**Actividad 5 - Propuesta de aplicativo con arquitectura**

**Imagen que contiene Logotipo

Descripción generada automáticamente**

Presentado Por:

Talia Yaritza Gelvez Gelvez (Ingeniería de Software)

Paula Andrea Ramirez Casilimas (Ingeniería de Software)

Presentado a:

 Docente: Joaquín Sánchez

Corporación Universitaria Iberoamericana

Facultad de Ingeniería

Bogotá, Colombia

2024

**Presentación del producto y sus ventajas de usar la arquitectura propuesta**

**Link repositorio:** <https://github.com/TaliaGelvez20/Gestor_De_Tareas.git>

El producto desarrollado es un Gestor de Tareas, el cual permite a los usuarios realizar acciones como agregar, editar y eliminar tareas, con un sistema que refleja el estado de cada tarea (pendiente, en proceso o finalizada). Esta aplicación, presentada en un entorno visual claro y accesible, emplea una interfaz minimalista, desarrollada en HTML, estilizada con CSS, y enriquecida con interactividad mediante JavaScript. Cada acción realizada por el usuario, como la adición de una nueva tarea o la modificación de una existente, se refleja inmediatamente en la interfaz, mostrando un flujo intuitivo para el manejo de las tareas diarias.

El uso de la arquitectura Modelo-Vista-Controlador (MVC) en el proyecto proporciona numerosas ventajas. La separación de responsabilidades entre la vista (frontend), el controlador (backend) y el modelo (los datos y la lógica de negocio) es fundamental para mejorar la organización y escalabilidad del código. En esta arquitectura, la vista se encarga únicamente de la presentación, es decir, de cómo los datos se muestran al usuario. En el caso del gestor, la vista es gestionada a través de un archivo index.html que presenta un formulario para agregar tareas, una lista que las muestra, y botones para editar o eliminarlas. El CSS asociado permite que cada tarea tenga un estilo diferente dependiendo de su estado, lo que ayuda a la claridad visual y a la experiencia de usuario​.

Por su parte, el controlador en Java maneja las operaciones lógicas y de negocio. Cada interacción del usuario en el frontend, como agregar una tarea o actualizar su estado, genera una solicitud HTTP hacia el backend, que se ocupa de procesar esa solicitud y devolver la respuesta adecuada a la vista. Esta interacción se implementa mediante la API REST en el servidor, donde las operaciones CRUD (crear, leer, actualizar y eliminar) se gestionan utilizando peticiones POST, GET, PUT y DELETE. El uso de Java como lenguaje para el controlador permite una gestión robusta de las solicitudes y respuestas, manteniendo la lógica del negocio fuera de la vista​.

Las ventajas de utilizar la arquitectura MVC en este tipo de proyectos son múltiples. En primer lugar, facilita la mantenibilidad del código. Al mantener separada la lógica de negocio de la presentación, cualquier modificación en la interfaz del usuario no afecta el funcionamiento interno de la aplicación. Por ejemplo, los cambios en el diseño del formulario de tareas, el cual se implementa en HTML y CSS, no requieren alteraciones en el controlador o el modelo. Esto permite a los desarrolladores modificar o actualizar la apariencia de la aplicación sin riesgo de introducir errores en el sistema principal.

Además, la escalabilidad es otra de las grandes ventajas de esta arquitectura. En proyectos que crecen en complejidad o en equipos de desarrollo grandes, es posible que varios desarrolladores trabajen en diferentes aspectos del proyecto de manera simultánea. Mientras unos se enfocan en mejorar o corregir el diseño del frontend, otros pueden optimizar las funciones de la API o mejorar el rendimiento del controlador. Esto es posible porque la arquitectura MVC segmenta claramente las diferentes capas del sistema, lo que permite una colaboración más eficiente entre los miembros del equipo.

Finalmente, esta arquitectura facilita la reutilización de componentes. En el contexto del gestor de tareas, la misma lógica de negocio y controlador podrían ser utilizados para desarrollar otras aplicaciones que manejen diferentes tipos de entidades, como un gestor de proyectos o de eventos, sin necesidad de reescribir grandes porciones de código. Esta modularidad, característica del diseño MVC, asegura que las funcionalidades puedan ser extendidas o adaptadas fácilmente a otros contextos sin afectar la estructura básica del sistema.

