
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 este proyecto propone el desarrollo de 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 hará uso preferentemente de la tecnología *React Native* para el desarrollo de clientes multiplataforma. Algunos aspectos clave de la infraestructura a desarrollar serán su *modo offline* para permitir el funcionamiento de los clientes incluso sin conexión a Internet, y el empleo de estándares de autorización como *OAuth* para dotar de seguridad a la *API REST*. Para su implementación se hará uso del framework *Express.js* que ayudará a simplificar el desarrollo. Asimismo, también se explorarán métodos para interactuar con el servicio *REST GTD* desde asistentes conversacionales.

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

Abstract

Blah Blah ...

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

Resumen	iii
Abstract	v
Autorización de difusión y utilización	i
1 Diseño e Implementacion del Frontend	1
1.1 Percepción del funcionamiento del sistema	1
1.2 Metáforas, Expresiones y Conceptos de diseño	2
1.2.1 Proximidad y Consistencia	4
1.2.2 Visibilidad	4
1.3 Prototipos e Interfaces	9
1.4 Implementación: ¿Qué es React Native?	11
2 Guía de despliegue	29
2.1 Estructura de carpetas del proyecto	29
2.2 Despligue del <i>backend</i>	30
2.3 Compilación de la aplicación web	31
2.4 Compilación de la aplicación de escritorio	32
2.5 Endpoints	32
2.5.1 Tarea	32
2.5.2 Usuario	36
2.5.3 Auth	36
2.5.4 Proyecto	38
2.5.5 Contexto	40
2.5.6 Etiqueta	41
3 Bibliografía	43

Índice de cuadros

Índice de figuras

1.1	Menu lateral	2
1.2	Boton para añadir tarea/proyecto	4
1.3	Archivadas_Screen	5
1.4	CuantoAntes_Screen	6
1.5	Entrada_Screen	7
1.6	Hoy_Screen	8
1.7	nombre de usuario y fecha	9
1.8	Detalles	10
1.9	inicio de sesion	12
1.10	Registro	13
1.11	escritorio	14
1.12	Entrada_Screen	15
1.13	CuantoAntes_Screen	16
1.14	Programadas	17
1.15	Archivadas_Screen	18
1.16	Hoy_Screen	19
1.17	menu	20
1.18	Ajustes - Movil	21
1.19	Ajustes - Escritorio	21
1.20	OptionSettings	22
1.21	details	23
1.22	editTask	24
1.23	addProyectOrTask	25
1.24	Menu add task	26
1.25	visualizacionTarea	27

Capítulo 1

Diseño e Implementacion del Frontend

En este capítulo ahondaremos en varias de las técnicas de diseño de software que hemos usado para bocetar el conjunto de interfaces de usuario que conforman *SwiftDo*. Además, explicaremos las herramientas que hemos utilizado para prototipar e implementar el Frontend de la app. Para que una aplicación llegue a ser utilizada de manera cotidiana, deberá cumplir con los criterios y objetivos para los que fue concebida, así como disponer de un buen diseño que lo complemente.

— ¿Qué hace que una aplicación esté bien diseñada? — Para responder a esta pregunta hemos explorado algunos libros y/o asignaturas cursadas en la universidad y, después de una exhaustiva búsqueda, hemos encontrado “The Design of Everyday Things” de *D. Norman*, uno de los libros más importantes que relaciona el diseño de producto y la cognición humana. A continuación proporcionaremos de manera desglosada las técnicas en las que nos apoyamos:

1.1 Percepción del funcionamiento del sistema

Para empezar a diseñar la app, investigamos, no solo como funciona el método explicado en el capítulo 2 <(Hacer referencia)>, sino también como trasladar esa experiencia fielmente al software. La percepción de la filosofía GTD puede variar según cada usuario, sin embargo, la esencia principal de ésta debe ser la misma. Para ello debemos meditar qué tipo de elementos deben estar o no presentes en la app. De esta forma conseguimos que coincidan el **modelo mental**, es decir, cómo el usuario piensa que funciona la aplicación, y el **modelo tecnológico**, que representa el funcionamiento interno.

Al haber reunido los conceptos fundamentales en el mapa mental de la figura ?? conseguimos reducir la fricción cognitiva, es decir, reducir la fatiga visual mediante la simplificación de conceptos y hacer que su uso sea intuitivo y eficiente.

1.2 Metáforas, Expresiones y Conceptos de diseño

Con la finalidad de hacer que el usuario se vea familiarizado con los conceptos empleados en la app, nos hemos visto envueltos en un conjunto de metáforas y expresiones.

En primer lugar, usamos metáforas para asociar cada “acción” del método *GTD* con un icono que exprese fielmente lo que realiza esa categoría. Ejemplo de ello, pueden ser la “entrada”, representada por una bandeja de entrada de mensajes, la bandera roja indicando la urgencia o el símbolo de progreso de cada proyecto, tal y como se puede observar en las figura 1.1:

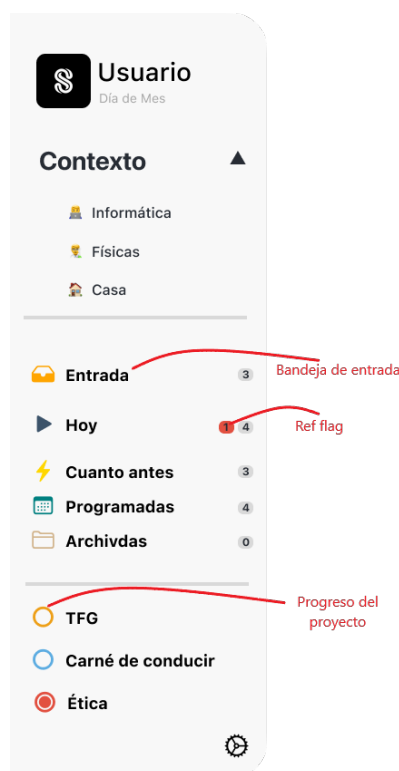
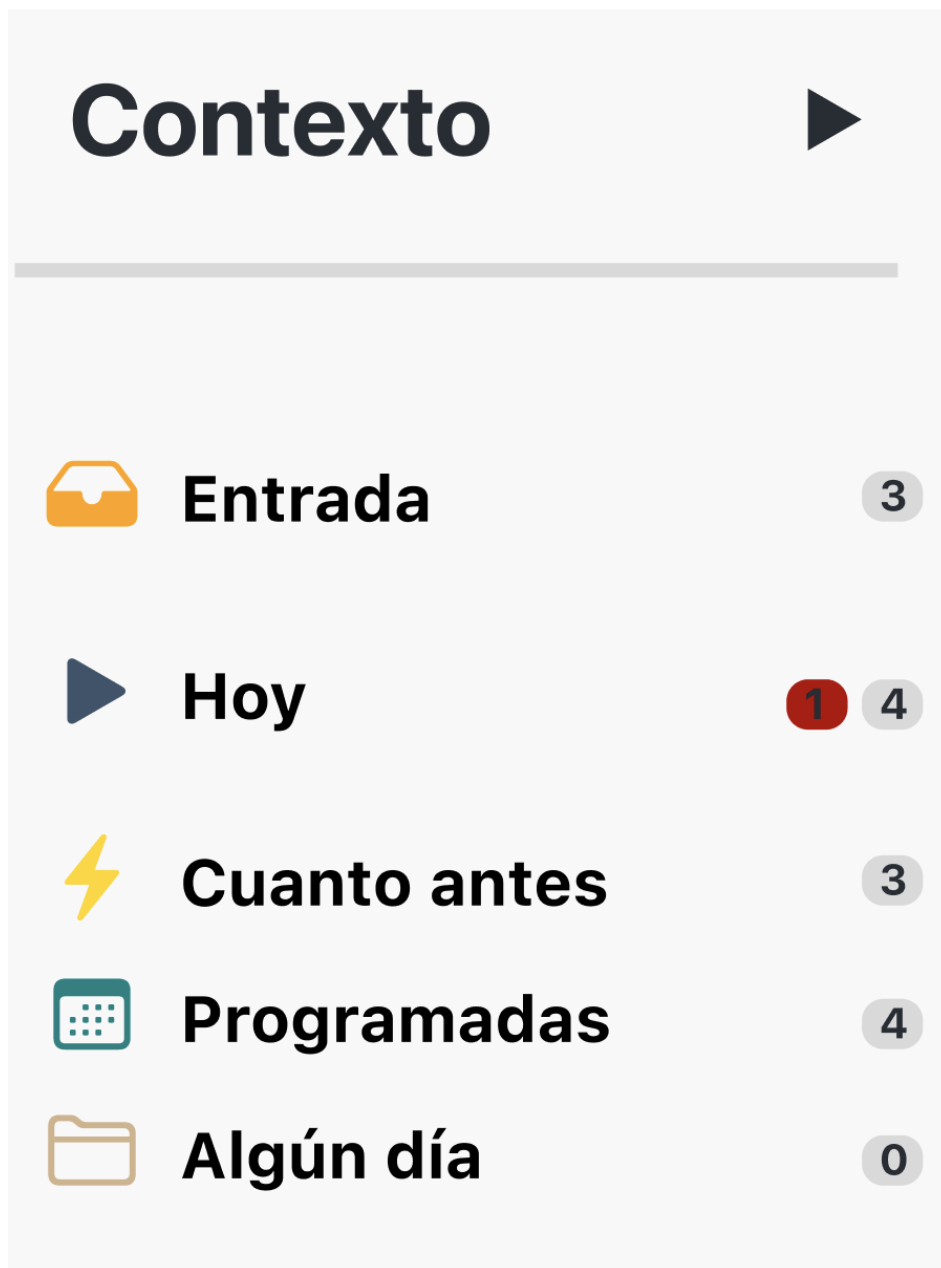


