso CU8

Este caso de uso extiende al anterior. Se produce en el caso de que un usuario desee adjuntar una imagen en la nueva información no oficial. Esto se realiza con el plugin Camera de Ionic. Para ello, deberá seleccionar el botón de hacer foto (a la izquierda de la botonera), o el botón de seleccionar foto de la galería (al centro de la botonera).

*Ilustración 29. Vista de la botonera para la imagen de crear información no oficial*

Si se pulsa en hacer foto, se llamará a la siguiente función dentro del controlador de nuevo-post.

```
takePhoto(){
    const options: CameraOptions = {
        quality: 100,
        destinationType: this.camera.DestinationType.FILE_URI,
        encodingType: this.camera.EncodingType.JPEG,
        saveToPhotoAlbum: true,
        mediaType: this.camera.MediaType.PICTURE
    }

    this.camera.getPicture(options).then((imageData) => {
        this.image = imageData;
        console.log('Imagen: ' + this.image);


    }, (err) => {
        // Handle error
        console.log(err)
    });
}
```

*Ilustración 30. Método para capturar foto con la cámara del teléfono*

Por el contrario, si lo que se desea es seleccionar una imagen de la galería, se llamará al siguiente método

```
takeFromGallery(){
    let cameraOptions = {
        quality: 100,
        destinationType: this.camera.DestinationType.FILE_URI,
        sourceType: this.camera.PictureSourceType.PHOTOLIBRARY,
        encodingType: this.camera.EncodingType.JPEG
    }

    this.camera.getPicture(cameraOptions).then( file_uri => {
        //Cogemos la imagen que hayamos seleccionado
        this.image = file_uri;
    }, (err) => {
        console.log(err)
    });
}
```

*Ilustración 31. Método para coger la imagen de la galería del teléfono*

La diferencia entre ambos se encuentra en las opciones que se le pasa al método getPicture del plugin Camera. La opción sourceType determina de dónde adquiere el recurso, si de la cámara o de la galería.

En el caso en el que el usuario se arrepienta de la imagen seleccionada, sólo tendrá que pulsar el botón de la papelera (a la derecha de la botonera).

Una vez se tenga preparada la imagen, se puede proceder a pulsar el botón “Enviar”. Entonces, el controlador se encarga de enviar, primero la información, y una vez recibida la respuesta de que se ha almacenado, enviar el archivo como String binario, el path donde se almacenará en Dropbox y el nombre del fichero.

```
this.nooficialService.crearPost(this.nombre, this.apellidos, this.descripcion, this.emergencia.id).subscribe(
    post => {
        const fileName = this.image.split('/').pop();
        const path = this.image.replace(fileName, '');
        var filePath = '/post_no_oficiales/emergencia_no_oficial_' + post.id;

        this.file.readAsBinaryString(path, fileName).then(
            (res) => {
                this.nooficialService.insertPathPhoto(post.id, filePath, fileName, res).subscribe(
                    (postEdited) => {
                        console.log('Se ha conseguido subir la foto');
                    }, (err) => {
                        console.log("No se ha podido insertar el path en la BD: " + err);
                    }
                );
            }, (err) => {
                console.log('Error parsing: ' + err);
            }
        );


        this.dismiss();
```

*Ilustración 32. Método para crear la información no oficial cuando se adjunta imagen*

En la API, se recoge el JSON con la información mencionada antes. Para poder enviar el archivo a la API de Dropbox, debemos obtener el array de bytes asociado al string recogido, en la codificación de caracteres recogido en el ISO-8859-1. Por último, el path del archivo en Dropbox se almacena en la base de datos, asociándolo a la información no oficial recién creada.

```
@POST
@Consumes({"application/json"})
@Produces({"application/json"})
@Path("editpost")
public EmergenciaNooficial editpost(String data){

    JSONObject json = new JSONObject(data);

    int id = (int)json.get("id");
    String filePath = (String)json.get("filePath");
    String fileName = (String)json.get("fileName");
    String imageAsText = (String)json.get("image");


    //Transformamos el string recibido al charset ISO-8859-1
    byte[] image = null;
    try {
        image = imageAsText.getBytes("ISO-8859-1");
    } catch (UnsupportedEncodingException ex) {
        Logger.getLogger(EmergenciaNooficialFacadeREST.class.getName()).log(Level.SEVERE, null, ex);
    }

    String fullPath = filePath + "/" + fileName;

    EmergenciaNooficial e = find(id);

    try {
        clientDropbox.upload(image, fullPath);
        e.setFoto(fullPath);
        edit(e);
    } catch (IOException | DbxException ex) {
        Logger.getLogger(EmergenciaNooficialFacadeREST.class.getName()).log(Level.SEVERE, null, ex);
    }

    return e;
}
```

*Ilustración 33. Método para almacenar imagen adjunta en la API*

Aunque se ha mencionado antes, destacar que, si la imagen no se consigue subir a Dropbox, la información no oficial será registrada igualmente.

### 4.3.6. Casos de uso CU9 y CU24

Este caso de uso trata de filtrar la lista de información no oficial. Para llevar a cabo esto, se necesita saber sobre qué columna de la base de datos se va a filtrar, y las condiciones que debe cumplir. Se decidió que se podía filtrar por nombre completo (nombre y apellidos), por descripción, y por tiempo.

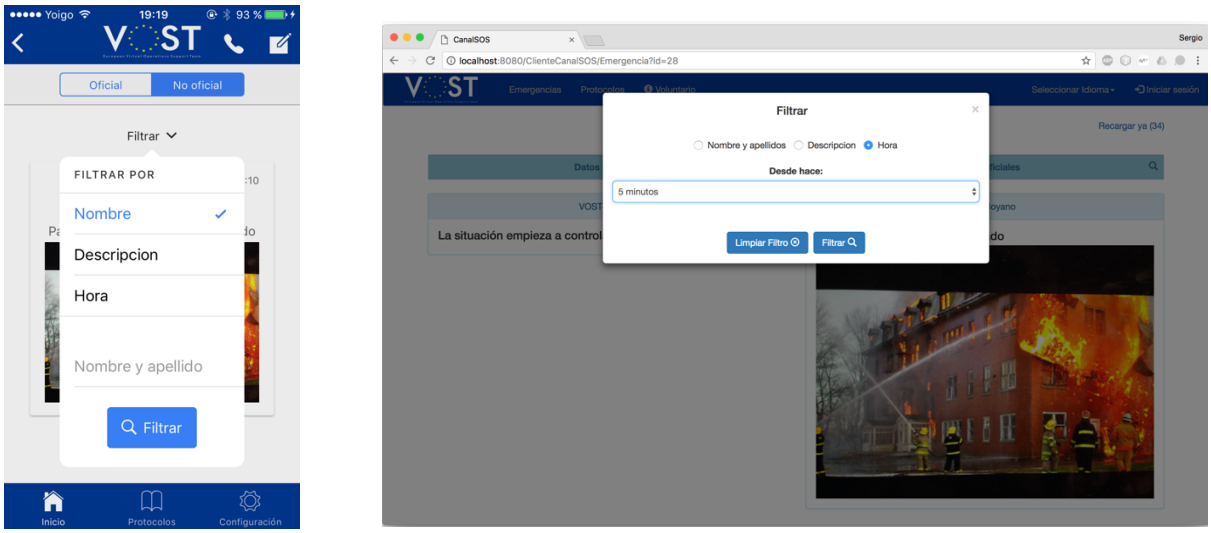


*Ilustración 34. Vistas de filtrado de emergencia no oficial en la app móvil y en la app web*

Al filtrar por tiempo, se va a mostrar un desplegable con varios intervalos de tiempo (5 minutos, 15 minutos, 1 hora, etc.).

Si el usuario está en la aplicación web, se arrepiente y desea cancelar el filtrado, puede pulsar en el botón “Limpiar filtro” y se cargará la lista completa.

