**Proyecto de desarrollo de aplicaciones multiplataforma**

**WhiskerWatch**

CICLO FORMATIVO DE GRADO SUPERIOR

**Desarrollo de Aplicaciones Multiplataforma (IFCS02)**

**Curso 2022-23**

Autor/a/es:

**Rocío Palao Fernández**

Tutor/a:

**José Luis González Sánchez**

Departamento de Informática y Comunicaciones

**I.E.S. Luis Vives**

Contenido

[2 Introducción 4](#_Toc136681949)

[3 Requisitos 5](#_Toc136681950)

[3.1 Requisitos Funcionales 5](#_Toc136681951)

[3.2 Requisitos No Funcionales 5](#_Toc136681952)

[3.3 Requisitos de Información 5](#_Toc136681953)

[4 Diagrama de Clases 6](#_Toc136681954)

[5 Diagrama de Casos de Uso 7](#_Toc136681955)

[5.1 Usuarios 7](#_Toc136681956)

[5.2 Administradores 7](#_Toc136681957)

[5.3 Asociaciones 8](#_Toc136681958)

[6 Competencia 9](#_Toc136681959)

[7 Elección Tecnológica 10](#_Toc136681960)

[7.1 Creación API Rest 10](#_Toc136681961)

[7.1.1 Spring 10](#_Toc136681962)

[7.1.2 Ktor 10](#_Toc136681963)

[7.2 Base de datos 11](#_Toc136681964)

[7.2.1 MySQL 11](#_Toc136681965)

[7.2.2 MariaDB 11](#_Toc136681966)

[7.2.3 Postgresql 11](#_Toc136681967)

[7.2.4 MongoDB 12](#_Toc136681968)

[7.3 Front-end 13](#_Toc136681969)

[7.3.1 Flutter 13](#_Toc136681970)

[7.3.2 Jetpack Compose 13](#_Toc136681971)

[7.3.3 Angular 13](#_Toc136681972)

[7.3.4 VueJS 14](#_Toc136681973)

[7.3.5 React 14](#_Toc136681974)

[7.4 Inyección de Dependencias 15](#_Toc136681975)

[7.4.1 Koin VS Dagger 15](#_Toc136681976)

[8 Back-end 16](#_Toc136681977)

[8.1 Modelos de Datos 16](#_Toc136681978)

[8.1.1 Usuarios 16](#_Toc136681979)

[8.1.2 Asociaciones 16](#_Toc136681980)

[8.1.3 Mapas 16](#_Toc136681981)

[8.1.4 Foro 17](#_Toc136681982)

[8.1.5 Mensajes Foro 17](#_Toc136681983)

[8.1.6 Calendario 17](#_Toc136681984)

[8.1.7 Tareas 17](#_Toc136681985)

[8.1.8 Notificaciones de adopción 17](#_Toc136681986)

[8.2 Repositorios 18](#_Toc136681987)

[8.2.1 Usuarios 18](#_Toc136681988)

[8.2.2 Asociaciones 19](#_Toc136681989)

[8.2.3 Mapas 20](#_Toc136681990)

[8.2.4 Foro 21](#_Toc136681991)

[8.2.5 Calendario 22](#_Toc136681992)

[8.3 Rutas 23](#_Toc136681993)

[8.3.1 Usuarios 23](#_Toc136681994)

[8.3.2 Asociaciones 24](#_Toc136681995)

[8.3.3 Mapas 26](#_Toc136681996)

[8.3.4 Foro 28](#_Toc136681997)

[8.3.5 Calendario 29](#_Toc136681998)

[8.4 Railway Programming 30](#_Toc136681999)

[8.5 Reactividad y Asincronía 30](#_Toc136682000)

[8.6 Seguridad 30](#_Toc136682001)

[8.6.1 SSL 30](#_Toc136682002)

[8.6.2 JWT 31](#_Toc136682003)

[8.7 Almacenamiento 32](#_Toc136682004)

[8.7.1 Guardar Imagen 32](#_Toc136682005)

[8.7.2 Mostrar Imagen 32](#_Toc136682006)

[8.7.3 Eliminar Imagen 32](#_Toc136682007)

[8.8 Tests 33](#_Toc136682008)

[8.8.1 Postman 33](#_Toc136682009)

[8.8.2 Junit y Mockit 33](#_Toc136682010)

[8.9 WebSockets y Notificaciones en tiempo real 34](#_Toc136682011)

[9 Front-end 35](#_Toc136682012)

# Introducción

Este proyecto principalmente se trata de una aplicación de localización de colonias de gatos callejeros.

En ella puede haber asociaciones de adopción los cuales tienen su apartado para difundirse, en el cuál pueden poner una imagen de su asociación con una descripción detallada de lo que realizan y un enlace a su página web, este dato es importante ya que esta aplicación puede ayudarles a rescatar gatos de la calle para poder buscarles una mejor vida y hogar.

