04/07/2025

Sistema de Gestión de Tareas en Tiempo Real

**Resumen**

El presente informe documenta el desarrollo del Sistema de Gestión de Tareas en Tiempo Real, una aplicación web diseñada para crear, visualizar, editar y eliminar tareas, integrando actualizaciones en tiempo real mediante WebSocket. El objetivo fue ofrecer una herramienta eficiente y funcional que permita manejar tareas entre múltiples clientes conectados al sistema.

Este sistema fue desarrollado en una arquitectura cliente-servidor utilizando tecnologías tanto en frontend como en backend. Las tareas se almacenan en una base de datos MongoDB Atlas, y el sistema está desplegado en Render.

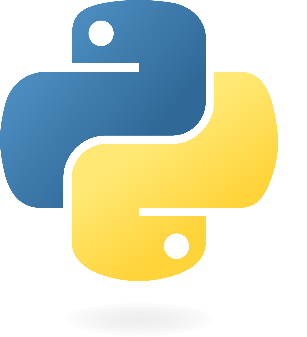
Tecnologías utilizadas:

* Frontend: HTML, CSS, JavaScript, Socket.IO-client
* Backend: Node.js, Express, Mongoose, Socket.IO
* Base de datos: MongoDB Atlas
* Despliegue: Render (Web Service)
* Repositorio GitHub:

<https://github.com/VictorRCortez/sistema-de-tareas-backend>

* URL del sistema desplegado:

<https://sistema-gestion-tareas-tpse.onrender.com/tareas/index>



**Índice**

[**Introducción** 3](#_Toc202349261)

[**Marco Teórico** 4](#_Toc202349262)

[**Estructura General del Proyecto** 5](#_Toc202349263)

[**Pruebas y Validación** 7](#_Toc202349264)

[**Despliegue** 7](#_Toc202349265)

[**Conclusión** 8](#_Toc202349266)

[**Anexos** 8](#_Toc202349267)

# **Introducción**

El proyecto nace de la necesidad de contar con una herramienta colaborativa para la gestión de tareas con actualización en tiempo real, especialmente útil en equipos de trabajo. La solución se desarrolló utilizando el stack MERN (sin React) y emplea WebSocket para sincronización de tareas entre usuarios.

**El sistema permite a los usuarios:**

* Crear nuevas tareas
* Editar tareas existentes
* Eliminar tareas
* Filtrar tareas por estado
* Recibir actualizaciones instantáneas gracias a WebSocket

# **Marco Teórico**

**Arquitectura Cliente-Servidor:**

El sistema responde a una arquitectura cliente-servidor. El cliente solicita operaciones a través de fetch y se comunica en tiempo real por WebSocket.

**Modelo MVC:**

Se aplicó el patrón MVC:

* Model: Definido con Mongoose (task.model.js).
* Controller: Lógica CRUD (tasks.controller.js).
* Routes: API REST (task.routes.js).

**WebSockets (socket.io):**

Para reflejar los cambios en tiempo real, se utilizó socket.io en el backend y frontend. Se emiten eventos en las acciones de crear, editar o eliminar tareas.

**MongoDB y Mongoose:**

MongoDB Atlas como base NoSQL, gestionada desde Mongoose. La URI de conexión está protegida mediante variables de entorno (.env).

# **Estructura General del Proyecto**

/SISTEMA-DE-TAREAS-BACKEND-MAIN

├── backend/

│ ├── controllers/ # Lógica de negocio

│ ├── middlewares/ # Validación de tareas

│ ├── models/ # Esquema Mongoose

│ ├── routes/ # Rutas de API REST

│ ├── sockets/ # Configuración de WebSocket

│ ├── app.js # Configuración principal del servidor

│ └── database.js # Conexión con MongoDB

├── frontend/

│ ├── public/ # Lógica JS del cliente

│ └── view/ # index.html

├── .env # Variables de entorno

└── README.md # Documentación general

**Backend**

API REST:

* GET /api/tareas – Listar todas las tareas
* POST /api/tareas – Crear una nueva tarea
* PUT /api/tareas/:id – Actualizar una tareaDELETE /api/tareas/:id – Eliminar tarea

WebSocket:

Conectado con socket.io y emitido desde el middleware.

