



TuDu – Aplicación web de gestión de tareas

Proyecto final de Master

Alumno: Desiderio Almansa Porrero

NIF: 05982106J

ESTRUCTURA DEL DOCUMENTO

1.	Introducción	1-0
1.1.	Presentación del proyecto	1-0
1.2.	Objetivos.....	1-0
1.3.	Metodología de trabajo.....	1-1
2.	Análisis de requisitos	2-3
2.1.	Descripción del problema.....	2-3
2.2.	Solución propuesta: Administrador de tareas.....	2-3
2.3.	Perfil del usuario.....	2-5
2.4.	Requerimientos técnicos	2-6
3.	Diseño de la aplicación web	3-9
3.1.	Arquitectura del sistema.....	3-9
3.2.	Diseño de la interfaz de usuario	3-16
3.3.	Diseño de la base de datos.....	3-22
3.4.	Diagrama de flujo de la aplicación	3-24
4.	Implementación	4-25
4.1.	Entorno de desarrollo	4-25
4.2.	Desarrollo del frontend (React con TypeScript y Tailwind).....	4-27
4.3.	Desarrollo del backend (Node.js con Express)	4-40
4.4.	Implementación de la API RESTful	4-49
4.5.	Pruebas y control de calidad	4-62
5.	Planificación del proyecto	5-63
6.	Evaluación y resultados	6-65
6.1.	Funcionalidades implementadas	6-65
6.2.	Usabilidad y experiencia de usuario.....	6-65
6.3.	Rendimiento y escalabilidad.....	6-66
6.4.	Seguridad y privacidad.....	6-66
7.	Conclusiones	7-69
7.1.	Resumen del proyecto	7-69
7.2.	Apunte del trabajo.....	7-69
7.3.	Limitaciones y futuras mejoras	7-70
8.	Anexos.....	8-71

8.1.	Código fuente.....	8-71
8.2.	Manual de usuario.....	8-71
8.3.	Capturas de pantalla	8-93
9.	Bibliografía.....	9-117

1. INTRODUCCIÓN

1.1. Presentación del proyecto

A todos nos ha pasado que hemos llegado a la universidad o al instituto y nos ha tocado hacer algún trabajo con un compañero que no se entera de nada o no se organiza bien, lo que dificultaba la elaboración del mismo. Casos parecidos a este se dan en muchos ámbitos: repartos de tareas del hogar al compartir un piso, desarrollo de un proyecto, etc.

El presente proyecto tiene como objetivo la creación de una aplicación web completa que demuestre las habilidades y conocimientos adquiridos en el master, abarcando tanto el frontend como el backend, así como la integración de servicios y la gestión de bases de datos.

La aplicación desarrollada busca resolver el problema planteado: la administración de tareas sencillas en las que se puede desglosar un proyecto/quehacer, mediante el uso de técnicas de desarrollo ágil, asegurando la escalabilidad, mantenibilidad y eficiencia del producto final.

1.2. Objetivos

Los principales objetivos del proyecto son:

- Identificación de los diferentes modelos de datos que habrán de ser puestos en común para su interoperabilidad, diseñando, por lo tanto, la estrategia de normalización más adecuada.
- Diseñar el flujo de trabajo que seguirá la gestión de tareas, desde su creación hasta su finalización. Este flujo determinará las interfaces de usuario necesarias y las interacciones entre diferentes componentes del sistema.
- Diseñar una interfaz de usuario que permita gestionar las tareas de manera cómoda e intuitiva, mejorando la experiencia del usuario. Esto incluye la creación, edición, eliminación, y visualización de tareas y sus respectivos proyectos.

- Implementar medidas robustas de seguridad para proteger la información sensible de los usuarios, como datos personales y detalles de las tareas. Asegurar que la aplicación cumpla con las normativas de protección de datos y que utilice técnicas de cifrado adecuadas para garantizar la confidencialidad y la integridad de los datos almacenados.

1.3. Metodología de trabajo

En este punto del trabajo se describe el método de trabajo que se ha seguido para el desarrollo del proyecto y el marco tecnológico usado.

- **Metodología Iterativa e Incremental**

La metodología iterativa e incremental consiste en seguir una planificación, para el desarrollo de un proyecto, por partes o bloques, en los que se va añadiendo nuevas funcionalidades y mejoras. Este modelo combina elementos del modelo en cascada, ya que fue desarrollado para reforzar sus debilidades (Caballero, 2019).

Este modelo construye una implementación incompleta de la aplicación y en cada iteración, de una secuencia lineal, se produce un incremento funcional en el desarrollo del producto final. Además de añadir nueva funcionalidad, también es posible modificar otras en caso de que fuera necesario, o bien integrar algunas nuevas que no se hayan definido en una primera instancia, permitiendo identificar nuevas mejoras en las etapas intermedias del desarrollo, lo que evita problemas en los procesos finales (Solano-Fernández, 2020). Cada iteración da como resultado un resultado funcional, ya que se desarrollan todas las fases del requisito y también sus pruebas.

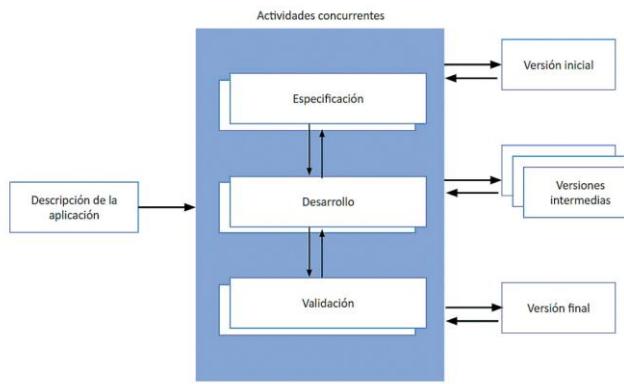


Ilustración 1: Modelo de desarrollo iterativo e incrementa (Solano-Fernández, 2020)

El principal objetivo del modelo es tomar como inicio un conjunto de requerimientos básicos, que se ya se han definido, y a partir de ese inicio, ir incorporando el resto de requerimientos que se han terminado de definir con mayor claridad o que han surgido durante el desarrollo.

Esta metodología también favorece la valoración y el aprovechamiento del aprendizaje que se ha ido realizando durante el desarrollo de las versiones que se van liberando.

Consta de cuatro fases o etapas: planificación, análisis y diseño, implementación y pruebas. Existe otra fase que solo se incluye en ciertos casos, la de evaluación. También dispone de un listado que contiene todas las tareas pendientes de realizar, y cada vez que se identifique una nueva funcionalidad se debe de indicar en ella (lista de control) .



Ilustración 2: Concepto de las fases del desarrollo iterativo e incremental (Diván, 2007)

2. ANÁLISIS DE REQUISITOS

2.1. Descripción del problema

Los principales problemas que se abordan son los siguientes:

- Falta de organización: cuando se pretende llevar a cabo un proyecto, ya sea personal o no, es complicado realizar un correcto seguimiento si no se dividen en tareas y no las centralizamos adecuadamente, ya que podemos ir apuntándolas en papeles sueltos, correos, en las notas del móvil, etc.
- Olvido y procrastinación: esto suele ocurrir más en proyectos individuales como comenzar un entrenamiento o un hábito de estudio para los exámenes que estén por llegar. Las personas tienden a olvidar o postergar tareas y actividades, debido a que no llevan un seguimiento apropiado.
- Colaboración ineficiente: este caso ocurre en proyectos grupales, como, por ejemplo, lo que se ha mencionado previamente, en trabajos escolares. Debido a la falta de comunicación, puede darse el caso de realizar esfuerzos por duplicado y exista una falta de claridad en las responsabilidades que desempeña cada miembro del grupo.
- Carga de trabajo desigual: en proyectos grupales es común que haya personas que completen más tareas que otras, llegando a ser incluso injusto.
- Falta de motivación: sin un sistema visual para ver el progreso de los proyectos o quehaceres es muy fácil perder la motivación.

2.2. Solución propuesta: Administrador de tareas

El proyecto planteado puede resolver los problemas abordados en el anterior punto de la siguiente manera:

- Un administrador de tareas centraliza todas las tareas en un solo lugar, facilitando su acceso y gestión.

- Permitir a los usuarios que se asignen tareas, compartir sus actualizaciones y dejar comentarios en ellas permite que todos ellos trabajen alineados y al tanto de sus responsabilidades.
- Definir un administrador y unos colaboradores permite distribuir de manera equitativa y eficiente la carga de trabajo.
- Permitir a los usuarios establecer metas claras, dividir proyectos grandes en tareas simples y manejables, permite que los usuarios tengan motivación y satisfacción al completarlas.

- **Funcionalidades**

1. Crear usuarios.
 - a. Confirmar usuario a través de correo y token de seguridad.
 - b. Y su edición a través del apartado de "Mi perfil".
2. Iniciar sesión y cerrar sesión.
3. Reestablecer contraseña.
 - a. A través de correo y token de seguridad en caso de olvido.
 - b. A través del apartado de "Mi perfil"
4. Visualizar el dashboard de los proyectos en los que participa.
5. Crear proyectos nuevos, en los que el usuario creador desempeña el rol de MANAGER.
6. Agregar o eliminar usuarios COLABORADORES a los proyectos en los que se es MANAGER.
7. Editar o eliminar aquellos proyectos en los que se es MANAGER.
8. Visualizar los proyectos en los que se participe.
9. Agregar tareas a los proyectos en los que se es MANAGER.

10. Editar o eliminar tareas si se es MANAGER.
11. Visualizar la tarea del proyecto en el que se participa.
12. Modificar el estado de la tarea y visualizar el historial de cambios.
13. Añadir, editar o eliminar comentarios en las tareas.
14. Dar feedback en cada acción realizada por el usuario para indicar éxito o error.

- **Roles de usuario**

MANAGER: usuario que crea el proyecto. Tiene permitido su edición y eliminación. Puede añadir tareas, editarlas, eliminarlas y visualizarlas, además de dejar comentarios en ellas. Puede agregar o eliminar COLABORADORES de los proyectos.

COLABORADOR: usuario al que han agregado a un proyecto. Tiene permitido visualizar el proyecto. Puede ver las tareas, dejar notas en ellas y modificar su estado.

2.3. Perfil del usuario

El perfil de usuario para el que está enfocada la aplicación comprende edades entre 15 y 60 años, pudiendo usarse tanto para el instituto, universidad o trabajo, con conocimientos básicos de informática hasta profesionales del sector.

El entorno de trabajo correspondería con centros educativos, oficinas o trabajo remoto.

Un perfil regular usaría la aplicación para tareas personales, a modo de recordatorio y seguimiento. Un usuario profesional la usaría para organizar a su propio equipo de trabajo y simplificar proyectos en tareas.

2.4. Requerimientos técnicos

Se ha seguido el marco tecnológico MERN (Erickson, 2024). MERN es un acrónimo del conjunto de tecnologías que lo componen: MongoDB, Express, React y Node.js. A continuación, veremos todas las tecnologías utilizadas:

- **Frontend:**

- React: biblioteca de JavaScript para construir interfaces de usuario. Versión Requerida: 17.x o superior. Consideraciones: Utilizar componentes funcionales y hooks para manejar el estado y los efectos secundarios. (React, 2024)
- TypeScript: lenguaje de programación que extiende JavaScript añadiendo tipos estáticos. Versión Requerida: 4.x o superior. Consideraciones: Configuración de tsconfig.json para optimizar el proyecto, asegurando el uso de tipado estático para evitar errores en tiempo de compilación. (TypeScript, 2024)
- CSS y Tailwind: Tailwind CSS es un framework de CSS utility-first. Versión Requerida de Tailwind: 2.x o superior. Consideraciones: Seguir las mejores prácticas de Tailwind para estilizar los componentes de manera eficiente y mantener el CSS modular y reutilizable. (Tailwind, 2024)
- Patrón de diseño basado en componentes, layouts y hooks (React).

- **Backend:**

- Node.js con Express: Node.js es un entorno de ejecución para JavaScript en el servidor, y Express es un framework minimalista para Node.js. Versión Requerida de Node.js: 14.x o superior. Versión Requerida de Express: 4.x o superior. Consideraciones: Utilizar middleware para la gestión de rutas, autenticación, y manejo de errores. Implementar controladores y servicios para organizar la lógica del negocio. (Express, 2024)

- MongoDB: Base de datos NoSQL orientada a documentos. Versión Requerida: 4.x o superior. Consideraciones: Diseñar los esquemas de datos usando Mongoose para la validación y estructuración de datos. (Se ha usado MongoDB Compass para visualizar la Base de datos). (Mongo, 2024)
- Patrón de diseño MVC (Modelo Vista Controlador) y modularización.

- **Control de versiones:**

- GitHub: Plataforma para control de versiones y colaboración en código. Consideraciones: Configurar repositorio para el frontend y backend. Utilizar ramas para desarrollo, pruebas y producción. Implementar pull requests y revisiones de código para mantener la calidad del código.

- **Entorno de desarrollo:**

- Visual Studio Code: Editor de código fuente. Consideraciones: Configurar extensiones recomendadas como Prettier, Tailwind CSS IntelliSense, entre otros para mejorar la productividad y mantener la consistencia del código. Configurar tareas y scripts en package.json para automatizar tareas comunes como la compilación y el linting.

- **Testing:**

- Thunder Client para Backend: Extensión de Visual Studio Code para testing de API Rest. (ProductHunt, s.f.)

- **Requerimientos específicos para funcionalidades de la aplicación:**
 - Autenticación y autorización haciendo uso de JSON Web Token.
Implementación de un middleware para verificar el token.
 - Hasheo de contraseñas al guardar en base de datos, usando bcrypt.
 - Toastify para notificaciones de feedback. (AP, 2023).
- **Otros:**
 - Photosop para el diseño del logo.
 - Mailtrap para crear emails ficticios temporales y poder dar de alta usuarios en la aplicación. (Rilsware, s.f.)
 - Balsamiq para diseñar bocetos de la interfaz de usuario. (Mockup, s.f.)

3. DISEÑO DE LA APLICACIÓN WEB

3.1. Arquitectura del sistema

Como se menciona en el marco tecnológico (apartado 2.5.), se ha utilizado el patrón de arquitectura MVC. Este patrón es un paradigma de diseño de software que separa una aplicación en tres componentes interconectados, promoviendo una separación de responsabilidades y facilitando el mantenimiento del sistema (Wikipedia, s.f.). Diferenciamos tres componentes:

Modelo: encargado de gestionar los datos y la lógica de negocio. Se encarga de recuperar, almacenar y acutalizar los datos, ya sea de una base de datos, archivos u otros medios de almacenamiento. Implementa las reglas de negocio y el procesamiento de los datos necesarios para que la aplicación funcione.

Controlador: actúa de intermediario entre el Modelo y la Vista. Se encarga de gestionar la lógica de flujo de la aplicación. Recibe las entradas del usuario a través de la vista, para posteriormente procesarlas, invocando métodos del modelo para actualizar los datos.

Vista: se encarga de presentar los datos al usuario e interacciona directamente con él. Muestra los datos que recibe del modelo de forma que el usuario pueda entenderlos. Captura las entradas y las envía al controlador para ser procesadas y, tras la respuesta de este, se actualiza reflejando el estado actual de los datos.

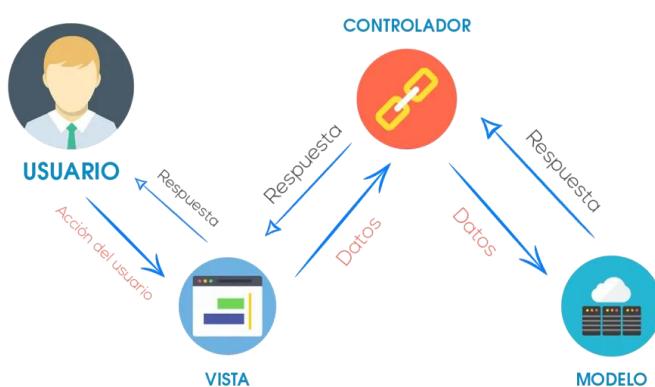


Ilustración 3: Flujo del modelo MVC (Precognis, 2022)

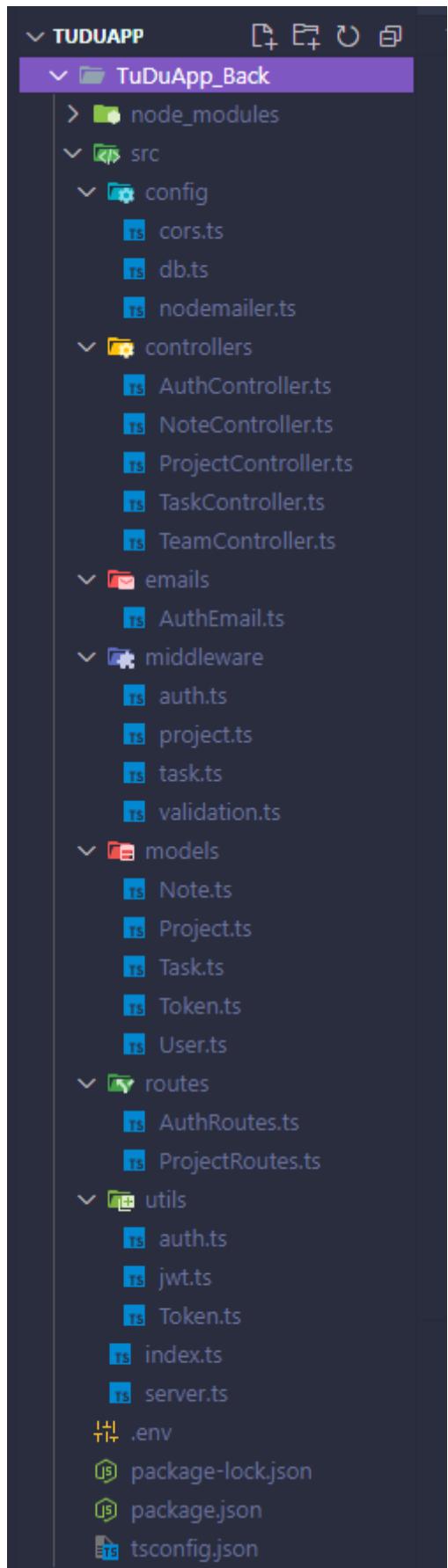
El proyecto desarrollado está dividido en Frontend y Backend, por lo que se ha dividido en dos proyectos TuDuApp_Back y TuDuApp_Front. En TuDuApp_Bakc encontramos el Modelo y el Controlador, y en TuDuApp_Front la Vista.

A continuación, se explica brevemente la estructura de carpetas:

- **TuDApp_Back:** En su raíz se encuentran los archivos de configuración de TypeScript y de los paquetes, además se ha añadido un archivo “.env” que contiene las variables de entorno que se han definido.
En él se encuentra la carpeta “src” que contiene todo el código desarrollado (explicado en la sección 4.3.):
 - server.ts: punto de entrada principal de la aplicación servidor. Configura y arranca el servidor.
 - index.ts: complementario a server.ts. Se encarga de importar el servidor, configurar el puerto en el que escuchará y manejar los errores.
 - Carpeta “config”: contiene archivos de configuración de CORS (White list para poder usar la API), de la base de datos (conexión) y de los envíos de correo.
 - Carpeta “controllers”: contiene los distintos controladores necesarios para la gestión de datos que se almacenan en nuestra base de datos. En ellos se definen los métodos que se ejecutarán cuando se llame a los endpoints de la API.
 - Carpeta “emails”: contiene el archivo AuthEmail.ts en el que se implementan los métodos que se encargan de enviar los emails de confirmación y de recuperación de contraseñas.
 - Carpeta “middleware”: como su propio nombre indica, contiene los middlewares de validación de nuestra API, entre ellos destaca el de

Autenticación, que se encarga de dar autorización a los usuarios para realizar determinadas acciones.

- Carpeta “Models”: en ella se encuentran los modelos de la base de datos utilizados.
- Carpeta “routes”: contiene los ficheros de configuración de rutas. Hay dos: AuthRoutes.ts, que contiene las rutas relacionadas con usuarios (inicio de sesión, crear cuenta, acceder a mi perfil, etc), y ProjectRoutes.ts, que contiene las rutas relacionadas con las distintas funcionalidades de proyectos y tareas (crear proyectos y tareas, agregar colaboradores, crear notas, etc).
- Carpeta “utils”: contiene varios ficheros con métodos auxiliares. Contiene auth.ts con métodos de hasheo y comprobación de contraseñas, jwt.ts con el método de generar el token JWT y Token.ts con el método que nos genera el código de 6 dígitos para verificación y recuperación de cuenta.



- **TuDuApp_Front:** al igual que en el backend, en la raíz del proyecto de frontend podemos observar diversos archivos de configuración de TailWind y TypeScript, además se ha añadido un archivo “.env.local” que contiene variables de entorno. También podemos encontrar el index.html, la carpeta public, que contiene el logo, y la carpeta “src” que contiene todo el código desarrollado, en la que destacan:
 - router.tsx: en el definimos todas las rutas que se utilizarán para la aplicación y sus respectivas vistas.
 - main.tsx: es el archivo principal del proyecto, encargado de renderizar las vistas definidas en el Router.
 - Carpeta “views”: contiene los componentes principales que representan las páginas completas de la aplicación. En ella se encuentran subcarpetas que clasifican las vistas en: 404 (error de no encontrado), auth (vistas relacionadas con la autenticación), profile (vistas relacionadas con el perfil del usuario), projects (vistas relacionadas con la gestión de proyectos y sus tareas) y por último el Dashboard (vista principal de la aplicación).
 - Carpeta “types”: contiene un archivo “index.ts” en el que se definen los Esquemas y el tipado de los datos que se manejarán en el resto de componentes del front.
 - Carpeta “services”: en ella se encuentran divididos en distintas clases los métodos encargados de comunicarse con los controladores definidos en el back. Se dividen en AuthService, NoteService, ProfileService, ProjectService, TaskService y TeamService.
 - Carpeta “locales”: únicamente tiene el archivo es.ts que nos sirve para traducir los estados de las tareas al español.
 - Carpeta “lib”: ahí encontramos el archivo “axios.ts” que contiene el código necesario para definir el servicio que comunicará nuestro frontend con el backend. Este servicio se usa en los archivos de la carpeta services para enviar las peticiones y recibir la respuesta.

- Carpeta “utils”: al igual que en el back, esta carpeta contiene ficheros con métodos auxiliares.
- Carpeta “layouts”: en ella se definen los distintos diseños o maquetaciones que definen la estructura general de la aplicación.
- Carpeta “hooks”: en el definimos los hooks, que nos sirven para encapsular lógica y reutilizarla donde se requiera, evitando código repetido.
- Carpeta “components”: en ella se encuentran divididos por sus carpetas los ficheros en los que definimos los distintos modales, formularios o paneles que se usarán en las vistas.

File tree for TuDuApp_Front:

- dist
- node_modules
- public
 - tudu_logo_morado.png
- src
 - components
 - auth
 - NewPasswordForm.tsx
 - NewPasswordToken.tsx
 - notes
 - AddNoteForm.tsx
 - NoteDetail.tsx
 - NotesPanel.tsx
 - profile
 - ProfileForm.tsx
 - Tabs.tsx
 - projects
 - DeleteProjectModal.tsx
 - EditProjectForm.tsx
 - ProjectForm.tsx
 - tasks
 - AddTaskModal.tsx
 - DropTask.tsx
 - EditTaskData.tsx
 - EditTaskModal.tsx
 - TaskCard.tsx
 - TaskForm.tsx
 - TaskList.tsx
 - TaskModalDetails.tsx
 - team
 - ErrorMessage.tsx
 - Logo.tsx
 - NavMenu.tsx
 - hooks
 - useAuth.ts

File tree for the main application directory:

- layouts
 - AppLayout.tsx
 - AuthLayout.tsx
 - ProfileLayout.tsx
- lib
 - axios.ts
- locales
 - es.ts
- services
 - AuthService.ts
 - NoteService.ts
 - ProfileService.ts
 - ProjectService.ts
 - TaskService.ts
 - TeamService.ts
- types
 - index.ts
- utils
 - policies.ts
 - utils.ts
- views
 - 404
 - NotFound.tsx
 - auth
 - ConfirmAccountView....
 - ForgotPasswordView.t...
 - LoginView.tsx
 - NewPasswordView.tsx
 - RegisterView.tsx
 - RequestNewCodeVie...
 - profile
 - ChangePasswordView...
 - ProfileView.tsx
 - projects
 - CreateProjectView.tsx
 - EditProjectView.tsx
 - ProjectDetailsView.tsx

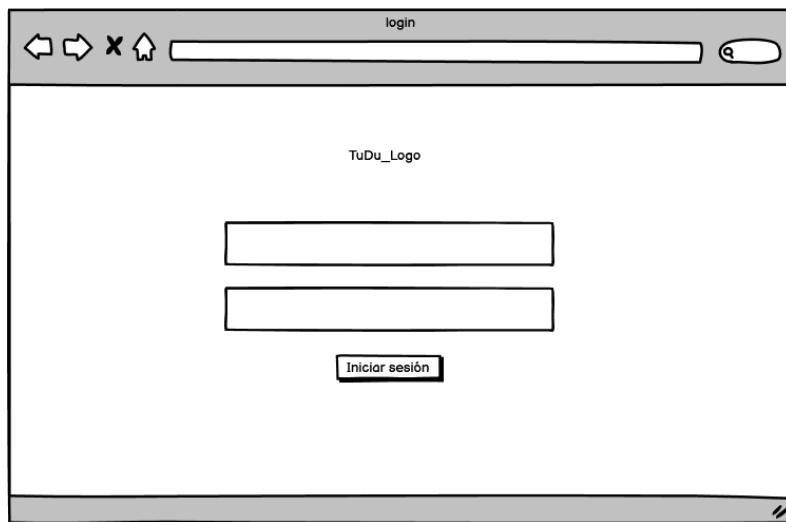
File tree for the vite configuration and other files:

- ProjectTeamView.tsx
- DashboardView.tsx
- index.css
- main.tsx
- router.tsx
- vite-env.d.ts
- .env.local
- .eslintrc.cjs
- .gitignore
- index.html
- package-lock.json
- package.json
- postcss.config.js
- README.md
- tailwind.config.js
- tsconfig.app.json
- tsconfig.json
- tsconfig.node.json
- vite.config.ts
- README.md

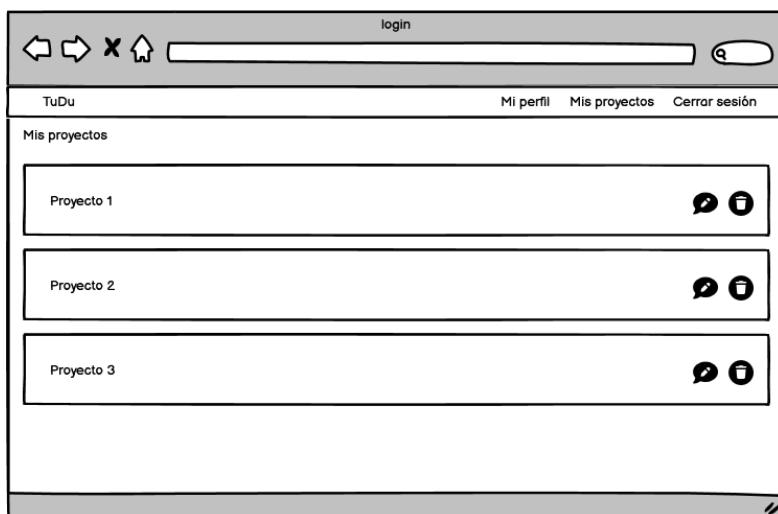
3.2. Diseño de la interfaz de usuario

Para el diseño de la interfaz de usuario se ha usado la aplicación Balsamiq. Esta aplicación nos permite elaborar bocetos de los diseños que tenemos en mente. Para este proyecto se han realizado los bocetos de las vistas principales, antes de comenzar con el desarrollo de la interfaz.

- Boceto vista de Inicio de sesión:



- Boceto vista de Mis proyectos:



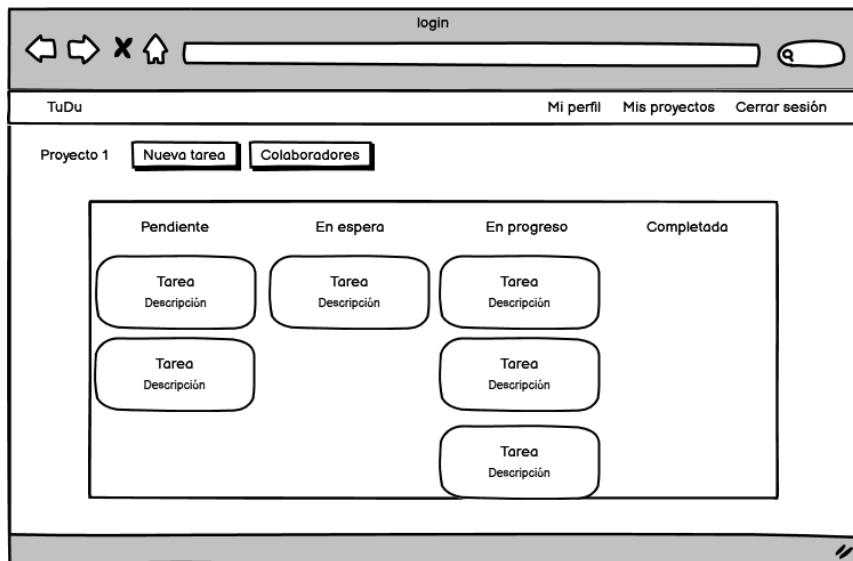
- Boceto vista de Crear proyecto:

The wireframe shows a top navigation bar with 'login' and search icons. Below it is a header with 'TuDu' and navigation links: 'Mi perfil', 'Mis proyectos', and 'Cerrar sesión'. The main content area is titled 'Crear proyecto' and contains three input fields labeled 'Nombre del proyecto', 'Nombre del cliente', and 'Descripción', followed by a 'Crear' button.

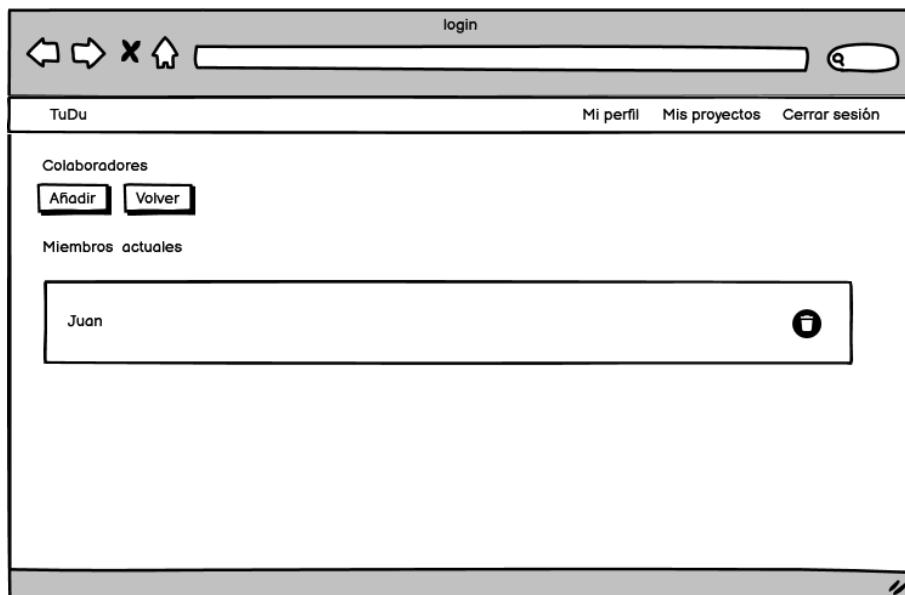
- Boceto vista de Editar proyecto:

The wireframe is identical to the 'Crear proyecto' page, showing the same top navigation, header, and form fields for project name, client name, description, and a 'Guardar cambios' (Save changes) button.

- Boceto vista de Proyecto:



- Boceto vista de Colaboradores:



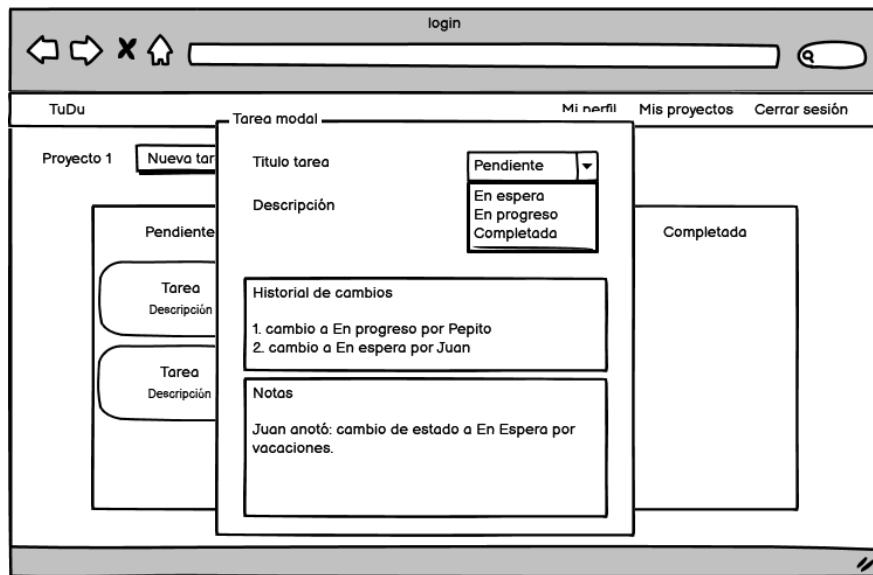
- Boceto vista de Añadir colaborador:

This wireframe shows the 'Colaboradores' (Collaborators) section of the TuDu application. At the top left are 'Añadir' and 'Volver' buttons. Below them is a list titled 'Miembros actuales' (Current members) containing a single entry 'Juan'. To the right is a form titled 'Añadir colaborador' (Add collaborator) with fields for 'Email del usuario' (User email), 'Buscar' (Search), and 'Usuario buscado' (User found). A '+' button is located next to the search field. At the bottom right of the page is a trash can icon.