Diagrama

Descripción generada automáticamentePor cada punto en el mapa de localización de las colonias tendremos un calendario de ayudas, para poner las tareas que se necesitan cierto día, en el que las personas que quieran ayudar puedan ir.

También para cada punto en el mapa tendremos un foro, por si ocurre algo en esa localización como por ejemplo un desperfecto en los bebederos de los gatos, que esté notificado en algún lugar y que también puedan poner sus opiniones sobre esa localización.

Podremos desde nuestro punto actual trazar una ruta para poder llegar a la localización de la colonia que deseemos.

Notificaciones en tiempo real cuando se realiza una recogida de colonia realizada por una asociación para buscarles un hogar.

Los usuarios se podrán registrar en la aplicación, dependiendo de si son usuarios normales o asociaciones tendrán unas características diferentes.

Todos los usuarios podrán modificar su perfil e incluso eliminar la cuenta si lo desean.

El objetivo de esta aplicación es para la ayuda y mejora de vida de los gatos callejeros.

# Requisitos

## Requisitos Funcionales

* Log-in a la aplicación.
* Registro a la aplicación.
* Eliminación de la cuenta.
* Modificación de datos de la cuenta.
* Cambiar imagen asociaciones.
* Señalización de colonias de gatos.
* Agregar localización de colonia de gatos.
* Eliminación de localización de colonia de gatos.
* Muestra de calendario de comidas de una localización concreta.
* Agregar comida al calendario.
* Muestra de asociaciones de adopción.
* Notificación de recogida de colonia por asociación.
* El registro no es el mismo para los usuarios y las asociaciones.
* Los usuarios no pueden notificar recogidas.
* Todos los usuarios deben haberse registrado / iniciado sesión para poder utilizar la aplicación.
* Solamente las asociaciones pueden eliminar del mapa la localización por recogida de colonia.
* Solamente administradores de la aplicación pueden eliminar del mapa la localización.
* Las contraseñas deben tener un mínimo de 6 caracteres.

## Requisitos No Funcionales

* La aplicación debe ser multiplataforma.
* Back-end desarrollado en Kotlin.
* La seguridad del back-end con JWT Bearer.
* Notificaciones de recogidas en tiempo real.

## Requisitos de Información

El proyecto tendrá 5 entidades:

* Usuarios
* Asociaciones
* Mapas
* Calendarios
* Foros

# Diagrama de Clases

Diagrama

Descripción generada automáticamente

En Whisker Watch tenemos localizaciones, cada localización tiene un foro y un calendario asignado, donde los usuarios pueden poner tareas (calendario) y mensajes (foro). Tenemos usuarios y asociaciones los cuales pueden consultar una localización y su respectivo, calendario y foro.

# Diagrama de Casos de Uso

## Usuarios

Diagrama

Descripción generada automáticamente

## Administradores

A parte de poder realizar todas las acciones que tienen permitidos los usuarios, también pueden:

Diagrama

Descripción generada automáticamente

## Asociaciones

Las asociaciones también pueden realizar las acciones que tienen los usuarios permitidos, pero también pueden:

Diagrama

Descripción generada automáticamente

# Competencia

Esta aplicación es sobre una especie muy concreta, de competencia he encontrado la siguiente información:

**Vaca de pie sobre superficie terrosa

Descripción generada automáticamente con confianza mediaDigitanimal**

Trata sobre un sistema de GPS para tener el ganado localizado en todo momento, para ello cada animal lleva un dispositivo atado a el cuello con el cual se puede saber en todo momento la localización del animal.

Con un recinto virtual en el que se indica por donde se podrán mover los animales y cuando estos pasan de ese límite el usuario recibe una alerta en la aplicación.

Además cada dispositivo nos muestra la información del animal que lo lleva puesto, nombre, sexo, raza, etc...

<https://digitanimal.com/ganaderia-extensiva/>

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

**Animal Tracker**

Esta aplicación es para que los usuarios puedan seguir la pista a la fauna y puedan ayudar a la comunidad científica consiguiendo imágenes de los ejemplares de animales, no son todos los animales ni todas las especies, sino unos animales de ciertas especies a los que se les ha colocado un GPS.

También te muestra información del animal y la ruta que toma a lo largo de un año.

Además también puedes marcar animales como favoritos y luego ver tu lista de favoritos.

<https://www.greenappsandweb.com/android/pista-a-la-fauna-animal-tracker/#:~:text=Animal%20Tracker%20es%20una%20aplicaci%C3%B3n,momento%20est%C3%A1n%20incluidas%2017%20especies>).

# Elección Tecnológica

## Creación API Rest

Para la creación de la API Rest tenemos dos opciones:

### Spring

Es el framework más utilizado en Java.

* Ventajas:

- Utiliza el patrón de inyección de dependencias.

- Está dividido en módulos que nos ayudan a gestionar diferentes necesidades.