Cuando se pulsa en el botón filtrar, se llama a los controladores, los cuales son los encargados de recoger el valor de la columna y la condición.

```
filtrar(){
    var data;
    if(this.filterBy == 'nombrePersona'){
        data = this.nombre;
    }else if(this.filterBy == 'descripcion'){
        data = this.descripcion;
    }else{
        console.log('Hora actual: ' + this.horaActual);
        this.horaFiltrada.setTime(this.horaActual.getTime());
        this.horaFiltrada.setHours(this.horaActual.getHours(), this.horaActual.getMinutes() - this.hora, this.horaActual.getSeconds(), 0);
        console.log('Hora fitrada: ' + this.horaFiltrada);
        data = this.horaFiltrada;
    }

    this.nooficialService.filterPostsByAttributeAndValue(this.filterBy, data, this.id_emergencia).subscribe(
        data => {
            this.viewCtrl.dismiss(data);
        }, err => {

        }
    );
}
```

*Ilustración 35. Método de filtrar en Ionic*

Según la columna seleccionada, la condición a pasar era diferente. Una vez que tenemos los datos necesarios para pasar a la API, se produce la llamada al método filtrar. Ésta nos devolverá la lista que cumpla dichas condiciones.

```
@POST
@Path("filter")
@Consumes({"application/json"})
@Produces({"application/json"})
public List<EmergenciaNooficial> filter(String data) throws ParseException {
    JSONObject json = new JSONObject(data);

    String columna = (String)json.get("columna");
    String value = (String)json.get("value");
    int id_emergencia = (int)json.getInt("id_emergencia");
    Emergencia e = emergenciaFacadeREST.find(id_emergencia);

    List<EmergenciaNooficial> listaPost = new ArrayList<EmergenciaNooficial>();
    Query q;

    String query = null;

    boolean cadena_caracteres = true;
    switch(columna){
        case "nombrePersona":
            query = "SELECT e FROM EmergenciaNooficial e WHERE "
                    + "(e.nombrePersona LIKE :VALUE OR e.apellidosPersona LIKE :VALUE) "
                    + "AND e.idEmergencia = :ID AND e.idUsuarioValidador != 0 ORDER BY e.destacado DESC, e.fechaHora DESC";
            break;
        case "descripcion":
            query = "SELECT e FROM EmergenciaNooficial e WHERE e.descripcion LIKE :VALUE "
                    + "AND e.idEmergencia = :ID AND e.idUsuarioValidador != 0 "
                    + "ORDER BY e.destacado DESC, e.fechaHora DESC";
            break;
        case "hora":
            query = "SELECT e FROM EmergenciaNooficial e WHERE e.fechaHora >= :HORA_FILTRO "
                    + "AND e.idEmergencia = :ID AND e.idUsuarioValidador != 0 ORDER BY e.destacado DESC, e.fechaHora DESC";
            cadena_caracteres = false;
            break;
        default:

    }

    if(query != null){
        q = em.createQuery(query);

        if(cadena_caracteres){
            q.setParameter("VALUE", "%"+value+"%");
        }else{
            //Convertimos la fecha que se tiene que filtrar a un formato reconocido por JAVA
            SimpleDateFormat formatterFechaFiltro = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSSXXX");
            Date fechaFiltro = formatterFechaFiltro.parse(value);

            q.setParameter("HORA_FILTRO", fechaFiltro);

        }

        q.setParameter("ID", e);

        listaPost = (List<EmergenciaNooficial>)q.getResultList();
    }

    try {
        this.getLinks(listaPost);
    } catch (DbxException ex) {
        Logger.getLogger(EmergenciaNooficialFacadeREST.class.getName()).log(Level.SEVERE, null, ex);
    }

    return listaPost;
}
```

*Ilustración 36. Método filtrar información no oficial de la API*

Este método acarreó bastantes problemas cuando se seleccionaba filtrar por tiempo, ya que el formato por defecto de los objetos Date de Java y el formato de TypeScript eran distintos, y al final tuvimos que darle el formato exacto de Java al objeto Date de TypeScript “horaFiltrada” para que funcionase.

```
this.horaFiltrada = new Date("2017-05-26T13:00:00+02:00");
```

*Ilustración 37. Definición del formato de la fecha actual en Ionic*

### 4.3.7. Caso de uso CU10

Este caso de uso trata de listar los protocolos almacenados en el sistema. Para eso, se hace una consulta a la API desde el provider protocolo, obteniendo la lista completa de protocolos.

Cuando la API devuelva la lista, quedará una vista como la siguiente en la aplicación:

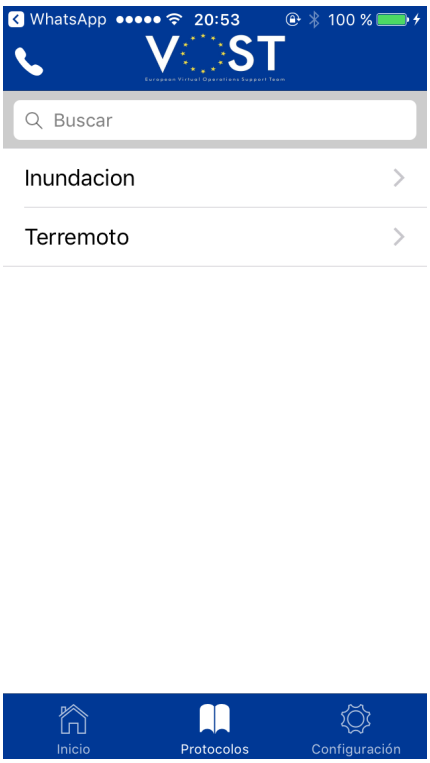


*Ilustración 38. Vista de la lista de protocolos*

### 4.3.8 Caso de uso CU11

Este caso de uso extiende al inmediatamente anterior. Como se puede comprobar en la imagen de ejemplo de la vista de vista de lista de protocolos, se ha implementado un buscador en la parte superior. Con lo que escribamos ahí, el

controlador pasará al provider el texto, y de aquí, al método de la API que se encarga de filtrar, devolviendo la lista que cumpla con la condición impuesta.

```
filter(){
    if(this.searchText != ''){
        this.protocolosService.filterProtocols(this.searchText).subscribe(
            data => {
                this.protocolos = data;
            }
        );
    }else{
        this.cargarProtocolos();
    }
}
```

*Ilustración 39. Método filtrar del controlador de protocolos en Ionic*

Como se puede comprobar, si no se escribe texto, simplemente se carga toda la lista de todos los protocolos.

En el provider, se llama al método filtrar pasando el texto escrito por parámetro

```
filterProtocols(protocol){
  let response =
     this.http.get(this.base_url + 'filter/' + protocol).map(res => res.json());
    return response;
}
```

*Ilustración 40. Método que llama a la función filtrar protocolos de la API*

Para finalizar, aquí se puede ver el método por el que se obtiene la lista de protocolos filtrada:

```
@GET
@Path("filter/{protocolo}")
@Produces({"application/json"})
public List<Protocolo> filter(@PathParam("protocolo") String protocolo){
    Query q;
    List<Protocolo> p;

    if(!protocolo.equals("")){
        q = em.createQuery("SELECT p FROM Protocolo p WHERE p.nombre LIKE :NAME");
        q.setParameter("NAME", "%"+protocolo+"%");

        p = (List<Protocolo>)q.getResultList();

        return p;
    }else{
        return this.findAll();
    }
}
```

*Ilustración 41. Método de obtener la lista de protocolos de la API*

### 4.3.9 Caso de uso CU12

Aquí se trata de seleccionar un cierto protocolo en la vista descrita anteriormente y ver su contenido asociado. Cuando se seleccione, la aplicación mostrará una nueva página, de la cual se podrá ir hacia atrás para volver a la lista de protocolos.

```
protocoloDetail(protocolo){
    this.navCtrl.push(ProtocoloDetailPage, {
        protocolo: protocolo
    });
}
```

*Ilustración 42. Método del controlador lanzado tras seleccionar un protocolo de la lista*

De esta vista, destacar que está preparado para mostrar texto con caracteres especiales de HTML, como la negrita, las listas, etc.

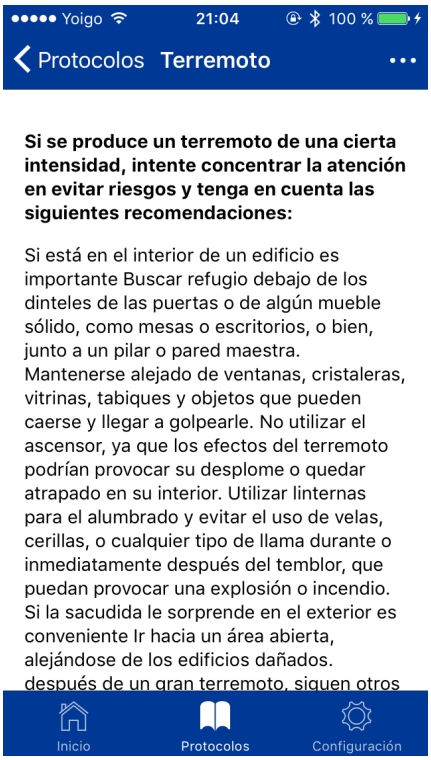


*Ilustración 43. Vista de un protocolo de actuación*

### 4.2.10 Caso de uso CU13

Este caso de uso relata la funcionalidad para poder llamar de forma directa al teléfono de emergencias 112. Es importante mencionar que este botón debía estar accesible desde todas las vistas de la aplicación, por lo que se decidió incluirlo en la barra de navegación. Para ello, se ha usado el plugin de Ionic llamado “Call Number”. La función que se encarga de esto es la siguiente:

```
call112(){
    this.callNumber.callNumber("112", true)
    .then(() => console.log('Aplicación llamada lanzada'))
    .catch(() => console.log('Error cargando aplicación llamada'));
}
```

*Ilustración 44. Método encargado de lanzar la llamada al 112*

El resultado en la aplicación tras pulsar el botón es el siguiente:


*Ilustración 45. Vista de llamada al 112*

### 4.3.11 Casos de uso CU14 y CU15

Estos dos casos de uso dependen uno del otro. Primero ocurre el caso CU15, cuando un administrador o un coordinador de equipo da de alta una nueva

emergencia. Cuando esto ocurre, se produce una llamada a las clases encargadas de producir las notificaciones push.

