Desarrollo de una infraestructura REST multiplataforma para GTD basada en Spring Boot
Development of a multiplatform REST infrastructure for GTD based on Spring Boot

Trabajo de Fin de Grado
Curso 2023–2024

Autor

Camilo Andres D'isidoro
Jhimmy Ender Candela Espinosa
Víctor Gallego Yañez
José David López Geraghty
Matías Antonio Alonso Eiras

Director

Juan Carlos Sáez Alcaide

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Desarrollo de una infraestructura REST multiplataforma para GTD basada en Spring Boot
Development of a multiplatform REST infrastructure for GTD based on Spring Boot

Trabajo de Fin de Grado en Ingeniería Informática

Autor

Camilo Andres D'isidoro
Jhimmy Ender Candela Espinosa
Víctor Gallego Yañez
José David López Geraghty
Matías Antonio Alonso Eiras

Director

Juan Carlos Sáez Alcaide

Convocatoria: *Junio 2024*

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

26 de mayo de 2024

Dedicatoria

A nuestros familiares y seres queridos, por iluminar nuestro camino, motivarnos cada día y hacer posible este momento.

Agradecimientos

De David a sus padres, por la educación que le han dado.

De Camilo a sus padres, por enseñarle a no rendirse, por motivarle a seguir sus sueños, a esforzarse en todo momento y a tener carácter; y a Felipe, por su constante apoyo y cariño, por motivarle a ser mejor cada día, y por creer en él incluso en los días que no podía creer en sí mismo.

De Jhimmy: Quisiera dedicar unas palabras de profundo agradecimiento a las personas más significativas en mi vida, sin cuyas contribuciones este viaje no hubiera sido posible. A mis padres Nancy y Feliciano gracias por ser mi fuente inagotable de energía. A Laura por ser mi luz en este viaje, gracias por tu comprensión, paciencia y amor incondicional. Tus palabras de aliento en los momentos de duda y tu capacidad para hacerme sonreír en los días difíciles han sido esenciales para alcanzar esta meta. Tu apoyo ha sido un regalo invaluable que valoro más allá de las palabras.

De Matías a sus padres, que le han apoyado en todo momento, y le han escuchado y ayudado a reflexionar. También al resto de su familia, con quienes ha contado con su apoyo durante los cuatro años de carrera.

De Víctor a sus padres por su cariño incondicional y a Ana por su apoyo constante y motivación para seguir adelante.

Y de parte del grupo a Juan Carlos, por su supervisión y motivación para sacar adelante el proyecto.

Resumen

Desarrollo de una infraestructura REST multiplataforma para GTD basada en Spring Boot

La metodología GTD (Getting Things Done) diseñada por David Allen constituye actualmente uno de los mecanismos más eficientes de organización personal. Su objetivo es hacer posible que logremos la máxima productividad mediante el almacenamiento de las tareas, proyectos y actividades a realizar en un lugar específico. Hoy en día, existen infinidad de aplicaciones para ayudarnos a poner en práctica la filosofía GTD. Algunas de estas aplicaciones almacenan nuestra información GTD en la nube, para así poder acceder a ella fácilmente desde distintos dispositivos (p.ej., PC de sobremesa, tablet o móvil). Lamentablemente, muchas de las aplicaciones disponibles son propietarias, y las empresas que ofrecen el servicio tienen acceso a nuestra información GTD personal, lo cual puede violar la privacidad de los usuarios.

Esta problemática ha sido la motivación por la cual se ha decidido desarrollar una infraestructura REST multiplataforma basada en Spring Boot que aplica la filosofía GTD, y que permitirá que las empresas interesadas en usar este sistema, almacenen su información en servidores propios. Este sistema protegerá sus comunicaciones con la API utilizando el protocolo HTTPS, y utilizará dos aplicaciones cliente para los usuarios finales: una web hecha utilizando Flutter, y una aplicación de Android empleando Kotlin.

Palabras clave

REST, Web, Android, Multiplataforma, GTD, Docker, Seguridad, Corporativo, SSO, Offline

Abstract

Development of a multiplatform REST infrastructure for GTD based on Spring Boot

The GTD methodology (Getting Things Done), designed by David Allen currently constitutes one of the most efficient personal organization mechanisms. It's aim is to make possible to achieve maximum productivity by storing tasks, projects and activities to be carried out in a specific place. Nowadays, there are countless applications to help us to put in practice the GTD philosophy. Some of these applications store our GTD information in the cloud, so we can access it easily from different devices (e.g. desktop computer, tablet or phone). Unfortunately, many of the available applications are propietary, and the companies that offer the service have access to our personal GTD information, which can violate the privacy of users.

This problem has been the motivation for which it has been decided to develop a multiplatform REST infrastructure based on Spring Boot which applies the GTD philosophy, and which will allow companies interested in using this system to store their information on their own servers. This system will protect its communications with the API using the HTTPS protocol, and will use two client applications for the final users: a website built using Flutter, and an Android application using Kotlin.

Keywords

REST, Web, Android, Multiplatform, GTD, Docker, Security, Corporative, SSO, Offline

Índice

Índice de figuras

Capítulo 1

Introducción

"No se pasa de lo posible a lo real, sino de lo imposible a lo verdadero"
— María Zambrano

1.1. Motivación

La filosofía GTD, desarrollada por David Allen, fue ideada como un sistema para gestionar y organizar la vida personal y laboral, permitiendo mejorar la productividad de las personas que aplican esta metodología[1]. Esta metodología consiste en 5 pasos fundamentales para su aplicación:

1. Capturar: Escribir o registrar todo lo que requiere nuestra atención en un solo sitio.

2. Aclarar: Definir si se puede realizar la tarea ahora mismo. De ser el caso, hay que decidir la próxima acción y proyecto. En caso contrario, decidir si es irrelevante, una referencia o algo que debe esperar.

3. Organizar: Crear recordatorios de las tareas categorizadas de forma apropiada.

4. Revisar: Actualizar y revisar todo lo necesario en el sistema para recuperar la concentración y enfoque.

5. Hacer: Utilizar el sistema para hacer las tareas pertinentes con confianza y claridad.

Para la aplicación de esta metodología existen diferentes aplicaciones que funcionan estrictamente siguiendo la filosofía GTD, y otras cuyo enfoque no sigue de una forma tan purista esta filosofía (Trello, Evernote, Notion, etc.). La mayor parte de estas aplicaciones permiten el almacenamiento de información en la nube, lo cual facilita su acceso desde cualquier dispositivo. Sin embargo, la vasta mayoría de estas aplicaciones son propietarias y almacenan los datos de sus usuarios en los servidores de los proveedores, por lo que los propietarios de estas aplicaciones tienen acceso a la información de los usuarios, afectando negativamente a su privacidad. Además, al

almacenarse esta información en servidores fuera del control de la empresa, podrían correrse riesgos de que la confidencialidad de la información que se almacene se vea comprometida en un ciberataque.

1.2. Objetivos

Con el fin de solucionar estos problemas, se propone el desarrollo de una infraestructura de un sistema GTD, la cual se compone de estos componentes:

- Una base de datos que almacenará la información que necesiten los miembros de la organización.
- Un servidor que implementa una API REST basada en Spring Boot, el cual hará posible consultar y manipular la información de la base de datos.
- Una aplicación web desarrollada en Flutter, que permitirá la interacción con la API REST y hará posible aplicar la filosofía GTD.
- Una aplicación de escritorio basada en la aplicación web, la cual adaptará la vista de la aplicación web y proporcionará una alternativa de acceso a la aplicación, además de proporcionar un acceso sin conexión a internet.
- Una aplicación nativa para dispositivos con sistema operativo Android.

También se busca que, dado que las organizaciones interesadas en utilizar este sistema tendrán que desplegar la infraestructura, su proceso de instalación sea sencillo. Para ello, se ha optado porque cada componente que deba instalarse en los servidores de la organización y sea ejecutable a través de contenedores, de modo que con un script de arranque sea posible desplegar estos componentes. Otro aspecto importante es el de la seguridad del sistema, para el cual se propone la implementación de estas medidas para garantizar la máxima seguridad posible:

- La comunicación con el servidor estará protegida haciendo uso del protocolo HTTPS, con el fin de cifrar las comunicaciones con el servidor y dificultar fugas de información.
- Con el fin de que los usuarios del sistema hagan uso solamente de la información que ellos mismos creen, se ha implementado un sistema de filtrado por autoría que impide que un usuario pueda acceder y/o modificar la información de otro usuario, sea de forma accidental o deliberada.

1.3. Arquitectura del sistema

El sistema dispondrá de la arquitectura descrita en la figura 1.1. Como se puede ver, la arquitectura tendrá los siguientes componentes:

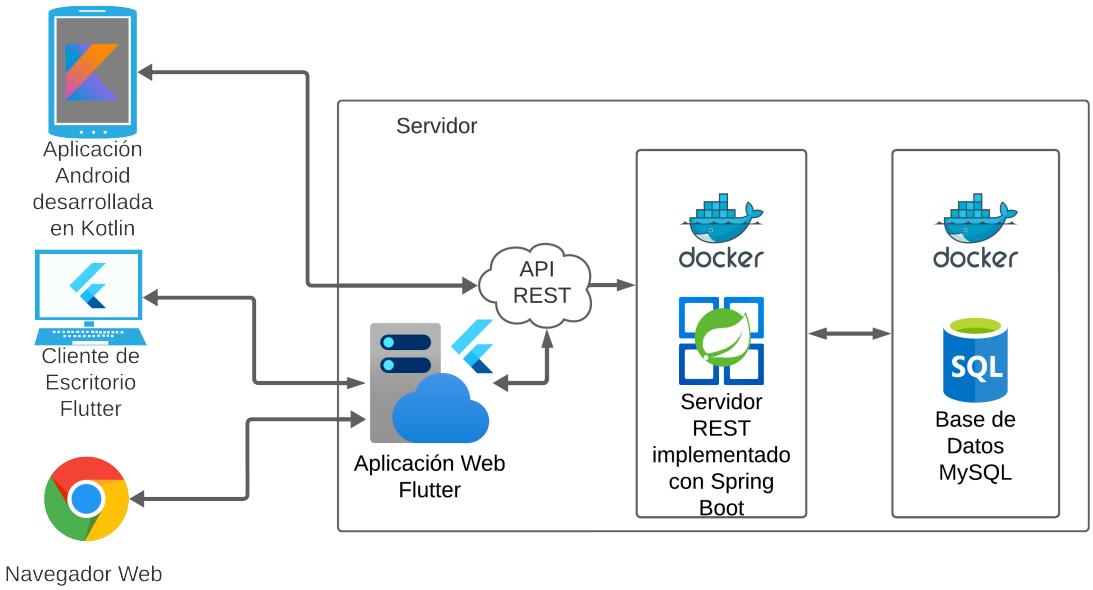

Figura 1.1: Arquitectura del Sistema

- Aplicación Android desarrollada en Kotlin: La aplicación móvil se conectará directamente con la API REST para la ejecución de las operaciones solicitadas por el usuario.
- Cliente de Escritorio Flutter: La aplicación de escritorio se desplegará en función de la aplicación web, e interactuará con esta para realizar las operaciones pertinentes.
- Aplicación Web Flutter: La aplicación web se encargará de proporcionar un acceso a la API para la aplicación de escritorio y para los navegadores web que accedan a ella.
- Servidor REST implementado con Spring Boot: Este servidor implementa todas las llamadas a la API y su lógica de negocio, comunicándose directamente con la base de datos para interactuar con la información existente o nueva.
- Base de Datos MySQL: Este sistema gestor de bases de datos SQL se despliega de forma independiente al servidor con el fin de distribuir la carga de trabajo de forma adecuada, y se encarga de persistir toda la información del sistema.

Las aplicaciones cliente se conectan a internet para interactuar con la API REST, en la cual el servidor realizará las solicitudes necesarias al servidor de base de datos, y devolverá la información o ejecutará las peticiones solicitadas por el cliente.

1.4. Plan de trabajo

Con el fin de cumplir con estos objetivos, se ha seguido una ruta de trabajo, que consta de las siguientes tareas:

T1. Diseño de la arquitectura: Definir los aspectos relevantes para el funcionamiento de la infraestructura, como la estructura de la base de datos.

T2. Estudio previo de las tecnologías: Realizar un estudio previo de cómo funcionan las tecnologías seleccionadas, mediante la lectura de documentación, visionado de videotutoriales, e implementación de prototipos relacionados con el sistema a desarrollar.

T3. Prototipo interfaz de usuario: Hacer el diseño de la interfaz de usuario con Figma[8] de las aplicaciones cliente, tanto en su versión de escritorio como su versión para móviles.

T4. Servidor y base de datos: Hacer la implementación de la base de datos y el servidor con Spring Boot, incluyendo funcionalidades básicas para la inserción, consulta, modificación y eliminación de los diferentes tipos de información existentes en la base de datos, así como las funciones de seguridad que sean pertinentes.

T5. Aplicación web: Implementar la aplicación multiplataforma con Flutter, implantando las medidas de seguridad necesarias, y todas las funcionalidades que permitan aplicar la filosofía GTD, aplicando todas las modificaciones necesarias al servidor.

T6. Aplicación Android: Desarrollar la aplicación Android con Kotlin, implementando un modo offline, que permita acceder a la información de los usuarios sin conexión a internet, y sincronizar cualquier posible modificación una vez recuperen la conexión a internet.

T7. Escribir la memoria: Redactar la memoria final del proyecto.

En las figuras 1.2 y 1.3 se presenta la planificación temporal del proyecto durante los años 2023 y 2024 respectivamente.

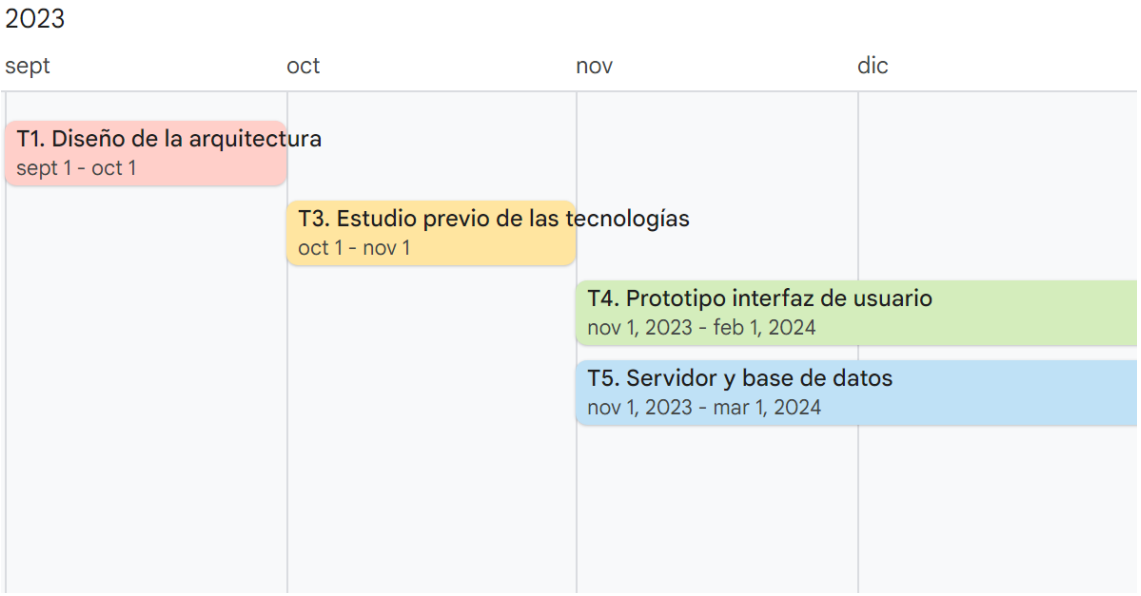

Figura 1.2: Diagrama de Gantt 1

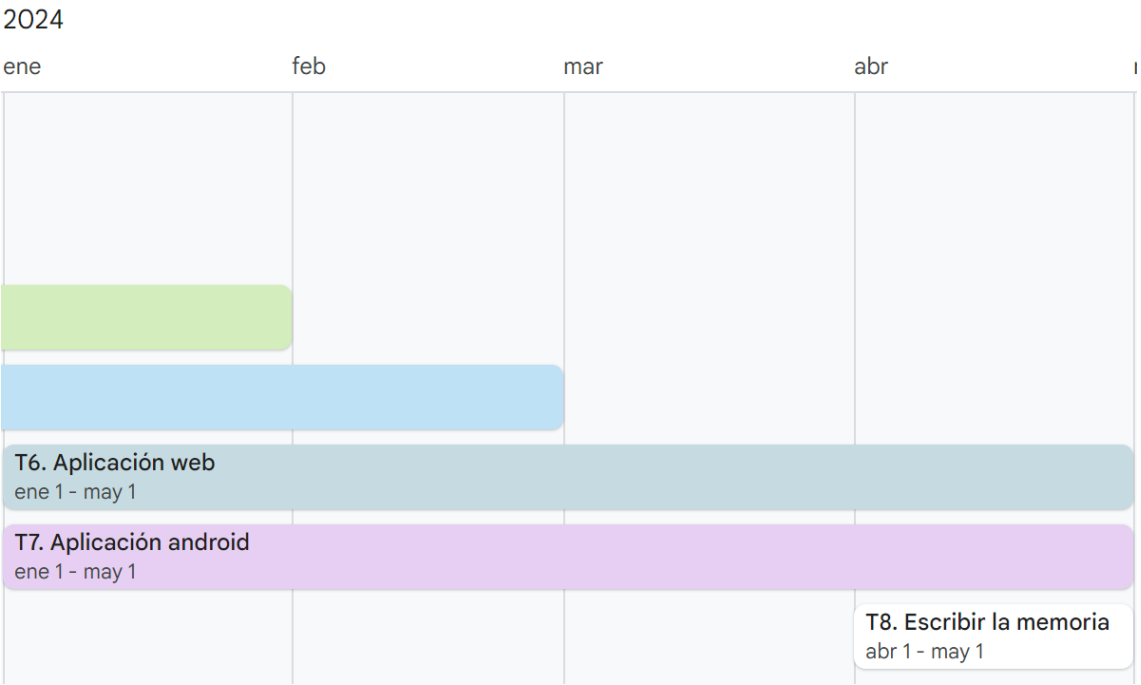

Figura 1.3: Diagrama de Gantt 2

1.5. Estructura de la memoria

La memoria de este proyecto se ha redactado definiendo la estructura que se detalla a continuación:

- En el capítulo 2 se detalla cómo se ha implementado la API basada en Spring Boot, así como ciertas decisiones de diseño como la motivación de haber seleccionado cada tecnología, la arquitectura interna del servidor, el modelo de datos, la orquestación de los contenedores del servidor y base de datos, y las medidas de seguridad implementadas para las comunicaciones con el servidor.
- En el capítulo 3 se describe y explica el diseño que tiene la interfaz de usuario para las aplicaciones cliente. También se lleva a cabo un recorrido por las pantallas que tiene dicha interfaz de usuario.
- En el capítulo 4 se explica cómo fue implementada la aplicación web basada en Flutter, discutiendo la arquitectura interna de la aplicación y el funcionamiento de ésta.
- En el capítulo 5 se encuentra detallada toda la implementación y decisiones de diseño relacionadas con la aplicación Android desarrollada con Kotlin, así como una guía orientativa de uso para los usuarios de la aplicación.
- En el capítulo 6 se recapitula todo el trabajo realizado en el proyecto, y se plantean posibles mejoras a realizar en el futuro en caso de que se continúe con el desarrollo del proyecto.

Adicionalmente, se ha dispuesto de una serie de apéndices que se detallan ahora:

- En el apéndice A se detallan las contribuciones personales de cada miembro del equipo en desarrollo de este proyecto.
- En el apéndice B se explica con un alto nivel de detalle las llamadas disponibles en la API, cómo funcionan, qué información devuelve cada una, etc.
- En el apéndice C se encuentra un manual de instalación y despliegue para el servidor Spring Boot.
- En el apéndice D se encuentra un manual de instalación y despliegue para la aplicación web desarrollada con Flutter.

Capítulo 2

Implementación del servidor y BD

En este capítulo se presenta todo el funcionamiento e implementación de la API REST y de la base de datos para la aplicación, así como varias decisiones de diseño.

En la sección de tecnologías utlizadas 2.1 se listan las tecnologías que se han empleado, y se explica la motivación por la que se han empleado cada una de ellas. Más adelante, en la sección diseño de la base de datos 2.2 se explica la estructura de la base de datos y el funcionamiento de cada tabla. Posteriormente, en el apartado orquestación de los componentes del servidor 2.3 se explica cómo se coordina la inicialización de la API y de la base de datos para una ejecución correcta. Finalmente, en el apartado de arquitectura y diseño de la API 2.4 se detalla el funcionamiento interno de la API, y cuáles son sus componentes.

De forma adicional, en el apéndice B se incluye la documentación detallada del funcionamiento de cada endpoint de la API. También se incluye en el apéndice C un instructivo para poder instalar y desplegar el servidor correctamente.

2.1. Tecnologías utilizadas

Para el desarrollo de la API REST se ha decidido hacer uso del framework Spring Boot[2, 3, 20], dado que es un framework con mucha presencia en la industria, lo que facilita el acceso a documentación y una comunidad que le da mucho soporte. Además, emplea el lenguaje de programación Java, un lenguaje muy utilizado a día de hoy y con mucho soporte.

Por otra parte, para la base de datos se ha empleado de una BD estructurada en SQL, utilizando MySQL, un sistema de gestión de bases de datos extremadamente popular dentro de la industria, y con el que se dispone de muy buen soporte para Spring Boot.

Finalmente, con la finalidad de facilitar el despliegue del servidor para el cliente final y proporcionar un nivel de aislamiento de la infraestructura donde se despliegue, se ha contenerizado el servidor y la base de datos en contenedores de Docker. [7]

2.2. Diseño del modelo de datos

La base de datos se ha diseñado en base al modelo de datos que se explica en la figura 2.1:

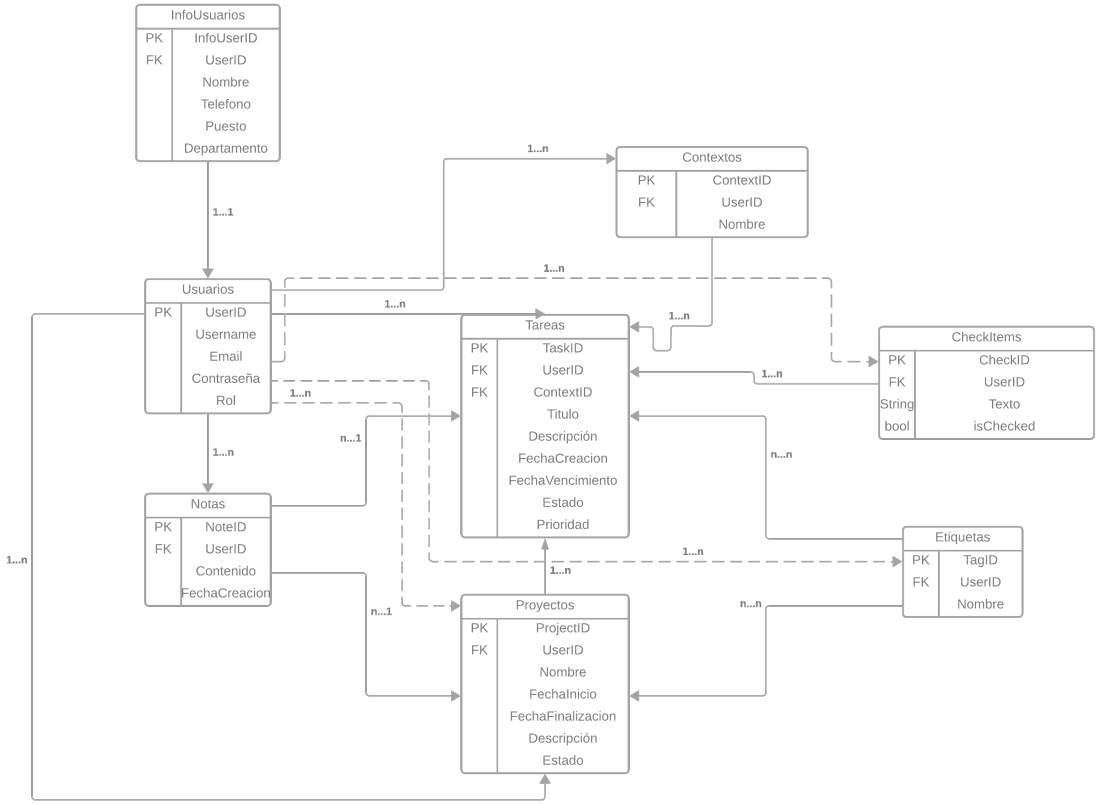

Figura 2.1: Estructura de la base de datos

A continuación se detallan todas las tablas y campos que conforman el modelo de datos:

- Tabla InfoUsuarios: Almacenará la información adicional de los usuarios de la aplicación. Mantiene una relación 1 a 1 con los usuarios, dado que una persona solo puede tener una identidad y ejercer un cargo en la organización.
  - InfoUserID (Clave Primaria): Identificador principal de la información adicional de un usuario.
  - UserID (Clave Foránea): Identificador del usuario al que le corresponde esta información adicional.
  - Nombre: Nombre real del usuario.
  - Teléfono: Número telefónico de contacto del usuario.
  - Puesto: Cargo que ocupa el usuario en la organización.
  - Departamento: Departamento de la organización en donde trabaja este usuario.
- Tabla Usuarios: Almacena la información básica de acceso de los usuarios. Mantiene una relación 1 a muchos con el resto de componentes del modelo de datos, con la finalidad de controlar la autoría de cada componente de la aplicación.
  - UserID (Clave Primaria): Identificador principal del usuario en el sistema.
  - Email: Correo electrónico con el que el usuario se ha registrado en el sistema. Este correo electrónico debe ser único en todo el sistema.
  - Contraseña: Contraseña de acceso para que el usuario pueda acceder al sistema. Esta contraseña se almacena cifrada utilizando el mecanismo de codificación BCrypt.
  - Username: Nombre de usuario con el que el usuario accederá al sistema. Este nombre de usuario debe ser único en todo el sistema.
  - Rol: Tipo de privilegios que tiene el usuario en el sistema. Los usuarios pueden tener únicamente dos tipos de rol:
    - Usuario: Representa a un usuario convencional en el sistema. Únicamente podrá acceder y manipular información de su autoría. Todos los usuarios se crean con este rol.
    - Administrador: Este tipo de usuario tiene acceso total a toda la información del sistema, sea de su autoría o no. Este tipo de usuario debe darse de alta manualmente desde el terminal de la base de datos, o bien ejecutando manualmente una petición HTTP de registro al API. Sin embargo, solo puede existir un usuario administrador a la vez.
- Tabla Contextos: Almacena los contextos que podría tener una tarea. Un contexto es el lugar físico en el que se ejecuta una tarea. Tiene una relación 1 a muchos con las tareas, dado que un contexto puede ser utilizado para varias tareas, mientras que una tarea solamente puede tener un contexto.

  - ContextID (Clave Primaria): Identificador principal del contexto.
  - UserID (Clave Foránea): Identificador del usuario autor del contexto.
  - Nombre: Nombre del contexto.

- Tabla CheckItems: Guarda los diferentes elementos de comprobación que puede tener una tarea.
  - CheckID (Clave Primaria): Identificador principal del elemento de comprobación.
  - UserID (Clave Foránea): Identificador del usuario autor del elemento de comprobación.
  - Texto: Descripción del elemento de comprobación.
  - isChecked: Indica si este elemento ha sido marcado como completado o no.

- Tabla Etiquetas: Contiene las diferentes etiquetas que puede tener una tarea. Una etiqueta permite categorizar una tarea o proyecto para darle una atención particular (Urgente, Por Documentar, Defecto...)
  - TagID (Clave Primaria): Identificador principal de la etiqueta.
  - UserID (Clave Foránea): Identificador del usuario autor de la etiqueta.
  - Nombre: Nombre de la etiqueta.

- Tabla Notas: Almacena las notas que pueden tener los proyectos o las tareas. Una nota puede almacenar información adicional de cara a desarrollar una tarea o proyecto.
  - NoteID (Clave Primaria): Identificador principal de la nota.
  - UserID (Clave Foránea): Identificador del usuario autor de la tarea.
  - Contenido: Contenido principal de la nota.
  - FechaCreacion: Fecha de creación de la nota.

- Tabla Tareas: Almacena las tareas que tenga que realizar los usuarios. Estas tareas deben estar asignadas a un proyecto concreto. Mantiene una relación 1 a muchos con las notas y checkItems, puesto que una tarea puede tener más de uno de estos componentes, mientras que dichos componentes solo pueden estar relacionados con una tarea. Además, tiene una relación muchos a muchos con las etiquetas, debido a que una tarea puede tener múltiples etiquetas, y una etiqueta puede utilizarse para varias tareas y proyectos.
  - TaskID (Clave Primaria): Identificador principal de la tarea.
  - UserID (Clave Foránea): Identificador del usuario autor de la tarea.
  - ContextID (Clave Foránea): Identificador del contexto en el que se realiza la tarea.
  - Título: Título de la tarea que se va a realizar.

  - Descripción: Descripción de en qué consiste la tarea a realizar.
  - FechaCreacion: Fecha de creación de la tarea.
  - FechaVencimiento: Fecha de vencimiento de la tarea, en caso de tener una.
  - Estado: Estado en que se encuentra la tarea.
  - Prioridad: Prioridad numérica de la tarea.

- Tabla Proyectos: Contiene los proyectos que crean los usuarios en el sistema. Un proyecto consiste en un conjunto de tareas que permiten completar una tarea más grande relacionada con un tema específico. Los proyectos tienen una relación 1 a muchos con las tareas, porque un proyecto puede tener una o varias tareas asociadas, mientras que una tarea solo puede asociarse a un proyecto. Además, también tiene una relación muchos a muchos con las etiquetas, debido a que un proyecto puede tener una o varias etiquetas asignadas, y una etiqueta puede utilizarse para varios proyectos o tareas.

  - ProjectID (Clave Primaria): Identificador principal del proyecto.
  - UserID (Clave Foránea): Identificador del usuario autor del proyecto.
  - Nombre: Nombre identificativo del proyecto.
  - FechaInicio: Fecha de inicio del proyecto.
  - FechaFinalización: Fecha de finalización del proyecto.
  - Descripción: Breve descripción de en que consiste el proyecto.
  - Estado: Estado en el que se encuentra el proyecto.

2.3. Orquestación de los componentes del servidor

Como se había comentado en el apartado 2.1 de tecnologías utilizadas, se está haciendo uso de Docker para contenerizar los dos componentes del servidor, con el fin de facilitar su instalación y proporcionar un entorno aislado a estos sistemas para su correcto funcionamiento.

La API y la base de datos se han desplegado en dos contenedores independientes para modularizar al máximo el backend. Sin embargo, si no se coordina el arranque de los contenedores podría provocar errores durante el inicio de la API. Específicamente, cuando se arranca el servidor Spring Boot, se intentará comunicar con el contenedor de la base de datos, y si este aún no ha terminado de arrancar o está apagado, dará un error y no será posible lanzar la aplicación.

Para solucionar este problema, se ha hecho uso de un fichero docker-compose, el cual además de especificar cierta configuración particular de los contenedores, permite diseñar una comprobación de estado de los contenedores, y programar el lanzamiento de uno de ellos en función del estado del otro.

A continuación se presenta un ejemplo del fichero docker compose utilizado para este proyecto:

```yaml
services:
    server:
        build:
        context: .
        target: development
        ports:
            - "443:443" #Mapeo del puerto 443 del host con el puerto 443 del contenedor
        environment:
            - SERVER_PORT=443
            - MYSQL_URL=jdbc:mysql://db/getTasksDone #URL de la base de datos
        volumes:
            - ./:/app
        depends_on: #Relacion de dependencia con la base de datos, bajo la condicion de que el servicio este activo
            db:
                condition: service_healthy
    db:
        image: mysql:8.0
        ports:
            - "3306:3306"
        environment: #Credenciales de acceso a la base de datos
            - MYSQL_ROOT_PASSWORD=
            - MYSQL_ALLOW_EMPTY_PASSWORD=true
            - MYSQL_USER=getTasksDone
            - MYSQL_PASSWORD=superSecur3P4ssw0rd123
            - MYSQL_DATABASE=getTasksDone
        volumes:
            - db_data:/var/lib/mysql
            - db_config:/etc/mysql/conf.d
            - ./database:/docker-entrypoint-initdb.d
        healthcheck: #Definicion de la comprobacion de actividad del servicio
            test: ["CMD", "mysqladmin", "ping", "-h", "localhost"] #Comando a ejecutar para validar la actividad del servicio
            interval: 10s #Intervalo en el que se ejecuta el comando
            timeout: 5s #Tiempo de espera de respuesta del comando
            retries: 5 #Numero de reintentos para la comprobacion de actividad
            start_period: 5s #Tiempo que tarda en iniciarse la comprobacion
volumes:
    db_data:
    db_config:

```

Algunas de las configuraciones que se han implementado en el fichero docker-compose son las siguientes:

- Asignar un puerto de la máquina host al puerto 443 de la API REST o de la base de datos.
- Definir la URL del servidor de base de datos.

- Establecer las credenciales de acceso para el servidor de base de datos.
- Proporcionar un fichero .SQL de entrada al contenedor de base de datos para importar la estructura de la BD.
- Definir una comprobación de estado para la base de datos con el fin de validar si está en un estado que admita conexiones entrantes o no.

  Esto se ha hecho solicitando que se envíe dentro del contenedor el comando `mysqladmin ping -h localhost`, y en caso de retornar una respuesta, se determina que el servidor admite conexiones. Esta comprobación se realiza cada minuto, en intervalos de 30 segundos por intento, con un límite de 5 intentos, para garantizar que dará tiempo suficiente a un dispositivo de bajos recursos para iniciar el contenedor.
- Implementar una relación de dependencia del servidor hacia la base de datos, bajo la condición de que la base de datos admita conexiones. Es decir, que la API REST no se iniciará hasta que la base de datos se haya inicializado por completo.