- Permite el desarrollo de aplicaciones flexibles y escalables, por lo que el mantenimiento del programa siempre será más fácil.

Icono

Descripción generada automáticamente - Está basado en la programación orientada a interfaces.

* Desventajas:

- Cuando hay actualizaciones del framework cuesta mucho trabajo migrarlo a la siguiente versión.

- Es complejo para poder utilizarlo.

- Si no tienes experiencia, cuesta un poco aprenderlo.

- Utiliza muchos XML.

### Ktor

Framework para crear servidores y clientes asíncronos creado por JetBrains.

* Ventajas:

- Código abierto.

- Fácil de empezar con Ktor.

- Utiliza DSL.

- Ligero.

- Flexible.

- Multiplataforma.

- Uso de corrutinas.

- Buena documentación.

* Desventajas:

- Solamente se puede utilizar el lenguaje Kotlin.

- Muy nuevo.

Para la creación de la API Rest he escogido ktor.

Porque es ligero, flexible y también es multiplataforma, usa el lenguaje kotlin y es fácil de empezar a utilizarlo.

Además es un framework que he utilizado anteriormente y me gustaría poder mejorar y descubrir más sobre él.

## Base de datos

### MySQL

Es una base de datos relacional, actualmente uno de los más utilizados y de código abierto.

* Ventajas:

- Es multiplataforma

- Requiere de pocos recursos (Hardware)

-Tiene un buen rendimiento en volúmenes de datos medianos o pequeños.

Imagen que contiene dibujo, alimentos

Descripción generada automáticamente- Realizar las consultas rápidamente.

- Baja probabilidad de corromper los datos.

- Esquema fijo.

- Seguro.

- Soporte de transacciones.

* Desventajas:

- En las modificaciones de las estructuras de la BDD pueden aparecer pequeños errores.

- Mal rendimiento con cantidades de datos muy grandes.

- No es intuitivo, se necesita un estudio previo para poder entenderlo.

### MariaDB

Es una versión mejorada de MySQL

* Imagen que contiene Forma

  Descripción generada automáticamente Ventajas:

- Mejor rendimiento con cantidades de datos más grandes.

- Mas seguridad de MySQL.

- Código abierto.

- Es más fácil de usar que MySQL.

* Desventajas:

- No es compatible con todas las aplicaciones.

- Poca ayuda de soporte.

### Postgresql

* Ventajas:

Logotipo, nombre de la empresa

Descripción generada automáticamente- Gratuito.

- Sencillo de utilizar.

- Multiplataforma.

- Gran rendimiento.

- Es un sistema robusto y fiable.

- Escalable.

* Desventajas:

- Rendimiento lento para bases de datos pequeñas.

- No tiene buen soporte técnico.

- Necesario tener una buena base del lenguaje SQL.

### MongoDB

Es una base de datos NoSQL ( no sólo SQL )

* Ventajas:

- Necesita pocos recursos.

- Código abierto.

- Buena documentación.

- Esquema no fijo.

- Escalable.

- Flexible por su buena adaptación a evoluciones.

Logotipo, nombre de la empresa

Descripción generada automáticamente- Optimización en las consultas con grandes cantidades de datos.

- Basado en JavaScript

* Desventajas:

- No tiene joins para las consultas.

- No admite todas las consultas SQL.

- No es correcta para las transacciones complejas.

Para el proyecto he elegido MongoDB

Ya que es escalable por lo que cuando aumenten los datos de localización no habrá tantos problemas, son más rápidas, son flexibles ya que las bases de datos NoSQL no siguen ningún esquema y tiene una alta disponibilidad por lo que cuando haya muchos accesos a la base de datos el rendimiento no se verá tan afectado como podría estarlo con otras bases de datos.

## Front-end

### Flutter

* Ventajas:

Logotipo, nombre de la empresa

Descripción generada automáticamente- Rápido desarrollo.

- Recarga de los cambios en caliente.

- Multiplataforma.

- Utiliza librerías de widgets.

- Rendimiento nativo.

- Open Source.

* Desventajas:

- Lenguaje de programación Dart.

- Pesa más que otros frameworks o aplicaciones nativas.

- Puede haber dificultad para integrar en el proyecto librerías nativas de terceros.

- Limitación de integración con hardware.

### Jetpack Compose

* Ventajas:

- Acelera el desarrollo de Interfaz de Usuario.

Icono

Descripción generada automáticamente- Adaptabilidad del dispositivo.

- Se usa menos código.

- Liberia independiente.

- Compatible con código XML.

- Compatible con todo el código existente.

* Desventajas:

- Problemas con el tiempo de compilación en la interoperabilidad.

- Problemas con el tamaño de aplicación en la interoperabilidad.

- Al ver la vista previa siempre tiene que renderizar.

- Algunos componentes no son compatibles.

- Más lento que XML para representar un cambio.

### Angular

* Ventajas:

- Utiliza TypeScript.

- Desarrollo rápido.