Eventos:

* task-created
* ask-updated
* ask-deleted

**Fronted**

Index.html:

El archivo principal de la interfaz del usuario. Contiene un formulario para el registro de tareas, un conjunto de botones que permiten filtrar las tareas según su estado (pendiente, en progreso o completada.

Main.js:

Comunicación con backend (fetch para API y socket.on para actualizaciones).

Funciones principales:

* renderizarTareas()
* editarTarea()
* eliminarTarea()
* filtrarTareas()

# **Pruebas y Validación**

**Pruebas Manuales**

* Creación de tarea exitoso.
* Edición y validación de estados: se impide cambiar de "en progreso" a "pendiente".
* Eliminación de tareas.

**Validación de Datos**

* Manejo de errores en el cliente mediante alertas.

**Comunicación WebSocket**

* Actualizaciones reflejadas en tiempo real en múltiples ventanas/usuarios conectados.
* Eventos sincronizados con acciones del backend.

# **Despliegue**

**Backend y WebSocket:**

Desplegado en Render (<https://sistema-gestion-tareas-tpse.onrender.com/tareas/index>)

como Web Service, con soporte para WebSockets.

**Base de Datos (MongoDB Atlas):**

Configurada con IP Whitelist 34.213.214.55/32 (IP de render) y variables .env para mantener segura la URI de conexión.

**Frontend:**  
Servido desde Express accediendo a view/index.hmtl.

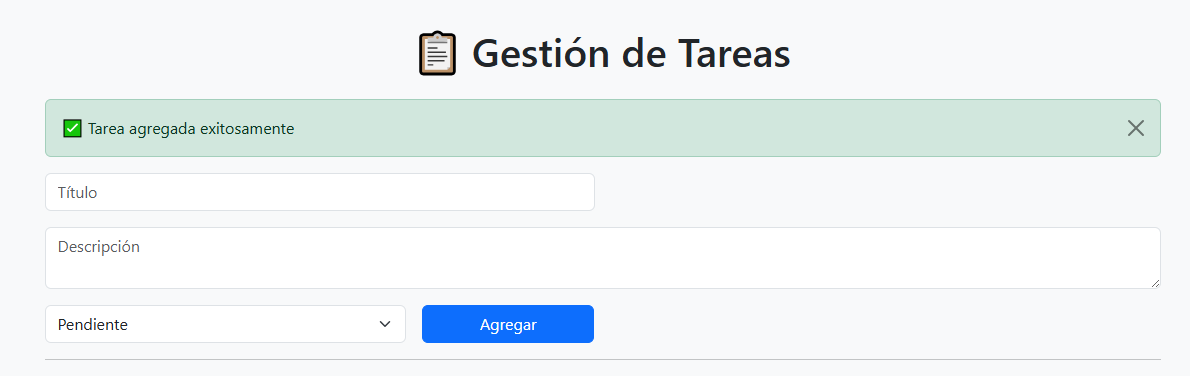
# **Conclusión**

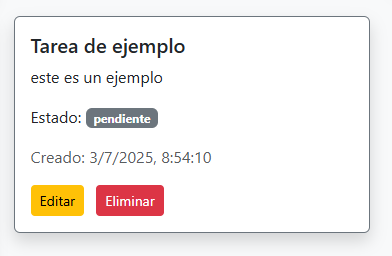
* El sistema cumple con su propósito como herramienta de gestión de tareas en tiempo real.
* Las operaciones CRUD funcionan correctamente y mediante WebSocket.
* El frontend es sencillo, funcional y usable.
* La separación en capas y carpetas permite fácil mantenimiento.
* Se implementaron validación y modularización.
* La conexión con MongoDB estable con validaciones.