Figura 1.1: Menu lateral

Por otro lado, tenemos las expresiones, éstas son un concepto fundamental, ya que ayudan a definir como interactúa el usuario con la aplicación, haciéndola mas eficiente y consistente. Éstas las podemos ver en el menú lateral, con las secciones ?? de “Entrada”, “Hoy”, “Cuanto antes”, “Programadas”, etc.



Finalmente, hemos optado por un diseño plano y elegante, con pocas trazas de esqueuomorfismo, evitando sobrecargar con detalles superfluos la interfaz. Para ello nos hemos basado en el concepto de *affordance*, haciendo que cada componente que se encuentre en la app sea autoexplicativo, como por ejemplo, el botón de añadir tarea y/o proyecto 1.2 Para explorar con mayor detalle estos aspectos, nos hemos basado en varios principios de diseño que describiremos a continuación.

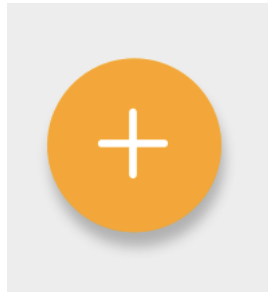


Figura 1.2: Boton para añadir tarea/proyecto

1.2.1 Proximidad y Consistencia

Estos principios se encuentran relacionados con la coherencia con la que diseñamos la interfaz. El primero de estos principios, **proximidad**, indica que todos los elementos que se encuentren relacionados entre sí deben agruparse visualmente, familiarizándolo y simplificando el proceso de aprendizaje del usuario con los conceptos tratados, como se observa en la figura ??

En segundo lugar, detallamos el principio de **consistencia**, que ha sido fundamental tanto para el diseño como para la implementación. Siendo de gran utilidad para la optimización del código, diseño 1.6 y aprendizaje de la interfaz por parte del usuario final.

En segundo lugar, detallamos el principio de **consistencia**, que ha sido fundamental tanto para el diseño como para la implementación. Siendo de gran utilidad para la optimización del código, diseño ?? y aprendizaje de la interfaz por parte del usuario final.

1.2.2 Visibilidad

D. Norman en su libro explica que todo componente debe proporcionar una representación clara, tanto de su estado como de las funciones que éstas desempeñan. Tanto es así, que explica que cuanto mas visible sea un objeto, mayor será la interacción que tendrá el usuario con él.

Por esto, hemos usado un diseño plano y minimalista, donde los detalles que tienen alta relevancia y que se relacionan con el método GTD, pasan a un primer plano, como son las “acciones” ?? , mientras los demás ocupan un segundo plano.

Por último, es conveniente gestionar el estado visible de los componentes de la app, es decir, que el usuario pueda observar claramente el estado actual del sistema. Ejemplo de ello son los detalles en el menú lateral, como el nombre de usuario, que le informa que ha iniciado sesión en la app, así como la fecha actual, que puede ser de especial relevancia para la creación de tareas 1.7 Por otro lado, las etiquetas y el contexto asociado a cada tarea en la pantalla de “detalles” ?? dirigen la atención a lo que realmente importa.

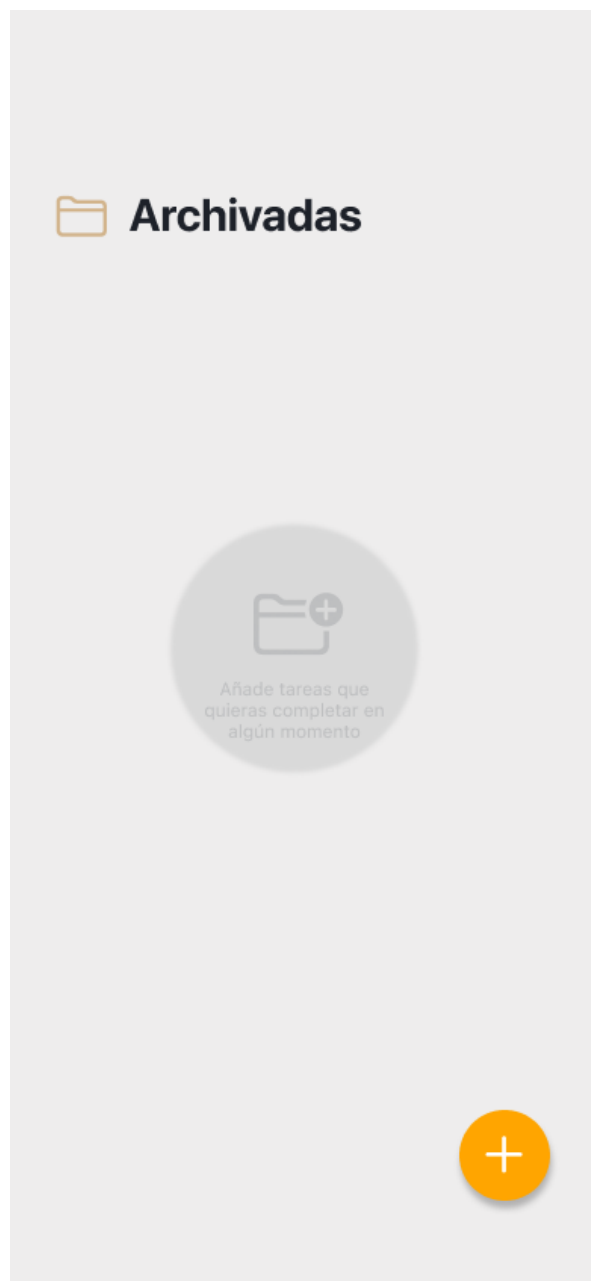


Figura 1.3: Archivadas_Screen

Cuanto antes


- ☐ Enviar diseño de prototipado 
- ☐ Terminar tema 1 de AC
- ☐ Descargar documentos CV