2.4. Arquitectura y diseño de la API

La API que se ha desarrollado tiene la organización por paquetes definida en la figura 2.2, los cuales se irán describiendo durante los próximos segmentos:

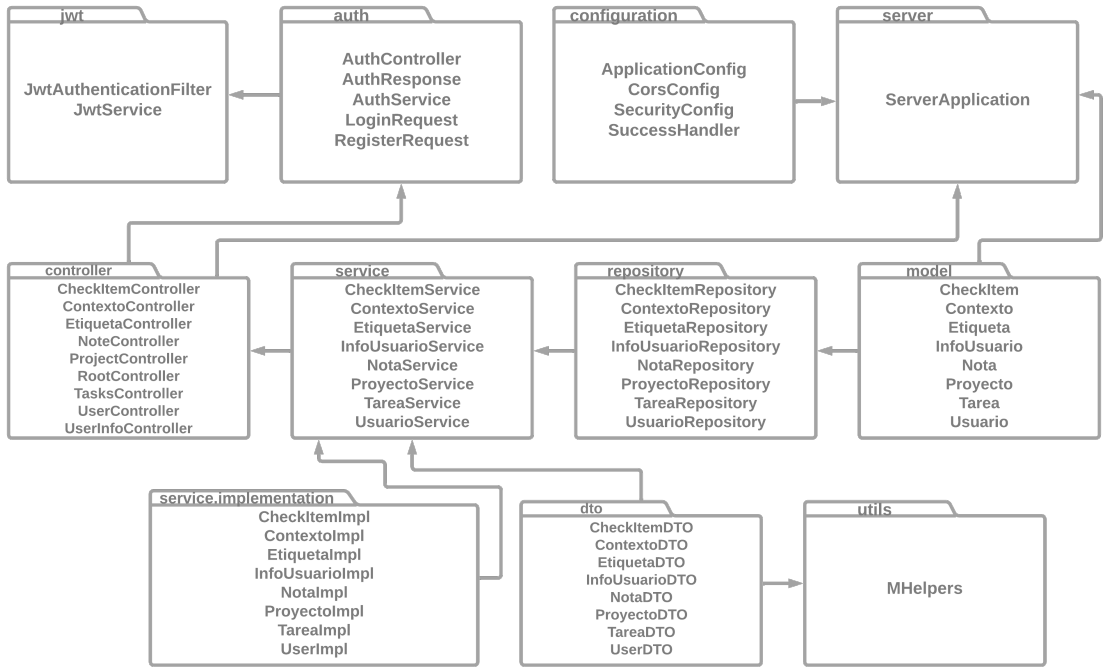

Figura 2.2: Paquetes de la API Spring

2.4.1. Implementación del sistema de autorización

La aplicación emplea un sistema de autenticación adaptado a las arquitecturas modernas, ya que permite iniciar sesión con un nombre de usuario y contraseña, habiendo hecho un registro previo, o usando el inicio de sesión de Google sin tener que hacer un registro previo.

Las ventajas de este sistema son claras y va destinado a que cada usuario tenga la libertad de elegir qué método de autenticación quiere utilizar, según le sea más conveniente y práctico.

Además, la autenticación sigue un esquema *sin estado* utilizando una tecnología llamada `JSON Web Token`, o por sus siglas *JWT*.

2.4.1.1. Autenticación sin estado

Las aplicaciones pueden usar autenticación con estado o sin estado. Mientras que una aplicación con estado mantiene información de la sesión y de los eventos, una sin estado trata cada interacción como independiente. Sin entrar en detalle, estas son algunas de las ventajas de las aplicaciones sin estado frente a las que sí guardan el estado:

- Escalabilidad: Las peticiones se atienden de forma desagregada.
- Simplicidad: Facilidad de desarrollo.
- Menor consumo de recursos, especialmente memoria.
- Mayor resiliencia a fallos.

Para consultar más información acerca de la autenticación sin estado frente a la autenticación con estado se puede consultar a Neal K. Davis[6].

2.4.1.2. JWT

Para lograr este sistema de autenticación, es muy ventajoso emplear tokens *JWT* desde el comienzo en la aplicación.

El objetivo de esta sección no es explicar detalladamente cómo funciona este estándar, si no explicar cómo ha sido vinculado en esta aplicación, si se quiere aprender más sobre la teoría detrás de esta tecnología, se puede consultar la página de documentación oficial de JWT[21].

Se ha elegido JWT por que permite fácilmente hacer autenticación sin estado, es muy seguro, robusto y también sigue el mismo estándar que los tokens que responden los proveedores de inicio de sesión social como Google.

El funcionamiento de JWT se describe en el diagrama de flujo mostrado en la figura 2.3:

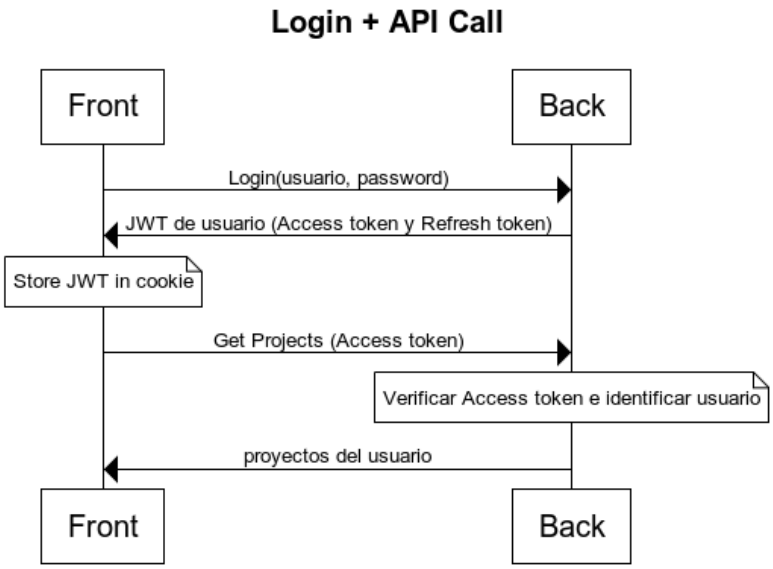

Figura 2.3: Login con JWT

Este diagrama describe el siguiente proceso de ejecución:

1. La aplicación frontend realiza una operación de inicio de sesión con el backend.
2. El backend responde al servidor con un JWT que ha generado para que pueda autenticarse en las llamadas posteriores.
3. El frontend almacena el JWT en una cookie para facilitar su acceso.
4. Ahora el frontend realiza una llamada de consulta de datos al back, anexando a la petición el JWT.
5. El backend recoge el JWT del cliente y verifica su identidad.
6. Una vez verificada la identidad del usuario, el backend envía la respuesta a la llamada de consulta del frontend.

A continuación se presenta un ejemplo de JWT que genera el servidor (Utilizando el algoritmo de cifrado HMAC con SHA-256) para el cliente en cuanto se ejecuta una operación de inicio de sesión:

```
eyJhbGciOiJIUzI1NiJ9.eyJpZCI6IjEiLCJyb2wiOiJVU1VBUklPIiwic3ViIjoiZ
GF2aWQubG9wZXoiLCJpYXQiOjE3MTE5MTU1NjQsImV4cCI6MTcxMTkxNzAwNH0.0AJ
2L7dpVKWgTjCInpci-6LAZ_ZMnWQR1tkugBtGVLg
```

Si se decodifica el siguiente token, se obtendrá la siguiente información (Solo se muestra el campo de datos):

```
{
    "id": "1",
```

```
    "rol": "USUARIO",
    "sub": "david.lopez",
    "iat": 1711915564,
    "exp": 1711917004
}
```

La información proporcionada por el token se explica a continuación:

- id: Representa el identificador único del usuario en la aplicación. Este identificador se envía en el JWT para validar que el usuario autenticado intenta acceder a información de su propiedad.

- rol: Indica el nivel de acceso que tiene el usuario en la aplicación. Enviar esta información permite detectar fácilmente si el usuario autenticado es un administrador o un usuario sin privilegios, con el fin de autorizar o negar el acceso más fácilmente a llamadas del servidor que requieran privilegios.

- sub (Subject): Contiene el nombre de usuario de la persona autenticada, con el fin de identificarle.

- iat (Issued at): Indica la fecha en milisegundos en que fue creado el token.

- exp (Expires): Indica la fecha en milisegundos en la que el token dejará de ser válido.

2.4.1.3. Implementación dentro de la aplicación

Antes de comenzar con el desarrollo de la API REST fue necesario añadir una serie de dependencias al proyecto que permitirán acceder a toda la funcionalidad de JWT. Estas dependencias se han añadido en el fichero `pom.xml` del proyecto.

Para implementar la funcionalidad de JWT en el proyecto se han implementado dos clases localizadas en el paquete `jwt`, las cuales se explican resumidamente a continuación[1]:

- `jwt/JwtService.java`: En esta clase se implementan los métodos que permiten crear un JWT nuevo, comprobar si un token es válido, si ha expirado, obtener su fecha de caducidad, el identificador del usuario autenticado, entre otras cosas.

- `jwt/JwtAuthenticationFilter.java`: Esta clase se encarga de realizar el filtrado interno de los token a nivel de peticiones HTTP. Es decir, que aquí se recogen las peticiones, se extrae el JWT de la petición, y valida que el token sea correcto, autorizando el paso a las llamadas de la API.

[1]Para conocer más a profundidad la implementación del código, se puede consultar en el repositorio donde está alojado el proyecto

2.4.1.4. Configuración del SSO de Google

Para realizar la configuración del SSO de Google se requiere la inclusión de una dependencia nueva en el fichero `pom.xml`.

`org.springframework.boot`: `spring-boot-starter-oauth2-client`

Una vez implementada la dependencia, se debe configurar el sistema de OAuth 2.0 de Google para obtener un identificador de cliente y una clave secreta que permitirá a la aplicación Spring identificarse correctamente con Google. Los detalles de cómo se puede solicitar esta información a Google se pueden consultar en la página web de documentación de Google Identity[16].

También es necesario añadir una configuración adicional para que se permita realizar la autenticación con Google. Esta configuración se hace en el fichero `configuration/SecurityConfig.java`.

Y finalmente, se debe crear la clase `configuration/SuccessHandler.java`, la cual implementa la interfaz `AuthenticationSuccessHandler`. Esta clase se encargará de que, cuando se haya realizado un inicio de sesión correcto y Google retorne su JWT, obtenga la información del token y con esa información, se redirija a uno de los controladores de la aplicación para generar un JWT propio de la API. Los detalles de los controladores se explicarán posteriormente en la sección 2.4.5.

2.4.2. Medidas de segurización de las comunicaciones con el servidor

Para garantizar la seguridad de las comunicaciones del servidor, se han tomado las siguientes medidas:

- Se ha implementado el protocolo HTTPS en el servidor para cifrar toda la información que viaje hacia las aplicaciones cliente. Para ello, Spring requiere que se cargue un certificado SSL/TLS dentro de los ficheros del proyecto, y hacer referencia al certificado en el fichero de configuración localizado en `src/main/resources/application.properties`.

- Se impide que los usuarios no autenticados puedan realizar cualquier llamada a la API que pueda resultar en la consulta y/o modificación de información en la base de datos, a excepción de, realizar un inicio de sesión, un registro en el sistema, o comprobar el estado del servidor llamando a un endpoint de ping.

  Otra excepción es cualquier paquete HTTP de tipo OPTIONS[19] (Encargado de solicitar las configuraciones de comunicación permitidas por el servidor), esto es debido a que los navegadores de internet requieren de enviar este tipo de peticiones para saber qué tipo de información puede recibir el servidor para esa llamada, y bloquearlos bajo autorización impediría el correcto funcionamiento de la aplicación cliente web.

  Todas estas configuraciones están definidas en el fichero:

  `configuration/SecurityConfig.java`

- Se restringe que un usuario pueda consultar, modificar y/o eliminar contenido que haya sido creado por otro usuario, a menos que sea un usuario administrador. Esto se ha logrado obteniendo el id y rol del usuario autenticado del JWT, y en cada operación que corresponda llamar a una función auxiliar `checkAccess` la cual validará si el usuario autenticado es propietario de dicho contenido o si es un usuario administrador. Esta configuración se encuentra en el fichero `utils/MHelpers.java` en el método `checkAccess()`

2.4.3. Arquitectura del modelo de datos

Para manejar el modelo de datos, se han creado diferentes clases que manejan una tabla determinada de la base de datos como se puede ver en la figura 2.4:

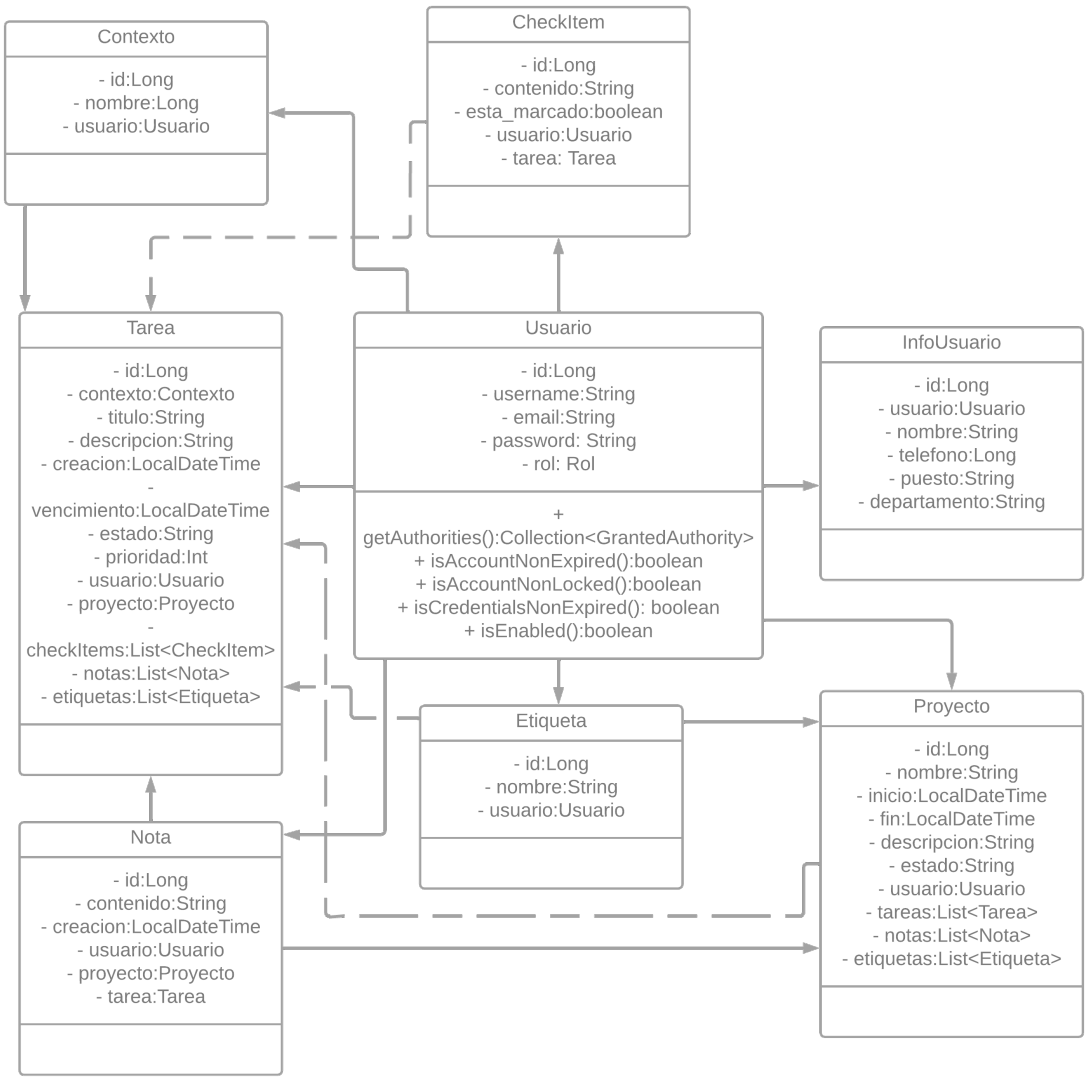

Figura 2.4: Diagrama de clases del modelo de datos de la API Spring Boot

A continuación se describen detalladamente los contenidos de cada clase del modelo de datos:

- `CheckItem`: Esta clase se encarga de procesar los elementos de comprobación de las tareas de la base de datos.
- `Contexto`: Esta clase tiene la labor de procesar los contextos de las tareas de la base de datos.
- `Etiqueta`: Esta clase procesa las etiquetas de las tareas de la base de datos.
- `InfoUsuario`: Esta clase se encarga de gestionar la información adicional de los usuarios de la base de datos.
- `Nota`: Esta clase se encarga de procesar las notas de las tareas o proyectos de la base de datos. Se puede observar que los campos de proyecto y tarea son opcionales, esto es debido a la naturaleza de las notas, que permite que se asignen o bien a una tarea o a un proyecto, pero no permiten asignarse a ambas cosas a la vez o a ninguna de las dos. Esta funcionalidad debe ser gestionada por las aplicaciones cliente.
- `Proyecto`: Esta clase se encarga de procesar los proyectos de la base de datos.
- `Tarea`: Esta clase se encarga de procesar las tareas de la base de datos.
- `Usuario`: Esta clase se encarga de procesar los usuarios registrados en la base de datos.

2.4.4. Desarrollo de la comunicación con la BD a través de repositorios

Spring Boot proporciona una herramienta muy útil para facilitar la interacción con la base de datos denominada repositorios, que se proporcionan por los siguientes paquetes Java en:

`org.springframework.data.jpa.repository.JpaRepository`

Estas interfaces permiten automatizar las consultas SQL, y facilita la obtención de la información de la base de datos. Sin embargo, es necesario especificar qué tipo de operaciones se van a realizar para cada elemento del modelo de datos. Para ello se han creado múltiples interfaces cada una encargada de administrar las consultas relacionadas a su tabla correspondiente en la base de datos, los cuales se detallan en la figura 2.5.

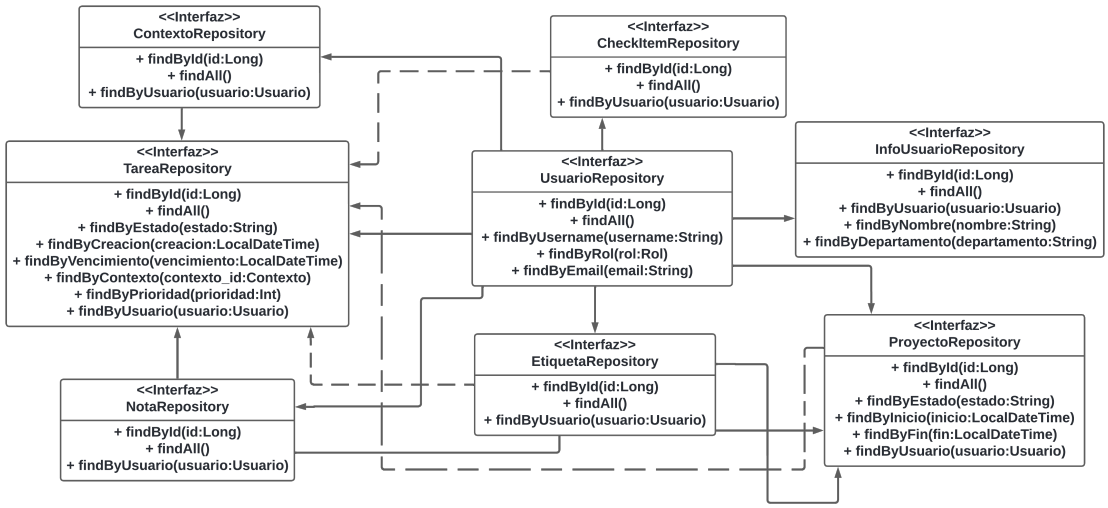

Figura 2.5: Diagrama de clases de los repositorios de la API Spring Boot

Dado que todas las interfaces tienen un funcionamiento similar, se mostrará únicamente un ejemplo de cómo están implementadas.

```java
@Repository
@SuppressWarnings("null")
public interface TareaRepository extends JpaRepository<Tarea, Long>{
    List<Tarea> findAll();
    Optional<Tarea> findById(Long id);
    List<Tarea> findByEstado(@Param("estado") String estado);
    List<Tarea> findByCreacion(@Param("creacion") LocalDateTime creacion);
    List<Tarea> findByVencimiento(@Param("vencimiento") LocalDateTime vencimiento);
    List<Tarea> findByContexto(@Param("contexto_id") Contexto contexto_id);
    List<Tarea> findByPrioridad(@Param("prioridad") int prioridad);
    List<Tarea> findByUsuario(Usuario usuario);
}
```

Figura 2.6: Repositorio de tareas de Spring

Como se puede ver en la figura 2.6, se definen un conjunto de funciones que permiten buscar tareas en base a ciertos parámetros como el estado, la prioridad, o el usuario que ha creado la tarea. Estas funciones se utilizan posteriormente para dar soporte a las clases `Service` e `Implementation` con el fin de entregar la información solicitada a las aplicaciones cliente en forma de `DTOs (Data Transfer Objects)`, los cuales simplifican la información enviada al frontend y se explicarán con más detalle en la sección 2.4.6.

2.4.5. Implementación de las llamadas REST

La gestión de las llamadas que se pueden hacer a la API, así como las operaciones de inicio de sesión se realizan en clases denominadas `Controladores`. En estos se implementa toda la lógica necesaria para que las llamadas a la API funcionen correctamente.

En el proyecto se organizan los controladores en dos paquetes:

1. `auth`: Se encargan de la autenticación del usuario.
2. `controller`: Se encargan de todo lo relacionado al resto de elementos de la base de datos.

2.4.5.1. Desarrollo de los controladores y lógica de autenticación

La definición de los controladores de autenticación y su lógica interna se encuentran en las clases definidas en la siguiente lista:

- Paquete `auth`: Su estructura se ve representada en el diagrama de la figura 2.7.

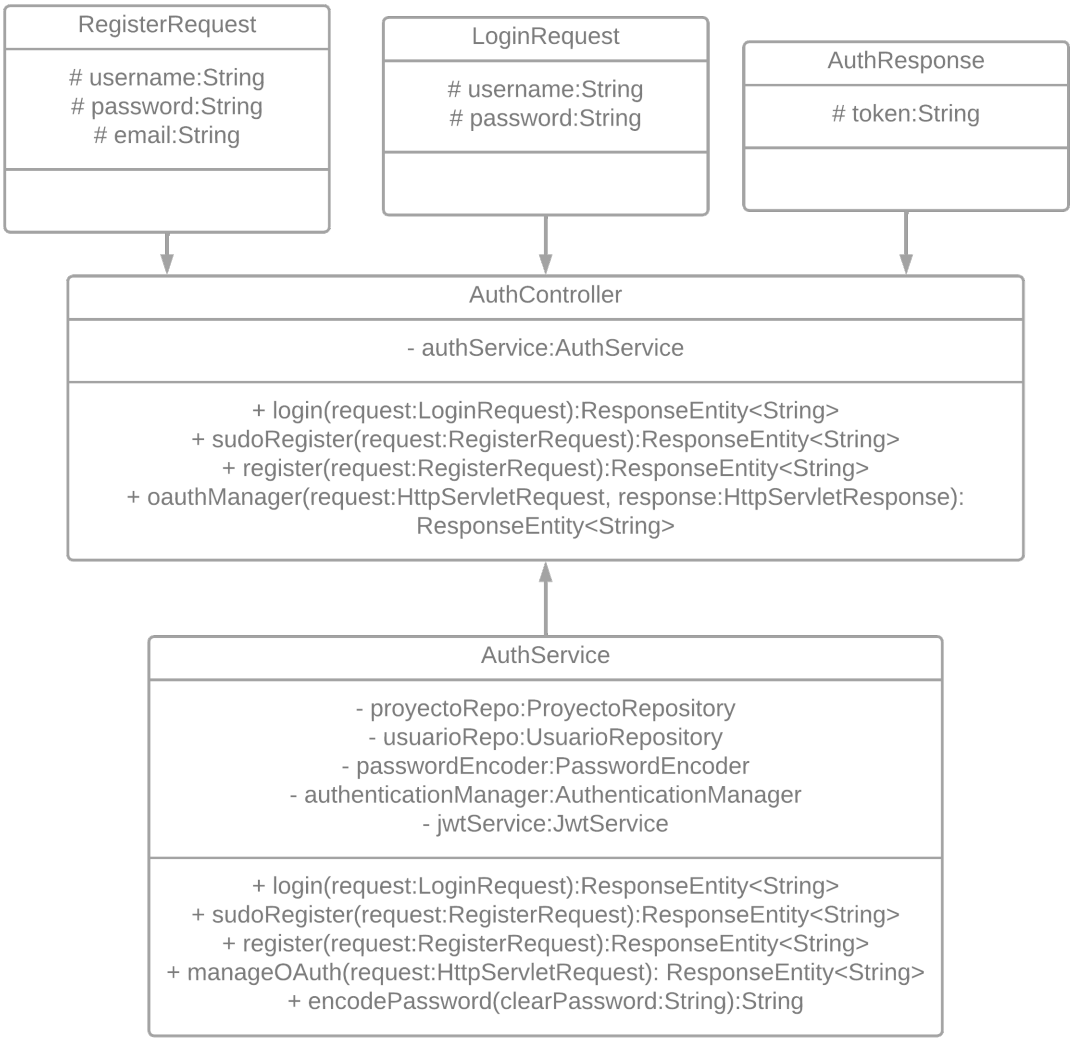

Figura 2.7: Diagrama de clases del paquete auth de la API Spring Boot

- Paquete `controller`: Su estructura se muestra en la figura 2.8

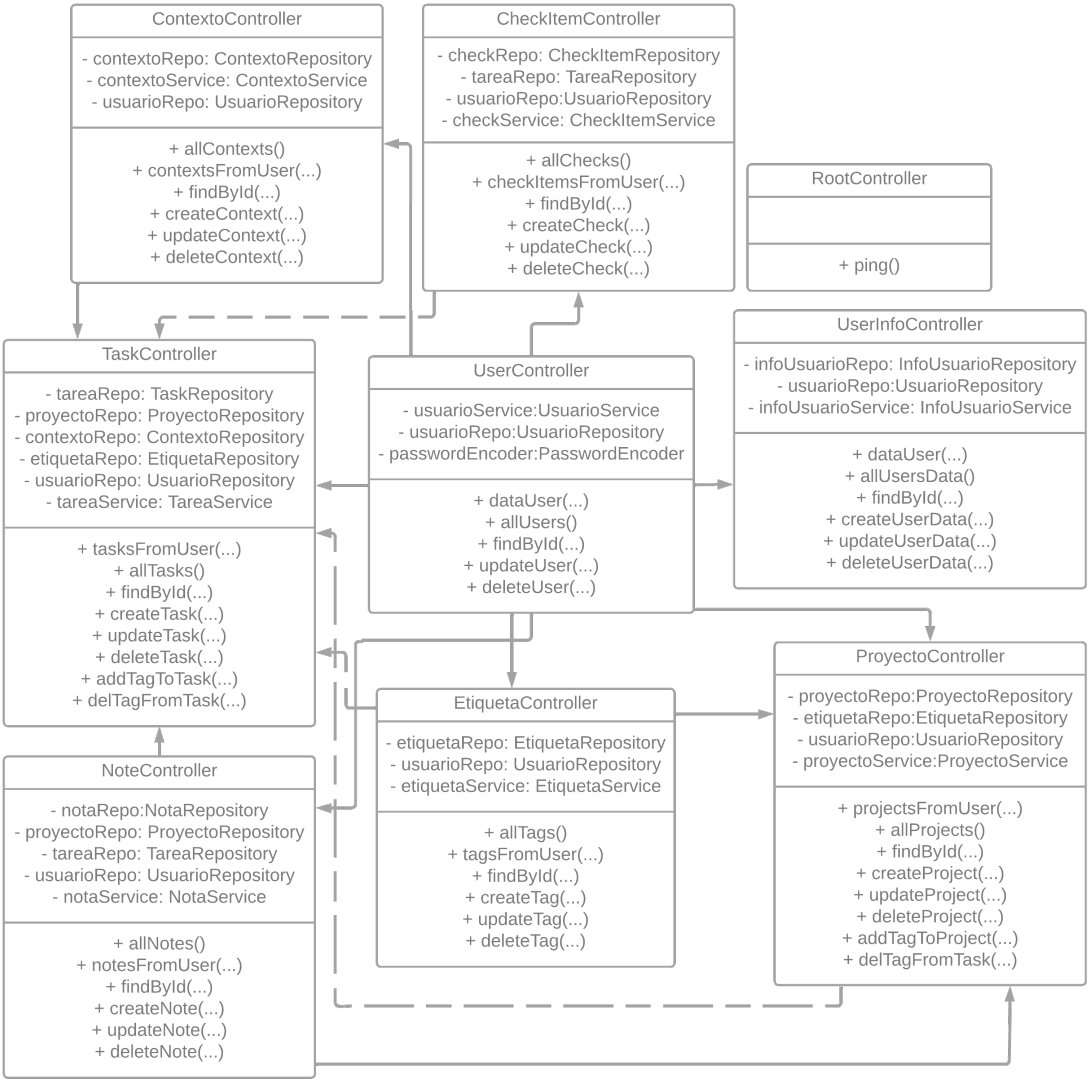

Figura 2.8: Diagrama de clases del paquete controller de la API Spring Boot

- Clase `AuthController.java`: Esta clase contiene los controladores disponibles relacionados a autenticar un usuario en el sistema.

  Este controlador tiene definida una etiqueta `@RequestMapping("/auth")`, la cual indica que todas las rutas dentro de este controlador deberán empezar por `/auth` para que funcionen correctamente.

  A su vez, cada función dentro del controlador tiene una etiqueta Mapping, que le permite asignar una ruta específica y un protocolo HTTP a cada función. Por ejemplo, si la URL del servidor es `https://localhost/`, para poder llamar a la función de login se debería hacer una petición `POST` a la URL `https://localhost/auth/login`.

- Clase `AuthService.java`: Esta clase es la encargada de administrar todas las operaciones relacionadas a la autenticación de los usuarios.

En esta clase también nos encontramos con la función `manageOAuth`, esta función controla el inicio de sesión a través del SSO de Google. En caso de que el usuario existiera antes en la BD, se le generaría un token JWT con su información y podría iniciar sesión. En caso de que no existiera el usuario, se le daría de alta y le daría un JWT para realizar sus operaciones en las aplicaciones cliente.

- Clases `RegisterRequest.java`, `LoginRequest.java`, y `AuthResponse.java`: Estas clases son clases de datos, las cuales se encargan de definir que campos deben recibir las operaciones definidas en `AuthService.java`

2.4.5.2. Implementación de los controladores de datos

Los controladores que administran el resto del contenido de la aplicación están localizados en el paquete `controller` que se muestra en la lista al principio de la sección 2.4.5.1.

Dado lo extenso que resultaría explicar detalladamente el funcionamiento de cada llamada de cada controlador, se dará un ejemplo de uno de los controladores de la API, siendo posible consultar el resto de llamadas en el Apéndice B, que contiene toda la documentación de la API.

```java
@RestController
@RequestMapping("/project")
@SuppressWarnings("null")
public class ProjectController {

    @Autowired
    private ProyectoRepository proyectoRepo;
    @Autowired
    private EtiquetaRepository etiquetaRepo;
    @Autowired
    private UsuarioRepository usuarioRepo;
    @Autowired
    private ProyectoService proyectoService;

    @GetMapping("/authed")
    public ResponseEntity<?> projectsFromUser(HttpServletRequest request) {···


    @PreAuthorize("hasAuthority('ADMINISTRADOR')")
    @GetMapping("/getProjects")
    public ResponseEntity<?> allProjects(){···

    @GetMapping("/{id}")
    public ResponseEntity<?> findById(@PathVariable("id") Long id, HttpServletRequest request){···

    @PostMapping("/create")
    public ResponseEntity<?> createProject(@RequestBody Proyecto project, HttpServletRequest request){···

    @PatchMapping("/update/{id}")
    public ResponseEntity<?> updateProject(@PathVariable("id") Long id, @RequestBody Proyecto project, HttpServletRequest request){···

    @DeleteMapping("/delete/{id}")
    public ResponseEntity<String> deleteProject(@PathVariable("id") Long id, HttpServletRequest request){···

    @PatchMapping("/addTag/{id}")
    public ResponseEntity<?> addTagToProject(@RequestParam("TagID") Long tagId, @PathVariable("id") Long id, HttpServletRequest request){···

    @PatchMapping("/removeTag/{id}")
    public ResponseEntity<?> delTagFromProject(@RequestParam("TagID") Long tagId, @PathVariable("id") Long id, HttpServletRequest request){···
}
```

Figura 2.9: Controladores de proyectos Spring

En el controlador de la figura 2.9 se puede ver que el endpoint inicial se define en:

`@RequestMapping("/project")`

Además, este controlador dispone de las siguientes funciones:

- `projectsFromUser()`: En esta llamada se obtienen todos los projectos que ha creado el usuario que se encuentra actualmente autenticado.
- `allProjects()`: Esta llamada limitada solo para los administradores permite obtener todos los proyectos registrados en la base de datos.
- `findById()`: En esta función se recibe el identificador de un proyecto, el cual se busca en la base de datos, y en caso de existir y que el usuario que realiza la consulta sea el propietario de ese proyecto o sea un administrador, se le devolverá la información asociada a ese proyecto.
- `createProject()`: Esta función se encarga de dar de alta un proyecto nuevo en base a la información proporcionada por las aplicaciones cliente.
- `updateProject()`: En esta función se obtiene el identificador de un proyecto existente, y se modifica su información en la base de datos.
- `deleteProject()`: Este método se hace cargo de eliminar un proyecto del sistema, recibiendo como entrada el identificador del proyecto a eliminar, y borrará el proyecto si el usuario está autorizado a efectuar esa operación sobre ese proyecto.
- `addTagToProject()`: En este método se reciben dos parámetros: El identificador de una etiqueta, y el identificador de un proyecto. En caso de que la etiqueta y el proyecto existan y sean propiedad del usuario autenticado, se asignará esa etiqueta al proyecto.
- `delTagFromProject()`: Esta función recibe el identificador de una etiqueta y el de un proyecto, y eliminará la asignación de la etiqueta a ese proyecto.

2.4.6. Limitando la información enviada a las aplicaciones cliente usando DTOs

Con el fin de simplificar la información enviada a las aplicaciones cliente y evitar enviar información innecesaria a estos, se han implementado clases `DTO` (Data Transfer Object). Los DTO son clases que permiten definir qué datos se van a enviar al frontend, con el fin de ocultar algunas propiedades particulares del modelo de datos que el frontend no debe ver[4]. Sin embargo, estas clases por sí solas no pueden cumplir este cometido, por lo que se debe implementar una serie de unas interfaces `Service` y unas clases `Implementation` que permitan mapear la información que se recibe de los repositorios con la información que se desea enviar a través de los `DTO`. La interacción de los paquetes Service y DTO se explica en la figura 2.10.

