

PROJECT TRACK

Especificación del Sistema



11 de septiembre de 2024

CodeMentoria

Contenido

[Introducción 2](#_Toc176978824)

[Propósito 2](#_Toc176978825)

[Glosario de términos 2](#_Toc176978826)

[Descripción del proyecto 3](#_Toc176978827)

[Backend 3](#_Toc176978828)

[Interfaz 3](#_Toc176978829)

[Dependencias Principales 4](#_Toc176978830)

[Requisitos del sistema 4](#_Toc176978831)

[Casos de uso 5](#_Toc176978832)

[Asignación de Tareas 5](#_Toc176978833)

[Modelado del sistema 7](#_Toc176978834)

[Diagrama de clases 7](#_Toc176978835)

[Prototipos y bocetos 8](#_Toc176978836)

[Análisis técnico 9](#_Toc176978837)

[1. Tecnologías utilizadas 9](#_Toc176978838)

[2. Requisitos del sistema 10](#_Toc176978839)

[3. Diagrama de infraestructura 10](#_Toc176978840)

[4. Consideraciones de escalabilidad y seguridad 10](#_Toc176978841)

[Planificación del proyecto 12](#_Toc176978842)

[1. Cronograma 12](#_Toc176978843)

[2. Hitos y entregables 12](#_Toc176978844)

[3. Recursos necesarios 12](#_Toc176978845)

[4. Riesgos y plan de mitigación 13](#_Toc176978846)

[Anexos 14](#_Toc176978847)

# Introducción

**Project Track** es una plataforma de gestión de proyectos diseñada para ofrecer una solución flexible y escalable, adaptada a las necesidades de los equipos de desarrollo y, en el futuro, a diversos tipos de proyectos empresariales. Aunque en su versión inicial puede parecer una aplicación básica en comparación con otras herramientas de gestión de proyectos, su verdadera fortaleza radica en su capacidad de crecimiento y personalización. La plataforma está pensada para ser una base sólida que permita a los usuarios ir añadiendo nuevas funcionalidades de acuerdo con sus necesidades, sin la necesidad de migrar a otras aplicaciones o herramientas.

Lo que distingue a **Project Track** de otras soluciones es su enfoque en la centralización y la integración. A medida que crece la plataforma, los usuarios no necesitarán cambiar de programas o aplicaciones para gestionar diferentes aspectos de su proyecto. Por ejemplo, así como en Visual Studio Code se pueden integrar extensiones para gestionar bases de datos o realizar solicitudes HTTP, **Project Track** busca ofrecer una experiencia similar. Los usuarios podrán gestionar tareas, manejar bases de datos, realizar consultas, enviar solicitudes HTTP, e incluso comunicarse con su equipo, todo desde una sola plataforma.

El sistema está diseñado para adaptarse a nuevas funcionalidades como un gestor de mensajes, notificaciones, calendarios y posibles herramientas de colaboración como videollamadas o compartición de pantalla, sin la necesidad de recurrir a aplicaciones externas. La meta es que todas las necesidades del equipo de trabajo en torno a un proyecto sean cubiertas en un solo lugar, evitando la fragmentación de herramientas y mejorando la productividad.

En resumen, **Project Track** no solo permite gestionar tareas y proyectos de manera eficiente, sino que también está diseñado para crecer con el usuario, integrando herramientas necesarias para cada proyecto y asegurando que todas las necesidades queden cubiertas en un solo espacio.

# Propósito

El principal propósito de Project Track es ser una plataforma flexible y personalizable que facilita el seguimiento y la gestión integral de proyectos. Inicialmente orientado a desarrolladores de software, Project Track incluye características pensadas específicamente para este público, como la integración de enlaces a repositorios y herramientas de desarrollo, pero su flexibilidad permitirá que se adapte a otros tipos de proyectos y equipos.

La idea principal es que los usuarios puedan crear un proyecto y personalizarlo de acuerdo con sus necesidades, agregando secciones como la gestión de tareas, los enlaces a repositorios, la asignación de responsables, entre otras. Al esto permitir la personalización, Project Track se adapta a proyectos de cualquier tipo y tamaño, desde un pequeño equipo de desarrolladores hasta empresas más grandes que requieren una mayor complejidad en la gestión de sus procesos.

