Documento de Desarrollo del software

Brayan Stiven López Méndez

23 de Agosto de 2024

Contents

1 Introducción

En los últimos años, el acceso remoto a laboratorios ha ganado importancia como una solución innovadora para superar las limitaciones geográficas y de recursos en la educación y la investigación. En la Universidad de Nariño, esta plataforma software surge como una herramienta esencial para modernizar el acceso a los laboratorios, permitiendo a estudiantes, investigadores y académicos interactuar de manera remota con experimentos robóticos reales en tiempo real.

Este documento describe el desarrollo e implementación de una plataforma que integra tecnologías avanzadas para gestionar experimentos remotos, garantizando una experiencia fluida, interactiva y segura para sus usuarios. A través de la implementación de una API RESTful en el backend, una interfaz web intuitiva para los usuarios, y un sistema robusto de transmisión de video en tiempo real, la plataforma se convierte en una solución escalable y eficiente para la gestión de laboratorios remotos.

Además de proporcionar acceso remoto, la plataforma incluye un sistema de autenticación para asegurar que solo usuarios autorizados puedan acceder a los experimentos, así como un sistema de cola que organiza la ejecución de múltiples experimentos en paralelo, optimizando el uso de los recursos.

1.1 Objetivo del Documento

El objetivo principal de este documento es proporcionar una descripción detallada de la arquitectura, tecnologías, y componentes clave involucrados en el desarrollo de la plataforma software de acceso remoto al laboratorio de robótica. Esto incluye:

1. Describir la arquitectura del sistema: Presentar una visión general de la arquitectura cliente-servidor utilizada en el proyecto, especificando cómo interactúan los componentes del backend y frontend.

2. Explicar la implementación del backend: Detallar el uso de FastAPI para gestionar las operaciones CRUD (crear, leer, actualizar y eliminar) de experimentos, robots y usuarios, así como la autenticación basada en tokens.

3. Presentar la interfaz de usuario: Describir cómo se ha diseñado y desarrollado el frontend utilizando HTML, CSS, y JavaScript, para proporcionar una experiencia de usuario fluida y fácil de usar.

4. Describir la configuración de Nginx: Explicar el despliegue del frontend utilizando Nginx como servidor web y proxy inverso, manejando tanto las solicitudes de la API como la transmisión de video.

5. Detallar la transmisión de video en tiempo real: Describir el uso de ROS y web_video_server para la transmisión de video, permitiendo a los usuarios visualizar los experimentos de manera remota.

6. Explicar la seguridad y autenticación: Describir el sistema de autenticación implementado para asegurar el acceso controlado a la plataforma mediante tokens.

7. Describir el sistema de gestión de experimentos: Explicar cómo se gestiona la ejecución de experimentos en la plataforma, incluyendo la ejecución de scripts y la asignación de robots a experimentos.

1.2 Contexto

El acceso remoto a laboratorios ha sido una solución cada vez más adoptada por instituciones académicas alrededor del mundo debido a sus múltiples beneficios. En el contexto de la educación en robótica, esta modalidad resulta particularmente ventajosa, ya que permite a los estudiantes experimentar con sistemas reales sin estar sujetos a las restricciones del espacio o del tiempo. Asimismo, en un entorno de investigación, el acceso remoto a equipos y experimentos permite una mayor flexibilidad, colaboraciones internacionales, y una mayor eficiencia en la gestión de recursos.

El desarrollo de esta plataforma de acceso remoto responde a la necesidad de la Facultad de Ingeniería de la Universidad de Nariño de ofrecer una herramienta moderna y eficiente para la realización de experimentos robóticos a distancia, mejorando tanto la calidad de la enseñanza como las capacidades de investigación. Los estudiantes y profesores podrán acceder a un entorno controlado donde pueden diseñar y ejecutar experimentos en robots reales, mientras que la interfaz del sistema permite la monitorización y el control de los experimentos en tiempo real a través de un navegador web.

La plataforma fue diseñada tomando en cuenta los retos asociados con la interacción remota con dispositivos físicos, tales como la latencia, la sincronización en tiempo real, y la seguridad. Estos retos se abordaron mediante la integración de soluciones tecnológicas avanzadas que permiten la transmisión de video en tiempo real, la ejecución segura de scripts personalizados y la administración de múltiples usuarios simultáneos en la plataforma.

En este contexto, la implementación de una plataforma web personalizada se presenta como la solución más adecuada para la Universidad de Nariño, dado que permite la escalabilidad y la integración con los planes académicos y de investigación actuales, asegurando una experiencia de usuario óptima. La plataforma no solo mejora la accesibilidad, sino que también habilita el acceso a un número mayor de usuarios, desde cualquier parte del mundo, con un enfoque en la usabilidad, seguridad y eficiencia.

1.3 Necesidades Cubiertas

El desarrollo de la plataforma de acceso remoto para el laboratorio de robótica de la Universidad de Nariño responde a diversas necesidades identificadas en los ámbitos de la educación, la investigación y la gestión de recursos. Estas necesidades se presentan como barreras en el acceso físico a laboratorios, la disponibilidad limitada de equipos y la necesidad de optimizar el uso de los recursos tecnológicos. A continuación, se describen las principales necesidades cubiertas por la plataforma:

1. Acceso remoto a equipos robóticos: La imposibilidad de estar presente físicamente en el laboratorio para interactuar con los robots limita la experiencia educativa y de investigación. La plataforma cubre esta necesidad al permitir que los usuarios controlen los robots en tiempo real desde cualquier ubicación geográfica a través de

un navegador web. Esto es particularmente importante para estudiantes que, por restricciones geográficas o de horario, no pueden acceder al laboratorio de forma presencial.