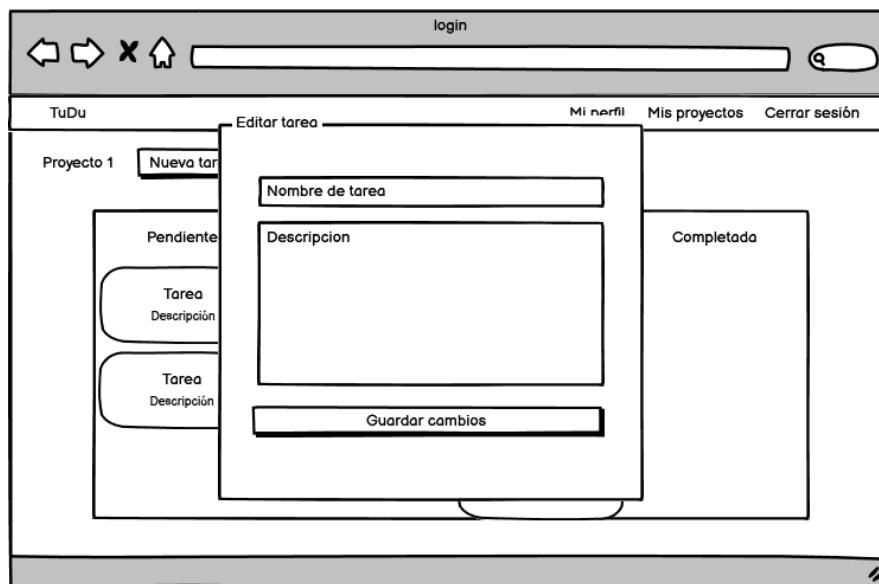
- Boceto vista de Crear tarea:

This wireframe shows the 'Crear tarea' (Create task) view. On the left, there's a sidebar for 'Proyecto 1' (Project 1) with a 'Nueva tarea' (New task) button, and two task items: 'Pendiente' (Pending) and 'Completada' (Completed). The main area has a 'Crear tarea' (Create task) title. It contains fields for 'Nombre de tarea' (Task name) and 'Descripción' (Description). A 'Crear' (Create) button is at the bottom. To the right of the task creation form is a large empty box.

- Boceto vista de Tarea:



- Boceto vista de Editar tarea:

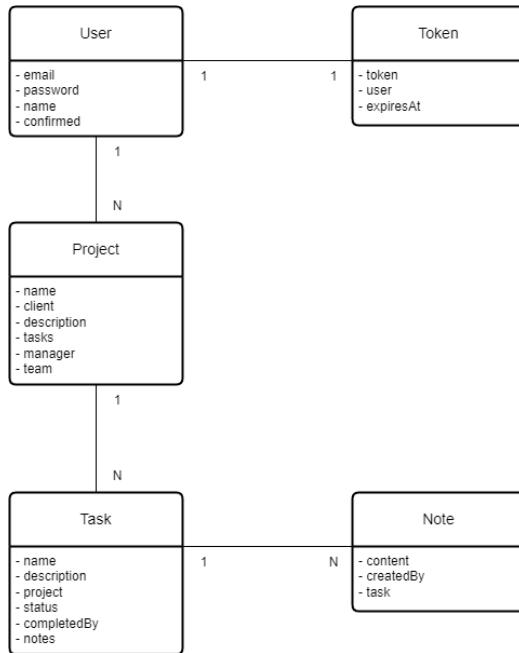


- Boceto vista de Mi perfil:

The wireframe depicts the 'Mi perfil' (My Profile) section of a web application. At the top, there's a header bar with icons for back, forward, search, and refresh, followed by the word 'login'. Below this is a navigation menu with 'TuDu' on the left and links for 'Mi perfil', 'Mis proyectos', and 'Cerrar sesión' (Logout) on the right. The main content area is titled 'Mi perfil' and contains two input fields: 'Nombre' (Name) and 'Email'. Below these fields are two buttons: 'Editar' (Edit) and 'Cambiar contraseña' (Change Password). A horizontal scrollbar is located at the bottom right of the content area.

3.3. Diseño de la base de datos

Para el diseño de la base de datos se ha utilizado un modelo no relacional, ya que se ha usado MongoDB. Para mayor compresión, en el siguiente esquema se representan los modelos con sus atributos y como se relacionarían si se hubiese usado un modelo relacional:



Descripción de los modelos:

- **Project**

Representa los proyectos o quehaceres que se quieren organizar a través de TuDu. Sus atributos son:

- name: nombre del proyecto.
- client: nombre del cliente para el que vamos a realizar el proyecto.
- description: descripción del proyecto.
- tasks: array de identificadores de las tareas que pertenecen al proyecto.
- manager: identificador del usuario que creó el proyecto (desempeñará el rol de MANAGER en él)
- team: array de identificadores de los usuarios que colaborarán en el proyecto (desempeñarán el rol de COLABORADOR en él)

- **User**

Representa los usuarios que utilizarán la aplicación. Sus atributos son:

- email: correo electrónico con el que se iniciará sesión y al que se le enviarán los tokens de verificación de cuenta y recuperación de contraseña.
- password: contraseña que se usará junto con el email para iniciar sesión y para confirmar la eliminación de proyectos. Irá hasheada para mayor seguridad.
- name: nombre del usuario.
- confirmed: indica si el usuario ha confirmado su email (true) o no (false).

- **Task**

Representa las tareas en las que se irán subdividiendo los proyectos. Sus atributos son:

- name: nombre de la tarea.
- description: descripción breve de qué consiste la tarea.
- Project: proyecto al que pertenece la tarea.
- status: estado en el que se encuentra la tarea (pending, onHold, inProgress, completed)
- completedBy: array de tuplas [id user, status] en el que almacenamos los cambios producidos en la tarea y quién los ha realizado para mostrar un historial.
- notes: array de identificadores de las notas que pertenecen a la tarea.

- **Note**

Representa los comentarios o notas que los usuarios pueden ir dejando en las tareas a medida que van realizándolas. Sus atributos son:

- content: se trata del contenido de la nota, el texto que el usuario escribirá.
- createdBy: identificador del usuario que creó la nota.
- task: identificador de la tarea a la que pertenece.

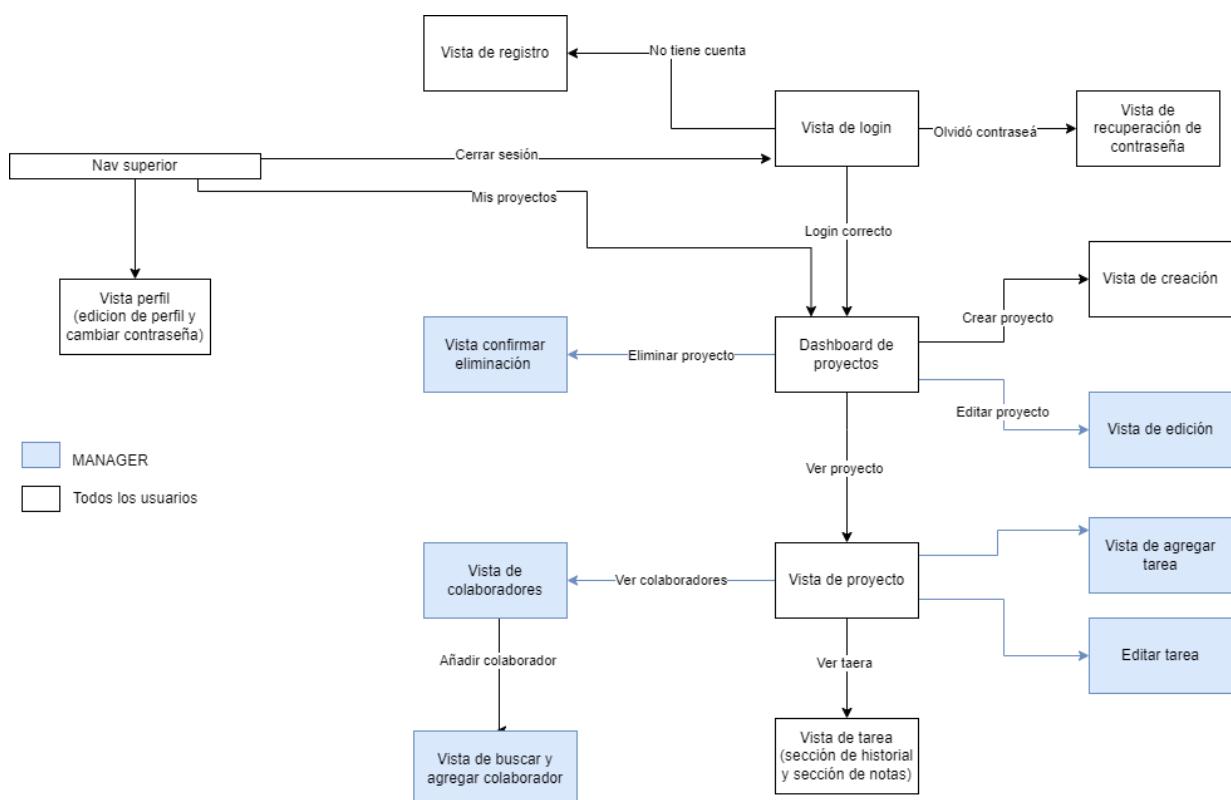
- o **Token**

Se trata de la clave de confirmación de cuenta que se envía al usuario tras registrarse o tras solicitar recuperar la contraseña en caso de olvido.

Sus atributos son:

- Token: código de 6 dígitos generado aleatoriamente.
- User: identificador del usuario al que pertenece.
- expiresAt: fecha en la que expirará el token y se eliminará de la base de datos automáticamente.

3.4. Diagrama de flujo de la aplicación



4. IMPLEMENTACIÓN

4.1. Entorno de desarrollo

Para poder desarrollar el proyecto se ha usado Visual Studio Code, como se ha mencionado en apartados anteriores. Este entorno de desarrollo cuenta con la capacidad de instalar plugins que mejoran considerablemente la experiencia de desarrollo y facilita mucho el mismo. Se han usado los siguientes:

- DotENV: Facilita la gestión de archivos .env en Visual Studio Code, proporcionando resaltado de sintaxis y autocompletado para variables de entorno.
- Tailwind CSS IntelliSense: Proporciona autocompletado, sugerencias y verificación de clases de Tailwind CSS para mejorar la productividad y precisión en el desarrollo frontend.
- Live Preview: Permite visualizar y actualizar en tiempo real los cambios realizados en archivos HTML y CSS dentro del editor.
- Thunder Client: Una alternativa ligera a Postman integrada en Visual Studio Code, diseñada para probar y realizar peticiones HTTP directamente desde el editor.
- Code Runner: Ejecuta rápidamente fragmentos de código en múltiples lenguajes de programación dentro de Visual Studio Code.
- Vscode-icons: Añade iconos visualmente atractivos y reconocibles para diferentes tipos de archivos y carpetas en Visual Studio Code, mejorando la navegación y organización del proyecto.

Por otro lado, se han instalado diversas dependencias tanto en frontend como en backend que nos proporcionan múltiples herramientas de desarrollo.

- Dependencias destacables de Backend:
 - Bcrypt: Biblioteca para hashing de contraseñas, proporcionando funciones para generar y comparar hashes de manera segura.
 - Colors: Biblioteca para agregar colores y estilos a la consola en Node.js, mejorando la legibilidad de los logs.
 - Cors: Middleware para habilitar CORS (Cross-Origin Resource Sharing) en aplicaciones Express, permitiendo o restringiendo peticiones de dominios externos.

- Dotenv: Herramienta para cargar variables de entorno desde un archivo .env en process.env, facilitando la configuración del entorno.
 - Express: Framework web minimalista y flexible para Node.js, utilizado para construir aplicaciones y APIs robustas.
 - Express-validator: Conjunto de middlewares para validar y sanitizar datos de entrada en aplicaciones Express, asegurando la integridad y seguridad de los datos.
 - Jsonwebtoken: Biblioteca para crear y verificar JSON Web Tokens (JWT), utilizados para la autenticación y autorización segura de usuarios.
 - Mongoose: Librería de modelado de objetos (ODM) para MongoDB y Node.js, proporcionando una solución estructurada para interactuar con bases de datos MongoDB.
 - Morgan: Middleware para generar logs HTTP en aplicaciones Express, ayudando en la monitorización y depuración de solicitudes.
 - Nodemailer: Biblioteca para enviar correos electrónicos desde aplicaciones Node.js, soportando múltiples servicios de email y métodos de autenticación.
 - Tipados de las dependencias que ayudan al autocompleteo entre otras cosas.
- Dependencias destacables de Frontend:
- Tipados de algunas de las dependencias mencionadas.
 - @chakra-ui/pin-input: un componente de Chakra UI para crear inputs de PIN con soporte para accesibilidad y personalización.
 - @dnd-kit/core: una biblioteca de arrastrar y soltar (drag and drop) altamente personalizable y de bajo nivel para React.
 - @headlessui/react: componentes accesibles y totalmente desprovistos de estilos para React, diseñados para integrarse fácilmente con cualquier CSS.
 - @heroicons/react: un conjunto de iconos SVG gratuitos y de código abierto, optimizados para proyectos de React.
 - @tailwindcss/forms: un complemento para Tailwind CSS que proporciona estilos prediseñados y consistentes para elementos de formulario.
 - @tanstack/react-query: una biblioteca para la gestión de estados asincrónicos en React, facilitando el manejo de datos remotos y la sincronización de servidores.

- @tanstack/react-query-devtools: herramientas de desarrollo para React Query, que permiten la inspección y depuración de estados de consulta.
- Axios: una biblioteca para realizar peticiones HTTP en JavaScript, popular por su simplicidad y soporte para promesas.
- React: una biblioteca de JavaScript para construir interfaces de usuario basadas en componentes.
- React-dom: proporciona métodos específicos del DOM que permiten que React se relacione con el navegador.
- React-hook-form: una biblioteca para manejar formularios en React con validación, integración con resolvers de esquemas y mejor rendimiento.
- React-router-dom: una colección de componentes de navegación dinámica para aplicaciones web React.
- React-toastify: una biblioteca para mostrar notificaciones no intrusivas en aplicaciones React.
- Zod: una biblioteca de validación de esquemas que se integra bien con TypeScript para asegurar la validación de datos.
- Vite: un constructor de aplicaciones web que proporciona un entorno de desarrollo rápido y optimizado, con soporte para React y otros marcos.
- Tailwindcss: Un framework CSS utilitario que permite construir interfaces de usuario rápidamente con clases de utilidad altamente configurables.
- Typescript: un superconjunto tipado de JavaScript que añade tipado estático opcional y otras características avanzadas de programación.

4.2. Desarrollo del frontend (React con TypeScript y Tailwind)

El frontend se encuentra dividido en varios modulos que recogen los componentes, los hooks, los layouts, los servicios, las vistas, los utils y la configuración de axios.

En la raíz del proyecto, se encuentra el index.html que renderizará todo a través del main.tsx.

```
TuDuApp_Front > index.html > ...
1  <!doctype html>
2  <html lang="en">
3  |  <head>
4  |  |  <meta charset="UTF-8" />
5  |  |  <link rel="icon" type="image/svg+xml" href="/vite.svg" />
6  |  |  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7  |  |  <title>TuDuApp - TFM Desiderio Almansa Porrero - Master FullStack</title>
8  |  </head>
9  |  <body class="bg-gray-100">
10 |  |  <div id="root"></div>
11 |  |  <script type="module" src="/src/main.tsx"></script>
12 |  </body>
13 </html>
```

El resto de código a explicar se encuentra en la carpeta src.

- main.tsx

```
TuDuApp_Front > src > main.tsx > ...
1  import React from 'react'
2  import ReactDOM from 'react-dom/client'
3  import { QueryClient, QueryClientProvider } from '@tanstack/react-query'
4  import { ReactQueryDevtools } from '@tanstack/react-query-devtools'
5  import './index.css'
6  import Router from './router'
7
8  const queryClient = new QueryClient()
9
10 ReactDOM.createRoot(document.getElementById('root')!).render(
11   <React.StrictMode>
12     <QueryClientProvider client={queryClient}>
13       <Router/>
14       <ReactQueryDevtools/>
15     </QueryClientProvider>
16   </React.StrictMode>,
17 )
```

En este fichero se crea un queryClient haciendo uso de react-query, que nos permitirá gestionar el estado de los asíncronos y la caché.

Haciendo uso de ReactDOM.createRoot, se inicializa una nueva aplicación de react y la monta haciendo uso del id "root".

El componente QuertyClientProvider hace que la instancia de queryClient esté disponible en toda la aplicación y nos permite usar hooks de react-query para realizar las consultas y gestionar los datos en componentes hijos.

El componente Router define las rutas de la aplicación. Gracias al se maneja la navegación y errutamiento.

ReactQueryDevTools es un componente que se añadió para visualizar las consultas, mutaciones y cache de la aplicación mientras se desarrollaba.

- router.tsx

```
TuDuApp_Front > src > router.tsx > Router
14 import ProjectTeamView from './views/projects/ProjectTeamView'
15 import ProfileView from './views/profile/ProfileView'
16 import ChangePasswordView from './views/profile/ChangePasswordView'
17 import ProfileLayout from './layouts/ProfileLayout'
18 import NotFound from './views/404/NotFound'
19
20 export default function Router() {
21
22     return (
23         <BrowserRouter>
24             <Routes>
25                 <Route element={<AppLayout/>}>
26                     <Route path='/' element={<DashboardView/>} index />
27                     <Route path='/projects/create' element={<CreateProjectView/>} />
28                     <Route path='/projects/:projectId' element={<ProjectDetailsView/>} />
29                     <Route path='/projects/:projectId/edit' element={<EditProjectView/>} />
30                     <Route path='/projects/:projectId/team' element={<ProjectTeamView/>} />
31
32                     <Route element={<ProfileLayout />}>
33                         <Route path='/profile' element={<ProfileView />} />
34                         <Route path='/profile/password' element={<ChangePasswordView />} />
35                     </Route>
36
37                 </Route>
38
39                 <Route element={<AuthLayout/>}>
40                     <Route path='/auth/login' element={<LoginView/>} />
41                     <Route path='/auth/register' element={<RegisterView/>} />
42                     <Route path='/auth/confirm-account' element={<ConfirmAccountView/>} />
43                     <Route path='/auth/request-code' element={<RequestNewCodeView/>} />
44                     <Route path='/auth/forgot-password' element={<ForgotPasswordView/>} />
45                     <Route path='/auth/new-password' element={<NewPasswordView/>} />
46                 </Route>
47
48                 <Route element={<AuthLayout/>}>
49                     <Route path='*' element={<NotFound/>}/>
50                 </Route>
51             </Routes>
52         </BrowserRouter>
53     )
54 }
```

En el se definen las rutas del navegador, diferenciando 3 elementos de enrutado (Route) que hacen referencia a 2 layouts:

- Un enrutado para AppLayout: se establecen rutas para las vistas relacionadas con los proyectos y para el ProfileLayout, que a su vez tiene rutas definidas dentro.

- Un enrutado para AuthLayout: se establecen rutas para las vistas relacionadas con inicio de sesión y registro de usuarios.
- Otro enrutado para la página de NotFound (se usa el AuthLayout).

▪ Tipados

En el fichero “types/index.ts” se definen los esquemas de validación y tipado utilizando zod. Estos esquemas y tipos se usan para modelar y validar los datos que se manejan en todo el frontend de la aplicación y en el servicio que usa.

```
TuDuApp_Front > src > types > index.ts > [?] dashboardProjectSchema
1 import { z } from 'zod'
2
3 /** Auth and Users */
4 const authSchema = z.object({
5   name: z.string(),
6   email: z.string().email(),
7   current_password: z.string(),
8   password: z.string(),
9   password_confirmation: z.string(),
10  token: z.string()
11 })
12
13 type Auth = z.infer<typeof authSchema>
14 export type UserLoginForm = Pick<Auth, 'email' | 'password'>
15 export type UserRegistrationForm = Pick<Auth, 'name' | 'email' | 'password' | 'password_confirmation'>
16 export type RequestConfirmationCodeForm = Pick<Auth, 'email'>
17 export type ForgotPasswordForm = Pick<Auth, 'email'>
18 export type NewPasswordForm = Pick<Auth, 'password' | 'password_confirmation'>
19 export type UpdateCurrentUserPasswordForm = Pick<Auth, 'current_password' | 'password' | 'password_confirmation'>
20 export type ConfirmToken = Pick<Auth, 'token'>
21 export type CheckPasswordForm = Pick<Auth, 'password'>
22
23
24 /** User */
25 export const userSchema = authSchema.pick({
26   name: true,
27   email: true
28 }).extend({
29   _id: z.string()
30 })
31 export type User = z.infer<typeof userSchema>
32 export type UserProfileForm = Pick<User, 'name' | 'email'>
33
34
35 /** Notes */
36 const noteSchema = z.object({
37   _id: z.string(),
38   content: z.string(),
39   createdBy: userSchema,
40   task: z.string(),
41   createdAt: z.string()
42 })
43 export type Note = z.infer<typeof noteSchema>
44 export type NoteFormData = Pick<Note, 'content'>
```

```
46  /** Task */
47  export const taskStatusSchema = z.enum(["pending", "onHold", "inProgress", "completed"])
48  export type TaskStatus = z.infer<typeof taskStatusSchema>
49
50  export const taskSchema = z.object({
51    _id: z.string(),
52    name: z.string(),
53    project: z.string(),
54    description: z.string(),
55    status: taskStatusSchema,
56    createdAt: z.string(),
57    updatedAt: z.string(),
58    completedBy: z.array(z.object({
59      _id: z.string(),
60      user: userSchema,
61      status: taskStatusSchema
62    })),
63    notes: z.array(noteSchema.extend({
64      createdBy: userSchema
65    }))
66  })
67
68  export const taskProjectSchema = taskSchema.pick({
69    _id: true,
70    name: true,
71    description: true,
72    status: true
73 })
74
75  export type Task = z.infer<typeof taskSchema>
76  export type TaskFormData = Pick<Task, 'name' | 'description'>
77  export type TaskProject = z.infer<typeof taskProjectSchema>
78
```

```
79  /** Project */
80
81  export const projectSchema = z.object({
82    _id: z.string(),
83    name: z.string(),
84    client: z.string(),
85    description: z.string(),
86    manager: z.string(userSchema.pick({ _id: true })),
87    tasks: z.array(taskProjectSchema),
88    team: z.array(z.string(userSchema.pick({ _id: true })))
89  })
90
91  export const dashboardProjectSchema = z.array(
92    projectSchema.pick({
93      _id: true,
94      name: true,
95      client: true,
96      description: true,
97      manager: true
98    })
99  )
100
101 export const editProjectSchema = projectSchema.pick({
102   name: true,
103   client: true,
104   description: true
105 })
```

```

106  export type Project = z.infer<typeof projectSchema>
107  export type ProjectFormData = Pick<Project, 'name' | 'client' | 'description'>
108
109  /** Team */
110  const teamMemberSchema = userSchema.pick({
111    name: true,
112    email: true,
113    _id: true
114  })
115  export const teamMembersSchema = z.array(teamMemberSchema)
116  export type TeamMember = z.infer<typeof teamMemberSchema>
117  export type TeamMemberForm = Pick<TeamMember, 'email'>

```

Haciendo uso de la función inferir (z.infer) y de Pick a partir de un mismo esquema o tipo, podemos crear otros tipos con los campos que queramos. De esta forma si alguno de los tipos principales crece, no es necesario actualizar ningún componente ya que esta utilizando los “subtipos” del mismo.

- Servicio de API

En el archivo “lib/axios.ts” se crea el servicio que nos servirá para enviar las peticiones al backend gracias a la librería axios.

```

TuDuApp_Front > src > lib >  axios.ts > [?] default
1  import axios from "axios"
2
3
4  const service = axios.create({
5    baseURL: import.meta.env.VITE_API_URL
6  })
7
8  service.interceptors.request.use(config => {
9    const tokenJWT = localStorage.getItem('AUTH_TOKEN_TUDUAPP')
10   if(tokenJWT){
11     config.headers.Authorization = `Bearer ${tokenJWT}`
12   }
13   return config
14 })
15
16  export default service

```

Establecemos la url base que almacenamos en una variable de entorno y configuramos un interceptor del servicio que nos añadira el jwt token almacenado en el localstorage del navegador a la cabecera Authentication de nuestras requests.

Este servicio se usa en los llamados “services” de la aplicación que se encuentran en la carpeta con el mismo nombre. En ellos se establecen las funciones asíncronas que se encargarán de enviar los datos requeridos por cada endpoint y de recibir y devolver la respuesta a los componentes que los utilicen, para así poder actualizar los datos o realizar las acciones que el usuario solicite.

A continuación, se adjuntan varios ejemplos de cómo se han implementado estas funciones:

```
38  export async function login(formData: UserLoginForm){
39      try{
40          const { data } = await service.post<string>('/auth/Login', formData)
41          localStorage.setItem('AUTH_TOKEN_TUDUAPP', data)
42          return data
43      } catch (error){
44          if(isAxiosError(error) && error.response){
45              throw new Error(error.response.data.error)
46          }
47      }
48 }
```

Los tipos post suelen recibir un formData como entrada (del tipo

- Hooks

En esta aplicación únicamente se ha utilizado un Hook, “hooks/useAuth.ts”.

```
TuDuApp_Front > src > hooks > [?] useAuth.ts > [?] useAuth
1  import { getUser } from "@/services/AuthService";
2  import { useQuery } from "@tanstack/react-query";
3
4  export const useAuth = () => {
5      const {data, isError, isLoading} = useQuery({
6          queryKey: ['user'],
7          queryFn: getUser,
8          retry: 1,
9          refetchOnWindowFocus: false
10     })
11
12     return {data, isError, isLoading}
13 }
```

El hook useAuth utiliza useQuery de react-query para gestionar la autenticación del usuario obteniendo los datos del usuario a través de la función getUser. Configura la consulta con una clave ['user'], deshabilita la actualización automática al enfocar la ventana (refetchOnWindowFocus: false), y permite un solo reintento en caso de error (retry: 1). Retorna el estado de la consulta, incluyendo los datos del usuario (data), si hay un error (isError), y si la consulta está en proceso de carga (isLoading), facilitando así la gestión del estado de autenticación en componentes React.

- Utils e idiomas

En el fichero “locales/es.ts” se implementa la traducción/formateo de los estados de las tareas, ya que los recibimos como se encuentran en base de datos, en inglés y sin espacios.

```
TuDuApp_Front > src > locales > es.ts > statusTranslation
1  export const statusTranslation : {[key: string] : string} = [
2    pending: 'PENDIENTE',
3    onHold: 'EN ESPERA',
4    inProgress: 'EN PROGRESO',
5    completed: 'COMPLETADA'
6  ]
```

Por otro lado en la carpeta “utils”, al igual que en el backend se implementan varias funciones que centralizamos ahí para tenerlas a mano en otros componentes del frontend. Contamos con dos utils:

- “policies.ts”, donde definimos un método que comprueba si un usuario es el manager. Esto nos sirve para mostrar en cada proyecto la etiqueta MANAGER si el usuario es el manager o COLABORADOR si no. También para mostrar u ocultar opciones como eliminar, editar y añadir tareas, entre otros. Se usa en el DashBoardView y en el ProjectDetailsView.

```
TuDuApp_Front > src > utils > policies.ts > isManager
1  import { Project, TeamMember } from "../types"
2
3  export const isManager = (managerId : Project['manager'], userId : TeamMember['_id']) => [
4    return managerId === userId
5  ]
```

- "utils.ts", en el que definimos un método que formatea las fechas.

```
TuDuApp_Front > src > utils > utils.ts > formatDate
1  export function formatDate(isoString: string) : string{
2    const date = new Date(isoString)
3    const formatter = new Intl.DateTimeFormat('es-ES', {
4      year: 'numeric',
5      month: 'Long',
6      day: 'numeric'
7    })
8    return formatter.format(date)
9 }
```

- Implementacion de componentes

Para explicar cómo se ha implementado el frontend se ha preferido explicar las implementaciones más importantes, ya que se usan en casi todos los componentes, vistas.

Todos los componentes y vistas siguen la siguiente estructura:

```
export default function NombreComponenteOView({children} : {children: React.ReactNode}){
  //CODIGO HANDLERS, REALIZAR PETICIONESM, COMPROBACIONES, ETC
  return (
    //CODIGO HTML
  )
}
```

Se ha usado lo siguiente:

- useNavigate() de react-router-dom para modificar la ruta del navegador. Un ejemplo de uso sería cuando estamos añadiendo una tarea nueva, mientras el formulario está abierto la url del navegador es <http://localhost:5173/projects/idProject?newTask=true>, pero si cerramos el formulario la ruta se modifica a <http://localhost:5173/projects/idProject>.

- useQueryClient() de @tanstack/react-query para invalidar queries, por lo que evitamos que los datos manejados por la aplicación en cada momento sean los correctos. En palabras más técnicas invalidamos los datos de la caché que consideremos desactualizados. Siguiendo el ejemplo anterior, en el AddTaskModal que usamos para registrar tareas, se usa para invalidar el dato almacenado en cache del id del proyecto.
- useParams() de react-router-dom y URLSearchParams, nos sirven para recoger los parámetros de la url que nos dan información como los ids de los proyectos y de las tareas, o para mostrar modales en caso de que exista un parámetro en concreto.
- useForm() de react-hook-form, nos sirve para gestionar los datos formularios, permitiéndonos validarlos y manipularlos de forma eficiente y fácil.
- useMutation() de @tanstack/react-query, nos permite realizar las peticiones al backend y manejar los errores o casos de éxito. En este proyecto usamos principalmente las key:
 - mutationFn: que indica el método de los servicios que se van a ejecutar al llamar la mutación.
 - onError: que captura los errores y nos permite realizar ciertas acciones para controlarlos, en este proyecto lo que se hace es usar Toastify para mostrar una notificación con el error.
 - onSuccess: si todo ha ido bien tras ejecutar la mutationFn, sigue ejecutando lo que se encuentre dentro de esta key. Principalmente ejecutaremos, invalidaciones de query, resets de formularios, useNavigate() para cambiar url y Toastify para mostrar una notificación con el mensaje de éxito. A continuación se muestra un ejemplo:

```
const {mutate} = useMutation({  
  mutationFn: serviceFunction,  
  onError: (error) => {  
    toast.error(error.message)  
  },  
  onSuccess: (data) => {  
    queryClient.invalidateQueries({queryKey: ['key', key]})  
    toast.success(data)  
    reset()  
    navigate(location.pathname, {replace: true})  
  }  
})
```

- Handlers, los handlers son los encargados de llamar a la mutación en casi todos los casos. Estos son llamados desde los botones u otros casos como en el Drag N Drop. A continuación se muestra un ejemplo.

```
const handleCreateTask = (formData: TaskFormData) => {  
  const data = {  
    formData,  
    projectId  
  }  
  mutate(data)  
}
```

En el apartado 8.3. se adjuntarán las capturas del código implementado para crear las vistas y los components.

- Layouts

Los layouts serán los contenedores principales desde los que irán mostrándose las views, y desde las views los modales/formularios/paneles. En esta aplicación se han definido 3 layouts.

AppLayout que cuenta con un header en el que definimos la barra de navegación, una sección que contiene el Outlet (encargado de renderizar y representar los componentes definidos en los routes), el footer y el

ToastContainer que se encargará de mostrarnos las notificaciones de Toastify.

```
TuDuApp_Front > src > layouts > AppLayout.tsx > AppLayout
1 import { Link, Navigate, Outlet } from "react-router-dom"
2 import { ToastContainer } from 'react-toastify'
3 import 'react-toastify/dist/ReactToastify.css'
4 import Logo from "@/components/Logo"
5 import NavMenu from "@/components/NavMenu"
6 import { useAuth } from "@/hooks/useAuth"
7
8
9 export default function AppLayout(){
10
11     const {data, isError, isLoading} = useAuth()
12
13     if(isLoading) return 'Loading...'
14     if(isError) {
15         return <Navigate to='auth/Login'/>
16     }
17
18     if(data) return [
19         <>
20             <header className="bg-gray-800 py-5">
21                 <div className="max-w-screen-2xl mx-auto flex flex-col lg:flex-row justify-between items-center">
22                     <div className="w-64">
23                         <Link to="/">
24                             <Logo/>
25                         </Link>
26                     </div>
27
28                     <NavMenu userName={data.name}/>
29                 </div>
30             </header>
31
32             <section className="max-w-screen-2xl mx-auto mt-10 p-5">
33                 <Outlet/>
34             </section>
35
36             <footer className="py-5">
37                 <p className="text-center">
38                     Todos los derechos reservados {new Date().getFullYear()}
39                 </p>
40             </footer>
41
42             <ToastContainer
43                 pauseOnHover={false}
44                 pauseOnFocusLoss={false}
45             />
46
47         </>
48     ]
49 }
```

El AuthLayout cuenta con un div, anidado dentro de otro div, que contiene el logo y el Outlet de las rutas definidas, y el ToastContainer. Lo que se pretende es que el logo quede en el centro de la ventana y justo debajo se vayan renderizando los formularios de login, registro, recuperación de contraseñas, etc.

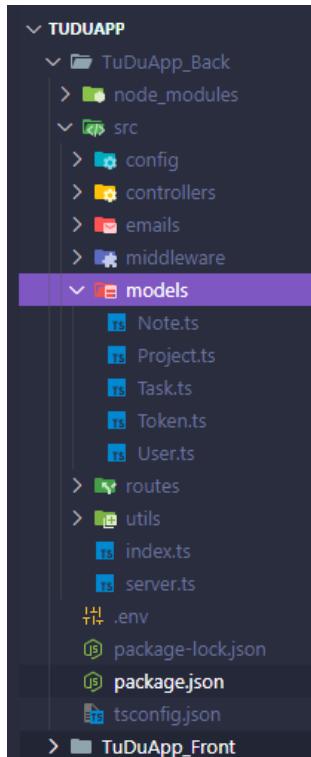
```
TuDuApp_Front > src > layouts > AuthLayout.tsx > ...
1 import Logo from "@/components/Logo"
2 import { Outlet } from "react-router-dom"
3 import { ToastContainer } from "react-toastify"
4
5 export default function AuthLayout() {
6     return (
7         <>
8             <div className="bg-gray-800 min-h-screen">
9                 <div className="py-10 lg:py-20 mx-auto w-[450px]">
10                    <Logo/>
11
12                    <div className="mt-10">
13                        <Outlet/>
14                    </div>
15                </div>
16            </div>
17
18            <ToastContainer
19                pauseOnHover={false}
20                pauseOnFocusLoss={false}>
21            </>
22        </>
23    )
24}
```

Y, por último, el ProfilerLayout que únicamente tiene el componente Tabs, que representa el menú de navegación de la vista de “Mi perfil” y el Outlet, que renderiza el formulario de editar cuenta y el de cambiar contraseña.

4.3. Desarrollo del backend (Node.js con Express)

- **Implementación de la lógica de negocio e integración con MongoDB**

Toda la lógica de negocio se recoge en la carpeta “models” del proyecto TuDuApp_Back. En ella podemos encontrar un fichero para cada uno de los modelos previamente definidos en el punto **3.3. Diseño de la lógica de negocio**.



Para no repetir texto, explicaré la construcción de todos los modelos y posteriormente adjuntaré el código de cada uno. Podemos destacar varios apartados sin contar con las importaciones: la interfaz que usaremos para los objetos de TypeScript, el objeto de la base de datos de Mongoose y la conexión entre estos dos.

Arriba del todo, definimos la estructura del Document, en la interfaz, especificando los tipos de datos y las relaciones con otros documentos. Para relacionar atributos a otros documentos usamos PopulatedDoc.

