ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
INGENIERÍA DEL SOFTWARE

ENTORNO SOFTWARE PARA LA PREVENCIÓN DE EMERGENCIAS

SOFTWARE ENVIROMENT FOR EMERGENY PREVENTION

Realizado por
IGNACIO OLIVARES SUBÍAS
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: En este Trabajo de fin de Grado, se aborda la problemática que tenemos hoy en día ante la desinformación de la sociedad ante una emergencia, y la cantidad de información errónea que podemos obtener a través de las redes sociales. Para ello se ha creado un entorno software que mediante una API REST, y dos clientes (web y móvil), se pretende informar a la ciudadanía de cómo proceder en el caso de que acontezca una emergencia. La funcionalidad del cliente móvil ha sido planteada para que el usuario pueda informarse debidamente de lo que sucede y con información contrastada y a su vez poder aportar información e imágenes (que posteriormente debe ser aceptada), todo ello en tiempo real. Además, también podrá consultar las pautas básicas a seguir en las emergencias más comunes. Por otro lado, el cliente web mantiene la misma funcionalidad que el cliente móvil, como canal de información, y un lugar donde poder aportarla. Además, añade las funcionalidades de administración de todo el sistema, como el lanzamiento de las emergencias y la contribución de información oficial contrastada para hacer frente a todos los engaños que circulan por la red. Así mismo también colaborar con la información para aquellas emergencias consideradas más comunes.

Palabras claves: emergencia, emergencia oficial, emergencia no oficial, API REST, Ionic, protocolos

Abstract: In this final degree Project, we face the problem about the lack of information that society has in case of an emergency and the amount of wrong information that we can get through social networks. For this purpose a software environment has been created, which through a REST API, and two clients (web and mobile), is intended to inform the public of how to proceed in the event of an emergency. The functionality of the mobile client proposes that the user can properly be informed about what is happening and with verified information and in return be able to provide information and image (wich subsequently must be accepted), all in real time. You will also be able to consult the basic guidelines to follow in the most common emergencies. On the other hand, the web client maintains the same functionality as the mobile client, as an information channel, and a place where they can contribute. In adition, its system-wide management adds features such as the launch of emergencies and the contribution of verified official information to deal with all the deceptions that circulate on the network. Also it collaborates with the information for those more common emergencies.

Keywords: emergency, oficial emergency, non oficial emergency, REST API, Ionic, protocols

Índice

1. Introducción 1
1.1 Objetivos. 1
1.2 Problemática a cubrir 2
1.3 Tecnologías utilizadas 3
2. Análisis de Requisitos 5
2.1 Requisitos funcionales 5
2.2 Requisito no funcionales 8
2.3 Actores 9
2.4 Diagrama de Casos de uso 9
2. 5 Descripción de los Casos de uso 12
2.6 Matriz de trazabilidad 24
3. Diseño y Modelado del software 25
3.1 Modelo de datos 25
3.2 Diagrama de Clases 26
3.3 Diagrama de despliegue 27
3.4 Patrones de diseño 28
4.Implementacion 29
4.1 Proyecto 29
4.2 Desarrollo de las funcionalidades 31
4.2.1 Gestión de la Sesión y los Perfiles. 31
4.2.2 Gestión de los Protocolos 44
4.2.3 Gestion de las Emergencias y su información. 50
4.2.4 Gestion de los usuarios 64
5. Conclusiones y líneas futuras 70
5.1 Líneas futuras 71
6. Bibliografía 72

1. Introducción

En esta introducción se explicarán las causas que han motivado la creación de este proyecto, la problemática que se pretende resolver y los objetivos que se marcaron cuando se propuso el Trabajo de Fin de Grado.

En este trabajo se van a abordar principalmente el desarrollo de una API REST en la cual abstraer el manejo de la base de datos y mantener la lógica del sistema, y un cliente web, con el que realizar peticiones a la API y de esta manera consumir sus servicios por parte de los usuarios.

Este proyecto se ha planteado en equipo con mi compañero Sergio Rodríguez Moyano.

1.1 Objetivos.

Este proyecto se plantea como la base para crear un Entorno Software para la Prevención de Emergencias, destinado a estar implantado en España como el principal canal de comunicación general entre las administraciones y la población para otorgar un mayor control de la situación ante una Emergencia. De esta manera la ciudadanía, frente una situación de emergencia tiene un lugar al que acudir en busca de pautas y recomendaciones para una mayor seguridad. De esta manera se pretende garantizar tanto una mayor protección de los individuos, como una mayor prevención de riesgos.

Así mismo también poder otorgar conocimiento e información sobre protocolos o recomendaciones de actuación ante emergencias más comunes como pueden ser el caso de los terremotos o la crecida de algún rio por altas precipitaciones.

Como hemos mencionado uno de los principales objetivos es proveer de una canal de comunicación a las administraciones con el cual poder llegar a los ciudadanos de una manera más directa. Desde el cliente web las administraciones podrán lanzar emergencias que podrán ser consultadas por la ciudadanía en tiempo real mostrando siempre dos canales de información.

El primer canal lo conoceremos como información oficial donde solo se mostrarán mensajes y recomendaciones dadas por medios públicos como (Policía, Bomberos, Protección Civil…) y otro como información no oficial que estará formado por datos contribuidos por la población desde los clientes web y móvil. Esta información no oficial para poder ser mostrada, antes deberá ser validada por los equipos VOST (equipos de voluntarios digitales en emergencias) los cuales estarán encargados de dejar pasar solo información 100% real y de esta manera poder erradicar así bulos y montajes de internet.

Lo usuarios del sistema estarán organizados mediante equipos, los cuales tendrán como gestor a un usuario con el rol de coordinador. Este podrá dar de alta tanto miembros para su equipo, encargados de gestionar los protocolos, el canal de información oficial y de validar la información no oficial. Por otro lado, habrá siempre un administrador, encargado de crear y gestionar los equipos, y la asignación de su coordinador. También obtiene la capacidad de ver a todos los usuarios dados de alta en el sistema. Finalmente, los únicos usuarios con la capacidad de crear emergencias serán los coordinadores y los administradores.

Mi parte del proyecto incluirá la mayor parte de las funcionalidades del cliente web, como gestión de los protocolos y su información, el control de las emergencias y de su información, la sesión de los usuarios y el control de su perfil y la creación de nuevos usuarios y su visualización.

1.2 Problemática a cubrir

Este proyecto es planteado por el representante de VOST España, “Rafael Gálvez Rivas”, para Protección Civil como respuesta al problema que tenemos con la tendencia de buscar respuestas en redes sociales e internet sin llegar a acceder a canales oficiales de nuestras administraciones públicas. Hay un gran número de situaciones donde se ha generado peligro a partir de bulos o engaños propagados por las redes sociales.

Ante esto se pretenden conseguir paliar estos diversos problemas:

- Prevenir que la aparición de bulos pueda desestabilizar una zona creando situaciones de peligro
- Proveer una canal de información estándar contrastado para que toda la población pueda acceder a él en busca de recomendaciones de seguridad.
- Dar la posibilidad a las personas envueltas en una situación de peligro contar su situación y aportar información útil sobre la situación del entorno o la suya propia.

1.3 Tecnologías utilizadas

En este apartado se comentarán las tecnologías y librerías que han sido utilizadas a lo largo del desarrollo de este proyecto. Se ha intentado siempre buscar soluciones con licencias de software libre.

- Java EE: plataforma de desarrollo basada en Java orientada al desarrollo de sistemas web.
- Servicios REST: Conjunto de servicios que permiten la obtención y modificación de datos mediante servicios, que pueden consumidos en cualquier lenguaje de programación que pueda utilizar peticiones HTTP.
- MySQL: sistema de base de datos SQL desarrollado por Oracle.
- JavaScript: lenguaje de programación, muy utilizado en la programación de clientes web.
- JQuery: biblioteca basada en JavaScript que permite simplificar muchas de las interacciones con los elementos HTML para modificar sus valores o para facilitar la interacción con AJAX.
- Bootstrap: framework que contiene mejoras para el diseño de la web, con la que programadores con nociones básicas de diseño, pueden crear sus primeras interfaces adaptativas, utilizar iconos para ayudar a la comprensión de los elementos y mejorar el aspecto en general de la web que se está desarrollando.

- HTML5: lenguaje utilizado para construir el esquema de las vistas de la página web.
- CSS: lenguaje de diseño gráfico utilizado para mejorar el aspecto visual de los elementos HTLM de la web.
- NetBeans: IDE utilizado para la creación y desarrollo de todo el proyecto web. Teniendo funcionalidad para obtener las clases de las entidades y los servicios REST a partir de la conexión con la Base de datos.
- Git: herramienta utilizada para el control de revisiones y darnos la posibilidad de trabajar simultáneamente pudiendo así compartir nuestras novedades de una manera rápida y sencilla.
- Sweetalert: plugin utilizado para sustituir las ventanas emergentes convencionales, por unas más agradables a la vista y que puedan aportar una mayor información.
- Tinymce: plugin que permite la utilización en nuestra web de un editor de texto con formato.

2. Análisis de Requisitos

Para la recogida de requisitos y la mayor comprensión del proyecto que se debía desarrollar elegimos tener una entrevista con el cliente para así explicarnos cuál era su visión de las diversas funcionalidades. Este es el resultado de su captura y posterior desarrollo.

2.1 Requisitos funcionales

ID	Nombre	Descripción
AWEB1	Iniciar Sesión	El usuario podrá iniciar sesión en el sistema mediante su correo electrónico y una 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
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

ID	Nombre	Descripción
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 de voluntario	El usuario podrá enviar una 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.
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 los 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

ID	Nombre	Descripción
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 información oficial
AM3	Listado de información no oficial	El usuario podrá ver el listado de información no oficial
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 información 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 oficial
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 a 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 Requisito no funcionales

ID	Nombre	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 tendrán una recarga automática, después de un período de tiempo establecido
Seguridad	**RNF4**	No se permitirá el acceso a funcionalidades con privilegios a ciertos usuarios.

2.3 Actores

El conjunto de actores que se describe a continuación serán los encargados de interactuar con el sistema y su interfaz.

- **Usuario no identificado**: este actor representa a todas aquellas personas que acceden tanto a la aplicación web como a la móvil de forma anónima.
- **Voluntario**: este actor representa aquellos usuarios que se les ha dado una cuenta con la única funcionalidad de validar información no oficial.
- **Miembro**: este actor representa aquellos usuarios que pertenecen a un equipo VOST, un nivel superior en los roles con el que poder tener acceso a alguna funcionalidad extra.
- **Coordinador**: este actor representa aquellos usuarios encargados de un equipo VOST. Es un rol con el que poder controlar las emergencias que comprendan la zona donde tenga actuación el equipo que a su vez el mismo gestiona.
- **Administrador**: este actor representa aquellos usuarios con el nivel de privilegios máximo, teniendo la capacidad de realizar cualquier funcionalidad del sistema.

2.4 Diagrama de Casos de uso

En esta sección se mostrarán los diagramas de casos de uso obtenidos a partir de los requisitos funcionales.