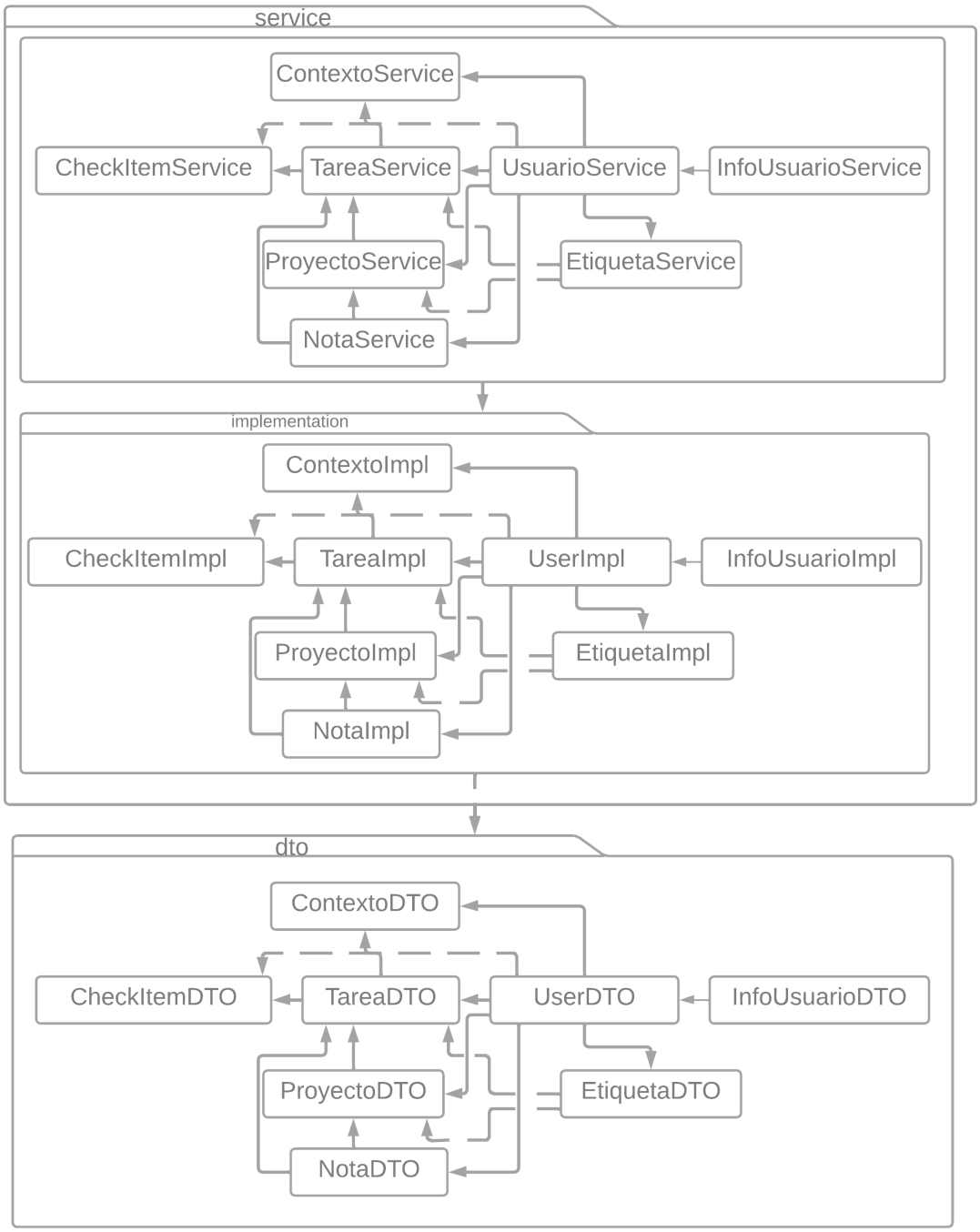

Figura 2.10: Diagrama de clases de los paquetes service y dto de la API Spring Boot

A continuación se explica la funcionalidad de cada tipo de objeto dentro de estos paquetes:

- Interfaces Service: Estas interfaces funcionan como un espejo de las interfaces Repository vistas en la sección 2.4.4, salvo que los Service retornan la información al cliente a través de los DTO.
- Clases Implementation: Estas clases se encargan de realizar el mapeo entre la información recibida desde la base de datos y los objetos DTO.
- Clases DTO: Las clases DTO son una version minimalista de las clases Model, permitiendo elegir cuánta información se desea enviar a las aplicaciones cliente.

Capítulo 3

Diseño de la interfaz de usuario

En este capítulo se presenta el diseño del que se ha partido para elaborar las aplicaciones descritas en los capítulos 4 y 5.

Para este diseño inicial se ha usado la herramienta profesional Figma, que es una muy buena aplicación para hacer prototipos gráficos de aplicaciones e incluso trabajar las interacciones entre las diferentes pantallas y elementos.

Se ha de tener en cuenta que este diseño se toma como una base para desarrollar las aplicaciones, pero debido a cuestiones relacionadas con cada tecnología usada a la hora de programar las aplicaciones, en el resultado final pueden existir algunas variaciones o diferencias respecto a las imágenes que se encuentran en esta sección.

Dado que este proyecto consta de principalmente dos aplicaciones: una multiplataforma y otra nativa para móvil, se han hecho dos diseños: uno para pantallas de escritorio y otro para pantallas de móvil. Sin embargo, hay que tener en cuenta que la aplicación multiplataforma se adapta a distintos tipos de pantallas, y que la aplicación nativa Android se ha adaptado a tablets también.

En general en cuanto a las líneas de diseño se ha optado por hacerlo limpio y minimalista. En la sección 3.1 se exponen las principales pantallas de las que consta la aplicación y se argumenta por qué se han tomado estas deciciones en base al modelo de datos y el flujo GTD. En la sección 3.2 se explica el diseño de los temas la aplicación para lograr una mayor personalización de la misma.

3.1. Pantallas

3.1.1. Inicio de sesión

En las figuras 3.1 y 3.2 se pueden ver las pantallas de inicio de sesión, tanto para escritorio como para móvil respectivamente. Como se observa, esta pantalla es sencilla y consta de un formulario con tres campos:

- URL del backend: Como esta aplicación puede ser desplegada por una organización en concreto, por ejemplo la UCM, la idea es que la URL del servidor se pueda cambiar dependiendo de dónde quede alojada, es por esto que existe este campo donde se puede introducir la URL de una instancia particular.

- Nombre de usuario: Identificador único del usuario.
- Contraseña.

A continuación existe un botón para iniciar sesión. Adicionalmente se puede hacer login con Google. Abajo del todo se puede encontrar un texto con un link que permite ir a la página de registro para crear una cuenta nueva.

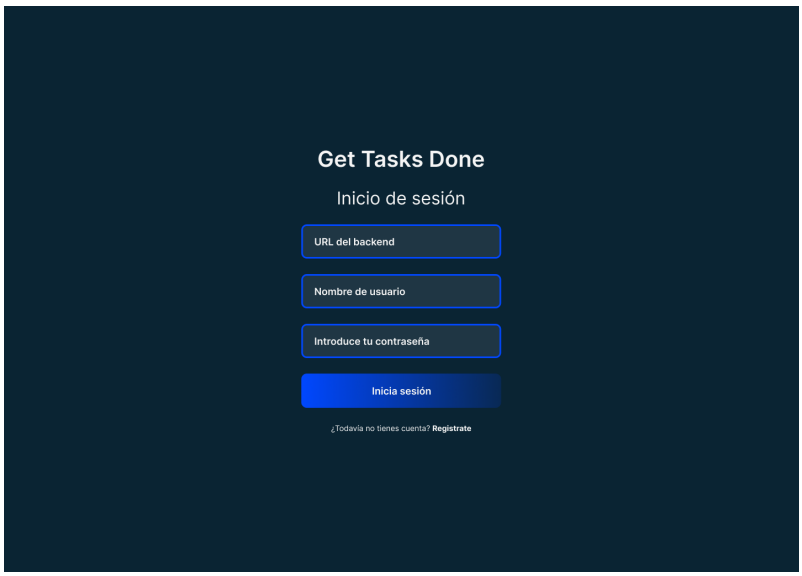

Figura 3.1: Inicio de sesión en escritorio

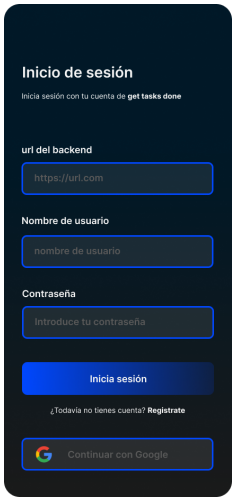

Figura 3.2: Inicio de sesión en móvil

3.1.2. Registro

En las figuras 3.3 y 3.4 se pueden ver las pantallas de registro de un usuario nuevo tanto para escritorio como para móvil respectivamente.

Como puede observarse, esta pantalla es sencilla y consta de un formulario con cuatro campos:

- URL del backend.
- Nombre de usuario.
- Correo electrónico.
- Contraseña.

A continuación existe un botón para crear la cuenta. Abajo del todo se puede visualizar un texto con un link que permite ir a la página de inicio de sesión en el caso de que ya se tenga una cuenta.

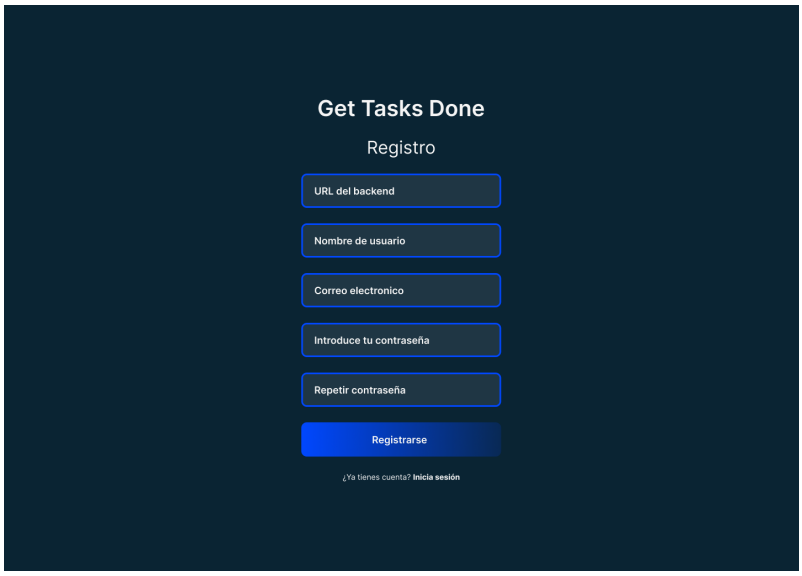

Figura 3.3: Registro de usuario en escritorio

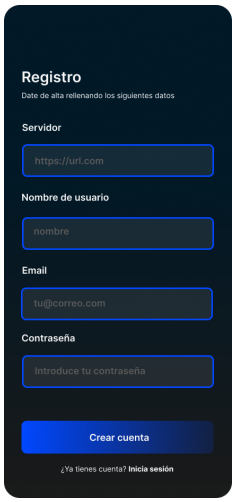

Figura 3.4: Registro de usuario en móvil

3.1.3. Completar registro

Después de crear un usuario, este debe introducir algunos datos adicionales, para lo que se ha hecho esta pantalla. En las figuras 3.5 y 3.6 se puede ver las pantallas de completar registro de un usuario nuevo tanto para escritorio como para móvil respectivamente. Como puede observarse, esta pantalla es sencilla y consta de un formulario con cuatro campos:

- Nombre.
- Número de teléfono: Este campo es opcional.
- Puesto de trabajo.
- Departamento dentro de la empresa.

A continuación existe un botón para completar registro.

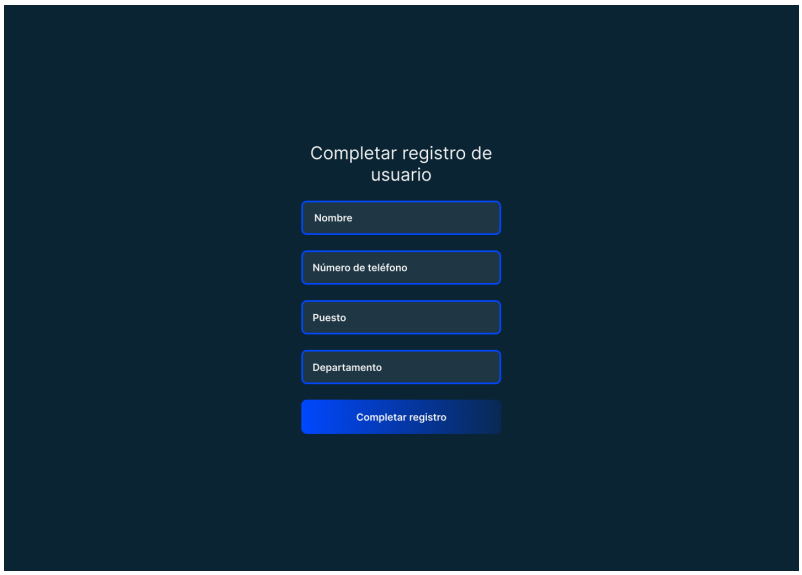

Figura 3.5: Completar registro de usuario en escritorio

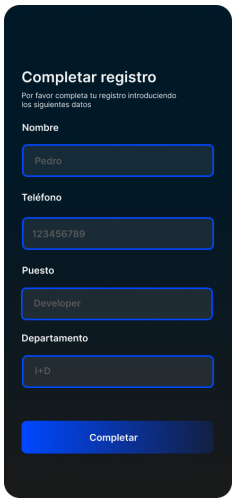

Figura 3.6: Completar registro de usuario en móvil

3.1.4. Página principal

Una vez el usuario ha pasado por el proceso de creación y completado de datos o de login, se llega a la pantalla principal. En las figuras 3.7 y 3.8 se pueden ver las pantallas principales tanto para escritorio como para móvil respectivamente. En el lado izquierdo de la pantalla podemos observar un menú, que sirve para navegar por dentro de la aplicación y tiene los siguientes elementos:

- Bandeja de entrada: Aquí aparecen todas las tareas que todavía no hemos completado, independientemente de si están marcadas con algún atributo que vemos a continuación.
- Lo siguiente: Aquí aparecen todas las tareas que están marcadas como siguiente.
- Esperando: Aquí se listan todas las tareas que están marcadas como esperando.
- Agendado: Aquí se listan todas las tareas que tienen una fecha de finalización.
- Algún día: Aquí aparecen las tareas que están marcadas con algún día.
- Importante: Aquí se listan todas las tareas que están marcadas como importantes.
- Proyectos: Aquí podremos ver los proyectos.
- Etiquetas: Aquí podemos ver las etiquetas que hayamos creado.
- Cerrar sesión.

Por defecto, cuando se accede a la aplicación se envía al usuario a la página de Bandeja de entrada, por lo que a la derecha, o en el panel principal, se podrá ver una lista de tareas que se tienen en esta sección.

En la parte inferior derecha de la pantalla, se puede observar un botón, que sirve para crear tareas o proyectos.

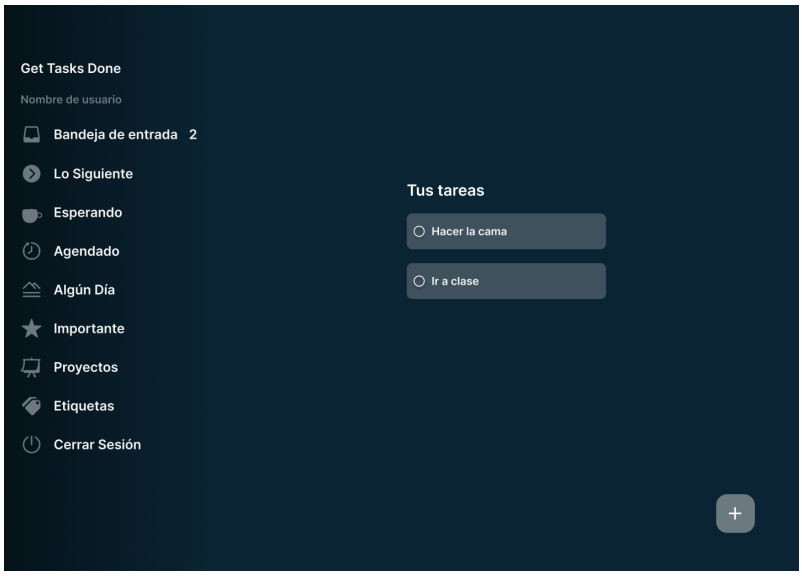

Figura 3.7: Pantalla principal en escritorio

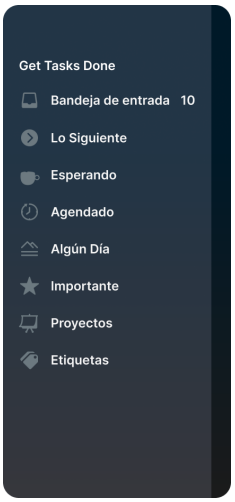

Figura 3.8: Pantalla menú en móvil

3.1.5. Crear tareas

Ahora el usuario ya puede crear tareas, para lo que hay un formulario, que se presenta en las figuras 3.9 y 3.10.

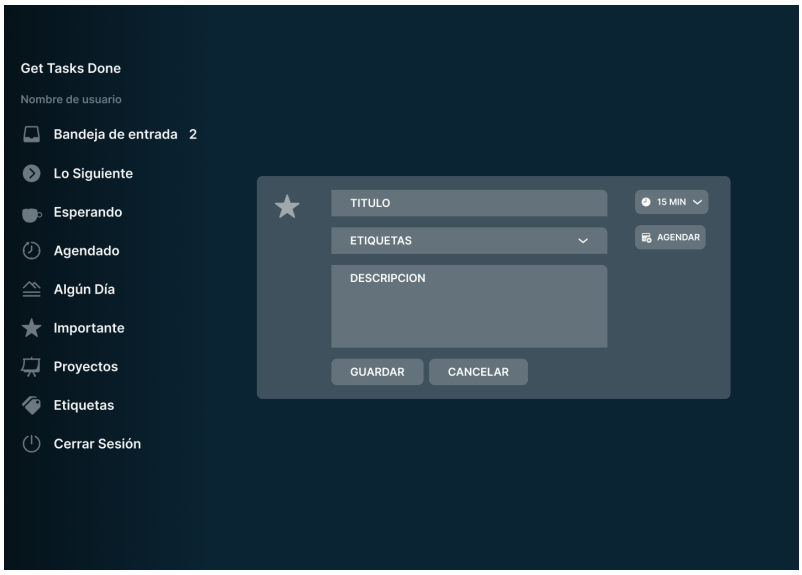

Figura 3.9: Formulario crear tarea en escritorio

Figura 3.10: Formulario crear tarea en móvil

Para crear una tarea, debemos rellenar, al menos, los siguientes campos:

- Titulo.
- Descripción.
- Etiquetas: Aquí es donde podríamos marcar una tarea para que tenga un determinado contexto, o se le aplique un filtro.
- Prioridad: Podemos marcar una tarea como prioritaria presionando sobre la estrella.
- Agendar: Podemos asignar una fecha a una tarea.

Si revisamos el modelo de datos, veremos que hay otros campos como el proyecto al que está asignado la tarea que son obligatorios a la hora de crearla, en el diseño inicial se ha considerado no incluir este campo ya que las aplicaciones serán capaces de detectar el proyecto en el que está trabajando y por tanto dentro del cual se va a crear la tarea.

3.1.6. Resto de pantallas

Habiendo definido el estilo y distribución de las anteriores pantallas, el resto van a ser muy similares a estas, pues consistirán en su mayoría de formularios como el de la pantalla de crear tareas, o bien de vistas como las de la pantalla principal. En las aplicaciones finales se hará uso de recursos tanto para agregar funcionalidades como para ”adornar” que no merece la pena especificar en este diseño.

3.2. Temas

Para lograr una mayor personalización de la aplicación, se ha implementado un sistema de temas sencillo, que permite cambiar los colores principales de la aplicación en tiempo de ejecución. Se ha optado por el enfoque tradicional de tener dos temas, el oscuro y el claro. Con este objetivo en mente, se diseña cómo deberían quedar las páginas para cada uno de estos temas. En las figuras 3.11 y 3.12 se puede ver las pantallas principales para escritorio con el tema oscuro y claro respectivamente.

Los temas no implican cambios en los elementos que se muestran en las pantallas, tan solo en sus estilos y colores.

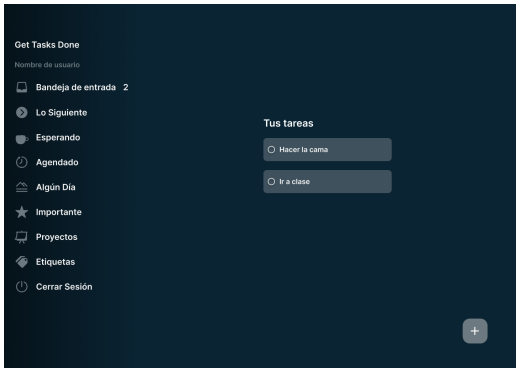

Figura 3.11: Pantalla principal en escritorio tema oscuro

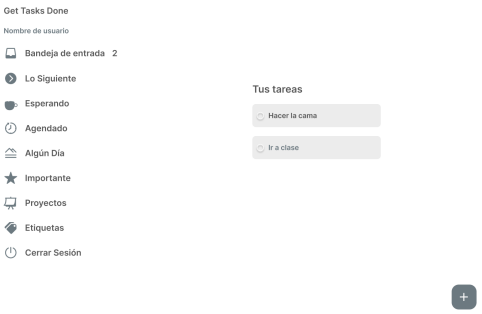

Figura 3.12: Pantalla principal en escritorio tema claro

Capítulo 4

Desarrollo del cliente web y de escritorio

En este capítulo se describe la implementación y funcionamiento del cliente multiplataforma de la aplicación, tanto para navegador como escritorio. También se proporciona una explicación parcial del diseño.

En la sección 4.1 se presenta las tecnologías de las que se han hecho uso, justificando dicho uso. Posteriormente, en la sección 4.2 se comenta la estructura del proyecto. En la siguiente sección, 4.3, se explican las pautas de la adaptación del diseño propuesto en el capítulo 3 y las consideraciones tomadas para la realización de una interfaz amoldable a cambios de dimensionalidad. A continuación, en la sección 4.4 se exponen las políticas de manejo y registro de datos del usuario de forma persistente. Seguidamente, en la sección 4.5 se exhibe el funcionamiento de la lógica del cliente. Posteriormente, la sección 4.6 se presenta la comunicación del cliente con el servidor, y las políticas de sincronización entre estos. Siguientemente, en la sección 4.7 se explica la forma de distribución de los distintos clientes. Finalmente en la sección 4.8 se ilustra un manual de uso del cliente.

4.1. Tecnologías utilizadas

Para la implementación del cliente de escritorio y web se decidió optar por una tecnología multiplataforma relativamente puntera. Por lo tanto, se decidió desarrollar los clientes en el framework Flutter [10], una librería para el lenguaje Dart. Tanto Dart como Flutter son tecnologías que se han desarrollado por Google. Flutter como tecnología aspira a facilitar el desarrollo de apliaciones para múltiples plataformas a través de un código fuente único para todos los clientes. La solución propuesta por Flutter «compila» el código fuente para cada cliente, dando por resultado un objeto nativo de la plataforma. Por ejemplo, si se decide construir la aplicación para un escritorio de Linux, Flutter transforma el código fuente que se escribió en Dart a C, y genera un ejecutable nativo de Linux compilando el proyecto convertido en C.

Otra ventaja de Flutter reside en el ecosistema de librerías o paquetes asociadas al framework o a Dart, habiendo habido paquetes de los cuales se han hecho uso en esta solución.

A parte de la ambición asociada al framework utilizado, existen una serie de

desventajas que se deben analizar de forma meticulosa. Tanto Dart, pero sobre todo Flutter, son tecnologías relativamente nuevas, por lo que no existe un ecosistema extenso asociado, de la misma forma que su uso en la industria tampoco es extenso.

4.2. Arquitectura

Debido a la naturaleza del desarrollo en Flutter y su paradigma, la estructura del proyecto y la arquitectura siguen el mismo patrón, tratándose de una estructura horizontal. El paradigma de Flutter revuelve sobre la composición, a diferencia de la herencia, por lo que la relación entre clases no es vertical, como padres e hijos, sino horizontal, donde una clase «hijo» puede tener varias clases «padres». Aunque se debe aclarar que, cuando se está hablando de elementos gráficos, es decir, código que pertenece a la representación de la interfaz, no se está hablando de clases, sino de más bien «widgets». Es en esta relación en la que se basa Flutter, no la herencia entre widgets, sino la composición.

En general todos los elementos del proyecto pertenecen de alguna manera u otra a este elemento de widget, justificando el razonamiento que se realizó ulteriormente.

Arquitectónicamente, el cliente está separado en las siguientes partes:

- **Diálogos**: Contiene todos elementos que sean cuadros de diálogo, pop-ups o similar. El manejo de este tipo de elementos en Flutter es un poco distinto al manejo de las pantallas, widgets o vistas. Los diálogos incluyen los formularios de creación y edición de objetos, avisos, y similar. Se parecen más a las pantallas, pero como se mencionó anteriormente, se manejan de forma distinta.

- **Lógica**: Toda la lógica independiente de elementos gráficos. La lógica está separada en dos partes, las llamadas a la API, y las estructuras de datos. Se expande sobre la lógica en la sección 4.5.

- **Mixins**: Código que se puede reutilizar en múltiples jerarquías de clases. Un «mixin» permite añadir miembros a un clase de forma horizontal, a diferencia de la herencia, donde la relación es vertical. El mixin permite agregar lógica a una clase independiente de que implemente, herede o que otros mixins incluya la clase.

- **Providers**: Elementos del paquete de Flutter «Riverpod» [5]. En la sección 4.5 se explica el uso de Riverpod de manera más extensa. Se enumeran los siguientes proveedores de estado en el cliente:

  - *Finalización del formulario de registro*: Indica el estado de finalización del proceso de registro.
  - *Contador de bandeja de entrada*: Como su nombre bien indica, comparte la cantidad de tareas que se encuentran en bandeja de entrada.
  - *Inicialización de la aplicación*: Indica el estado de inicialización del cliente.

Figura 4.1: Pantalla «aplicación» del cliente web

- *Nuevo usuario*: Notifica a la aplicación si el usuario es de nueva creación. Esto es de interés debido a que se obliga a los nuevos usuarios a crear un primer contexto en el momento de su creación.
- *Encaminamiento*: Proveedor que comparte el encaminador de la apliación. El encaminador se encarga de realizar el redireccionamiento dependiendo del estado de la aplicación, por ejemplo, redireccionar al usuario a la pantalla de inicio de sesión si su token de sesión es inválido. La navegación entre pantallas se realizar a través de este encaminador, que se ha implementado con el paquete «go_router» [11].
- *Token de sesión*: Guarda el token de sesión, que se usa para la autorización de varias de las peticiones al servidor, además de indicar la validez de la sesión del usuario.
- *Tema*: Guarda el tema del cliente.
- *Nombre de usuario*: Guarda el nombre del usuario, que se obtiene en el inicio de sesión o al registrarse.

- **Pantallas**: Las pantallas de la aplicación. Una pantalla es un elemento gráfico-lógico que encapsula el estado compartido entre todos elementos dentro de una pantalla. Estas pantallas están vinculadas a una URL, por lo que existe una forma de navegar a esta desde otra pantalla. Se presentan a continuación las pantallas:

  - *Aplicación*: La pantalla de la apliación en sí, es decir, la aplicación GTD donde el usuario puede realizar las acciones y cambios oportunos. Contiene el menú de la aplicación, la lógica de redimensionamiento, además del botón flotante para la creación de tareas en bandeja de entrada. Esta pantalla se puede ver en la figura 4.1.

Figura 4.2: Pantalla «completar registro» del cliente web

  - *Completar registro*: Pantalla de finalización de registro, que contiene los campos de nombre, número de teléfono, puesto y departamento. La apariencia de esta pantalla se puede ver en la figura 4.2
  - *Inicialización*: Una pantalla especial que sirve como inicializador de la aplicación. Comprueba el estado que se haya guardado anteriormente y toma las pautas necesarias para contruir el estado inicial de la aplicación. Esta pantalla se muestra como una barra de carga circular al abrir la apliación web.
  - *Inicio de sesión*: Pantalla de inicio de sesión, incluyendo los campos de URL del servidor, nombre de usuario y contraseña. La pantalla se puede ver en la imagen 4.3
  - *Registro*: Primer paso del proceso de registro. Contiene campos parecidos al inicio de sesión, pero adaptados al proceso de registro. Esta pantalla está definida en la figura 4.4

- **Utilidades**: Contiene varios elementos misceláneos que individuales o únicos en naturaleza. Por ejemplo, estructuras de datos generales o básicos, métodos auxiliares, declaración de los temas, entre otros.

- **Vistas**: Elemento gráfico-lógico que se puede comparar a una «sub-pantalla». Parecidas a las pantallas, pero la diferencia es que las vistas pertenecen a una pantalla, y no tienen URL asociada, por lo que no se puede navegar a una vista desde otras partes de la aplicación. Una pantalla puede tener varias vistas. Las vistas en el proyecto pertenecen a la pantalla de aplicación:

  - *Contextos*: Vista de listado de contextos. Permite la creación, edición y borrado de contextos.

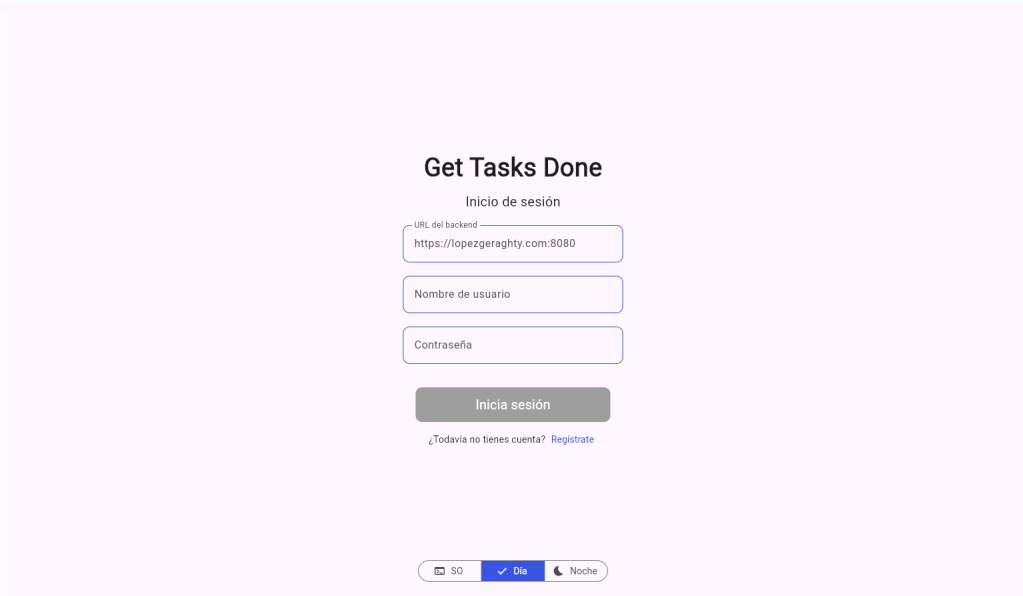

Figura 4.3: Pantalla «inicio de sesión» del cliente web

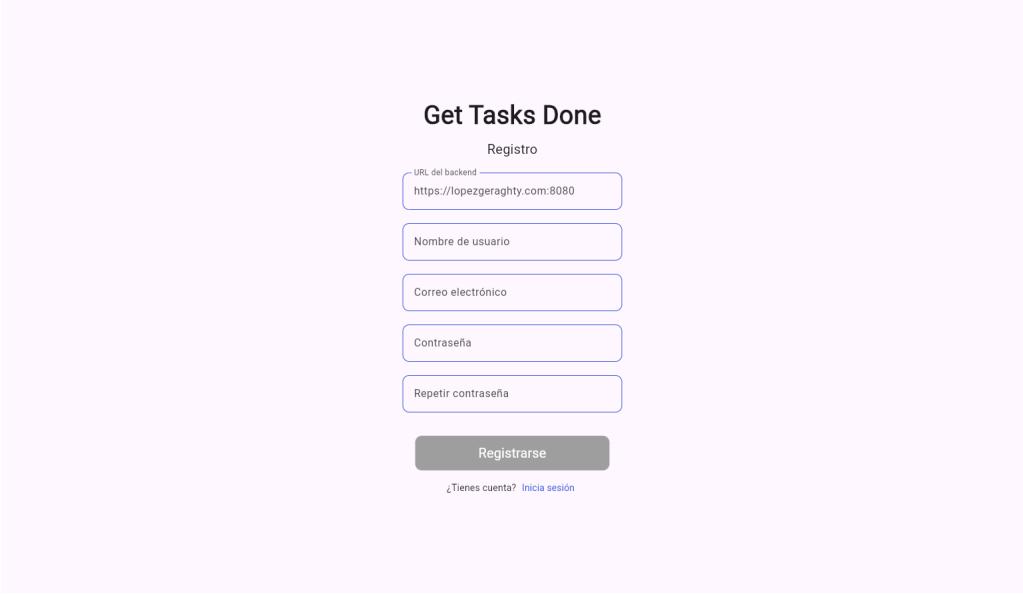

Figura 4.4: Pantalla «registro» del cliente web

  - *Proyecto*: Vista de listado de proyectos. Permite la creación, edición, borrado de proyectos, además de mostrar las tareas asociadas a cada proyecto, así como la creación de tareas asociadas a proyectos.
  - *Tareas*: Vista de listado de tareas. Permite la creación, edición y borrado de tareas. También permite la el filtrado de tareas que se listan. De esta manera se consigue representar las distintas categorías en el menú de la aplicación.

- **Widgets**: Son el elemento mínimo de la arquitectura, que representan los elementos interactivos de la aplicación, o son elementos lógicos que definen la relación entre widgets. Los campos, las tarjetas, los botones son todos widgets. En Flutter, los Widgets se definen a partir de la composición de otros widgets. En el proyecto existen elementos interactivos que son únicos, es decir, que solo se usan una vez. Este tipo de widgets se definen en línea. Por otra parte, existen widgets que se repiten en varias partes de la aplicación, por lo que conviene definir estos elementos como widgets propios. En Flutter existen dos tipos de widgets, sin estado y con estado. Los widgets sin estado no contienen ningún tipo de estado propio, como por ejemplo las tarjetas. Las tarjetas simplemente se encargan de mostrar información al usuario. En cambio, los widgets con estado tienen atributos y lógica de la que depende su representación, y en general contienen un método heredado «setState» que indica el estado del widget ha cambiado y requiere su reconstrucción (redibujado).

4.3. Diseño adaptivo

Debido a la posibilidad de poder acceder a la aplicación web desde cualquier navegador, se exigía la usabilidad de esta en con distintas dimensiones, tanto en un escritorio, como en un móvil. Flutter facilita este proceso de diseño adaptivo, debido a que en general, el tamaño de los widgets es flexible, y viene determinado por la relación que existe entre otros widgets y la disposición de todos estos en la pantalla. Por lo tanto, en general no existe la definición de dimensionalidad en número de píxeles, sino que Flutter lo calcula a partir de las relaciones mencionadas anteriormente. Por lo tanto, los widgets se pueden redimensionar con facilidad, y la composición de la pantalla no se pierde por el redimensionamiento.

Aún existiendo los mecanismos descritos en el párrafo ulterior, se tuvo que tener en cuenta pautas en el proceso del diseño de la interfaz del cliente, para que cumpliera con la usabilidad en todo tipo de pantallas. Este proceso requirió de varias iteraciones del diseño, pero culminó en un diseño suficiente. Para lograr este objetivo se tuvo que tener en cuenta la recomposición de algunos de los elementos de la aplicación que reaccionasen a los cambios de dimensión de la ventana de la aplicación, manteniendo su usabilidad. Uno de estos elementos es el menú, donde en las figuras 4.5 y 4.6 se puede ver cómo cambia su composición dependiendo del tamaño de la pantalla.

Figura 4.5: Menú de la aplicación en dimensión grande

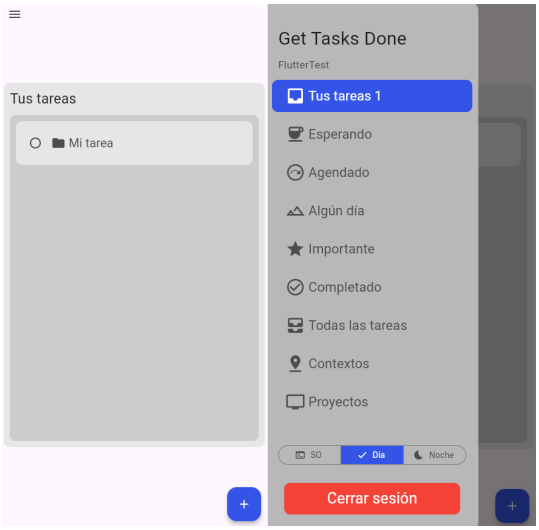

Figura 4.6: Menú de la aplicación en dimensión pequeña

Por otra parte, debido al hecho de realizar el diseño adaptivo, se da la posibilidad de uso de la aplicación en escritorio con la dimensión que el usuario desee. Finalmente, el diseño del cliente se basó en la interfaz propuesta en el capítulo 3, aunque no se trata de una réplica exacta, debido a las diferencias inherentes de las tecnologías.