Seguido de esto, encontramos el esquema de Moongoose, que define la estructura del Document en MongoDB, incluyendo los tipos de datos y otros atributos, como trimado, si es requerido, valor por defecto, etc. Además de las referencias a otros Documents (atributo “ref”).

Por último, se conecta la interfaz con el esquema de Moongose, creando así el modelo, y se exporta para poder usarlo en otras partes de la aplicación. La conexión de interfaz y esquema se lleva acabo de la siguiente forma:

```
const Model = mongoose.model<IModel>('Model', ModelSchema)
```

Siendo IModel la interfaz y ModelSchema el esquema de Moongose.

- user.ts

- Interfaz IUser:

```
1  import mongoose, {Schema, Document} from "mongoose";
2
3  export interface IUser extends Document{
4      email: string
5      password: string
6      name: string
7      confirmed: boolean
8  }
```

- Esquema de Moongose userSchema:

```
10 const userSchema: Schema = new Schema({
11     email:{
12         type: String,
13         required: true,
14         lowercase:true,
15         unique: true
16     },
17     password:{
18         type: String,
19         required: true
20     },
21     name:{
22         type: String,
23         required: true
24     },
25     confirmed:{
26         type: Boolean,
27         default: false
28     }
29 })
30
31
```

Destacamos que todos sus atributos excepto “confirmed” son requeridos. Además, email se guardará en minúscula y será único, es decir no se podrá repetir en otros registros de este modelo.

- Conexión y exportación del modelo User:

```
32  const User = mongoose.model<IUser>('User', userSchema)
33  export default User
```

- token.ts

- Interfaz IToken:

```
3  export interface IToken extends Document {
4    token: string
5    user: Types.ObjectId
6    createdAt: Date
7 }
```

- Esquema de Moongoose tokenSchema :

```
9  const tokenSchema: Schema = new Schema({
10    token: {
11      type: String,
12      required: true
13    },
14    user: {
15      type: Types.ObjectId,
16      ref: 'User'
17    },
18    expiresAt: {
19      type: Date,
20      default: Date.now(),
21      expires: "1d"
22    }
23 })
```

Destacamos que el campo “token” es requerido, “user” hace referencia al modelo User y “expiresAt” es una fecha que se inicia con valor por defecto y durará 1 día, tras el día se eliminará automáticamente.

- Conexión y exportación del modelo Token:

```
25  const Token = mongoose.model<IToken>('Token', tokenSchema)
26  export default Token
```

- task.ts

- Interfaz ITask:

```
13  export interface ITask extends Document {
14    name: string
15    description: string
16    project: Types.ObjectId
17    status: TaskStatus
18    completedBy: {
19      user: Types.ObjectId,
20      status: TaskStatus
21    }[]
22    notes: Types.ObjectId[]
23 }
```

- Esquema de Moongoose TaskSchema :

```

25  const TaskSchema: Schema = new Schema({
26    name: {
27      type: String,
28      required: true,
29      trim: true
30    },
31    description: {
32      type: String,
33      required: true,
34      trim: true
35    },
36    project: {
37      type: Types.ObjectId,
38      required: true,
39      ref: 'Project'
40    },
41    status: {
42      type: String,
43      enum: Object.values(taskStatus),
44      default: taskStatus.PENDING
45    },
46    completedBy: [
47      {
48        user: {
49          type: Types.ObjectId,
50          ref: 'User',
51          default: null
52        },
53        status: {
54          type: String,
55          enum: Object.values(taskStatus),
56          default: taskStatus.PENDING
57        }
58      ],
59    notes: [
60      {
61        type: Types.ObjectId,
62        ref: 'Note'
63      }
64    ]
65  }, {timestamps: true})

```

Destacamos que “name”, “description” y “project” son requeridos, “project” hace referencia al modelo Project, “status” guardara valores de un “enumerado” llamado “taskStatus”, “completedBy” es un array de tupas “user”-“status”, haciendo referencia al modelo User y el enumerado mencionado, y por último, que “notes” es un array de objetos que hacen referencia al modelo “Note”.

- Conexión y exportación del modelo Task:

```

74  const Task = mongoose.model<ITask>('Task', TaskSchema)
75  export default Task

```

- Este modelo también cuenta con un middleware que se encarga de eliminar las notas que pertenezcan a una tarea que va a ser eliminada. Este middleware se ejecutará antes de completar la acción en la base de datos.

```

68   TaskSchema.pre('deleteOne', {document: true}, async function (){
69     const taskId = this._id
70     if(!taskId) return
71     await Note.deleteMany({task: taskId})
72   })

```

- El enumerado de “taskStatus” también se define en este fichero, exportándose como el tipo TaskStatus:

```

4   const taskStatus = {
5     PENDING: 'pending',
6     ON_HOLD: 'onHold',
7     IN_PROGRESS: 'inProgress',
8     COMPLETED: 'completed',
9   } as const
10
11 export type TaskStatus = typeof taskStatus[keyof typeof taskStatus]
12 //> type TaskStatus = "pending" | "onHold" | "inProgress" | "completed"

```

- note.ts

- Interfaz INote:

```

3   export interface INote extends Document {
4     content: String
5     createdBy: Types.ObjectId
6     task: Types.ObjectId
7   }
8

```

- Esquema de Moongoose NoteSchema :

```

9   const NoteSchema: Schema = new Schema({
10     content: {
11       type: String,
12       required: true
13     },
14     createdBy: {
15       type: Types.ObjectId,
16       ref: 'User',
17       required: true
18     },
19     task: {
20       type: Types.ObjectId,
21       ref: 'Task',
22       required: true
23     }
24   }, {timestamps: true})
25

```

Destacamos que todos sus campos son requeridos, "createdBy" hace referencia al modelo User y "task" al modelo Task.

- Conexión y exportación del modelo Token:

```

26   const Note = mongoose.model<INote>('Note', NoteSchema)
27   export default Note

```

- project.ts

- Interfaz IToken:

```

7   export interface IProject extends Document {
8     name: string
9     client: string
10    description: string
11    tasks: PopulatedDoc<ITask & Document>[]
12    manager: PopulatedDoc<IUser & Document>
13    team: PopulatedDoc<IUser & Document>[]
14  }

```

- Esquema de Moongoose tokenSchema :

```

17  const ProjectSchema: Schema = new Schema({
18    name: {
19      type: String,
20      trim: true,
21      required: true
22    },
23    client: {
24      type: String,
25      trim: true,
26      required: true
27    },
28    description: {
29      type: String,
30      trim: true,
31      required: true
32    },
33    tasks:[
34      {
35        type: Types.ObjectId,
36        ref: 'Task'
37      }
38    ],
39    manager: {
40      type: Types.ObjectId,
41      ref: 'User'
42    },
43    team:[
44      {
45        type: Types.ObjectId,
46        ref: 'User'
47      }
48    ]
49  }, {timestamps: true})

```

Destacamos que “name”, “client” y “description” son requeridos, “tasks” es un array de objetos que referencia al modelo Task, “team” es un array de objetos que referencia al modelo User y “manager” hace referencia al modelo User.

- Conexión y exportación del modelo Token:

```

66  //connection Typescript Model with Mongoose Schema
67  const Project = mongoose.model<IPProject>('Project', ProjectSchema)
68  export default Project

```

- En este modelo también contamos con un middleware que se ejecutará antes de completar la acción de eliminar un proyecto. Lo que hará será eliminar las tareas que le pertenezcan y sus respectivas notas.

```

52  ProjectSchema.pre('deleteOne', {document: true}, async function (){
53    const projectId = this._id
54    if(!projectId) return
55
56    //Delete notes
57    const tasks = await Task.find({project: projectId})
58    for(const task of tasks) {
59      await Note.deleteMany({task: task.id})
60    }
61    //Delete tasks
62    await Task.deleteMany({project: projectId})
63  })
64

```

Por último, en la carpeta “config” se encuentra el archivo de conexión con la base de datos de MongoDB, usando Mongoose, a través del método dbConnect.

```

TuDuApp_Back > src > config > db.ts > dbConnect
1  import colors from 'colors'
2  import {exit} from 'node:process'
3  import mongoose from 'mongoose'
4
5
6  export const dbConnect = async () => [
7    try{
8      const connection = await mongoose.connect(process.env.DATABASE_CONNECTION_URL)
9      console.log(colors.cyan.bold(`MongoDB Connect to: ${connection.connection.host}:${connection.connection.port}`))
10    }catch(error){
11      console.log(colors.red.bold(`Error when trying to connect to mongodb --> ${error.message}`))
12      exit(1)
13    }
14 ]

```

En el bloque try-catch, creamos la conexión haciendo uso de mongo.connect y la variable de entorno que guarda la url de conexión. Tras intentar se nos mostrará un mensaje de log para informar si se ha establecido la conexión correctamente o se ha producido un error.

4.4. Implementación de la API RESTful

Para la implementación de la API se han creado varias carpetas que modularizan el código.

- Servidor

El servidor viene determinado por los ficheros de la raíz del proyecto server.ts e index.ts.

```
TuDuApp_Back > src > server.ts > default
  1 import dotenv from 'dotenv'
  2 import cors from 'cors'
  3 import morgan from 'morgan'
  4 import { corsConfig } from './config/cors'
  5 import express from 'express'
  6 import {dbConnect} from './config/db'
  7 import projectRoutes from './routes/ProjectRoutes'
  8 import authRoutes from './routes/AuthRoutes'
  9
 10
 11 dotenv.config()
 12 dbConnect();
 13 const app = express()
 14 app.use(cors(corsConfig))
 15
 16 //Logger
 17 app.use(morgan('dev'))
 18
 19 //Read forms data
 20 app.use(express.json())
 21
 22 //Routes
 23 app.use('/api/auth', authRoutes)
 24 app.use('/api/projects', projectRoutes)
 25
 26
 27 export default app
```

En server.ts se establece la configuración y conexión necesaria para correr la aplicación. Haciendo uso de express() se crea una nueva aplicación Express. Para poder ver logs de las peticiones que se van realizando se ha usado el middleware Morgan. Con express.json() realizamos las lecturas de las solicitudes entrantes con las que establecemos el req.body. Y para terminar con lo más destacable del fichero, se configuran las rutas principales de la aplicación haciendo referencia a authRoutes y projectRoutes respectivamente.

```
TuDuApp_Back > src > index.ts > ...
1 import server from './server'
2 import colors from 'colors'
3
4 const port = process.env.PORT || 5500
5
6 server.listen(port, () => {
7   console.log(colors.green.bold(`REST API running on port ${port}`));
8 }) .on('error', (err: NodeJS.ErrnoException) => {
9   if (err.code === 'EADDRINUSE') {
10     console.error(colors.red.bold(`Port ${port} is already in use. Please choose a different port.`));
11   } else {
12     console.error(colors.red.bold(`Failed to start server: ${err.message}`));
13   }
14 });


```

Por otro lado, en index.ts pone en marcha el servidor HTTP configurado en server.ts, manejando los posibles errores que puedan producirse durante su arranque.

■ Rutas

Las rutas se establecen en la carpeta “routes”. Se han implementado dos, una para api/auth, AuthRoutes.ts, y otra para api/project ProjectRoutes.ts.

La estructura que siguen ambas es la siguiente:

- 1º Creamos el router con la función Router().
- 2º Se crean las rutas post, get, put o delete haciendo uso de dicho router.
- 3º Para cada ruta creada, se valida manejando los errores con el handlerInputErrors o con authenticate (si es necesario autenticación), y en caso de llevar body se valida haciendo uso de la librería express-validator.
- 4º Se especifica el método del controlador que se ejecutara cuando se llame.
- 5º Algunas de ellas llevan datos en la propia url que van representados por /ruta/:<**nombreDato**>.

En AuthRoutes.ts se definen todas las rutas que tienen que ver con usuarios. Son las siguientes:

- /create-account → para crear cuentas.
- /confirm-account → para confirmar la cuenta

- /login → para iniciar sesión.
- /request-code → para solicitar un token de 6 dígitos nuevo.
- /forgot-password → para solicitar el cambio de contraseña.
- /validate-token → para validar el token ya sea para cambio de contraseña o para confirmar cuenta.
- /update-password/:token → para actualizar la cuenta, recibe el token por url.
- /user → para obtener el user
- /profile → para actualizar el perfil del usuario (nombre y correo)
- /update-password → para actualizar la contraseña desde la ventana de “Mi perfil”, no lleva el token pero solicita la contraseña anterior en su defecto.
- /check-password → comprueba si la contraseña es correcta, usado para la eliminación de proyectos.

```
TuDuApp_Back > src > routes > AuthRoutes.ts > ...
1 import { Router } from "express";
2 import { body, param } from 'express-validator'
3 import { AuthController } from '../controllers/AuthController';
4 import { handlerInputErrors } from "../middleware/validation";
5 import { authenticate } from "../middleware/auth";
6
7 const router = Router()
8
9 router.post('/create-account',
10   body('name').notEmpty().withMessage('name es obligatorio'),
11   body('password').isLength({min: 8}).withMessage('La contraseña es demasiado corta, debe contener al menos 8 caracteres'),
12   body('password_confirmation').custom((value, {req}) => {
13     if(value !== req.body.password){
14       throw new Error('Las contraseñas no coinciden')
15     }
16     return true
17   }),
18   body('email').isEmail().withMessage('email no válido'),
19   handlerInputErrors,
20   AuthController.createAccount
21 )
22
23 router.post('/confirm-account',
24   body('token').notEmpty().withMessage('token es obligatorio'),
25   handlerInputErrors,
26   AuthController.confirmAccount
27 )
```

```

29   router.post('/Login',
30     body('email').isEmail().withMessage('email no válido'),
31     body('password').notEmpty().withMessage('password es obligatorio'),
32     handlerInputErrors,
33     AuthController.login
34   )
35
36   router.post('/request-code',
37     body('email').isEmail().withMessage('email no válido'),
38     handlerInputErrors,
39     AuthController.requestConfirmationCode
40   )
41
42   router.post('/forgot-password',
43     body('email').isEmail().withMessage('email no válido'),
44     handlerInputErrors,
45     AuthController.forgotPassword
46   )
47
48   router.post('/validate-token',
49     body('token').notEmpty().withMessage('token es obligatorio'),
50     handlerInputErrors,
51     AuthController.validateToken
52   )
53
54   router.post('/update-password/:token',
55     param('token').isNumeric().withMessage('token no válido'),
56     body('password').isLength({min: 8}).withMessage('La contraseña es demasiado corta, debe contener al menos 8 caracteres'),
57     body('password_confirmation').custom((value, {req}) => {
58       if(value !== req.body.password){
59         throw new Error('Las contraseñas no coinciden')
60       }
61       return true
62     }),
63     handlerInputErrors,
64     AuthController.updatePasswordWithToken
65   )

```

```

67   router.get('/user',
68     authenticate,
69     AuthController.user
70   )
71
72   //PROFILE ROUTES
73   router.put('/profile',
74     authenticate,
75     body('name').notEmpty().withMessage('name es obligatorio'),
76     body('email').isEmail().withMessage('email no válido'),
77     AuthController.updateProfile
78   )
79
80
81   router.post('/update-password',
82     authenticate,
83     body('current_password').notEmpty().withMessage('La contraseña actual es obligatoria'),
84     body('password').isLength({min: 8}).withMessage('La contraseña es demasiado corta, debe contener al menos 8 caracteres'),
85     body('password_confirmation').custom((value, {req}) => {
86       if(value !== req.body.password){
87         throw new Error('Las contraseñas no coinciden')
88       }
89       return true
90     }),
91     handlerInputErrors,
92     AuthController.updateCurrentUserPassword
93   )
94
95   router.post('/check-password',
96     authenticate,
97     body('password').notEmpty().withMessage('password es obligatorio'),
98     handlerInputErrors,
99     AuthController.checkPassword
100 )
101 export default router

```

En ProjectRoutes.ts se definen el resto de rutas de la aplicación relacionadas con la gestión de proyectos y tareas. Son las siguientes:

- / → para la creación de proyectos
- /:id tipo GET → para obtener el proyecto al que corresponda el id que se pasa por url.
- /:id tipo PUT → para actualizar el proyecto al que corresponda el id que se pasa por url.
- /:id tipo DELETE → para eliminar el proyecto al que corresponda el id que se pasa por url.
- /:projectId/tasks tipo POST → para obtener las tareas del proyecto especificado.
- /:projectId/tasks tipo GET → para crear tareas nuevas en el proyecto especificado.
- /:projectId/tasks/:taskId tipo GET → para obtener una única tarea especificada. projectId hace referencia al proyecto y taskId a la tarea.
- /:projectId/tasks/:taskId tipo DELETE → para eliminar la tarea especificada.
- /:projectId/tasks/:taskId/status → para actualizar el estado de la tarea especificada.
- /:projectId/team tipo GET → para obtener los miembros colaboradores del proyecto especificado.
- /:projectId/team/find → para buscar un usuario a través de su email.
- /:projectId/team tipo POST → para añadir un usuario a los colaboradores del proyecto.
- /:projectId/team/:userId → para eliminar el usuario especificado de los colaboradores del proyecto.
- /:projectId/tasks/:taskId/notes tipo POST → para crear notas en la tarea especificada.
- /:projectId/tasks/:taskId/notes tipo GET → para obtener las notas de la tarea especificada.
- /:projectId/tasks/:taskId/notes/:noteId → para eliminar la nota especificada.

```

TuDuApp_Back > src > routes > ProjectRoutes.ts > ...
1  import { Router } from "express";
2  import { ProjectController } from "../controllers/ProjectController";
3  import { TeamController } from "../controllers/TeamController";
4  import { body, param } from 'express-validator'
5  import { handlerInputErrors } from "../middleware/validation";
6  import { TaskController } from "../controllers/TaskController";
7  import { validateProjectExists } from "../middleware/project";
8  import { hasAuthorization, taskBelongsToProject, validateTaskExists } from "../middleware/task";
9  import Task from "../models/Task";
10 import { authenticate } from "../middleware/auth";
11 import { NoteController } from "../controllers>NoteController";
12
13 const router = Router()
14
15 router.use(authenticate)
16
17 router.post('/', [
18   body('name').notEmpty().withMessage('name es obligatorio'),
19   body('client').notEmpty().withMessage('client es obligatorio'),
20   body('description').notEmpty().withMessage('description es obligatorio'),
21   handlerInputErrors,
22   ProjectController.createProject
23 ])
24
25 router.get('/:id',
26   param('id').isMongoId().withMessage('ID no válido'),
27   handlerInputErrors,
28   ProjectController.getProjectById
29 )
30
31 router.get('/', ProjectController.getProjects)
32
33 router.put('/:id',
34   param('id').isMongoId().withMessage('ID no válido'),
35   body('name').notEmpty().withMessage('name es obligatorio'),
36   body('client').notEmpty().withMessage('client es obligatorio'),
37   body('description').notEmpty().withMessage('description es obligatorio'),
38   handlerInputErrors,
39   ProjectController.updateProject
40 )
41
42 router.delete('/:id',
43   param('id').isMongoId().withMessage('ID no válido'),
44   handlerInputErrors,
45   ProjectController.deleteProjectById
46 )
47
48 //TASK ROUTES
49 router.param('projectId', validateProjectExists)
50
51 router.post('/:projectId/tasks',
52   hasAuthorization,
53   body('name').notEmpty().withMessage('name es obligatorio'),
54   body('description').notEmpty().withMessage('description es obligatorio'),
55   handlerInputErrors,
56   TaskController.createTask
57 )
58
59 router.get('/:projectId/tasks',
60   TaskController.getProjectTasks
61 )
62
63 router.param('taskId', validateTaskExists)
64 router.param('taskId', taskBelongsToProject)
65

```

```

66  router.get('/:projectId/tasks/:taskId',
67    param('taskId').isMongoId().withMessage('ID de Tarea no válido'),
68    handlerInputErrors,
69    TaskController.getTaskById
70  )
71
72  router.put('/:projectId/tasks/:taskId',
73    hasAuthorization,
74    param('taskId').isMongoId().withMessage('ID de Tarea no válido'),
75    body('name').notEmpty().withMessage('name es obligatorio'),
76    body('description').notEmpty().withMessage('description es obligatorio'),
77    handlerInputErrors,
78    TaskController.updateTask
79  )
80
81  router.delete('/:projectId/tasks/:taskId',
82    hasAuthorization,
83    param('taskId').isMongoId().withMessage('ID de Tarea no válido'),
84    handlerInputErrors,
85    TaskController.deleteTaskById
86  )
87
88  router.post('/:projectId/tasks/:taskId/status',
89    param('taskId').isMongoId().withMessage('ID de Tarea no válido'),
90    body('status').notEmpty().withMessage('state es obligatorio'),
91    handlerInputErrors,
92    TaskController.updateStatus
93  )
94
95 //TEAM ROUTES
96  router.get('/:projectId/team',
97    TeamController.getProjectTeam
98  )
99
100 router.post('/:projectId/team/find',
101   body('email').isEmail().toLowerCase().withMessage('Email no válido'),
102   handlerInputErrors,
103   TeamController.findMemberByEmail
104 )
105
106
107 router.post('/:projectId/team',
108   body('id').isMongoId().withMessage('ID no válido'),
109   handlerInputErrors,
110   TeamController.addMemberById
111 )
112
113 router.delete('/:projectId/team/:userId',
114   param('userId').isMongoId().withMessage('ID no válido'),
115   handlerInputErrors,
116   TeamController.removeMemberById
117 )
118
119
120 //NOTES ROUTES
121 router.post('/:projectId/tasks/:taskId/notes',
122   body('content').notEmpty().withMessage('Email no válido'),
123   handlerInputErrors,
124   NoteController.createNote
125 )

```

```

120
127 router.get('/:projectId/tasks/:taskId/notes',
128   NoteController.getNotes
129 )
130
131 router.delete('/:projectId/tasks/:taskId/notes/:noteId',
132   param('noteId').isMongoId().withMessage('ID no válido'),
133   handlerInputErrors,
134   NoteController.deleteNote
135 )
136
137 export default router

```

▪ Middlewares

Para controlar ciertas validaciones se ha creado la carpeta “middleware” que contiene los ficheros en los que definimos los métodos que realizarán las comprobaciones antes de que se ejecuten algunos de los métodos de los controladores.

auth.ts nos ayuda con la autenticación del usuario, se explicará más en detalle en el apartado **6.4. Seguridad y privacidad**.

project.ts se encarga de validar si el proyecto al que se hace referencia en la llamada existe o no en base de datos.

```

TuDuApp_Back > src > middleware > project.ts > validateProjectExists
1 import type {Request, Response, NextFunction} from 'express'
2 import Project, { IProject } from '../models/Project'
3
4 declare global{
5   namespace Express{
6     interface Request{
7       project: IProject
8     }
9   }
10 }
11
12 export async function validateProjectExists(req: Request, res: Response, next: NextFunction){
13   try{
14     const {projectId} = req.params
15     const project = await Project.findById(projectId)
16
17     if(!project){
18       const error = new Error(`Proyecto no encontrado`)
19       return res.status(404).json({error: error.message})
20     }
21     req.project = project
22     next()
23   } catch(error){
24     res.status(500).json({error: 'Ha ocurrido un error'})
25   }
26 }

```

task.ts contiene 3 validaciones: una para comprobar si la tarea referenciada existe en base de datos, otra para comprobar si la tarea referenciada pertenece al proyecto referenciado y otra para comprobar si el usuario que esta llamando tiene autorización para realizar la acción solicitada.

```
TuDuApp_Back > src > middleware > task.ts > hasAuthorization
1 import type {Request, Response, NextFunction} from 'express'
2 import Task, { ITask } from "../models/Task"
3
4 declare global{
5     namespace Express{
6         interface Request{
7             task: ITask
8         }
9     }
10 }
11
12 export async function validateTaskExists(req: Request, res: Response, next: NextFunction){
13     try{
14         const {taskId} = req.params
15
16         const task = await Task.findById(taskId)
17         if(!task){
18             const error = new Error(`No se ha encontrado la tarea`)
19             return res.status(404).json({error: error.message})
20         }
21
22         req.task = task
23         next()
24     } catch(error){
25         res.status(500).json({error: 'Ha ocurrido un error'})
26     }
27 }
28
29 export function taskBelongsToProject(req: Request, res: Response, next: NextFunction){
30     if(req.task.project.toString() !== req.project.id.toString()){
31         const error = new Error(`La tarea no pertenece al proyecto`)
32         return res.status(403).json({error: error.message})
33     }
34     next()
35 }
36
37 export function hasAuthorization(req: Request, res: Response, next: NextFunction){
38     console.log(req.project)
39     if(req.user.id.toString() !== req.project.manager.toString()){
40         const error = new Error(`No tienes permiso para realizar la acción`)
41         return res.status(403).json({error: error.message})
42     }
43     next()
44 }
```

Y, por último, pero no menos importante, validation.ts, donde definimos el handlerInputErrors que utilizarán todos los endpoints y que usamos para controlar errores.

```
TuDuApp_Back > src > middleware > validation.ts > handlerInputErrors
1 import type { Request, Response, NextFunction } from "express"
2 import { validationResult } from 'express-validator'
3
4
5 export const handlerInputErrors = (req: Request, res: Response, next: NextFunction) => {
6   let errors = validationResult(req)
7   if(!errors.isEmpty()){
8     return res.status(400).json({errors: errors.array()})
9   }
10  next()
11 }
```

▪ Controladores

En la carpeta “controllers” se encuentran los controladores que implementan los métodos que acceden a la base de datos y realizan las acciones que los usuarios requieren. Estos métodos son usados por los routers definidos en la carpeta “routes”. No se adjuntan capturas del código de los controladores debido a que se incrementaría mucho la longitud del documento, pero es posible verlos a través del enlace a Github del apartado 8.1. Hay 5 controladores:

- AuthController: en el se definen los métodos que tienen que ver con User. Métodos:
 - createAccount → añade un usuario nuevo a la tabla users
 - confirmAccount → actualiza el campo confirmed del usuario en base de datos.
 - login → envía el JWT token de respuesta para mantener la sesión activa.
 - requestConfirmationCode → crea un nuevo token de 6 dígitos para el user y envía el correo de confirmación a su email.
 - forgotPassword → crea un nuevo token de 6 dígitos para el user y envía el correo de actualización de contraseña.
 - validateToken → comprueba si el token existe en base de datos.

- updatePasswordWithToken → recibe el token y la nueva contraseña, y en caso de pasar las validaciones actualiza la contraseña del usuario.
 - user → devuelve el usuario
 - updateProfile → actualiza la información del usuario en base de datos
 - updateCurrentUserPassword → actualiza la contraseña en base de datos, si la contraseña anterior que recibe es correcta.
 - checkPassword → comprueba si la contraseña que recibe es correcta para el usuario que lo usa.
- NoteController.ts: en él se definen los métodos que tienen que ver con Note. Métodos:
- createNote: crea un registro nuevo en la tabla notes de la base de datos.
 - getNotes: recupera todas las notas que existen en base de datos de la tarea que se indique.
 - deleteNote: elimina la nota de la base de datos.
- TeamController.ts: en él se definen los métodos que tienen que ver con los colaboradores de los proyectos. Métodos:
- getProjectTeam → busca el proyecto en base de datos y devuelve la lista team.
 - findMemberByEmail → busca un usuario en base de datos a través de su email y lo devuelve.
 - addMemberById → añade el usuario a la lista team del proyecto y actualiza el proyecto en base de datos.
 - removeMemberById → elimina el usuario de la lista team del proyecto y actualiza el proyecto en base de datos.
- TaskController.ts: en el se definen los métodos que tienen que ver con Task. Métodos:
- createTask → añade un registro en la tabla tasks.
 - getProjectTask → busca todas las tareas que pertenezcan a un determinado proyecto y las devuelve.

- getTaskById → busca la tarea especificada en la tabla tasks y la devuelve.
 - updateTask → actualiza la información de la tarea en base de datos.
 - deleteTaskById → elimina la tarea especificada de la tabla tasks.
 - updateStatus → actualiza el campo completedBy de la tarea en base de datos.
- ProjectController.ts: en él se definen los métodos que tienen que ver con Project. Métodos:
- createProject → añade un registro en la tabla projects.
 - getProjectById → busca el proyecto especificado en base de datos y lo devuelve.
 - getProjects → busca los proyectos, a los que tiene acceso el user, en base de datos y los devuelve.
 - updateProject → actualiza el proyecto en base de datos con la información que recibe.
 - deleteProjectById → elimina el proyecto especificado de la tabla projects.

▪ Utils

En la carpeta “utils” se encuentran los ficheros en los que implementamos los métodos auxiliares. Esto se ha realizado para tenerlos centralizados y poder usarlos fácilmente desde otras clases. Entre estos podemos encontrar:

- auth.ts: en el usamos la librería bcrypt para hashear la contraseña y para comprobar una string normal con un string previamente hasheado.

```
TuDuApp_Back > src > utils > auth.ts > checkPassword
1 import bcrypt from 'bcrypt'
2
3 export const hashPassword = async (password : string) => {
4   const salt = await bcrypt.genSalt(10)
5   return await bcrypt.hash(password, salt)
6 }
7
8 export const checkPassword = async(enterPassword : string, storeHash: string) => {
9   return await bcrypt.compare(enterPassword, storeHash)
10 }
```

- jwt.ts: se explica en el punto 6.4.
- Token.ts: contiene el método con el algoritmo de generación de claves de 6 dígitos aleatorias.

```
TuDuApp_Back > src > utils > Token.ts ...  
1  export const generateToken = () => Math.floor(100000 + Math.random() * 900000) . toString()
```

■ Email

En la carpeta “emails” se encuentra el fichero AuthEmails.ts que contiene los métodos que crean los correos de confirmación de cuenta o de resetear contraseña. En estos métodos usamos nodemailer (configurado en la carpeta “config”) para enviar el correo creado al usuario.

```
TuDuApp_Back > src > emails > AuthEmails > authEmail > sendPasswordResetToken > info > html  
1  import { transporter } from '../config/nodemailer'  
2  
3  interface IEmail {  
4      email: string,  
5      name: string,  
6      token: string  
7  }  
8  
9  export class AuthEmail {  
10    static sendConfirmationEmail = async (user : IEmail) => {  
11      const info = await transporter.sendMail({  
12        from: 'TuDu <admin@tudu.com>',  
13        to: user.email,  
14        subject: 'TuDu - Confirma tu cuenta',  
15        text: 'Utiliza el siguiente código para confirmar tu cuenta:',  
16        html: `<p>Hola: ${user.name}, ya casi has creado tu cuenta en TuDu, solo falta confirmar tu cuenta. Para ello copia el siguiente código y visita el enlace:</p>  
17          <p>Código: <b>${user.token}</b></p>  
18          <a href="${process.env.FRONT_URL}/auth/confirm-account">Confirmar cuenta</a>  
19          <p>El código expirará en 10 minutos.</p>  
20      })  
21      console.log('Mensaje enviado', info.messageId)  
22  }  
23  
24  
25  static sendPasswordResetToken = async (user : IEmail) => {  
26    const info = await transporter.sendMail({  
27      from: 'TuDu <admin@tudu.com>',  
28      to: user.email,  
29      subject: 'TuDu - Restablece contraseña',  
30      text: 'Restablece tu contraseña',  
31      html: `<p>Hola: ${user.name}, has solicitado el cambio de contraseña, visita el siguiente enlace para crear una nueva:</p>  
32          <p>Código: <b>${user.token}</b></p>  
33          <a href="${process.env.FRONT_URL}/auth/new-password">Restablecer contraseña</a>  
34          <p>El código expirá en 10 minutos.</p>  
35      })  
36      console.log('Mensaje enviado', info.messageId)  
37  }  
38}  
39}
```

4.5. Pruebas y control de calidad

Para este proyecto no se han realizado pruebas unitarias debido a la falta de tiempo. Aun así, se pensó en utilizar Mocha para el backend y Jest para el frontend.

Las pruebas realizadas han sido a través de Thunder Client. Se adjunta enlace al JSON que contiene todas las llamadas realizadas:

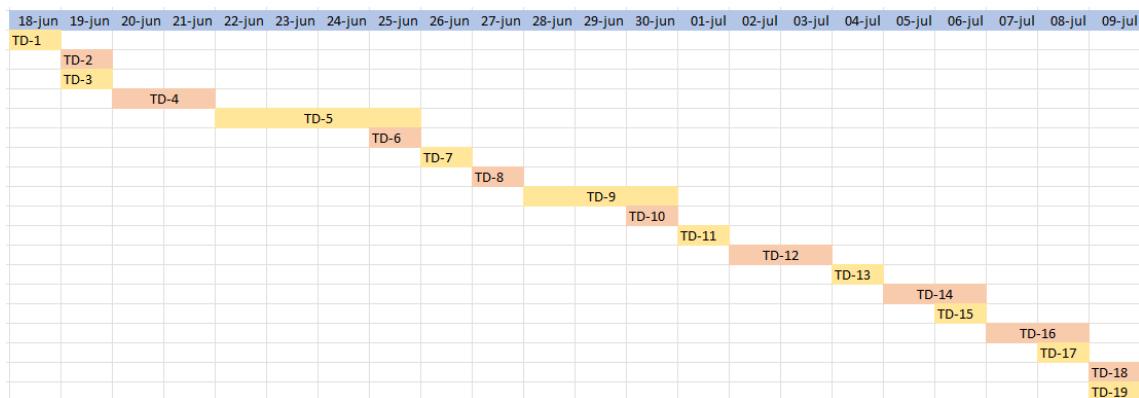
<https://github.com/DesiderioAlmansa/ThunderClientTuDuAPP.git>

5. PLANIFICACIÓN DEL PROYECTO

5.1. Cronograma de actividades

Cada tarea se ha identificado con las letras TD seguido del número de tarea. TD hacen referencia al nombre del proyecto TuDu. En la siguiente tabla se pueden ver las fechas de inicio y fin de cada una de ellas.

Actividad	Inicio estimado	Finalización estimada
TD-1 Create server	18-jun	18-jun
TD-2 Create bbdd in MongoDB and connect the project	18-jun	19-jun
TD-3 Create projects in TuDuApp	19-jun	19-jun
TD-4 Create tasks in TuDuApp	19-jun	21-jun
TD-5 Develop TuDuApp frontend base	21-jun	25-jun
TD-6 Implement frontend communication with backend	25-jun	25-jun
TD-7 Read backend errors from frontend	25-jun	26-jun
TD-8 Add the principals functions for projects and the respective views to frontend	26-jun	27-jun
TD-9 Add views and respective services to view and manage tasks	27-jun	30-jun
TD-10 Change tasks status	30-jun	30-jun
TD-11 Users authentication backend	30-jun	01-jul
TD-12 Users authentication frontend	01-jul	03-jul
TD-13 Add JWT Token	03-jul	04-jul
TD-14 Relationship between projects and authenticated users	04-jul	06-jul
TD-15 Add collaborators to projects	06-jul	06-jul
TD-16 Add roles for collaborators	06-jul	08-jul
TD-17 Add task change history and notes	08-jul	08-jul
TD-18 User Profile	08-jul	09-jul
TD-19 Add Drag N Drop to change tasks status	09-jul	09-jul



5.2. Recursos humanos y materiales

La elaboración del proyecto se ha llevado a cabo con un único integrante del equipo, Desiderio Almansa Porrero, encargado del desarrollo y de su documentación.

