Diseño de Arquitectura del Sistema - TaskManager

**Autor: Carlos Alberto Bolaños Gamarra**

**Rodrigo Holgado Quispe**

**Fabricio Huillca Perez**

**Raul Ppacsi Chillihuani**

**Versión:** 2.0

# **1. Introducción**

TaskManager nace con la idea de simplificar el día a día de los equipos de trabajo, reduciendo el estrés y mejorando la claridad en los procesos. En un mundo tan dinámico y lleno de desafíos, contar con un sistema que ayude a gestionar tareas de forma más eficiente es clave. Es por eso que su diseño se basa en una arquitectura simple y directa, sin complicaciones innecesarias, pensada para ser fácil de usar y escalable a medida que el equipo crece. La propuesta incluye tres capas (Presentación, Negocio y Datos), lo que garantiza un flujo de trabajo fluido y bien organizado sin sobrecargar a los usuarios ni a los desarrolladores.

La arquitectura de TaskManager no solo busca hacer más eficientes las operaciones diarias, sino también garantizar que el sistema crezca de manera sostenible y se adapte a las necesidades del equipo. Está pensada para ser flexible, lo que significa que puede escalar con el tiempo y ajustarse a los cambios que puedan surgir en el futuro. Usar herramientas robustas como React.js para el frontend y PostgreSQL para la gestión de datos asegura que el sistema sea rápidAdemás, la seguridad de los datos es un aspecto fundamental. TaskManager se protege mediante autenticación segura, roles bien definidos (RBAC), y el cifrado de contraseñas para mantener la información protegida. Pero, más allá de la tecnología, lo más importante es que el sistema tiene como objetivo hacer la vida más fácil tanto a los usuarios como a los administradores. De esta manera, las tareas cotidianas se hacen más fáciles y la experiencia del usuario final siempre es la prioridad.

Este enfoque de arquitectura está alineado con las mejores prácticas de desarrollo de software, buscando un balance entre simplicidad, seguridad y escalabilidad. El objetivo es crear un sistema que no solo cumpla con los requisitos técnicos, sino que también ofrezca una experiencia intuitiva y accesible para los usuarios.

# **2. Visón General de la Arquitectura**

## **2.1. Estilo Arquitectónico**

El sistema adopta una arquitectura en capas, siguiendo el patrón cliente-servidor. Esto significa que cada parte del sistema tiene un rol bien definido y se comunica con las demás de forma ordenada, lo que facilita la comprensión, el mantenimiento y la evolución futura.

Se han definido tres capas principales:

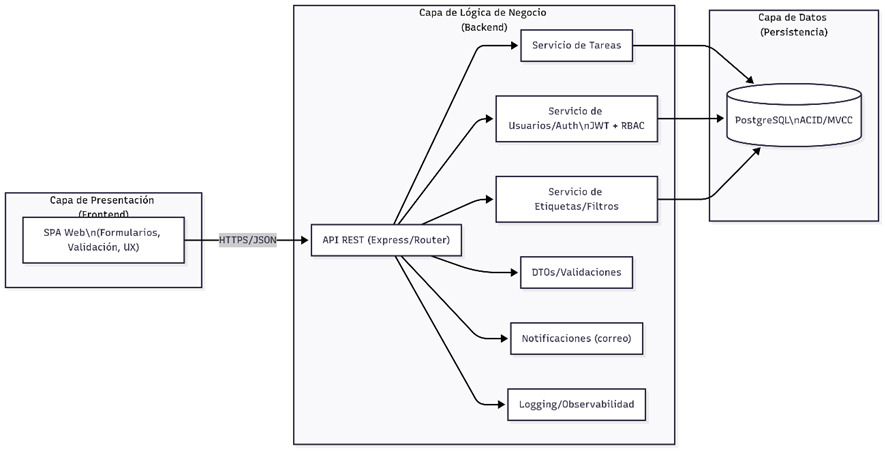
Presentación (Frontend): Es la cara visible para el usuario. Se implementa como una aplicación web de una sola página (SPA), con formularios fáciles de usar, validaciones claras y mensajes empáticos. Esta capa se conecta con el backend a través de servicios REST que envían y reciben datos en formato JSON.

