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 éstandares 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 Metodología GTD y aplicaciones **1**
1.1 ¿Qué es GTD? . . . 1
1.2 Método y técnica . . . 2
Collect . . . 2
Process . . . 4
Organize . . . 4
Review . . . 5
Do . . . 5
1.3 Aplicaciones . . . 6
1.4 Desafíos que encontramos . . . 7
1.5 Conclusiones . . . 9

2 Diseño e Implementacion del Frontend **11**
2.1 Percepción del funcionamiento del sistema . . . 11
2.2 Metáforas, Expresiones y Conceptos de diseño . . . 12
2.2.1 Proximidad y Consistencia . . . 13
2.2.2 Visibilidad . . . 14
2.3 Prototipos e Interfaces . . . 14
2.4 Implementación: ¿Qué es React Native? . . . 25

3 Guía de despliegue **27**
3.1 Estructura de carpetas del proyecto . . . 27
3.2 Despligue del *backend* . . . 28
3.3 Compilación de la aplicación web . . . 29
3.4 Compilación de la aplicación de escritorio . . . 30
3.5 Endpoints . . . 30

3.5.1 Tarea 30
3.5.2 Usuario 34
3.5.3 Auth 34
3.5.4 Proyecto 36
3.5.5 Contexto 38
3.5.6 Etiqueta 39

Bibliografía **41**

Índice de cuadros

Índice de figuras

1.1 Workflow GTD - Getting Things Done de David Allen. Fuente: [2] . . . 3

2.1 Menu lateral . . . 12
2.2 barra lateral . . . 13
2.3 Boton para añadir tarea/proyecto . . . 13
2.4 Prototipo de Bandeja de entrada, Cuanto antes, Programadas y Archivadas 15
2.5 nombre de usuario y fecha . . . 16
2.6 Detalles . . . 17
2.7 Inicio de sesión y Registro . . . 18
2.8 escritorio . . . 18
2.9 Pantalla de sección “hoy” . . . 19
2.10 Ajustes - Movil . . . 20
2.11 Ajustes - Escritorio . . . 20
2.12 Opciones de ajustes . . . 21
2.13 Editor de tareas . . . 21
2.14 Añadir tareas o proyectos . . . 22
2.15 Menu para añadir tareas . . . 23
2.16 Visualización de tarea . . . 24

Capítulo 1

Metodología GTD y aplicaciones

1.1 ¿Qué es GTD?

El desarrollo tecnológico, sumado a la creciente complejidad del mundo moderno, ha desembocado en un notable incremento de responsabilidades tanto personales como profesionales. Es por ello que, mantener la atención al detalle y la gestión eficaz del tiempo se ha vuelto un gran desafío para todo tipo de personas.

En la actualidad, existen varios métodos que afirman ser de gran utilidad para aumentar la productividad diaria y el desarrollo personal. Sin embargo, muchos de estos carecen de sentido para varios de los aspectos con los que nos topamos a diario. O bien son muy específicos, necesitando implementarlos en áreas muy concretas y no ayudan con todo tipo de tareas y actividades cotidianas, o bien, requieren de una alta curva de aprendizaje y dedicación del usuario para mantener dicha productividad que prometen.

En este contexto, David Allen, en la pasada década de los ochenta, tras años investigando para dar solución a la problemática anteriormente mencionada, creó un sistema que denominó como el método *Getting things Done* (al que nos referiremos más adelante como *GTD*), título que le daría a su libro publicado en 2001, [1]. En su obra, detalla un conjunto de técnicas que ayudarían a mejorar la productividad del individuo implementándolas a su rutina diaria.

A pesar del desarrollo tecnológico y el paso de los años, *GTD* sigue siendo uno de los métodos de referencia para las personas que buscan alejarse "del ruido" del día a día, permitiéndoles enfocarse en lo que realmente importa; tanto en los aspectos personales como profesionales. De esta forma, permite al individuo gestionar de forma óptima el tiempo del que dispone a diario. Y para ello, hace uso de un lenguaje sencillo, basándose en dos objetivos principales:

- Recopilar en un "almacén" todas las actividades que necesitan estar hechas, sin importar el momento de finalización o la importancia de las mismas.

- Obtener una disciplina que permita controlar y organizar lo anteriormente reunido en diferentes acciones.

El fijar objetivos tan abstractos y concisos, hace que *GTD* obtenga un carácter de mayor relevancia para tareas de ámbito general, sin encasillarlo en áreas muy específicas de la vida, como puede ser el trabajo, un proyecto o simples labores domésticas. Esto posibilita la integración de todo tipo de tareas, reduciendo la carga de trabajo mental que supone acordarse de todo constantemente, evitando posibles distracciones mientras nos mantenemos enfocados en lo que de verdad importa.

Por último, como comentaremos más adelante, para enfocarnos en un estado de máxima concentración, debemos seguir una serie de principios fundamentales que resumiremos en: 1.2 y 1.2.

1.2 Método y técnica

Como comentamos previamente, para seguir los principios en los que se basa, *GTD* se apoya en cinco fases "collect", "process", "organize", "review" y "do" que detallaremos en este apartado. El buen uso de estas, facilitará las gestiones cotidianas, a la par que nos libera del estrés puntual que supone la multitarea a la que el mundo moderno nos tiene acostumbrados. Asimismo, para implementar correctamente la metodología *Getting Things Done*, representada en la figura 1.1, es imprescindible seguir todos estos pasos de forma secuencial sin evitar ninguna fase, ya que de lo contrario podríamos vernos envueltos en un bucle de descontrol, desviándonos así del objetivo de la metodología explicada en el libro [1].

