Implementación de una infraestructura REST con privacidad para GTD basada en Express y React Native

TRABAJO FIN DE GRADO

Pablo Gamo González, Carlos Gómez López, Javier Gil Caballero, Alejandro Del Río Caballero

Dirigido por: Juan Carlos Sáez Alcaide

Grado en Ingeniería del Software y Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid
Curso 2023-2024

Implementación de una infraestructura REST con privacidad para GTD basada en Express y React Native

Memoria de Trabajo Fin de Grado

Pablo Gamo González, Carlos Gómez López, Javier Gil Caballero, Alejandro Del Río Caballero

Dirigido por: Juan Carlos Sáez Alcaide

Grado en Ingeniería del Software y Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid
Curso 2023-2024

Resumen

La metodología de productividad personal conocida como "*Getting Things Done*" (*GTD*), creada por David Allen, es uno de los métodos más efectivos para la organización de tareas en la actualidad. Su objetivo es maximizar la productividad a través de la consolidación de todas las tareas, proyectos y actividades en un solo lugar. Aunque existen muchas aplicaciones disponibles para ayudar a poner en práctica la filosofía *GTD*, la mayoría son propiedad de empresas que pueden tener acceso a la información personal de los usuarios, lo que puede violar su privacidad.

Para solventar este problema en este proyecto se ha desarrollado una infraestructura formada por una base de datos, un servicio *REST*, y un conjunto de clientes para entornos de escritorio y móvil. En el proyecto se ha empleado la tecnología *React Native* para el desarrollo de clientes multiplataforma. Algunos aspectos clave de la infraestructura desarrollada son su *modo offline*, para permitir el funcionamiento de los clientes incluso sin conexión a *Internet*, y el empleo de éstandares de autorización como *OAuth* para dotar de seguridad a la *API REST*. Para la implementación de esta *API* se ha hecho uso del framework *Express.js* que ha simplificado sustancialmente el desarrollo. Asimismo, también se han explorado métodos para interactuar con el servicio *REST GTD* desde asistentes conversacionales.

palabras clave: GTD, React Native, Express.js, REST API, OAuth 2.0, Multiplataforma

Autorización de difusión y utilización

Los abajo firmantes, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Grado "Implementación de una infraestructura REST con privacidad para GTD basada en Express y React Native", realizado durante el curso académico 2023-2024 bajo la dirección de Juan Carlos Sáez Alcaide en el Departamento de Arquitectura de Computadores y Automática, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Pablo Gamo González, Carlos Gómez López, Javier Gil Caballero, Alejandro Del Río Caballero

Juan Carlos Sáez Alcaide

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Introduction

1.1 Motivation

Immersed in the digital age, characterized by the culture of immediacy, it's not an easy task to keep focus and avoid distractions in the middle of a maelstorm of information and stimuli. The *GTD* (Getting Things Done) methodology aims to help people perform their daily tasks in a way that they do not depend on their memory and focus on the now, without being aware of future tasks. This method was created by David Allen, who collected it in his book entitled "Getting Things Done" [1] and was translated into Spanish as "Organízate con eficacia".

Likewise, the *GTD* system is known for its effectiveness both personally and professionally in planning, organizing and managing tasks. However, some of the apps that implement it have several problems. First of all, many of these applications do not clearly inform about the process of processing their users' data. (e.g., the data is hosted on third-party servers and their security protocols are unknown).

On the other hand, many of these apps have a high learning curve or are compatible with a small number of operating systems. A clear example of this is Things [2] which, despite being a very good application, is only available for Apple devices. This conditions the access of this methodology to a more general public.

Therefore, motivated to solve this problem, we seek to develop a multiplatform application that, in addition to implementing the *GTD* methodology, stands out for its intuitive and friendly interface, guided by a *REST* architecture, with the aim of allowing the user to have control of their data, guaranteeing their privacy and transparency. This application, called *SwiftDo*, aims to offer an accessible and versatile alternative for the efficient management of daily tasks.

1.2 Goals

The main objective of this project is to develop a cross-platform *GTD* application. To do this, we have used the *React Native* framework that allows us to create a consistent user interface that works on all major operating systems, such as *Android*, *iOS*, *MacOS*, *GNU/Linux* and *Windows*, and ensures that users can manage their tasks and projects efficiently from any device, no matter what platform they use.

The use of *React Native* simplifies the development, manteinance, and scalability of the project, resulting in an agile application that is adaptable to future updates and changes in the target platforms.

SwiftDo stands out for its offline mode, which allows users to manage tasks even without an internet connection (as can be the case on mobile devices). Unlike other alternatives, our app ensures an uninterrupted experience by storing data locally and automatically synchronizing with the server when the connection is restored, ensuring the constant availability of information on all the user's devices.

In the development of *SwiftDo*, the *REST API* plays a crucial role as it provides endpoints for the client-server communication, enabling CRUD operations on data, such as having an *endpoint* '/tareas' for creating and reading tasks. In this way, the *REST API* provides a standardized and efficient interface for data manipulation on out *GTD* platform. In addition, implementing solid security practices, such as authentication and data encryption, ensures the integrity and confidenciality of information, guaranteeing a secure experience for our users.

When developing our application, we pursue a series of specific objectives that were represented by the following high-level requirements.

- **Centralized Task Management:**

The application should allow users to create new tasks which, by default, will be added to the *Inbox* (unorganized task store within the *GTD* methodology). From there, users will be able to assign tags to them, link them to specific projects or areas, and move them between different sections.

- **Flexibility in the Organization:**

Users should have the ability to organize their tasks flexibly, adding them to relevant projects or areas, and assigning them labels for more detailed classification. In addition, the app should allow editing and elimination of tasks as needed.

- **Tracking & Prioritization:**

  The app should provide a number of functionalities for tracking and prioritizing tasks efficiently and effectively. Users must have the ability to complete tasks and assign them relative importance. In addition, the app should provide access to detailed information about each task, including its current status, due date, and

any associated notes. It's crucial that tasks are automatically prioritized based on their relevance to the user.

- **Security:**

SwiftDo must implement a robust authentication and authorization system to ensure that only authorized users can access and manipulate data via the *REST API*. In addition, end-to-end encryption is required to protect the confidentiality of information during its transmission and storage, ensuring a secure experience for all users.

- **Search & Filtering Features:**

Users should be able to search and filter their tasks based on tags, areas, projects, or other relevant criteria, making it easier to manage and view tasks.

- **Usability:**

The app provides an offline mode that allows offline tasks to be managed by storing data locally on the user's device and ensuring continuity of work regardless of online availability.

- **Integration with conversational agents:**

The app integrates with conversational agents such as *Alexa* offering a more versatile experience, allowing users to interact with the app using voice commands, further simplifying task entry and management intuitively and effortlessly.

These requirements provide a solid foundation for the development of the *SwiftDo* application, ensuring a comprehensive experience that complies with the fundamental principles of the *GTD* methodology and meets the needs of users in the effective management of their tasks and projects.

1.2.1 System Architecture

Figure 1.1 shows the block diagram of the system, which is composed of 2 main components: a client application, essentially made for various devices, and a backend which has a REST API with a secure authorization system.

The client application is implemented with the cross-platform framework *React Native*, and can be executed on various devices. Clients interact via *REST* over *HTTPS* with the backend.