2.4.1 Diagrama de Casos de uso cliente web

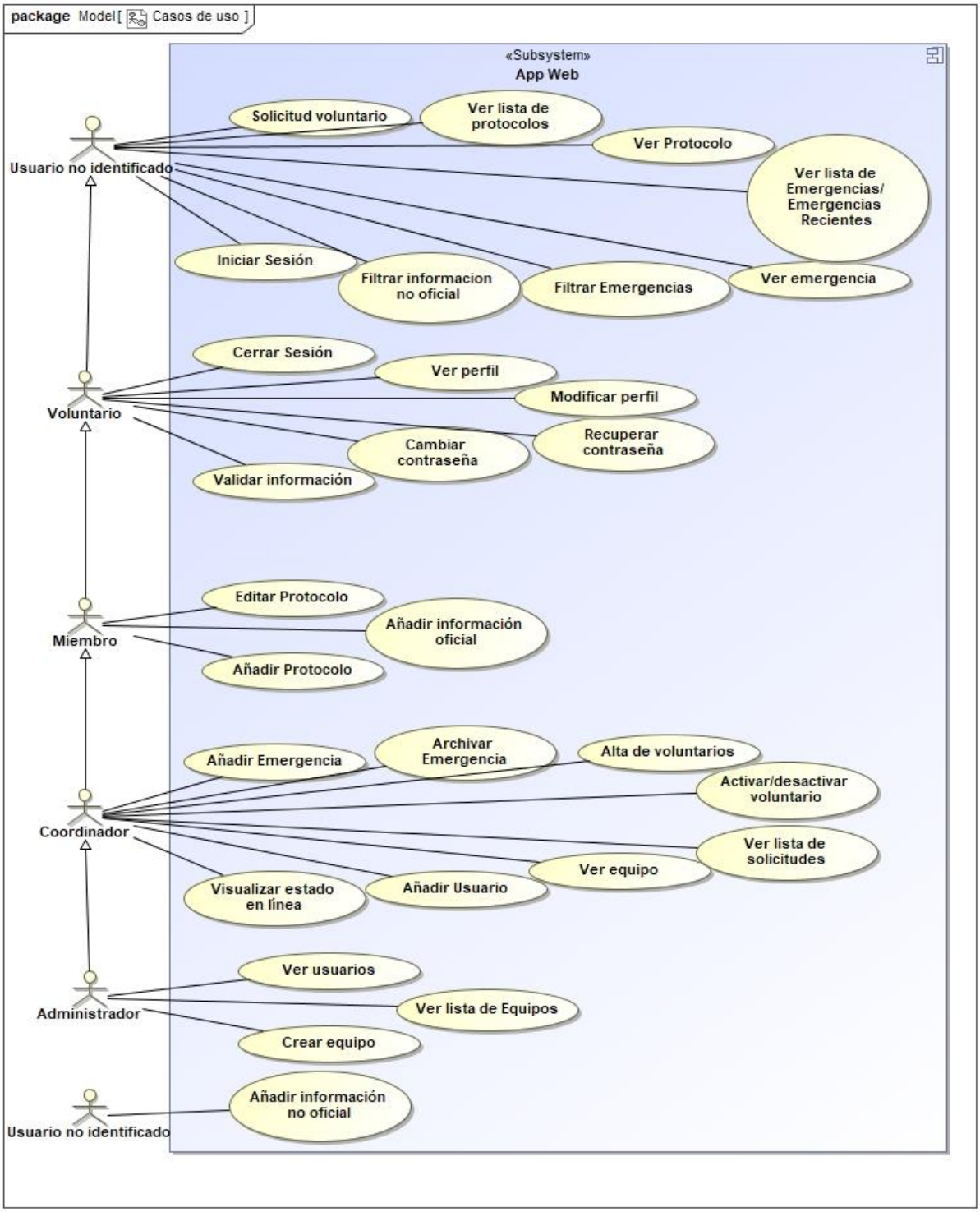

Ilustración 1. Diagrama casos de uso cliente web

2.4.2 Diagrama de Casos de uso cliente móvil.

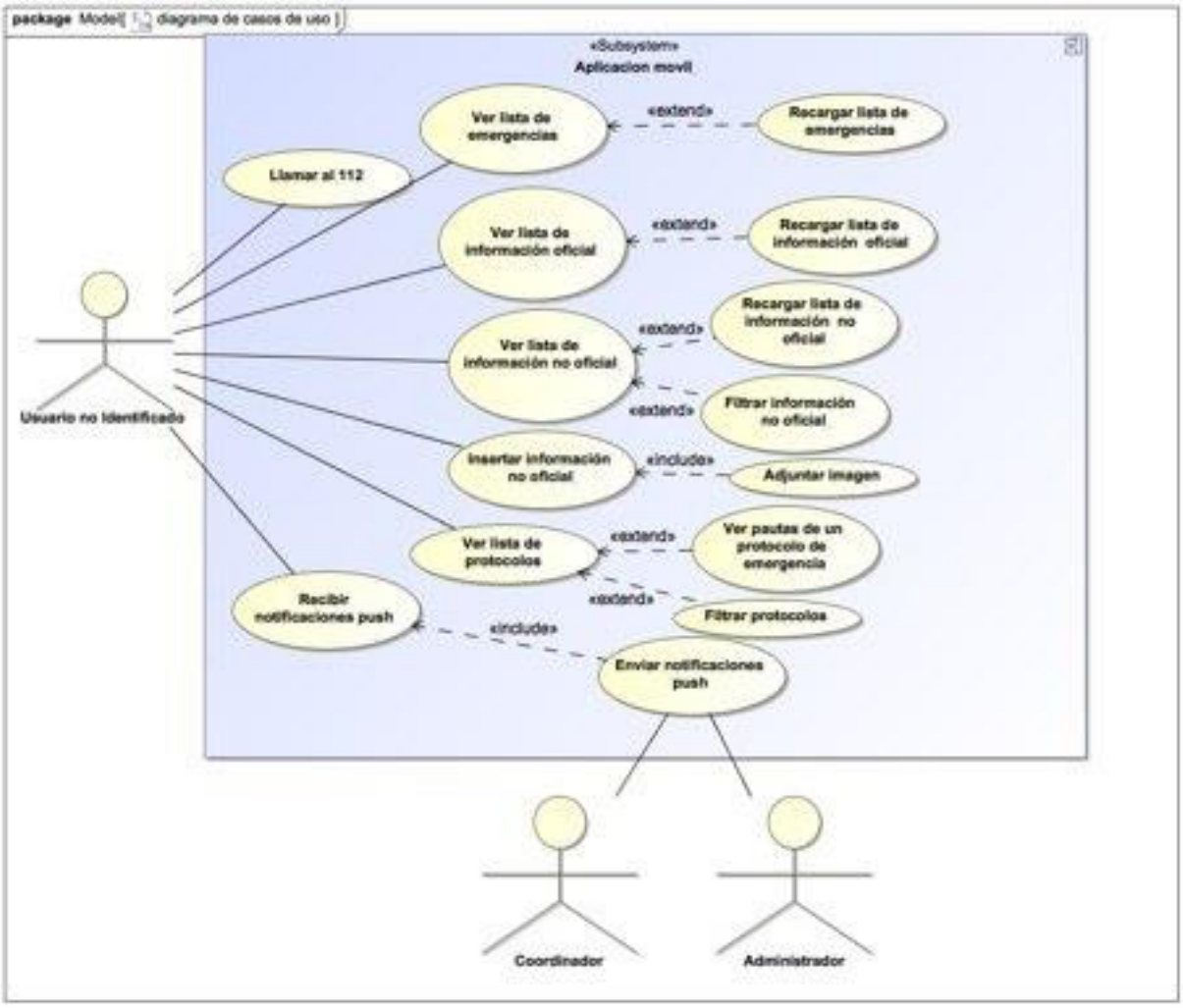

Ilustración 2. Diagrama casos de uso cliente móvil

2. 5 Descripción de los Casos de uso

ID	U1
Caso de uso	Ver lista de Emergencias
Descripción	El usuario podrá ver una lista con todas las emergencias
Precondición	Ninguna
Escenario principal	1. El usuario pulsa sobre el botón de “Emergencias” 2. El sistema realiza una búsqueda en base de datos de todas las emergencias no archivadas. 3. El sistema genera la vista mostrando la lista de emergencias.
Escenario alternativo	
Requisito asociado	AWEB10, AWEB11

ID	U2
Caso de uso	Filtrar Emergencias
Descripción	El usuario podrá aplicar un filtro por Comunidades autónomas a la lista de las emergencias
Precondición	Ninguna
Escenario principal	1.El usuario pulsa sobre el mapa en una de las comunidades autónomas 2.El sistema realiza la búsqueda en base de datos de las emergencias que no estén archivadas asociadas a dicha comunidad autónoma. 3. El sistema genera la vista mostrando la lista de emergencias
Escenario alternativo	1. El usuario despliega el filtro y selecciona una comunidad autónoma de la lista 2.El sistema realiza la búsqueda en base de datos de las emergencias que no estén archivadas asociadas a dicha comunidad autónoma. 3. El sistema genera la vista mostrando la lista de emergencias
Requisito asociado	AWEB12

ID	U3
Caso de uso	Ver lista de protocolos
Descripción	El usuario podrá ver una lista con todos los protocolos
Precondición	Ninguna
Escenario principal	1. El usuario pulsa sobre el botón de “Protocolos” 2. El sistema realiza la búsqueda en base de datos de todos los protocolos. 3. El sistema genera la vista mostrando la lista de protocolos
Escenario alternativo	
Requisito asociado	AWEB6

ID	U4
Caso de uso	Ver Protocolo
Descripción	El usuario podrá ver la descripción de un protocolo
Precondición	Haber realizado U3
Escenario principal	1. El usuario pulsa sobre el nombre de un protocolo 2. El Sistema obtiene la descripción de dicho protocolo. 3. El sistema genera un campo con la descripción en la vista
Escenario alternativo	
Requisito asociado	AWEB7

ID	U5
Caso de uso	Añadir Protocolo
Descripción	El usuario podrá añadir un nuevo protocolo
Precondición	Estar autentificado, tener un rol de miembro, coordinador o Administrador y haber realizado U3
Escenario principal	1. El usuario pulsa sobre el botón de “Añadir Protocolo”. 2. El sistema redirige a la vista de “Añadir Protocolo”. 3. El usuario escribe el nombre y la descripción del protocolo y pulsa sobre el botón de “Guardar”. 4. El sistema guarda el nuevo protocolo en base de datos. 5. El sistema hace una nueva búsqueda de la lista de protocolos. 6. El sistema genera la vista con la lista de protocolos actualizada.
Escenario alternativo	
Requisito asociado	AWEB8

ID	U6
Caso de uso	Editar Protocolo
Descripción	El usuario podrá editar la descripción de un protocolo existente
Precondición	Estar autentificado, tener un rol de miembro, coordinador o Administrador y haber realizado U4
Escenario principal	1. El usuario pulsa sobre el botón “Editar” 2. El sistema carga un editor de texto en el que modificar la descripción. 3. El usuario modifica la descripción y pulsa sobre el botón “Guardar” 4. El sistema actualiza el protocolo en base de datos. 5. El sistema elimina el editor de texto y vuelve a genera el campo con la descripción del protocolo actualizada.
Escenario alternativo	
Requisito asociado	AWEB9

ID	U7
Caso de uso	Añadir Usuario
Descripción	El usuario podrá dar de alta un nuevo usuario
Precondición	Estar autentificado y tener un rol de Coordinador o administrador
Escenario principal	1. El usuario pulsa sobre el desplegable con el nombre de Usuarios y selecciona la opción de “Añadir Usuario” 2. El sistema genera la vista de “añadir usuario” 3. El usuario introduce los datos del nuevo usuario y pulsa el botón “Añadir usuario” 4. El sistema genera una contraseña aleatoria y añade el nuevo usuario al sistema. 5. El sistema envía la contraseña aleatoria al correo electrónico del nuevo usuario. 6. El sistema genera la vista de la lista de usuarios.
Escenario alternativo	
Requisito asociado	AWEB21

ID	U8
Caso de uso	Ver lista de usuarios
Descripción	El usuario podrá ver la lista completa de usuarios y su estado en línea.
Precondición	Estar autentificado y tener un rol de Administrador
Escenario principal	1. El usuario pulsa sobre el desplegable con el nombre de Usuarios y selecciona la opción de “Lista de Usuarios”. 2. El sistema realiza la búsqueda en base de datos de los usuarios. 3. El sistema genera la vista de la lista de usuarios
Escenario alternativo	
Requisito asociado	AWEB22, AWEB23

ID	U9
Caso de uso	Ver perfil
Descripción	El usuario podrá ver sus datos de usuario
Precondición	Estar autentificado
Escenario principal	1. El usuario pulsa sobre el desplegable con el nombre del usuario y selecciona la opción de “Mi perfil” 2. El sistema realiza la búsqueda de los datos del usuario. 3. El sistema genera la vista con el perfil del usuario.
Escenario alternativo	
Requisito asociado	AWEB3

ID	U10
Caso de uso	Modificar perfil
Descripción	El usuario podrá modificar sus datos personales salvo el correo electrónico
Precondición	Estar autentificado y haber realizado U10
Escenario principal	1. El usuario modifica cualquiera de los datos y pulsa sobre el botón “Guardar” 2. El sistema actualiza los datos del perfil.
Escenario alternativo	
Requisito asociado	AWEB4

ID	U11
Caso de uso	Cambiar contraseña
Descripción	El usuario podrá cambiar su contraseña
Precondición	Estar autentificado y haber realizado U10
Escenario principal	1. El usuario introduce una nueva contraseña y la contraseña actual. 2. El sistema verifica que la contraseña es correcta. 3. El sistema actualiza la contraseña
Escenario alternativo	2. El sistema detecta que la contraseña actual es errónea y manda un mensaje de aviso.
Requisito asociado	AWEB5

ID	U12
Caso de uso	Ver Emergencia
Descripción	El usuario podrá ver toda la información de una emergencia, tanto oficial como no oficial
Precondición	Haber completado U1
Escenario principal	1. El usuario pulsa sobre el botón “Mas info” de una emergencia. 2. El sistema obtiene toda la información sobre la emergencia. 3. El sistema genera la vista de la emergencia
Escenario alternativo	
Requisito asociado	AWEB13,

ID	U13
Caso de uso	Añadir información no oficial
Descripción	El usuario podrá añadir información no oficial
Precondición	Haber completado U13
Escenario principal	1. El usuario pulsa sobre el botón de “Añadir información”. 2. El sistema muestra una ventana emergente donde poder inscribir información. 3. El usuario completa los campos sobre la información no oficial y pulsa el botón de “Enviar” 4. El sistema registra la nueva información no oficial y la deja a la espera de ser validada.
Escenario alternativo	
Requisito asociado	AWEB16

ID	U14
Caso de uso	Validar información
Descripción	El usuario podrá aceptar/rechazar información no oficial
Precondición	Estar autentificado y tener un rol de voluntario o superior.
Escenario principal	1. El usuario pulsa sobre el botón de “Validar Información”. 2. El sistema carga la lista de emergencias activas y redirige a la vista de validar información. 3. El usuario pulsa sobre una de las emergencias. 4. El sistema carga la lista de post no oficiales pendientes de validar. 5. El usuario entiende que la información es útil y pulsa el botón de aceptar. 6. El sistema actualiza la información no oficial como aceptada y es mostrada en la emergencia.
Escenario alternativo	5. El usuario entiende que la información no aporta nada a la emergencia y pulsa el botón de rechazar. 6. El sistema elimina dicha información no oficial.
Requisito asociado	AWEB17

ID	U15
Caso de uso	Añadir información Oficial
Descripción	El usuario podrá añadir información oficial
Precondición	Estar autentificado, tener un rol de Miembro, Coordinador o Administrador y haber completado U13
Escenario principal	1. El usuario pulsa sobre el botón de “Añadir información”. 2. El sistema muestra una ventana emergente donde poder inscribir información. 3. El usuario completa los campos sobre la información no oficial y pulsa el botón de “Enviar” 4. El sistema registra la nueva información oficial.
Escenario alternativo	
Requisito asociado	AWEB15

ID	U16
Caso de uso	Iniciar Sesión
Descripción	El usuario podrá iniciar sesión en el sistema mediante su correo electrónico y una contraseña
Precondición	Ninguna
Escenario principal	1. El usuario pulsa sobre el botón de “Iniciar Sesión” 2. El sistema genera la vista de iniciar sesión. 3. El usuario introduce su correo electrónico y contraseña. 4. El sistema comprueba que el correo electrónico y contraseña. 5. La contraseña es correcta y el sistema genera un token aleatorio y lo inserta en la base de datos asociándolo con el usuario que ha iniciado sesión. 6. El sistema envía el token al cliente el cual es guardado en sesión. 7. El sistema redirige a la página de origen.
Escenario alternativo	5. La contraseña es incorrecta y el sistema redirige de vuelta a la vista de iniciar sesión mostrando un mensaje de error.
Requisito asociado	AWEB1

