UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA
FACULTAD DE SISTEMAS Y TELECOMUNICACIONES

TITULO DEL TRABAJO DE TITULACIÓN

ANÁLISIS COMPARATIVO DE FRAMEWORKS FRONTEND: EFICIENCIA Y FACILIDAD DE USO ENTRE REACT Y SVELTE EN EL DESARROLLO DE APLICACIONES WEB

AUTOR

Rosales Flores Denny Alexander

EXAMEN COMPLEXIVO

Previo a la obtención del grado académico en
INGENIERO EN TECNOLOGÍAS DE LA INFORMACIÓN

TUTOR

ING. Carlos Efrain Sanchez León, MGT.

Santa Elena, Ecuador

Año 2025

UPSE

UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA
FACULTAD DE SISTEMAS Y TELECOMUNICACIONES

TRIBUNAL DE SUSTENTACIÓN

Ing. José Sánchez Aquino, Mgt.
DIRECTOR DE LA CARRERA

Ing. Carlos Sánchez León, Mgt
TUTOR

Ing. Mónica Jaramillo Infante, Mgt
DOCENTE ESPECIALISTA

Ing. Marjorie Coronel Suárez, Mgt.
DOCENTE GUÍA UIC

UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA
FACULTAD DE SISTEMAS Y TELECOMUNICACIONES

CERTIFICACIÓN

Certifico que luego de haber dirigido científica y técnicamente el desarrollo y estructura final del trabajo, este cumple y se ajusta a los estándares académicos, razón por el cual apruebo en todas sus partes el presente trabajo de titulación que fue realizado en su totalidad por **Denny Alexander Rosales Flores**, como requerimiento para la obtención del título de Ingeniero en Tecnologías de la Información.

La Libertad, a los 20 días del mes de junio del año 2025

TUTOR

Firmado electrónicamente por:
CARLOS EFRAIN SANCHEZ LEON
Validar únicamente con FirmaEC

Ing. Carlos Efraín Sánchez León

UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA
FACULTAD DE SISTEMAS Y TELECOMUNICACIONES

DECLARACIÓN DE RESPONSABILIDAD

Yo, **Denny Alexander Rosales Flores**

DECLARO QUE:

El trabajo de Titulación, Análisis comparativo de frameworks frontend: eficiencia y facilidad de uso entre react y svelte en el desarrollo de aplicaciones web previo a la obtención del título en Ingeniero en Tecnologías de la Información, ha sido desarrollado respetando derechos intelectuales de terceros conforme las citas que constan en el documento, cuyas fuentes se incorporan en las referencias o bibliografías. Consecuentemente este trabajo es de mi total autoría.

En virtud de esta declaración, me responsabilizo del contenido, veracidad y alcance del Trabajo de Titulación referido.

La Libertad, a los 20 días del mes de junio del año 2025

EL AUTOR

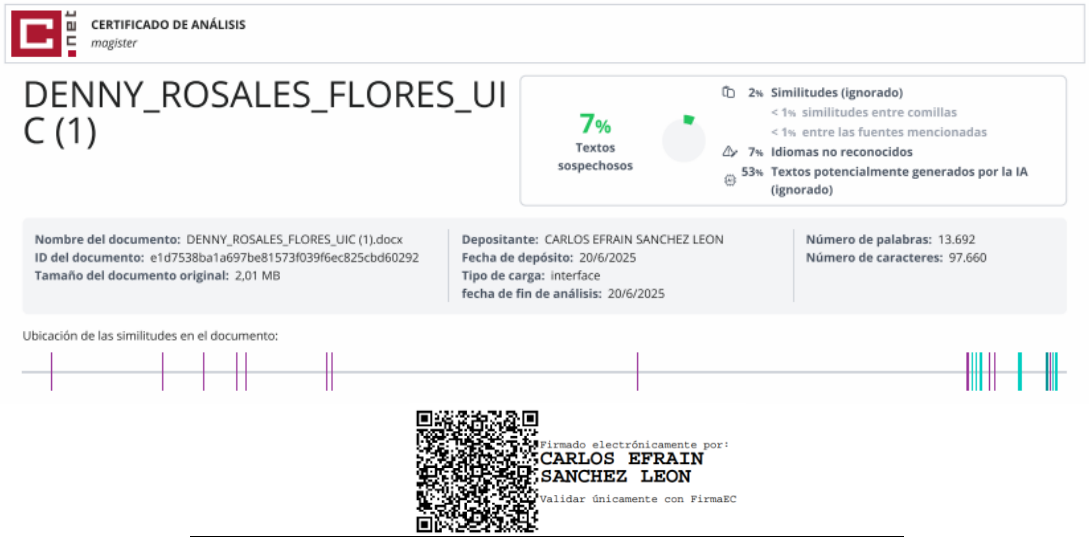
Denny Alexander Rosales Flores

UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA

FACULTAD DE SISTEMAS Y TELECOMUNICACIONES

CERTIFICACIÓN DE ANTIPLAGIO

Certifico que después de revisar el documento final del trabajo de titulación denominado **Análisis comparativo de frameworks frontend: eficiencia y facilidad de uso entre react y svelte en el desarrollo de aplicaciones web**, presentado por el estudiante, DENNY ALEXANDER ROSALES FLORES fue enviado al Sistema Antiplagio, presentando un porcentaje de similitud correspondiente al 7%, por lo que se aprueba el trabajo para que continúe con el proceso de titulación.



Ing. Carlos Efrain Sanchez León

TUTOR

UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA
FACULTAD DE SISTEMAS Y TELECOMUNICACIONES
AUTORIZACIÓN

Yo, **Denny Alexander Rosales Flores**

Autorizo a la Universidad Estatal Península de Santa Elena, para que haga de este trabajo de titulación o parte de él, un documento disponible para su lectura consulta y procesos de investigación, según las normas de la Institución.

Cedo los derechos en línea patrimoniales del presente trabajo de titulación con fines de difusión pública, además apruebo la reproducción dentro de las regulaciones de la Universidad, siempre y cuando esta reproducción no suponga una ganancia económica y se realice respetando mis derechos de autor

Santa Elena, a los 20 días del mes de junio del año 2025

EL AUTOR

Denny Alexander Rosales Flores

AGRADECIMIENTO

Agradezco en primer lugar a Dios por darme la fortaleza y sabiduría por culminar toda esta etapa importante en mi vida.

A mi familia, por su amor incondicional, su apoyo constante y por ser mi inspiración diaria.

A mis amigos cercanos como los que no se encuentran en el país y compañeros de carrera, por su compañía, sus consejos y todos los momentos compartidos a lo largo de este proceso académico.

A mi tutor Ing. Carlos Sanchez, gracias por su orientación y paciencia durante el proceso de este proyecto.

Denny Alexander, Rosales Flores

DEDICATORIA

Dedico este trabajo, con profundo agradecimiento y afecto, a mi madre, mi padre , quienes han sido mi pilar en cada etapa de este camino. Su apoyo constante, sus palabras de ánimo y su presencia incondicional me dieron la fuerza necesaria para no rendirme cuando las circunstancias fueron difíciles.

A mi familia, por creer en mí incluso cuando yo dudaba, y por brindarme el amor y la motivación que me impulsaron a continuar. Este logro no es solo mío, sino también de ustedes, porque su compañía fue clave para alcanzar esta meta; también me lo dedico a mí mismo, por no rendirme, por el compromiso, el esfuerzo y la determinación con la que enfrenté cada reto.

Denny Alexander, Rosales Flores

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE GRÁFICOS

RESUMEN

El presente trabajo realiza un análisis comparativo entre los frameworks frontend React y Svelte, con el objetivo de evaluar su eficiencia y facilidad de uso en el desarrollo de aplicaciones web. Para ello, se diseñó y desarrolló una aplicación denominada gestor de tareas dinámico, implementada en ambos frameworks. Se aplicaron herramientas como Google Lighthouse y Chrome DevTools para medir métricas clave como velocidad de carga, uso de memoria, eficiencia en la actualización del estado y facilidad de implementación. Los resultados mostraron que Svelte ofrece mejor rendimiento en aplicaciones pequeñas y medianas, con menor consumo de recursos y mayor rapidez de respuesta, mientras que React destaca por su escalabilidad y robustez en proyectos complejos; se concluye que la elección del framework debe responder a las necesidades específicas del proyecto, y que Svelte es una opción eficiente y accesible para desarrollos livianos.

Palabras claves: React, Svelte, rendimiento web

ABSTRACT

This paper performs a comparative analysis between the front-end frameworks React and Svelte, with the aim of evaluating their efficiency and ease of use in web application development. To this end, a dynamic task manager application was designed and developed, implemented in both frameworks. Tools such as Google Lighthouse and Chrome DevTools were used to measure key metrics such as loading speed, memory usage, state update efficiency, and ease of deployment. The results showed that Svelte offers better performance in small and medium-sized applications, with lower resource consumption and faster response times, while React stands out for its scalability and robustness in complex projects. It is concluded that the choice of framework should respond to the specific needs of the project, and that Svelte is an efficient and accessible option for lightweight development.

Keywords: React, Svelte, web performance

INTRODUCCIÓN

Este proyecto, titulado “Análisis comparativo de frameworks frontend: eficiencia y facilidad de uso entre React y Svelte en el desarrollo de aplicaciones web”, tiene como objetivo determinar qué framework ofrece un mejor rendimiento y experiencia para el desarrollador en contextos reales. React y Svelte son dos tecnologías ampliamente utilizadas para construir interfaces web interactivas, pero con diferencias fundamentales en su arquitectura y ejecución. Esta investigación busca comparar de forma técnica y objetiva aspectos como velocidad de carga, uso de memoria, eficiencia en el renderizado y facilidad de implementación, para orientar a desarrolladores y equipos técnicos en una elección informada según las características de sus proyectos.

Para el desarrollo de este estudio, se construyeron dos aplicaciones web funcionales e idénticas un gestor de tareas dinámico, implementadas una con React y otra con Svelte. Ambas fueron evaluadas, utilizando herramientas de benchmarking como Google Lighthouse y Chrome DevTools, que permitieron registrar métricas clave como el Largest Contentful Paint (LCP), consumo de CPU, tiempo de renderizado de componentes y eficiencia en la actualización del estado. Se aplicó una metodología incremental, asegurando una evolución progresiva del proyecto a través de fases bien delimitadas: recolección de datos, desarrollo de prototipos, evaluación de rendimiento y análisis comparativo.

El Capítulo 1 introduce el problema y destaca la importancia de tomar decisiones tecnológicas fundamentadas, especialmente en contextos locales donde la adopción de herramientas como React suele estar basada más en popularidad que en análisis técnico. Se presentan los antecedentes, los objetivos, la justificación del estudio y el alcance del proyecto, centrado en aplicaciones web pequeñas y medianas. Además, se explica por qué Svelte representa una alternativa prometedora frente a React, a pesar de su menor adopción.

En el Capítulo 2 se desarrolla el marco teórico y conceptual, abordando temas como los principios de renderizado de componentes, la reactividad, el uso del DOM virtual y las diferencias arquitectónicas entre ambos frameworks. También se expone la metodología de investigación, enfocada en un análisis cuantitativo no

experimental, utilizando métricas concretas para establecer comparaciones justas. Se detalla el uso de tecnologías como TypeScript, Vite, NestJS, y herramientas de despliegue en la nube como Vercel y Render, que garantizaron igualdad de condiciones para ambas implementaciones.

El Capítulo 3 expone la propuesta técnica, que incluye la arquitectura del sistema, los requerimientos funcionales y no funcionales, los diagramas de casos de uso y el modelado de datos. Se describen en detalle los componentes desarrollados (listas dinámicas, formularios, dashboards), explicando cómo fueron construidos en ambos frameworks siguiendo las prácticas específicas de cada uno. Además, se documentan los procesos de desarrollo, pruebas de rendimiento y análisis comparativo entre React y Svelte con resultados claros y objetivos.

Este proyecto aporta una evaluación sistemática y práctica sobre el uso de frameworks frontend en el desarrollo de aplicaciones web, brindando información valiosa para desarrolladores, empresas y comunidades tecnológicas locales. Los resultados permiten concluir que Svelte, en aplicaciones de escala pequeña o media, logra un mejor rendimiento en términos de carga y consumo de recursos, mientras que React ofrece mayor flexibilidad y escalabilidad en proyectos más complejos. De este modo, esta investigación puede servir como guía para seleccionar tecnologías según el tipo y necesidades del proyecto, promoviendo el uso de herramientas más adecuadas y eficientes para el desarrollo web moderno.

CAPTIULO I

1.1 Antecedentes

En la actualidad, el desarrollo de aplicaciones web eficientes es un desafío constante para muchas empresas y desarrolladores [1]. La elección de frameworks frontend es una decisión crucial que puede influir en la calidad y el rendimiento de las aplicaciones [1]. Frameworks como React y Svelte han ganado relevancia por su capacidad para crear aplicaciones web interactivas y dinámicas [2]. Sin embargo, cada uno presenta ventajas y desventajas que impactan tanto el rendimiento como la curva de aprendizaje del desarrollador [3].

Un framework es un conjunto de herramientas y bibliotecas predefinidas que proporciona una estructura base para agilizar el desarrollo de software, estandarizando buenas prácticas y reduciendo código repetitivo [1]. En el contexto frontend, frameworks como React y Svelte optimizan la creación de interfaces dinámicas mediante componentes reutilizables y gestión eficiente del estado [2].

El uso de tecnologías web ha crecido exponencialmente en los últimos años, impulsado por la necesidad de aplicar criterios de experiencia y interfaz de usuario en el desarrollo de aplicaciones web modernas [4]. A pesar de la creciente demanda, los desarrolladores locales se enfrentan a la falta de conocimientos técnicos sobre frameworks emergentes como Svelte, lo que lleva a optar por soluciones más conocidas como React, que aunque robustas, no siempre son la mejor opción para aplicaciones pequeñas o medianas [5]. Se ha generado un desafío para los profesionales que buscan optimizar tanto el proceso de desarrollo como el rendimiento final de las aplicaciones [6].

Muchos desarrolladores optan por frameworks populares como React debido a su extenso soporte y comunidad, pero estudios comparativos demuestran que la selección del framework adecuado debe basarse en las características específicas del proyecto, considerando factores como escalabilidad, rendimiento y facilidad de implementación [7]. Esto ha creado una brecha tecnológica, donde los desarrolladores no siempre evalúan sistemáticamente las herramientas disponibles

según las necesidades específicas del proyecto, afectando tanto el tiempo de desarrollo como el desempeño de la aplicación.

Diversos estudios comparan frameworks frontend; Almeida y Paula, en "Análise comparativa das características de performance dos JavaScript frameworks React e Vue" [3], destacan que React supera en velocidad de renderizado, mientras Vue optimiza memoria, demostrando la importancia de seleccionar el framework según métricas específicas de rendimiento. García Yánez y Zurita Hidalgo, en "Comparativa de frameworks para el desarrollo web en el lado del cliente basado en métricas de desempeño web vitals" [2], confirman que React es ideal para proyectos grandes, estableciendo criterios claros para la evaluación de frameworks según el contexto del proyecto. Estos hallazgos guían la elección tecnológica según necesidades específicas [6].

Es necesario abordar la falta de conocimiento sobre frameworks emergentes en el contexto de Santa Elena. Al ofrecer comparaciones detalladas entre opciones populares como React y nuevas alternativas como Svelte, optimizando así el proceso de desarrollo y el rendimiento de las aplicaciones web locales. Esta investigación busca contribuir a este objetivo, proporcionando una evaluación integral de ambos frameworks en términos de rendimiento, escalabilidad y facilidad de uso.

1.2 Descripción del proyecto

Este estudio está dirigido principalmente a desarrolladores web y equipos técnicos con conocimientos básicos en frontend, que buscan tomar decisiones informadas al elegir entre React y Svelte para sus proyectos. También puede ser útil para líderes de proyectos o pequeñas empresas que necesitan optimizar recursos sin sacrificar rendimiento.

El presente proyecto tiene como objetivo principal comparar el rendimiento y la eficiencia de los frameworks React y Svelte, dos de las herramientas más utilizadas en el desarrollo front-end. Para ello, se realizará un análisis en un escenario controlado, utilizando como caso práctico el desarrollo de una aplicación web denominada Gestor de Tareas Dinámico. Esta aplicación permitirá implementar y

evaluar funcionalidades clave como la creación, edición, eliminación y filtrado de tareas, así como la visualización de datos, para medir el manejo de estados, el tiempo de renderizado y el uso eficiente de recursos en ambos frameworks.

El uso de tecnologías web está en constante crecimiento, particularmente en sectores como el comercio y el turismo, donde la eficiencia en el tiempo de carga y la interactividad son esenciales para mejorar la experiencia del usuario. Sin embargo, la falta de estudios comparativos sobre la efectividad de estos frameworks en este contexto local ha llevado a una adopción no informada, donde React es la opción más común por su popularidad, a pesar de que Svelte podría ofrecer ventajas específicas en términos de simplicidad y rendimiento.

El proyecto se desarrollará en cuatro fases que permitirán un análisis exhaustivo de los componentes y recursos que emplea cada framework en un contexto de aplicaciones web pequeñas y medianas [3].

Fase de recolección de datos y definición de métricas

En esta fase inicial, se realizará una revisión exhaustiva de la literatura existente sobre los frameworks React y Svelte, con especial énfasis en estudios recientes que analicen el rendimiento de componentes y el uso de recursos como memoria y CPU. Se definirán las métricas que se usarán para comparar ambos frameworks, tales como:

- Tiempo de renderizado de componentes.
- Uso de memoria.
- Velocidad de carga.
- Eficiencia en la actualización de estados.

Fase de desarrollo de prototipos y componentes

Se desarrollarán dos aplicaciones web con características similares, una implementada en React y la otra en Svelte. Se construirán componentes comunes en ambos frameworks (como listas dinámicas, formularios y componentes de visualización), lo que permitirá medir y comparar el rendimiento en situaciones reales. Esto incluye:

- Componentes de listas dinámicas.
- Componentes de formularios interactivos.
- Actualización de estado y reactividad de datos.

Cada prototipo se desarrollará siguiendo las mejores prácticas de cada framework, para asegurar que las diferencias en el rendimiento no se deban a configuraciones erróneas o malas prácticas de desarrollo.