Negocio (Backend): Aquí viven las reglas más importantes del sistema. Se trata de una API REST encargada de procesar las solicitudes, validar información y aplicar las reglas del dominio (tareas, usuarios, etiquetas). Además, se asegura la seguridad mediante autenticación JWT con control de roles (RBAC). También gestiona integraciones internas, como el envío de correos y notificaciones, pero sin añadir complejidades innecesarias en esta primera versión.

Datos (Persistencia): Es el espacio donde se almacena la información. Se ha elegido PostgreSQL como base de datos por su robustez, ya que garantiza consistencia y permite trabajar con múltiples usuarios al mismo tiempo sin que se generen conflictos, gracias a sus mecanismos ACID y MVCC.

d.

## **2.2. Diagrama General de Arquitectura:**



# **3. Descripción de Componentes Principales**

## **3.1. Frontend (Presentación)**

El Frontend es la parte del sistema con la que los usuarios interactúan directamente. En TaskManager, hemos diseñado una interfaz que busca ser clara, intuitiva y fácil de usar, para que cada miembro del equipo se sienta cómodo utilizando la herramienta sin complicaciones.

* Framework: Utilizamos React.js, un framework de JavaScript muy popular y eficiente para la construcción de interfaces de usuario interactivas. Gracias a su enfoque basado en componentes, React permite que cada parte de la interfaz se gestione de manera modular, lo que facilita tanto su desarrollo como su mantenimiento a largo plazo. Esto asegura una experiencia de usuario ágil, con tiempos de carga rápidos y actualizaciones dinámicas en tiempo real.
* Estilo: Para mantener una interfaz limpia y coherente en todo el sistema, elegimos Tailwind CSS. Este framework de diseño utilitario nos permite aplicar estilos de forma rápida y consistente, haciendo que el diseño sea no solo visualmente atractivo, sino también funcional. Con Tailwind, se puede personalizar la apariencia de manera sencilla, asegurando que la plataforma sea accesible y cómoda en cualquier dispositivo.
* Responsabilidades:
* Arquitectura en capas + REST: El Frontend actúa como la capa de presentación del sistema, encargada de mostrar los datos a los usuarios y capturar sus interacciones. Este se conecta con el Backend a través de API REST, utilizando JSON como formato de intercambio de datos, lo que favorece la independencia entre ambas capas y facilita la evolución del sistema sin depender estrictamente de cambios en el backend.
* Interfaz accesible y validaciones: La interfaz incluye formularios claros y sencillos de completar, con validaciones en tiempo real para garantizar que los usuarios ingresen la información correctamente desde el inicio. Cada campo tiene una validación visual para mostrar los errores o la confirmación de éxito, asegurando que el proceso sea lo más fluido posible.
* Gestión de estado: El Frontend también se encarga de gestionar el estado de la aplicación, es decir, mantiene la información actualizada en tiempo real para que el usuario siempre vea los datos más recientes sin tener que recargar la página. Esto permite una experiencia dinámica y eficiente, eliminando tiempos de espera innecesarios.

Con este enfoque, el objetivo es ofrecer una plataforma que no solo sea útil, sino también agradable de usar, reduciendo al máximo la complejidad técnica para que los usuarios puedan centrarse en lo que realmente importa: realizar su trabajo de forma efectiva.

## **3.2. Backend (Lógica de negocio)**

El backend constituye el **núcleo del sistema**, donde se definen las reglas y operaciones que garantizan que la aplicación funcione de manera coherente, segura y escalable. Para ello se emplea **Node.js con Express**, un framework ligero y flexible que facilita el manejo de peticiones HTTP y la implementación de APIs REST.