Figura 1.4: CuantoAntes_Screen

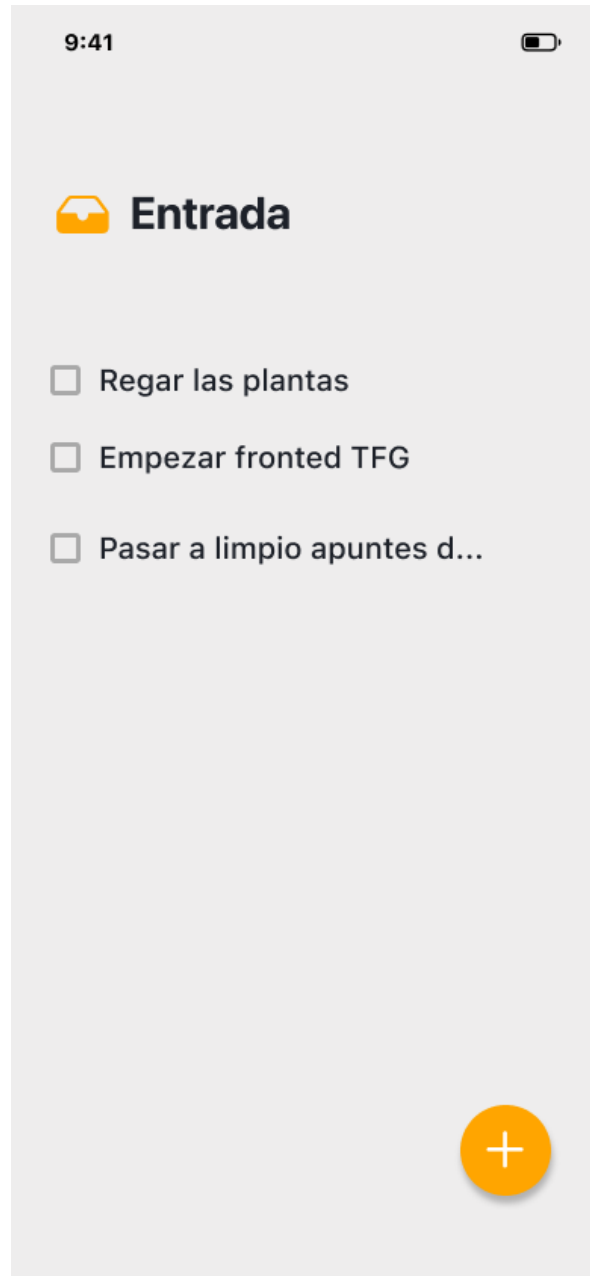


Figura 1.5: Entrada_Screen

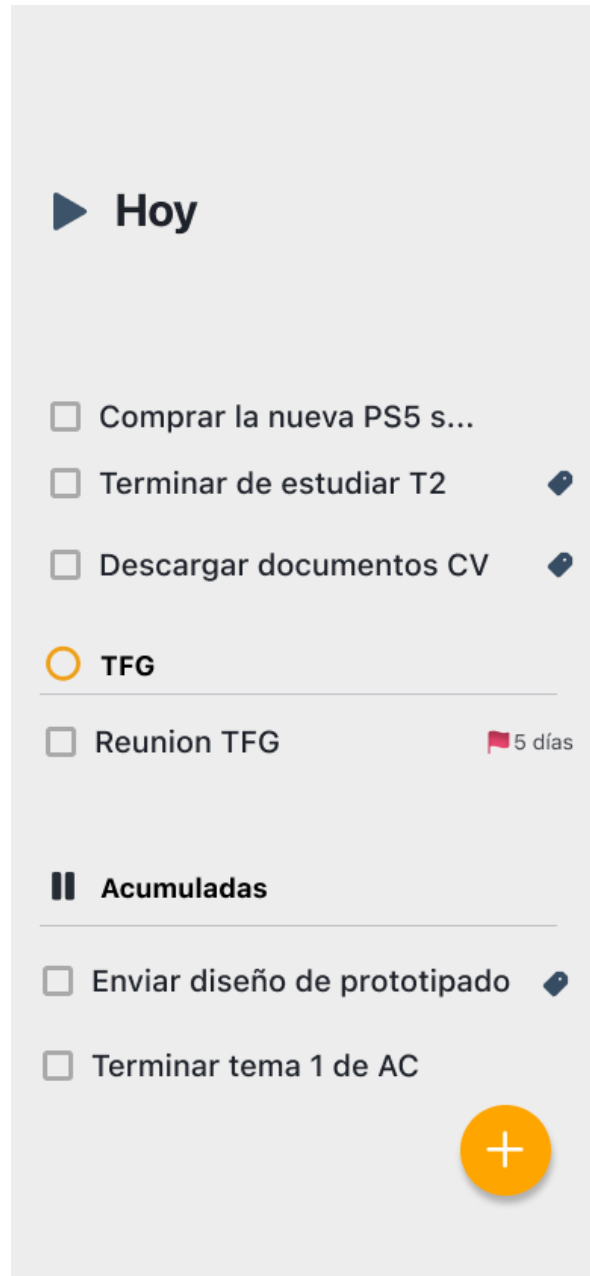


Figura 1.6: Hoy_Screen



Figura 1.7: nombre de usuario y fecha

Por esto, hemos usado un diseño plano y minimalista, donde los detalles que tienen alta relevancia y que se relacionan con el método GTD, pasan a un primer plano, como son las “acciones” ?? y el número de tareas que tienen tanto, mientras los demás ocupan un segundo plano.

<REFERENCIA A NUMEROS DE TAREAS EN ACCIONES; ACCIONES Y PROYECTOS Y TAREAS>

Por último, es conveniente gestionar el estado visible de los componentes de la app, es decir, que el usuario pueda observar claramente el estado actual del sistema. Ejemplo de ello son los detalles en el menú lateral, como el nombre de usuario, que le informa que ha iniciado sesión en la app, así como la fecha actual, que puede ser de especial relevancia para la creación de tareas . Por otro lado, las etiquetas y el contexto asociado a cada tarea en la pantalla de “detalles” , dirigen la atención a lo que realmente importa.

1.3 Prototipos e Interfaces

Durante el desarrollo prematuro de la app exploramos diferentes herramientas de diseño para poder elaborar varios de los mockups que componen la app, entre ellas destacamos **Balsamiq** y **Figma**. Sin embargo, nos decantamos por Figma, ya que es una herramienta más versátil y con mayores opciones de personalización, abarcando desde modelos de baja a alta fidelidad.

Figma es una plataforma de diseño colaborativo que permite desde la creación de prototipos interactivos hasta interfaces de usuario con alto nivel de detalle. Además de permitir diseñar fácilmente las interfaces, habilita a otros miembros del grupo a colaborar y compartir el contenido que se está diseñando, lo que ha ayudado, en gran parte, a la pronta implementación del Frontend del proyecto.