En su libro, Allen nos relata este apartado teniendo por analogía la *RAM* de un ordenador. Al igual que este, somos seres secuenciales, sin embargo, podemos realizar cierta multitarea y trabajar en muchos aspectos en paralelo. No obstante, para poder lograrlo, no sólo debemos tener bastante capacidad de almacenaje, sino también saber cómo administrarla de forma eficiente. Para ello, se deben seguir las cinco fases del método *GTD*:

Collect

Se trata de una de las fases fundamentales en las que se basa la filosofía *GTD*, en ella debemos recopilar todas las tareas que durante el día nos han ido surgiendo a la mente; independientemente de la importancia de las mismas, pudiendo ir desde tareas tan triviales como "limpiar los platos" a "entregar un proyecto antes de cierto plazo".

La idea es que, como indica Allen en su libro, "*saquemos todo lo que nos preocupa de la cabeza*" de forma que cada idea, tarea o asunto que nos vaya surgiendo a lo largo del día, la podamos almacenar para una organización posterior.

Sin embargo, de nada sirve que cumplamos lo anterior si periódicamente no hacemos

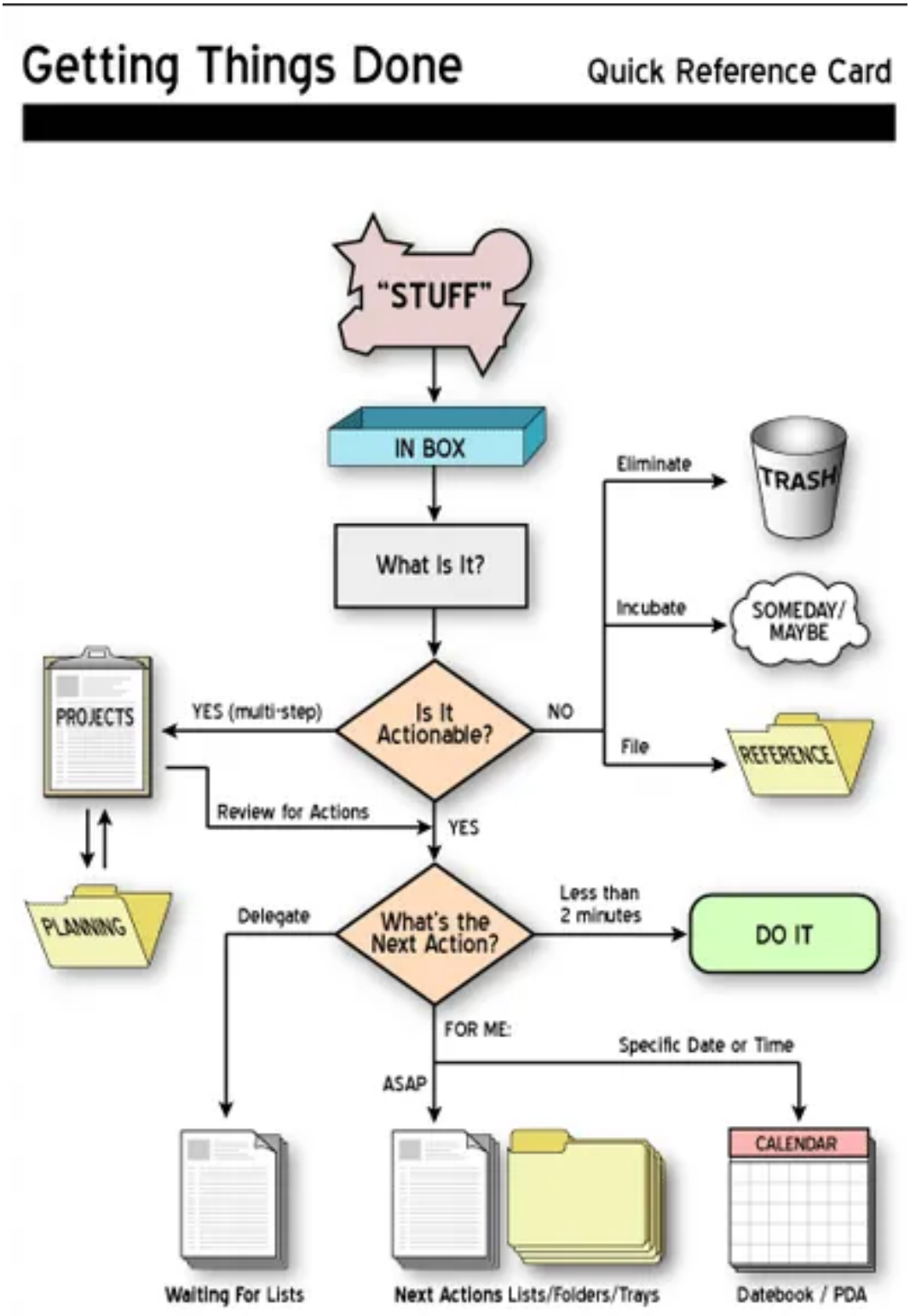

Figura 1.1: Workflow GTD - Getting Things Done de David Allen. Fuente: [2]

una limpieza a nuestra *RAM*. De lo contrario, tendríamos un cajón de sastre abocado a la procrastinación, motivo que queremos evitar con *GTD*.

Process

Una vez recopiladas, debemos encontrar un hueco para poder catalogarlas, es decir, clasificarlas y procesarlas. Para ello, llega el momento donde nos haremos ciertas preguntas como ¿qué es esto que he apuntado?, ¿requiere más de una acción? ¿necesito clarificarla?

1. "¿Qué es esto que he apuntado?"