2. Optimización de la infraestructura física: La capacidad limitada de los laboratorios físicos y la disponibilidad restringida de equipos suelen ser un obstáculo para que un mayor número de usuarios acceda a los experimentos robóticos. La plataforma facilita la gestión eficiente de los recursos al permitir que múltiples usuarios programen y accedan a los robots de manera remota, maximizando el tiempo de uso de los dispositivos y optimizando la infraestructura del laboratorio.

3. Interacción en tiempo real con baja latencia: Para garantizar una experiencia de usuario adecuada en experimentos robóticos remotos, es crucial que la plataforma minimice la latencia en la transmisión de datos y video. La plataforma implementa tecnologías avanzadas de streaming de video en tiempo real y optimización de la red para asegurar que los usuarios puedan controlar los robots con respuestas inmediatas, manteniendo la sincronización y precisión de los experimentos.

4. Escalabilidad y flexibilidad: A medida que crece el número de estudiantes e investigadores que necesitan acceder a los laboratorios, la plataforma ha sido diseñada para ser escalable, permitiendo la incorporación de más usuarios y robots sin comprometer el rendimiento. Además, se ha construido de manera modular para facilitar futuras expansiones, como la integración de nuevos dispositivos o la adición de funcionalidades específicas para proyectos de investigación.

5. Seguridad y gestión de usuarios: La seguridad es una preocupación fundamental en el acceso remoto a laboratorios, especialmente cuando se interactúa con dispositivos físicos. La plataforma aborda esta necesidad mediante un sistema robusto de autenticación y control de acceso que garantiza que solo usuarios autorizados puedan interactuar con los robots y realizar experimentos. Además, se ha implementado un sistema de gestión de usuarios que permite a los administradores del laboratorio crear, modificar y eliminar usuarios con diferentes niveles de permisos.

6. Soporte para la enseñanza y el aprendizaje: Uno de los principales objetivos de la plataforma es mejorar la calidad de la enseñanza en robótica, brindando a los estudiantes la oportunidad de realizar experimentos prácticos de forma remota. La plataforma ofrece una interfaz intuitiva que facilita la interacción con los robots y la ejecución de scripts personalizados, proporcionando un entorno educativo dinámico que se alinea con los requerimientos curriculares de la facultad.

7. Reducción de costos operativos: Al permitir que los experimentos se realicen de manera remota, la plataforma reduce la necesidad de mantener la infraestructura física del laboratorio en uso constante, lo que se traduce en una disminución de los costos operativos. Esto incluye el ahorro en mantenimiento de equipos y consumo energético, ya que los robots pueden ser gestionados y programados de manera eficiente sin necesidad de supervisión presencial.

2 Arquitectura del Sistema

2.1 Visión General

La plataforma de acceso remoto para el laboratorio de robótica de la Universidad de Nariño está diseñada con una arquitectura modular y escalable que permite la interacción remota con robots en tiempo real, garantizando un alto rendimiento, seguridad, y flexibilidad. La arquitectura se construye sobre una base de tecnologías de software y hardware que aseguran una comunicación eficiente entre los usuarios, el servidor, y los dispositivos robóticos, integrando varios componentes clave que operan en conjunto para ofrecer una experiencia robusta y fluida.

La visión general de la arquitectura se puede desglosar en cuatro capas principales, cada una con funciones específicas dentro del sistema:

1. Capa de Interfaz de Usuario (Frontend): Esta capa proporciona la interfaz gráfica accesible desde cualquier navegador web, donde los usuarios pueden interactuar directamente con la plataforma. El frontend está desarrollado utilizando Vue.js, una tecnología moderna y altamente reactiva que facilita la experiencia de usuario. Desde la interfaz, los usuarios pueden iniciar sesión, controlar robots, subir scripts personalizados, visualizar experimentos en tiempo real a través de streaming de video, y acceder a recursos educativos. Se garantiza una experiencia intuitiva y responsiva, adecuada tanto para estudiantes como para administradores del sistema.

2. Capa de Servicios Web (API y Backend): La lógica de negocio y las operaciones del sistema están gestionadas por una API RESTful desarrollada con FastAPI. Esta API actúa como intermediario entre el frontend y los robots, permitiendo el control de los dispositivos y la gestión de los experimentos. Las funcionalidades de la API incluyen la autenticación de usuarios, el manejo de experimentos (CRUD), la programación de tareas, la ejecución remota de scripts y la transmisión de video en tiempo real. La integración con PostgreSQL como base de datos asegura un almacenamiento eficiente y seguro de la información relacionada con los usuarios y los experimentos.

3. Capa de Control de Dispositivos (Robots y ROS): Los robots son controlados a través de la plataforma mediante la Robot Operating System (ROS), que facilita la comunicación y coordinación de los dispositivos robóticos. ROS actúa como un middleware que permite enviar y recibir comandos entre la API y los robots, asegurando que las operaciones se ejecuten en tiempo real con mínima latencia. En este nivel, se integra también la transmisión de video en vivo desde los robots, permitiendo que los usuarios visualicen el estado del experimento a medida que interactúan con los dispositivos. Los drivers de los robots, como el Sphero RVR, están configurados para recibir comandos desde la API, lo que posibilita el control remoto.

4. Capa de Infraestructura (Servidores y Redes): La infraestructura del sistema está diseñada para soportar múltiples usuarios y dispositivos de forma simultánea, garantizando la estabilidad y el rendimiento del sistema. Los servidores que alojan la API y la interfaz web están desplegados con Nginx como servidor web, lo que permite manejar de manera eficiente las peticiones concurrentes. Se implementan políticas de seguridad avanzadas para asegurar la privacidad de los usuarios y la integridad

de los experimentos. Además, el sistema está configurado para escalar horizontalmente, permitiendo la incorporación de más robots o usuarios sin comprometer la calidad del servicio.