Es así, como la implementación del Gestor de Tareas con la arquitectura MVC demuestra cómo la correcta separación de responsabilidades, junto con el uso de tecnologías web estándar, proporciona un producto final eficiente, fácil de mantener y escalar, garantizando una experiencia de usuario fluida y un sistema backend robusto para gestionar todas las operaciones.

**Integrantes: Talia Yaritza Gelvez Gelvez - Paula Andrea Ramirez Casilimas**

**Gestor de Tareas**

Este proyecto es una aplicación sencilla para la gestión de tareas, implementada con un backend en Java y un frontend en HTML, CSS y JavaScript. Permite a los usuarios agregar, editar y eliminar tareas, las cuales se almacenan en un servidor que expone una API REST. La arquitectura de software utilizada es el patrón Modelo-Vista-Controlador (MVC), donde el frontend actúa como la vista, el backend como el controlador, y el modelo representa los datos y la lógica de negocio. La API REST facilita la comunicación entre la vista y el controlador.

**La aplicación se divide en dos partes:**

**Frontend:** Implementado en HTML, CSS y JavaScript, donde se permite al usuario interactuar con el gestor de tareas. Las tareas se pueden agregar, editar y eliminar, con cambios que se reflejan en la interfaz de usuario.

**Backend:** Implementado en Java, expone una API REST que maneja las operaciones CRUD para las tareas. Las peticiones fueron probadas usando Postman, lo cual permitió verificar el correcto funcionamiento de las solicitudes y respuestas del API, facilitando la depuración y pruebas de las operaciones.

**Funcionalidades**

**Agregar nuevas tareas con un estado (Pendiente, En Proceso, Finalizada).**

**Editar tareas existentes.**

**Eliminar tareas.**

**Mostrar todas las tareas desde el backend.**

**Estructura del Proyecto**

**Frontend:** Consiste en un archivo index.html que estructura la página, style.css para los estilos, y script.js que maneja la lógica de interacción con el backend.

**Backend:** Consiste en un servidor Java que expone una API REST para la gestión de las tareas. Se incluyen controladores para las operaciones de creación, actualización y eliminación de tareas.

**Instalación y Uso Requisitos Previos**

**Java:** Asegúrate de tener una versión reciente de Java instalada en tu sistema.

**Postman:** Instalado para probar y enviar solicitudes HTTP al servidor backend.

**Navegador Web:** Cualquier navegador moderno funcionará para acceder alfrontend.

**Instrucciones**

**Clonar el repositorio:**

bash

git clone <URL\_DEL\_REPOSITORIO> cd gestor-de-tareas

**Ejecutar el Backend:**

Para ejecutar el backend, navega a la carpeta que contiene los archivos del backend y ejecuta el servidor Java. Asegúrate de que el servidor escuche en localhost:4560.

bash

java -jar servidor.jar

**Probar las APIs con Postman:**

**Abre Postman y utiliza las siguientes configuraciones para probar las peticiones:**

**GET /tareas:** Obtiene todas las tareas.

**POST /tareas:** Crea una nueva tarea enviando un cuerpo en formato JSON.

**PUT /tareas/{id}:** Actualiza una tarea existente con un JSON que incluya los cambios.

**DELETE /tareas/{id}: Elimina una tarea.**

**Ejemplo de solicitud POST:**

json

{

"nombre": "Tarea de ejemplo",

"estado": "pendiente"

}

**Abrir el Frontend:**

**Abre el archivo index.html en tu navegador.** La aplicación interactuará automáticamente con el backend para mostrar, agregar, editar y eliminar tareas.

**Uso**

Para agregar una tarea, ingresa un nombre en el campo de texto y selecciona el estado de la tarea, luego presiona "Agregar".

Para editar una tarea, presiona el botón de edición al lado de la tarea y realiza los cambios necesarios.

Para eliminar una tarea, presiona el botón de eliminar junto a la tarea que deseas borrar.