4.4. Almacenamiento local

Para el manejo de los datos de usuario persistentes se ha hecho uso del paquete «flutter_secure_storage» [22]. Este paquete proporciona una API multiplataforma para el almacenamiento seguro de los datos de la aplicación en local, que posibilita el desarrollo de un único código para el almacenamiento en cualquier plataforma,

como se puede ver en el siguiente ejemplo del manejo del token de sesión:

```
import '.../flutter_secure_storage.dart';

class SessionToken {
  static const SecureStorage _storage = SecureStorage(); //
   Referencia al objeto interfaz de almacenamiento.
  static const String _key = 'session_token';

  String? _token;

  // Leer del almacenamiento local. Si no se encuentra la clave, no
    se lee nada.
  static Future<String?> readFromStorage() async {
    if (await _storage.containsKey(key: _key)) {
      return await _storage.read(key: _key);
    }

    return null;
  }

  void set(String? token) async {
    _token = token;

    // Escribir en nuevo token de sesion en local.
    if (token != null) {
      await _storage.write(key: _key, value: token);
    } else { // Borrar el token de local en invalidacion.
      await _storage.delete(key: _key);
    }
  }
}

```

De esta forma se realiza el guardado de los datos oportunos. A continuación se listan justificadamente estos datos:

- **URLs del servidor**: Se guardan la lista de URLs que se hayan introducido en el campo de configuración de URL del servidor. De esta manera el usuario tendrá acceso a sugerencias para la URL de configuración, independientemente de que servidor se haya conectado.

- **Token de sesión**: Se almacena el token de sesión si no se ha cerrado sesión en la aplicación, para evitar el inicio de sesión en la aplicación cuando se vuelva a abrir. Si hay un token guardado en la inicialización del cliente, se comprueba su validez y se redirecciona al usuario adecuadamente.

- **Tema**: La elección del tema se guarda de forma persistente independientemente del estado de la aplicación, y es un atributo completamente estético. La preferencia del tema es a nivel local, y no se comparte entre instancias.

- **Nombre de usuario**: Se guarda debido a la persistencia del token de sesión, ya que en la aplicación en sí se muestra el nombre de usuario.

4.5. Lógica del cliente

La lógica está separada en tres partes principales, las llamadas a la API, las estructuras de datos, y el estado. Como se ha indicado, las llamadas a la API son las peticiones del cliente a la API del servidor. Se manejan desde un mismo fichero, debido a que las peticiones se realizan en varias partes del código de la aplicación, además de que existe una lógica compartida entre las distintas peticiones.

Otra parte de la lógica son las estructuras de datos. Una es la la configuración del servidor, que se encarga de almacenar la URL del servidor que se haya configurado por el cliente, entre otras cosas. Otra estructura de datos se corresponde a los datos de usuario, un *singleton* que se encarga de almacenar todos los elementos del modelo de datos del usuario, análogos a los elementos del modelo de datos del servidor, presentado en la sección 2.2. Para cada uno de los elementos de datos del usuario, como las tareas o proyectos, existe una lógica específica asociada, además de una lógica compartida entre todos estos. Se trata de la funcionalidad correspondiente a la codificación y decodificación de JSON, que es la notación utilizada por la API del servidor. De esta forma todos los elementos de datos tienen métodos para codificarlos o decodificarlos de forma sencilla en otros componentes de la aplicación.

Finalmente está la lógica asociada al estado de la aplicación. El estado suele estar encapsulado por los *widgets*, desde elementos como los botones hasta la pantalla. Pero existe un estado compartido entre todos estos elementos, que se maneja con Riverpod. Riverpod es un framework de caché y enlazado de datos, permitiendo encapsular la lógica de comunicación de estado entre varias pantallas o elementos lógicos de forma relativamente sencilla e idiomática. Proporciona la notificación automática del cambio de estado entre distintas partes de la aplicación. De esta forma, se puede notificar el número de tareas en la bandeja de entrada, que se representa en la pantalla de aplicación, desde el formulario de creación de tarea, por ejemplo. A continuación se muestra un proveedor del cliente con Riverpod:

```
1  import 'package:riverpod_annotation/riverpod_annotation.dart';
2
3  part 'inbox_count.g.dart';
4
5  @Riverpod(keepAlive: true) // Anotacion para marcar la clase como
       un proveedor Riverpod, con el flag keepAlive.
6  class InboxCount extends _$InboxCount {
7    int _count = 0; // Estado especifico del proveedor.
8
9    @override
10   int build() { // Metodo para leer el estado, a traves de ref.
       watch().
11     return _count;
12   }
13
14   // Funciones especificas al proveedor.
15   void set(int count) {
16     _count = count;
17
18     ref.invalidateSelf(); // Metodo que permite la notificacion del
       cambio de estado a partes del codigo donde se este escuchando
```

```
    con ref.watch().
  }

  void addOne() {
    set(_count + 1);
  }

  void substractOne() {
    set(_count - 1);
  }
}

```

4.6. Llamadas a la API y sincronización con el servidor

Como se describió en la sección 4.5, el cliente realiza peticiones a la API de servidor. Dependiendo de la petición, hace falta adjuntar el token de sesión como autorización. Estas llamadas ocurren en varias partes de la aplicación, algunas por parte del cliente en sí, como en la inicialización, y otras por interacción del usuario. En general, las respuestas de estas llamadas suelen tener un cuerpo, en formato JSON, que es el que describe el elemento de datos.

Para mantener el mismo estado entre servidor y cliente, además de realizar la petición al servidor, el cliente realiza dos acciones. El primer caso, como por ejemplo, la creación de una tarea, se maneja en local, es decir, se hace la petición de creación de tarea al servidor, y en el cliente se crea la tarea en el modelo de datos al mismo tiempo. El segundo caso se maneja invalidando el estado del cliente. Este caso se da en acciones que tenga alta repercusión en el modelo de datos del usuario, como por ejemplo, eliminar un contexto. La eliminación de contexto en el servidor también elimina todas las tareas asociadas, que a su vez pueden estar asociadas a un proyecto. Por lo tanto, debido al posible alto nivel de repercusión de la acción, en cliente se invalida el estado completamente y se vuelve a cargar desde servidor, de la misma manera que se carga en la inicialización. A continuación se muestra el código representativo del último ejemplo:

```
// Metodo a ejecutar en el borrado de contexto.
await deleteContext(
  ref, // Objeto para trabajar con la interfaz de Riverpod.
  context.id,
  () async {
    userData.clear(); // Invalidacion del estado local.
    userData.loadUserData( // Cargar estado del servidor.
      ref,
      await getUserDataResponse(ref), // Bytes de la respuesta del servidor para los datos de usuario.
    );

    if (buildContext.mounted) { // Comprobar si ha finalizado la construccion del arbol de widgets.
```

```
13        buildContext.pop(); // Desapilar el dialogo de confirmacion
      de borrado de contexto.
14        buildContext.pop(); // Desapilar el formulario de edicion de
      contexto.
15
16        setParentState(); // Actualizar el estado del widget padre (
      la pantalla).
17      }
18    },
19 );
20
```

4.7. Distribución

La distribución de la aplicación de Flutter es distinta dependiendo de la plataforma a la que se quiera compilar, ya que Flutter empaqueta la aplicación en código nativo de la plataforma. Primero, se requiere compilar el proyecto para la la plataforma en cuestión, que se puede realizar con el siguiente comando:

```
1 flutter build <PLATAFORMA> --release
2
```

Una vez que se haya compilado el proyecto para la plataforma específica, se puede distribuir. Si es una aplicación web, tan solo hay que dejar el proyecto compilado donde se sirvan los ficheros del servidor. El paquete web contiene un fichero `index.html`, que es el que hay enviar al cliente desde el servidor, y el resto de ficheros los cargará la aplicación de Flutter. También cabe destacar que la aplicación web de Flutter permite instalarse en local desde un navegador Chromium, como se muestra en la figura 4.7.

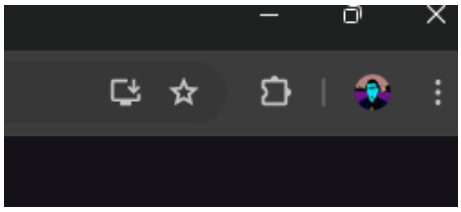

Figura 4.7: Navegador Chrome con posibilidad de instalar la aplicación web en local

El paquete de escritorio será distinto dependiendo para que sistema operativo se haya compilado, además que también existirán opciones de distribución dependiendo del sistema. Pero en general para todos los escritorios se compilará un paquete que incluirá un ejecutable.

4.8. Manual de usuario

4.8.1. Introducción

La interfaz está compuesta de varias pantallas, que se dividen en dos categorías: manejo de sesión y aplicación. El usuario tendrá que iniciar sesión o registrarse para

continuar a la aplicación, donde podrá manejar sus tareas y proyectos.

El objetivo del cliente web y escritorio es el de proporcionar una interfaz de usuario amigable y usable para interactuar con el servidor, de forma que el usuario en cuestión pueda usar la metodología GTD según la interpretación de la aplicación.

4.8.2. Requerimientos e instalación

Para el uso del cliente desde la web tan solo hace falta cualquier navegador, pero debido a la naturaleza de la tecnología, el cliente está optimizado para el uso en navegadores Chromium.

Es posible la instalación del cliente como aplicación web desde un navegador Chromium, como se mencionó en la sección 4.7 (o referirse a la figura 4.7). Este método de instalación se puede hacer tanto en equipos de escritorio como telefónos móviles. La descarga e instalación de la aplicación web no requiere de ninguna configuración posterior.

4.8.3. Comenzando a usar la aplicación

Para poder usar el cliente, hace falta iniciar sesión en el servidor con una cuenta perteneciente a tal servidor. El servidor de la aplicación se puede configurar desde el campo `URL del backend`, disponible tanto en la pantalla de inicio de sesión, como la de registro. Si uno no posee de usuario en el servidor en cuestiçon, tendrá que registrarse a través de la pantalla de registro, accesible al presionar sobre el texto `Regístrate` en la pantalla de inicio de sesión. Esta pantalla se muestra en la figura 4.3. En esta pantalla como en la siguiente también hay un control al final que permite cambiar el tema de la aplicación.

La pantalla de registro es similar a la pantalla de inicio de sesión. Para empezar el proceso de registro, el usuario tendrá que rellenar los campos de `Nombre de usuario`, `Correo eletrónico`, `Contraseña` y `Repetir contraseña`. Una vez rellenados los campos, al presionar `Regístrate`, el usuario se encontrará con la pantalla de finalización de registro. La pantalla de registro es visible en la figura 4.4.

En la pantalla de finalización de registro, el usuario tiene que rellenar los campos de `Nombre`, `Puesto` y `Departamento`. El campo `Número de teléfono` es opcional, y no es necesario para completar el proceso de registro. Esta pantalla se muestra en la figura 4.2.

Si se ha iniciado sesión con una cuenta existente, el usuario podrá seguir a la aplicación con normalidad. Si la cuenta es de recién creación, al entrar en la aplicación, el usuario tendrá que crear su primer contexto a través del formulario que se encontrará abierto en pantalla, visible en la figura 4.8. La creación del primer contexto no es opcional, pero una vez creado, el nuevo usuario podrá proseguir a usar el resto de la aplicación libremente.

Figura 4.8: Formulario de creación de contexto

4.8.4. Características y funciones del cliente

- **Menú**: El menú de la aplicación contiene un listado de botones que permiten navegar entre las varias vistas y filtros del cliente. Al entrar en la aplicación, la vista de `Bandeja de entrada` se muestra por defecto. El listado de botones comprende los elementos desde el mencionado previamente hasta `Proyectos`. Al final del menú se encuentran dos controles, uno que permite seleccionar el tema de la aplicación y otro control para cerrar sesión. Este menú se muestra en al figura 4.9.

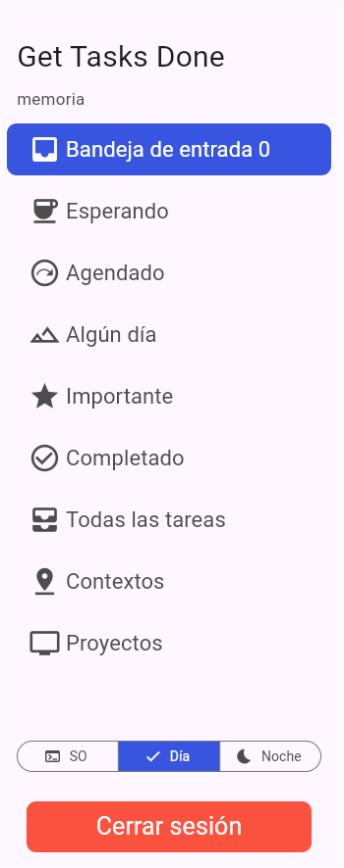

Figura 4.9: Menú del cliente web

- **Botón flotante de creación de tarea**: En la esquina inferior derecha, de forma persistente, se encuentra el botón de creación rápida de tarea. Al presionar sobre este botón se abrirá un formulario de creación de tarea reducido, como se muestra en la figura 4.10. Para crear una tarea con este formulario tan solo hay que rellenar el campo de `Nombre` y seleccionar un contexto con el control `Contexto`.

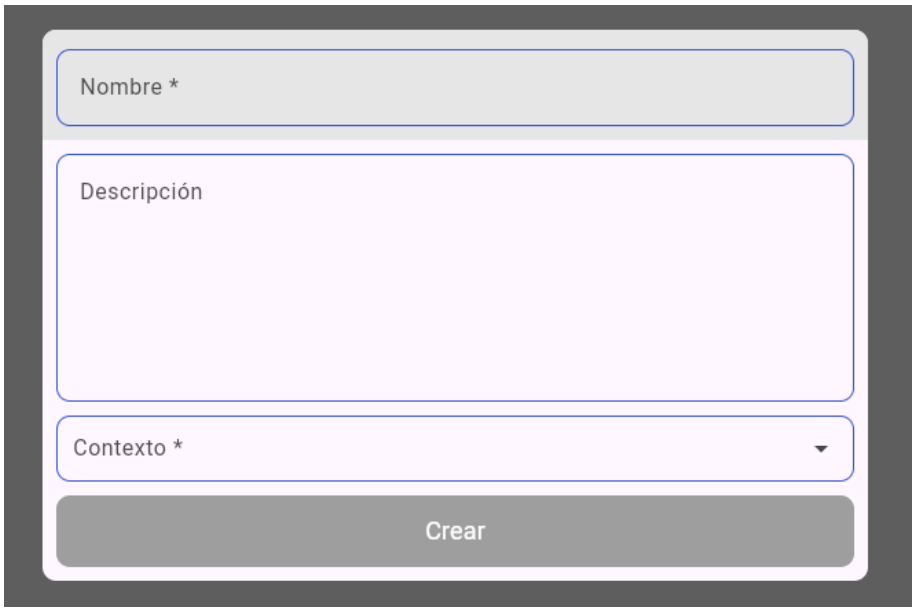

Figura 4.10: Formulario de creación rápida de tarea

- **Vista de bandeja de entrada**: En la bandeja de entrada se listan todas las tareas que se hayan creado a través del botón flotante, además de otras tareas que no se encuentren en alguno de los filtros o no tengan un proyecto asignado. Desde esta pantalla se pueden editar las tareas presionando sobre estas. También se pueden las completar tareas presionando sobre el círculo a la izquierda del nombre de la tarea. Finalmente, a través de gestos también se pueden completar o borrar tareas. Si se desliza una tarea hacia la izquierda, se completa. Si se desliza hacia la derecha, la tarea se borra. La vista de bandeja de entrada se muestra en la figura 4.1.
- **Formulario de edición de tarea**: Este formulario sirve para la edición (y borrado) de una tarea, además de ser el formulario de creación de tarea si se usa el botón `Agregar tarea` en alguno de los filtros. El formulario está compuesto por una serie de campos pertenecientes a la tarea, algunos de estos visibles desde el formulario de creación rápida, visibles en la figura 4.11.

Figura 4.11: Formulario de creación de tarea

- **Filtros de tareas**: Como descrito previamente, existen en el cliente una serie de filtros para el mostrado de tareas. La mayoría de estos filtros contienen un botón `Agregar tarea`, visible en la figura 4.12. Al presionar este botón, un formulario completo de creación de tarea se abrirá, con algunos de los campos ya rellenados. Por ejemplo, si se presiona el botón en el filtro `Importante`, el campo que marca una tarea como importante en el formulario ya estará marcado. A continuación se listan cada uno de los filtros:
  - *Esperando*: Se muestran las tareas que tienen en el campo `Estado` el estado `esperando`.
  - *Agendado*: Se muestran las tareas que tienen asginada una fecha en el campo de `Fecha de expiración`.
  - *Algún día*: Se muestran las tareas que tienen en el campo `Estado` el estado `algún día`.
  - *Importante*: Se muestran las tareas que tienen marcado el campo `Importante`, descrito en el ejemplo anterior.
  - *Completado*: Se muestran las tareas que tienen marcado el control a la izquierda del nombre de la tarea en la lista, es decir, si están marcadas como completa. En este filtro no hay botón `Agregar tarea`.

- *Todas las tareas*: Se muestran todas las tareas, independientemente de cualquier característica de la tarea. En este filtro no hay botón `Agregar tarea`.

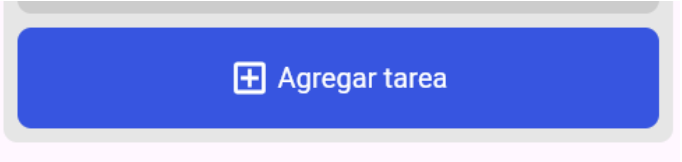

Figura 4.12: Botón agregar tarea dispobible en los filtros

- **Vista de contextos**: En esta vista se muestran todos los contextos del usuario, visible en la figura 4.13. La funcionalidad es similar a la de la vista de bandeja de entrada. A diferencia de esa vista, los contextos no tienen un control para completarlos. En relación a los gestos, existe el deslizamiento a la derecha de un contexto para borrarlo, pero solo se puede realizar el gesto si el contexto en cuéstion no está signado a ninguna tarea. Para la creación de contextos y edición de estos se usa un mismo formulario, mostrado en la figura 4.8. A la hora de borrar un contexto a través de este formulario, si el contexto tiene tareas asignadas, aparecerá un diálogo de confirmación con el listado de tareas que se eliminarían al borrar el contexto en cuestión, como se aprecia en la figura 4.14. Finalmente, al final de la lista hay un botón `Agregar contexto`, que permite la creación de contextos.

Figura 4.13: Vista contextos del cliente web

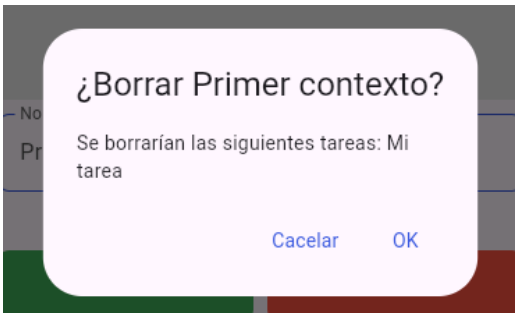

Figura 4.14: Mensaje de confirmación de borrado de contexto

- **Vista de proyectos**: Esta vista muestra los proyectos del usuario, representada en al figura 4.15. Cada elemento de la lista representa un proyecto, con una serie de controles asociado a este. Si se presiona sobre el elemento de la lista, el proyecto despliega las tareas asociadas a este. Este listado de tareas tiene la misma funcionalidad que el de las vistas de tareas. Si se presiona el botón con icono de lápiz, aparecerá el formulario de edición de proyecto, que se muestra en la figura 4.16. Finalmente si se presiona el botón con el símbolo de la suma, aparecerá un formulario de creación de tarea con el campo `Proyecto` teniendo seleccionado a este proyecto. Finalmente, al final de la lista hay un botón `Agregar proyecto`, que permite la creación de proyectos.

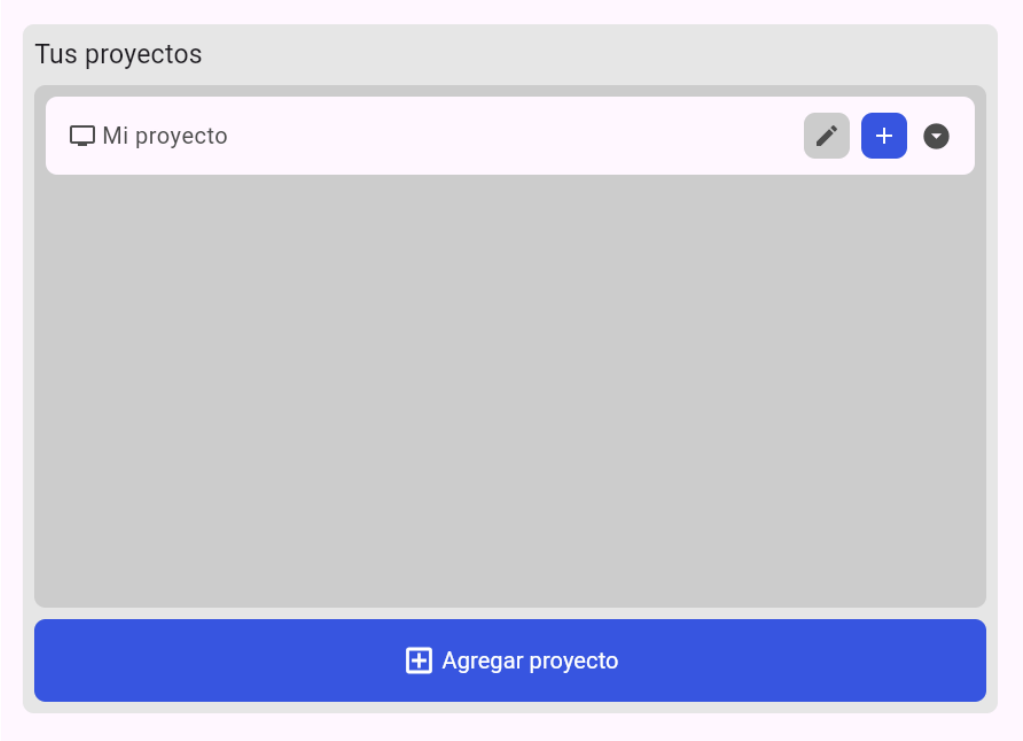

Figura 4.15: Vista proyectos del cliente web

Figura 4.16: Formulario de creación y edición de proyectos

Capítulo 5

Implementación de la aplicación móvil para Android

En este capítulo se presenta el funcionamiento y la implementación de la aplicación móvil para Android[13]. Cabe resaltar que esta aplicación se ha desarrollado de forma nativa para esta plataforma, con los beneficios y desventajas que ello conlleva. La aplicación se ha diseñado de tal forma que puede ser usada tanto en la pantalla de un móvil como de una tablet.

En la sección 5.1 se presentan las tecnologías que se han empleado. Posteriormente, en la sección 5.2 se explican las diferentes capas que componen la aplicación. Más adelante en la sección 5.3 se describe la parte gráfica de la aplicación. En la sección 5.4 hay un breve manual de usuario de la aplicación Finalmente, en la sección 5.5 se detalla cómo se puede obtener facilmente la aplicacion.

5.1. Tecnologías utilizadas

Para el desarrollo de esta aplicación Android nativa, hemos optado por Kotlin como nuestro lenguaje principal de programación. Kotlin, ampliamente respaldado por Google, ofrece una sintaxis moderna y concisa que mejora la productividad del desarrollador y reduce la probabilidad de errores. Al adoptar Kotlin, hemos aprovechado las características de seguridad de tipos, interoperabilidad sin problemas con el código Java existente y la capacidad de escribir menos código con una mayor expresividad.

Además, para aprovechar al máximo el ecosistema de desarrollo de Android, hemos empleado Android Studio como nuestro entorno integrado de desarrollo (IDE). Android Studio proporciona herramientas poderosas y eficientes que facilitan la creación, la depuración y la optimización de aplicaciones Android. Con funcionalidades como la vista previa en tiempo real, el análisis de rendimiento y las sugerencias de corrección de errores, Android Studio ha sido fundamental para nuestro flujo de trabajo de desarrollo.

En cuanto a la gestión de dependencias y la automatización de la compilación del proyecto, hemos confiado en Gradle. Gradle es una herramienta de construcción

avanzada que nos permite definir y gestionar eficazmente las dependencias del proyecto, compilar el código fuente, ejecutar pruebas automatizadas y generar artefactos distribuibles. Gracias a la flexibilidad y la potencia de Gradle, hemos podido personalizar nuestro proceso de construcción según las necesidades específicas de nuestro proyecto, lo que ha contribuido a una mayor eficiencia y calidad en el desarrollo.

En resumen, la combinación de Kotlin, Android Studio y Gradle ha sido fundamental para el éxito de nuestro proyecto de desarrollo de aplicaciones Android, permitiéndonos crear una aplicación robusta, eficiente y altamente funcional que cumple con los estándares más exigentes de calidad y rendimiento.

5.2. Arquitectura de la aplicación

En la figura 5.1 se proporciona una visión global de la arquitectura de la aplicación.

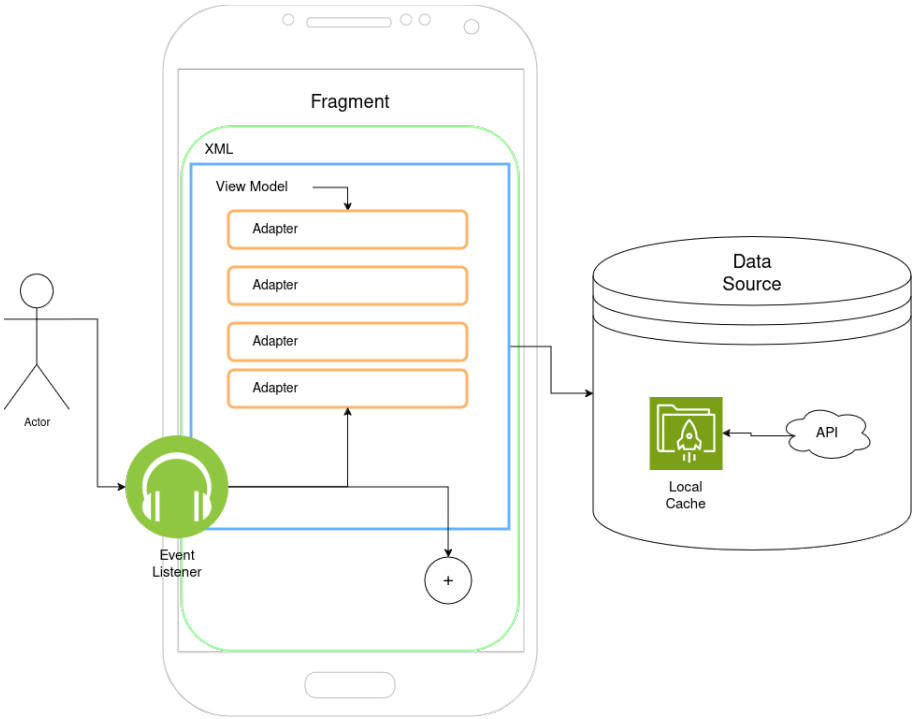

Figura 5.1: Arquitectura de la aplicación Android

Como se puede ver, esta aplicación tiene varios componentes bien diferenciados, que se describen en las subsecciones posteriores.

5.2.1. Vistas

Definen dónde y cómo aparecen en la pantalla los diferentes elementos visuales con los que el usuario va a interactuar, por ejemplo los botones, textos, formularios, imagenes, etc. Se implementan mediante archivos XML, y se pueden encontrar dentro del proyecto en la carpeta: `app/src/main/res`. Por ejemplo, en el archivo:

`activity_anadir_task.xml` se puede ver cómo se construye una pantalla que contiene un formulario para crear tareas. Se puede ver un ejemplo de la definición de una vista en la figura 5.2.

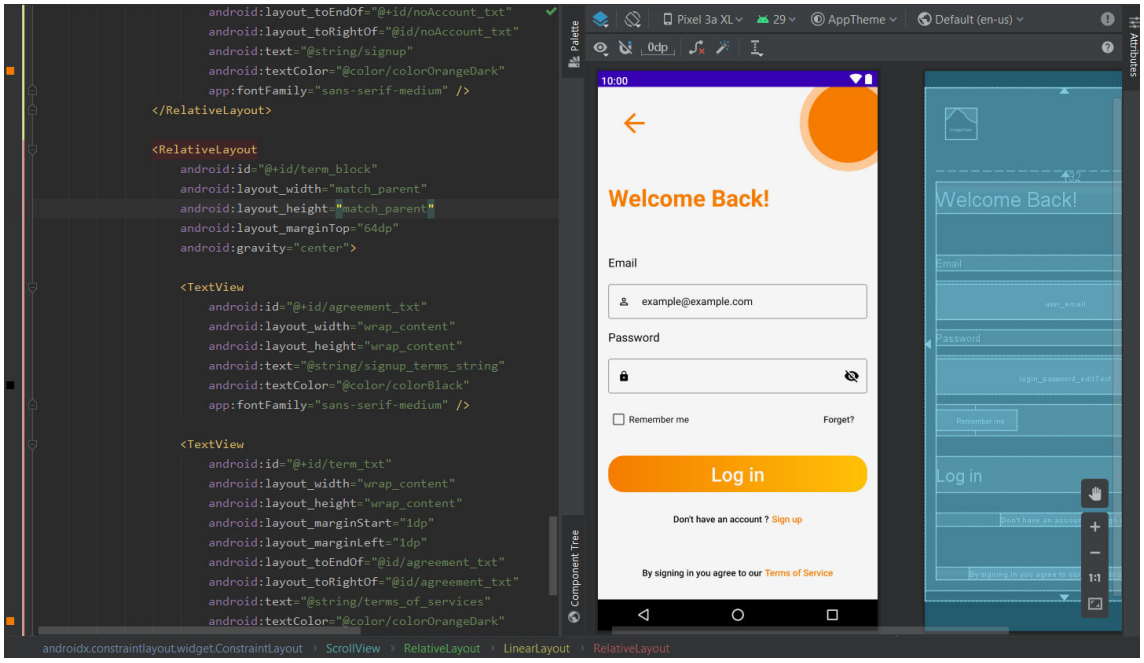

Figura 5.2: Ejemplo de diseño XML

5.2.2. Controladores

Definen qué ocurre cuando el usuario interactúa con la aplicación y se encarga de rellenar las vistas con la información que proviene de la capa de datos.

Los controladores de la aplicación se implementan en Kotlin, y se guardan en el directorio: `app/src/main/java/com/gettasksdone/gettasksdone`.

Siguen la estructura típica de Android de Actividades, Fragmentos, View Models y Adapters.

- Actividades: Son los componentes fundamentales que manejan la interfaz de usuario de la aplicación[18]. Se puede encontrar un ejemplo de actividad en el archivo: `./MainActivity.kt`.

- Fragmentos: Son una parte de la actividad. Puede haber varios fragmentos dentro de la misma actividad, representando multiples pantallas. Dentro de los fragmentos se pueden emplear varias funciones que escuchan eventos, por ejemplo dentro del archivo: `./ui/inBox/InBoxFragment.kt` se puede ver que hay un método que escucha la creación de la vista y se encarga de invocar al Adapter y al ViewModel para obtener la lista de tareas, entre otras cosas [17].

- View Model: Este componente sirve para obtener los datos y facilitarselos al fragmento cuando los necesite, por ejemplo obteniendo la lista de tareas, dentro

del ciclo de vida del fragmento [12]. Se puede encontrar un ejemplo de View Model en el archivo: `./ui/inBox/InBoxViewModel.kt`.

- Adapter: Provee acceso a elementos de datos con interfaz personalizada[9]. En el caso de este proyecto, se ha empleado por ejemplo para modificar la forma en que se ve cada elemento en la lista de tareas, como se se puede ver en este archivo: `./ui/inBox/TaskAdapter.kt`.

5.2.3. Datos

Teniendo en cuenta que los teléfonos en ocasiones son utilizados sin conexión a internet, se ha seguido un esquema avanzado para la capa de datos que permite utilizar un modo sin conexión dentro de la aplicación.

En la figura 5.3 se muestra cómo funciona la estrategia de caché, ejemplificando el caso de las tareas, pero es similar para el resto de entidades.

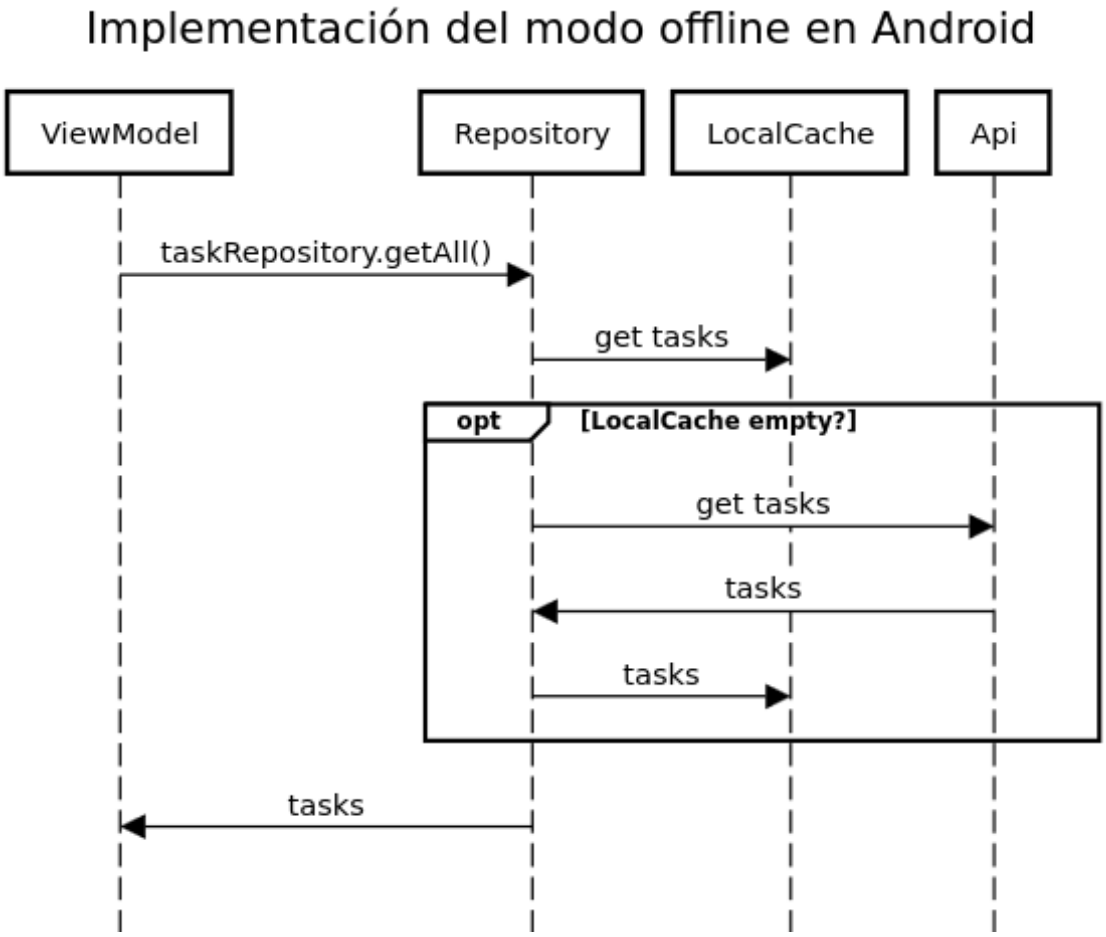

Figura 5.3: Modo offline Android

Como hemos discutido anteriormente, el View Model es el componente encargado de obtener los datos y entregarlos al fragmento, por lo que es este componente el que inicia una llamada hacia el repositorio pidiendole un recurso, en este caso las tareas.