2.1.1 Flujo de Datos

El flujo de datos dentro de la arquitectura comienza cuando el usuario interactúa con la interfaz gráfica a través del navegador. Las solicitudes de control de robots o ejecución de scripts son enviadas al backend mediante la API, que a su vez se comunica con ROS para transmitir los comandos a los robots. Simultáneamente, el sistema de video streaming permite que los usuarios visualicen el experimento en tiempo real. Toda la información es almacenada en la base de datos y gestionada mediante procedimientos seguros y eficientes para garantizar la consistencia y disponibilidad de los datos.

2.1.2 Seguridad y Autenticación

La plataforma implementa un sistema robusto de autenticación de usuarios basado en JSON Web Tokens (JWT), asegurando que solo usuarios autorizados puedan acceder al sistema. Además, se implementan medidas de seguridad a nivel de red y de aplicación para proteger las comunicaciones y los dispositivos. La arquitectura está diseñada para cumplir con altos estándares de seguridad, garantizando la protección de los datos de los usuarios y el acceso controlado a los robots.

2.2 Diagrama de Arquitectura y Flujo de Datos

El diagrama de arquitectura de la plataforma de acceso remoto para el laboratorio de robótica educativa de la Universidad de Nariño ofrece una representación visual detallada de cómo se integran y comunican los diversos componentes del sistema. A través de este diagrama, se pueden observar las relaciones y flujos de información entre la interfaz de usuario, los servidores, el middleware de control de dispositivos (ROS), y los robots.

2.2.1 Descripcion del diagrama de Arquitectura

- Capa de Interfaz de Usuario (Frontend): Representada como el punto de acceso para los usuarios finales, ya sean estudiantes o administradores, que se conectan a través de un navegador web. En esta capa, las acciones realizadas por el usuario, como iniciar sesión, controlar robots, o visualizar experimentos en vivo, son enviadas hacia la API mediante solicitudes HTTP (REST).

- Capa de Servicios Web (API y Backend): Esta capa intermedia recibe las solicitudes del frontend y las procesa mediante la API RESTful desarrollada en FastAPI. Aquí, la lógica de negocio se encarga de validar y ejecutar las acciones solicitadas. Las operaciones del CRUD (crear, leer, actualizar, eliminar) de los experimentos y usuarios se gestionan en esta capa, junto con la autenticación y autorización de accesos, conectada a la base de datos PostgreSQL para el almacenamiento seguro de la información.

- Capa de Control de Dispositivos (Robots y ROS): La API, una vez validada la solicitud, transmite los comandos hacia la capa de robots mediante ROS (Robot Operating System), que actúa como el middleware encargado de la comunicación

con los dispositivos robóticos. Esta capa maneja la interacción directa con los robots, como el control del movimiento, la captura de datos sensoriales, y la transmisión de video en vivo desde los robots hacia la interfaz de usuario.

- Capa de Infraestructura (Servidores y Redes): En esta capa se ilustra el servidor de la aplicación, donde Nginx actúa como el servidor web que gestiona las conexiones entre los diferentes componentes. Los servidores también se encargan de gestionar la transmisión de video en tiempo real desde los robots hacia la interfaz web, asegurando una baja latencia y alta disponibilidad del sistema. La infraestructura está configurada para ser escalable, permitiendo añadir más robots o usuarios según las necesidades del laboratorio.

2.2.2 Descripcion del Flujo de Datos

El flujo de datos dentro del sistema sigue una trayectoria bien definida para asegurar la correcta comunicación entre los usuarios, los servidores y los robots:

- Interacción del Usuario: El flujo de datos comienza cuando el usuario interactúa con la interfaz gráfica desde su navegador. Las acciones del usuario, como iniciar sesión o controlar un robot, son enviadas como solicitudes HTTP a la API.

- Procesamiento de Solicitudes: La API RESTful recibe estas solicitudes y, dependiendo de la acción requerida, realiza diferentes procesos. Por ejemplo, para el control de robots, la API valida la solicitud y la envía hacia ROS, mientras que para el manejo de usuarios o experimentos, interactúa directamente con la base de datos PostgreSQL.

- Comunicación con ROS: En el caso de que la solicitud implique la interacción con un robot, la API envía los comandos a ROS, que es el encargado de transmitir estos comandos a los robots en tiempo real. ROS actúa como puente entre la lógica de la aplicación y los robots, permitiendo la ejecución de comandos como movimientos, ejecución de scripts, o captura de imágenes.

- Ejecución en los Robots: Los robots, al recibir los comandos de ROS, realizan las acciones indicadas, como moverse, ejecutar un script o capturar un video en tiempo real. El video capturado se transmite desde el robot hasta el servidor web para su visualización en la interfaz del usuario.

- Streaming de Video en Tiempo Real: A medida que el robot ejecuta los experimentos, se envía el video en tiempo real al servidor a través de un nodo ROS especializado en video streaming, que lo pone a disposición del usuario en la interfaz web. Esta transmisión es gestionada por Nginx, que asegura la fluidez y estabilidad de la señal de video.

- Actualización en la Interfaz del Usuario: Una vez los experimentos han sido ejecutados y los datos capturados, el frontend refleja los resultados a través de actualizaciones automáticas. El sistema está diseñado para manejar múltiples experimentos y usuarios de manera simultánea, asegurando que los datos se presenten en tiempo real sin demoras significativas.