A medida que Project Track evolucione, será capaz de manejar todos los aspectos relacionados con un proyecto, desde la colaboración en tiempo real entre los miembros del equipo hasta la gestión de calendarios, reuniones y notificaciones, todo dentro de la misma plataforma. El objetivo final es proporcionar una herramienta unificada que elimine la necesidad de recurrir a múltiples aplicaciones para gestionar un solo proyecto.

## Público Meta:

Project Track está dirigido inicialmente a desarrolladores de software que buscan una plataforma flexible y escalable para gestionar sus proyectos de manera más eficiente. Sin embargo, su potencial de crecimiento lo hace ideal para otros sectores y equipos de trabajo que requieren una herramienta robusta y personalizable para gestionar cualquier tipo de proyecto.

Esta plataforma permitirá a los equipos de desarrollo tener un control más centralizado sobre sus proyectos, evitando la necesidad de recurrir a múltiples herramientas para realizar tareas específicas. Con su capacidad de personalización, Project Track también podrá adaptarse fácilmente a las necesidades de otros tipos de usuarios y sectores, como gestión empresarial, marketing, diseño, etc., convirtiéndose en una solución integral para cualquier proyecto que requiera seguimiento y colaboración efectiva.

# Glosario de términos

* **PostgreSQL**: Base de datos relacionales utilizada para almacenar datos de proyectos, tareas, usuarios y roles.
* **Sequelize**: ORM (Object-Relational Mapping) utilizado para interactuar con PostgreSQL, permitiendo manejar las consultas y modelos de base de datos de manera más sencilla.
* **2FA (Autenticación de Dos Factores)**: Método de autenticación que añade una segunda capa de seguridad, requiriendo una verificación adicional al iniciar sesión.
* **JWT (JSON Web Token)**: Token que se utiliza para la autenticación y autorización en la plataforma, asegurando que las interacciones entre el cliente y el servidor sean seguras.
* **React**: Biblioteca de JavaScript utilizada en el frontend para construir interfaces de usuario.
* **Sass**: Preprocesador de CSS que permite escribir estilos de manera más organizada y eficiente.

# Descripción del proyecto

## Descripción técnica del backend:

El backend está construido con **Node.js** y el framework **Express.js**, utilizando **PostgreSQL** como base de datos principal y **Sequelize** como ORM para manejar modelos y consultas. El sistema de autenticación se maneja con **JWT** y es compatible con **2FA** para garantizar la seguridad.

Además, se cuenta con varios scripts principales para la gestión del proyecto:

1. **Creación de componentes (controladores, rutas, modelos)**:

* Los componentes pueden crearse individualmente o en combinaciones utilizando diferentes flags: -r (rutas), -m (modelos), -c (controladores).
* Ejemplos de combinaciones:
  + npm run create -- <name> -rmc / Crea las rutas, el modelo y el controlador.
  + npm run create -- <name> -c / Solo crea el controlador.
* También se incluye la bandera -respara reescribir archivos existentes.

1. **Importación de rutas**:

* Se ejecuta con node importRoutes o npm run import.
* Este script importa automáticamente todas las rutas en un archivo index.js dentro de la carpeta de rutas, y luego este archivo se importa en el index.js principal de la API.

## Interfaz

El frontend está desarrollado en **React**, utilizando una estructura modular de componentes con estilos organizados a través de **Sass**. La estructura de estilos incluye:

* **Base**: Estilos globales (reset.scss, typography.scss, global.scss).
* **Temas**: Definición de temas (dark.scss, light.scss).
* **Utilidades**: Utilidades y mixins (mixins.scss, functions.scss).

Los componentes de React se organizan en carpetas dentro de src/components, con archivos .jsy .scss asociados para cada componente.

Además, cuenta con un script para la creación automatizada de componentes:

* **Crear componente**: npm run create-c -- <nombre\_del\_componente> / Crea la carpeta, el archivo .jsy el archivo .scss del componente.