Asimismo, usamos gran parte de las opciones que ofrece para probar varios de los

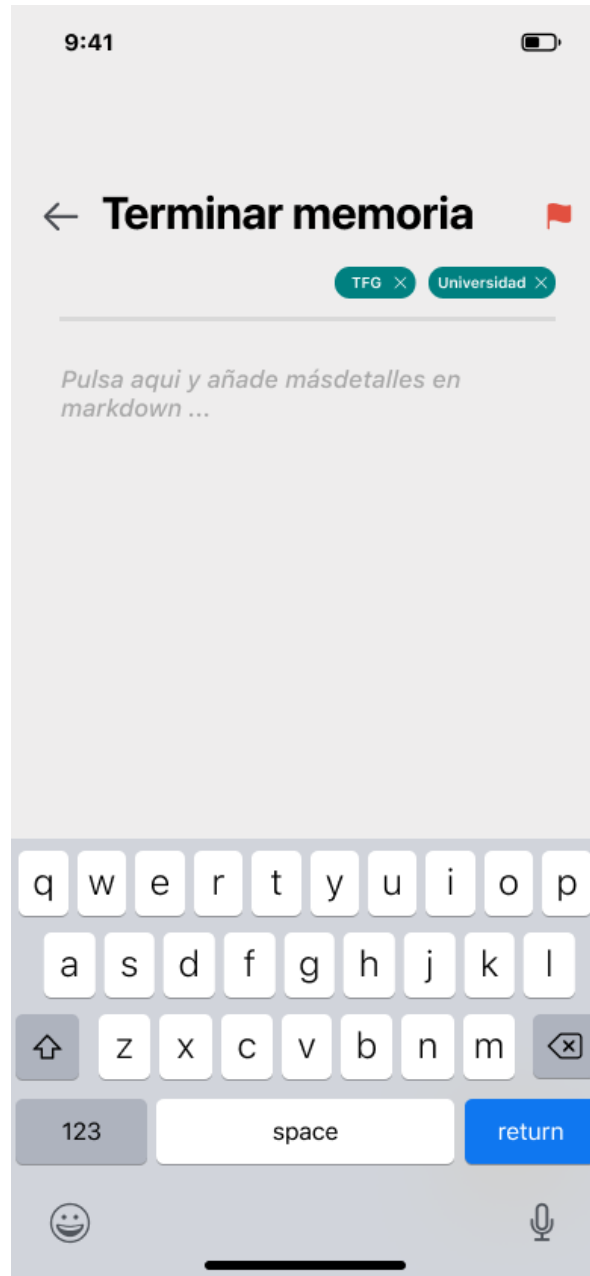


Figura 1.8: Detalles

principios de diseño que hemos logrado implementar en la aplicación final, además de las fuentes y colores utilizados en la misma.

Finalmente, una vez creados todos los prototipos junto a sus componentes, probamos el flujo de diseño que habíamos creado inicialmente, mediante el uso de la aplicación móvil, llegando a emular fielmente 1:1 el comportamiento final de la app.

A continuación presentamos todos las interfaces que componen SwiftDo, junto a una breve explicación que las relaciona con el método GTD

En primer lugar tenemos las interfaces relacionadas con el registro e inicio de sesión:

En segundo lugar tenemos las tareas relacionadas con la gestion del flujo de *GTD* y sus acciones tanto en escritorio, como en dispositivos móviles:

En adición a las categorías de *GTD* mencionadas en el capítulo 2, hemos querido añadir una nueva categoría “**Hoy**” ?? que actue como resumen diario inteligente. Éste detectará cuyas tareas pertenecen al día de “hoy”, tareas atrasadas cuya finalización ha expirado y por último, en caso de tener pocas tareas para realizar en el día, se irán adjuntando, otras pertenecientes a la categoría “**cuanto antes**”.

En tercer lugar tenemos interfaces relacionadas con el menú o barra lateral y ajustes tanto en formato escritorio como en dispositivos móviles:

Por último tenemos interfaces más detalladas como son la de añadir tarea/proyecto, interfaz de añadir detalles a la tarea, pudiendo insertar markdown texto en formato markdown y distintos modales para la creacion o edicion de tareas.

1.4 Implementación: ¿Qué es React Native?

Para implementar el Frontend de la app junto a las interfaces de usuario anteriormente descritas, necesitábamos hacer uso de un framework que pudiera adaptarse no solo a interfaces táctiles sino también a interfaces de escritorio. Para lograrlo hemos hecho uso de **React Native*** y ***Expo***.

Expo se trata de una plataforma que agiliza el desarrollo y despliegue de aplicaciones en React Native sin la necesidad de lidiar con configuraciones complejas.

Por otro lado, React Native se trata de un framework desarrollado por Facebook que tiene por lema “*Learn once, write anywhere*”, ya que mediante el uso de React y JavaScript, permite la creación de aplicaciones multiplataforma, ya sea en *MacOS*, *iOS*, *Windows* o *Android*.

Éste framework no solo destaca por la capacidad de reutilización de código sino por ofrecer un comportamiento nativo de los componentes utilizados, es decir, permitir un rendimiento similar al de las aplicaciones desarrolladas con lenguajes nativos (*Swift* para *IOS* o *Kotlin* para *Android*).



**Domina el caos,
conquista tu día.**

¿Aún no tienes cuenta? [Regístrate](#)

Iniciar sesión

Figura 1.9: inicio de sesión
12



**Domina el caos,
conquista tu día.**

Nombre (opcional)

Correo electrónico

Contraseña

¿Ya estás registrado? [Inicia sesión](#)

Registrar

Figura 1.10: Registro

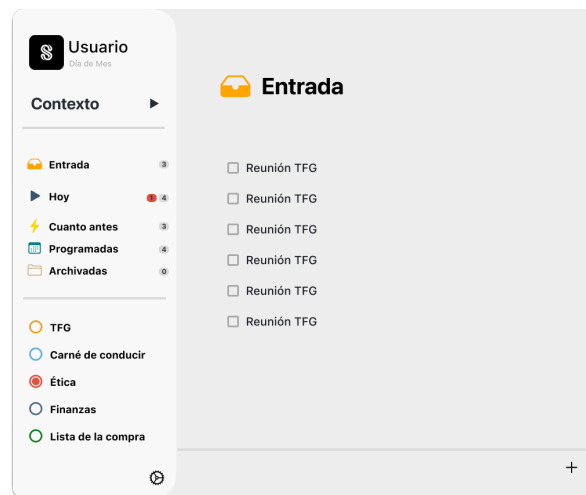


Figura 1.11: escritorio

Por último, destacar la gran comunidad que tiene, ya que además de la documentación que podemos consultar en la pagina web oficial de React Native, disponemos de bibliotecas externas que se detallan en otras páginas web que son útiles para la implementación de otros componentes que no se encuentran en las fuentes oficiales. Las bibliotecas externas que hemos empleado son las siguientes:

- **React Navigation:**¹ Esta librería permite definir y enrutar las diferentes pantallas de la aplicación para poder navegar entre ellas. Ha sido de gran importancia en la aplicación ya que la navegación es un aspecto clave en cualquier aplicación. Además este paquete es muy sencillo de entender y usar gracias a su buena documentación.
- **Drawer Navigation:**² Muestra un cajón de navegación en el costado de la pantalla que se puede abrir y cerrar mediante gestos.
- **Axios:**³ Librería de *JavaScript* que se utiliza para hacer solicitudes *HTTP* desde el navegador. Este paquete facilita la gestión y el envío de *requests* y *responses*, esto nos ha servido para gestionar los *headers* de autorización.
- **React Native Calendars:**⁴ Un componente de calendario declarativo multiplataforma *React Native* para *iOS* y *Android*. Se ha utilizado para implementar la pantalla de Programadas.
- **React Native Async Storage:**⁵ Este paquete ha sido de gran importancia en el desarrollo ya que permite guardar información en el dispositivo. Se ha utilizado