**API del Backend**

**El backend expone los siguientes endpoints:**

**GET /tareas:** Obtiene todas las tareas.

**POST /tareas:** Crea una nueva tarea.

**PUT /tareas/{id}:** Actualiza una tarea existente.

**DELETE /tareas/{id}:** Elimina una tarea.

**Tecnologías Utilizadas**

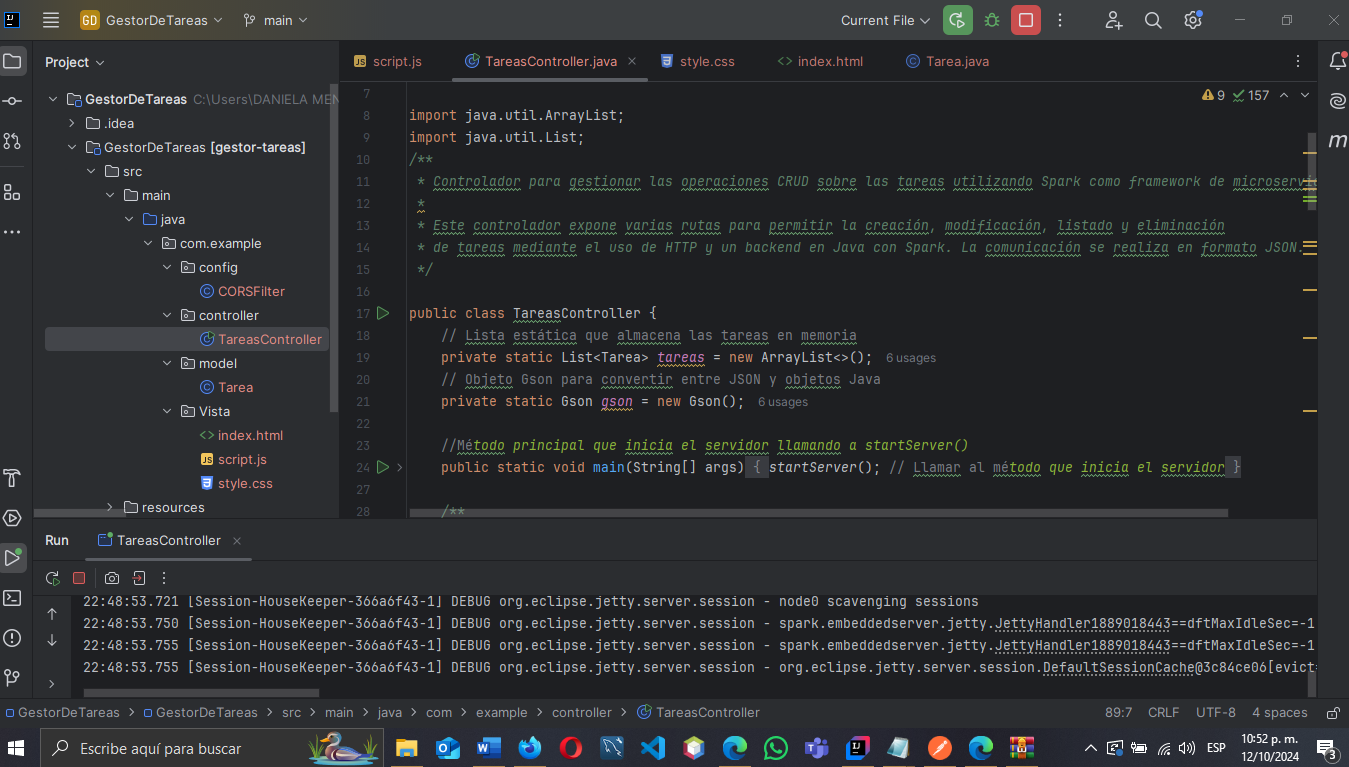
**Frontend:** HTML, CSS, JavaScript.

**Backend:** Java, API REST.

**Postman:** Para pruebas y verificación de solicitudes HTTP.

**Dependencias:** Ninguna para el frontend. Para el backend, se utilizan las dependencias estándar de Java para la creación de API REST.