# **Anexos**

**Agregar Tarea:**

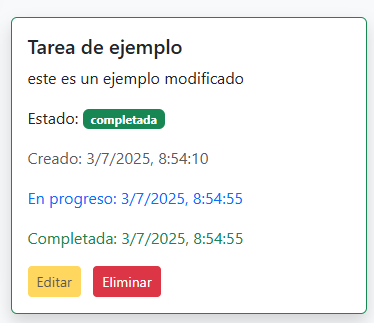




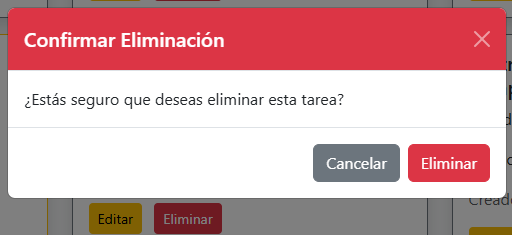


**Modificar Tarea:**

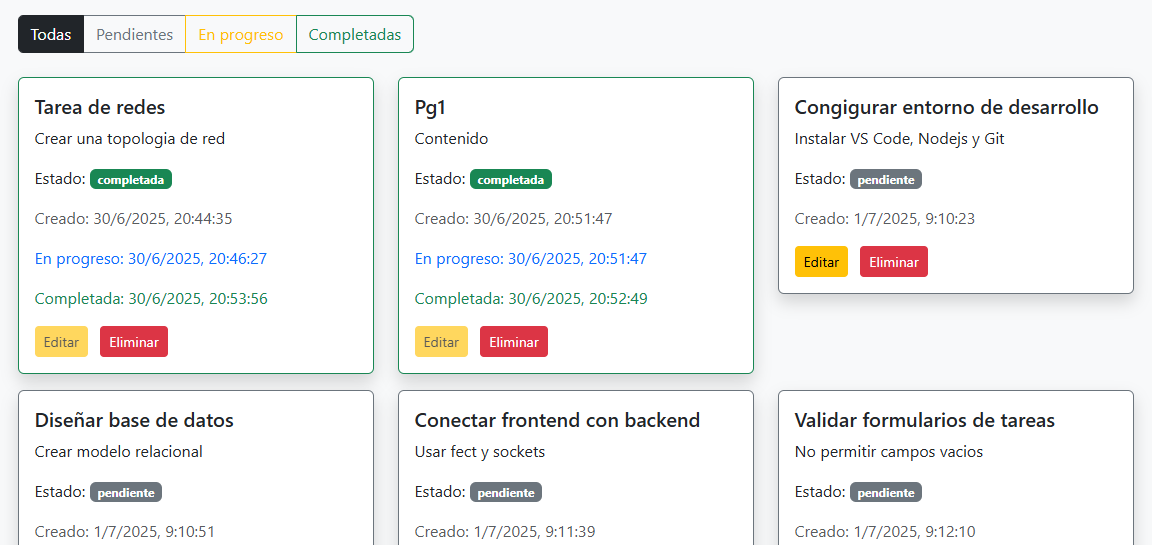


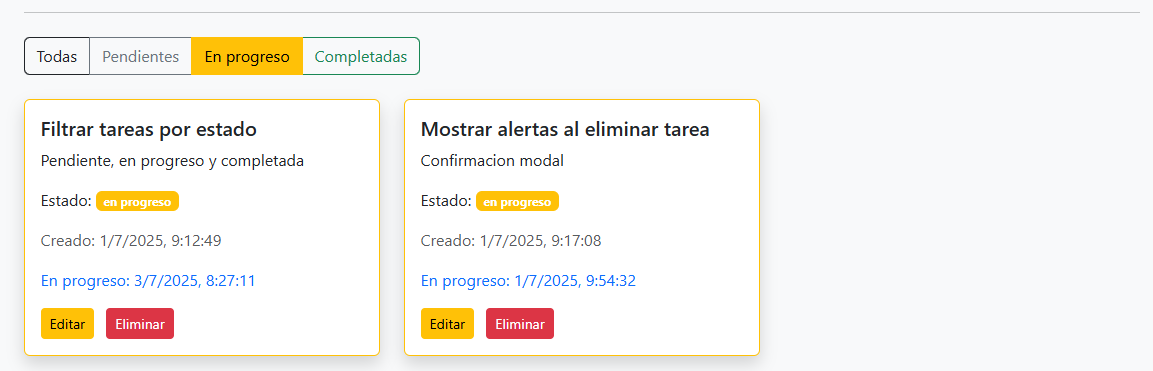
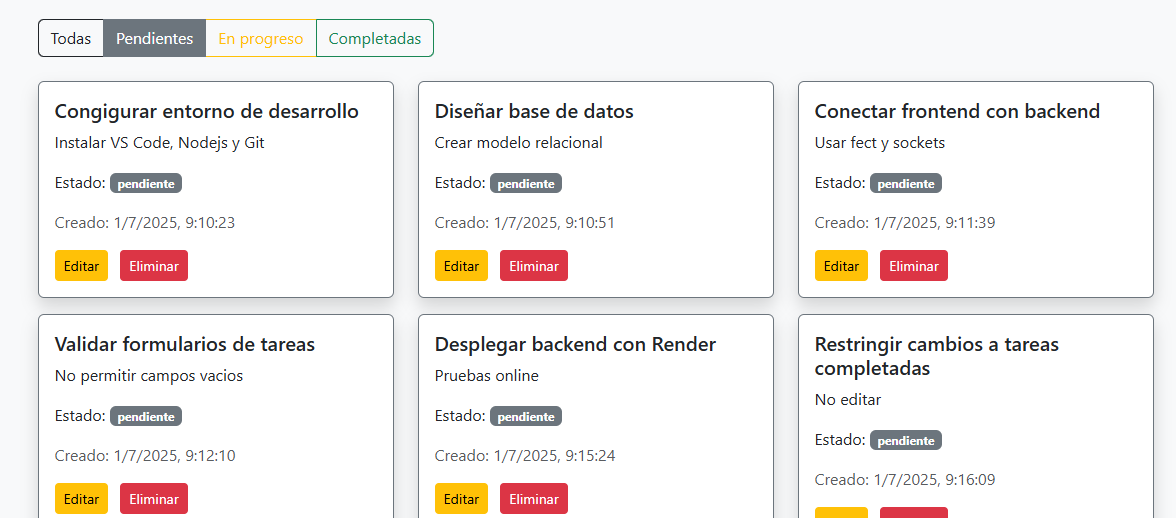