#### **Responsabilidades principales:**

* Gestionar la autenticación de usuarios y el control de accesos.
* Administrar las operaciones de negocio (creación, actualización y seguimiento de tareas).
* Implementar reglas de validación, flujos de autorización y envío de notificaciones.
* Integrar servicios externos de manera eficiente sin generar sobrecarga.

### **3.2.1. Módulos del Backen**

* **authController.js**: Responsable del login y registro de usuarios, incluyendo encriptación de contraseñas y emisión de tokens.
* **taskController.js**: Gestiona las operaciones CRUD (crear, leer, actualizar y eliminar) sobre las tareas.
* **listController.js**: Administra las listas de tareas creadas por los usuarios, permitiendo organizarlas por categorías.
* **middleware/auth.js**: Valida los tokens JWT y asegura que solo usuarios autorizados accedan a recursos protegidos.
* **services/emailService.js**: Gestiona el envío de notificaciones, como recordatorios o avisos de vencimiento de tareas.

## **3.3. Base de Datos (Persistencia)**

La Base de Datos es el núcleo donde se almacenan todos los datos importantes del sistema, y es esencial para garantizar que la información esté organizada de forma eficiente y accesible. En TaskManager, hemos optado por una arquitectura de base de datos robusta que asegura tanto la consistencia como el rendimiento a lo largo del ciclo de vida del sistema.

* Sistema Gestor: Para manejar la persistencia de datos, hemos elegido PostgreSQL, un sistema de gestión de bases de datos relacional conocido por su fiabilidad y capacidad para manejar grandes volúmenes de datos de manera eficiente. PostgreSQL es ideal para sistemas que requieren transacciones seguras y consistentes, ya que ofrece características avanzadas como ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad) y MVCC (Control de Concurrencia Multiversión), que aseguran que los datos no se pierdan, incluso en situaciones de alta concurrencia.
* ORM (Object-Relational Mapping): Para facilitar la interacción con la base de datos desde el backend, utilizamos un ORM (Sequelize o Prisma). Estos ORM permiten trabajar con la base de datos utilizando objetos de JavaScript en lugar de escribir consultas SQL manualmente. Esto no solo hace que el código sea más limpio y fácil de entender, sino que también acelera el desarrollo y mejora la mantenibilidad del sistema. Al utilizar un ORM, podemos realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) de manera eficiente y con menos riesgo de errores.
* Esquema de Datos: El esquema de la base de datos está diseñado para reflejar las entidades principales del sistema y sus relaciones. A continuación, se describen las principales tablas que componen la base de datos:
* Tabla users: Esta tabla almacena la información de los usuarios que interactúan con el sistema. Cada registro contiene los siguientes campos:
* id: Identificador único del usuario.
* nombre: Nombre del usuario.
* correo: Correo electrónico del usuario.
* contraseña: Contraseña encriptada del usuario para su autenticación.
* Tabla tasks: Aquí se almacenan las tareas de los usuarios, representando cada tarea que un usuario debe completar. Los campos principales son:
* id: Identificador único de la tarea.
* título: Título de la tarea.
* descripción: Descripción detallada de la tarea.
* fecha\_vencimiento: Fecha límite para completar la tarea.
* estado: Estado de la tarea (por ejemplo, "pendiente", "completada").
* user\_id: Relación con el usuario que está encargado de la tarea.
* list\_id: Relación con la lista de tareas a la que pertenece la tarea.
* Tabla lists: Esta tabla organiza las tareas en diferentes listas, permitiendo que los usuarios agrupen sus tareas por categorías. Los campos principales son:
* id: Identificador único de la lista.
* nombre: Nombre de la lista de tareas (por ejemplo, "Trabajo", "Personal").
* user\_id: Relación con el usuario que posee la lista.

La estructura modular de la base de datos permite gestionar de manera eficiente tanto los usuarios como las tareas, manteniendo las relaciones entre ellos claras y bien definidas. Esta organización asegura que la información se almacene de forma consistente y se pueda acceder rápidamente para realizar cualquier operación.