The backend implements 2 main components and includes a database. On the one hand, there is the *OAuth 2.0* module, which is responsible for managing the authentication and authorization flow, that is, it manages the access of the users of the client applications to the information and services provided by the backend. On the other hand, the backend is also composed of an API that follows the *REST* architecture and implements and exposes the different services of the application through various endpoints. Finally, the

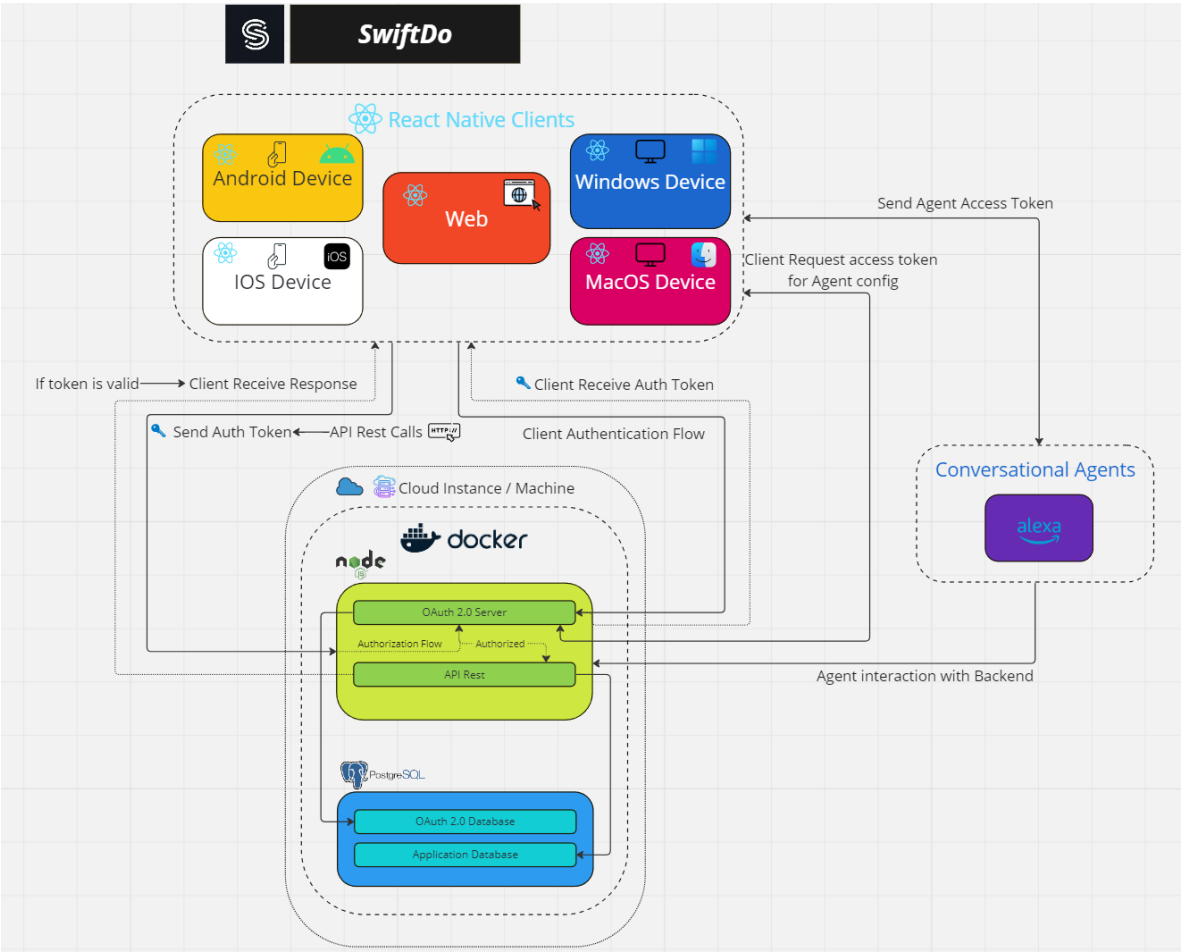

Figura 1.1: System Block Diagram

backend also contains the application's database, which contains both the tables used by the *API REST* module and the *OAuth 2.0* module. All the services contained in the backend are managed through *Docker* containers, so it is possible to start, connect and configure the various modules in a simple and agile way, in order to be able to deploy these services on any machine without the need for further configuration specific to each environment.

Finally, the system contains a third component which allows you to connect some conversational agents with the application. From the clients, it is possible to carry out this configuration of agents by generating a special key for them, so that the agents, once configured, can access the services of the backend through voice commands.

1.3 Work Plan

1.3.1 Tasks

To carry out the development of the project, we have divided the tasks to be carried out into several phases that are discussed below.

First of all, and individually, we carry out a search by comparing our project model with other existing applications on the market and then sharing the different ideas. Each member of the team installed and analyzed one of these applications and pointed out the possible improvements that we could implement to add more value to our product. Thanks to this exercise, we reached many of the conclusions explained in 1.1.

Subsequently, there was a design phase in which we made a mockup of what we would like our application to look like.

Next, we had a learning phase in which we searched for information and experimented with the different tools and components needed to build our application. Among these components, we sought to familiarize ourselves with *Express*, *Docker*, *AWS*, *React Native*, *firebase* among others to make a comparison and choose the technologies that best suited our needs for the future development of our application.

Once we finished the various pre-development tasks discussed above, we started with the programming of the backend which took us approximately four months. During this time we were implementing the main services of the application in addition to the authentication and authorization system. Once the essential part of the backend was finished, we started with the development of the frontend, which has been the most costly phase in terms of time and effort, since we have dedicated a lot of work to create a friendly and usable interface as well as extensive functionality (filtering, searching, editing…). That is why we have been making the necessary updates to the backend for the correct functioning of the application.

1.3.2 Time planning

The figures 1.2 and 1.3 show the tasks associated with the development of the backend and frontend respectively, as well as the start date, end date, and duration of each task:

Backend

ID	Task name	Start	Finish	Duration
1	Despliegue AWS	26/10/2023	15/11/2023	22.0 d.
2	Crear BBDD	28/10/2023	4/11/2023	9.0 d.
3	Esquema CRUD Projects	18/11/2023	5/12/2023	18.0 d.
4	CRUD Tasks	18/11/2023	26/12/2023	39.0 d.
5	CRUD Context	18/11/2023	26/12/2023	39.0 d.
6	CRUD Tags	18/11/2023	26/12/2023	39.0 d.
7	Validaciones	19/12/2023	25/1/2024	38.0 d.
8	Servicios	27/12/2023	23/1/2024	28.0 d.
9	Routes	27/12/2023	23/1/2024	28.0 d.
10	Actualizar username sidebar	25/1/2024	1/2/2024	8.0 d.
11	Actualizar números sidebar	25/1/2024	1/2/2024	8.0 d.
12	Añadir ajustes sidebar	1/2/2024	8/2/2024	8.0 d.

Figura 1.2: Backend Gantt chart

Frontend

1.3.3 Memory Structure

The rest of the report is organized as follows:

- In **chapter 2**, an analysis is made of what GTD is, its method and technique. It also describes the several applications in the market that implement *GTD*.
- **Chapter 3** discusses project planning, covering points such as the version control system, the development and integration environments used, and the deployment system.
- In **chapter 4**, we introduce the data model and the implementation of the database. This describes the entities, the physical model of the database, its performance and scalability, and the security of the database.
- **Chapter 5** introduces the design and implementation of the backend, explaining our use of *REST*, *API* design, application endpoints, *API* security aspects, and *OAuth 2.0* implementation.
- **Chapter 6** details the design principles that have guided the creation of the app, along with the different technologies used in its development, from the initial prototyping stage to its implementation, exploring how the fundamentals of the *GTD* method have been implemented in *SwiftDo*.