Ahora, el repositorio va a manejar diferentes casuisticas, dependiendo de si los datos se encuentran en local o no.

En primer lugar, y con la filosofía *offline first*, va a consultar la cache local. Solo en el caso de que esta esté vacía, hace una llamada a la API para lo que tiene que salir a la red.

Como se puede ver la estrategia es muy sencilla, de esta forma, se guardan los datos en el dispositivo cuando hay conexión, pero cuando esta se pierde, se pueden seguir leyendo e incluso manipulando. Para ello se ha utilizado la librería Room, que permite hacer uso de una base de datos en local con alto rendimiento y un esquema eficiente que garantiza la consistencia de los datos.

5.3. Diseño de la aplicación

Para diseñar la parte gráfica de la aplicación, se toma en cuenta el diseño inicial planteado en el capítulo 3 y fue utilizado como guía, pudiendo hacer algunas modificaciones cuando era necesario por limitaciones de la tecnología o por requisitos funcionales.

Algunos elementos del diseño provienen de Android de forma nativa. Android nos provee una serie de elementos gráficos con ciertos estilos predeterminados, por ejemplo podemos usar las cajas de texto que vienen por defecto dentro de nuestros formularios, ahorrando tiempo de desarrollo con un buen acabado. Igualmente pasa con otros elementos como los popup, que no tenemos que diseñar desde cero, por que ya vienen con estilos predeterminados que encajan bien con nuestra aplicación.

Otros elementos como el menú o la lista de tareas han sido completamente personalizados para cumplir los requisitos de diseño y funcionales.

En general se hace uso de diferentes elementos y herramientas de Android, como pueden ser el sistema de gestos o las ventanas emergentes, que permiten construir una aplicación intuitiva y fácil de usar.

5.4. Manual de usuario

A continuación, se explicará cómo utilizar la app de la manera más eficiente posible para tener una experiencia completa y satisfactoria de esta app. Esta aplicación permite cambiar el tema con el visualizas la app, por lo que en el manual se mostrarán imágenes de ambos temas, para hacer más fácil e intuitiva la vista del usuario.

5.4.1. Pantalla principal

Esta es la pantalla principal de la aplicación, esta será la primera pantalla que aparezca una vez se inicie la aplicación y como detalle añadido se incluye un botón con el que se puede cambiar la temática con la que el usuario visualizará la app, como se puede observar en las figuras 5.4 y 5.5.

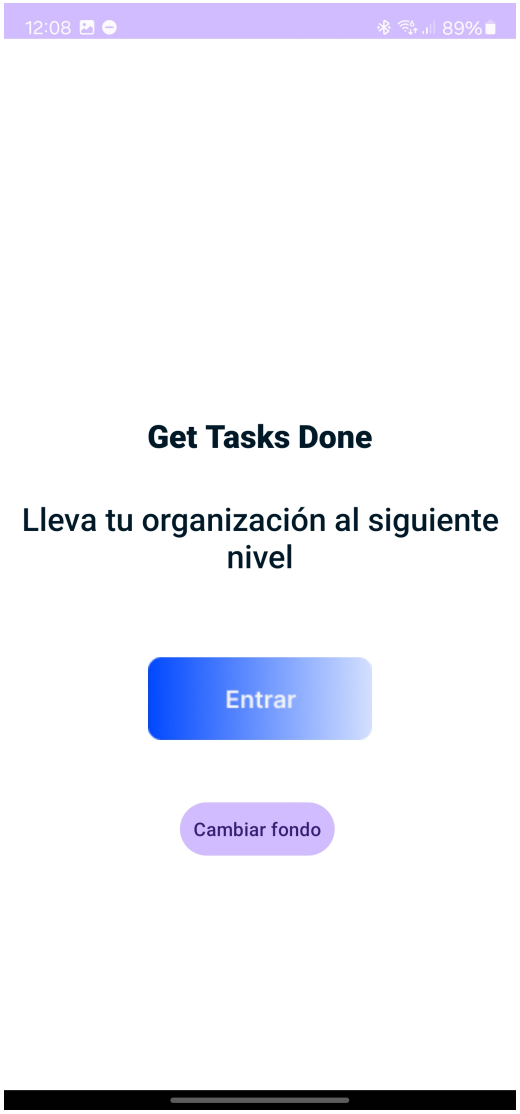

Figura 5.4: Pantalla principal en tema claro

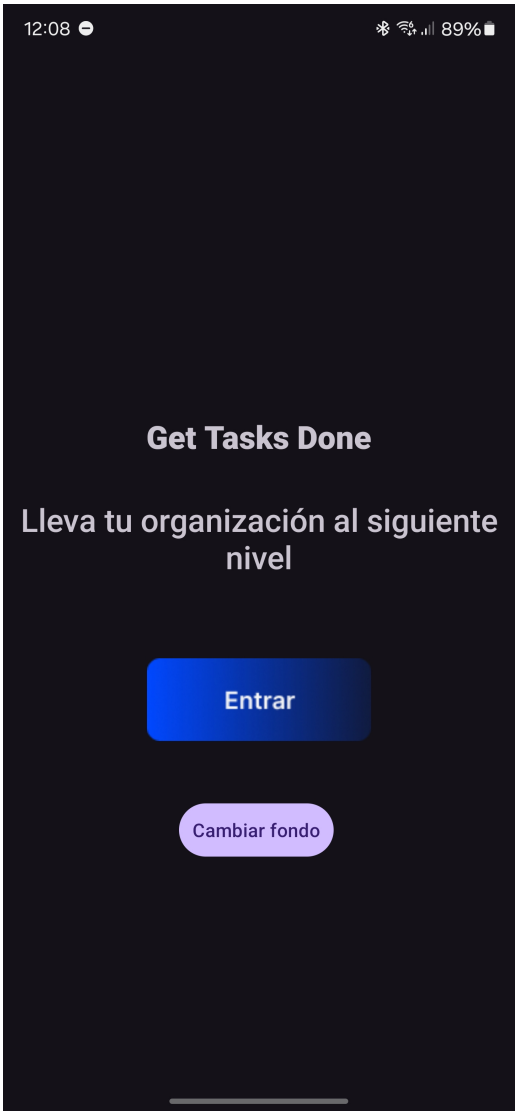

Figura 5.5: Pantalla principal en tema oscuro

5.4.2. Pantalla de inicio de sesión

Esta es la pantalla siguiente a la pantalla principal a la que el usuario accederá después de pulsar el botón “entrar” en la pantalla anterior. En esta pantalla aparece un formulario para introducir los datos de inicio de sesión, nombre de usuario y contraseña. Además, aparece un selector desplegable para seleccionar el servidor al que se desea que se conecte la aplicación. Por último, se presentan los botones de inicio de sesión y la posibilidad de acceder a la pantalla de registro si el usuario no dispone todavia de una cuenta. La apariencia de esta pantalla se puede ver en las figuras 5.6 y 5.7 respectivamente.

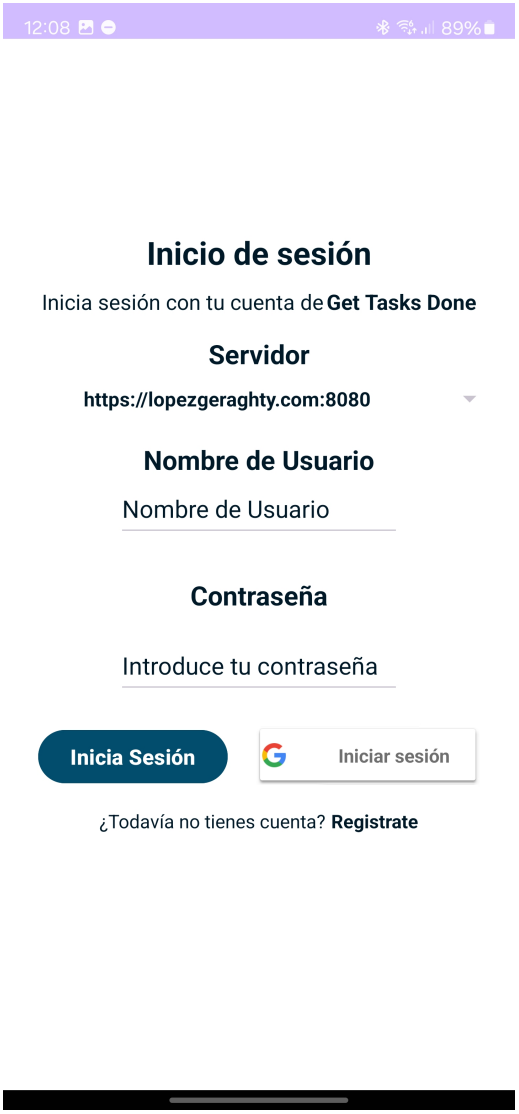

Figura 5.6: Pantalla de inicio de sesión en tema claro

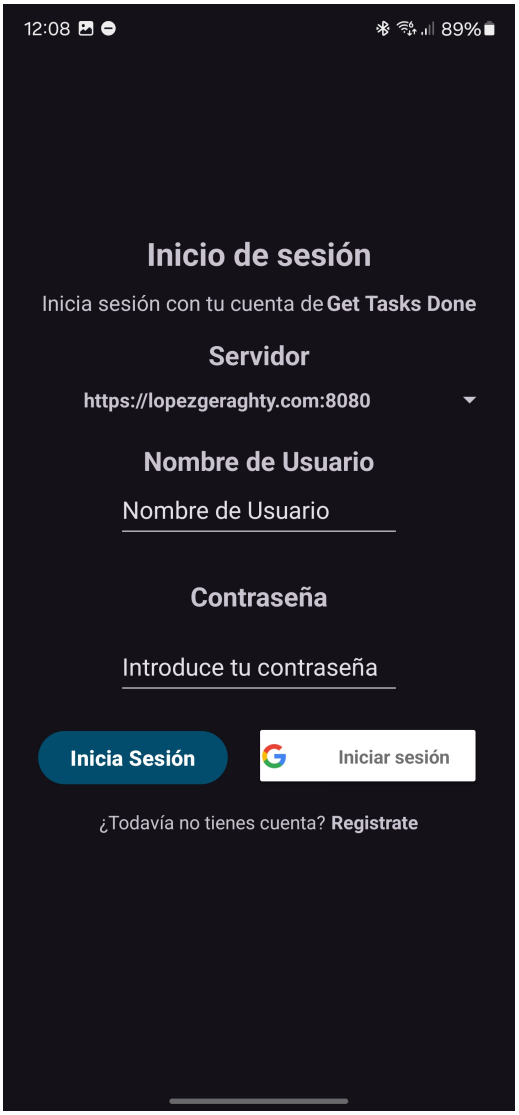

Figura 5.7: Pantalla de inicio de sesión en tema oscuro

5.4.3. Pantalla de registro

Esta es la pantalla donde el usuario podrá introducir los datos que la organización necesita para dar de alta a un nuevo usuario, que son respectivamente, nombre de usuario, contraseña y email, además se podrá seleccionar el servidor igual que en la pantalla anterior y una vez creada la cuenta se iniciará sesión automáticamente. Estas pantallas se pueden observar en las figuras 5.8 y 5.9.

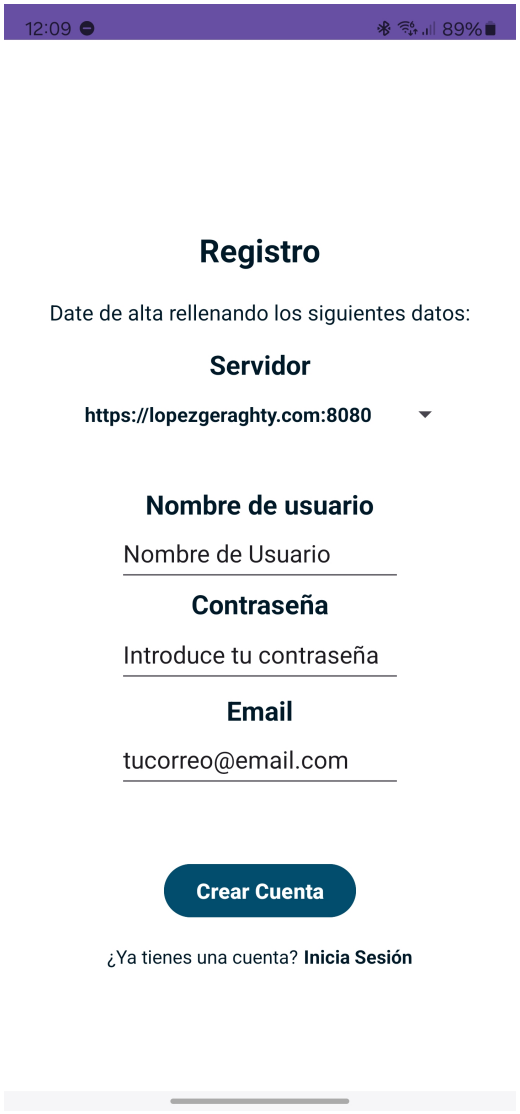

Figura 5.8: Pantalla de registro en tema claro

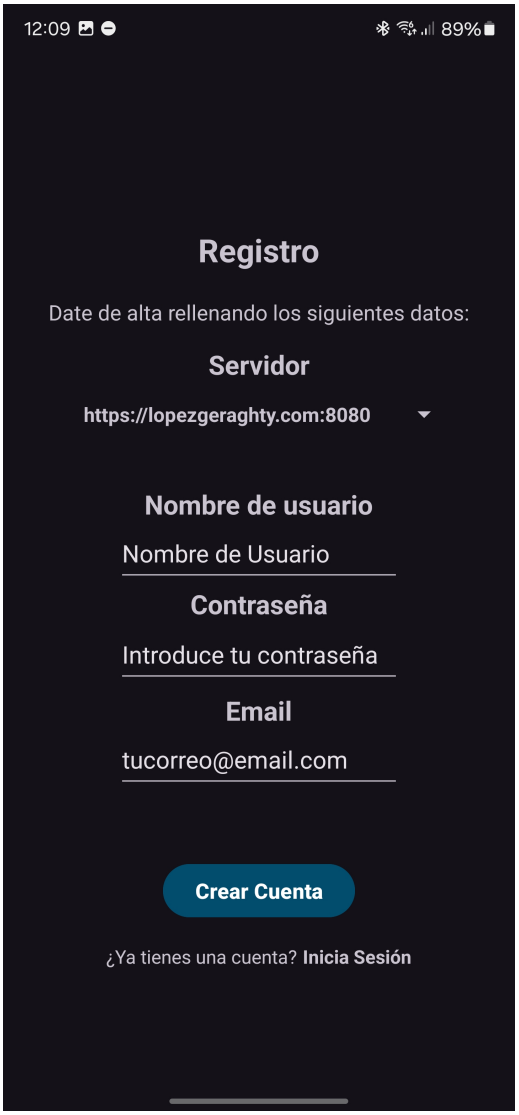

Figura 5.9: Pantalla de registro en tema oscuro

5.4.4. Pantalla de Completar Registro

Esta es la pantalla que le aparecerá al usuario tras registrarse en la aplicación por primera vez para completar sus datos personales, los cuales son, nombre, teléfono, puesto dentro de la organización y el departamento al que pertenece el usuario dentro de la organización. Su aspecto se puede ver en las figuras 5.10 y 5.11.

Figura 5.10: Pantalla de completado de registro en tema claro

Figura 5.11: Pantalla de completado de registro en tema oscuro

5.4.5. Pantalla de Menú

Esta es la pantalla principal con la que se encontrará el usuario una vez inicie sesión en la aplicación. El fragmente que aparezca en la pantalla principal y que por tanto el usuario visualizará dependerá de la selección que el usuario realice. El usuario puede elegir entre ver una de las siguientes pantallas, Bandeja de entrada, Proyectos, Esperando, Agendado, Algún Día y Completadas. Las tareas y proyectos que aparezcan en cada uno de estos fragmentos dependerán de como el usuario quiera organizar sus tareas y proyectos a la hora de crearlos. También en la parte inferior el usuario puede acceder a la lista de contextos que aparecen, puede también visualizar sus datos y existen las herramientas de cambiar el tema de la aplicación y de cerrar sesión por si el usuario quiere desconectarse y volver a la primera pantalla. Esta pantalla de menú se puede ver en las figuras 5.12 y 5.13.

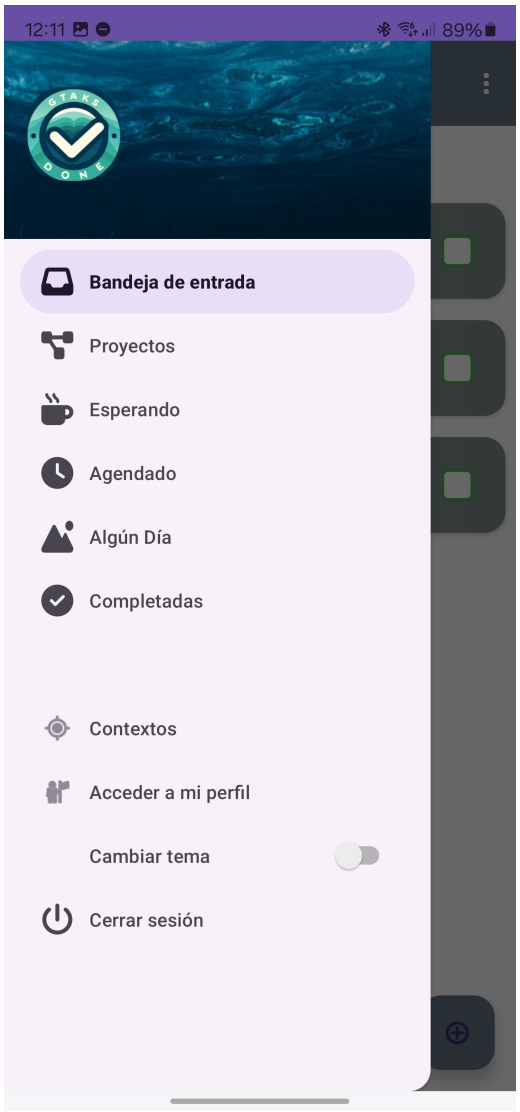

Figura 5.12: Pantalla de menú en tema claro

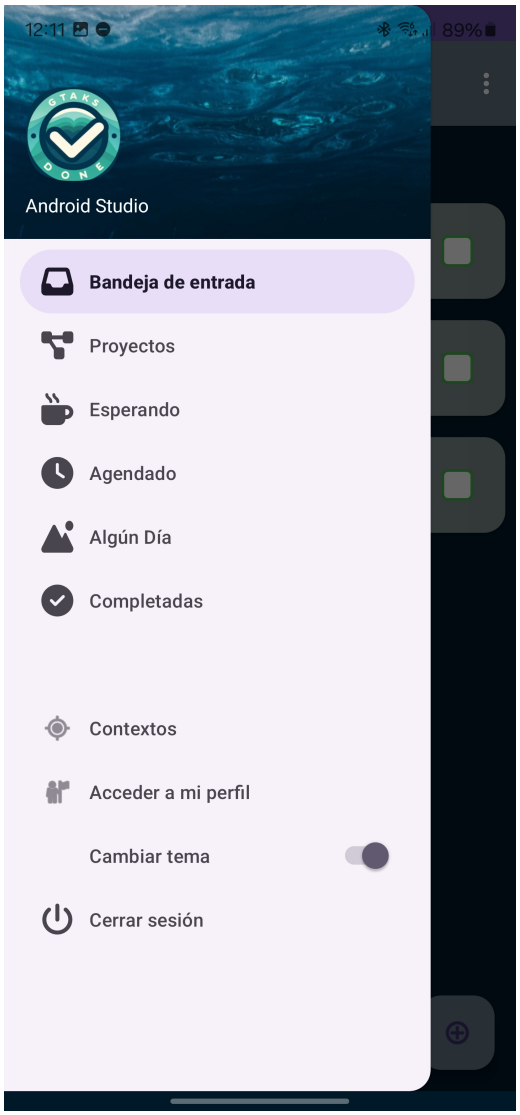

Figura 5.13: Pantalla de menú en tema oscuro

5.4.6. Fragmentos del Menú

Esta es la visión que tendrá el usuario de los distintos fragmentos a los que tiene visión el usuario y mencionados anteriormente, es un ejemplo de la "Bandeja de Entrada" pero es extrapolable al resto de fragmentos, salvo que en el fragmento de proyectos en vez de las tareas aparecerán todos los proyectos que tenga creados el usuario. En la esquina inferior derecha aparece un botón flotante que al pulsarlo dará la opción de elegir entre crear una tarea o crear un proyecto. Cuando el usuario quiera tener información sobre una tarea solamente tendrá que pinchar sobre ella y aparecerá una pantalla con toda la información de la tarea para verla incluso poder modificarla. Si el usuario desea marcar como completada la tarea tiene un botón en la parte derecha del item de la tarea para marcarla y si desea eliminarla, simplemente tendrá que deslizar la tarea hacia la izquierda.

Para el fragmento de los proyectos al pinchar sobre un proyecto aparece la lista de tareas que compone ese proyecto de manera similar a lo explicado anteriormente.

Esta pantalla de fragmentos se puede ver en las figuras 5.14 y 5.15 respectivamente.

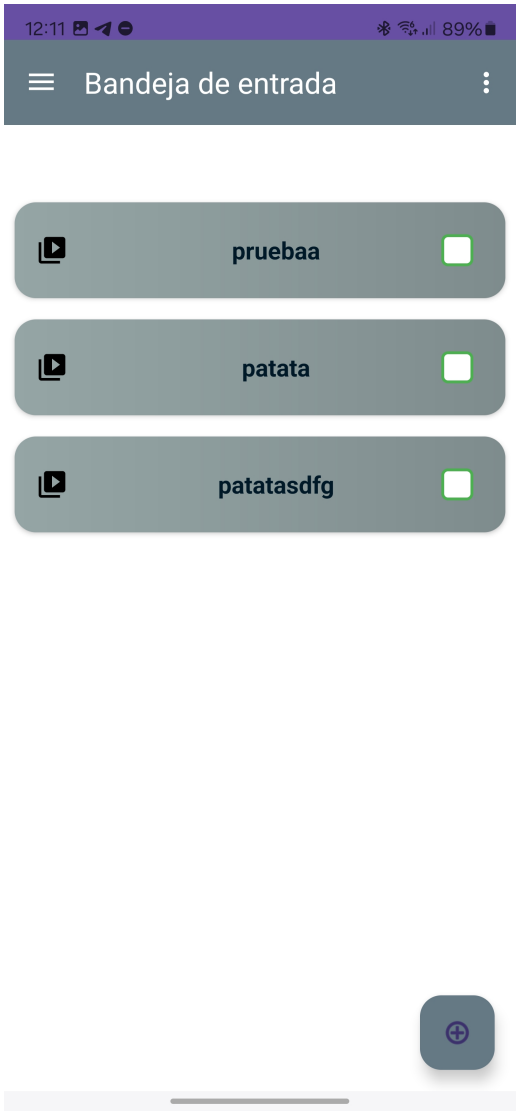

Figura 5.14: Pantalla de tareas en tema claro

Figura 5.15: Pantalla de tareas en tema oscuro

5.4.7. Pantalla de Crear Tarea

Esta pantalla muestra al usuario un formulario mediante el cual esta podrá crear una nueva tarea. Los campos que aparecen en el formulario para crear la nueva tarea son una estrella en la parte superior de la pantalla para que el usuario determine si su tarea sea prioritaria o no, un campo para introducir la fecha de realización de la tarea, el nombre de la tarea y su descripción, además de tres desplegables que son Estado, Contexto y Proyecto para seleccionar entre la lista de opciones que aparecerán.

Este formulario de creación de tareas se puede visualizar en las figuras 5.16 y 5.17.

Figura 5.16: Pantalla de creación de tareas en tema claro

Figura 5.17: Pantalla de creación de tareas en tema oscuro

5.4.8. Pantalla de Crear Proyecto

Esta pantalla es muy similar a la anterior, solo cambian los campos necesarios para crear un nuevo proyecto, los cuales son, el nombre del proyecto, dos campos de fecha para introducir el inicio y el fin del proyecto y un campo para introducir la descripción de este. Además, aparece un desplegable para seleccionar el estado del proyecto.

Este formulario se puede visualizar en las figuras 5.18 y 5.19.

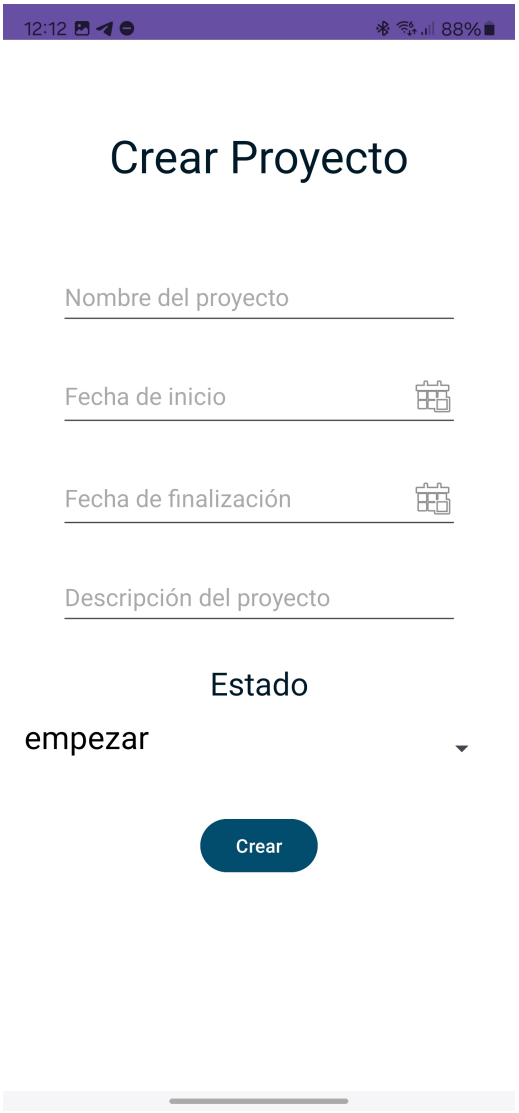

Figura 5.18: Pantalla de creación de proyectos en tema claro

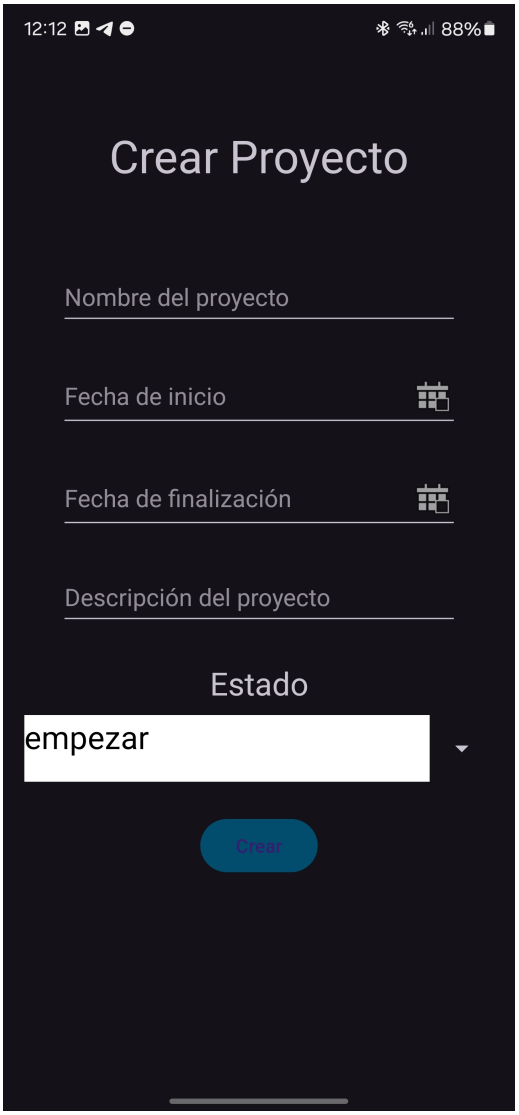

Figura 5.19: Pantalla de creación de proyectos en tema oscuro

5.5. Distribución de la aplicación

Obtener la aplicación Android e instalarla en un dispositivo es algo muy sencillo, ya que en cualquier dispositivo Android se puede instalar un APK en cuestión de segundos.

Un APK (Android Installation Package), no es más que un paquete que contiene los datos y archivos que componen la aplicación que se ha desarrollado. Los teléfonos Android permiten instalar estos archivos de forma nativa, sencilla y rápida. Sólo debemos descargar el archivo APK y abrirlo en nuestro teléfono.

Para que nos hagamos una idea, en la figura 5.20 se muestra la estructura interna de un APK.

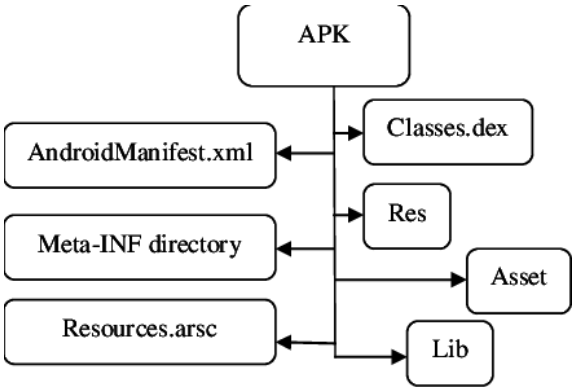

Figura 5.20: Diagrama de un APK

Un archivo clave dentro de estos es `AndroidManifest.xml`, que contiene mucha información sobre la aplicación como los permisos que va a requerir y la versión.

Además, la aplicación se puede distribuir a través de tiendas de aplicaciones como la Play Store de Google, aunque esto queda sujeto a las restricciones y condiciones de la tienda de aplicaciones. En el caso del Play Store de Google usan un formato especial algo distinto a un APK, con el que hacen un preprocesamiento y permiten que el usuario final se descargue una versión optimizada para su tipo de dispositivo. Nosotros no hemos explorado la posibilidad de publicar la aplicación en una de estas tiendas.

Debido a la facilidad de distribución mediante un archivo APK, hemos optado por esta opción. Como se menciona anteriormente, subir la aplicación a una tienda es un sencillo pero requiere burocracia y puede implicar pagos

Capítulo 6

Conclusiones y Trabajo Futuro

En conclusión, se ha desarrollado la infraestructura REST para GTD detallada a lo largo de esta memoria, con la finalidad de que las empresas puedan tener el control absoluto de su información al usar este sistema y desplegando la infraestructura en sus propios servidores. Este sistema muestra una importante mejora en privacidad y seguridad a comparación de utilizar una aplicación de productividad desarrollada y mantenida por empresas terceras, las cuales podrían tener acceso a información sensible de la empresa, y correr mayor riesgo de sufrir una fuga de información. A su vez, la implementación de la API Spring Boot ha permitido una correcta interacción con las aplicaciones web basada en Flutter y móvil para Android, y con el uso de tokens de sesión JWT y comunicación con HTTPS se garantiza la seguridad del sistema. Además, las aplicaciones web y móvil se han diseñado para ofrecer una experiencia intuitiva y fácil de entender para los usuarios finales, y siguiendo la filosofía GTD.

En el proceso se ha podido ver de primera mano la importancia de una buena coordinación de cara a implementar una aplicación servidor y sus aplicaciones cliente, dado que es fundamental que estos dos componentes coexistan correctamente desde las etapas más tempranas del proyecto. Así la comunicación entre estas dos partes ha de estar en perfectas condiciones, para agilizar la etapa de desarrollo.

También se ha podido contemplar el grado de dificultad que implica construir una aplicación cliente-servidor desde cero con un enfoque aplicado a la vida real, pues se deben tener en cuenta bastantes variables, como la seguridad de la infraestructura, la facilidad de instalación y despliegue, las limitaciones que podrían presentarse en la infraestructura donde se despliegue el sistema, entre otras cosas.

A su vez, se ha experimentado cierta dificultad en el desarrollo de la aplicación Android nativa por varios motivos. Uno de ellos ha sido la elevada curva de aprendizaje de Kotlin al tener un acceso limitado a documentación actualizada y completa.

Como trabajo futuro, se plantea la implementación de las siguientes características adicionales:

- Ampliar la funcionalidad del Single Sign-On (SSO) para permitir el uso de otras plataformas, como GitHub o Microsoft.

- Implementar ventanas en las aplicaciones cliente que permitan utilizar las funcionalidades administrativas disponibles en el backend.
- Desarrollar un sistema de instalación asistida para el backend, que permita, entre otras cosas, definir la ruta donde está almacenado el certificado SSL/TLS, las credenciales de acceso para el uso del SSO, o especificar qué sistemas de SSO se desean utilizar.
- Extender la funcionalidad sin conexión de la aplicación móvil, para permitir poner en cola operaciones de escritura en la aplicación mientras se está sin conexión a internet.
- Mejorar la apariencia visual de la aplicación móvil.
- Implementar en las aplicaciones cliente la funcionalidad relacionada con las etiquetas, notas y elementos de comprobación.

Introduction

6.1. Motivation

The GTD philosophy, developed by David Allen, was planned as a system to manage and organize the personal and professional life, improving the productivity of the people who apply this methodology[1]. This methodology consist on 5 fundamental steps:

1. Capture: Write or record everything that requires our attention into one tool.

2. Clarify: Define if that task is actionable right now. If so, we must decide the next task and project. Otherwise, we have to decide if it's irrelevant, a reference or something to put on hold.

3. Organize: Create reminders of the categorized tasks in appropiate places.

4. Reflect: Update and review all necessary on the system to recover concentration and focus.

5. Engage: Use the system to make the tasks with confidence and clarity.

For the application of this methodology there are different programs that work strictly following the GTD philosophy, and others which approach does not follow this philosophy in such a purist way (Trello, Evernote, Notion, etc.). Most of these applications allow storing information in the cloud, which makes easier its access from any device. However, most of these applications are proprietary and store the users' information on the providers' servers, so the owners of these applications have access to the users' information, negatively affecting their privacy. Also, by storing this information in servers out of the company control, there could be a risk that the confidenciality of the information stored could be compromised in a cyber attack.

6.2. Objectives

In order to solve these problems, the development of a GTD application infrastructure is proposed, which is made up of these components:

- A database that will store the information needed by the members of the organization.
- A server that hosts a REST API based on Spring Boot, which makes possible to read and manipulate the information of the database.
- A web application developed using Flutter, that allows the interaction with the REST API and makes possible the execution of the GTD philosophy.
- A desktop application based on the web application, which adapts the web app view and provides an alternative access to the application, in addition to providing access without an internet connection.
- An Android native mobile application.

It is also sought that, due to the interested organizations will have to mount the infrastructure, the installation process must be easy. In order to accomplish that, every component that must be installed in the organization servers and will be executable through containers, so it's possible to launch the application with a simple script. Another important aspect is the security of the application, so these measures are proposed in order to ensure de maximum level of security:

- The communication with the server will be protected by using the HTTPS protocol in order to encrypt communications with the server and prevent information leaks.
- In order for system users to only make use of the information created by themselves, a filtering system by authorship has been implemented, so the access and/or modification of the information creatrun ed by another user is prevented, whether it is by accident or deliberately.