ID	U17
Caso de uso	Cerrar sesión
Descripción	El usuario podrá salir del sistema
Precondición	Estar autentificado
Escenario principal	1. El usuario pulsa sobre el desplegable con el nombre del usuario y selecciona la opción de “Cerrar Sesión” 2. El sistema elimina el token de usuario de la base de datos y actualiza su última conexión. 3. El sistema elimina los datos en sesión y las cookies. 4. El sistema carga la vista del home
Escenario alternativo	
Requisito asociado	AWEB2

ID	U18
Caso de uso	Archivar Emergencia
Descripción	El usuario podrá archivar una emergencia
Precondición	Estar autentificado, tener un rol de Coordinador o Administrador y haber realizado U1.
Escenario principal	1. El usuario pulsa sobre el botón con el icono de una carpeta situado en la emergencia. 2. El sistema muestra una ventana preguntado al usuario, la confirmación de la acción. 3. El usuario pulsa sobre el botón aceptar. 4. El sistema actualiza la emergencia como archivada. 5. El sistema recarga la vista de emergencias.
Escenario alternativo	
Requisito asociado	AWEB26

ID	U19
Caso de uso	Recuperar Contraseña
Descripción	El usuario podrá recuperar su contraseña
Precondición	Estar en la vista de iniciar sesión y tener un rol de voluntario o superior.
Escenario principal	1. El usuario pulsa sobre el de recuperar contraseña 2. El sistema muestra una ventana preguntado al usuario, su correo electrónico. 3. El usuario introduce su correo electrónico y pulsa sobre él captcha. 4. El sistema comprobara que el correo electrónico está en el sistema. 5. El sistema genera una contraseña aleatoria para el usuario y se la envía por correo electrónico.
Escenario alternativo	4. El sistema comprueba que el correo electrónico no está en el sistema y redirige de vuelta a la vista.
Requisito asociado	AWEB30

2.6 Matriz de trazabilidad

Para visualizar los casos de uso desarrollados por el alumno se ha generado esta matriz de trazabilidad para enlazar cada caso de uso con sus respectivos requisitos

	U1	U2	U3	U4	U5	U6	U7	U8	U9	U10	U11	U12	U13	U14	U15	U16	U17	U18	U19
AWEB1																X			
AWEB2																	X		
AWEB3									X										
AWEB4										X									
AWEB5											X								
AWEB6			X																
AWEB7				X															
AWEB8					X														
AWEB9						X													
AWEB10	X																		
AWEB11	X																		
AWEB12		X																	
AWEB13												X							
AWEB14																			
AWEB15															X				
AWEB16													X						
AWEB17														X					
AWEB18																			
AWEB19																			
AWEB20																			
AWEB21							X												
AWEB22								X											
AWEB23								X											
AWEB24																			
AWEB25																			
AWEB26																		X	
AWEB27																			
AWEB28																			
AWEB29																			
AWEB30																			X

Ilustración 3. Matriz de trazabilidad

3. Diseño y Modelado del software

En esta sección vamos a ver como hemos planteado las diversas soluciones de diseño para realizar todas las funcionalidades del proyecto.

3.1 Modelo de datos

Este es modelo de datos que planteamos para poder organizar todos aquellos elementos que vamos a tener la necesidad de almacenar y utilizar.

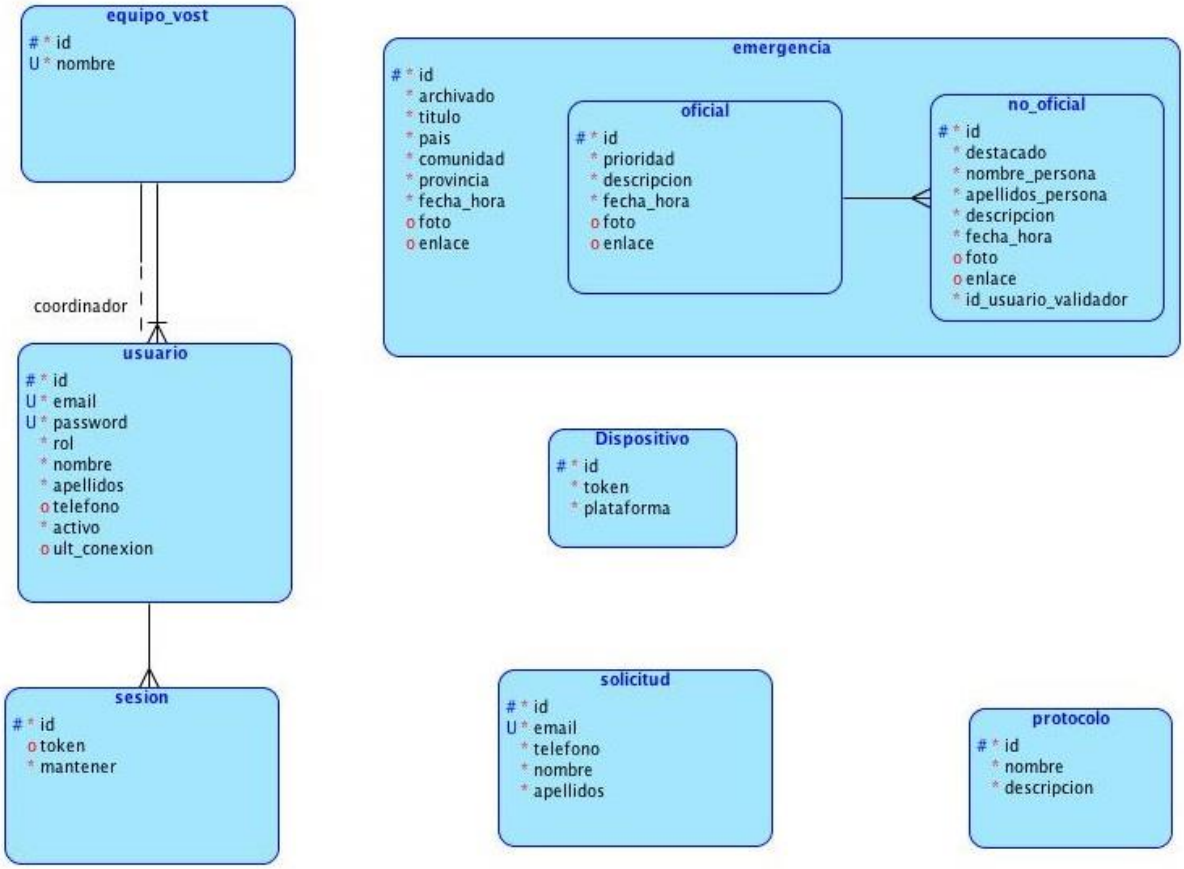

Ilustración 4. Modelo de datos

3.2 Diagrama de Clases

A partir del modelo de datos obtenemos este diagrama de clases con las entidades que vamos a utilizar en el proyecto.

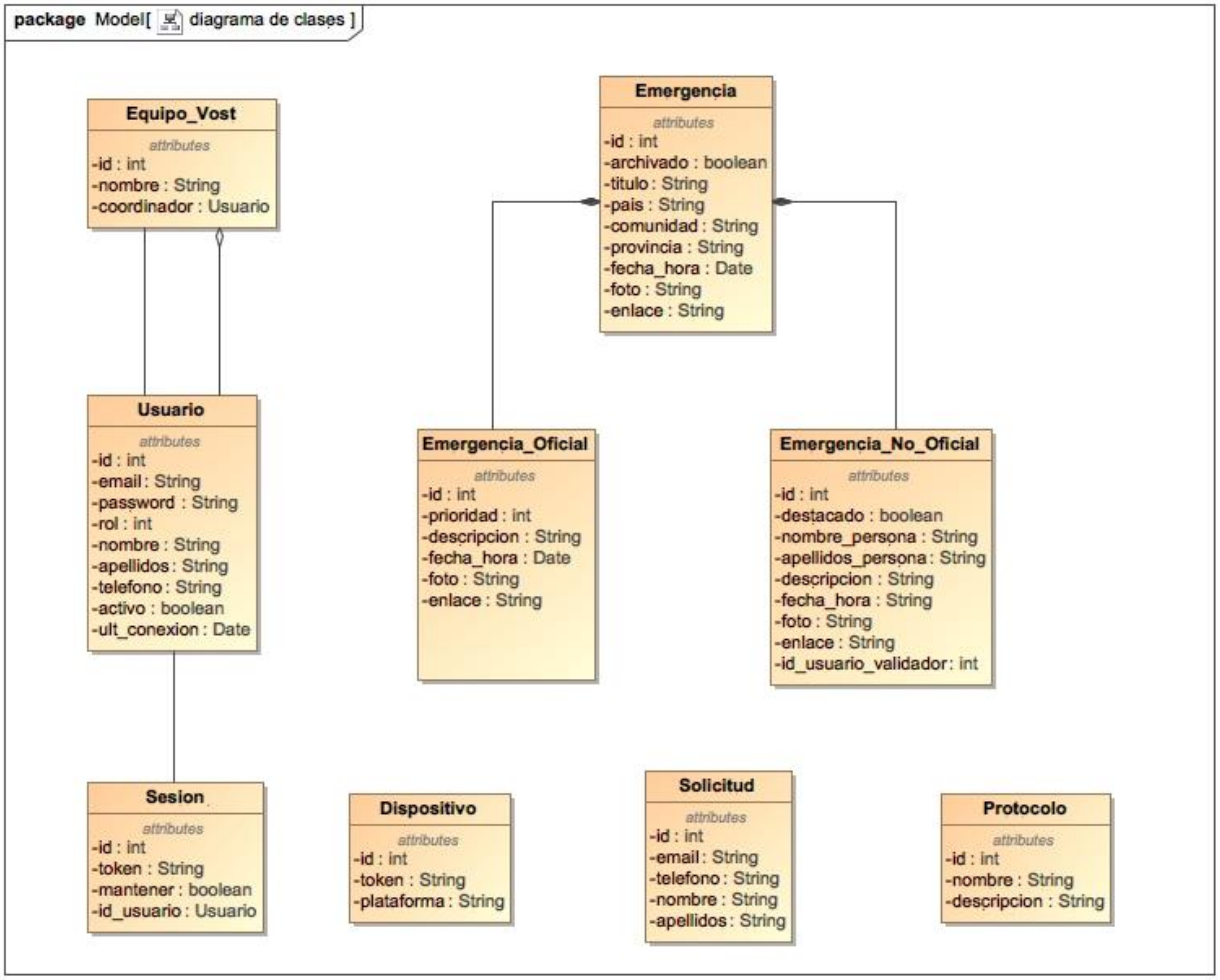

Ilustración 5. Diagrama de clases

3.3 Diagrama de despliegue

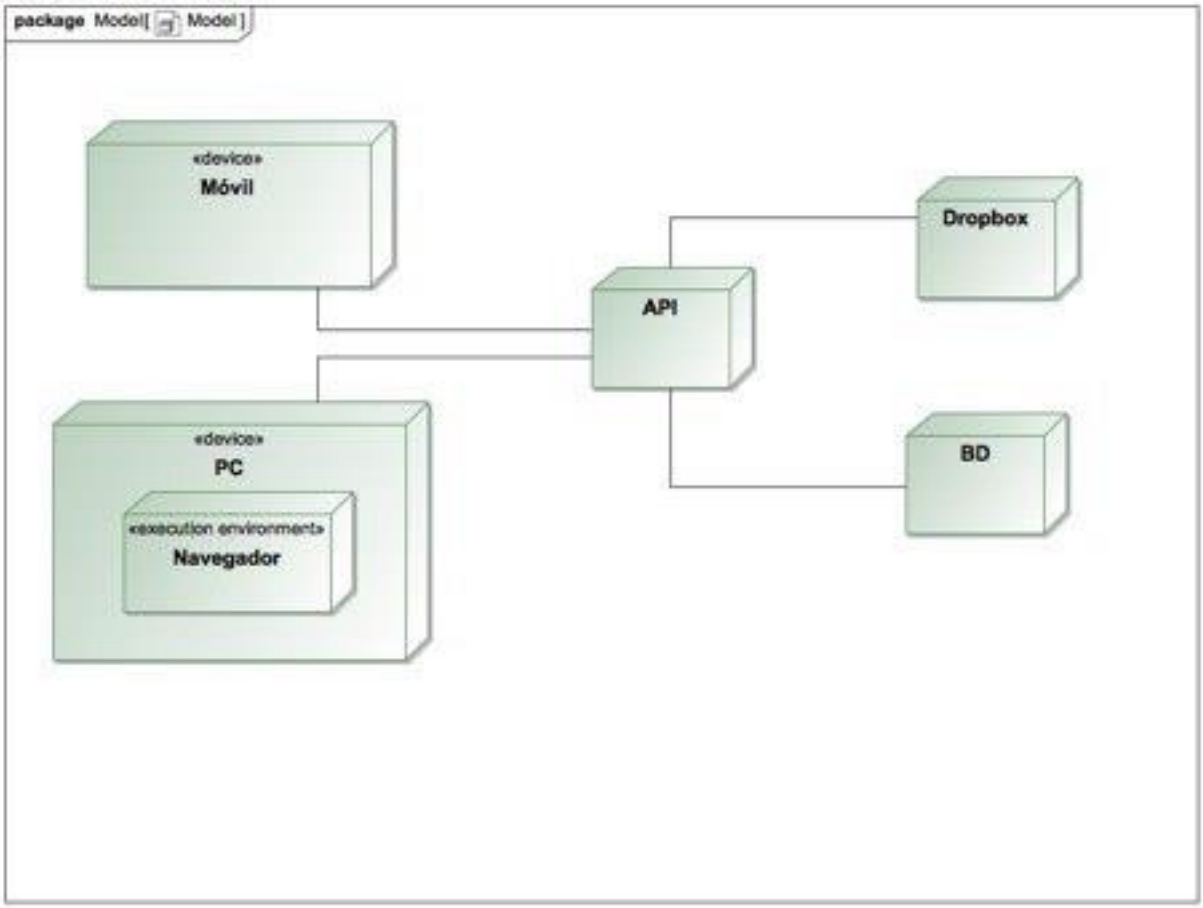

Ilustración 6. Diagrama de despliegue

3.4 Patrones de diseño

Para el cumplir con los objetivos se han implementado el uso de dos patrones.

