



PROYECTO FIN DE CICLO



miTiempo

miTiempo

Autor: Pablo Herrero Sánchez

Tutor individual y colectivo: Jose Luis Arias Cobreros

Ciclo Formativo de Grado Superior en Desarrollo de Aplicaciones Multiplataforma

Curso 2020/2021

Sotrondio, Junio 2021

A mi tutor, Jose Luis Arias Cobreros, y a todos los profesores de D.A.M. del I.E.S. Juan José Calvo Miguel, sin cuyo apoyo, ayuda, labor educativa y guía hubiera sido imposible el abordaje de este proyecto y la finalización de este Ciclo Formativo.

No puedo dejar de dedicar este proyecto y de agradecer a Sara todo su amor, paciencia, generosidad y apoyo incondicional, siendo compañera, luz y guía no sólo en este recorrido, sino en la vida.

A Alejandro, Omar y Pablo, compañeros y camaradas, fieles defensores del código y maestros en el día a día sin los que hubiera sido infinitamente más difícil alcanzar la meta.

A mi familia, indispensable y vital.

A The Beatles, Led Zeppelin y The Rolling Stones, banda sonora durante el desarrollo.

Contenido

MEMORIA DEL PROYECTO.....	5
Resumen de la motivación	5
Objetivos y alcance del proyecto.....	5
Objetivos.....	5
Alcance	6
INTRODUCCIÓN.....	6
Justificación del proyecto.....	6
Estudio de la situación actual.....	7
Evaluación de las alternativas.....	7
Aspectos teóricos.....	9
FASES.....	9
FORMACIÓN	10
ANÁLISIS	10
Requisitos.....	10
Requisitos no funcionales	10
Requisitos funcionales.....	12
Especificación casos de uso	15
Actores del sistema.....	15
Diagrama de casos de uso	15
Especificación de casos de uso.....	16

DISEÑO DEL SISTEMA	22
Arquitectura del sistema.....	22
Landing page	23
Frontend	24
Backend.....	25
Base de datos.....	26
Diseño de la base de datos	27
Diseño de la interfaz.....	29
Prototipado	29
Flujo de pantallas	33
IMPLEMENTACIÓN DEL SISTEMA	34
Estándares y normas seguidos.....	34
Estándares Javascript	34
Buenas prácticas en React y Node.js	35
Clean Code	36
Lenguajes de programación.....	36
Tecnologías y herramientas	36
Tecnologías.....	36
Frameworks.....	37
Frameworks UI	37
Editores	37
Herramientas.....	38
Herramientas de diseño y prototipado	38

APIs y servicios	38
Paquetes y librerías	38
Deploy	39
Control de versiones y organización del proyecto.....	40
Creación del sistema	40
Preparación y planificación del desarrollo.....	40
Backend.....	42
Frontend	58
Landing page	94
Problemas encontrados	100
Despliegue	105
DOCUMENTACIÓN	115
Manual de instalación.....	115
Manual de usuario.....	116
MEJORAS FUTURAS.....	123
CONCLUSIONES.....	123
REFERENCIAS BIBLIOGRÁFICAS	124
APÉNDICES.....	126
Glosario	126
Contenido entregado	130
Diario de desarrollo.....	130

MEMORIA DEL PROYECTO

Resumen de la motivación

Cursando el segundo trimestre de 2º de DAM, amenazado por los exámenes, al acecho de tutorías, rodeado de materia pendiente de estudiar y de tareas por hacer, me di cuenta de lo importante que es saber organizarse y tener control sobre nuestro tiempo, no existiendo aplicaciones sencillas, directas e intuitivas para ello.

Ante esta necesidad, **miTiempo** nace como una app de productividad multiplataforma (web, Android e iOS) que integra un gestor de tareas, un temporizador basado en Pomodoro y un rastreador de tiempo (time-tracker), permitiendo temporizar y planificar tareas y rutinas, logrando que el usuario saque el máximo provecho de su tiempo y maximizar su rendimiento. Es ideal para cualquier persona que quiera organizarse de forma adecuada, siendo especialmente aconsejable para estudiantes y profesionales de cualquier sector.

A su vez, este proyecto se plantea como un desafío donde poner en práctica no sólo los conocimientos adquiridos a lo largo del ciclo, sino estudiar e implementar nuevas e interesantes tecnologías.

Objetivos y alcance del proyecto

Objetivos

El objetivo del proyecto es la realización de una **aplicación multiplataforma** que permita a cualquier usuario, particular o profesional, gestionar su tiempo de la forma más productiva y sencilla posible.

Para ello, esta app será capaz de:

- **Gestionar las tareas** del usuario mediante un interface amigable, llamativo e intuitivo.
- Permitir la ejecución de las tareas del usuario implementando la técnica **Pomodoro** si así se desea.
- Implementar la operatividad de un **time-tracker**, dando al usuario un feedback visual del tiempo implementado en sus tareas y los distintos tipos de las mismas.

Aunque de cara a este proyecto se pretendía implementar funciones automáticas para el time-tracker (detección de aplicaciones y dispositivos durante la ejecución de una tarea), así como funciones avanzadas de concentración (impedir que el usuario utilice el terminal móvil mientras se ejecuta una tarea), estas funciones se delegan a una segunda versión de la

aplicación, ya que las mismas no pueden ser desarrolladas en el intervalo de tiempo que compete a este proyecto.

Alcance

Se pretende desplegar una **primera versión operativa**, donde el usuario pueda registrarse, crear, planificar y ejecutar tareas, así como ver un registro del tiempo usado en ellas.

Para ello, se pretende investigar e implementar la realización de un **desarrollo full-stack**, a partir de la construcción de un frontend (móvil y web) que se conecta a un backend, el cual intercambia información con una base de datos no documental, así como una *landing page* que sirve de acceso y gancho a nuevos usuarios.

Todas las tecnologías se tratan de desarrollar desde un punto de vista **Cloud**, dada la repercusión que los servicios en la nube están teniendo actualmente y las ventajas que provéen. Por ello, el front web y el backend son desplegados en un servicio en la nube (Heroku), la *landing page* es alojada en un hosting (Netlify) y la base de datos corre desde un cluster (MongoDB).

Por otro lado, la implementación de un sistema de control de versiones y el uso de paquetes y librerías anexas se antoja totalmente imprescindible para abordar este proyecto.

INTRODUCCIÓN

Justificación del proyecto

Se pretende realizar un uso integral de todos los **conocimientos adquiridos** a lo largo del ciclo formativo, aplicando y ampliando lo aprendido durante estos años.

Pese a que podría enmarcarse inicialmente en el **marco** de la asignatura de Programación Multimedia y Dispositivos Móviles, la mayor parte del resto de asignaturas confluyen en esta aplicación, desde asignaturas troncales como Programación, Lenguajes de Marcas, Bases de Datos, Entornos de Desarrollo o Sistemas Informáticos, hasta asignaturas más específicas como Acceso a Datos o Desarrollo de Interfaces, e incluso Lengua Extranjera, haciéndose la escritura del código y el interface de usuario en lengua inglesa.

Por otro lado, cabe destacar que para este desarrollo se hace uso del stack **MERN**, uno de los stacks o pilas tecnológicas más innovadoras, más usadas y de mayor repercusión actualmente en el sector, formado por MongoDB, Express, React y Node.js. Todas estas tecnologías no se han visto a lo largo de las asignaturas del ciclo, por lo que este proyecto me permite realizar una inmersión y un estudio intensivo de las mismas, salvo en el caso de

MongoDB, de la cual se abordó un curso introductorio en la asignatura Acceso a Datos, requiriendo una ampliación y profundización.

También es reseñable el uso de **herramientas** de vanguardia en el desarrollo, muchas de ellas no abarcadas a lo largo del curso, incluyendo entornos de desarrollo, herramientas de desarrollo, librerías, frameworks de UI, herramientas de diseño, herramientas de despliegue (deploy), APIs o sistemas de gestión de versiones.

Teniendo todo lo expuesto en cuenta, este proyecto me permite **aplicar y ampliar mi formación** de cara a encontrar mi primera oportunidad laboral en el sector.

Estudio de la situación actual

La importancia de poder gestionar el tiempo y aumentar en la medida de posible la productividad es clara en los tiempos que corren. Sea un entorno profesional o personal, los usuarios se acogen a calendarios, gestores de tareas, pomodoros y trackers para organizar su día a día. De hecho, grandes empresas como Apple o Microsoft, han desplegado sus propias versiones de este tipo de programas, siendo cada vez más las compañías que se aventuran al desarrollo de este tipo de soluciones. **La gestión del tiempo es tendencia.**

Actualmente existe una basta oferta de gestores de tareas, pomodoros y trackers de tiempo en el mercado. Curiosamente no se encuentran, o servidor no lo ha conseguido, **apenas alternativas que abarquen las tres opciones** deseadas para este proyecto: gestor de tareas, técnica Pomodoro y tracker de tiempo. Tenerlas unificadas en un solo programa, es una opción atractiva para el usuario, simplificando su día a día y repercutiendo favorablemente en su productividad para cualquier tipo de tarea que quiera realizar.

Evaluación de las alternativas

Como se indicaba en el apartado anterior, realmente hay pocas alternativas en el mercado donde confluyan las tres características principales de miTiempo: gestor de tareas, Pomodoro y tracker de tiempo. Teniendo en cuenta esto, a continuación se destacan las aplicaciones más llamativas que abarcan alguna o varias de esas características, y terminaré detallando cómo miTiempo las mejora o trata de tomar lo mejor de ellas:

- **Things 3:** Es un renombrado gestor de tareas. Destaca, además de por su cuidado interface, por permitir organizar las tareas por categorías, por proyectos, incluye entradas rápidas, una opción de posponer una tarea para la última hora del día y enviar un correo a Things para que se transforme en una tarea dentro de la app. Como desventaja, cabe destacar que Things 3 no es gratuito y es exclusivo para sistemas iOS y MacOS.

- **PomoDoneApp:** Se trata de aplicación multiplataforma muy interesante que abarca un gestor de tareas junto con un temporizador Pomodoro muy completo. Permite crear tareas e implementar la técnica Pomodoro de una forma muy sencilla, siendo muy interesante su integración con un sinfín de aplicaciones y servicios. Su único inconveniente es que carece de un tracker incorporado.
- **Forest:** Es una app de productividad disponible para iOS, Android y como extensión para Chrome que promueve que el usuario se concentre en lo que quiere o necesita hacer. Implementando la gamificación, por cada tarea, y según la duración, el usuario recibe árboles con los que va generando bosques. Estos bosques son realmente gráficas que muestran el tiempo en el que el usuario ha estado concentrado durante un determinado periodo de tiempo. Pocas desventajas tiene esta aplicación, la sencillez, su gran virtud, quizás sea también su gran desventaja, no incluyendo ninguna otra característica aparte de la anteriormente expuesta.
- **Focus To-Do:** App multiplataforma muy completa. Integra un gestor de tareas con la técnica Pomodoro, incluyendo recordatorios, notas, subtareas, repeticiones y un interesante sistema de feedback al usuario del tiempo mediante gráficas muy interesantes. Es una opción muy interesante con el hándicap de contar, quizás, con demasiadas opciones y no contar con la suficiente repercusión en el mercado.
- **Toggl:** Este time-tracker multiplataforma se centra exclusivamente en medir el tiempo del usuario. Este debe introducir el nombre de la tarea que va a realizar, darle a play y parar el temporizador cuando termine. De esta forma, la aplicación va almacenando datos y mostrándoselos al usuario mediante gráficas. Toggl cumple perfectamente su función y en determinados entornos puede ser ideal, aunque quizás sea demasiado sencilla.
- **Timely:** A diferencia de la anterior opción, Timely es una app que se encarga de rastrear el tiempo automáticamente, generando estadísticas en tiempo real con datos tan interesantes como las aplicaciones utilizadas en el dispositivo en cada espacio temporal. Es una gran solución que tiene como gran desventaja el no disponer de una versión gratuita y el tener como público objetivo el sector empresarial, donde brilla en la gestión de equipos de trabajo.

Una vez expuestas algunas de las alternativas más interesantes para el tipo de aplicación que compete a este proyecto, y teniendo en cuenta el sinfín de opciones que existen, es difícil afrontar un nuevo desarrollo sin caer en los aciertos y errores de quienes ya han realizado algo parecido.

miTiempo es una aplicación que engloba un poco de cada una de las opciones vistas. No es su competencia convertirse en un gestor de tareas excelso, como en el caso de Things 3. Tampoco aspira a ser una aplicación con un sinfín de opciones e integraciones, como

PomoDoneApp y Focus To-Do. Por otro lado, abarca la sencillez de alternativas como Forest y Toggle, y le hubiera gustado llegar al nivel de automatización de Timely. Sí es competencia de miTiempo el tratar de implementar una **solución sencilla** en la que el usuario disponga en una sola aplicación y a través de un interface amigable de la opción de gestionar tareas, ejecutarlas con o sin la técnica Pomodoro y un ligero pero efectivo time-tracker .

Aspectos teóricos

miTiempo es un proyecto desarrollado sobre el stack o pila tecnológica **MERN**, formado por **MongoDB**, **Express**, **React** y **Node.js**, siendo su interactividad y eficacia contrastada, y contando con gran documentación y recursos de ayuda al desarrollador.

En concreto, este conglomerado de tecnologías permite hacer uso del lenguaje **JavaScript** en todas las etapas de desarrollo, tanto en lado cliente como en lado servidor, facilitando en gran medida el desarrollo y la comunicación entre las distintas partes del proyecto. A su vez, se hace uso de una base de datos no documental por la agilidad, velocidad y escalabilidad que provee este tipo de base de datos para el tipo de aplicación de que se trata.

Cabe destacar que miTiempo hace uso de un framework de React, **React Native**. Gracias a él, a partir de un solo código se desarrollan aplicaciones nativas para Android e iOS y una aplicación web. Las ventajas que esto implica, como veremos a lo largo de la presente documentación, son enormes, aunque también los problemas que pueden surgir.

Aunque las tecnologías que incluye MERN son la columna vertebral del proyecto y en este apartado nos centramos tan sólo en él a modo de introducción, para el desarrollo de miTiempo se hace uso de numerosos frameworks, entornos y herramientas de desarrollo, herramientas de diseño, servicios, api, paquetes, librerías y sistemas de control de versiones. Estas serán detalladas en profundidad en el apartado “Implementación del sistema”.

FASES

Para la realización de este proyecto se siguen 4 fases: **formación**, **análisis**, **diseño** e **implementación**.

A lo largo de los siguientes apartados se detallan cada una de ellas.

FORMACIÓN

La primera fase de la creación de miTiempo nace con el **estudio del stack MERN**: MongoDB, Express, React/React Native y Node.js. Al ser tecnologías no vistas en gran parte a lo largo del ciclo, requerían un estudio previo (Node.js, Express, React y React Native) o una ampliación de los conocimientos adquiridos (MongoDB). Esta fase no es baladí, pues los conocimientos adquiridos durante ella son los que permiten el desarrollo completo del proyecto.

Se parte de lo concreto a lo general, comenzando el estudio de cada tecnología por separado, realizando varios proyectos de ejemplo, para terminar con el estudio de la relación entre las mismas.

Este periodo de formación abarca del 9 de marzo al 8 de abril, comprendiendo la realización de las siguientes lecturas, cursos y prácticas:

- **MongoDB:**
 - o MongoDB for Javascript Developers M220JS, MongoDB University.
- **Node.js + Express:**
 - o Beginning Node.js, Express & MongoDB Development, Greg Lim.
- **React:**
 - o The Road to React 2020 edition, Robin Wieruch.
 - o Modern React with Redux, Stephen Grider.
- **React Native:**
 - o The Complete React Native + Hooks Course, Stephen Grider.
- **MERN:**
 - o Node with React: Fullstack Web Development, Stephen Grider.

ANÁLISIS

En la fase de análisis se definen los **requisitos**, no funcionales y funcionales, con el objetivo de tener claras las **necesidades** de la aplicación de cara a su desarrollo.

Requisitos

En el presente apartado se detallan los distintos requisitos de la aplicación, es decir, se hace un desglose de los **servicios** que debe ofrecer y las **restricciones** asociadas a su funcionamiento.

Requisitos no funcionales

Son aquellos requisitos que son “**cualidades**”, definen cómo debe ser el sistema:

Número	Requisito	Descripción	Prioridad
RNF1	Multiplataforma	<p>El sistema debe correr sobre diversas plataformas, en concreto:</p> <ul style="list-style-type: none"> - Web. - Android. - iOS. 	5
RNF2	Seguridad	<ul style="list-style-type: none"> - Registro necesario para hacer uso del sistema. - El registro y el ingreso se implementará a partir de usuarios definidos y contraseñas. - Debe existir una capa opcional de autenticación para el alta con la api de Google. - Las contraseñas serán cifradas. - Los usuarios tendrán acceso sólo a su información y no a la del resto de usuarios. - El usuario no necesitará iniciar sesión cuando vuelva a abrir la aplicación, salvo que hubiera cerrado la sesión. 	5
RNF3	Usabilidad	<p>El interface debe:</p> <ul style="list-style-type: none"> - Ser amigable, ágil y fácil de usar para el usuario. - Los textos y los diversos controles deben ser distinguibles a 1 metro de distancia para la plataforma web y 0.5 metros para la plataforma móvil. - Las peticiones de datos serán claras y estructuradas. 	5
RNF4	Portabilidad	<p>El sistema:</p> <ul style="list-style-type: none"> - Debe hacer uso de una conexión a internet. - Debe implementarse haciendo uso de tecnologías Cloud que permitan el acceso a los datos desde cualquiera de las plataformas 	5
RNF5	Rendimiento	<ul style="list-style-type: none"> - Los tiempos de respuesta para peticiones al servidor no deben superar un segundo de duración. - Los tiempos de respuesta para consultas a la base de datos no deben superar los 2 segundos. - Los tiempos de carga entre pantallas serán inapreciables. 	3
RNF6	Soporte	<p>El sistema debe quedar preparado para implementar un sistema de tests unitarios.</p>	1
RNF7	Documentación	<ul style="list-style-type: none"> - Código claro y documentado. - Manual de usuario. 	5

Tabla 1 - Requisitos no funcionales

Requisitos funcionales

Se trata de aquellos requisitos que indican **cómo interacciona el sistema con su entorno**, cuáles van a ser su estado y su funcionamiento, definiendo qué debe hacer el sistema:

Número	Requisito	Descripción	Prioridad
RF1	Acceso a la app mediante landing page, url o app nativa.	<p>El acceso a la aplicación se realizará mediante estas vías:</p> <ul style="list-style-type: none">- Landing page que centraliza la experiencia y redirecciona al resto de opciones.- URL webapp.- App Android/iOS.	5
RF2	Alta de usuario mediante formulario o cuenta de Google.	<p>El alta en el sistema se realizará mediante dos formas:</p> <ul style="list-style-type: none">- Mediante formulario de alta en el que se solicitará al usuario un email y contraseña y se realizará una validación. Si la validación no es superada, se mostrará un error en pantalla.- Mediante capa de autenticación de Google, donde el usuario seleccionará su cuenta y le serán enviadas las credenciales de acceso.	5
RF3	Inicio de sesión del usuario mediante credenciales.	<p>El ingreso en la app será mediante email y contraseña y validación de los mismos.</p> <p>Si la validación no es superada, se mostrará un error en pantalla y no se permitirá el acceso. En caso contrario, el usuario podrá acceder.</p>	5
RF4	Inicio de sesión mediante cuenta de Google.	El ingreso en la app se podrá realizar mediante autenticación mediante cuenta de Google.	1
RF5	Visualizar la relación de tareas del usuario para el día actual	El sistema mostrará las tareas planificadas para el día de la semana en el que se encuentre el usuario así como el color seleccionado para cada una.	5
RF6	Visualizar la relación de todas las tareas del usuario.	El sistema mostrará todas las tareas del usuario así como el color seleccionado para cada una.	5
RF7	Visualizar las tareas por categoría.	El sistema permitirá filtrar las tareas por categorías preestablecidas.	5
RF8	Búsqueda de tareas.	El sistema permitirá realizar la búsqueda de tareas mediante una barra de búsqueda que filtre las tareas por su título.	5

RF9	Acceso al detalle de una tarea.	El sistema permitirá acceder al detalle de una tarea, mostrando la información correspondiente a la misma.	5
RF10	Edición de una tarea.	El sistema permitirá la edición de una tarea ya creada, mediante la edición de los campos de texto y las opciones de la misma en la pantalla de detalle, y la habilitación del botón superior “Ok” para guardar los cambios.	5
RF11	Eliminación de una tarea.	El sistema permitirá la eliminación de una tarea desde la pantalla de detalle de la misma, habilitándose el botón superior “Delete” desde donde realizar esta acción.	5
RF12	Marcar tarea como completada	<ul style="list-style-type: none"> - El sistema permitirá marcar una tarea como completada, clickando sobre el switch circular que se encuentra a la izquierda de cada tarea en la pantalla principal. - El sistema marcará la tarea automáticamente como completada cuando se ejecute su temporizador y este llegue a cero. 	5
RF13	Ejecutar una tarea.	<p>El sistema permitirá ejecutar una tarea:</p> <ul style="list-style-type: none"> - Pulsando el switch “play” que se encuentra a la izquierda de cada tarea en la pantalla principal. - Pulsando sobre el botón “play” que se encuentra en la pantalla de detalle de la tarea. <p>Esto mostrará un temporizador en la pantalla que mostrará diversos mensajes según el usuario haya seleccionado Pomodoro-Yes o Pomodoro-No.</p> <p>Al terminar el temporizador, la tarea se marcará como completada.</p>	5
RF14	Creación de una tarea.	<p>El sistema permitirá la creación de tareas con la siguiente información:</p> <ul style="list-style-type: none"> - Título (obligatorio). - Descripción (obligatorio). - Día (un día de la semana, de lunes a domingo). - Repetir (nunca, cada día o cada semana). - Categoría (Rutinas, Studio, Familia, Ocio, Lectura, Cocina, Deportes, Otros). - Duración (5, 15, 30, 45 minutos, 1, 2, 3, 4, 5, 6, 7, 8 horas). - Pomodoro (sí o no). 	5

		<ul style="list-style-type: none"> - Color. <p>Se habilitarán los botones superiores “Cancel” y “Add” para cancelar la creación o añadir la tarea.</p>	
RF15	Visualizar gráficas sobre las tareas del usuario.	<p>El sistema generará automáticamente gráficas que sirvan de feedback y tracker temporal al usuario:</p> <ul style="list-style-type: none"> - Tareas hechas vs pendientes. - Tareas por categoría. - Tareas por día de creación. - Minutos concentrado (minutos de tareas completadas vs minutos de tareas pendientes). - Duración de tareas con técnica pomodoro vs tareas sin técnica pomodoro. - Tareas Pomodoro vs no Pomodoro. 	5
RF16	Visualizar dispositivos y aplicaciones utilizados durante una determinada tarea	El sistema deberá dar feedback al usuario de las aplicaciones y dispositivos utilizados durante la ejecución de una tarea.	1
RF17	Mostrar los datos de la cuenta del usuario.	<p>El sistema permitirá acceder a los datos de la cuenta del usuario:</p> <ul style="list-style-type: none"> - Email. - Nombre de usuario. 	5
RF18	Edición de los datos de la cuenta del usuario.	<p>El sistema permitirá la edición de los datos de la cuenta del usuario “email” y “nombre”. También se permitirá el cambio de contraseña mediante el ingreso de una nueva y su confirmación. Se realizará una validación de la nueva contraseña mediante coincidencia con la contraseña de validación, mostrando un mensaje de aviso y deshabilitando el botón de “Update” en caso de que la validación sea incorrecta.</p> <p>El sistema habilitará un botón “Cancel” para anular la edición.</p>	5
RF19	Cierre de sesión	El sistema permitirá el cierre de la sesión del usuario desde la pantalla de cuenta de usuario, pulsando sobre el botón “Sign out”, debiendo volver a iniciar sesión para acceder a su información.	5

Tabla 2 - Requisitos funcionales

Especificación casos de uso

Actores del sistema

Entendiendo como actor del sistema a aquellas entidades que interactúan con el sistema, el **usuario** sería el único actor en lo que respecta a la aplicación que compete.

Diagrama de casos de uso

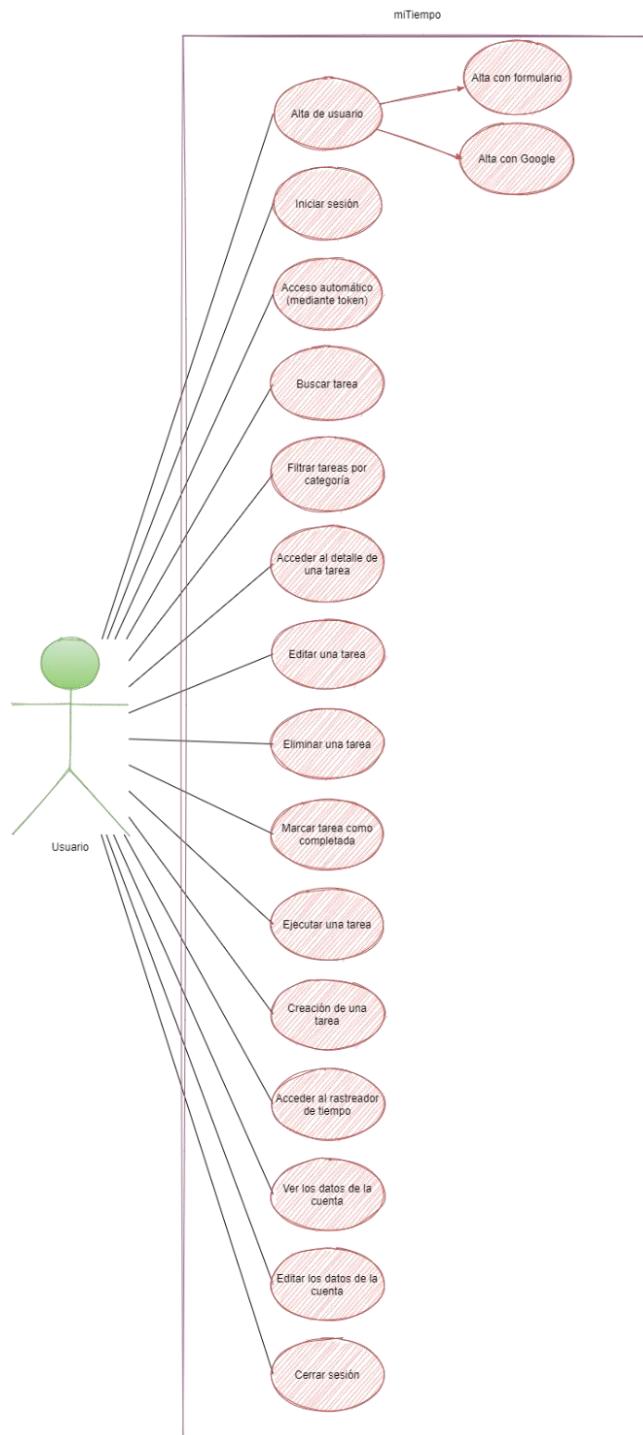


Ilustración 1 - Diagrama de casos de uso

Especificación de casos de uso

En las siguientes tablas se detalla cada uno de los casos de uso:

01A – Alta de usuario con formulario	
Descripción	El usuario desea registrarse en la aplicación mediante email y contraseña.
Precondición	No debe haber ninguna sesión abierta.
Secuencia principal	<ol style="list-style-type: none"> 1. El usuario rellena los campos “email” y “Password”. 2. El usuario pulsa el botón “Sign up”. 3. El sistema valida el email y la contraseña. 4. Se accede a la aplicación: se muestra la pantalla principal mostrándose las “Today tasks” y “My tasks” del usuario.
Errores / alternativas	<ol style="list-style-type: none"> 1. No se supera la validación del email y la contraseña. 2. Se muestra un mensaje de error. 3. El usuario tiene que repetir la secuencia principal.
Postcondición	Sesión iniciada.

Tabla 3 - Casos de uso 01A

01B – Alta de usuario con cuenta de Google	
Descripción	El usuario desea registrarse en la aplicación mediante su cuenta de Google.
Precondición	No debe haber ninguna sesión abierta.
Secuencia principal	<ol style="list-style-type: none"> 1. El usuario pulsa el botón “Sign up with G”. 2. Se redirige al autenticador de Google. 3. Se envía un email con las credenciales de acceso (email registrado y contraseña). 4. Se redirige a la página de confirmación de envío del email. 5. El usuario debe iniciar sesión (caso 02).
Errores / alternativas	<ol style="list-style-type: none"> 1. No se supera la autenticación de Google. 2. El usuario tiene que repetir la secuencia principal.
Postcondición	Usuario creado en el sistema.

Tabla 4 - Casos de uso 01B

02 – Iniciar sesión	
Descripción	El usuario desea iniciar sesión en la aplicación.
Precondición	No debe haber ninguna sesión abierta.
Secuencia principal	<ol style="list-style-type: none"> 1. El usuario pulsa sobre “Already have an account? Sign in instead!”. 2. El usuario rellena los campos “email” y “Password”. 3. El usuario pulsa el botón “Sign in”. 4. El sistema valida el email y la contraseña.

	5. Se accede a la aplicación: se almacena token en el dispositivo, se muestra la pantalla principal mostrándose las “Today tasks” y “My tasks” del usuario.
Errores / alternativas	<ol style="list-style-type: none"> 1. No se supera la validación del email y la contraseña. 2. Se muestra un mensaje de error. 3. El usuario tiene que repetir la secuencia principal.
Postcondición	Sesión iniciada.

Tabla 5 - Casos de uso 02

03 – Acceso automático a la app	
Descripción	El usuario desea acceder a la app.
Precondición	El usuario ha tenido que iniciar sesión y el token de seguridad encontrarse en su dispositivo.
Secuencia principal	<ol style="list-style-type: none"> 1. El usuario abre la app (web, Android o iOS). 2. El sistema comprueba que hay un token válido. 3. Se accede a la pantalla principal.
Errores / alternativas	<ol style="list-style-type: none"> 1. No hay un token válido. 2. Se redirige a la pantalla de alta.
Postcondición	Se accede a la aplicación con la sesión correspondiente al token.

Tabla 6 – Casos de uso 03

04 – Buscar tarea	
Descripción	El usuario desea buscar una tarea mediante la barra de búsqueda en la pantalla principal.
Precondición	Sesión iniciada.
Secuencia principal	<ol style="list-style-type: none"> 1. El usuario pulsa sobre la barra de búsqueda de la pantalla principal. 2. El usuario teclea uno o más caracteres. 3. Se muestran las tareas que incluyen en su título el carácter o los caracteres tecleados.
Errores / alternativas	<ol style="list-style-type: none"> 1. Si no hay ninguna tarea con título coincidente, no se muestra ningún resultado. 2. Si el usuario elimina la búsqueda, se muestran todas las tareas de nuevo.
Postcondición	Se muestra la búsqueda realizada.

Tabla 7 - Casos de uso 04

05 – Filtrar tareas por categoría	
Descripción	El usuario desea filtrar las tareas por categoría en la pantalla principal.
Precondición	Sesión iniciada.

Secuencia principal	<ol style="list-style-type: none"> El usuario pulsa sobre la categoría deseada en la lista horizontal de categorías de la parte superior de la pantalla. El sistema filtra y muestra las tareas por la categoría seleccionada.
Errores / alternativas	<ol style="list-style-type: none"> Si no hay ninguna tarea para la categoría seleccionada, no se muestra ningún resultado. El usuario tiene que seleccionar otra categoría u “All”.
Postcondición	Se muestran las tareas con la categoría seleccionada.

Tabla 8 – Casos de uso 05

06 – Acceder al detalle de una tarea	
Descripción	El usuario desea acceder al detalle de una tarea.
Precondición	Sesión iniciada y alguna tarea creada por el usuario.
Secuencia principal	<ol style="list-style-type: none"> El usuario pulsa sobre el chevron situado en el lado derecho de la tarea en la pantalla principal. El sistema redirige al usuario a la pantalla de detalle de tarea donde se muestra la información de la misma.
Errores / alternativas	<ol style="list-style-type: none"> Ninguna
Postcondición	Se muestra la pantalla de detalle de tarea con la información de la tarea.

Tabla 9 - Casos de uso 06

07 – Editar una tarea	
Descripción	El usuario desea editar una tarea.
Precondición	Encontrarse en la pantalla de detalle de la tarea seleccionada.
Secuencia principal	<ol style="list-style-type: none"> El usuario modifica los campos “title”, “description” o alguna de las opciones en el panel de selección central. El usuario pulsa en el botón superior derecho “Ok”. Se modifica la tarea. Se redirige a la pantalla principal.
Errores / alternativas	<ol style="list-style-type: none"> El usuario pulsa en el botón “home” de la barra de navegación para volver a la pantalla principal.
Postcondición	Tarea seleccionada editada.

Tabla 10 – Casos de uso 07

08 – Eliminar una tarea	
Descripción	El usuario desea eliminar una tarea.
Precondición	Sesión iniciada y encontrarse en la pantalla de detalle de la tarea seleccionada.
Secuencia principal	<ol style="list-style-type: none"> 4. El usuario pulsa sobre el botón superior izquierdo “Delete”. 5. Se elimina la tarea. 6. Se redirige a la pantalla principal.
Errores / alternativas	Ninguna.
Postcondición	Tarea seleccionada eliminada.

Tabla 11 - Casos de uso 08

09 – Marcar una tarea como completada	
Descripción	El usuario desea eliminar una tarea.
Precondición	Sesión iniciada y encontrarse en la pantalla principal.
Secuencia principal	<ol style="list-style-type: none"> 1. El usuario pulsa sobre el switch circular situado en la parte izquierda de la tarea. 2. La tarea se cambia a la categoría “Done”. 3. La tarea se muestra traslúcida y el switch se rellena con un punto. 4. En la próxima renderización la tarea aparecerá sólo bajo la categoría “Done”.
Errores / alternativas	Ninguna.
Postcondición	Tarea seleccionada marcada como completada.

Tabla 12 - Casos de uso 09

10 – Ejecutar una tarea	
Descripción	El usuario desea ejecutar una tarea.
Precondición	Sesión iniciada y encontrarse en la pantalla principal o en la pantalla de detalle de la tarea.
Secuencia principal	<ol style="list-style-type: none"> 1. El usuario pulsa sobre el switch con forma de “play” situado a la izquierda de la tarea en la pantalla principal, o en el botón “play” situado en la zona inferior de la pantalla de detalle de tarea. 2. Se muestra un temporizador con la duración y la configuración preestablecida por la tarea (Pomodoro y duración). 3. Cuando el temporizador termina se muestra un mensaje y se marca la tarea como completada. 4. Se cambia el botón “play” por el botón “Go back” para dar la opción al usuario de volver a la pantalla principal.

Errores / alternativas	<ol style="list-style-type: none"> 1. El usuario pulsa el botón “Cancel” dentro del temporizador. 2. Se anula la ejecución de la tarea. 3. Se vuelve a la pantalla principal.
Postcondición	Se marca como completada la tarea.

Tabla 13 - Casos de uso 10

11 – Creación de una tarea	
Descripción	El usuario desea crear una tarea.
Precondición	Sesión iniciada.
Secuencia principal	<ol style="list-style-type: none"> 1. El usuario pulsa sobre el botón “+” de la barra de navegación. 2. Se abre la pantalla de creación de tarea. 3. El usuario rellena los campos title y description. 4. El usuario selecciona las opciones deseadas en el selector de opciones. 5. El usuario pulsa “Add” para añadir la tarea.
Errores / alternativas	<ol style="list-style-type: none"> a. El usuario pusa “Cancel”: se cancela la creación de tarea y se redirige a la pantalla principal. b. El usuario no inserta los campos obligatorios (title y description): no se permite guardar la tarea.
Postcondición	Se añade una nueva tarea.

Tabla 14 - Casos de uso 11

12 – Acceder al rastreador de tiempo	
Descripción	El usuario desea acceder al rastreador de tiempo para recibir feedback visual.
Precondición	Sesión iniciada.
Secuencia principal	<ol style="list-style-type: none"> 1. El usuario pulsa sobre el botón “reloj” en la barra de navegación. 2. Se redirige al usuario a la pantalla de rastreador de tiempo. 3. Se muestra por defecto la tabla “Tasks by category”. 4. El usuario selecciona una opción en el selector de opciones. 5. Se muestra la gráfica seleccionada.
Errores / alternativas	Ninguna.
Postcondición	Ninguna.

Tabla 15 - Casos de uso 12

13 – Ver los datos de la cuenta	
Descripción	El usuario desea acceder a los datos de su cuenta.
Precondición	Sesión iniciada.

Secuencia principal	<ol style="list-style-type: none"> 1. El usuario pulsa sobre el botón “tuerca” en la barra de navegación. 2. Se redirige al cliente a la pantalla de la cuenta del usuario. 3. Se muestra el email y el nombre de usuario
Errores / alternativas	Ninguna.
Postcondición	Ninguna.

Tabla 16 - Casos de uso 13

14 – Editar los datos de la cuenta	
Descripción	El usuario desea editar los datos de su cuenta.
Precondición	Sesión iniciada y encontrarse en la pantalla de la cuenta del usuario.
Secuencia principal	<ol style="list-style-type: none"> 1. El usuario pulsa sobre el botón “Edit”. 2. Se redirige al usuario a la pantalla de edición. 3. El usuario puede modificar los campos “email” y “name”, así como establecer una nueva contraseña. 4. Se realiza la validación de los campos. 5. Al pulsar en el botón update se almacenan los cambios y se redirige a la pantalla de la cuenta del usuario.
Errores / alternativas	<ul style="list-style-type: none"> - Si la nueva contraseña no coincide con su confirmación, se inhabilita el botón “Update” y se muestra un error en pantalla. - Si el usuario pulsa el botón “Cancel” se cancelan los cambios.
Postcondición	Datos de la cuenta editados.