- Tiene compatibilidad para el desarrollo de aplicaciones móviles o de escritorio.

Una señal de alto

Descripción generada automáticamente con confianza baja- Desarrollo web más rápido.

- Soporte de Google.

- Pruebas sencillas de escribir.

- Productivo.

- Más opciones de lenguajes para escribir el código.

* Desventajas:

- Puede llegar a ser complejo.

- Puede tener un aprendizaje un poco difícil.

- Aplicaciones relativamente grandes.

### VueJS

* Ventajas:

- Pesa poco.

- División en componentes individuales.

- Fácil de aprender.Logotipo, nombre de la empresa

Descripción generada automáticamente

- Fácil de utilizar.

- Reactividad.

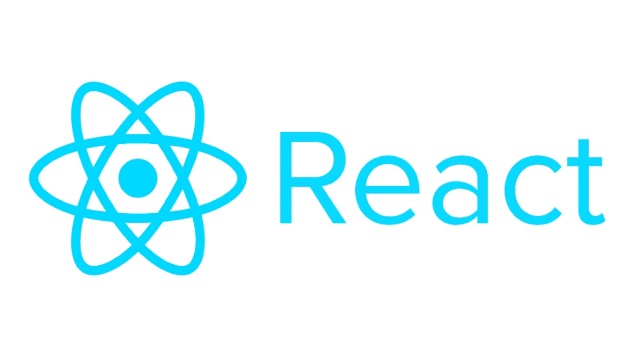
* Desventajas:

- Necesario combinarlo con Electron para hacer aplicaciones de escritorio y móviles.

### React

* Ventajas:

- Acelera el desarrollo de las aplicaciones.

- JavaScript.

- Código abierto.

- Recarga de cambios en caliente.

- Buen rendimiento.

- Apariencia nativa.

* Desventajas:

- Problemas para depurar.

- Necesario desarrolladores nativos.

- Depende de su desarrollador Facebook.

- No muy buena gestión de la memoria.

- Actualización lenta de funciones recientes.

- Mal rendimiento con funciones complejas.

Para el front del proyecto utilizaré Angular ya que es un framework muy utilizado en españa, utiliza typescript, es de desarrollo rápido, tiene mucha comunidad y me puede servir para el futuro.

## Inyección de Dependencias

### Koin VS Dagger

Dagger es más eficiente que koin ya que genera el código necesario para la inyección de dependencias en tiempo de compilación mientras que koin lo genera en tiempo de ejecución. Por lo que las aplicaciones con koin tardarán más en ejecutarse.

Dagger es más complejo de aprender ya que es un inyector de dependencias y necesita mucha configuración antes de empezar a usarlo, mientras que koin es un localizador de servicios y es mucho más fácil de aprender.

En un localizador de servicios cuando tenemos nuevas dependencias, tendremos que añadir manualmente como se debe generar la dependencia, mientras que en un inyector de dependencias al principio es muy complejo pero el resto del tiempo no tendremos que tocar casi nada de las clases que inyectan las dependencias.

He elegido Koin con Anotaciones porque ayuda a crear lo necesario para la inyección de dependencias muy fácilmente, se entiende, es muy rápido e intuitvo.



# Back-end

## Modelos de Datos

### Interfaz de usuario gráfica, Texto, Aplicación Descripción generada automáticamenteUsuarios

El usuario tiene, nombre, contraseña (cifrada), su nombre de usuario que usaremos para poner en los mensajes que escriban en los foros, correo con el que iniciarán sesión, id (Identificador único del usuario) y el rol con el que podremos saber si es administrador o no.

### Imagen de la pantalla de una computadora Descripción generada automáticamente con confianza bajaAsociaciones

La asociación tiene, el nombre, la descripción de la asociación, la contraseña (cifrada), la imagen de la asociación, la url de su web/ redes sociales, el email con el que iniciarán sesión, el rol con el que nos indican que son una asociación y el id (identificador único de la asociación)

### Mapas

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Los mapas tienen el id (identificador único de mapa) y la longitud y latitud que es lo que se necesita para calcular la localización en un mapa.

### Foro

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

El foro tiene el id (identificador único de foro), el id del mapa al que le asignamos este foro y la lista de los mensajes del foro.

### Interfaz de usuario gráfica, Texto, Aplicación Descripción generada automáticamenteMensajes Foro

Los mensajes de foro tienen el id (identificador único de mensaje de foro), el nombre de usuario del usuario que ha escrito el mensaje, el mensaje y la fecha en la que se ha creado ese mensaje.

### Interfaz de usuario gráfica, Texto, Aplicación Descripción generada automáticamenteCalendario

El calendario tiene el id (identificador de calendario), el id del mapa al que le asignamos este calendario y la lista de tareas que hay en ese calendario.

### Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto Descripción generada automáticamenteTareas

En las tareas tenemos el id (identificador único de la tarea), la tarea que se va a realizar y la fecha en la que se tiene que realizar.