   Al haber reunido todo lo que se nos ha venido a la cabeza, es muy común que tengamos ciertos asuntos que han perdido importancia o simplemente han perdido el sentido de ser que tenían en su momento. Además, debemos esclarecer si las tareas que tenemos por procesar requieren más de una acción o no. Si la respuesta es negativa, debemos clasificarlas en tres categorías, en caso contrario deberíamos ir directamente al *punto 2:*

   - **Eliminarla**: Si la tarea ha dejado de tener importancia, la eliminamos o la marcamos como completada.
   - **Incubarla**: Se necesita de la finalización de una tarea previa para poder empezar a trabajar con la otra.
   - **Referenciarla**: A diferencia de la incubación, en esta categoría, se almacenan las tareas que no requieren de una acción previa para poder ser finalizadas. Es decir, se almacenan tareas que contienen información relevante a la que queramos acceder en algún momento futuro.

2. "¿Necesito clarificarla?"

   Si la respuesta a esta pregunta es afirmativa, volvemos a tener 3 posibles acciones:

   - **Hacerla**: Siempre y cuando completarla nos lleve menos de 2 minutos.
   - **Aplazarla**: Si la tarea nos ocupa más de 2 minutos en completarla.
   - **Diferir su naturaleza**: Se tarda más de 2 minutos en completarla y necesitamos que se finalice una acción previa para clasificarla en los puntos 1 o 2.

Organize

En esta fase tratamos con mayor énfasis las tareas anteriormente procesadas. De esta forma, facilitamos su seguimiento y ejecución, listándolas en categorías más concretas como por ejemplo "Siguientes" o "Calendario" para tareas que requieren menos de 2 minutos y que debíamos "Diferir su naturaleza" y no tienen múltiples pasos o por el contrario, proyectos. Por otro lado, tenemos la posibilidad de listarlas en "Esperando".

Esta en sí tiene una pequeña peculiaridad ya que se trata de tareas que necesitamos *aplazarlas*, porque estamos pendientes de que finalice un evento anterior que nos permita avanzar. Sin embargo, estas pueden estar en "Siguientes" teniendo en cuenta lo mencionado anteriormente.

Por último Allen, deja claro en su libro qué tipo de material puede ir o no en cada listado:

- **Proyectos**: tareas que requieren una serie de pasos para ser completadas. Los proyectos a diferencia de las tareas, solo se usan en forma de índice y los artefactos generados se deben ir listando en otras categorías.

  Por ejemplo, si necesitamos grabar un vídeo para YouTube, en el proyecto iremos listando un índice de todo lo que debemos hacer para dar esta tarea por finalizada. Si uno de los ítems era "grabar en x localización" podremos tener otra tarea u proyecto relacionado con las labores que tengamos que hacer en dicha localización.

- **Calendario**: Tareas con acciones de tiempo específicas e información relacionada con horarios, día o semana concreta. Sin embargo, debemos tener en cuenta que no tenemos que caer en el error de utilizarlo como un listado para tareas diarias.

- **Siguientes**: Categoría encargada de listar las tareas que requieren hacerse "cuanto antes". Es decir, tareas que sin tener fecha específica de finalización, conviene acabarlas en el menor tiempo posible.

Review

Llegados a este momento, debemos realizar un seguimiento, aproximadamente semanal, en el que volvemos a ejecutar la fases anteriores observando si ha cambiado la naturaleza de alguna tarea o por el contrario existan algunas "rezagadas", evitando que estas queden en el olvido.

En otras palabras, si queremos que un ordenador funcione como el primer día, debemos ejecutar periódicamente un mantenimiento, una *fase de review*, eliminando todos los archivos que no se estén utilizando, depurando el sistema y aligerando la carga de la memoria *RAM*.

Do

Por último, tenemos la fase *Do*. En esta fase ya hemos terminado de *capturar*, *procesar*, *organizar* y *revisar* por tanto deberíamos tener un sistema coherente y preciso. No obstante, Allen en su libro nos explica que aún podemos mejorar el sistema, ya que si sobrecargamos la categoría "Siguientes", podríamos llegar a cuestionarnos, ¿Por qué tarea deberíamos empezar? La respuesta corta es seguir nuestra intuición, sin embargo, esto puede que no sea correcto en algunos casos. Por ello, a pesar de que en el libro

nos explican tres modelos, creemos que para mantener una visión simple e intuitiva deberíamos seguir uno de los tres, en concreto, el modelo de los *cuatro criterios*:

1. **Contexto**: conjunto de acciones que se pueden realizar según la localización en la que nos encontremos.
2. **Tiempo disponible**: ¿Disponemos de suficiente tiempo para ejecutar esta tarea?
3. **Energía disponible**: ¿Sabiendo el gasto de energía que supone realizar esta tarea, es factible ejecutarla ahora mismo?
4. **Prioridad**: ¿Existe alguna otra tarea que podamos realizar y tenga mayor importancia sobre otras?

1.3 Aplicaciones

En este apartado trataremos de cumplimentar todas las fases anteriormente descritas, teniendo en cuenta las herramientas que nos proporciona Allen en su libro, aunque adaptadas a la década en la que nos encontramos.

Por lo general, el libro nos da ciertas herramientas para que podamos implementar la filosofía *GTD* como una extensión más allá de nuestra rutina cotidiana. Sin embargo, debemos tener en cuenta que el libro fue escrito a comienzos de los años dos mil. Por ello, en lugar de usar términos como "papeleras", "notas de voz" o "apuntes en papel", usaremos herramientas actuales que vemos todos los días, tales como los ordenadores, móviles o *iPads*, de forma que podamos integrarlo con mayor facilidad en nuestro día a día.