# Dependencias Principales

* Node.js: versión 22.7.0
* Expresar: v4.19.2
* PostgreSQL: versión 8.12.0
* Secuela: v6.37.3
* Reaccionar: v18.2.0
* Sass: versión 1.77.8

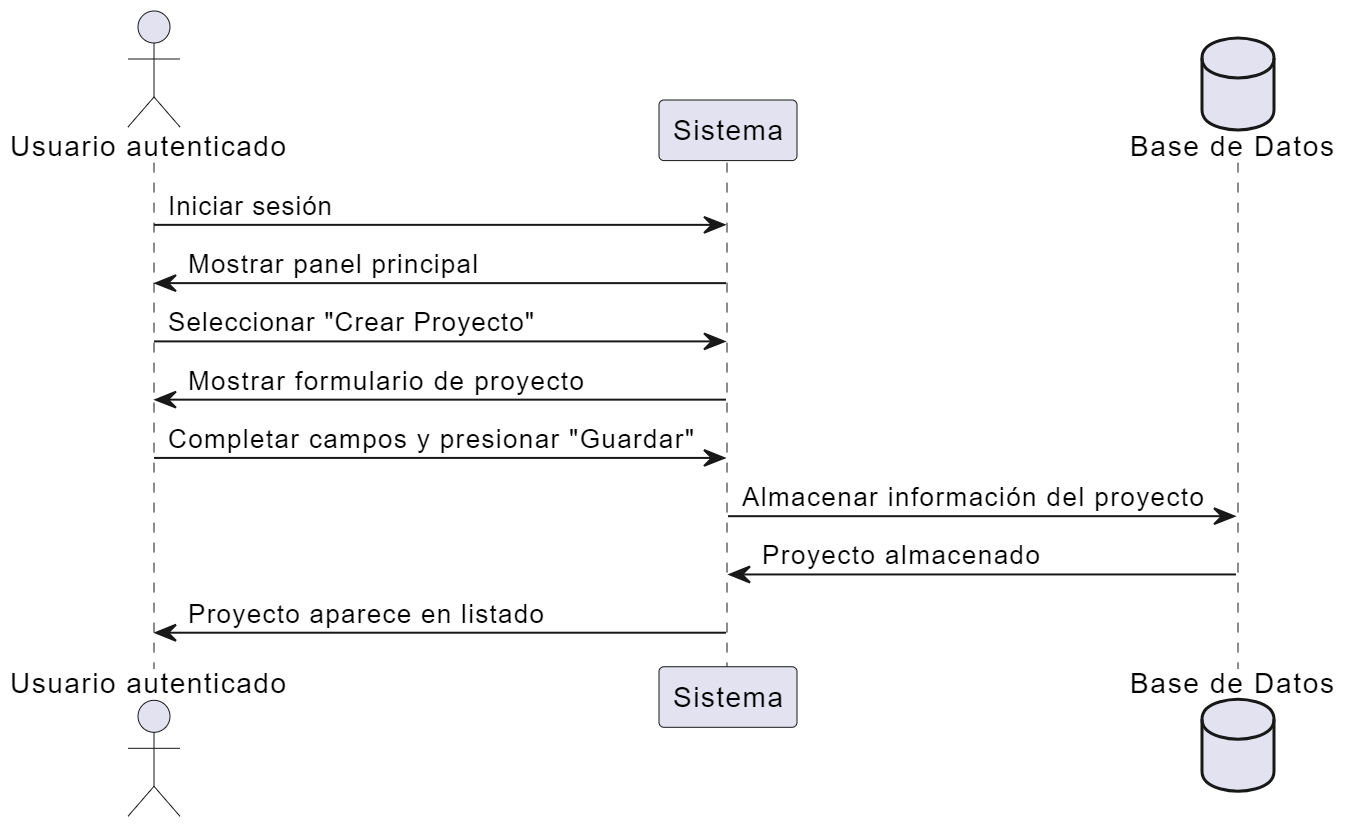
**Otras dependencias:**

* bcryptjs: versión 2.4.3
* cursos: v2.8.5
* dotenv: versión 16.4.5
* jsonwebtoken: versión 9.0.2
* nodo-cron: v3.0.2
* tienda pg: v2.3.4
* Código QR: v1.5.4
* nodemon: v3.1.4