Lo primero, es rellenar el formulario de nueva emergencia, que es como el que se ve a continuación:

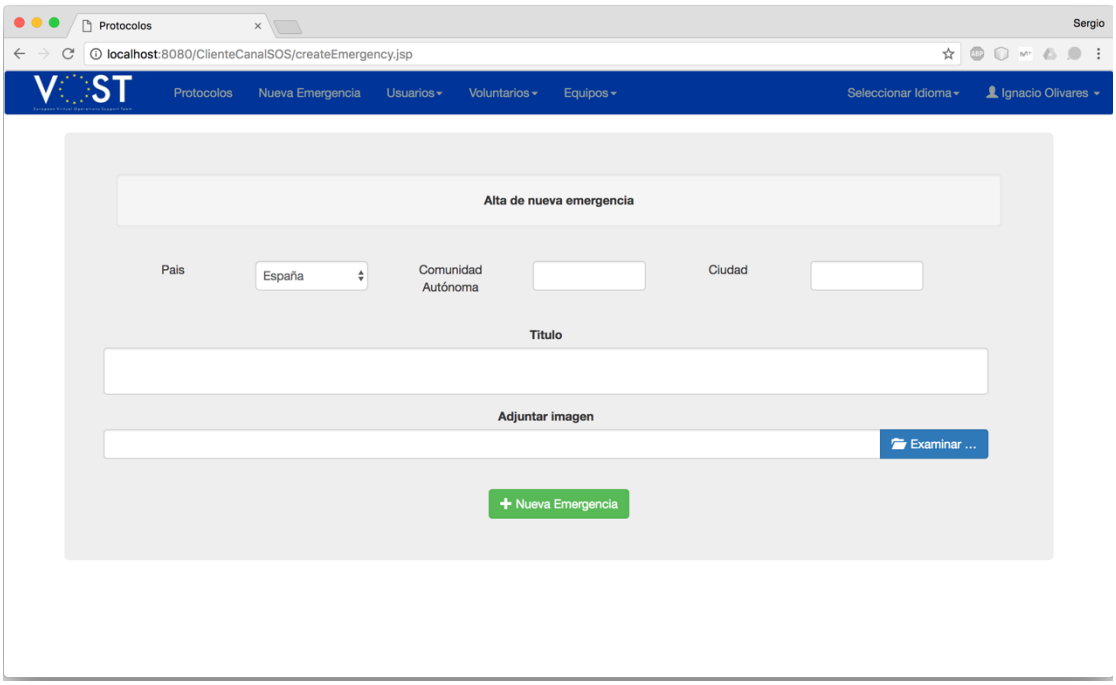


*Ilustración 46. Vista de formulario de nueva emergencia*

Al rellenar el formulario de nueva emergencia, el Servlet recoge los datos introducidos:

```
String pais = (String)request.getParameter("pais");
String comunidad = (String)request.getParameter("comunidad");
String provincia = (String)request.getParameter("provincia");
String titulo = (String)request.getParameter("titulo");
String foto = null;
String enlace = null;
Part file = request.getPart("file");
```

*Ilustración 47. Recogida de información del formulario de nueva emergencia*

Una vez que tenemos los datos, necesitamos transformar el fichero obtenido en String en base 64, para poder enviárselo a la API para que lo procese:

La llamada a la API se hace en dos pasos. Primero, creamos el objeto emergencia pasándole toda la información anterior para almacenarlo en la base de

datos. La fecha que se pasa es la fecha completa (hora, minuto y segundo incluido) en la que se llama a este Servlet.

```
Date fecha_hora = new Date();

Emergencia emergencia = new Emergencia();
emergencia.setPais(pais);
emergencia.setComunidad(comunidad);
emergencia.setProvincia(provincia);
emergencia.setTitulo(titulo);
emergencia.setFechaHora(fecha_hora);
emergencia.setArchivada(false);
emergencia.setFoto(foto);
emergencia.setEnlace(enlace);

EmergencyClient client = new  EmergencyClient();
GenericType<Emergencia> genericType = new GenericType<Emergencia>() {};
Emergencia aux = client.createEmergency(emergencia, genericType);
```

*Ilustración 48. Creación de la nueva emergencia*

Una vez hecho esto, se comprueba si dentro del formulario se incluye alguna imagen. Si es así, se llama al método de editar emergencia, que es la que se encarga de subir la imagen a Dropbox dado un path y dicho fichero.

```
if(file.getSize() != 0){
    InputStream in = file.getInputStream();
    ByteArrayDataOutputStream out = new ByteArrayDataOutputStream();
    byte[] imagen = new byte[Integer.parseInt(String.valueOf(file.getSize()))];
    while(in.read(imagen) != -1){
        out.write(imagen);
    }
    String imagen64String = Base64.encodeBase64String(imagen);

    JSONObject json = new JSONObject();
    json.put("id", aux.getId());
    json.put("imagen", imagen64String);
    json.put("fileName", file.getSubmittedFileName());

    aux = client.editEmergency(json.toString(), genericType);
}
```

*Ilustración 49. Comprobación y posterior subida de imagen*

Cuando finalice la subida del fichero, se enviarán las notificaciones push por parte del servidor, pasándole la emergencia creada. Finalmente, el sistema enviará al usuario a la vista de lista de emergencias.

```
client.sendNotifications(aux);

response.sendRedirect("emergencias");
```

*Ilustración 50. Envío de notificaciones push*

Por parte de la API, lo primero que se hace es crear una emergencia en la base de datos.

```
@POST
@Path("create")
@Consumes({"application/json"})
@Produces({"application/json"})
public Emergencia createEmergency(Emergencia entity) {

    em.persist(entity);
    em.flush();

    return entity;
}
```

*Ilustración 51. Registro en la base de datos de la nueva emergencia*

Para terminar con el caso CU15, una vez que se termina de insertar en la base de datos la nueva emergencia, el sistema devuelve su identificador para así poder guardar el path asociado a la foto que se va a subir también en el mismo método en la columna path.

```
@POST
@Path("edit")
@Consumes({"application/json"})
public Emergencia editEmergency(String data) throws DbxException, IOException {
    JSONObject json = new JSONObject(data);
    int id = (int)json.get("id");
    String imagen = (String)json.get("imagen");
    String fileName = (String)json.get("fileName");

    //Transformamos el string recibido al charset ISO-8859-1
    byte[] image = Base64.decodeBase64(imagen);

    String path = "/emergencias/emergencia_" + id + "/" + fileName;

    Emergencia e = find(id);

    e.setFoto(path);

    clientDropbox.upload(image, path);

    return e;

}
```

*Ilustración 52. Almacenamiento del path en la base de datos y subida a Dropbox*

Una vez que se haya almacenado la imagen asociada, debemos llamar a los métodos responsables de las notificaciones push. Antes de nada, se comprueba si la emergencia que se ha pasado como parámetro tiene una imagen asociada en Dropbox. Esto se hace para obtener un enlace temporal o no. A continuación, se obtienen en todos los dispositivos almacenados en la base de datos, ordenados por plataforma (iOS y Android). Con el token asociado a cada uno de ellos y un mensaje (en este caso, el título de la nueva emergencia), se llaman a las clases IOSPush y AndroidPush respectivamente.

En estas clases existe un solo método, denominado sendPushNotification, el cual nos permite realizar la función demandada.

```
public static void sendPushNotification(String device_token, String message) throws Exception {
    ApnsService service =
            APNS.newService()
                    .withCert(PATH_TO_P12_CERT, CERT_PASSWORD)
                    .withSandboxDestination()
                    .build();

    String payload = APNS.newPayload()
            .alertBody(message)
            .sound("default")
            .build();

    service.push(device_token, payload);
    System.out.println("The message has been hopefully sent...");
}
```

*Ilustración 53. Método que envía notificaciones push a dispositivos Apple*

La diferencia entre ambas clases, es que cada una accede a diferentes servicios. En Android se hace a través de la herramienta de Google, Firebase. Mientras tanto, Apple lo hace a través de su propio servicio de notificaciones, denominado APNS. En Android, debemos pasar algunos parámetros como JSON para poder configurar como nosotros deseamos la notificación push, como la prioridad, si lleva una imagen que deseemos, etc.

```
public static void sendPushNotification(String device_token, String title, String message, String enlace)

    JSONObject data = new JSONObject();
    data.put("title", title);
    data.put("message", message);
    data.put("priority", 2);
    data.put("visibility", 1);
    data.put("content-available", "1");
    if(enlace != null){
        data.put("image", enlace);
    }

    JSONObject pushMessage = new JSONObject();
    pushMessage.put("to", device_token);
    pushMessage.put("data", data);

    System.out.println(pushMessage.toString());

    // Create connection to send FCM Message request.
    URL url = new URL("https://fcm.googleapis.com/fcm/send");
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setRequestProperty("Authorization", "key=" + SERVER_KEY);
    conn.setRequestProperty("Content-Type", "application/json");
    conn.setRequestMethod("POST");
    conn.setDoOutput(true);

    // Send FCM message content.
    OutputStream outputStream = conn.getOutputStream();
    outputStream.write(pushMessage.toString().getBytes());
```