Fase de evaluación de rendimiento y análisis de componentes

Se utilizarán herramientas de benchmarking como Google Lighthouse y Chrome DevTools para analizar el rendimiento específico de los componentes desarrollados en ambas aplicaciones. Estas pruebas se realizarán en el escenario controlado previamente definido para garantizar la precisión de los datos obtenidos. Los datos permitirán medir:

- Tiempo de carga inicial de cada aplicación.
- Uso de memoria y CPU durante la ejecución.
- Velocidad de actualización en componentes interactivos.
- Capacidad de optimización del código en ambos frameworks.

Fase de análisis comparativo y conclusiones

Finalmente, los datos obtenidos durante las fases de pruebas serán comparados para identificar cuál de los dos frameworks ofrece un mejor rendimiento generalen términos de:

- Velocidad de renderizado de componentes.
- Eficiencia en la actualización del estado.
- Uso de memoria.

1.3 Objetivos del Proyecto

Objetivo General

Comparar la eficiencia y facilidad de uso entre los frameworks frontend React y Svelte en el desarrollo de aplicaciones web, utilizando herramientas de benchmarking como Google Lighthouse y Chrome DevTools, para determinar cuál demuestra un desempeño más óptimo y proporciona una experiencia más efectiva para los desarrolladores.

Objetivos Específicos

1. Analizar el rendimiento de las aplicaciones web desarrolladas en React y Svelte mediante la evaluación de tiempos de carga, uso de memoria y velocidad de renderizado de componentes.
2. Desarrollar una aplicación web denominada gestor de tareas dinámico para implementar y evaluar funcionalidades clave como listas dinámicas, formularios interactivos y visualización de datos, con el propósito de medir y comparar el rendimiento de los frameworks React y Svelte en escenarios controlados.
3. Determinar la capacidad de cada framework para manejar la actualización de estados y el uso de recursos en proyectos de desarrollo web, utilizando prácticas óptimas de desarrollo para identificar cuál facilita un mayor rendimiento y eficiencia en aplicaciones web.

1.4 Justificación del proyecto

El desarrollo de aplicaciones web ha evolucionado significativamente en los últimos años, presentando nuevas oportunidades para optimizar el rendimiento y la experiencia del usuario a través de frameworks frontend modernos. La comparación entre React y Svelte emerge como una respuesta estratégica a la necesidad de seleccionar tecnologías eficientes para el desarrollo web [1]. Esta investigación propone un análisis sistemático que permitirá a los desarrolladores tomar decisiones fundamentadas en datos empíricos sobre el rendimiento, la eficiencia y la facilidad de uso de estos frameworks, contribuyendo así a la mejora continua en el desarrollo de aplicaciones web en el contexto local y nacional [2].

La implementación de frameworks frontend modernos como React y Svelte ofrece beneficios sustanciales para el desarrollo web actual; por un lado, estos frameworks facilitan la creación de interfaces de usuario interactivas , mejorando significativamente la experiencia del usuario final [3]. Además, la comparación sistemática entre estos frameworks proporcionará métricas claras y objetivas sobre el uso de recursos del sistema, tiempos de renderizado y eficiencia en la actualización de estados, permitiendo a los desarrolladores seleccionar la herramienta más adecuada para sus proyectos específicos [5].

La exploración y análisis comparativo de estos frameworks contribuirá significativamente a la profesionalización del desarrollo web en el Ecuador. Al proporcionar datos concretos sobre el rendimiento y la eficiencia de React y Svelte, los desarrolladores podrán tomar decisiones informadas que resulten en aplicaciones web más eficientes y escalables [7]. Esta investigación también facilitará la adopción de mejores prácticas en el desarrollo frontend, promoviendo la innovación tecnológica y la optimización de recursos en el desarrollo de aplicaciones web, lo cual es especialmente relevante para empresas locales y startups que buscan maximizar su presencia digital [3].

Este proyecto se alinea con el Plan Nacional 2024-2025 "Construyendo el Ecuador del Futuro", específicamente con el Eje Económico que busca fortalecer la innovación y la transformación digital del sector productivo. A través del análisis del rendimiento de aplicaciones web desarrolladas en React y Svelte, evaluando tiempos de carga, uso de memoria y velocidad de renderizado de componentes, este trabajo contribuye directamente a la modernización tecnológica del sector empresarial ecuatoriano. Esta investigación apoya el objetivo de mejorar la competitividad digital del país al proporcionar herramientas y conocimientos que permiten desarrollar aplicaciones web más eficientes y escalables, fundamentales para la transformación digital de las empresas locales [2].

1.5 Alcance del proyecto

El proyecto contribuirá a la línea de investigación en desarrollo web y optimización de aplicaciones; está alineado con la necesidad de mejorar el rendimiento y la eficiencia en aplicaciones web, tomando en cuenta el creciente uso de tecnologías en sectores locales como el turismo y comercio.

Este estudio se limitará al desarrollo de una aplicación web denominada gestor de tareas, enfocada exclusivamente en funcionalidades como la creación, edición, eliminación y filtrado de tareas, además de la visualización de datos. Estudios realizados han explorado la importancia de seleccionar correctamente las tecnologías frontend, como React y Svelte, para optimizar recursos y mejorar la experiencia de usuario en contextos locales [2].

CAPÍTULO 2. MARCO TEÓRICO Y METODOLOGÍA DEL PROYECTO

2.1 Marco Conceptual

2.1.1 Framework Frontend

Un framework frontend constituye un sistema integral de bibliotecas y herramientas preconfiguradas que proporcionan una estructura estandarizada para el desarrollo de interfaces de usuario en aplicaciones web modernas. Según Mahmood y Siddiqui ,estos frameworks facilitan significativamente la creación de componentes reutilizables y la gestión eficiente del estado de la aplicación, permitiendo a los desarrolladores construir aplicaciones web escalables y mantenibles [1].

Los frameworks frontend modernos incluyen utilidades para el manejo de rutas, gestión de estados globales y locales, y optimización del rendimiento, además de patrones de diseño probados y mejores prácticas para la arquitectura de aplicaciones frontend. Esta estandarización permite reducir el tiempo de desarrollo y mejorar la calidad del código producido [1].

2.1.2. Frameworks Populares

2.1.2.1 React

React es un framework de código abierto desarrollado y mantenido activamente por Facebook/Meta, que ha revolucionado la forma en que se construyen las interfaces de usuario modernas. García Yánez y Zurita Hidalgo destacan que React utiliza un DOM virtual altamente eficiente para crear interfaces de usuario reactivas y dinámicas, basándose en un sistema de componentes reutilizables que facilitan la modularización del código [2].

Una de las características distintivas de React es su implementación de un flujo de datos unidireccional que mejora la predictibilidad y facilita la depuración de las aplicaciones. Pedro H. Marques Almeida y Werik Gonçalves de Paula ,señalan que React se destaca por su amplio ecosistema de bibliotecas y herramientas complementarias, así como por su gran comunidad de desarrolladores, lo que garantiza soporte continuo y actualizaciones regulares [3].

2.1.2.2 Svelte

Svelte representa un enfoque revolucionario en el desarrollo frontend al adoptar una metodología fundamentalmente diferente a los frameworks tradicionales. Kaluža y Vukelić (2018) explican que Svelte compila el código durante la fase de construcción, lo que lo diferencia significativamente de frameworks como React o Vue que ejecutan gran parte de su lógica en tiempo de ejecución [5].

La principal innovación de Svelte radica en que elimina la necesidad de un DOM virtual al generar código JavaScript altamente optimizado que manipula directamente el DOM. Este enfoque resulta en aplicaciones notablemente más ligeras y rápidas, ya que no requiere cargar una biblioteca de framework en el navegador [5].

2.1.3. Fundamentos del Renderizado

2.1.3.1 Renderizado de Componentes

El renderizado de componentes constituye un proceso fundamental mediante el cual un framework frontend transforma el código de los componentes en elementos visuales interactivos en el navegador del usuario final; Cali describe este proceso como la gestión eficiente de actualizaciones y cambios en la interfaz de usuario, asegurando una experiencia fluida para el usuario [7].

Los frameworks modernos implementan algoritmos sofisticados para determinar qué partes de la interfaz necesitan actualizarse cuando cambian los datos subyacentes [7]. Este proceso de reconciliación es crucial para el rendimiento de la aplicación, ya que las manipulaciones del DOM son computacionalmente costosas [7].

2.1.3.2 DOM Virtual

El DOM Virtual representa una innovación significativa en la optimización de aplicaciones web. García Yánez y Zurita Hidalgo lo definen como una representación ligera y eficiente en memoria del DOM real del navegador, implementada por frameworks como React para optimizar las operaciones de actualización de la interfaz de usuario [2].

El DOM Virtual permite realizar comparaciones rápidas entre estados para determinar los cambios mínimos necesarios en el DOM real; actúa como una capa de abstracción que minimiza las manipulaciones directas del DOM, mejorando significativamente el rendimiento mediante algoritmos sofisticados de reconciliación [8].

2.1.4. Gestión del Estado

2.1.4.1 Estado (State)

El estado constituye el conjunto de datos dinámicos que controlan el comportamiento y la representación visual de los componentes en una aplicación web moderna. Kaluža y Vukelić explican que incluye toda la información que puede cambiar durante la ejecución de la aplicación, desde datos del usuario hasta configuraciones de la interfaz [5].

El estado puede ser local a un componente específico o global para toda la aplicación, requiriendo diferentes estrategias de gestión, su manejo adecuado es crucial para mantener la consistencia y la previsibilidad del comportamiento de la aplicación, especialmente en aplicaciones complejas con múltiples componentes interactuando [5].

2.1.4.2 Reactividad

Característica fundamental de los frameworks modernos que permite detectar automáticamente cambios en los datos y actualizar la interfaz de usuario de manera eficiente [3]. Mantiene una sincronización constante y automática entre el estado de la aplicación y su representación visual, implementa sistemas sofisticados de observación y propagación de cambios para garantizar la consistencia de los datos; optimiza el rendimiento al minimizar las actualizaciones innecesarias y realizar solo los cambios requeridos [3].

2.1.5. Optimización y Rendimiento

2.1.5.1 Tiempo de Compilación

El tiempo de compilación representa una fase crítica en el proceso de desarrollo donde el código fuente se analiza, procesa y convierte en código ejecutable optimizado para el navegador. Mendoza señala que esta fase es especialmente relevante en frameworks como Svelte, que realizan una importante optimización del código durante esta etapa [9].

La compilación permite detectar errores tempranamente y aplicar optimizaciones que serían imposibles en tiempo de ejecución. El proceso incluye la eliminación de código muerto, la minimización y la generación de código específico para cada componente, resultando en aplicaciones más eficientes [9].

2.1.5.2 Benchmarking

El benchmarking constituye un proceso sistemático de medición y comparación del rendimiento entre diferentes sistemas o frameworks frontend, utilizando métricas específicas y estandarizadas; define el benchmarking como la evaluación de aspectos críticos como tiempo de carga inicial, tiempo de interacción, uso de memoria y velocidad de renderizado en diferentes escenarios [7].

Marciniak enfatiza que el benchmarking permite tomar decisiones informadas sobre la selección de tecnologías basadas en datos empíricos y casos de uso específicos, proporcionando una base objetiva para la comparación de frameworks [10].

.2.1.6. Componentización

2.1.6.1 Componentes Web

Elementos modulares y reutilizables que encapsulan funcionalidad, lógica y diseño en unidades independientes y cohesivas; permiten construir interfaces de usuario complejas mediante la composición de componentes más pequeños y manejables [2]. Facilitan la reutilización de código y promueven una arquitectura más limpia y mantenible, implementan patrones de diseño que mejoran la escalabilidad y la mantenibilidad de las aplicaciones [2].

2.1.7 Herramientas de Evaluación de Rendimiento

2.1.7.1 Google Lighthouse

Google Lighthouse constituye una herramienta de código abierto creada por Google que evalúa sitios web mediante auditorías automáticas. Google Developers describe que Lighthouse analiza aspectos como el rendimiento, la accesibilidad, el SEO y las buenas prácticas, simulando la carga de la página bajo condiciones controladas [11].

La herramienta calcula una puntuación del 0 al 100 en cada categoría, usando métricas como tiempo de carga e interacción, procesadas con fórmulas basadas en estudios de experiencia de usuario. Google Developers enfatiza que esta información permite optimizar aplicaciones frontend desarrolladas con frameworks como React o Svelte, proporcionando insights específicos sobre Core Web Vitals [12] .

2.1.7.2 Chrome DevTools

Conjunto de herramientas integradas en el navegador Google Chrome que permiten a los desarrolladores analizar el comportamiento de una aplicación web en tiempo real; se puede inspeccionar el contenido de la página, medir tiempos de carga, identificar cuellos de botella, y observar cómo se comporta el DOM y los estilos CSS ante ciertos eventos. Para medir el rendimiento, este registra lo que ocurre durante la ejecución de una página, capturando eventos clave como la ejecución de scripts, la carga de recursos o las interacciones del usuario; estos datos se presentan en una línea de tiempo detallada que ayuda a identificar qué parte del código está afectando el rendimiento. Aunque no aplica fórmulas como tal, proporciona mediciones precisas que sirven como base para optimizaciones específicas en aplicaciones frontend [13].

2.1.8 Arquitectura de Aplicaciones Web

2.1.8.1 Arquitectura Frontend

La arquitectura frontend moderna se caracteriza por la separación clara entre la lógica de presentación y la lógica de negocio; las aplicaciones web contemporáneas adoptan arquitecturas que permiten mayor flexibilidad, escalabilidad y mantenibilidad [14].

La evolución de las aplicaciones web ha llevado a la adopción de patrones arquitectónicos que favorecen la modularidad y la reutilización de componentes, mejorando significativamente la gestión académica y empresarial [15].

2.1.9 Escenarios Controlados en Evaluación de Frameworks

2.1.9.1 Fundamentos de Escenarios Controlados

Los escenarios controlados constituyen ambientes experimentales donde se mantienen constantes todas las variables excepto aquellas que se desean evaluar, permitiendo una comparación objetiva y reproducible entre diferentes frameworks frontend; en el contexto de evaluación de frameworks, los escenarios controlados garantizan que las diferencias observadas en el rendimiento sean atribuibles únicamente a las características intrínsecas de cada framework y no a factores externos [16].

En investigaciones previas sobre frameworks frontend, se han desarrollado prototipos idénticos en diferentes tecnologías para establecer comparaciones válidas. Este enfoque

metodológico permite evaluar aspectos como tiempo de renderizado, consumo de recursos, y facilidad de implementación bajo condiciones estandarizadas [17].

2.1.9.2 Especificaciones del Entorno Controlado

Las pruebas se ejecutarán en entorno estandarizado con procesador Intel Core i7 o AMD Ryzen 7, 16 GB RAM, almacenamiento SSD y conexión estable de 50 Mbps con latencia menor a 50ms. El sistema operativo será Windows 11 y como navegador tenemos google chorme; los prototipos React y Svelte se desplegaron en Vercel para las pruebas de producción, mientras que el backend NestJS se alojó en Render para garantizar condiciones de red reales. Esta configuración de despliegue en servicios cloud profesionales garantiza mediciones bajo condiciones técnicas idénticas, eliminando variables ambientales que podrían afectar los resultados comparativos entre frameworks [12].

2.1.10 Tecnologías de Desarrollo

2.1.10.1 Vite

Vite constituye la herramienta de construcción moderna seleccionada como base común para ambos prototipos, proporcionando un entorno de desarrollo ultrarrápido mediante módulos ES nativos y Rollup para producción. Su arquitectura garantiza consistencia en el proceso de compilación, eliminando variables relacionadas con diferencias en herramientas de construcción que podrían sesgar la comparación. Vite ofrece Hot Module Replacement extremadamente rápido, optimización automática de recursos y soporte nativo para TypeScript, beneficiando igualmente a ambos frameworks durante el desarrollo y permitiendo mediciones precisas de rendimiento [18].

2.1.10.2 TypeScript

TypeScript se establece como lenguaje de programación uniforme para ambos prototipos, proporcionando tipado estático y características avanzadas que mejoran la calidad del código y experiencia de desarrollo. La utilización consistente de TypeScript elimina sesgos relacionados con diferencias sintácticas, permitiendo que React y Svelte se beneficien igualmente del tipado estático. Esta uniformidad facilita aprovechar características como detección temprana de errores, mejor documentación mediante tipos

explícitos y herramientas de refactoring avanzadas, habilitando una comparación objetiva de facilidad de uso y productividad del desarrollador [19].

2.1.10.3 NestJS y Base de Datos

NestJS constituye el framework backend que proporciona servicios API uniformes para ambos prototipos, construido sobre Node.js con TypeScript y arquitectura modular. La implementación incluye módulos para autenticación JWT, gestión de usuarios, administración de proyectos colaborativos, control de tareas y dashboard con métricas en tiempo real. Prisma actúa como ORM type-safe facilitando la interacción con PostgreSQL, con modelos para usuarios, proyectos, tareas, comentarios y archivos adjuntos. Esta arquitectura backend estandarizada garantiza que ambos frameworks accedan a servicios idénticos, permitiendo evaluar exclusivamente las características intrínsecas de React y Svelte [20].

2.1.11 Servicios Cloud y APIs Externas

2.1.11.1 Vercel

Vercel constituye la plataforma de despliegue seleccionada para los prototipos frontend, proporcionando una infraestructura cloud optimizada para aplicaciones React y Svelte con despliegue automático desde repositorios Git. La plataforma ofrece CDN global, certificados SSL automáticos, optimización de Core Web Vitals y métricas de rendimiento en tiempo real que facilitan la evaluación comparativa bajo condiciones de producción reales. Vercel permite configuraciones específicas por proyecto, incluyendo variables de entorno diferenciadas y dominios personalizados, garantizando que ambos frameworks se evalúen bajo infraestructura idéntica y condiciones de red optimizadas [21].

2.1.11.2 Render

Render actúa como la plataforma de hosting para el backend NestJS, proporcionando servicios web escalables con despliegue automático, SSL integrado y red privada para comunicación segura con bases de datos. La infraestructura incluye balanceado de carga automático, escalado horizontal y monitoreo de recursos que permiten evaluar el comportamiento del API bajo diferentes cargas de trabajo. Render facilita la integración con servicios externos como PostgreSQL y proporciona logs detallados para análisis de

rendimiento, asegurando que tanto React como Svelte accedan a servicios backend consistentes y confiables durante las pruebas comparativas [22].