Tabla 17 - Casos de uso 14

15 – Cerrar sesión	
Descripción	El usuario desea cerrar la sesión.
Precondición	Sesión iniciada y encontrarse en la pantalla de la cuenta del usuario.
Secuencia principal	<ol style="list-style-type: none"> 1. El usuario pulsa sobre el botón “Sign out”. 2. Se elimina el token del dispositivo del usuario, cerrando la sesión. 3. Se redirige al usuario a la pantalla de alta
Errores / alternativas	Ninguno.
Postcondición	Sesión cerrada.

Tabla 18 - Casos de uso 15

DISEÑO DEL SISTEMA

Tras haber completado las fases de formación y análisis, se realiza el **diseño** del sistema: la arquitectura del sistema, el diseño de la base de datos y el diseño de la interfaz.

Cabe decir que el diseño original de la aplicación a nivel de componentes y clases sufrió ligeras modificaciones por necesidades y variaciones durante el desarrollo, mostrándose a continuación el resultado final.

Arquitectura del sistema

A continuación se muestra una visión global de la **arquitectura** de la aplicación:

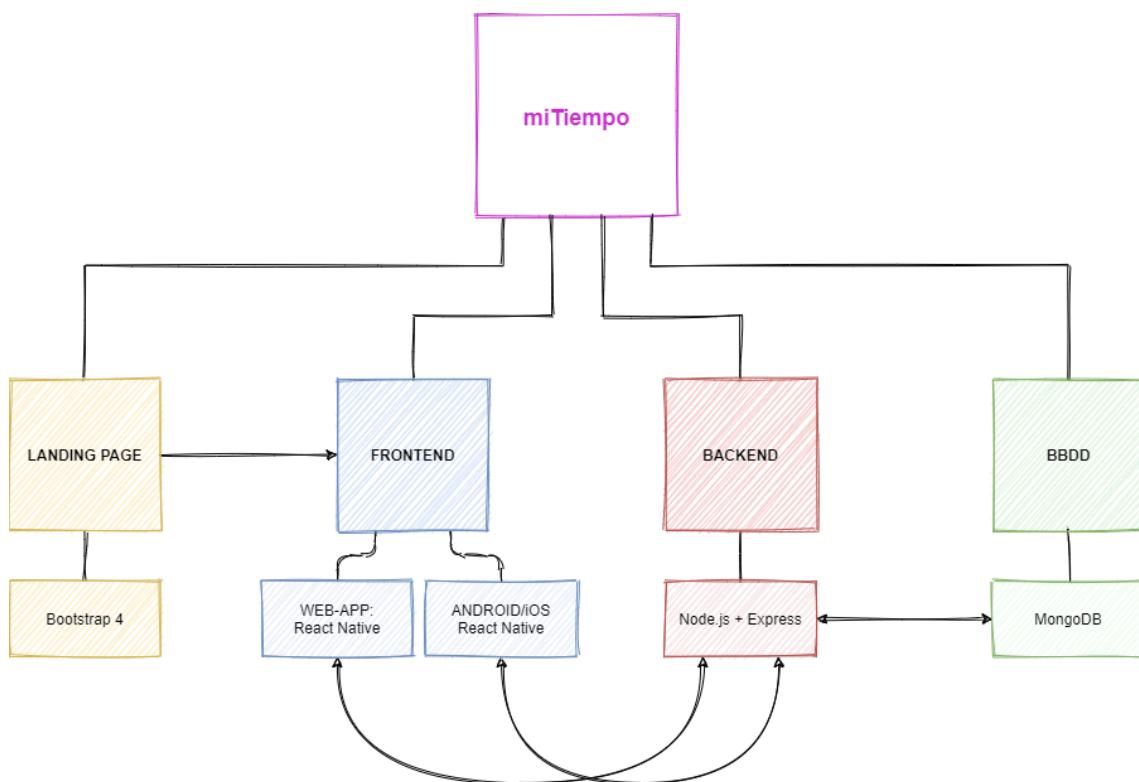


Ilustración 2 - Arquitectura del sistema

Como se puede ver, este proyecto consta de **4 grandes bloques intercomunicados** entre sí, con una arquitectura basada en el modelo cliente-servidor:

- **Landing page:** Consiste en una página web desarrollada sobre Bootstrap 4 y alojada en Netlify que sirve de gancho para un posible nuevo usuario, así como forma de acceso para redirigir al cliente a la url de la app web o a las tiendas de aplicaciones de las app móviles.
- **Frontend:** Desarrollado con Expo, framework que trabaja sobre React Native, permite a partir de un único código la construcción de una web app y aplicaciones

nativas para las plataformas móviles (en el presente caso, Android y iOS). El frontend está comunicándose constantemente con el backend, delegando la lógica no correspondiente al interface a este.

- **Backend:** Desarrollado con Express sobre Node.js y alojado en Heroku, es el encargado de recibir las peticiones desde el frontend, procesarlas, hacer las consultas pertinentes a la base de datos y enviar una respuesta al frontend.
- **Base de datos:** Los datos de la aplicación se almacenan en un *cluster* en la nube de MongoDB, la cual recibe las consultas pertinentes desde el back end. Por otro lado, también es la encargada de suministrar al frontend directamente las gráficas que se muestran en la pantalla del timer-tracker.

A continuación se muestra la estructura de cada uno de estos bloques:

Landing page

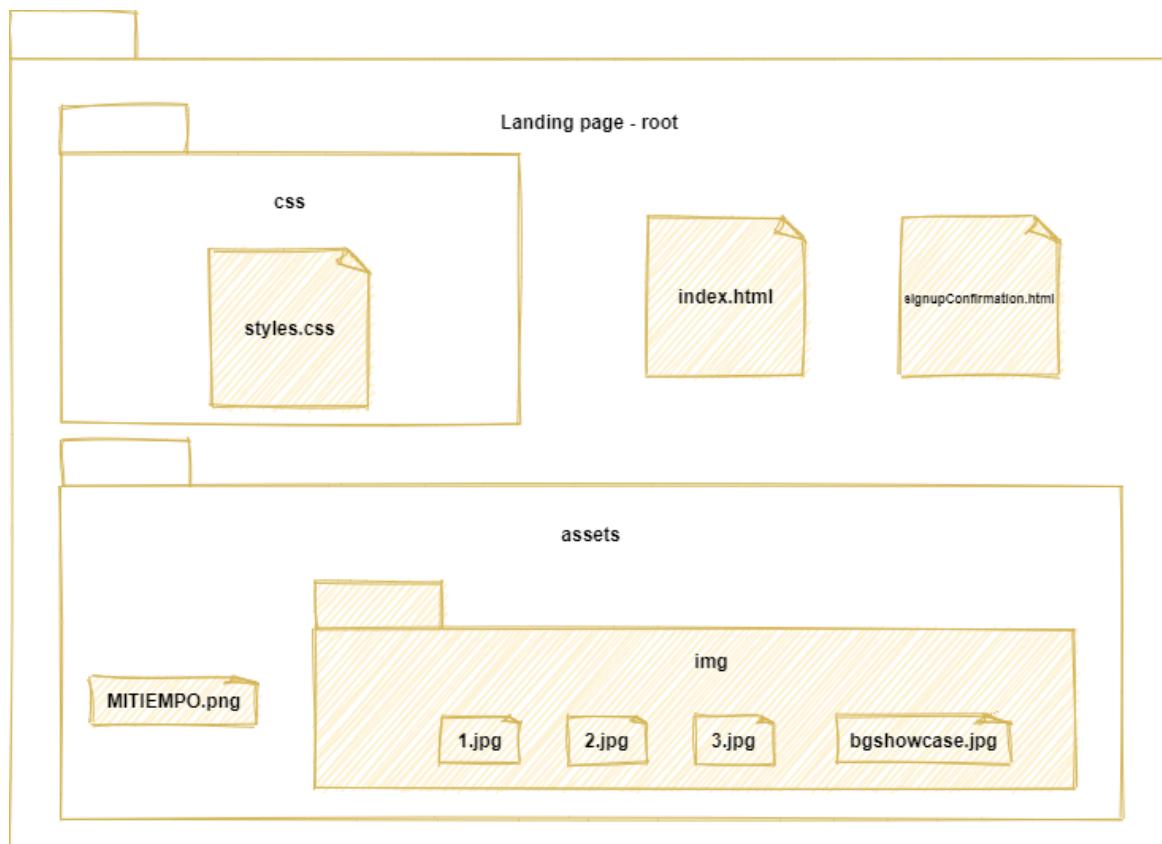


Ilustración 3 - Estructura Landing Page

Frontend

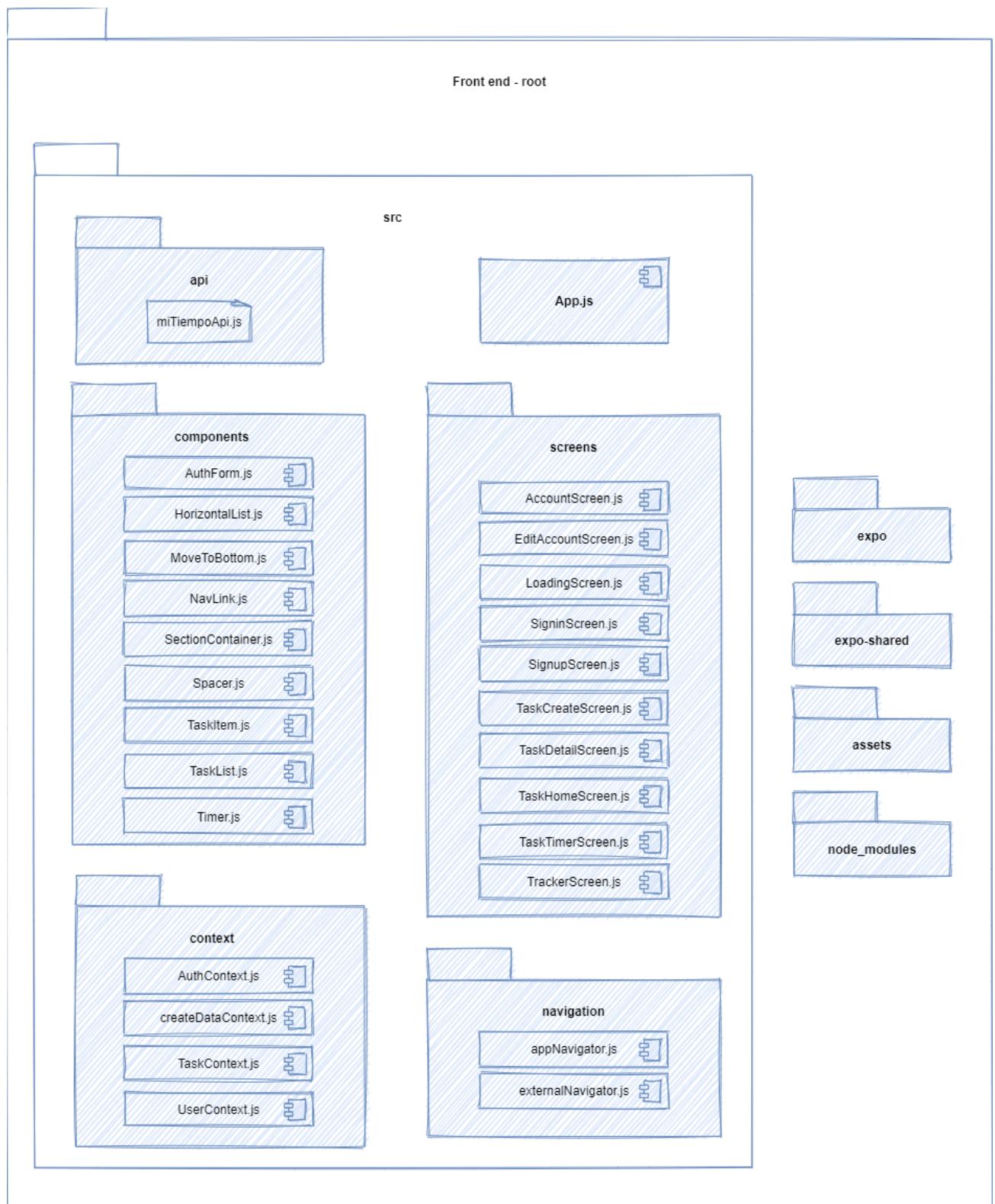


Ilustración 4 - Estructura Frontend

Backend

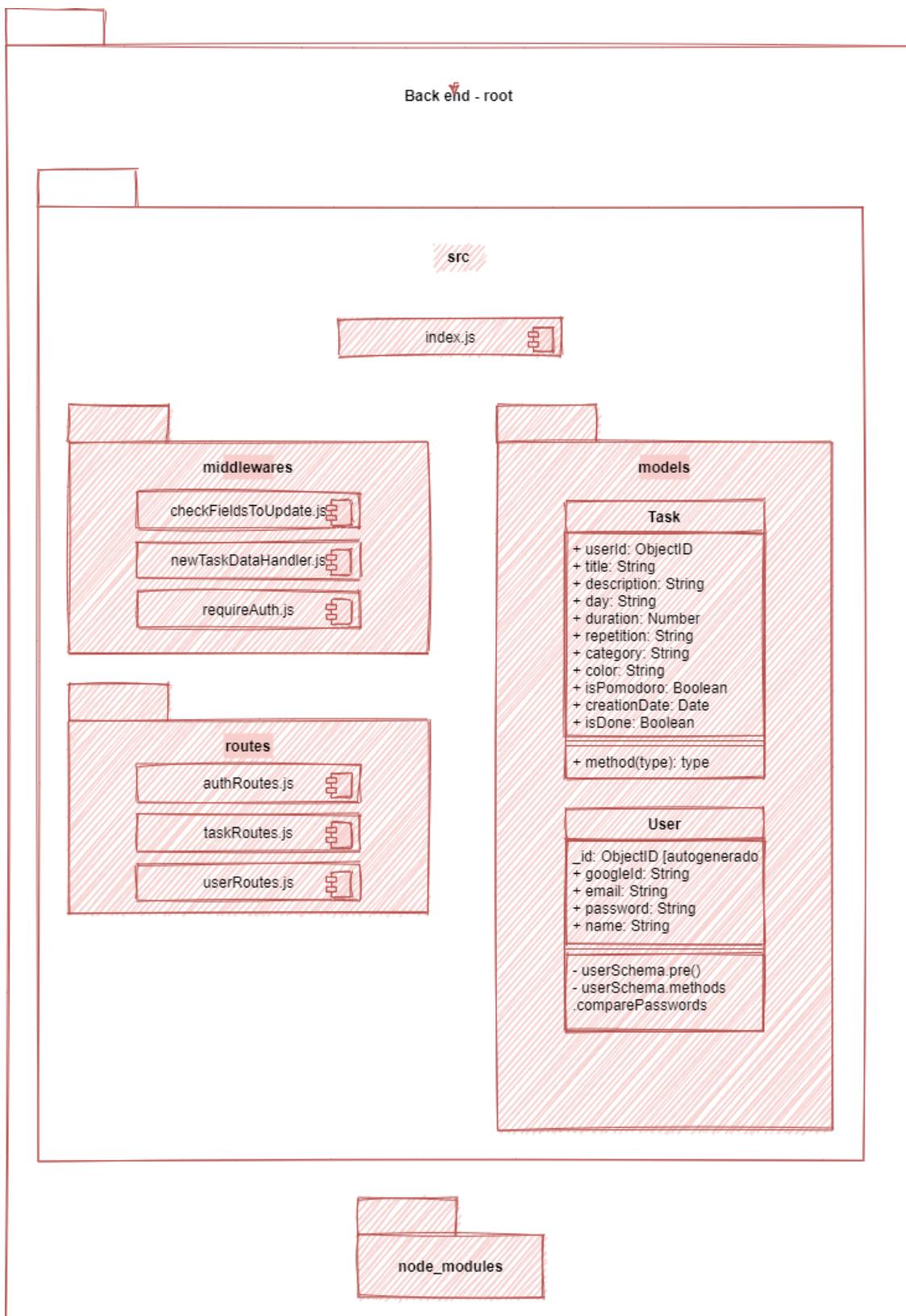


Ilustración 5 - Estructura Backend

Base de datos

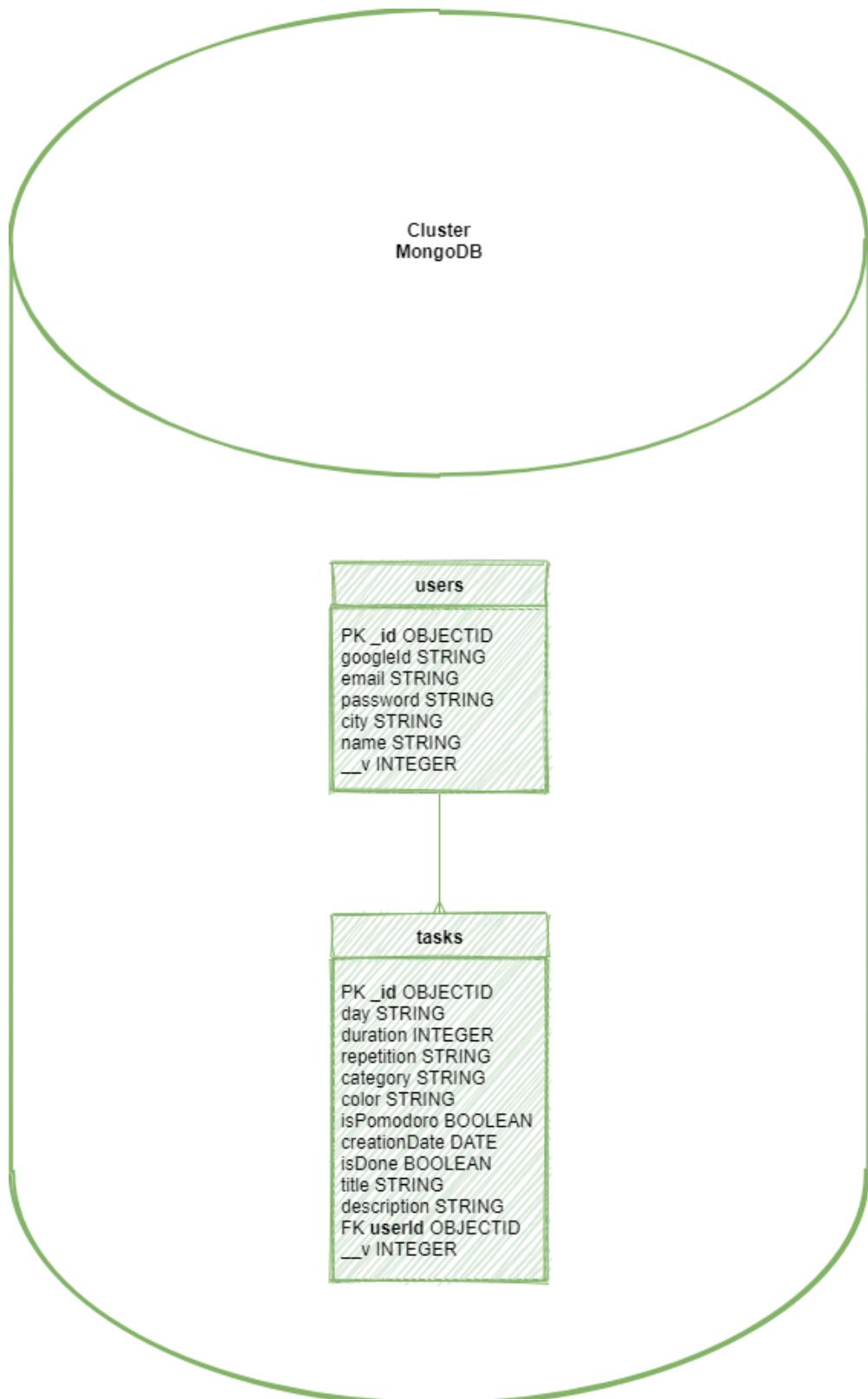


Ilustración 6 - Estructura Base de datos

Diseño de la base de datos

La elección de la base de datos para la aplicación es sencilla, puesto que el stack a usar, MERN, ya propone el uso de **MongoDB**. Aun así, se podría haber optado por una base de datos relacional, pero hay varios **factores** que hacen ideal esta base de datos documental o noSQL para miTiempo:

- A diferencia de una base de datos relacional, MongoDB se basa en **colecciones**, las cuales son estructuras de datos que almacenan documentos, normalmente con campos compartidos. Estos **documentos** son elementos dentro de una colección que almacenan la información en un formato similar a JSON, BSON, que es simplemente un “superset” de JSON, la representación binaria del mismo, permitiendo más tipos de datos dentro de MongoDB. Como decíamos, normalmente los campos son compartidos por todos los documentos de la colección, pero esto puede no ser así, ofreciendo gran flexibilidad.
- Precisamente, una de las grandes virtudes de MongoDB es que se basa en **esquemas flexibles**, siendo muy ágil de cara al desarrollo. Esto quiere decir que no todos los documentos tienen por qué tener los mismos campos, o que, si nuestra aplicación evoluciona, se pueden añadir y quitar campos o añadir nuevas colecciones sin entrar en conflictos con los datos ya almacenados.
- Al basarse en **JSON** (BSON), MongoDB implementa una extraordinaria interoperabilidad con aplicaciones basadas en JavaScript. Por tanto, la **comunicación** con el resto de tecnologías de la app está asegurada, ya que tanto desde el frontend como desde el backend se pueden manejar este tipo de datos con gran soltura.
- A su vez, MongoDB dispone de la opción de desplegar la base de datos en un cluster en vez de en un servidor local. Un **cluster** son un conjunto de servidores sobre los que corre la base de datos. Las ventajas son claras: los datos siempre están disponibles en la nube, descentralizados, seguros, y disponibles para ser consumidos en cualquier momento.
- Por último, esta base de datos dispone de una prestación esencial para miTiempo: **MongoDB Charts**: permite extraer los datos y generar gráficas a gusto del desarrollador, las cuales se pueden implementar en una aplicación de diversas formas. La ventaja que esto supone de cara a este proyecto, es clara, evitando horas de programación generando gráficas propias o mediante librerías de terceros.

Para el diseño de la base de datos de miTiempo se hace uso de **Mongoose**. Esta librería para Node.js está basada en Object Data Modeling (ODM). Permite especificar modelos que contienen esquemas, patrones de propiedades que ese modelo debe seguir. A partir de

estos modelos se crean **objetos** en la aplicación. Estos objetos son mapeados por Mongoose, almacenando su información en documentos, dentro de colecciones que se generan automáticamente en MongoDB, o recuperando información de esta mediante consultas para ser almacenada en nuevos objetos.

miTiempo necesita **dos modelos**, **User** y **Task**. A partir de cada uno de estos modelos, y siempre que desde el código del backend se especifique, se crean nuevos objetos, existiendo dos opciones:

- El objeto creado, User o Task, con información suministrada por el frontend o por el backend, **es almacenado** via Mongoose en un nuevo documento dentro de la colección Users o Tasks en el cluster de MongoDB.
- El objeto creado, sirve para **recibir la información** de un documento almacenado en una colección Users o Tasks, a partir de una consulta que realiza Mongoose desde el backend. Posteriormente, ese objeto puede ser usado dentro de la aplicación.

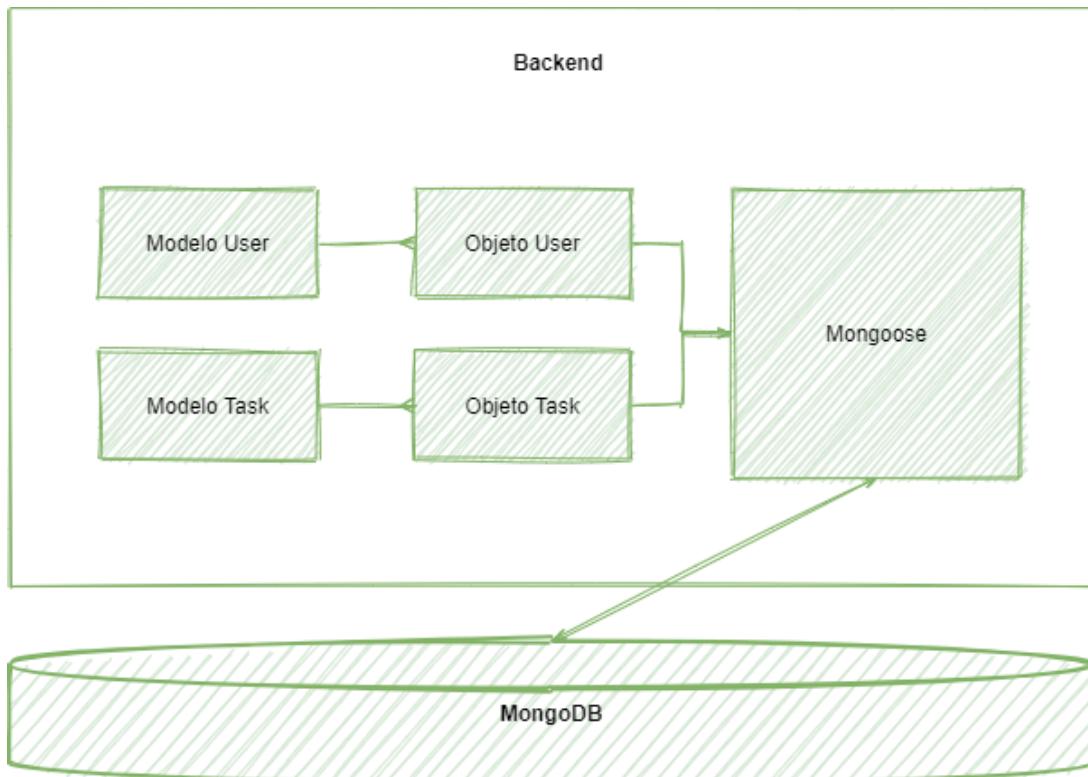


Ilustración 7 - Diseño de la base de datos

De esta manera, la base de datos de miTiempo queda definida mediante **dos colecciones**, **users** y **tasks**, encargadas de almacenar documentos basados en los modelos creados en el backend a partir de Mongoose:

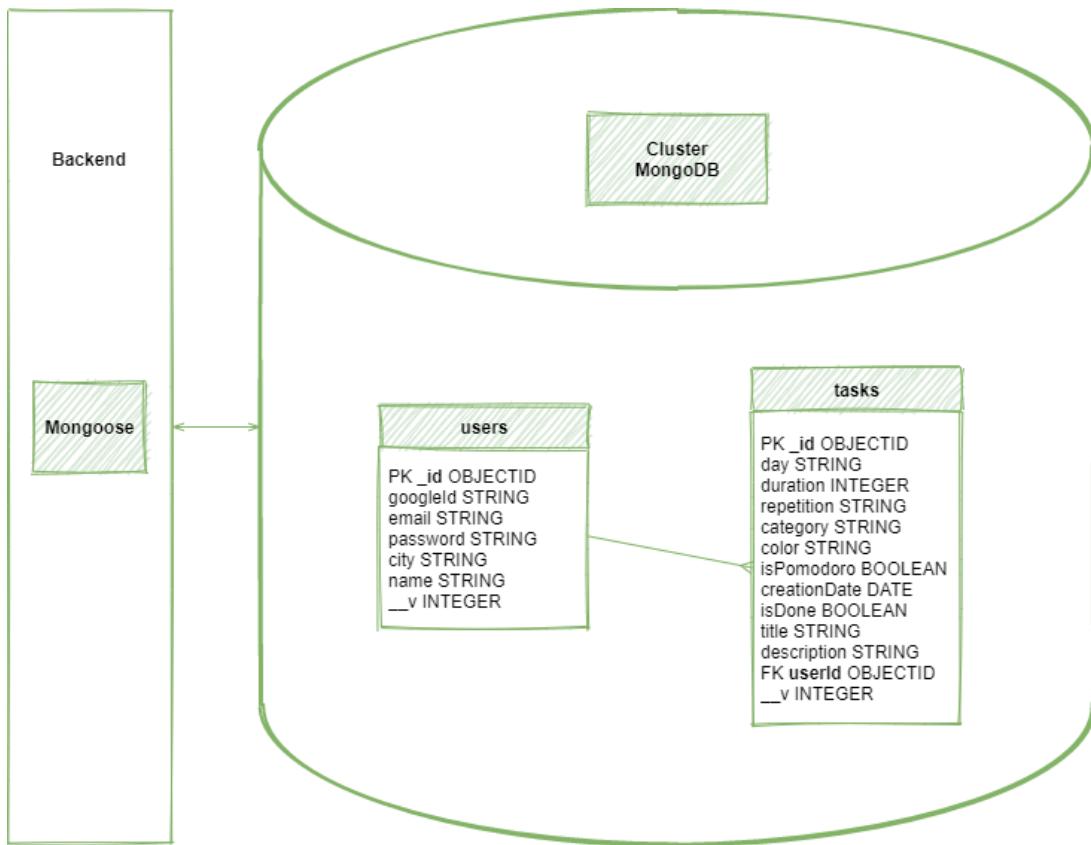


Ilustración 8 - Comunicación Backend y BBDD

Diseño de la interfaz

El interfaz de miTiempo es una parte muy importante de la aplicación. No sólo abarca conceptos de diseño, teniendo que ser atractivo y disponer de una buena funcionalidad (User Interface o **UI**), sino que debe proporcionar una buena experiencia de usuario (User Experience o **UX**) tratando de ser lo más amigable, intuitivo, fluido y eficaz posible.

Además, el interfaz de este proyecto está muy ligado al frontend. **React**, pese a su capacidad de extenderse hasta donde el desarrollador quiera, es en esencia una biblioteca para construir **interfaces de usuario “reactivas”**, mediante las cuales se consiguen páginas y aplicaciones dinámicas y muy atractivas. La planificación del diseño se produce en una etapa anterior al comienzo de la escritura de código, dejando constancia de la **importancia** que el diseño tiene para esa capa de la app.

Prototipado

Como se verá en el apartado de implementación del sistema, una de las primeras fases de este proyecto es el **prototipado del diseño del interfaz** de usuario mediante la herramienta **Figma**.

Se trata de diseñar un interfaz teniendo estos **objetivos** en cuenta:

- El interfaz debe ser atractivo, eficaz y amigable, así como proporcionar una buena experiencia de usuario (**UI y UX**).
- El prototipado realizado **debe servir de guía** de cara al desarrollo. Es decir, debe servir de mapa a seguir, evitando lo máximo posible tener que “diseñar sobre la marcha”, siendo de gran ayuda para concentrarse en la funcionalidad y en la maquetación durante la programación del sistema.

A continuación se muestra el **prototipado** realizado:



Ilustración 9 - Prototipo 1

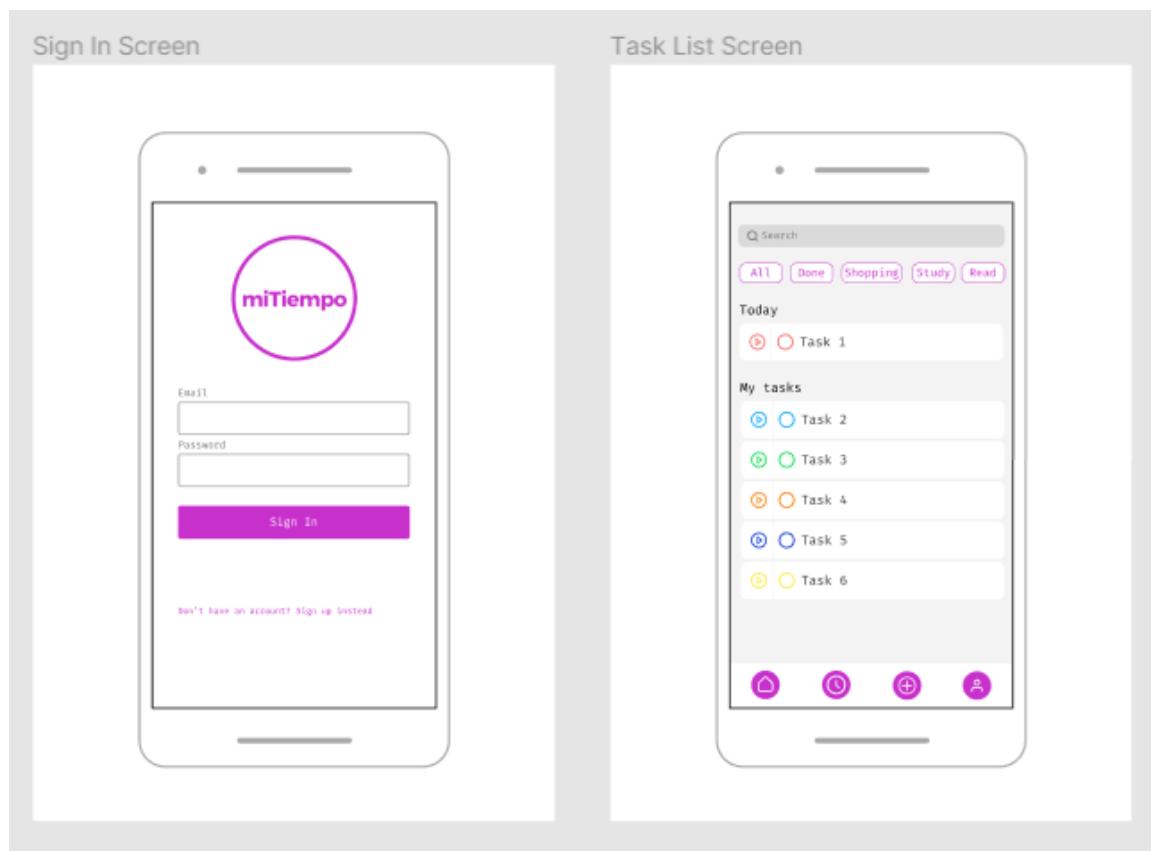


Ilustración 10 - Prototipo 2

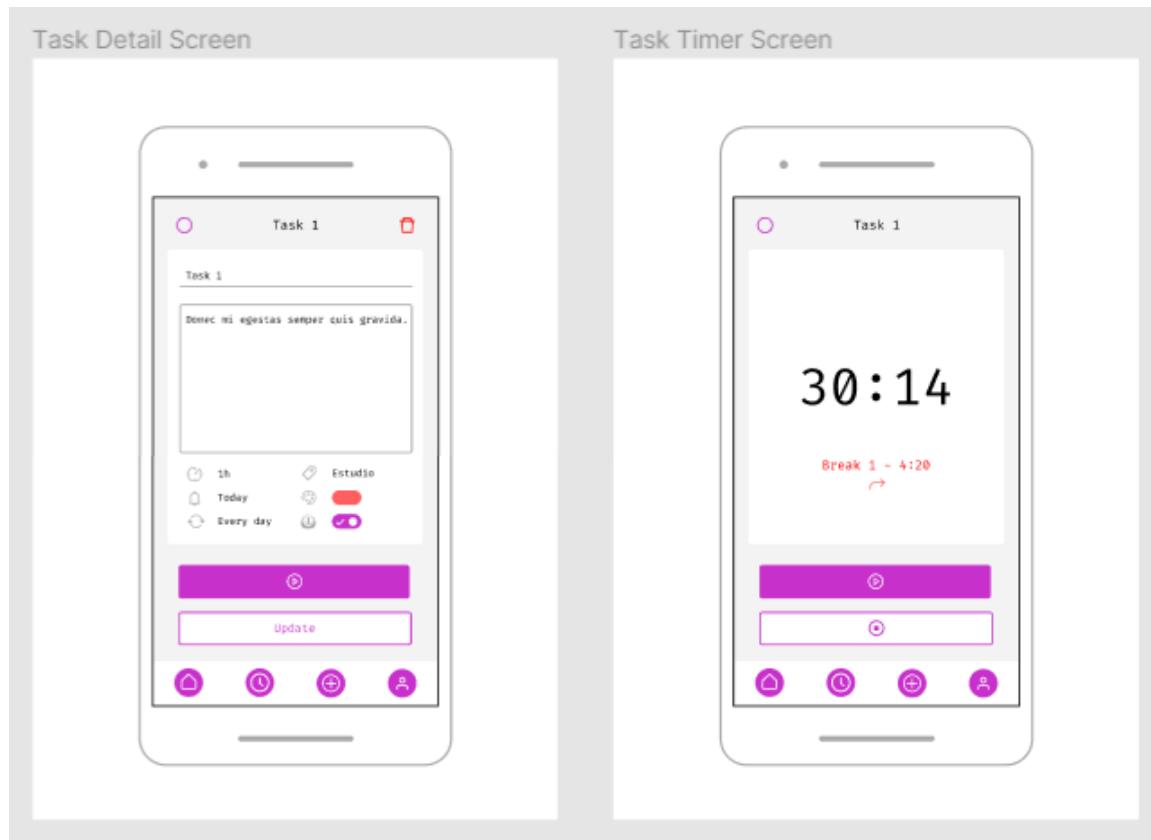


Ilustración 11 - Prototipo 3

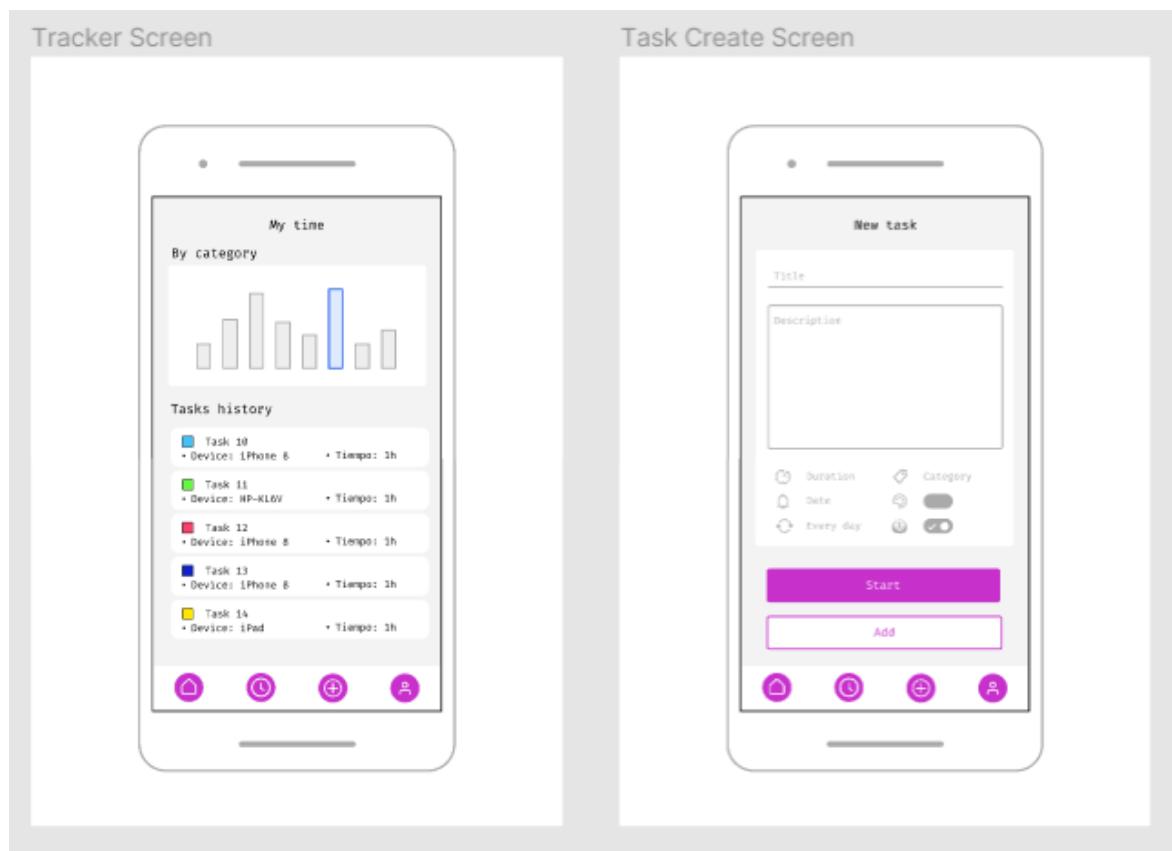


Ilustración 12 - Prototipo 4

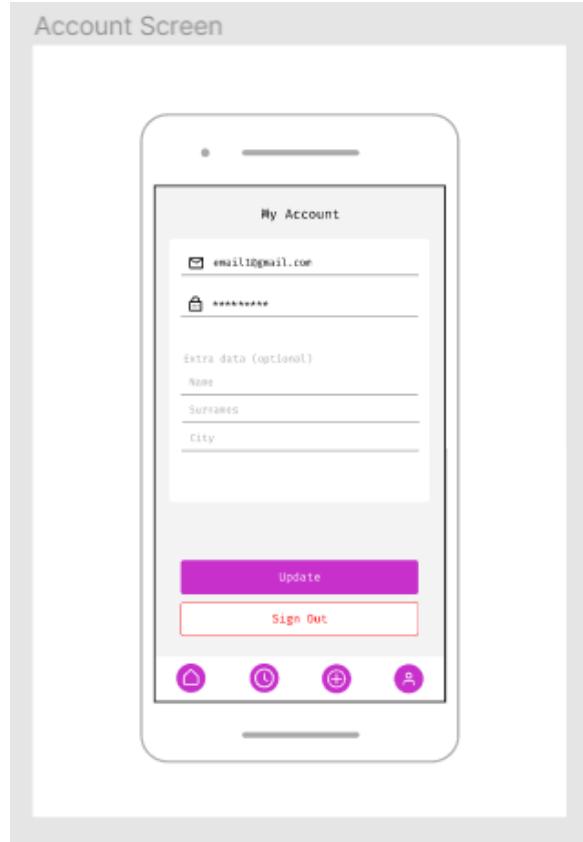


Ilustración 13 - Prototipo 5

Flujo de pantallas

Parte fundamental del diseño del interfaz es el flujo entre pantallas. Conseguir **cambios de pantalla intuitivos y naturales** es esencial para que el usuario disfrute de la aplicación.