**Confirmación de eliminación:**



**Categoría de Tareas:** (Todas, Pendientes, En progreso, Completadas)

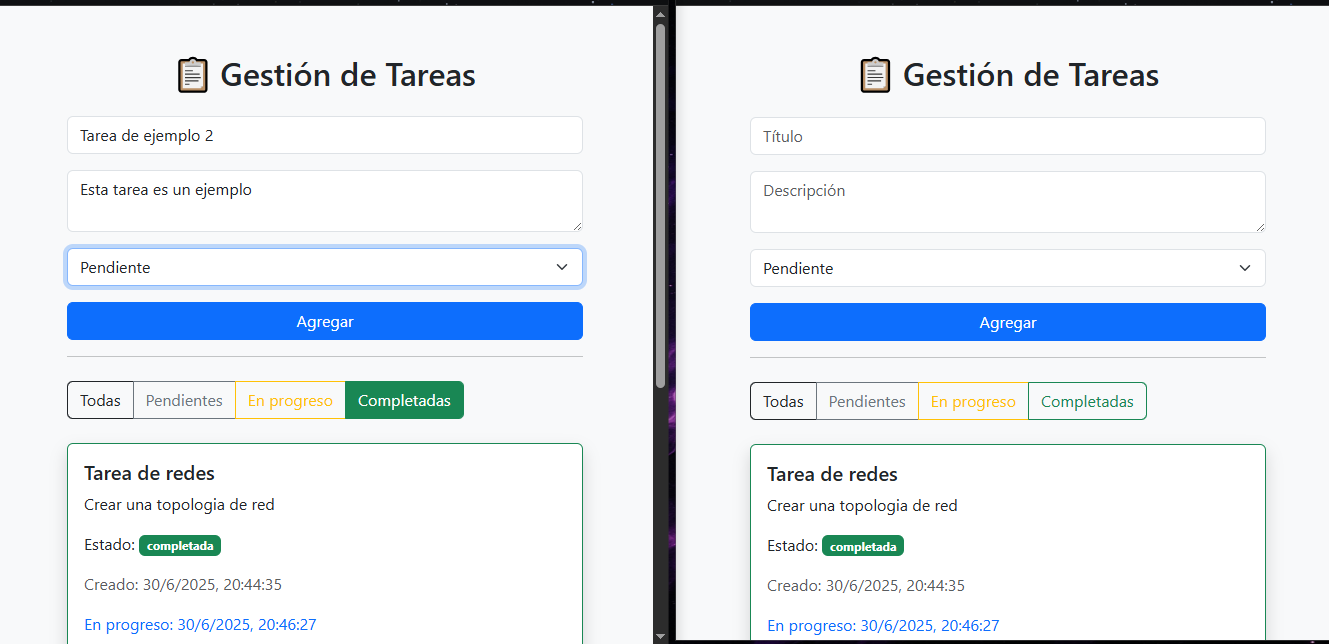