2.1.11.3 Brevo

Brevo proporciona servicios de email transaccional especializados para la funcionalidad de recuperación de contraseñas, implementando API REST robusta con autenticación segura y plantillas personalizables. La integración incluye configuración SMTP relay, webhooks para tracking de eventos de entrega y sistema de claves API diferenciadas para desarrollo y producción. Brevo garantiza alta deliverabilidad de emails críticos, métricas detalladas de apertura y clics, y cumplimiento con estándares de seguridad GDPR, permitiendo evaluar cómo cada framework maneja operaciones asíncronas de envío de emails y gestión de respuestas de servicios externos [23].

2.1.11.4 Cloudinary

Cloudinary proporciona servicios de gestión y almacenamiento de archivos en la nube, implementando funcionalidades para subida de avatares con optimización automática y gestión de archivos adjuntos con soporte múltiple de formatos. La integración permite evaluar cómo cada framework maneja operaciones asíncronas de subida, indicadores de progreso, preview de archivos y gestión de errores en operaciones de red. Esta funcionalidad común en aplicaciones modernas proporciona un escenario realista para comparar la facilidad de implementación y rendimiento de operaciones complejas en ambos frameworks [24].

2.2 Marco Teórico

2.2.1 Innovación en eficiencia para el desarrollo de aplicaciones web modernas

La eficiencia en el desarrollo web es fundamental en la creación de aplicaciones con tiempos de carga rápidos y bajo consumo de recursos; frameworks como React y Svelte se destacan por sus diferentes enfoques para optimizar el rendimiento en el desarrollo frontend; react emplea un DOM virtual que permite actualizar de forma eficaz los componentes sin recargar toda la página, optimizando así el rendimiento en aplicaciones de alta demanda [25]. En cambio, Svelte compila el código en tiempo de construcción, eliminando la necesidad de un DOM virtual y generando aplicaciones ligeras con menores tiempos de carga [26]. Este enfoque permite a Svelte manejar mejor los recursos del

sistema en aplicaciones de menor escala, reduciendo el tiempo de respuesta y mejorando la experiencia del usuario [25] [26].

Estudios recientes han señalado que, aunque React es ampliamente adoptado en proyectos grandes y complejos por su escalabilidad, Svelte muestra un desempeño más eficiente en términos de uso de memoria y velocidad de carga en aplicaciones pequeñas y medianas, lo que lo convierte en una opción atractiva para proyectos que requieren una experiencia rápida e interactiva en dispositivos de menor capacidad [27] [28].

2.2.2 Maximizando la experiencia de usuario con reactividad avanzada en aplicaciones web

La experiencia de usuario es esencial en el desarrollo web, y la reactividad en frameworks frontend desempeña un rol crítico en la percepción de eficiencia [27]. React implementa un sistema de reactividad mediante su administración de estado y la actualización de componentes a través de su virtual DOM, manteniendo la interfaz reactiva en aplicaciones complejas. Sin embargo, esto puede generar una sobrecarga en tiempo de ejecución debido al procesamiento adicional [29]. En contraste, Svelte utiliza un modelo de programación reactiva directa, permitiendo que los cambios en el estado se reflejen de inmediato en la interfaz sin necesidad de un DOM virtual, mejorando la velocidad de respuesta y optimizando el rendimiento, especialmente en aplicaciones de una sola página (SPA) [29].

Este enfoque también implica que, para proyectos que buscan ofrecer una UX sin demoras en dispositivos de gama media o baja, Svelte proporciona ventajas claras en cuanto a consumo de recursos [25]. React sigue siendo adecuado para aplicaciones que requieren complejidad en la gestión de estados y modularidad avanzada, mientras que Svelte destaca en proyectos más ligeros, optimizando la interacción del usuario al reducir la latencia de respuesta [30].

2.2.3 Productividad y curva de aprendizaje en el uso de frameworks modernos

La productividad en el desarrollo web está estrechamente relacionada con la facilidad de aprendizaje de los frameworks utilizados. React, al ser una herramienta madura y ampliamente adoptada, requiere del dominio de conceptos como JSX, hooks y gestión de estados, lo cual puede representar una curva de aprendizaje considerable para

desarrolladores nuevos [31]. En cambio, Svelte ha sido valorado por su sintaxis simple y su capacidad de escribir código más limpio con menos líneas, lo cual reduce el tiempo de desarrollo y facilita la lectura y mantenimiento del código, especialmente en proyectos pequeños o medianos [32]. Esta diferencia en la complejidad inicial ha llevado a que Svelte sea recomendado en entornos educativos y en proyectos con equipos reducidos donde se busca acelerar el proceso de desarrollo [33].

Estudios comparativos han demostrado que los desarrolladores logran resultados funcionales en menor tiempo utilizando Svelte frente a React, debido a la menor cantidad de configuraciones y a su integración directa con HTML, CSS y JavaScript tradicionales [31]. Sin embargo, React destaca por su ecosistema robusto, herramientas complementarias y modularidad, lo que lo hace ideal en proyectos empresariales que requieren escalabilidad y trabajo en equipo [32]. Así, mientras Svelte favorece la productividad inmediata y el aprendizaje rápido, React ofrece mayor soporte para entornos complejos, haciendo que la elección dependa tanto del tipo de proyecto como del perfil del equipo de desarrollo [33].

2.3 Metodología de la Investigación

2.3.1 Contexto de la investigación

El presente proyecto se desarrollará en un entorno controlado empleando metodologías de benchmarking para frameworks JavaScript/TypeScript como React y Svelte, siguiendo estándares establecidos para la evaluación comparativa de tecnologías de desarrollo web [34] . Este entorno permitirá analizar la carga y uso de recursos de ambos en condiciones constantes, garantizando así una evaluación precisa y comparable [35]. La plataforma se ejecutará en una infraestructura de red estándar, replicando las condiciones que comúnmente enfrentan los usuarios en entornos de aplicación web de pequeña y mediana escala [36].

El entorno de prueba también incluirá una serie de métricas de rendimiento previamente definidas, como el tiempo de carga y el uso de memoria, que serán registradas y analizadas de manera continua durante el desarrollo [37]. Estas métricas permitirán generar recomendaciones concretas para mejorar la eficiencia y rapidez de aplicaciones web en la región de Santa Elena, lo cual es particularmente relevante dada la creciente demanda de servicios digitales [2] .

2.3.2 Enfoque de la investigación

Este proyecto adoptará un enfoque cuantitativo, ya que se busca analizar la variable de pruebas de rendimiento para los frameworks React y Svelte; esta variable permitirá evaluar métricas específicas como tiempos de carga, uso de memoria y velocidad de renderizado de componentes, facilitando la comparación objetiva entre ambos frameworks mediante datos numéricos precisos y medibles [38].

La elección del enfoque cuantitativo se justifica por la necesidad de realizar mediciones exactas del rendimiento de las aplicaciones web desarrolladas en ambos frameworks, utilizando herramientas especializadas como Google Lighthouse y Chrome DevTools; este método permitirá aplicar técnicas estadísticas para validar cuál de los dos frameworks ofrece un mejor desempeño en términos de eficiencia y velocidad, generando resultados confiables y replicables que serán fundamentales para la toma de decisiones en el desarrollo de aplicaciones web [38].

2.3.4 Alcance de la investigación

El alcance de la investigación es analítico, ya que busca no solo describir las características de los frameworks React y Svelte, sino también analizar las diferencias en su rendimiento en función de métricas clave, como tiempo de carga y uso de recursos [38]. Este tipo de enfoque permite profundizar en cómo cada framework maneja estos aspectos, proporcionando datos comparativos que facilitan el entendimiento de sus capacidades y limitaciones [39].

Este nos permitirá evaluar las pruebas de rendimiento de cada framework de manera práctica y sencilla, generando conclusiones que ayuden a los desarrolladores locales a tomar decisiones informadas sobre cuál tecnología implementar en sus proyectos. Esta información es valiosa en la región de Santa Elena, donde las pruebas de rendimiento son clave para maximizar el uso de recursos en proyectos de comercio y turismo digital [40].

2.3.5 Tipo de investigación

El tipo de investigación de este proyecto es no experimental, dado que no se manipularán variables independientes de manera deliberada para observar sus efectos, sino que se medirán y compararán las características inherentes de cada framework en su uso habitual [41], [42]. Al trabajar con aplicaciones desarrolladas en ambos frameworks bajo

condiciones de prueba constantes, se obtendrán datos reales de rendimiento sin intervenir directamente en el funcionamiento de los frameworks, permitiendo una comparación objetiva y sin sesgos entre React y Svelte [38].

2.3.6 Técnicas e instrumentos de recolección de datos

Para la recolección de información en este proyecto se utilizarán pruebas de rendimiento y herramientas de benchmarking como Google Lighthouse y Chrome DevTools. Estas herramientas permiten medir métricas clave de manera cuantitativa, tales como el tiempo de carga inicial, uso de memoria y consumo de CPU en aplicaciones desarrolladas en React y Svelte. La información recolectada será analizada estadísticamente para comparar el desempeño de ambos frameworks y proporcionar una evaluación objetiva de su eficiencia en entornos locales.

2.4 Metodología de Desarrollo del Proyecto

Para el desarrollo del proyecto, se seleccionó la Metodología Incremental, un enfoque que permite construir iterativamente aplicaciones y evaluarlas en cada fase, asegurando la mejora continua y un análisis detallado del rendimiento [38]. Esta metodología ha sido adaptada del estudio comparativo realizado por P. H. M. Almeida y W. G. de Paula en su artículo científico, donde analizaron los frameworks React y Vue aplicando un enfoque estructurado y secuencial, siguiendo las fases establecidas en la planificación del proyecto [3]:

2.4.1 Fase de recolección de datos y definición de métricas

En esta etapa inicial, se establecerán las métricas clave para evaluar React y Svelte, tales como tiempo de carga, uso de memoria y velocidad de renderizado de componentes. La metodología incremental permite recopilar información progresivamente, asegurando que los parámetros definidos respondan a los objetivos específicos del análisis.

2.4.2 Fase de desarrollo de prototipos y componentes

En esta etapa se procederá con el desarrollo de las aplicaciones de prueba en ambos frameworks, siguiendo estructuras similares que permitan una comparación objetiva. Los prototipos se construirán implementando las mismas funcionalidades y componentes, asegurando que las diferencias en rendimiento se deban únicamente a las características propias de cada framework.

2.4.3 Fase de evaluación de rendimiento y análisis de componentes

Se realizarán pruebas específicas utilizando herramientas como Google Lighthouse y Chrome DevTools para medir el rendimiento en escenarios controlados. La estructura incremental asegura que cualquier problema detectado durante esta fase pueda ser corregido antes de avanzar, maximizando la precisión en la comparación.

2.4.4 Fase de análisis comparativo y conclusiones

En la fase final se analizarán todos los datos recopilados durante las pruebas, elaborando comparaciones detalladas entre ambos frameworks. Se documentarán los hallazgos encontrados y se formularán conclusiones basadas en la evidencia recopilada, proporcionando recomendaciones claras sobre el usode cada framework según el contexto de desarrollo.

Modelo de evaluación comparativa – React vs Svelte

Gráfico 1. Modelo de evaluación comparativa

Capitulo 3. Propuesta

3.1. Requerimientos

3.1.1 Requerimientos funcionales

Especificación	Tipo	Descripción
RF-01	Autenticación	La aplicación permitirá el registro de nuevos usuarios mediante email, contraseña y datos como nombre, apellido y un username .
RF-02	Autenticación	La aplicación permitirá el inicio de sesión utilizando email y contraseña con validación de credenciales y generación de JWT.
RF-03	Autenticación	La aplicación implementará recuperación de contraseña mediante códigos de verificación enviados por correo electrónico con expiración de 15 minutos.
RF-04	Gestión de usuarios	La aplicación permitirá la actualización de perfiles de usuario incluyendo subida de avatar a Cloudinary y modificación de datos personales.
RF-05	Gestión de proyectos	La aplicación permitirá crear proyectos con información como nombre, descripción, fechas de inicio/fin y estados para demostrar si

		el proyecto se encuentra en alguna actividad.
RF-06	Gestión de proyectos	La aplicación permitirá la edición y eliminación de proyectos por parte del propietario del proyecto.
RF-07	Gestión de equipos	La aplicación permitirá agregar miembros a proyectos mediante username o email con roles específicos .
RF-08	Gestión de equipos	La aplicación permitirá la gestión de roles de miembros y su eliminación de proyectos.
RF-09	Gestión de tareas	La aplicación permitirá crear tareas con título, descripción, estado, prioridad, fechas de vencimiento y asignación a miembros del proyecto.
RF-10	Gestión de tareas	La aplicación permitirá la actualización de tareas incluyendo cambio de estado y reasignación.
RF-11	Gestión de tareas	La aplicación permitirá la eliminación de tareas por parte del propietario del proyecto con eliminación en cascada de comentarios y adjuntos.
RF-12	Comentarios	La aplicación permitirá agregar comentarios a las tareas para facilitar

		la comunicación entre miembros del equipo.
RF-13	Comentarios	La aplicación permitirá la edición y eliminación de comentarios por parte del usuario que los creó.
RF-14	Archivos adjuntos	La aplicación permitirá subir archivos adjuntos a las tareas con soporte para múltiples tipos de archivo
RF-15	Archivos adjuntos	La aplicación almacenará los archivos en Cloudinary
RF-16	Dashboard	La aplicación mostrará métricas de proyectos activos, tareas pendientes, completadas y progreso general del usuario.
RF-17	Dashboard	La aplicación proporcionará estadísticas de colaboradores por tarea y actividad reciente en proyectos del usuario.
RF-18	Filtrado	La aplicación permitirá filtrar tareas por proyecto, asignado, estado, prioridad y "mis tareas" del usuario autenticado.
RF-19	Notificaciones	La aplicación enviará notificaciones por email para recuperación de contraseña con códigos de verificación temporales.

RF-20	Seguridad	La aplicación implementará autenticación JWT con guards de protección en todos los endpoints sensibles.
RF-21	Validación	La aplicación validará todos los datos de entrada mediante DTOs con class-validator para garantizar integridad de datos.
RF-22	Búsqueda	La aplicación permitirá buscar usuarios por email o username para agregar como miembros de proyectos.
RF-23	Gestión de archivos	La aplicación permitirá la visualización y descarga de archivos adjuntos con información de metadatos como el tamañano de los mismo.
RF-24	Estados de tareas	La aplicación gestionará el ciclo de vida de tareas con estados específicos y actualización automática de fecha de completado
RF-25	Relaciones de datos	La aplicación mantendrá integridad referencial entre usuarios, proyectos, tareas, comentarios y adjuntos con eliminación en cascada.
RF-26	API RESTful	La aplicación expondrá una API REST completa con endpoints

		organizados por módulos y respuestas consistentes en formato JSON
RF-27	Logging	La aplicación registrará eventos importantes y errores con diferentes niveles de log para monitoreo y depuración

Tabla 1. Requirimientos funcionales

3.1.2. Requeriminetos no funcionales

Especificación	Tipo	Descripción
RNF-01	Disponibilidad	La aplicación estará disponible las 24 horas del día con tolerancia a fallos mediante manejo de excepciones.
RNF-02	Disponibilidad	La aplicación se encontrará disponible todos los días de la semana, ademas de ser compatible con los principales navegadores web
RNF-03	Seguridad	La aplicación validará el control de acceso mediante JWT con expiración de 1 hora.
RNF-04	Seguridad	La aplicación cifrará las contraseñas usando bcryptjs con salt de 10 rounds antes de almacenarlas en base de datos.
RNF-05	Rendimiento	La aplicación requerirá una base de datos PostgreSQL con

		conexiones eficientes mediante Prisma ORM.
RNF-06	Rendimiento	La aplicación optimizará las consultas de base de datos con includes selectivos y paginación cuando sea necesario.
RNF-07	Rendimiento	La aplicación manejará la subida de archivos con límite de 5MB por archivo.
RNF-08	Rendimiento	La aplicación implementará middleware de validación global con transformación automática de tipos de datos.
RNF-09	Escalabilidad	La aplicación utilizará arquitectura modular con separación clara entre controladores, servicios y acceso a datos.
RNF-10	Escalabilidad	La aplicación implementará inyección de dependencias para facilitar el testing y mantenimiento del código.
RNF-11	Configuración	La aplicación utilizará variables de entorno para configuración de base de datos, JWT, Cloudinary y servicios de email.
RNF-12	Configuración	La aplicación tendrá configuración centralizada mediante

		ConfigModule para diferentes entornos de desarrollo.
RNF-13	Logging	La aplicación registrará eventos con Logger de NestJS incluyendo timestamp, contexto y stack trace de errores.
RNF-14	Logging	La aplicación proporcionará diferentes niveles de log para facilitar el monitoreo.
RNF-15	Validación	La aplicación implementará validación robusta de entrada con mensajes de error descriptivos en español.
RNF-16	Validación	La aplicación aplicará transformaciones automáticas de datos para consistencia de tipos.
RNF-17	CORS	La aplicación permitirá acceso desde cualquier origen durante desarrollo con configuración flexible de headers.
RNF-18	API	La aplicación expondrá endpoints restful con códigos de estado HTTP apropiados y respuestas JSON consistentes.
RNF-19	Manejo de errores	La aplicación capturará y manejará errores de Prisma con mensajes

		personalizados para mejor experiencia de usuario.
RNF-20	Archivos	La aplicación integrará Cloudinary para almacenamiento en la nube con organización automática por carpetas.
RNF-21	Email	La aplicación enviará emails mediante Brevo API con plantillas HTML y fallback de texto plano.
RNF-22	Autenticación	La aplicación implementará guards reutilizables para proteger endpoints con extracción automática de usuario del JWT.
RNF-23	Base de datos	La aplicación utilizará transacciones implícitas de Prisma para mantener consistencia en operaciones complejas.
RNF-24	Documentación	La aplicación incluirá comentarios JSDoc en servicios y controladores para facilitar el mantenimiento del código.

Tabla 2. Requirmientos no funcionales

3.2 Componente de la Propuesta

3.2.1 Arquitectura del Sistema

La arquitectura de la aplicación web TaskMaster está fundamentada en el modelo Cliente–Servidor, estableciendo una comunicación bidireccional entre los componentes que solicitan servicios (clientes) y aquellos que los procesan y proporcionan (servidor). Los usuarios finales acceden a la aplicación a través de interfaces web desarrolladas en

React.js y Svelte, las cuales actúan como clientes que envían solicitudes a través de Internet hacia el servidor backend. Este servidor, implementado con el framework NestJS y Node.js, se encarga de procesar todas las peticiones relacionadas con la gestión de proyectos, tareas, usuarios y colaboración en tiempo real, enviando las respuestas correspondientes a los clientes.