3.4.1 Patrón Modelo Vista Controlador (MVC)

Este patrón se diseñó para separar los datos y la lógica de negocio de la aplicación de la interfaz de usuario. Para ello se utilizan tres componentes: el modelo la vista y el controlador.

- **El modelo**: parte encargada de abstraer la obtención de la información de los datos. Es el encargado de tanto obtener los datos como de manipularlos (inserción, edición, borrado). Recibe las peticiones de los usuarios de los datos que se necesitan mostrar en la vista.
- **Vista**: encargada de la interacción con el usuario, presenta los datos obtenidos en el modelo y enviados por el controlador de la manera en la que el usuario lo entiende. También se encarga de recibir las peticiones del usuario y comunicárselas al controlador.
- **Controlador**: parte encargada de la comunicación entre la vista y el modelo. Es el responsable de recibir las peticiones del usuario a través de la vista y enviarlas al modelo para que, una vez recibidas sean enviadas a la vista, cuya función es gestionar los datos obtenidos.

3.4.2 Patrón fachada abstracta.

El patrón fachada es un patrón estructural, centrado en la creación de objetos. Es el patrón bajo el que se desarrolla la lógica de los servicios REST desarrollados en la API.

4.Implementacion

En esta sección se pretende dar respuesta a las decisiones tomadas para el desarrollo del entorno software.

4.1 Proyecto

Este proyecto se ha estructurado como ya mencionamos con una arquitectura MVC (Modelo Vista Controlador), para la cual desglosamos cada cual de los componentes que funcionalidad va a cumplir en esta arquitectura.

Modelo: este papel se va a cubrir mediante el desarrollo de una API que mediante servicios REST va a gestionar el acceso al modelo de datos.

Controlador: este papel se va a cubrir mediante el desarrollo de Servlet que van a ser los encargados gestionar las llamadas a los servicios REST mediante la utilización de clientes, los cuales serán clientes generados utilizando el framework “Project Jersey”.

Vista: este papel se va a cubrir mediante el desarrollo de jsp que utilizando HTML5 y java mostrarán al usuario toda la información recogida en los controladores.

4.1.1 Estructura de la API

Creada mediante un proyecto web estándar de NetBeans presenta esta estructura:

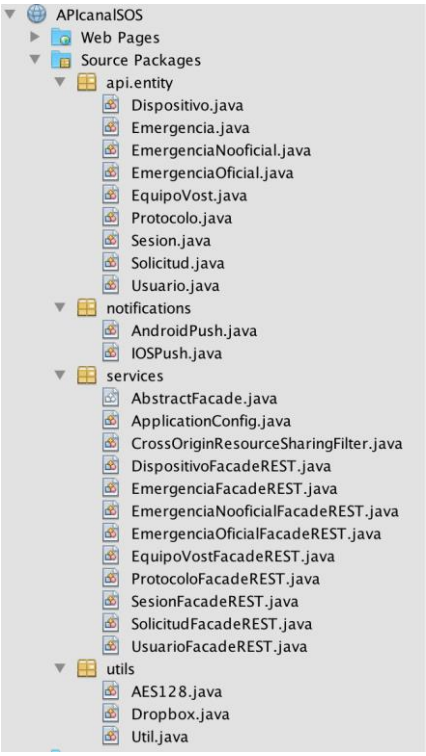

Ilustración 7. Esquema API

Cada entidad tiene asociada su fachada donde se desarrollan los métodos y servicios necesarios para realizar las funciones del sistema actualizando o recogiendo datos de la base de datos si fuera necesario.

Además, se han utilizado los métodos de las clases “AES128.java” y “Util.java”, clases que hemos reutilizado de la asignatura Seguridad en servicios y aplicaciones” para poder generar HASH en MD5 para la codificación de las contraseñas. También la clase Dropbox con la que podemos subir imágenes.

4.1.2 Estructura del cliente web

Creado mediante un proyecto web estándar de NetBeans

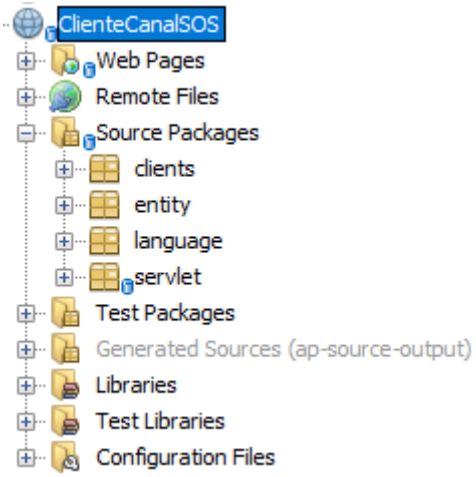

Ilustración 8. Esquema cliente web

El proyecto está formado por web Pages siendo estas paginas “jsp”. En el paquete “clients” están situados todos los clientes para los servicios web REST. En el paquete “entity”, se almacenan todas las entidades generadas a través de la Base de datos, para que los controladores y las vistas puedan tratar los objetos. En el paquete “language”, están situados los archivos para internacionalización. Finalmente, en el paquete “servlet” están todos los Servlet que harán las funciones de controladores.

4.2 Desarrollo de las funcionalidades.

En esta sección vamos a hablar de la implementación llevada a cabo en cada grupo de funcionalidades comunes, hablando en cada apartado a partir de su caso de uso. Explicaremos las clases que se han creado (controladores) y los diferentes métodos que han desarrollado para realizar las funciones de servicios de la API.

4.2.1 Gestión de la Sesión y los Perfiles.

Para cumplir con los requisitos y casos de uso asociados a estas funcionalidades se han desarrollado los siguientes servicios y clases. Este apartado supone los casos de uso: U9, U10, U11, U16, U17 y U19.

4.2.1.1 Inicio de Sesión

Cuando el usuario necesite iniciar sesión, deberá pulsar sobre el botón de “Iniciar sesión” situado en la parte derecha de la barra de navegación, y este nos llevara hasta la vista de inicio de sesión.

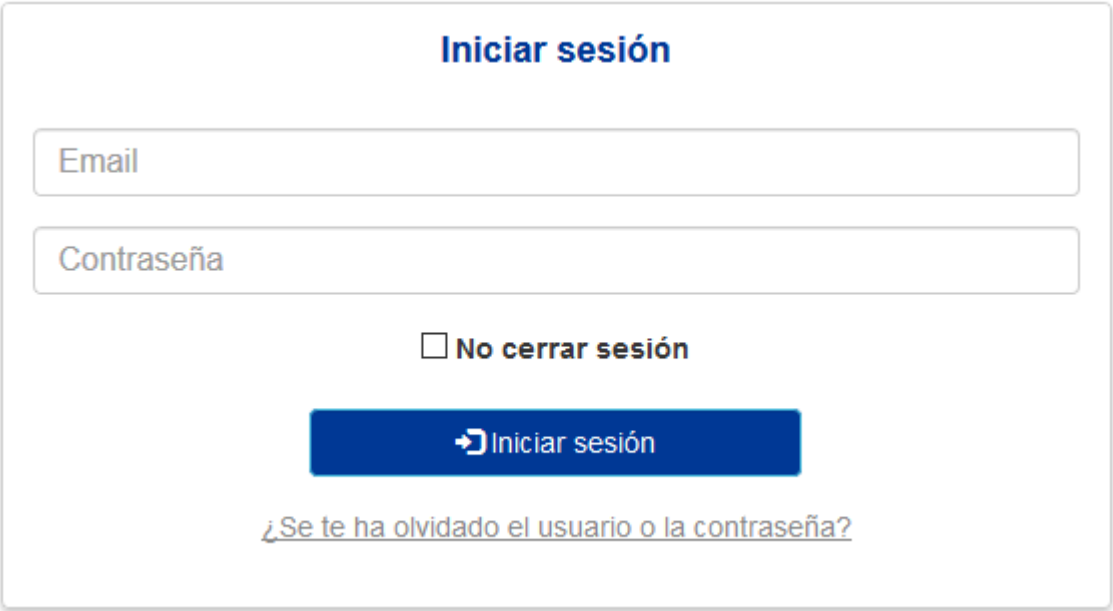

Ilustración 9.Vista de Inicio de sesión

Una vez allí el usuario introduce su correo electrónico y contraseña y pulsa sobre el botón de Iniciar sesión. Tambien comentar que si se selecciona el campo de no cerrar sesion el sistema guardara el inicio de sesión en una cookie de duracion de 1 año, en caso contrario la cookie expirara en un día.

Una vez recogidos los parámetros por el controlador “Login” este crea un objeto JSON con ambos parametros y envía la petición de login mediante su servicio correspondiente.

```
UserClient client = new  UserClient();

JSONObject login = new JSONObject();
login.put("email", name);
login.put("pass", pass);
if(mantener == null){
    login.put("mantener", false);
}else{
    login.put("mantener", true);
}

String output = client.login(login.toString());
JSONObject json = new JSONObject(output);
```

Ilustración 10. 1º Parte controlador de inicio de sesión

Posteriormente en la API se comprueban si el correo electrónico y la contraseña coinciden con la de un usuario del sistema. Como se puede observar se genera el HASH a través de la contraseña introducida, para poder compararlo con el almacenado en base de datos.

```
public String login(String data) throws UnsupportedEncodingException, NoSuchAlgorithmException{
    Query q;
    Usuario u=null;

    JSONObject json = new JSONObject(data);
    String email = json.getString("email");
    String pass = json.getString("pass");
    boolean mantener = json.getBoolean("mantener");

    byte[] bytesOfMessage = pass.getBytes("UTF-8");
    MessageDigest md = MessageDigest.getInstance("MD5");
    byte[] thedigest = md.digest(bytesOfMessage);


    String passCifrada = Util.bytesToHexString(thedigest).toLowerCase();

    q = em.createQuery("SELECT u FROM Usuario u WHERE u.email = :EMAIL AND u.password = :PASS AND u.activo != 0");
    q.setParameter("EMAIL", email);
    q.setParameter("PASS", passCifrada);

    List<Usuario> userlist = q.getResultList();
```

Ilustración 11. 1º Parte servicio de inicio de sesión

Una vez obtenido resultado se comprueba si el usuario obtenido no es nulo. Una vez comprobado, lo primero es ver si existe un token asociado al usuario que acabamos de obtener, si ese fuera el caso, obtendríamos dicho token y lo enviaríamos como respuesta. Si no existiera un token asociado debemos generarlo e insertarlo en la Base de datos. Finalmente guardamos el token, el rol y el nombre del usuario generando un objeto JSON con esos tres valores para ser enviados como respuesta.

```
if(!userlist.isEmpty()){
    u=userlist.get(0);
    rol = u.getRol();
    nombre = u.getNombre()+ " " + u.getApellidos();
Date ultConexion = new Date();
u.setUltConexion(ultConexion);
edit(u.getId(), u);
    //Si no esta en session se crea
    Sesion sesionUser = checkUserSession(u);
    if(sesionUser == null){
        //obtenemos un token que no exista e insertamos fila en la tabla temporal
        do{
            token = generateRandomString();
        }while(tokenExist(token));

        Sesion sesion = new Sesion();
        sesion.setIdUsuario(u);
        sesion.setToken(token);
        sesion.setMantener(mantener);

        sesionFacadeREST.create(sesion);

    }else{
        token = sesionUser.getToken();
    }
}
JSONObject result = new JSONObject();
result.put("token", token);
result.put("rol", rol);
result.put("nombre", nombre);
return result.toString();
```

Ilustración 12. 2º Parte servicio de inicio de sesión

Para terminar en el controlador comprobaremos si en la respuesta el token es nulo, si lo fuera el inicio de sesión habría fallado al introducir datos erróneos por lo que el sistema enviaría al usuario de vuelta al inicio de sesión mostrando un mensaje de error. Si en cambio tenemos un token válido el sistema lo almacena en sesión junto con el rol del usuario y redirigirá al usuario al home.

```
String token = null;
String nombre = null;
int rol = -1;
if(!json.isNull("token")){
    token = json.getString("token");
    rol = json.getInt("rol");
    nombre = json.getString("nombre");
}

if(token==null){
    RequestDispatcher rdp;
    rdp = this.getServletContext().getRequestDispatcher("/login.jsp");
    rdp.forward(request, response);
}else{
    session.setAttribute("token", token);
    session.setAttribute("rol", rol);
    session.setAttribute("nombre", nombre);
    Cookie miCookie = new Cookie("session", token);
    if(mantener == null){
        miCookie.setMaxAge(86400);
    }else{
        miCookie.setMaxAge(31536000);
    }

    response.addCookie(miCookie);

    response.sendRedirect("home");
}
```

Ilustración 13. 2º Parte del controlador de inicio de sesión

4.2.1.2 Cierre de Sesión

Una vez el usuario pulsa en el botón con su nombre en la parte derecha de la barra de navegación se despliega un menú, dentro debe pulsar la opción de “Cerrar Sesión”. Una vez pulsa el botón el sistema llama al controlador de “Logout”.

Este controlador recoge el token de la sesión. Si no existiera un token en sesión querría decir que no hay una sesión iniciada y redirigiría directamente al home. Si existe un token en sesión entonces el sistema primero eliminara la cookie que alberga el token de sesión asociado. A continuación, realizara una llamada a la API para cerrar sesión mediante el cliente enviando el token en un objeto JSON donde se eliminaría la asociación con el usuario en la tabla de sesión. Para finalizar se destruirá la sesión y se redirigirá al “home”.

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

String token = (String)session.getAttribute("token");

if(token==null){
    response.sendRedirect("home");
}else{
    Cookie[] mis_cookies = request.getCookies();

    for(Cookie mi_cookie : mis_cookies) {
        if (mi_cookie.getName().equals("session")) {
            mi_cookie.setMaxAge(0);
            response.addCookie(mi_cookie);
        }
    }

    UserClient client = new UserClient();
    JSONObject json = new JSONObject();
    json.put("token", token);
    client.logout(json.toString());

    session.invalidate();

    response.sendRedirect("home");
}
```

Ilustración 14. Controlador de cierre de sesión

En la llamada a la API para cerrar sesión, se buscará al usuario para actualizar su última hora de conexión y se eliminará su token de la base de datos.