A continuación se muestra el **flujo** entre las pantallas de miTiempo:

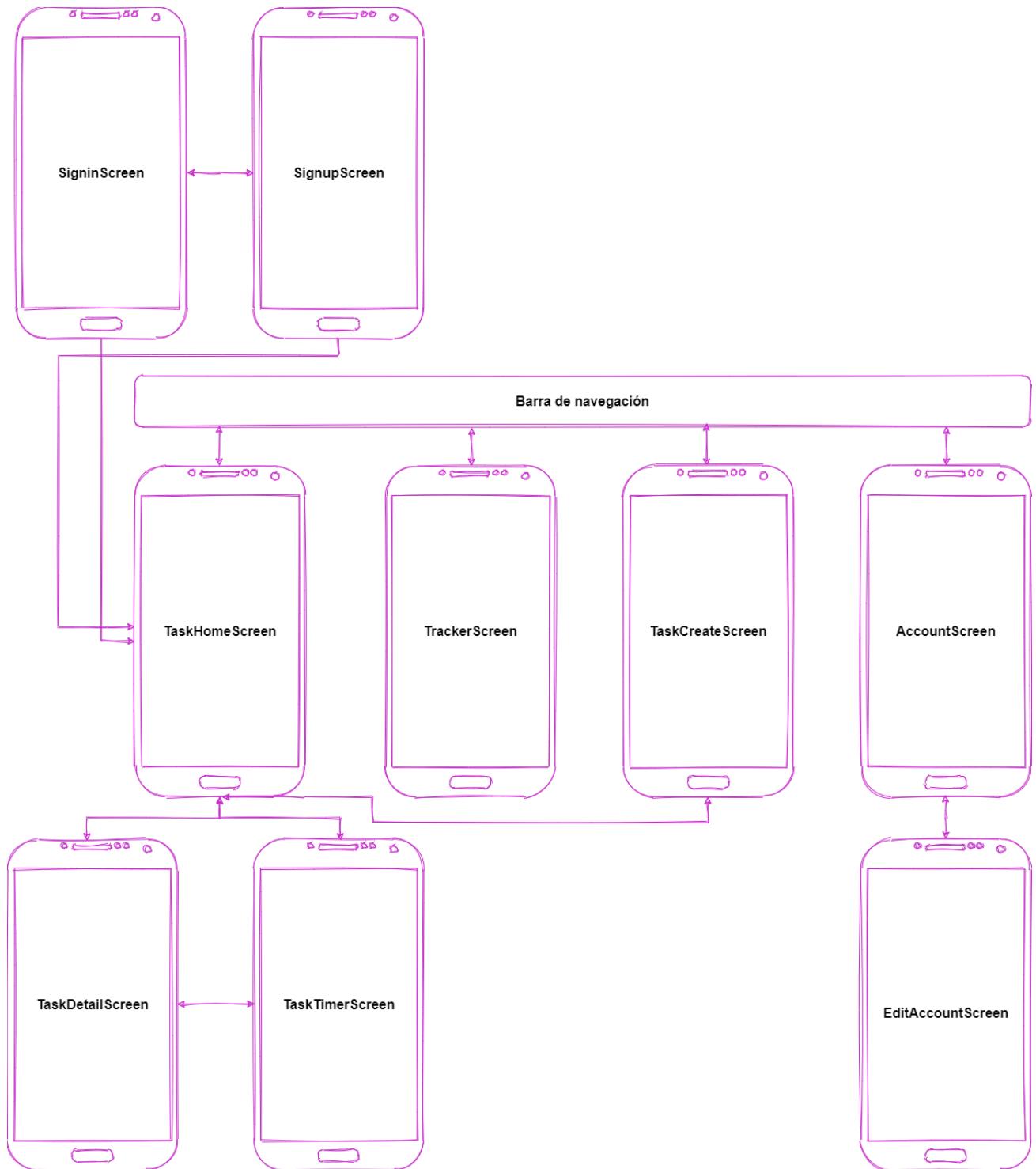


Ilustración 14 - Flujo de pantallas

IMPLEMENTACIÓN DEL SISTEMA

A lo largo de la fase de implementación se realiza el **desarrollo** de la aplicación, tomando de referencia todas las fases anteriores hasta ahora vistas, en ocasiones siendo modificadas por necesidades imperiosas.

Se comienza este apartado haciendo un recorrido a lo largo de los estándares, normas lenguajes, tecnologías y herramientas utilizados a lo largo del desarrollo, para pasar posteriormente a la creación del sistema.

Estándares y normas seguidos

A continuación se indican los estándares y normas seguidos en el desarrollo de miTiempo.

Estándares Javascript

Dado que todas las tecnologías del proyecto, como veremos en los siguientes apartados, se basan en **JavaScript**, miTiempo se basa en los estándares de este lenguaje emitidos por **ECMA Internacional bajo la especificación ECMA-262**, que contiene los estándares para un lenguaje de scripting de carácter general. Teniendo esto en cuenta, miTiempo se acoge a los siguientes estándares específicos para JavaScript:

ECMAScript 6

ECMAScript 6, ES6 o ECMAScript 2015 fue la segunda mayor revisión de JavaScript, publicada en 2015. Las principales **novedades** que incorpora al lenguaje y que se siguen en este proyecto son:

- **Let keyword:** Permite utilizar variables con alcance de bloque o block scope.
- **Const keyword:** Permite declarar constantes, una variable con un valor inmutable.
- **Arrow functions:** Permiten una sintaxis más corta que las funciones tradicionales, sin necesidad de las palabras reservadas “function” y “return”. No tienen “this” o, mejor dicho, su “this” siempre se refiere al objeto que las define.
- **Template literals:** Cadenas que, mediante *Interpolación*, permiten la incorporación de expresiones de JavaScript.
- **Call spread (...):** Permite a una función tener un número indefinido de parámetros
- **Classes:** Plantillas a partir de las cuales se crean objetos de JavaScript.
- **Destructuring:** Expresión de JavaScript que permite desempacar valores específicos de arrays o propiedades de objetos en distintas variables. De esta manera, podemos extraer directamente datos de arrays y objetos y asignarlos a variables.

- **Modules:** Piezas de código que se escriben en ficheros independientes y que se pueden reutilizar en otros archivos mediante una importación.
- **Promises:** Objeto de JavaScript que representa una actividad asíncrona, un código que consume y una promesa, un código que es recibido como consecuencia del primero.
- **Array.find():** Método que retorna el valor del primer elemento en un array que cumple la condición.
- **Array.findIndex():** Método que devuelve el índice del primer elemento en un array que cumple la condición.

ECMAScript 8

ECMAScript 8, ES8 o ECMAScript 2017 es una actualización menor del estándar de ECMA para Javascript. Las principales **novedades** que incorpora al lenguaje y que se siguen en este proyecto son:

- **Async/Await:** Define una función asíncrona, simplificando la forma de trabajar con las promesas y las actividades asíncronas. El operador Async asociado a una función, indica que la misma siempre devolverá una promesa. Por otro lado, Await hace referencia a la acción asíncrona que se va a realizar, como una función, por ejemplo, a la cual se espera para, suspendiendo la ejecución de la función, hasta que se establece la promesa y se reanuda la función. O bien se devuelve una excepción.

Buenas prácticas en React y Node.js

React y Node.js son librerías/entornos que delegan en el desarrollador la **libertad** de afrontar el proyecto según su propio criterio. Aun así, hay varias **buenas prácticas** a seguir que facilitan el progreso de la creación de la aplicación.

Convenciones de estilo

- Una carpeta/subcarpeta siempre con lowerCamelCase (la primera letra minúscula).
- Los componentes, clases y módulos deben ser escritos en CamelCase.
- Las funciones deben ser escritas en lowerCamelCase.
- Ausencia de espacio entre dos imports.
- El nombre de una clase/componente debe ser el mismo que el del archivo a la que pertenece.
- La declaración de objetos y variables se debe realizar siempre en lowerCamelCase.
- Para la declaración de variables const, se permite el uso del lowerCamelCase para indicar variables que no van a ser reasignadas. Se limitan las mayúsculas para constantes puras o constantes de clase/componente.

Estructura de carpetas

- Todas las carpetas deben estar incluidas dentro de la carpeta “src”.
- Todos los tipos de componentes, funciones, archivos deben situarse en una estructura de carpetas lógica y ordenada dentro de la carpeta “src”.

Clean Code

En la escritura del código de miTiempo se intenta implementar lo máximo posible una escritura limpia o, como diría Robert C. Martin, “**Clean Code**”, un código limpio. Por lo tanto, se trata de seguir estos puntos clave, dados en el libro del citado autor y seguidos desde hace años por la comunidad de desarrolladores, para tener un código lo más inteligible posible:

- **Nombres** de variables, funciones, clases y componentes significativos, con sentido.
- **Funciones** cortas y fáciles de leer, intentando que hagan sólo una acción por función y con el menor número de argumentos posible.
- Una clase/componente debe ser sólo responsable de ella misma (SRC o **Single Responsibility Principle**).
- Uso de **comentarios** que aporten información necesaria.
- **Formato** eficaz, en torno a los 80 y 100 caracteres de ancho, así como un orden del mismo verticalmente, permitiendo una mejor lectura del código,
- **Don't Repeat Yourself**: no repetir código, refactorizar para implementar código que se pueda reutilizar y evitar duplicaciones.

Lenguajes de programación

El lenguaje de programación usado en miTiempo es **JavaScript**, dado que es el que implementan las tecnologías del stack seleccionado. Es un lenguaje interpretado basado en prototipos, multiparadigma, de un solo hilo, con soporte para programación orientada a objetos, imperativa y declarativa. Su uso es amplio como lenguaje de scripting en páginas web, así como fuera del navegador en entornos como Node.js o Apache CouchDB.

Tecnologías y herramientas

Tecnologías

El proyecto se basa en el stack **MERN**, pila de tecnologías de desarrollo que tienen a JavaScript como nexo de unión:

- **MongoDB**: Base de datos documental (NoSQL), orientada a documentos, estructuras de datos BSON (representación binaria de JSON), los cuales se

almacenar en colecciones de forma dinámica, permitiendo llevar a cabo una integración de datos más fácil y rápida.

- **Express**: Framework para Node.js que facilita la creación de aplicaciones web y API en ese entorno.
- **React**: Biblioteca de JavaScript desarrollada por Facebook para construir interfaces de usuario. Declarativo, basado en componentes y multiplataforma gracias a frameworks como **React Native**, el cual permite desarrollar aplicaciones nativas para Android e iOS, además de web-apps con React.
- **Node.js**: Entorno de ejecución para JavaScript en entorno servidor, orientado a eventos asíncronos y con Entradas/Salidas, permitiendo gran escalabilidad y desarrollos completos con JavaScript, no teniendo que manejar otro lenguaje del lado del servidor.

Frameworks

- **React Native**: Framework creado por Facebook que permite desarrollar aplicaciones móviles para iOS, Android, Windows, macOS, entre todos. A diferencia de otros frameworks, permite realizar aplicaciones nativas con JavaScript para varias plataformas, siendo el framework el encargado de transformar el código escrito por el desarrollador en elementos renderizados con código nativo, siendo el resultado un aplicación nativa.
- **Expo**: Framework para desarrollo multiplataforma de aplicaciones React Native. Corriendo encima de este, proporciona un SDK que permite utilizar y administrar fácilmente los proyectos de React Native, componentes, herramientas de arranque, debug y despliegue.
- **Passport.js**: Framework para Node.js que ayuda en la gestión de autenticaciones.

Frameworks UI

- **React Native Elements**: Framework para React Native con componentes personalizados con apariencia alineada a lo largo de las diversas plataformas.
- **Bootstrap 4**: Framework HTML, CSS y JavaScript para realizar webs mobile-first responsivas de forma ágil.

Editores

- **Visual Studio Code**: Editor de texto multiplataforma desarrollado por Microsoft, ampliable y customizable con innumerables extensiones para añadir más características al mismo.

Herramientas

- **MongoDB Compass:** Herramienta para explorar, analizar e interactuar con el contenido almacenado en una base de datos de MongoDB, ya sea una base de datos almacenada en local o en un cluster.
- **AVD Manager (Android Studio):** Simulador de terminales Android.
- **Postman:** Utilidad que permite testear y analizar APIs.
- **Windows Terminal:** Aplicación que unifica todas las Shell de Windows en una nueva terminal con una UI redefinida que favorece la productividad del desarrollador. Permite añadir shells externas.
- **Git Bash:** Aplicación para entornos Windows que instala Bash, utilidades comunes a Bash y Git.

Herramientas de diseño y prototipado

- **Figma:** Herramienta para diseñar y probar prototipos de interfaces de usuario para gran variedad de proyectos, con el punto de mira puesto en el desarrollo de la UX y UI de los productos.
- **Draw.io:** Herramienta de creación y edición de diagramas.
- **Canva:** Aplicación de diseño, de gran utilidad para realizar logotipos o pequeñas composiciones.
- **Gimp:** Programa de edición y retoque de imágenes digitales.
- **Adobe Color:** Utilidad de Adobe que permite generar paletas y combinaciones de colores.
- **React Native Shadow Generator:** Utilidad que, mediante un interfaz sencillo, permite crear sombras para los componentes de React Native.

APIs y servicios

- **OAuth Consent Screen:** API de Google que permite utilizar la cuenta de Google del usuario para darse de alta o iniciar sesión.
- **Sendgrid:** Plataforma para la gestión de correos electrónicos transaccionales y comerciales. Sirve de intermediario entre un servidor y un proveedor de correo electrónico tradicional, como Gmail.

Paquetes y librerías

- **Axios:** Librería de JavaScript que hace las funciones de cliente HTTP, permitiendo la realización de peticiones a servicios y servidores de forma sencilla. Soporta Promesas y Async/Await.

- **BCrypt**: Librería que permite la encriptación de contraseñas implementando una función de hashing basada en el cifrado de Blowfish. Incorpora un fragmento aleatorio llamado “salt”, el cual evita que dos contraseñas iguales generen el mismo hash.
- **Jsonwebtoken**: Librería con la que crear Json Web Tokens (JWT), tokens de seguridad creados en el momento en que el usuario inicia sesión con sus credenciales, siendo devuelto al cliente y teniendo que ser enviado por este al servidor para acceder automáticamente a la aplicación.
- **Mongoose**: Esta herramienta permite realizar esquemas fuertemente tipados que son aplicados a modelos, los cuales sirven para generar objetos que son mapeados para interactuar con la base de datos. Mongoose transforma estos objetos en documentos en MongoDB.
- **Nodemon**: Librería que permite reiniciar automáticamente la aplicación de Node.js cuando se realizan cambios en el código.
- **Dotenv**: Utilidad que permite hacer uso de variables de entorno en una aplicación de Node.js, permitiendo almacenar las variables en el entorno, sin ser expuestas y separadas del código.
- **Cors**: Herramienta que provee formas de configurar las CORS (Cross-Origin Resource Shariing), mecanismo que utiliza cabeceras HTTP para permitir que un agente obtenga permiso para acceder a recursos seleccionados desde un servidor en un origen distinto al que pertenece. Sin esta librería, una app de Node.js tendría problemas para hacer peticiones a APIs de terceros.
- **Nodemailer**: Librería que permite enviar emails desde Node.js al email del usuario, a través del proveedor seleccionado.
- **Nodemailer-sendgrid-transport**: Plugin para Nodemailer que permite al mismo hacer uso de la api de SendGrid.
- **AsyncStorage**: Librería que implementa un sistema de almacenamiento local para React Native. Equivaldría a las SharedPreferences de Android o al keychain de iOS.
- **Moment.js**: Librería con la que poder manejar, operar y formatear fechas y horas.
- **Animate.css**: Librería de animaciones css con la que animar fácilmente elementos de páginas web.
- **AOS**: Librería de animaciones de scroll para páginas web.

Deploy

- **Heroku**: Plataforma como servicio (PaaS) que sirve para alojar aplicaciones de diversos tipos en la nube y con integraciones con otras plataformas, como GitHub.
- **Netlify**: Hosting web con gran integración con servicios como GitHub.

Control de versiones y organización del proyecto

- **Git:** Sistema de control de versiones que permite llevar un registro de los cambios en los archivos de un proyecto, siendo especialmente útil para coordinar el trabajo entre varias personas sobre archivos compartidos.
- **GitHub:** Plataforma de hospedaje de repositorios de código en la nube utilizando Git. Incluye **GitHub Projects**, proyectos en forma de tableros y vinculados a repositorios que sirven para organizar el trabajo sobre los mismos.

Creación del sistema

Preparación y planificación del desarrollo

Para este proyecto se hace uso integral de **GitHub**, implementando **Git** como sistema de control de versiones y **GitHub Projects** como sistema de ayuda a la planificación del proyecto, así como un **diario de desarrollo**, incluido en los anexos de la presente documentación.

Git & GitHub

La implementación de un sistema de control de versiones como **Git no es baladí**. Aunque este proyecto se desarrolla en solitario y quizás su uso no sea tan obvio como en un trabajo en equipo, hay varias razones para implementarlo:

- Permite un **registro** de los archivos del proyecto, pudiendo volver a cualquier etapa y a cualquier punto del desarrollo.
- Al usar Git con GitHub, hace funciones de **respaldo** del código, al alojarse en un repositorio remoto en el cloud de GitHub.
- Permite **estructurar y organizar** el proyecto por ramas, siendo lo más habitual tener una rama “dev”, en la que se trabaja durante el desarrollo, y una rama “prod”, donde se destina el código de producción, el que es desplegado y publicado finalmente.
- GitHub está **integrado** en gran cantidad de plataformas, por lo que facilita el trabajo con las mismas. Para este proyecto, la integración de GitHub con plataformas de despliegue como Heroku o Netlify han hecho más sencillas las tareas de despliegue.

Por lo tanto, se crean **3 repositorios** con la misma estructura: cada uno de ellos posee una rama **main** o rama de entorno de desarrollo, y una rama **prod** o rama de entorno de producción. El flujo de la información siempre parte de la rama main, que es sobre la que se trabaja, realizando un **merge** o traspaso de información de la rama de desarrollo a la de producción, cuando la funcionalidad está depurada y se quiere implementar en el entorno de destino.

Estos son los **3 repositorios** creados:

- **Landing page:** <https://github.com/pablohs1986/miTiempo-landingPage>
- **Frontend:** <https://github.com/pablohs1986/miTiempo-front>
- **Backend:** <https://github.com/pablohs1986/miTiempo-back>

A continuación se muestra el uso que se hace de **Git y el flujo de información** en cada uno de esos repositorios, partiendo de un entorno local hasta llegar al entorno de despliegue.

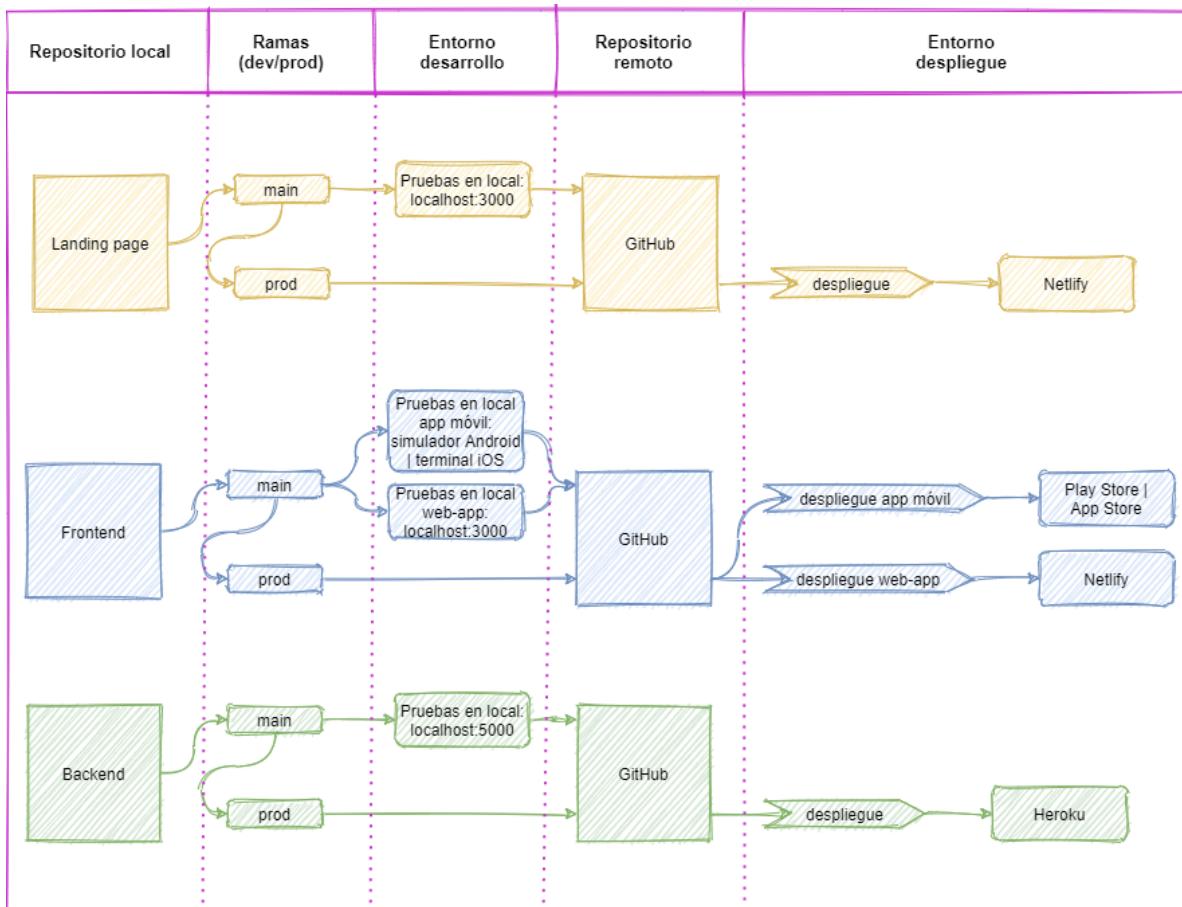


Ilustración 15 - Flujo Git

GitHub Projects

GitHub implementa un **sistema de planificación de proyectos** denominado GitHub Projects. Consiste en un **tablero de planificación** vinculado a los repositorios del proyecto. Dicho tablero posee 3 columnas por defecto que son las utilizadas para miTiempo, aunque se podrían añadir tantas como se quieran:

- **To-do:** Corresponden a tareas por realizar
- **In progress:** A esta columnas pertenecen las tareas en las que se está trabajando, las tareas en curso
- **Done:** Tareas ya realizadas, terminadas.

A su vez, también dispone de un **histórico** de actividad, muy útil a lo largo de toda la vida del proyecto, permitiendo saber siempre en qué estado está el mismo.

Las **ventajas** de utilizar GitHub Projects son claras, pues se centraliza la planificación del proyecto en un solo lugar, pudiendo crear al principio del desarrollo todas las tareas pendientes por hacer e ir asignando poco a poco el trabajo, así como poder añadir y reestructurar tareas sobre la marcha, según las necesidades.

Este es un detalle del proyecto de **miTiempo en GitHub Projects**, el cual se puede encontrar en la dirección <https://github.com/pablohs1986/projects/1> :

The screenshot shows the GitHub Projects interface for the 'miTiempo' project. It features three main columns: 'To do', 'In progress', and 'Done'. The 'To do' column contains two items: 'Fix on front web: Titles of tabs.' and 'Extras: Landing page to link to mobile / web app.', both added by 'pablohs1986'. The 'In progress' column contains three items: 'Front/back: Handle routines', 'Front/Back: Default order on Tasks.', and 'Front: create screens.', all added by 'pablohs1986'. The 'Done' column contains 45 items, many of which have checkmarks next to them, indicating completed tasks. A sidebar on the right shows a history of recent activity, including moves and updates made by 'pablohs1986' over the past few days. The URL at the bottom of the screenshot is <https://github.com/pablohs1986/projects/1#card-59421971>.

Ilustración 16 - GitHub Projects

Backend

Consiste en una aplicación desarrollada **Express sobre Node.js** y hospedada en **Heroku**, siendo la encargada de alojar los modelos de datos, recibir las peticiones desde el frontend, procesarlas, hacer las consultas pertinentes a la base de datos y enviar una respuesta al frontend.

Conceptos esenciales

Antes de explicar y detallar su funcionamiento, es esencial explicar tres **conceptos** esenciales:

- **Model:** Un modelo de Mongoose contiene esquemas gracias a los que se pueden crear objetos que son mapeados por esta librería para crear documentos en una base de datos MongoDB o recibir datos de la misma.

- **Route:** Es un método de Express que va ligado a una petición HTTP (GET, POST, PUT, DELETE), a una URL/Path y a una función que recibe dos objetos, Request y Response, que es llamada para manejar esa petición y devolver o no una respuesta. Entre la petición y la respuesta, pueden recibir Middlewares para realizar operaciones y validaciones intermedias.
- **Middleware:** Es una función que recibe objetos Request y Response y la función Next(). Se implementan sobre la app o sobre cada route para realizar operaciones que se pueden extender a varios routes (como por ejemplo, requerir una autenticación de usuario para procesar una petición). Se pueden ejecutar varias Middleware en cadena (pueden ser apiladas).

Funcionamiento

A partir de estos conceptos se procede a explicar el **funcionamiento** general del backend en **4 pasos**:

- **Petición desde frontend:** Recibe una petición desde el backend (GET, POST, PUT, DELETE).
- **Aplicación de Middleware:** Según el caso, se aplica una middleware para realizar algún tipo de acción antes de procesar la petición.
- **Procesamiento de la respuesta:** Se procesa la respuesta, haciendo uso o no de los modelos creados con Mongoose para recibir o almacenar información.
- **Envío de la respuesta:** Se envía la respuesta al frontend.

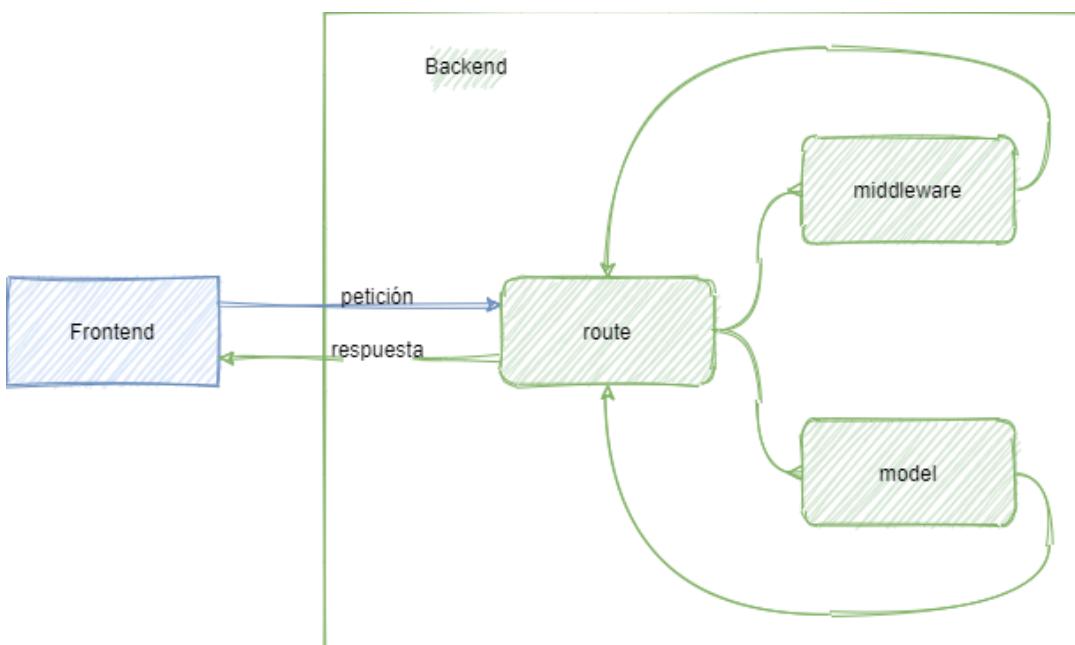


Ilustración 17 - Funcionamiento Backend

Detalle

A continuación se procede a explicar en detalle el backend.

index.js

Es el **punto de entrada** de la aplicación de Node.js. En ella se realiza la importación de todas las dependencias necesarias, se implementa la instancia de Express, el uso de Cors y se establece la conexión con el cluster de MongoDB.

Resaltar el uso de **dotenv**, librería que permite hacer uso de variables del sistema, para no exponer información sensible como claves de la base de datos, tokens, uris, claves de api, etc. Estas variables se almacenan, en local, en un archivo .env, mientras que en el entorno de producción se almacenan dentro del mismo, no exponiéndose nunca al exterior:

```
require('dotenv').config();
require('./models/User');
require('./models/Task');
const express = require('express');

const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const authRoutes = require('./routes/authRoutes');
const userRoutes = require('./routes/userRoutes');
const taskRoutes = require('./routes/taskRoutes');
const cors = require('cors');

// Express instance
const app = express();
app.use(
  cors({
    allowedHeaders: ['authorization', 'Content-Type'],
    exposedHeaders: ['authorization'],
    origin: '*',
    methods: 'GET,HEAD,PUT,PATCH,POST,DELETE',
    preflightContinue: false,
  })
);
```

Ilustración 18 - index.js 1

```

app.use(bodyParser.json());
app.use(authRoutes);
app.use(userRoutes);
app.use(taskRoutes);

// MongoDB connection
mongoose.connect(process.env.DB_URI, {
  useNewUrlParser: true,
  useCreateIndex: true,
});

checkConnection();

// Server initialization
const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {
  console.log(`Listening on ${PORT}.`);
});

```

Ilustración 19 - index.js 2

Models

User

Modelo de Mongoose que **representa un usuario**, a partir del cual se toman y almacenan en la base de datos los documentos con las propiedades que se indican en su esquema:

```

const userSchema = new mongoose.Schema({
  googleId: String,
  email: {
    type: String,
    unique: true,
    required: true,
  },
  password: {
    type: String,
    required: true,
  },
  name: String,
  city: String,
});

```

Ilustración 20 – User

Se establecen **dos métodos** o Hooks de Mongoose:

- **userSchema.pre**: Este método se ejecuta siempre antes de almacenar un nuevo usuario en la base de datos. En nuestro caso se utiliza para realizar la encriptación de la contraseña antes de guardar los datos en la base de datos.

```

userSchema.pre('save', function (next) {
  const user = this;

  if (!user.isModified('password')) {
    return next();
  }

  bcrypt.genSalt(10, (err, salt) => {
    if (err) {
      return next(err);
    }

    bcrypt.hash(user.password, salt, (err, hash) => {
      if (err) {
        return next(err);
      }

      user.password = hash;
      next();
    });
  });
});

```

Ilustración 21 - userSchema.pre

- **userSchema.methods.comparePassword**: Si se recibe una contraseña como parámetro, la compara con la contraseña del usuario encriptada (mediante la librería bcrypt), retornando una promesa con el resultado (se usa, por ejemplo, en el login de usuario).

```

userSchema.methods.comparePassword = function (candidatePassword) {
  const user = this;

  return new Promise((resolve, reject) => {
    bcrypt.compare(candidatePassword, user.password, (err, isMatch) => {
      if (err) {
        return reject(err);
      }

      if (!isMatch) {
        return reject(false);
      }

      resolve(true);
    });
  });
};

```

Ilustración 22 - userSchema.methods.comparePassword

Task

Modelo de Mongoose que **representa una tarea**, a partir del cual se toman y almacenan en la base de datos los documentos con las propiedades que se indican en su esquema:

```
const mongoose = require('mongoose');

const taskSchema = new mongoose.Schema({
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },
  title: {
    type: String,
    required: true,
```

Ilustración 23 - Task 1

```
const mongoose = require('mongoose');

const taskSchema = new mongoose.Schema({
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },
  title: {
    type: String,
    required: true,
  },
  description: {
    type: String,
    required: true,
  },
  day: {
    type: String,
    default: '',
  },
  duration: {
    type: Number,
    default: 0,
  },
  repetition: {
    type: String,
    default: '',
  },
  ...
```

Ilustración 24 - Task 2

```

        category: {
            type: String,
            default: 'Other',
        },
        color: {
            type: String,
            default: '#C830CC',
        },
        isPomodoro: {
            type: Boolean,
            default: false,
        },
        creationDate: {
            type: Date,
            default: new Date(),
        },
        isDone: {
            type: Boolean,
            default: false,
        },
    });
}

mongoose.model('Task', taskSchema);

```

Ilustración 25 - Task 3

Middlewares

checkFieldsToUpdate

Esta middleware está pensada para todas las **peticiones que requieren la actualización de datos de un documento**. Toma todos los campos del “body” de una petición y retorna sólo los campos a actualizar. Internamente se ayuda de dos métodos auxiliares (getFieldsFromBody() y deleteFieldsWithoutValue()) para comprobar qué campos tienen asociado un valor, descartando de esta manera cuáles deben ser actualizados y cuáles no.

```

module.exports = (req, res, next) => {
    let fields = getFieldsFromBody(req.body);
    fieldsToUpdate = deleteFieldsWithoutValue(fields);
    req.fieldsToUpdate = fieldsToUpdate;
    next();
};

```

Ilustración 26 - checkFieldsToUpdate

newTaskDataHandler

Como su propio nombre indica, esta middleware se encarga de **manejar los datos** cuando se recibe la petición para **crear una nueva tarea**. Por ejemplo, el dato “duration” se recibe como un String desde el frontend, teniendo que castearlo al valor oportuno del tipo Number.