2.2.3 Diagramas y Ejemplos

El diagrama de arquitectura ilustra este flujo de datos, con flechas que muestran la dirección de las solicitudes y respuestas, así como las comunicaciones entre los diferentes componentes del sistema. El diagrama también resalta cómo las capas están interconectadas para proporcionar una experiencia continua, desde la interacción del usuario hasta la ejecución remota de los experimentos.

2.3 Justificación de las Tecnologías Usadas

La elección de tecnologías para la plataforma de acceso remoto al laboratorio de robótica educativa de la Universidad de Nariño ha sido cuidadosamente justificada en función de su robustez, escalabilidad, facilidad de integración y capacidad para satisfacer los requisitos específicos del proyecto. Cada tecnología utilizada ha sido seleccionada para optimizar el rendimiento, asegurar la mantenibilidad y facilitar el crecimiento futuro del sistema. A continuación, se presentan las razones que respaldan las decisiones tecnológicas clave del sistema:

2.3.1 FastAPI: API RESTful para el backend

La elección de FastAPI como framework para el backend de la plataforma está basada en su rendimiento sobresaliente y su enfoque moderno para el desarrollo de aplicaciones web rápidas y eficientes. FastAPI es una herramienta asíncrona y ligera que permite manejar una gran cantidad de solicitudes simultáneamente, lo cual es esencial para la naturaleza del sistema, donde múltiples usuarios pueden estar interactuando con robots en tiempo real.

- Rendimiento: FastAPI es conocido por su alta velocidad, comparable a frameworks como Node.js o Go, lo que asegura respuestas rápidas a las solicitudes de los usuarios.
- Facilidad de uso: Su integración nativa con Python y su compatibilidad con estándares como OpenAPI y JSON Schema simplifican el desarrollo de las rutas API y permiten la creación automática de documentación interactiva, lo que facilita la mantenibilidad y la escalabilidad del sistema.
- Asincronía: La capacidad de FastAPI para manejar procesos asíncronos es fundamental en un entorno donde la ejecución de comandos y el streaming de video deben gestionarse sin interrupciones en la interacción del usuario.

2.3.2 Vue.js: Framework de JavaScript para el frontend

Vue.js ha sido seleccionado para el desarrollo de la interfaz de usuario debido a su naturaleza progresiva y su facilidad de integración con otros componentes de la aplicación. Este framework de frontend permite construir interfaces dinámicas y reactivas con una curva de aprendizaje relativamente baja en comparación con otros frameworks como React o Angular.

- Reactividad: Vue.js es ideal para la creación de interfaces altamente interactivas y actualizaciones en tiempo real, lo cual es crucial en la plataforma para reflejar los cambios en los experimentos y los resultados del laboratorio.

- Modularidad: La estructura de componentes de Vue.js permite organizar el código de manera más limpia y modular, facilitando el mantenimiento a largo plazo y la escalabilidad del sistema. Además, su capacidad de integración con bibliotecas de terceros permite una fácil personalización.

- Comunidad y soporte: Vue.js tiene una comunidad activa, lo que asegura la disponibilidad de recursos y soluciones para problemas técnicos.

2.3.3 ROS (Robot Operating System): Middleware para la comunicación con los robots

La plataforma emplea ROS como el middleware que permite la comunicación y el control de los robots utilizados en el laboratorio. ROS es la opción más adecuada para este entorno debido a su diseño modular y su popularidad en aplicaciones robóticas tanto educativas como industriales.

- Estandarización: ROS es un estándar de facto en la robótica, lo que asegura que la plataforma sea compatible con una amplia gama de hardware y sensores, y pueda integrarse fácilmente con otros sistemas robóticos.

- Comunicaciones distribuidas: ROS maneja las interacciones entre nodos de manera distribuida, lo que permite una gestión eficiente del control robótico, la recolección de datos sensoriales, y la transmisión de video sin sobrecargar los recursos del servidor central.

- Capacidad de simulación: ROS permite simular robots en entornos virtuales, lo que resulta útil tanto para pruebas antes de los experimentos en hardware real como para desarrollar y validar nuevos scripts.

2.3.4 PostgreSQL: Base de datos relacional

La plataforma utiliza PostgreSQL como base de datos relacional debido a su robustez, rendimiento y compatibilidad con entornos de alta concurrencia. PostgreSQL es una de las bases de datos más avanzadas y seguras disponibles, lo que la convierte en una elección sólida para gestionar la información crítica del sistema.

- Escalabilidad y rendimiento: PostgreSQL es altamente escalable y puede manejar grandes volúmenes de datos y múltiples conexiones concurrentes, lo cual es crucial a medida que la plataforma crece en número de usuarios y experimentos.

- Seguridad: Ofrece múltiples niveles de seguridad, incluyendo autenticación avanzada y cifrado de datos, lo cual es indispensable para proteger la información sensible de los usuarios y los datos generados durante los experimentos.

- Compatibilidad con ORM: PostgreSQL se integra de manera eficiente con SQLAlchemy, el ORM (Object Relational Mapper) utilizado en el backend de FastAPI, lo que facilita el manejo de consultas y operaciones de base de datos.

2.3.5 Nginx: Servidor web

Nginx ha sido seleccionado como el servidor web debido a su capacidad para manejar una gran cantidad de conexiones simultáneas y su uso como balanceador de carga y proxy inverso. Estas características son esenciales para garantizar que la plataforma responda de manera fluida a las solicitudes de los usuarios sin demoras, especialmente durante la transmisión de video en vivo.

- Manejo de alto rendimiento: Nginx es conocido por su eficiencia en la gestión de solicitudes HTTP y su capacidad para gestionar múltiples conexiones, lo que lo hace ideal para la entrega de contenido estático y dinámico.
- Proxy inverso: Su uso como proxy inverso permite una distribución equilibrada de la carga entre los servidores y facilita la integración de los diferentes componentes del sistema, como la API y el frontend.
- Seguridad: Nginx puede configurarse para ofrecer una capa adicional de seguridad mediante certificados SSL/TLS, protegiendo la comunicación entre los usuarios y el servidor.