*Ilustración 54. Método para enviar notificaciones push a dispositivos Android*

Terminado el caso CU15, llegamos al caso CU14, que es recibir notificaciones push en el dispositivo móvil. Para ello, en la aplicación móvil, lo primero que se hace es registrar el token del dispositivo en el sistema para que se puedan enviar mensajes a dicho dispositivo.

```
if (!this.platform.is('cordova')) {
    console.warn("Push notifications not initialized. Cordova is not available - Run in physical device");
    return;
}
const options: PushOptions = {
    android: {
        senderID: "883625243011",
        sound: true,
        vibrate: true,
        forceShow: true
    },
    ios: {
        alert: "true",
        badge: false,
        sound: "true"
    },
    windows: {}
};

const pushObject: PushObject = this.push.init(options);

pushObject.on('registration').subscribe((data: any) => {

    //TODO - send device token to server

    //Definimos la plataforma en la que estamos
    let platform = '';
    if(this.platform.is('ios')) {
        platform = 'ios';
    }else if(this.platform.is('android')){
        platform = 'android';
    }

    //Enviamos el token al servidor para registrar el dispositivo en nuestra BD
    this.dispositivoService.insert(data.registrationId, platform).subscribe(
        data => {
            console.log("device token ->", data.registrationId);
        }, err => {

        }
    );

});
```

*Ilustración 55. Registro del token generado del dispositivo en la base de datos*

Una vez registrado, se añade el evento “notification” que actúa tanto cuando la aplicación está en primer plano como cuando está en el segundo plano.

```
pushObject.on('notification').subscribe((data: any) => {
    console.log('message', data.message);
    //if user using app and push notification comes
    console.log("Push notification clicked");

});
```

*Ilustración 56. Definición del comportamiento del dispositivo al lanzarse el evento notification*

Por último, se define el comportamiento si se produce un error en una notificación push.

```
pushObject.on('error').subscribe(error => console.error('Error with Push plugin', error));
```

*Ilustración 57. Definición del comportamiento del dispositivo cuando ocurra un error en la notificación*

### 4.3.12 Caso de uso CU16

Este caso de uso trata de registrar una nueva solicitud para ser voluntario. Este caso es para cualquier tipo de usuario. Para registrar, se ha planteado que el usuario debe rellenar un pequeño formulario, almacenando su correo electrónico, su nombre y apellidos, y su número de teléfono. Se ha puesto este último como obligatorio para que, ya que los administradores y los coordinadores son los que lo darán de alta al voluntario, estos puedan contactar más fácilmente cuando se llegue al caso.


*Ilustración 58. Vista de formulario de nueva solicitud de voluntario*

Un aspecto importante que se ha incluido es el “captcha” de Google. El motivo de la inclusión es que, al no tener que iniciar sesión, cualquier robot podría enviar formularios sin parar. Con esto, evitamos una sobrecarga producido por un ataque dirigido al sistema. Además, se comprueba que el correo electrónico introducido al menos tiene un formato para tal fin.

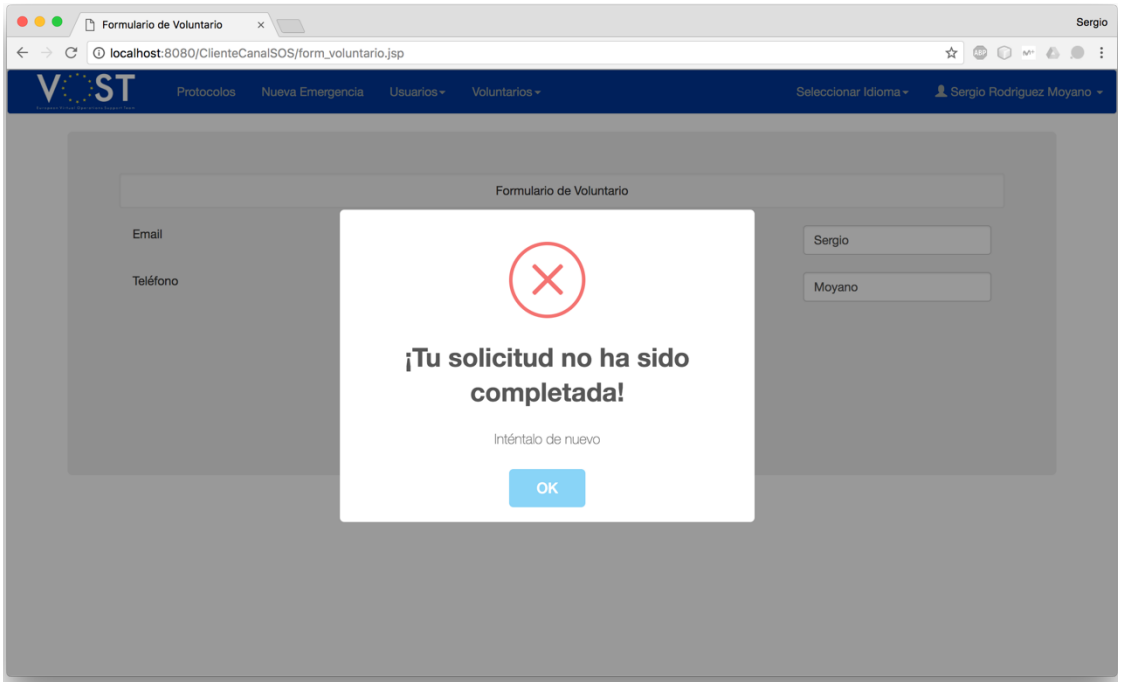


*Ilustración 59. Vista de error al introducir un email que ya existe en la base de datos*

En el Servlet, lo primero que se hace es comprobar si es una petición AJAX. En el caso de que no lo sea, simplemente dirige a la página principal. Si no, comienza el registro. A continuación, se recogen los datos introducidos.

```
if(!ajax){
    response.sendRedirect("home");
}else{

    RequestClient reqClient = new RequestClient();
    GenericType<Solicitud> genericType = new GenericType<Solicitud>() {};

    String email = (String)request.getParameter("email");
    String telefono = (String)request.getParameter("telefono");
    String nombre = (String)request.getParameter("nombre");
    String apellidos = (String)request.getParameter("apellidos");
```

*Ilustración 60. Obtención de información del formulario de nueva solicitud*

A partir de ahí, se crea un objeto de tipo Solicitud con dichos datos. Pero antes de confirmar el registro de la solicitud, debemos comprobar que no exista ni ninguna solicitud, ni ningún usuario existente en el sistema con ese mismo email. Si no existe ningún usuario ni solicitud, se registra.

```
JSONObject data = new JSONObject();
data.put("email", email);
Solicitud aux = reqClient.findByEmail(data.toString(), genericType);

GenericType<Usuario> genericTypeUser = new GenericType<Usuario>() {};
UserClient uClient = new UserClient();
Usuario u = uClient.recover(genericTypeUser, email);

boolean ok = false;
if(aux == null && u == null){
    ok = true;
    reqClient.create(s);
}
```

*Ilustración 61. Comprobación de email existente y registro de nueva solicitud*

Para finalizar, se devuelve un JSON con una variable boolean, para informar al usuario si se ha creado el registro de la solicitud o no.

```
JSONObject output = new JSONObject();
output.put("ok", ok);

response.setContentType("application/json");
response.setCharacterEncoding("UTF-8");
response.getWriter().write(output.toString());
```

*Ilustración 62. Envío de JSON a la vista con información de si se había creado la solicitud o no*

### 4.3.13 Caso de uso CU17

Se trata de mostrar en una vista la lista de solicitudes registradas. En el Servlet, primero se comprueba si se tiene los privilegios para llegar a esta vista. Si es así, comienza el proceso de carga de la lista.

```
HttpSession session = request.getSession();

String token = (String)session.getAttribute("token");
int rol = session.getAttribute("rol") == null ? -1 : (Integer)session.getAttribute("rol");

if(token==null || rol<2 ){
    response.sendRedirect("home");
}else{
```

*Ilustración 63. Comprobación de privilegios para acceder a la lista de solicitudes*

Tras esto, simplemente se hace una llamada a la API para almacenar dicha lista y pasársela a la vista.

```
RequestClient reqClient = new RequestClient();

GenericType<List<Solicitud>> genericType = new GenericType<List<Solicitud>>() {};
List<Solicitud> solicitudesList = reqClient.findAll(genericType);

request.setAttribute("solicitudesList", solicitudesList);
```

*Ilustración 64. Obtención de solicitudes*

Con todo esto, un ejemplo de una vista sería el siguiente:

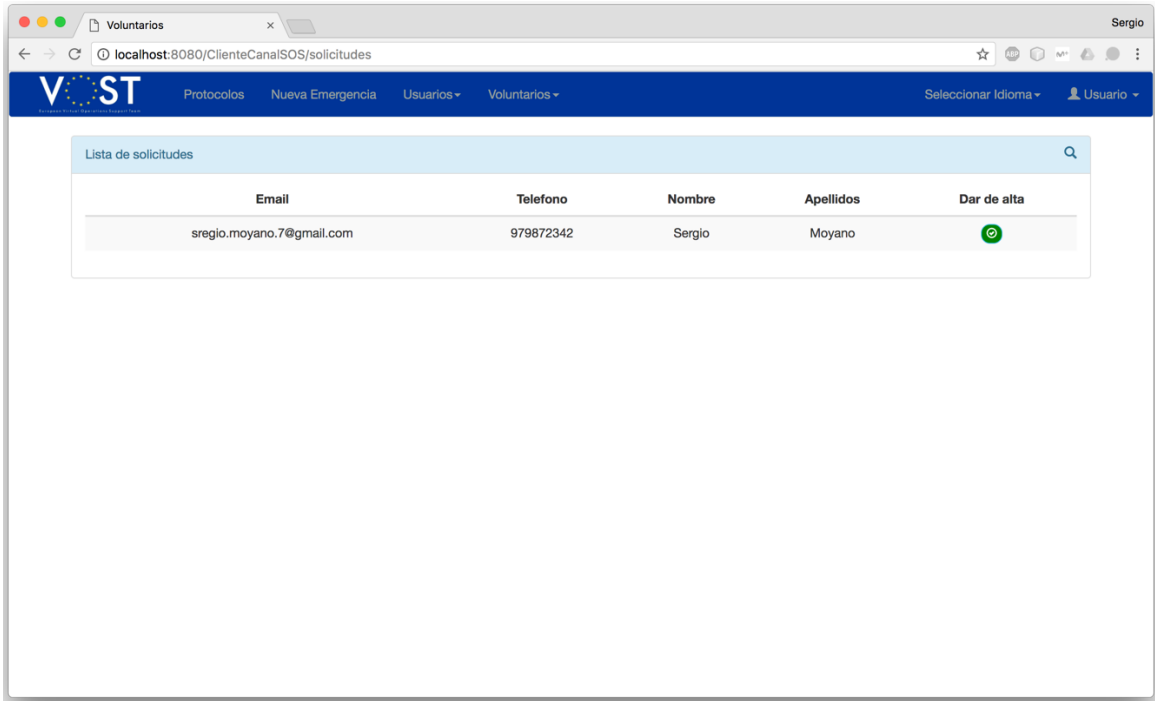


*Ilustración 65. Vista de lista de solicitudes de voluntarios*

### 4.3.14 Caso de uso CU18

Con este caso de uso, se pretende dar de alta como usuarios a las solicitudes que interesen. Para eso, se debe acceder a la lista de solicitudes del caso de uso CU17. Una vez allí, y como podemos comprobar en la imagen anterior, tenemos un botón verde en la columna de la derecha, que tiene la funcionalidad de dar de alta. Para que la este proceso sea lo más rápido posible, se ha implementado un buscador que filtra la lista por email.

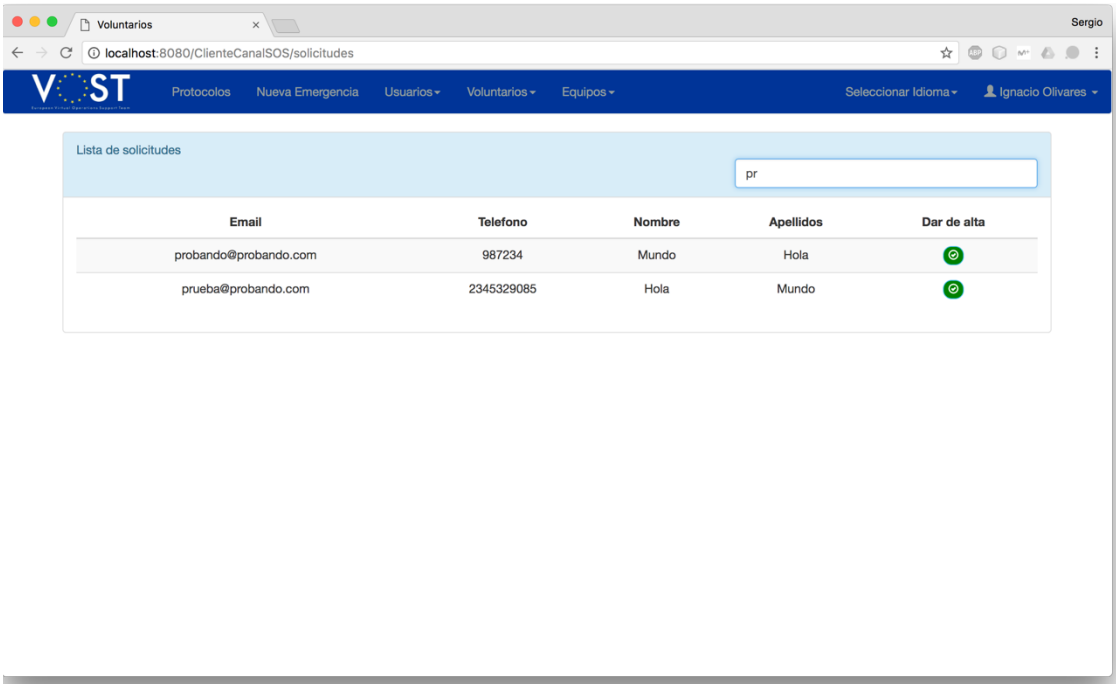


*Ilustración 66. Vista de lista de solicitudes de voluntarios filtrada*

Es una función AJAX que llama al Servlet FilterVolunteers. Aquí se recoge el texto escrito en el buscador y se obtiene la lista que coincida con dicho valor.

```
if(!ajax){
    response.sendRedirect("home");
}else{

    String email = (String)request.getParameter("email");
    JSONObject data = new JSONObject();
    data.put("email", email);

    RequestClient reqClient = new RequestClient();
    GenericType<List<Solicitud>> genericType = new GenericType<List<Solicitud>>() {};
    List<Solicitud> listaSolicitudes = reqClient.findAllByEmail(data.toString(), genericType);

    JSONObject output = new JSONObject();
    output.put("solicitudes", listaSolicitudes);

    response.setContentType("application/json");
    response.setCharacterEncoding("UTF-8");
    response.getWriter().write(output.toString());
}
```

*Ilustración 67. Función AJAX encargada del filtrado de solicitudes*

Según el usuario que haya iniciado sesión, el proceso será diferente.

Si el usuario que ha iniciado sesión es un coordinador de equipo, este sólo tendrá que pulsar en el botón de dar de alta y se le asignará automáticamente a su propio equipo.

Sin embargo, si el usuario es el administrador, tendrá que seleccionar a qué equipo destina al nuevo voluntario. Para ello, se hace uso de un Modal de Bootstrap, como se puede ver a continuación:

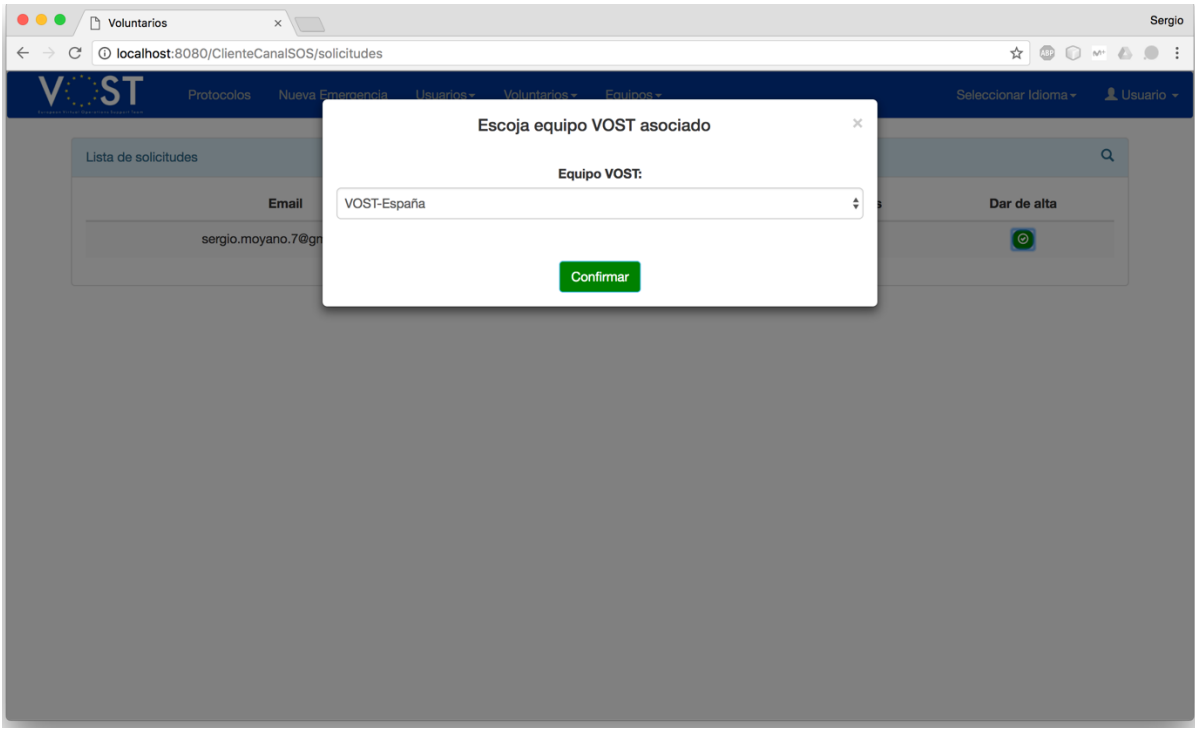


*Ilustración 68. Vista de dar de alta una solicitud de la lista para el administrador*

En el Servlet, la primera comprobación es si es una petición AJAX. Si no lo es, se redirige a la página principal.