6.3. System architecture

The system will have the architecture defined in the figure 6.1. As you can see, the architecture will have the following components:

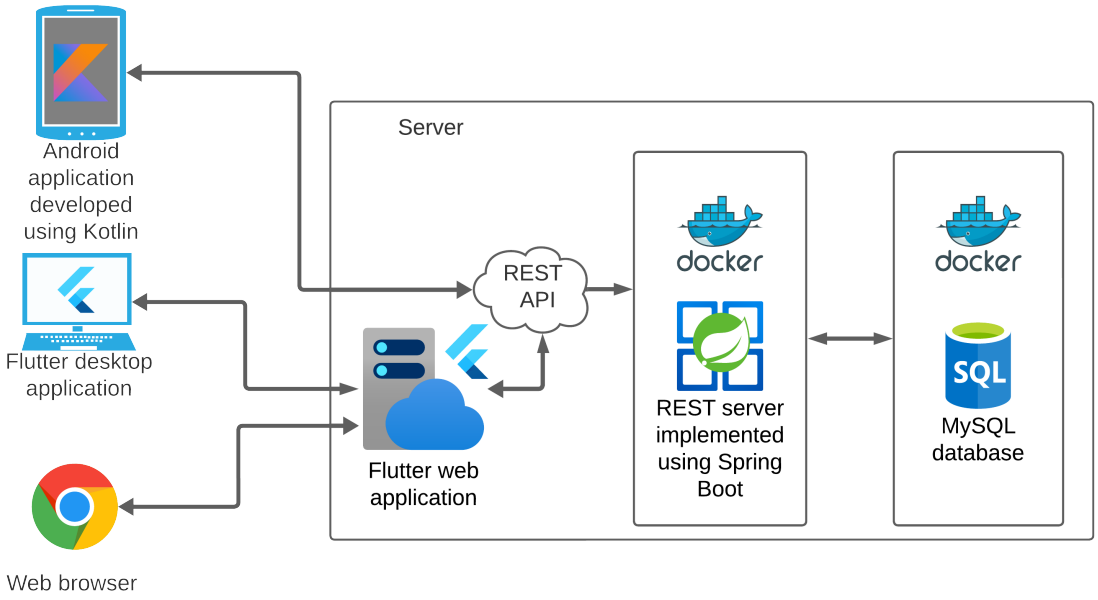

Figure 6.1: System architecture

- Android application developed using Kotlin: The mobile application will connect directly with the REST API in order to execute the operations requested by the user.

- Flutter desktop application: The desktop application will be deployed depending on the web application, and will interact with it to carry out the relevant operations.

- Flutter web application: The web application will provide an access to the API for the desktop application and the web browsers that uses it.

- REST server implemented using Spring Boot: This server implements the API calls and its business logic, communicating directly with the database in order to interact with the existing or new information.

- MySQL database: This SQL database management system is deployed independently of the server in order to distribute the workload appropiately, and is responsible for persisting all system information.

The client applications will connect to the internet so they can interact with the REST API, in which the server will execute the relevant requests to the database server, and will return the requested information or execute the operations requested by the client.

6.4. Work road

In order to achieve these objectives, we followed this work road:

T1. Design of the architecture: Define the relevant aspects for the functionality of the infrastructure, like the database structure.

T2. Previous study of the technologies: Make a previous research of how does the chosen technologies work, through reading documentation, watching videotutorials, and building prototypes related to the application to develop.

T3. Prototype of the user interface: Design the user interface with Figma[8], in it's desktop version and in it's mobile version.

T4. Server and database: Implement the database and server with Spring Boot, including basic functionality to the insertion, reading, modification and elimination of the different data types existing in the database, and also implementing the security measures considered necessary.

T5. Web application: Develop the multiplatform web application with Flutter, adding the necessary security measures, and every functionality that allows to implement the GTD philosophy, applying the necessary modifications to the server.

T6. Android application: Develop the Android application with Kotlin, adding a offline mode that will allow to access and modify the information of the users without internet access, and synchronize any possible modification once the internet access is recovered.

T7. Write the report: Write the final report of the project.

In the figures 6.2 and 6.3 the temporal planification during the years 2023 and 2024 is presented.

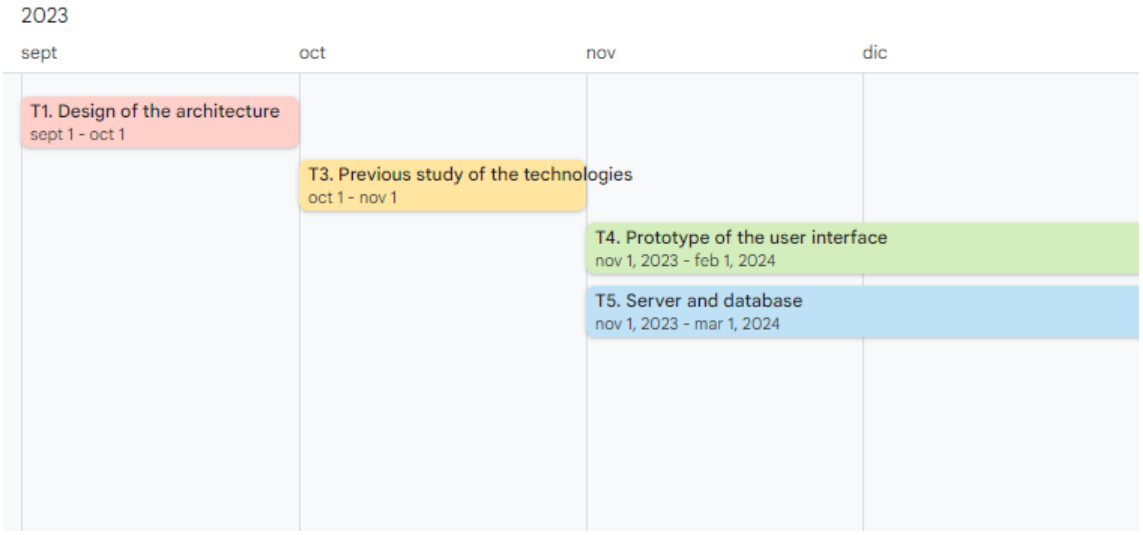

Figure 6.2: Gantt diagram 1

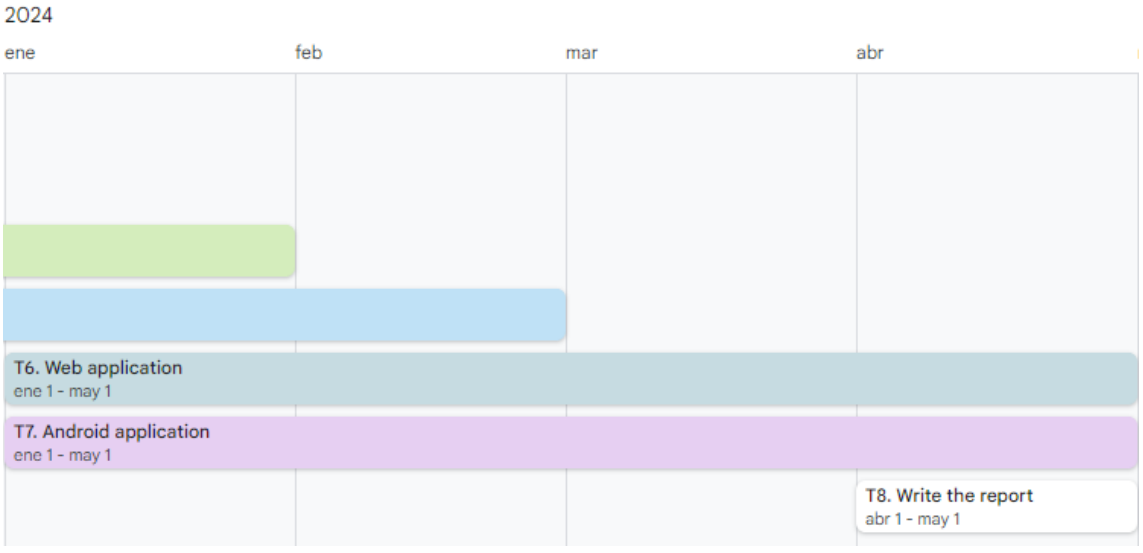

Figure 6.3: Gantt diagram 2

6.5. Report structure

The report of this project has been written defining the detailed structure below:

- In chapter 2 it is detailed how the Spring Boot based API is implemented, as well as certain design decisions such as the motivation for having selected each technology, the internal architecture of the server, the data model, the orchestration of the server and database containers, and the security measures implemented for the communications with the server.
- In chapter 3 the design of the user interface is detailed. A tour of the user interface screens is also carried out.
- In chapter 4 it is explained how the Flutter based web application was implemented, discussing the internal architecture of the application and its operation.
- In chapter 5 it is detailed all of the implementation and design decisions made related to the Android application developed using Kotlin, as well as a use guide for the application users.
- Chapter 6.5 recapitulates all the work carried out in the project, and possible improvements to be made in the future are proposed if the development of the project continues.

Additionally, a series of appendices have been provided, detailed below.

- Appendix A details the personal contributions to the project made for each one of the team members.
- In appendix B it is detailed with a high level of detail every available API call, how do they work, which information returns, etc.
- Appendix C provides a installation and deployment manual for the Spring Boot server.
- Appendix D contains a installation and deployment manual for the web application developed using Flutter.

Conclusions and Future Work

In conclussion, a REST infrastructure for GTD has been developed as described throughout this report, with the purpose that companies can have absolute control of their information when using this system and deploying the infrastructure on their own servers. This system shows an important improvement in privacy and security compared to using third-party applications, which could have access to sensitive information of the company, and run a greater risk of suffering information leaks. At the same time, the Spring Boot API implementation allowed a correct interaction with the website based on Flutter and with the Android application, and with the usage of JWT session tokens and communication with HTTPS, the security of the system is guaranteed. Also, the web and mobile applications have been designed to offer an intuitive and easy to understand experience for the final users, while following the GTD philosophy.

In the process it has been possible to see first-hand the importance of good coordination in order to implement a server application and its client applications, since it is essential that these two components coexist correctly from the earliest stage of the project. So the communication between these two components must be in perfect conditions, in order to speed up the development stage.

It has also been possible to contemplate the degree of difficulty involved in building a client-server application from scratch with an approach applied to real life, because many variables must be taken into account, like the security of the infrastructure, the ease of installation and deployment, the limitations that could arise where the system is deployed, among other things.

At the same time, some difficulty has been experienced during the development of the native Android application due to multiple reasons. One of them is the elevated learning curve of Kotlin due to the limited access of updated and complete documentation.

As future work, it is proposed the implementation of the following additional features:

- Extend the functionality of the Single Sign-On (SSO) access to allow the use of other platforms, like GitHub or Microsoft.

- Implement screens on the client applications that allow to use the additional administrative features available in the backend.

- Develop an assisted installation system for the backend that allows, among other things, to define the path where the SSL/TLS certificate is located, the access credentials for the SSO usage, or specify which SSO systems will be used.
- Extend the offline functionality of the mobile application, to allow enqueue writing operations in the application while internet access is not available.
- Improve the visual appearance of the mobile application.
- Implement on the client applications the features related to tags, notes and check items.

Bibliografía

[1] David Allen. *Organízate con eficacia: El arte de la productividad sin estrés*. Empresa Activa, 2017. ISBN: 9788417180577.

[2] Baeldung. *Build your API with Spring*. eBook, 2024.

[3] Baeldung. *Learn Spring Boot*. 2024. URL: https://www.baeldung.com/spring-boot.

[4] Oscar Blancarte. *Data Transfer Object (DTO) - Patron de diseño*. 2018. URL: https://www.oscarblancarteblog.com/2018/11/30/data-transfer-object-dto-patron-diseno/.

[5] dash-overflow.net. *Riverpod Framework*. 2022. URL: https://flutter.dev/.

[6] Neal K. Davis. *Stateful vs. Stateless Applications - Differences, Pros & Cons, Use Cases*. LinkedIn, 2023. URL: https://www.linkedin.com/pulse/stateful-vs-stateless-applications-differences-pros-cons-neal-davis-febge/.

[7] Docker. *Official Docker Documentation*. Docker, 2024. URL: https://docs.docker.com/.

[8] *Figma*. URL: https://www.figma.com/.

[9] Data Flair. *Android Adapters - The art of accessing the data items in Android*. 2020. URL: https://data-flair.training/blogs/android-adapters/.

[10] Flutter. *Flutter Framework*. Google, 2017. URL: https://flutter.dev/.

[11] flutter.dev. *go_router - Flutter package*. 2022. URL: https://pub.dev/packages/go_router.

[12] Lyla Fujiwara. *ViewModels : A Simple Example*. 2017. URL: https://medium.com/androiddevelopers/viewmodels-a-simple-example-ed5ac416317e.

[13] *GetTasksDone - Android Client*. GitHub Repository. 2024. URL: https://github.com/gettasksdone/GetTasksDone-android-app.

[14] *GetTasksDone - Server*. GitHub Repository. 2024. URL: https://github.com/gettasksdone/gettasksdone-server.

[15] *GetTasksDone - Web Client*. GitHub Repository. 2024. URL: https://github.com/gettasksdone/GetTasksDone-web-client.

[16] Google. *Cómo usar OAuth 2.0 para acceder a las API de Google*. Google, 2024. URL: https://developers.google.com/identity/protocols/oauth2?hl=es-419.

[17] javatpoint. *Android Fragments*. 2023. URL: https://www.javatpoint.com/android-fragments.

[18] Oğulcan Kirtay. *What is Activity in Android?* 2023. URL: https://medium.com/@ogulcankirtay/what-is-activity-in-android-25a55ed2fcba.

[19] Mozilla Developer Network. *OPTIONS*. 2024. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/OPTIONS.

[20] *Official Spring Boot Documentation*. 2024. URL: https://spring.io/guides.

[21] Okta. *JWT Documentation*. 2024. URL: https://jwt.io/introduction.

[22] steenbakker.dev. *flutter_secure_storage - Flutter package*. 2020. URL: https://pub.dev/packages/flutter_secure_storage.

[23] *Swagger Documentation*. 2024. URL: https://swagger.io/docs/specification/about/.

Apéndice A

Contribuciones Personales

Camilo Andres D'isidoro

En la asignación de carga de trabajo, Camilo se ha encargado principalmente del desarrollo e implementación del API REST en Spring Boot en los siguientes aspectos:

- Definición de las clases modelo que hacen de reflejo de la estructura de la base de datos.
- Implementación de los controladores donde se definen las llamadas al API y devuelven la información solicitada de la base de datos.
- Construcción de las clases repository, encargadas de comunicarse directamente con la base de datos y realizar las consultas pertinentes.
- Implementación de las clases service e implementation, encargadas de procesar la información recibida a través de las clases repository y filtrar con ayuda de las clases DTO la información que se desea devolver a los clientes.
- Creación de las clases DTO, que consisten en una versión limitada de las clases modelo con el fin de limitar la información que se le devuelve a los clientes al hacer operaciones de consulta.
- Configuración de los controladores para que los usuarios del sistema únicamente tengan permitido consultar y gestionar la información de la base de datos que ellos hayan creado a través de la asignación de dicha información a ese usuario en la base de datos.
- Colaboración con Jhimmy para la generación de tokens de sesión JWT a través del acceso a la aplicación por el sistema Single Sign-On (SSO) de Google. Esto se ha conseguido mediante el procesamiento del token que proporciona Google una vez finalizado correctamente el inicio de sesión para recoger el correo electrónico de la persona que ha iniciado sesión, y en caso de que ese email exista en la base de datos construir el token en base a la información

ya existente. En caso de que se tratase de un nuevo usuario, se le crearía un usuario nuevo y generaría un token con el usuario nuevo.

- Establecimiento de una comunicación segura con el API por HTTPS. Esto se consigue a través de añadir un certificado electrónico en formato .p12 que permite validar la autenticidad de la conexión.
- Coordinación con Matías para implementar una correcta interacción entre el cliente web y el servidor.

También se ha hecho cargo de la definición de la estructura de la base de datos en los siguientes aspectos:

- Definición de las tablas en las que se guardará la información.
- Implementación de las claves primarias y foráneas de cada tabla.
- Definición de las relaciones y restricciones de actualización o borrado en la base de datos en base a las claves foráneas.

A su vez, Camilo se ha encargado de construir la arquitectura Docker del servidor:

- Creación del fichero Dockerfile para el API REST para su contenerización.
- Definición de un fichero Docker Compose con el fin de definir el comportamiento interno del contenedor, como sus dependencias, puertos de escucha, variables de entorno, pruebas de salud de los contenedores, y rutas de los ficheros de configuración necesarios para la correcta instalación de los contenedores.
- Implementación de la imagen Docker mysql 8.0 para la gestión de la base de datos, y su correcta interacción con el API a través del fichero Docker Compose. Este contenedor al lanzarse por primera vez en un sistema importará el fichero de configuración DBStructure.sql definido en la carpeta database del servidor, el cual contiene toda la estructura y configuración de la base de datos.
- Implementación de una comprobación de estado para el contenedor de la base de datos que permitirá arrancar el contenedor del API REST únicamente cuando el contenedor MySQL haya terminado de arrancar. Esta comprobación de estado consiste en ejecutar el comando `mysqladmin ping -h localhost` en el contenedor de la base de datos para comprobar que el servicio se encuentra activo en intervalos de 1 minuto con 5 reintentos cada 30 segundos y un tiempo de expiración de 30 segundos para cada intento, con el fin de darle tiempo suficiente al servicio de ejecutarse incluso en máquinas con recursos limitados para poder lanzar el API.

Finalmente como parte de la creación de manuales de instalación y documentación de la aplicación, Camilo se ha encargado de:

- Documentar todas las posibles llamadas que puede recibir el API a través de la herramienta Swagger.

- Documentar los pasos para instalar y configurar correctamente el servidor en el manual de instalación.

El impacto del trabajo de Camilo con el servidor ha sido bastante importante, puesto que una correcta implementación del servidor y de la base de datos es fundamental para la posterior implementación y diseño de las aplicaciones cliente.

Jhimmy Ender Candela Espinosa

Durante el desarrollo del proyecto, Jhimmy ha tenido la oportunidad de colaborar estrechamente con sus compañeros, contribuyendo en varios aspectos cruciales del desarrollo, innovación y la implementación de la aplicación.

- Automatización y eficiencia con GitHub Actions: Se diseñaron flujos de trabajo automatizados para la Integración Continua y el Despliegue Continuo (CI/CD) principalmente en el repositorio del servidor, pero también se translado a la realización del build en la parte del cliente web con Flutter.Este enfoque permitiría detectar errores de forma temprana y mejorar la calidad del código antes de su despliegue final, asegurando una entrega más rápida y fiable del software.

- Innovación en la implementación de clases DTO: Junto a Camilo se concibió la idea y posteriormente implementación de las clases de Transferencia de Datos (DTOs). Esta tarea no solo implicó programación técnica, sino también una visión estratégica para modelar datos garantizando seguridad y privacidad, filtrando con precisión la información que fluye entre el servidor y los clientes. Esto permite garantizar que solo los datos necesarios y seguros sean transmitidos.

- Fortalecimiento de la seguridad con JWT: En colaboración estrecha con sus compañeros, Jhimmy ideó, desarrolló e implementó la autenticación basada en JSON Web Tokens (JWT). Este proceso es crucial para la seguridad del sistema, ya que los JWT permiten la creación de sesiones seguras y gestionables sin necesidad de almacenar el estado del usuario en el servidor, ofreciendo así una solución eficiente y escalable para manejar identidades en aplicaciones modernas.

    - **Colaboración de David para el Diseño y implementación:** David, con su experiencia previa en desarrollo de aplicaciones, aportó una perspectiva valiosa para la estructuración inicial del sistema de autenticación. Juntos, conceptualizaron cómo los JWT podrían integrarse eficazmente en la infraestructura existente, evaluando diferentes métodos de implementación y estableciendo un prototipo funcional que cumplía con los requisitos de seguridad y escalabilidad.
    - **Implementación y Configuración con Camilo:** Una vez definido el prototipo, Jhimmy trabajó mano a mano con Camilo para integrar esta

solución en el sistema de Single Sign-On (SSO) de Google. Su objetivo era asegurar una transición fluida y segura de las sesiones del usuario. De esta menera se gestionaría correctamente las identidades y las sesiones a través de múltiples aplicaciones y servicios. Este entorno de pruebas fue crucial para identificar y resolver cualquier problema de integración.

- Automatización y Eficiencia mediante Scripting:
  - **Desarrollo de scripts para la automatización de tareas:** Se desarrollaron varios scripts que han sido fundamentales para automatizar procesos repetitivos y laboriosos dentro del flujo de desarrollo. Estos scripts incluyeron la automatización de pruebas, la configuración de entornos de desarrollo, y la gestión de despliegues, lo que significativamente incrementó la eficiencia.
  - **Integración con herramientas:** Los scripts que Jhimmy diseñó fueron integrados con las herramientas de Integración Continua y Despliegue Continuo existentes, tales como GitHub Actions. Esto permitió ejecuciones de código más limpias y un ciclo de lanzamiento de producto más ágil y controlado. Los scripts garantizaron que cada build pasara por un conjunto estandarizado de pruebas y configuraciones antes de ser considerado para el paso a producción.

- **Contribuciones en el desarrollo de la aplicación cliente Android:** Junto a sus compañeros se implementaron diversos aspectos necesarios para la correcta funcionalidad de la aplicación.
  - Se profundizó en aspectos concretos como la interfaz de usuario.
  - La adaptabilidad a diferentes dispositivos. Él equipo empleó un enfoque riguroso para asegurar que la aplicación funcionara sin problemas en una amplia variedad de dispositivos Android, desde los más básicos hasta los de gama alta.
  - Se utilizó un diseño adaptable que se basa en unidades de medida flexibles y layouts que se expanden o contraen basándose en las dimensiones del dispositivo, asegurando así una presentación óptima en cada dispositivo.
  - Flexibilidad para Cambiar la URL a la que Apunta el Cliente Android: Para ofrecer mayor flexibilidad en la configuración de la aplicación y facilitar la integración con diferentes entornos de backend, se implementó una funcionalidad que permite a los usuarios o administradores cambiar la URL de conexión del cliente Android. Esto se realizó a través de un Spinner de configuración accesible dentro de la aplicación, donde se pueden ingresar y almacenar diferentes URLs. Este ajuste puede ser especialmente útil para pruebas, donde se requiere conectar con múltiples entornos de prueba antes de realizar el despliegue en producción. También permite a la aplicación adaptarse rápidamente a cambios en la infraestructura del servidor sin necesidad de actualizar la aplicación y redistribuirla.

Además, Jhimmy ha realizado un trabajo significativo en la realización de pruebas, la identificación y solución de errores, lo que ha mejorado la fiabilidad de los entornos de aplicación tanto en la parte de servidor como en el cliente android.

Víctor Gallego Yañez

A lo largo del desarrollo de este Trabajo de Fin de Grado, Víctor ha llevado a cabo una serie de tareas esenciales, p rincipalmente enfocadas en el desarrollo del frontend del proyecto GTD. Estas contribuciones personales se detallan a continuación:

- Colaboración en el Diseño Inicial: Víctor colaboró estrechamente con David en el diseño inicial de las vistas y mockups de la aplicación, tanto para el cliente web y de escritorio como para el cliente Android, utilizando la herramienta Figma, para la que hubo que hacer un trabajo de aprendizaje y conocimiento previo ya que era una herramienta desconocida.

- Aprendizaje de Kotlin y Desarrollo de Android: Víctor asumió un desafío significativo al embarcarse en el aprendizaje autodidacta del lenguaje de programación Kotlin y el desarrollo de aplicaciones Android. A pesar de que ni él ni ningún otro miembro del equipo tenían conocimientos previos en estas áreas, Víctor demostró una notable capacidad de adaptación y aprendizaje rápido. Se dedicó a buscar información relevante, estudiar tutoriales, practicar con ejemplos de código y experimentar con diferentes aspectos de Kotlin y Android. Su esfuerzo no sólo le permitió adquirir una nueva habilidad, sino que también sentó las bases para el desarrollo de la aplicación Android del proyecto. Gracias a su dedicación, el equipo pudo contar con una aplicación Android funcional y eficiente, que se integró perfectamente con el backend y ofreció una experiencia de usuario fluida y agradable.

- Montaje y Desarrollo de la Aplicación Android: Víctor se encargó del primer montaje y del desarrollo inicial de la aplicación Android, así como de la construcción del proyecto.

- Selección de Herramientas Auxiliares: Víctor realizó la selección de las herramientas auxiliares que se utilizarían a lo largo del proyecto. Esta tarea fue fundamental para alcanzar un alto nivel de comunicación entre el backend y el frontend por medio de la API.

- Diseño de Logos y Temáticas del Proyecto: Víctor diseñó los logos y las temáticas del proyecto que se utilizaron tanto en la aplicación Android como en el resto de los clientes, web y de escritorio.

- Desarrollo de la Estructura de la Aplicación: Víctor desarrolló la estructura de la aplicación, estableciendo las bases para su funcionamiento y crecimiento. Siendo esto importante para que otros integrantes del grupo pudieran entender de una manera más sencilla el desarrollo de la aplicación y el lenguaje utilizado para su posterior contribución.

- Desarrollo de Pantallas de la Aplicación: En colaboración con Camilo y David, Víctor desarrolló varias pantallas de la aplicación, contribuyendo a su funcionalidad y apariencia.
- Diseño de Nuevas Funcionalidades: Víctor diseñó nuevas funcionalidades que no estaban contempladas previamente en los objetivos principales del proyecto. Para ello, tuvo que realizar un trabajo de investigación para examinar su viabilidad en el proyecto.
- Realización de Pruebas: Víctor llevó a cabo una serie de pruebas para garantizar el correcto funcionamiento de la aplicación basándose en los conocimientos adquiridos durante la carrera en asignaturas como Testing de Software. Estas pruebas incluyeron pruebas unitarias, pruebas de integración y pruebas de usuario.
- Identificación y Solución de Errores: Víctor jugó un papel crucial en la identificación de errores en la aplicación. Una vez identificados, trabajó diligentemente para solucionar estos errores, mejorando así la estabilidad y la fiabilidad de la aplicación.
- Documentación de la Aplicación y Creación de Guías de Usuario: Víctor ha desempeñado un papel fundamental en la documentación de la aplicación y la creación de guías de usuario. Su trabajo en este aspecto ha sido esencial para garantizar que los usuarios puedan entender y utilizar la aplicación de manera efectiva. Víctor ha elaborado documentación detallada y fácil de entender, cubriendo todas las funcionalidades de la aplicación. Además, ha creado guías de usuario intuitivas y completas, proporcionando instrucciones claras y paso a paso para ayudar a los usuarios a navegar por la aplicación. Esta contribución ha mejorado la accesibilidad de la aplicación y ha facilitado su adopción por parte de los usuarios.

Estas contribuciones han sido vitales para el éxito del proyecto y demuestran el compromiso y la dedicación de Víctor en el desarrollo de software de alta calidad para el correcto desempeño del proyecto, consiguiendo que el resultado final, sea una aplicación que cumple con los criterios que ha de tener una buena aplicación móvil. Su colaboración en el diseño inicial y su aprendizaje autodidacta de Kotlin y desarrollo de Android han permitido que la aplicación tenga una usabilidad intuitiva y una integrabilidad eficiente con el backend. Su trabajo en el montaje y desarrollo de la aplicación Android, así como la selección de bibliotecas auxiliares, han asegurado un rendimiento óptimo de la aplicación y una compatibilidad amplia. Gracias al trabajo de otros compañeros creando un backend seguro se ha conseguido con la integración de este backend y utilizando las herramientas apropiadas para el frontend una aplicación segura para el usuario, además de tener una funcionalidad robusta y un diseño atractivo.

Además, Víctor ha realizado un trabajo significativo en la realización de pruebas, la identificación y solución de errores, lo que ha mejorado la fiabilidad de la aplicación. Su contribución a la personalización y accesibilidad de la aplicación ha mejorado la experiencia del usuario, haciendo que la aplicación sea más atractiva y

accesible para todos los usuarios. En resumen, el trabajo de Víctor ha sido fundamental para el éxito del proyecto y ha resultado en una aplicación que cumple con los criterios de una buena aplicación móvil.

José David López Geraghty

Con una sólida experiencia en el desarrollo de aplicaciones en entornos profesionales, José David López Geraghty lideró la conceptualización y la planificación inicial de la arquitectura de la aplicación. Su enfoque estratégico y su conocimiento técnico fueron fundamentales para establecer los cimientos de un proyecto robusto y escalable. Entre sus contribuciones destacadas se incluyen:

- Diseño arquitectónico robusto: David propuso una arquitectura clásica basada en 3 capas, donde la separación clara de la base de datos, el backend y los clientes garantizaba la modularidad y la mantenibilidad del sistema a largo plazo.

- Tecnologías modernas seleccionadas: Identificando las necesidades del proyecto, recomendó el uso de tecnologías líderes en el mercado. Spring Boot se destacó como una opción robusta para el backend, mientras que Docker ofrecía una solución eficiente para la distribución de la aplicación, alineándola con las últimas tendencias en desarrollo de software.

- Implementación de prácticas de desarrollo ágil: Con el objetivo de promover la eficiencia y la calidad del código, José David propuso la adopción de Github Actions para establecer flujos de desarrollo continuo e integración continua, permitiendo entregas frecuentes y confiables.

- Diseño del prototipo Figma: Reconociendo la importancia del diseño de la interfaz de usuario, lideró el proceso de diseño inicial utilizando Figma. Su dedicación a la investigación y su habilidad para adaptar el diseño a diferentes dispositivos y necesidades del sistema gtd, logrando un resultado intuitivo y fácil de usar.

- Desarrollo de la aplicación nativa móvil Android: Como parte del equipo de desarrollo de la aplicación nativa, José David sugirió la adopción de Kotlin como lenguaje de programación para la aplicación, aprovechando sus características modernas y su interoperabilidad con el ecosistema de Android. Dentro de este desarrollo se ha centrado más específicamente en las integraciones de las vistas con el api, en desarrollar un sistema de clases con sentido y en formar parte del desarrollo del modo offline en la aplicación. Además ha participado en la parte gráfica diseñando layouts reutilizables y logos.

- Desarrollo de la memoria: Ha escrito el capítulo del diseño de la interfaz de usuario así como el que trata sobre la implementación de la aplicación nativa para móvil Android al haber participado muy activamente en estas dos tareas durante el proyecto. Además, ha ayudado en otras partes de la memoria y ha incidido en la importancia de incluir diagramas.

Gracias a su agilidad en el desarrollo, ha desbloqueado a otros miembros del equipo cuando experimentaban problemas técnicos como fue el caso del login con jwt en el backend, la integración del api en la aplicación Android y el correcto funcionamiento del modo offline entre otras tareas.

Además de su papel en la planificación y la conceptualización, José David desempeñó un papel activo en la implementación técnica del proyecto. Se destacó por su capacidad para resolver problemas complejos, especialmente en la implementación de soluciones de autenticación seguras utilizando JWT y la integración de proveedores de terceros como Google para el inicio de sesión.

Como coordinador del equipo, José David demostró habilidades excepcionales de liderazgo y gestión. Supervisó de cerca el progreso del equipo, asignando tareas de manera equitativa y organizando reuniones regulares para revisar el avance del proyecto. Su compromiso con la excelencia y su capacidad para fomentar un ambiente colaborativo fueron fundamentales para el éxito del equipo.

Además de sus responsabilidades principales, José David también brindó soporte durante el desarrollo de la aplicación Android, contribuyendo al desarrollo de características clave y asegurando la coherencia entre las diferentes partes del sistema.

En un esfuerzo por facilitar el acceso y la colaboración, José David proporcionó un espacio en un servidor público para alojar el proyecto, garantizando su disponibilidad y accesibilidad durante todas las etapas del desarrollo y las pruebas. Su dedicación a la gestión de recursos fue fundamental para mantener un entorno de desarrollo eficiente y productivo.

Matías Antonio Alonso Eiras

El núcleo del trabajo que ha realizado Matías A. Alonso Eiras del Trabajo de Fin de Grado se ha concentrado en el desarrollo e implementación del cliente de navegador y de escritorio. Su conocimiento adquirido por la labor que ha realizado en su carrera profesional fue increíblemente valiosa a el desarrollo e implementación del cliente, además de la ayuda proporcionada a sus compañeros. A continuación se enumeran el grueso de de sus aportaciones:

- Desarrollo e implementación completa del cliente de escritorio y navegador a través del lenguaje Dart y el framework de Flutter:

- **Ampliación de los conocimientos en las tecnologías**: Debido a la naturaleza puntera de Dart y Flutter, y el desconocimiento que tenían los miembros del grupo sobre estas, Matías asumió la responsabilidad de aprender y manejar con soltura Dart y Flutter para el desarrollo del cliente.
- **Estructura del proyecto**: Matías diseñó la estructura del proyecto de Flutter, asegurándose de que los componentes estuvieran bien organizados y que el código fuera fácilmente mantenible y escalable.
- **Arquitectura**: Implementó una arquitectura robusta que facilitó el desarrollo modular y la reutilización del código, optimizando así el proceso de desarrollo.
- **Implementación del modelo de datos lógico del cliente**: Desarrolló el modelo de datos lógico de la aplicación, garantizando una gestión eficaz de los datos y su integración fluida con la interfaz de usuario y el servidor.
- **Pantallas**: Creó interfaces de usuario intuitivas y atractivas, diseñando cada pantalla para mejorar la experiencia del usuario y asegurando su funcionalidad con diversos tipos de datos.
- **Peticiones a la API del servidor**: Desarrolló y gestionó las peticiones a la API del servidor, asegurando la comunicación efectiva entre el cliente y el servidor.
- **Métodos fromJson y toJson**: Implementó estos métodos para facilitar la serialización y deserialización de datos, crucial para el intercambio de datos entre el cliente y el servidor.
- **Ensayo del cliente de Flutter**: Lideró las pruebas del cliente, empleando técnicas de testing para verificar la funcionalidad y robustez de la aplicación.
- **Diseño adaptivo de la interfaz**: Aseguró que la aplicación fuera usable en dimensiones variables, adaptando la interfaz para diferentes dispositivos y orientaciones de pantalla.
- **Iteración del diseño de la interfaz con Víctor y David**: Colaboró estrechamente con Víctor y David para iterar sobre el diseño de la interfaz, mejorando continuamente la experiencia del usuario.
- **Programación de nuevas funcionalidades**: Implementó funcionalidades avanzadas como gestos y selección de tema, que no estaban inicialmente previstas, añadiendo valor adicional al proyecto.
- **Aportaciones en el diseño de los temas y estética de la interfaz**: Contribuyó significativamente en el diseño visual de la aplicación, asegurando que fuera tanto funcional como estéticamente agradable.

- Colaboración con sus compañeros para aspectos cruciales del TFG:
  - **Estrategia de integración del SSO de Google**: Aportó la estrategia para integrar el SSO de Google con el modelo de datos del backend de

forma sencilla, facilitando un acceso cómodo, seguro y eficiente para los usuarios.