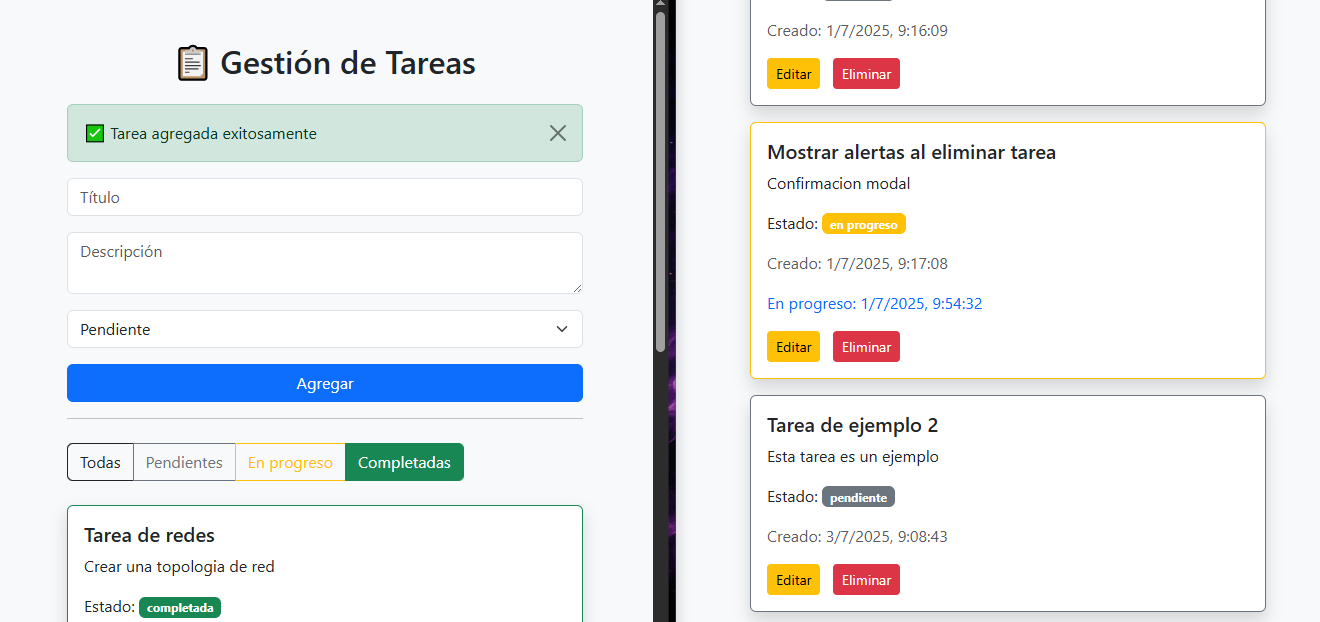




**Comprobación de WebSocket:**

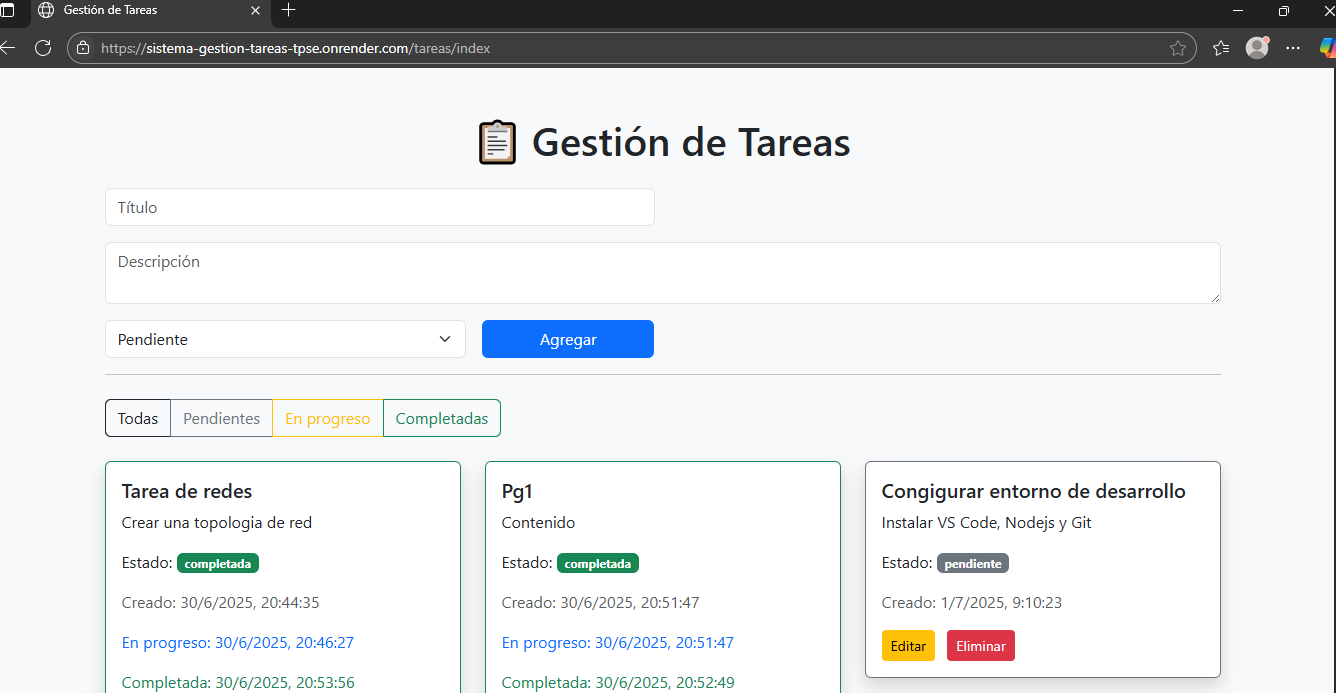
dos pestañas simultáneamente para mostrar la actualización a tiempo real.