```
public void logout(String data){
    JSONObject json = new JSONObject(data);
    String token = json.getString("token");

    Usuario u = findUserByToken(token);

    Date ultConexion = new Date();

    u.setUltConexion(ultConexion);

    edit(u.getId(), u);

    sesionFacadeREST.deleteToken(token);
}
```

Ilustración 15. Servicio de cierre de sesión

4.2.1.3 Ver perfil y modificar datos

Una vez el usuario pulsa en el botón con su nombre en la parte derecha de la barra de navegación se despliega un menú, dentro debe pulsar la opción de “Mi perfil”. Una vez pulsa el botón el sistema llama al controlador de “Account” y redirige a la vista.

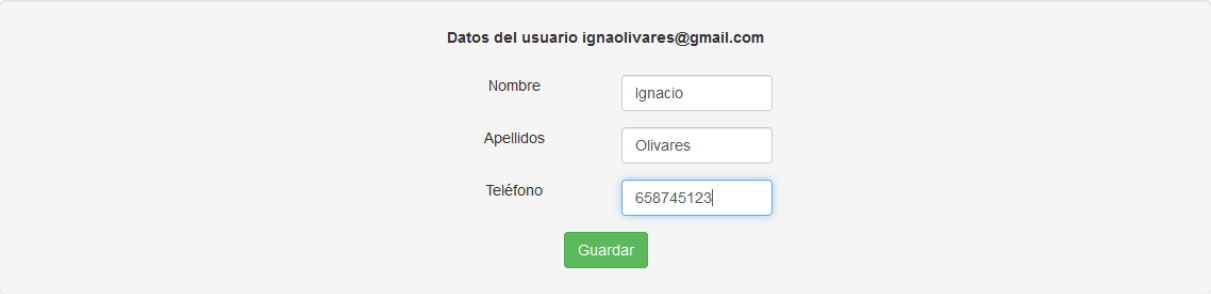

Ilustración 16. Vista de modificar perfil

Una vez allí el usuario puede cambiar cualquiera de sus datos y pulsar el botón “Guardar. Ese botón llama a la función JavaScript que recoge los datos introducidos y mediante una petición AJAX llama al controlador “editUser”. Si uno de los campos obligatorios como el nombre o apellidos estuviera vacío, aparecería una ventana emergente Sweetalert para avisarnos. Una vez completada la petición mostrara un mensaje de confirmación de éxito del cambio con otra ventana emergente.

```
function submitChangeProfile(){
    var nombre = $('#nombre').val();
    var apellidos = $('#apellidos').val();
    var telefono = $('#telefono').val();
    if(nombre === "" || apellidos === ""){
        swal("", "Ups el Nombre o el Apellido no pueden estar vacios", "error");
    }else{
        $.ajax({
            type:'POST',
            url: base_url + 'editUser',
            data: {
                'nombre':nombre,
                'apellidos':apellidos,
                'telefono':telefono

            },
            success: function(){
                swal("", "Tus datos han sido actualizados", "success");
            }
        });
    }
}
```

Ilustración 17. Petición AJAX de modificar perfil

La lógica del controlador recoge los parámetros y los añade a un objeto JSON junto con el token identificativo del usuario. Posteriormente crea la llamada al servicio de la API y la envía junto con el objeto JSON.

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

UserClient userClient = new  UserClient();

JSONObject json = new JSONObject();
json.put("token", token);
json.put("nombre", nombre);
json.put("apellidos", apellidos);
json.put("telefono", telefono);

userClient.editUser(json.toString());
```

Ilustración 18. Controlador de modificar perfil

Finalmente, en la API se recoge el objeto JSON y se obtienen los parámetros que fueron enviados. Utilizando el token buscamos el Usuario, se modifican sus datos y finalmente se actualiza la base de datos.

```java
@POST
@Consumes({"application/json"})
@Path("editUser")
public void editUser(String data){
    JSONObject json = new JSONObject(data);
    String token = json.getString("token");
    String nombre = json.getString("nombre");
    String apellidos = json.getString("apellidos");
    String telefono = json.getString("telefono");

    Usuario u = findUserByToken(token);

    u.setNombre(nombre);
    u.setApellidos(apellidos);
    u.setTelefono(telefono);

    edit(u);
}
```

Ilustración 19. Servicio de modificar perfil

4.2.1.4 Cambiar contraseña

Para cambiar contraseña debemos repetir el caso de uso U9 e introducirnos en nuestro perfil. Una vez allí veremos los siguientes campos donde realizaremos el cambio de contraseña.

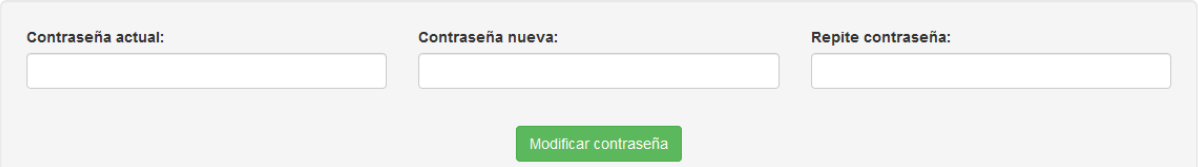

Ilustración 20. Vista de cambiar contraseña

Debemos introducir nuestra contraseña actual y 2 veces la nueva contraseña y pulsar sobre el botón de “Modificar contraseña”. Esto accionará la llamada a una función JavaScript que comprobará si la contraseña actual es la correcta mediante una petición AJAX. Si lo es, se realizará a la función encargada de realizar el cambio de contraseña. En caso contrario mostrará una ventana emergente Sweetalert con un mensaje de error.

```
function submitChangePass(){
    var pwd = $('#pwd').val();
    $.ajax({
        type:'POST',
        url: base_url + 'Usuario',
        success: function(output){

            if(output.password === md5(pwd)){
                changePass()
            }else{
                swal("", "La contraseña actual no es correcta", "error");
            }
        }
    });

}
```

Ilustración 21. Petición AJAX de comprobación de contraseña

La función recoge los dos campos de la nueva contraseña y comprueba que no estén vacíos, mostrando un mensaje de error si se diera el caso. A continuación, llama al controlador, que será el encargado de actualizar la contraseña. Si todo ha salido bien, mostrará una ventana emergente con una confirmación.

```
function changePass(){
    var newpass = $('#newpass').val();
    var newpass1 = $('#newpass1').val();
    if(newpass === newpass1){
        $.ajax({
            type:'POST',
            url: base_url + 'ChangePass',
            data: {
                'newpass':newpass
            },
            success: function(){
                swal("", "Tus contraseña ha sido cambiada correctamente", "success");

            }
        })
    }else{
        swal("", "Ups las contraseñas no eran iguales", "error");
    }

}
```

Ilustración 22. Petición AJAX de cambio de contraseña

En el controlador crearemos el objeto JSON que contendrá la nueva contraseña y el token identificativo del usuario. Este objeto se enviará junto a la petición al servicio REST gestionada por el “UserClient”.

```
UserClient client = new UserClient();

String newpass = (String)request.getParameter("newpass");

JSONObject data = new JSONObject();
data.put("token", token);
data.put("newpass", newpass);
client.changePass(data.toString());
```

Ilustración 23. Controlador de cambio de contraseña

En el servicio recogemos el objeto JSON con el token y la contraseña. Generamos el hash que vamos a utilizar para actualizar la contraseña y mediante el token identificativo obtenemos el objeto Usuario. Finalmente se actualiza el objeto y la base de datos

```
@POST
@Consumes({"application/json"})
@Path("changePass")
public void changePass(String data) throws UnsupportedEncodingException, NoSuchAlgorithmException{
    JSONObject json = new JSONObject(data);
    String token = json.getString("token");
    String newpass = json.getString("newpass");

    byte[] bytesOfMessage = newpass.getBytes("UTF-8");
    MessageDigest md = MessageDigest.getInstance("MD5");
    byte[] thedigest = md.digest(bytesOfMessage);

    String passCifrada = Util.bytesToHexString(thedigest).toLowerCase();

    Usuario u = findUserByToken(token);

    u.setPassword(passCifrada);

    edit(u);
}
```

Ilustración 24. Servicio de cambio de contraseña

4.2.1.5 Recuperar contraseña

Una vez el usuario está en la vista de iniciar sesión si pulsa sobre el enlace de “¿Has olvidado tu contraseña?”, este redirigirá a la vista correspondiente.

Ilustración 25. Vista de recuperación de contraseña

EL usuario debe introducir su correo electrónico y pulsar sobre el captcha. Una vez completos, el usuario pulsa sobre el botón de “Recuperar contraseña”, el cual produce una llamada al controlador “RecoverPass”. Primero comprobamos si el correo electrónico esta dado de alta en la base de datos, si no lo está el sistema redirigirá de vuelta a la vista de recuperar contraseña. Si el usuario si está, se creará un objeto JSON que contiene el correo electrónico del usuario y llamará al servicio de la API REST.

```
UserClient client = new UserClient();
GenericType<Usuario> genericType = new GenericType<Usuario>() {};
Usuario user = client.recover(genericType, email);

if(user==null){
    RequestDispatcher rdp;
    rdp = this.getServletContext().getRequestDispatcher("/resetPassword.jsp");
    rdp.forward(request, response);
}else{

    JSONObject json = new JSONObject();
    json.put("email", email);
    client.sendEmailRecover(json.toString());

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

Ilustración 26. Controlador de recuperación de contraseña

En el servicio de la API recogemos el correo electrónico y utilizándolo obtenemos el usuario. A continuación, generamos una contraseña aleatoria obtenemos su hash. Una vez obtenida la contraseña, actualizamos el objeto usuario con su nueva contraseña y enviamos un correo electrónico al usuario, con la contraseña que debe utilizar para volver a entrar.

```
public void sendEmailRecover(String data) throws UnsupportedEncodingException, NoSuchAlgorithmException{
    JSONObject json = new JSONObject(data);
    String email = json.getString("email");

    Query q;
    Usuario u;

    q = em.createQuery("SELECT u FROM Usuario u WHERE u.email = :EMAIL");
    q.setParameter("EMAIL", email);

    u = (Usuario)q.getSingleResult();
    String newpass = generateRandomString().substring(0, 8);

    byte[] bytesOfMessage = newpass.getBytes("UTF-8");
    MessageDigest md = MessageDigest.getInstance("MD5");
    byte[] thedigest = md.digest(bytesOfMessage);

    String passCifrada = Util.bytesToHexString(thedigest).toLowerCase();

    u.setPassword(passCifrada);

    edit(u);

    String message = "Tu nueva contraseña es " + newpass;
    sendEmail(email, message);
}
```

Ilustración 27. Servicio de recuperación de contraseña

4.2.2 Gestión de los Protocolos

Para cumplir con los requisitos y casos de uso asociados a estas funcionalidades se han desarrollado los siguientes servicios y clases. Este apartado supone los casos de uso: U3, U4, U5 y U6.

4.2.2.1 Ver lista de protocolos

Cuando el usuario pulsa sobre el botón de “Protocolos” situado en la barra de navegación, el sistema produce una llamada al controlador “ListProtocolos”. Este mediante el cliente asociado al servicio REST de los Protocolos realiza la siguiente llamada:

```
ProtocolsClient pClient = new ProtocolsClient();

GenericType<List<Protocolo>> genericType = new GenericType<List<Protocolo>>() {};
List<Protocolo> protocolosList = pClient.findAll(genericType);

session.setAttribute("pList", protocolosList);
```

Ilustración 28. Controlador de lista de protocolos

De esta manera el controlador recoge la lista completa de los protocolos actuales para que posteriormente sera entregada a la siguiente vista a la que el sistema redirigirá mostrando este resultado:

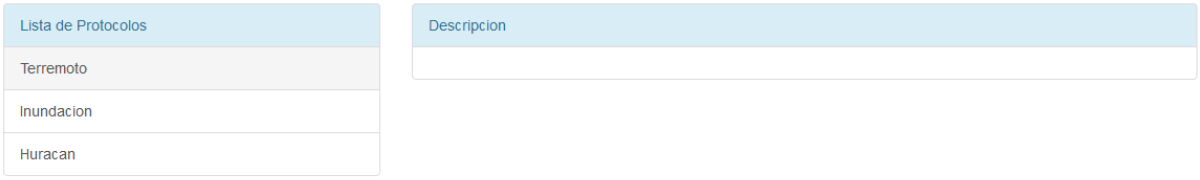

Ilustración 29. Vista de lista de protocolos

4.2.2.2 Ver Protocolo

Una vez dentro de la vista con la lista de protocolos el usuario podrá pulsar sobre cualquier título dentro de los protocolos, generando una petición AJAX, la cual creará una petición para obtener la descripción del protocolo.

```
function chargeProtocol(protocol){

    $.ajax({
        type:'POST',
        url: base_url + 'Protocolo',
        data: {
            'protocolo':protocol
        },
        success: function(output){
            $('#pDescripcion').empty();
            $('#pDescripcion').append(output.descripcion);
            protocol_selected = output.id;
            $('#cancel').hide();
            $('#edit').show();
        }
    });
}
```

Ilustración 30. Petición AJAX de obtener protocolo

Esta petición llama al controlador “ProtocoloServlet” el cual mediante el identificador del protocolo hará una llamada al servicio REST para obtener el objeto protocolo.