La existencia de esta función se debe a que hay ciertos datos que se manejan con distintos tipos de datos en el frontend y en el backend/base de datos, y es necesaria una conversión para manejarlos.

```
4 module.exports = (req, res, next) => {
5   req.body.day = handleDataType('day', req.body.day);
6   req.body.duration = handleDataType('duration', req.body.duration);
7   req.body.color = handleDataType('color', req.body.color);
8   req.body.isPomodoro = handleDataType('isPomodoro', req.body.isPomodoro);
9   next();
10};
```

Ilustración 27 - newTaskDataHandler 1

```
13 function handleDataType(dataType, data) {
14   switch (dataType) {
15     case 'day':
16       if (data === 'Today') {
17         return new Date().toLocaleString('en-GB', { weekday: 'long' })
18       }
19     case 'duration':
20       if (data === '5 min') {
21         return 5;
22       }
23       if (data === '15 min') {
24         return 15;
25       }
26       if (data === '30 min') {
27         return 30;
28       }
29       if (data === '45 min') {
30         return 45;
31       }
32       if (data === '1 h') {
33         return 60;
34       }
35       if (data === '2 h') {
36         return 120;
37       }
38       if (data === '3 h') {
39         return 180;
40       }
41       if (data === '4 h') {
42         return 240;
43       }
44       if (data === '5 h') {
45         return 300;
```

Ilustración 28 – newTaskDataHandler 2

requireAuth

Esta función **valida al usuario según su token**. Se implementa en todas esas rutas en las que debe ser validada la identidad del usuario para ser ejecutadas. El funcionamiento es sencillo:

Toma el token que recibe en la petición y lo verifica con la librería “jsonwebtoken”, utilizando la key que se encuentra en las variables de entorno. Si se valida el token, se retorna el usuario con la id asociada al mismo.

```
module.exports = (req, res, next) => {
  const { authorization } = req.headers;

  if (!authorization) {
    return res.status(401).send({ error: 'You must be logged in.' });
  }

  const token = authorization.replace('Bearer ', '');

  jwt.verify(token, process.env.TOKEN_KEY, async (err, payload) => {
    if (err) {
      return res.status(401).send({ error: 'You must be logged in.' });
    }

    const { userId } = payload;

    const user = await User.findById(userId);
    req.user = user;
    next();
  });
};
```

Ilustración 29 – requireAuth

Routes

A continuación se detallan las routes implementadas en el backend. Como ya se ha explicado, las route son un **método de Express que manejan peticiones y respuestas**. Para comprobar su funcionalidad en entorno de desarrollo, se decide hacer uso de la aplicación **Postman**, el cual nos permite hacer peticiones a nuestro servidor local o en producción, especificando las cabeceras y los datos que se pasan en el cuerpo de las mismas, pudiendo de esta forma probar nuestras routes de forma aislada sobre el backend, independientemente del frontend.

Method	Endpoint	Description
POST	addTask	
GET	listTodayTasks	
GET	listTasks	
DET	getCategories	
DEL	deleteTask	
POST	updateTask	

Authorization Header:

Key	Value
Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2Vy...

Body (Pretty):

```

1  []
2   "Category",
3   "Done",
4   "Leisure",
5   "Readings",
6   "Routines",
7   "Study"
8  []

```

Ilustración 30 - Detalle Postman 1

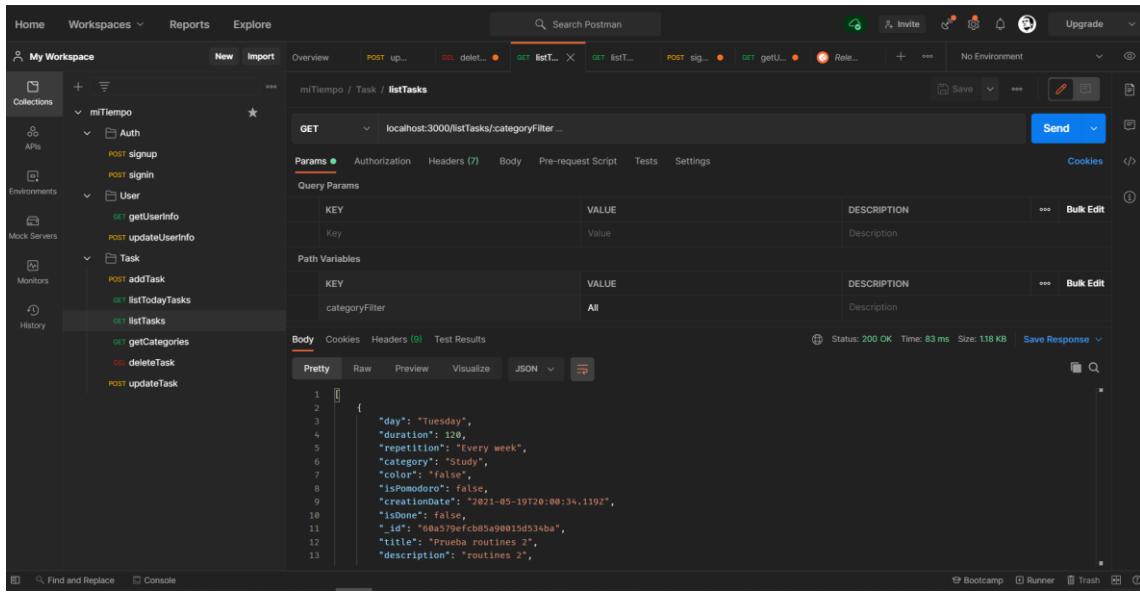


Ilustración 31 - Detalle Postman 2

authRoutes

Implementa todas las routes que tienen que ver con la **autenticación del usuario**:

/signup

Trata de **almacenar un nuevo usuario** en la base de datos si el mismo no existe, devolviendo al usuario (al frontend) un token que es generado en ese mismo momento.

```
router.post('/signup', async (req, res) => {
  const { email, password } = req.body;

  try {
    const user = new User({ email, password });
    await user.save();
    const token = jwt.sign({ userId: user._id }, process.env.TOKEN_KEY);
    res.send({ token });
  } catch (err) {
    return res.status(422).send(err.message);
  }
});
```

Ilustración 32 - /signup

/signin

Trata de **logar a un usuario**. Para ello, comprueba si hay algún usuario con el email que recibe en el cuerpo de la petición y verifica, en tal caso, la contraseña. Si el proceso tiene éxito, se devuelve al frontend un token.

```

router.post('/signin', async (req, res) => {
  const { email, password } = req.body;

  if (!email || !password) {
    return res
      .status(422)
      .send({ error: 'Must provide email and password.' });
  }

  const user = await User.findOne({ email });

```

Ilustración 33 - /signin 1

```

if (!user) {
  return res.status(422).send({ error: 'Invalid password or email.' });
}

try {
  await user.comparePassword(password);
  const token = jwt.sign({ userId: user._id }, process.env.TOKEN_KEY);
  res.send({ token });
} catch (err) {
  return res.status(422).send({ Error: 'Invalid password or email.' });
}
);

```

Ilustración 34 - /signin 2

/auth/google

Ruta de entrada para el **alta de usuario** a través del sistema de autenticación de **Google**. Destacar que para todo este proceso se implementa el framework Passport.js, el cual es de suma utilidad para manejar autenticaciones complejas en este entorno, como es el caso.

Se implementa la **estrategia de Google en Passport**, redirigiéndonos a la misma.

```

router.get(
  '/auth/google',
  passport.authenticate('google', {
    scope: ['profile', 'email'],
  })
);

```

Ilustración 35 - /auth/Google

Configuración de la estrategia de Google en Passport

Ligada a la ruta /auth/google, **permite hacer uso de la estrategia de Google en Passport**. La idea detrás de todo este proceso consiste en que cuando el usuario quiere darse de alta en la aplicación con Google, se le redirige a la pantalla de autenticación de Google y el usuario introduce o selecciona su cuenta de Google para darse de alta. Esta estrategia es la encargada de gestionar la respuesta a esta acción. Para ello, toma las claves

almacenadas de forma segura en las variables de entorno y verifica si el usuario ya existe en la base de datos:

- Si el **usuario existe**, no hace nada, más que asignar un mensaje informando que ya dispone de una contraseña.
- Si **no existe**, se genera un nuevo usuario en la base de datos a partir de la GoogleId e email de la cuenta de Google del usuario, además de una clave aleatoria que se genera con el método auxiliar generatePassword().

Esta estrategia de Google tiene una **callback** especificada, que es la ruta a la que se retorna una vez el usuario selecciona o introduce la cuenta de Google.

```
< passport.use(
  < new GoogleStrategy(
    < {
      clientID: process.env.GOOGLE_CLIENT_ID,
      clientSecret: process.env.GOOGLE_CLIENT_SECRET,
      callbackURL: '/auth/google/callback',
    },
    < async (accessToken, refreshToken, profile, done) => {
      const existingUser = await User.findOne({ googleId: profile.id });
      let newPassword = generatePassword();
      this.password = newPassword;

      if (existingUser) {
        this.password =
          'You already have a valid password. Check the registration email or reply to this email to request a new password.';
        return done(null, existingUser);
      }
      const user = await new User({
        googleId: profile.id,
        email: profile.emails[0].value,
        password: newPassword,
        name: profile.displayName,
      }).save();
      done(null, user);
    }
  );
< /**
  * Function that generates a random password */
< function generatePassword() {
  return Math.floor(10000000 + Math.random() * 90000000);
}
```

Ilustración 36 - Estrategia Google

/auth/Google/callback

Esta ruta **envía un email al usuario** con la contraseña generada en la estrategia de Google haciendo uso del servicio Nodemailer.

```
router.get(
  '/auth/google/callback',
  passport.authenticate('google', { failureRedirect: '/', session: false }),
  async (req, res) => {
    var transporter = nodemailer.createTransport(
      sendGridTransport({
        auth: {
          api_key: process.env.SENDGRID_API,
        },
      })
    );
  }
);
```

Ilustración 37 - /auth/Google/callback

```

        transporter.sendMail({
          from: 'mitiempo.phesan@gmail.com',
          to: req.user.email,
          subject: 'Welcome to miTiempo',
          html: `<strong><h1 style="color: #C830CC">Welcome to miTiempo</h1></strong>
<br>Here are your account details, necessary to log into the app:
<br><br>- <strong style="color: #C830CC">Email</strong>: ${req.user.email}
<br>- <strong style="color: #C830CC">Password</strong>: ${this.password}
<br><br>Thank you and enjoy miTiempo!
<br><br><br><em>miTiempo - Made with ❤️ by Pablo Herrero</em>`,
        });
      }

      res.redirect('https://mitiempoapp.netlify.app/signupconfirmation');
    );
  }
);

```

Ilustración 38 - /auth/Google/callback 2

taskRoutes
/addTask

Esta route **almacena una nueva tarea** asociada al usuario de la id que recibe en la petición. Cabe destacar la implementación de la middleware requireAuth, la cual se implementa en el resto de las rutas de esta sección, así como de newTaskDataHandler.

```

try {
  const task = new Task({
    title,
    description,
    day,
    duration,
    repetition,
    category,
    color,
    isPomodoro,
    creationDate,
    isDone,
    userId: req.user._id,
  });
  await task.save();
  res.send(task);
} catch (error) {
  res.status(422).send({ error: error.message });
}
);

```

Ilustración 39 - /addTask

/listTasks/:categoryFilter

A partir de esta ruta se devuelve una **lista de todas las tareas** almacenadas en la base de datos para el usuario que se valida. Estas tareas pueden estar filtradas por categoría o no, según el parámetro “categoryFilter” que se recibe como parámetro en la petición.

```

router.get('/listTasks/:categoryFilter', requireAuth, async (req, res) => {
  try {
    if (req.params.categoryFilter === 'All') {
      const tasks = await Task.find({ userId: req.user._id })
        .sort({
          creationDate: -1,
        })
        .where('category')
        .ne('Done');
      res.send(tasks);
    } else {
      const tasksFiltered = await Task.find({
        userId: req.user._id,
        category: req.params.categoryFilter,
      }).sort({ creationDate: -1 });

      res.send(tasksFiltered);
    }
  } catch (error) {
    return res.status(422).send({
      Error: 'Something went wrong retrieving tasks. Try again.',
    });
  }
});

```

Ilustración 40 - /listTasks

/listTodayTasks/:categoryFilter

Similar a la anterior, pero **filtrando las tareas asignadas al día actual**.

```

router.get('/listTodayTasks/:categoryFilter', requireAuth, async (req, res) => {
  try {
    if (req.params.categoryFilter === 'All') {
      const tasks = await Task.find({
        userId: req.user._id,
        day: new Date().toLocaleString('en-GB', { weekday: 'long' }),

```

Ilustración 41 - listTodayTasks 1

```

        .sort({ creationDate: -1 })
        .where('category')
        .ne('Done');
      res.send(tasks);
    } else {
      const tasksFiltered = await Task.find({
        userId: req.user._id,
        category: req.params.categoryFilter,
        day: new Date().toLocaleString('en-GB', { weekday: 'long' }),
      }).sort({ creationDate: -1 });
      res.send(tasksFiltered);
    }
  } catch (error) {
    return res.status(422).send({
      Error: 'Something went wrong retrieving today tasks. Try again.',
    });
  }
});

```

Ilustración 42 - /listTodayTasks 2

/listCategories

Esta ruta **lista todas las categorías** asignadas a las tareas del usuario que valida.

```
router.get('/listCategories', requireAuth, async (req, res) => {
  try {
    const categories = await Task.find().distinct('category');
    res.send(categories);
  } catch (error) {
    return res.status(422).send({
      Error: 'Something went wrong retrieving categories. Try again.',
    });
  }
});
```

Ilustración 43 - /listCategories

/updateTask

Route que **actualiza** en la base de datos **una tarea** específica a partir de su id. Implementa todas las middlewares vistas, pues necesita validar el usuario, manejar los datos que se reciben desde el frontend y comprobar los campos a actualizar.

```
router.post(
  '/updateTask',
  requireAuth,
  newTaskDataHandler,
  checkFieldsToUpdate,
```

Ilustración 44 - /updateTask 1

```
router.post(
  '/updateTask',
  requireAuth,
  newTaskDataHandler,
  checkFieldsToUpdate,
  async (req, res) => {
    const taskId = req.body.taskId;
    let fieldsToUpdate = req.fieldsToUpdate;

    try {
      const task = await Task.findByIdAndUpdate(
        taskId,
        { $set: { ...fieldsToUpdate } },
        {
          runValidators: true,
          new: true,
        }
      );
      res.send(task._id + ' successfully updated.');
    } catch (error) {
      res.status(422).send({ error: error.message });
    }
  }
);
```

Ilustración 45 - /updateTask 2

/deleteTask

Esta ruta permite **eliminar** de la base de datos **una tarea** específica a partir de la id de la misma que recibe en el cuerpo de la petición.

```
router.delete('/deleteTask', requireAuth, async (req, res) => {
  const taskId = req.body.taskId;

  try {
    const taskToDelete = await Task.findByIdAndDelete(taskId);
    res.send(taskToDelete._id + ' successfully deleted.');
  } catch (error) {
    res.status(422).send({ error: error.message });
  }
});
```

Ilustración 46 - /deleteTask

userRoutes

/getUserInfo

Haciendo uso de esta ruta se consigue la **información de un usuario** específico. Para ello, se recibe un usuario en la petición, del cual se coge su id para extraer los datos de la base de datos, siempre habiendo superado la validación a través de la middleware de autorización.

```
router.get('/getUserInfo', requireAuth, async (req, res) => {
  try {
    const user = await User.findById(req.user._id);
    const email = user.email;
    const name = user.name;
    const city = user.city;
    res.send({ email, name, city });
  } catch (error) {
    return res.status(422).send({
      error: 'Something went wrong retrieving user information. Try again.',
    });
  }
});
```

Ilustración 47 - /getUserInfo

/updateUserInfo

Route que permite **actualizar la información del usuario** que recibe en la petición. Para ello, implementa las middlewares requireAuth y checkFieldsToUpdate. Si el proceso de validación del usuario tiene éxito, actualiza el usuario con los nuevos datos.

```
router.post(
  '/updateUserInfo',
  requireAuth,
  checkFieldsToUpdate,
  async (req, res) => {
    const userId = req.user._id;
    let fieldsToUpdate = req.fieldsToUpdate;
```

Ilustración 48 - /updateUserInfo 1

```
try {
  const user = await User.findByIdAndUpdate(
    userId,
    { $set: { ...fieldsToUpdate } },
    {
      runValidators: true,
      new: true,
    }
  );
  await user.save();
  res.send(user);
} catch (error) {
  res.status(422).send({ error: error.message });
}
```

Ilustración 49 – /updateUserInfo 2

Frontend

El **frontend** o la interfaz de usuario de miTiempo está desarrollado con **React Native**, framework que permite crear aplicaciones multiplataforma haciendo uso de la librería de JavaScript React. Gracias a estas tecnologías, **se consiguen tres aplicaciones nativas** con un solo código, las cuales están **comunicadas con el backend**:

- **Web-app:** Aplicación web alojada en Heroku y desde la que cualquier usuario puede acceder introduciendo la url en su navegador o desde la landing page.
- **Aplicación Android:** Aplicación de Android nativa, que el usuario podría descargar desde el Play Store.
- **Aplicación iOS:** Aplicación de iOS nativa, que el usuario podría descargar desde el App Store.

También se hace uso de **Expo**, framework para React Native consistente en un conjunto de servicios y herramientas para ayudar al desarrollador en el manejo de React Native, así como del framework de UI **React Native Elements**, el cual nos permite implementar componentes prefabricados con mayores funciones y opciones de personalización.

Conceptos esenciales

Antes de pasar a detallar y explicar el funcionamiento del frontend, se antoja indispensable explicar los **conceptos esenciales** en los que se basa:

- **Funcionamiento de React Native:** Al desarrollar con React Native estamos hablando de **desarrollo nativo**, pero ¿qué quiere decir esto? Con un solo código en JavaScript creamos aplicaciones nativas en cada una de las plataformas. Cada **componente de React Native tiene su equivalente en la plataforma nativa y mediante un puente o “bridge” se ejecuta en la plataforma específica**. A diferencia de otros frameworks como Ionic, los cuales hacen uso de Cordova para realizar aplicaciones híbridas que renderizan o muestran la UI en una webview, con React Native la renderización se implementa sobre componentes nativos.

Para comprender el **funcionamiento interno**, es conveniente detallar cómo se ejecuta una aplicación realizada con React Native. Aunque la lógica nos puede llevar a pensar que React Native transforma en tiempo de compilación el código escrito por el desarrollador en JavaScript a código nativo de la plataforma específica (Objective-C/Swift para iOS, Java/Kotlin para Android), esta aproximación es errónea, puesto que hablamos de lenguajes con características e implementaciones muy distintas que harían muy complicada esa transformación.

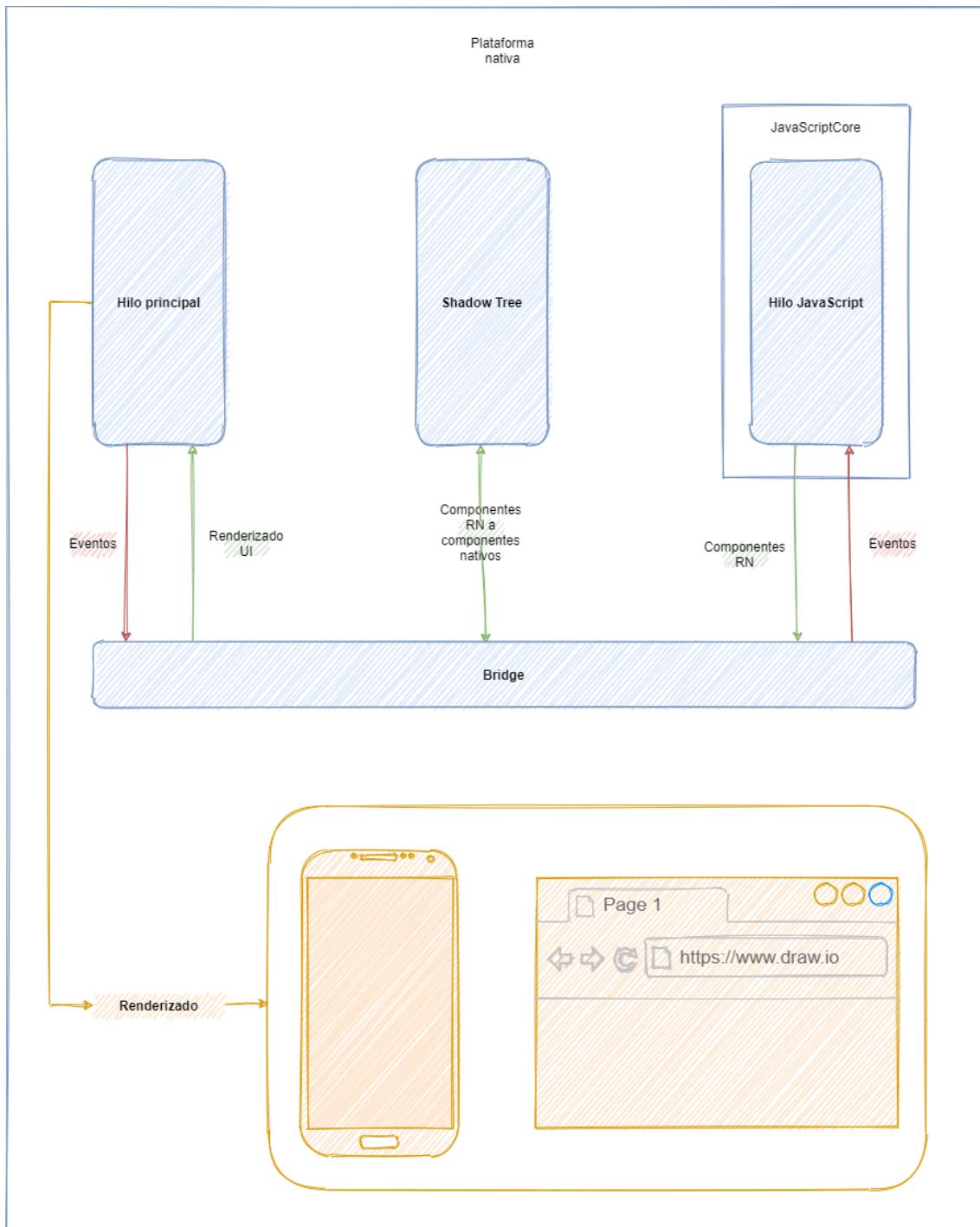
React Native se basa en una serie de **componentes que tienen su equivalencia en componentes nativos**. Es decir, un TextView de React Native tiene su equivalencia en React (para la plataforma Web), en Objective-C/Swift (para iOS) y en Java/Kotlin (para Android). Lo más inteligente y sensato es **comunicar** los componentes de React Native con los componentes nativos.

Cuando una aplicación React Native **se ejecuta**, internamente se lanzan **tres hilos**:

- Inicialmente se ejecuta un **hilo principal**, encargado de modificar la interfaz nativa y que está constantemente escuchando y enviando eventos e información al resto de hilos, además de modificar y renderizar la interfaz en la pantalla del usuario.
- El hilo principal se encarga en primera instancia de iniciar el **hilo JavaScript**, encargado de correr el código escrito por el desarrollador y que se ejecuta sobre el motor JavaScriptCore (incluido en iOS, pero no en Android, por lo que se incorpora al ejecutable de esa plataforma).
- El hilo JavaScript envía la información de lo que necesita que sea renderizado en pantalla. Esta información es usada por un tercer hilo, el hilo sombra o **Shadow tree**, el cual consiste en un árbol de nodos, donde cada nodo es una representación nativa dinámica de los componentes declarados por el desarrollador en el archivo JavaScript. Es en este hilo donde se

transforma la UI de React Native a la UI nativa, haciendo uso del motor de diseño **Yoga**, que convierte el diseño basado en flexbox de React Native al sistema que entiende la plataforma nativa.

- La comunicación entre todos los hilos se hace mediante un puente o **Bridge**: escrito en Java/C++, mapea e implementa una comunicación asíncrona y bidireccional sobre los hilos indicados. Simplemente, serializa los datos en formato JSON y los transfiere a través del mismo hacia y desde los hilos.



Como **resultado**, tenemos una aplicación que corre de forma nativa en la plataforma específica, con el mismo rendimiento y experiencia de cara al usuario.

Los **beneficios** en conceptos de tiempo y forma con este tipo de desarrollos son enormes, pues un solo equipo puede encargarse del mantenimiento de una aplicación en tres plataformas a la vez. Por otro lado, es **tendencia**, grandes compañías han desarrollado sus propios frameworks para el desarrollo multiplataforma nativo, como el caso de Flutter (Google).

- **JSX**: Extensión de la sintaxis de JavaScript que permite combinar JavaScript y HTML. Utilizado por React Native para el renderizado de los componentes.
- **Componente**: Es una clase o función que puede recibir una serie de *Props* o propiedades y manejar el *State* o estado, retornando código JSX a renderizar. En este proyecto, dado el uso de *Hooks* que se implementa en el mismo, se hace uso de componentes de función o funcionales, los cuales se basan en funciones, a diferencia de los componentes de clase, basados en clases de JavaScript.

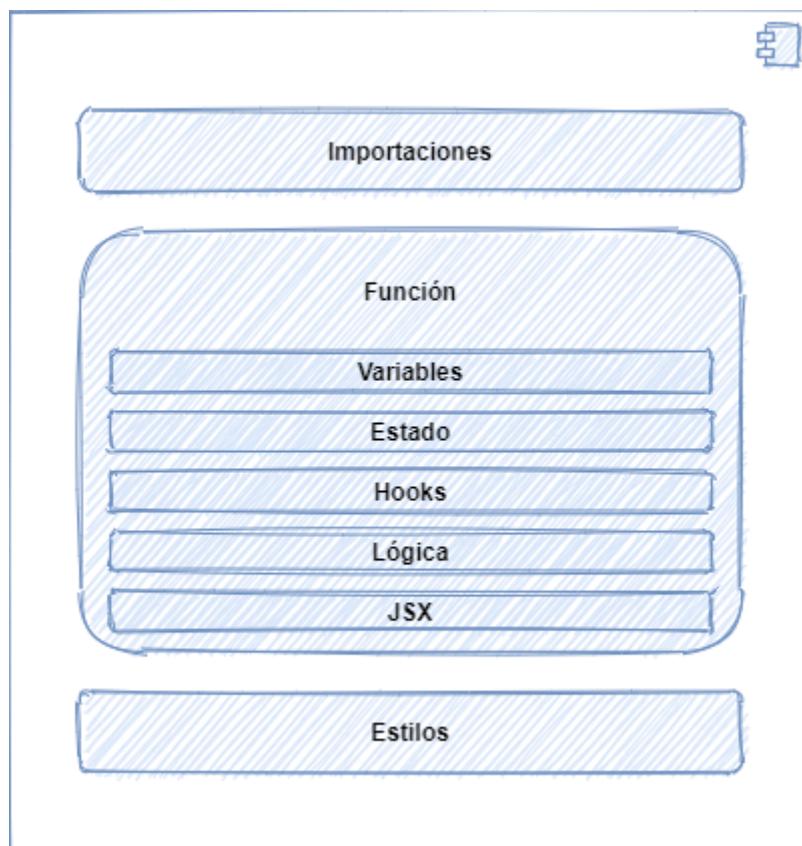


Ilustración 51 - Componente React Native

- **Ciclo de vida de un componente:** Un componente en React tiene tres partes vitales esenciales:
 - o Montaje o primer renderizado.
 - o Actualización (re-renderizado).
 - o Desmontaje.
 - o Error.
- **Comunicación en React:** La comunicación en React es **unidireccional**, realizándose el paso de *Props* o pequeñas piezas de información de un componente padre a un hijo. El padre, puede responder a eventos lanzados desde el componente hijo para manejar las respuestas de este, pero no recibir información directa desde este, como sí se hace en frameworks como Angular.
- **Event bubbling:** Cuando se genera un evento en un componente hijo, se propaga hacia arriba hacia los componentes padre.
- **Estado:** Objeto de JavaScript que contiene información relevante del componente. A efectos, podría decirse que un estado está compuesto de diferentes variables del componente, las cuales pueden ser modificadas mediante *setters*. Una actualización del estado implica la re-renderización del componente.
- **Prop:** Es un sistema para pasar información de un componente padre a un componente hijo. Estas propiedades son variables que se reciben como parámetro en el componente hijo.

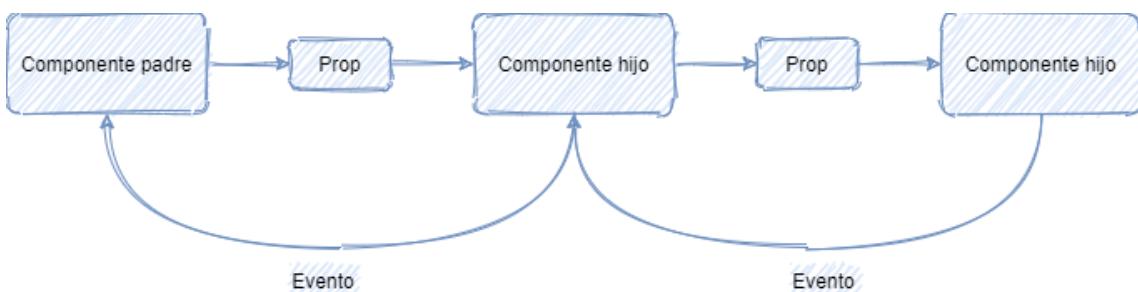


Ilustración 52 – Props

- **Hook:** Son funciones que aportan funcionalidad adicional a los componentes. Las hay de diversos tipos y con diversas funcionalidades. Entre las más destacadas y, a su vez, más utilizadas en miTiempo, se encuentra:
 - o **useState()**: Utilizada para manejar el estado del componente.
 - o **useEffect()**: Con la que se maneja el ciclo de vida del componente.
 - o **useContext()**: Permite hacer uso del *Context*.
- **Context:** Es un objeto que permite pasar datos y manejar el estado a través del árbol de componentes sin tener que pasar *props* manualmente en cada nivel. Cuando un componente se suscribe al *Provider* o proveedor de un contexto, permite el manejo de la información de dicho contexto entre un componente padre y sus componentes hijos.

hijos anidados, permitiendo pasar información fácilmente a un componente hijo específico, sin necesidad de pasar props a través de cada nivel de componentes hijos.

- **Provider:** Función que provee de un contexto a los componentes hijos asignados.
- **Reducer:** Es una función que determina cambios en el estado, usando la acción o *action function* que recibe para determinar ese cambio. Tiene acceso a todos los estados de la app y se implementa con el Hook `useReducer()`.
- **Action function:** Función que toma o modifica el *State* dentro de un *Reducer*.

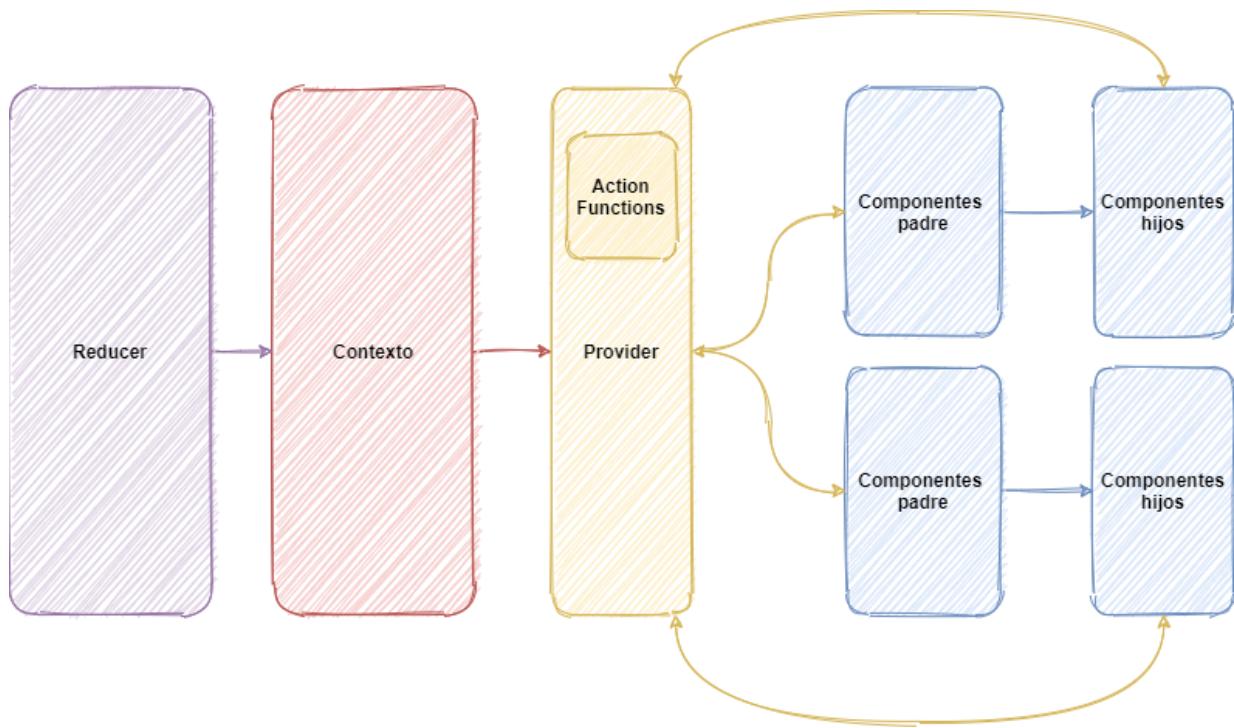


Ilustración 53 - Flujo de Context

Funcionamiento

Teniendo en cuenta los conceptos anteriormente explicados, se procede a dar una explicación general del **funcionamiento** del frontend:

- **Inicialización:** El interfaz nace en App.js, donde se realiza la configuración de los proveedores de contexto, los navegadores.
- **Renderizado:** Se procede a renderizar el componente de inicio, pudiéndose realizar una primera petición a los proveedores para cargar datos.
- **Interacción del usuario:** Cuando el usuario realiza una acción en el interfaz, la aplicación **reacciona**, generando **dos tipos de flujos** a seguir, según la interacción realizada:
 - Se manda una **orden al navegador o al navegador externo** para realizar un cambio de pantalla.

- O se lanza una **Action function** con la que se modifica el estado en un proveedor, pudiendo hacerse una petición al backend.
- **Re-renderizado:** Si el proveedor genera un cambio en el estado del componente, se realiza una actualización del mismo, volviéndose a renderizar y mostrando la información actualizada en la pantalla del usuario.

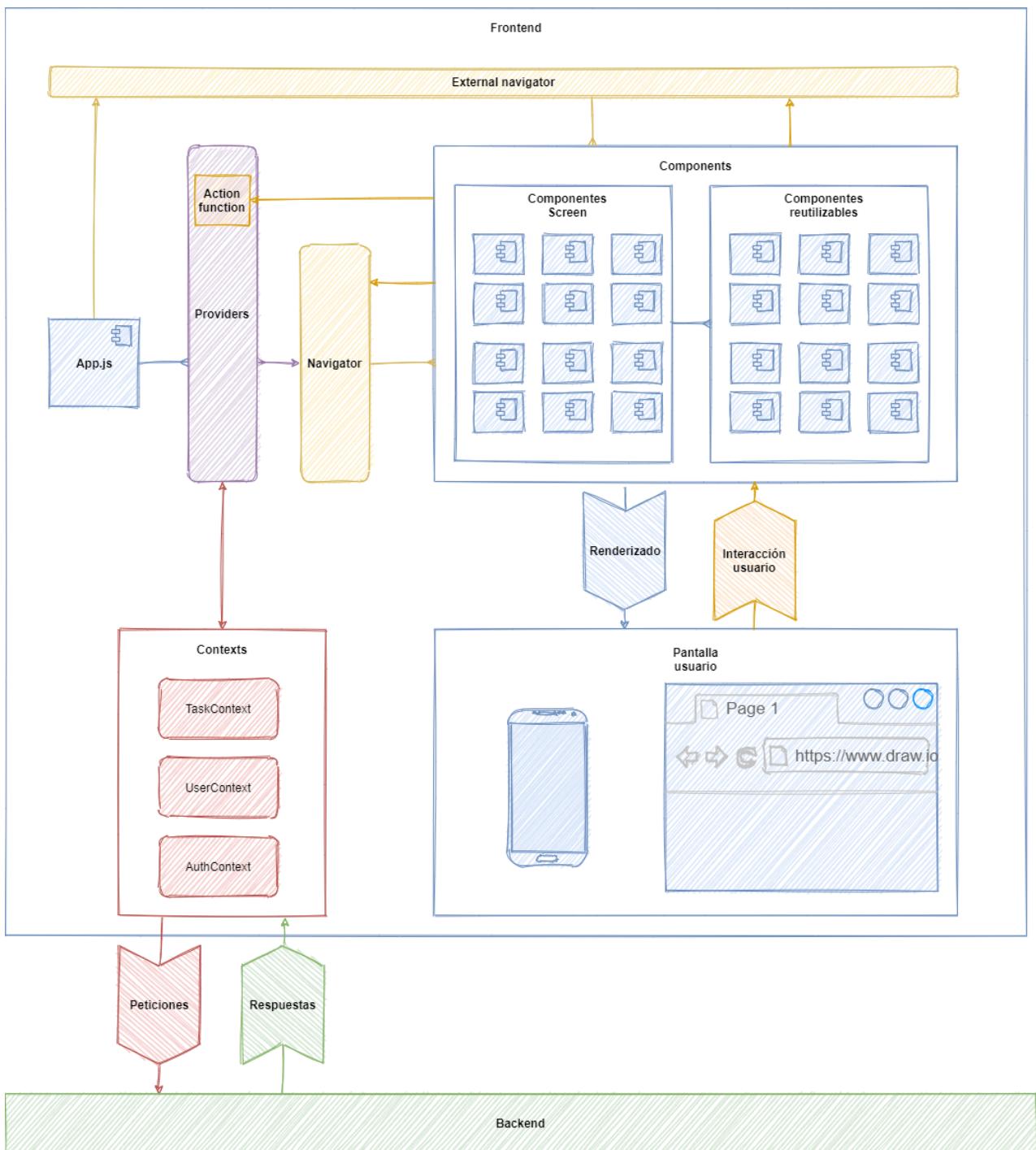


Ilustración 54 - Funcionamiento Frontend

Detalle

A continuación se procede a explicar en detalle el frontend.

App.js

Es el **punto de entrada** de la aplicación de React Native. En ella se realiza la configuración de los sistemas de navegación y de los proveedores de contexto.

```
const App = createAppContainer(appNavigator());

export default () => {
  return (
    <SafeAreaProvider>
      <TaskProvider>
        <UserProvider>
          <AuthProvider>
            <App ref={setNavigator} />
          </AuthProvider>
        </UserProvider>
      </TaskProvider>
    </SafeAreaProvider>
  );
};
```

Ilustración 55 - App.js

api

miTiempoApi.js

Instancia de la librería Axios, cliente HTTP utilizado para realizar peticiones al backend.