Basándonos en esta idea, hemos explorado el mercado de aplicaciones que existen tanto en *App Store* y *Google Play*, encontrándonos con las aplicaciones más destacadas para poder implementar la metodología, siendo estas:

- **NirvanaHQ**: Aplicación web y móvil diseñada específicamente para *GTD*. Ofrece características comolistas de proyectos, listas de acciones y enfoque en el flujo de trabajo *GTD*.
- **TodoIst**: Aplicación de gestión de tareas. Permite crear listas de tareas, recordatorios, establecer etiquetas y organizar tus tareas según los principios básicos de *GTD*.
- **Notion**: *Notion* es una herramienta de productividad versátil que puede personalizarse para adaptarsea *GTD*. Puedes crear bases de datos, tablas y tableros para organizar tus tareas y proyectos.
- **Things**: Se trata de una aplicación de gestión de tareas disponible para dispositivos *Apple*. Tiene una interfaz elegante y herramientas que se alinean con *GTD*, como áreas, proyectos y tareas.

- **Omnifocus**: Ofrece una amplia gama de características de organización y permite crear proyectos, tareas y contextos para seguir la metodología *GTD*.
- **TickTick**: *TickTick* es una aplicación de gestión de tareas con múltiples características que pueden utilizarse para implementar *GTD*. Ofrece listas de tareas, calendario y recordatorios.

1.4 Desafíos que encontramos

Cuando planteamos este trabajo de fin de grado teníamos como premisa, fusionar lo mejor de cada aplicación que nos encontramos en el mercado en una sola, dando así una solución completa para todo tipo de usuarios.

A pesar de que las apps mencionadas en el anterior apartado son excelentes, observamos ciertas deficiencias en cada una de ellas que hacen que dificulten el acceso y la implementación del método de David Allen, tal y como detallamos en la siguiente tabla:

Aplicación	**¿Multiplataforma?**	**Aspectos positivos**	**Aspectos negativos**
NirvanaHQ	Si	Diseñada especialmente para GTD	Interfaz demasiado simple pudiendo confundirse
TodoIst	Si	Interfaz intuitiva y fácil de utilizar Soporte multiplataforma Funciones avanzadas de recordatorio, etiquetas y filtrado	Versión gratuita con muchas restricciones Difícil integración de *GTD* ya que carece de la función de creación de proyectos
Notion	Si	Altamente personalizable y con grandes capacidades (Uso de base de datos, tablas...) Se puede integrar gran cantidad de métodos no solo *GTD*	Alta curva de aprendizaje combinado con un sentimiento abrumador dada la cantidad de características y flexibilidad que ofrece
Things	Solo dispositivos *Apple*	Diseño elegante, minimalista y centrado en *GTD*	Uso reducido a dispositivos *Apple* Carece de características avanzadas
Omnifocus	Solo dispositivos *Apple*	Alta capacidad de organización para proyectos, tareas y manejos de contextos	Uso reducido a dispositivos *Apple* Alta curva de aprendizaje
TickTick	Si	Interfaz intuitiva y fácil de usar	La versión gratuita difiere bastante de la versión premium

Teniendo esto en cuenta, debemos tener en consideración que no solo hay falta una aplicación con una buena interfaz y que integre todas las características del método, sino que además debe ser fácil e intuitiva de utilizar para que se pueda poner en práctica diariamente.

1.5 Conclusiones

En resumen, la filosofía *Getting Things Done* pone por escrito muchas de las funciones que usualmente tenemos en mente, pero que no conseguimos ejecutar correctamente por motivos diversos, estando estos relacionados con la organización. Es por esto por lo que tener fases bien definidas, permite gestionar mejor el día a día, consiguiendo enfocarnos en lo que realmente importa.

A pesar de que, como todo método, puede presentar ciertas dificultades, merece la pena implementarlo en la rutina diaria pues tiene una corta curva de aprendizaje siendo fácilmente adaptable a múltiples facetas de la vida, a diferencia de otros métodos y/o técnicas (p.ej.: técnica pomodoro).

Teniendo esto en cuenta, detallaremos en el Capitulo 6 **??** la implementación del frontend de la app, así como qué características destacamos para hacer llegar este método a los dispositivos personales de todo individuo.

Capítulo 2

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:

2.1 Percepción del funcionamiento del sistema

Para empezar a diseñar la app, investigamos, no solo como funciona el método explicado en el capítulo 1, 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.

2.2 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 2.1:

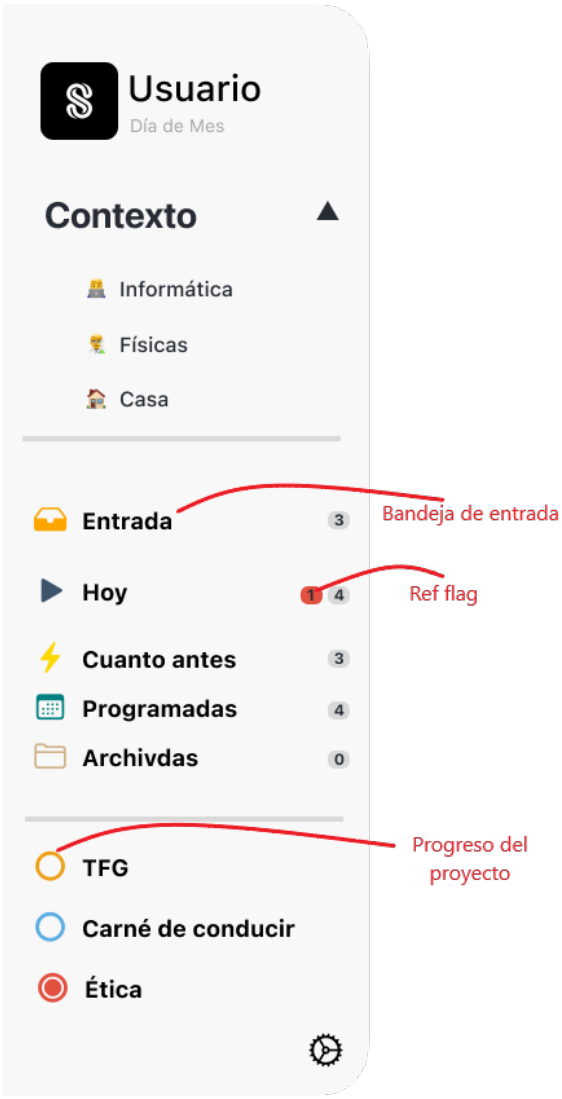

Figura 2.1: Menu lateral

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 2.2 de “Entrada”, “Hoy”, “Cuanto antes”, “Programadas”, etc.