### Notificaciones de adopción

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Esta clase es para las notificaciones en tiempo real sobre si una colonia ha sido recogida por una asociación para encontrarles una vida mejor.

Esta clase tiene el mapa el cual ha sido quitada la colonia y la fecha en la que se ha recogido la colonia.

## Repositorios

### Usuarios

Captura de pantalla de computadora

Descripción generada automáticamente

- **findByEmail**: Buscamos un usuario por un email dado, puede devolver el usuario si lo ha encontrado o nulo.

- **findById**: Para buscar un usuario por un id dado, puede devolver el usuario si lo ha encontrado o nulo.

- **save**: Para guardar un usuario, guarda el usuario y devuelve el usuario guardado.

- **update**: Para actualizar un usuario que ya está guardado en la base de datos, devuelve el usuario actualizado.

- **delete**: Eliminar un usuario por su id, devuelve si se ha eliminado correctamente.

- **findAll**: Buscar todos los usuarios que hay almacenados en la base de datos, devuelve un flujo de usuarios.

- **deleteAll**: Para eliminar todos los usuarios almacenados en la base de datos, devuelve si se ha eliminado correctamente.

### 

### Asociaciones

Captura de pantalla de computadora

Descripción generada automáticamente con confianza media

- **findByEmail**: Buscar una asociación por un email dado, devuelve la asociación si se encuentra o nulo.

- **findById**: Buscar una asociación por un id dado, devuelve la asociación si se encuentra o nulo.

- **save**: Guardar en la base de datos una asociación, devuelve la asociación guardada.

- **update**: Actualizar una asociación que ya está almacenada en la base de datos, devuelve la asociación actualizada.

- **delete**: Eliminar una asociación de la base de datos, devuelve si se ha eliminado la asociación correctamente.

- **findAll**: Buscar todas las asociaciones que hay almacenadas en la base de datos, devuelve un flujo de asociaciones.

- **deleteAll:** Eliminar todas las asociaciones almacenadas en la base de datos, devuelve si se ha eliminado correctamente.

### Mapas

Texto

Descripción generada automáticamente

- **findById**: Buscar un mapa por un id dado, devuelve un mapa si se encuentra o nulo.

- **save**: Guardar un mapa en la base de datos, devuelve el mapa guardado.

- **update**: Actualizar un mapa que ya está almacenado en la base de datos, devuelve el mapa actualizado.

- **delete**: Eliminar un mapa de la base de datos, devuelve si se ha eliminado correctamente.

- **findAll**: Buscar todos los mapas almacenados en la base de datos, devuelve un flujo de mapas.

Texto

Descripción generada automáticamente- **deleteAll**: Eliminar todos los mapas almacenados en la base de datos, devuelve si se han eliminado correctamente.

- **findMapByLatLon**: Para buscar un mapa por si latitud y longitud, devuelve si se ha encontrado o nulo.

### Foro

Captura de pantalla de computadora

Descripción generada automáticamente- **findByMapsId:** Buscar un foro por un id de mapa, devuelve el foro si se ha encontrado o nulo.

- **findById:** Buscar un foro por un id, devuelve el foro si se ha encontrado o nulo.

- **save**: Guardar en la base de datos un foro, devuelve el foro guardado.

- **update:** Actualizar un foro que ya está almacenado en la base de datos, devuelve el foro actualizado.

- **delete:** Eliminar un foro de la base de datos, devuelve si se ha eliminado correctamente.

- **findAll:** Buscar todos los foros almacenados en la base de datos, devuelve un flujo de foros.

- **deleteAll:** Eliminar todos los foros almacenados en la base de datos, devuelve si se han eliminado correctamente.

### Calendario



- **findByMapsId:** Buscar un calendario por un id de mapa, devuelve el calendario si se ha encontrado o nulo.

- **findById:** Buscar un calendario por un id, devuelve el calendario si se ha encontrado o nulo.

- **save**: Guardar en la base de datos un calendario, devuelve el calendario guardado.

- **update:** Actualizar un calendario que ya está almacenado en la base de datos, devuelve el calendario actualizado.

- **delete:** Eliminar un calendario de la base de datos, devuelve si se ha eliminado correctamente.

- **findAll:** Buscar todos los calendarios almacenados en la base de datos, devuelve un flujo de calendarios.

- **deleteAll:** Eliminar todos los calendarios almacenados en la base de datos, devuelve si se han eliminado correctamente.

## Rutas

### Usuarios

Texto

Descripción generada automáticamente

Tenemos dos rutas de post, una para el registro y otra para el login, en estas dos rutas no se necesita autentificarse.

- **Register**: Para el registro recibimos unos datos específicos que si no pasan la validación serán rechazados, si son correctos, se llama al servicio para que almacene el usuario y si todo ha salido correctamente generamos el token del nuevo usuario para que pueda usar las rutas en las que se es necesario estar autentificado.