- **Coordinación con Camilo**: Trabajó en estrecha colaboración con Camilo para implementar la interacción entre el cliente y el servidor, garantizando una comunicación fluida y eficaz.
- **Creación y configuración del contenedor de desarrollo**: Estableció un entorno de contenedor de desarrollo que permite a otros equipos desarrollar el cliente de manera eficiente y rápido, evitando problemas.
- **Documentación provisional**: Redactó documentación provisional que detallaba la instalación y configuración del proyecto y sus dependencias, facilitando el proceso para futuros desarrolladores.
- **Colaboración con Jhimmy en GitHub Actions**: Colaboró con Jhimmy para implementar el flujo de GitHub Actions, asegurando la integración continua con el servidor.
- **Contribuciones al código fuente del servidor y examinación**: Realizó pequeñas mejoras en el código del servidor para optimizar la interacción cliente-servidor y participó en el testing del servidor, realizando pruebas cruciales que aseguraron la calidad y estabilidad del sistema.

El trabajo de Matías fue esencial para el éxito del proyecto, teniendo un impacto significativo en la calidad y en la entrega del mismo. Sus habilidades técnicas y su capacidad para trabajar en colaboración con el equipo jugaron un papel fundamental en la culminación exitosa del proyecto de grado.

Apéndice B

Documentación detallada de la API Spring Boot

Para la generación de la documentación de la API se ha empleado de `Swagger`[23], una tecnología que permite automatizar la generación de documentación para APIs bajo el estándar OpenAPI 3.0.

Esta documentación es relevante para el desarrollo de las aplicaciones cliente, dado que especifica qué endpoints existen, cómo se llaman, cuál es su función, que respuestas se pueden esperar al llamar a cada endpoint, etc.

Get Tasks Done - Definición del API v1 OAS 3.0

Controlador de notas

En este controlador se encuentran todas las operaciones relativas a gestionar las notas de la aplicación.

POST `/note/create` Crea una nota.

Se da de alta una nota en la aplicación.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
Target * required `string($Target={id})` *(query)*	Entidad a la que se le asignará la nota. Puede ser Project o Task Target=Task
ID * required `string($ID={id})` *(query)*	Identificador de la tarea/proyecto donde se asignará la nota. ID=1

Request body required — application/json

Parámetros de entrada de la nota.

Example Value | Schema

```
{
  "contenido": "Nota de prueba",
  "creacion": "2024-12-31 23:59:59"
}
```

Responses

Code	Description	Links
200	La nota se ha creado correctamente. Se retorna el identificador de la nota nueva. Media type string	*No links*

Code	Description	Links
	Controls `Accept` header. **Example Value** | Schema `0`	
400	Ha ocurrido un error al realizar la llamada. Media type: **text/plain** Examples: **No se ha encontrado el proyecto o no se tier** **Example Value** `Project not found.` **Example Description** No se ha encontrado el proyecto o no se tiene permisos.	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

PATCH `/note/update/{id}` Modifica una nota.

Se modifica una nota existente en la aplicación. Solo el usuario que ha creado la nota o un administrador pueden modificarla.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
id * required `string(${id})` *(path)*	Identificador de la nota a modificar 1

Request body required — application/json

Parámetros de entrada de la nota.

Example Value | Schema

```
{
  "contenido": "Nota de prueba",
  "creacion": "2024-12-31 23:59:59"
}
```

Responses

Code	Description	Links
200	La nota se ha modificado correctamente. Media type string Controls `Accept` header. **Example Value** | Schema `Note updated.`	*No links*
400	No se ha encontrado la nota, o no se tiene autorización para modificarla. Media type string **Example Value** | Schema `Note not found.`	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

GET `/note/{id}` Busca una nota.

Se busca una nota específica en la aplicación. Solamente el usuario que haya creado la nota o un administrador podrán verla.

Parameters

Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
id * required `string(${id})` *(path)*	Identificador de la nota a consultar 1

Responses

Code	Description	Links
200	La llamada ha respondido correctamente. Media type: application/json Controls `Accept` header. **Example Value** | Schema `{ "id": 1, "contenido": "Esto es una nota.", "creacion": "2024-05-01T09:40:43.639Z", "tareaId": 0, "proyectoId": 0 }`	*No links*
404	La nota no existe en la aplicación, o no se tiene permiso para consultarla. Media type: string **Example Value** | Schema `Note not found.`	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

GET `/note/getNotes` Obtiene todas las notas.

Se obtienen todas las notas registradas en la aplicación. REQUIERE PRIVILEGIOS DE ADMINISTRADOR.

Parameters Try it out

Name	Description
Authorization * required `string($Bearer [token])` *(header)*	Token de autenticación Authorization

Responses

Code	Description	Links
200	La llamada ha respondido correctamente. Media type: application/json Controls `Accept` header.	*No links*

Code	Description	Links

Example Value | Schema


```
[
  {
    "id": 1,
    "contenido": "Esto es una nota.",
    "creacion": "2024-05-01T09:40:43.642Z",
    "tareaId": 0,
    "proyectoId": 0
  }
]
```


CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

GET `/note/authed` Obtiene las notas del usuario autenticado.

Se obtienen las notas registradas en la aplicación por el usuario autenticado actualmente.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer [token])` *(header)*	Token de autenticación Authorization

Responses

Code	Description	Links
200	La llamada ha respondido correctamente. Media type application/json Controls `Accept` header. **Example Value** | Schema	*No links*


```
[
  {
    "id": 1,
    "contenido": "Esto es una nota.",
    "creacion": "2024-05-01T09:40:43.646Z",
    "tareaId": 0,
    "proyectoId": 0
  }
]
```


CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

DELETE `/note/delete/{id}` Elimina una nota.

Se borra una nota de la aplicación. Solo el usuario que ha creado la nota o un administrador pueden borrarla.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
id * required `string(${id})` *(path)*	Identificador de la nota a eliminar 1

Responses

Code	Description	Links
200	La nota se ha eliminado correctamente. Media type string Controls `Accept` header. **Example Value** | Schema `Note deleted.`	*No links*
404	No se ha encontrado la nota, o no se tiene autorización para eliminarla. Media type string **Example Value** | Schema `Note not found.`	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

Controlador de tareas

En este controlador se encuentran todas las operaciones relativas a gestionar las tareas de la aplicación.

POST `/task/create` Crea una tarea.

Se da de alta una tarea en la aplicación.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
ProjectID `string($ProjectID={id})` *(query)*	Identificador del proyecto a donde se desea asignar la tarea ProjectID=3

Request body required — application/json

Parámetros de entrada de la tarea.

Example Value | Schema

```
{
  "contexto": {
    "id": 1
  },
  "titulo": "Tarea de prueba",
  "descripcion": "Este es un proyecto de prueba",
  "creacion": "2024-12-31 23:59:59",
  "vencimiento": "2024-12-31 23:59:59",
  "estado": "agendado",
  "prioridad": 0
}
```

Responses

Code	Description	Links
200	La tarea se ha creado correctamente. Media type: string Controls `Accept` header. **Example Value** | Schema `0`	*No links*
400	Ha ocurrido un error al procesar la solicitud. Media type: text/plain Examples: El contexto especificado no se ha encontrad **Example Value**	*No links*

Code	Description	Links
	`Context not found.` **Example Description** El contexto especificado no se ha encontrado en el sistema, o no se tiene permisos para utilizarlo	
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

PATCH `/task/update/{id}` Modifica una tarea.

Se modifica una tarea existente en la aplicación. Solo el usuario que ha creado la tarea o un administrador pueden modificarla.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
id * required `string(${id})` *(path)*	Identificador de la tarea a modificar 1
ProjectID `string($ProjectID={id})` *(query)*	Identificador del proyecto a donde se desea mover la tarea ProjectID=3

Request body required — application/json

Parámetros de entrada de la tarea.

Example Value | Schema

```
{
  "contexto": {
    "id": 1
  },
  "titulo": "Tarea de prueba",
  "descripcion": "Este es un proyecto de prueba",
  "creacion": "2024-12-31 23:59:59",
  "vencimiento": "2024-12-31 23:59:59",
  "estado": "agendado",
  "prioridad": 0
}
```

Responses

Code	Description	Links
200	La tarea se ha modificado correctamente. Media type: string Controls `Accept` header. **Example Value** | Schema `Task updated.`	*No links*
400	Ha ocurrido un error al procesar la solicitud. Media type: text/plain Examples: No se ha encontrado la tarea, o no se tiene a **Example Value** `Task not found.` **Example Description** No se ha encontrado la tarea, o no se tiene autorización para modificarla.	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

PATCH `/task/removeTag/{id}` Elimina una etiqueta de una tarea.

Se quita una etiqueta de una tarea existente en la aplicación. Solo el usuario que ha creado la tarea y etiqueta o un administrador puede interactuar con estos componentes.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
id * required `string(${id})` *(path)*	Identificador de la tarea al que se le eliminará la etiqueta 1
TagID * required `string($TagID={id})` *(query)*	Identificador de la etiqueta a remover TagID=1

Responses

Code	Description	Links
200	La etiqueta se ha eliminado de la tarea correctamente. Media type: string Controls `Accept` header. **Example Value** | Schema `Tag deleted from task.`	*No links*
400	Ha ocurrido un error al procesar la solicitud. Media type: text/plain Examples: No se ha encontrado la tarea, o no se tiene a **Example Value** `Task not found.` **Example Description** No se ha encontrado la tarea, o no se tiene autorización para modificarla.	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

PATCH `/task/addTag/{id}` Añade una etiqueta a una tarea.

Se añade una etiqueta a una tarea existente en la aplicación. Solo el usuario que ha creado la tarea y etiqueta o un administrador puede interactuar con estos componentes.

Parameters

Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
id * required `string(${id})` *(path)*	Identificador de la tarea al que se le añadirá la etiqueta 1
TagID * required `string($TagID={id})` *(query)*	Identificador de la etiqueta a agregar TagID=1

Responses

Code	Description	Links
200	La etiqueta se ha añadido a la tarea correctamente. Media type: string Controls `Accept` header. **Example Value** | Schema `Tag added to task.`	*No links*
400	Ha ocurrido un error al procesar la solicitud. Media type: text/plain Examples: No se ha encontrado la tarea, o no se tiene a **Example Value** `Task not found.` **Example Description** No se ha encontrado la tarea, o no se tiene autorización para modificarla.	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

GET `/task/{id}` Busca una tarea.

Se busca una tarea específica en la aplicación. Solamente el usuario que haya creado la tarea o un administrador podrán verlo.

Parameters Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
id * required `string(${id})` *(path)*	Identificador de la tarea a consultar 1

Responses

Code	Description	Links
200	La llamada ha respondido correctamente. Media type application/json Controls `Accept` header. **Example Value** | Schema `{ "id": 1, "titulo": "Tarea de prueba", "descripcion": "Esta es la descripción de una tarea.", "creacion": "2024-05-01T09:40:43.670Z", "vencimiento": "2024-05-01T09:40:43.670Z", "estado": "string", "prioridad": 0, "contexto": { "id": 1, "nombre": "Base de datos" }, "checkItems": [{ "id": 1, "contenido": "Este es un elemento de comprobación.", "esta_marcado": false, "tareaId": 0 }], "notas": [{ "id": 1, "contenido": "Esto es una nota.", "creacion": "2024-05-01T09:40:43.670Z", "tareaId": 0, "proyectoId": 0 }`	*No links*
404	La tarea no existe en la aplicación, o no se tiene permiso para consultarla. Media type string **Example Value** | Schema `Task not found.`	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

GET `/task/getTasks` Obtiene todos las tareas.

Se obtienen todos las tareas registradas en la aplicación. REQUIERE PRIVILEGIOS DE ADMINISTRADOR.

Parameters Try it out

Name	Description
Authorization * required `string($Bearer [token])` *(header)*	Token de autenticación Authorization

Responses

Code	Description	Links
200	La llamada ha respondido correctamente. Media type application/json Controls `Accept` header. **Example Value** | Schema	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*


```
[
  {
    "id": 1,
    "titulo": "Tarea de prueba",
    "descripcion": "Esta es la descripción de una tarea.",
    "creacion": "2024-05-01T09:40:43.674Z",
    "vencimiento": "2024-05-01T09:40:43.674Z",
    "estado": "string",
    "prioridad": 0,
    "contexto": {
      "id": 1,
      "nombre": "Base de datos"
    },
    "checkItems": [
      {
        "id": 1,
        "contenido": "Este es un elemento de comprobación.",
        "esta_marcado": false,
        "tareaId": 0
      }
    ],
    "notas": [
      {
        "id": 1,
        "contenido": "Esto es una nota.",
        "creacion": "2024-05-01T09:40:43.674Z",
        "tareaId": 0,
        "proyectoId": 0
```


GET /task/authed Obtiene las tareas del usuario autenticado.

Se obtienen las tareas registradas en la aplicación por el usuario autenticado actualmente.

Parameters

Try it out

Name	Description
Authorization * required	Token de autenticación

Name	Description
string($Bearer [token]) *(header)*	Authorization

Responses

Code	Description	Links

200	La llamada ha respondido correctamente.
	
	Media type
	application/json
	Controls Accept header.
	
	Example Value | Schema
	
	[
	{
	"id": 1,
	"titulo": "Tarea de prueba",
	"descripcion": "Esta es la descripción de una tarea.",
	"creacion": "2024-05-01T09:40:43.677Z",
	"vencimiento": "2024-05-01T09:40:43.677Z",
	"estado": "string",
	"prioridad": 0,
	"contexto": {
	"id": 1,
	"nombre": "Base de datos"
	},
	"checkItems": [
	{
	"id": 1,
	"contenido": "Este es un elemento de comprobación.",
	"esta_marcado": false,
	"tareaId": 0
	}
	],
	"notas": [
	{
	"id": 1,
	"contenido": "Esto es una nota.",
	"creacion": "2024-05-01T09:40:43.677Z",
	"tareaId": 0,
	"proyectoId": 0
		No links

CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

DELETE /task/delete/{id} Elimina una tarea.

Se borra una tarea de la aplicación. Solo el usuario que ha creado la tarea o un administrador pueden borrarla.

Parameters

Try it out

Name	Description
Authorization * required string($Bearer {token}) *(header)*	Token de autenticación Authorization

Name	Description
id * required `string(${id})` *(path)*	Identificador de la tarea a eliminar 1

Responses

Code	Description	Links
200	La tarea se ha eliminado correctamente. Media type string Controls `Accept` header. **Example Value** | Schema `Task deleted.`	*No links*
404	No se ha encontrado la tarea, o no se tiene autorización para eliminarla. Media type string **Example Value** | Schema `Task not found.`	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

Controlador de acceso de usuarios

En este controlador se encuentran todas las operaciones relativas a gestionar las credenciales de acceso de los usuarios de la aplicación junto el controlador de autorización.

PATCH `/user/update/{id}` Modifica las credenciales de un usuario.

Se modifican las credenciales de acceso de un usuario existente en la aplicación. Solo el usuario o un administrador pueden modificar esta información.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer {token})`	Token de autenticación

Name	Description
(header)	Authorization
id * required `string(${id})` *(path)*	Identificador del usuario a modificar 1

Request body required — application/json

Parámetros de entrada del usuario.

Example Value | Schema

```
{
  "email": "user@gettasksdone.com",
  "password": "newPassword"
}
```

Responses

Code	Description	Links
200	Las credenciales del usuario se han modificado correctamente. Media type: string Controls `Accept` header. **Example Value** | Schema `Update completed.`	*No links*
404	No se ha encontrado el usuario, o no se tiene autorización para modificarlo. Media type: string **Example Value** | Schema `User not found.`	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

GET `/user/{id}` Busca las credenciales de un usuario.

Se buscan las credenciales de acceso de un usuario específico en la aplicación. REQUIERE PRIVILEGIOS DE ADMINISTRADOR.

Parameters

Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
id * required `string(${id})` *(path)*	Identificador del usuario a consultar 1

Responses

Code	Description	Links
200	La llamada ha respondido correctamente. Media type application/json Controls `Accept` header. **Example Value** | Schema `{ "id": 1, "username": "usuarioPrueba", "email": "prueba@gettasksdone.com", "rol": "0" }`	*No links*
404	El usuario no existe en la aplicación. Media type string **Example Value** | Schema `User not found.`	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

GET `/user/users` Obtiene las credenciales de todos los usuarios registrados en el sistema.

Se obtienen todas las credenciales de los usuarios registrados en la aplicación. REQUIERE PRIVILEGIOS DE ADMINISTRADOR.

Parameters

Try it out

Name	Description
Authorization * required `string($Bearer [token])` *(header)*	Token de autenticación Authorization

Responses

Code	Description	Links
200	La llamada ha respondido correctamente. Media type application/json Controls `Accept` header. **Example Value** | Schema `[ { "id": 1, "username": "usuarioPrueba", "email": "prueba@gettasksdone.com", "rol": "0" } ]`	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

GET `/user/authed` Obtiene las credenciales del usuario autenticado.

Se obtienen las credenciales del usuario autenticado actualmente.

Parameters Try it out

Name	Description
Authorization * required `string($Bearer [token])` *(header)*	Token de autenticación Authorization

Responses

Code	Description	Links
200	La llamada ha respondido correctamente. Media type	*No links*

Code	Description	Links
	application/json Controls `Accept` header. **Example Value** | Schema `[ { "id": 1, "username": "usuarioPrueba", "email": "prueba@gettasksdone.com", "rol": "0" } ]`	
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

DELETE `/user/delete/{id}` Elimina un usuario.

Se da de baja a un usuario de la aplicación. REQUIERE PRIVILEGIOS DE ADMINISTRADOR.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
id * required `string(${id})` *(path)*	Identificador del usuario a eliminar 1

Responses

Code	Description	Links
200	El usuario se ha eliminado correctamente. Media type string Controls `Accept` header. **Example Value** | Schema `User deleted.`	*No links*
404	No se ha encontrado el usuario. Media type string	*No links*

Code	Description	Links
	Example Value | Schema `User not found.`	
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

Controlador de proyectos

En este controlador se encuentran todas las operaciones relativas a gestionar los proyectos de la aplicación.

POST `/project/create` Crea un proyecto.

Se da de alta un proyecto en la aplicación.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization

Request body required — application/json

Parámetros de entrada del proyecto.

Example Value | Schema

```
{
  "nombre": "Proyecto de prueba",
  "inicio": "2024-12-31 23:59:59",
  "fin": "2024-12-31 23:59:59",
  "descripcion": "Este es un proyecto de prueba",
  "estado": "agendado"
}
```

Responses

Code	Description	Links
200	El proyecto se ha creado correctamente. Se retorna el identificador del proyecto nuevo. Media type string Controls `Accept` header. **Example Value** | Schema	*No links*

Code	Description	Links
	0	
400	Se ha intentado crear un proyecto inbox nuevo. Media type: text/plain Examples: Se intenta crear un proyecto de nombre inbc Example Value Inbox is a reservated project name. **Example Description** No es posible crear un proyecto con nombre inbox, ya que es un nombre de proyecto reservado para la bandeja de entrada.	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

PATCH `/project/update/{id}` Modifica un proyecto.

Se modifica un proyecto existente en la aplicación. Solo el usuario que ha creado el proyecto o un administrador pueden modificarlo.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
id * required `string(${id})` *(path)*	Identificador del proyecto a modificar 1

Request body required — application/json

Parámetros de entrada del proyecto.

Example Value | Schema

```
{
  "nombre": "Proyecto de prueba",
  "inicio": "2024-12-31 23:59:59",
  "fin": "2024-12-31 23:59:59",
  "descripcion": "Este es un proyecto de prueba",
  "estado": "agendado"
}
```

Responses

Code	Description	Links
200	El proyecto se ha modificado correctamente. Media type: string Controls `Accept` header. **Example Value** | Schema `Project updated.`	*No links*
400	Ha ocurrido un error al procesar la solicitud. Media type: text/plain Examples: No se ha encontrado el proyecto, o no se tie **Example Value** `Project not found.` **Example Description** No se ha encontrado el proyecto, o no se tiene autorización para modificarlo.	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

PATCH `/project/removeTag/{id}` Elimina una etiqueta de un proyecto.

Se quita una etiqueta de un proyecto existente en la aplicación. Solo el usuario que ha creado el proyecto y etiqueta o un administrador puede interactuar con estos componentes.

Parameters

Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
id * required `string(${id})` *(path)*	Identificador del proyecto al que se le eliminará la etiqueta 1
TagID * required	Identificador de la etiqueta a remover

Name	Description
string($TagID={id}) *(query)*	TagID=1

Responses

Code	Description	Links
200	La etiqueta se ha eliminado del proyecto correctamente. Media type: string Controls Accept header. Example Value | Schema `Tag added to project.`	*No links*
400	Ha ocurrido un error al procesar la solicitud. Media type: text/plain Examples: No se ha encontrado el proyecto, o no se tie Example Value `Project not found.` **Example Description** No se ha encontrado el proyecto, o no se tiene autorización para modificarlo.	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

PATCH `/project/addTag/{id}` Añade una etiqueta a un proyecto.

Se añade una etiqueta a un proyecto existente en la aplicación. Solo el usuario que ha creado el proyecto y etiqueta o un administrador puede interactuar con estos componentes.

Parameters Try it out

Name	Description
Authorization * required string($Bearer {token}) *(header)*	Token de autenticación Authorization

Name	Description
id * required `string($\{id\})` *(path)*	Identificador del proyecto al que se le añadirá la etiqueta 1
TagID * required `string($TagID={id})` *(query)*	Identificador de la etiqueta a agregar TagID=1

Responses

Code	Description	Links
200	La etiqueta se ha añadido al proyecto correctamente. Media type: string Controls `Accept` header. **Example Value** | Schema `Tag added to project.`	*No links*
400	Ha ocurrido un error al procesar la solicitud. Media type: text/plain Examples: No se ha encontrado el proyecto, o no se tie **Example Value** `Project not found.` **Example Description** No se ha encontrado el proyecto, o no se tiene autorización para modificarlo.	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

GET /project/{id} Busca un proyecto.

Se busca un proyecto específico en la aplicación. Solamente el usuario que haya creado el proyecto o un administrador podrán verlo.

Parameters

Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
id * required `string(${id})` *(path)*	Identificador del proyecto a consultar 1

Responses

Code	Description	Links
200	La llamada ha respondido correctamente. Media type application/json Controls `Accept` header. **Example Value** | Schema	*No links*


```
{
  "id": 1,
  "nombre": "Proyecto de pruebas",
  "inicio": "2024-05-01T09:40:43.717Z",
  "fin": "2024-05-01T09:40:43.717Z",
  "descripcion": "string",
  "estado": "string",
  "tareas": [
    {
      "id": 1,
      "titulo": "Tarea de prueba",
      "descripcion": "Esta es la descripción de una tarea.",
      "creacion": "2024-05-01T09:40:43.717Z",
      "vencimiento": "2024-05-01T09:40:43.717Z",
      "estado": "string",
      "prioridad": 0,
      "contexto": {
        "id": 1,
        "nombre": "Base de datos"
      },
      "checkItems": [
        {
          "id": 1,
          "contenido": "Este es un elemento de comprobación.",
          "esta_marcado": false,
          "tareaId": 0
        }
      ],
```


404	El proyecto no existe en la aplicación, o no se tiene permiso para consultarlo. Media type string **Example Value** | Schema	*No links*

```
Project not found.
```

Code	Description	Links
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

GET `/project/getProjects` Obtiene todos los proyectos.

Se obtienen todos los proyectos registrados en la aplicación. REQUIERE PRIVILEGIOS DE ADMINISTRADOR.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer [token])` *(header)*	Token de autenticación Authorization

Responses

Code	Description	Links
200	La llamada ha respondido correctamente. Media type application/json Controls `Accept` header. **Example Value** | Schema	*No links*


```
[
  {
    "id": 1,
    "nombre": "Proyecto de pruebas",
    "inicio": "2024-05-01T09:40:43.723Z",
    "fin": "2024-05-01T09:40:43.723Z",
    "descripcion": "string",
    "estado": "string",
    "tareas": [
      {
        "id": 1,
        "titulo": "Tarea de prueba",
        "descripcion": "Esta es la descripción de una tarea.",
        "creacion": "2024-05-01T09:40:43.723Z",
        "vencimiento": "2024-05-01T09:40:43.723Z",
        "estado": "string",
        "prioridad": 0,
        "contexto": {
          "id": 1,
          "nombre": "Base de datos"
        },
        "checkItems": [
          {
            "id": 1,
            "contenido": "Este es un elemento de comprobación.",
            "esta_marcado": false,
            "tareaId": 0
          }
```


CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

GET `/project/authed` Obtiene los proyectos del usuario autenticado.

Se obtienen los proyectos registrados en la aplicación por el usuario autenticado actualmente.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer [token])` *(header)*	Token de autenticación Authorization

Responses

Code	Description	Links
200	La llamada ha respondido correctamente. Media type: application/json Controls `Accept` header. **Example Value** | Schema	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*


```
[
  {
    "id": 1,
    "nombre": "Proyecto de pruebas",
    "inicio": "2024-05-01T09:40:43.726Z",
    "fin": "2024-05-01T09:40:43.726Z",
    "descripcion": "string",
    "estado": "string",
    "tareas": [
      {
        "id": 1,
        "titulo": "Tarea de prueba",
        "descripcion": "Esta es la descripción de una tarea.",
        "creacion": "2024-05-01T09:40:43.726Z",
        "vencimiento": "2024-05-01T09:40:43.726Z",
        "estado": "string",
        "prioridad": 0,
        "contexto": {
          "id": 1,
          "nombre": "Base de datos"
        },
        "checkItems": [
          {
            "id": 1,
            "contenido": "Este es un elemento de comprobación.",
            "esta_marcado": false,
            "tareaId": 0
          }
```


DELETE `/project/delete/{id}` Elimina un proyecto.

Se borra un proyecto de la aplicación. Solo el usuario que ha creado el proyecto o un administrador pueden borrarlo.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
id * required `string(${id})` *(path)*	Identificador del proyecto a eliminar 1

Responses

Code	Description	Links
200	El proyecto se ha eliminado correctamente. Media type: **string** Controls `Accept` header. **Example Value** | Schema `Project deleted.`	*No links*
400	No es posible eliminar el proyecto inbox, ya que es un proyecto reservado para la bandeja de entrada. Media type: **text/plain** **Example Value** | Schema `Inbox project is protected from deletion or modification`	*No links*
404	No se ha encontrado el proyecto, o no se tiene autorización para eliminarlo. Media type: **string** **Example Value** | Schema `Project not found.`	*No links*

Code	Description	Links
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

Controlador de etiquetas

En este controlador se encuentran todas las operaciones relativas a gestionar las etiquetas de la aplicación.

POST `/tag/createTag` Crea una etiqueta.

Se da de alta una etiqueta en la aplicación.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization

Request body required — application/json

Parámetros de entrada de la etiqueta.

Example Value | Schema

```
{
  "nombre": "Etiqueta de prueba"
}
```

Responses

Code	Description	Links
200	La etiqueta se ha creado correctamente. Se retorna el identificador de la etiqueta nueva. Media type: string Controls `Accept` header. **Example Value** | Schema `0`	*No links*

Code	Description	Links
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

PATCH `/tag/update/{id}` Modifica una etiqueta.

Se modifica una etiqueta existente en la aplicación. Solo el usuario que ha creado la etiqueta o un administrador pueden modificarla.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
id * required `string(${id})` *(path)*	Identificador de la etiqueta a modificar 1

Request body required — application/json

Parámetros de entrada de la etiqueta.

Example Value | Schema

```
{
  "nombre": "Etiqueta de prueba"
}
```

Responses

Code	Description	Links
200	La etiqueta se ha modificado correctamente. Media type string Controls `Accept` header. **Example Value** | Schema `Tag updated.`	*No links*
400	No se ha encontrado la etiqueta, o no se tiene autorización para modificarla. Media type	*No links*

Code	Description	Links
	string **Example Value** | Schema `Tag not found.`	
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

GET `/tag/{id}` Busca una etiqueta.

Se busca una etiqueta específica en la aplicación. Solamente el usuario que haya creado la etiqueta o un administrador podrán verla.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
id * required `string(${id})` *(path)*	Identificador de la etiqueta a consultar 1

Responses

Code	Description	Links
200	La llamada ha respondido correctamente. Media type application/json Controls `Accept` header. **Example Value** | Schema `{ "id": 1, "nombre": "Urgente" }`	*No links*
404	La etiqueta no existe en la aplicación, o no se tiene permiso para consultarla. Media type string **Example Value** | Schema	*No links*

Code	Description	Links
	Tag not found.	
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

GET `/tag/getTags` Obtiene todas las etiquetas.

Se obtienen todas las etiquetas registradas en la aplicación. REQUIERE PRIVILEGIOS DE ADMINISTRADOR.

Parameters Try it out

Name	Description
Authorization * required `string($Bearer [token])` *(header)*	Token de autenticación Authorization

Responses

Code	Description	Links
200	La llamada ha respondido correctamente. Media type application/json Controls `Accept` header. **Example Value** | Schema `[ { "id": 1, "nombre": "Urgente" } ]`	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

GET `/tag/authed` Obtiene las etiquetas del usuario autenticado.

Se obtienen las etiquetas registradas en la aplicación por el usuario autenticado actualmente.

Parameters Try it out

Name	Description
Authorization * required `string($Bearer [token])` *(header)*	Token de autenticación Authorization

Responses

Code	Description	Links
200	La llamada ha respondido correctamente. Media type application/json Controls `Accept` header. **Example Value** | Schema `[ { "id": 1, "nombre": "Urgente" } ]`	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

DELETE `/tag/delete/{id}` Elimina una etiqueta.

Se borra una etiqueta de la aplicación. Solo el usuario que ha creado la etiqueta o un administrador pueden borrarla.

Parameters Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
id * required `string(${id})` *(path)*	Identificador de la etiqueta a eliminar 1

Responses

Code	Description	Links
200	La etiqueta se ha eliminado correctamente. Media type string Controls `Accept` header. **Example Value** | Schema `Tag deleted.`	*No links*
404	No se ha encontrado la etiqueta, o no se tiene autorización para eliminarla. Media type string **Example Value** | Schema `Tag not found.`	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

Controlador de contextos

En este controlador se encuentran todas las operaciones relativas a gestionar los contextos de la aplicación.

POST `/context/createContext` Crea un contexto.

Se da de alta un contexto en la aplicación.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization

Request body required — application/json

Parámetros de entrada del contexto.

Example Value | Schema

```
{
  "nombre": "Contexto de prueba"
}
```

Responses

Code	Description	Links
200	El contexto se ha creado correctamente. Se retorna el identificador del contexto nuevo. Media type string Controls `Accept` header. **Example Value** | Schema `0`	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

PATCH `/context/update/{id}` Modifica un contexto.

Se modifica un contexto existente en la aplicación. Solo el usuario que ha creado el contexto o un administrador pueden modificarlo.

Parameters

Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
id * required `string(${id})` *(path)*	Identificador del contexto a modificar 1

Request body required — application/json

Parámetros de entrada del contexto.

Example Value | Schema

```
{
  "nombre": "Contexto de prueba"
}
```

Responses

Code	Description	Links
200	El contexto se ha modificado correctamente. Media type string Controls `Accept` header. **Example Value** | Schema `Context updated.`	*No links*
400	No se ha encontrado el contexto, o no se tiene autorización para modificarlo. Media type string **Example Value** | Schema `Context not found.`	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

GET `/context/{id}` Busca un contexto.

Se busca un contexto específico en la aplicación. Solamente el usuario que haya creado el contexto o un administrador podrán verlo.

Parameters Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
id * required `string(${id})` *(path)*	Identificador del contexto a consultar 1

Responses

Code	Description	Links
200	La llamada ha respondido correctamente. Media type	*No links*

Code	Description	Links

application/json

Controls `Accept` header.

Example Value | Schema

```
{
  "id": 1,
  "nombre": "Base de datos"
}
```

404	El contexto no existe en la aplicación, o no se tiene permiso para consultarlo. Media type: string **Example Value** | Schema `Context not found.`	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

GET `/context/getContexts` Obtiene todos los contextos.

Se obtienen todos los contextos registrados en la aplicación. REQUIERE PRIVILEGIOS DE ADMINISTRADOR.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer [token])` *(header)*	Token de autenticación Authorization

Responses

Code	Description	Links
200	La llamada ha respondido correctamente. Media type: application/json Controls `Accept` header. **Example Value** | Schema	*No links*

```
[
  {
    "id": 1,
    "nombre": "Base de datos"
```

Code	Description	Links
	<pre> }]</pre>	
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

GET `/context/authed` Obtiene los contextos del usuario autenticado.

Se obtienen los contextos registrados en la aplicación por el usuario autenticado actualmente.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer [token])` *(header)*	Token de autenticación Authorization

Responses

Code	Description	Links

200	La llamada ha respondido correctamente.
	Media type
	application/json
	Controls `Accept` header.
	Example Value | Schema
	<pre>[
	  {
	    "id": 1,
	    "nombre": "Base de datos"
	  }
	]</pre>
		No links

CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

DELETE `/context/delete/{id}` Elimina un contexto.

Se borra un contexto de la aplicación. Solo el usuario que ha creado el contexto o un administrador pueden borrarlo.

Parameters | Try it out

Name	Description
Authorization * required	Token de autenticación

Name	Description
`string($Bearer {token})` *(header)*	Authorization
id * required `string(${id})` *(path)*	Identificador del contexto a eliminar 1

Responses

Code	Description	Links
200	El contexto se ha eliminado correctamente. Media type: string Controls `Accept` header. **Example Value** | Schema `Context deleted.`	*No links*
404	No se ha encontrado el contexto, o no se tiene autorización para eliminarlo. Media type: string **Example Value** | Schema `Context not found.`	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

Controlador de elementos de comprobación

En este controlador se encuentran todas las operaciones relativas a gestionar los elementos de comprobación de la aplicación.

POST `/check/create` Crea un elemento de comprobación.

Se da de alta un contexto para una tarea en la aplicación.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
TaskID * required `string($TaskID={id})` *(query)*	Identificador de la tarea que contendrá al elemento de comprobación TaskID=1

Request body required — application/json

Parámetros de entrada del elemento de comprobación.

Example Value | Schema

```
{
  "contenido": "Check de prueba",
  "esta_marcado": false
}
```

Responses

Code	Description	Links
200	El elemento de comprobación se ha creado correctamente. Se retorna el identificador del elemento de comprobación nuevo. Media type: string Controls `Accept` header. **Example Value** | Schema `0`	*No links*
404	La tarea no existe en la aplicación, o no se tiene permiso para interactuar con ella. Media type: string **Example Value** | Schema `Task not found.`	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

PATCH `/check/update/{id}` Modifica un elemento de comprobación.

Se modifica un elemento de comprobación existente en la aplicación. Solo el usuario que ha creado el elemento de comprobación o un administrador pueden modificarlo.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
id * required `string(${id})` *(path)*	Identificador del elemento de comprobación a modificar 1

Request body required — application/json

Parámetros de entrada del elemento de comprobación.

Example Value | Schema

```
{
  "contenido": "Check de prueba",
  "esta_marcado": false
}
```

Responses

Code	Description	Links
200	El elemento de comprobación se ha modificado correctamente. Media type string Controls `Accept` header. **Example Value** | Schema `Check Item updated.`	*No links*
400	No se ha encontrado el elemento de comprobación, o no se tiene autorización para modificarlo. Media type string **Example Value** | Schema `Check Item not found.`	*No links*

Code	Description	Links
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

GET `/check/{id}` Busca un elemento de comprobación.

Se busca un elemento de comprobación específico en la aplicación. Solamente el usuario que haya creado el elemento de comprobación o un administrador podrán verlo.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
id * required `string(${id})` *(path)*	Identificador del elemento de comprobación a consultar 1

Responses

Code	Description	Links
200	La llamada ha respondido correctamente. Media type: application/json Controls `Accept` header. **Example Value** | Schema	*No links*
404	El elemento de comprobación no existe en la aplicación, o no se tiene permiso para consultarlo. Media type: string **Example Value** | Schema	*No links*


```
{
  "id": 1,
  "contenido": "Este es un elemento de comprobación.",
  "esta_marcado": false,
  "tareaId": 0
}
```


```
Check Item not found.
```

Code	Description	Links
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

GET `/check/getChecks` Obtiene todos los elementos de comprobación.