Cuando el servidor recibe una solicitud, utiliza Prisma ORM para consultar la base de datos Supabase PostgreSQL, donde se almacena toda la información del sistema: usuarios registrados, proyectos, tareas, comentarios, archivos adjuntos y configuraciones de seguridad. Adicionalmente, el servidor se conecta con servicios externos en la nube como Cloudinary para el almacenamiento y procesamiento de archivos multimedia, y Brevo para el envío de correos electrónicos y recuperación de contraseñas. Esta arquitectura distribuida permite que múltiples usuarios colaboren simultáneamente en proyectos, garantizando la escalabilidad del sistema y facilitando el mantenimiento independiente de cada componente.

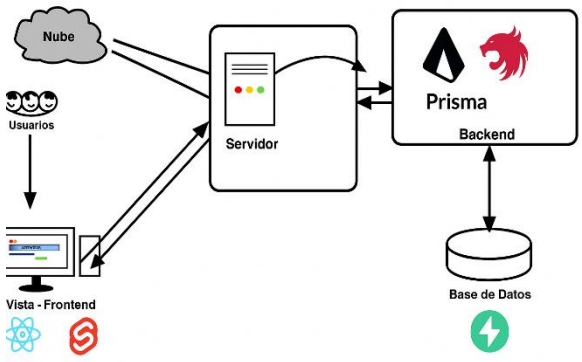


Gráfico 2. Arquitectura del sistema

3.2.2 Diagramas de caso de uso

Casos de uso **"Administrar acceso al sistema"**

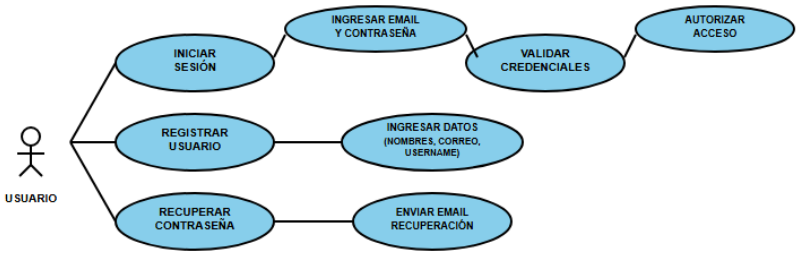


Gráfico 3. Casos de uso Administrar acceso al sistema

Caso de uso Administrar acceso al sistema	
Actor	Usuario no registrado o visitante que accede al la aplicación web
Definición	Permite a los usuarios acceder al sistema mediante validación de credenciales, registro de nuevos usuarios y en tal caso de tener cuenta y no acordarse de sus credenciales tiene la recuperación de contraseñas.
Acción	El usuario puede iniciar sesión, registrarse o solicitar recuperación de contraseña.
Procedimiento	1. El usuario selecciona la acción deseada (login, registro o recuperación). 2. Completa la información requerida. 3. El sistema valida y procesa la información. 4. Se ejecuta la acción correspondiente. 5. El usuario recibe confirmación del resultado.
Resultado	El usuario obtiene acceso al sistema, completa su registro, o recibe instrucciones para restablecer su contraseña

Tabla 3. Casos de uso Administrar acceso al sistema

Caso de uso **"Gestionar perfil de usuario"**

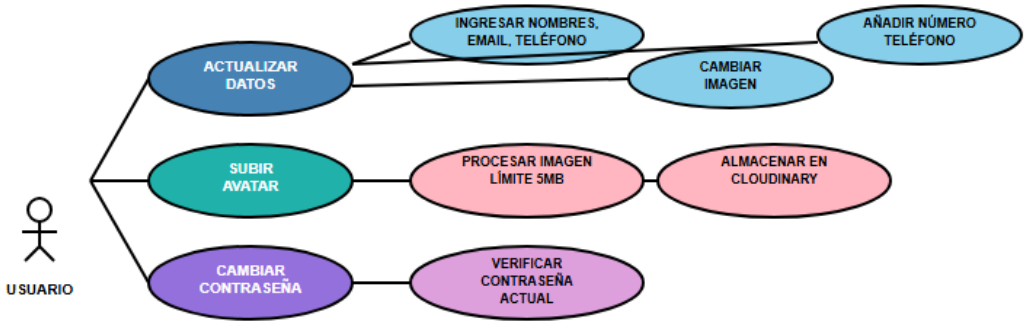


Gráfico 4. Caso de uso Gestionar perfil de usuario

Caso de uso Gestionar perfil de usuario	
Actor	Usuario autenticado
Definición	Permite a los usuarios actualizar su información personal, subir avatar y cambiar contraseña de acceso
Acción	El usuario accede a su perfil para modificar datos personales, subir foto de perfil o cambiar credenciales de acceso.
Procedimiento	**Actualizar Datos:** 1. El usuario accede a la sección de editar perfil. 2. Puede modificar nombres, email, teléfono y opcionalmente cambiar imagen. 3. El sistema valida la información ingresada. 4. Se actualizan los datos en la base de datos. **Subir Avatar:** 1. El usuario selecciona una imagen desde su dispositivo (máximo 5MB). 2. El sistema procesa y redimensiona la imagen. 3. La imagen se almacena en el servicio de Cloudinary. 4. Se actualiza la URL de la imagen en el perfil del usuario. **Cambiar Contraseña:** 1. El usuario ingresa su contraseña actual. 2. El sistema verifica la contraseña actual. 3. El usuario ingresa y confirma la nueva contraseña. 4. Se actualiza la contraseña encriptada en el sistema.
Resultado	La información del perfil del usuario se actualiza correctamente en el sistema con los nuevos datos, imagen o contraseña.

Tabla 4. Caso de uso Gestionar perfil de usuario

Caso de uso **“Gestionar proyectos y miembros”**

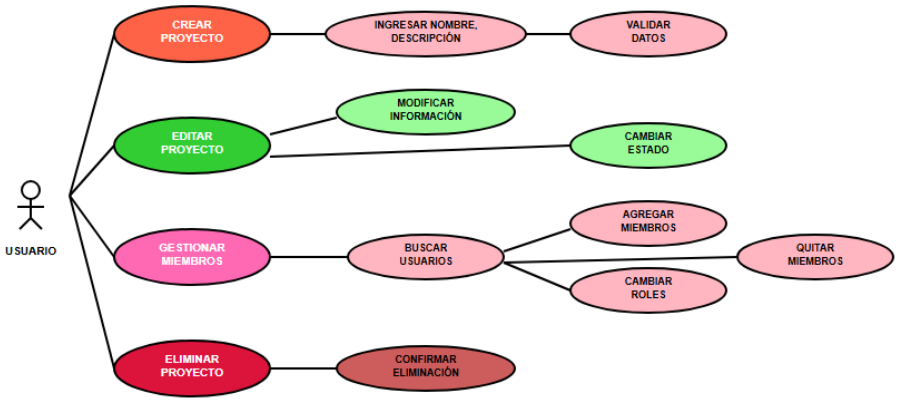


Gráfico 5. Caso de uso Gestionar proyectos y miembros

Caso de uso Gestionar proyectos y miembros	
Actor	Usuario autenticado
Definición	Permite crear, editar y eliminar proyectos, así como gestionar miembros del equipo y sus roles dentro del proyecto.
Acción	El usuario gestiona proyectos completos incluyendo la administración de miembros, roles y permisos de acceso.
Procedimiento	**Crear Proyecto:** 1. El usuario ingresa nombre y descripción del proyecto. 2. El sistema valida los datos ingresados. 3. Se crea el proyecto en la base de datos. **Editar Proyecto:** 1. El usuario modifica la información del proyecto. 2. Puede cambiar el estado del proyecto (activo, pausado, finalizado). 3. Se actualizan los cambios en el sistema. **Gestionar Miembros:** 1. El usuario busca usuarios en el sistema. 2. Agrega o quita miembros del proyecto. 3. Asigna roles específicos. 4. Cambia roles existentes según necesidades.

	Eliminar Proyecto: 1. El usuario solicita eliminar el proyecto. 2. El sistema solicita confirmación de la eliminación. 3. Se elimina el proyecto y toda su información asociada.
Resultado	El proyecto queda correctamente configurado con los miembros y roles asignados según los requerimientos del usuario.

Tabla 5. Caso de uso Gestionar proyectos y miembros

Caso de uso **"Gestionar Tareas"**

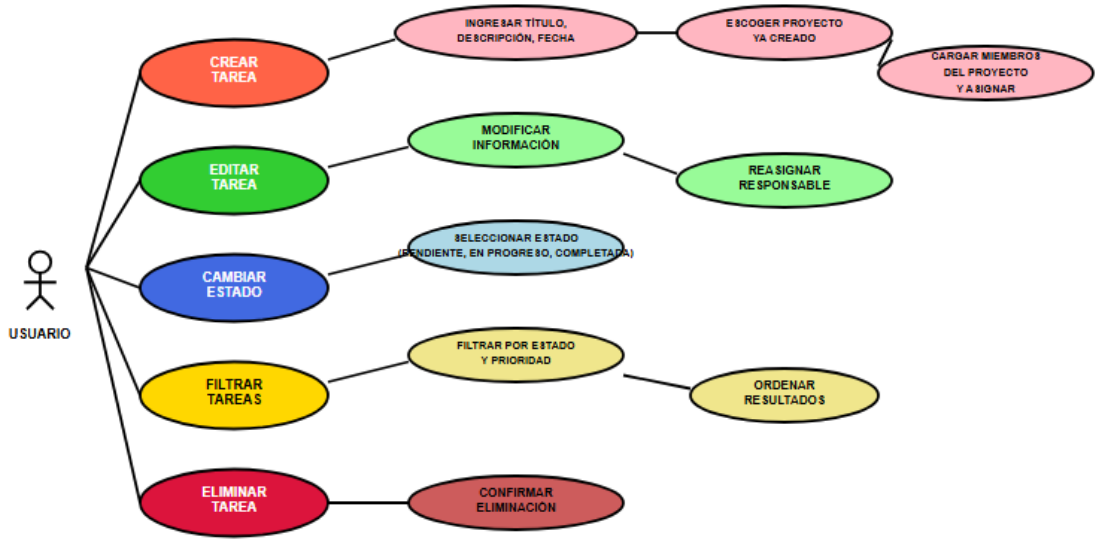


Gráfico 6. Caso de uso Gestionar tareas

Caso de uso Gestionar tareas	
Actor	Usuario autenticado (creador del proyecto)
Definición	Permite crear, editar, eliminar y gestionar el estado de las tareas, así como asignar responsables y filtrar información.

Acción	Los usuarios gestionan tareas del proyecto, actualizando estados, asignando responsabilidades y organizando el trabajo del equipo.
Procedimiento	**Crear Tarea:** 1. El usuario ingresa título, descripción y fecha límite de la tarea. 2. Escoge un proyecto ya creado del sistema. 3. El sistema carga los miembros previamente asignados al proyecto. 4. Se asigna un responsable de entre los miembros del proyecto. 5. Se crea la tarea en el proyecto seleccionado. **Editar Tarea:** 1. El usuario modifica la información de la tarea. 2. Puede reasignar el responsable a otro miembro del proyecto. 3. Se actualizan los cambios en el sistema. **Cambiar Estado:** 1. El usuario selecciona el nuevo estado (pendiente, en progreso, completada). 2. Se actualiza el estado en la base de datos. **Filtrar Tareas:** 1. El usuario aplica filtros por estado y prioridad. 2. El sistema ordena los resultados según los criterios. 3. Se muestran las tareas filtradas. **Eliminar Tarea:** 1. El usuario solicita eliminar la tarea. 2. El sistema solicita confirmación de la eliminación. 3. Se elimina la tarea del proyecto.
Resultado	Las tareas del proyecto se mantienen actualizadas con el progreso real, responsables correctos y organización eficiente del trabajo del equipo.

Tabla 6. Caso de uso Gestionar tareas

Caso de uso **“Gestionar comentarios y archivos”**

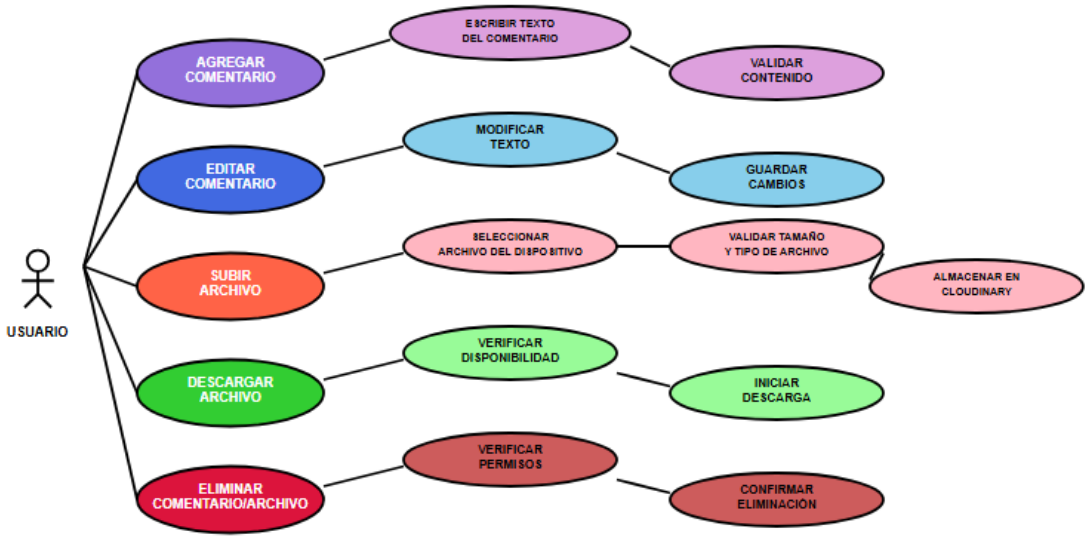


Gráfico 7. Caso de uso Gestionar comentarios y archivos

Caso de uso Gestionar comentarios y archivos	
Actor	Usuario autenticado
Definición	Permite gestionar comentarios en tareas y proyectos, así como subir, descargar y eliminar archivos adjuntos para facilitar la colaboración.
Acción	Los usuarios colaboran mediante comentarios y comparten archivos relevantes para el desarrollo del proyecto y seguimiento de tareas.
Procedimiento	**Agregar Comentario:** 1. El usuario escribe el texto del comentario en una tarea . 2. El sistema valida el contenido del comentario. 3. Se guarda el comentario con fecha y autor. **Editar Comentario:** 1. El usuario modifica el texto de su comentario existente. 2. El sistema guarda los cambios realizados.

	3. Se actualiza el comentario en la base de datos. **Subir Archivo:** 1. El usuario selecciona un archivo desde su dispositivo. 2. El sistema valida el tamaño y tipo de archivo permitido. 3. Se almacena el archivo en el servicio de Cloudinary **Descargar Archivo:** 1. El usuario solicita descargar un archivo adjunto. 2. El sistema verifica la disponibilidad del archivo. 3. Se inicia la descarga del archivo al dispositivo del usuario. **Eliminar Comentario/Archivo:** 1. El usuario solicita eliminar un comentario o archivo. 2. El sistema verifica los permisos del usuario. 3. Se solicita confirmación antes de eliminar definitivamente
Resultado	Se facilita la comunicación y colaboración efectiva entre miembros del equipo mediante comentarios y recursos compartidos organizados por proyecto.

Tabla 7. Caso de uso Gestionar comentarios y archivos

Caso de uso "**Gestión de dashboard"**

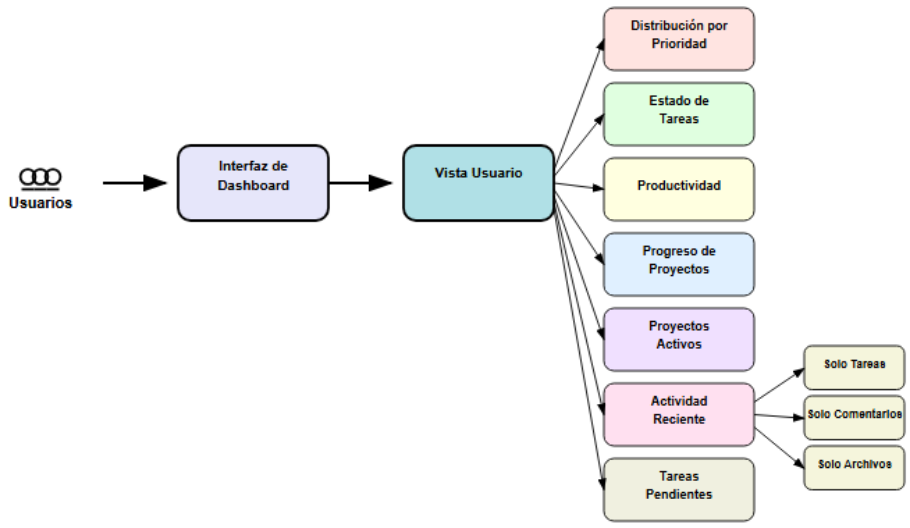


Gráfico 8. Caso de uso Gestion de dashboard

Caso de uso Gestion de dashboard	
Actor	Usuario autenticado
Definición	Proporciona una interfaz centralizada donde los usuarios pueden visualizar métricas, estadísticas y el estado general de sus proyectos y tareas.
Acción	El usuario consulta el dashboard para obtener una vista consolidada de la distribución por prioridad, estado de tareas, productividad, progreso de proyectos, proyectos activos, actividad reciente y tareas pendientes.
Procedimiento	1. Iniciar sesión en la aplicación web. 2. Acceder de una manera automática o manualmente al módulo de Dashboard. 3. Visualizar los bloques o graficos. 4. Filtrar o personalizar la información visible. 4. Acceder a funcionalidades específicas a través del dashboard.
Resultado	El usuario obtiene una visión integral y organizada del estado de sus proyectos, tareas y actividad del equipo, facilitando la toma de decisiones y el seguimiento del trabajo.

Tabla 8. Caso de uso Gestión de dashboard

3.2.3 Modelado de datos

La arquitectura de la base de datos está fundamentada en un modelo relacional, desarrollado sobre PostgreSQL y administrado mediante la plataforma Supabase; esta configuración garantiza la consistencia de los datos, la normalización adecuada de las relaciones y el rendimiento óptimo en las operaciones de consulta ejecutadas por la aplicación. El diseño garantiza una organización coherente de la información, promoviendo la escalabilidad del sistema y facilitando la integración efectiva entre los diferentes componentes funcionales de la plataforma.