2.3.6 WebSocket para transmisión de video: Tecnología para streaming en tiempo real

Para gestionar la transmisión de video en tiempo real desde los robots a la interfaz del usuario, se ha optado por utilizar WebSocket, una tecnología que permite la comunicación bidireccional entre el servidor y el cliente.

- Baja latencia: WebSocket permite el envío continuo de datos con una latencia mínima, lo cual es fundamental para asegurar una experiencia fluida durante el control de los robots y la visualización en tiempo real de los experimentos.
- Eficiencia: A diferencia de las conexiones HTTP tradicionales, WebSocket mantiene una conexión abierta entre el cliente y el servidor, reduciendo el tiempo necesario para transmitir grandes volúmenes de datos, como el video en vivo.

3 Componentes del Backend

3.1 Descripción General de FastAPI

FastAPI es el framework principal que sustenta la arquitectura del backend de la plataforma de acceso remoto al laboratorio de robótica educativa. Este framework, basado en Python, ha sido elegido por su alto rendimiento, facilidad de uso y capacidad para manejar operaciones asíncronas de manera eficiente. Su diseño está orientado a desarrollar APIs RESTful de forma rápida y sencilla, garantizando a su vez la escalabilidad y mantenibilidad del sistema a medida que este crece en complejidad y número de usuarios.

3.1.1 Características clave de FastAPI

1. Alto rendimiento: FastAPI ha demostrado ser uno de los frameworks más rápidos disponibles para construir APIs en Python, gracias a su uso intensivo de Python

asíncrono (asyncio) y al motor de tipado estático basado en Python 3.6+. Esto lo coloca al nivel de otros frameworks de alto rendimiento como Node.js y Go, siendo ideal para gestionar múltiples solicitudes concurrentes, algo esencial en un entorno donde varios usuarios pueden estar interactuando simultáneamente con los robots en el laboratorio.

2. Facilidad de desarrollo: FastAPI permite la creación de rutas API de manera rápida y eficiente, ofreciendo una sintaxis limpia y simple que reduce significativamente el tiempo de desarrollo. Su capacidad para autogenerar documentación interactiva, utilizando OpenAPI y Swagger UI, facilita a los desarrolladores y usuarios finales comprender y explorar las rutas disponibles, así como probar la funcionalidad de la API en tiempo real.

3. Soporte para asincronía: Una de las principales razones para elegir FastAPI es su capacidad para soportar tareas asíncronas, permitiendo el manejo simultáneo de múltiples solicitudes de manera eficiente. En el contexto de la plataforma, esto es fundamental para gestionar las operaciones en tiempo real, como el envío y recepción de comandos a los robots, el control remoto de los experimentos, y la transmisión de datos de sensores y video en vivo sin interrupciones.

4. Compatibilidad con SQLAlchemy: FastAPI se integra perfectamente con SQLAlchemy, un Object Relational Mapper (ORM) que facilita la interacción con bases de datos relacionales, como PostgreSQL, utilizada en la plataforma. Esta combinación permite a los desarrolladores realizar consultas y operaciones de base de datos de manera sencilla y segura, manteniendo la integridad de los datos de los usuarios y los experimentos.

5. Seguridad y autenticación: FastAPI ofrece soporte nativo para la implementación de medidas de seguridad y autenticación, como la protección mediante OAuth2 y JSON Web Tokens (JWT). En la plataforma, estas características son cruciales para asegurar que solo usuarios autorizados puedan acceder y controlar los robots, así como para proteger la información sensible almacenada en la base de datos.

6. Escalabilidad: La arquitectura modular y asíncrona de FastAPI facilita la escalabilidad de la plataforma. A medida que el número de usuarios y experimentos aumenta, el sistema puede ser fácilmente ajustado para soportar más conexiones sin comprometer el rendimiento. Además, su capacidad para manejar microservicios permite una expansión más controlada y específica de las funcionalidades.

7. Mantenimiento y documentación: FastAPI proporciona una ventaja significativa en términos de mantenimiento y documentación. Su capacidad para generar automáticamente documentación actualizada en formato OpenAPI asegura que los desarrolladores y usuarios puedan acceder siempre a la información más reciente sobre las rutas y funcionalidades disponibles, facilitando tanto el uso como la evolución del sistema.

3.1.2 Implementación en la plataforma

En el backend de la plataforma, FastAPI se encarga de gestionar las siguientes operaciones clave:

- Creación y gestión de usuarios: FastAPI maneja las rutas para registrar, autenticar y gestionar usuarios dentro del sistema, asegurando que las operaciones sean rápidas y seguras.
- Control de experimentos: A través de FastAPI, los usuarios pueden interactuar con los robots del laboratorio de manera remota, enviando comandos en tiempo real y recibiendo respuestas inmediatas.
- Transmisión de datos: La API también facilita la transmisión de datos sensoriales y de video desde los robots hacia la interfaz de usuario, manteniendo una comunicación fluida y eficiente entre los distintos componentes del sistema.

3.2 Endpoints CRUD (con ejemplos de solicitudes/respuestas)

Los endpoints CRUD (Create, Read, Update, Delete) son fundamentales en el backend de la plataforma para gestionar los principales recursos del sistema, como los usuarios, experimentos y robots. Estos endpoints están implementados utilizando FastAPI, que permite gestionar las solicitudes HTTP y manipular los datos almacenados en la base de datos de manera eficiente y segura.