Se obtienen todos los elementos de comprobación registrados en la aplicación. REQUIERE PRIVILEGIOS DE ADMINISTRADOR.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer [token])` *(header)*	Token de autenticación Authorization

Responses

Code	Description	Links
200	La llamada ha respondido correctamente. Media type: application/json Controls `Accept` header. **Example Value** | Schema	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

```
[
  {
    "id": 1,
    "contenido": "Este es un elemento de comprobación.",
    "esta_marcado": false,
    "tareaId": 0
  }
]
```

GET `/check/authed` Obtiene los elementos de comprobación del usuario autenticado.

Se obtienen los elementos de comprobación registrados en la aplicación por el usuario autenticado actualmente.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer [token])` *(header)*	Token de autenticación Authorization

Responses

Code	Description	Links
200	La llamada ha respondido correctamente. Media type application/json Controls `Accept` header. **Example Value** | Schema `[ { "id": 1, "contenido": "Este es un elemento de comprobación.", "esta_marcado": false, "tareaId": 0 } ]`	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

DELETE `/check/delete/{id}` Elimina un elemento de comprobación.

Se borra un elemento de comprobación de la aplicación. Solo el usuario que ha creado el elemento de comprobación o un administrador pueden borrarlo.

Parameters Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
id * required `string(${id})` *(path)*	Identificador del elemento de comprobación a eliminar 1

Responses

Code	Description	Links
200	El elemento de comprobación se ha eliminado correctamente. Media type string Controls `Accept` header. **Example Value** | Schema `Check Item deleted.`	*No links*
404	No se ha encontrado el elemento de comprobación, o no se tiene autorización para eliminarlo. Media type string **Example Value** | Schema `Check Item not found.`	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

Controlador de información adicional de usuarios

En este controlador se encuentran todas las operaciones relativas a gestionar la información adicional de los usuarios registrados en la aplicación.

POST `/userData/create` Crea un nodo de información adicional.

Se da de alta un nodo de información adicional en la aplicación.

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization

Request body required — application/json

Parámetros de entrada del nodo de información adicional.

Example Value | Schema

```
{
  "nombre": "Pedro",
  "telefono": "123456789",
  "puesto": "Reclutador",
  "departamento": "Recursos Humanos"
}
```

Responses

Code	Description	Links
200	El nodo de información adicional se ha creado correctamente. Media type: string Controls `Accept` header. Example Value | Schema `User data created.`	*No links*
400	El usuario que ha intentado crear el nodo de información adicional ya tiene un nodo propio. Media type: string Example Value | Schema `User data already exist for this user.`	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

PATCH `/userData/update/{id}` Modifica un nodo de información adicional.

Se modifica un nodo de información adicional existente en la aplicación. Solo el usuario que ha creado el nodo de información adicional o un administrador pueden modificarlo.

Parameters

Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization

Name	Description
id * required `string(${id})` *(path)*	Identificador del nodo de información adicional a modificar 1

Request body required — application/json

Parámetros de entrada del nodo de información adicional.

Example Value | Schema

```
{
  "nombre": "Pedro",
  "telefono": "123456789",
  "puesto": "Reclutador",
  "departamento": "Recursos Humanos"
}
```

Responses

Code	Description	Links
200	El nodo de información adicional se ha modificado correctamente. Media type: string Controls `Accept` header. **Example Value** | Schema `User data updated.`	*No links*
404	No se ha encontrado el nodo de información adicional, o no se tiene autorización para modificarlo. Media type: string **Example Value** | Schema `User data not found.`	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

GET `/userData/{id}` Busca un nodo de información adicional.

Se busca un nodo de información adicional específico en la aplicación. REQUIERE PRIVILEGIOS DE ADMINISTRADOR.

Parameters

Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
id * required `string(${id})` *(path)*	Identificador del nodo de información a consultar 1

Responses

Code	Description	Links
200	La llamada ha respondido correctamente. Media type application/json Controls `Accept` header. **Example Value** | Schema `{ "id": 1, "nombre": "Pedro", "telefono": 123456789, "puesto": "Recultador", "departamento": "Recursos Humanos", "usuarioId": 0 }`	*No links*
404	El nodo de información adicional no existe en la aplicación. Media type string **Example Value** | Schema `User data not found.`	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

GET `/userData/getUserData` Obtiene la información adicional de todos los usuarios registrados.

Se obtiene la información adicional de todos los usuarios registrados en la aplicación. REQUIERE PRIVILEGIOS DE ADMINISTRADOR.

Parameters

Try it out

Name	Description
Authorization * required `string($Bearer [token])` *(header)*	Token de autenticación Authorization

Responses

Code	Description	Links
200	La llamada ha respondido correctamente. Media type application/json Controls `Accept` header. **Example Value** | Schema `[ { "id": 1, "nombre": "Pedro", "telefono": 123456789, "puesto": "Recultador", "departamento": "Recursos Humanos", "usuarioId": 0 } ]`	*No links*
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

GET `/userData/authed` Obtiene la información adicional del usuario autenticado.

Se obtiene la información adicional del usuario autenticado en la aplicación.

Parameters Try it out

Name	Description
Authorization * required `string($Bearer [token])` *(header)*	Token de autenticación Authorization

Responses

Code	Description	Links
200	La llamada ha respondido correctamente.	*No links*

Code	Description	Links

Media type

application/json

Controls `Accept` header.

Example Value | Schema

```
{
  "id": 1,
  "nombre": "Pedro",
  "telefono": 123456789,
  "puesto": "Reculatador",
  "departamento": "Recursos Humanos",
  "usuarioId": 0
}
```

CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

DELETE `/userData/delete/{id}` Elimina un nodo de información adicional.

Se borra un nodo de información adicional de la aplicación. REQUIERE PRIVILEGIOS DE ADMINISTRADOR

Parameters | Try it out

Name	Description
Authorization * required `string($Bearer {token})` *(header)*	Token de autenticación Authorization
id * required `string(${id})` *(path)*	Identificador del nodo de información adicional a eliminar 1

Responses

Code	Description	Links
200	El nodo de información adicional se ha eliminado correctamente. Media type string Controls `Accept` header. **Example Value** | Schema `User data deleted.`	*No links*
404	No se ha encontrado el nodo de información adicional. Media type	*No links*

Code	Description	Links
	string **Example Value** | Schema `User data not found.`	
CONN_REFUSED	El servidor ha rechazado la conexión porque no se tiene autorización para acceder a esta llamada.	*No links*

Controlador de autorización

Este controlador se encarga de manejar todas las operaciones de inicio de sesión y registro.

POST `/auth/sudoRegister` Se da de alta un usuario administrador.

Se intenta registrar a un usuario administrador en el sistema.

Parameters | Try it out

No parameters

Request body required | application/json

Credenciales de registro del usuario.

Example Value | Schema

```
{
  "username": "usuarioDePrueba",
  "password": "userPassword",
  "email": "sudoEmail@gettasksdone.com"
}
```

Responses

Code	Description	Links
200	Se ha dado de alta al usuario. Se retorna el token JWT generado por el servidor. Media type: string Controls `Accept` header. Examples: Se proporciona el token de autenticación **Example Value** `eyJhbGciOiJIUzI1NiJ9.eyJpZCI6IjMiLCJyb2wiOiJVU1VBUklPIiwic3ViIjoiY2Rpc2lkb3IiLCJpYXQiOjE3MDkyMTk1MTMsImV4cCI6MTcwOTIyMDk1M30.-k-k_kTqTJ_B1zzNwRHO7vtEY6fD17M4CkN2cVLh-r4`	*No links*

Code	Description	Links
	Example Description Se proporciona el token de autenticación	
400	Ha ocurrido un error en la solicitud. Media type: text/plain Examples: El nombre de usuario ya está en uso dentro de la **Example Value** `The username already exists.` **Example Description** El nombre de usuario ya está en uso dentro de la aplicación.	*No links*
401	Ya se ha dado de alta un usuario administrador en el sistema, por lo que no pueden darse de alta más administradores. Media type: string Examples: Ya existe un usuario administrador en el sistema. **Example Value** `Unauthorized` **Example Description** Ya existe un usuario administrador en el sistema.	*No links*

POST `/auth/register` Se da de alta un usuario.

Se intenta registrar a un usuario en el sistema.

Parameters | Try it out

No parameters

Request body required | application/json

Credenciales de registro del usuario.

Example Value | Schema

```
{
  "username": "usuarioDePrueba",
  "password": "userPassword",
```

```
  "email": "userEmail@gettasksdone.com"
}
```


Responses

Code	Description	Links
200	Se ha dado de alta al usuario. Se retorna el token JWT generado por el servidor. Media type: string Controls `Accept` header. Examples: Se proporciona el token de autenticación **Example Value** `eyJhbGciOiJIUzI1NiJ9.eyJpZCI6IjMiLCJyb2wiOiJVU1VBUklPIiwic3ViIjoiY2Rpc2lkb3IiLCJpYXQiOjE3MDkyMTk1MTMsImV4cCI6MTcwOTIyMDk1M30.-k-k_kTqTJ_B1zzNwRHO7vtEY6fD17M4CkN2cVLh-r4` **Example Description** Se proporciona el token de autenticación	*No links*
400	Ha ocurrido un error en la solicitud. Media type: text/plain Examples: El nombre de usuario ya está en uso dentro de la **Example Value** `The username already exists.` **Example Description** El nombre de usuario ya está en uso dentro de la aplicación.	*No links*

POST `/auth/login` Inicia la sesión de un usuario.

Se intenta iniciar la sesión de un usuario en el sistema.

Parameters

Try it out

No parameters

Request body required

application/json

Credenciales de acceso del usuario.

Example Value | Schema

```
{
  "username": "usuarioDePrueba",
  "password": "userPassword"
}
```

Responses

Code	Description	Links
200	Se ha iniciado la sesión del usuario. Se retorna el token JWT generado por el servidor. Media type: string Controls `Accept` header. Examples: Se proporciona el token de autenticación **Example Value** `eyJhbGciOiJIUzI1NiJ9.eyJpZCI6IjMiLCJyb2wiOiJVU1VBUklPIiwic3ViIjoiY2Rpc2lkb3IiLCJpYXQiOjE3MDkyMTk1MTMsImV4cCI6MTcwOTIyMDk1M30.-k-k_kTqTJ_B1zzNwRHO7vtEY6fD17M4CkN2cVLh-r4` **Example Description** Se proporciona el token de autenticación	*No links*
403	Ha ocurrido un error en la solicitud. Media type: text/plain Examples: Las credenciales de acceso no son válidas **Example Value** `Invalid credentials.` **Example Description** Las credenciales de acceso no son válidas	*No links*

Controlador Raíz

GET `/api/ping` Hace un ping al servidor.

Se hace una llamada ping al servidor para comprobar que la conexión funciona correctamente, retornando Pong.

Parameters | Try it out

No parameters

Responses

Code	Description	Links
200	La llamada ha respondido correctamente. Media type */* Controls `Accept` header. **Example Value** | Schema Pong	*No links*

Schemas

GrantedAuthority {

```
description:    Obtiene el nivel de privilegios del usuario.

authority       string
}
```

InfoUsuario {

```
description:    Recurso para gestionar la información adicional del usuario.

id*             integer($int64)
                example: 1
usuario*        Usuario {...}
nombre*         string
                example: Pedro
telefono        integer($int64)
                example: 123456789
puesto*         string
                example: Reclutador
departamento*   string
                example: Recursos Humanos
}
```

Usuario {

description:	Recurso para la gestión de las credenciales de acceso en el sistema.
id*	integer($int64) *example: 1*
username*	string *example: usuarioPrueba*
email*	string *example: prueba@gettasksdone.com*
password*	string *example: contraseñaMuySegura1234*
rol*	string Nivel de privilegios del usuario. Enum: [USUARIO, ADMINISTRADOR]
authorities	[Obtiene el nivel de privilegios del usuario. GrantedAuthority {...}]
accountNonExpired	boolean Comprueba si la cuenta está caducada.
accountNonLocked	boolean Comprueba si la cuenta está bloqueada.
credentialsNonExpired	boolean Comprueba si las credenciales del usuario han caducado.
enabled	boolean Comprueba si el usuario está habilitado.

}

CheckItem {

description:	Recurso para los elementos de comprobación.
id*	integer($int64) *example: 1*
contenido*	string *example: Marcar esta casilla como completada*
esta_marcado*	boolean *example: true* Determina si este paso de tarea ha sido completado.
usuario*	Usuario {...}
tarea*	{...}

}

Contexto {

description:	Recurso para la gestión de contextos en la aplicación.
id*	integer($int64) *example: 1*
nombre*	string *example: Base de datos*
usuario*	Usuario {...}

}

Etiqueta {

description:	Recurso para la gestión de etiquetas en el sistema.
id*	integer($int64) *example: 1*
nombre*	string *example: Urgente*
usuario*	**Usuario** { ... }

}

Nota {

description:	Recurso para la gestión de notas dentro de la aplicación.
id*	integer($int64) *example: 1*
contenido*	string *example: Esto es una nota.*
creacion*	string($date-time)
usuario*	**Usuario** { ... }
proyecto	{ ... }
tarea	{ ... }

}

Proyecto {

description:	Recurso para gestionar los proyectos de la aplicación.
id*	integer($int64) *example: 1*
nombre*	string *example: Proyecto de pruebas*
inicio	string($date-time)
fin	string($date-time)
descripcion	string *example: Descripción del proyecto de pruebas.*
estado*	string **Estado actual del proyecto.**
usuario*	**Usuario** { ... }
tareas	[**Lista de tareas asignadas al proyecto.** { ... }]
notas	[**Lista de notas asignadas al proyecto.** **Nota** { ... }]
etiquetas	[**Lista de etiquetas asignadas al proyecto.** **Etiqueta** { ... }]

}

Tarea ⌄ {

description:	Recurso para la gestión de tareas en la aplicación.
id*	integer($int64) *example: 1*
contexto*	**Contexto** › {...}
titulo*	string *example: Tarea de prueba*
descripcion	string *example: Esta es una tarea de prueba.*
creacion	string($date-time)
vencimiento	string($date-time)
estado*	string Estado actual de la tarea.
prioridad*	integer($int32) Indica si la tarea es prioritaria (1) o no (0).
usuario*	**Usuario** › {...}
proyecto*	**Proyecto** › {...}
checkItems	⌄ [Lista de elementos de comprobación asignados a la tarea. **CheckItem** › {...}]
notas	⌄ [Lista de notas asignadas a la tarea. **Nota** › {...}]
etiquetas	⌄ [Lista de etiquetas asignadas a la tarea. **Etiqueta** › {...}]

}

RegisterRequest ⌄ {

description:	Recurso para la petición de registro en el sistema.
username*	string *example: usuarioDePrueba*
password*	string *example: contraseñaMuySegura*
email*	string *example: prueba@gettasksdone.com*

}

LoginRequest ⌄ {

description:	Contiene los parámetros necesarios para iniciar sesión.
username*	string *example: userGTD*
password*	string *example: contraseñaMuySegura1234*

}

InfoUsuarioDTO ⌄ {

description:	Recurso para enviar la información adicional de los usuarios.
id*	integer($int64) *example: 1*
nombre*	string *example: Pedro*
telefono	integer($int64) *example: 123456789*
puesto*	string *example: Recultador*
departamento*	string *example: Recursos Humanos*
usuarioId*	integer($int64) **Identificador del usuario propietario de esta información adicional.**

}

UserDTO ⌄ {

description:	Recurso para el envío de la información de acceso de un usuario
id*	integer($int64) *example: 1*
username*	string *example: usuarioPrueba*
email*	string *example: prueba@gettasksdone.com*
rol*	string *example: 0* **Nivel de privilegios** **Enum:** › Array [2]

}

CheckItemDTO ⌄ {

description:	Recurso para enviar los elementos de comprobación.
id*	integer($int64) *example: 1*
contenido*	string *example: Este es un elemento de comprobación.*
esta_marcado*	boolean *example: false*
tareaId*	integer($int64) **Identificador de la tarea asociada a este elemento de comprobación.**

}

ContextoDTO ⌄ {

description:	Recurso para enviar como respuesta la información de un contexto.
id*	integer($int64) *example: 1*
nombre*	string *example: Base de datos*

}

EtiquetaDTO ⌄ {

description:	Recurso para enviar como respuesta la información de una etiqueta
id*	integer($int64) *example: 1*
nombre*	string *example: Urgente*

}

NotaDTO ⌄ {

description:	Recurso para el envío de la información de una nota.
id*	integer($int64) *example: 1*
contenido*	string *example: Esto es una nota.*
creacion*	string($date-time)
tareaId*	integer($int64) Identificador de la tarea asociada a la nota.
proyectoId*	integer($int64) Identificador del proyecto asociado a la nota.

}

TareaDTO ⌄ {

description:	Recurso para el envío de información de una tarea.
id*	integer($int64) *example: 1*
titulo*	string *example: Tarea de prueba*
descripcion	string *example: Esta es la descripción de una tarea.*
creacion	string($date-time)
vencimiento	string($date-time)
estado*	string Estado actual de la tarea.
prioridad*	integer($int32) Indica si la tarea es prioritaria o no.
contexto*	ContextoDTO › {...}
checkItems	⌄ [Lista de elementos de comprobación de la tarea CheckItemDTO › {...}]
notas	⌄ [Lista de notas de la tarea NotaDTO › {...}]
etiquetas	⌄ [Lista de etiquetas de la tarea EtiquetaDTO › {...}]
proyectoId*	integer($int64) Identificador del proyecto asociado a la tarea.
contextoId*	integer($int64) Identificador del contexto asociado a la tarea.

}

ProyectoDTO ⌄ {

description:	Recurso para el envío de información de un proyecto.
id*	integer($int64) *example: 1*
nombre*	string *example: Proyecto de pruebas*
inicio	string($date-time)
fin	string($date-time)
descripcion	string Descripción del proyecto
estado*	string Estado actual del proyecto
tareas	⌄ [Lista de tareas del proyecto **TareaDTO** › {...}]
notas	⌄ [Lista de notas del proyecto **NotaDTO** › {...}]
etiquetas	⌄ [Lista de etiquetas del proyecto **EtiquetaDTO** › {...}]

}

Apéndice C

Instalación y configuración de la API y la base de datos

Uno de los componentes más importantes, si no el más importante para el funcionamiento de la aplicación, es el servidor y su base de datos, fundamentales para el procesamiento y persistencia de la información de la organización dentro de la aplicación. En este apartado se explicarán los requisitos mínimos necesarios para ejecutar correctamente el servidor y los pasos para poder instalarlo y configurarlo correctamente.

C.1. Requisitos del sistema

Como requisitos técnicos, la plataforma donde se vaya a instalar el servidor tiene los siguientes requisitos:

- Sistema operativo Linux de 64 bits (Ubuntu 16.04 o superior).
- Mínimo 4GB de RAM.
- Mínimo 10GB de espacio libre en el disco duro.
- Un procesador que soporte tecnologías de virtualización.

A su vez, se requiere tener los siguientes componentes en el sistema:

- Sistema de control de versiones Git.
- Plataforma de contenedores Docker.
- Se recomienda tener instalada la interfaz de usuario de Docker.
- Un certificado SSL/TLS para habilitar la comunicación por HTTPS (Se desaconseja rotundamente continuar con la instalación sin disponer de un certificado válido, aunque se proporcionarán instrucciones para configurar el servidor de modo que utilice el protocolo HTTP en vez de HTTPS en caso de no tener un certificado válido).

C.2. Proceso de instalación

C.2.1. Descargando y configurando el servidor

Para descargar el repositorio del servidor[14] se debe descargar de GitHub ejecutando el siguiente comando:

```
git clone https://github.com/gettasksdone/gettasksdone-server.git

```

Una vez descargado el código del servidor, hay que realizar varias configuraciones que se explicarán a continuación:

C.2.1.1. Configurando los contenedores

Para configurar el funcionamiento de los contenedores se debe modificar el fichero `docker-compose.dev.yml`. Para ello abriremos el fichero con nuestro editor de texto de preferencia:

```
services:
    server:
        build:
        context: .
        target: development
        ports:
            - "443:443" #Mapeo del puerto 443 del host con el puerto 443 del contenedor
        environment:
            - SERVER_PORT=443
            - MYSQL_URL=jdbc:mysql://db/getTasksDone #URL de la base de datos
        volumes:
            - ./:/app
        depends_on: #Relacion de dependencia con la base de datos, bajo la condicion de que el servicio este activo
            db:
                condition: service_healthy
    db:
        image: mysql:8.0
        ports:
            - "3306:3306"
        environment: #Credenciales de acceso a la base de datos
            - MYSQL_ROOT_PASSWORD=
            - MYSQL_ALLOW_EMPTY_PASSWORD=true
            - MYSQL_USER=getTasksDone
            - MYSQL_PASSWORD=superSecur3P4ssw0rd123
            - MYSQL_DATABASE=getTasksDone
        volumes:
            - db_data:/var/lib/mysql
            - db_config:/etc/mysql/conf.d
            - ./database:/docker-entrypoint-initdb.d
        healthcheck: #Definicion de la comprobacion de actividad del servicio
            test: ["CMD", "mysqladmin", "ping", "-h", "localhost"] #Comando a ejecutar para validar la actividad del servicio
```

```
32              interval: 10s #Intervalo en el que se ejecuta el
    comando
33              timeout: 5s #Tiempo de espera de respuesta del comando
34              retries: 5 #Numero de reintentos para la comprobacion
    de actividad
35              start_period: 5s #Tiempo que tarda en iniciarse la
    comprobacion
36 volumes:
37     db_data:
38     db_config:
39
```

En este fichero la configuración más relevante para editar es:

- `services/server/ports`: En esta configuración se asigna un puerto de la máquina host con un puerto del contenedor. En caso de querer configurar la aplicación en un puerto distinto al 8080, se debe modificar este campo. Por ejemplo, "443:443" para que el servidor funcione en el puerto estándar de HTTPS.
- `db/environment`: Esta sección de la configuración permite hacer ajustes en la configuración del servidor MySQL. Puede ser interesante modificar el usuario y contraseña de acceso (De hacerlo, también hay que añadir los cambios en el fichero `src/main/resources/application.properties`).
- `db/healthcheck`: Este apartado de la configuración permite efectuar la prueba de estado del servidor MySQL, y permitirá que una vez superada la prueba se arranque el servidor API. Aquí se pueden modificar los valores `interval`, `timeout`, `retries` y `start_period` como se desee.

C.2.1.2. Configurando el servidor API

Otro fichero esencial para la configuración de la API es el fichero denominado `application.properties` localizado en `src/main/resources`.

```
1  spring.jpa.hibernate.ddl-auto=update
2  spring.datasource.url=jdbc:mysql://db:3306/getTasksDone?
       serverTimezone=UTC
3  spring.datasource.username=<USUARIO DE LA BASE DE DATOS>
4  spring.datasource.password=<CONTRASENA DE LA BASE DE DATOS>
5  spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
6  spring.mvc.format.date-time=YYYY-MM-DD hh:mm:ss
7  logging.level.org.springframework.web=TRACE
8  #---Certificate Setup---
9  #Enables accept-only HTTPS
10 server.ssl.enabled=true
11 #Format of keystore (Recommended PKCS12)
12 server.ssl.key-store-type=PKCS12
13 #Path of the certificate
14 server.ssl.key-store=classpath:keystore/keystore.p12
15 #Password of the certificate
16 server.ssl.key-store-password=
17 #Alias of the certificate
18 server.ssl.key-alias=tomcat
```

```
19 spring.security.oauth2.client.registration.google.client-id=<ID DE
      CLIENTE OAUTH DE GOOGLE>
20 spring.security.oauth2.client.registration.google.client-secret=<
      SECRET DE CLIENTE OAUTH DE GOOGLE>
21
```

- `spring.datasource.username/password`: Estos parámetros almacenan las credenciales de acceso para el servidor MySQL. Deben ser idénticos a lo especificado en el fichero docker-compose.dev.yml.
- `server.ssl.enabled`: Este parámetro indica que se debe utilizar el protocolo HTTPS (true) o HTTP (false). Se debe recordar que para utilizar HTTPS se necesita de un certificado SSL/TLS válido. Si se desea utilizar el protocolo HTTP, se debe establecer este valor a false, aunque se desaconseja puesto que podría causar que la comunicación con el servidor sea interceptada y propensa a fugas de información.
- `server.ssl.key-store`: Este parámetro define la ruta donde está almacenado el certificado SSL/TLS. Se recomienda que se utilice un fichero con formato .p12, en caso de utilizar un formato distinto se debe modificar el campo server.ssl.key-store-type.
- `server.ssl.key-store-password`: Este parámetro debe contener la contraseña del certificado SSL/TLS en caso de requerir una contraseña.
- `server.ssl.key-store-alias`: Este parámetro debe contener el alias del certificado SSL/TLS.
- `spring.security.oauth2.client.registration.google`: Estos parámetros deben contener el ID de cliente y la clave secreta del cliente de OAuth de Google que se tenga configurado para establecer el inicio de sesión por SSO. Si no se hace uso del SSO de Google se puede modificar esta configuración por lo que proceda con el otro sistema, o dejar los campos vacíos en caso de no utilizar autenticación por SSO.

C.2.2. Ejecutando el servidor

Una vez realizadas todas las configuraciones pertinentes se puede ejecutar el proyecto. Para ello, en el terminal se debe acceder al directorio raíz del proyecto y buscar un script de ejecución llamado `prodRun.sh`, y ejecutarlo.

A continuación, se muestra un menú de 3 opciones, siendo de especial relevancia:

- `Iniciar servidor`: Ejecutar esta opción hará que se arranque el servidor. Si se desea realizar otra actividad en segundo plano mientras el servidor está activo se debería ejecutar la secuencia CTRL + A, y posteriormente pulsar la tecla d. Cuando se desee volver al servidor se utilizará el comando `screen -R`.
- `Restablecer servidor`: Lanzar esta acción requerirá una posterior confirmación. En esta opción se borrarán todos los datos relacionados con el servidor, impidiendo su recuperación.

```
Selecciona una opción:
1) Iniciar servidor
2) Restablecer servidor (ZONA DE PELIGRO)
0) Para salir
Introduce tu opción [0-2]:
```

Figura C.1: Ejecución del script ProdRun.sh

Para iniciar el servidor se escogerá la primera opción. La primera vez que se ejecute puede demorar en iniciar, dado que se deben descargar las dependencias y el contenedor MySQL.

```
[System] [MY-010931] [Server]
/usr/sbin/mysqld: ready for connections.
Version: '8.0.36'  socket: '/var/run/mysqld/mysqld.sock'
port: 3306  MySQL Community Server - GPL.
```

Figura C.2: Logs del contenedor SQL que indican que está listo

Una vez aparezca la salida de la figura C.2 en el terminal se asegura que el servidor MySQL está preparado para recibir conexiones, por lo que el servidor API podrá iniciar su ejecución libremente.

```
INFO 41 --- [  restartedMain]
c.gettasksdone.server.ServerApplication  :
Started ServerApplication in 5.262 seconds
(process running for 5.563)
```

Figura C.3: Logs del contenedor Spring Boot que indican que está listo

Y una vez se muestre la salida de la figura C.3, se considerará que la API ya permite recibir conexiones. Para probar que la API está funcionando correctamente, se puede acceder a la URL

`https://<NombreDeDominioDelServidor>:<Puerto>/api/ping`

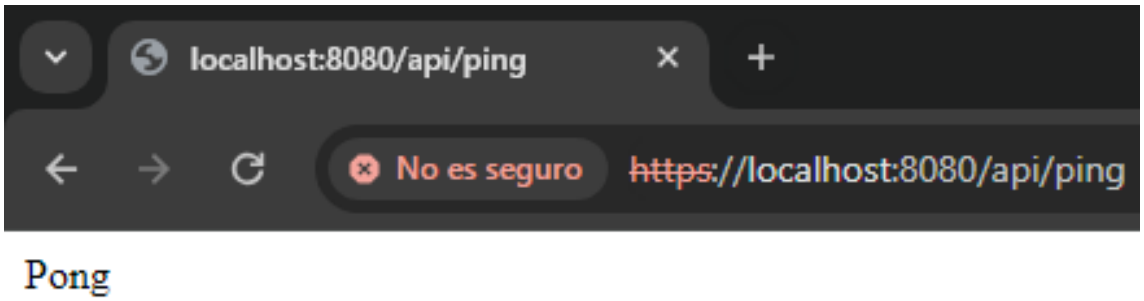

Figura C.4: Comprobación de ejecución correcta del servidor

Y si aparece una página respondiendo Pong, quiere decir que la API funciona correctamente. ¡Enhorabuena! Ya se puede proceder con la configuración e instalación de la aplicación web, que se detalla en el próximo capítulo.

Apéndice D

Instalación y configuración del cliente web

Este apéndice detalla una guía paso a paso para montar el cliente hecho con Flutter tanto en un sistema operativo Windows, macOS o Linux usando contenedores en Visual Studio Code (VSCode) o sobre una instalación previa de Flutter. Esta flexibilidad permite adaptarse tanto a entornos de desarrollo estandarizados como a configuraciones personalizadas.

D.1. Requisitos del sistema

Para la virtualización de Flutter en cualquier entorno de desarrollo, se recomienda considerar los siguientes requisitos del sistema:

- Un procesador con al menos 2 núcleos, aunque 4 núcleos son ideales para mejorar el rendimiento
- Memoria RAM mínima de 4 GB, preferiblemente 8 GB o más para manejar las tareas de desarrollo y los emuladores de manera eficiente.
- Un espacio mínimo en disco de 10 GB para alojar el sistema operativo, Flutter SDK, las dependencias del proyecto y herramientas adicionales.
- Un procesador con soporte de tecnologías de virtualización.

D.2. Proceso de despliegue del proyecto

El despliegue del proyecto del cliente web puede realizarse dentro de un contenedor Docker gestionado por VSCode o en un sistema donde Flutter ya esté instalado de forma nativa. Para desarrollar aplicaciones utilizando Flutter dentro del ecosistema de VSCode, se requiere configurar un ambiente de desarrollo integrado que permita la creación y gestión eficiente de código. Este proceso asegura un entorno replicable y consistente para el desarrollo. A continuación se detallan los pasos necesarios:

1. **Instalación de herramientas básicas**:

- **Visual Studio Code**: Instalar VSCode, necesario para gestionar el desarrollo y los contenedores.
- **Docker**: Instalar Docker, tecnología necesaria para el manejo de los contenedores de desarrollo.

2. **Instalación de extensiones**:

   - **Dev Containers**: Esta extensión para VSCode facilita el uso de contenedores Docker como entornos de desarrollo integrados, permitiendo abrir cualquier carpeta dentro de un contenedor y manejar sus configuraciones. Además permite tener una configuración estable y replicable para ejecutar los contenedores al instante.

D.2.1. Configuración del proyecto

1. **Clonación del repositorio**:

   - El código fuente del proyecto se encuentra alojado en un repositorio Git[15]. Este repositorio debe ser clonado en el sistema local para poder usarlo dentro del contenedor, y se clona ejecutando el siguiente comando:

```
1 git clone git@github.com:gettasksdone/GetTasksDone-web-client.
    git
2
```

2. **Inicialización del contenedor de desarrollo**:

   - Al abrir el repositorio clonado en VSCode, se debería mostrar un pop-up en la esquina inferior derecha con la opción “Abrir en contenedor de desarrollo”. Al seleccionar esta opción, VSCode prepara un contenedor Docker para el entorno de desarrollo específico del proyecto, lo cual puede tardar algunos minutos.

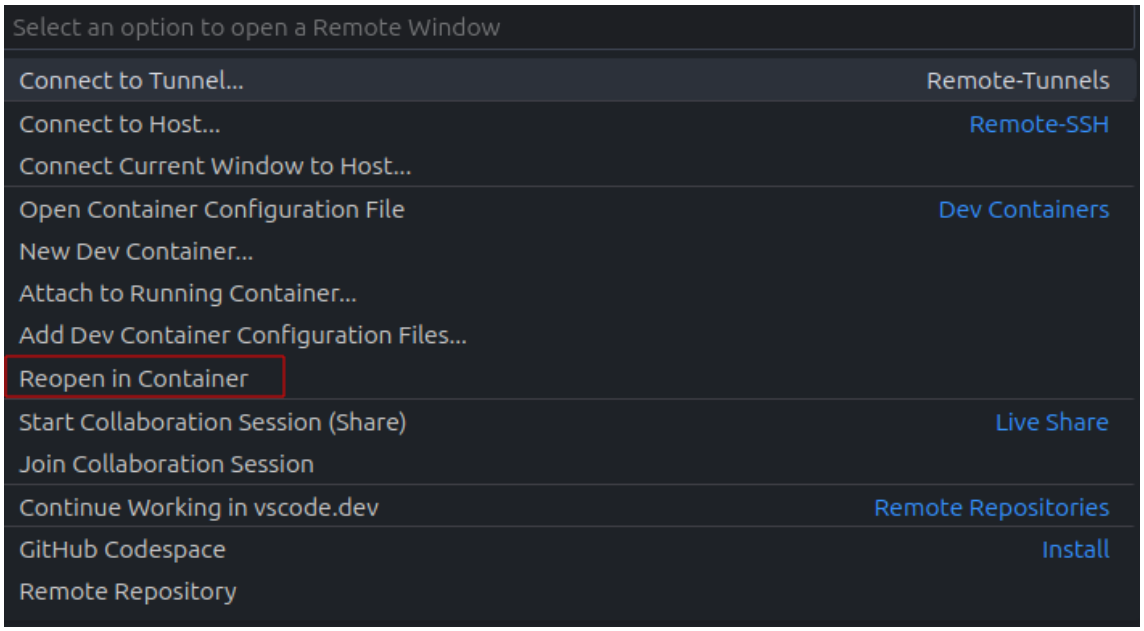

Figura D.1: Reabrimos proyecto en contenedor de desarrollo

Como vemos en la figura D.1 se muestra la opción `Reopen in Container` para lanzar el contenedor de desarrollo en VSCode.

D.2.2. Inicialización del proyecto Flutter

1. **Configuración del proyecto**:
   - Una vez inicializado el contenedor de desarrollo, es necesario abrir una terminal dentro de VSCode y ejecutar los siguientes comandos:
     - `flutter pub get`: Este comando recupera las dependencias del proyecto especificadas en el archivo `pubspec.yaml`.
     - `flutter pub upgrade`: Este comando actualiza las librerías de Flutter a las últimas versiones compatibles, asegurando que el proyecto utilice las funcionalidades más recientes y estables.

D.2.3. Verificación y trabajo continuo

1. **Finalización de la configuración**:
   - Con todas las dependencias instaladas y actualizadas, el contenedor de desarrollo está ahora completamente configurado y listo para ser usado en el desarrollo del proyecto Flutter.
2. **Lanzado del cliente web**:
   - Una vez se ha finalizado el proceso de configuración, la aplicación web se puede lanzar en modo depurador en un puerto local con el siguiente comando:

```
flutter run -d web-server --web-port <PUERTO>

```

3. **Desarrollo continuo**:

   - El entorno de desarrollo configurado permite ahora un desarrollo fluido y consistente, facilitando la colaboración y el mantenimiento del código a través de diferentes entornos y máquinas.