En caso contrario, se comienza recogiendo los valores pasados en la llamada, que son los identificadores de la solicitud y del equipo. Con el identificador de la solicitud, se extrae el objeto de la base de datos. Así, tenemos los datos necesarios de un usuario para darle de alta.

```
String id_solicitud = (String)request.getParameter("id_solicitud");
String id_equipo = (String)request.getParameter("id_equipo");

//Primero obtenemos el objeto de la base de datos
RequestClient reqClient = new RequestClient();
GenericType<Solicitud> genericType = new GenericType<Solicitud>() {};
Solicitud solicitud = reqClient.find(genericType, id_solicitud);
```

*Ilustración 69. Obtención de parámetros para dar de alta a un voluntario*

Se crea un JSON con toda la información del usuario. En este JSON se inserta también el identificador del equipo al que se asociará el nuevo usuario. Pero para ello, se debe distinguir lo que contiene el valor pasado desde la llamada AJAX. Si el valor está vacío, significa que el usuario que se está dando de alta es un coordinador, por lo que se debe buscar los datos del usuario dado el token de la sesión. En caso contrario, se pasará el identificador del equipo seleccionado en la vista. Para finalizar, se inserta a través de la API el nuevo usuario y se elimina la solicitud.

```
JSONObject u = new JSONObject();
u.put("email", solicitud.getEmail());
u.put("rol", "0");
u.put("name", solicitud.getNombre());
u.put("surname", solicitud.getApellidos());
u.put("phone", solicitud.getTelefono());
if(id_equipo.equals("")){
    //Obtenemos el usuario de la sesion para obtener el equipo asociado
    HttpSession session = request.getSession();
    String token = (String)session.getAttribute("token");
    JSONObject json = new JSONObject();
    json.put("token", token);

    GenericType<Usuario> genericTypeUser = new GenericType<Usuario>() {};
    Usuario user = uClient.getUserFromSesion(json.toString(), genericTypeUser);

    u.put("equipoVost", String.valueOf(user.getIdEquipo()));
}else{
    u.put("equipoVost", id_equipo);
}


uClient.newuser(u.toString());

//Borramos la solicitud
reqClient.remove(id_solicitud);
```

*Ilustración 70. Creación del nuevo usuario a partir de la solicitud seleccionada*

Por último, se obtienen las demás solicitudes que quedan pendientes y se devuelven en un JSON.

### 4.3.15 Caso de uso CU19

Esta funcionalidad de listar equipos sólo está disponible para el administrador. Consiste en pulsar sobre la barra de navegación en el desplegable Equipos, y dentro de él, en el botón Lista de equipos.

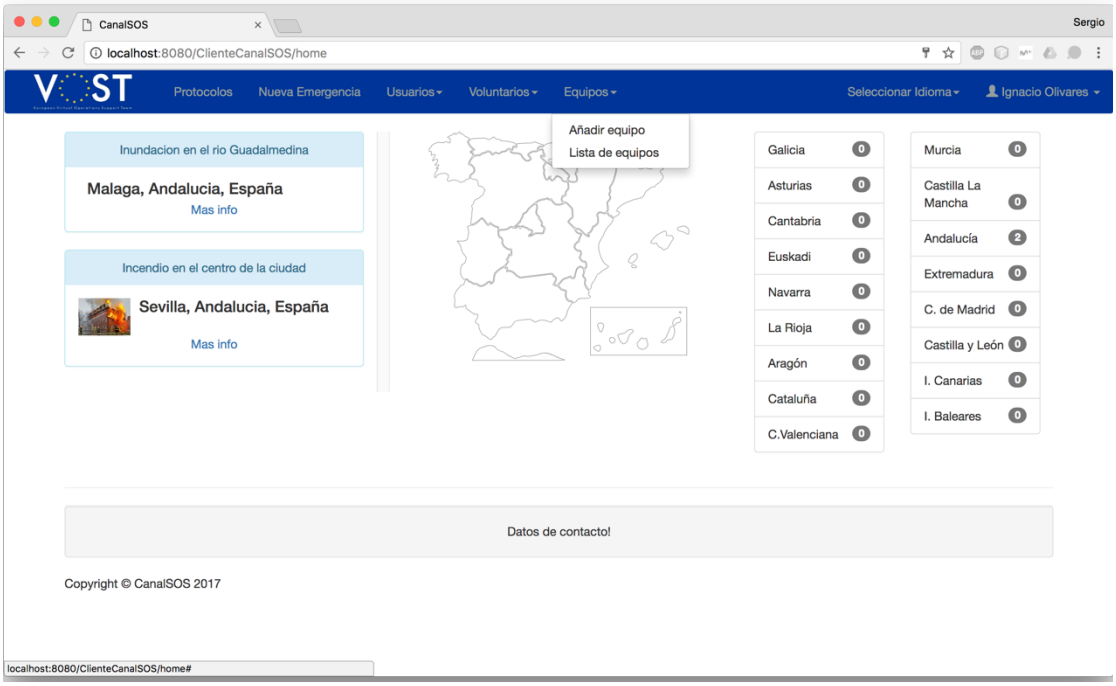


*Ilustración 71. Vista de la opción de la barra de navegación que se debe seleccionar*

El Servlet se encarga, primero de la comprobación de los privilegios, y posteriormente, de comunicarse con la API para obtener todos los equipos registrados.

```
HttpSession session = request.getSession();

String token = (String)session.getAttribute("token");
int rol = session.getAttribute("rol") == null ? -1 : (Integer)session.getAttribute("rol");

if(token==null || rol<3 ){
    response.sendRedirect("home");
}else{

    TeamVostClient eClient = new TeamVostClient();

    GenericType<List<EquipoVost>> genericType = new GenericType<List<EquipoVost>>() {};
    List<EquipoVost> equiposList = eClient.findAll(genericType);

    request.setAttribute("equiposList", equiposList);

    RequestDispatcher rdp;
    rdp = this.getServletContext().getRequestDispatcher("/teamList.jsp");
    rdp.forward(request, response);
}
```

*Ilustración 72. Servlet de lista de equipos*

Finalmente, el resultado del proceso sería una vista como la siguiente:

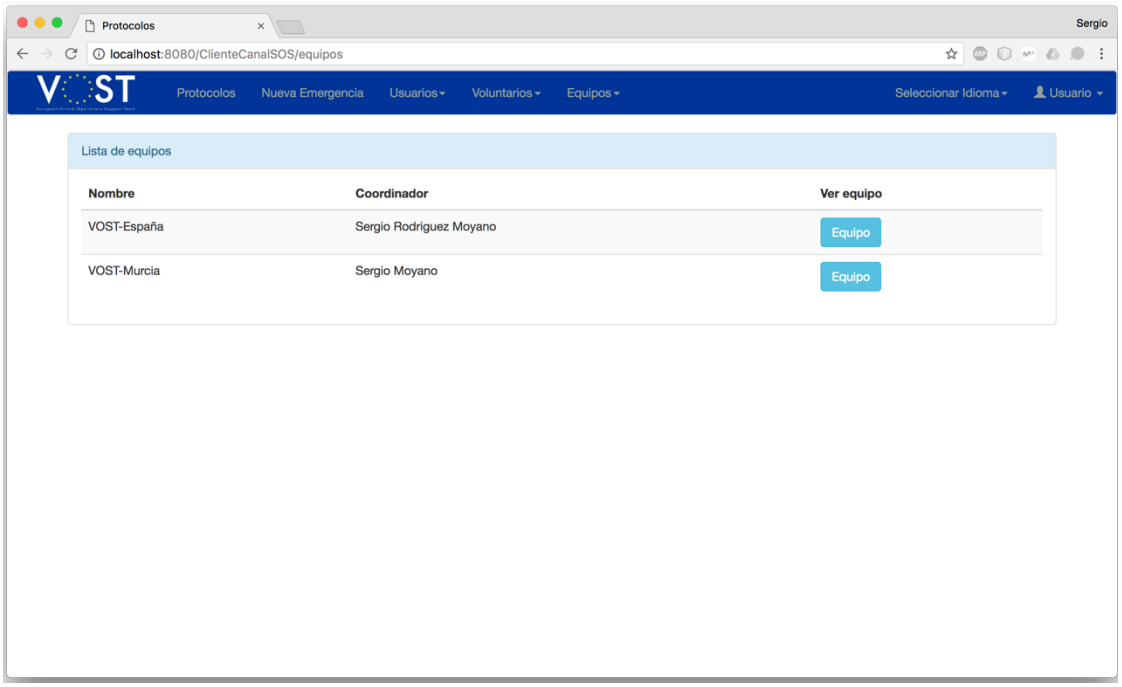


*Ilustración 73. Vista de lista de equipos*

### 4.3.16 Casos de uso CU20 y CU21

La funcionalidad de ver equipo contempla dos casos de uso. Uno en el que es el coordinador de quipo el que visualiza su propio equipo, y otro en el que es el administrador el que elige qué equipo quiere observar. En ambos casos de uso, se ha aprovechado la funcionalidad realizada por mi compañero de ver la lista de usuarios del sistema.