ID	Task name	Start	Finish	Duration
1	Project Service	26/12/2023	1/2/2024	38.0 d.
2	Diseño Sidebar	27/12/2023	1/2/2024	38.0 d.
3	Diseño Programadas	26/12/2023	4/2/2024	41.0 d.
4	Cuanto antes	26/12/2023	4/2/2024	41.0 d.
5	Diseño Inbox	26/12/2023	4/2/2024	41.0 d.
6	Service Task	26/12/2023	4/2/2024	41.0 d.
7	Diseño Project	21/1/2024	1/2/2024	12.0 d.
8	Inicio de sesión	29/1/2024	5/2/2024	7.0 d.
9	Sidebar proyectos actualizados	4/2/2024	11/2/2024	7.0 d.
10	Adaptar Vista proyecto con Actions Screen	4/2/2024	12/2/2024	9.0 d.
11	Crear Context + Listar Context	4/2/2024	13/2/2024	10.0 d.
12	Editar Proyecto	4/2/2024	19/2/2024	16.0 d.
13	Crear Proyecto	7/2/2024	14/2/2024	8.0 d.
14	Área y números sidebar	14/2/2024	21/2/2024	8.0 d.
15	Estado de proyectos y action	14/2/2024	24/2/2024	11.0 d.
16	Modo oscuro	14/2/2024	11/3/2024	27.0 d.
17	Desarrollo Etiquetas	28/2/2024	11/3/2024	13.0 d.
18	Versión escritorio	28/2/2024	20/3/2024	22.0 d.
19	Filtrado context/etiquetas	28/2/2024	11/4/2024	43.0 d.
20	Administrar contenido modal etiquetas	10/3/2024	17/3/2024	7.0 d.
21	Previsualización markdown	11/4/2024	17/4/2024	7.0 d.
22	Pantalla hoy	11/4/2024	6/5/2024	26.0 d.
23	Implementación modal settings	11/4/2024	8/5/2024	28.0 d.

Figura 1.3: Frontend Gantt chart

- **Chapter 7** describes the integration of our *GTD* app with conversational agents, in our case with *Alexa*. We'll explain how we set up the *Alexa* skill, how we linked our user account, the implementation of the *OAuth 2.0* authorization flow, and the issues we encountered during this process.
- In **chapter 8** you will find the user manual, showing the full functionality of our application in an easy and intuitive way.
- **Chapter 9** sets out the main conclusions we have reached and discusses future lines of work.
- This report also consists of four appendices. The first two (Appendices A and B) correspond to the English translation of the introduction and conclusions, the third (Appendix C) lists the contributions that each team member has made to the project, and the last one, (Appendix D) provides a deployment guide so that anyone who wants to use our application can do so.

Capítulo 2

Contribución de cada miembro del equipo

Desde el inicio del curso escolar 23/24 y con ello, desde el inicio de este trabajo de fin de grado, cada miembro del grupo ha ido aportando una serie de características que han sido cruciales para el desarollo de la aplicación. A continuación se precisa la contribución de cada integrante a las fases del proyecto, destacando cómo la coordinación grupal ha permitido el objetivo final.

2.1 Alejandro del Río Caballero

2.1.1 Puesta en escena: Aprendizaje

Tras la primera reunión con el tutor y con los objetivos estipulados, Alejandro asumió la responsabilidad inicial de familiarizarse con la metodología *GTD* con "THE MAIN IDEA" [3], un resumen de apenas 12 páginas donde explicaban los fundamentos de *GTD*. De esta forma, comenzó a escribir la motivación en el Capítulo **??**.

Por otro lado, Alejandro se documentó sobre las tecnologías que íbamos a utilizar, tales como *Node.js* y *React Native* para el desarrollo de *backend* y *frontend*. Además de Figma para el diseño de interfaces de usuario.

2.1.2 Documentación de la metodología

Una vez conocida la metodología y las tecnologías que íbamos a utilizar, Alejandro, con el objetivo de analizar la competencia a la que nos enfrentábamos, investigó las aplicaciones más populares de la *AppStore y Google Play Store* que implementan *GTD*, sacando sus ventajas y desventajas. De esta forma, pudimos comprender los puntos fuertes que queríamos destacar y los puntos débiles que debíamos evitar para el desarrollo de *SwiftDo*.

Por otro lado, al ver las aplicaciones más populares, Alejandro leyó el libro de David Allen, [1], pudiendo retener el conocimiento para comunicárselo a los demás integrantes del equipo. Este estudio permitió enfatizar los aspectos y características que debían estar en el diseño final de la aplicación, además de poder utilizarlo para poder escribir el Capítulo **??**.

2.1.3 Diseño: Desarrollo de la Interfaz de Usuario (UI)

El diseño de la interfaz de usuario fue hecha por Alejandro, asegurándose que el diseño fuera atractivo y coherente con la metodología. Para ello utilizó Figma como herramienta de diseño lo que facilitó la creación de los prototipos y componentes que conforman *SwiftDo*.

Para desarrollar la interfaz de usuario, Alejandro se guió por los principios de diseño que establece Donald Norman en su libro [4], y en el diseño de aplicaciones ya exitosas como Things [2], dado su enfoque minimalista. De esta forma permitimos un diseño centrado en el usuario, permitiendo centrarse en las acciones del método *GTD*, eliminando las distracciones y facilitando la gestión de tareas.

El uso de los principios de diseño, sumado a la capacidad de identificar y aplicar características ya exitosas de otras aplicaciones, contribuyó a que Alejandro pudiese escribir el Capítulo **??**.

2.1.4 Desarrollo

Backend

Al inicio de la aplicación nos centramos fundamentalmente en el desarrollo del *backend*, concretamente en las acciones *CRUD*, "Crear, Leer, Actualizar y Borrar" (del inglés: *Create, Read, Update and Delete*). Una vez repartidas las *CRUD* mas relevantes de la aplicación entre todos los integrantes del grupo, Alejandro se encargó de la sección de Proyectos.

Para ello, creó un "validador de datos" a modo de esquema en un archivo "projectValidator.js", asegurándose que los campos "título" y "descripción" de un proyecto se crearan y modificaran correctamente en la base de datos. Luego, implementó las rutas necesarias mediante el framework de *Node.js* y *Express*, manejando así las solicitudes HTTP correspondientes para crear, completar, mostrar y actualizar los campos de un proyecto. Así como la adición de los códigos de estado HTTP para notificar al usuario en caso de error.

Para llevar a cabo los aspectos de *backend*, Alejandro aprendió las bases de *Node.js y Express.js* en un curso de freeCodeCamp [5] , ya que la titulación de Ingeniería Informática carece de asignaturas en las que se imparta dicho *framework* en profundidad. También aprendió mediante el mismo curso el funcionamiento de Postman, para probar

y verificar la API que íbamos creando.

Frontend

En cuanto al *frontend*, hemos mantenido un desarrollo consistente y cohesionado gracias a la participación activa de todos los integrantes del equipo en todos las áreas de la aplicación.

En un primer lugar, Alejandro contribuyó en gran parte al diseño e implementación de componentes reutilizables como las pantallas principales, así como con la navegación de la aplicación, reduciendo significativamente el número de archivos creados y el número de líneas de código por archivo.

Uno de los componentes principales de los que se encargó Alejandro fue de la *sidebar*, proporcionando una navegación rápida a diversas secciones y funcionalidades de la aplicación. Ya que como vimos en el Capítulo **??** está cargada de varios elementos, tales como el nombre de usuario en la parte superior de la misma (subcomponente que se encargó de implementar Carlos Gómez), las acciones de *GTD* explicadas en el Capítulo **??** junto al número de tareas asociadas, además de los proyectos creados por el usuario (subcomponente que se encargo Alejandro de forma conjunta con Javier Gil) y la representación del porcentaje de completitud de estos, así como un archivo a modo de esquema reutilizable que utilizarían todas las pantallas que implementan las acciones de *GTD*.

En cuanto a otros aspectos, también se encargó de implementar las interfaces de inicio y registro de sesión, el rediseño e implementación al completar tareas, así como la pantalla para agregar detalles en las tareas y la previsualización de dichos detalles en *markdown*. Además de adaptar los estilos de la aplicación para hacerlo compatible con el modo oscuro.

Para llevar a cabo el desarrollo de la aplicación en *React Native*, igualmente se documentó a través del libro [6], además de la documentación de la página oficial de *React Native* [7] y *React Navigation* [8], además de realizar un curso en FreeCodeCamp [9] para entender el funcionamiento de *React Native y Expo* .

2.1.5 Conclusiones

La participación de cada miembro del grupo ha sido fundamental para el desarrollo de la aplicación. Demostrando la cohesión y consistencia de los conocimientos adquiridos tanto en ingeniería de software como en informática, así como la importancia del trabajo en equipo, pudiendo destacar las habilidades y destrezas adquiridas.