- **Login**: Para el login recibimos unos datos específicos, si estos no pasan la validación serán rechazados. Si todo es correcto pasamos a buscar en el servicio un usuario con el email que se ha recibido en los datos, si este es encontrado procedemos a ver si la contraseña es correcta y también si es correcto generamos el token.

Captura de pantalla con letras

Descripción generada automáticamente

Entramos en las rutas en las que hay que estar autenticado.

- **Get**: Para pedir todos los usuarios que tenemos almacenados.

- **Get Id**: Para pedir un usuario por el id indicado, buscamos en el servicio, si este existe lo devolvemos y sino respondemos con un No Encontrado.

Texto

Descripción generada automáticamente

- **Put**: Para actualizar un usuario por un id dado.

( Esta acción solamente la pueden realizar los administradores y los usuarios ). Recibimos unos datos específicos, realizamos la actualización del usuario, dependiendo de si sale todo correctamente devolvemos el usuario actualizado y si no dependiendo del error se le envía una respuesta u otra.

Texto

Descripción generada automáticamente

- **Delete**: Para eliminar un usuario por un id dado. ( Esta acción solamente la pueden realizar los administradores y los usuarios). Realizamos la eliminación, si todo sale correctamente no devolvemos contenido y sino devolvemos el error.

### Asociaciones

Texto

Descripción generada automáticamente

Tenemos dos rutas de post, una para el registro y otra para el login, en estas dos rutas no se necesita autentificarse.

- **Register**: Para el registro recibimos unos datos específicos que si no pasan la validación serán rechazados, si son correctos, se llama al servicio para que almacene la asociación y si todo ha salido correctamente generamos el token de la nueva asociación para que pueda usar las rutas en las que se es necesario estar autentificado.

- **Login**: Para el login recibimos unos datos específicos, si estos no pasan la validación serán rechazados. Si todo es correcto pasamos a buscar en el servicio una asociación con el email que se ha recibido en los datos, si este es encontrado procedemos a ver si la contraseña es correcta y también si es correcto generamos el token.

Texto

Descripción generada automáticamente

Entramos en las rutas en las que hay que estar autenticado.

- **Get**: Para pedir todas las asociaciones que tenemos almacenadas.

- **Get Id**: Para pedir una asociación por el id indicado, buscamos en el servicio, si este existe lo devolvemos y sino respondemos con un No Encontrado.

Texto

Descripción generada automáticamente

- **Delete**: Para eliminar una asociación por un id dado. ( Esta acción solamente la pueden realizar los administradores y las asociaciones). Realizamos la eliminación, si todo sale correctamente no devolvemos contenido y sino devolvemos el error.

Texto

Descripción generada automáticamente

Las asociaciones también tienen rutas para el guardado, vista y eliminación de imágenes.

- **Post Image Id**: Para almacenar una imagen por el id de una asociación.

( Esta acción solamente la pueden realizar los administradores y las asociaciones.)

Recibimos un multi parte (fichero), cuando tenemos el fichero que necesitamos, guardamos la imagen con el servicio y si sale todo correctamente devolvemos que se ha creado y sino devolvemos el error.

Texto

Descripción generada automáticamente

- **Get Image Id**: Para mostrar una imagen por el id de la asociación. Buscamos la imagen en el servicio, si ha salido correctamente devolvemos la imagen y sino devolvemos error.

Texto

Descripción generada automáticamente

- **Delete Image Id**: Para eliminar la imagen de una asociación por un id dado. ( Esta acción solamente la pueden realizar los administradores y las asociaciones).

Eliminamos la imagen con el servicio, si ha salido correctamente devolvemos sin contenido y sino devolvemos error.

### Mapas

Texto

Descripción generada automáticamente

Todas las rutas están autenticadas.

- **Get**: Para pedir todos los mapas que tenemos almacenados.

- **Get Id**: Para pedir un mapa por un id dado, buscamos en el servicio si lo encontramos lo devolvemos y si no mostramos un error.

- **Post**: Salvar un mapa, recibimos unos datos específicos, estos datos deben pasar una validación, si la pasan salvamos en el servicio y si ha salido todo correctamente creamos el calendario y el foro correspondiente a la nueva localización y devolvemos el dato salvado, sino devolvemos error.

Texto

Descripción generada automáticamente

- **Put**: Actualizar mapa por un id dado ( Esta acción solamente la pueden realizar los administradores). Recibimos unos datos específicos que deben pasar una validación, si la pasan actualizamos en el servicio el mapa y si ha salido todo correctamente devolvemos el dato actualizado, sino devolvemos error.

- **Delete**: Eliminar un mapa por un id dado ( Esta acción solamente la pueden realizar los administradores). Realizamos el eliminado en el servicio, si todo ha salido correctamente eliminamos el foro y el calendario asignado a esa localización y devolvemos sin contenido y sino devolvemos error.