```
ProtocolsClient protClient = new ProtocolsClient();

String id_protocol = request.getParameter("protocolo");

GenericType<Protocolo> generic = new GenericType<Protocolo>() {};
Protocolo prot = protClient.find(generic, id_protocol);

JSONObject output = new JSONObject();

output.put("descripcion", prot.getDescripcion());
output.put("id", prot.getId());
```

Ilustración 31. Controlador de obtener protocolo

Posteriormente se creará un objeto JSON en el que añadiremos la descripción y el identificador del protocolo que seran recogidas en el succed de la petición AJAX, para poder así construir la descripcion y generar el botón de “editar” que será necesario para el siguiente caso de uso. El resultado es el que podemos apreciar en la siguiente imagen.

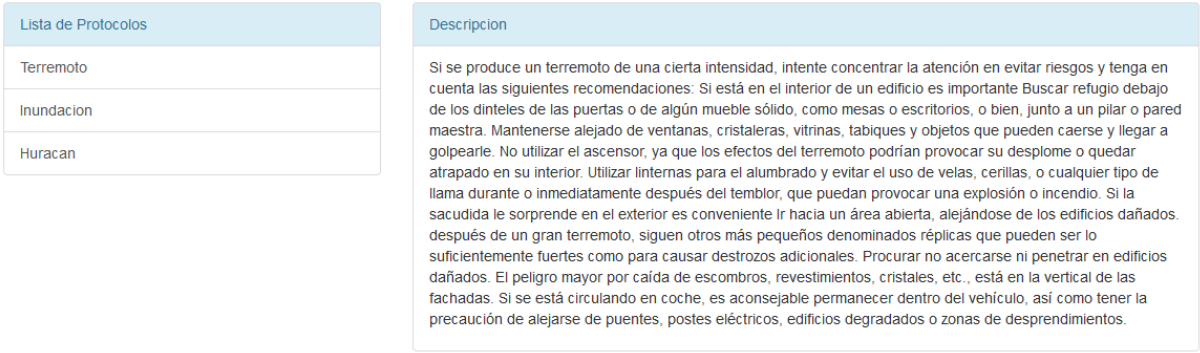

Ilustración 32. Vista con la descripción del protocolo

4.2.2.3 Editar Protocolo

Para la edición de un protocolo se necesita estar autentificado con un rol de miembro o superior. Cuando accedemos a la vista con autentificación esta cambia dejándonos a la vista los botones de editar y de Añadir nuevo protocolo.

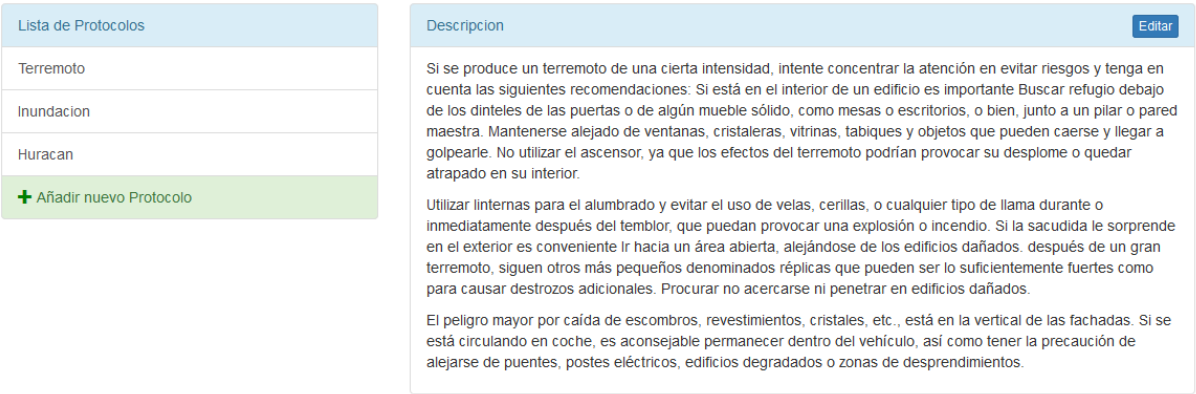

Ilustración 33. Vista del administrador de la lista de protocolos

Si pulsamos sobre el botón editar el sistema, mediante JavaScript carga el editor de texto con formato que hemos utilizado para que la edición de texto (TinyMce), ya que dado que el texto puede llegar a ser extenso consideramos necesario poder dar al usuario la capacidad de darle formato al texto para que así sea más visible y fácil de leer.

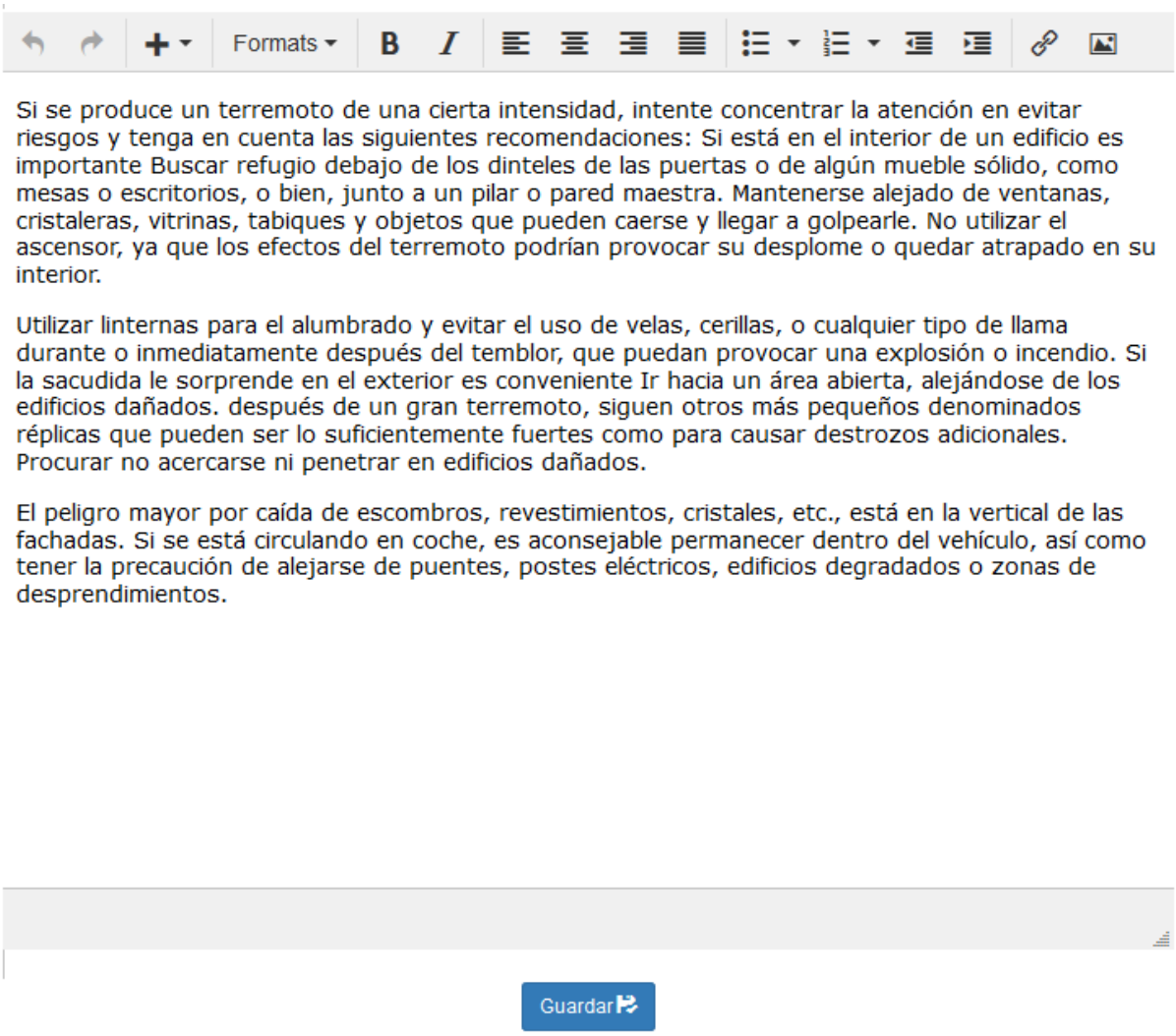

Ilustración 34. Editor de texto de la descripción del protocolo

Una vez modificado el texto el usuario pulsa sobre el botón “Guardar” y mediante una petición AJAX se recoge el nuevo valor del protocolo y se guarda en la base de datos llamando al correspondiente servicio.

```
function submitProtocol(){
    tinymce.triggerSave();
    var descripcion = $('#protocoloDesc').val();
    $.ajax({
        type:'POST',
        url: base_url + 'saveProtocol',
        data: {
            'protocolo':protocol_selected,
            'descripcion':descripcion
        },
        success: function(output){
            chargeProtocol(output.protocolo);
            $('#pDescripcion').show();
            tinymce.remove();
            $('#protocoloDesc').hide();
            $('#save').hide();

        }
    });
}
```

Ilustración 35. Petición AJAX de editar protocolo

La llamada para el Servlet “SaveProtocolo” envia por parámetro el identificador del protocolo y su nueva descripción, así el controlador llamará al servicio y podrá obtener el protocolo mediante su id. Despues se modificará el objeto para añadirle su nueva descripción y de esta manera realizar el update en Base de datos. Finalmente en un objeto JSON se guarda el identificador y que es enviado en el succes de la petición AJAX, donde se llamará a la función javaScript del caso de uso anterior para que se vea la descripción actualizada.

```
String protocoloDesc = request.getParameter("descripcion");
String id_protocolo = request.getParameter("protocolo");

ProtocolsClient protClient = new ProtocolsClient();

GenericType<Protocolo> protocolType = new GenericType<Protocolo>() {};
Protocolo prot = protClient.find(protocolType, id_protocolo);

prot.setDescripcion(protocoloDesc);

protClient.edit(prot,id_protocolo);

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

Ilustración 36. Controlador de editar protocolo

4.2.2.4 Añadir Protocolo

Como en el caso de uso anterior se necesita estar autentificado con un rol de miembro o superior. En este caso al pulsar sobre el botón de “Añadir nuevo Protocolo”, el sistema nos redirigirá a la vista donde podremos completar los datos del nuevo protocolo.

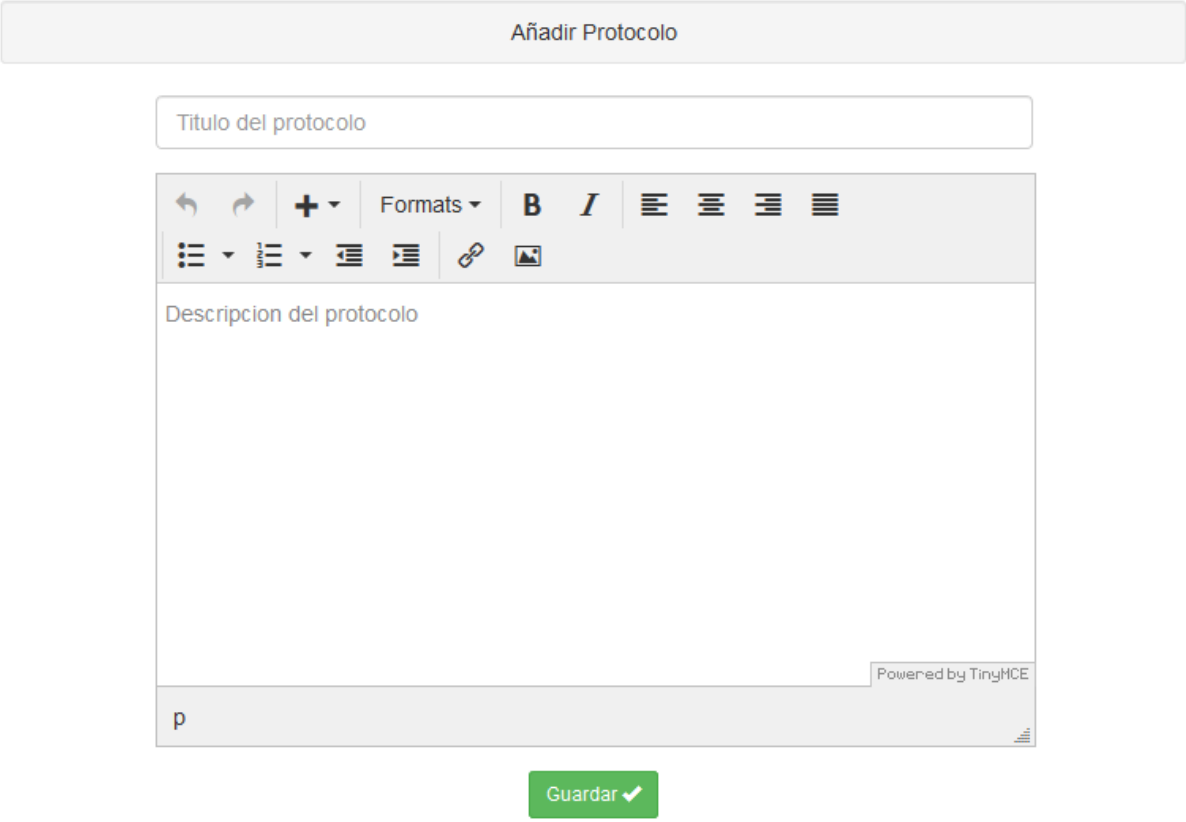

Ilustración 37. Vista de añadir protocolo

Una vez el usuario ha completado los campos, al pulsar el boton de “Guardar” el sistema realiza el envío del formulario al controlador “AddProtocol”, que recoge los parámetros y crea el nuevo objeto Protocolo.

```
String protocoloDesc = request.getParameter("protocoloDesc");
String titulo = request.getParameter("titulo");

ProtocolsClient protClient = new ProtocolsClient();

Protocolo prot = new Protocolo();
prot.setNombre(titulo);
prot.setDescripcion(protocoloDesc);

protClient.create(prot);
```

Ilustración 38. Controlador de añadir protocolo

Finalmente el sistema te redirige a la vista donde podemos ver la lista de protocolos, mostrando el nuevo protocolo en ella.

4.2.3 Gestion de las Emergencias y su información.

Para cumplir con los requisitos y casos de uso asociados a estas funcionalidades se han desarrollado los siguientes servicios y clases. Este apartado supone los casos de uso: U1, U2, U12, U13, U14, U15 y U18.

4.2.3.1 Ver lista de Emergencias, y filtrado

El Usuario pulsa sobre el botón de “Emergencias” situado en la barra de navegación. El sistema le redirige al controlador llamado “Emergencias”. El cual obtiene la lista de emergencias mediante una petición al servicio de la API usando la clase EmergencyClient.

```
String zona = request.getParameter("zona");

EmergencyClient eClient = new EmergencyClient();
GenericType<List<Emergencia>> genericType = new GenericType<List<Emergencia>>() {};
List<Emergencia> emergenciaList = null;

if(zona == null){
   emergenciaList = eClient.findNotArchivedEmergencies(genericType);
}else{
    emergenciaList = eClient.findByCommunity(genericType, zona);
}

session.setAttribute("eList", emergenciaList);
```

Ilustración 39. Controlador de la lista de emergencias

En el caso de pulsar en una de las regiones del mapa en vez de haber pulsado sobre alguna de las comunidades del mapa hubiéramos llamado al mismo controlador, pero pasando el parámetro de dicha comunidad como zona.