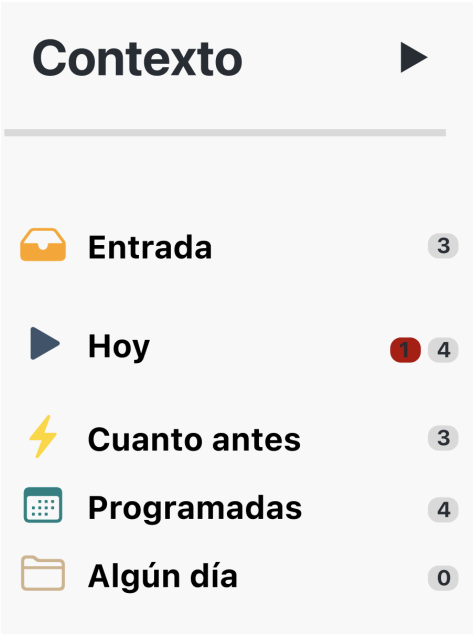

Figura 2.2: barra lateral

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 2.3 Para explorar con mayor detalle estos aspectos, nos hemos basado en varios principios de diseño que describiremos a continuación.

Figura 2.3: Boton para añadir tarea/proyecto

2.2.1 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 2.2

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 2.4 y aprendizaje de la interfaz por parte del usuario final.

2.2.2 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" 2.4, 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 2.5 Por otro lado, las etiquetas y el contexto asociado a cada tarea en la pantalla de "detalles" 2.6 dirigen la atención a lo que realmente importa.

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" 2.2 y el número de tareas que tienen tanto, mientras los demás ocupan un segundo plano.

2.3 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.

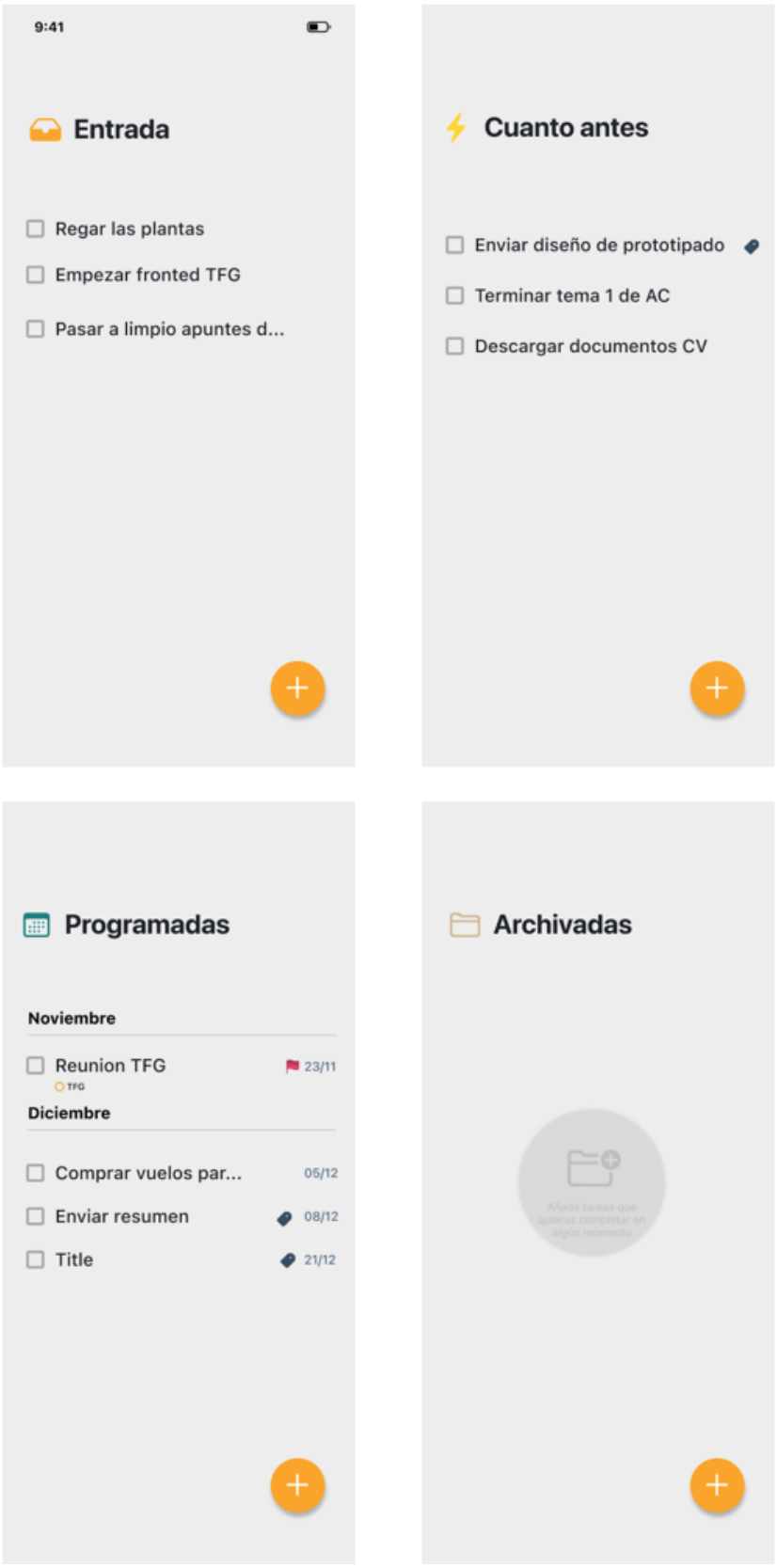

Figura 2.4: Prototipo de Bandeja de entrada, Cuanto antes, Programadas y Archivadas