Texto

Descripción generada automáticamente

- **Delete Adoption**: Eliminar un mapa por causa de adopción de la colonia en esa localización por un id dado (Esta acción solamente la pueden realizar las asociaciones y los administradores ). Realizamos el eliminado por adopción en el servicio, si todo ha salido correctamente eliminamos el foro y el calendario asignados a esa localización y devolvemos sin contenido y sino devolvemos error.

### Foro

Texto

Descripción generada automáticamente

Todas las rutas están autenticadas.

- **Get**: Para pedir todos los foros que hay almacenados.

- **Get Id:** Para buscar un foro por un id dado. Buscamos en el servicio si está el foro, si todo sale correctamente devolvemos el foro y sino devolvemos error.

- **Get MapsId**: Para buscar un foro por el id de mapa dado. Buscamos en el servicio si está el foro, si todo sale correctamente devolvemos el foro encontrado y sino devolvemos el error.

- **Post**: Para salvar un foro. Recibimos unos datos específicos que deben ser validados, si son validados correctamente salvamos en el servicio y devolvemos el foro guardado, sino devolvemos el error.

Texto

Descripción generada automáticamente

* **Put**: Actualizar un foro por un id dado. Recibimos unos datos específicos que deben ser validados, si son validados correctamente actualizamos en el servicio, si sale todo correctamente devolvemos el dato actualizado y sino devolvemos el error.
* **Delete**: Eliminar un foro por un id dado (Esta acción solamente pueden realizarla los administradores). Eliminamos en el servicio el foro, si todo ha salido correctamente devolvemos sin contenido y sino devolvemos el error.
* **Delete MapsId**: Eliminar un foro por un id de mapa dado (Esta acción solamente la pueden realizar los administradores). Eliminamos en el servicio el foro, si todo ha salido correctamente devolvemos sin contenido y sino devolvemos el error.

### Calendario

Texto

Descripción generada automáticamente

Todas las rutas están autenticadas.

- **Get**: Para pedir todos los calendarios que hay almacenados.

- **Get Id:** Para buscar un calendario por un id dado. Buscamos en el servicio si está el calendario, si todo sale correctamente devolvemos el calendario y sino devolvemos error.

- **Get MapsId**: Para buscar un calendario por el id de mapa dado. Buscamos en el servicio si está el calendario, si todo sale correctamente devolvemos el calendario encontrado y sino devolvemos el error.

- **Post**: Para salvar un calendario. Recibimos unos datos específicos que deben ser validados, si son validados correctamente salvamos en el servicio y devolvemos el calendario guardado, sino devolvemos el error.

* Texto

  Descripción generada automáticamente**Put**: Actualizar un calendario por un id dado. Recibimos unos datos específicos que deben ser validados, si son validados correctamente actualizamos en el servicio, si sale todo correctamente devolvemos el dato actualizado y sino devolvemos el error.
* **Delete**: Eliminar un calendario por un id dado (Esta acción solamente pueden realizarla los administradores). Eliminamos en el servicio el calendario, si todo ha salido correctamente devolvemos sin contenido y sino devolvemos el error.
* **Delete MapsId**: Eliminar un calendario por un id de mapa dado (Esta acción solamente la pueden realizar los administradores). Eliminamos en el servicio el calendario, si todo ha salido correctamente devolvemos sin contenido y sino devolvemos el error.

## rop funcional flow en aplicacionRailway Programming

Es un tipo de programación funcional, utilizado en el proyecto para no lanzar excepciones ya que por su nombre son **algo excepcional**.

Esto nos permite devolver a la vez el resultado que habría si se realiza todo correctamente y todos los errores que pueden ocurrir.

Texto

Descripción generada automáticamente

## Reactividad y Asincronía

La **asincronía** nos permite realizar una acción y seguir realizando otras acciones a la vez, aunque no hayamos recibido respuesta todavía de la primera acción, la asincronía no bloquea el hilo del programa.

Y la reactividad es un paradigma de programación enfocado en el trabajo de flujos de datos finitos o infinitos de **manera asíncrona**, lo que nos permite “reaccionar” a cambios cuando los datos cambian al ejecutar una serie de eventos.

De la reactividad aparece el **Patrón Observer** donde nos suscribimos a algo, algo/alguien nos avisa frente a cambios y reaccionamos.

En los repositorios, cuando buscamos todos los datos, devolvemos flujos ya que se pueden usar de forma asíncrona.

## Seguridad

### SSL

SSL (Secure Sockets Layer o capa de sockets seguros) es una tecnología estandarizada que nos permite crear una conexión segura, cifrando el tráfico de datos de un punto a otro. Lo que nos ayuda a impedir a personas externas poder interceptar la información que se esté transmitiendo de un punto a otro.

En nuestro Back utilizamos **HTTPS** ya que usamos HTTP y la seguridad que nos aporta los certificados SSL.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

### JWT

JWT (Json Web Tokens) Es un texto que está codificado y contiene la identidad de un determinado usuario y sus claims y/o privilegios.