# **4. Integraciones Externas (Opcionales)**

El sistema contempla una serie de integraciones complementarias que, si bien no son esenciales para la primera versión, aportan valor agregado y mejoran la experiencia de los usuarios, así como la gestión técnica del proyecto:

* Servicios de Notificación: Se plantea integrar herramientas como EmailJS, SendGrid o Nodemailer, que permitan enviar correos automáticos a los usuarios para recordarles tareas próximas a vencer o notificarles cuando una actividad ya está atrasada. Esto fomenta la productividad y evita olvidos.
* Autenticación Social: En versiones futuras, se podría habilitar el inicio de sesión mediante OAuth2 con Google, lo que facilitará a los usuarios registrarse o ingresar al sistema sin necesidad de crear una nueva cuenta, reduciendo fricción y mejorando la accesibilidad.
* Monitorización y Seguimiento: Para garantizar un funcionamiento estable, se recomienda la integración de herramientas como LogRocket o Sentry, que permiten registrar errores, caídas o comportamientos inesperados en tiempo real. Con ello, el equipo de desarrollo puede detectar y resolver problemas de forma rápida y proactiva.

# **5. Seguridad**

La seguridad es un pilar esencial del sistema, ya que busca proteger la información sensible de los usuarios y garantizar la confianza en la plataforma. Para ello, las contraseñas se almacenarán de manera segura utilizando algoritmos de encriptación como bcrypt, evitando que datos críticos queden expuestos. La autenticación se gestionará con tokens JWT, los cuales contarán con un tiempo de expiración para prevenir accesos no autorizados prolongados. Asimismo, todas las entradas de los usuarios serán validadas y sanitizadas, reduciendo riesgos como inyecciones SQL o XSS. Finalmente, todo el tráfico entre cliente y servidor se realizará mediante HTTPS, asegurando la confidencialidad e integridad de la comunicación.

# **6. Escalabilidad y Despliegue**

Posibles problemas que pueden surgir durante el desarrollo.

El sistema ha sido diseñado pensando en una evolución progresiva y sostenible, lo que permitirá adaptarse a un mayor número de usuarios y cargas de trabajo sin comprometer el rendimiento. El frontend podrá desplegarse en plataformas modernas como Vercel o Netlify, que ofrecen rapidez y flexibilidad. El backend, en cambio, podrá ejecutarse en servicios como Render, Railway o Heroku, que simplifican la gestión de infraestructura y permiten un escalado horizontal cuando sea necesario. La base de datos estará alojada en soluciones confiables como Supabase o ElephantSQL, lo que garantiza disponibilidad y respaldo continuo. En caso de un crecimiento mayor, la arquitectura permitirá separar la base de datos en instancias dedicadas y añadir más servidores de aplicación, asegurando que el sistema se mantenga estable, eficiente y preparado para el futuro.

# **7. Conclusiones**

La arquitectura planteada busca alcanzar un equilibrio entre simplicidad, seguridad y escalabilidad, respondiendo a las necesidades reales de los usuarios y del equipo de desarrollo. Al estructurarse en capas bien definidas, permite un mantenimiento ordenado y la incorporación progresiva de nuevas funciones sin generar complejidad innecesaria. Además, prioriza la experiencia del usuario mediante interfaces claras y procesos ágiles, respaldados por un backend robusto y una base de datos confiable. Con ello, se asegura no solo un desempeño eficiente en el presente, sino también una plataforma preparada para evolucionar de manera sostenible en el futuro.