Se configura con la url donde está ubicado el backend, en Heroku, y un interceptor que fuerza la validación del token del usuario para cada petición que se hace a la api.

Esta instancia de Axios es llamada desde cualquier punto del frontend siempre que sea necesario hacer una **petición a la api** del backend.

```
import axios from 'axios';
import AsyncStorage from '@react-native-community/async-storage';

const instance = axios.create({
  baseURL: 'https://mitiempo-back.herokuapp.com/',
});
```

Ilustración 56 - miTiempoApi.js 1

```

instance.interceptors.request.use(
  async (config) => {
    const token = await AsyncStorage.getItem('token');
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => {
    return Promise.reject(error);
  }
);

export default instance;

```

Ilustración 57 - miTiempoApi.js 2

context

createDataContext.js

Función que **automatiza el proceso de creación de un contexto y su proveedor** a partir de un reducer, para ser usados posteriormente en la aplicación. Esta función es esencial a lo largo de toda la aplicación puesto que, sin ella, tendríamos que crear un reducer, un proveedor y un contexto para cada tipo de dato a manejar, duplicando código innecesariamente.

Recibe **tres parámetros**:

- **Reducer**: Función que determina cambios en el estado según las acciones o Action Functions que recibe.
- **Actions**: Action Functions a utilizar por el reducer.
- **defaultValue**: El valor inicial del estado.

Con ellos, la función crea un contexto y un proveedor que son retornados por la misma.

```

export default (reducer, actions, defaultValue) => {
  const Context = React.createContext();

  const Provider = ({ children }) => {
    const [state, dispatch] = useReducer(reducer, defaultValue);

    const boundActions = {};
    for (let key in actions) {
      boundActions[key] = actions[key](dispatch);
    }

    return (
      <Context.Provider value={{ state, ...boundActions }}>
        {children}
      </Context.Provider>
    );
  };

  return { Context, Provider };
};

```

Ilustración 58 - createDataContext.js

AuthContext.js

Objeto Context que **representa el estado referente a la autenticación** en la aplicación, es decir, la información referente al alta, login y cierre de sesión de un usuario en la misma.

Este objeto se crea en **tres partes**:

- Instancia **reducer**: Se implementa un reducer que recibe el estado global y una serie de Action Functions, modificando el state según las acciones a implementar.

```
const authReducer = (state, action) => {
  switch (action.type) {
    case 'signInUp':
      return { errorMessage: '', token: action.payload };
    case 'add_error':
      return { ...state, errorMessage: action.payload };
    case 'clear_error_message':
      return { ...state, errorMessage: '' };
    case 'signout':
      return { token: null, errorMessage: '' };
    default:
      return state;
  }
}
```

Ilustración 59 – authReducer

- Definición de **Action Functions**: Son funciones destinadas a modificar el estado a través del reducer definido. Con ellas, se realizan las peticiones necesarias al backend, modificando en consecuencia el estado con la respuesta que se recibe del mismo.
 - o **signup()**: Valida el formulario y envía una petición al backend para dar de alta a un nuevo usuario. Si el alta es aceptada, toma el token que recibe desde el backend para logar al usuario.

```
const signup =
  (dispatch) =>
  async ({ email, password }) => {
    let infoMessage = validateForm(email, password);

    if (infoMessage === '') {
      try {
        const response = await miTiempoApi.post('/signup', {
          email,
          password,
        }); // make post request to backend
        await AsyncStorage.setItem('token', response.data.token); // store token
        dispatch({ type: 'signInUp', payload: response.data.token });
        navigate('mainFlow'); // navigate to main flow
      } catch (error) {
        console.error(error);
        dispatch({ type: 'add_error', payload: error.message });
      }
    }
  }
}
```

Ilustración 60 - signup() 1

```

        } catch (error) {
            dispatch({
                type: 'add_error',
                payload: 'The email entered is already registered',
            });
        }
    } else {
        dispatch({
            type: 'add_error',
            payload: infoMessage,
        });
    }
};

```

Ilustración 61 - signup() 2

- signin(): Valida el formulario y envía una petición al backend para logar al usuario en el sistema. Si el login es aceptado, toma el token que recibe desde el backend para logar al usuario.

```

const signin =
  (dispatch) =>
  async ({ email, password }) => {
    let infoMessage = validateForm(email, password);

    if (infoMessage === '') {
      try {
        const response = await miTiempoApi.post('/signin', {
          email,
          password,
        }); // make post request to backend
        await AsyncStorage.setItem('token', response.data.token); // store token
        dispatch({ type: 'signInUp', payload: response.data.token }); // dispatch action
        navigate('mainFlow'); // navigate to main flow
      } catch (error) {
        dispatch({
          type: 'add_error',
          payload: 'Invalid password or email',
        });
      }
    } else {
      dispatch({
        type: 'add_error',
        payload: infoMessage,
      });
    }
  };

```

Ilustración 62 - signin()

- tryLocalSignIn(): Función que comprueba si el usuario tiene un token de validación almacenado en su equipo. De ser así, loga al usuario automáticamente, si no, redirige a la pantalla de alta.

```

const tryLocalSignIn = (dispatch) => async () => {
  const token = await AsyncStorage.getItem('token');
  if (token) {
    dispatch({ type: 'signInUp', payload: token });
    navigate('mainFlow');
  } else {
    navigate('Signup');
  }
};

```

Ilustración 63 - tryLocalSignIn()

- o signout(): Función que elimina el token de validación del dispositivo del usuario, cerrando la sesión del mismo.

```

const signout = (dispatch) => async () => {
  await AsyncStorage.removeItem('token');
  dispatch({ type: 'signout' });
  navigate('loginFlow');
};

```

Ilustración 64 - signout()

- Finalmente, el objeto se crea haciendo la llamada a la función **createDataContext()**, vista en el apartada anterior:

```

export const { Provider, Context } = createDataContext(
  authReducer,
  {
    signup,
    signin,
    tryLocalSignIn,
    signout,
    clearErrorMessage,
  },
  { token: null, errorMessage: '' }
);

```

Ilustración 65 - createdataContext()

Como se puede apreciar, **createDataContext()** recibe el reducer y las acciones definidas para crear el contexto y el proveedor del mismo, así como los valores iniciales de las variables del estado que este contexto va a manejar.

Es interesante el **uso de la desestructuración** de JavaScript para tomar el Provider y el Context de **createDataContext()**.

TaskContext.js

Objeto Context que **representa el estado referente a las tareas** en la aplicación. Por lo tanto, este contexto manejará acciones como añadir, listar, actualizar y eliminar tareas, así como inicializar diversos datos iniciales para los paneles de control de las pantallas referentes a las tareas en la aplicación.

Su creación se puede dividir en **tres partes**:

- Instancia **reducer**: Se implementa un reducer que recibe el estado global y una serie de Action Functions, modificando el state según las acciones a implementar.

```
const taskReducer = (state, action) => {
  switch (action.type) {
    case 'listTasks':
      return {
        ...state,
        tasks: action.payload,
      };
    case 'listTodayTasks':
      return {
        ...state,
        todayTasks: action.payload,
      };
    case 'getDays':
      return {
        ...state,
        days: action.payload,
      };
  }
};
```

Ilustración 66 – taskReducer

- Definición de **Action Functions**:

- o addTask(): Función que envía una petición al backend con los datos de una nueva tarea a añadir. Si la petición tiene éxito, se redirecciona al usuario a la pantalla de inicio.

```
const addTask = (dispatch) => async ({
  title,
  description,
  day,
  duration,
  repetition,
  category,
  color,
  isPomodoro,
}) => {
  try {
    await miTiempoApi.post('/addTask', {
      title,
      description,
      day,
      duration,
      repetition,
      category,
      color,
      isPomodoro,
    });
    navigate('TaskHome');
  } catch (error) {
    console.error(error);
  }
};
```

Ilustración 67 - addTask()

- o listTasks(): Función que hace una petición al backend listando las tareas según la categoría que recibe como parámetro. Si la petición tiene éxito, en la respuesta se recibe el listado de tareas filtrado.

```

const listTodayTasks = (dispatch) => async ({ category }) => {
  try {
    const response = await miTiempoApi.get(`/listTodayTasks/${category}`);
    dispatch({ type: 'listTodayTasks', payload: response.data });
  } catch (error) {
    dispatch({
      type: 'add_error',
      payload: 'Something went wrong retrieving today tasks data.',
    });
  }
}

```

Ilustración 68 - listTasks()

- `listTodayTasks()`: Función que hace una petición al backend listando las tareas según la categoría que recibe como parámetro para el día de hoy. Si la petición tiene éxito, en la respuesta se recibe el listado de tareas filtrado.

```

const listTodayTasks = (dispatch) => async ({ category }) => {
  try {
    const response = await miTiempoApi.get(`/listTodayTasks/${category}`);
    dispatch({ type: 'listTodayTasks', payload: response.data });
  } catch (error) {
    dispatch({
      type: 'add_error',
      payload: 'Something went wrong retrieving today tasks data.',
    });
  }
}

```

Ilustración 69 - listTodayTasks()

- `updateTask()`: Función que hace una petición al backend para actualizar una determinada tarea a partir de los datos de la misma que se reciben como parámetros. Si tiene éxito, se redirige al usuario a la pantalla de inicio, habiéndose actualizado la tarea indicada.

```

try {
  await miTiempoApi.post('/updateTask', {
    taskId,
    title,
    description,
    day,
    duration,
    repetition,
    category,
    color,
    isPomodoro,
    isDone,
  });
  navigate('TaskHome');
}

```

Ilustración 70 - updateTask()

- `deleteTask()`: Función que realiza una petición al backend para eliminar la tarea con el id que se recibe como parámetro. Si la petición tiene éxito, se redirige al usuario a la pantalla de inicio.

```

const deleteTask = (dispatch) => async ({ taskId }) => {
  try {
    await miTiempoApi.delete('/deleteTask', { data: { taskId } });
    navigate('TaskHome');
  } catch (error) {
    dispatch({
      type: 'add_error',
      payload: 'Something went wrong deleting the task. Try again.',
    });
  }
}

```

Ilustración 71 - updateTask()

- getDays(), getDaysArray(), getDurations(), getRepetitions(), getCategories(), getColors() y getPomodoro() son acciones auxiliares que se encargan de realizar la carga de los valores iniciales de las opciones del panel de control de las pantallas TaskCreateScreen y TaskDetailScreen.
- Finalmente, el objeto se crea haciendo la llamada a la función **createDataContext()**. Se omite la explicación, pues es la misma que la dada para AuthContext:

```

export const { Provider, Context } = createDataContext(
  taskReducer,
  {
    addTask,
    listTasks,
    listTodayTasks,
    updateTask,
    deleteTask,
    getDays,
    getDurations,
    getRepetitions,
    getCategories,
    getColors,
    getPomodoro,
  },
  {
    tasks: [],
    todayTasks: [],
    days: [],
    durations: [],
    repetitions: [],
    categories: [],
    colors: [],
    isPomodoro: [],
    errorMessage: '',
  }
);

```

Ilustración 72 - createDataContext()

UserContext.js

Objeto Context que **representa el estado referente al usuario** en la aplicación. Por lo tanto, este contexto hace uso de acciones que se encargan de tomar y actualizar la información del usuario.

Su creación se divide en **tres partes**:

- Instancia **reducer**: Se implementa un reducer que recibe el estado global y una serie de Action Functions, modificando el state según las acciones a implementar.

```
const userReducer = (state, action) => {
  switch (action.type) {
    case 'getUserInfo':
      return {
        ...state,
        _id: action.payload.id,
        email: action.payload.email,
        name: action.payload.name,
        city: action.payload.city,
      };
    case 'add_error':
      return { ...state, errorMessage: action.payload };
    default:
      return state;
  }
};
```

Ilustración 73 – userReducer

- Definición de **Action Functions**:

- o `getUserInfo()`: Función que envía una petición al backend para recibir como respuesta, si tiene éxito, la información del usuario logado.

```
const getUserInfo = (dispatch) => async () => {
  try {
    const response = await miTiempoApi.get('/getUserInfo');
    dispatch({
      type: 'getUserInfo',
      payload: response.data,
    });
  } catch (error) {
    dispatch({
      type: 'add_error',
      payload: 'Something went wrong retrieving user data.',
    });
}
```

Ilustración 74 - getUserInfo()

- o `updateUserInfo()`: Función que valida el formulario y envía una petición al backend para actualizar la información del usuario logado a partir de los datos a actualizar que recibe como parámetros. Si la petición tiene éxito, además de actualizar los datos se redirige al usuario a la pantalla `AccountScreen`.

```

const updateUserInfo =
  (dispatch) =>
    async ({ email, name, city, newPassword }) => {
      let infoMessage = validateForm(email, newPassword);
      if (infoMessage === '') {
        try {
          if (newPassword === '') {
            await miTiempoApi.post('/updateUserInfo', {
              email,
              name,
              city,
            });
          } else {
            await miTiempoApi.post('/updateUserInfo', {
              email,
              name,
              city,
              newPassword,
            });
          }
        } catch (error) {
          dispatch({
            type: 'add_error',
            payload: 'Something went wrong updating your data. Try again.',
          });
        }
      } else {
        dispatch({
          type: 'add_error',
          payload: infoMessage,
        });
      }
    }

    navigate('Account');
  } catch (error) {
    dispatch({
      type: 'add_error',
      payload: 'Something went wrong updating your data. Try again.',
    });
  }
} else {
  dispatch({
    type: 'add_error',
    payload: infoMessage,
  });
}

```

Ilustración 75 - updateUserInfo()

- Finalmente, el objeto se crea haciendo la llamada a la función **createDataContext()**. Se omite la explicación, pues es la misma que la dada para AuthContext:

```

export const { Provider, Context } = createDataContext(
  userReducer,
  { getUserInfo, updateUserInfo },
  { email: '', name: '', city: '', errorMessage: '' }
);

```

Ilustración 76 - createDataContext()

navigation
appNavigator.js

Método que retorna el **navegador principal** de la aplicación. Está basado en varios navegadores apilados y flujos de navegación:

- **SwitchNavigator**: Según el caso (switch), redirecciona a dos flujos de información: loginFlow, para todo el proceso de alta y autenticación, y mainFlow, flujo de la app con una sesión ya iniciada.

- **StackNavigator:** El loginFlow se basa en un StackNavigator que permite pasar de la pantalla de alta a la de login y viceversa.
- **BottomTabNavigator:** El mainFlow se basa en una barra de navegación con 4 opciones básicas: homeFlow, Tracker, TaskCreate y accountFlow.

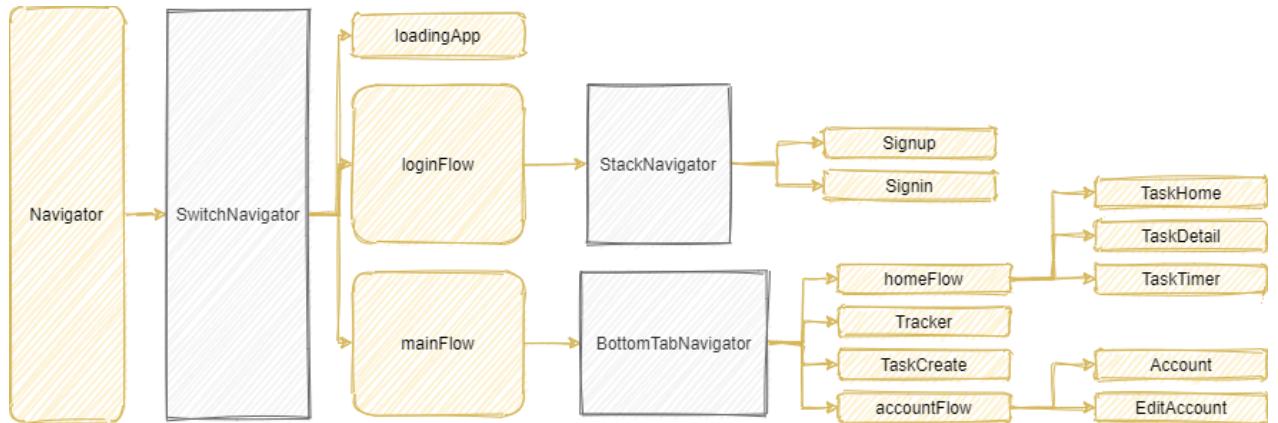


Ilustración 77 - Detalle navegación

```
export default () => {
  const homeFlow = createStackNavigator({
    TaskHome: TaskHomeScreen,
    TaskDetail: TaskDetailScreen,
    TaskTimer: TaskTimerScreen,
  });

  homeFlow.navigationOptions = {
    tabBarIcon: ({ tintColor }) => (
      <FontAwesome5 name="home" size={20} style={{ color: tintColor }} />
    ),
  };
}
```

Ilustración 78 - appNavigator.js 1

```
export default () => {
  const homeFlow = createStackNavigator({
    TaskHome: TaskHomeScreen,
    TaskDetail: TaskDetailScreen,
    TaskTimer: TaskTimerScreen,
  });

  homeFlow.navigationOptions = {
    tabBarIcon: ({ tintColor }) => (
      <FontAwesome5 name="home" size={20} style={{ color: tintColor }} />
    ),
  };

  const accountFlow = createStackNavigator({
    Account: AccountScreen,
    EditAccount: EditAccountScreen,
  });

  accountFlow.navigationOptions = {
    tabBarIcon: ({ tintColor }) => (
      <FontAwesome5 name="cog" size={20} style={{ color: tintColor }} />
    ),
  };
}
```

Ilustración 79 - appNavigator.js 2

```

const switchNavigator = createSwitchNavigator({
  loadingApp: LoadingScreen,
  loginFlow: createStackNavigator({
    Signup: SignupScreen,
    Signin: SigninScreen,
  }),
  mainFlow: createBottomTabNavigator(
    [
      homeFlow,
      Tracker: TrackerScreen,
      TaskCreate: TaskCreateScreen,
      accountFlow,
    ],
    {
      tabBarOptions: {
        showLabel: false,
        activeTintColor: '#C830CC',
        inactiveTintColor: 'grey',
      },
    },
  ),
});

```

Ilustración 80 - appNavigator.js 3

externalNavigator.js

Para situaciones en las que no se tiene **acceso directo al sistema de navegación** de la app, se implementa este método que modifica el estado en la navegación para redirigir a la pantalla deseada.

```

export const navigate = (routeName, params) => {
  externalNavigator.dispatch(
    NavigationActions.navigate({
      routeName,
      params,
    })
);
}

```

Ilustración 81 - externalNavigator.js

Screens

A continuación se muestra el detalle del funcionamiento de las pantallas de la aplicación.

SignupScreen.js

Es la **pantalla inicial** de la aplicación cuando **no hay un usuario con una sesión iniciada**. Desde la misma se ofrece al usuario la opción de darse de **alta mediante un formulario** (email y contraseña), usando el autenticador de **Google** o iniciar sesión si ya dispone de cuenta.

Hace uso de varios componentes personalizados, como AuthForm, NavLink y Spacer, e implementa AuthContext para utilizar, validar y modificar el estado, realizando todas las acciones de alta de usuario.

Si se supera el proceso de alta, se redirige al usuario a TaskHomeScreen.

```
const SignupScreen = () => {
  const { state, signup, clearErrorMessage } = useContext(AuthContext);

  return (
    <SafeAreaView style={styles.container} forceInset={{ top: 'always' }}>
      <NavigationEvents onWillFocus={clearErrorMessage} />

      <Spacer>
        <Image
          source={require('../assets/img/logoMiTiempo.png')}
          containerStyle={styles.logo}
        />
      </Spacer>
    
```

Ilustración 82 - SignupScreen 1

```
<AuthForm
  errorMessage={state.errorMessage}
  buttonText="Sign up"
  icon={<FontAwesome name="google" style={styles.googleIcon} />}
  showGoogleButton={true}
  onSubmit={signup}
>

<View style={styles.linkContainer}>
  <NavLink
    text="Already have an account? Sign in instead!"
    routeName="Signin"
  />
</View>
</SafeAreaView>
);
};
```

Ilustración 83 - SignupScreen 2

SigninScreen.js

Pantalla que permite al usuario **iniciar sesión** en la aplicación. Similar en funcionamiento a SignupScreen, redirige al usuario a TaskHomeScreen si se supera la validación.

```

const SigninScreen = () => {
  const { state, signin, clearErrorMessage } = useContext(AuthContext);

  return (
    <SafeAreaView style={styles.container} forceInset={{ top: 'always' }}>
      <NavigationEvents onWillFocus={clearErrorMessage} />

      <Spacer>
        <Image
          source={require('../assets/img/logoMiTiempo.png')}
          containerStyle={styles.logo}
        />
      </Spacer>
  );
};

```

Ilustración 84 - SigninScreen 1

```

<AuthForm
  errorMessage={state.errorMessage}
  buttonText="Sign in"
  icon={<FontAwesome name="google" style={styles.googleIcon} />}
  showGoogleButton={false}
  onSubmit={signin}
/>

<View style={styles.linkContainer}>
  <NavLink
    text="Don't have an account? Sign up instead!"
    routeName="Signup"
  />
</View>
</SafeAreaView>
};


```

Ilustración 85 - SigninScreen 2

TaskHomeScreen.js

Pantalla principal de la aplicación una vez **hay una sesión iniciada**. En ella se muestra una barra de búsqueda en la parte superior, un selector de categorías y las tareas del usuario, desglosadas en dos listas: tareas para el día actual y en todas las tareas.

Desde aquí **el usuario puede realizar varias acciones**, como iniciar una tarea, marcarla como realizada o acceder al detalle de la misma, además de tener en la parte inferior de la pantalla la barra de navegación de la app.

```

    return (
      <SafeAreaView style={styles.container} forceInset={{ top: 'always' }}>
        <NavigationEvents onWillFocus={refreshData} />

        <SearchBar
          round
          lightTheme
          placeholder="Search"
          showCancel
          containerStyle={styles.searchBarContainer}

```

Ilustración 86 - TaskHomeScreen 1

```

          containerStyle={styles.searchBarContainer}
          inputStyle={styles.searchBarInputStyle}
          inputContainerStyle={styles.searchBarInputContainerStyle}
          onChangeText={setSearchTerm}
          value={searchTerm}
        >
      <View style={styles.horizontalList}>
        <HorizontalList
          data={state.categories}
          onSubmit={setCategory}
          style={styles.horizontalList}
        </HorizontalList>
      </View>

```

Ilustración 87 - TaskHomeScreen 2

```

      <ScrollView
        showsVerticalScrollIndicator={false}
        showsHorizontalScrollIndicator={false}
      >
        <Spacer>
          <Text h4>Today</Text>
          <TaskList data={state.todayTasks} searchTerm={searchTerm} />
        </Spacer>
        <Spacer>
          <Text h4>My Tasks</Text>
          <TaskList data={state.tasks} searchTerm={searchTerm} />
        </Spacer>
      </ScrollView>
    </SafeAreaView>
  );
};


```

Ilustración 88 - TaskHomeScreen 3

Se implementa una **barra de búsqueda**, la cual realiza el rastreo de los caracteres que introduce el usuario en la misma y los pasa como propiedad en la variable searchTerm al componente personalizado TaskList, que es el encargado de filtrar y mostrar las listas.

Por otro lado, tomando el contexto TaskContext, hace uso del **Hook useEffect()** para actualizar las tareas a mostrar según la categoría seleccionada. Realmente, cuando el usuario selecciona una tarea, se modifica la tarea seleccionada en el State del TaskContext,

desencadenando una petición al backend que tiene como respuesta el renderizado de las tareas filtradas.

```
useEffect(() => {
  listTasks({ category });
  listTodayTasks({ category });
  getCategories();
  handleRoutines();
}, [category]);
```

Ilustración 89 - useEffect()

También se hace uso del método `refreshData()` mediante `NavigationEvents`, que permite disparar dicho método cuando se tiene foco sobre la pantalla.

```
function reloadData() {
  listTasks({ category });
  listTodayTasks({ category });
}
```

Ilustración 90 - reloadData()

Esta pantalla dispone del método `handleRoutines()`, encargado de gestionar las rutinas (tareas repetitivas) mediante la librería Moment.

```
function handleRoutines() {
  const today = moment().format('dddd');
  const inAWeek = moment().add(6, 'days').format('dddd');

  state.tasks.forEach((task) => {
    if (task.day === moment().subtract(1, 'days').format('dddd')) {
      let taskId = task._id;
      let day = '';
      let isDone = false;

      switch (task.repetition) {
        case 'Every day':
          day = today;
          updateTask({ taskId, day, isDone });
          break;

        case 'Every week':
          day = inAWeek;
          updateTask({ taskId, day, isDone });
          break;

        default:
          break;
      }
    }
  });
}
```

Ilustración 91 - handleRoutines()

TaskDetailScreen.js

Esta pantalla muestra el **detalle de una tarea**. La id de la tarea es tomada por el navegador cuando el usuario pulsa sobre la tarea en la pantalla TaskHomeScreen, y es recibida en esta en la pantalla de detalle como Prop donde, haciendo uso del TaskContext, se toma la tarea deseada de la lista de tareas que se encuentra en el estado de ese contexto.

```
const TaskDetailScreen = ({ navigation }) => {
  const taskId = navigation.getParam('id');
```

Ilustración 92 - TaskDetailScreen 1

El usuario **puede modificar** los campos de la tarea y cambiar las opciones seleccionadas en el panel de opciones para actualizar la misma mediante el botón “Ok”, siempre que se supere la validación (deben rellenarse los campos “title” y “description”), el cual lanza una Action Function al contexto que ejecuta una petición en el backend para actualizar dicha tarea.

```
rightComponent={
  <Button
    buttonStyle={styles.headerButtonRight}
    titleStyle={styles.titleHeaderButtonRight}
    type="clear"
    title="Ok"
    onPress={() =>
      updateTask({
        taskId,
        title,
        description,
        day,
        duration,
        repetition,
        category,
        color,
        isPomodoro,
      })
    }
    disabled={!([title && description])}
  />
```

Ilustración 93 - TaskDetailScreen 2

Por otro lado, el usuario también **puede borrar** la tarea mediante el botón “Delete”, ejecutándose una action function en el contexto que desencadena una petición para dicho objetivo en el backend.

```
<Header>
    <leftComponent=>
        <Button>
            buttonStyle={styles.headerButtonLeft}
            titleStyle={styles.titleHeaderButtonLeft}
            type="clear"
            title="Delete"
            onPress={() => deleteTask({ taskId })}
        </Button>
    </leftComponent>
</Header>
```

Ilustración 94 - TaskDetailScreen 3

Para terminar, el usuario también **puede pulsar el botón “Play”**, redirigiendo al usuario a la pantalla TaskTimerScreen, donde se ejecutará la misma.

```
<MoveToBottom>
    <Spacer>
        <Button>
            buttonStyle={styles.solidButton}
            icon={
                <FontAwesome5
                    name="play"
                    size={20}
                    color={'white'}
                </FontAwesome5>
            }
            onPress={() =>
                navigation.navigate('TaskTimer', {
                    id: task._id,
                })
            }
        </Button>
    </Spacer>
</MoveToBottom>
```

Ilustración 95 - TaskDetailScreen 4

En esta pantalla se hacen uso de diversos métodos para el manejo de la información a mostrar en pantalla al usuario en el panel central de opciones, puesto que se reciben datos en distinto formato que el que se quiere mostrar al usuario. Por ejemplo, el color negro está almacenado en el backend como “#000000”, mientras que en el panel de opciones queremos mostrar “Negro”. Estos métodos se encargan de realizar este tipo de conversiones.

TaskTimerScreen.js

Pantalla que **ejecuta una tarea**:

- Recibe la **id** de una tarea, la cual es buscada en la lista de tareas del estado de TaskContext.
- Una vez se tiene la tarea, se hace uso del método **timerLoadHandler()**, método que **configura** el componente personalizado **Timer** según los valores de la tarea, teniendo en cuenta su duración y si es Pomodoro. En este último caso, gestiona los intervalos de tiempo.

- A medida que el temporizador avanza y según el caso, se muestra un **feedback** al usuario mediante mensajes y un panel de información.
- Al terminar un temporizador, se marca la tarea como **realizada**.

```
const timerLoadHandler = () => {
  if (isPomodoro) {
    // If pomodoro
    if (!isBreakTime) {
      // If is not breaktime
      setMessage('Work!');
      if (taskDuration - 25 ≤ 0) {
        // Work time less than break
        setTimerLoad(taskDuration);
        setIsLastLoad(true);
      }

      if (taskDuration - 25 > 0) {
        // Work time longer than break
        setTimerLoad(25);
        setTaskDuration(taskDuration - 25);
      }
    }
  }
}
```

Ilustración 96 - TaskTimerScreen 1

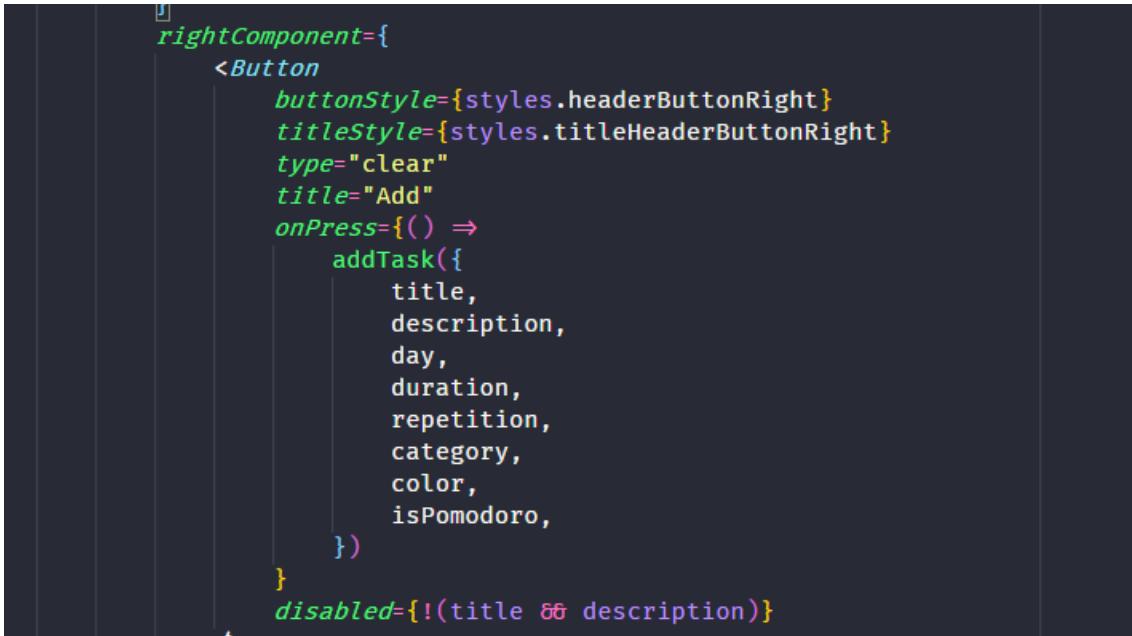
```
setPomodorosCounter(pomodorosCounter + 1);
} else {
  // If is breaktime
  if (pomodorosCounter ≥ 1 && pomodorosCounter ≤ 4) {
    // Short breaks handler
    setTimerLoad(4);
    setMessage(
      'Short break ' +
      pomodorosCounter +
      ' of 4!\nNew session starts in:'
    );
  } else if (pomodorosCounter === 5) {
    // Long breaks handler
    setTimerLoad(14);
    setMessage(`Long break!\nNew session starts in:`);
    setPomodorosCounter(0);
  }
} else {
  // If not pomodoro
  setTimerLoad(taskDuration);
  setIsLastLoad(true);
}
};
```

Ilustración 97 - TaskTimerScreen 2

TaskCreateScreen.js

Pantalla que permite al usuario introducir el título y la descripción de una **nueva tarea**, así como seleccionar una serie de opciones mediante el panel de opciones.

El usuario puede almacenar la nueva tarea mediante el botón “Add”, si se supera la validación del formulario (deben rellenarse los campos “title” y “description”), el cual lanza una Action Function al TaskContext para almacenar la nueva tarea en el servidor.



```
rightComponent={
  <Button
    buttonStyle={styles.headerButtonRight}
    titleStyle={styles.titleHeaderButtonRight}
    type="clear"
    title="Add"
    onPress={() =>
      addTask({
        title,
        description,
        day,
        duration,
        repetition,
        category,
        color,
        isPomodoro,
      })
    }
  disabled={! (title && description)}
}
```

Ilustración 98 - TaskCreateScreen 1

A su vez, puede cancelar la creación mediante el botón “Cancel”, retornando a la pantalla anterior.



```
<Button
  buttonStyle={styles.headerButtonLeft}
  titleStyle={styles.titleHeaderButtonLeft}
  type="clear"
  title="Cancel"
  onPress={() => navigation.navigate('homeFlow')}
/>
```

Ilustración 99 - TaskCreateScreen 2

TrackerScreen.js

La pantalla de **Tracker** permite al usuario **visualizar una serie de gráficas** mediante la lista horizontal que se sitúa en la parte baja de la misma, dando un feedback al usuario con diversa información sobre sus tareas.

Aunque se plantea la opción de utilizar librerías de terceros para realizar las gráficas a partir de los datos de la base de datos, se opta finalmente por implementar un servicio de MongoDB denominado **MongoDB Charts**. Esta herramienta permite la creación de gráficos a partir de los datos almacenados en la base de datos para su posterior uso en una aplicación, ya sea mediante el SDK propio de MongoDB Charts, o con la generación de código incrustado (*embedded code*) mediante un *iframe*..

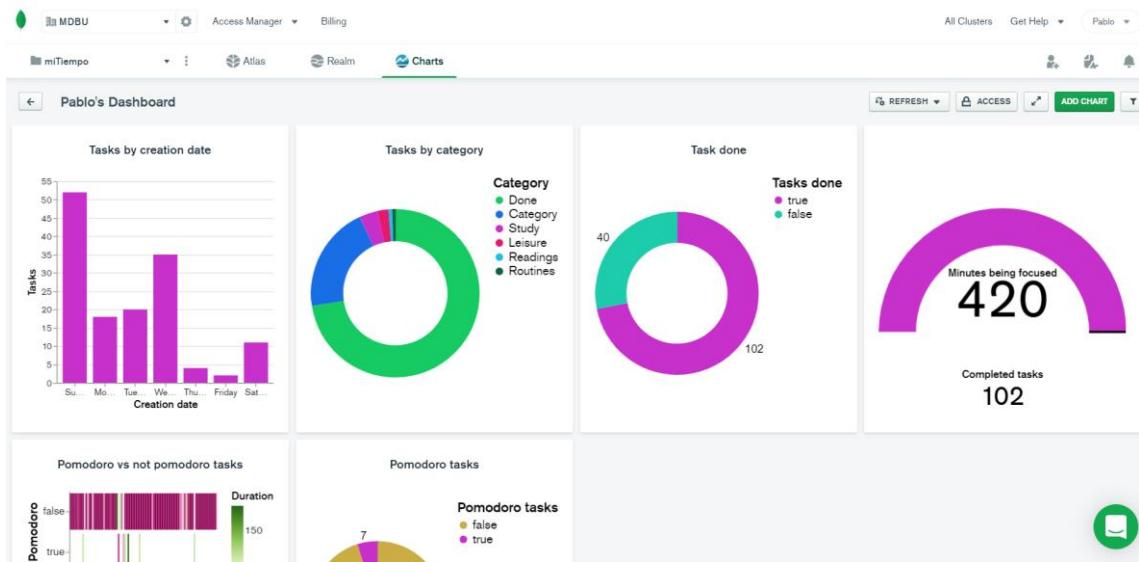


Ilustración 100 - MongoDB Charts 1

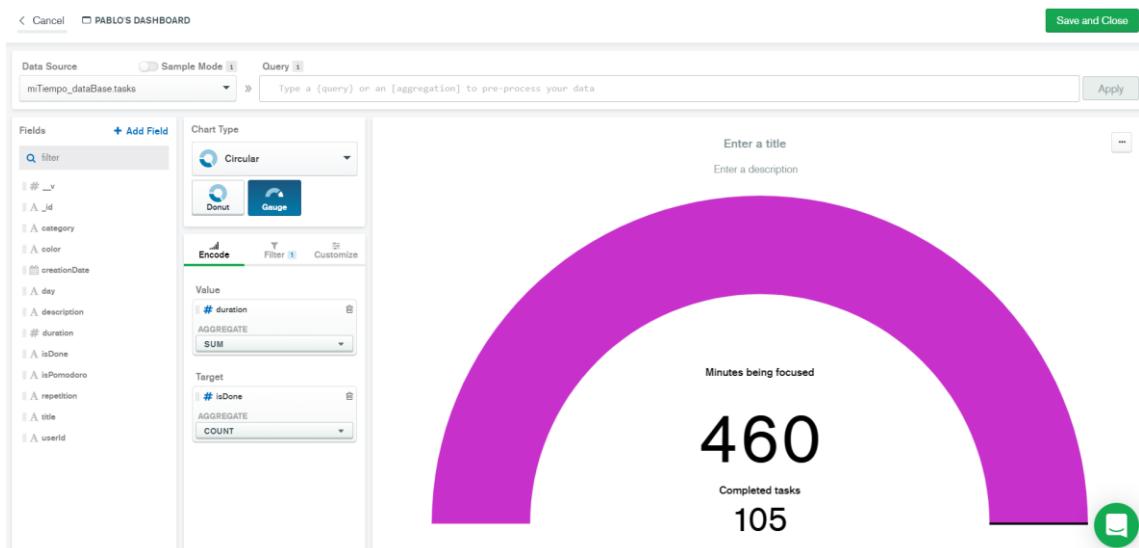


Ilustración 101 - MongoDB Charts 2

EMBED CODE:

```
<iframe style="background: #FFFFFF; border: none; border-radius: 2px; box-shadow: 0 2px 10px 0 rgba(70, 76, 79, .2); width="640" height="480" src="https://charts.mongodb.com/charts-mitiempo-trqxx/embed/charts?id=88615f15-7209-4048-a1f0-e8944d467c26&theme=light"></iframe>
```

Ilustración 102 - MongoDB Charts 3

Una vez creadas las gráficas, se toman sus direcciones para ser utilizadas en la aplicación, aplicándoles un filtro con la id del usuario del que extraer los datos:

```
const userIdFilter = `filter={"userId":%20['$oid':%20`${state._id}`]}`;  
const CHART_OPTIONS = {  
  TASKS_BY_CATEGORY: `https://charts.mongodb.com/charts-mitiempo-trqxx/embed/charts?id=93ce989d-3c06-4c03-ab13-f043470138f58${userIdFilter}&theme=light`,  
  TASKS_BY_CREATION_DATE: `https://charts.mongodb.com/charts-mitiempo-trqxx/embed/charts?id=f8bb6578-7add-431d-975c-2dcfb668ea95${userIdFilter}&theme=light`,  
  TASKS_DONE: `https://charts.mongodb.com/charts-mitiempo-trqxx/embed/charts?id=cacf875-a36f-4e78-8355-71ca516d7c6d6${userIdFilter}&theme=light`,  
  CONCENTRATED_MINUTES: `https://charts.mongodb.com/charts-mitiempo-trqxx/embed/charts?id=88615f15-7209-4048-af0-e8944d467c266${userIdFilter}&theme=light`,  
  POMODORO_VS_NOTPOMODORO: `https://charts.mongodb.com/charts-mitiempo-trqxx/embed/charts?id=621b0122-e7b2-445c-af4d-99ae61be51ed${userIdFilter}&theme=light`,  
  POMODORO_TASKS: `https://charts.mongodb.com/charts-mitiempo-trqxx/embed/charts?id=c51e6898-7d28-449c-bb6b-1f189fcfdcbf6${userIdFilter}&theme=light`,  
};
```

Ilustración 103 - TrackerScreen 1

Las gráficas **se generan de dos formas** distintas dentro de TrackerScreen, teniendo que hacer uso de **código específico** para la parte web y para la parte móvil, no pudiendo unificar el código por incompatibilidades en React Native para procesar la información en ambas capas:

- En la parte **web** se opta por hacer uso de un **iframe**, un elemento HTML que permite incrustar otra página web, en este caso, se incrusta la url de la gráfica seleccionada.

```
if (Platform.OS === 'web') {  
  return (  
    <SafeAreaView  
      style={styles.container}  
      forceInset={{ top: 'always' }}  
    >  
    <View style={styles.container2}>  
      <Text style={styles.header} h4>  
        My Time  
      </Text>  
      <SectionContainer>  
        <View style={styles.webChart}>  
          <iframe  
            src={selectedChart}  
            height={400}  
            width={'100%' }  
            height={500}  
            frameBorder="0"  
          />  
        </View>
```

Ilustración 104 - TrackerScreen 2

- En la parte **móvil** se afronta con el mismo planteamiento que en la parte web, pero se hace uso de **WebView**. Puesto que el elemento iframe no es compatible con la capa móvil de React Native, WebView es un componente de compatibilidad de React Native que permite incrustar iframe en plataformas móviles.

```
<WebView  
    source={/  
        html: `<iframe width="100%" height="100%" src=${selectedChart} frameborder="0" ></iframe>`,  
    }  
    style={styles.mobileChart}  
/>
```

Ilustración 105 - TrackerScreen 3

AccountScreen.js

Pantalla que permite al usuario **ver datos de su cuenta de usuario**, como el email o el usuario, así como editar su cuenta o cerrar sesión.