Ambos casos de uso se han resuelto en un mismo Servlet. En él comprobamos si se ha pasado un parámetro, el identificador del equipo asociado. Si es que sí, es que viene de la vista de lista de equipos. Si no, es que viene de pulsar en Lista de usuarios de la barra de navegación.

```
if(id_equipo!=null){
    //Viene de lista de equipos
    usuariosList = pClient.findUsersByTeam(genericType, id_equipo);
    voluntariosList = pClient.findVolunteersByTeam(genericType, id_equipo);
}else{
```

*Ilustración 74. Comprobación de la vista desde donde se invoca al Servlet*

Se ha planteado que, para el primer caso (CU20), el administrador tiene que dirigirse a ver la lista de equipos, y seleccionar el equipo que quiere visualizar, como se puede apreciar en la *Ilustración 73*. Una vez que haya pulsado en el botón asociado al equipo seleccionado, se llamará al Servlet de lista de usuarios pasando el identificador del equipo, el cual terminará por mostrar al usuario la lista de usuarios asociado a ese equipo.

Para llegar al segundo caso (CU21), se pulse en la barra de navegación en el botón Usuarios, y dentro de este, en el botón Lista de usuarios.

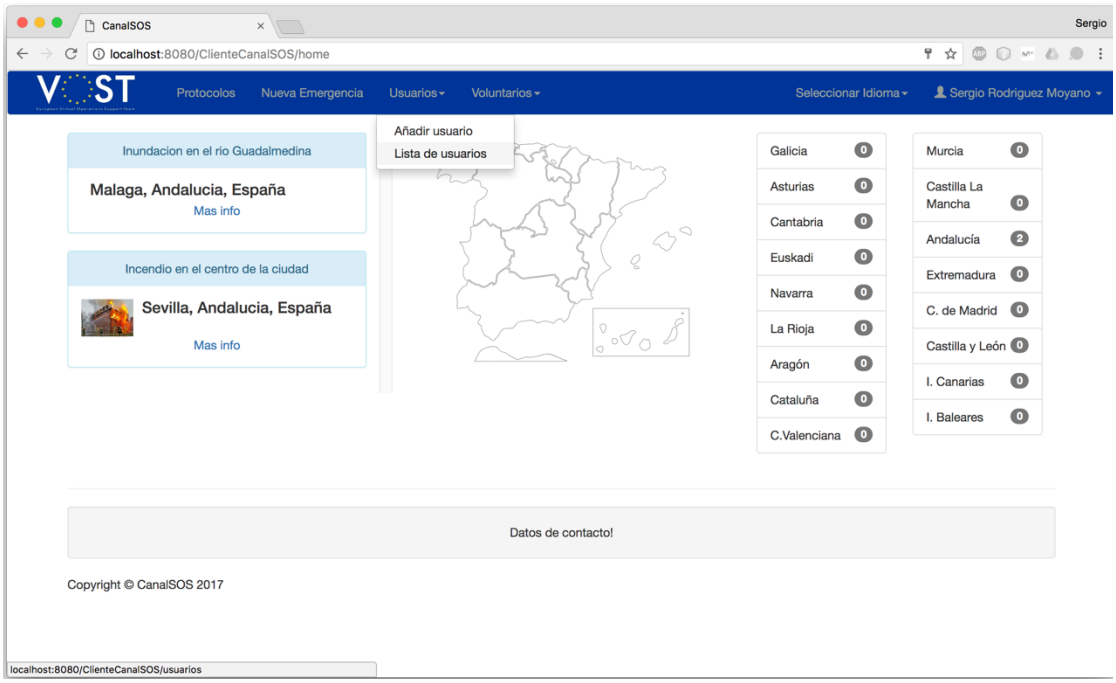


*Ilustración 75. Vista de la opción de la barra de navegación que se debe seleccionar*

Esto llamará al Servlet, el cual se encargará de analizar si el rol del usuario que tiene iniciada la sesión es el de coordinador o el de administrador. Si es de administrador, cargará la lista completa de usuarios del sistema, caso de uso desarrollado por mi compañero Ignacio. Si es de coordinador, se busca su identificador de equipo asociado y se filtra a través de él. Así se visualizará la lista de usuarios asociada a su propio equipo.

```
if(rol == 2){
    JSONObject json = new JSONObject();
    json.put("token", token);
    GenericType<Usuario> genericTypeUser = new GenericType<Usuario>() {};
    Usuario u = pClient.getUserFromSesion(json.toString(), genericTypeUser);

    usuariosList = pClient.findUsersByTeam(genericType, String.valueOf(u.getIdEquipo()));
    voluntariosList = pClient.findVolunteersByTeam(genericType, String.valueOf(u.getIdEquipo()));
}else{
    usuariosList = pClient.findUsers(genericType);
    voluntariosList = pClient.findVolunteers(genericType);
}
```

*Ilustración 76. Comprobación del rol del usuario almacenado en sesión*

Un ejemplo de la vista resultante sería:

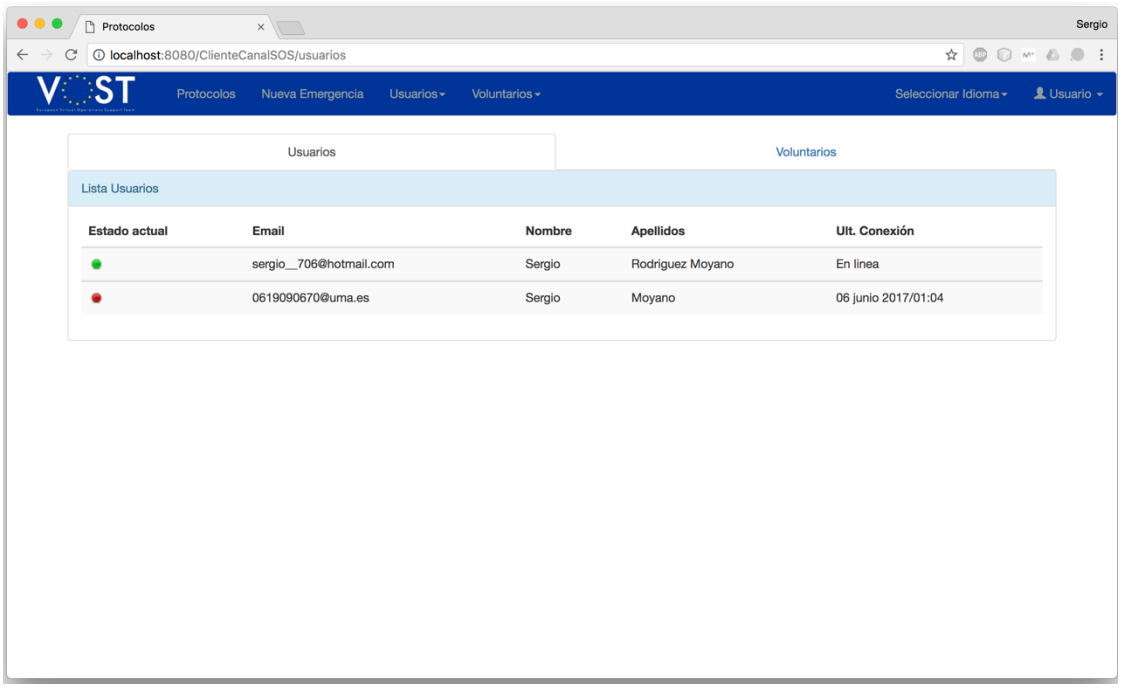


*Ilustración 77. Vista de la lista de usuarios*

### 4.3.17 Caso de uso CU22

Este caso de uso se basa en ver la lista de usuarios, ya sea del sistema (cuando el usuario es administrador) o del propio equipo (cuando el usuario es coordinador).