Los materiales utilizados son:

- Portatil HP Omen 16 – Intel Core i7 – 16 gb RAM
- Monitor Samsung Basic 27 pulgadas
- Monitor AOC Gaming Curvo 24 pulgadas
- Raton Logitech M185
- Teclado Mars Gaming

6. EVALUACIÓN Y RESULTADOS

6.1. Funcionalidades implementadas

En el apartado 2.2. se definieron las funcionalidades principales a implementar. Todas las funcionalidades se han completado con éxito.

6.2. Usabilidad y experiencia de usuario

Para saber si se ha logrado una usabilidad y experiencia de usuario satisfactorias debemos contestar a las siguientes preguntas que hacen referencia a aspectos clave de las mismas:

- ✓ **Efectividad:** ¿Pueden los usuarios completar sus tareas con precisión y sin errores? Si, se controlan todos los errores y en caso de que ocurriera alguno inesperado, la aplicación le avisa mediante notificaciones.
- ✓ **Eficiencia:** ¿Cuánto tiempo y esfuerzo necesitan los usuarios para completar sus tareas? Muy poco ya que no tiene se da información en todo momento de lo que se está haciendo y donde está situado.
- ✓ **Aprendibilidad:** ¿Cuán fácil es para los usuarios nuevos aprender a usar el producto? Muy fácil, ya que no presenta un flujo de uso excesivamente complejo.
- ✓ **Memorabilidad:** ¿Es fácil para los usuarios recordar cómo usar el producto después de no haberlo usado por un tiempo? Sí, es muy intuitivo.
- ✓ **Utilidad:** ¿El producto satisface una necesidad del usuario? ¿Es útil? Partiendo de que es un proyecto sencillo, cumple su principal objetivo: dividir proyectos o quehaceres en tareas para organizarse mejor.
- ✓ **Desempeño:** ¿El producto es rápido y confiable? El proyecto no tiene problemas de carga y a menos que no existan problemas de red, funciona correctamente.
- ✓ **Atractivo:** ¿El diseño del producto es atractivo y agradable visualmente? Se han elegido colores llamativos y que tienen buena armonía entre sí. Al no tener un diseño extravagante y complejo se considera que es atractivo y agradable (opinión del desarrollador).

6.3. Rendimiento y escalabilidad

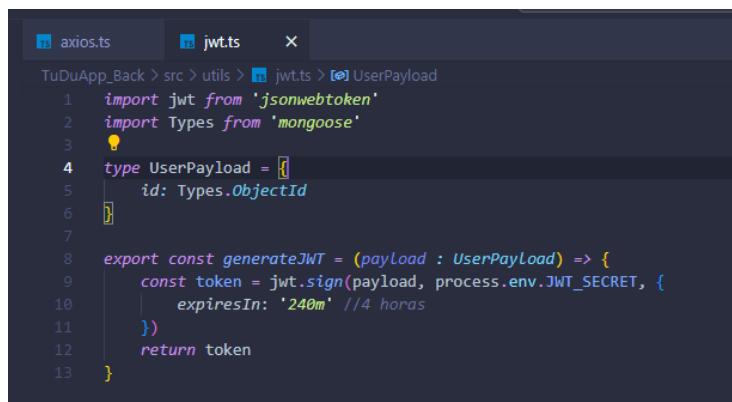
El rendimiento alcanzado es bueno, pero se considera que puede mejorar haciendo uso de técnicas de caching. Por ejemplo, indexando campos clave usando Redis, que reduciría la carga en la base de datos y mejoraría significativamente los tiempos de respuesta en caso de que exista una gran demanda de usuarios y muchos registros en la base de datos.

En cuanto a la escalabilidad, al haber usado el patrón MVC y modularizado funcionalidades, se considera que se ha asegurado satisfactoriamente una escalabilidad óptima en caso de que se quiera seguir añadiendo funcionalidad a la aplicación.

6.4. Seguridad y privacidad

En el desarrollo de aplicaciones web, garantizar la seguridad y privacidad de los usuarios es crucial. Una de las estrategias más efectivas y utilizada en el desarrollo de este proyecto es la autenticación mediante JSON Web Token (JWT).

Desde el backend, cuando un usuario inicia sesión, se genera un token jwt usando su ID y una palabra secreta que almacenamos en las variables de entorno.



```
axios.ts          jwt.ts      x
TuDuApp_Back > src > utils > jwt.ts > UserPayload
1 import jwt from 'jsonwebtoken'
2 import Types from 'mongoose'
3
4 type UserPayload = {
5   id: Types.ObjectId
6 }
7
8 export const generateJWT = (payload: UserPayload) => {
9   const token = jwt.sign(payload, process.env.JWT_SECRET, {
10     expiresIn: '240m' //4 horas
11   })
12   return token
13 }
```

Ilustración 4: Util que contiene el método para generar token JWT (Backend)

El token generado se devuelve y se almacena en el localstorage del navegador del usuario, con la palabra clave “AUTH_TOKEN_TUDUAPP”.

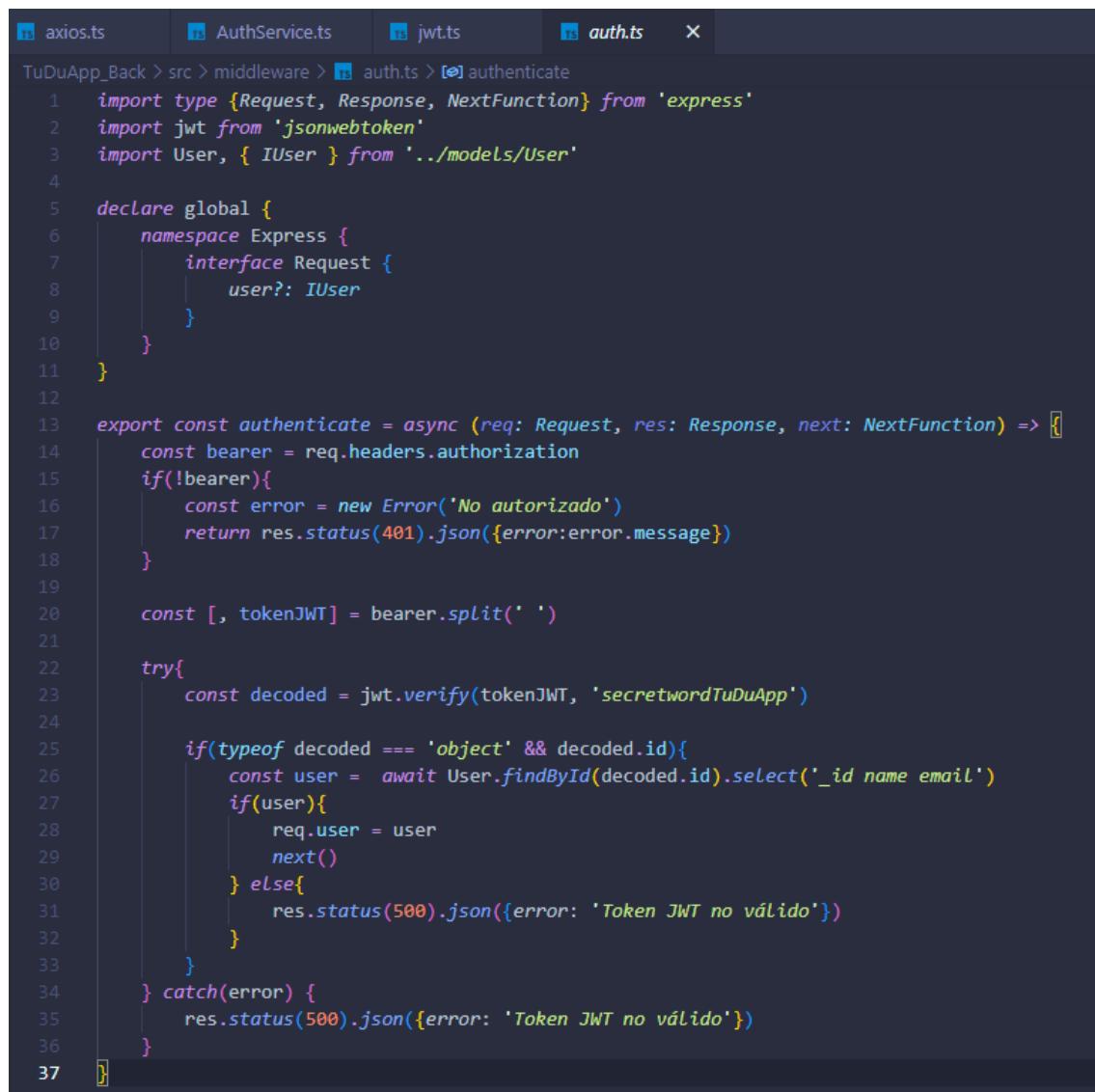
```

export async function login(formData: UserLoginForm){
    try{
        const { data } = await service.post<string>('/auth/login', formData)
        localStorage.setItem('AUTH_TOKEN_TUDUAPP', data)
        return data
    } catch (error){
        if(isAxiosError(error) && error.response){
            throw new Error(error.response.data.error)
        }
    }
}

```

Ilustración 5: AuthService del Frontend, metodo login

Una vez se tiene almacenado, el frontend usará ese token para construir las peticiones y el backend comprobará si le llega una petición con un token válido, para autenticar al usuario haciendo uso del middleware "auth.ts".



```

// auth.ts
import type {Request, Response, NextFunction} from 'express'
import jwt from 'jsonwebtoken'
import User, { IUser } from '../models/User'

declare global {
    namespace Express {
        interface Request {
            user?: IUser
        }
    }
}

export const authenticate = async (req: Request, res: Response, next: NextFunction) => [
    const bearer = req.headers.authorization
    if(!bearer){
        const error = new Error('No autorizado')
        return res.status(401).json({error:error.message})
    }

    const [, tokenJWT] = bearer.split(' ')

    try{
        const decoded = jwt.verify(tokenJWT, 'secretwordTuDuApp')

        if(typeof decoded === 'object' && decoded.id){
            const user = await User.findById(decoded.id).select('_id name email')
            if(user){
                req.user = user
                next()
            } else{
                res.status(500).json({error: 'Token JWT no válido'})
            }
        }
    } catch(error) {
        res.status(500).json({error: 'Token JWT no válido'})
    }
]

```

En este middleware decodificamos el token con la palabra secreta y si el resultado coincide con el id de un usuario la validación pasara correctamente.

Este middleware se utiliza en todos los endpoints definidos en “ProjectRoutes.ts”, por lo que todos los endpoints que contienen quedan protegidos de usuarios maliciosos. Y también se utiliza en los endpoints definidos en “AuthRoutes.ts” que tienen que ver con el perfil del usuario, actualización de contraseñas y datos, entre otros.

Por otro lado, en cuanto a la autorización, se controla mediante middlewares para tareas y proyectos, en los que se comprueba si el usuario que realizo la petición es el manager del proyecto. En caso de que si lo sea se le da autorización para realizar otras opciones que los colaboradores (no managers) no tienen.

También se ha utilizado cors para crear una whitelist que controla quien puede realizar peticiones al backend. De este modo restringimos su uso únicamente al frontend y al thunderclient. A continuación se muestra la configuración establecida en el fichero “config/cors.ts”.

```
TuDuApp_Back > src > config > cors.ts > corsConfig
  3  export const corsConfig: CorsOptions = {
  4    origin: function(origin, callback){
  5      callback(null, true)
  6    }
  7  }
  8  else{
  9    callback(new Error('CORS Error'))
 10  }
 11}
 12}
 13}
 14}
 15}
 16}
```

Con todo lo mencionado se protegen los endpoints, por lo que pueden compartirse sin problema, y los datos, ningún usuario no autorizado podrá realizar determinadas acciones.

7. CONCLUSIONES

7.1. Resumen del proyecto

El presente trabajo de fin de máster en Desarrollo Web FullStack se centra en la creación de una aplicación web utilizando el stack MERN. El proyecto se desarrolló usando el patrón de arquitectura MVC y modularización con el objetivo de construir una aplicación eficiente, escalable y fácil de mantener.

Durante el desarrollo del proyecto, se implementaron diversas funcionalidades clave, tales como la gestión de proyectos y tareas, autenticación y autorización, y la interacción con la base de datos no relacional MongoDB. Gracias a esto se ha conseguido resolver satisfactoriamente el problema planteado inicialmente.

Este proyecto no solo sirvió para aplicar los conocimientos adquiridos durante el máster, sino que también sirvió para aprender mucho más en lo que se refiere a desarrollo web y para proporcionar experiencia práctica. A lo largo del proceso, se han reforzado habilidades críticas en la programación frontend, lo que servirá para poder afrontar futuros desafíos profesionales.

7.2. Aporte del trabajo

El desarrollo de este trabajo de fin de máster en Desarrollo Web FullStack ha supuesto un significativo enriquecimiento tanto a nivel personal como profesional. La creación de un proyecto utilizando Express.js me ha permitido consolidar mis conocimientos en el desarrollo backend, específicamente en el uso de frameworks modernos para la construcción de aplicaciones web eficientes y escalables.

Este proyecto me ha aportado una comprensión profunda de la importancia de la modularización del código, facilitando así el mantenimiento y la compresión del código de la aplicación. Además, al enfrentar y resolver desafíos relacionados con Frontend, he adquirido habilidades prácticas en la implementación de vistas previamente no tenía. En conjunto, estas experiencias han reforzado mi capacidad para abordar

proyectos complejos de desarrollo web, preparándome mejor para enfrentar retos profesionales en el campo del desarrollo FullStack.

7.3. Limitaciones y futuras mejoras

Aunque la aplicación conseguida cumple los objetivos y funcionalidades planteadas al inicio del proyecto, tiene ciertas limitaciones y mejoras que pueden aplicarse en un futuro si se desea seguir ampliando el proyecto.

La principal limitación que se puede observar es que la aplicación no está preparada para manejar grandes volúmenes de datos. Esto puede resolverse aplicando técnicas de Sharding (distribución de datos en diferentes servidores para balancear cargas y mejorar rendimiento) u optimizando la aplicación mediante Caching para consultas frecuentes (por ejemplo usando Redis).

En cuanto a las mejoras más importantes que pueden implementarse pueden ser:

- Asignación de tareas: sería una gran mejora que los usuarios pudieran asignarse tareas, para saber quien la está llevando a cabo con mayor exactitud.
- Estados de tareas customizados: permitir editar o agregar los estados de las tareas a gusto del usuario.
- Añadir más información de los usuarios: como por ejemplo fotos, número de teléfono, etc.
- Separar el historial de cambios de las tareas del modal principal de la tarea.
- Añadir otras vistas para el tablero de tareas: como por ejemplo un listado.

8. ANEXOS

8.1. Código fuente

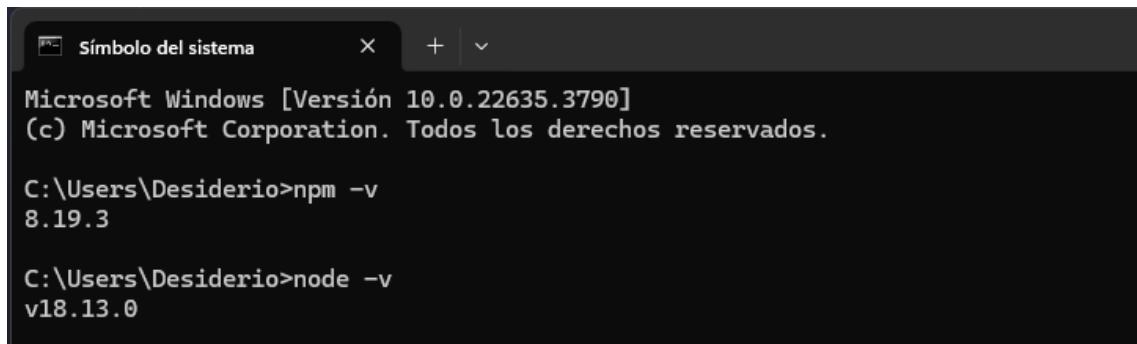
Enlace a GitHub: <https://github.com/DesiderioAlmansa/TuDApp.git>

8.2. Manual de usuario

○ Instalación

Al no estar desplegada la aplicación, es necesario seguir los siguientes pasos si se quiere ejecutar en local para probarla:

1º) Tener instalado: npm y node.js.

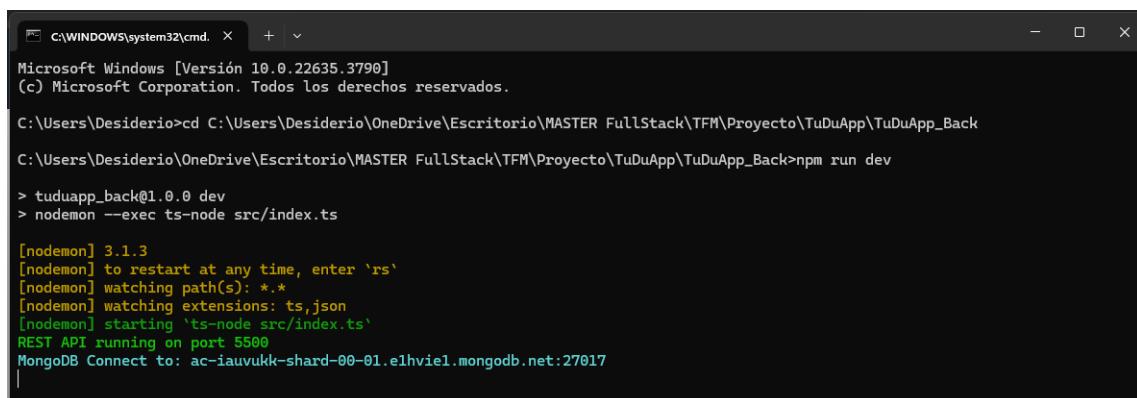


```
Símbolo del sistema Microsoft Windows [Versión 10.0.22635.3790]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Desiderio>npm -v
8.19.3

C:\Users\Desiderio>node -v
v18.13.0
```

2º) Levantar Backend: En un terminal cmd nos situamos en la carpeta ...\\TuDuApp\\TuDuApp_Back e ingresamos el comando **npm run dev**.



```
C:\WINDOWS\system32\cmd. Microsoft Windows [Versión 10.0.22635.3790]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Desiderio>cd C:\Users\Desiderio\OneDrive\Escritorio\MASTER_FullStack\TFM\Proyecto\TuDuApp\TuDuApp_Back
C:\Users\Desiderio\OneDrive\Escritorio\MASTER_FullStack\TFM\Proyecto\TuDuApp\TuDuApp_Back>npm run dev
> tduapp_back@1.0.0 dev
> nodemon --exec ts-node src/index.ts

[nodemon] 3.1.3
[nodemon] to restart at any time, enter 'rs'
[nodemon] watching path(s): *.*
[nodemon] watching extensions: ts,json
[nodemon] starting 'ts-node src/index.ts'
REST API running on port 5500
MongoDB Connect to: ac-iauvukk-shard-00-01.e1hv1el.mongodb.net:27017
```

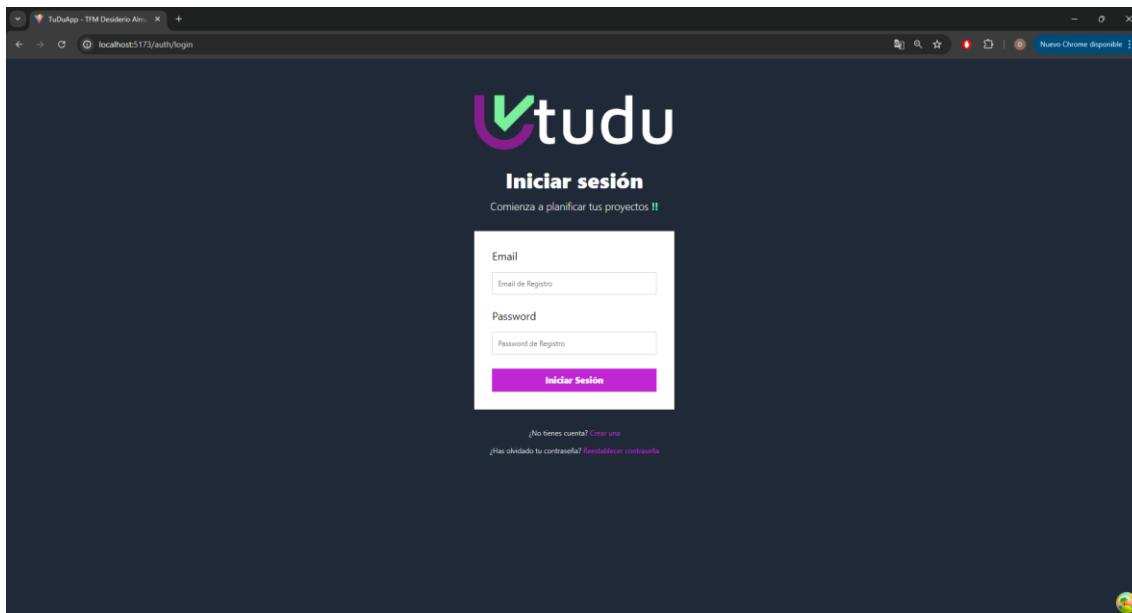
3º) Levantar Frontend: En un terminal cmd nos situamos en la carpeta ...\\TuDuApp\\TuDuApp_Front e ingresamos el comando **npm run dev**.

```
C:\WINDOWS\system32\cmd. x + v
Microsoft Windows [Versión 10.0.22635.3790]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Desiderio>cd C:\Users\Desiderio\OneDrive\Escritorio\MASTER FullStack\TFM\Proyecto\TuDuApp\TuDuApp_Front
C:\Users\Desiderio\OneDrive\Escritorio\MASTER FullStack\TFM\Proyecto\TuDuApp\TuDuApp_Front>npm run dev
> tuduapp-front@0.0.0 dev
> vite

VITE v5.3.1 ready in 947 ms
→ Local: http://localhost:5173/
→ Network: use --host to expose
→ press h + enter to show help
```

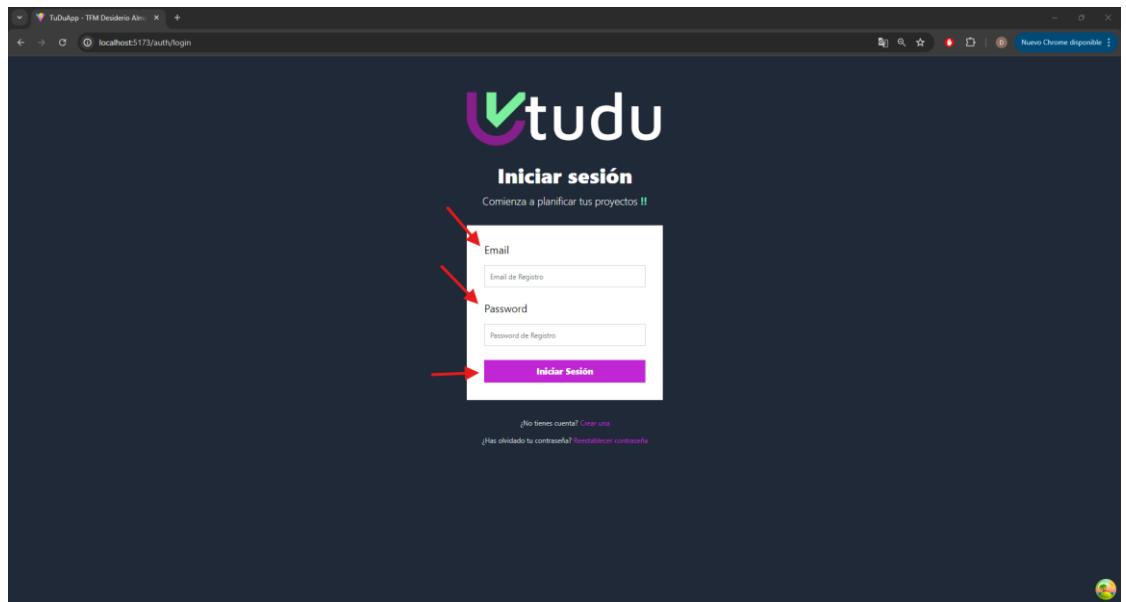
4º) Accedemos a la url <http://localhost:5173/> desde nuestro navegador.



- **Manual de uso**

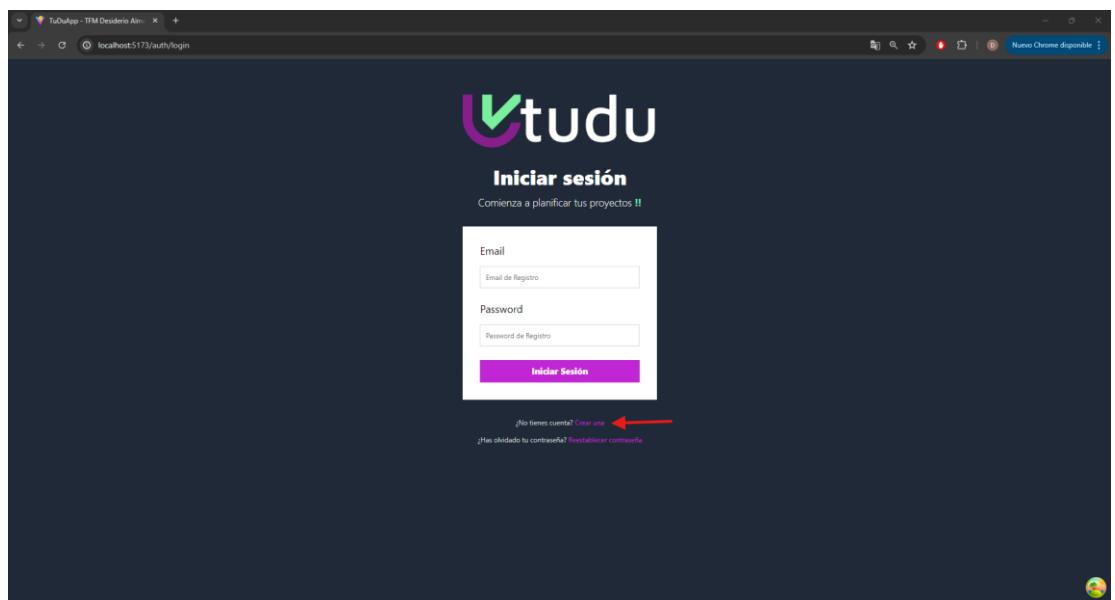
- Inicio de sesión

Al acceder a la ruta <http://localhost:5173/> se abrirá automáticamente la ventana de login, en caso de que no tengamos sesión activa. En dicha ventana se solicitan dos datos: email y contraseña. Tras introducirlos deberemos pulsar el botón de “Iniciar sesión” para logearnos y comenzar a usar la aplicación.

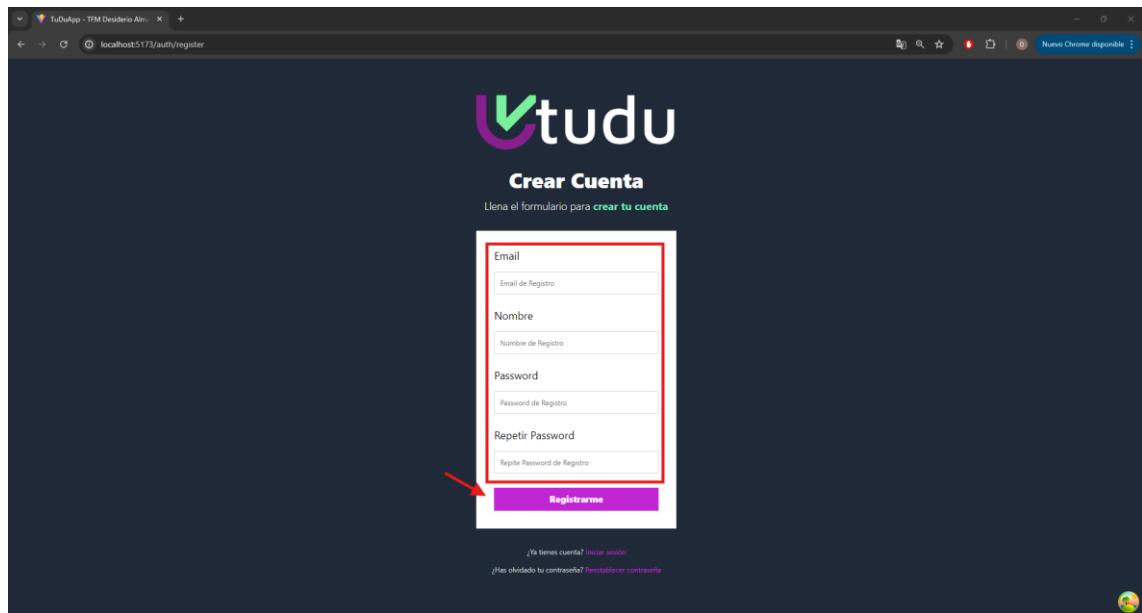


- [Crear cuenta](#)

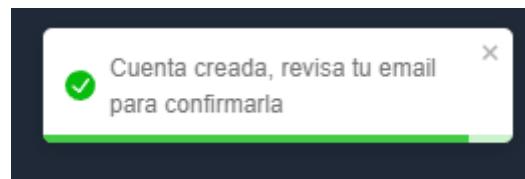
Si aún no tenemos una cuenta creada, en la parte inferior de la ventana de inicio de sesión encontramos un enlace “Crear cuenta”.



Debemos de hacer clic en él y nos redirigirá a la ventana de registro.
Debemos de introducir los datos que se solicitan y pulsar el botón
“Resgistrarme”.



Una vez realizado esto, la aplicación nos notificará que nos ha enviado un correo de verificación.



Debemos de acceder a la url que se nos proporciona e ingresar el código de 6 dígitos que también nos llegara en el correo.

TuDu - Confirma tu cuenta

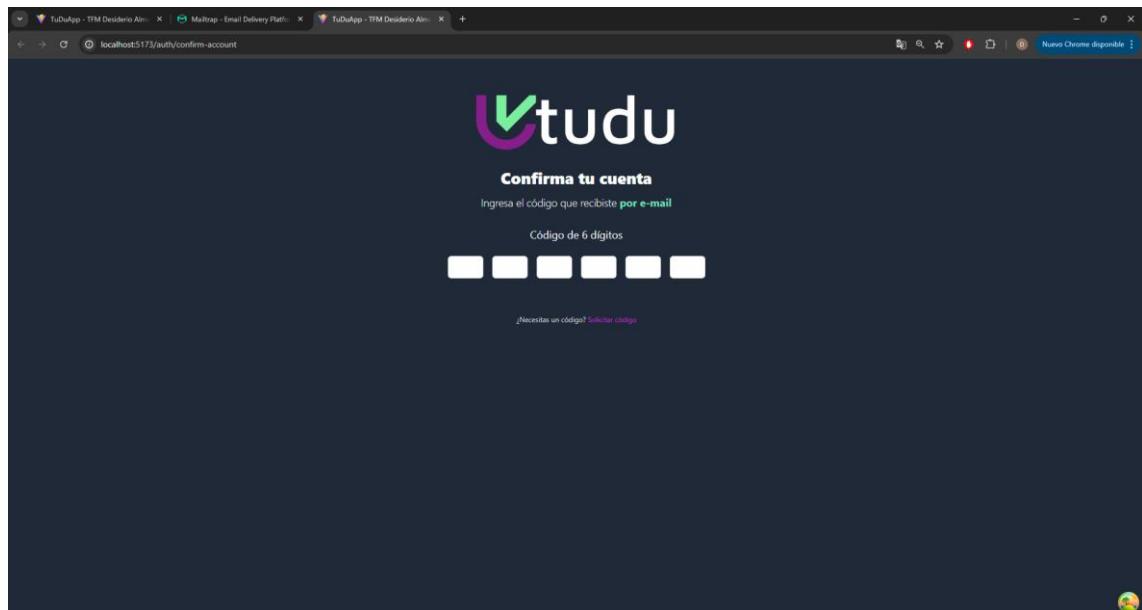
From: TuDu <admin@tudu.com>
To: <pepe@mail.es>

Show Headers

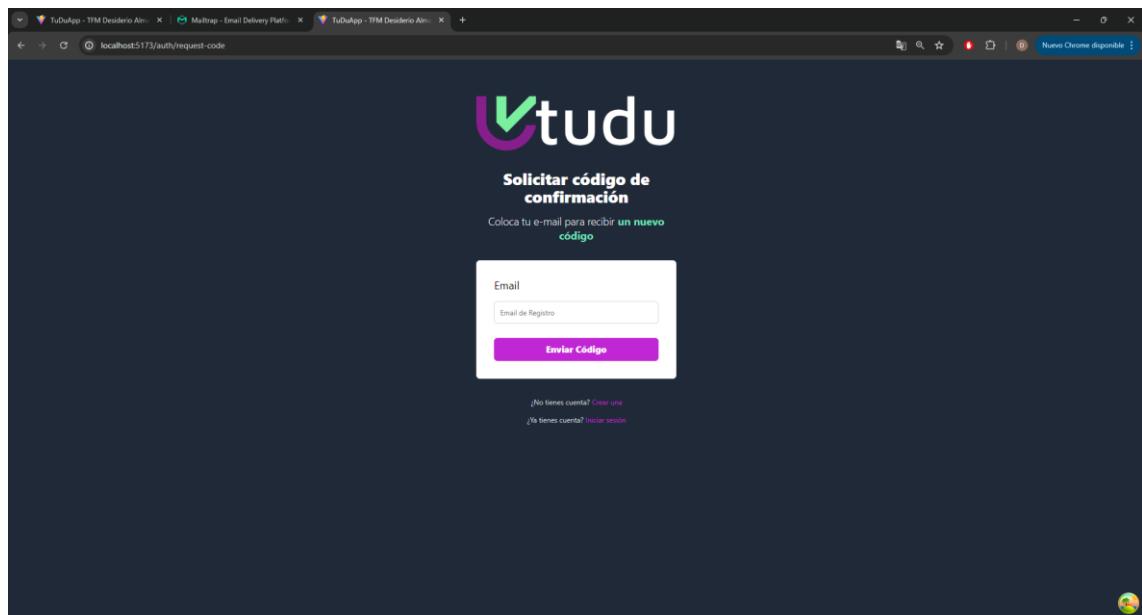
HTML HTML Source Text Raw Spam Analysis HTML Check Tech Info

Hola: Pepe Garcia, ya casi has creado tu cuenta en TuDu, solo falta confirmar tu cuenta. Para ello copia el siguiente código y visita el enlace:
Código: **459125**
[Confirmar cuenta](#)
El código expirará en 10 minutos.