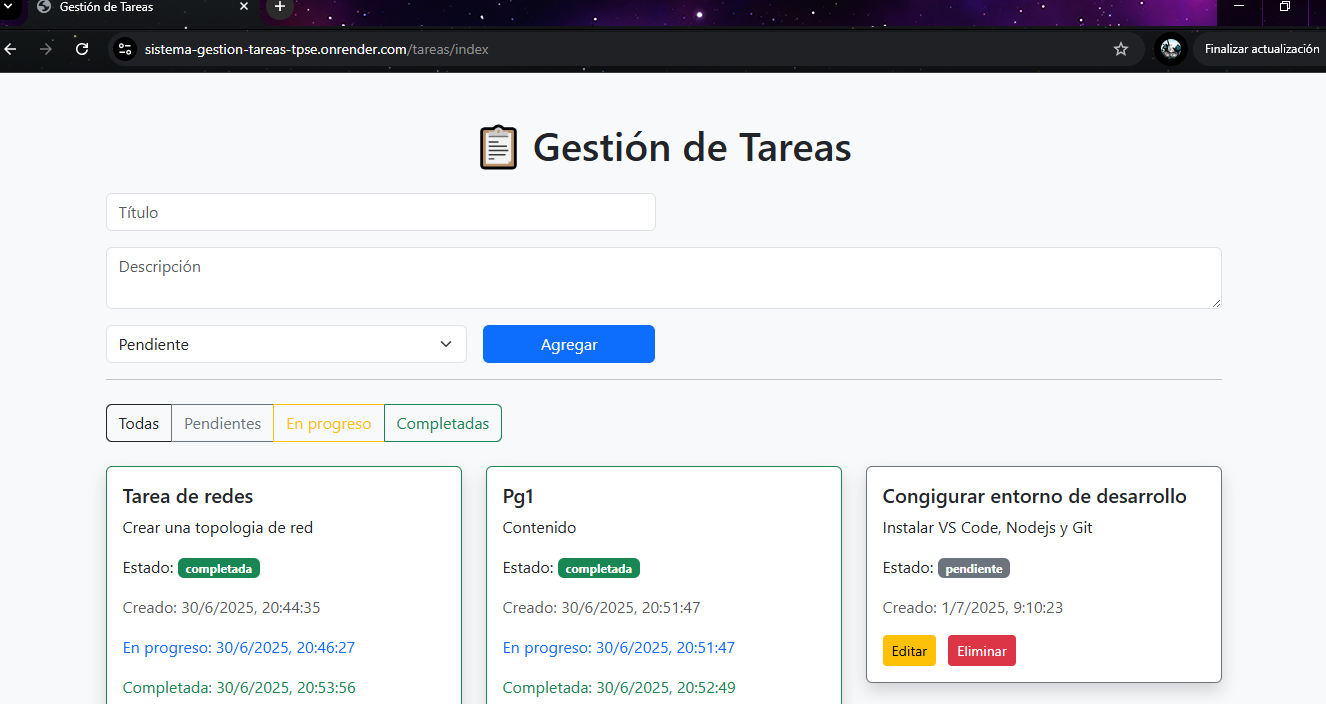


**Compatibilidad con navegadores y dispositivos:**

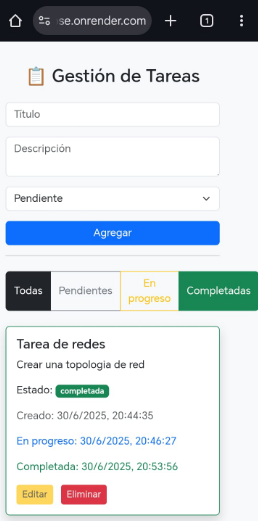
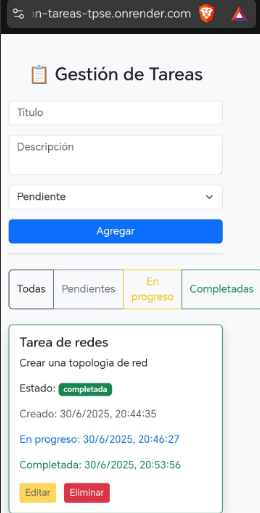
Microsoft Edge: (Windows 10, laptop)



Google Chrome: (Windows 10, laptop)



Google Chrome: (Móvil Android) Brave: (Móvil Android)

Firefox: (Móvil Android)