A continuación, se describen los endpoints CRUD principales, junto con ejemplos de solicitudes y respuestas para cada operación.

3.2.1 Gestión de Usuarios

Los endpoints CRUD para la gestión de usuarios permiten la creación, consulta, actualización y eliminación de usuarios dentro de la plataforma. Estos endpoints son accesibles solo para usuarios autorizados y, en algunos casos, requieren permisos de administrador.

3.2.2 Gestión de Experimentos

Estos endpoints permiten gestionar los experimentos disponibles en la plataforma, asegurando que los usuarios puedan acceder, crear y modificar los experimentos de manera controlada.

3.2.3 Gestión de Robots

La plataforma también incluye endpoints CRUD para gestionar los robots, permitiendo crear, consultar, actualizar y eliminar la configuración de robots en el laboratorio.

3.3 Gestión de Autenticación y Seguridad

La gestión de autenticación y seguridad en la plataforma de acceso remoto al laboratorio de robótica educativa está diseñada para garantizar la protección de los datos y la integridad de los recursos. Este sistema utiliza prácticas modernas para la verificación de usuarios, el manejo de credenciales y el control de acceso a las funcionalidades más sensibles.

3.3.1 Autenticación basada en tokens JWT

La plataforma utiliza JSON Web Tokens (JWT) como mecanismo de autenticación. JWT es una solución eficiente y segura que permite la autenticación sin necesidad de mantener

sesiones en el servidor, lo que reduce la carga en el backend y mejora la escalabilidad del sistema.

El flujo de autenticación se gestiona de la siguiente manera:

- Inicio de sesión: Cuando un usuario se autentica proporcionando sus credenciales, el backend verifica su identidad y, si es correcto, genera un token JWT. Este token contiene información cifrada sobre la identidad del usuario y su nivel de acceso.
- Uso de tokens: El token JWT es enviado al cliente y almacenado localmente (usualmente en cookies seguras o en el almacenamiento local del navegador). Para cada solicitud posterior a los endpoints protegidos, el cliente incluye este token en el encabezado de la solicitud (Authorization: Bearer ¡token¿), lo que permite al servidor verificar la autenticidad y los permisos del usuario sin requerir credenciales adicionales.

3.3.2 Encriptación de contraseñas

Para garantizar la protección de las credenciales de los usuarios, el sistema nunca almacena contraseñas en texto plano. Las contraseñas se procesan utilizando algoritmos de hashing de alta seguridad, en este caso, bcrypt. Este método genera un hash único e irrepetible incluso si dos usuarios tienen la misma contraseña, añadiendo una capa extra de seguridad mediante el uso de salt.

1. El usuario introduce su contraseña al registrarse.
2. La contraseña es procesada con bcrypt para generar un hash seguro.
3. Solo el hash de la contraseña es almacenado en la base de datos.

Al momento de iniciar sesión, la contraseña ingresada se somete al mismo proceso de hashing y se compara el resultado con el hash almacenado.

3.3.3 Control de acceso basado en roles

El sistema implementa un control de acceso basado en roles (RBAC), que permite gestionar qué acciones pueden realizar los usuarios en función de su nivel de permisos. Los roles típicos incluyen:

- Administrador: Tiene control total sobre la plataforma, incluyendo la gestión de usuarios, robots, experimentos, y otros recursos críticos.
- Usuario estándar: Puede acceder a los experimentos, ejecutar tareas dentro de los robots asignados y consultar sus resultados.

Al usar este enfoque, los endpoints críticos están protegidos por reglas de acceso que verifican si el usuario tiene el nivel de permisos adecuado antes de permitir el acceso o la modificación de recursos sensibles.

3.3.4 Protección contra ataques comunes

La plataforma está diseñada para mitigar los riesgos de los ataques más comunes en aplicaciones web, asegurando la integridad y disponibilidad del sistema.

- Protección contra ataques de fuerza bruta: La plataforma implementa un sistema de limitación de intentos de inicio de sesión para evitar que un atacante pueda adivinar contraseñas mediante intentos repetidos.
- Prevención de inyección SQL: Las interacciones con la base de datos se realizan mediante SQLAlchemy, un ORM (Object Relational Mapper) que abstrae las consultas SQL y protege contra ataques de inyección de SQL mediante la gestión segura de los parámetros de las consultas.
- Uso de HTTPS: Todas las comunicaciones entre el cliente y el servidor están protegidas mediante HTTPS, asegurando que los datos transmitidos, incluidos los tokens de autenticación, estén encriptados y no puedan ser interceptados por atacantes.

3.3.5 Medidas adicionales de seguridad

- Autenticación de doble factor (2FA): En el futuro, el sistema está preparado para integrar una capa adicional de seguridad mediante autenticación de doble factor, permitiendo que los usuarios verifiquen su identidad no solo con sus credenciales, sino también mediante un segundo factor como un código temporal enviado a su dispositivo móvil.
- Caducidad de sesiones: Los tokens JWT tienen un tiempo de vida limitado para evitar que, en caso de robo o pérdida del token, un atacante pueda tener acceso indefinido al sistema.

3.4 Manejo de Errores y Excepciones

El manejo de errores y excepciones en el backend de la plataforma de acceso remoto al laboratorio de robótica educativa está diseñado para garantizar una experiencia de usuario fluida y robusta, proporcionando respuestas claras y útiles ante fallos, mientras se protege la integridad del sistema y se evitan posibles vulnerabilidades.

3.4.1 Gestión centralizada de errores

En la arquitectura de FastAPI, se emplea un sistema de gestión centralizada de errores mediante la implementación de excepciones controladas. Esto garantiza que cualquier fallo en el backend, tanto en las operaciones del servidor como en la interacción con la base de datos, sea capturado, gestionado y reportado de manera adecuada al cliente. El objetivo principal es evitar que el sistema presente errores inesperados o que exponga información sensible del servidor o de la base de datos.