2.2 Carlos Gómez López

2.2.1 Puesta en escena: Aprendizaje

El desarrollo de *SwiftDo*, nuestra aplicación basada en la metodología *GTD*, comenzó con una fase de búsqueda y familiarización con esta metodología. Nos sumergimos en la literatura y recursos disponibles sobre *GTD* para entender mejor sus principios y cómo podríamos aplicarlos a nuestra aplicación. Además, nos inspiramos en otras aplicaciones existentes que implementan la metodología *GTD*, lo que ayudó a visualizar cómo estructurar las funcionalidades y la interfaz de usuario. Durante esta etapa, surgieron muchas dudas sobre la arquitectura de la aplicación y las "acciones" específicas que necesitábamos implementar. Estas dudas las fuimos resolviendo con la orientación de nuestro profesor y mediante investigaciones exhaustivas.

Paralelamente, dedicamos tiempo a investigar las tecnologías que utilizaríamos para el desarrollo. Decidimos optar por *React Native* y *Express.js*. Aunque algunos ya teníamos experiencia previa con *Express.js* de una asignatura en el grado universitario, *React Native* representó un nuevo desafío. A pesar de que algunos miembros del equipo tenían conocimientos de *React*, nos dimos cuenta de que *React Native* presentaba diferencias significativas y requería un enfoque de estudio y aprendizaje específico. Esta fase inicial fue crucial para sentar las bases técnicas y conceptuales de *SwiftDo* y preparar a nuestro equipo para el entorno de desarrollo.

2.2.2 Documentación

Después de familiarizarnos con la metodología *GTD* y las tecnologías que íbamos a utilizar, nos enfocamos en identificar aplicaciones similares a las que queríamos desarrollar, con especial atención en aquellas que también implementaban la metodología *GTD*. Para llevar a cabo esta investigación, dividimos la tareas entre los miembros del equipo, asignando a cada uno la responsabilidad de analizar diferentes aplicaciones.

Durante esta fase, recopilamos información sobre las ventajas y desventajas de cada aplicación, lo que nos permitió desarrollar una visión clara de qué características eran esenciales para nuestra aplicación. A partir de este análisis, obtuvimos ideas tanto de funcionalidades como diseño de la interfaz de usuario, que consideramos importantes integrar en *SwiftDo*.

Paralelamente, dado que una de nuestras metas era integrar nuestra aplicación con agentes conversacionales, Carlos se encargó de investigar las opciones disponibles para esta integración. Exploró la viabilidad de trabajar con agentes como *Alexa* y *Google Assistant*, evaluando cuál de ellos ofrecía mejores características y facilidades para su integración.

2.2.3 Diseño: Desarrollo de la Interfaz de Usuario (UI)

Para desarrollar la interfaz de usuario de *SwiftDo*, Alejandro y Carlos lideraron respectivamente el diseño para dispositivos móviles y web/aplicación. Si bien se evaluaron ambas propuestas, se destacaron y se incorporaron más ideas provenientes del diseño de Alejandro. La colaboración resultó en una interfaz equilibrada que reflejaba la visión del equipo y satisfacía las necesidades de los usuarios.

2.2.4 Desarrollo

Base de datos

En el desarrollo de la base de datos, Javier asumió principalmente la responsabilidad, aunque Carlos realizó ajustes específicos en las relaciones entre las entidades "Tasks" y "Tags", incluyendo la creación de una tabla nueva llamada "TagToTask".

Backend

En el desarrollo del *backend*, Carlos asumió varias responsabilidades. Inicialmente, se encargó de la configuración de los *routes* y *services* de la entidad "Task", así como en la implementación del *CRUD* para esta entidad. Además, desarrolló la funcionalidad para la creación de áreas/contextos y un buscador de etiquetas, tanto específicas como para obtener todas.

Carlos también realizó modificaciones significativas en la sección de filtros previamente creada por Pablo, adaptándola para generar filtros específicos para la acción "Hoy". Por último, cabe mencionar su contribución en la personalización del diseño predeterminado de *OAuth* para la integración con *Alexa*.

Frontend

En el *frontend*, Carlos lideró una serie de cambios y diseños significativos. En la sección de "Inbox", se enfocó en desarrollar la interfaz de usuario, integrando funciones clave como la adición y edición de tareas. Diseñó varios modales, incluyendo los de adición, edición, completado, movimiento y selección de tareas, así como el selector entre tarea o proyecto. También implementó la integración de etiquetas en las tareas, contando con la colaboración de Javier.

Para la mejora de la experiencia del usuario, se llevó a cabo la introducción de funcionalidades de desplazamiento y la capacidad de seleccionar tareas. Además, Carlos se encargó de la visualización de etiquetas y la importancia de las tareas en las vistas de las "acciones", lo que proporcionó una mejora significativa en esta sección de la aplicación.

En el "Sidebar", Carlos diseñó el perfil, la funcionalidad para añadir y listar "áreas", y los indicadores de las tareas disponibles en cada "acción".

Carlos trabajó en el diseño la sección “Hoy” junto con Javier, aunque las funcionalidades las desarrolló de manera individual. Esta parte de la aplicación resultó ser una de las más tediosas debido a la complejidad de las adaptaciones necesarias para garantizar el funcionamiento como en el resto de la aplicación.

Además, Carlos también realizó adaptaciones para web y escritorio, y se ocupó de implementar el apartado de “Tutorial” y “Vincular con Alexa” en el menú de configuración.

Finalmente, se dedicó a realizar ajustes finos y correcciones de errores en toda la aplicación.

2.2.5 Vinculación con Alexa

Carlos fue el encargado de desarrollar la skill de *SwiftDo* desde cero para la integración con Alexa, partiendo de una plantilla base proporcionada por Amazon Developer. En esta fase, el hecho de que la plantilla base ofreciera la opción de elegir entre Node.js y Python facilitó el proceso para Carlos, ya que contaba con conocimientos sólidos en Node.js. Después de estudiar en solitario el funcionamiento del agente conversacional, enfrentó los desafíos técnicos que surgieron en el desarrollo de la skill. Una vez que logró tener todo funcional, solicitó la ayuda de Pablo para poner en marcha la integración, ya que requería de sus conocimientos en OAuth. Tras un arduo trabajo y pruebas exhaustivas, lograron la vinculación y el correcto funcionamiento de la skill, realizando pruebas Carlos desde su propio dispositivo Alexa.

2.2.6 Memoria

La redacción de la memoria se organizó distribuyendo el trabajo en capítulos, a excepción de los capitulos de “Introducción” y “Conclusiones”. Una vez terminado cada capítulo, este era revisado por los miembros del equipo, tras lo cual se agendaba una reunión con el profesor para obtener su *feedback* y realizar las correcciones necesarias.

Para la “Introducción”, se siguieron las indicaciones del profesor y Carlos se encargó de redactar la sección de “Objetivos” (sección **??**). Además, Carlos fue responsable de los capítulos sobre el “Modelo de datos e implementación de la base de datos” (capítulo **??**), así como de la “Integración con agentes conversacionales” (capítulo **??**).

El capítulo de “Conclusiones” se elaboró mediante una sesión de *brainstorming*, donde discutimos qué aspectos destacar y cómo reflejar mejor nuestras experiencias y aprendizajes. Finalmente, tras completar todos los capítulos, se llevó a cabo una revisión conjunta del documento completo antes de la entrega final.

2.2.7 Conclusiones

La experiencia de llevar a cabo este proyecto nos ha brindado valiosas oportunidades para crecer y mejorar nuestras habilidades en diversas áreas, desde la programación hasta

el diseño de interfaces de usuario y la gestión de proyectos. Afrontar los obstáculos nos ha permitido desarrollar soluciones prácticas y ganar confianza en nuestras capacidades técnicas y de trabajo en equipo.