¹React Navigation

²Drawer Navigation

³Axios

⁴React Native Calendars

⁵React Native Async Storage

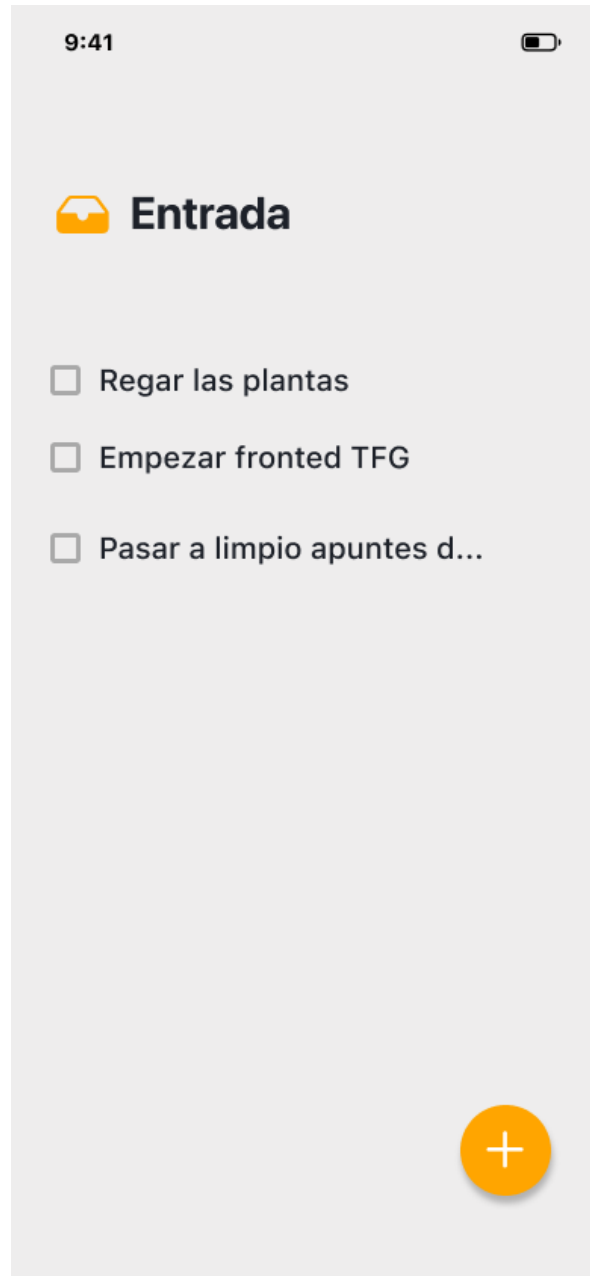


Figura 1.12: Entrada_Screen

Cuanto antes


- ☐ Enviar diseño de prototipado 
- ☐ Terminar tema 1 de AC
- ☐ Descargar documentos CV