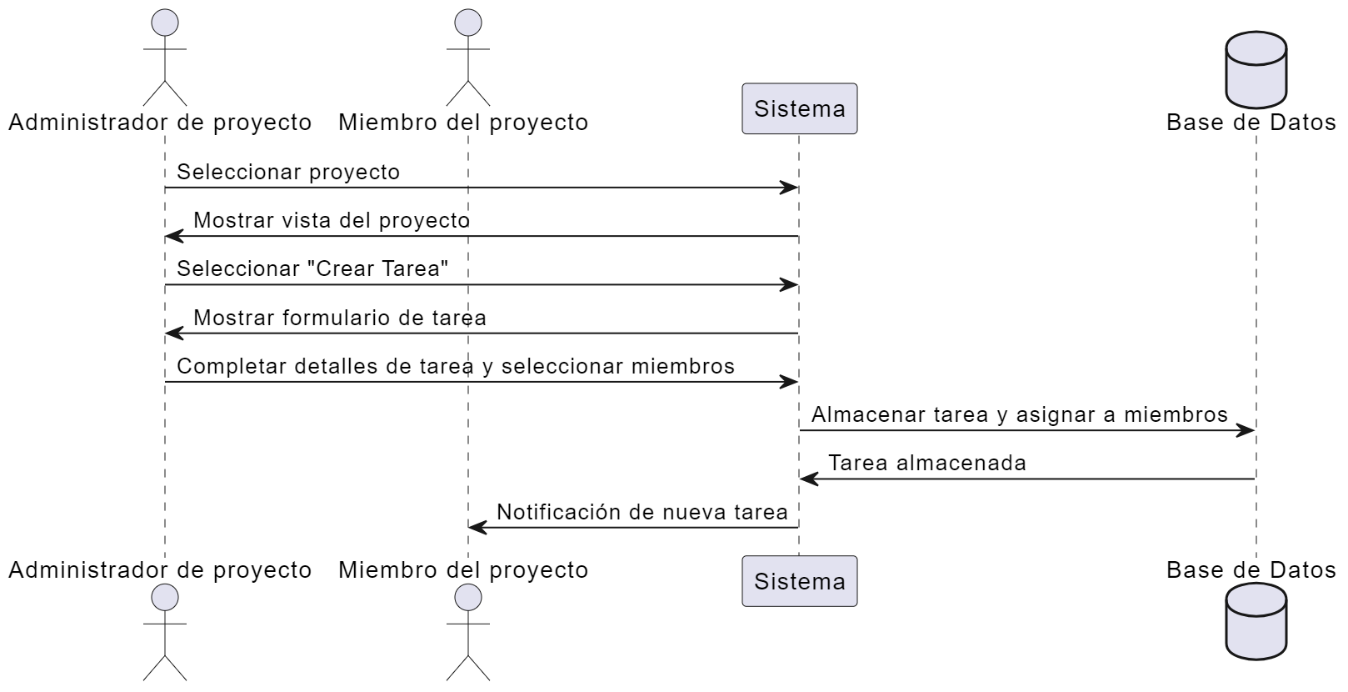
# Requisitos del sistema

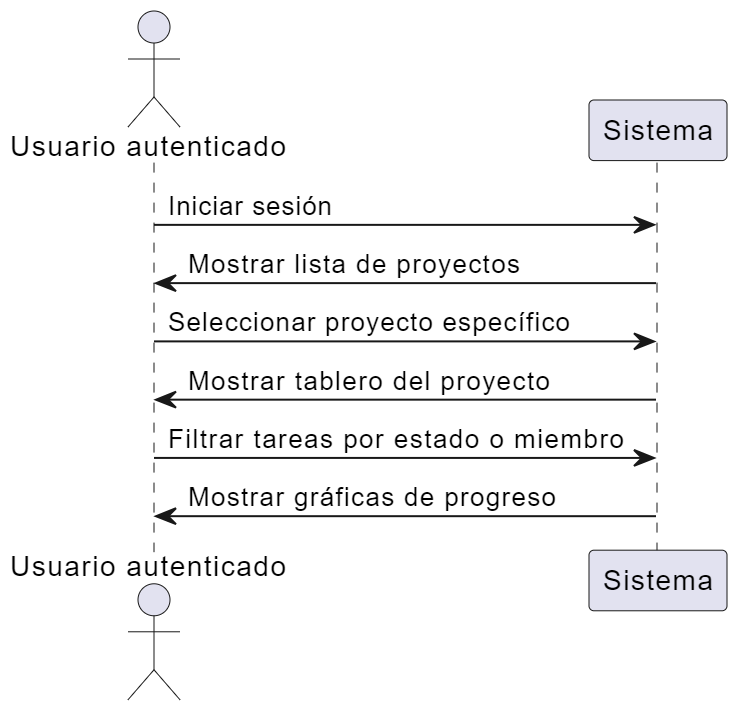
* **Node.js**: v22.7.0 o superior.
* **PostgreSQL**: Instalado y configurado para almacenar datos de usuarios y proyectos.
* **Sass**: Requerido para compilar los estilos del frontend.