2.3 Pablo Gamo González

2.3.1 Puesta en escena: Aprendizaje

Al igual que el resto de los compañeros el primer paso que dió Pablo para empezar el proyecto fue informarse bien sobre la metodología *GTD*. Para ello investigó por internet acerca de los principios de este método para entender su funcionamiento. También realizó un proceso de búsqueda de aplicaciones similares, que aunque no implementaran la metodología, fueran de organización personal. El fin de está búsqueda fue preguntarnos qué podría aportar o mejorar nuestra aplicación, respecto a otras existentes.

Una vez que tuvimos claros los conceptos, llegó el momento de definir las tecnologías a utilizar para el desarrollo del proyecto. Desde que acordamos realizar este trabajo con el tutor, en las primeras reuniones Pablo fue el principal defensor de utilizar la tecnología *React Native* para la implementación de los clientes ya que tenía experiencia previa con ella y sabía del potencial que tenía. Por esta razón, una vez empezado el trabajo llegamos a la conclusión de usar esta tecnología. En cuanto a las tecnologías con las que se ha implementado el *backend*, también propuso utilizar *Node.js* y *Express.js* ya que varios de nosotros teníamos ya experiencia en ellas. Dada la previa experiencia de Pablo con las tecnologías mencionadas, no tuvo que familiarizarse con ellas, pero sí realizó un proceso de búsqueda de posibles soluciones para la aplicación, donde pudo ir aprendiendo sobre paquetes tanto de *backend* como de *frontend* que fueron de utilidad en el desarrollo.

2.3.2 Documentación

Sobre el trabajo realizado en el desarrollo de la memoria, Pablo se ha encargado de documentar el capítulo del *backend* en su totalidad, ya que es la persona que más conocimiento tenía sobre las tecnologías así como sobre el estándar *OAuth 2.0*. También ha realizado gran parte del capítulo de planificación, sobre todo la parte en la que se habla del sistema de despliegue ya que fue él quien desarrolló dicho sistema. Además ha documentado completamente los apéndices de la guía de despliegue y los *endpoints*. En cuanto la introducción y la conclusión, estos capítulos fueron redactados en conjunto por todos los participantes del proyecto, aunque Pablo fue quién realizó el diagrama de bloques del sistema de la introducción.

2.3.3 Desarrollo de la aplicación

El rol de Pablo en el desarrollo de aplicación ha sido principal, ya que al ser la persona que más experiencia tenía con las tecnologías ha aportado gran cantidad de soluciones a los desafíos técnicos afrontados.

Backend

Sobre el desarrollo del *backend* Pablo realizó el diseño de la estructura de la *API* aplicando los principios de la arquitectura *REST*. Además, realizó la creación y estructuración de la estructura de carpetas y de ficheros del *backend*. En cuanto a funcionalidades desarrolladas, Pablo realizó la mayoría de las operaciones de la entidad Tarea y aportó también al desarrollo del resto de funcionalidad, apoyando al resto de compañeros en los problemas surgidos.

Implementación del estándar *OAuth2.0*

En cuanto a la implementación del estándar *OAuth2.0* en la aplicación, Pablo se encargó completamente. En primer lugar Pablo tuvo que informarse sobre el funcionamiento del estándar, para ello realizó varias lecturas, entre ellas el trabajo realizado por los autores. Después, realizó un proceso de búsqueda de una posible forma de implementar el estándar, donde encontró el paquete *oauth2-server* mencionado en el capítulo **??** y procedió a realizar su implementación en el *backend*.

Frontend

En la parte del *frontend*, de la misma manera que en *backend* Pablo desarrolló los primeros pasos para empezar a implementar funcionalidad, entre estos estaba la implementación de un *navigator* o navegador, donde se utilizó el *framework* mencionado en el capítulo **??** *React-Navigation*. También realizó la integración del proceso de autorización de *OAuth* implementado en el *backend* con el *frontend* donde realizó el sistema de guardado de información en el dispositivo necesario para almacenar los *tokens* de acceso. En cuanto a la funcionalidad desarrollada, Pablo se ha centrado más en la parte funcional que en la parte de diseño de las pantallas, destacando la implementación de las operaciones de la entidad Tarea, aunque también sirvió de apoyo al resto de los compañeros en prácticamente todas las partes de la aplicación.

Pablo también realizó completamente el proceso de configuración de servidores, donde utilizó un mecanismo similar al de gestión de *tokens* para almacenar la información sobre los servidores en los dispositivos.

Modo Offline

Pablo ha desarrollado completamente el modo *offline* de la aplicación.

Implementación y compilación de la aplicación de escritorio

Pablo ha desarrollado completamente la implementación y la compilación de la aplicación de escritorio. Para ello buscó en un primer lugar cuál era la mejor forma de generar las aplicaciones de escritorio en las diferentes plataformas. Tras explorar la posibilidad de utilizar *React-Native for Windows and MacOS* [10], se descartó al estar dicho proyecto en un estado de desarrollo en el que muchos componentes que ya se habían implementado eran incompatibles. Por ello se accedió a implementar los clientes con el *framework Electron*, explicado en el apéndice 3.

2.3.4 Conclusiones

El resultado final del trabajo ha cumplido con nuestras expectativas, ya que hemos desarrollado una aplicación que cumple con los objetivos establecidos y consideramos que puede aportar gran valor a la solución de un problema como es el de organizar tus tareas. Además, este viaje nos ha hecho aprender por el camino tecnologías, estándares, metodologías, gestión de proyectos y otros aspectos que nos han hecho crecer en lo académico y en lo personal.

2.4 Javier Gil Caballero

2.4.1 Puesta en escena: Aprendizaje

Con el comienzo del curso y con el tema ya seleccionado del TFG, iniciamos una labor de búsqueda para familiarizarnos con la metodología *GTD*. Durante esta investigación nos surgieron varias dudas de conceptos las cuales fuimos resolviendo con nuestro tutor en diversas reuniones, esto nos ha permitido desarrollar nuestra aplicación siguiendo los principios que se establecen en *GTD*.

Una vez decididas las tecnologías que íbamos a utilizar para el desarrollo de la aplicación que en nuestro caso ha sido *Express.js* y *React Native*, comenzamos una labor de aprendizaje de las mismas. Aprender *Express.js* fue más sencillo ya que varios miembros del equipo lo habíamos utilizado en la carrera y *React Native* fue todo un desafío ya que nadie conocía este *framework* y nos tocó aprenderlo desde cero.

2.4.2 Documentación

Al finalizar esta investigación y conociendo los principales aspectos de *GTD*, Javier buscó información sobre distintas aplicaciones que implementan esta tecnología tanto en dispositivos móviles como en ordenadores, aportando datos como los sistemas operativos que permiten instalar cada aplicación, sus funcionalidades y sus puntos fuertes y débiles con el propósito de realizar un primer diseño de *SwiftDo* utilizando las características más útiles, mejorando aquellas que no nos convencieron en el resto de aplicaciones e

implementando nuevas funcionalidades para desarrollar una aplicación más completa que cualquiera de la competencia.

2.4.3 Desarollo:

En este apartado distinguiremos las aportaciones de Javier al proyecto en relación con la base de datos, el *backend* y el *frontend*.

Base de datos

Javier se encargó de crear la primera versión de la base de datos, creando las tablas "users", "projects", "tags", "contexts", y "tasks" con sus respectivos atributos. Además definió las claves primarias de cada tabla e incluyó las relaciones con el resto de entidades añadiendo sus *foreign keys*. Carlos y Pablo se encargaron de modificar la base de datos a medida que avanzaba el proyecto y surgían nuevas necesidades.

Backend

En cuanto al desarrollo de la aplicación, decidimos comenzar por el *backend*. Javier inició la configuración del *backend* desarrollando el archivo "app.js" para organizar y estructurar el código y las distintas rutas que se utilizarían a lo largo de este proyecto. Como equipo, distribuimos el trabajo del *backend* asignando cada entidad a un miembro del equipo. Javier se encargó de desarrollar la entidad "contexto", implementando las operaciones *CRUD* y funciones complementarias.