Figura 1.13: CuantoAntes_Screen

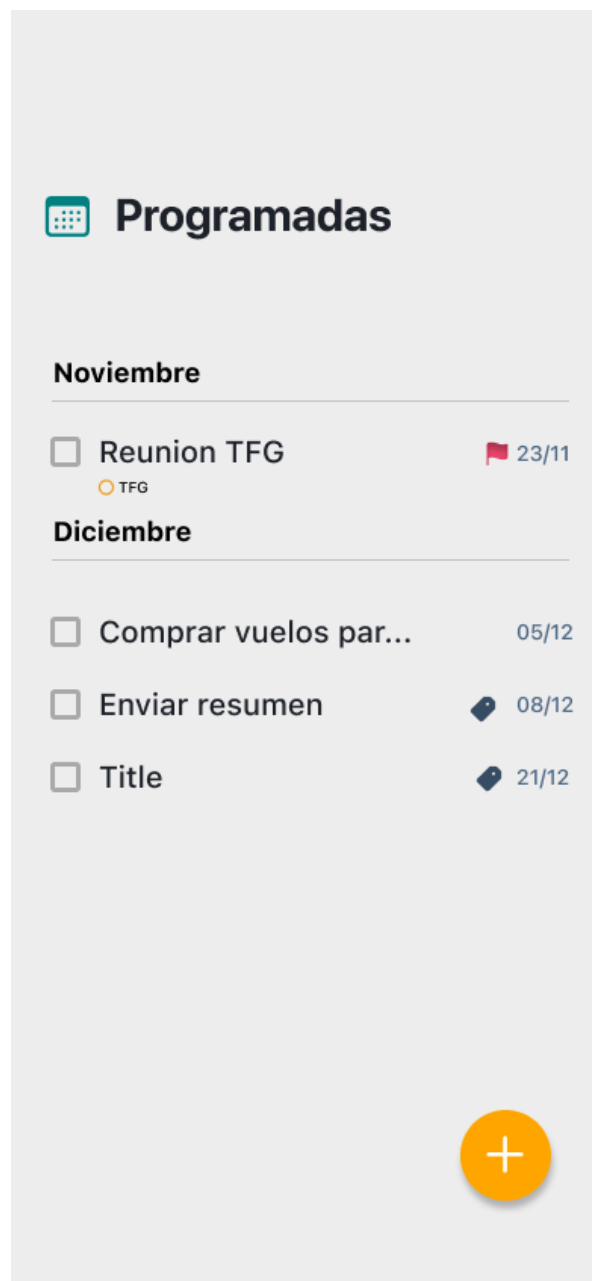


Figura 1.14: Programadas

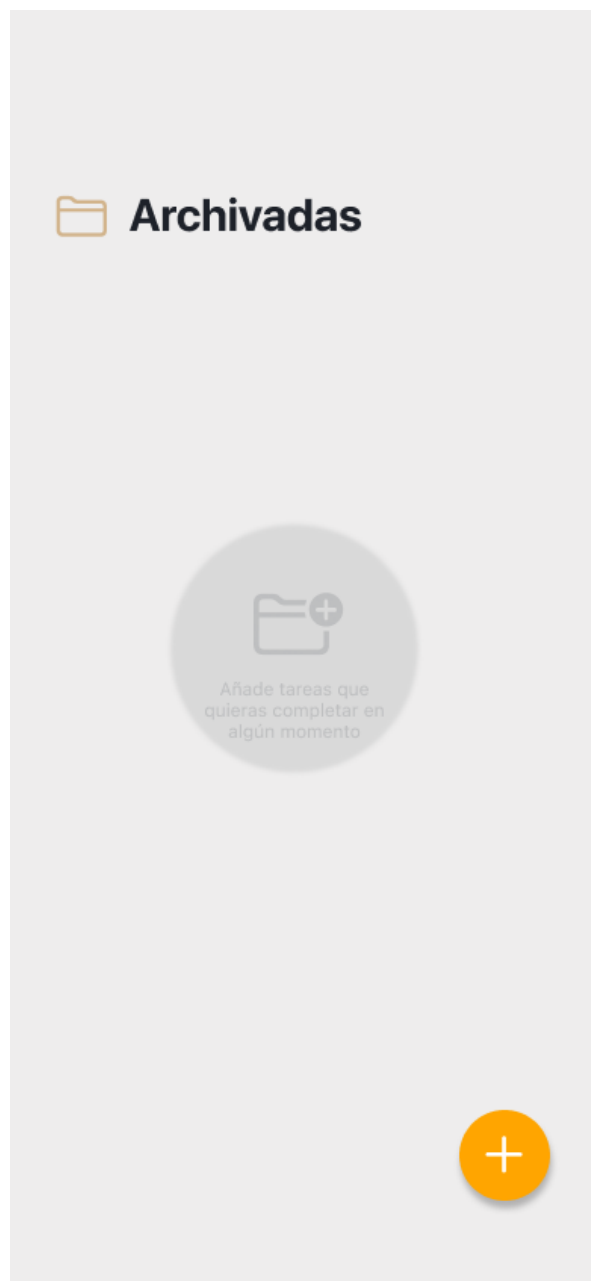


Figura 1.15: Archivadas_Screen

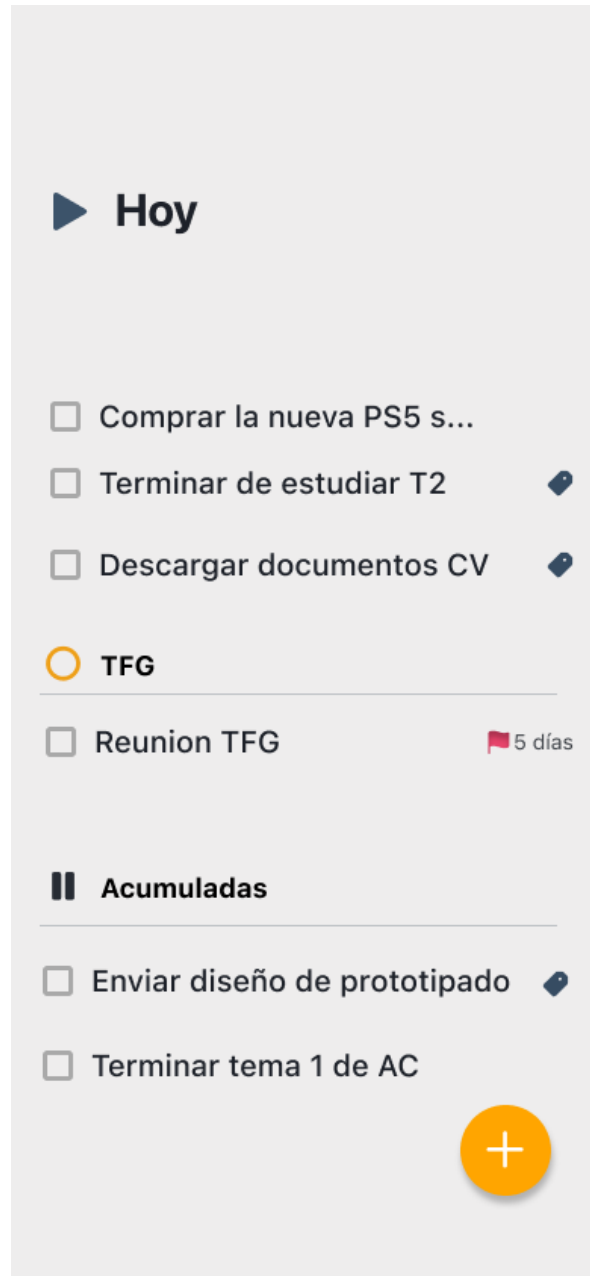


Figura 1.16: Hoy_Screen

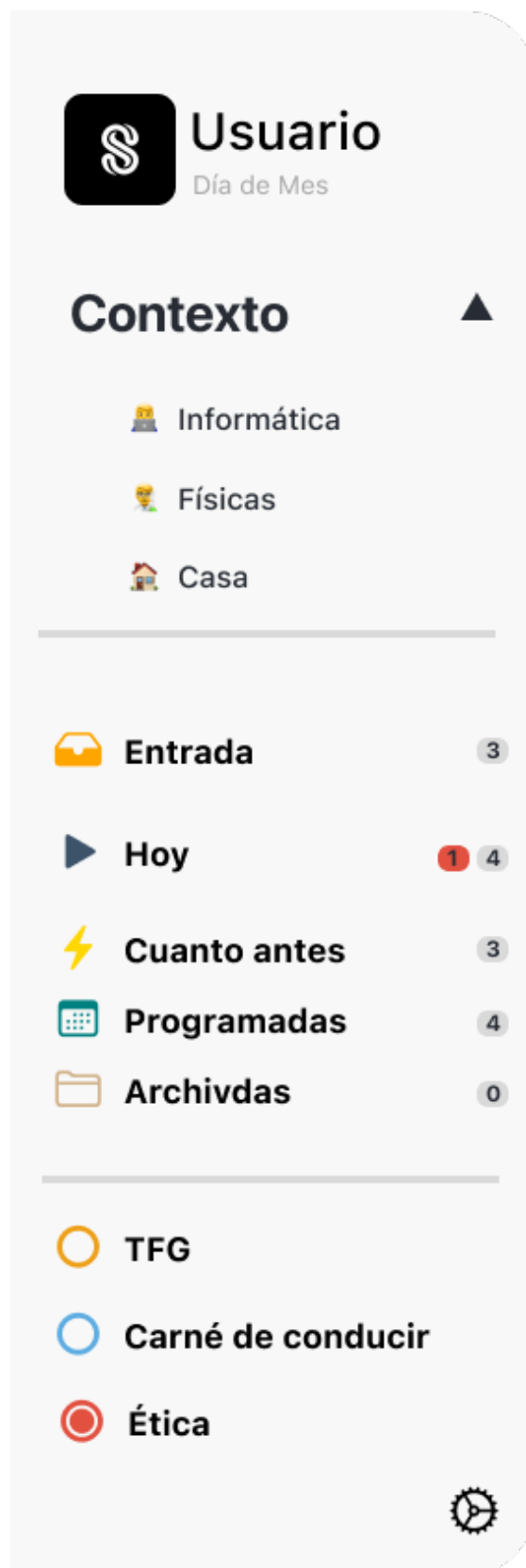


Figura 1.17: menu
20



Figura 1.18: Ajustes - Movil

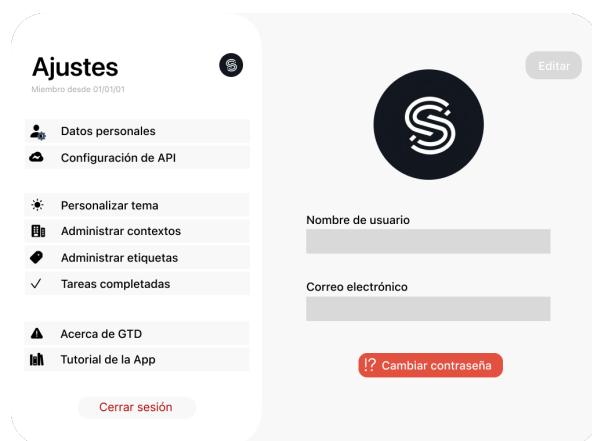


Figura 1.19: Ajustes - Escritorio

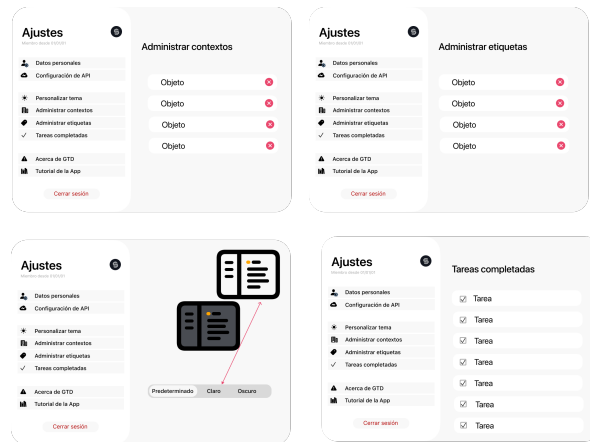


Figura 1.20: OptionSettings

para almacenar la información de las sesiones iniciadas como los *tokens* de acceso o la configuración aplicada como el tema seleccionado o los servidores configurados.

- **Markdown Display:**⁶ Renderizador de *Markdown*.
- **Modern Datepicker:**⁷ Se ha utilizado este paquete para implementar el selector de fecha para una tarea.
- **Swipeable:**⁸ Componente el cual permite implementar filas deslizables o interacción similar. Representa a sus hijos dentro de un contenedor panable que permite el deslizamiento horizontal hacia la izquierda y hacia la derecha.
- **Wheel Color Picker:**⁹ Componente seleccionador de colores. Se ha utilizado en el modal de crear proyecto para seleccionar el color del mismo.

⁶Markdown Display

⁷Modern Datepicker

⁸Swipeable

⁹Wheel Color Picker

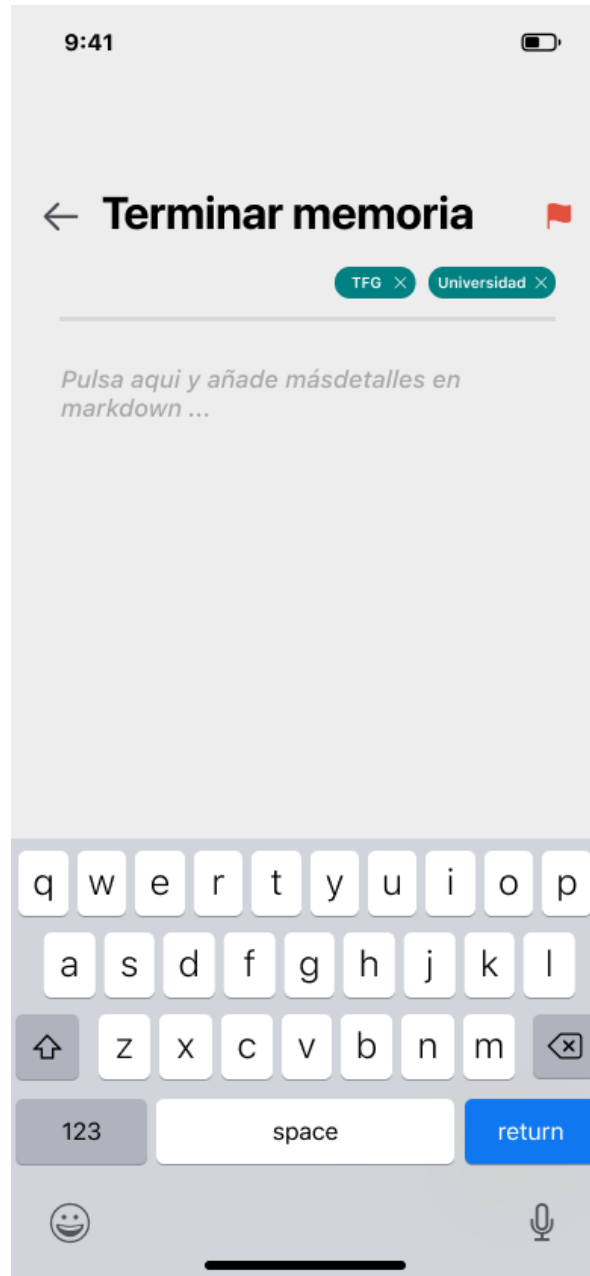


Figura 1.21: details

Reunion TFG

Descripción...