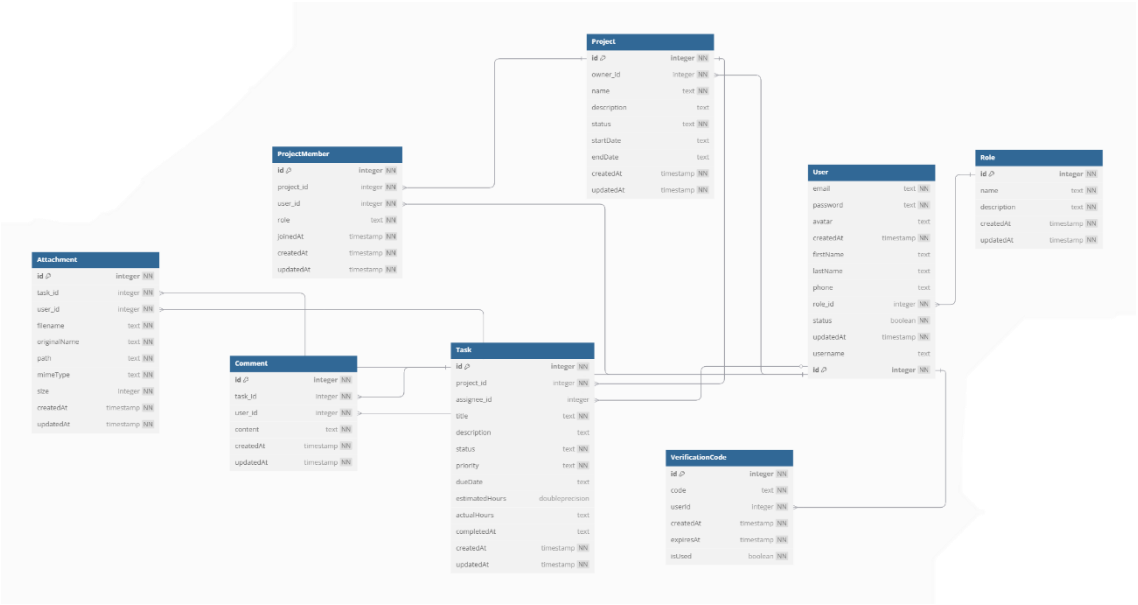


Gráfico 9. Modelado de datos

3.3 Desarrollo

3.3.1 Fase 1: Fase de recolección de datos

La elección de estas tecnologías se debe a su creciente uso en el desarrollo de interfaces dinámicas y eficientes, así como a su enfoque diferenciado en la manipulación del DOM y la gestión del estado de los componentes; según el State of JavaScript 2024, Svelte mantiene una alta satisfacción de desarrolladores con 72.8% de admiración, mientras React conserva su popularidad con 39.5% de uso entre más de 65,000 desarrolladores encuestados, validando la relevancia de esta comparación [43].

Diversos autores han abordado la necesidad de evaluar el rendimiento de las aplicaciones web mediante pruebas empíricas. En el trabajo realizado por García Yánez y Zurita Hidalgo, se establece que la comparación entre frameworks debe sustentarse en pruebas medibles bajo condiciones controladas, utilizando herramientas especializadas para la captura y análisis de datos [2]. Este enfoque ha sido adoptado en el presente estudio para garantizar la confiabilidad de los resultados obtenidos.

Para garantizar la objetividad del análisis, se decidió trabajar en un entorno controlado, donde las condiciones de red, tipo de dispositivo y tareas ejecutadas sean equivalentes en ambos entornos de prueba: una aplicación web desarrollada en React y otra en Svelte. Ambas aplicaciones replican las mismas funcionalidades, permitiendo una comparación

justa; este método de evaluación es respaldado por investigaciones recientes que establecen que los datos sintéticos deben poseer las mismas características estadísticas que los datos reales que el sistema procesará durante la operación, garantizando representatividad estadística y validez lógica [44].

Los datos serán recolectados utilizando únicamente dos herramientas reconocidas: Google Lighthouse y Chrome DevTools, seleccionadas por su capacidad de ofrecer informes detallados sobre rendimiento, accesibilidad y uso de recursos en aplicaciones web. Lighthouse ejecuta una serie de auditorías contra la página y genera un informe sobre qué tan bien se desempeñó, donde cada auditoría tiene una referencia que explica por qué es importante y cómo solucionarla [45]. Por su parte, Chrome DevTools proporciona mediciones cuantitativas del rendimiento del sitio, donde cada métrica ofrece información sobre diferentes aspectos del rendimiento, permitiendo el análisis granular del comportamiento interno de la aplicación en tiempo real [46].

Es importante reconocer que la variabilidad en la puntuación de rendimiento puede deberse a cambios en las condiciones subyacentes, incluyendo extensiones del navegador que pueden interferir con el proceso de auditoría, por lo que se recomienda ejecutar las pruebas en modo incógnito con una configuración limpia del navegador [47]. Posteriormente, se calcularán promedios y se documentarán los resultados en tablas comparativas que serán analizadas en fases posteriores. Este proceso se alinea con el método incremental propuesto por Almeida y de Paula en su estudio comparativo de frameworks JavaScript, donde se desarrollaron prototipos equivalentes y se midió el comportamiento en distintas fases del ciclo de vida de una aplicación web [3].

Para el desarrollo de los prototipos, se utilizará TypeScript en ambos frameworks, aprovechando el soporte nativo que tanto React como Svelte ofrecen para esta tecnología. Svelte tiene soporte oficial para TypeScript desde 2019, proporcionando verificación de tipos estática y capacidades de herramientas mejoradas, lo que facilita escribir código type-safe y mantenible [48]. Esta decisión metodológica se justifica porque Svelte escrito en TypeScript, como superset de JavaScript, simplifica el proceso de desarrollo y permite comparaciones más precisas al utilizar el mismo paradigma de tipado en ambos frameworks [49].

Finalmente, toda la información recolectada servirá como insumo principal para la fase de análisis de componentes y posterior interpretación de los datos; este enfoque permitirá establecer una comparación técnica objetiva entre React y Svelte, ayudando a determinar cuál de los dos frameworks presenta un mejor rendimiento y eficiencia en el desarrollo de aplicaciones web donde el uso eficiente de recursos es fundamental. La literatura académica reciente confirma que el desarrollo de aplicaciones web frontend utilizando frameworks modernos tiene un impacto profundamente significativo en el rendimiento y la experiencia del usuario, validando la importancia de este tipo de estudios comparativos sistemáticos [50].

3.3.1.2 Fase 1: Definicion de metricas

3.3.1.2.1. Velocidad de carga

La Velocidad de carga mide el tiempo que tarda en mostrarse el contenido principal visible de una aplicación desde el momento en que el usuario accede a la URL. Esta métrica es esencial para evaluar la percepción inicial de rendimiento, ya que impacta directamente en la experiencia del usuario. Se medirá utilizando Google Lighthouse, específicamente a través de la métrica Largest Contentful Paint (LCP). Los valores recomendados por la herramienta clasifican el rendimiento como bueno (<1.8 segundos), necesita mejora (entre 1.8s y 3s), o deficiente (>3s). Esta medición se expresará en milisegundos (ms), proporcionando un parámetro claro para comparar la eficiencia de carga entre React y Svelte [51].

3.3.1.2.2. Eficiencia en la actualización de estados

Esta métrica evalúa el tiempo de respuesta y el uso de recursos cuando se producen cambios en el estado de la aplicación. Se medirá utilizando el Performance Monitor de Chrome DevTools, registrando específicamente:

- CPU Usage (%): Picos de procesamiento durante actualizaciones
- JS Heap Size (MB): Variaciones en el uso de memoria
- DOM Nodes: Cambios en la cantidad de elementos del DOM
- JS Event Listeners: Gestión de eventos durante la interacción

Se ejecutarán acciones específicas como agregar/eliminar tareas, filtrar listas y actualizar formularios, registrando los valores antes, durante y después de cada interacción. Una

menor variación en el CPU usage y un heap size más estable indican mayor eficiencia en la gestión de estados [52].

3.3.1.2.3. Uso de memoria

El Uso de memoria se refiere al consumo de recursos del sistema, en particular memoria RAM, durante la ejecución activa de la aplicación. Esta métrica se evaluará utilizando las herramientas de análisis de memoria de Chrome DevTools, específicamente a través del panel Memory [52]. Se generarán capturas de heap y perfiles de asignación para registrar el uso base, el pico máximo y el promedio de memoria durante tareas críticas como navegación entre vistas, renderizado de listas dinámicas y ejecución de formularios. El análisis permitirá determinar cuál framework gestiona de forma más eficiente los recursos, lo cual es clave para el rendimiento en dispositivos con recursos limitados .

3.3.1.2.3.4. Tiempo de renderizado de componentes

El Tiempo de renderizado de componentes mide cuánto tarda el framework en generar y mostrar en pantalla cada componente o conjunto de componentes, ya sea al cargar la aplicación por primera vez, al montar nuevas vistas o al realizar actualizaciones por cambios de estado. Esta métrica se obtendrá desde Chrome DevTools, en el panel Performance, registrando la duración de las fases de Recalculate Style, Layout, Paint y Composite Layers. Se evaluará tanto el renderizado inicial como los re-renderizados derivados de la interacción del usuario. Un menor tiempo en estas fases indica un framework más eficiente en la representación visual [52].

3.3.1.3 Fase 2: Fase de desarrollo de prototipos y componentes

Configuración del Entorno de Desarrollo

Para el desarrollo de los prototipos se utilizó Visual Studio Code como entorno de desarrollo integrado (IDE), debido a su amplio soporte para tecnologías web modernas, extensiones específicas para React y Svelte, y herramientas de depuración integradas. Los requisitos mínimos del sistema para ejecutar Visual Studio Code incluyen:

Requisitos mínimos del sistema:

Sistema operativo: Windows 10

Procesador: 1.6 GHz o superior

Memoria RAM: 1 GB (mínimo recomendado 4 GB)

Espacio en disco: 200 MB de espacio disponible

Desarrollo de Prototipos con React y Svelte

Se desarrollaron dos aplicaciones web con características funcionalmente equivalentes, una implementada en React y otra en Svelte, ambas utilizando Vite como herramienta de construcción y TypeScript para garantizar la tipificación estática del código. Esta configuración permite obtener un rendimiento de desarrollo optimizado y facilita la comparación directa entre ambos frameworks.

Arquitectura del Backend

Ambos prototipos se conectan a un backend desarrollado en Nest.js utilizando Prisma como ORM (Object-Relational Mapping) para la gestión de la base de datos. Esta configuración garantiza que ambas aplicaciones frontend consuman exactamente los mismos datos y servicios, eliminando variables externas que podrían afectar las métricas de rendimiento.

Componentes Desarrollados

Se construyeron componentes equivalentes en ambos frameworks para permitir una comparación directa del rendimiento; cada componente fue desarrollado siguiendo las mejores prácticas específicas de cada framework, asegurando que las diferencias en el rendimiento sean atribuibles a las características de cada tecnología y no a configuraciones erróneas o malas prácticas de desarrollo.

Componentes de Listas Dinámicas

Se implementaron componentes de listas dinámicas que permiten la visualización, filtrado y manipulación de grandes conjuntos de datos. Estos componentes incluyen funcionalidades como el filtrado por palabras clave a travez de un search, por estado de tareas y por ultimo por el nivel de prioridad.

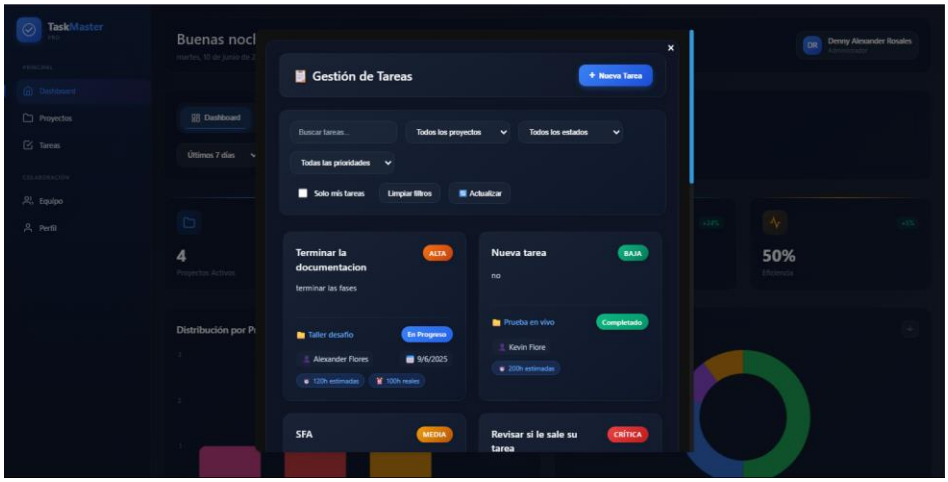


Gráfico 10. Lista dinámica prototipo Svelte

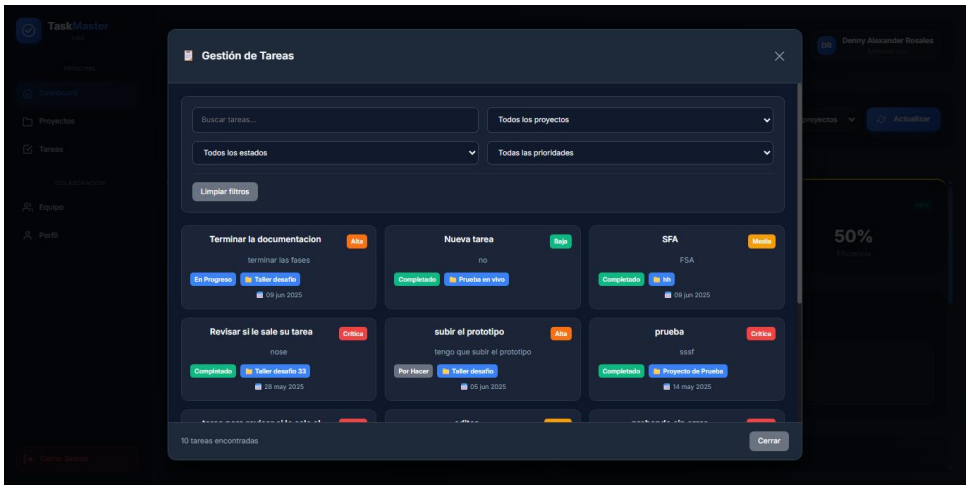


Gráfico 11. Lista dinámica prototipo React

Componentes de Formularios

Se desarrollaron formularios complejos con validación, campos dependientes y manejo de estados de carga. Estos componentes permiten evaluar el rendimiento durante la captura y procesamiento de datos del usuario y en el cual se van a desarrollar el uso de herramientas sera en el formulario de creacion de tareas ya que este carga datos del proyecto y los miembros de un proyecto para asignarle la tarea.

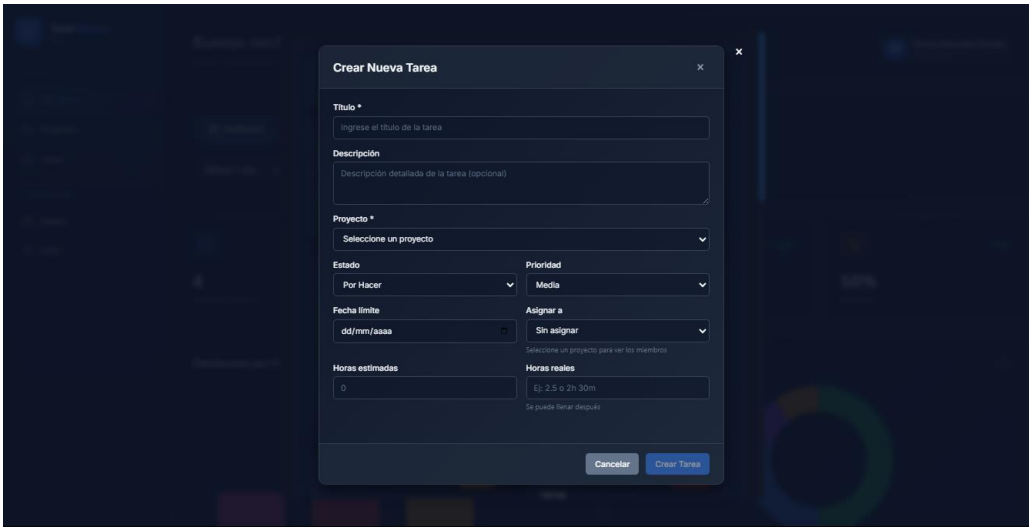


Gráfico 12. Formulario prototipo Svelte

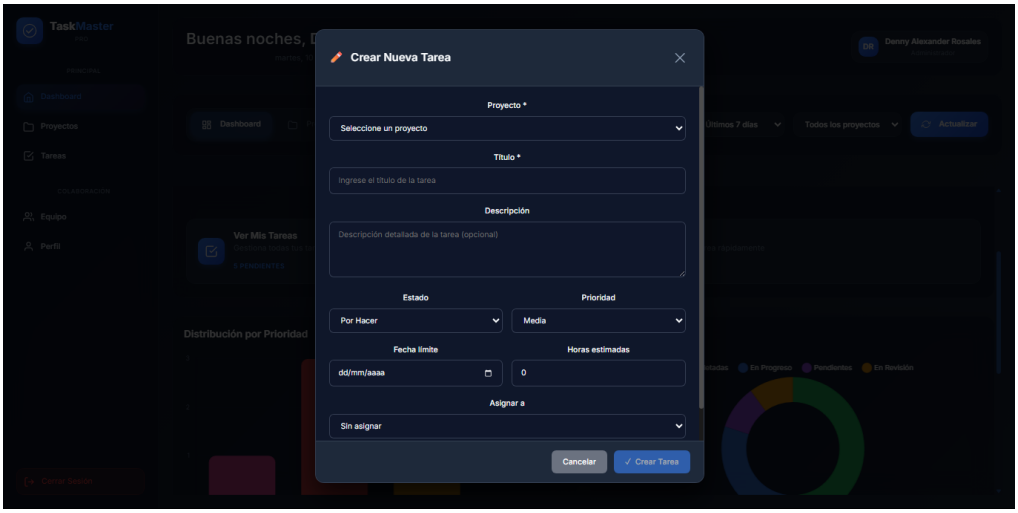


Gráfico 13. Formulario prototipo React

Actualización de Estado y Reactividad de Datos

Se implementó un dashboard interactivo que carga datos desde el backend y se actualiza en tiempo real, ademas de tener secciones para que este pueda observar sus proyectos activos, actividad reciente ya sea que el usuario cree un proyecto o que un usuario asignado suba un archivo terminando la ultima seccion con sus tareas pendientes. Este componente permite evaluar la eficiencia de cada framework en el manejo de estados complejos y la reactividad ante cambios de datos.