Además, Javier creó un validador de datos para los contextos, garantizando que los nuevos contextos esten vacíos y tengan una longitud máxima de 50 caracteres. También desarrolló las rutas necesarias para manejar las solicitudes *HTTP*, permitiendo crear, modificar, eliminar, mostrar un contexto y mostrar todos los contextos asociados a un usuario, gestionando los errores en caso de fallos en las rutas.

Por último, Javier sirvió de apoyo en otras entidades, desarrollando la ruta principal de la aplicación, rutas de usuarios y las funciones de buscar tareas por su ID y por el ID del usuario.

Frontend

Respecto al *frontend*, hemos distribuido las tareas de forma diferente, manteniendo un orden de prioridad. De este modo, todos los miembros del equipo hemos participado en todas las áreas de la aplicación.

Para empezar, Javier ha desarrollado todas las funcionalidades relacionadas con proyectos. Esto incluye el servicio de proyectos que permite crear, mostrar, modificar, completar un proyecto y mostrar los proyectos de un usuario, así como el diseño de la pantalla de proyectos.

Además, ha diseñado el modal de selección para la creación de tareas o proyectos, y el modal específico para crear proyectos, el cual permite asignar un nombre, una descripcion y un color al proyecto.

También colaboró con Carlos en el desarrollo de las etiquetas y del diseño de la pantalla de "hoy" y trabajó junto a Alejandro en el desarrollo de la barra lateral, especialmente en el apartado de proyectos y la representación del porcentaje de completitud de los mismos, así como en la numeración de las tareas para cada acción de la barra lateral.

Por último, se encargó de diseñar el modal de configuración y de implementar varias de las funcionalidades que este ofrece. Específicamente, trabajó en los apartados de datos personales, gestión de contextos y etiquetas, tareas completadas y acerca de GTD.

2.4.4 Memoria

Para la realización de la memoria, organizamos el trabajo por capítulos, realizando versiones incrementales que se revisaban antes de cada entrega. Sin embargo, para la introducción, debido a su extensión, cada miembro del equipo contribuyó con una parte.

Javier fue el encargado de realizar las secciones **??** "Plan de Trabajo" y **??** "Estructura de la memoria" en el capítulo de introducción y redactó los capítulos **??** "Planificación del proyecto" con la ayuda de Pablo y **??** Manual de usuario.

En cuanto al apartado de conclusiones y trabajo futuro, hicimos un brainstorming entre todos y lo redactamos de manera conjunta.

2.4.5 Conclusiones

Realizar un proyecto de estas características nos ha permitido aprender nuevas tecnologías, lo que ha mejorado nuestras habilidades como programadores. Gracias al trabajo en equipo, hemos afrontado y superado juntos todos los errores que surgieron a lo largo del proyecto, demostrando nuestra capacidad de colaboración y comunicación. Gracias a la buena gestión de proyecto que hemos llevado, hemos sido capaces de crear una aplicación funcional y útil, dando lugar a un producto de alta calidad.

Capítulo 3

Guía de despliegue

En este apéndice se va a detallar cómo desplegar o compilar los distintos componentes de la aplicación. También se va a explicar la estructura de carpetas del proyecto para ayudar a realizar dicho proceso de despliegue/compilación de los distintos componentes.

3.1 Estructura de carpetas del proyecto

El proyecto está formado por 3 directorios principales. El primer directorio es *BBDD* y en este donde se encuentran los *scripts* de definición de las tablas de la aplicación. Después tenemos el directorio *Backend* que contiene todo el código referente al servidor de autorización *OAuth* y la *API REST*, además de un fichero compose.yaml el cual sirve para ejecutar los componentes con *docker*. Por últlimo podemos encontrar el directorio *Frontend* el cual contiene el código fuente referente a la aplicación cliente, este código se encuentra a partir del subdirectorio */src*.

```
BBDD
    bbdd.sql
    oauth.sql
Backend
    app.js
    bd
    compose.yaml
    instalation
    oauth
    package-lock.json
    package.json
    public
    routes
    services
```

```
Frontend
    app.json
    babel.config.js
    package-lock.json
    package.json
    src
    desktop
README.md
```

3.2 Despliegue del *backend*

Requisitos

- *Docker* y *Docker-compose*

Para desplegar el *backend* debemos utilizar la herramienta *docker-compose* y para ello debemos definir un fichero *compose.yaml*. Este fichero puede definirse desde cero ajustándolo a las necesidades de infraestructura que tenga cada usuario, aun así proporcionamos la configuración que hemos utilizado nosotros el cual también se puede aplicar a cualquier necesidad. Dicho fichero se encuentra en la raíz del proyecto *Backend* y sigue la estructura que se puede ver en el siguiente definición:

```
services:
backend:
  image: node:18-alpine
  command: sh -c "npm install && cp ./instalation/oauth/author..."
  ports:
    - 3000:3000
  working_dir: /Backend
  volumes:
    - ./:/Backend
  environment:
    PORT: 3000
    POSTGRES_HOST: db
    POSTGRES_USER: tfggtdUser
    POSTGRES_PASSWORD: nodenodito@69
    POSTGRES_DB: tfggtd

db:
  image: postgres
  volumes:
    - posgresql-data:/var/lib/postgresql/data
    - ../BBDD/:/docker-entrypoint-initdb.d/
```

```
    environment:
      POSTGRES_USER: tfggtdUser
      POSTGRES_PASSWORD: nodenodito@69
      POSTGRES_DB: tfggtd
    ports:
      - 127.0.0.1:5432:5432

  volumes:
  posgresql-data:
```

Se definen dos servicios y por lo tanto dos contenedores separados, uno para la parte de la *API REST* utilizando una imagen de *Node* y otro para la base de datos utilizando una imagen de *PostgreSQL*. Dentro de cada servicio definimos las distintas variables de entorno que se utilizan en la aplicación, estas pueden modificarse a gusto del usuario. Cabe recalcar que las variables de entorno referentes a la conexión a la base de datos deben coincidir en ambos servicios.

El primer servicio llamado *backend* realizará tres acciones, la primera es instalar las dependencias con *npm install*, después realiza la sustitución de un fichero del paquete para implementar OAuth por el mismo pero modificado para la solución de un bug de este paquete. Dicho fichero modificado se encuentra en la carpeta del proyecto */Backend/instalaton*. Por último se inicia el *backend* con *npm run start*.

En cuanto al segundo servicio, el llamado *db*, iniciará el sistema de gestión de bases de datos de *PostgreSQL*. Para que el sistema cree las tablas automáticamente al iniciar la ejecución, incluimos los *scripts* de definición de datos. Las tablas solo se crearán si no existen.

Por último, en el fichero *compose.yaml* también se define un volumen para los datos de la base de datos, esto permitirá que cuando finalicemos la ejecución de los contenedores, los datos no se borren y sigan estando cuando los levantemos de nuevo.

Para ejecutar el *backend* completo basta con ejecutar el siguiente comando:

```
docker-compose up -d
```

El flag *-d* ejecutará los contenedores en segundo plano liberando el terminal donde se haya ejecutado el comando.

3.3 Compilación de la aplicación web

Requisitos

- *Node.js - (18.18.2)*
- *npm (9.8.1)*

Las versiones indicadas son las que hemos utilizado nosotros, con versiones más modernas debería de funcionar correctamente, no así con versiones anteriores.

Para la compilación y generación de la aplicación web basta con ejecutar los siguientes comandos situándonos en el directorio *Frontend*:

```
npm install
npx expo export -p web
```

La ejecución de estos dos comandos generará una carpeta en el mismo directorio llamada *dist*, la cual contiene el *bundle* o paquete de la aplicación generada para web. Este contiene un *index.html* y los *scripts* necesarios. Por lo que para ejecutar la aplicación, bastará con iniciar cualquier servidor web sirviendo dicho directorio y el *index.html* como página de entrada.