Fecha

Añadir accion

Añadir a proyecto

Registrar

Figura 1.22: editTask

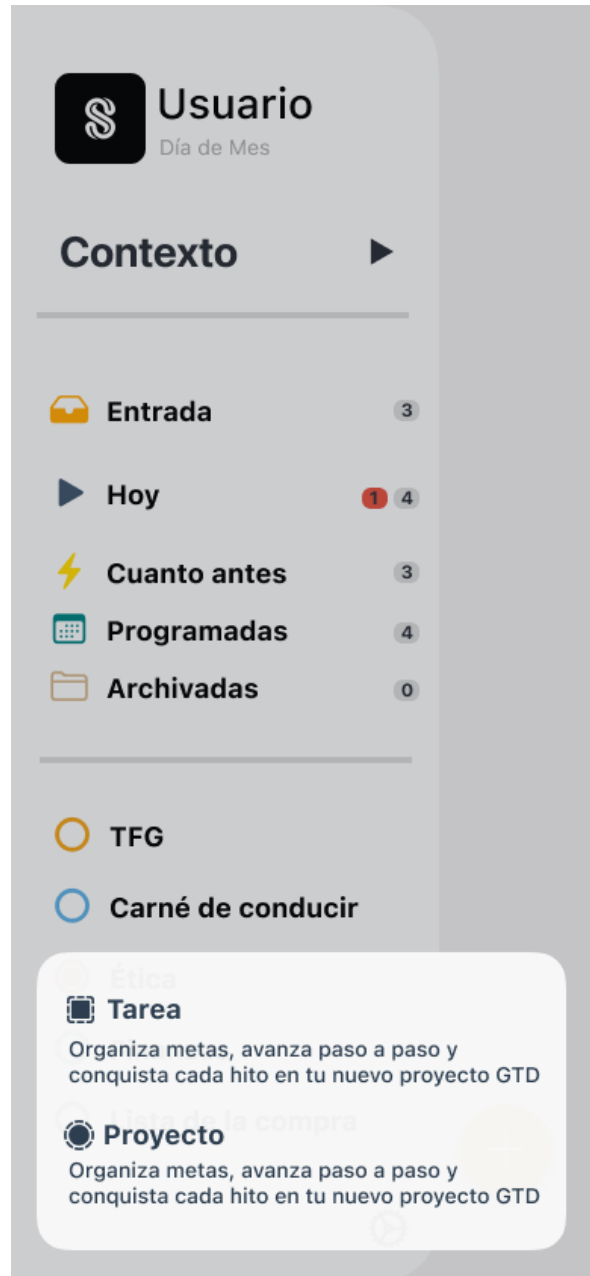


Figura 1.23: addProyectOrTask

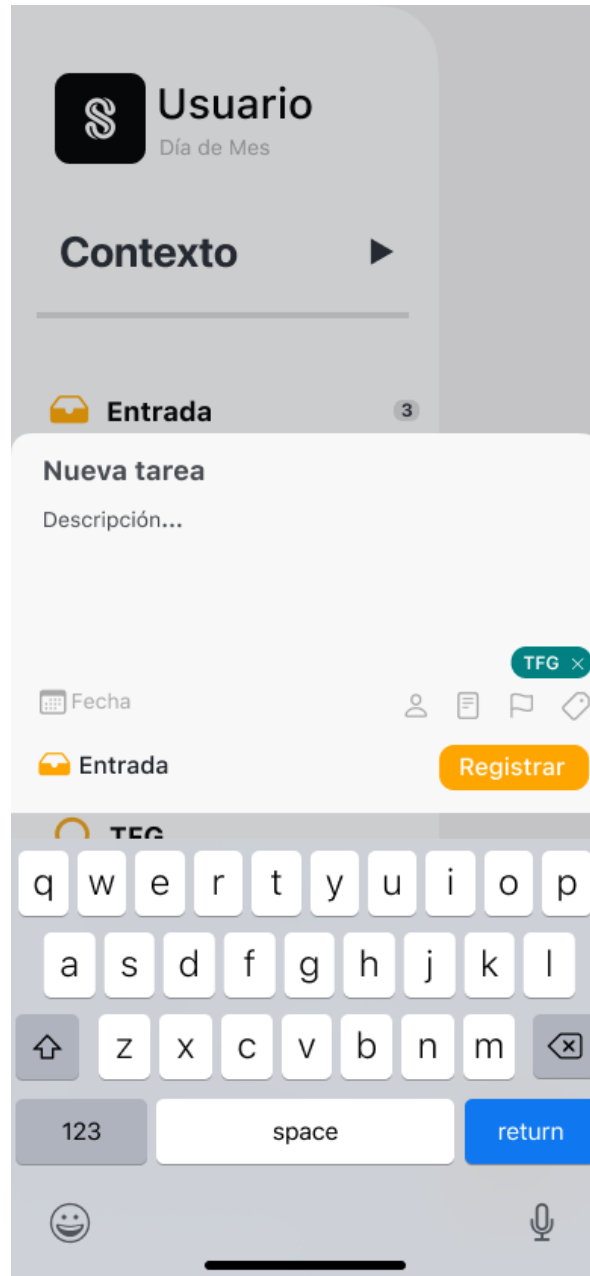


Figura 1.24: Menu add task

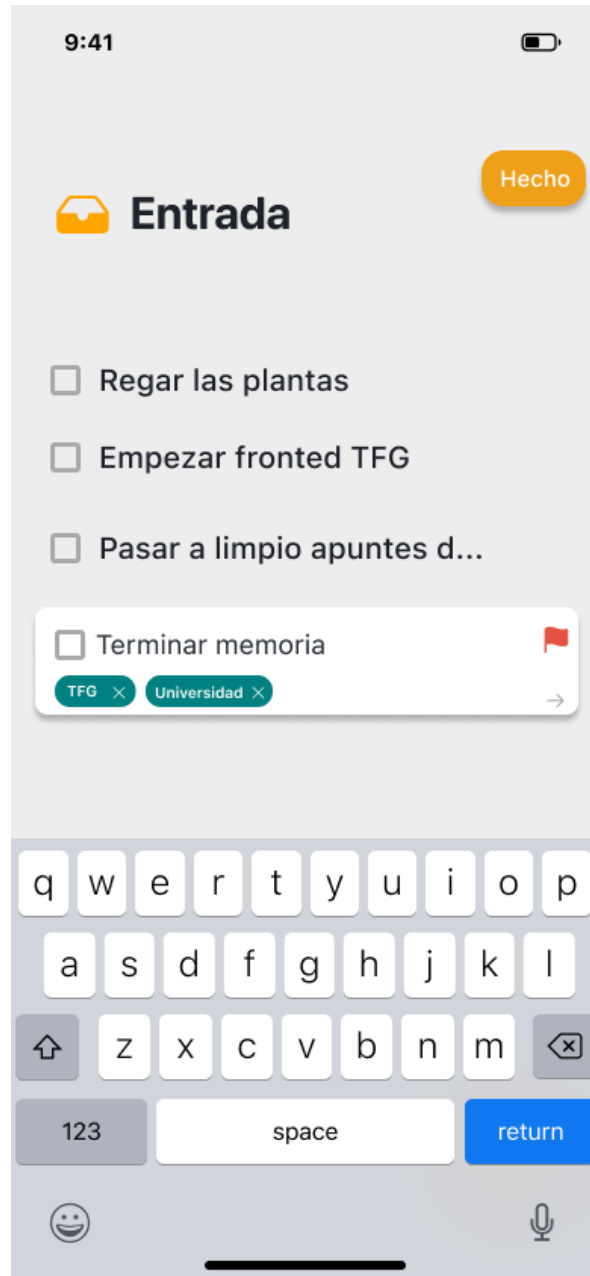


Figura 1.25: visualizacionTarea

Capítulo 2

Guía de despliegue

En este apéndice se va a detallar como 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.