Toma la información del usuario a partir del UserContext y redirige al usuario a la pantalla de edición mediante el botón “Edit”.

```
<MoveToBottom>  
  <Spacer>  
    <Button  
        buttonStyle={styles.solidButton}  
        title="Edit"  
        onPress={() => navigation.navigate('EditAccount')}  
    />
```

Ilustración 106 - AccountScreen 1

Por otro lado, también hace uso del AuthContext o contexto de autorización para realizar el cierre de sesión, que consiste en el envío de una Action Function a ese contexto que genera la eliminación del token del usuario, provocando el cierre instantáneo de la aplicación y el retorno a la página de alta.

```
<Button  
    buttonStyle={styles.outlineButton}  
    titleStyle={styles.titleColorOutlineButton}  
    type="outline"  
    title="Sign out"  
    onPress={signout}  
/>  
</Spacer>
```

Ilustración 107 - AccountScreen 2

EditAccountScreen.js

Pantalla que permite al usuario **editar los datos de su cuenta de usuario** (email y nombre de usuario), así como realizar un cambio de contraseña.

El usuario puede modificar los campos email y nombre de usuario. A su vez, si el usuario decide crear una nueva contraseña, esta debe ser igual a la contraseña de confirmación, si no, no se habilitará el botón “Update”, mostrándose un mensaje al usuario en pantalla.

```
<MoveToBottom>
  <Spacer>
    <Button
      buttonStyle={styles.solidButton}
      title="Update"
      disabled={
        !(  

          newPassword === newPasswordConfirmation ||  

          newPassword.length > 0) ||  

          (newPassword === '' ||  

          newPasswordConfirmation === '')
```

Ilustración 108 – EditAccountScreen 1

A su vez, si el email introducido está registrado en la base de datos, se muestra un mensaje al usuario.

Todas estas acciones se hacen mediante el uso del UserContext.

```
  onPress={() =>
    updateUserInfo({
      email,
      name,
      city,
      newPassword,
    })
  }
<Button
  buttonStyle={styles.outlineButton}
  titleStyle={styles.titleColorOutlineButton}
  type="outline"
  title="Cancel"
  onPress={() => navigation.navigate('Account')}
/>
```

Ilustración 109 - EdicAccountScreen 2

Components

En este apartado se detallan los **componentes utilizados en el frontend** de miTiempo. Estos componentes, a diferencia de los componentes de las pantallas, son componentes **reutilizables** a lo largo del frontend, pudiendo ser implementados en una o en varias pantallas de la aplicación.

AuthForm.js

Componente que implementa un **formulario con los campos email y Password**. Recibe como propiedades el texto del botón de submit, un ícono para el botón secundario (en esta app destinado al alta de Google), un mensaje de error, una función callback para enviar una respuesta al componente padre y un boolean que indica si se debe mostrar el botón de Google o no.

Está diseñado de tal forma que con un único componente, podemos reutilizar código para implementar ambos formularios, el de alta y el de login.

```
const AuthForm = ({  
  navigation,  
  buttonText,  
  icon,  
  showGoogleButton,  
  errorMessage,  
  onSubmit,  
}) => {  
  const [email, setEmail] = useState('');  
  const [password, setPassword] = useState('');
```

Ilustración 110 – AuthForm

HorizontalList.js

Componente que implementa una **lista horizontal**. Recibe como propiedades la información a mostrar en la lista, así como un callback para enviar una respuesta al componente padre con la opción seleccionada.

Es un componente **usado ampliamente** a lo largo de la aplicación y que es diseñado ante la inexistencia de un componente prediseñado que cumpliera la funcionalidad requerida. Está presente en el selector de categorías de TaskHomeScreen, en el selector de gráficas de TrackerScreen, y en el selector propio de cada opción de TaskDetail y TaskCreateScreen.

```
return (  
  <FlatList  
    contentContainerStyle={styles.flatlistContainer}  
    horizontal  
    showsVerticalScrollIndicator={false}  
    showsHorizontalScrollIndicator={false}  
    data={data}  
    keyExtractor={(element) => element._id}  
    renderItem={({ item }) => {
```

Ilustración 111 - HorizontalList 1

```

        return (
          <ListItem containerStyle={styles.container}>
            <ListItem.Content>
              <ListItem.Title
                titleStyle={styles.itemContainer}
              >
                <Button
                  buttonStyle={handleButtonStyles(
                    item
                  )}
                  titleStyle={handleButtonTitlesStyles(
                    item
                  )}
                  type="outline"
                  title={item}
                  onPress={() => {
                    onSubmit(item);
                  }}
                />
              </ListItem.Title>
            </ListItem.Content>
          </ListItem>
        );
      }
    );
  }
}

export default HorizontalList;

```

Ilustración 112 - HorizontalList 2

MoveToBottom.js

Componente que **sitúa el componente hijo que recibe en el footer** de la pantalla. Es usado a lo largo de la aplicación para colocar los botones siempre al mismo nivel y para colocar cualquier otro elemento en la parte baja de la pantalla.

```

const MoveToBottom = ({ children }) => {
  return <View style={styles.container}>{children}</View>;
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'flex-end',
    marginBottom: 10,
  },
});

```

Ilustración 113 – MoveToBottom

NavLink.js

Componente que implementa un **botón textual que enlaza a otra pantalla**. Para ello, recibe como propiedades o parámetros el navegador de la aplicación, el texto a mostrar en el botón y la ruta de destino del mismo.

Es usado en las pantallas de alta y login, en la opción que se da al usuario debajo de los botones para navegar entre ambas pantallas.

```

const NavLink = ({ navigation, text, routeName }) => {
  return (
    <TouchableOpacity onPress={() => navigation.navigate({ routeName })}>
      <Spacer>
        <Text style={styles.link}>{text}</Text>
      </Spacer>
    </TouchableOpacity>
  );
};

```

Ilustración 114 – NavLink

SectionContainer.js

Componente que añade un **contenedor con fondo blanco y margen al componente hijo** que recibe. Usado ampliamente en las pantallas de la aplicación para hacer más vistosas y atractivas las secciones de las mismas.

```

const SectionContainer = ({ children }) => {
  return <View style={styles.container}>{children}</View>;
};

const styles = StyleSheet.create({
  container: {
    backgroundColor: 'white',
    margin: 10,
    borderRadius: 4,
  },
});

```

Ilustración 115 – SectionContainer

Spacer.js

Componente que **añade margen al componente hijo** que recibe como parámetro. Es utilizado a lo largo de todo el frontend no sólo para aplicar márgenes, sino como **separador** entre componentes.

```

const Spacer = ({ children }) => {
  return <View style={styles.spacer}>{children}</View>;
};

const styles = StyleSheet.create({
  spacer: {
    margin: 15,
  },
});

```

Ilustración 116 - Spacer

TaskList.js

Componente que **implementa una lista**, recibiendo como parámetros las tareas a listar y un posible término de búsqueda para filtrar las mismas por el mismo mediante el método `filterTasks()`.

Este componente hace uso de otro componente, **TaskItem**, encargado de gestionar cada uno de los elementos de la lista.

```
const TaskList = ([{ data, searchTerm }]) => {
  let tasks = filterTasks(data, searchTerm);

  return (
    <FlatList
      showsVerticalScrollIndicator={false}
      showsHorizontalScrollIndicator={false}
      style={styles.listContainer}
      data={tasks}
      keyExtractor={( task ) => task._id}
      renderItem={({ item }) => {
        return <TaskItem item={item} />;
      }}
    >
  );
};
```

Ilustración 117 - TaskList 1

```
function filterTasks(data, searchTerm) {
  return searchTerm === ''
    ? data
    : data.filter((task) =>
      task.title.toLowerCase().includes(searchTerm.toLowerCase())
    );
}
```

Ilustración 118 - TaskList 2

TaskItem.js

Componente que **implementa un ítem de la lista**. Ligado al componente TaskList, recibe desde este la tarea y el navegador principal de la aplicación.

```
const TaskItem = ({ item, navigation }) => {
  const { updateTask } = useContext(TaskContext);
  const [isTaskDone, setIsTaskDone] = useState(item.isDone);
```

Ilustración 119 - TaskItem 1

Implementa varias **funcionalidades**:

- Botón para ejecutar la tarea.

```
<TouchableOpacity  
    onPress={() =>  
        navigation.navigate('TaskTimer', {  
            id: item._id,  
        })  
    }>  
    <FontAwesome5 name="play" size={20} color={item.color} />
```

Ilustración 120 - TaskItem 2

- Switch para marcar la tarea como realizada.

```
<TouchableOpacity  
    onPress={() =>  
        markTaskDone(item._id, item.isDone, item.category)  
    }>  
    {isTaskDone ? (  
        <FontAwesome5  
            name="dot-circle"  
            size={20}  
            color={item.color}  
        />
```

Ilustración 121 - TaskItem 3

- Botón para ir al detalle de la tarea.

```
<TouchableOpacity  
    onPress={() =>  
        navigation.navigate('TaskDetail', {  
            id: item._id,  
        })
```

Ilustración 122 - TaskItem 4

Timer.js

Componente ligado a la pantalla TaskTimerScreen. Implementa un **temporizador** a partir de los parámetros que recibe: la carga de tiempo y un callback que indica al componente padre que actualizar el temporizador.

Durante el diseño de este componente se pretende que se encargue simplemente de una cuenta atrás de X segundos, **delegando la lógica** de los intervalos Pomodoro y mensajes de información a TaskTimerScreen.

```

const Timer = ({ timerLoad, refreshTimer }) => {
  const [minutes, setMinutes] = useState(timerLoad);
  const [seconds, setSeconds] = useState(0);

  const timerMinutes = minutes < 10 ? `0${minutes}` : minutes;
  const timerSeconds = seconds < 10 ? `0${seconds}` : seconds;

```

Ilustración 123 - Timer 1

```

const timer = () => {
  let interval = setInterval(() => {
    clearInterval(interval);
    if (seconds === 0) {
      if (minutes !== 0) {
        setSeconds(59);
        setMinutes(minutes - 1);
      }
    } else {
      setSeconds(seconds - 1);
    }
    if (minutes === 0 && seconds === 0) {
      refreshTimer();
    }
  }, 5);
};

```

Ilustración 124 - Timer 2

Landing page

Esta capa de la aplicación consiste en una **web desarrollada con Bootstrap 4 y alojada en Netlify**. Su función es proveer al usuario de una **forma de acceso** hacia el resto de frontends (web-app, aplicación Androir en Play Store o aplicación iOS en App Store), además de servir de **gancho** para nuevos clientes. Por otro lado, también abarca la página de **confirmación de envío de credenciales** cuando el usuario decide darse de alta mediante Google.

Conceptos esenciales

Aunque esta capa de miTiempo es sencilla, conviene detallar los siguientes **conceptos**:

- **Bootstrap 4**: Framework de HTML, CSS y JavaScript que permite la creación ágil de páginas web *responsive* o diseño adaptativo, es decir, adaptadas a las pantallas de cualquier dispositivo. Se basa en elementos y propiedades predefinidas, implementadas directamente a través del HTML, fácilmente personalizables a través de CSS.
- **Animate.css**: Librería de animaciones CSS. Permite enfatizar elementos web de forma muy llamativa para el usuario.
- **AOS**: Similar a Animate.css, AOS se centra en animaciones que se *disparan* con el uso del scroll dentro de la web en la que se implementa.

Funcionamiento

Para la creación de la Landing Page de este proyecto, se hace uso de la siguiente **plantilla** de Bootstrap de licencia MIT para agilizar su desarrollo:

<https://startbootstrap.com/theme/landing-page>

Sobre dicha plantilla se realiza una **personalización** de cada una de las secciones para amoldarla al estilo de miTiempo y a su objetivo como página de bienvenida. Para los Scripts de las distintas librerías, se opta por su implementación mediante **CDN** (Content Delivery Network) para agilizar el proceso.

Respecto a su **funcionamiento**, se trata de dos páginas web ligadas a una plantilla de global y a varios scripts de JavaScript que se reciben vía CDN:

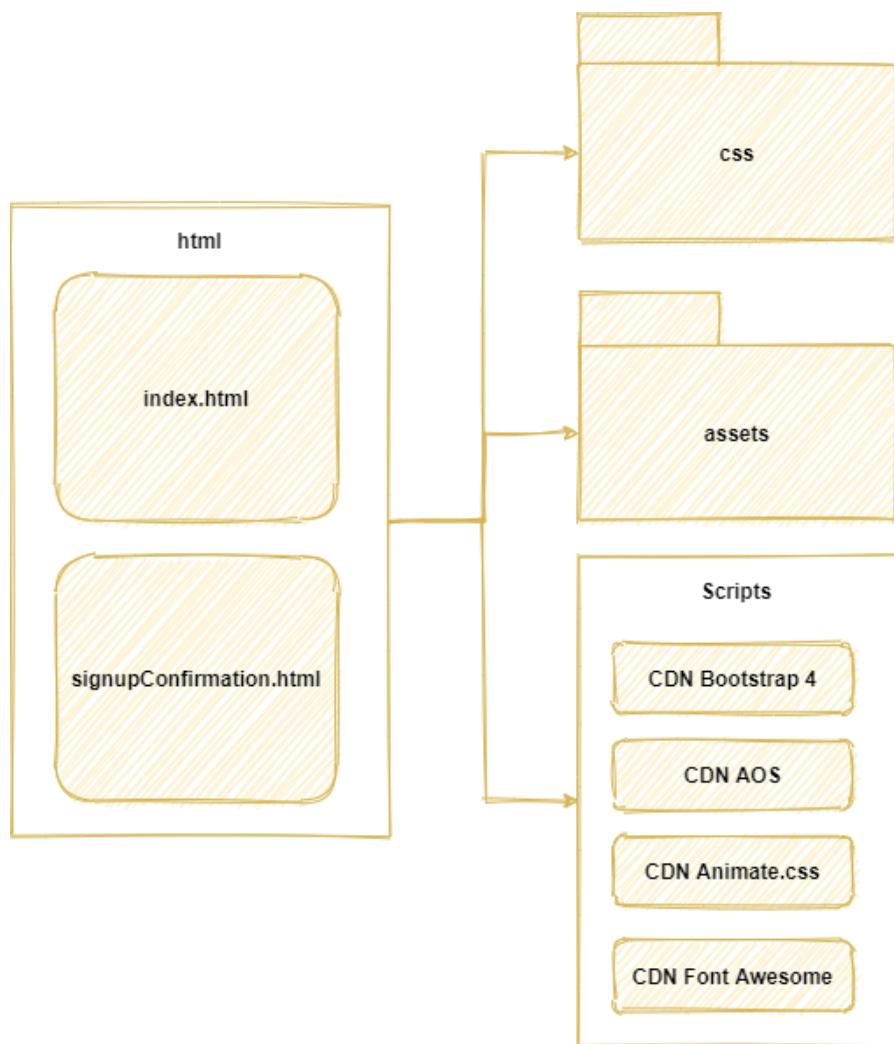


Ilustración 125 - Funcionamiento Landing Page

Secciones

Para reflejar la estructura anterior, a continuación se realiza una visión general de cada una de las secciones de la web.

Cabecera

La cabecera es el **punto de entrada** al universo de miTiempo, por lo que se opta por un diseño sencillo a partir del logo de miTiempo con una animación giratoria, además de su eslogan y una llamada a la acción mediante un *chevron* para indicar al usuario que haga scroll vertical.

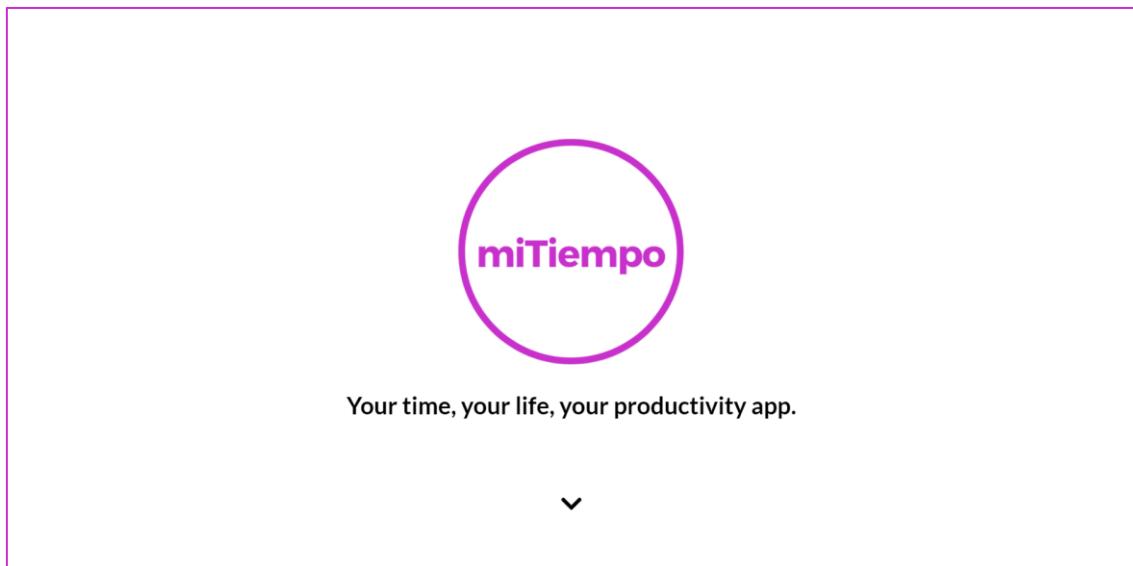


Ilustración 126 - Cabecera Landing Page

Características

Mediante animaciones de scroll, **se muestran de forma visual** las características de la aplicación. Esta sección, Showcase y Testimonios tienen el objetivo de “enganchar” al nuevo usuario, para que comience a utilizar miTiempo.

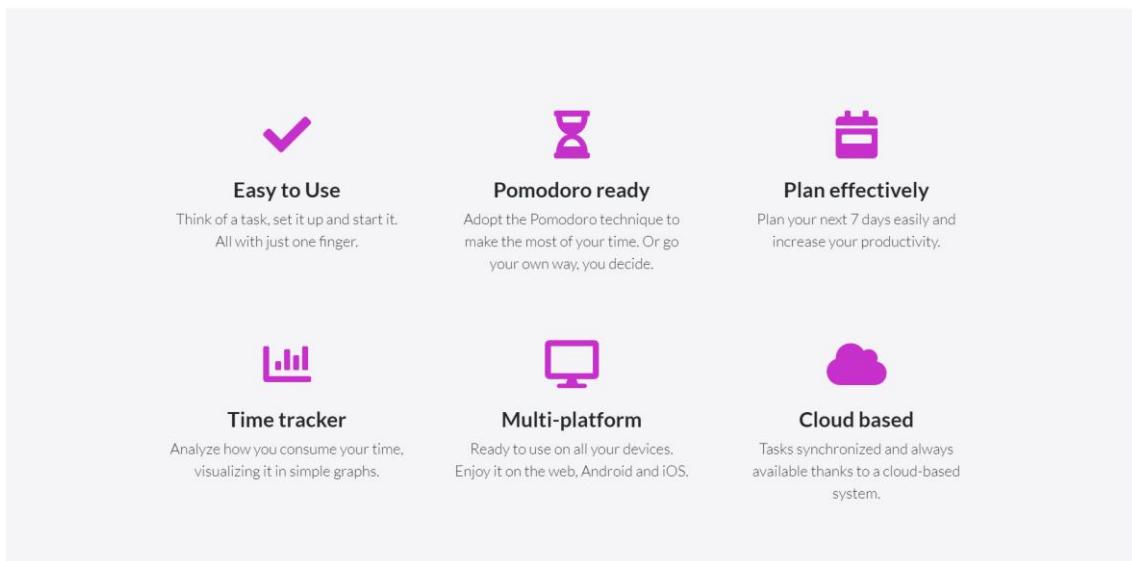


Ilustración 127 - Características Landing Page

Showcase

En el escaparate o showcase, se muestran nuevos **titulares para la aplicación**, así como una visión de la misma en un terminal Android, haciendo uso de animaciones de scroll.

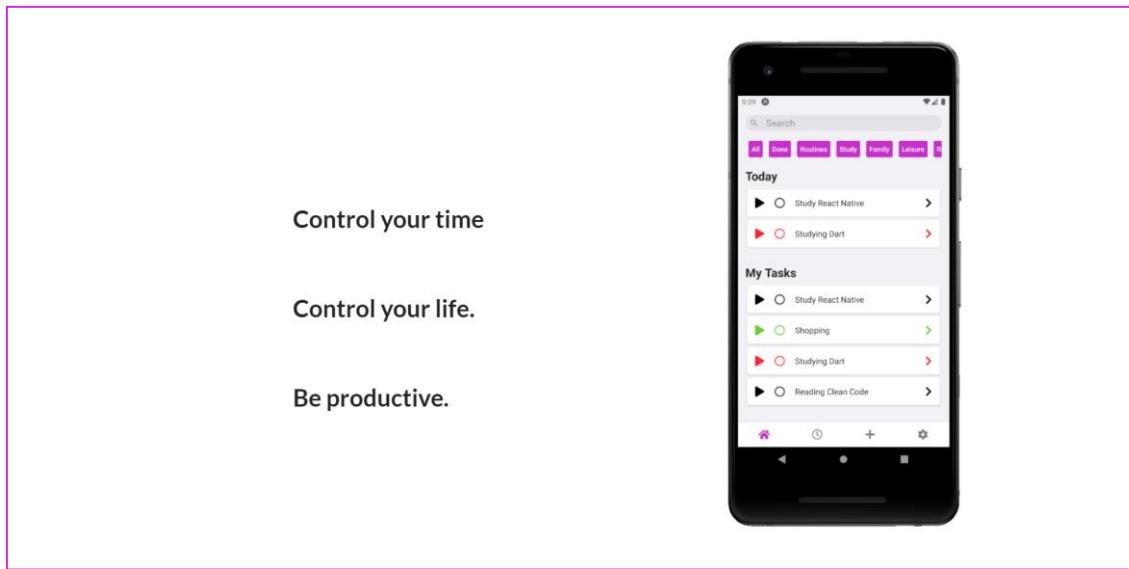


Ilustración 128 - Showcase Landing Page

Testimonios

A lo largo de esta sección se muestran una serie de diversos testimonios (ficticios) para atraer a un usuario potencial.

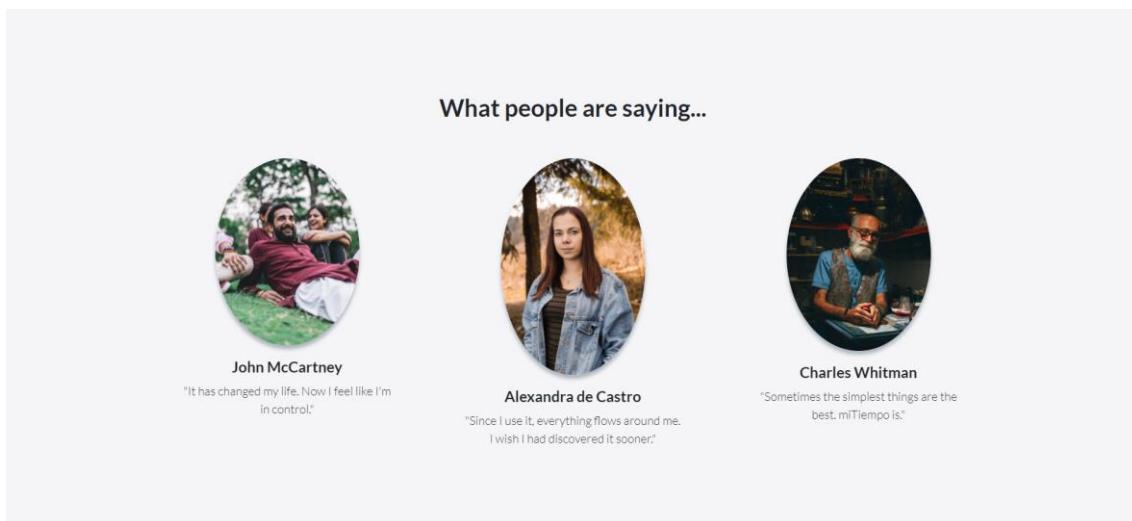


Ilustración 129 - Testimonios Landing Page

Llamada a la acción

Una vez mostradas las características de la aplicación, **se redirige al usuario hacia los diversos frontends** de la aplicación, pudiendo elegir entre la app-web, la app Android a descargar desde el Play Store, o la app iOS a descargar desde la App Store.

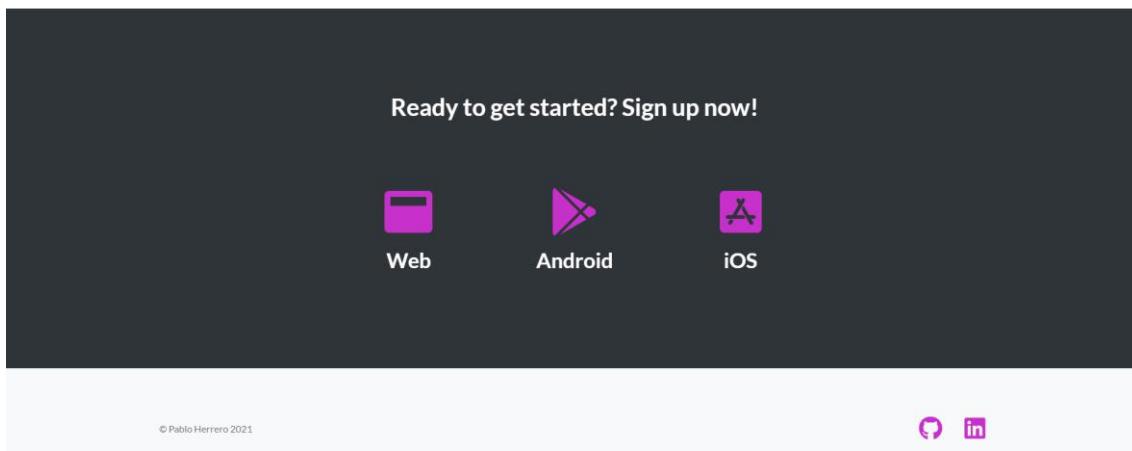


Ilustración 130 - Llamada a la acción Landing Page

Confirmación de alta

A esta sección, que es una página HTML aparte, es redirigido el usuario por parte del backend cuando el **proceso de alta con Google tiene éxito y se envía el email** con las nuevas credenciales al email del usuario.

Este es el **email** que recibe el cliente:

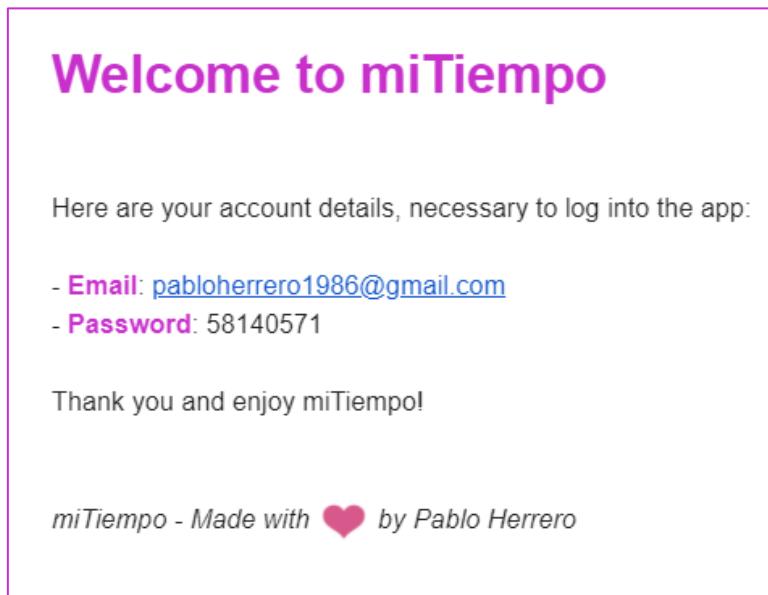


Ilustración 131 - Email de confirmación de alta

Esta es la **página de confirmación**:

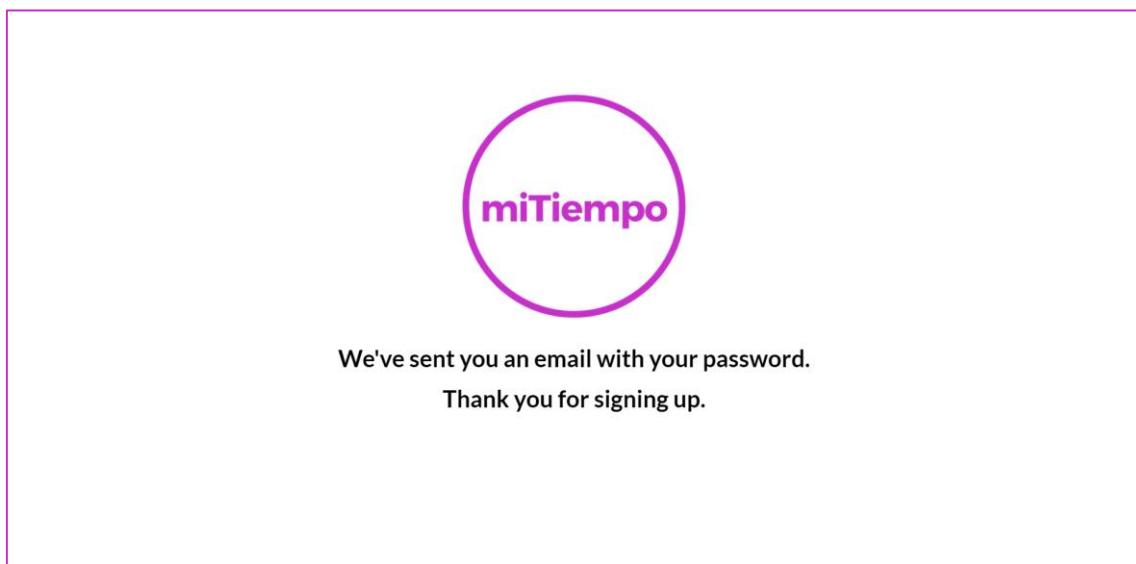


Ilustración 132 - Confirmación de alta Landing Page

Problemas encontrados

En el siguiente apartado se hace un repaso por los **diversos problemas** encontrados en el desarrollo de miTiempo.

Sencillez de React

React tiene una **gran virtud que es a la vez su gran defecto: la sencillez**. Pese a su potencial, en inicio es simplemente una librería para realizar interfaces de usuario por lo que el desarrollador debe buscar e implementar librerías de terceros para añadir nuevas funcionalidades.

Esta característica de React Native es lo que la transforma en una **librería y no en un Framework** completo, como Angular.

La investigación necesaria, la tasa elevada de manejo de dependencias, los problemas e incompatibilidades originados por las librerías de terceros, han sido un problema con el que se lucha a lo largo de toda la creación de este proyecto.

Manejo de State, Context, Provider y Reducer en React

Uno de los grandes retos de este proyecto y, por ende, del aprendizaje de React es la **comprensión de conceptos** como State, Context, Provider y Reducer explicados a lo largo de esta documentación, e indispensables para el correcto manejo y flujo de la información a lo largo de la aplicación.

Incompatibilidades, inconsistencias entre plataformas y código específico en React Native

En más de una ocasión a lo largo del desarrollo de esta aplicación, servidor se ha encontrado con la imposibilidad de utilizar un componente por no estar disponible dentro de React Native para las tres plataformas, web, Android e iOS. Cuando hablamos de un framework multiplataforma, este tipo de problemas es lo último que uno desea encontrar. Genera varios y muy molestos problemas en el desarrollo que tienen como consecuencias retrasos en la producción.

A modo de ejemplo de esta situación, se plantea el **problema del Picker**:

Al realizar el diseño de las pantallas de miTiempo, para la pantalla de creación de tareas se pensó en utilizar un picker o selector para que el usuario pudiera realizar la elección de la configuración de cada una de las opciones de la tarea. Al intentar implementar el picker en desarrollo, no se encuentra un componente que funcione en todas las plataformas, o se

encuentra, pero genera inconsistencias y errores en la aplicación. Se plantea utilizar un picker para la parte web y otro para la parte móvil, pero se descarta por inconsistencia en el diseño.

Finalmente la solución pasa por implementar un panel de opciones personalizado:

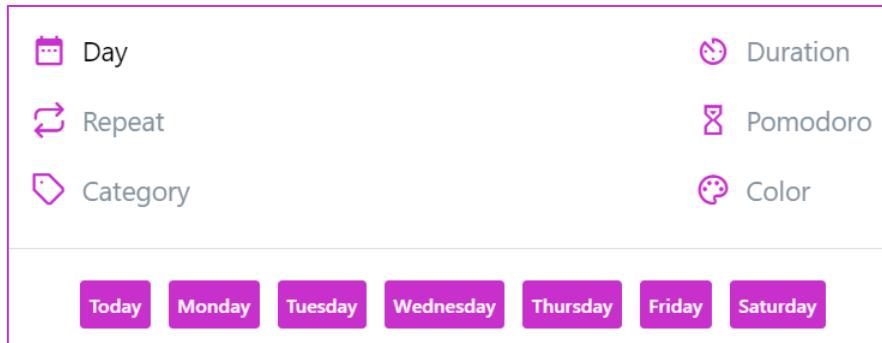


Ilustración 133 - Panel de opciones personalizado

A su vez, también se han sufrido inconsistencias entre componentes nativos de la propia librería, siendo el caso más claro el de la **altura del header en Android**. Al implementar una header con botones a sus laterales como en las pantallas TaskCreateScreen y TaskDetailScreen, se genera un bug que hace que en las plataformas web e iOS la altura sea correcta, pero en Android dicha altura sea desproporcionada. La solución pasó por la investigación, encontrando luz en diversos foros como <https://github.com/react-navigation/react-navigation/issues/4336#issuecomment-392185562>.

Otra inconsistencia digna de relatar es la que se sufre con el **componente SafeAreaView**. Este componente se aplica en las pantallas de la aplicación para instaurar un área segura en la pantalla, libre de, por ejemplo, los notch de los iPhone y otros dispositivos. Es decir, especifica al componente el área donde puede renderizar. Al desarrollar miTiempo e implementar este componente se recibe este error:

A screenshot of a code editor showing a red error message. The message reads: "Error: No safe area insets value available. Make sure you are rendering `<SafeAreaProvider>` at the top of your app." Below the message is a stack trace:

```
useSafeAreaInsets
C:/Users/pablo/Documents/DAM/Segundo/PROY/miTiempo/code/miTiempo-front/SafeAreaContext.tsx:104
101 | export function useSafeAreaInsets(): EdgeInsets {
102 |   const safeArea = React.useContext(SafeAreaInsetsContext);
103 |   if (safeArea === null) {
> 104 |     throw new Error(
105 |       'No safe area insets value available. Make sure you are rendering `<SafeAreaProvider>` at the top of your app.'
106 |     );
107 |   }
View compiled
```

Ilustración 134 - Error "No safe area"

La solución, en este caso, la ofrece el framework usado para manejar React Native: Expo. A través de este se pudo instalar react-native-safe-area-context, un componente que solventa los problemas originales del SafeArea original.