3.4 Compilación de la aplicación de escritorio

- *Node.js - (18.18.2)*
- *npm (9.8.1)*

Las versiones indicadas son las que hemos utilizado nosotros, con versiones más modernas debería de funcionar correctamente, no así con versiones anteriores.

La versión de escritorio está desarrollada con *Electron*, un framework que permite ejecutar aplicaciones desarrolladas con tecnologías web en escritorio, esto se consigue incluyendo *Chromiun* y *Node.js* en los binarios generados, dicho en otras palabras el binario generado con la aplicación web se ejecuta en un navegador incrustado en el mismo binario. *Electron* permite generar binarios para todas las principales plataformas, por ello nuestra aplicación puede ser compilada para cualquiera de estas.

Para realizar la generación del binario de la aplicación de escritorio debemos situarnos en el directorio *Frontent/desktop*, donde hemos incluido un script tanto en *bash* como en *powershell* que realiza los pasos necesarios. Una vez ejecutado el *script*, se generará en la misma carpeta un directorio llamado *out* el cual contendrá el ejecutable de la aplicación de escritorio. Es importante recalcar que tal compilación se hará para la misma plataforma donde se realice la misma, por ejemplo si deseamos compilar para *windows* se deberá de realizar este proceso en *windows*.

3.5 Compilación de la aplicación de *Android*

- *Docker*
- Cuenta *Expo*

Las versiones indicadas son las que hemos utilizado nosotros, con versiones más modernas debería de funcionar correctamente, no así con versiones anteriores. Para poder compilar

la aplicación de *Android* es necesario realizar el proceso desde una máquina *linux* por ello lo hemos realizado en un contenedor de *docker* con una imagen de *ubuntu* que además contiene todas las dependencias necesarias para realizar la compilación. Todas estas dependencias se encuentran definidas en el *dockerfile* localizado en la carpeta */Frontend/dockerfile*.

La compilación de la aplicación de *Android* se realiza con la herramienta *expo* y por lo tanto es necesario disponer de una cuenta de esta plataforma. El proceso se ejecuta totalmente en local, aunque *expo* dispone también de un servicio de compilación en sus servidores.

Para realizar la compilación, nos situaremos en primer lugar en el directorio */Frontend/dockerfile* y realizaremos la *build* del contenedor de docker con el siguiente comando:

```
docker build -t apkswiftdo .
```

La *flag* -t define una *tag* para nuestro contenedor con el objetivo de luego localizarlo con facilidad. Una vez ejecutado el anterior comando comenzará la *build*, cabe destacar que puede que este proceso tarde varios minutos. A continuación según acabe la *build* accederemos interactivamente al contenedor refiriéndonos a él con la *tag* previamente definida con el siguiente comando:

```
docker run -it apkswiftdo
```

Tras ejecutar el anterior comando entraremos en el terminal del contenedor. Por último ejecutamos el siguiente comando para generar el fichero *.apk* de la aplicación:

```
eas build --platform android --local --profile preview
```

Este proceso puede alargarse varios minutos. Una vez acabado observaremos el siguiente mensaje:

```
Build successful
You can find the build artifacts in /project/build-1716755505051.apk
```

Tras finalizar el proceso, ya disponemos del fichero *.apk* el cual podemos exportar a nuestra máquina desde el contenedor con el siguiente comando:

```
docker cp <containerId>:/project/build-1716755505051.apk ./
```

Para obtener el identificador del contenedor podemos ejecutar el siguiente comando:

```
docker ps
```

Capítulo 4

Endpoints

En el este apéndice se detallan los *endpoints* implementados en la *API REST*. Cada *endpoint* se describe mediante dos tablas, una que define las características de este y otra que define los parámetros que puede recibir.

Tarea

Endpoint	/task/
Descripción	Crea una tarea para el usuario que lo solicita
Método HTTP	POST
Cabecera de Autorización	Bearer token
Código HTTP (OK)	201
Código HTTP (KO)	409

Parámetros	Descripción	Tipo	Opcional
title (*Body*)	Titulo de la tarea	*String*	No
description (*Body*)	Descripción de la tarea	*String*	Si
state (*Body*)	Estado de la tarea	*Integer*	Si
important_fixed (*Body*)	Valor de importante	*Boolean*	Si
completed (*Body*)	Valor de completado	*Boolean*	Si
project_id (*Body*)	Proyecto de la tarea	*Integer*	Si
context_id (*Body*)	Contexto de la tarea	*Integer*	Si
date_limit (*Body*)	Fecha de la tarea	*Timestamp*	Si

Endpoint	/task/:id
Descripción	Modifica la tarea con el id indicado en la url del usuario que lo solicita
Método HTTP	POST
Cabecera de Autorización	Bearer token
Código HTTP (OK)	200
Código HTTP (KO)	409

Parámetros	Descripción	Tipo	Opcional
task_id (*Path*)	id de la tarea a modificar	*Integer*	No
title (*Body*)	Titulo de la tarea modificado	*String*	Si
description (*Body*)	Descripción de la tarea modificado	*String*	Si
state (*Body*)	Estado de la tarea modificado	*Integer*	Si
important_fixed (*Body*)	Valor de importante	*Boolean*	Si
completed (*Body*)	Valor de completado	*Boolean*	Si
project_id (*Body*)	Proyecto de la tarea	*Integer*	Si
context_id (*Body*)	Contexto de la tarea	*Integer*	Si
date_limit (*Body*)	Fecha de la tarea	*Timestamp*	Si

Endpoint	/task/:id
Descripción	Devuelve el contenido de tarea con el id solicitado
Método HTTP	GET
Cabecera de Autorización	Bearer token
Código HTTP (OK)	200
Código HTTP (KO)	404

Parámetros	Descripción	Tipo	Opcional
task_id (*Path*)	id de la tarea a consultar	*Integer*	No

Endpoint	/task/
Descripción	Devuelve todas las tareas no completadas del usuario que las solicita
Método HTTP	GET
Cabecera de Autorización	Bearer token

Endpoint	/task/
Código HTTP (OK)	200
Código HTTP (KO)	404

Endpoint	/task/movelist
Descripción	Mueve de estado un listado de tareas
Método HTTP	POST
Cabecera de Autorización	Bearer token
Código HTTP (OK)	200
Código HTTP (KO)	409

Parámetros	Descripción	Tipo	Opcional
list_ids (*Body*)	Listado de ids de las tareas a mover de estado	*array[Integer]*	No
state (*Body*)	Estado al que se mueven las tareas	*Integer*	No

Endpoint	/task/completelist
Descripción	Completa un listado de tareas
Método HTTP	POST
Cabecera de Autorización	Bearer token
Código HTTP (OK)	200
Código HTTP (KO)	409

Parámetros	Descripción	Tipo	Opcional
list_ids (*Body*)	Listado de ids de las tareas a mover de estado	*array[Integer]*	No
completed(*Body*)	Valor de completar al que modificar las tareas	*Boolean*	No

Endpoint	/task/addTag
Descripción	Añade una etiqueta a una tarea
Método HTTP	POST
Cabecera de Autorización	Bearer token
Código HTTP (OK)	200
Código HTTP (KO)	409

Parámetros	Descripción	Tipo	Opcional
task_id (*Body*)	Id de la tarea a la cual se le añade la etiqueta	*Integer*	No
tag(*Body*)	Etiqueta a añadir a la tarea	*Object*	No

Endpoint	/task/:id/tags
Descripción	Devuelve el listado de etiquetas de la tarea con el id solicitado.
Método HTTP	GET
Cabecera de Autorización	Bearer token
Código HTTP (OK)	200
Código HTTP (KO)	404

Parámetros	Descripción	Tipo	Opcional
task_id (*Path*)	Id de la tarea a obtener sus etiquetas	*Integer*	No

Endpoint	/task/info
Descripción	Devuelve un resumen de la información de las tareas del usuario que lo solicita. Para mostrar en la sidebar.
Método HTTP	GET
Cabecera de Autorización	Bearer token
Código HTTP (OK)	200
Código HTTP (KO)	404