# Casos de uso

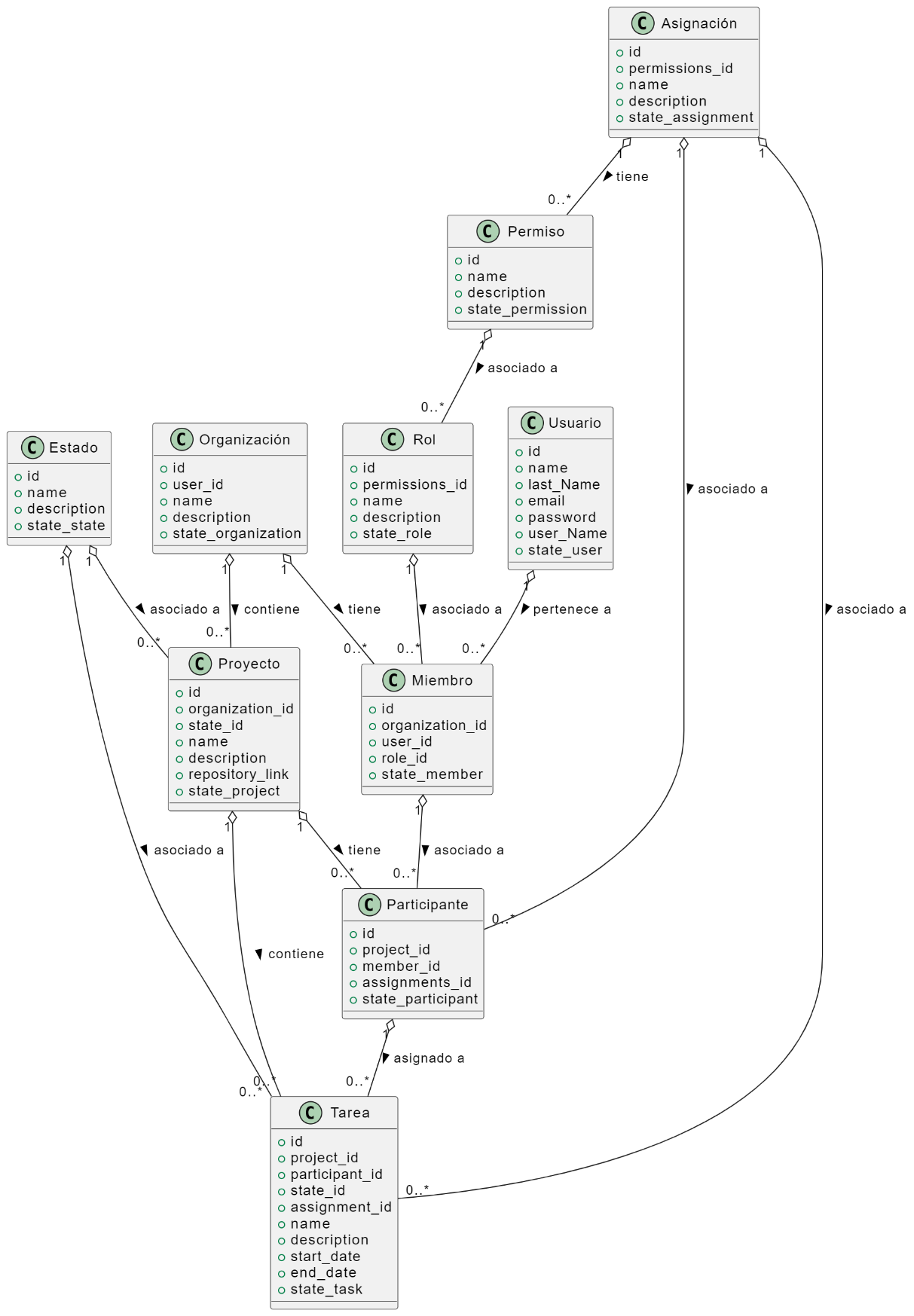
Creación de un Proyecto

## Asignación de Tareas



Visualización del Progreso del Proyecto

# Modelado del sistema

Diagrama de clases 

# Prototipos y bocetos

# Análisis técnico

## 1. Tecnologías utilizadas

Interfaz:

* + **React**: La interfaz de usuario del proyecto está construida con React, un framework JavaScript eficiente para construir interfaces interactivas. Se utiliza para gestionar la vista de la aplicación y la interacción del usuario.
  + **Sass**: Sass se utiliza para el manejo de los estilos y hojas de estilo dinámicos, facilitando la escritura de CSS modular y reutilizable.

Parte trasera:

* + **Node.js con Express**: El servidor back-end está construido sobre Node.js, utilizando Express para gestionar las rutas y las interacciones con las API. Express proporciona un marco minimalista y flexible para la creación de aplicaciones web y API.
  + **Sequelize**: Sequelize es el ORM (Object-Relational Mapping) utilizado para interactuar con la base de datos PostgreSQL. Permite manejar fácilmente las entidades de la base de datos mediante modelos y sus relaciones.

Base de datos:

* + **PostgreSQL**: Como base de datos relacional, PostgreSQL se utiliza para almacenar de estructura manerada los datos relacionados con usuarios, publicaciones, programación de tareas, tokens de autenticación, etc.

Autenticación:

* + **OAuth 2.0**: El sistema de autenticación está basado en OAuth 2.0 para acceder a APIs de redes sociales como Twitter, Mastodon y Reddit. OAuth asegura que los usuarios puedan autorizar a la aplicación a actuar en su nombre sin necesidad de compartir sus credenciales.
  + **JWT (JSON Web Tokens)**: Los JWT se utilizan para gestionar la autenticación de los usuarios en el sistema, permitiendo un método seguro y escalable para manejar sesiones.

Tareas y posteo en redes sociales:

* + **Fetch API**: Para realizar solicitudes HTTP a las API de redes sociales (por ejemplo, Twitter, Reddit, Mastodon) desde el front-end y back-end.
  + **Manejo de Tokens**: El sistema gestiona los tokens de acceso de manera dinámica, incluyendo la renovación de tokens vencidos y el almacenamiento seguro.

## 2. Requisitos del sistema

Equipo:

* + **Servidor**: El sistema servidores requiere para manejar las solicitudes de los usuarios y las interacciones con las redes sociales. Se recomienda utilizar un servidor con al menos 4 GB de RAM y 2 vCPU para entornos de desarrollo.
  + **Almacenamiento**: Dependerá del volumen de usuarios, pero para un prototipo inicial se recomienda un mínimo de 20 GB de almacenamiento.

Software:

* + **Sistemas operativos**: El servidor puede correr en distribuciones basadas en Linux, como Ubuntu, o en contenedores Docker para mayor escalabilidad.
  + **Dependencias**: Node.js, PostgreSQL, Redis (opcional para manejo de colas), y otras dependencias relacionadas a React y Express (como nodemailer, axios, bcrypt).