FastAPI proporciona la clase HTTPException, que permite devolver errores HTTP personalizados. Estos errores incluyen tanto un código de estado específico como un mensaje detallado para el cliente.

3.4.2 Errores de validación

Los errores de validación de datos son manejados de manera automática y eficiente por Pydantic, la biblioteca que FastAPI utiliza para la validación de modelos. Cuando un cliente envía datos que no cumplen con los requisitos de formato o tipo esperados, Pydantic genera una excepción detallada, devolviendo una respuesta al cliente con el código de estado 422 Unprocessable Entity, acompañada de un mensaje que explica cuáles campos no cumplen los requisitos.

3.4.3 Manejo de excepciones globales

Para capturar cualquier excepción no controlada que pueda ocurrir durante la ejecución del backend, se implementa un manejador global de excepciones. Este enfoque garantiza que el sistema no se detenga abruptamente por errores imprevistos y que el cliente reciba siempre una respuesta controlada, incluso en situaciones en las que se produzcan fallos no anticipados.

3.4.4 Errores de base de datos

Los errores que surgen durante la interacción con la base de datos, tales como violaciones de integridad o consultas inválidas, se gestionan utilizando los manejadores de excepciones de SQLAlchemy. Este sistema asegura que cualquier fallo en la capa de persistencia de datos sea identificado de manera precisa y controlada.

3.4.5 Respuestas consistentes

Todas las respuestas de error en el sistema siguen un formato estándar para asegurar la consistencia en la comunicación con el cliente. Los mensajes de error incluyen un código de estado HTTP claro, un mensaje descriptivo y, en algunos casos, un detalle adicional sobre la naturaleza del error. Esto facilita la depuración y mejora la experiencia del usuario al proporcionar información precisa sobre lo que ha fallado.

3.4.6 Logging de errores

Además del manejo de excepciones, el sistema incorpora un sistema de logging (registro) para registrar cada error ocurrido en el backend. Esto permite a los administradores del sistema monitorear en tiempo real cualquier incidencia, realizar análisis detallados y tomar acciones correctivas de manera oportuna.

Los registros de errores incluyen detalles como:

- Fecha y hora del error.
- Ruta de la solicitud que generó el error.
- Detalles de la excepción.
- Información adicional del entorno, como el usuario autenticado en ese momento.

4 Componentes del Frontend

4.1 Descripcion General

El frontend de la plataforma de acceso remoto al laboratorio de robótica educativa está diseñado con un enfoque centrado en la experiencia del usuario, asegurando que las interacciones sean intuitivas, eficientes y accesibles desde cualquier dispositivo con acceso a internet. Utilizando tecnologías modernas de desarrollo web, el sistema permite una gestión fluida de experimentos y robots de forma remota, con una interfaz atractiva y responsiva.

4.1.1 Framework de desarrollo

La interfaz de usuario está construida con Vue.js 3, un framework progresivo de JavaScript que facilita la creación de interfaces de usuario interactivas y modulares. Vue.js es ideal para el desarrollo de aplicaciones web escalables debido a su flexibilidad, rendimiento y simplicidad en la integración con otras bibliotecas o proyectos existentes.

Vue.js permite gestionar eficientemente el estado de la aplicación mediante su sistema de componentes, lo que facilita la organización del código y su reutilización. Esto es clave en la plataforma, ya que cada módulo del sistema, desde el control de robots hasta la carga de scripts, está encapsulado en componentes que pueden ser actualizados y mejorados sin afectar al resto del sistema.

4.1.2 Interfaz responsiva y diseño modular

El diseño del frontend sigue un enfoque modular y responsivo, optimizado tanto para dispositivos de escritorio como móviles. Se utilizan tecnologías estándar de desarrollo web como HTML5, CSS3 y JavaScript, apoyadas por el framework Bootstrap para garantizar que la interfaz se adapte a diferentes tamaños de pantalla y dispositivos, manteniendo la funcionalidad y usabilidad en cualquier entorno.

4.1.3 Integración con el backend

El frontend interactúa con el backend de la plataforma mediante Axios, una biblioteca HTTP que facilita la realización de solicitudes API asincrónicas. La comunicación entre frontend y backend sigue el modelo RESTful, donde el frontend envía solicitudes y recibe respuestas en formato JSON. Esta estructura permite que los usuarios puedan:

- Realizar operaciones CRUD (crear, leer, actualizar, eliminar) sobre experimentos, robots y usuarios.
- Autenticarse en el sistema y gestionar sesiones de usuario.
- Ejecutar experimentos y obtener datos en tiempo real.

4.1.4 Componentes clave del sistema

El frontend está dividido en varios componentes clave, cada uno con una función específica dentro de la plataforma:

- LoginPage.vue: Encargado del inicio de sesión de los usuarios. Este componente permite la autenticación con credenciales y está vinculado al sistema de seguridad del backend.
- MainDashboard.vue: El componente central de la aplicación donde se agrupan las funcionalidades principales, como la gestión de experimentos, robots y visualización de datos en tiempo real.
- PythonCode.vue: Interfaz para la carga y ejecución de scripts Python, permitiendo a los usuarios cargar código y ver su ejecución en el sistema.
- ExperimentControl.vue: Facilita la interacción con los experimentos remotos, proporcionando controles para iniciar, detener y monitorear experimentos en vivo.