Figura 2.5: nombre de usuario y fecha

Asimismo, usamos gran parte de las opciones que ofrece para probar varios de los 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 movil, 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 en dispositivos móviles 2.4, además del prototipo de escritorio:

En adición a las categorias de *GTD* mencionadas en el capítulo 2, hemos querido añadir una nueva categoría "**Hoy**" 2.9 que actúe 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.

Figura 2.6: Detalles

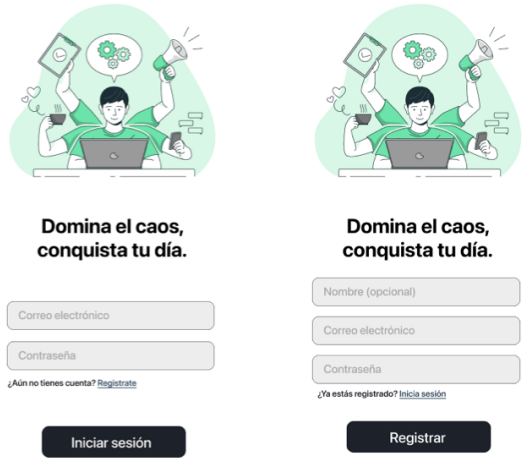

Figura 2.7: Inicio de sesión y Registro

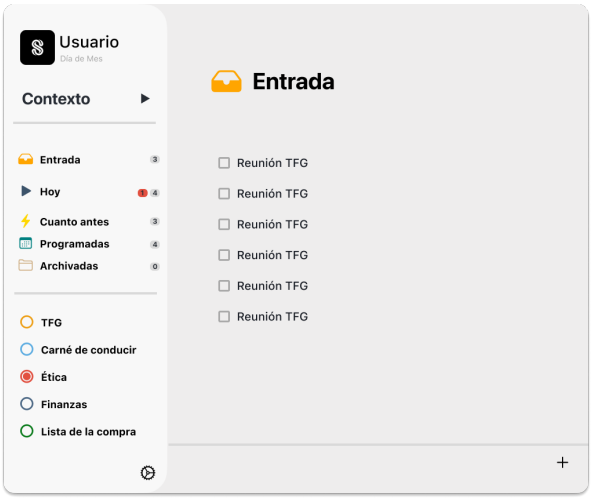

Figura 2.8: escritorio

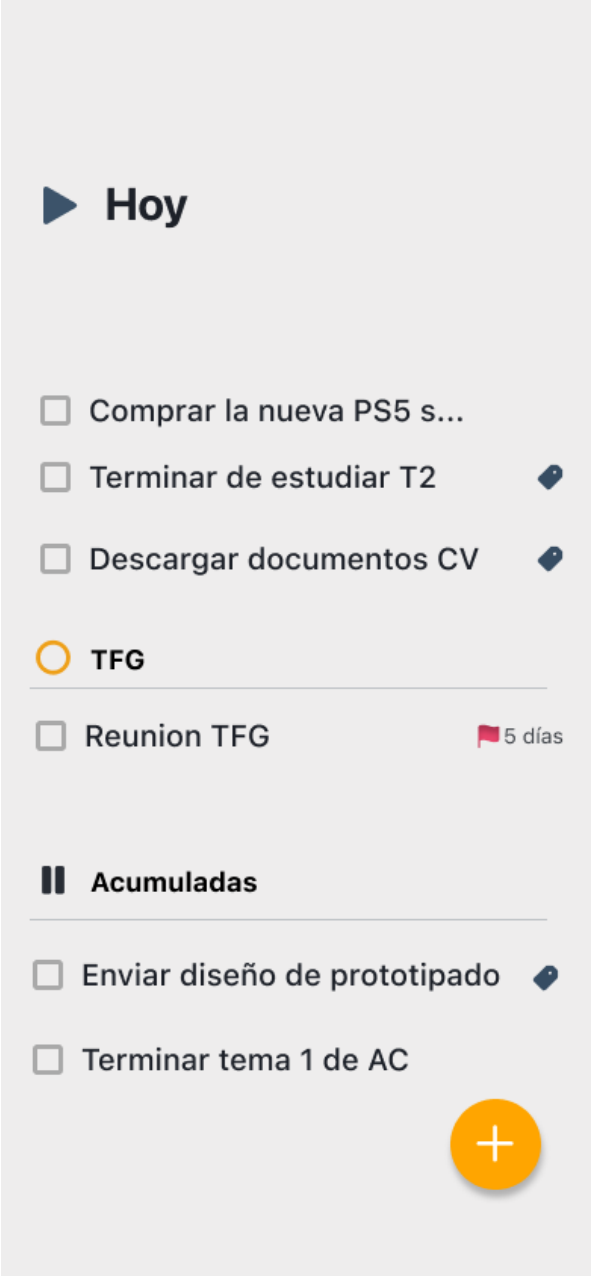

Figura 2.9: Pantalla de sección “hoy”