Ilustración 40. Vista de bienvenida

Como podemos observar a la izquierda del mapa se muestran las emergencias activas más recientes que siempre son mostradas en la vista home. Finalmente, cuando el controlador carga la lista de Emergencias muestra la siguiente vista.

Ilustración 41. Vista de emergencias

4.2.3.2 Ver Emergencia

Una vez el usuario esta sobre cualquier lista de emergencias, este solo debe pulsar sobre el enlace de “Mas info” que tiene cada emergencia, el cual llamará al controlador “Emergencia” que será el encargado de llevarnos a la vista donde ver la información de la emergencia.

El controlador mediante el identificador de la emergencia que recoge por parámetro obtendrá las listas de información oficial e información no oficial asociadas a la emergencia.

```
String id_emergencia = request.getParameter("id");

EmergencyOfficialClient emOfClient = new EmergencyOfficialClient();
EmergencyNoOfficialClient emNoOfClient = new EmergencyNoOfficialClient();

GenericType<List<EmergenciaNooficial>> genericTypeNoOficial = new GenericType<List<EmergenciaNooficial>>() {};
GenericType<List<EmergenciaOficial>> genericTypeOficial = new GenericType<List<EmergenciaOficial>>() {};

List<EmergenciaOficial> emergenciaOficialList = null;
List<EmergenciaNooficial> emergenciaNoOficialList = null;

emergenciaOficialList = emOfClient.findByIdEmergency(genericTypeOficial, id_emergencia);
emergenciaNoOficialList = emNoOfClient.posts(genericTypeNoOficial, id_emergencia);

session.setAttribute("eListOficial", emergenciaOficialList);
session.setAttribute("eListNoOficial", emergenciaNoOficialList);

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

Ilustración 42 Controlador de selección de emergencia

El sistema mostrara la vista correspondiente con toda la información. Esta vista incluye una recarga automática cada cierto periodo de tiempo para que el usuario siempre vea la información más actualizada.

Ilustración 43. Vista de la emergencia seleccionada

4.2.3.3 Añadir información no Oficial

Una vez el usuario ha completado el caso de uso de “Ver Emergencia” y pulsa sobre el botón de “Añadir información” el sistema desplegara una ventana emergente, con los diferentes campos a completar.

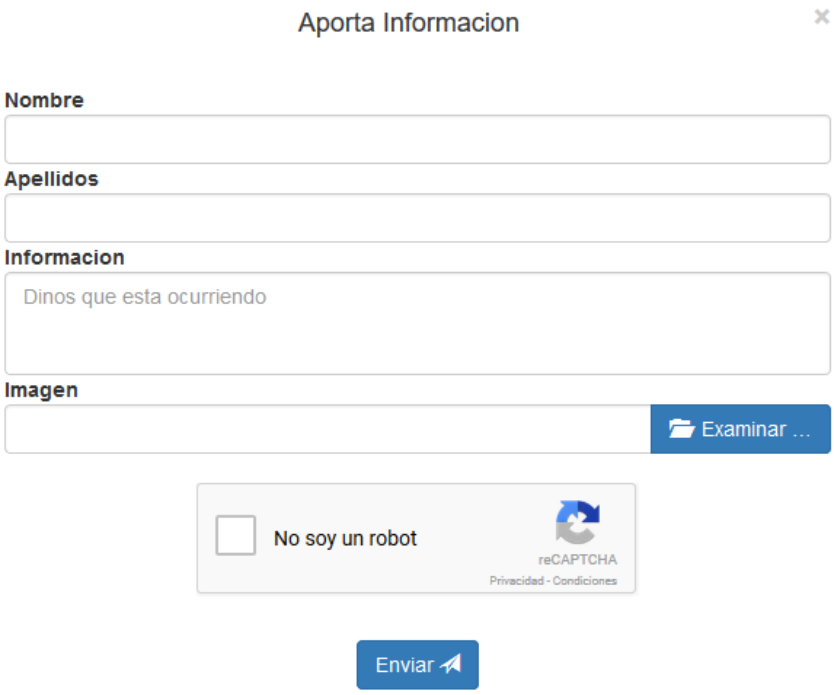

Ilustración 44. Ventana emergente para añadir información no oficial

Al completar el nombre, apellidos, el campo información y pulsar sobre el captcha (la imagen es opcional) el usuario puede pulsar sobre el botón “Enviar”. El sistema envía el formulario y muestra un mensaje de confirmación, avisando de que la información está a la espera de ser validada.

Una vez en el controlador, primero se recogen los parámetros de Nombre y Apellidos y la descripción de la información, se crea el objeto, se crea un objeto JSON con los valores y se envía en la llamada al servicio REST para crear la información no oficial. Este devuelve el objeto creado para la posterior inserción de la imagen (si la hubiera). Para que el almacenamiento de las imágenes siga un cierto orden, cada imagen es asociada al identificador de la información no oficial que la contiene, por eso debe ser previamente creado ya que al tener un id autoincrementado hasta que el objeto no ha sido creado no podemos obtenerlo.

```
EmergencyNoOfficialClient emNoClient = new EmergencyNoOfficialClient();

String name = (String)request.getParameter("nombre");
String surname = (String)request.getParameter("apellidos");
String info = request.getParameter("informacion");
String id_emergency = request.getParameter("id_emergencia");

JSONObject json = new JSONObject();

json.put("nombre", name);
json.put("apellidos", surname);
json.put("descripcion", info);
json.put("id_emergencia", id_emergency);

GenericType<EmergenciaNooficial> emNoGeneric = new GenericType<EmergenciaNooficial>() {};
EmergenciaNooficial post = emNoClient.newpost(json.toString(), emNoGeneric);
```

Ilustración 45. 1º Parte controlador de añadir información no oficial

Una vez en el servicio recogemos todos los valores que enviamos en el JSON, creamos el objeto de Emergencia no Oficial y le añadimos todos los valores. También ponemos el valor de validado a “0”, ya que esta información no debe aparecer hasta no ser validada.

```
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
    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 46. Servicio de añadir información no oficial

Después de vuelta en el controlador, al ser la imagen una parte opcional se comprueba si existe un fichero. Una vez que sabemos que hay un fichero para subir, leemos la imagen y la transformamos en un array de bytes y después los bytes se codifican utilizando el sistema de numeración posicional base64. Finalmente generamos un objeto JSON en el que introducimos el id de la información no oficial, la ruta donde se debe guardar, el nombre del fichero, y la imagen codificada. Y realizamos la llamada al servicio editar, para así añadir la imagen.

```
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);
    String filePath = "/post_no_oficiales/emergencia_no_oficial_"+post.getId();
    JSONObject edit = new JSONObject();
    edit.put("id", post.getId());
    edit.put("filePath", filePath);
    edit.put("fileName", file.getSubmittedFileName());
    edit.put("image", imagen64String);
    edit.put("web", 1);

    emNoClient.editpost(edit.toString(), emNoGeneric);
}
```

Ilustración 47. 2º Parte controlador de añadir información no oficial

El servicio recoge los parámetros del JSON, descodifica la imagen y mediante la API de Dropbox que mi compañero Sergio Rodríguez ha preparado subimos el fichero. Finalmente, el sistema nos redirigirá a la página de la emergencia de nuevo, donde podremos ver una vez validada, la información que hemos aportado.

```
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;
    if(json.isNull("web")){
        try {
            image = imageAsText.getBytes("ISO-8859-1");
        } catch (UnsupportedEncodingException ex) {
            Logger.getLogger(EmergenciaNooficialFacadeREST.class.getName()).log(Level.SEVERE, null, ex);
        }
    }else{
        image = Base64.decodeBase64(imageAsText);
    }

    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);
    }
```

Ilustración 48. Servicio de editar la emergencia no oficial para la imagen

4.2.3.4 Validar información no Oficial

Para realizar esta funcionalidad el usuario debe estar autentificado y pulsar sobre el botón de “Validar Información”

En el controlador el sistema cargará la lista de Emergencias que están activas, y redirigirá a la vista de validar información.

```
EmergencyClient eClient = new EmergencyClient();
GenericType<List<Emergencia>> genericType = new GenericType<List<Emergencia>>() {};

List<Emergencia>  emergenciaList = eClient.findNotArchivedEmergencies(genericType);

request.setAttribute("eList", emergenciaList);

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

Ilustración 49. Controlador de ver la lista de emergencias no archivadas

Una vez llega a la vista el sistema muestra todas las emergencias y un botón con forma de flecha que dará paso a cargar la información no oficial pendiente de validar. Esta vista siempre será actualizada automáticamente cada cierto periodo de tiempo para que el usuario que realiza la función de validar siempre vea información actualizada.

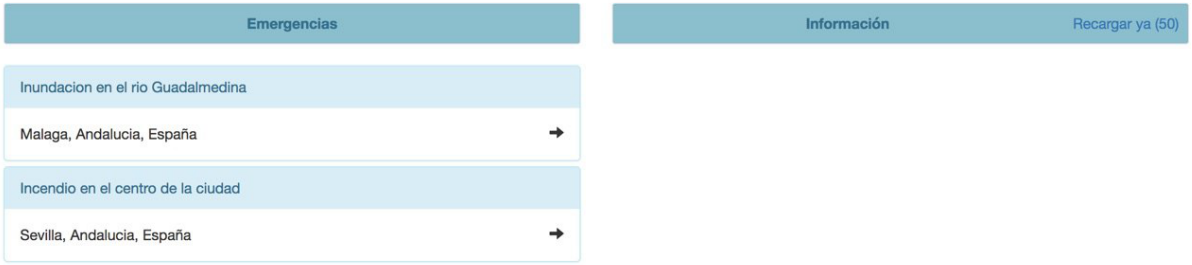

Ilustración 50. Vista de la lista de emergencias activas sin seleccionar

Cuando el usuario pulse sobre el botón el sistema llamará por JavaScript a la función “searchPost” que mediante una petición AJAX construirá la lista de información no oficial pendiente de validar

```
function searchPost(emergencia){

    $.ajax({
        type:'POST',
        url: base_url + 'postToValidate',
        data: {
            'id_emergencia': emergencia
        },
        success: function(output){

            $('#lista').empty();
            $('#id_emergencia').val(output.id_emergencia);
            for(var i=0; i<output.post.length; i++){

                var post = createPost(output.post[i]);
                $('#lista').append(post);
            }
        }
    });
}
```

Ilustración 51. Petición AJAX para obtener la información no oficial sin validar

Una vez el usuario ha obtenido la lista, en cada información hay 2 botones, uno para validar e introducir la información en el sistema, y otro para borrar toda aquella información que no aporte utilidad o pueda llegar a ser ofensiva.

Ilustración 52. Vista de la información no oficial sin validar

Cuando el usuario pulsa el botón llama a la función JavaScript que nuevamente mediante una petición AJAX llama al controlador encargado de validar la información.

```
function validate(id_post){
    var id_emergencia= $('#id_emergencia').val();
    $.ajax({
        type:'POST',
        url: base_url + 'validateNoOfficial',
        data: {
            'id_post': id_post,
            'validate': 1,
            'id_emergencia': id_emergencia
        },
        success: function(output){

            $('#lista').empty();

            for(var i=0; i<output.post.length; i++){

                var post = createPost(output.post[i]);
                $('#lista').append(post);
            }
        }
    });
}
```

Ilustración 53. Petición AJAX para validar información no oficial

Tanto para validar como para eliminar ambas peticiones llaman al mismo controlador, ya que mandan un parámetro para saber si deben validar o no la información para el sistema. Primero mediante el token identificativo, obtenemos al usuario que va a validar la información. A continuación, utilizando los parámetros que recibimos en la petición los introducimos en un objeto JSON, y generamos la petición al servicio de la API

```
String id_emergencia = request.getParameter("id_emergencia");

int id_post = Integer.parseInt(request.getParameter("id_post"));
int validate = Integer.parseInt(request.getParameter("validate"));

String token = (String)session.getAttribute("token");

JSONObject jsonToken = new JSONObject();
jsonToken.put("token", token);

UserClient userClient = new UserClient();

GenericType<Usuario> genericType = new GenericType<Usuario>() {};
Usuario u = userClient.getUserFromSesion(jsonToken.toString(), genericType);

JSONObject jsonRequest = new JSONObject();

jsonRequest.put("id_post", id_post);
jsonRequest.put("validar", validate);
jsonRequest.put("id_usuario", u.getId());

EmergencyNoOfficialClient emNoClient = new EmergencyNoOfficialClient();

emNoClient.validatePost(jsonRequest.toString());
```

Ilustración 54. Controlador de validar información no oficial

Una vez en el servicio REST recogemos el JSON de la petición y con sus valores buscamos la información y la validamos cambiando el valor de “id_usuario_validador” por el id del usuario que ha validado, (así podemos tener una trazabilidad de la información validada por un usuario), se actualiza en la base de datos y se elimina si la hubiera las imágenes de Dropbox.