REFERENCIAS

[1] FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures* [en línea]. Irvine: University of California, 2000. Disponible en: [https://ics.uci.edu/~fielding/pubs/dissertation/fielding\_dissertation.pdf](https://ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf?utm_source=chatgpt.com) (Consulta: 08 sep. 2025). [ics.uci.edu](https://ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf?utm_source=chatgpt.com)

[2] OWASP. *Application Security Verification Standard (ASVS) v4.0.3* [en línea]. 2020. Disponible en: [https://raw.githubusercontent.com/OWASP/ASVS/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf](https://raw.githubusercontent.com/OWASP/ASVS/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf?utm_source=chatgpt.com) (Consulta: 08 sep. 2025). [GitHub](https://raw.githubusercontent.com/OWASP/ASVS/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf?utm_source=chatgpt.com)

[3] POSTGRESQL GLOBAL DEVELOPMENT GROUP. *MVCC and Transaction Isolation* [en línea]. 2025. Disponible en: [https://www.postgresql.org/docs/current/mvcc-intro.html](https://www.postgresql.org/docs/current/mvcc-intro.html?utm_source=chatgpt.com) (Consulta: 08 sep. 2025). [PostgreSQL](https://www.postgresql.org/docs/current/mvcc-intro.html?utm_source=chatgpt.com)

[4] W3C. *Web Content Accessibility Guidelines (WCAG) 2.2* [en línea]. 12 dic. 2024. Disponible en: [https://www.w3.org/TR/WCAG22/](https://www.w3.org/TR/WCAG22/?utm_source=chatgpt.com) (Consulta: 08 sep. 2025). [W3C](https://www.w3.org/TR/WCAG22/?utm_source=chatgpt.com)

[5] IDEO.org. *The Field Guide to Human-Centered Design* [en línea]. 2015. Disponible en: https://openlab.citytech.cuny.edu/.../Field-Guide-to-Human-Centered-Design\_IDEO\_lo.pdf (Consulta: 08 sep. 2025). [openlab.citytech.cuny.edu](https://openlab.citytech.cuny.edu/bassjrcomd3701spr2021ol75/files/2021/02/Field-Guide-to-Human-Centered-Design_IDEO_lo.pdf?utm_source=chatgpt.com)

[6] COSTANZA-CHOCK, S. *Design Justice: Community-Led Practices to Build the Worlds We Need* [en línea]. MIT Press, 2020. Edición de acceso abierto. Disponible en: [https://direct.mit.edu/books/oa-monograph/4605/Design-JusticeCommunity-Led-Practices-to-Build-the](https://direct.mit.edu/books/oa-monograph/4605/Design-JusticeCommunity-Led-Practices-to-Build-the?utm_source=chatgpt.com) (Consulta: 08 sep. 2025).

[7] FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures* [en línea]. Irvine: UCI, 2000. Disponible en: ics.uci.edu/~fielding/pubs/dissertation/fielding\_dissertation.pdf (consulta: 08 sep. 2025). [ics.uci.edu](https://ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf?utm_source=chatgpt.com) [8] OWASP. *Application Security Verification Standard v4.0.3* [en línea]. 2020. Disponible en: raw.githubusercontent.com/OWASP/ASVS/.../ASVS%204.0.3-en.pdf (consulta: 08 sep. 2025). [GitHub](https://raw.githubusercontent.com/OWASP/ASVS/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf?utm_source=chatgpt.com)[owasp.org](https://owasp.org/www-project-application-security-verification-standard/?utm_source=chatgpt.com) [9] POSTGRESQL GLOBAL DEVELOPMENT GROUP. *Multiversion Concurrency Control — Introduction* [en línea]. Documentación oficial (versión actual). Disponible en: postgresql.org/docs/current/mvcc-intro.html (consulta: 08 sep. 2025).

[8] FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures* [en línea]. Irvine: University of California, 2000. Disponible en:<https://ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf> (consulta: 08 sep. 2025).

[9] POSTGRESQL GLOBAL DEVELOPMENT GROUP. *MVCC and Transaction Isolation* [en línea]. 2025. Disponible en:<https://www.postgresql.org/docs/current/mvcc-intro.html> (consulta: 08 sep. 2025).

[10] OWASP. *Application Security Verification Standard (ASVS) v4.0.3* [en línea]. 2020. Disponible en:<https://raw.githubusercontent.com/OWASP/ASVS/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf> (consulta: 08 sep. 2025).