Para terminar con este punto trataremos el **código específico**. React Native permite al desarrollador escribir código que vaya dedicado exclusivamente a una o a varias plataformas. En miTiempo es habitual el uso de código específico para la aplicación de estilos ya que el diseño se antoja inconsistente o requiere de retoques en la front web respecto al móvil. Este es, por ejemplo, el código de uno de los botones de TaskDetailScreen, donde se aplica el código específico:

```
solidButton: {
  ...Platform.select({
    android: {
      backgroundColor: '#C830CC',
      marginBottom: 10,
    },
    ios: {
      backgroundColor: '#C830CC',
      marginBottom: 10,
    },
    default: {
      alignSelf: 'center',
      backgroundColor: '#C830CC',
      marginBottom: 10,
      width: '25%',
      padding: 10,
    },
  }),
},
```

Ilustración 135 - Detalle código específico

Manejo CORS

El Intercambio de Recursos de Origen Cruzado, o CORS, es uno de los grandes problemas con los que el creador de este proyecto se encuentra en su desarrollo. El **CORS** es un mecanismo de cabeceras HTTP adicionales para permitir acceder a recursos adicionales de un servidor en un origen distinto al que pertenece.

Cuando desde el frontend de la web-app se intenta hacer una **petición al backend sin las CORS configuradas**, se recibe el siguiente mensaje de error:

```
* Access to XMLHttpRequest at 'https://mitiempo-back.herokuapp.com/signin' from origin 'http://localhost:19006' has been blocked by CORS policy: Request header field content-type is not allowed by Access-Control-Allow-Headers in preflight response.
```

Ilustración 136 - Error CORS

La **solución**, inspirada por la respuesta dada en <https://stackoverflow.com/questions/57873186/solved-cannot-get-cors-to-work-no-matter-what-i-try>, fue la implementación en el backend de la librería “cors”, la cual permite realizar una configuración inicial de las mismas para permitir este tipo de peticiones desde cualquier destino.

```

app.use(
  cors({
    allowedHeaders: ['authorization', 'Content-Type'],
    exposedHeaders: ['authorization'],
    origin: '*',
    methods: 'GET,HEAD,PUT,PATCH,POST,DELETE',
    preflightContinue: false,
  })
);

```

Ilustración 137 - Solución CORS

Actualización dinámica de campos en MongoDB

Se tienen varias dificultades cuando se intenta **actualizar un solo campo** de un documento de MongoDB mediante Mongoose en el backend, a partir de una petición realizada desde el frontend.

Tras realizar una investigación, tanto en StackOverflow como en MongoDB, e inspirado por la respuesta dada en <https://stackoverflow.com/questions/30602057/how-to-update-some-but-not-all-fields-in-mongoose>, se decide implementar una **Middleware** en el backend para las rutas que desean actualizar los campos de cierto documento. Dicha Middleware, comprueba los campos recibidos sin valor, eliminándolos y retornando sólo los que deben actualizarse, que son los que son procesados finalmente.

```

module.exports = (req, res, next) => {
  ...
  let fields = getFieldsFromBody(req.body);
  fieldsToUpdate = deleteFieldsWithoutValue(fields);
  req.fieldsToUpdate = fieldsToUpdate;
  next();
};

```

Ilustración 138 - Campos dinámicos

Dentro ya de la petición, esos campos se procesan mediante el operador **\$set**, que permite la actualización en un documento de los campos especificados.

```

const user = await User.findByIdAndUpdate(
  userId,
  { $set: { ... fieldsToUpdate } },
  {
    runValidators: true,
    new: true,
  }
);

```

Ilustración 139 - Campos dinámicos 2

Manejo de peticiones - Paso de parámetros en una petición GET

Quizás por **no estar habituado al entorno web**, al no haber trabajado en él a lo largo del ciclo, una **dificultad añadida fue el manejo de peticiones**. En concreto, el **paso de parámetros** desde el frontend al backend en una petición Get, requiere en primera instancia de investigación del desarrollador, además de ensayo y error hasta conseguir la forma correcta de hacerlo.

Para ilustrar el problema, se muestra la Action Function listTasks(), en el frontend, y la ruta a la que realiza la petición en el backend, /listTasks.

```
const listTasks = (dispatch) => async ({ category }) => {
  try {
    const response = await miTiempoApi.get(`/listTasks/${category}`);
    dispatch({ type: 'listTasks', payload: response.data });
  } catch (error) {
    dispatch({
      type: 'add_error',
      payload: 'Something went wrong retrieving tasks data.',
    });
  }
};
```

Ilustración 140 - Get 1

```
router.get('/listTasks/:categoryFilter', requireAuth, async (req, res) => {
  try {
    if (req.params.categoryFilter === 'All') {
      const tasks = await Task.find({ userId: req.user._id })
      res.json(tasks);
    }
  } catch (error) {
    res.status(500).json(error);
  }
});
```

Ilustración 141 - Get 2

Implementación de alta y login con Google – OAuth2

La **idea original** para el sistema de alta y logado de miTiempo es ofrecer al usuario dos opciones diferenciadas:

- Alta/login mediante email y contraseña.
- Alta/login mediante cuenta de Google.

La capa de Google permite introducir al usuario su cuenta de Google y contraseña, siendo logado automáticamente al sistema de miTiempo. El escaso tiempo y la falta de conocimientos para abordar los problemas surgidos con claridad, hacen al desarrollador replantear la situación y abordar la funcionalidad requerida de otra forma.

Surge un **gran obstáculo** al intentar implementar la autenticación con Google:

No se encuentra la forma de enviar el token al front desde el backend. Se consigue que el backend remita al usuario al autenticador de Google a través de Passport, y que, una vez

el usuario inicia sesión en el sistema de Google, se almacene al nuevo usuario y se genere un token de autenticación para el mismo. Pero no se consigue devolver este token al frontend, por lo que el usuario sigue sin poder logarse.

Aunque se intentan distintos planteamientos para intentar mantener la funcionalidad tal cual se planteó en principio, finalmente y como ya se ha mostrado a lo largo de la documentación, **se decide usar Google como forma de validar** el email del usuario a través del autenticador de Google, enviando, tras esta validación, un email al usuario con sus credenciales.

Despliegue

En este apartado hablaremos de la última etapa abordada en la implementación del sistema de miTiempo: el despliegue. El **despliegue o deploy** es la publicación de la aplicación para disfrute del usuario final. Dado que este proyecto es afín a las tecnologías **Cloud**, se intenta hacer el despliegue haciendo uso de **servicios en la nube**, de forma que la aplicación esté disponible para cualquier usuario.

Backend

Para el deploy del backend se decide usar **Heroku**, una plataforma que permite alojar aplicaciones basadas en Node.js, teniendo integración con GitHub. Esta integración implica que el despliegue de Heroku se enlaza con el repositorio del backend en GitHub, por lo que con cada nuevo *commit* al repositorio, el despliegue en Heroku se actualiza automáticamente.

Estos son los **pasos** que se siguen para hacer el despliegue en Heroku, teniendo ya creada una cuenta en la plataforma:

1. **Configuración** del backend: Heroku indica a la aplicación, en cada nueva ejecución, un puerto donde correr. Ese puerto, nos viene dado automáticamente a través de la variable de entorno PORT, por lo que debemos indicarle al backend que debe inicializarse también bajo los valores dados por esa variable:

```
const PORT = process.env.PORT || 3000;
```

Ilustración 142 - Configuración Puerto Heroku

2. Creación de la app en Heroku:

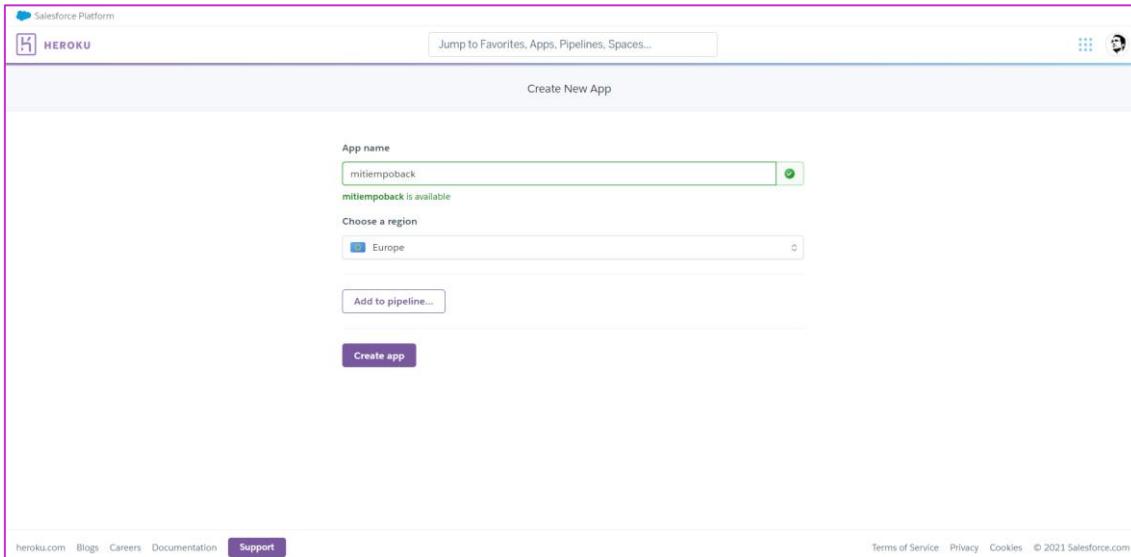


Ilustración 143 - Creación App Heroku

3. Heroku puede trabajar con distintas **tecnologías**, en este caso hay que indicar que el servidor de miTiempo es una aplicación de Node.js:

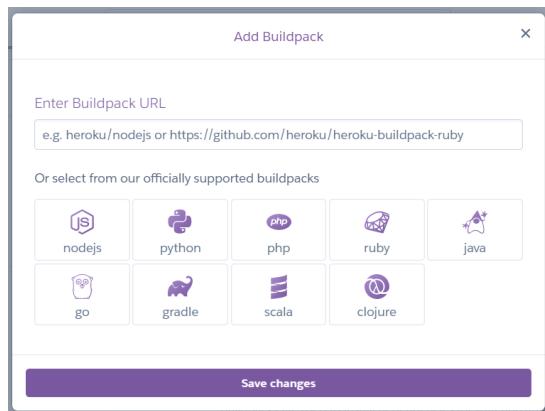


Ilustración 144 - Selección tecnología Heroku

4. **Configuración del repositorio** de GitHub. En este paso se le indica a Heroku desde qué repositorio y rama debe hacer el despliegue.

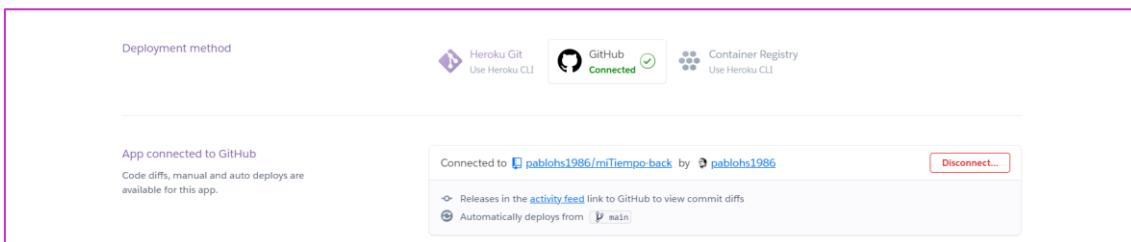


Ilustración 145 - Selección repositorio Heroku

5. Configuración de las variables de entorno:

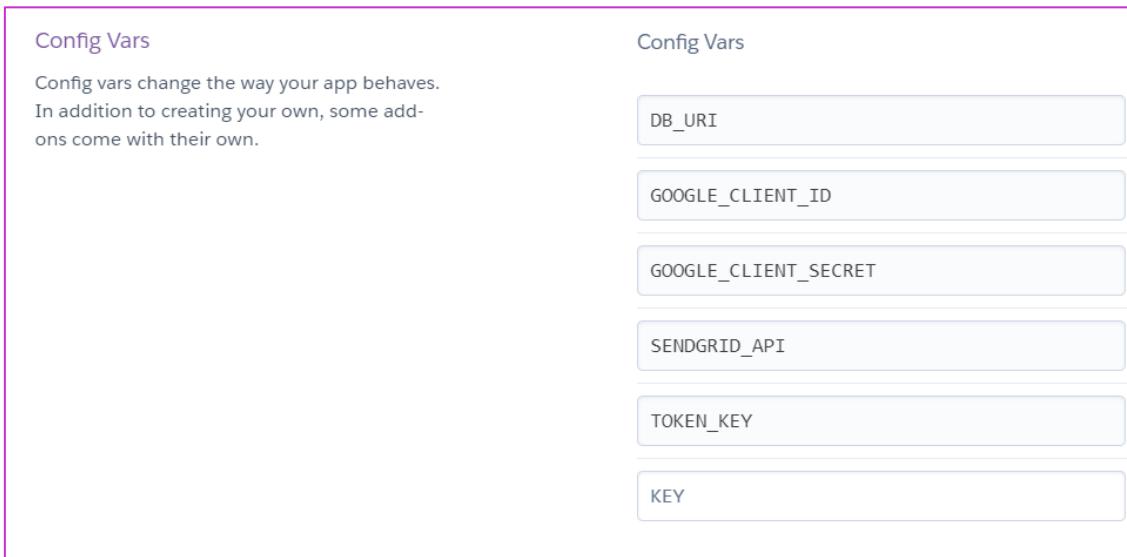


Ilustración 146 - Variables de entorno Heroku

Tras estos pasos, el servidor ya está corriendo en la nube mediante la plataforma Heroku. La siguiente captura muestra el log de Heroku, donde se pueden ver distintas peticiones que ha realizado el frontend al servidor en la nube:

A screenshot of a terminal window titled 'Heroku'. The log output shows several application logs and router logs. Application logs include: 'ed in a future version. To use the new Server Discover and Monitoring engine, pass option { useUnifiedTopology: true } to the MongoClient constructor.', '2021-05-30T15:25:35.694855+00:00 app[web.1]: (Use `node --trace-warnings ...` to show where the warning was created)', '2021-05-30T15:25:36.146799+00:00 heroku[web.1]: State changed from starting to up', '2021-05-30T15:25:35.980255+00:00 app[web.1]: Connected to MongoDB.', '2021-05-30T15:25:37.602824+00:00 heroku[router]: at=info method=OPTIONS path="/listTodayTasks/All" host=mitiempo-back.herokuapp.com request_id=4d5cecba-2aca-4651-acb6-e347d4a19473 fwd="81.36.75.209" dyno=web.1 connect=1ms service=15ms status=204 bytes=323 protocol=https', '2021-05-30T15:25:37.648705+00:00 heroku[router]: at=info method=OPTIONS path="/listTasks/All" host=mitiempo-back.herokuapp.com request_id=921338bd-fe9b-4404-94d3-3430c7a2eea6 fwd="81.36.75.209" dyno=web.1 connect=0ms service=2ms status=204 bytes=323 protocol=https', '2021-05-30T15:25:37.987715+00:00 heroku[router]: at=info method=GET path="/listTodayTasks/All" host=mitiempo-back.herokuapp.com request_id=57203f3c-e629-41e4-a520-5ceeaaff56e49 fwd="81.36.75.209" dyno=web.1 connect=1ms service=329ms status=304 bytes=227 protocol=https', '2021-05-30T15:25:38.072421+00:00 heroku[router]: at=info method=GET path="/listTasks/All" host=mitiempo-back.herokuapp.com request_id=d7a59a23-2ef0-4e06-9f03-645b9ed67108 fwd="81.36.75.209" dyno=web.1 connect=0ms service=371ms status=304 bytes=229 protocol=https'. Router logs are also present between these application logs.

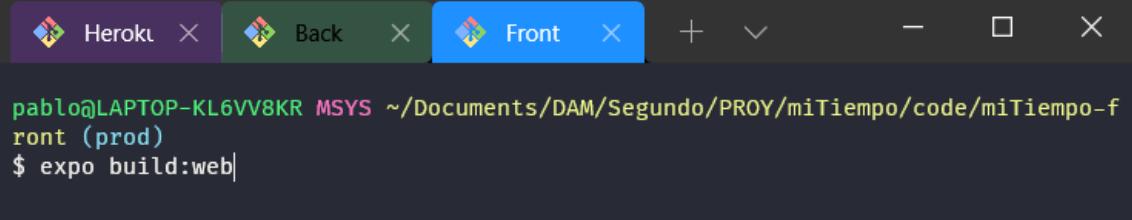
Ilustración 147 - Heroku Log

Web-app

El despliegue de la web-app se realiza sobre **Netlify**, un hosting web que también tiene integración con repositorios de GitHub. En este caso, se hace una **publicación manual**, puesto que Netlify debe tomar el build generado por Expo, no el código de GitHub. Esto quiere decir que cada vez que se haga un cambio en el código de la aplicación y se quiera actualizar la publicación, hay que repetir este proceso.

A continuación se muestran los **pasos** para realizar la publicación en Netlify:

1. **Build** del frontend mediante el framework Expo:



```
pablo@LAPTOP-KL6VV8KR MSYS ~/Documents/DAM/Segundo/PROY/miTiempo/code/miTiempo-front (prod)
$ expo build:web
```

Ilustración 148 - Build Expo 1



```
✓ Expo Webpack
Compiled successfully in 47.29s
```

Ilustración 149 - Build Expo 2

2. **Instalación del CLI** de Netlify:



```
pablo@LAPTOP-KL6VV8KR MSYS ~/Documents/DAM/Segundo/PROY/miTiempo/code/miTiempo-front (prod)
$ npm install netlify-cli -g
```

Ilustración 150 - Instalación CLI Netlify

3. Situado en el proyecto y rama de producción sobre la que se hizo el **build**, se realiza el despliegue. El CLI de Netlify nos va guiando por una serie de pasos:



```
pablo@LAPTOP-KL6VV8KR MSYS ~/Documents/DAM/Segundo/PROY/miTiempo/code/miTiempo-front (prod)
$ netlify deploy
```

Ilustración 151 - Despliegue Netlify 1

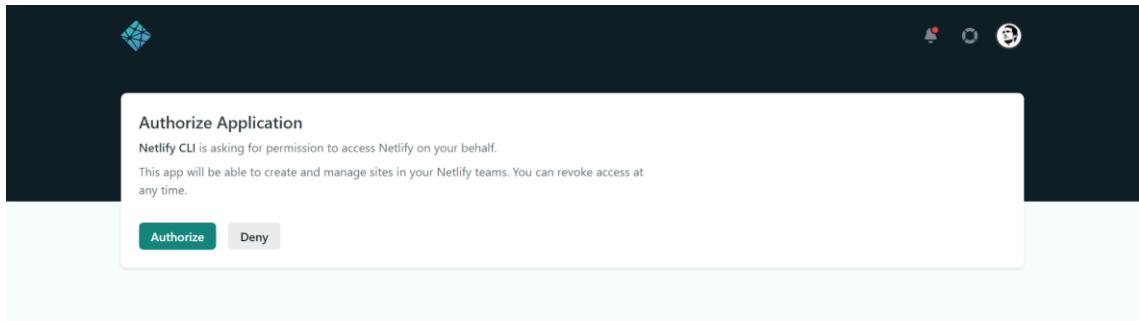


Ilustración 152 - Despliegue Netlify 2

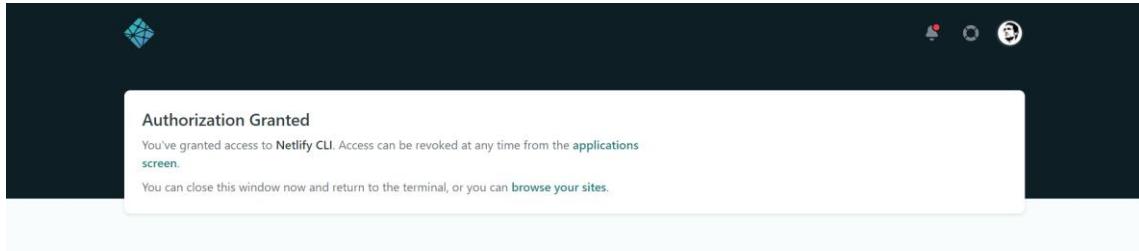


Ilustración 153- Despliegue Netlify 3

```
You are now logged into your Netlify account!
Run netlify status for account details
To see all available commands run: netlify help
This folder isn't linked to a site yet
? What would you like to do?
  Link this directory to an existing site
> + Create & configure a new site
```

Ilustración 154 - Despliegue Netlify 4

```
This folder isn't linked to a site yet
? What would you like to do? + Create & configure a new site
? Team:
> Pablo Herrero's team
```

Ilustración 155 - Despliegue Netlify 5

```
? Site name (optional): miTiempoWeb|
```

Ilustración 156- Despliegue Netlify 6

```
Site Created

Admin URL: https://app.netlify.com/sites/miTiempoWeb
URL: https://miTiempoWeb.netlify.app
Site ID: fe8b3c1b-0087-4e5d-a953-d971cfaaf40b
Please provide a publish directory (e.g. "public" or "dist" or "."):
C:\Users\pablo\Documents\dam\Segundo\PROY\miTiempo\code\miTiempo-front
? Publish directory web-build|
```

Ilustración 157- Despliegue Netlify 7

```

Deploy path: C:\Users\pablo\Documents\ DAM\Segundo\PROY\miTiempo\code\miTiempo-front\web-build
Deploying to draft URL...
✓ Finished hashing 45 files
✓ CDN requesting 7 files
✓ Finished uploading 7 assets
✓ Deploy is live!

Logs: https://app.netlify.com/sites/mitiempoweb/deploy/60b3b7c3a83485cf3fd84aa8
Website Draft URL: https://60b3b7c3a83485cf3fd84aa8--mitiempoweb.netlify.app

If everything looks good on your draft URL, deploy it to your main site URL with
the --prod flag.
netlify deploy --prod

```

Ilustración 158- Despliegue Netlify 8

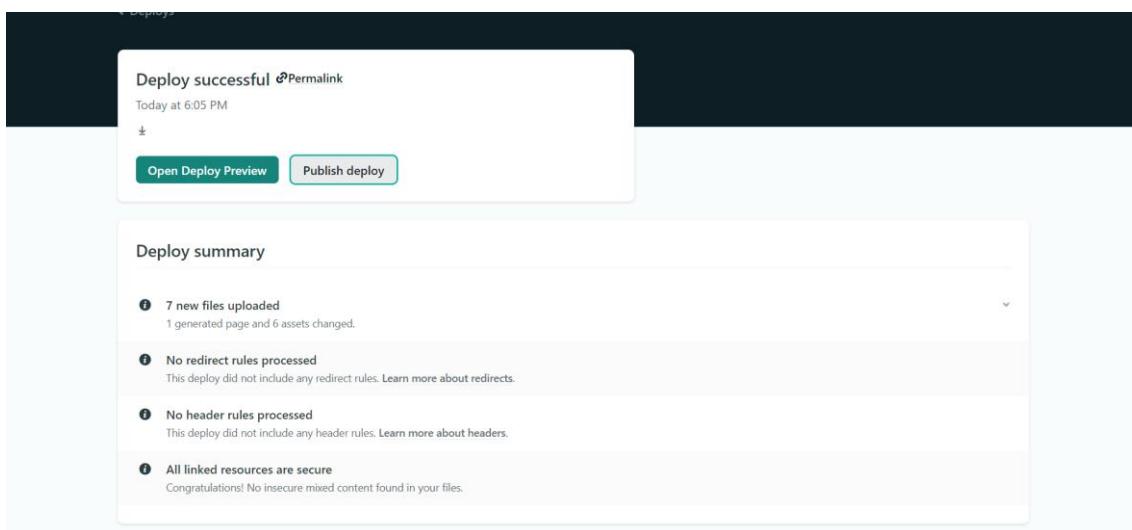


Ilustración 159 - Despliegue Netlify 9

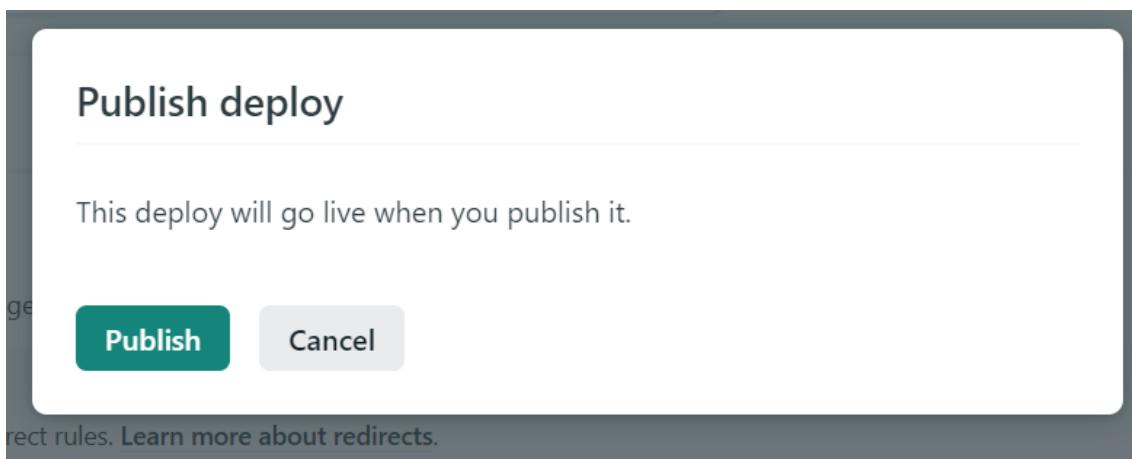


Ilustración 160 - Despliegue Netlify 10

De esta manera ya tendríamos publicado el front web en Netlify, pudiendo acceder el usuario desde la dirección <https://mitiempoweb.netlify.app/> .

Landing page

El deploy de la Landing Page se realiza también sobre **Netlify**. A diferencia del frontend, esta se asocia al repositorio de GitHub de esta capa de miTiempo, por lo que cada vez que se realiza un commit al repositorio de la Landing Page, se actualiza el código publicado en Netlify.

Estos son los **pasos** para realizar el deploy e una web estática en Netlify:

1. Se selecciona “**Create new site from Git**” y se elige el proveedor de repositorios:

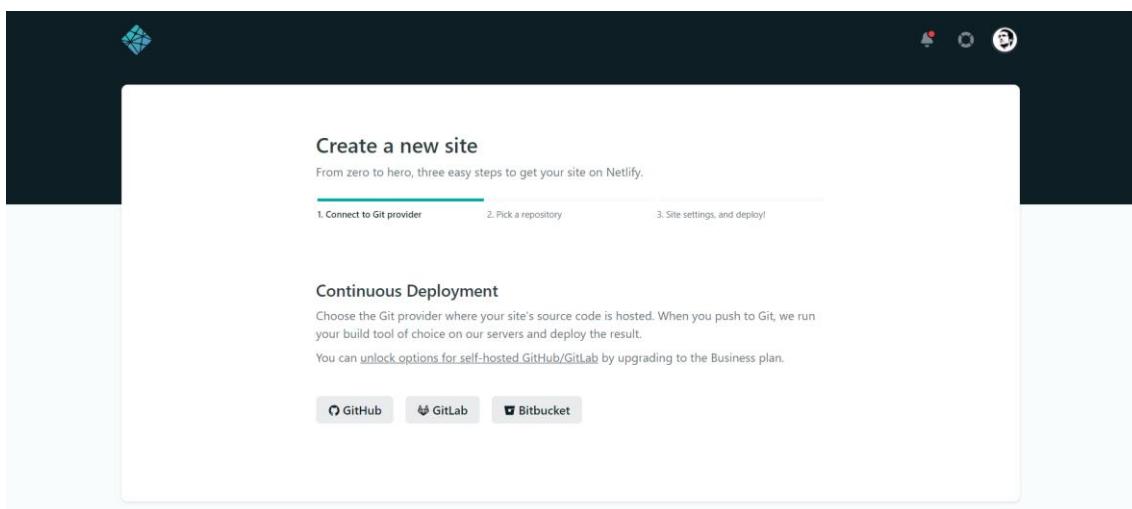


Ilustración 161 - Creación nuevo sitio Netlify

2. Se selecciona el **repositorio** y la rama:

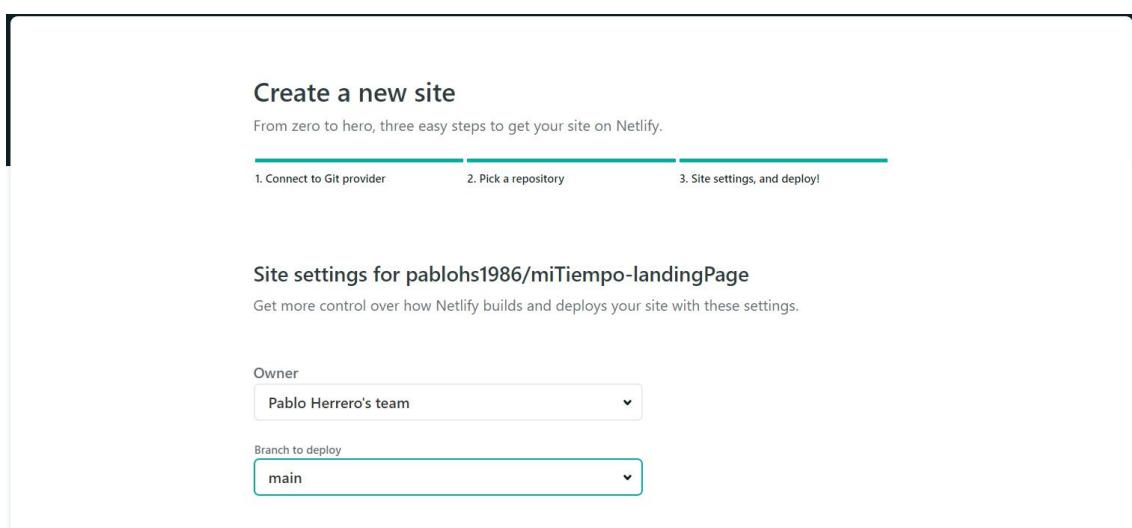


Ilustración 162 - Selección repositorio Netlify

De esta forma la Landing Page ya está publicada y el usuario puede acceder desde la dirección <https://mitiempoapp.netlify.app/>.

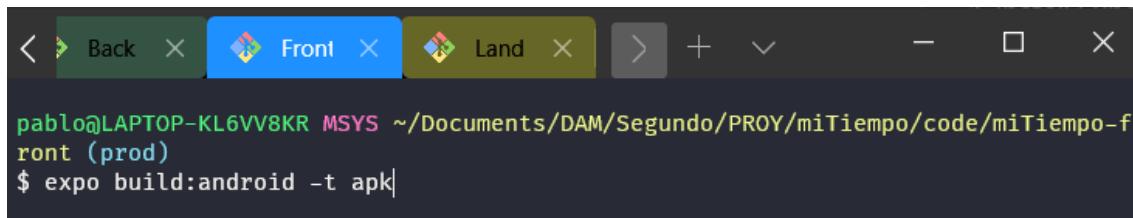
Apps Android y iOS

Para el despliegue de las aplicaciones de Android e iOS en las tiendas de aplicaciones **Play Store** y **App Store** hace falta registrarse como desarrollador y pagar una tasa, paso que, dada la finalidad de este proyecto, se va a obviar.

Por otro lado, aunque se ha probado la aplicación a lo largo de todo el desarrollo en un terminal **iOS**, asegurando su total compatibilidad para funcionar en esa plataforma, no se va a generar el ejecutable de la app para la misma (.ipa) que sería publicado en el App Store, puesto que no se dispone de un equipo MacOS configurado para tal fin.

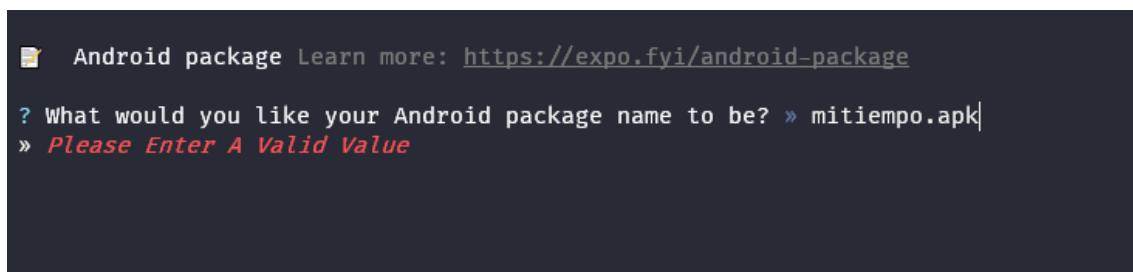
Por lo tanto, en el presente apartado se muestra cómo hacer el build o .apk de **Android**, que tendría como objetivo su despliegue en el Play Store de Google.

1. Desde la consola, situados sobre la ruta del proyecto y en la rama de producción, lanzamos el comando de Expo para realizar el **build**:



```
pablo@LAPTOP-KL6VV8KR MSYS ~/Documents/DAM/Segundo/PROY/mitiempo/code/mitiempo-front (prod)
$ expo build:android -t apk|
```

Ilustración 163 - Build Android Expo



```
Android package Learn more: https://expo.fyi/android-package
? What would you like your Android package name to be? » mitiempo.apk
» Please Enter A Valid Value
```

Ilustración 164 - Build Android Expo 2

2. Se crea una **cuenta en Expo** y se accede a ella desde la terminal, para proceder a crear el .apk, el cual es subido inicialmente a la web de Expo:

```
Accessing credentials for pherrero in project miTiempo-front
? Would you like to upload a Keystore or have us generate one for you?
If you don't know what this means, let us generate it! :) » - Use arrow-keys. Return to submit.
> Generate new keystore
I want to upload my own file
```

Ilustración 165 - Procesamiento .apk Expo 1

```
> Expo SDK: 40.0.0
> Release channel: default
> Workflow: Managed

Building optimized bundles and generating sourcemaps ...
Starting Metro Bundler
Building JavaScript bundle [=====] 57%
```

Ilustración 166 - Procesamiento .apk Expo 2

```
Uploading JavaScript bundles
Publish complete

Manifest: https://exp.host/@pherrero/miTiempo-front/index.exp?sdkVersion=40.0.0 Learn more: https://expo.fyi/manifest-url
Project page: https://expo.io/@pherrero/miTiempo-front Learn more: https://expo.fyi/project-page

Checking if this build already exists ...

Build started, it may take a few minutes to complete.
You can check the queue length at https://expo.io/turtle-status

You can monitor the build at

https://expo.io/accounts/pherrero/builds/eb6a5aea-13f7-4509-a54f-201199a0ea68

Waiting for build to complete.
You can press Ctrl+C to exit. It won't cancel the build, you'll be able to monitor it at the printed URL.
- Build queued ...
```

Ilustración 167 - Procesamiento .apk Expo 3

3. Desde la web de Expo se puede hacer un **seguimiento** del build, que tarda unos **15 minutos** en realizarse:

Ilustración 168 - Procesamiento .apk Expo 4

- Una vez finalizado, se procede a **descargar el .apk** desde el link que nos indica Expo en la terminal o desde nuestra cuenta de Expo:

```
You can monitor the build at
https://expo.io/accounts/pherrero/builds/eb6a5aea-13f7-4509-a54f-201199a0ea68

Waiting for build to complete.
You can press Ctrl+C to exit. It won't cancel the build, you'll be able to monitor it at the printed URL.
✓ Build finished.

Successfully built standalone app: https://expo.io/artifacts/9dc2814c-179b-4a13-90d8-f7423a6d5fa1
```

Ilustración 169 - Finalización Build Android Expo 1

Ilustración 170 - Finalización Build Android Expo 2

Con el .apk en nuestro poder, el paso lógico sería realizar la publicación del mismo en el Play Store, paso que, como se indicó al comienzo de este apartado, se omite por cuestiones obvias.

DOCUMENTACIÓN

Manual de instalación

Web-app

El usuario puede acceder a la **web-app** de miTiempo sin necesidad de instalaciones, simplemente mediante una de estas dos vías:

- Landing Page: Accediendo a la dirección <https://mitiempoapp.netlify.app/> y pulsando en el botón “web”:

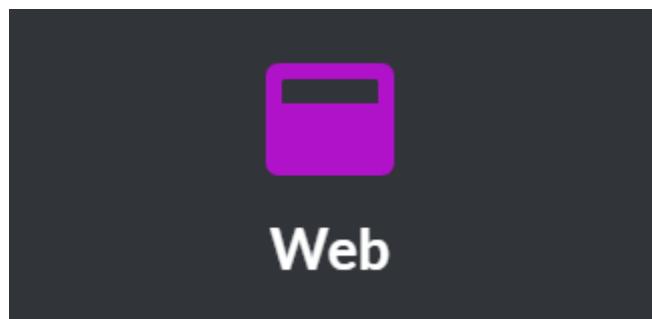


Ilustración 171 - Acceso Web-App

- Web-app: Accediendo directamente a la dirección de la web-app, <https://mitiempoweb.netlify.app/>

Android

El usuario debe dirigirse al **Play Store**, buscar la aplicación y descargar la misma. Dispone de un acceso directo en la **Landing Page**.

App Store

El usuario debe dirigirse al **App Store**, buscar la aplicación y descargar la misma. Dispone de un acceso directo en la **Landing Page**.

Manual de usuario

Alta de usuario

Con formulario

- Situarse en la pantalla de inicio.



Ilustración 172 - Pantalla alta

- Introducir un email y una contraseña válidos.