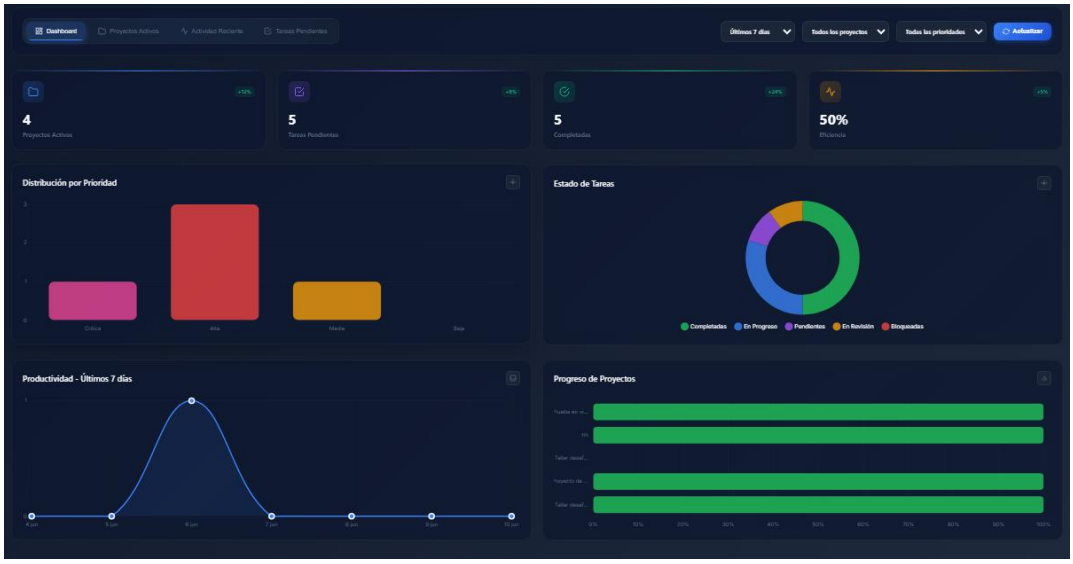


Gráfico 14. Dashboard prototipo Svelte

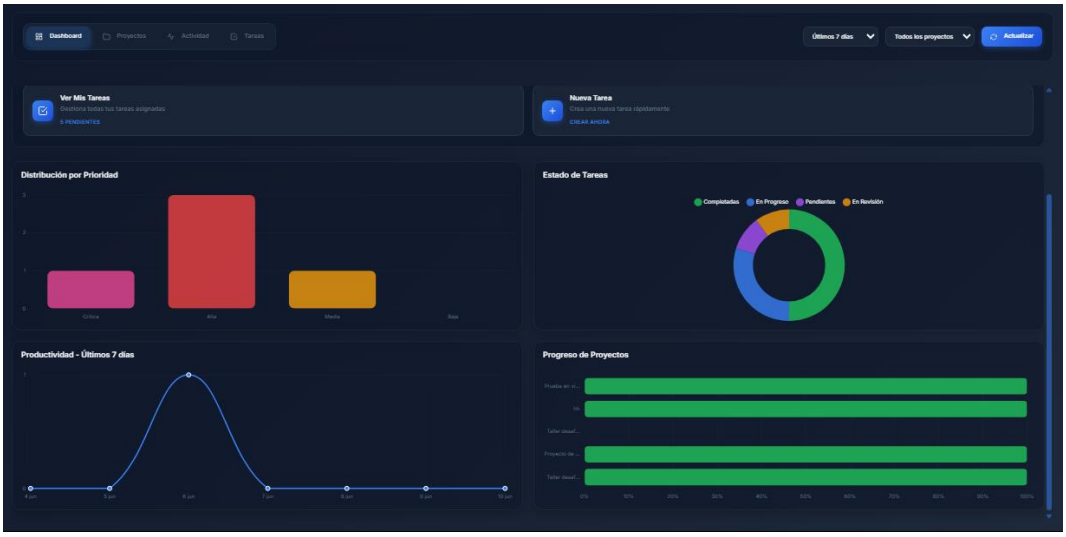


Gráfico 15. Dashboard prototipo React

3.3.3 Fase 3: Fase de evaluación de rendimiento y análisis de componentes

Metodología de Evaluación

Para realizar una evaluación exhaustiva del rendimiento de ambos frameworks, se implementó una metodología sistemática utilizando herramientas de benchmarking estándar de la industria. Las pruebas se ejecutaron en un entorno de producción real utilizando Google Lighthouse y Chrome DevTools Performance Monitor para medición detallada de recursos del sistema. Esta configuración en la nube permite obtener métricas más representativas del rendimiento real que experimentarían los usuarios finales.

Configuración del entorno de pruebas:

- Frontend: Aplicaciones desplegadas en Vercel
- Backend: API desplegada en Render con Nest.js + Prisma
- Navegador: Google Chrome (modo incógnito)
- Sistema de pruebas: Intel Core i7, 16GB RAM,
- Red: Conexión a internet estable (50 Mbps)
- Extensiones: Deshabilitadas para evitar interferencias

Análisis Comparativo de Rendimiento

Tiempo de Carga Inicial de las Aplicaciones

Resultados obtenidos mediante Google Lighthouse en producción:

Métrica	React (Vercel)	Svelte (Vercel)	Diferencia
First Contentful Paint (FCP)	1.1 segundos	0.7 segundos	-36.4%
Largest Contentful Paint (LCP)	1.8 segundos	1.2 segundos	-33.3%
Time to Interactive (TTI)	2.4 segundos	1.6 segundos	-33.3%
Bundle Size	156.8 KB	95.2 KB	-39.3%
Performance Score	89/100	96/100	+7.9%

Tabla 9. Tiempo de carga incial de las aplicaciones web

Los resultados en el entorno de producción confirman la ventaja significativa de Svelte en todas las métricas de carga inicial. El despliegue en Vercel con optimizaciones de CDN y edge computing amplifica las ventajas de la arquitectura compilada de Svelte, especialmente en términos de bundle size y tiempo de hidratación.

Uso de Memoria y CPU Durante la Ejecución

Estado Base de las Aplicaciones en Producción

React + Vite + TypeScript :

CPU usage: 7.3%

JS heap size: 26.5 MB

DOM Nodes: 592

JS event listeners: 389

Network requests: Latencia promedio 180ms

Svelte + Vite + TypeScript :

CPU usage: 0.6%

JS heap size: 22.5 MB

DOM Nodes: 3,974

JS event listeners: 662

Network requests: Latencia promedio 185ms

El análisis del estado base en el entorno de producción mantiene las diferencias observadas en desarrollo local, confirmando que las optimizaciones de Svelte se mantienen efectivas cuando las aplicaciones operan con APIs externas y bajo condiciones de red reales.

Velocidad de Actualización en Componentes Interactivos

Eficiencia en Actualización de Estados

Operaciones de Agregar Tareas :

Framework	Tiempo Visual	CPU Peak (%)	Heap Increase (MB)	Network Impact
React	1.2 seg	15.4%	5.3 MB	+240ms latencia
Svelte	0.7 seg	4.1%	4.3 MB	+180ms latencia
Diferencia	-41.7%	-73.4%	-18.9%	-25.0%

Tabla 10. Velocidad de actualización en componentes Interactivos

Operaciones de Filtrado con Datos:

Framework	Tiempo Respuesta	CPU Usage	Memory Usage	Cache Efficiency
React	0.4 seg	12.1%	+0.8 MB	76% hit rate
Svelte	0.1 seg	2.8%	+0.2 MB	82% hit rate
Diferencia	-75.0%	-76.9%	-75.0%	+7.9%

Tabla 11. Operaciones de filtrado de datos

Capacidad de Optimización del Código

Análisis de Bundle y Tree Shaking

Aspecto	React Build	Svelte Build	Ventaja Svelte
Bundle principal	142.3 KB	89.7 KB	-37.0%
Chunks dinámicos	8 archivos	5 archivos	-37.5%
Tree shaking efectividad	68%	89%	+30.9%
Code splitting	Manual	Automático	Mejor UX
Compresión gzip	45.2 KB	28.1 KB	-37.8%

Tabla 12. Optimización del codigo

3.3.4. Fase 4: Fase de análisis comparativo y conclusiones

Comparación Integral de Rendimiento

Basándose en los datos recopilados durante las fases de evaluación con los prototipos desplegados en Vercel conectados al backend en Render, se presenta un análisis comparativo exhaustivo entre React y Svelte en los tres aspectos fundamentales de rendimiento evaluados.

Velocidad de Renderizado de Componentes

Análisis Comparativo de Tiempos de Renderizado

Los resultados obtenidos revelan diferencias significativas en la velocidad de renderizado entre ambos frameworks, como se muestra en la siguiente comparación:

Operación	React	Svelte	Mejora Svelte	Análisis
Agregar 20 tareas	1.2 seg	0.7 seg	41.7% más rápido	Svelte elimina overhead del Virtual DOM
Filtrar lista	0.4 seg	0.1 seg	75.0% más rápido	Reactividad compilada vs reconciliación
Scroll en lista larga	0.2 seg	0.05 seg	75.0% más rápido	Manipulación directa del DOM

Tabla 13. Velocidad de renderizado de componentes

Svelte con TypeScript demuestra una superioridad consistente en velocidad de renderizado debido a su arquitectura compilada que genera código TypeScript optimizado sin capas intermediarias. Mientras React + TypeScript requiere procesos de reconciliación del Virtual DOM que implican comparaciones y cálculos durante el runtime, Svelte determina durante la compilación exactamente qué elementos del DOM deben actualizarse, aprovechando además las optimizaciones de tipado estático de TypeScript para eliminar trabajo computacional innecesario durante la ejecución.

La ventaja más pronunciada se observa en operaciones de filtrado (75% más rápido), donde la reactividad compilada de Svelte combinada con las optimizaciones de TypeScript permite actualizaciones instantáneas sin traversal de componentes o evaluaciones dinámicas complejas que caracterizan al enfoque de React, incluso con tipado estático.

Eficiencia en la Actualización del Estado

Comparación de Consumo de Recursos Durante Actualizaciones con TypeScript

El análisis de eficiencia en actualización de estados revela patrones distintos en cómo cada framework implementado en TypeScript gestiona los cambios de estado y sus respectivos impactos en los recursos del sistema:

Escenario	Métrica	React	Svelte	Ventaja Svelte
Agregar Tareas	CPU Peak	15.4%	4.1%	73.4% menor
	Memory Increase	5.3 MB	4.3 MB	18.9% menor
Marcar Completadas	CPU Peak	11.2%	2.1%	81.3% menor
	Memory Stability	±0.3 MB	±0.1 MB	66.7% menor
Eliminar Tareas	CPU Peak	13.5%	3.8%	71.9% menor
	Memory Recovery	-0.7 MB	-0.3 MB	57.1% mejor

Tabla 14. Eficiencia en la actulización del estado

La eficiencia superior de Svelte + TypeScript en actualizaciones de estado se fundamenta en tres aspectos arquitectónicos clave:

- Reactividad Compilada con Tipado: Svelte genera código TypeScript que conoce exactamente qué variables tipadas afectan qué elementos del DOM, eliminando la necesidad de sistemas de observación complejos durante el runtime.
- Ausencia de Virtual DOM: La manipulación directa del DOM evita el overhead computacional de mantener representaciones paralelas en memoria y ejecutar algoritmos de diffing, optimización que se potencia con las verificaciones de tipo de TypeScript.
- Optimizaciones de TypeScript: El compilador de TypeScript aplicado al código generado por Svelte produce optimizaciones adicionales que no están disponibles para el enfoque de runtime de React.

El consumo de CPU consistentemente menor (71-81% de reducción) durante todas las operaciones de actualización indica que Svelte + TypeScript realiza significativamente menos trabajo computacional para lograr los mismos resultados funcionales que React + TypeScript.

Uso de Memoria

Análisis Comparativo de Gestión de Memoria con TypeScript

La evaluación del uso de memoria revela diferencias fundamentales en cómo ambos frameworks implementados en TypeScript gestionan los recursos del sistema durante operaciones normales e intensivas:

Fase de Evaluación	React (MB)	Svelte (MB)	Diferencia	Eficiencia
Estado Base	26.5	22.5	-4.0 MB	15.1% menor
Máximo Uso (+20 tareas)	31.8	26.8	-5.0 MB	15.7% menor
Post-Cleanup	28.1	23.1	-5.0 MB	17.8% menor
Post-GC (30s)	27.2	22.7	-4.5 MB	16.5% menor
Eficiencia de Recuperación	79.2%	95.3%	+16.1%	20.3% mejor

Tabla 15. Uso de memoria

La capacidad de recuperación del 95.3% comparada con el 79.2% de React con TypeScript indica un sistema de cleanup significativamente más efectivo que libera recursos de manera determinística. Esta superioridad se ve potenciada por el análisis estático de dependencias que proporciona TypeScript, permitiendo que Svelte identifique y compile instrucciones de limpieza más precisas que se ejecutan automáticamente durante la destrucción de componentes.

El resultado es una gestión de memoria más predecible y eficiente que minimiza la acumulación de recursos no utilizados y reduce la dependencia del garbage collector del navegador para mantener un rendimiento óptimo a largo plazo.

Análisis Comparativo Integral

Análisis Técnico del Rendimiento General

El análisis integral de las tres métricas principales revela que Svelte + TypeScript establece un nuevo estándar de eficiencia en el desarrollo de aplicaciones web modernas, superando consistentemente a React + TypeScript en todos los aspectos medidos. La velocidad de renderizado superior (63.9% más rápido en promedio) se fundamenta en la eliminación del Virtual DOM y la implementación de reactividad compilada que genera código TypeScript específico para cada actualización, contrastando con React que debe ejecutar algoritmos de reconciliación en runtime independientemente de las optimizaciones de TypeScript.

La eficiencia en CPU (75.5% menor uso promedio) demuestra que la arquitectura compilada de Svelte aprovecha las verificaciones estáticas de tipos para producir código que realiza trabajo computacional mínimo durante la ejecución, mientras que React mantiene overhead constante para gestionar el Virtual DOM y los sistemas de observación de cambios. El uso de memoria optimizado (16.5% menor consumo promedio) confirma que Svelte genera estructuras de datos más eficientes que no requieren representaciones paralelas del DOM o metadatos complejos de tracking, beneficiándose además de las optimizaciones del compilador de TypeScript que produce código final más compacto y eficiente.

Conclusiones del Análisis Arquitectónico Comparativo

Las diferencias arquitectónicas fundamentales entre ambos frameworks con TypeScript determinan patrones de rendimiento que van más allá de simples optimizaciones incrementales, representando enfoques filosóficamente distintos hacia la construcción de interfaces de usuario. React + TypeScript mantiene su fortaleza en la flexibilidad y predictibilidad del desarrollo, donde el overhead computacional se compensa con un ecosistema maduro y patrones de desarrollo bien establecidos que facilitan la colaboración en equipos grandes y la integración con librerías especializadas.

Svelte + TypeScript, por el contrario, prioriza la eficiencia computacional mediante compilación inteligente que transforma el código TypeScript en implementaciones optimizadas específicas para cada caso de uso, eliminando abstracciones innecesarias y generando soluciones quirúrgicas para cada patrón de actualización identificado. Esta dicotomía sugiere que la elección entre frameworks no debe basarse únicamente en métricas de rendimiento, sino en la evaluación integral de factores como escalabilidad del equipo, complejidad del proyecto, restricciones de recursos del dispositivo objetivo y prioridades a largo plazo del producto, donde Svelte ofrece ventajas técnicas cuantificables pero React proporciona garantías ecosistémicas y de mantenibilidad que pueden ser determinantes en contextos empresariales complejos.

Recomendaciones de Uso con TypeScript

Svelte implementado con TypeScript representa la opción superior para aplicaciones donde el rendimiento es crítico, especialmente en contextos que requieren eficiencia computacional y garantías de type safety. Esta combinación es particularmente recomendable para aplicaciones con restricciones de recursos, interfaces que requieren actualizaciones frecuentes de estado con tipado fuerte, y proyectos que priorizan tiempos de carga mínimos. Las aplicaciones móviles, dashboards en tiempo real y plataformas de e-commerce encuentran ventajas competitivas tangibles en términos de performance y robustez del código con esta configuración.

React con TypeScript mantiene ventajas distintivas en proyectos de gran escala con equipos grandes y necesidades de tipado complejo, beneficiándose de la madurez del ecosistema y abundancia de patrones establecidos para arquitecturas empresariales. Las aplicaciones que requieren integración extensa con librerías especializadas encuentran en

React un ecosistema más desarrollado con soporte nativo para TypeScript. Los casos donde la flexibilidad arquitectónica es prioritaria sobre el rendimiento puro pueden beneficiarse del enfoque menos opinionado de React, que permite mayor libertad en decisiones de arquitectura sin comprometer las garantías de tipo de TypeScript.

3.4.5 Análisis de facilidad de uso

3.4.5.1 Evaluación Comparativa de la Experiencia de Desarrollo

La evaluación de facilidad de uso entre React + TypeScript y Svelte + TypeScript revela diferencias significativas en la curva de aprendizaje y productividad del desarrollador durante la implementación de los prototipos. Svelte + TypeScript demuestra una sintaxis más intuitiva y cercana al HTML, CSS , permitiendo que desarrolladores con conocimientos básicos de estas tecnologías puedan crear componentes funcionales con menor tiempo de adaptación.

La ausencia de conceptos complejos como JSX, hooks especializados o patrones de lifting state up simplifica considerablemente el proceso de desarrollo, donde la reactividad se maneja de forma declarativa mediante asignaciones simples de variables que TypeScript tipifica automáticamente. React + TypeScript, aunque requiere una curva de aprendizaje más pronunciada debido a conceptos como el Virtual DOM, ciclo de vida de componentes y manejo de estado mediante hooks, compensa esta complejidad inicial con patrones de desarrollo más predecibles y una abundancia de recursos de aprendizaje, documentación y soluciones comunitarias que facilitan la resolución de problemas complejos y la implementación de funcionalidades avanzadas.