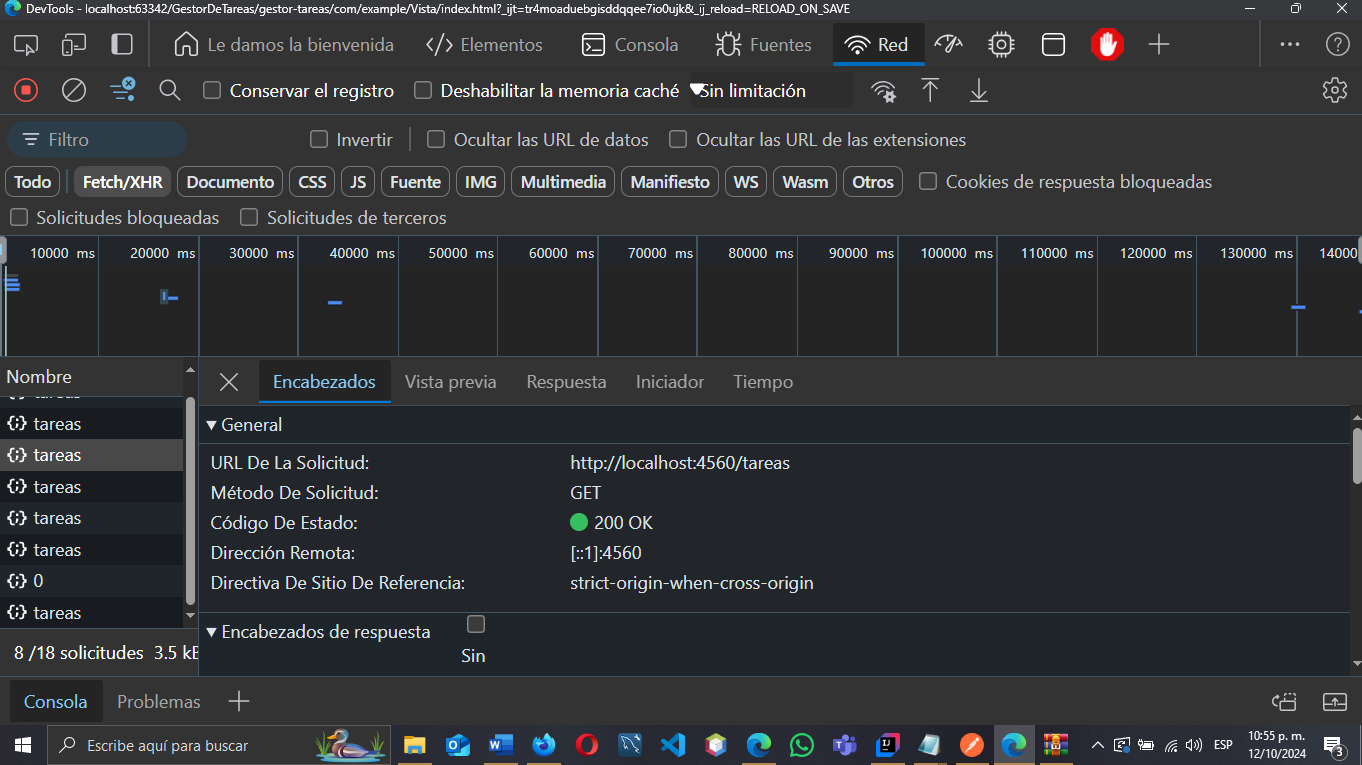
**IntelliJ**



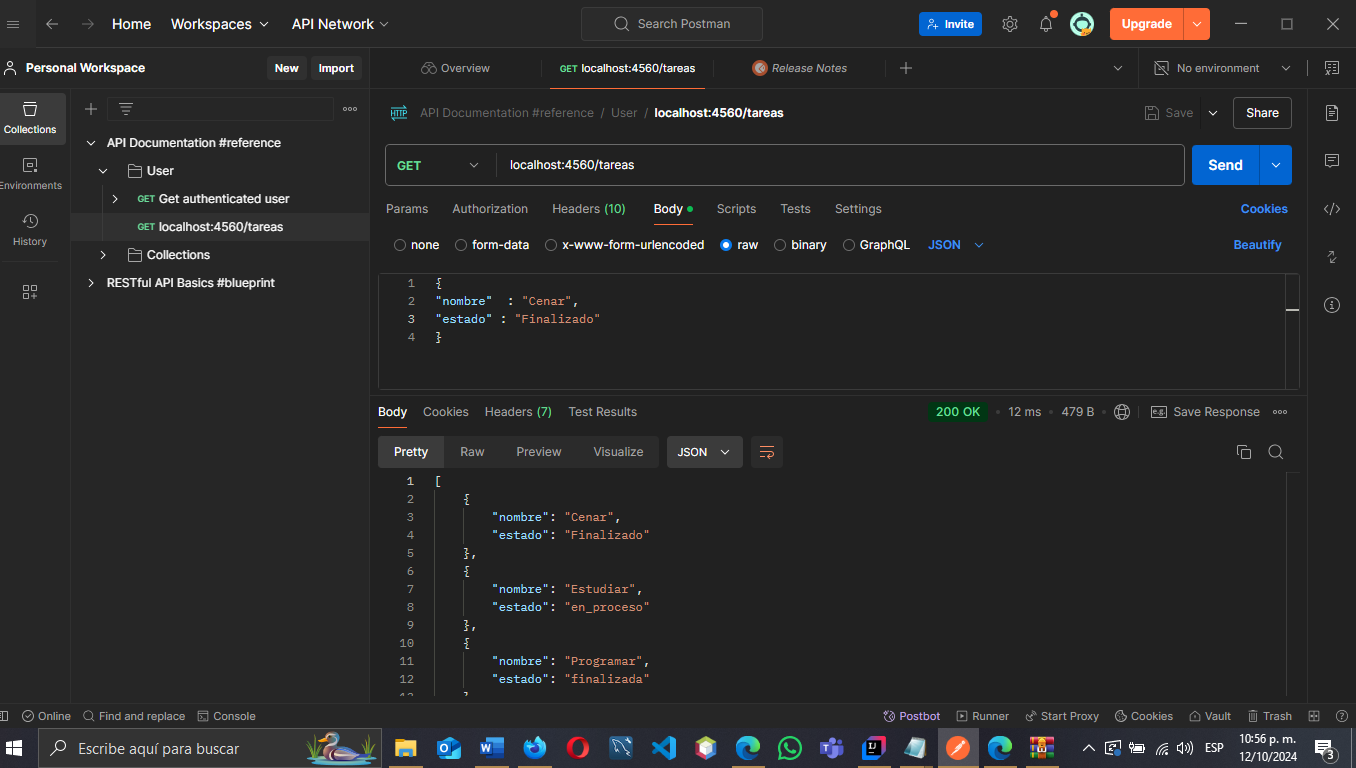
**Front**



**Inspeccionador del Navegador**



**Peticiones Postman**



**Conclusiones**

el desarrollo del Gestor de Tareas utilizando la arquitectura Modelo-Vista-Controlador (MVC) ha permitido una clara separación de responsabilidades entre la vista, el controlador y el modelo. Esta organización no solo facilita el mantenimiento del sistema, sino que también mejora su escalabilidad, permitiendo la implementación de nuevas funcionalidades sin comprometer la estabilidad del código existente.

La utilización de Java para el backend y HTML, CSS y JavaScript para el frontend ha demostrado ser una combinación eficiente. El manejo de peticiones a través de una API REST ha sido clave para mantener la interoperabilidad entre el cliente y el servidor, garantizando una experiencia de usuario fluida y robusta. El uso de Postman para las pruebas ha permitido verificar que las operaciones CRUD se realicen correctamente, proporcionando una base sólida para la gestión de tareas.

Además, la arquitectura MVC asegura que la aplicación sea fácilmente escalable y extensible. Esto es especialmente importante en proyectos de software más grandes, donde diferentes equipos de desarrollo pueden trabajar en distintas capas de la aplicación sin interferencias. La modularidad proporcionada por este patrón de diseño también facilita la reutilización de componentes, lo que resulta en una mayor eficiencia y optimización del desarrollo.

Finalmente, el proyecto no solo ha cumplido con los requisitos funcionales esperados, sino que también ha proporcionado a los desarrolladores una experiencia valiosa en el diseño e implementación de sistemas distribuidos, sentando una base sólida para futuras aplicaciones más complejas.