Esa url nos abrirá una ventana de confirmación de cuenta en la que se nos pide ingresar el código de 6 dígitos.



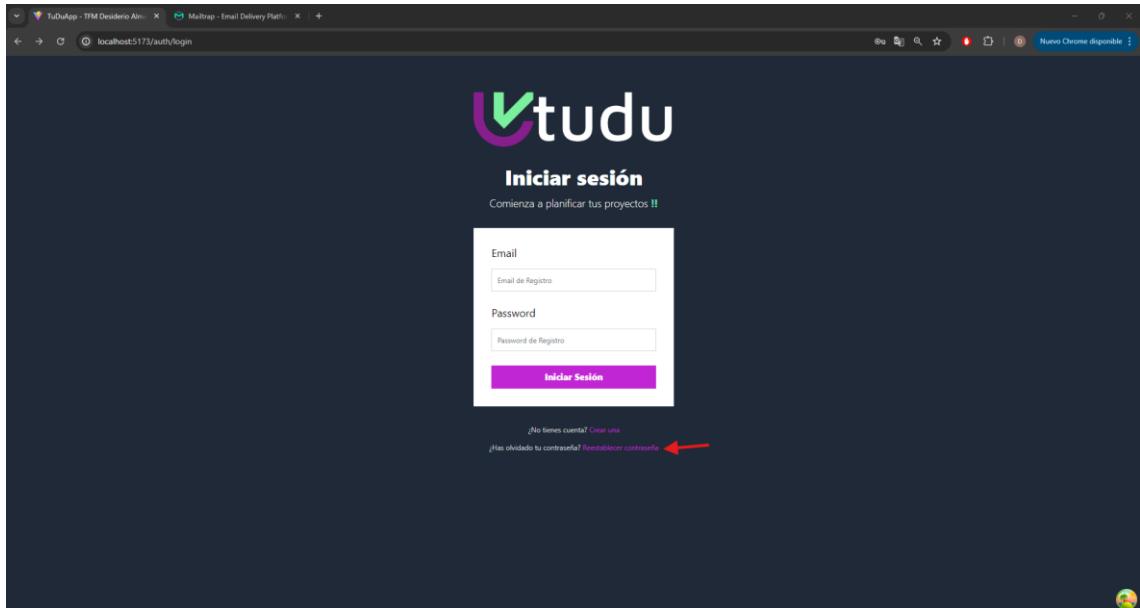
En caso de perder el código, podemos solicitar otro haciendo uso del enlace “Solicitar código”, que nos dirigirá a la ventana de solicitar código en la que debemos indicar el email del correo que hemos utilizado para registrarnos.



Una vez ingresado el código, si es correcto, nos notificará que la cuenta se ha confirmado correctamente y podremos cerrar la ventana e iniciar sesión.

- [Restablecer contraseña](#)

En caso de no recordar la contraseña, en la ventana de inicio de sesión podemos encontrar un enlace para reestablecerla.



Si accedemos a él, nos redirigirá a la ventana de recuperar contraseña en la que se solicita el correo de registro del usuario. Debemos ingresarlo y pulsar el botón “Enviar instrucciones” para que nos envíe un correo indicando lo que debemos hacer.

TuDu - Reestablece contraseña

From: TuDu <admin@tudu.com>
To: <pepe@email.es>

Show Headers

[HTML](#) [HTML Source](#) [Text](#) [Raw](#) [Spam Analysis](#) [HTML Check](#) [Tech Info](#)



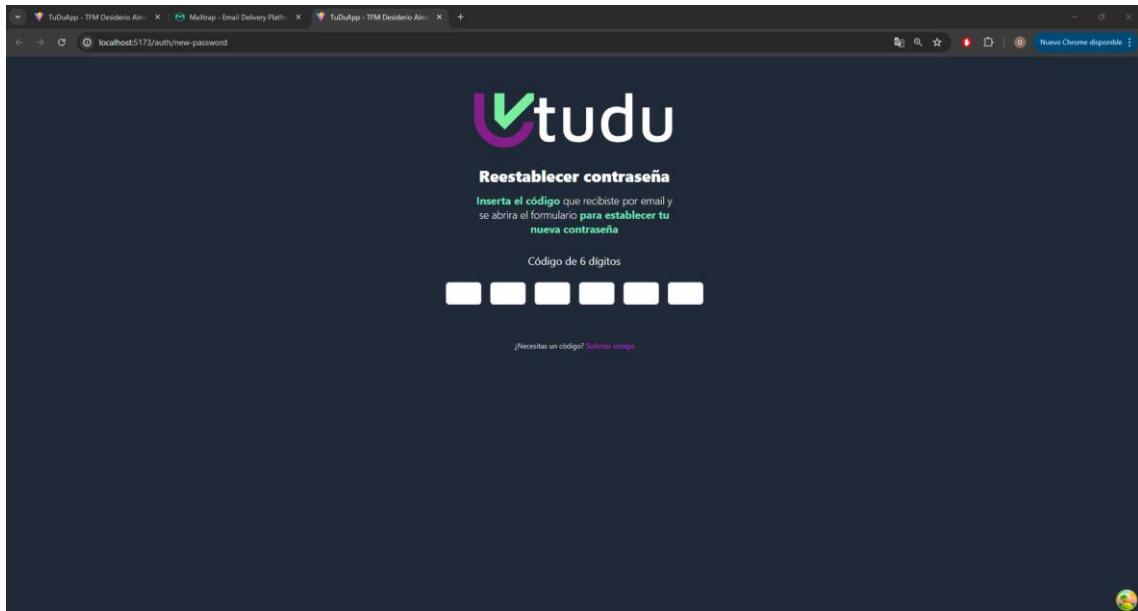
Hola: Pepe García, has solicitado el cambio de contraseña, visita el siguiente enlace para crear una nueva:

Código: **271412**

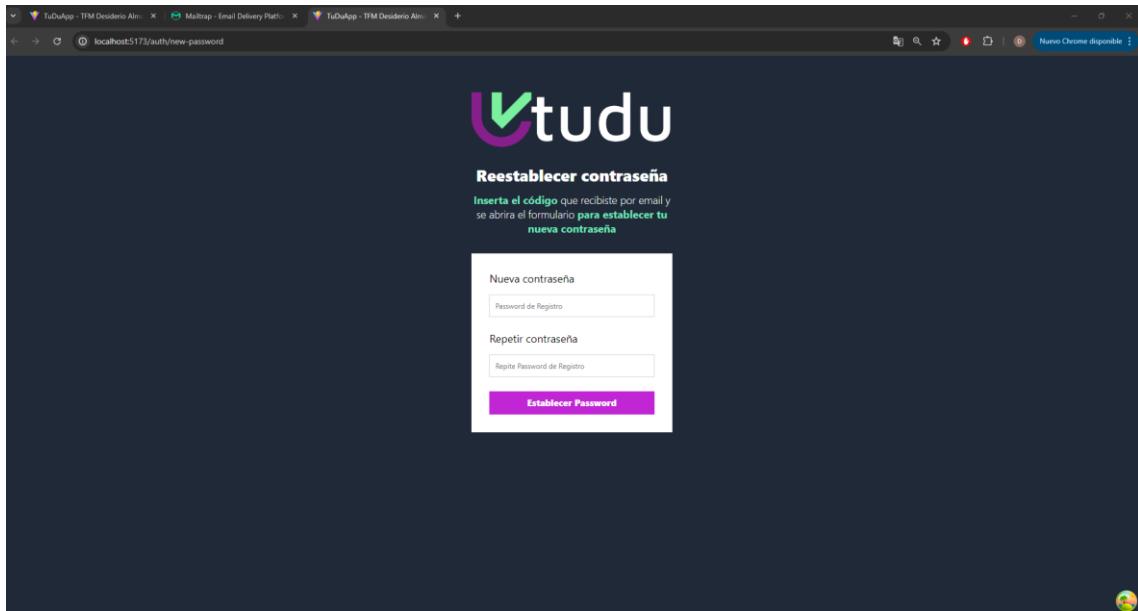
[Restablecer contraseña](#)

El código expirará en 10 minutos.

Al igual que el correo de verificación, el correo de reestablecer contraseña nos proporciona un código de 6 dígitos y un enlace. Si accedemos al enlace nos abrirá una nueva ventana en la que se pide el código proporcionado.



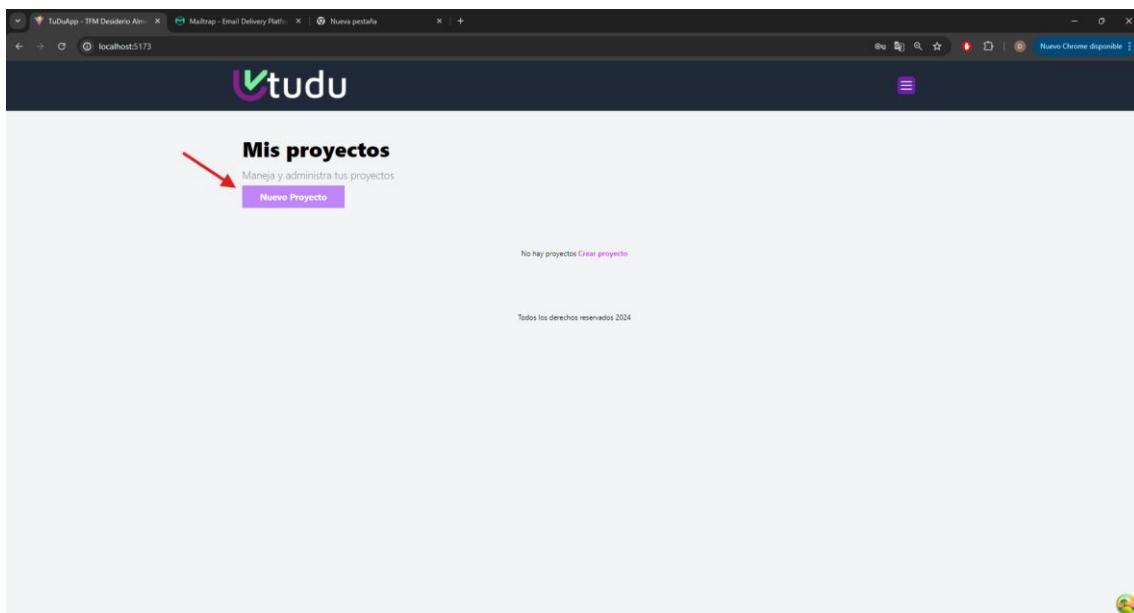
Una vez introducido el código correcto, nos mostrará un formulario en el que debemos de insertar la nueva contraseña.



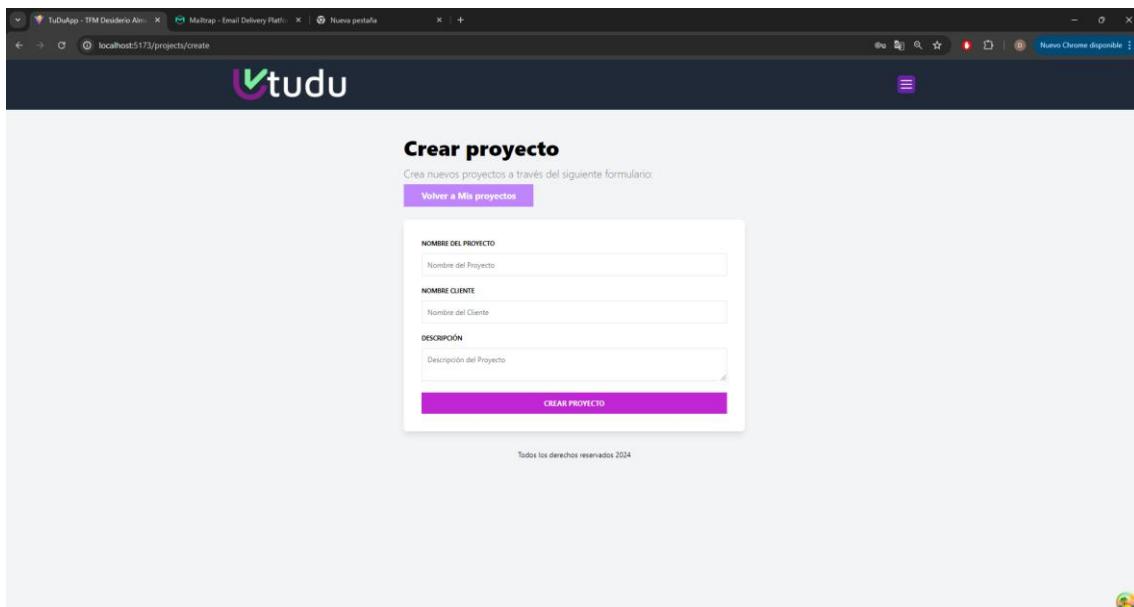
Tras restablecerla, nos redirigirá automáticamente a la ventana de inicio de sesión.

- Crear proyectos

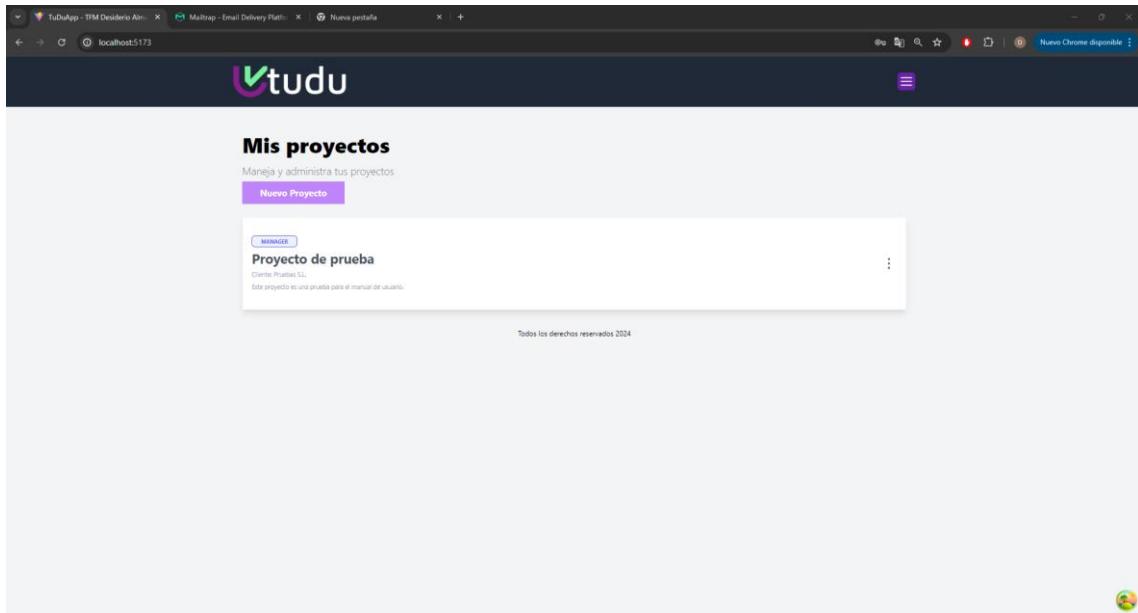
Tras iniciar sesión se nos muestra el dashboard de los proyectos. Para crear uno nuevo debemos pulsar el botón “Nuevo proyecto”.



Tras esto, nos abrirá un formulario que nos solicita la información del proyecto a crear.



Una vez completado, pulsamos el botón de “Crear proyecto” y nos redirigirá al dashboard de “Mis proyectos”.



En cada proyecto tendremos una etiqueta indicativa de si se es Manager o Colaborador.

En la parte derecha de cada proyecto tendremos un menú representado por 3 puntos. Al pulsar nos desplegará 3 opciones: Ver, editar o eliminar proyecto, si se es Manager; o una única opción, ver proyecto si se es colaborador.

COLABORADOR

Trabajo de Laravel
Cliente: Asignatura de Programación Web - 3º Ing.Informática - UCLM
Elabora un documento con los pasos a seguir para instalar Laravel y su configuración. Explica en que consiste, migraciones, edición de migraciones y ejecución de migraciones.

MANAGER

Proyecto de prueba
Cliente: Pruebas S.L.
Este proyecto es una prueba para el manual de usuario.

Todos los derechos reservados 2024

Ver Proyecto
Editar Proyecto
Eliminar Proyecto

COLABORADOR

Trabajo de Laravel
Cliente: Asignatura de Programación Web - 3º Ing.Informática - UCLM
Elabora un documento con los pasos a seguir para instalar Laravel y su configuración. Explica en que consiste, migraciones, edición de migraciones y ejecución de migraciones.

MANAGER

Proyecto de prueba
Cliente: Pruebas S.L.
Este proyecto es una prueba para el manual de usuario.

Todos los derechos reservados 2024

▪ Editar proyecto

Si hacemos clic en la opción de “Editar proyecto” nos abrirá un formulario en el que podremos modificar tanto el nombre, como el cliente o la descripción. Tras modificar lo que queramos pulsaremos “Guardar cambios” y se nos redirigirá al dashboard principal con los proyectos actualizados.

localhost:5173/projects/66910a070a26265bf1f092a1ac/edit

utudu

Editar proyecto
Edita el proyecto a través del siguiente formulario:

Volver a Mis proyectos

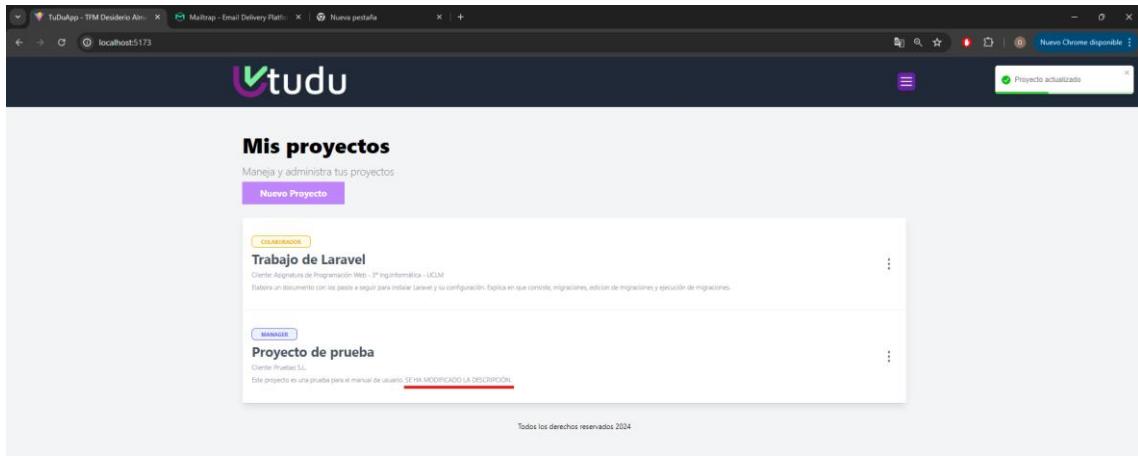
NOMBRE DEL PROYECTO
Proyecto de prueba

NOMBRE CLIENTE
Pruebas S.L.

DESCRIPCIÓN
Este proyecto es una prueba para el manual de usuario.

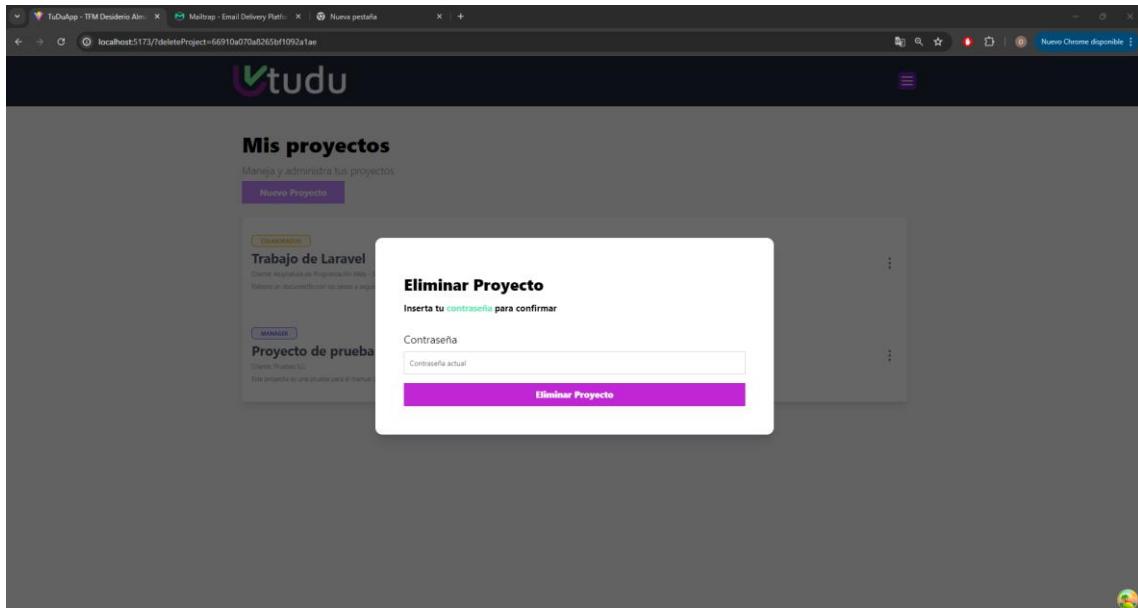
GUARDAR CAMBIOS

Todos los derechos reservados 2024



- Eliminar proyecto

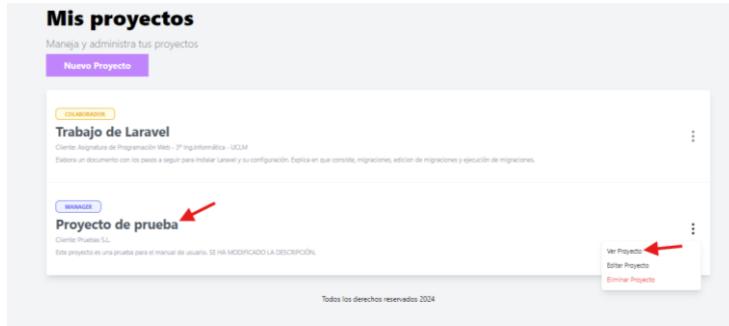
Si hacemos clic en la opción de eliminar proyecto se nos abrirá un modal que nos pedirá la contraseña del usuario para confirmar la eliminación.



En caso de introducir la contraseña incorrecta se notificará y no hará nada. En caso de introducirla correctamente el proyecto se eliminará y el dashboard de proyectos se actualizará.

- Ver proyecto

Si pulsamos la opción de ver proyecto o el título de este,

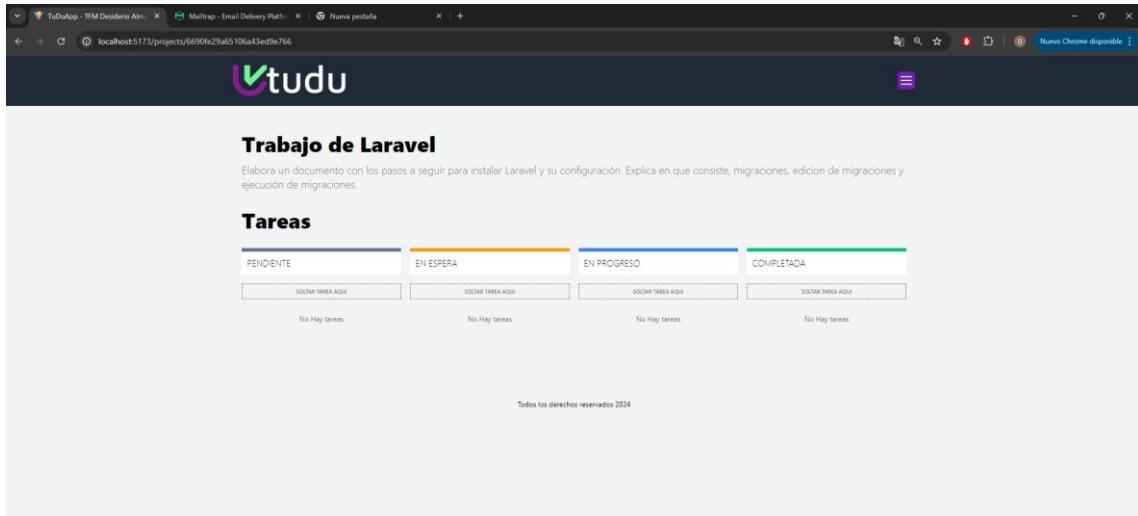


nos mostrará el tablero de tareas de este. Si se es manager del proyecto nos aparecerán dos opciones en la parte superior: "Añadir tarea, y "Colaboradores".

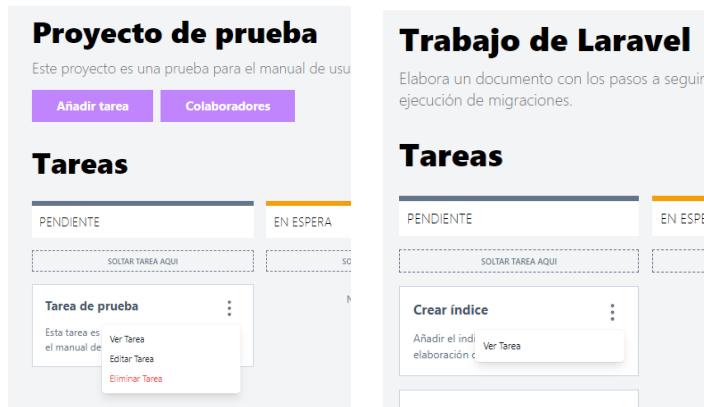
A screenshot of a task board for the project "Proyecto de prueba". The board is divided into four columns: PENDIENTE, EN ESPERA, EN PROGRESO, and COMPLETADA. Each column has a placeholder area labeled "SOLTAR TAREA AQUÍ". Below each column, it says "No Hay tareas". A red arrow points to the "EN ESPERA" column. The URL in the browser is "localhost:5173/projects/66910a070a8265bf1092a1ae".

Además podremos editar la información de las tareas y también eliminarlas.

Si se es colaborador estas opciones no saldrán, por lo que únicamente podremos ver las tareas y modificar su estado.

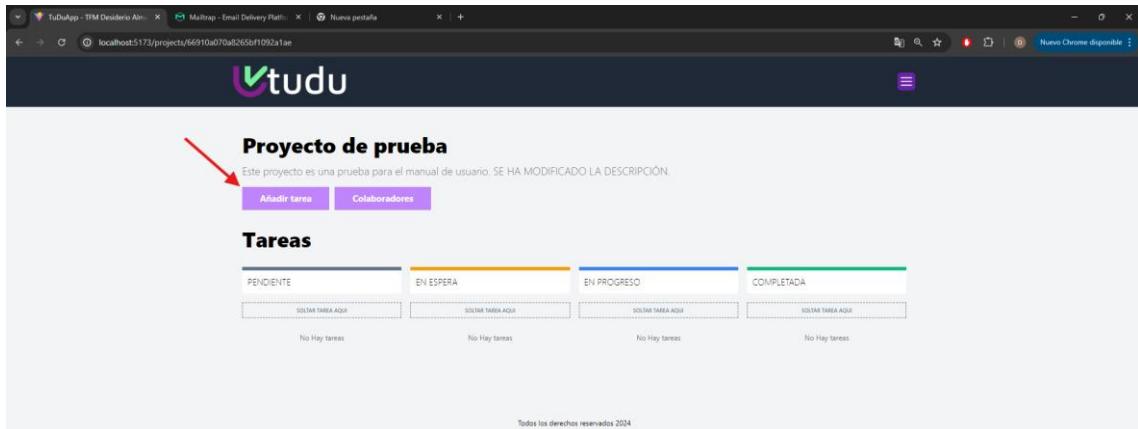


Cada tarea en el tablero del proyecto cuenta con un ícono con 3 puntos (igual que los proyectos en el dashboard), si hacemos clic en el se nos despliegan las opciones disponibles (3 si se es manager o 1 si se es colaborador).

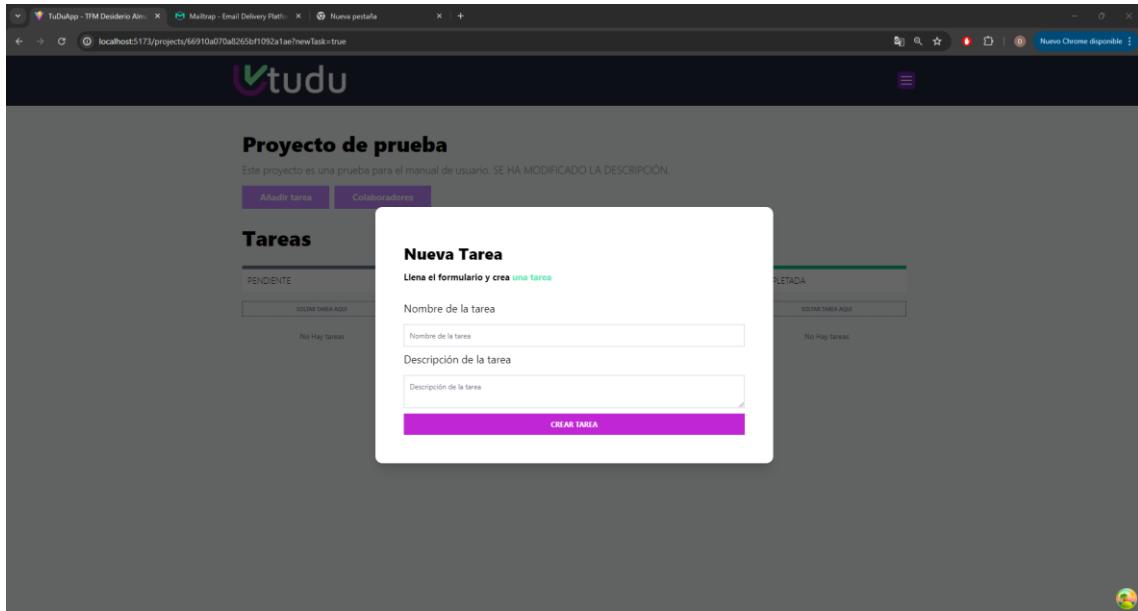


- Añadir tarea (permitido a Manager del proyecto)

Para añadir tareas a un proyecto debemos de pulsar el botón “Añadir tarea”.



Lo que nos abrirá un modal con un formulario, en el que se solicita el Nombre y la descripción de la tarea.



Una vez creada, el modal se cerrará y nos actualizará el tablero de tareas del proyecto automáticamente.

The screenshot shows a web browser window with the URL localhost:5173/projects/66910a070a8265bf1092a1ae. The page title is "TuDuApp - TiM Desiderio Alm...". The main content area is titled "Proyecto de prueba" with the sub-instruction "Este proyecto es una prueba para el manual de usuario. SE HA MODIFICADO LA DESCRIPCIÓN.". Below this are two buttons: "Añadir tarea" and "Colaboradores". A section titled "Tareas" displays four columns: "PENDIENTE" (with one card), "EN ESPERA" (empty), "EN PROGRESO" (empty), and "COMPLETADA" (empty). A specific task card for "Tarea de prueba" is shown in the PENDIENTE column, with the note "Esta tarea es de prueba para elaborar el manual de usuario". At the bottom right of the card is a small green circular icon with a person icon.

- Editar tarea (permitido a Manager del proyecto)

Haciendo clic en el ícono de 3 puntos de la tarea y después en la opción de “Editar tarea” se nos abrirá un formulario en un modal, similar al de crear tareas, que nos permite editar la información de la tarea.

The screenshot shows the same project page as before, but with a modal window overlaid on the "Tareas" section. The modal is titled "Editar Tarea" and contains the instruction "Realiza cambios a una tarea en [este formulario](#)". It has three input fields: "Nombre de la tarea" (with value "Tarea de prueba"), "Descripción de la tarea" (with value "Esta tarea es de prueba para elaborar el manual de usuario"), and a large text area below it. At the bottom of the modal is a purple button labeled "Guardar Tarea". The background of the page is dimmed to indicate the modal is active.

Una vez hechos los cambios, tras pulsar el botón “Guardar tarea”, el modal se cerrará y nos mostrará la información actualizada en el tablero de tareas.

The screenshot shows a web browser window with the URL <localhost:5173/projects/66910a070a0265bf1092a1ae>. The page title is "TuDuApp - TFM Desiderio Alm...". The main content area is titled "Proyecto de prueba" with the sub-instruction "Este proyecto es una prueba para el manual de usuario. SE HA MODIFICADO LA DESCRIPCIÓN.". Below this are two buttons: "Añadir tarea" and "Colaboradores". A section titled "Tareas" displays four horizontal bars representing task status: PENDIENTE (grey), EN ESPERA (orange), EN PROGRESO (blue), and COMPLETADA (green). Each bar has a corresponding button below it labeled "SOLICITAR TAREA AQUÍ". Under the EN ESPERA bar, there is a modal window titled "Tarea de prueba" containing the text "Esta tarea es de prueba para electoral: el manual de usuario EDITADA." At the bottom right of the page is a footer note: "Todos los derechos reservados 2024".

- Eliminar tarea (permitido a Manager del proyecto)

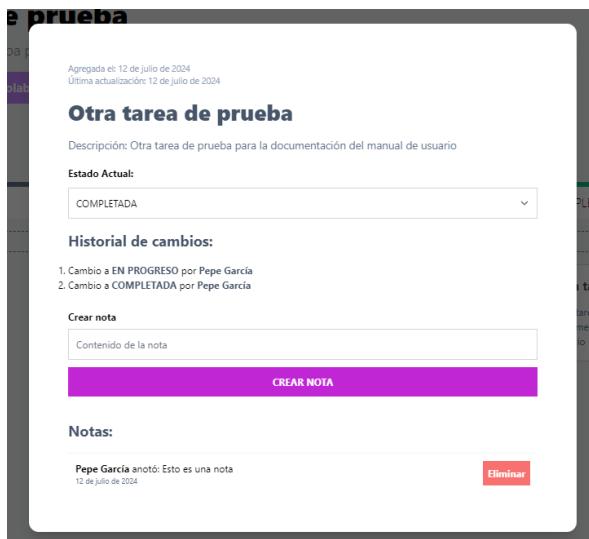
Haciendo clic en el icono de 3 puntos de la tarea y después en la opción de "Eliminar tarea", la tarea se eliminará y se quitará del tablero de tareas. En este caso no se pide confirmación.

- Ver tarea

Haciendo clic en el icono de 3 puntos de la tarea y después en la opción de "Ver tarea" se abrirá el modal en el que podremos ver toda la información de la tarea: fechas de creación y actualización, título, descripción, estado actual (que podremos modificar con un combo box), notas (junto a una caja de texto y un botón para añadirlas) y un historial (solo aparece si se han realizado cambios previamente).

The screenshot shows a modal window titled "Otra tarea de prueba" with the sub-instruction "Descripción: Otra tarea de prueba para la documentación del manual de usuario". The modal includes fields for "Estado Actual:" (set to "PENDIENTE"), "Crear nota" (with a text input field and a "CREAR NOTA" button), and a note history section with the message "No hay notas". The background of the modal is white, while the rest of the page is dark grey. The modal has a close button in the top right corner.