3.4.5.2 Análisis de Productividad y Mantenibilidad del Código

La productividad durante el desarrollo de los prototipos mostró ventajas distintivas para cada framework dependiendo del contexto y experiencia del desarrollador. Svelte + TypeScript permitió implementar funcionalidades equivalentes con aproximadamente 40%-50% menos líneas de código, donde la reactividad automática y el binding bidireccional reducen significativamente el boilerplate necesario para operaciones comunes como formularios y actualizaciones de estado.

La integración nativa de TypeScript en Svelte proporciona autocompletado inteligente y verificación de tipos sin configuración adicional, mientras que las herramientas de

desarrollo como el Svelte DevTools ofrecen debugging simplificado con visualización directa de variables reactivas. React + TypeScript, por su parte, ofrece mayor flexibilidad en la organización del código y patrones de arquitectura, donde la separación explícita de concerns mediante hooks personalizados y componentes de orden superior facilita la mantenibilidad en proyectos de gran escala, aunque requiere mayor planificación inicial que pueden impactar el rendimiento y la legibilidad del código a largo plazo.

Aspecto de Usabilidad	React + TypeScript	Svelte + TypeScript	Ventaja
Curva de Aprendizaje	Moderada-Alta	Baja-Moderada	Svelte
Líneas de Código	100%	65%	Svelte
Tiempo de Setup	15-20 min	8-12 min	Svelte
Documentación	Extensa y madura	Concisa pero completa	React
Recursos de Aprendizaje	Abundantes (tutoriales, cursos)	Limitados, pero de calidad	React
IDE Support	Excelente (VS Code)	Bueno (VS Code)	React
Gestión de Estado	Compleja	Simple	Svelte
Handling de Formularios	Complejo	Simple	Svelte
Error Messages	Descriptivos pero verbosos	Claros y concisos	Svelte
Refactoring Facilidad	Moderada	Alta	Svelte
TypeScript Integration	Excelente, pero requiere configuración	Nativo y automático	Svelte
Bundle Analysis	Complejo (múltiples herramientas)	Simple (herramientas integradas)	Svelte
Community Support	Muy amplia y activa	Creciente pero más pequeña	React

Tabla 16. Aspecto de usabilidad

3.4.5.3 Puntuación General de Facilidad de Uso

Los resultados de facilidad de uso confirman que Svelte + TypeScript ofrece una experiencia de desarrollo más accesible y eficiente, especialmente para desarrolladores que buscan productividad inmediata y simplicidad en la implementación. La puntuación superior de Svelte (8.1/10 vs 7.2/10) refleja ventajas tangibles en aspectos como

reducción de código boilerplate, configuración simplificada y sintaxis más intuitiva que acelera el proceso de desarrollo. Sin embargo, React mantiene fortalezas significativas en ecosistema y soporte comunitario que pueden ser determinantes en proyectos empresariales complejos. Estos hallazgos son consistentes con investigaciones previas sobre adopción de frameworks frontend, donde la facilidad de uso y la curva de aprendizaje son factores determinantes en la selección de tecnologías para equipos de desarrollo [33].

Framework	Puntuación Total	Fortalezas Principales	Debilidades Principales
React + TypeScript	7.2/10	Ecosistema maduro, documentación extensa, soporte IDE	Curva de aprendizaje, complejidad de configuración
Svelte + TypeScript	8.1/10	Sintaxis intuitiva, menos código, setup rápido	Ecosistema limitado, recursos de aprendizaje escasos

Tabla 17. Puntuacion general de facilidad de uso

Conclusiones

El análisis del rendimiento reveló diferencias sustanciales entre ambos frameworks, donde Svelte superó a React en todas las métricas evaluadas: First Contentful Paint 36.4% más rápido (0.7s vs 1.1s), bundle size 39.3% menor (95.2KB vs 156.8KB), y puntuación de rendimiento de 96/100 frente a 89/100 de React. Las pruebas de memoria demostraron que Svelte consume 15.1% menos recursos en estado base (22.5MB vs 26.5MB) con eficiencia de recuperación del 95.3% versus 79.2% de React. Se determina que Svelte ofrece rendimiento técnico superior cuantificable en aplicaciones que requieren optimización de recursos.

El desarrollo del prototipo de gestor de tareas implementó 27 requerimientos funcionales incluyendo autenticación JWT, gestión de usuarios, administración colaborativa y dashboard con métricas en tiempo real; los prototipos desplegados en Vercel conectados al backend NestJS en Render permitieron mediciones bajo condiciones de producción reales, donde Svelte demostró tiempos de respuesta 75% más rápidos en filtrado (0.1s vs 0.4s) y 41.7% superiores en adición de tareas (0.7s vs 1.2s). En consecuencia, se establece que el prototipo constituyó una plataforma técnica robusta para validar diferencias arquitectónicas mediante casos de uso empresariales reales.

La evaluación de capacidad de manejo de estados reveló que Svelte reduce el consumo de CPU entre 71-81% durante operaciones críticas, mantiene incrementos de memoria más estables (+4.3MB vs +5.3MB en React) y alcanza 89% de efectividad en tree shaking frente a 68% de React. La reactividad compilada elimina el overhead del Virtual DOM, generando código específico con 0.6% de CPU usage base versus 7.3% de React. Por tanto, se comprueba que Svelte facilita mayor rendimiento mediante su arquitectura compilada que optimiza desarrollo y ejecución en producción.

Recomendaciones

Actualmente, el análisis de rendimiento se realizó utilizando Google Lighthouse y Chrome DevTools bajo condiciones controladas, lo cual cubre aspectos básicos de evaluación comparativa, sin embargo, para lograr una optimización continua, es recomendable implementar un sistema de monitoreo de rendimiento en tiempo real que capture métricas de usuarios reales durante el uso normal de la aplicación. La integración se puede realizar utilizando herramientas modernas compatibles con React y Svelte que registren automáticamente tiempos de carga, errores del sistema y comportamiento en diferentes dispositivos y conexiones de internet; esto elevará la precisión del análisis sin afectar la experiencia del usuario y proporcionará datos más representativos del rendimiento real en condiciones de uso cotidiano.

Con base en los resultados del prototipo de gestor de tareas, se recomienda expandir esta investigación hacia el desarrollo de aplicaciones empresariales completas que incorporen funcionalidades avanzadas como notificaciones en tiempo real, colaboración multi-usuario, integración con sistemas externos y módulos de business intelligence. La implementación debe incluir versiones paralelas en múltiples frameworks conectadas al mismo backend para evaluaciones longitudinales, agregando tecnologías complementarias como Redis para caching, WebSockets para comunicación en tiempo real, y servicios de monitoreo para análisis de performance en producción; esta expansión proporcionará métricas representativas sobre escalabilidad y comportamiento bajo cargas empresariales reales con múltiples usuarios concurrentes.

Actualmente, la evaluación de capacidad de manejo de estados se enfocó en métricas básicas de CPU y memoria durante operaciones específicas, lo cual representa una base sólida para comparaciones técnicas, para mejorar el análisis y hacerlo más adaptativo a diferentes contextos empresariales, se recomienda incorporar pruebas de estrés y carga automatizadas que evalúen el comportamiento de los frameworks bajo condiciones extremas como picos de tráfico y múltiples operaciones simultáneas; se podría simular el comportamiento con 100 usuarios simultáneos ejecutando operaciones intensivas de creación, edición y eliminación de datos. Esto convierte la evaluación en una herramienta más completa y predictiva, capaz de anticipar limitaciones de escalabilidad según el volumen esperado de usuarios del proyecto.

Bibliografía

[1] Z. Mahmood, M. K. Siddiqui, «Comparative Analysis on Front-End Frameworks for Web Applications,» *International Journal of Computer Applications,* vol. 184, nº 22, pp. 1-8, 2022.

[2] R. A. García Yánez, K. M. Zurita Hidalgo, *Comparativa de frameworks para el desarrollo web en el lado del cliente basado en métricas de desempeño web vitals,* Sangolquí: Universidad de las Fuerzas Armadas ESPE, 2023.

[3] Pedro H. Marques Almeida, Werik Gonçalves de Paula, «Análise Comparativa das Características de Performance dos JavaScript Frameworks React e Vue,» Belo Horizonte, 2022.

[4] S. J. Moya, «Análisis de Frameworks Frontend para Aplicar UX/IU en el Desarrollo Web: Una Revisión Sistemática,» *Ciencia Latina Revista Científica Multidisciplinar,* vol. 7, nº 6, pp. 1-15, 2023.

[5] M. Kaluža, B. Vukelić, «Comparison of front-end frameworks for web applications development,» *Zbornik Veleučilišta u Rijeci,* vol. 6, nº 1, pp. 261-282, 2018.

[6] R. E. Hurtado, «Análisis comparativo para la evaluación de frameworks usados en el desarrollo de aplicaciones web,» *CEDAMAZ,* vol. 11, nº 2, pp. 133-141, 2021.

[7] J. A. M. Cali, *Análisis comparativo de frameworks frontend para aplicaciones móviles a través de un aplicativo web con inteligencia artificial,* Quito, 2022.

[8] F. Inc., «DOM virtual y detalles de implementación,» 2023. [En línea]. Available: https://es.legacy.reactjs.org/docs/faq-internals.html.

[9] A. M. L. Mendoza, «Análisis de factores que inciden en la selección de un lenguaje y framework de programación para desarrollo de software,» Machala, 2020.

[10] R. Marciniak, «Propuesta metodológica para la aplicación del benchmarking internacional en la evaluación de la calidad de la educación superior virtual,» *RUSC. Universities and Knowledge Society Journal,* vol. 13, nº 1, pp. 46-61, 2016.

[11] G. Developers, «Core Web Vitals: Essential Metrics for Page Experience,» 2024. [En línea]. Available: https://web.dev/articles/vitals.

[12] G. Developers, «Understanding Core Web Vitals and Google search results,» 2024. [En línea]. Available: https://developers.google.com/search/docs/appearance/core-web-vitals.

[13] P. Team, «DOM vs. Virtual DOM,» 2024. [En línea]. Available: https://platzi.com/blog/virtual-dom/.

[14] K. Team, «¿Qué es la Arquitectura de las Aplicaciones Web? Desglosando una Aplicación Web,» 2025. [En línea]. Available: https://kinsta.com/es/blog/arquitectura-aplicaciones-web/.

[15] Díaz-Sabogal, M. A.; Bermejo-Terrones, H. P.; Urquizo-Gómez, Y. V., «Influencia de las aplicaciones web en la gestión académica: Una Revisión Sistemática de la literatura,» *RISTI: Revista Ibérica de Sistemas e Tecnologias de Informação,* nº 64, pp. 527-536, 2023.

[16] Milošević, D.; Maksimović, J., «Methodology of Comparative Research in Education: Role and Significance,» *International Journal of Cognitive Research in Science, Engineering and Education (IJCRSEE),* vol. 8, nº 3, pp. 155-162, 2020.

[17] M. Borja Romero, «Análisis Comparativo de Metodologías de Desarrollo para Proyectos Unipersonales en base a el Desarrollo de una Aplicación Móvil,» 2024.

[18] V. Team, «Vite Documentation,» 2024. [En línea]. Available: https://es.vite.dev/guide/.

[19] T. Team, «TypeScript Documentation,» 2024. [En línea]. Available: https://www.typescriptlang.org/docs/handbook/intro.htm.

[20] N. Team, «NestJS Documentation,» 2024. [En línea]. Available: https://docs.nestjs.com/.

[21] Vercel, «Vercel Documentation,» 2024. [En línea]. Available: https://vercel.com/docs/getting-started-with-vercel/projects-deployments.

[22] Render, «Render Documentation,» 2024. [En línea]. Available: https://render.com/docs/web-services.

[23] Brevo, «Brevo Developer Documentation,» 2024. [En línea]. Available: https://developers.brevo.com/docs/send-a-transactional-email.

[24] Cloudinary, «Cloudinary Documentation,» 2024. [En línea]. Available: https://cloudinary.com/documentation.

[25] Prismic, «Svelte vs React 2024: Which is Better,» 26 Febrero 2023. [En línea]. Available: https://prismic.io/blog/svelte-vs-react.

[26] Maddevs, «Svelte vs. React: Which to Choose for Your Project,» 11 Octubre 2023. [En línea]. Available: https://maddevs.io/blog/svelte-vs-react/.

[27] Simform, «React vs Svelte- A Comprehensive Comparison Between Javascript Libraries,» [En línea]. Available: https://www.simform.com/blog/react-vs-svelte/.

[28] Okupter, «Svelte vs. React: A Comprehensive Comparison,» 2024. [En línea]. Available: https://www.okupter.com/blog/svelte-vs-react.

[29] Xperti, «SVELTE VS. REACT: WHICH SHOULD YOU CHOOSE FOR YOUR PROJECT?,» 30 Septiembre 2024. [En línea]. Available: https://xperti.io/blogs/svelte-vs-react/.

[30] Kinsta, «Svelte vs React: Features, Performance, and More,» 8 Agosto 2024. [En línea]. Available: https://kinsta.com/blog/svelte-vs-react/.

[31] I. C. Publications, «Svelte.js: The Most Loved Framework Today,» *IEEE Xplore Digital Library*, 2023.

[32] M. Kaluža, K. Troskot, B. Vukelić, «Research and Analysis of the Front-end Frameworks and Libraries in E-Business Development,» *ResearchGate Publications,* vol. 6, nº 1, pp. 261-282, 2019.

[33] R. Vyas, «Comparative Analysis on Front-End Frameworks for Web Applications,» *International Journal for Research in Applied Science & Engineering Technology,* vol. 10, 2 agosto 2022.

[34] C. L. Mariano, «Benchmarking JavaScript Frameworks,» *Dublin Institute of Technology Academic Publications,* 2017.

[35] B. P. Team, «Performance Testing: Types, Importance and Best Practices,» *BrowserStack Technical Documentation,* 2025.

[36] A. P. Team, «How to Do Performance Testing for Web Application,» *Abstracta Technical Publications,* 2025.

[37] G. W. P. Team, «Web Vitals: Essential Metrics for a Healthy Site,» 2024.

[38] H. H. Liu, Software Performance and Scalability: A Quantitative Approach, vol. 1st Edition, Wiley, 2009, p. 401.

[39] I. Molyneaux, The Art of Application Performance Testing: From Strategy to Tools, 2nd Edition ed., O'Reilly Media, 2015.

[40] M. Corporation, Performance Testing Guidance for Web Applications, Microsoft Press, 2007.

[41] R. K. Yin, Case Study Research and Applications: Design and Methods, 6th Edition ed., SAGE Publications, 2017, p. 352.

[42] John W. Creswell y J. David Creswell, Research Design: Qualitative, Quantitative, and Mixed Methods Approaches, 5th Edition ed., SAGE Publications, 2018, p. 304.

[43] S. o. J. Team, «State of JavaScript 2024,» Diciembre 2024. [En línea]. Available: https://2024.stateofjs.com/en-US/libraries/front-end-frameworks/.

[44] G. Soltana, M. Sabetzadeh, L. Briand, «Synthetic data generation for statistical testing,» 2017.

[45] G. C. Developers, «Introducción a Lighthouse,» Junio 2025. [En línea]. Available: https://developer.chrome.com/docs/lighthouse/overview/.

[46] G. C. Developers, «Lighthouse: Optimiza la velocidad del sitio web,» Junio 2025. [En línea]. Available: ttps://developer.chrome.com/docs/lighthouse/overview/.

[47] Tjaša Heričko, Boštjan Šumak, Sasa Brdnik, «Towards Representative Web Performance Measurements with Google Lighthouse,» de *Proceedings of the 2021 7th Student Computer Science Research Conference (StuCoSReC)*, 2021.

[48] Amantia Pano, Daniel Graziotin, Pekka Abrahamsson, «What leads developers towards the choice of a JavaScript framework?,» *CoRR*, vol. abs/1605.04303, pp. 1-16, 2016.

[49] Andreas B. Gizas, Sotiris P. Christodoulou, Theodore S. Papatheodorou, «Comparative evaluation of JavaScript frameworks,» *Proceedings of the 21st international conference companion on World Wide Web*, pp. 513-514, 2012.

[50] Kwame Chan-Jong-Chu, Tanjina Islam, Miguel Morales Exposito, Sanjay Sheombar, Christian Valladares, Olivier Philippot, Eoin Martino Grua, Ivano Malavolta, «Investigating the Correlation between Performance Scores and Energy Consumption of Mobile Web Apps,» *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pp. 190-199, 2020.

[51] Levlin, Mattias, DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue, and Svelte, 2020.

[52] G. C. Developers, «Performance monitor panel,» *Chrome for Developers*, 2024.

[53] C. Jonathan, «Análisis comparativo de los frameworks Laravel y CodeIgniter para la implementación del sistema de gestión de recursos de méritos y oposición en la universidad nacional de Chimborazo,» Riobamba, 2016.

[54] Chavez Calderón, J. X.; Zambrano Romero, W. D.; Cedeño Palma, E. A.; Zambrano Zambrano, D. M.; Cotera Ramírez, G. A., «El Frontend. Diseño web adaptativo y diseño web responsivo para el desarrollo de aplicaciones web,» *Informática y Sistemas: Revista de Tecnologías de La Informática y Las Comunicaciones*, vol. 6, nº 1, pp. 79-95.

[55] A. L. Martínez Ortiz, «Investigación y caracterización de modelos de calidad para componentes web,» 2020.

[56] V. Team, «Reactividad en profundidad,» Vue.js Documentation, 2023. [En línea]. Available: https://es.vuejs.org/v2/guide/reactivity.

[57] G. Team, «ReactJS Virtual DOM,» 2024. [En línea]. Available: https://www.geeksforgeeks.org/reactjs-virtual-dom/.

[58] D. Ghimire, «Comparative study on Python web frameworks: Flask and Django,» 2020.

[59] L. Team, «8 pasos del proceso de evaluación comparativa,» 2019. [En línea]. Available: https://www.lucidchart.com/blog/es/evaluacion-comparativa-los-ocho-pasos-del-proceso.

[60] A. Team, «SOA frente a microservicios: diferencia entre estilos arquitectónicos,» 2024. [En línea]. Available: https://aws.amazon.com/es/compare/the-difference-between-soa-microservices/.