Email
Password

Ilustración 173 - Formulario alta

- Pulsar en el botón "Sign up".



Ilustración 174 - Botón alta

- Se accede a la aplicación.

Con Google

- Situarse en la pantalla de inicio.
- Pulsar en el botón "Sign up with G".
- Iniciar sesión en Google

- Si el proceso ha ido bien, se muestra la confirmación de envío de credenciales.

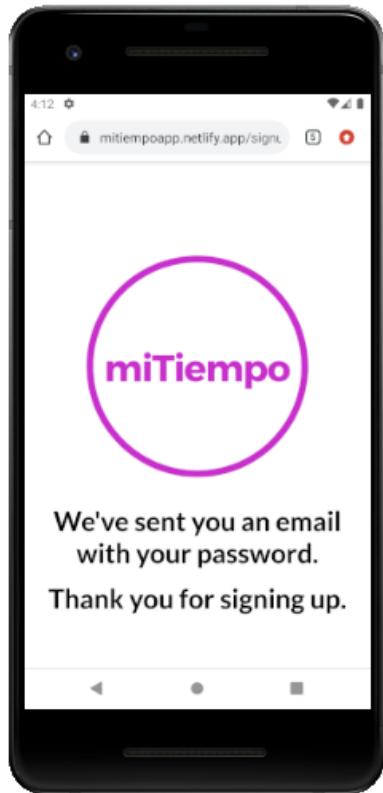


Ilustración 175 - Confirmación de envío credenciales

- El email recibido es como este.

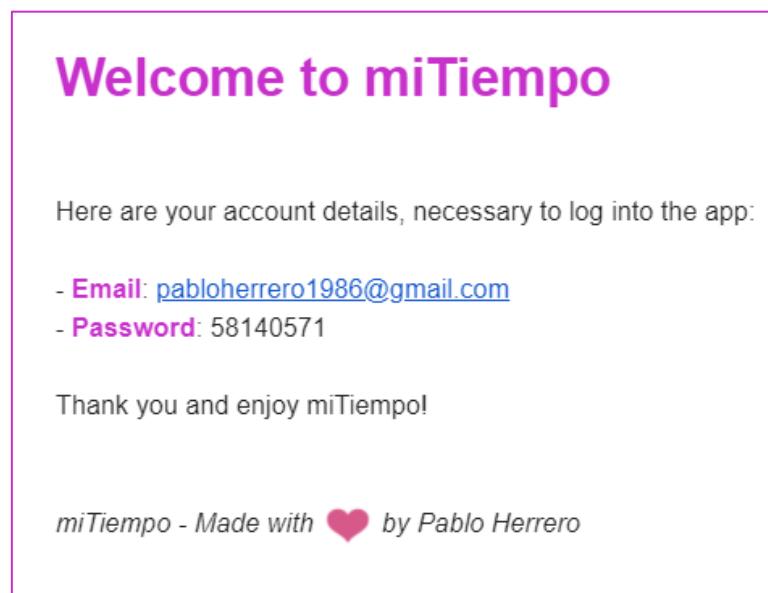


Ilustración 176 - Email credenciales

- El usuario puede logarse con dichas credenciales.

Iniciar sesión

- Situarse en la pantalla de inicio.
- Pulsar sobre “Sign in instead!”.

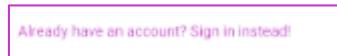


Ilustración 177 - Link inicio sesión

- Introducir email y contraseña.
- Pulsar en el botón “Sign in”.



Ilustración 178 - Botón inicio sesión

Creación de tarea

- Situarse en la pantalla principal.

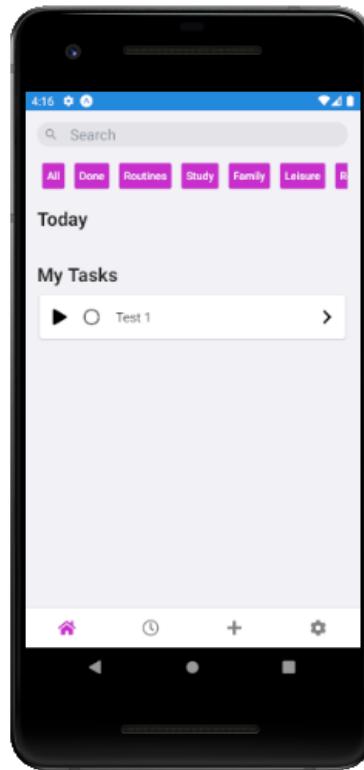


Ilustración 179 - Pantalla principal

- Pulsar sobre el botón “+” en la barra de navegación.



Ilustración 180 - Navegación

- Rellenar los campos “Title” y “Description”.

The form contains two input fields: 'Title' and 'Description'. Both fields have placeholder text: 'Enter a title for the task' and 'Enter a description for the task' respectively.

Ilustración 181 - Formulario creación de tarea

- Seleccionar las opciones deseadas (opcional).

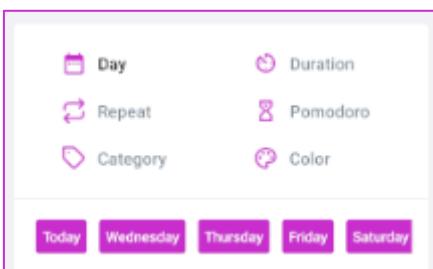


Ilustración 182 - Panel de opciones

- Cancelar o guardar la tarea con los botones “Cancel” y “Add”.



Ilustración 183 - Panel superior

Detalle de tarea

- Situarse en la pantalla principal.
- Pulsar sobre el chevrón a la derecha de la tarea.

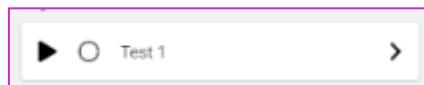


Ilustración 184 - Tarea

Edición de tarea

- Seguir los pasos dados en “Detalle de tarea”.
- Editar los valores deseados.

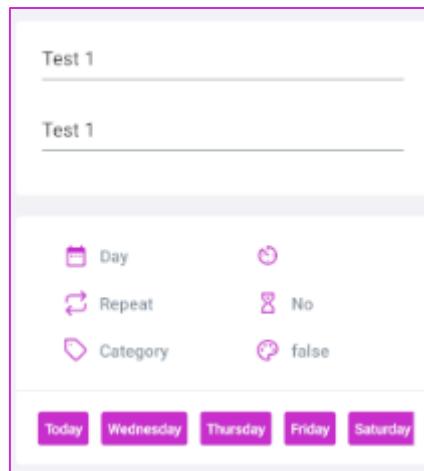


Ilustración 185 - Edición de tarea

- Pulsar en el botón “Ok”.

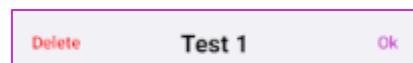


Ilustración 186 - Panel superior

Borrar tarea

- Seguir los pasos dados en “Detalle de tarea”.
- Pulsar el botón “Delete”.

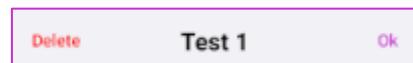


Ilustración 187 - Borrar tarea

Ejecutar tarea

- En la pantalla principal:
 - o Pulsar sobre el botón “Play” a la izquierda de la tarea.

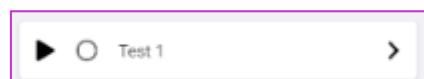


Ilustración 188 - Tarea

- En la pantalla de detalle de tarea
 - o Pulsar sobre el botón “Play” en la parte inferior de la pantalla.



Ilustración 189 - Botón ejecutar tarea

- Se ejecuta el temporizador de la tarea.

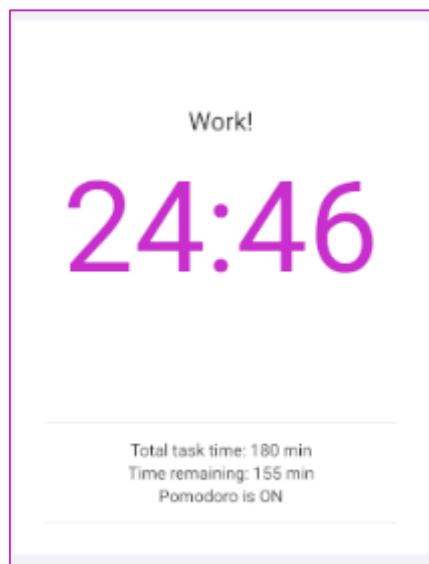


Ilustración 190 - Temporizador

Completar tarea

- Cualquier tarea se da por completada al ejecutarla y terminar su temporizador.
- Cualquier tarea se puede marcar como completada en cualquier momento pulsando el botón circular a la izquierda de la misma.

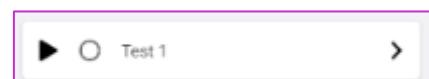


Ilustración 191 - Tarea

Tracker

- Pulsar sobre el botón “Reloj” en la barra de navegación.



Ilustración 192 – Navegación

- Seleccionar la gráfica deseada en el selector.

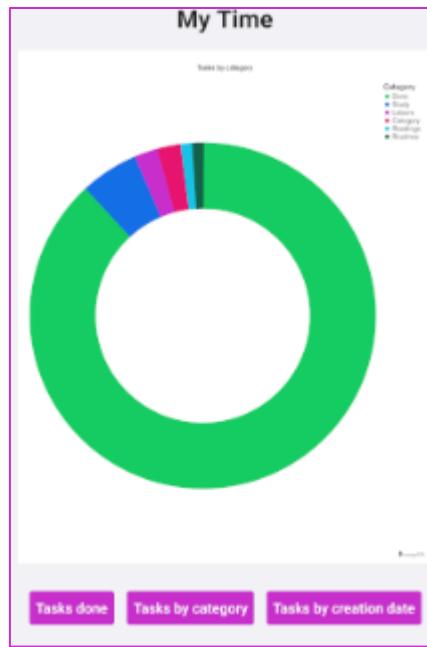


Ilustración 193 - Detalle Tracker

Cuenta de usuario

- Pulsar sobre el botón “Tuerca” en la barra de navegación.



Ilustración 194 - Navegación

- Pulsar sobre el botón “Edit” para editar los datos de la cuenta.



Ilustración 195 - Botón editar

- Pulsar en “Sign out” para cerrar sesión.



Ilustración 196 - Botón cerrar sesión

MEJORAS FUTURAS

A la finalización de la primera versión de la aplicación, se detectan una serie de **mejoras futuras** que se exponen a continuación:

- **Pruebas unitarias:** Las pruebas unitarias son la forma más eficaz de testear y mantener una aplicación a lo largo del tiempo, ya que permite comprobar el funcionamiento de la misma de una forma eficaz. Se espera implementar librerías de testeo como Jest o Detox en un futuro próximo.
- **Animaciones:** La implantación de animaciones en el interfaz de miTiempo se descartó por falta de tiempo y por no sobrecargar el código. Librerías como React-Native-Reanimated son ideales para dar un toque extra a los interfaces de React.
- **Depuración del diseño:** Aunque se tiene una primera versión funcional, el diseño de la aplicación debe ser depurado, puliendo las pequeñas diferencias que hay entre las distintas plataformas y modificando ciertas partes del interfaz para una mejor calidad de la app.
- **Modo concentración:** Se espera abordar el modo concentración a la mayor brevedad posible, implementando un aviso si el usuario sale de la pantalla del temporizador.
- **Tracker automático:** Se desea investigar e implementar la forma de que el tracker analice en segundo plano las aplicaciones que usa el usuario en su dispositivo, así como el tiempo que destina a ellas.

CONCLUSIONES

Como se ha visto a lo largo de esta documentación, **miTiempo** es una aplicación multiplataforma realizada con el stack **MERN** que ha supuesto todo un **reto y experiencia de aprendizaje para el desarrollador** que escribe estas líneas. Con ella no sólo se ponen en práctica los conocimientos adquiridos a lo largo del ciclo, sino que se aprenden tecnologías de vanguardia y se hace uso de arquitecturas, herramientas y conceptos de estilo de código que son tendencia actualmente en el sector.

Para finalizar, queda la duda, a modo de **reflexión**, de si el uso de otro stack diferente o la realización de aplicaciones nativas con Android/Swift/Web, hubiera supuesto un ahorro de *miTiempo* en el desarrollo de esta aplicación. Las virtudes de los frameworks multiplataforma como React Native están claras, pero con este proyecto también quedan patentes sus defectos. Sea como sea, el paso del tiempo definirá el futuro de este tipo de tecnologías que, de una forma u otra, han llegado para quedarse.

REFERENCIAS BIBLIOGRÁFICAS

Cursos online

- MongoDB Basics M001, MongoDB University.
- MongoDB for Javascript Developers M220JS, MongoDB University.
- Modern React with Redux, Stephen Grider.
- The Complete React Native + Hooks Course, Stephen Grider.
- Node with React: Fullstack Web Development, Stephen Grider.

Libros

- JavaScript: The Definitive Guide, Flanagan.
- Beginning Node.js, Express & MongoDB Development, Greg Lim.
- The Road to React 2020 edition, Robin Wieruch.
- Clean Code, Robert C. Martin.
- The Pragmatic Programmer, David Thomas, Andrew Hunt.

Recursos web

Los recursos web consultados a lo largo del proyecto son innumerables. Algunos de los más destacados son los siguientes:

- <https://stackoverflow.com/>
- <https://dev.to/alekswritescode/infinite-pomodoro-app-in-react-52jj>
- http://www.lsi.us.es/~javierj/cursos_ficheros/03.%20Sokoban.%20Un%20ejemplo%20de%20plantillas.pdf
- https://repositorio.uam.es/bitstream/handle/10486/662281/gomez_matesanz_alfonso_tfg.pdf?sequence=1
- <https://dev.to/alekswritescode/infinite-pomodoro-app-in-react-52jj>
- <https://creately.com/blog/es/diagramas/tutorial-diagrama-caso-de-uso/>
- <https://repositorio.grial.eu/bitstream/grial/1155/1/UML%20-%20Casos%20de%20uso.pdf>
- https://ingenieria.udistrital.edu.co/pluginfile.php/38829/mod_resource/content/5/20130820-%20Casos%20de%20Uso.pdf
- <http://elvex.ugr.es/idbis/db/docs/design/2-requirements.pdf>
- <https://victorgraciaweb.com/ventajas-y-desventajas-entre-mysql-vs-mongodb/>
- <https://openwebinars.net/blog/ventajas-y-desventajas-de-mongodb/>
- <https://www.adolfocuadros.com/bases-de-datos/aprendiendo-mongodb/>
- <https://dzone.com/articles/view-mongodb-collections-as-diagrams-yes-it-is-pos>

- <https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/>
- <https://www.techopedia.com/definition/30736/object-data-model>
- <https://www.izertis.com/es/-/blog/criptacion-de-password-en-nodejs-y-mongodb-bcrypt>
- https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/mongoose
- https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes
- <https://medium.com/@selvaganesh93/how-node-js-middleware-works-d8e02a936113#:~:text=A%20middleware%20is%20basically%20a,once%20your%20middleware%20code%20completed.>
- <https://medium.com/@aarnlpezsosa/middleware-en-express-js-5ef947d668b>
- <https://hackernoon.com/understanding-react-native-bridge-concept-e9526066ddb8>
- <https://www.izertis.com/es/-/blog/react-native-de-javascript-al-desarrollo-de-aplicaciones-moviles>
- <https://www.reactnative.guide/3-react-native-internals/3.1-react-native-internals.html>
- <https://mobidev.biz/blog/how-react-native-app-development-works>
- <https://stackoverflow.com/questions/36012239/is-shadow-dom-fast-like-virtual-dom-in-react-js>
- https://www.youtube.com/watch?v=TU_kTuz2i9Y
- <https://www.freecodecamp.org/news/how-react-native-constructs-app-layouts-and-how-fabric-is-about-to-change-it-dd4cb510d055/>
- <https://www.paradigmadigital.com/dev/desarrollando-aplicaciones-moviles-nativas-con-react-native/>
- <https://medium.com/@chenfeldmn/react-native-a-bridge-to-project-fabric-part-2-1f082415b881#:~:text=Shadow%20Node%2FTree%20%E2%80%94%20Represent%20a,and%20the%20DOM%20in%20ReactJS.>
- <https://css-tricks.com/understanding-how-reducers-are-used-in-redux/#:~:text=A%20reducer%20is%20a%20function,so%20that%20they%20behave%20consistently.>
- <https://www.netguru.com/codestories/react-native-lifecycle>
- <https://medium.com/@simonhoyos/qu%C3%A9-es-javascript-95006a2f94f9>
- https://www.w3schools.com/react/react_components.asp#:~:text=Components%20are%20independent%20and%20reusable,will%20concentrate%20on%20Class%20components.

- <https://es.reactjs.org/docs/faq-state.html#:~:text=state%20representan%20los%20valores%20renderizados,despu%C3%A9s%20de%20llamar%20a%20setState%20.>
 - <https://platzi.com/blog/estructura-organizacion-y-tipos-de-componentes-en-react/>
 - <https://www.twilio.com/blog/react-choose-functional-components#:~:text=First%20of%20all%2C%20the%20clear,JavaScript%20class%20that%20extends%20React.&text=The%20JSX%20to%20render%20will%20be%20returned%20inside%20the%20render%20method.>
 - <https://es.reactjs.org/docs/react-component.html>
 - <https://es.reactjs.org/docs/context.html>
 - <https://blog.expo.io/create-and-deploy-web-apps-and-pwas-with-expo-a286cc35d83c>
 - <https://docs.expo.io/distribution/building-standalone-apps/>
 - <https://programacionymas.com/blog/diferencia-entre-javascript-y-ecmascript>
 - https://www.w3schools.com/js/js_es6.asp
 - https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment
 - <https://reactnative.dev/docs/javascript-environment>
 - <https://stackoverflow.com/questions/40291766/naming-convention-for-const-object-keys-in-es6/40291962>
 - <https://gilshaan.medium.com/react-native-coding-standards-and-best-practices-5b4b5c9f4076>
-

APÉNDICES

Glosario

Action function

Función que toma o modifica el *State* dentro de un *Reducer*.

Animate.css

Librería de animaciones CSS. Permite enfatizar elementos web de forma muy llamativa para el usuario.

AOS

Similar a Animate.css, AOS se centra en animaciones que se *disparan* con el uso del scroll dentro de la web en la que se implementa.

Bootstrap 4

Framework de HTML, CSS y JavaScript que permite la creación ágil de páginas web responsive o diseño adaptativo, es decir, adaptadas a las pantallas de cualquier dispositivo. Se basa en elementos y propiedades predefinidas, implementadas directamente a través del HTML, fácilmente personalizables a través de CSS.

Colecciones

Estructuras de datos que almacenan documentos, normalmente con campos compartidos.

Componente

Es una clase o función que puede recibir una serie de *Props* o propiedades y manejar el *State* o estado, retornando código JSX a renderizar. En este proyecto, dado el uso de *Hooks* que se implementa en el mismo, se hace uso de componentes de función o funcionales, los cuales se basan en funciones, a diferencia de los componentes de clase, basados en clases de JavaScript.

Context

Es un objeto que permite pasar datos y manejar el estado a través del árbol de componentes sin tener que pasar *props* manualmente en cada nivel. Cuando un componente se suscribe al *Provider* o proveedor de un contexto, permite el manejo de la información de dicho contexto entre un componente padre y sus componentes hijos anidados, permitiendo pasar información fácilmente a un componente hijo específico, sin necesidad de pasar *props* a través de cada nivel de componentes hijos.

Documentos

Elementos dentro de una colección que almacenan la información en un formato similar a JSON, BSON, que es simplemente un “superset” de JSON, la representación binaria del mismo, permitiendo más tipos de datos dentro de MongoDB.

ECMAScript 6

ECMAScript 6, ES6 o ECMAScript 2015 fue la segunda mayor revisión de JavaScript, publicada en 2015.

ECMAScript 8

ECMAScript 8, ES8 o ECMAScript 2017 es una actualización menor del estándar de ECMA para Javascript.

Estado

Objeto de JavaScript que contiene información relevante del componente. A efectos, podría decirse que un estado está compuesto de diferentes variables del componente, las cuales pueden ser modificadas mediante *setters*. Una actualización del estado implica la re-renderización del componente.

Event bubbling

Cuando se genera un evento en un componente hijo, se propaga hacia arriba hacia los componentes padre

Git: Sistema de control de versiones que permite llevar un registro de los cambios en los archivos de un proyecto, siendo especialmente útil para coordinar el trabajo entre varias personas sobre archivos compartidos.

GitHub: Plataforma de hospedaje de repositorios de código en la nube utilizando Git. Incluye **GitHub Projects**, proyectos en forma de tableros y vinculados a repositorios que sirven para organizar el trabajo sobre los mismos.

Heroku

Plataforma como servicio (PaaS) que sirve para alojar aplicaciones de diversos tipos en la nube y con integraciones con otras plataformas, como GitHub.

Hook

Son funciones que aportan funcionalidad adicional a los componentes. Las hay de diversos tipos y con diversas funcionalidades.

JSX

Extensión de la sintaxis de JavaScript que permite combinar JavaScript y HTML. Utilizado por React Native para el renderizado de los componentes.

MERN

Pila de tecnologías de desarrollo que tienen a JavaScript como nexo de unión: MongoDB, Express, React y Node.js.

Middleware

Es una función que recibe objetos Request y Response y la función Next(). Se implementan sobre la app o sobre cada route para realizar operaciones que se pueden extender a varios

routes (como por ejemplo, requerir una autenticación de usuario para procesar una petición). Se pueden ejecutar varias Middleware en cadena (pueden ser apiladas).

Model

Un modelo de Mongoose contiene esquemas gracias a los que se pueden crear objetos que son mapeados por esta librería para crear documentos en una base de datos MongoDB o recibir datos de la misma.

Netlify

Hosting web con gran integración con servicios como GitHub.

Pomodoro

La **Técnica Pomodoro** es un método para mejorar la administración del tiempo dedicado a una actividad. Fue creado por **Francesco Cirillo** a fines de la década de 1980.¹ Se basa en usar un temporizador para dividir el tiempo en intervalos fijos, llamados *pomodoros*, de 25 minutos de actividad, seguidos de 5 minutos de descanso, con pausas más largas cada cuatro pomodoros.

Prop

Es un sistema para pasar información de un componente padre a un componente hijo. Estas propiedades son variables que se reciben como parámetro en el componente hijo.

Provider

Función que provee de un contexto a los componentes hijos asignados.

Reducer

Es una función que determina cambios en el estado, usando la acción o *action function* que recibe para determinar ese cambio. Tiene acceso a todos los estados de la app y se implementa con el Hook `useReducer()`.

Route

Es un método de Express que va ligado a una petición HTTP (GET, POST, PUT, DELETE), a una URL/Path y a una función que recibe dos objetos, Request y Response, que es llamada para manejar esa petición y devolver o no una respuesta. Entre la petición y la respuesta, pueden recibir Middlewares para realizar operaciones y validaciones intermedias.

Contenido entregado

En este apartado se hace un desglose del contenido entregado en el momento de la entrega del proyecto.

Documentación

La presente documentación en formato .pdf, se encuentra en la carpeta **Documentación** con el nombre de archivo **DocumentacionMiTiempoPabloHerreroSanchez.pdf**

Código y ejecutable

En la carpeta **codigoFuente** se encuentra el código de las tres capas del proyecto, Landing Page, Frontend y Backend, así como el ejecutable de Android .apk.

El código de cada una de las partes también se encuentra en los siguientes repositorios de GitHub:

- **Landing page:** <https://github.com/pablohs1986/miTiempo-landingPage>
- **Frontend:** <https://github.com/pablohs1986/miTiempo-front>
- **Backend:** <https://github.com/pablohs1986/miTiempo-back>

Dichas capas se encuentran corriendo en los siguientes hostings:

- **Landing page:** <https://mitiempoapp.netlify.app/>
- **Frontend:** <https://mitiempoweb.netlify.app/>
- **Backend:** <https://mitiempo-back.herokuapp.com/>

Presentación

En la carpeta **Presentacion** se encuentran los ejecutables y enlaces a la presentación del proyecto, así como los vídeos mostrando el código y las tecnologías (**CodigoMiTiempoPabloHerreroSanchez.mp4**) y las funcionalidades de la app (**DemoMiTiempoPabloHerreroSanchez.mp4**). También hay una subcarpeta **OtrosVideos** con los vídeos usados en la presentación.

Diario de desarrollo

En el presente apartado se muestra el diario de desarrollo de miTiempo, utilizado junto con GitHub Projects a modo de histórico y guía para la realización del proyecto, dejando constancia del trabajo realizado.

09/03 al 08/04 – FORMACIÓN

MongoDB

- MongoDB for Javascript Developers M220JS, MongoDB University.

Node.js + Express

- Beginning Node.js, Express & MongoDB Development, Greg Lim.

React

- The Road to React 2020 edition, Robin Wieruch.
- Modern React with Redux, Stephen Grider.

React Native

- The Complete React Native + Hooks Course, Stephen Grider.

MERN

- Node with React: Fullstack Web Development.

08/04 – COMIENZO DEL DESARROLLO

- Desglose de funciones de la app.
- Creación de proyecto en GitHub.

09/04

- Mockup del front en Figma.
- Creación de repositorios en GitHub.
- **Back:**
 - o Generación del proyecto.
 - o Instalación de dependencias.
 - o Creación de cluster de MongoDB
 - o Conexión al cluster de MongoDB.
 - o Configuración de nodemon y dotenv.
- **Front:**
 - o Generación del proyecto.

10 y 11/04

- **Back:**
 - o Se implementa .env donde albergar las variables de entorno y keys a mantener seguras.
 - o Primer intento (fallido) de implementar el registro y el login con Google y Express. Se toma como toma de contacto.
 - o Implementación del registro y el login con usuario/contraseña. Pruebas con Postman.
 - o Creación del modelo User.
- **Front:**
 - o Generación de la estructura base de la navegación.
 - o Generación de la estructura base de las pantallas.
 - o Creación de componente de ayuda Spacer.
 - o Instalación del framework de UI React Native Elements.

- Creación de la pantalla signup.

12/04

- **Back:**
 - Se configuran variables de entorno en Heroku para habilitar la conexión entre Front y Back.
- **Front:**
 - Iconos en barra de navegación.
 - Refactorización en componentes para pantallas de sign up/in.
 - Creación pantalla sign in.
 - Conexión a la api mediante Axios.
 - Se implementa la funcionalidad de sign up con email/pass.
 - Se implementa el contexto:
 - Se genera un creador de contexto a partir de un provider.
 - Se genera un contexto propio para el sistema de login mediante action functions y el provider generado.

13/04

- **Back:**
 - Se implementa una ruta getUserInfo para conseguir la información de un usuario logado a partir de su token.
 - Se solventa un error en la versión web de la app: un problema con los cors impedía que ciertas request fueran aceptadas por el backend. Se implementa la librería cors para solventarlo.
- **Front:**
 - Se refactoriza la navegación en una carpeta/archivo independiente.
 - Se crea un navegador externo para cuando no se puede hacer uso directo de la navegación de React.
 - Se implementa la funcionalidad de sign in con email/pass.
 - Se implementa la funcionalidad de tryLocalSign (con token almacenado en dispositivo).
 - Se solventa un error de las pantallas de sign in/up: cuando se mostraba un error y se cambiaba de pantalla, este no se eliminaba. Se implementa un método para solventarlo.

14/04

- **Back:**
 - Se refactoriza la ruta getUserInfo en un archivo aparte.
- **Front:**
 - Se finaliza al estructura de la AccountScreen. Se intenta implementar un modal, pero se descarta por incompatibilidad de RN con parte web.
 - Se implementa la función de signOut.
 - Se comienza a trabajar en el UserContext.

15/04

- **Back:**
 - Se refactoriza, documenta y limpia la route getUserInfo.
- **Front:**
 - Se refactoriza accountScreen y se elimina el componente de formulario (no era necesario, no se iba a reutilizar).
 - Se implementa UserContext y la action getUserInfo. Se añaden docs en todos los context y createDataContext.

- Se actualiza miTiempoApi para implementar un interceptor de request que fuerce la autenticación en cada una de ellas.
- Se implementa tryLocalSign con una página de carga.
- Se trabaja en el manejo del state en la accountScreen para mostrar la información del usuario.

16/04

- **Back:**
 - Se implementa updateUserInfo.
 - Limpieza en documentación de userRoutes y authRoutes.
- **Front:**
 - Se trabaja en implementar updateDataInfo en la pantalla Account. Varios problemas con el manejo del state, se opta por dividir la pantalla en dos, Account y EditScreen, para tener una ui más clara para el usuario y un manejo del state más claro.

17/04

- **Back:**
 - Creado el modelo de Task.
 - Creado taskRoute e implementadas y testeadas las routes listTasks, listTaskToday y addTask.
- **Front:**
 - Implementadas AccountScreen y EditAccountScreen junto con updateUserInfo y getUserInfo.
 - Se renombra la pantalla TackListScreen a TaskListScreen. Se hace la base para la misma.

18/04

- **Back:**
 - Se actualizan las routes listTask y listTodaysTasks para que puedan recibir un parámetro para filtrar por categoría.
 - Se implementa las routes updateTask junto con una middleware checkFieldsToUpdate que detecta los campos a actualizar en un documento.
 - Se implementa la middleware checkFieldsToUpdate sobre updateUserInfo.
- **Front:**
 - Se elimina el botón Back en EditAccountScreen (en contexto móvil, no es necesario, tampoco para la web, en esta app).
 - Se trabaja en TaskContext, TaskList y TaskHomeScreen. Problemas pasando el parámetro Category a través de una GET request.

19/04

- **Back:**
 - Solucionada la forma de recibir parámetros en una petición get, así como diversos errores, para las routes listTasks y ListTodayTasks.
 - Se implementa la route getCategory.
 - Se implementa el manejo de errores en las routes indicadas.
- **Front:**
 - Se solventa la forma de enviar parámetros a través de una petición get para listTasks y listTodayTasks.
 - Se implementa listCategories.
 - Se trabaja en TaskHomeScreen, implementando el contexto y aplicando el estilo.

- Se implementa las funciones de filtrar por categoría y búsqueda en TaskHomeScreen.

20/04

- **Front:**

- Se trabaja en implementar el diseño de TaskCreateScreen. Algunos problemas para utilizar pickers en ambas capas de la app (móvil y web). Se busca una opción cómoda para el usuario y que no implique navegación excesiva entre pantallas, por lo que se decide reutilizar un componente ya creado en TasksHomeScreen para el panel de opciones de TaskCreateScreen, categoriesList, rebautizándolo como horizontalList.
- Se modifica taskContext para añadir la carga de los presets de las opciones, implementándolo en TaskCreateScreen.

21/04

- **Front:**

- Se termina de integrar el diseño y el contexto en TaskCreateScreen.
- Se crea el componente MoveToBottom para ayudar a situar determinados componentes en el pie de la pantalla.
- Se añade alguna documentación faltante al proyecto.

22/04

- **Back:**

- Se realizan algunos cambios en el modelo de Task y TaskRoutes, para alinear campos en front y back.

- **Front:**

- Se realizan cambios en el horizontalList para las categorías.
- Se comienza a trabajar en implementar addTask en TasksCreateScreen.

23/04

- **Back:**

- Se implementa la middleware newTaskDataHandler y la route addTask.
- Se soluciona un problema en listTodayTasks con las fechas, para alinear la forma de tratarlas con el front.
- Se ordenan las tareas por fecha de creación en listTasks and listTodayTasks.

- **Front:**

- Se implementa addTask.

24/04

- **Front:**

- Se detecta que .toLocaleString() usado en getDaysArray, funciona para dispositivos iOS, pero no en Android. Se investiga y se opta por utilizar una librería externa, Moments.js.
- Se implementa refreshPresets en TaskCreateScreen para refrescar todos los campos al volver a la pantalla.
- Se trabaja en refining el diseño de TaskHomeScreen.

25/04

- **Front:**
 - o Se implementa un nuevo componente, sectionContainer, para resaltar secciones dentro de cada pantalla.
 - o Se replantea y modifican los controles de TaskCreateScreen. Se trata de implementar en el resto de pantallas sin éxito por incoherencia del resultado entre iOS y Android, por lo que se decide dejar el resto de pantallas como estaban.

01/05

- **Back:**
 - o Se crea la route getTaskById, aunque posteriormente se elimina al no ser totalmente necesaria para realizar las acciones sobre una sola tarea en el front (desde este, teniendo acceso a la lista de tareas, se puede iterar y realizar acciones sobre una de las tareas, por lo que esta route no era necesaria).
 - o Se implementan las middlewares ya creadas sobre updateTask.
 - o Se realizan cambios para tener un código más claro en deleteTask.
- **Front:**
 - o Se soluciona el error "No safe area insets value available" en plataforma web provocado por safearea de React Native. Se implementa safearea de Expo para solventarlo.
 - o Se implementa la pantalla TaskDetail.
 - o Se implementan las funciones updateTask y deleteTask sobre la pantalla TaskDetail.

02/05

- **Back:**
 - o Se excluyen las tareas de categoría "done" en listTasks and listTodayTasks.
- **Front:**
 - o Se trabaja en marcar las tareas como completadas.
 - o Se centran las listas horizontales en plataforma web, haciendo uso de código específico para ella en el componente horizontalList.

03/05 a 09/05

- **Back:**
 - o Se actualizan las duraciones en newTaskDataHandler para implementar los cambios realizados en el front.
 - o Se soluciona un error con isPomodoro en newTaskDataHandler, el cual estaba devolviendo siempre false, independientemente del valor recibido.
- **Front:**
 - o Se trabaja en TaskTimerScreen y en el temporizador, implementando toda la lógica:
 - Para simplificar, se controla toda la lógica desde la pantalla TaskTimerScreen, y se realiza un componente por separado, Timer, que tan solo se encarga de recibir la cantidad de tiempo a gestionar y mostrar la cuenta atrás correspondiente.
 - Se implementa que se marque una tarea como hecha cuando se consume su tiempo.
 - o Se aplica el color de cada tarea a los controles en TaskHomeScreen.
 - o Se modifica el ancho de los botones para toda la app en la versión web, realizando código específico para ella.

- Se trabaja en TrackerScreen:
 - Se valoran varios abordajes: uso de librerías externas que lean de la base de datos o unas MongoDB Charts. Se decide implementar esta segunda opción, “embediendo” las gráficas directamente desde esta opción de Mongo.
 - Se implementa el componente HorizontalList para actuar como selector entre las distintas gráficas.
- Se modifica la forma de buscar en la searchBar de la pantalla TaskHome, realizando la búsqueda por inclusión y sin distinguir mayúsculas y minúsculas. Se consigue realizar una búsqueda más dinámica y vistosa para el usuario.

10/05 a 16/05

- **Back y front:**
 - Se busca la forma de implementar el alta/login con Google mediante Passport y OAuth2.
 - Se encuentran varios problemas:
 - No se encuentra la forma de enviar el token desde el back al front a través del callback de la estrategia de Google.
 - No se encuentra la forma de redirigir con Axios al prompt de Google.
 - Se opta por usar la capa de autenticación de Google, a la cual se redirecciona desde el front, para crear el usuario en la base de datos, junto con una clave aleatoria que se envía al usuario mediante un email generado en el back por Nodemailer y SendGrid.
- **Landing page:**
 - Se desarrolla una página de entrada para la app, que sirva de gancho e introducción al usuario, así como de enlace a las versiones de la app móvil y web.
 - A su vez, se despliega una página de confirmación de envío de email a la que redireccionar en el alta con Google.
 - Se despliega en Netlify.

20/05

- **Front:**
 - Se realizan cambios en el diseño, aplicando código específico para la parte web.
- **Landing page:**
 - Se añade iOS como plataforma.
 - Se modifican algunos colores.

22/05

- **Front:**
 - Se soluciona el error ““TypeError: Cannot read property 'title' of undefined”” al borrar una tarea. Era provocado porque se intentaba asignar estados sobre una tarea inexistente.

30/05

- **Back:**
 - Limpieza de código en AuthRoutes.
- **Front:**

- Se soluciona un bug con las gráficas en TrackerScreen. Por algún motivo se dejaron de mostrar en la parte móvil. Se saca el componente del View y se soluciona.
- Se implementan Dimensions para hacer responsive la parte web.
- Se realiza el deploy de la parte web y el build de la app Android.
- **Landing:**
 - Se actualizan los enlaces de la sección de acciones.

31/05

- **Back:**
 - Se arregla el valor por defecto para la duración de la tarea en newTaskDataHandler().
- **Front:**
 - Se añaden validaciones extra en Signup, Signin, Account, EditAccount, Task, TaskCreateScreen.
 - Se añaden iconos y una pantalla de carga.
 - Se realiza un nuevo despliegue para la parte web y build para Android.
- **Landing page:**
 - Se añade favicon.
 - Se realiza nuevo despliegue.

01/06

- **Back y Front:**
 - Se detecta un bug en la pantalla TrackerScreen, cargándose siempre las tablas con los datos de todos los usuarios, no del especificado. Se modifica la userRoutes para devolver la id del usuario. Se especifica el campo a filtrar en MongoDB Charts y se implementa un filtro en TrackerScreen.

02/06

- **Front:**
 - Se soluciona un error al cargar las gráficas en TrackerScreen en el que no se carga la gráfica por defecto al renderizar la pantalla. Se implementa la Hook useEffect() para solucionarlo.
 - Se implementa autorefresh en las gráficas de MongoDB para actualización de las gráficas tras modificaciones en el estado.
 - Ligeros retoques en el diseño (centrado de elementos, márgenes).
 - Se realiza limpieza de código.

08/06

- **Front:**
 - Se detecta un bug en la lógica de carga de tiempo de TimerTaskScreen. No se pone los minutos restantes a 0 en la última ronda de Pomodoros. Se trabaja en solucionarlo.
- **Landing:**
 - Se añade enlace al canal de YouTube de miTiempo.

09/06

- **Front:**
 - Se soluciona un bug con la lógica del temporizador indicado en el 08/06.
- **Landing:**
 - Se modifican los márgenes para los enlaces a RRSS.