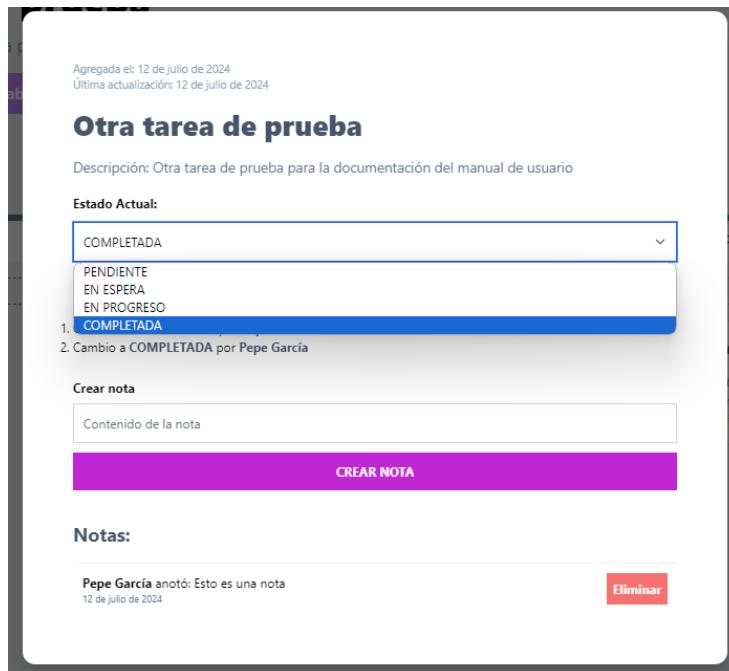
Con historial y notas:



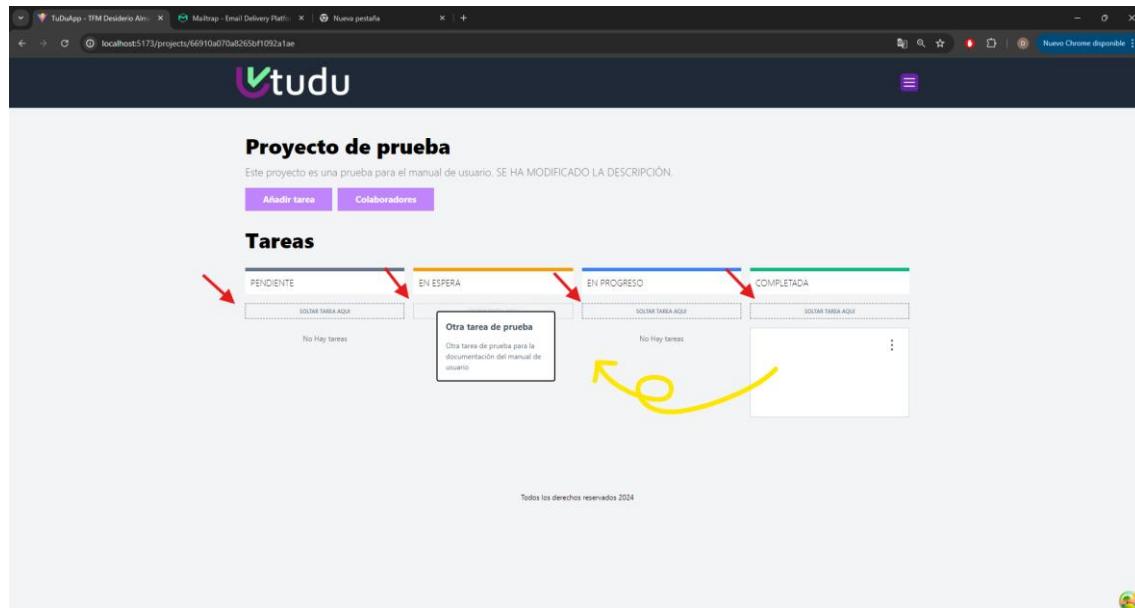
- Cambiar estado de tareas

Se puede realizar de dos formas desde el modal de “Ver tarea” o desde el propio tablero de tareas del proyecto.

Para cambiarlo desde el modal de “Ver tarea” únicamente hay que pulsar en el estado y se mostrarán las opciones:

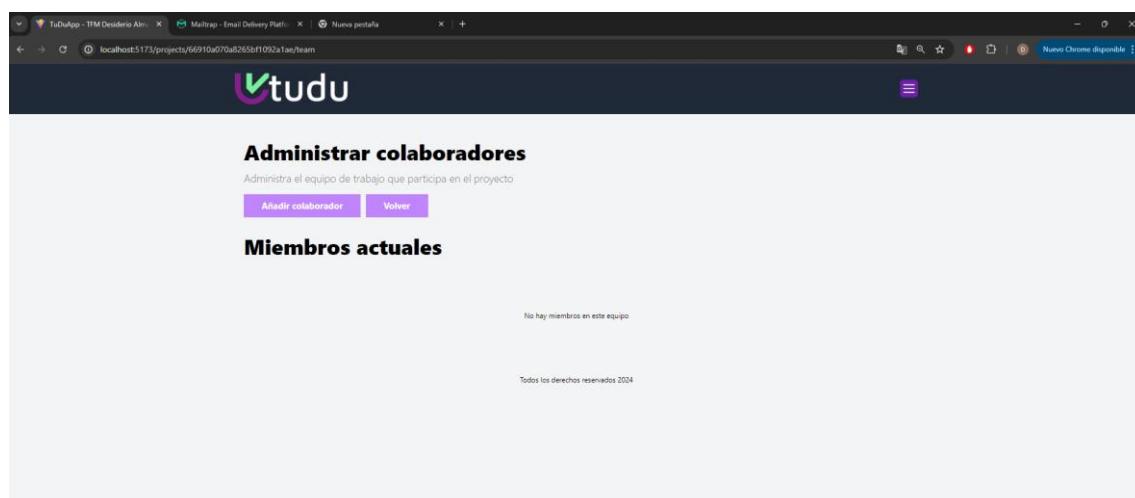


Para hacerlo desde el tablero, simplemente tenemos que pinchar en la tarea y sin soltarla, arrastrarla hasta el recuadro que dice “Soltar tarea aquí” debajo de cada estado.



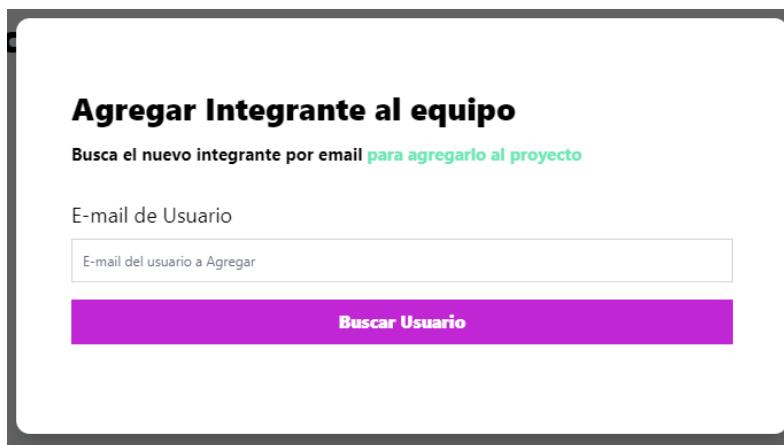
- Colaboradores (permitido a Manager del proyecto)

Si accedemos a la ventana de colaboradores a través del botón “Colaboradores” se mostrará los miembros del equipo que están trabajando en el proyecto. También se dará la opción de añadir un nuevo colaborador al equipo con el botón “Añadir colaborador” o de volver al dashboard de proyectos con el botón “Volver”.

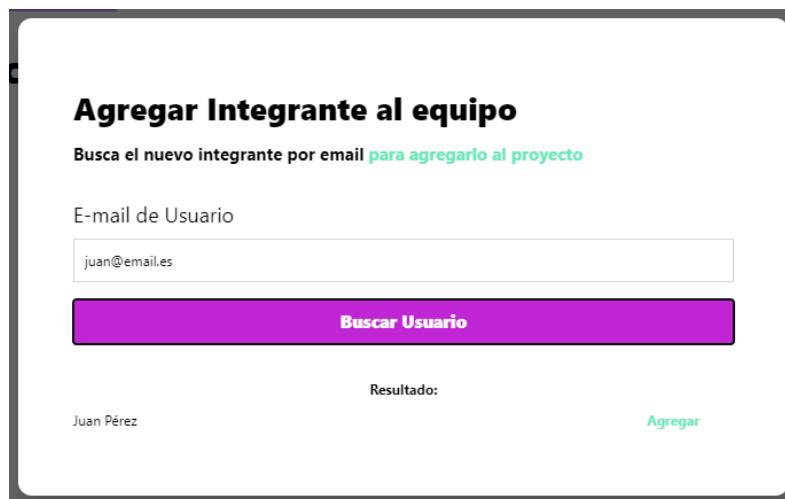


- Añadir colaboradores (permitido a Manager del proyecto)

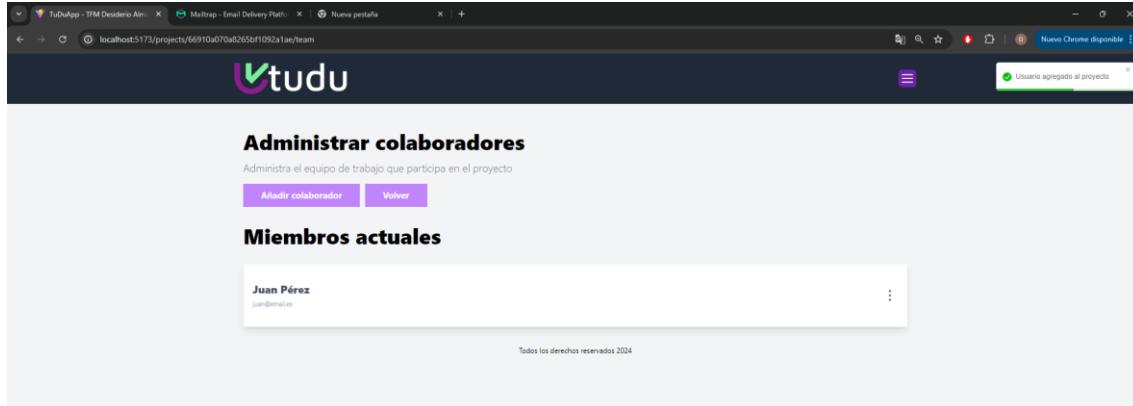
Para añadir colaboradores debemos de hacer clic en “Añadir colaborador” y nos abrirá un modal en el que podremos buscar el usuario.



Tras ingresar el correo del usuario, nos mostrará en la parte inferior si lo ha encontrado o no. En caso de encontrarlo nos dará la opción de añadirlo a través de un botón verde “Aregar”.



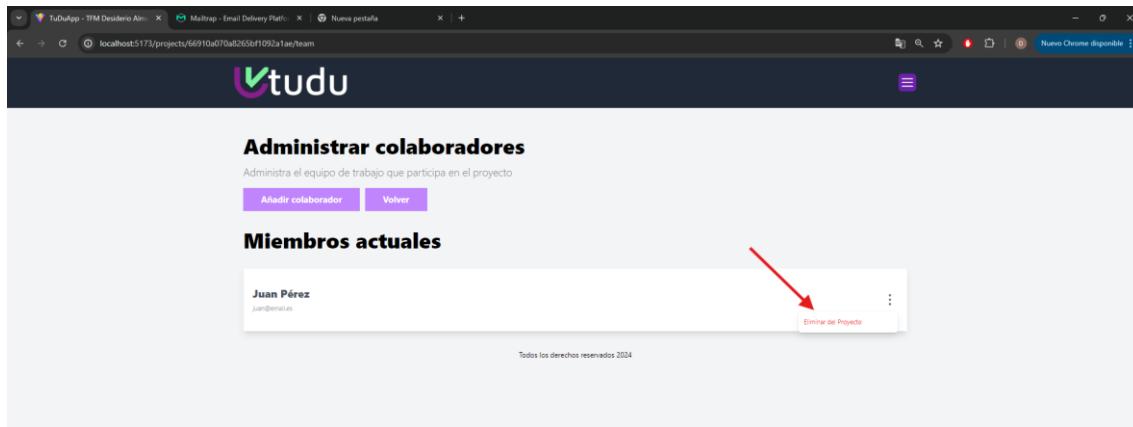
Tras pulsar el botón de agregar, se cerrará el modal y nos mostrará los miembros del equipo actualizados.



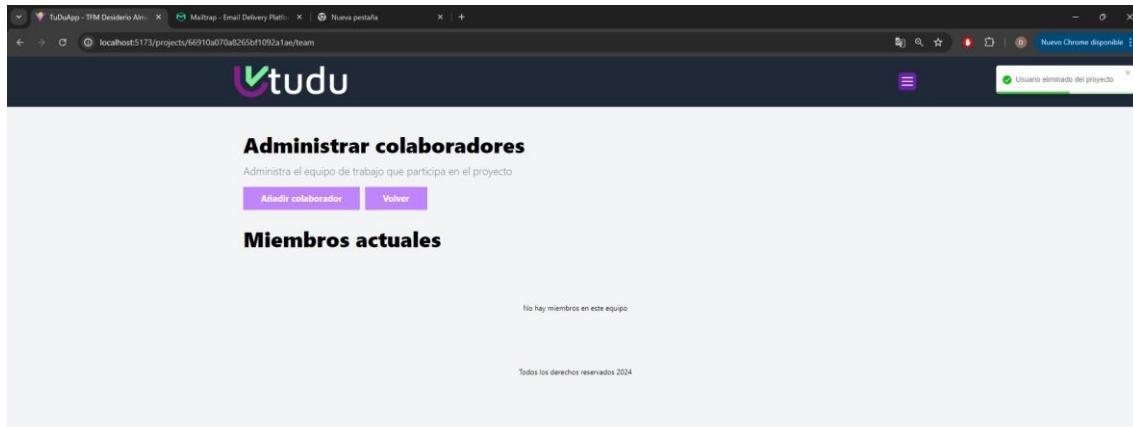
The screenshot shows the 'Administrar colaboradores' (Manage Collaborators) page. At the top, there are two buttons: 'Añadir colaborador' (Add collaborator) and 'Volver' (Back). Below this, the heading 'Miembros actuales' (Current Members) is displayed. A single member, 'Juan Pérez' (juan@enlaces), is listed. To the right of the member's name is a three-dot menu icon. At the bottom of the page, a copyright notice reads 'Todos los derechos reservados 2024'.

- [Eliminar colaboradores](#) (permitido a Manager del proyecto)

Para eliminar colaboradores simplemente debemos pulsar en el ícono de 3 puntos de la derecha de cada colaborador, que nos mostrará la opción de "Eliminar del Proyecto". Si pulsamos el colaborador se eliminará y se actualizarán los colaboradores.



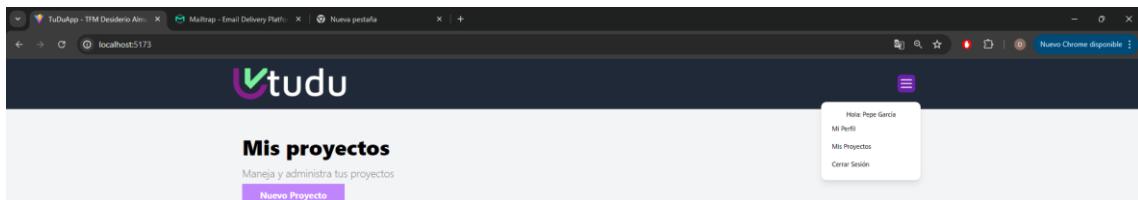
This screenshot is identical to the one above, but a red arrow points to the three-dot menu icon next to 'Juan Pérez'. A tooltip labeled 'Eliminar del Proyecto' (Delete from Project) is visible over the menu icon.



This screenshot shows the same interface after the deletion. A green success message 'Usuario eliminado del proyecto' (User deleted from project) is displayed at the top right. The member 'Juan Pérez' is no longer listed in the 'Miembros actuales' section. A small note at the bottom states 'No hay miembros en este equipo' (There are no members in this team).

- Barra superior de navegación

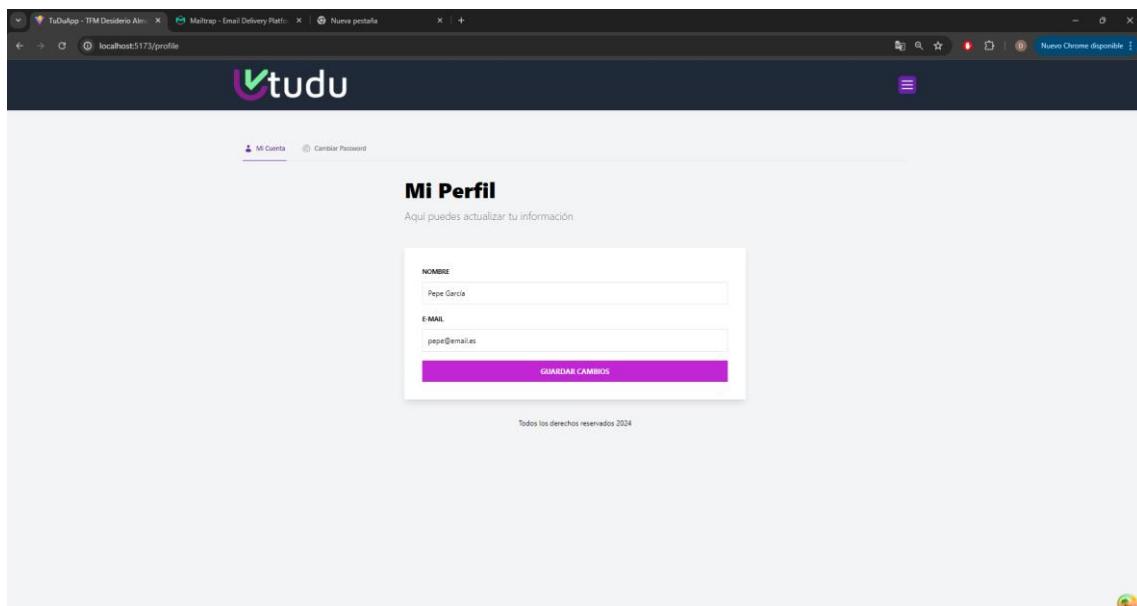
En la barra superior podremos ver el logo y un ícono morado con 3 rayas. Si hacemos clic en el logo la aplicación nos llevará al dashboard principal de “Mis proyectos”. Si hacemos clic en el ícono nos desplegará un menú de opciones en el que también podremos ver nuestro nombre de usuario.



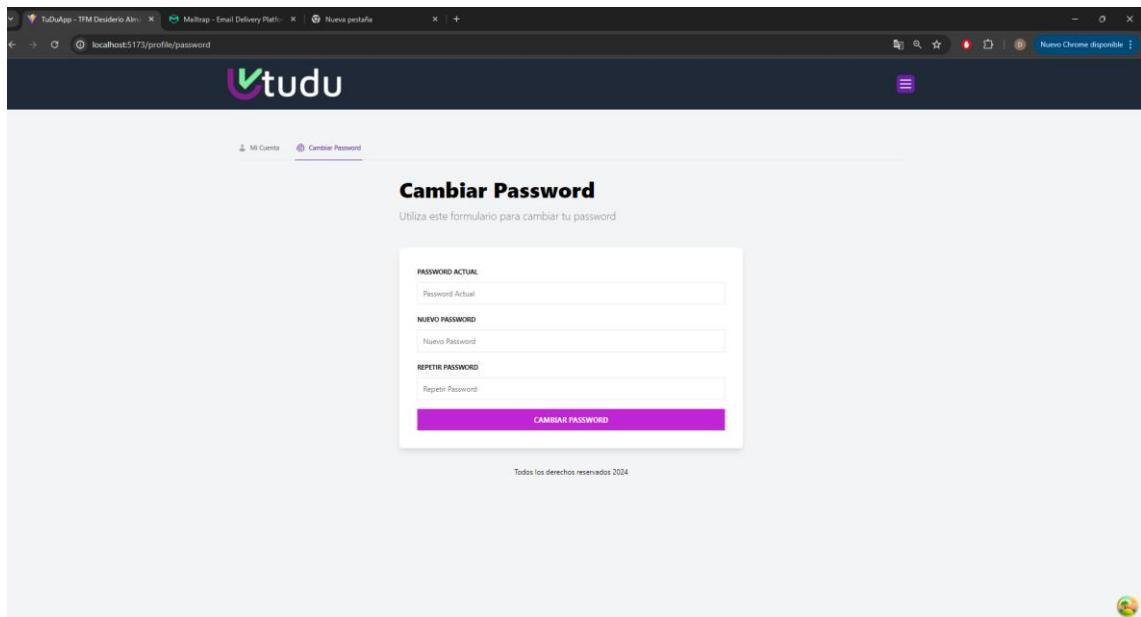
La opción de “Mis proyectos” como su propio nombre indica nos llevará al dashboard de proyectos.

- Mi Perfil

Si pulsamos la opción de “Mi perfil” de la barra superior, nos dirigirá a la vista del perfil en la que podremos editar nuestra información y cambiar la contraseña.



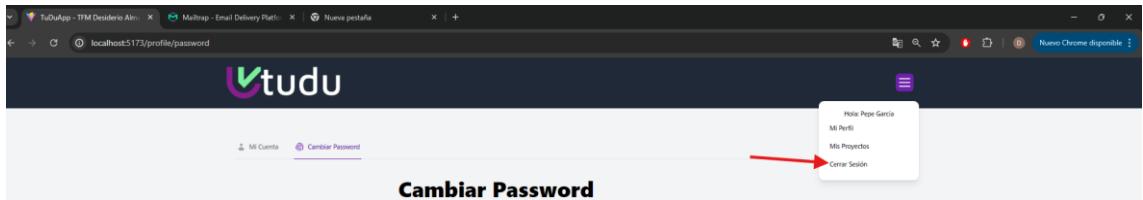
Para editar nuestra información únicamente debemos modificar los campos “Nombre” o “E-Mail” y pulsar “Guardar cambios”.



Para actualizar la contraseña debemos de ingresar la contraseña actual y la nueva (2 veces) y pulsar “Cambiar password”.

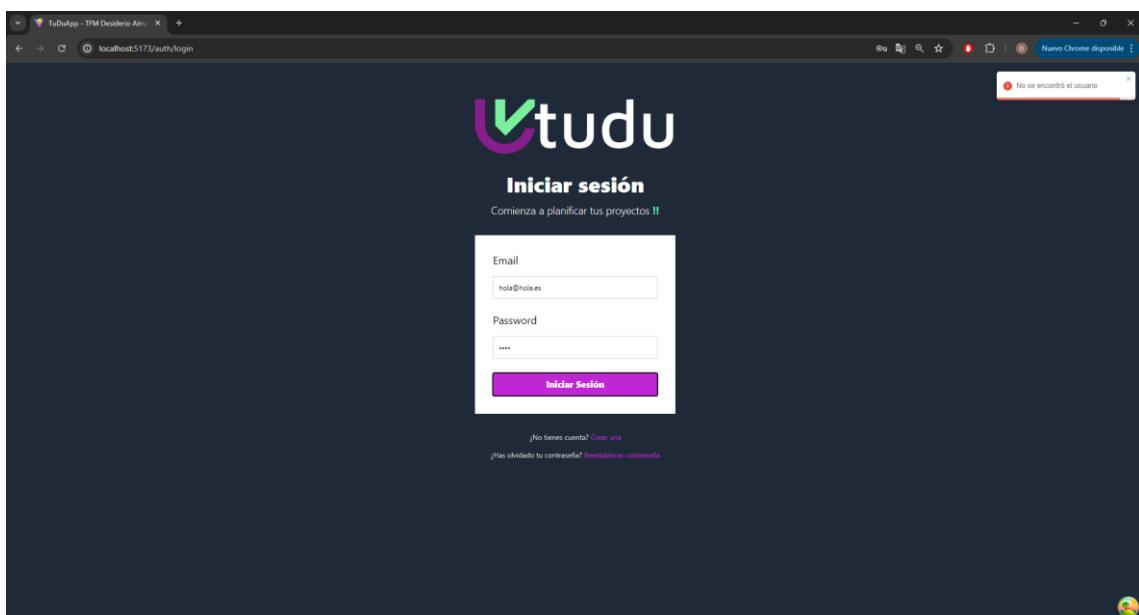
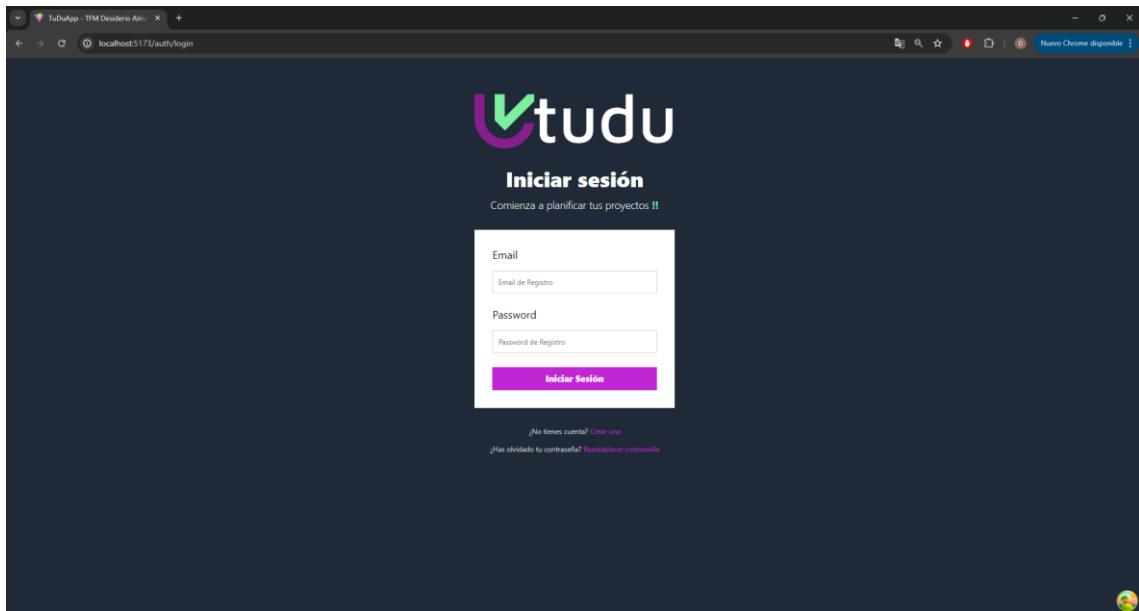
- Cerrar sesión

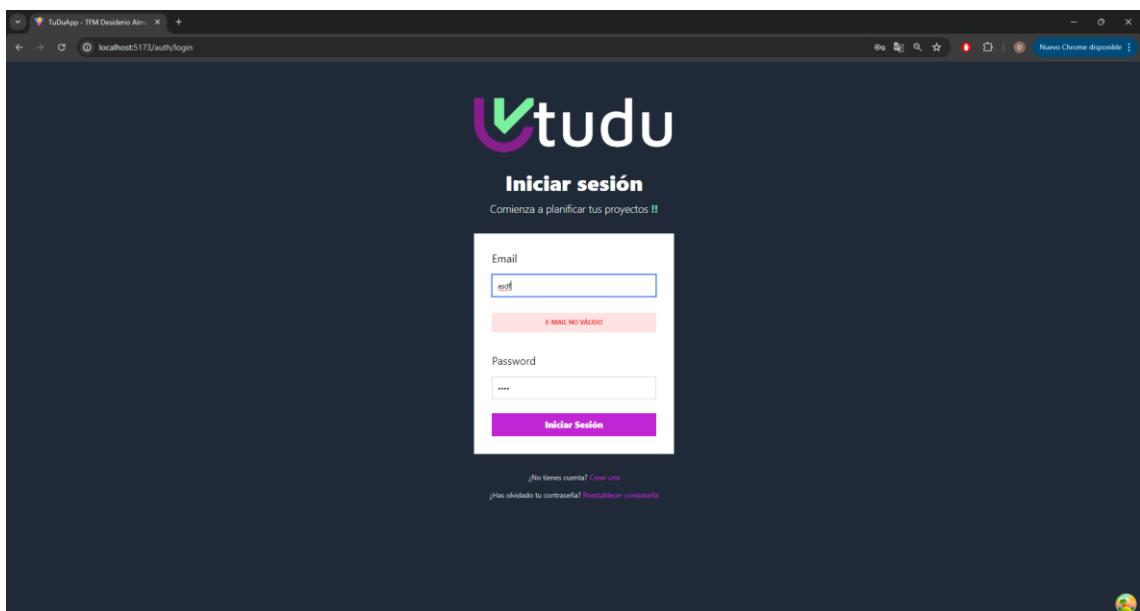
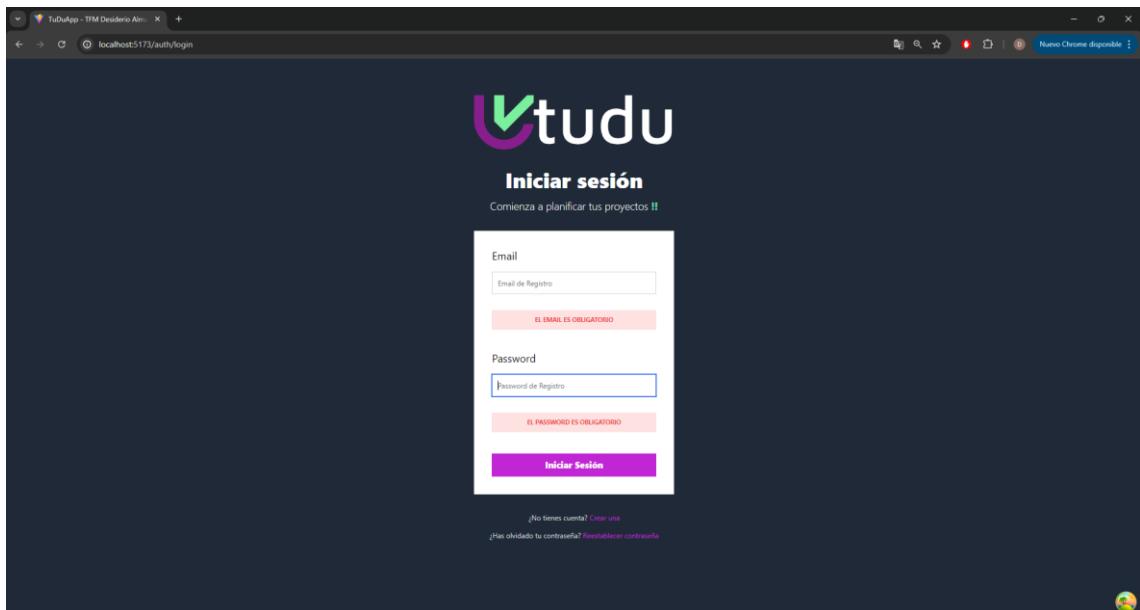
Para cerrar sesión debemos pulsar la opción de “Cerrar sesión” de la barra superior.

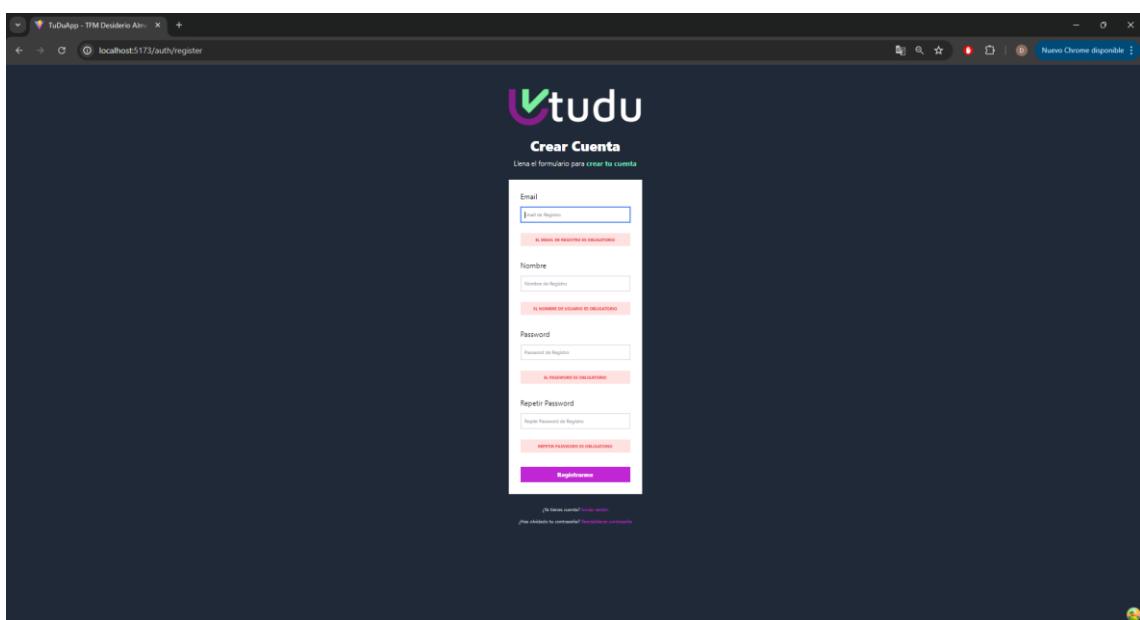
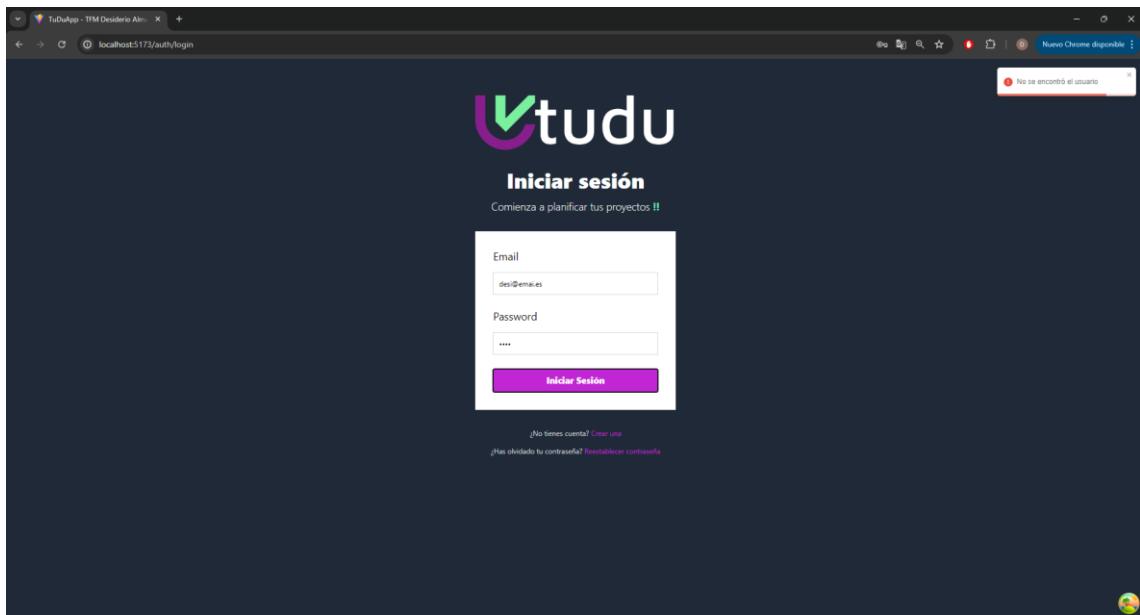


Tras hacerlo nos redirigirá a la ventana de inicio de sesión.