## 3. Diagrama de infraestructura

Aquí incluirías un diagrama que ilustra la arquitectura de tu sistema:

* **Front-end**: Cliente React -> Servidor Node.js (Express)
* **Back-end**: Servidor Node.js conectado a la base de datos PostgreSQL y servicios externos (APIs de Twitter, Mastodon, Reddit)
* **Almacenamiento**: PostgreSQL almacena los datos de usuarios, tareas, publicaciones y tokens de acceso.
* **Escalabilidad**: Se puede considerar utilizar Docker para la orquestación de contenedores y la implementación de microservicios.

## 4. Consideraciones de escalabilidad y seguridad

Escalabilidad:

* + El sistema puede escalar fácilmente al implementar balanceadores de carga para manejar múltiples instancias del servidor back-end e distribuir las solicitudes entrantes.
  + Uso de servicios de caché como Redis o Memcached para reducir la carga en la base de datos.
  + Almacenamiento en la nube y bases de datos distribuidas para grandes volúmenes de datos.

Seguridad:

* + **Encriptación**: Toda la información sensible, como contraseñas y tokens de acceso, se encripta usando algoritmos como bcrypt y los datos en tránsito se protegen mediante HTTPS.
  + **Autenticación segura**: Uso de OAuth 2.0 y JWT con renovación de tokens para gestionar accesos y permisos de los usuarios.
  + **Protección contra ataques**: Implementación de políticas CORS, uso de casco.js para mejorar la seguridad de las cabeceras HTTP, y protección contra ataques de fuerza bruta mediante servicios como limitación de velocidad.

# Planificación del proyecto

## 1. Cronograma

|  |  |  |  |
| --- | --- | --- | --- |
| **Fase** | **Duración** | **Responsable** | **Descripción** |
| Análisis de requisitos | 1 semana | Equipo de análisis | Definir las necesidades del sistema y casos de uso. |
| Diseño del sistema | 2 semanas | Equipo de desarrollo | Crear diagramas UML, ER y arquitectura general. |
| |  | | --- | |  |   Creación de la base de datos | 1 semana | |  | | --- | | Desarrolladores backend |  |  | | --- | |  | | Diseñar y configurar la base de datos PostgreSQL, modelos y relaciones (con Sequelize). |
| Creación de la API REST | 4 semanas | Desarrolladores backend | Implementación de la API REST para la lógica de negocios. |
| Desarrollo  Front-end | 5 semanas | Desarrolladores frontend | Creación de la interfaz de usuario con React. |
| Pruebas | 2 semanas | Equipo de control de calidad | Realizar pruebas funcionales y de integración. |
| Despliegue | 1 semana | DevOps | Despliegue del sistema en el entorno de producción. |

## 2. Hitos y entregables

* **Sistema de implementación de autenticación** (Semana 4): Implementar la autenticación con OAuth 2.0 y JWT.
* **Funcionalidad de creación de proyectos completados** (Semana 6): Los usuarios pueden crear proyectos, tareas y programar publicaciones.
* **Integración con redes sociales** (Semana 7): Los usuarios pueden vincular cuentas de redes sociales y publicar directamente desde la aplicación.

## 3. Recursos necesarios

* **Recursos humanos**: Un equipo de desarrollo full-stack (front-end y back-end), un diseñador de UI/UX, y un especialista en DevOps.
* **Recursos económicos**: Presupuesto para servidores en la nube, licencias de software (si se usan herramientas no gratuitas), y herramientas de pruebas.

## 4. Riesgos y plan de mitigación

* **Riesgo técnico**: Fallos en la integración con redes sociales debido a cambios en las API externas.
  + **Mitigación**: Implementar mecanismos de monitoreo para detectar cambios en las APIs y tener plan B (documentación actualizada).
* **Retrasos en el desarrollo**: Cambios en los requisitos o dificultades técnicas no previstas.
  + **Mitigación**: Planificar márgenes de tiempo en cada fase del proyecto y tener reuniones semanales de revisión.

# Anexos