2.1 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 último 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
```

2.2 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/authorize-handler.js ./node_modules"
    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:
      - postgresql-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:
  postgresql-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 crearan 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.

2.3 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 situandonos 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.

2.4 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 *Chromium* 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 *Frontend/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*.

2.5 Endpoints

2.5.1 Tarea

Endpoint	/task/
Descripción	Crea una tarea para el usuario que lo solicita

Endpoint	/task/
Método HTTP	POST
Cabecera de Autorización	Bearer token
Código HTTP (OK)	201
Código HTTP (KO)	409

Parametros	/task/	Tipo	Opcional
title (<i>Body</i>)	Titulo de la tarea	<i>String</i>	No
description (<i>Body</i>)	Descripción de la tarea	<i>String</i>	Si
state (<i>Body</i>)	Estado de la tarea	<i>Integer</i>	Si
important_fixed (<i>Body</i>)	Valor de importante	<i>Boolean</i>	Si
completed (<i>Body</i>)	Valor de completado	<i>Boolean</i>	Si
project_id (<i>Body</i>)	Proyecto de la tarea	<i>Integer</i>	Si
context_id (<i>Body</i>)	Contexto de la tarea	<i>Integer</i>	Si
date_limit (<i>Body</i>)	Fecha de la tarea	<i>Timestamp</i>	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

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

Parametros	/task/:id	Tipo	Opcional
task_id (<i>Path</i>)	id de la tarea a consultar	<i>Integer</i>	No

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

Parametros	/task/movelist	Tipo	Opcional
list_ids (<i>Body</i>)	Listado de ids de las tareas a mover de estado	<i>array[Integer]</i>	No
state (<i>Body</i>)	Estado al que se mueven las tareas	<i>Integer</i>	No

Endpoint	/task/completelist
Descripción	Completa un listado de tareas

Endpoint	/task/completelist
Método HTTP	POST
Cabecera de Autorización	Bearer token
Código HTTP (OK)	200
Código HTTP (KO)	409

Parametros	/task/completelist	Tipo	Opcio- nal
list_ids (<i>Body</i>)	Listado de ids de las tareas a mover de estado	<i>array[Integer]</i>	No
comple- ted(<i>Body</i>)	Valor de completar al que modificar las tareas	<i>Boolean</i>	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

Parametros	/task/addTag	Tipo	Opcional
task_id (<i>Body</i>)	Id de la tarea a la cual se le añade la etiqueta	<i>Integer</i>	No
tag(<i>Body</i>)	Etiqueta a añadir a la tarea	<i>Object</i>	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

Parametros	/task/:id/tags	Tipo	Opcional
task_id (<i>Path</i>)	Id de la tarea a obtener sus etiquetas	<i>Integer</i>	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

2.5.2 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

Parametros	/user/register	Tipo	Opcional
email (<i>Body</i>)	E-mail del usuario	<i>String</i>	No
name (<i>Body</i>)	Nombre de usuario	<i>String</i>	No
password (<i>Body</i>)	Contraseña del usuario	<i>String</i>	No

2.5.3 Auth

Endpoint	/oauth/authorize
Descripción	Obtiene el codigo de autorización de <i>OAuth</i> si el la autenticación es correcta.
Método HTTP	POST
Cabecera de Autorización	

Endpoint	/oauth/authorize
Código HTTP (OK)	304
Código HTTP (KO)	401

Parametros	/oauth/authorize	Tipo	Op- cio- nal
client_id (<i>Body</i>)	id del cliente a autorizar	<i>Integer</i>	No
response_type (<i>Body</i>)	<i>code</i> por defecto ya que estamos obteniendo el código de autorización	<i>String</i>	No
email (<i>Body</i>)	E-mail del usuario	<i>String</i>	No
password (<i>Body</i>)	Contraseña del usuario	<i>String</i>	No

Endpoint	/oauth/token
Descripción	Si el código de autorización es correcto devuelve el token de acceso. También permite realizar la renovación del token.
Método HTTP	POST
Cabecera de Autorización	
Código HTTP (OK)	200
Código HTTP (KO)	401

Parame- tros	/oauth/token	Tipo	Op- cio- nal
client_id (<i>Body</i>)	id del cliente desde donde se requiere el token	<i>Integer</i>	No
client_secret (<i>Body</i>)	secreto del cliente autorizado	<i>String</i>	No

Parametros	/oauth/token	Tipo	Opcional
grant_type (Body)	El proceso de <i>OAuth</i> puede ser <i>authorization_code</i> para obtener el token a partir de un código o <i>refresh_token</i> para refrescar un token ya existente	String	No
code (Body)	Si estamos en el proceso <i>authorization_code</i> 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 <i>refresh_token</i> 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

2.5.4 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

Parametros	/project/	Tipo	Opcional
title (Body)	Título del proyecto	String	No
description (Body)	Descripción del proyecto	String	Si

Parametros	/project/	Tipo	Opcional
color (<i>Body</i>)	Color del proyecto	<i>String</i>	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		
Código HTTP (OK)	200		
Código HTTP (KO)	404		

Parametros	/project/:id/complete	Tipo	Opcional
project_id (<i>Path</i>)	Id del proyecto	<i>Integer</i>	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		

Parametros	/project/:id	Tipo	Opcional
project_id (<i>Path</i>)	Id del proyecto	<i>Integer</i>	No

Endpoint	/project/:id
Descripción	Modifica el proyecto con id pasado por la url.
Método HTTP	POST
Cabecera de Autorización	Bearer token
Código HTTP (OK)	200
Código HTTP (KO)	404

Parametros	/project/	Tipo	Opcional
title (<i>Body</i>)	Titulo del proyecto	<i>String</i>	Si
description (<i>Body</i>)	Descripción del proyecto	<i>String</i>	Si
color (<i>Body</i>)	Color del proyecto	<i>String</i>	Si
completed (<i>Body</i>)	Valor de completado	<i>Boolean</i>	Si

2.5.5 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

Parametros	/context/	Tipo	Opcional
name (<i>Body</i>)	Nombre del contexto	<i>String</i>	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

Parametros	/context/:id	Tipo	Opcional
context_id (<i>Path</i>)	id del contexto	<i>Integer</i>	No

Endpoint	/context/:id
Descripción	Modifica el contexto con el id pasado por la url.
Método HTTP	POST
Cabecera de Autorización	Bearer token
Código HTTP (OK)	200
Código HTTP (KO)	404

Parametros	/context/:id	Tipo	Opcional
context_id (<i>Path</i>)	id del contexto	<i>Integer</i>	No
name (<i>Body</i>)	Nuevo nombre del contexto	<i>String</i>	Si

2.5.6 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

Parametros	/tag/	Tipo	Opcional
name (<i>Body</i>)	Nombre de la etiqueta	<i>String</i>	No

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

Parametros	/tag/	Tipo	Opcional
name (<i>Path</i>)	Nombre de la etiqueta a eliminar	<i>String</i>	No

Capítulo 3

Bibliografía