8.3. Capturas de pantalla







TuDuApp - TiM Desiderio Alm... X +

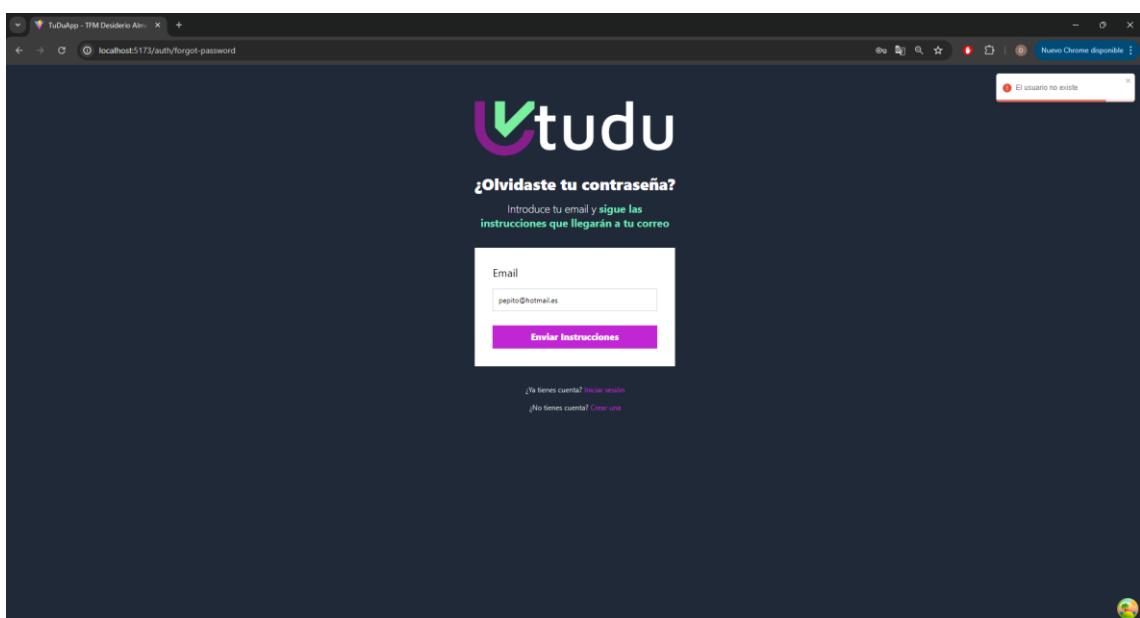
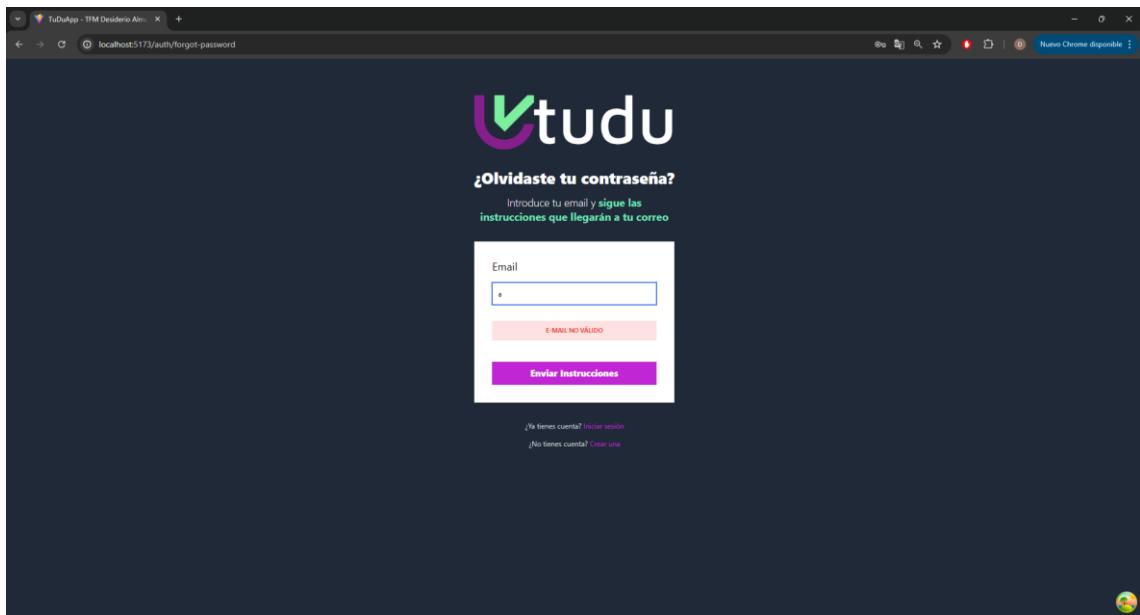
localhost:5173/auth/register

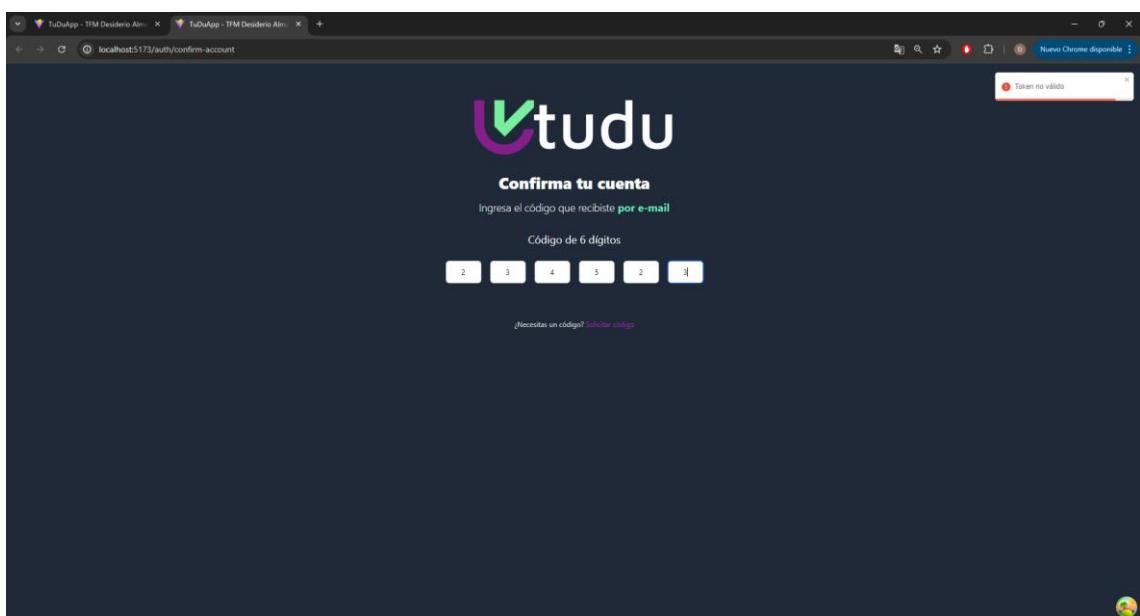
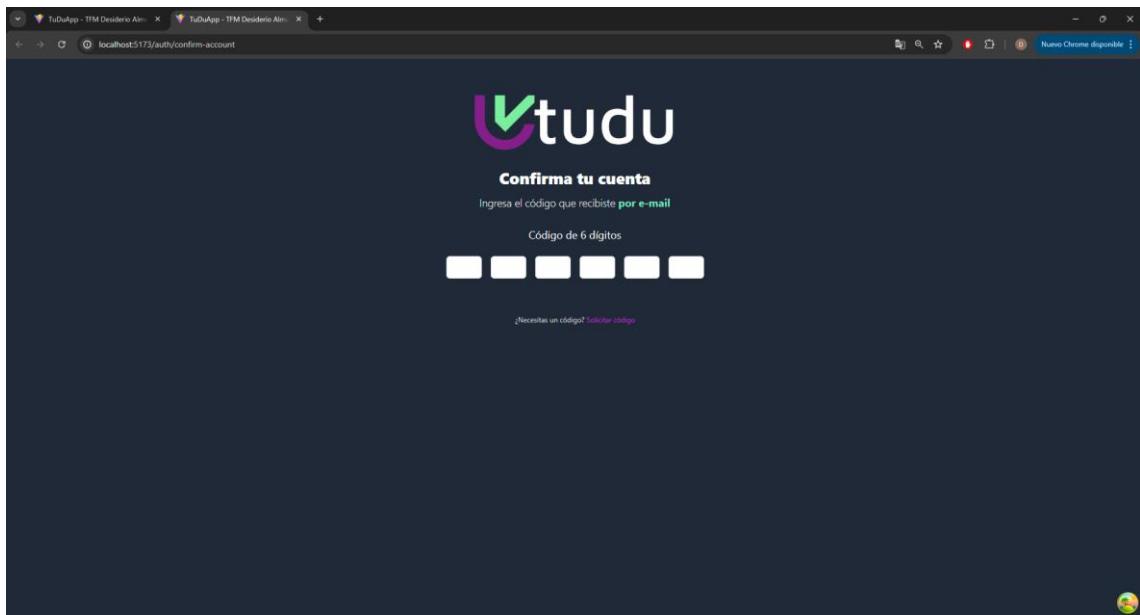
Nuevo Chrome disponible

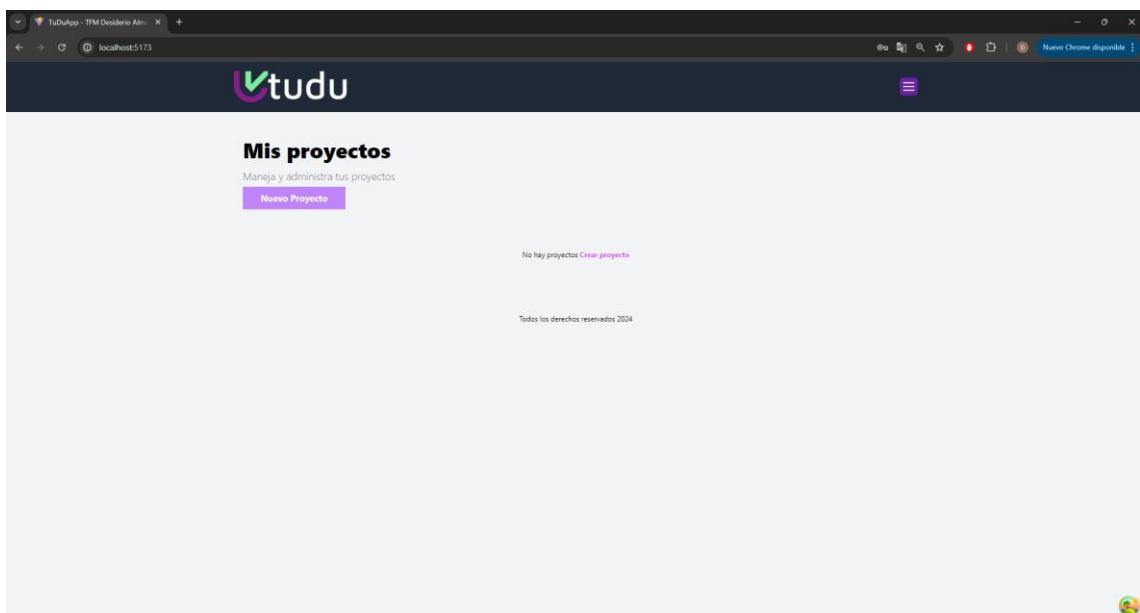
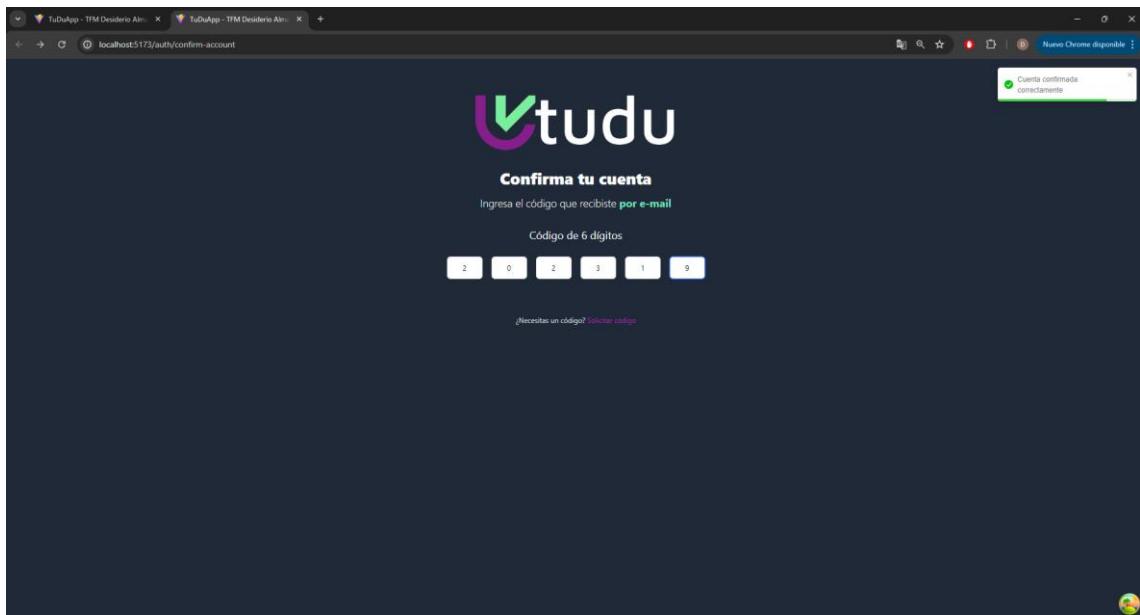
TuDuApp - TiM Desiderio Alm... X +

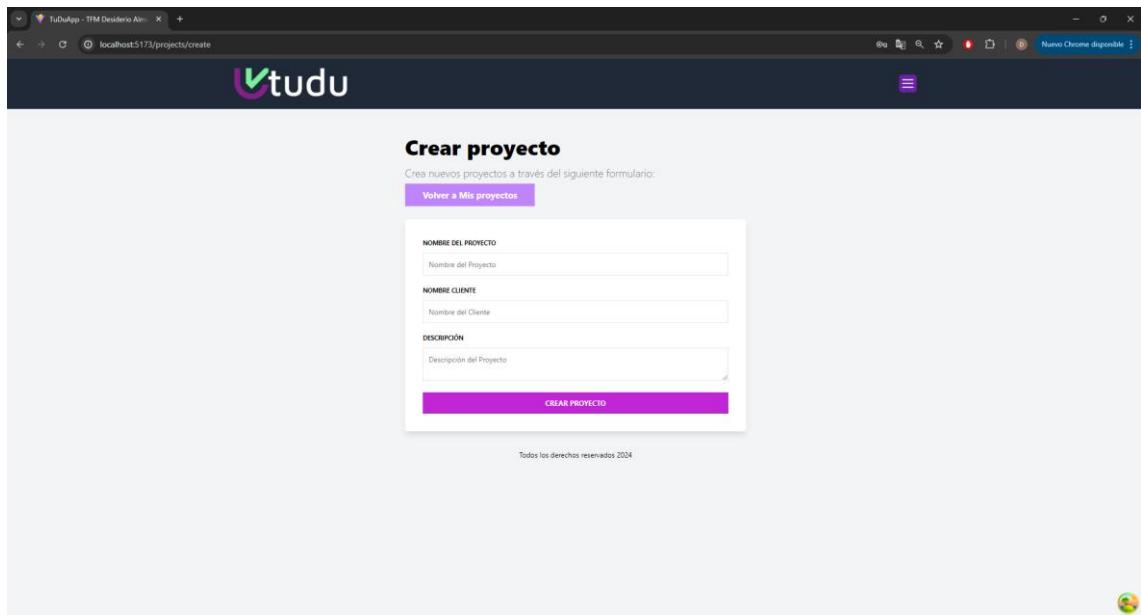
localhost:5173/auth/forgot-password

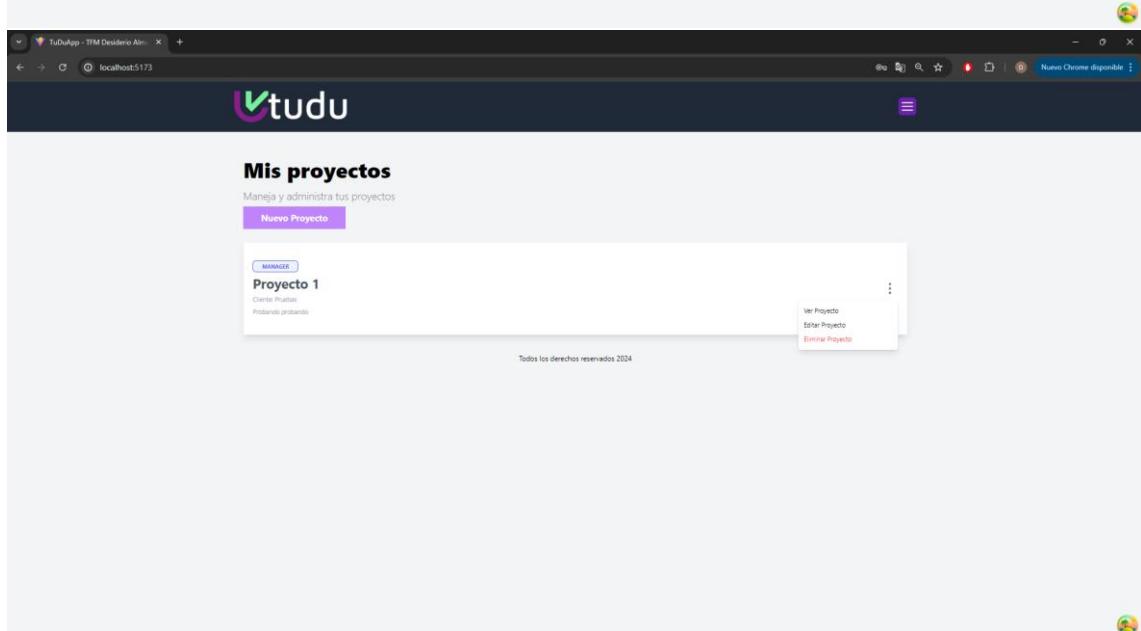
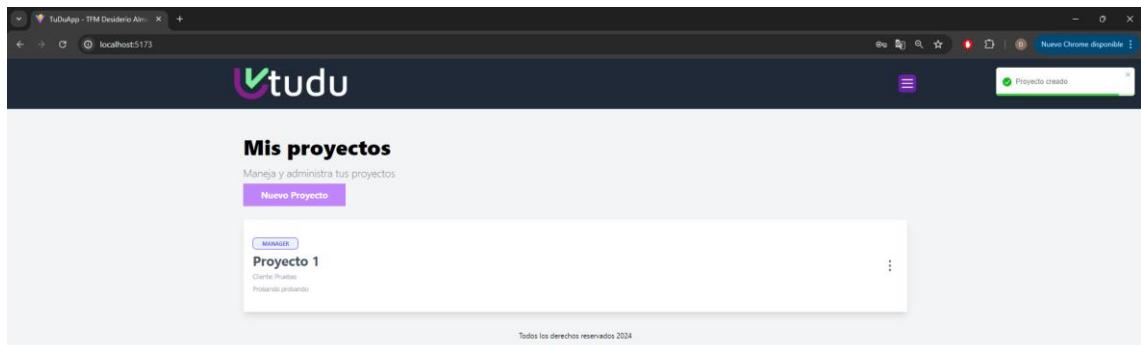
Nuevo Chrome disponible











TuDuApp - TiM Desiderio Alm... X +

localhost:5173/projects/6691cdafe9770eb74-443bb/edit

utudu

Editar proyecto

Edita el proyecto a través del siguiente formulario:

[Volver a Mis proyectos](#)

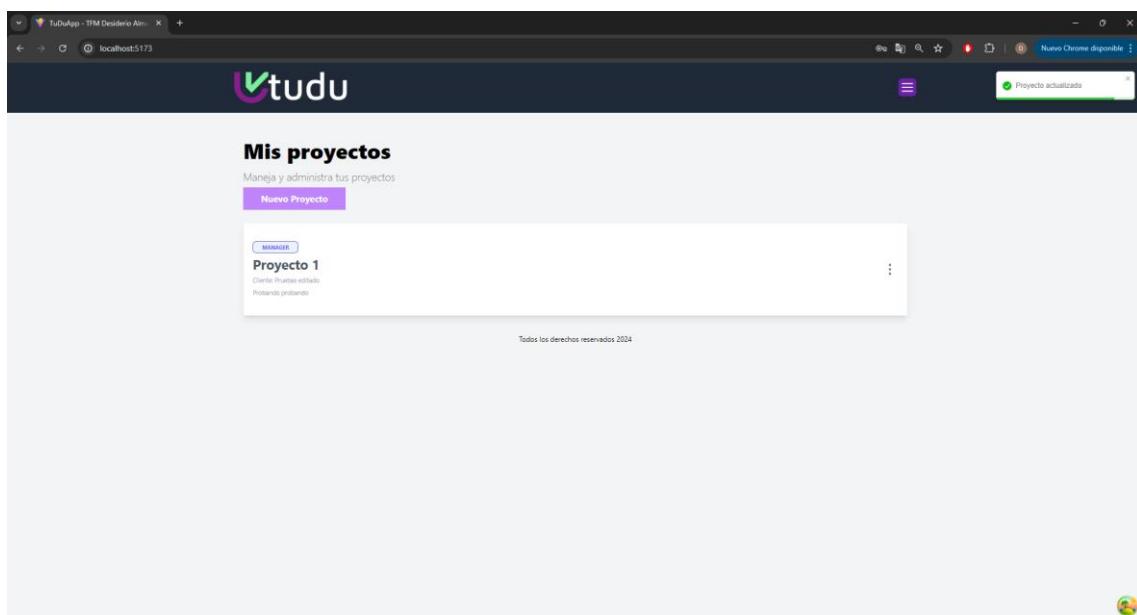
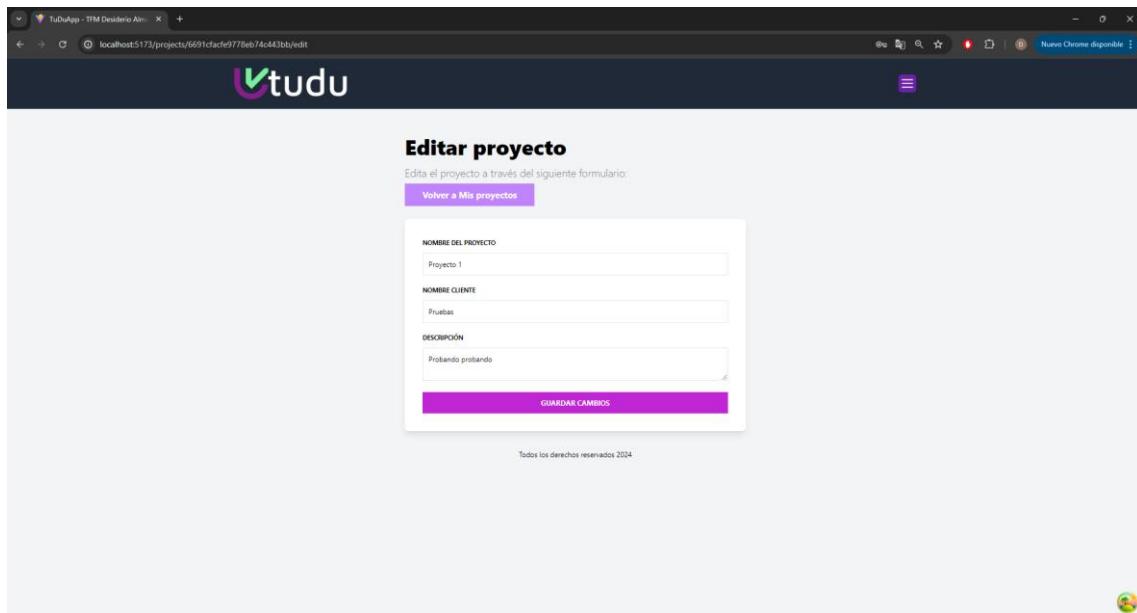
NOMBRE DEL PROYECTO
Proyecto 1

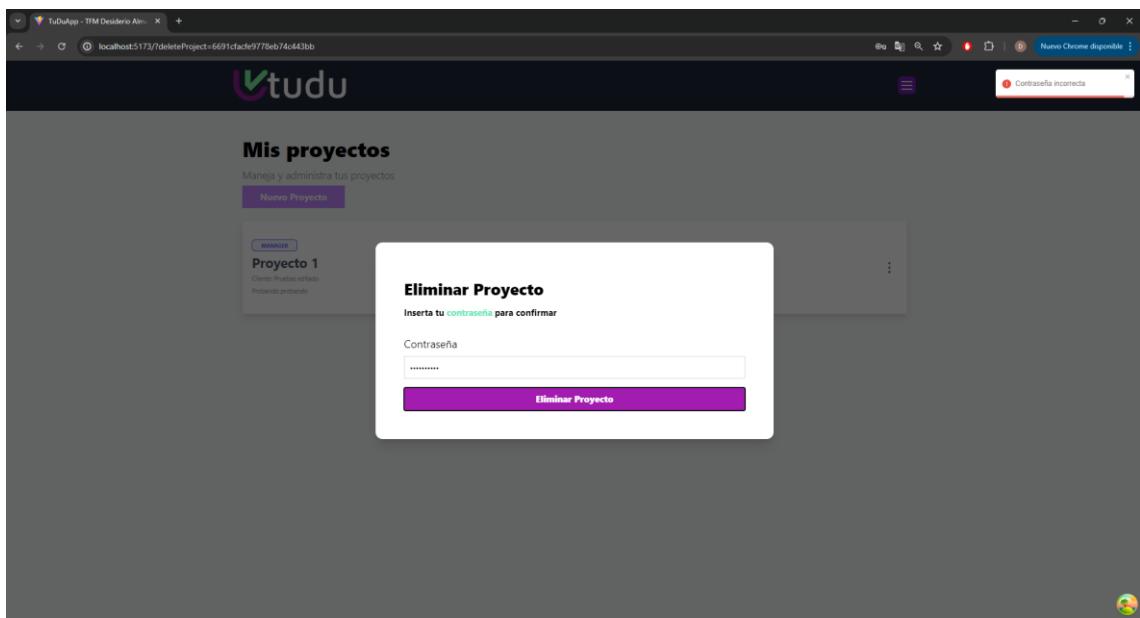
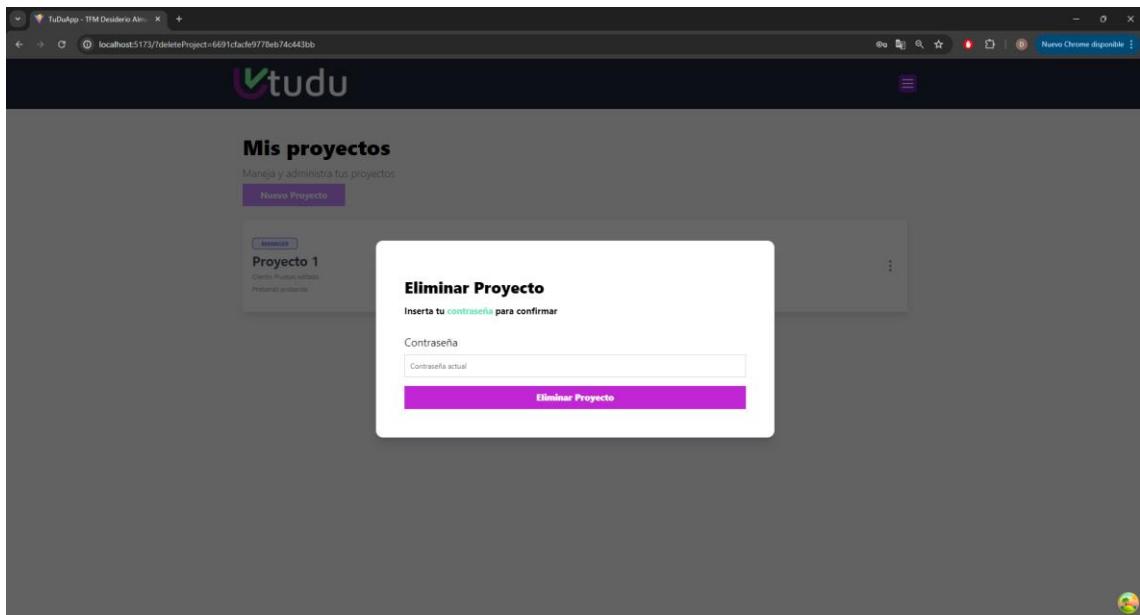
NOMBRE CLIENTE
Pruebas

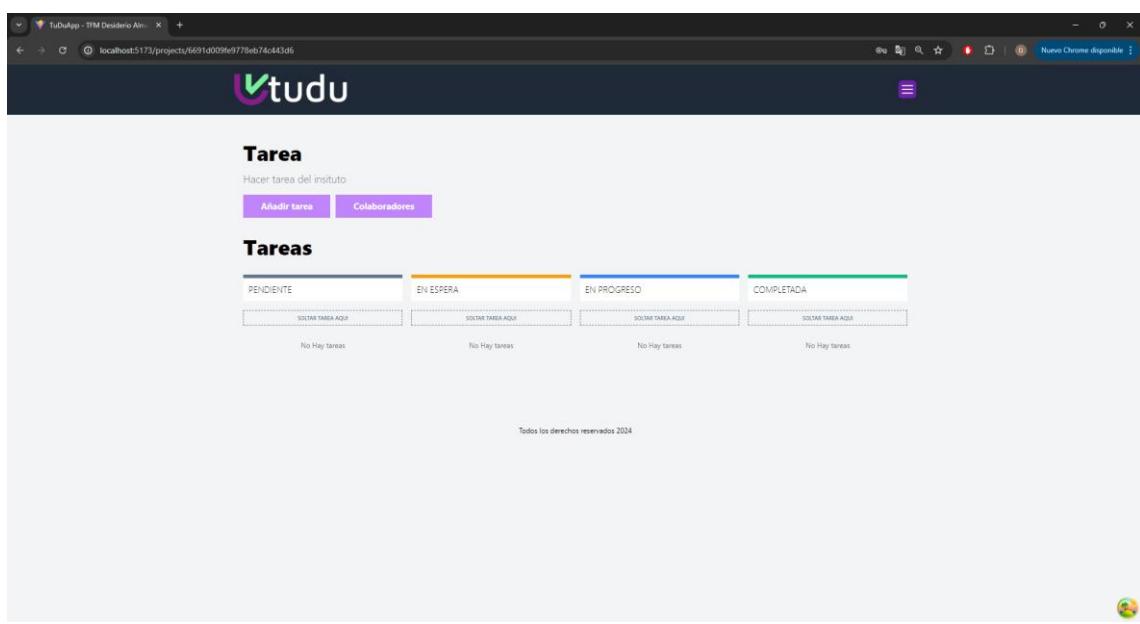
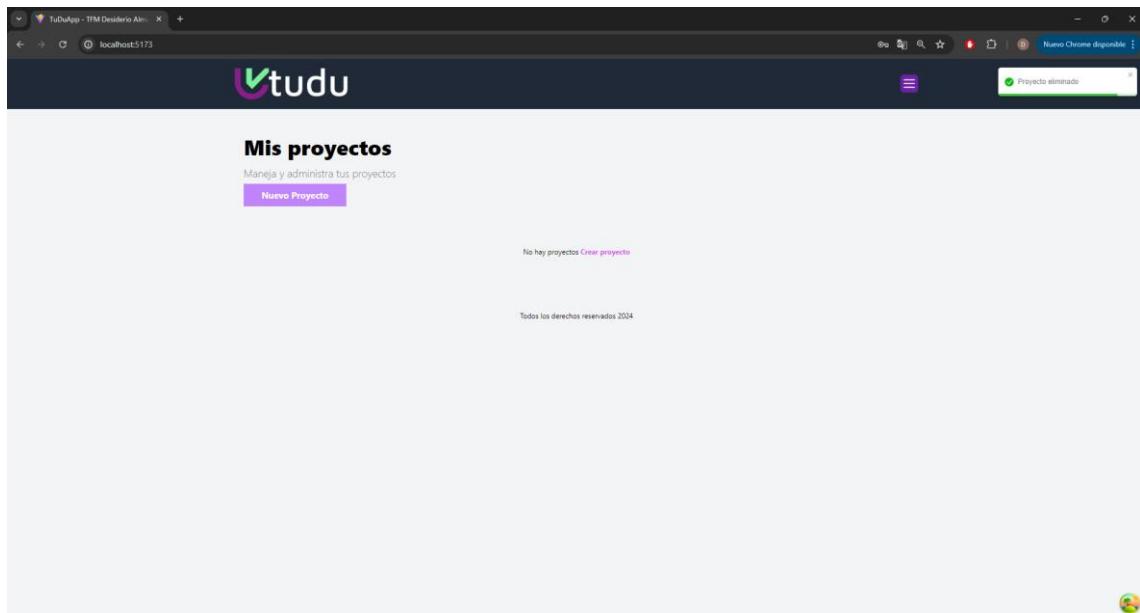
DESCRIPCIÓN
Probando probando

GUARDAR CAMBIOS

Todos los derechos reservados 2024







TuDuApp - TFM Desiderio Alm... X +

localhost:5173/projects/6691d009fe9778eb74c443df/team

Nuevo Chrome disponible

Vtudu

Administrar colaboradores

Administra el equipo de trabajo que participa en el proyecto

[Añadir colaborador](#) [Volver](#)

Miembros actuales

No hay miembros en este equipo

Todos los derechos reservados 2024

TuDuApp - TFM Desiderio Alm... X +

localhost:5173/projects/6691d009fe9778eb74c443df/team?addMember=true

Nuevo Chrome disponible

Vtudu

Administrar colaboradores

Administra el equipo de trabajo que participa en el proyecto

[Añadir colaborador](#) [Volver](#)