Cuando se llega a la vista, se puede ver que existen dos pestañas, una con los usuarios que son miembros fijos de los equipos (coordinadores y miembros) y otra con los voluntarios. Se selecciona la pestaña de voluntarios y se queda una vista como la siguiente:

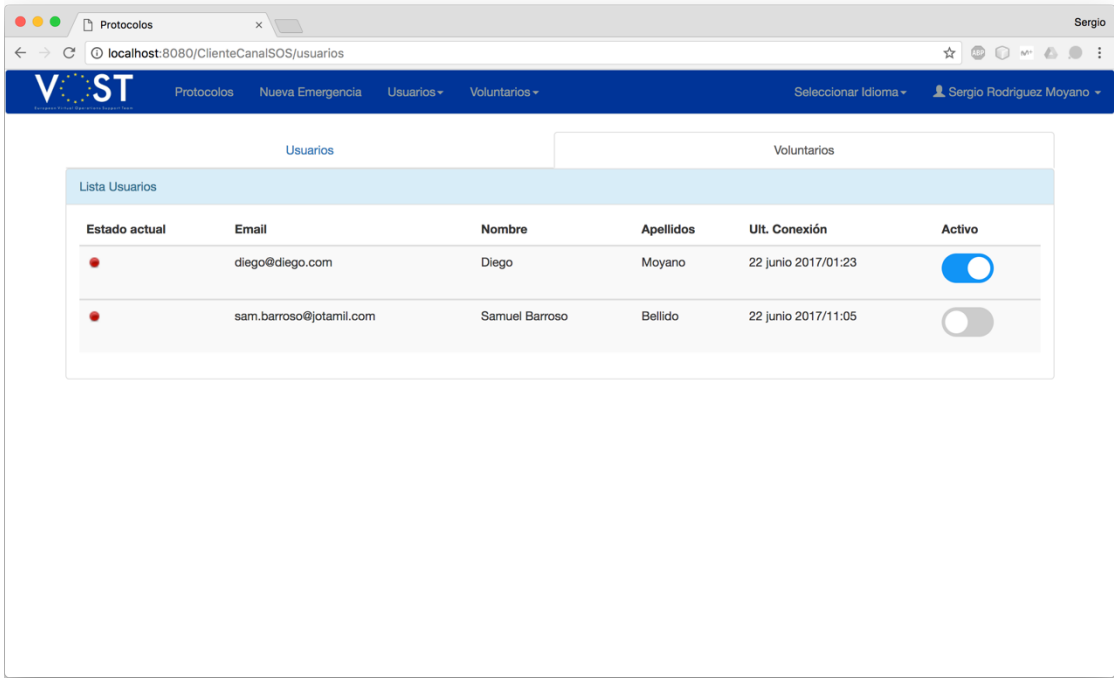


*Ilustración 78. Vista de lista de voluntarios*

Se ha implementado un “switch toggle” para que activar y desactivar sea muy intiutivo. Para ello se han incluido clases de CSS que no son propias de Bootstrap, pero su página web te documenta las clases que se deben incluir en un checkbox.

Cuando se pulsa sobre el toggle, se manda una petición AJAX al Servlet ActiveVolunteer con el identificador del usuario y el valor del toggle. Con esto, se llama a la API para cambiar en la base de datos el atributo activo del usuario asociado al identificador. De esta forma, el usuario estará habilitado (o no) para iniciar sesión en el sistema.

```
boolean ajax = "XMLHttpRequest".equals(
                request.getHeader("X-Requested-With"));

if(!ajax){
    response.sendRedirect("home");
}else{
    String id_usuario = request.getParameter("id_usuario");
    boolean activar = Boolean.parseBoolean(request.getParameter("activo"));

    UserClient client = new UserClient();
    GenericType<Usuario> genericType = new GenericType<Usuario>() {};
    Usuario u = client.find(genericType, id_usuario);

    u.setActivo(activar);
    client.edit(u, id_usuario);
}
```

*Ilustración 79. Servlet encargado de activar/desactivar voluntarios*

### 4.3.18 Caso de uso CU23

El administrador podrá añadir equipos VOST al sistema. Para que esto ocurra, previamente debe existir algún usuario en el sistema que sea coordinador sin equipo asociado. Sin él, no se podrá dar de alta ningún equipo.

Esta función se hace mediante petición AJAX, por lo que, en el Servlet, antes de nada, se comprueba que esto es así. Posteriormente, se recogen los valores pasados en la petición, que son el nombre del equipo y el coordinador asociado.

```
boolean ajax = "XMLHttpRequest".equals(
                request.getHeader("X-Requested-With"));

if(!ajax){
    response.sendRedirect("home");
}else{
    String nombre = (String)request.getParameter("nombre");
    String coordinador = (String)request.getParameter("coordinador");
```

*Ilustración 80. Comprobación de privilegios y obtención de parámetros*

A continuación, se comprueba que no exista ningún equipo con el mismo nombre en la base de datos del sistema, ya que deben ser únicos. Para ello, se hace una consulta a la API pasando el nombre recogido por parámetro.

```
GenericType<EquipoVost> genericTypeTeam = new GenericType<EquipoVost>() {};
EquipoVost aux = tClient.findByName(genericTypeTeam, nombre);
```

*Ilustración 81. Comprobación de nombre de equipo inexistente*

Si el resultado de esa consulta es null, significa que no existe ningún equipo con ese nombre, por lo que se procede a registrar el nuevo equipo.

```
boolean ok = false;
if(aux == null){
    ok = true;
    UserClient uClient = new UserClient();
    GenericType<Usuario> genericTypeUser = new GenericType<Usuario>() {};
    Usuario coord = uClient.find(genericTypeUser, coordinador);

    EquipoVost equipo = new EquipoVost();
    equipo.setNombre(nombre);
    equipo.setCoordinador(coord);
    System.out.println("llego y la variable ok es " + ok);
    EquipoVost newTeam = tClient.createTeam(equipo, genericTypeTeam);

    coord.setIdEquipo(newTeam.getId());
    uClient.edit(coord, String.valueOf(coord.getId()));
}
```

*Ilustración 82. Registro del nuevo equipo*

La variable booleana se usa para devolver un JSON informando al usuario si el registro del equipo ha sido satisfactorio, o si por el contrario no ha podido registrarse.

```
JSONObject output = new JSONObject();
output.put("ok", ok);

response.setContentType("application/json");
response.setCharacterEncoding("UTF-8");
response.getWriter().write(output.toString());
```

*Ilustración 83. Envío de JSON a la vista con información de si se había creado el equipo o no*

# 5. CONCLUSIONES Y LÍNEAS FUTURAS

Este proyecto me ha servido personalmente para poner en práctica todo lo aprendido en la carrera. Y no sólo eso, sino que he tenido que aprender otras tecnologías para poder desarrollar mi parte del proyecto.

Por un lado, al ser un proyecto en grupo, ha habido muchas puestas en común a la hora del desarrollo. El trabajo en equipo comenzó en la captura de requisitos, ya que pensamos que definiríamos así mejor la lista de funcionalidades que debíamos cumplir. Este trabajo en equipo se extendió durante el modelado, e incluso, hasta al desarrollo, donde tuvimos que colaborar para el desarrollo de la API.

Por otro lado, cada uno tenía que desarrollar su parte, que en mi caso era la aplicación móvil. Esto no ha supuesto ningún inconveniente, ya que, si en algún momento nos hemos sentido bloqueados durante el desarrollo, nos ayudábamos mutuamente. Además, viendo que la aplicación móvil tenía una lista de requisitos menos abultada, me encargué de desarrollar algunas de las funcionalidades de la aplicación web.

Una de las cuestiones que más problemas me ha dado es el trato con los plugins existentes en Ionic, como son la cámara, las notificaciones push o el almacenamiento, por ejemplo. Algunos de ellos están poco (o incluso mal) documentados, por lo que buscar información fue bastante tedioso y costó bastante tiempo. Por lo demás, aunque fuese una tecnología desconocida para mí, sí que supe adaptarme sin demasiada dificultad.

De cara al futuro, este proyecto se ha dejado como base sobre la cual seguir desarrollando y ampliando sus funcionalidades. En un principio, está pensado para utilizarla a nivel estatal, pero podría ampliarse fácilmente a nivel europeo, abarcando varios países. Posteriormente, se debería mejorar la forma de mostrar el mapa con las emergencias activas, ya que es básico. Además, podría establecerse un chat interno entre miembros de un mismo equipo, para que la colaboración entre los mismos sea más sencilla y directa, además de entre coordinadores de diferentes equipos. Por último, otra funcionalidad futura sería la inclusión de usuarios temporales que se encargarían de temas de administración, ante una gran carga de trabajo.

## 6. BIBLIOGRAFÍA

http://ionicframework.com/docs/ - Documentación de Ionic

http://api.jquery.com/ - Documentación de jQuery

https://www.w3schools.com/ - Documentación de tecnologías web

https://www.w3schools.com/howto/howto_css_switch.asp - toggle switch

https://github.com/dropbox/dropbox-sdk-java - API Java de Dropbox

http://plugins.krajee.com/file-input/demo - Plugin FileInput de Bootstrap

http://t4t5.github.io/sweetalert/ - Plugin SweetAlert

https://es.stackoverflow.com/ - Consultas múltiples

https://medium.com/@ankushaggarwal/push-notifications-in-ionic-2-658461108c59 - Tutorial para notificaciones push en ionic 2

https://console.firebase.google.com/ - Notificaciones push para Android

https://www.dropbox.com/developers - Documentación API de Dropbox