Los tokens están divididos en 3 partes:

* Header: El encabezado, aquí indicamos, como mínimo el algoritmo y el tipo de token.
* Payload: Aquí se encuentran los datos del usuario y sus privilegios y datos.
* Signature: Es una firma que nos permite verificar si el token es válido.

La firma nos sirve para verificar que el remitente es quien dice ser y el mensaje no ha sido modificado, tenemos una clave secreta que nos ayuda a la verificación del token.

El token en el proyecto nos sirve para verificar quien está accediendo y si no estás validado o no tienes los permisos necesarios, no puedes obtener la información de dicha ruta.

## Swagger

Se ha agregado swagger al proyecto para la especificación de requisitos y de resultados de cada una de las rutas de nuestra API

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

## Almacenamiento

En Whisker Watch únicamente tienen almacenamiento las asociaciones, guardamos la imagen de cada asociación.

### Guardar Imagen

Lo que hacemos para guardar la imagen, primero leemos los bytes del ítem, averiguamos cual es la extensión del fichero proporcionado y creamos el nombre del fichero a almacenar.

Creamos el fichero en el directorio establecido para el almacenamiento y escribimos los bytes leídos al principio.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

### Mostrar Imagen

Para devolver un fichero, buscamos el fichero por el nombre proporcionado, si existe lo devolvemos y sino devolvemos nulo.

Texto

Descripción generada automáticamente

### Eliminar Imagen

Muy similar que el de mostrar imagen pero si existe, eliminamos el fichero que hemos conseguido.

Texto

Descripción generada automáticamente

## Tests

### Postman

Con Postman se han probado todas las rutas del back.

### Junit y Mockit

En las pruebas usamos ***runTest*** ya que son funciones suspendidas (Corrutinas).

#### Repositorios

Tenemos un orden para que en cada prueba no debamos crear y eliminar la entidad que usemos para testear. Comprobamos que los datos sean los mismos.

Texto

Descripción generada automáticamente

#### Servicios

Utilizamos **Mockito** para simular las repuestas que nos devolverían los repositorios, también llamados ***dobles de prueba.***

Para realizar las pruebas debemos por cada llamada que se haga a él repositorio, en la prueba del servicio a probar:

* Usar *coEvery{}* para indicarle cuando tenga que utilizar una función concreta del repositorio que debe de devolvernos.
* *Usar coVerify{}* para verificar que se ha llamado a una función concreta del repositorio un número de veces exacta.

Texto

Descripción generada automáticamente

#### Rutas

Para las rutas deberemos configurar el entorno, crearnos un cliente e instalarle todo lo necesario para poder usar la ruta a testear.

Si se necesita realizar acciones previas para poder testear otra ruta, hay que hacerlo.

En las rutas que haya que pasarle contenido deberemos especificar que tipo de contenido vamos a pasarle y en el body añadir el contenido.

Texto

Descripción generada automáticamente

## WebSockets y Notificaciones en tiempo real

Los websockets son conexiones cliente servidor continuas, que permanecen abiertas, que sirven para recibir datos sin realizar más de una petición y tener las notificaciones de los cambios de esos datos por actualizaciones en tiempo real.

Texto

Descripción generada automáticamente

En el proyecto usamos los websocket para notificar las recogidas de colonias de gatos.

Cuando se accede a la ruta lo añadimos como suscriptor para poder luego ir enviándole los nuevos datos que se vayan generando y cuando el cliente cierra la ruta lo sacamos de los suscriptores.

# Front-end

## Primeras Ideas

Para el front ya que es con Angular que es un framework web las primeras ideas fueron las siguientes:

Un dibujo de un pizarrón blanco

Descripción generada automáticamente con confianza media

Un dibujo de un pizarrón blanco

Descripción generada automáticamente con confianza media

## Realización del Front

### Modelos

Creación de los modelos necesarios para las consultas a nuestra API (iguales a las del back).

Fue necesario crear mapeadores para las actualizaciones de listas como calendario y foro.

### Servicios

Utilización de **HttpClient** para consultar las rutas de nuestra API y también para el servicio de notificación.

*API*

*Texto

Descripción generada automáticamente*

*Texto

Descripción generada automáticamenteNotificaciones*

Realizado con el *MatSnackBar* de **Material** para mostrar las notificaciones en pantalla

### Vistas

#### Mapas

Para la creación de la vista de mapas ha sido necesario el uso de **OpenLayers**.

OpenLayers es una librería que facilita la colocación de mapas dinámicos en páginas web, es gratuito y usa javascript.

En OpenLayers se puede personalizar, añadir acciones, marcadores.

#### Calendario

Para la creación del calendario ha sido necesario el uso de **FullCalendar.**

FullCalendar es ligero, es de codigo abierto y se pueden hacer muchas cosas, como la creación de eventos, utilizada en el proyecto para crear, mostrar y eliminar eventos