Figura 2.10: Ajustes - Movil

Figura 2.11: Ajustes - Escritorio

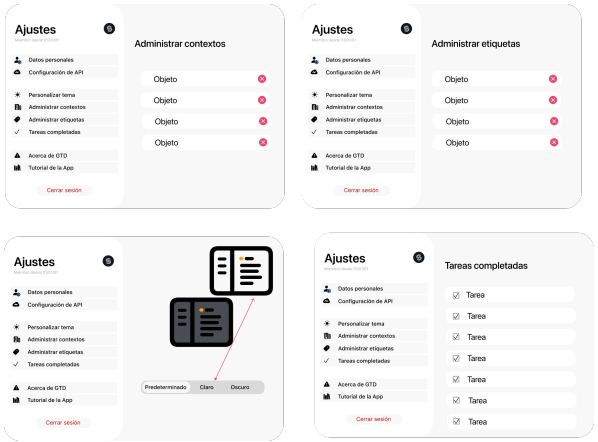

Figura 2.12: Opciones de ajustes

Figura 2.13: Editor de tareas

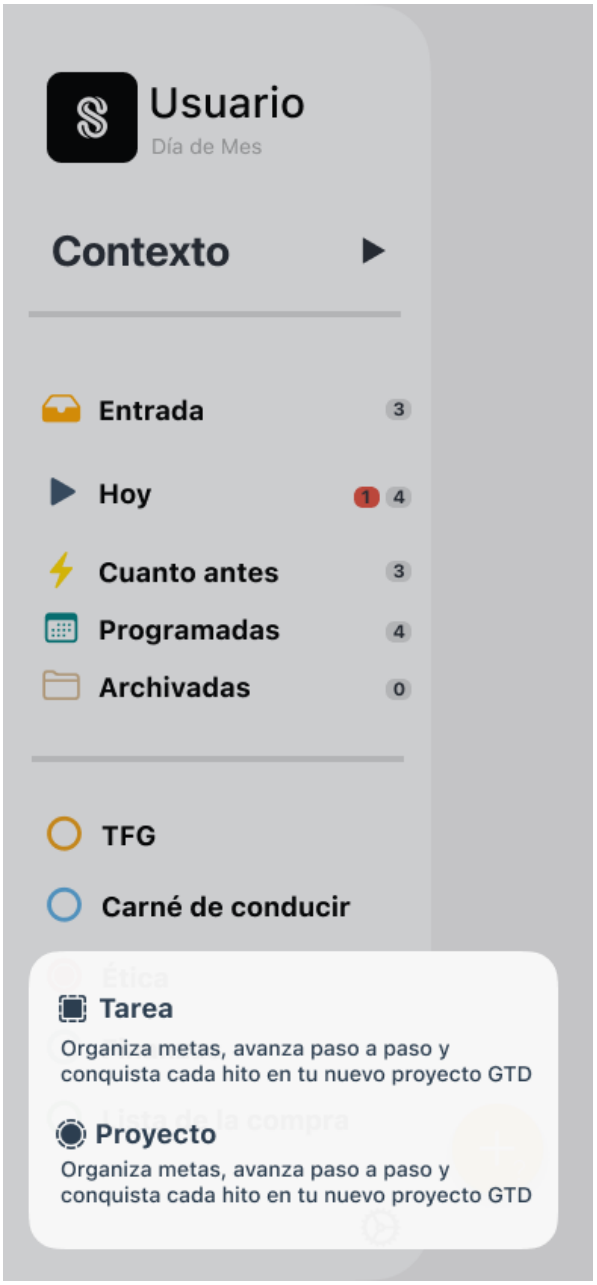

Figura 2.14: Añadir tareas o proyectos

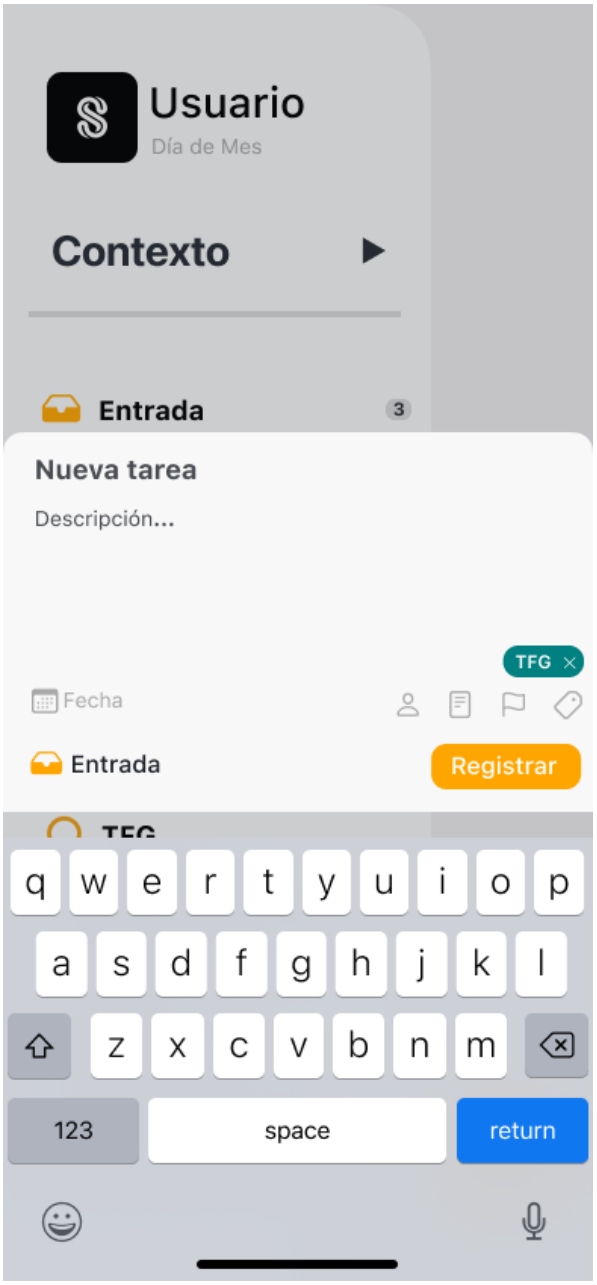

Figura 2.15: Menu para añadir tareas

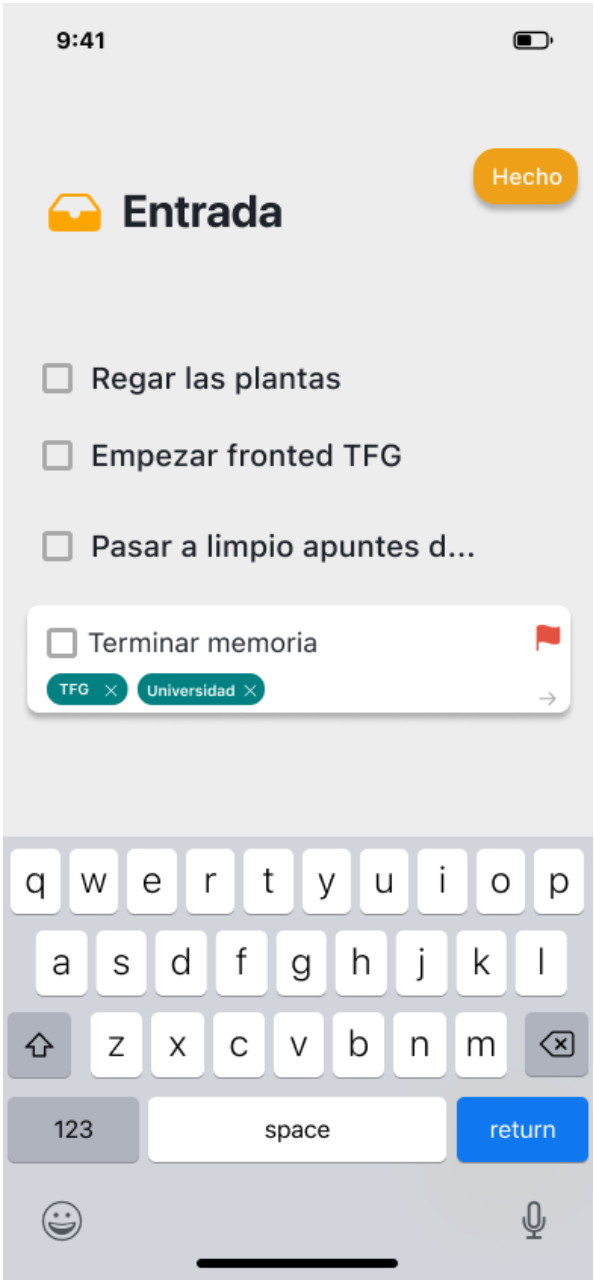

Figura 2.16: Visualización de tarea

2.4 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*).

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 enrrutar 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 envio de *requests* y *respones*, 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

[1]React Navigation
[2]Drawer Navigation
[3]Axios
[4]React Native Calendars
[5]React Native Async Storage

desarrollo ya que permite guardar información en el dispositivo. Se ha utilizado 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.

[6]Markdown Display
[7]Modern Datepicker
[8]Swipeable
[9]Wheel Color Picker

Capítulo 3

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.

3.1 Estructura de carpetas del proyecto

El proyecto está formado por 3 directóricos 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 útlimo podemos encontrar el directorio *Frontend* el cual contiene el código fuente referente a la aplicación cliente, este código se cuentra 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 Despligue 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 ajustandolo 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_
    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 lo 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 definicion 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.

3.3 Compilación de la aplicación web

Requisitos

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

Las versiónes 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.

3.4 Compilación de la aplicación de escritorio

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

Las versiónes 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 aplicacion 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 apliación de escritorio. Es importante recalcar que tal compilación se hará para la misma plataforma donde se realize la misma, por ejemplo si deseamos compilar para *windows* se deberá de realizar este proceso en *windows*.

3.5 Endpoints