Usuario

Endpoint	/user/register
Descripción	Realiza el registro de un usuario creandolo en la BBDD
Método HTTP	POST
Cabecera de Autorización	
Código HTTP (OK)	200
Código HTTP (KO)	409

Parámetros	Descripción	Tipo	Opcional
email (*Body*)	E-mail del usuario	*String*	No
name (*Body*)	Nombre de usuario	*String*	No

Parámetros	Descripción	Tipo	Opcional
password (*Body*)	Contraseña del usuario	*String*	No

Auth

Endpoint	/oauth/authorize
Descripción	Obtiene el codigo de autorización de *OAuth* si el la autenticación es correcta.
Método HTTP	POST
Cabecera de Autorización	
Código HTTP (OK)	304
Código HTTP (KO)	401

Parámetros	Descripción	Tipo	Opcional
client_id (*Body*)	id del cliente a autorizar	*Integer*	No
response_type (*Body*)	*code* por defecto ya que estamos obtieniendo el codigo de autorización	*String*	No
email (*Body*)	E-mail del usuario	*String*	No
password (*Body*)	Contraseña del usuario	*String*	No

Endpoint	/oauth/token
Descripción	Si el codigo de autorización es correcto devuelve el token de acceso. Tambien permite realizar la renovación del token.
Método HTTP	POST
Cabecera de Autorización	

Endpoint	/oauth/token
Código HTTP (OK)	200
Código HTTP (KO)	401

Parámetros	Descripción	Tipo	Opcional
client_id (*Body*)	Id del cliente desde donde se requiere el token	*Integer*	No
client_secret(*Body*)	secreto del cliente autorizado	*String*	No
grant_type (*Body*)	Tipo de proceso de *OAuth* puede ser *authorization_code* para obtener el token a partir de un codigo o *refresh_token* para refrescar un token ya existente	*String*	No
code (*Body*)	Si estamos en el proceso *authorization_code* es el código de autorización obtenido previamente. Si no, no es necesario	*String*	Si
redirect_uri (*Body*)	uri a la que redireccionará al obtener el token	*String*	No
refresh_token (*Body*)	Si estamos en el proceso *refresh_token* es el token de renovación necesario para obtener un nuevo token. Si no, no es necesario.	*String*	Si
email (*Body*)	E-mail del usuario	*String*	No
password (*Body*)	Contraseña del usuario	*String*	No

Proyecto

Endpoint	/project/
Descripción	Devuelve todos los proyectos del usuario que lo requiere.
Método HTTP	GET
Cabecera de Autorización	Bearer token
Código HTTP (OK)	200
Código HTTP (KO)	404

Endpoint	/project/
Descripción	Crea un proyecto para el usuario que lo requiere.
Método HTTP	POST
Cabecera de Autorización	Bearer token
Código HTTP (OK)	200
Código HTTP (KO)	409

Parámetros	Descripción	Tipo	Opcional
title (*Body*)	Titulo del proyecto	*String*	No
description (*Body*)	Descripción del proyecto	*String*	Si
color (*Body*)	Color del proyecto	*String*	No

Endpoint	/project/:id/complete
Descripción	Completa el proyecto referente al id pasado por la url para el usuario que lo requiere. Si tiene tareas asociadas sin completar, también las completa.
Método HTTP	POST
Cabecera de Autorización	Bearer token

Endpoint	/project/:id/complete
Código HTTP (OK)	200
Código HTTP (KO)	404

Parámetros	Descripción	Tipo	Opcional
project_id (*Path*)	Id del proyecto	*Integer*	No

Endpoint	/project/:id
Descripción	Devuelve toda la información referente al proyecto del id pasado por la url.
Método HTTP	GET
Cabecera de Autorización	Bearer token
Código HTTP (OK)	200
Código HTTP (KO)	404

Parámetros	Descripción	Tipo	Opcional
project_id (*Path*)	Id del proyecto	*Integer*	No

Endpoint	/project/:id
Descripción	Modifica el proyeto con id pasado por la url.
Método HTTP	POST
Cabecera de Autorización	Bearer token

Endpoint	/project/:id
Código HTTP (OK)	200
Código HTTP (KO)	404

Parámetros	Descripción	Tipo	Opcional
title (*Body*)	Titulo del proyecto	*String*	Si
description (*Body*)	Descripción del proyecto	*String*	Si
color (*Body*)	Color del proyecto	*String*	Si
completed (*Body*)	Valor de completado	*Boolean*	Si

Contexto

Endpoint	/context/
Descripción	Crea un context para el usuario que lo requiere.
Método HTTP	POST
Cabecera de Autorización	Bearer token
Código HTTP (OK)	200
Código HTTP (KO)	404

Parámetros	Descripción	Tipo	Opcional
name (*Body*)	Nombre del contexto	*String*	No

Endpoint	/context/
Descripción	Devuelve todos los contextos del usuario que lo requiere.
Método HTTP	GET
Cabecera de Autorización	Bearer token
Código HTTP (OK)	200
Código HTTP (KO)	404

Endpoint	/context/:id
Descripción	Elimina el contexto con el id pasado por la url. Si ese contexto tiene tareas, se modifican para que no tengan contexto.
Método HTTP	DELETE
Cabecera de Autorización	Bearer token
Código HTTP (OK)	200
Código HTTP (KO)	404

Parámetros	Descripción	Tipo	Opcional
context_id (*Path*)	id del contexto	*Integer*	No

Endpoint	/context/:id
Descripción	Modifica el contexto con el id pasado por la url.
Método HTTP	POST
Cabecera de Autorización	Bearer token

Endpoint	/context/:id
Código HTTP (OK)	200
Código HTTP (KO)	404

Parámetros	Descripción	Tipo	Opcional
context_id (*Path*)	id del contexto	*Integer*	No
name (*Body*)	Nuevo nombre del contexto	*String*	Si

Etiqueta

Endpoint	/tag/
Descripción	Crea una etiqueta para el usuario que lo requiere.
Método HTTP	POST
Cabecera de Autorización	Bearer token
Código HTTP (OK)	200
Código HTTP (KO)	409

Parámetros	Descripción	Tipo	Opcional
name (*Body*)	Nombre de la etiqueta	*String*	No

Endpoint	/tag/gettags
Descripción	Obtiene las etiquetas del usuario que lanza la operación.
Método HTTP	GET

Endpoint	/tag/gettags
Cabecera de Autorización	Bearer token
Código HTTP (OK)	200
Código HTTP (KO)	404

Endpoint	/tag/:name
Descripción	Elimina la etiqueta con el nombre pasado por parametro.
Método HTTP	DELETE
Cabecera de Autorización	Bearer token
Código HTTP (OK)	200
Código HTTP (KO)	404

Parámetros	Descripción	Tipo	Opcional
name (*Path*)	Nombre de la etiqueta a eliminar	*String*	No

Bibliografía

[1] D. Allen, *Getting Things Done. The Art of Stress-Free Productivity*. Penguin, 2003.

[2] C. Code, *Things*. 2024.

[3] J. David-Lang, «Summary of the Summary: Getting Things Done: The Art of Stress-Free Productivity by David Allen». 2017.

[4] D. A. Norman, *The Design of Everyday Things*. USA: Basic Books, Inc., 2002.

[5] freeCodeCamp, «Node.js and Express.js - Full Course». abr-2021.

[6] H. Djirdeh, A. Accomazzo, y S. Shoemaker, *Fullstack React Native: Create Beautiful Mobile Apps with JavaScript and React Native*. Amazon Digital Services LLC - KDP Print US, 2019.

[7] M. Platforms, «React Native Learn once, write anywhere.» 2024.

[8] «Drawer Navigation». https://reactnavigation.org/docs/drawer-navigator/.

[9] freeCodeCamp, «React Native Course - Android and iOS App Development». 2023.

[10] «React Native for Windows + macOS». https://microsoft.github.io/react-native-windows/.