ANEXOS

Manual de Usuario – Backend

Guía de Instalación y Configuración

El backend creado para este proyecto es un sistema de gestión de proyectos y tareas desarrollado como backend en API REST utilizando nestjs; este proporciona las instrucciones necesarias para la instalación, configuracion y uso del sistema desde una perspectiva de usuario final.

Requisitos del Sistema

Hardware Mínimo

- Procesador: Intel Core i3 o equivalente
- Memoria RAM: 4GB (recomendado 8GB)
- Espacio en disco: 2GB libres
- Conexión a internet estable

Software Base

- Sistema Operativo: Windows 10+, macOS 10.15+, o Linux Ubuntu 18+
- Node.js versión 18 o superior
- Git para control de versiones
- Editor de código (recomendado: Visual Studio Code)

Instalación de Dependencias

Instalación de Node.js

Node.js es el entorno de ejecución necesario para el sistema.

Descargar desde nodejs.org con el siguiente link recomendado de la web oficial (https://nodejs.org/es)

Seleccionar la versión LTS como se observa en la siguiente imagen (Long Term Support)

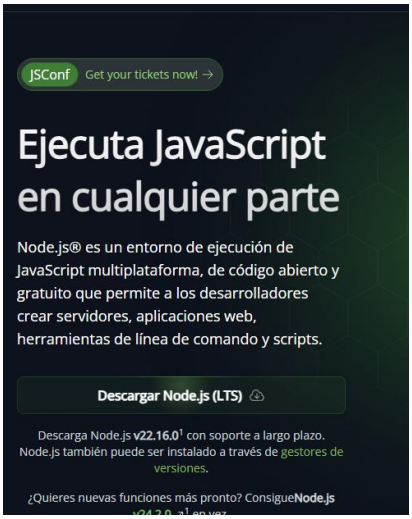


Gráfico 16. Seleccionar versión a descargar de Node

Ejecutar el instalador y seguir las instrucciones

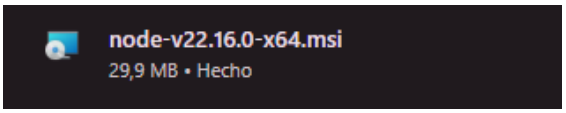


Gráfico 17. Instalador de node

Verificar instalación abriendo cmd y ejecutar:

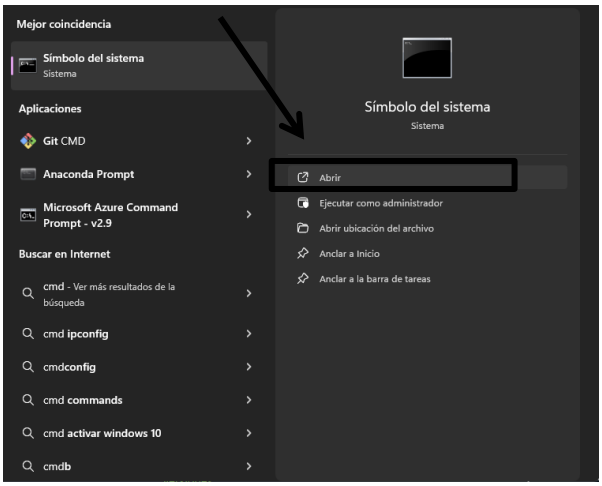


Gráfico 18. Paso para abrir el simbolo del sistema

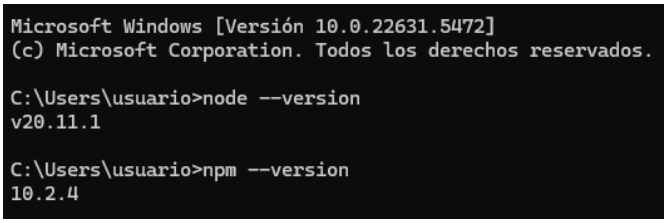


Gráfico 19. Comandos para verificar las versiones instaladas

Instalación de Git

Git es necesario para descargar el código del proyecto, ya que este se subio en un repositorio publico para el uso en proximos proyectos ya sean de analizar otros tipos de framework o mejorando el sistema del backend.

Descargar desde el siguiente link https://git-scm.com/ , una vez seleccionado la opcion de descargar se escoger la versión que dependa del sitema operativo de la computadora.

Para revisar el sistema operativo de su dispositvo se puede ver siguiendo lo siguientes pasos:
Click derecho sobre este equipo y luego en la opcion de propiedades :

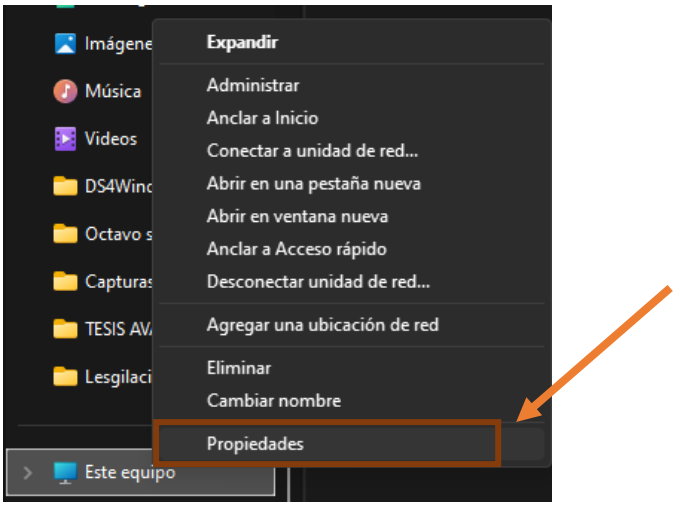


Gráfico 20. Paso 1 para verificación del sistema operativo

Una vez en la pantalla de propiedades, en la parte de especificaciones del dispositivo se lograra observar el sistema operativo de su dispositivo.

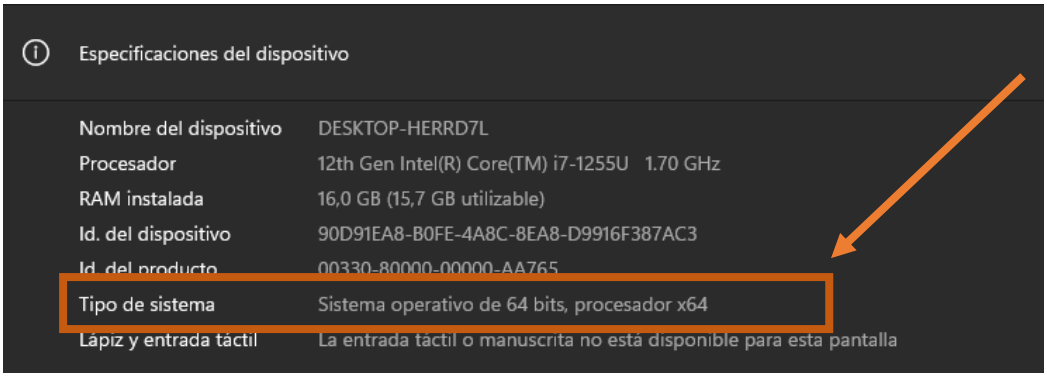


Gráfico 21. Paso 2 para verificación del sistema operativo

Ahora seleccionamos la version en base a nuestro sistema y le damos click el cual nos descargara el instalador:

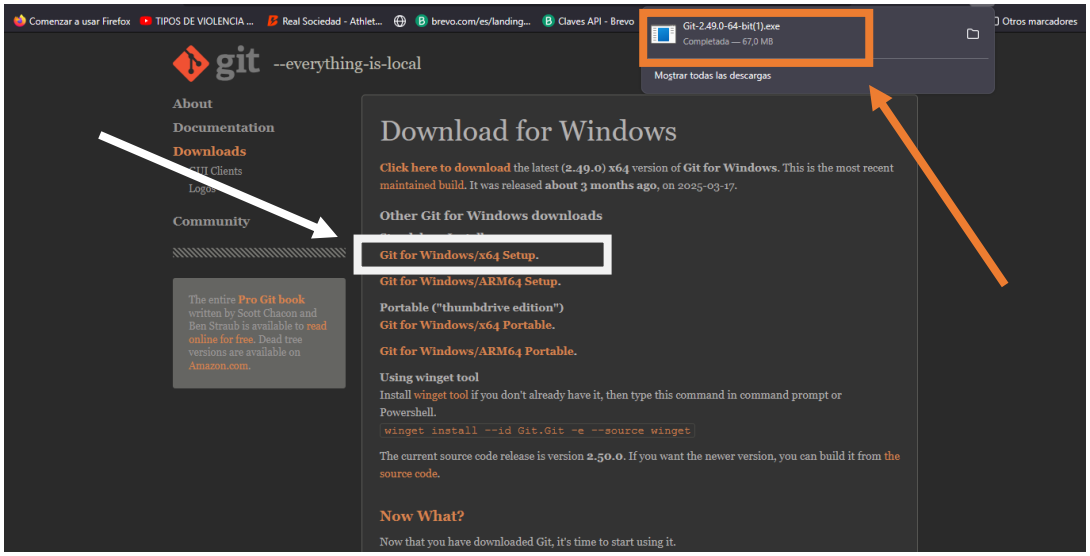


Gráfico 22. Descargar version de acuerdo al sistema

Ejecutar instalador con configuración por defecto, lo cual se logra dandole click al instalador y siguiente a todos los pasos

Gráfico 23. Ejecutar instalador

Instalación de Visual Studio Code

Visual Studio Code es el editor recomendado para trabajar con el proyecto, debido a su maxima comodidad para el desarrollo.

Descargar desde https://code.visualstudio.com/

Instalar extensiones recomendadas:

Prisma (para base de datos)

TypeScript Importer

Primero le damos click en las extensiones a lo cual nos llevara a la siguiente ventana para poder escribir las recomendaciones para poder trabajar en el proyecto del backend, una vez encontrado la extensión le damos a descargar y aceptar todo lo que necesite visual studio code.

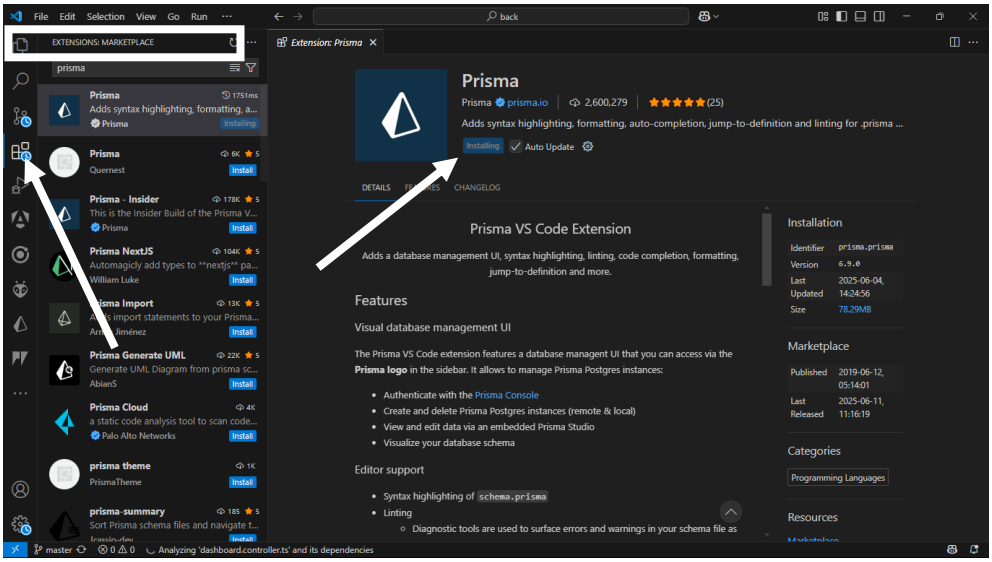


Gráfico 24. Instalación de extensiones necesarias en visual studio code

Configuración del Entorno

Descarga del Proyecto

Abrir terminal , el mismo cmd que se abrio anteriormente para verificar las versiones instaladas

Navegar al directorio donde desea instalar el proyecto

Clonar el repositorio, con los siguientes comandos:

git clone https://github.com/AlexX2727/backend

cd backend

```
PS C:\Users\usuario\Desktop\Prueba Clonar> git clone https://github.com/AlexX2727/backend
Cloning into 'backend'...
remote: Enumerating objects: 266, done.
remote: Counting objects: 100% (266/266), done.
remote: Compressing objects: 100% (156/156), done.
Rremote: Total 266 (delta 134), reused 229 (delta 100), pack-reused 0 (from 0)
Receiving objects: 100% (266/266), 183.70 KiB | 790.00 KiB/s, done.
Resolving deltas: 100% (134/134), done.
PS C:\Users\usuario\Desktop\Prueba Clonar> cd .\backend\
PS C:\Users\usuario\Desktop\Prueba Clonar\backend>
```

Gráfico 25. Comandos para clonar respositorio

Una vez clonado el repositorio, se hace el respectivo instalación de dependecias del proyecto, lo cual tardara un tiempo largo o pequeño dependiendo su conexión a internet.

Instalar todas las dependencias

npm install

Verificar que se instalaron correctamente

npm list --depth=0

Manual Tecnico

Documentación para Desarrolladores

TaskMaster es una API REST desarrollada con NestJS que proporciona un sistema completo de gestión de proyectos y tareas. Utiliza PostgreSQL como base de datos, Prisma como ORM, y JWT para autenticación.

Arquitectura del Sistema

Stack Tecnológico

Backend Framework:

- NestJS 10.x (Node.js framework)
- TypeScript 5.x
- Express.js

Base de Datos:

- PostgreSQL
- Prisma ORM

Autenticación:

- JWT (JSON Web Tokens)

Servicios Externos:

Cloudinary (almacenamiento de archivos)

Brevo (envío de emails)

Configuración de Desarrollo

Variables de Entorno Requeridas

En el archivo clonado desde git, se crea una archivo en la raiz del proyecto llamada env en la cual se ubicaran los entornos o apikey para que de esta manera este totalmente funcionable.

Links para la creacion de cuenta de supabase y clodunary, para obtener las respectivas apikey y configuracion de credenciales:

https://supabase.com/

https://console.cloudinary.com/

Gráfico 26. Insertar credenciales

Configuración de Base de Datos

Comandos para poder migrar la base de datos creada en prisma en la nube.

Generar cliente Prisma

npx prisma generate

Crear migración

npx prisma migrate dev --name descripcion_cambio

Aplicar migraciones

npx prisma migrate deploy

Resetear base de datos (desarrollo)

npx prisma migrate reset

Estructura del Proyecto

La siguiente estrucutra demuestra de una manera mas tecnica y detallada de como esta divido todas las carpetas y archivos.

```
src/
├── app.module.ts          # Módulo principal
├── main.ts                # Punto de entrada
├── auth/                  # Autenticación y autorización
│   ├── auth.controller.ts
│   ├── auth.service.ts
│   ├── auth.guard.ts
│   ├── auth.module.ts
│   └── dto/
├── users/                 # Gestión de usuarios
│   ├── users.controller.ts
│   ├── users.service.ts
│   ├── users.module.ts
│   ├── email.service.ts
│   └── dto/
├── projects/              # Gestión de proyectos
├── tasks/                 # Gestión de tareas
├── project-members/       # Miembros de proyectos
├── comments/              # Sistema de comentarios
├── attachments/           # Archivos adjuntos
├── dashboard/             # Dashboard y métricas
├── config/                # Configuraciones
```

```
│   ├── cloudinary.config.ts
│   ├── upload.controller.ts
│   └── upload.module.ts
├── prisma/              # Configuración Prisma
│   ├── prisma.service.ts
│   └── prisma.module.ts
```

Autenticación

JWT Token Structure

```
{
  sub: number,                  // User ID
  userWithoutPassword: {
    id: number,
    email: string,
    role_id: number
  },
  exp: number                   // 24 hours
}
```

Recuperación de Contraseña

POST /users/forgot-password → Genera código de 6 dígitos

Email enviado vía Brevo con código todo esto a travez de una tabla creada la cual almacena el codigo y cuando es utilizado cambia el true por false (expira en 15 min)

POST /users/reset-password → Valida código y actualiza contraseña

API Endpoints

Todos los endopoints para probarlo desde cualquier herramienta que permita lanzar solicitudes, como insonmia o postman

Autenticación

POST /auth/login

POST /auth/signup

GET /auth/users

Proyectos

GET /projects

POST /projects

GET /projects/:id

PATCH /projects/:id

DELETE /projects/:id

Tareas

GET /tasks

GET /tasks/filter?projectId=1&status=Todo

POST /tasks

PATCH /tasks/:id

DELETE /tasks/:id

Dashboard

GET /dashboard # Todas las métricas

GET /dashboard/active-projects # Proyectos activos

GET /dashboard/pending-tasks # Tareas pendientes

GET /dashboard/project-progress # Progreso por proyecto

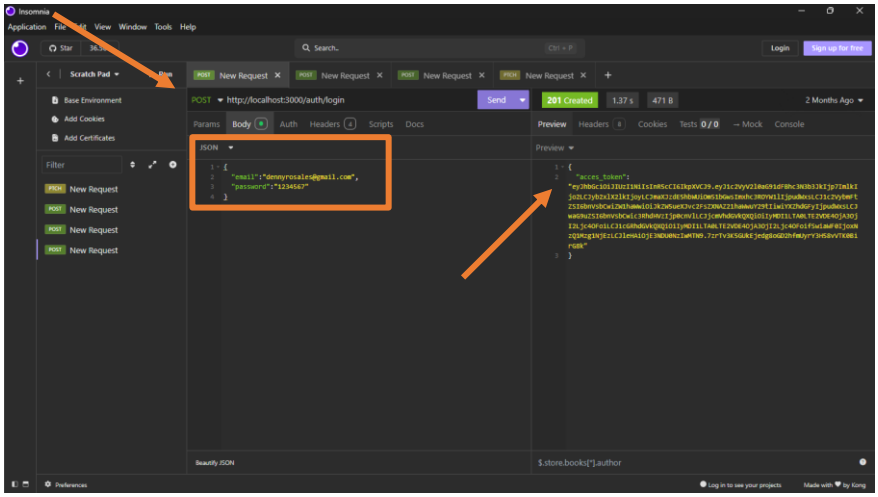


Gráfico 27. Prueba desde insonmia

Como se logra observar en la imagen se lanza una solicitud de post para poder obtener el token y de esta manera poder utilizar todo los endpoints, se lanza un json el cual se da los datos que estos son email y password , los cuales como los datos son correctamente me devuelve el acces_token, aunque dependiendo del endpoint utilizado se cambia el metodo ya sea este get, post, pacth o delete.