3.5.1 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 (*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

Parametros	/task/:id	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

Parametros	/task/:id	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
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 (*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

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

Parametros	/task/completelist	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

Parametros	/task/addTag	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

Parametros	/task/:id/tags	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

3.5.2 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 (*Body*)	E-mail del usuario	*String*	No
name (*Body*)	Nombre de usuario	*String*	No
password (*Body*)	Contraseña del usuario	*String*	No

3.5.3 Auth

Endpoint	/oauth/authorize
Descripción	Obtiene el codigo de autorización de *OAuth* 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	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	
Código HTTP (OK)	200
Código HTTP (KO)	401

Parametros	/oauth/token	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

Parametros	/oauth/token	Tipo	Opcional
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

3.5.4 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*)	Titulo del proyecto	*String*	No
description (*Body*)	Descripción del proyecto	*String*	Si

Parametros	/project/	Tipo	Opcional
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
Código HTTP (OK)	200
Código HTTP (KO)	404

Parametros	/project/:id/complete	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

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

Parametros	/project/	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

3.5.5 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 (*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

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

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

3.5.6 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 (*Body*)	Nombre de la etiqueta	*String*	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 (*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] «Getting Things Done: The Art of Stress Free Productivity by David Allen (Nicole Schlinger Book Review)». https://medium.com/@nicoleschlinger/getting-things-done-the-art-of-stress-free-productivity-by-david-allen-nicole-schlinger-book-688e1440290d.