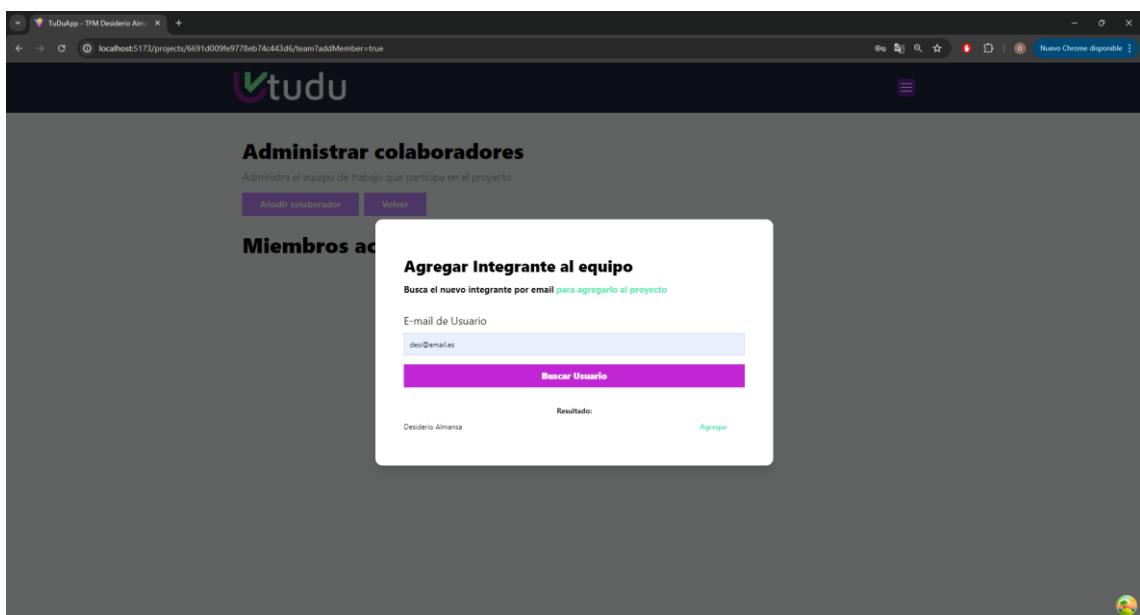
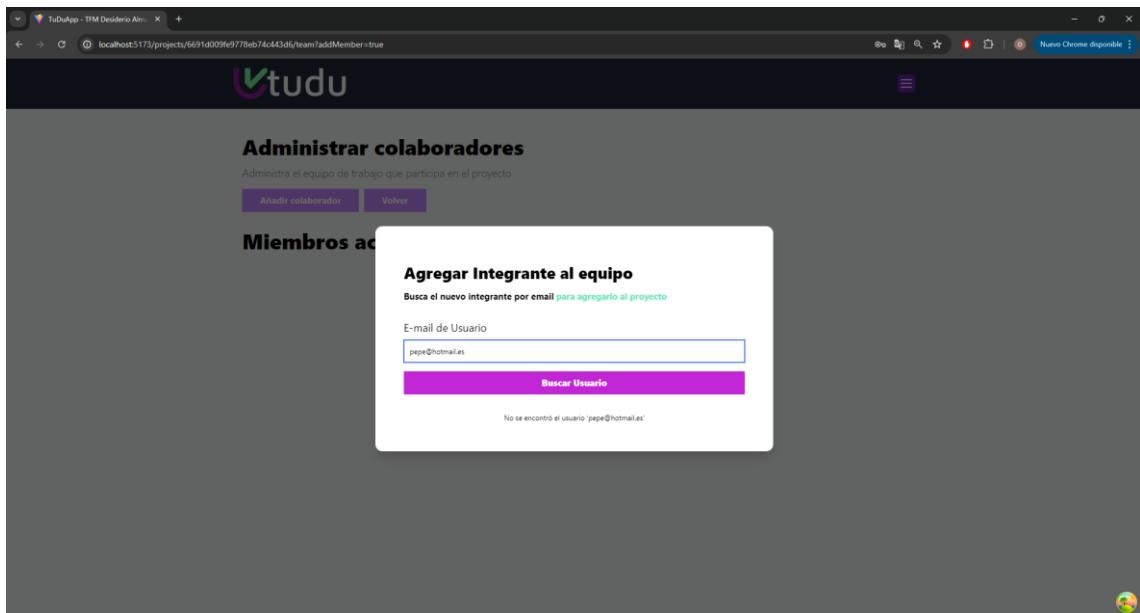
Miembros actuales

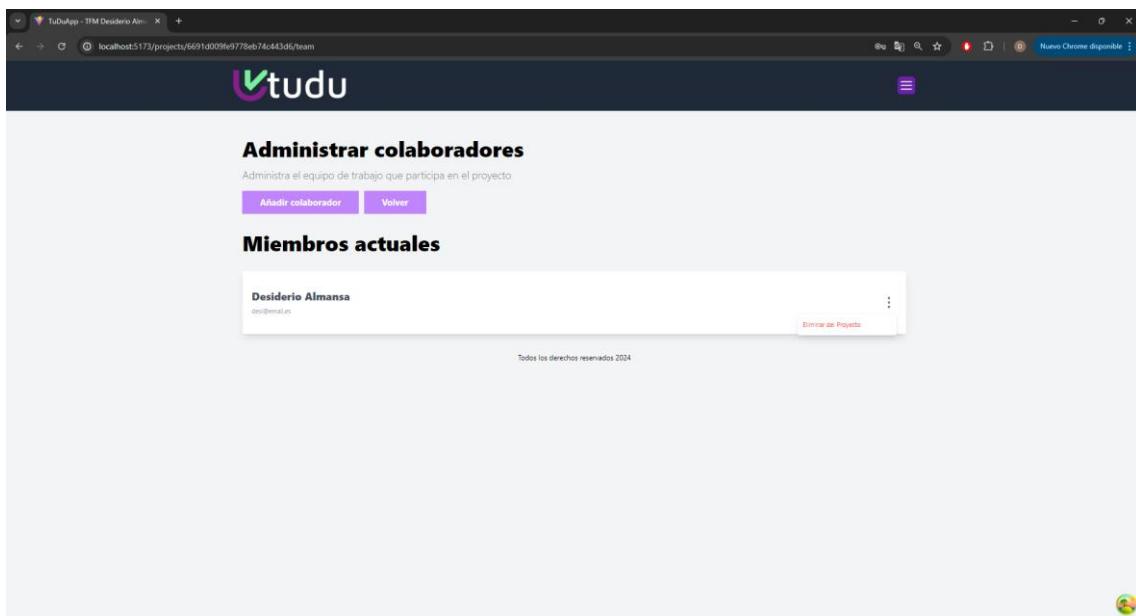
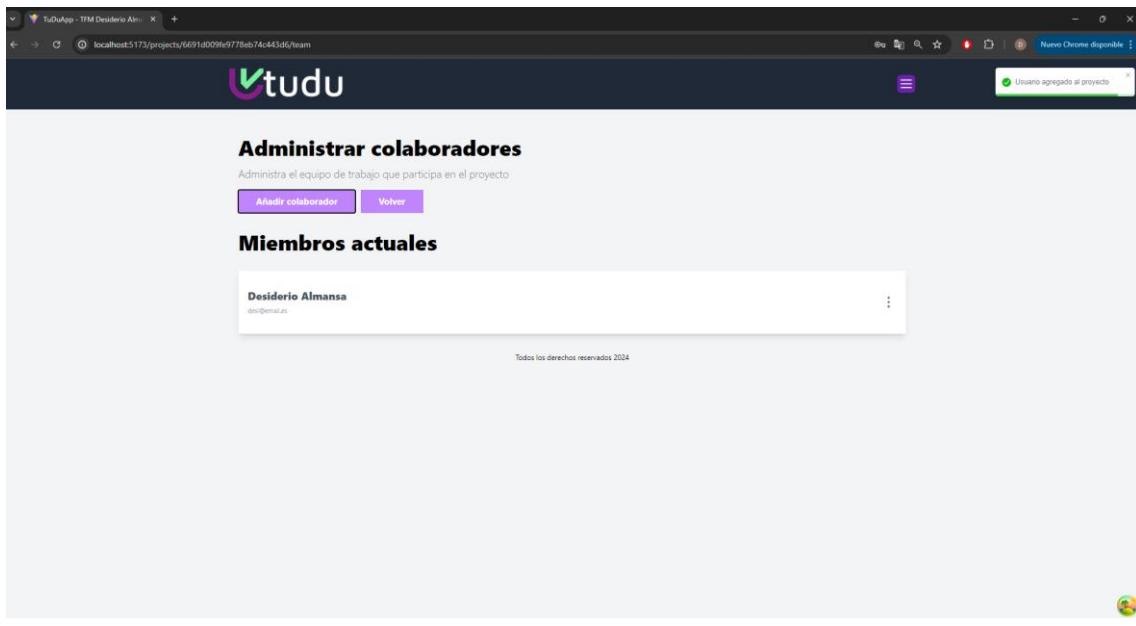
Agregar Integrante al equipo

Busca el nuevo integrante por email para agregarlo al proyecto

E-mail de Usuario

[Buscar Usuario](#)





TuDuApp - TFM Desiderio Alm... X +

localhost:5173/projects/6691d009fe9778eb74c443d6/team

Administrador de colaboradores

Administra el equipo de trabajo que participa en el proyecto

[Añadir colaborador](#) [Volver](#)

Miembros actuales

No hay miembros en este equipo

Todos los derechos reservados 2024

This screenshot shows the 'Administrador de colaboradores' (Collaborator Management) section of the TuDuApp. It displays a message stating 'No hay miembros en este equipo' (There are no members in this team). There are two buttons at the top: 'Añadir colaborador' (Add collaborator) and 'Volver' (Back). A notification bar at the top right says 'Usuario eliminado del proyecto' (User removed from the project).

TuDuApp - TFM Desiderio Alm... X +

localhost:5173/projects/6691d009fe9778eb74c443d6?newTask=true

Tarea

Hacer tarea del instituto

[Añadir tarea](#) [Colaboradores](#)

Tareas

PENDIENTE

SOLTAR TAREA AQUÍ

No Hay tareas

Nueva Tarea

Llena el formulario y crea una tarea

Nombre de la tarea

Descripción de la tarea

CREAR TAREA

FLETADA

SOLTAR TAREA AQUÍ

No Hay tareas

This screenshot shows the 'Tarea' (Task) creation page. It features a central modal window titled 'Nueva Tarea' (New Task) with fields for 'Nombre de la tarea' (Task name) and 'Descripción de la tarea' (Task description), and a 'CREAR TAREA' (Create Task) button. The background shows a dark interface with a sidebar labeled 'Tareas' (Tasks) and a status bar indicating 'PENDIENTE' (Pending) and 'FLETADA' (Planned). A message at the top of the page says 'Llena el formulario y crea una tarea' (Fill out the form and create a task). The URL in the address bar includes '?newTask=true'.

Tarea

Hacer tarea del instituto

Adadir tarea Colaboradores

Tareas

PENDIENTE	EN ESPERA	EN PROGRESO	COMPLETADA
SOLTAR TAREA AQUÍ	SOLTAR TAREA AQUÍ	SOLTAR TAREA AQUÍ	SOLTAR TAREA AQUÍ
Tarea1 Este es la tarea1	No Hay tareas	No Hay tareas	No Hay tareas

Todos los derechos reservados 2024

Tarea

Hacer tarea del instituto

Adadir tarea Colaboradores

Tareas

PENDIENTE	EN ESPERA	EN PROGRESO	COMPLETADA
SOLTAR TAREA AQUÍ	SOLTAR TAREA AQUÍ	SOLTAR TAREA AQUÍ	SOLTAR TAREA AQUÍ
No Hay tareas	Tarea1 Este es la tarea1	No Hay tareas	No Hay tareas

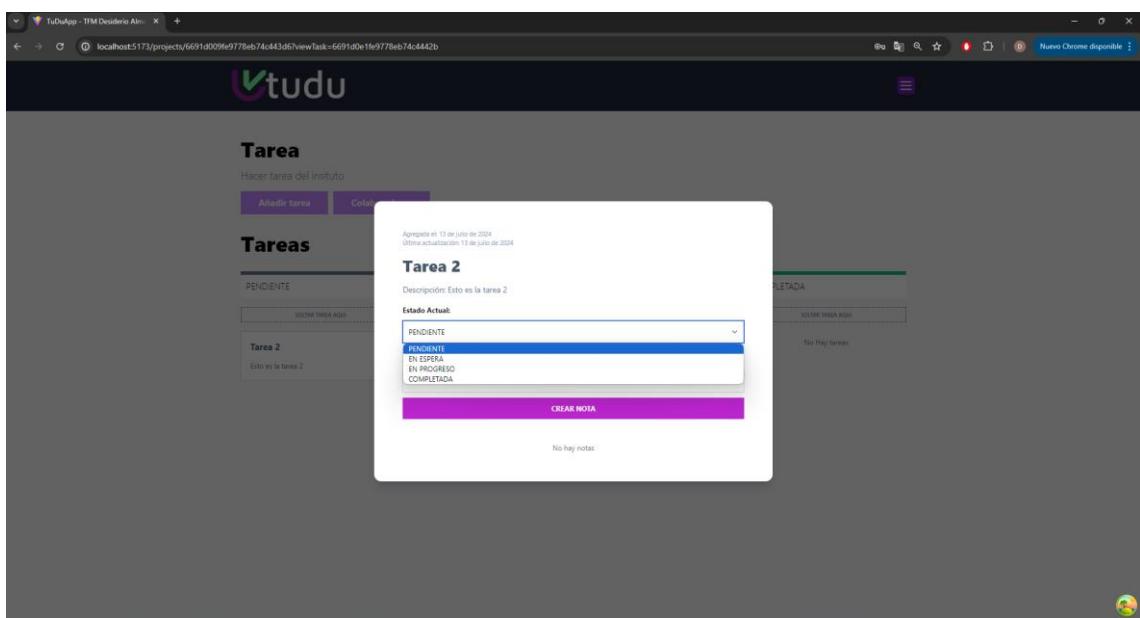
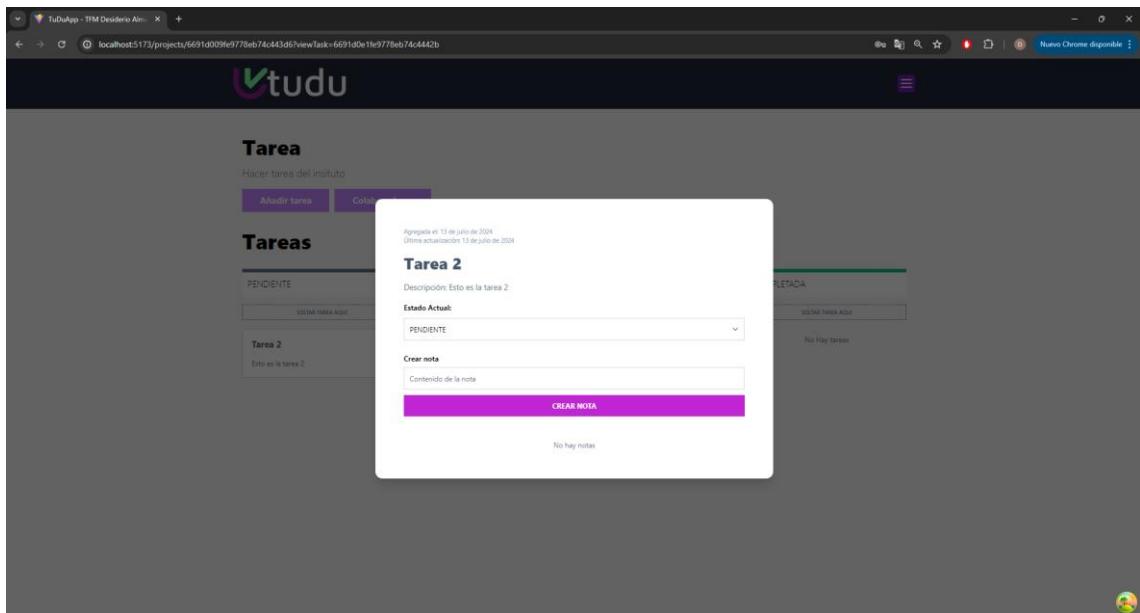
Todos los derechos reservados 2024

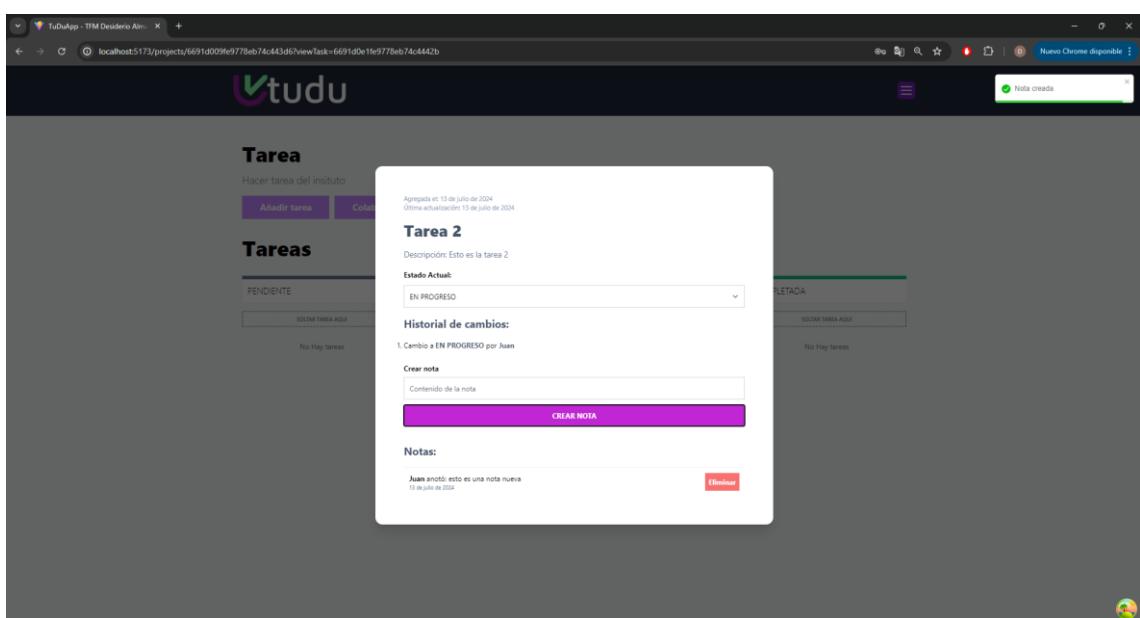
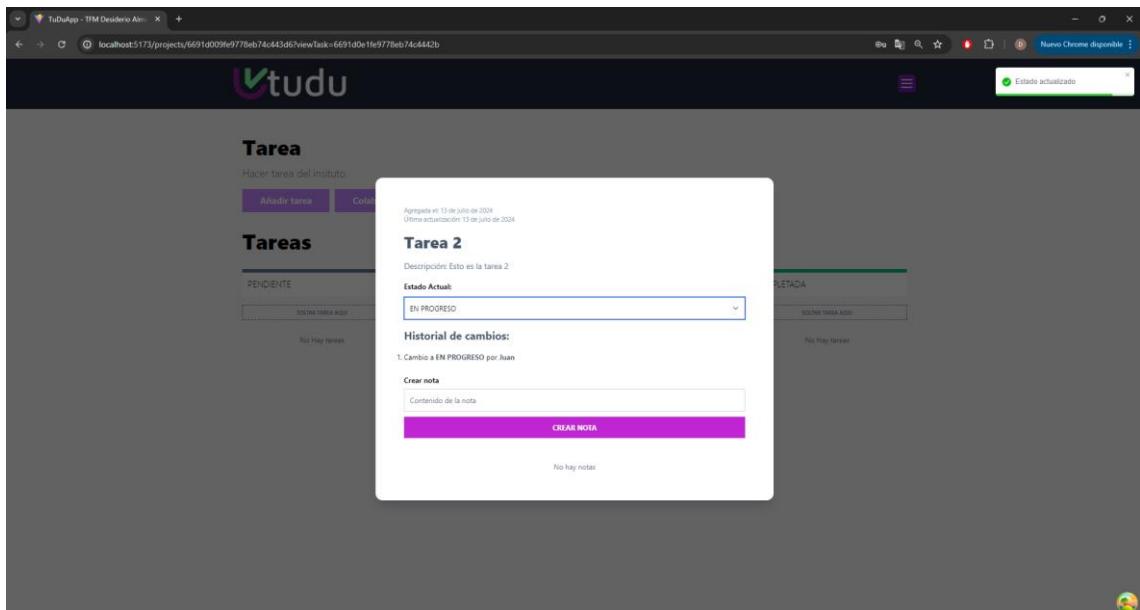
The screenshot shows the main interface of the TuDuApp task management application. At the top, there is a navigation bar with the logo 'vtudu' and a search bar. Below the navigation bar, there is a section titled 'Tarea' with a sub-section 'Hacer tarea del instituto'. There are two buttons: 'Añadir tarea' and 'Colaboradores'. The main area is titled 'Tareas' and features four tabs: 'PENDIENTE' (grey), 'EN ESPERA' (orange), 'EN PROGRESO' (blue), and 'COMPLETADA' (green). Under each tab, there is a placeholder text 'SOLTAR TAREA AQUÍ' and a message 'No Hay tareas'. In the center, there is a modal window for a task named 'Tarea1'. The modal has a title 'Tarea1', a description 'Esto es la tarea', and three buttons: 'Ver tarea', 'Editar Tarea', and 'Eliminar tarea' (in red).

The screenshot shows the 'Edit Task' form from the TuDuApp application. The form is titled 'Editar Tarea' and contains instructions 'Realiza cambios a una tarea en este formulario'. It has two input fields: 'Nombre de la tarea' (with value 'Tarea1') and 'Descripción de la tarea' (with value 'Esto es la tarea'). At the bottom, there is a large purple button labeled 'Guardar Tarea'.

The screenshot shows a web browser window for the TuDuApp task management application. The URL is `localhost:5173/projects/6691d009fe9778eb74c443d6`. The page title is "TuDuApp - TFM Desiderio Alm...". The main header features the "tudu" logo. Below the header, there's a section titled "Tarea" with the subtitle "Hacer tarea del instituto". Two buttons are visible: "Añadir tarea" and "Colaboradores". A "Tareas" section follows, displaying a horizontal navigation bar with four categories: PENDIENTE (grey), EN ESPERA (orange), EN PROGRESO (blue), and COMPLETADA (green). Under the EN ESPERA tab, there is one task card: "Tarea1 editada" with the subtitle "Esto es la tarea!". The footer contains the text "Todos los derechos reservados 2024".

This screenshot shows the same TuDuApp interface after the task has been moved from the EN ESPERA category to the COMPLETADA category. The task card "Tarea1 editada" is now displayed under the green "COMPLETADA" tab. The other three categories (PENDIENTE, EN ESPERA, and EN PROGRESO) are empty. The rest of the page structure, including the header, buttons, and footer, remains identical to the first screenshot.





The screenshot shows a web browser window for 'TuDuApp - TIM Desiderio Alm...' at 'localhost:5173'. The main interface has a dark header with the 'vtudu' logo. Below it, there's a sidebar titled 'Tareas' with sections for 'PENDIENTE' and 'COMPLETADA'. A central area displays a list of tasks. A modal dialog box is open, centered over the tasks. The modal has a white background and contains the following information:

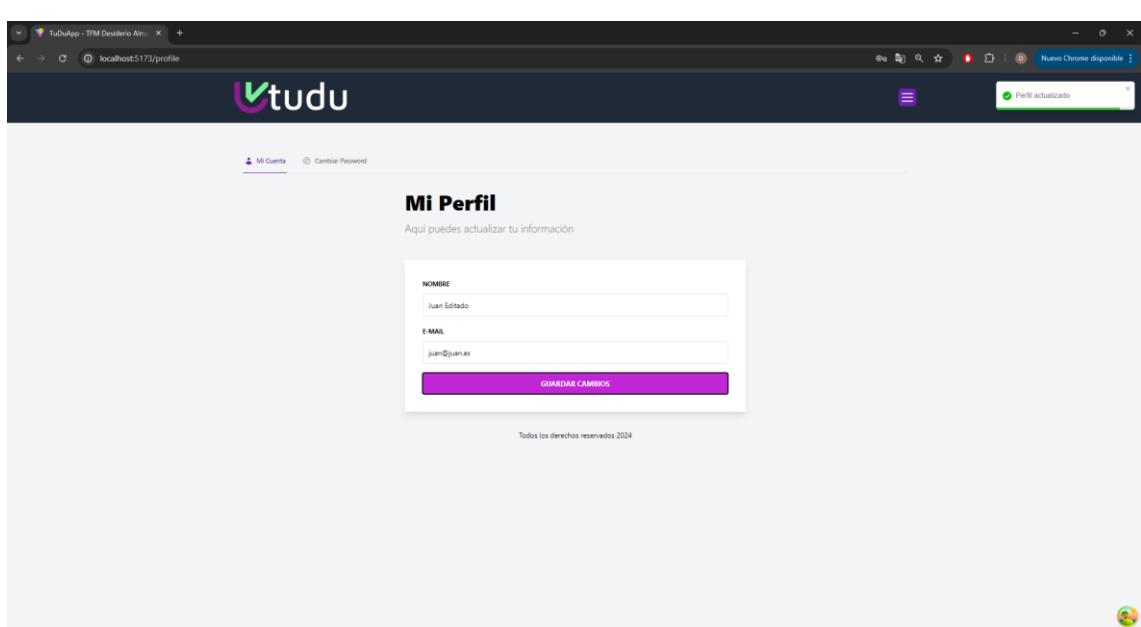
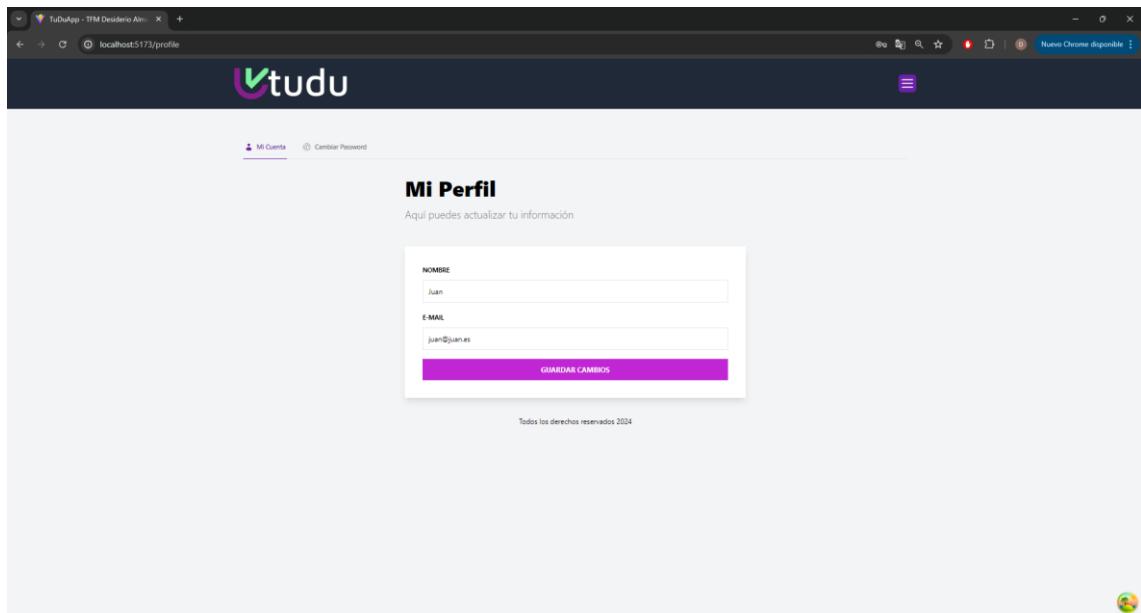
- Tarea 2**
- Descripción: Esto es la tarea 2.
- Estado Actual: EN PROGRESO
- Historial de cambios:
 - Cambio a EN PROGRESO por Juan
- Crear nota:
Contenido de la nota
CLEAR NOTA

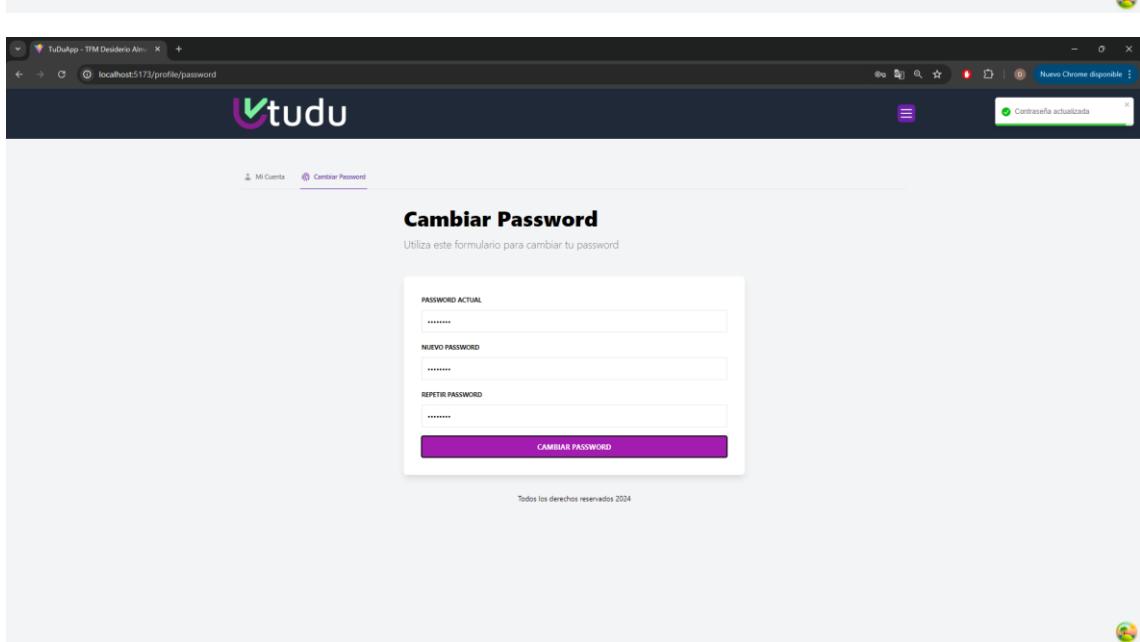
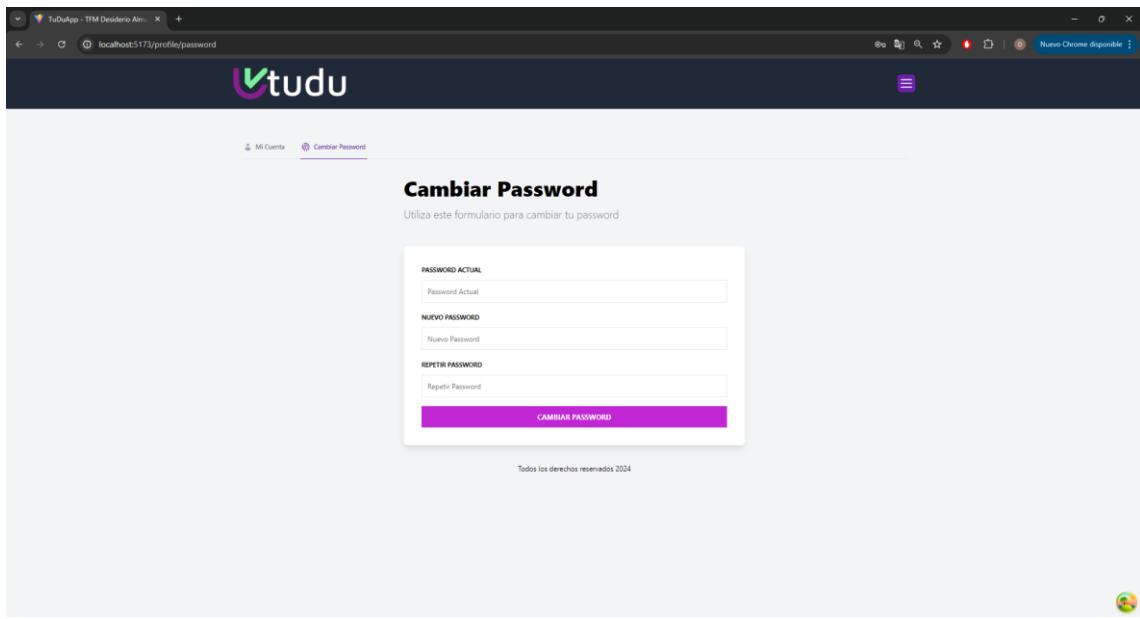
At the bottom of the modal, it says 'No hay notas'.

The screenshot shows a web browser window for 'TuDuApp - TIM Desiderio Alm...' at 'localhost:5173'. The main interface has a dark header with the 'vtudu' logo. Below it, there's a sidebar titled 'Mis proyectos' with the sub-section 'Tarea'. A user profile dropdown menu is open in the top right corner, showing the following options:

- Hola Juan
- Mi Perfil
- Mis Proyectos
- Cerrar Sesión

At the bottom of the page, it says 'Todos los derechos reservados 2024'.





9. BIBLIOGRAFÍA

- AP, V. (2023). Obtenido de <https://apvarun.github.io/toastify-js/>
- Caballero, M. R. (2019). *Apuntes de Ingeniería del Software II. Metodologías de Desarrollo*. Ciudad Real: UCLM.
- Diván, M. J. (2007). *Concepto de Desarrollo Iterativo*.
- Erickson, J. (30 de 1 de 2024). Obtenido de <https://www.oracle.com/es/database/mern-stack/>
- Express. (11 de 7 de 2024). Obtenido de <https://expressjs.com/>
- Mockup, B. (s.f.). Obtenido de <https://balsamiq.com/>
- Mongo. (11 de 7 de 2024). Obtenido de <https://www.mongodb.com/>
- Precognis. (10 de Ago de 2022). Obtenido de <https://www.precognis.com/blog/modelo-vista-controlador/>
- ProductHunt. (s.f.). *Thunder Client*. Obtenido de <https://www.thunderclient.com/>
- React. (11 de 7 de 2024). Obtenido de <https://es.react.dev/>
- Rilsware. (s.f.). Obtenido de <https://mailtrap.io/>
- Solano-Fernández, D. P.-A. (2020). *El modelo iterativo e incremental para el desarrollo de la aplicación de realidad aumentada*. AmónR .
- Tailwind. (11 de 7 de 2024). Obtenido de <https://tailwindcomponents.com/>
- TypeScript. (11 de 7 de 2024). Obtenido de <https://www.typescriptlang.org/>
- Wikipedia. (s.f.). Obtenido de <https://es.wikipedia.org/wiki/Modelo%20%93vista%20%93controlador>
- Mongo. (11 de 7 de 2024). How to use Mern Stack: A complete guide. Obtenido de <https://www.mongodb.com/resources/languages/mern-stack-tutorial>
- Express. (2024). Framework Express js. Obtenido de <https://expressjs.com/>