```
JSONObject json = new JSONObject(data);

int id_post = (Integer)json.get("id_post");
int validar = (Integer)json.get("validar");
int u  = (Integer)json.get("id_usuario");

EmergenciaNooficial post = find(id_post);

if(validar == 1){
    post.setIdUsuarioValidador(u);

    edit(id_post, post);
}else{
    if(post.getFoto()!=null){
        try {
            clientDropbox.delete(post.getFoto());
        } catch (DbxException ex) {
            Logger.getLogger(EmergenciaNooficialFacadeREST.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    remove(id_post);
}
```

Ilustración 55. Servicio de validar información no oficial

Para finalizar, después de validar la información no oficial, el controlador carga de nuevo la lista de información no validada y la envía para que en el succes de la petición AJAX se reconstruya la lista actualizada y así el usuario pueda seguir trabajando.

4.2.3.5 Añadir información Oficial

Para el caso de la información oficial es prácticamente igual que la no oficial solo que cambian los requisitos, ya que el usuario debe estar autentificado y tener un rol de miembro o superior.

Los campos a rellenar también son diferentes, aquí se decide que prioridad tiene, ya que de ello depende en que posición aparecerá en la lista y la descripción de la información, en este caso la imagen es también opcional. Una vez el usuario pulsa el botón “Enviar”, esta se almacena en base de datos y se muestra directamente, ya que no tiene que ser validada.

Ilustración 56. Ventana emergente para añadir información oficial

Finalmente, el controlador sigue el mismo flujo que cuando es una información no oficial, se recogen los parámetros y se llama al servicio para crear el objeto y añadirlo en base de datos. Una vez creado, se comprobará si hay una imagen, si es el caso se realizará el proceso de obtenerla y se llamará al servicio para añadirla a Dropbox. Para terminar, volverá a la vista de la emergencia con la información ya actualizada.

4.2.3.6 Archivar emergencia

Para realizar esta funcionalidad el usuario debe estar en la vista con una lista de emergencias y tener un rol de coordinador o superior. Una vez allí pulsa sobre el botón archivar (icono del fichero), el sistema mostrará un mensaje confirmando si quiere realizar la acción. El usuario pulsa el boton aceptar y mediante una peticion AJAX el sistema realizara una llamada al controlador “ArchiveEmergency”.

```
function archivar(id_emergencia){
    swal({
        title: "Estas seguro?",
        text: "Una vez archivada la Emergencia no se podra reactivar!",
        type: "warning",
        showCancelButton: true,
        confirmButtonColor: "#5CB85C",
        confirmButtonText: "Aceptar",
        cancelButtonText: "Cancelar",
        closeOnConfirm: false
    },
    function(){
        $.ajax({
            type:'POST',
            url: base_url + 'archiveEmergency',
            data: {
                'id_emergencia':id_emergencia
            },
            success: function(output){
                $('#lista').empty();
                for(var i=0; i<output.emergencias.length; i++){

                    var emergency = createRowEmergency(output.emergencias[i]);
                    $('#lista').append(emergency);
                }
                swal("Archivado!", "La emergencia ha sido archivada.", "success");
            }
        });
```

Ilustración 57. Petición AJAX para archivar una emergencia

Mediante el id de la Emergencia el sistema actualizará el objeto para poner el valor de “archivada” a true. A continuacion obtendra la lista de emergencias no archivadas actualizada y la introducirá en un objeto JSON que será enviado al succes de la petición.

```
String id_emergencia = request.getParameter("id_emergencia");

EmergencyClient emClient = new EmergencyClient();

GenericType<Emergencia> genericEM = new GenericType<Emergencia>() {};
Emergencia em = emClient.find(genericEM, id_emergencia);

em.setArchivada(true);

emClient.edit(em, id_emergencia);

GenericType<List<Emergencia>> genericType = new GenericType<List<Emergencia>>() {};
List<Emergencia> emergenciaList = emClient.findNotArchivedEmergencies(genericType);

JSONObject output = new JSONObject();
output.put("emergencias", emergenciaList);

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

Ilustración 58. Controlador de archivar una emergencia

Finalmente el succes de la funcion AJAX recogerá la lista y mediante un bucle “for” reconstruirá emergencia a emergencia la lista actualizada.

4.2.4 Gestión de los usuarios

Para cumplir con los requisitos y casos de uso asociados a estas funcionalidades se han desarrollado los siguientes servicios y clases. Este apartado supone los casos de uso: U7 y U8.

4.2.4.1 Añadir usuario

Para acceder a esta funcionalidad el usuario debe estar autentificado y tener un rol de Coordinador o superior. El usuario deberá pulsar en el desplegable de “Usuarios” y elegir la opcion de “Añadir usuario”. El sistema redirigirá a la vista correspondiente para añadir usuario.

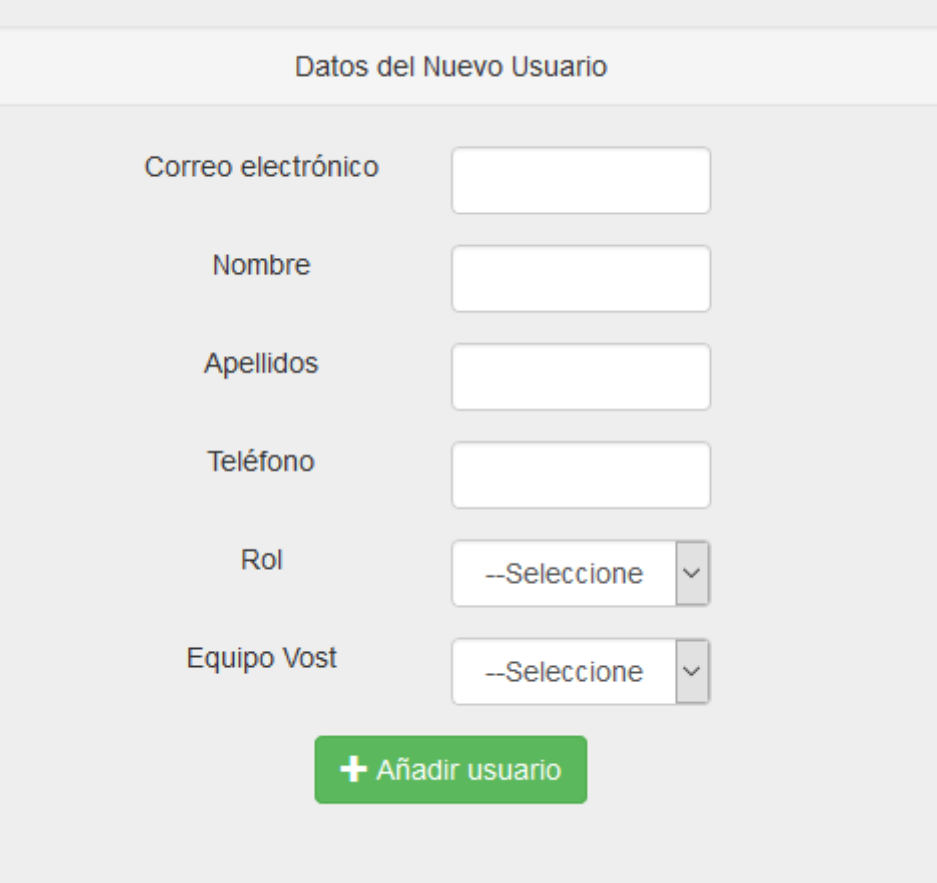

Ilustración 59. Vista de añadir usuario

Una vez allí el usuario completará los distintos campos y pulsará sobre el botón “Añadir Usuario”. Esto desencadenará la llamada al controlador “NewUser”. Una vez en el controlador, este recoge todos los parámetros y los introduce en un objeto JSON que será enviado en la petición del servicio REST.

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

JSONObject user = new JSONObject();
user.put("email", email);
user.put("name", nombre);
user.put("surname", apellidos);
user.put("phone", telefono);
user.put("rol", rol);
user.put("equipoVost", equipoVost);

UserClient client = new  UserClient();
client.newuser(user.toString());

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

Ilustración 60. Controlador de añadir usuario

Una vez en el servicio se recogen todos los valores del nuevo usuario y se genera una contraseña aleatoria. A continuación se crea un objeto usuario y se le asignan todos los nuevos valores y el hash de su contraseña, que serán almacenados en la base de datos. Cuando el usuario ya esta dado de alta se le envía un correo electrónico, recordandole que su usuario es su correo electrónico y dándole su contraseña, para que una vez que acceda al sistema, este la cambie.

```
String email = (String)json.get("email");
String password = generateRandomString().substring(0, 8);
String nombre = (String)json.get("name");
String apellidos = (String)json.get("surname");
String telefono = (String)json.get("phone");
int rol = Integer.parseInt((String)json.get("rol"));
int equipoVost = Integer.parseInt((String)json.get("equipoVost"));

byte[] bytesOfMessage = password.getBytes("UTF-8");
MessageDigest md = MessageDigest.getInstance("MD5");
byte[] thedigest = md.digest(bytesOfMessage);


String passCifrada = Util.bytesToHexString(thedigest).toLowerCase();
Date ultConexion = new Date();

Usuario u = new Usuario();
u.setEmail(email);
u.setNombre(nombre);
u.setApellidos(apellidos);
u.setTelefono(telefono);
u.setPassword(passCifrada);
u.setRol(rol);
u.setIdEquipo(equipoVost);
u.setActivo(true);
u.setUltConexion(ultConexion);

create(u);

String message = "Su usuario ha sido creado con éxito. Recuerde, su usuario es " + email + " y su contraseña es " + password;
sendEmail(email, message);
```

Ilustración 61. Servicio de añadir usuario

4.2.4.2 Ver la lista de usuarios

Esta funcionalidad es exclusiva del Administrador el usuario pulsa sobre el desplegable de “Usuarios” y elige la opcion de “Lista de Usuarios”. Esto desencadena la llamada al controlador “ListaUsuarios”.

En el controlador mediante el cliente de Usuarios y Sesión, obtenemos la lista de Usuarios, la lista de voluntarios, y la lista de sesiones activas. Se comprueba el rol del usuario, ya que si el usuario es Coordinador este solo puede ver aquellos usuarios pertenecientes a su equipo. A continuación se enviaran las listas obtenidas como atributos y se mostraran en la vista con lo que obtenemos dos pestañas.

```
String id_equipo = request.getParameter("id_equipo");
UserClient pClient = new UserClient();
SesionClient sClient = new SesionClient();

GenericType<List<Usuario>> genericType = new GenericType<List<Usuario>>() {};
List<Usuario> usuariosList = null;
List<Usuario> voluntariosList = null;

if(id_equipo!=null){
    //Viene de lista de equipos
    usuariosList = pClient.findUsersByTeam(genericType, id_equipo);
    voluntariosList = pClient.findVolunteersByTeam(genericType, id_equipo);
}else{
    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);
    }
}
GenericType<List<Sesion>> genericTypeSesion = new GenericType<List<Sesion>>() {};
List<Sesion> sesionList = sClient.findAll(genericTypeSesion);

request.setAttribute("userList", usuariosList);
request.setAttribute("voluntariosList", voluntariosList);
request.setAttribute("sessionList", sesionList);
```

Ilustración 62. Controlador de la lista de usuarios

La pestaña de Usuarios donde podemos ver todos los usuarios, su última conexión y su información básica.

Ilustración 63. Vista de la lista de usuarios

Y por otro lado la pestaña de Voluntarios, donde podemos realizar la acción de activar o desactivar un voluntario.

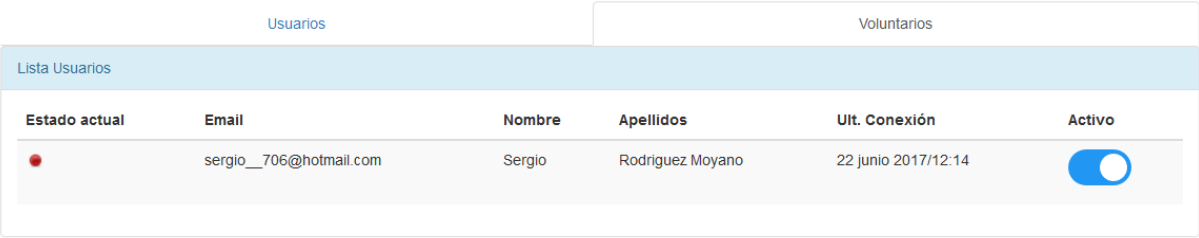

Ilustración 64. Vista de la lista de voluntarios

5. Conclusiones y líneas futuras

Desde el primer día nos enfrentamos al Trabajo de fin de Grado con respeto, sabíamos lo importante que sería, pero estábamos motivados como para intentar dejar nuestra huella en este proyecto. Este era el paso final a dar dentro de nuestro aprendizaje en estos años en la universidad y la oportunidad para demostrar todo aquello que hemos aprendido.

La organización del trabajo, al ser en equipo, ha sido siempre mediante consenso, intentando repartir nuestras tareas de la forma más equitativamente posible. En primera instancia pensamos en realizar todo el trabajo de captura de requisitos, desarrollo de los diagramas, y el desarrollo de la API conjuntamente; posteriormente Sergio se encargaría del cliente móvil y yo del cliente web, considerando que ambos tendrían las mismas funcionalidades. Pero a medida que fuimos modelando el proyecto y hablando con el cliente vimos que la aplicación móvil tendría funcionalidades más simples, por lo que Sergio también ha tomado parte en el desarrollo del cliente web en algunas de sus implementaciones.

Así mismo ambos hemos formado un buen grupo ya que nos conocemos desde primer año de la carrera. Además, siempre que alguno de los dos se ha sentido bloqueado en sus tareas hemos podido aportar nuestra visión y completarlas de manera satisfactoria.

El desarrollo del proyecto nos ha servido para ver que podemos ser capaces de aprender nuevas tecnologías por nosotros mismos, aunque no siempre ha sido fácil. Algunos plugin han sido complejos de utilizar, por su mala documentación, pero con darte un tiempo para buscar soluciones es posible reparar esos problemas que van surgiendo.

5.1 Líneas futuras

Este proyecto ha sido desde el primer día planteado como una base para que en el futuro pueda ser continuado. Las grandes aportaciones que se pueden realizar a este proyecto pueden ser:

- Aunque a día de hoy ha sido planteada a nivel estatal, esta puede ser perfectamente ampliable a nivel europeo.
- Las mejoras de la ubicación de las emergencias, dando la posibilidad, de en un gran mapa, poder dibujar las emergencias y sus zonas de peligro.
- La creación de un chat interno entre miembros de un mismo equipo para favorecer la comunicación, así mismo como entre coordinadores para facilitar la ayuda ante una emergencia.
- Dar de alta usuarios temporales a aquellas administraciones ajenas a la plataforma, para que en un momento de necesidad puedan aportar información.

6. Bibliografía

https://www.w3schools.com/

https://www.w3schools.com/bootstrap/

https://stackoverflow.com/

https://www.tinymce.com/docs/

https://github.com/mohan/tinymce-placeholder

http://api.jquery.com/

http://t4t5.github.io/sweetalert/