4.1.5 Flujo de usuario simplificado

El diseño de la experiencia de usuario (UX) está pensado para ser lo más sencillo posible, reduciendo la curva de aprendizaje y facilitando el uso de la plataforma para usuarios con diferentes niveles de experiencia. Los elementos interactivos, como botones y formularios, están diseñados de forma clara y accesible, mientras que las notificaciones y mensajes brindan retroalimentación constante sobre el estado de las operaciones en curso.

4.1.6 Optimización y rendimiento

La plataforma utiliza diversas técnicas de optimización para mejorar la velocidad de carga y la eficiencia general del sistema. Entre ellas, destacan:

- Lazy loading de componentes: Carga de componentes únicamente cuando son necesarios, reduciendo el tiempo de inicio de la aplicación.
- Minificación de archivos CSS y JavaScript: Compresión de archivos para reducir el tamaño y mejorar los tiempos de respuesta.
- Almacenamiento en caché de elementos estáticos: Mejora el rendimiento del sistema al minimizar las solicitudes HTTP repetitivas.

4.2 Capturas de la Interfaz

La interfaz de usuario de la plataforma está diseñada para ofrecer una experiencia fluida y amigable, tanto para la gestión de experimentos como para la interacción con robots de manera remota. A continuación, se presentan capturas de pantalla de las principales secciones del frontend, mostrando el diseño, la funcionalidad y la organización de los elementos de la plataforma.

4.2.1 Pantalla de inicio de sesión (LoginPage.vue)

Esta captura muestra la pantalla inicial donde los usuarios deben ingresar sus credenciales para acceder a la plataforma. La interfaz es simple y eficiente, con campos claramente etiquetados para el nombre de usuario y la contraseña. Los botones de acción son prominentes, asegurando que el proceso de autenticación sea intuitivo.

4.2.2 Pantalla de inicio de sesión (LoginPage.vue)

El panel principal es el núcleo de la plataforma, donde los usuarios pueden gestionar y supervisar los experimentos. Aquí, se presenta una interfaz organizada con secciones dedicadas a la selección de experimentos, control de robots, visualización de datos y otras operaciones clave. El diseño es modular, con pestañas o paneles laterales que facilitan la navegación.

4.2.3 Control de experimentos (ExperimentControl.vue)

En esta captura se puede observar la interfaz que permite a los usuarios interactuar directamente con los experimentos. El diseño incluye botones de control (iniciar, detener, pausar) y visualizaciones en tiempo real del estado del experimento, como transmisión de video en vivo y datos recibidos de los sensores.

4.2.4 Carga y ejecución de scripts (PythonCode.vue)

Esta sección permite a los usuarios cargar scripts de Python que se ejecutarán como parte de los experimentos. La interfaz incluye un formulario de carga de archivos y una sección donde los usuarios pueden ver el progreso y el resultado de la ejecución del código. También se muestran mensajes de error y confirmación cuando los scripts son procesados.

4.2.5 Gestión de usuarios (Admin Panel)

Los administradores de la plataforma tienen acceso a una sección exclusiva para la gestión de usuarios. Aquí, pueden crear, editar y eliminar cuentas, así como asignar roles y permisos. La interfaz está diseñada para facilitar la navegación por la lista de usuarios y realizar operaciones de manera rápida y eficiente.

4.3 Interacción con la API

El frontend de la plataforma está diseñado para interactuar de manera eficiente con la API desarrollada en FastAPI. Esta interacción sigue una estructura de cliente-servidor, donde el cliente (frontend) envía solicitudes HTTP a los distintos endpoints de la API, y recibe respuestas en formato JSON. La comunicación entre ambos componentes está optimizada para garantizar una experiencia de usuario fluida y con baja latencia.

4.3.1 Uso de Axios para las solicitudes HTTP

Para gestionar las solicitudes HTTP en el frontend, se utiliza Axios, una librería JavaScript que facilita la comunicación con la API. Axios permite realizar peticiones asíncronas, lo que garantiza que la interfaz siga siendo responsiva mientras espera los datos del servidor.

La configuración de Axios está centralizada en el archivo utils/axios.js, desde donde se gestionan las solicitudes de autenticación, obtención y envío de datos relacionados con los experimentos y la gestión de usuarios.

4.3.2 Autenticación de usuarios

La autenticación de usuarios en el frontend se gestiona mediante tokens JWT (JSON Web Tokens). Cuando el usuario ingresa sus credenciales, se realiza una solicitud POST

al endpoint de autenticación de la API. Si la autenticación es exitosa, se recibe un token JWT que se almacena en el localStorage del navegador, y se incluye en las cabeceras de todas las solicitudes posteriores para autorizar el acceso a las funcionalidades protegidas de la API.

4.3.3 Manejo de datos de experimentos

El frontend se comunica constantemente con la API para gestionar la información de los experimentos. Esto incluye la creación de nuevos experimentos, la consulta del estado de experimentos en curso, y la obtención de datos experimentales almacenados en la base de datos. Estas solicitudes se hacen a través de los endpoints CRUD (crear, leer, actualizar, eliminar) proporcionados por la API.

4.3.4 Gestión de errores y respuestas

El frontend está diseñado para manejar tanto los errores devueltos por la API como las respuestas exitosas. En caso de error, Axios captura la respuesta y se muestra un mensaje de alerta en la interfaz para notificar al usuario sobre el problema. De esta manera, se garantiza que el usuario tenga una experiencia clara y que pueda tomar acción cuando sea necesario (por ejemplo, reintentar una operación).

4.3.5 Sincronización en tiempo real

Para ciertas operaciones críticas, como el control de robots o la visualización de datos en tiempo real, el frontend necesita estar sincronizado con el backend. Aunque Axios es utilizado para la mayoría de las interacciones, se pueden emplear tecnologías como WebSockets o eventos del servidor (Server-Sent Events) para mantener una comunicación en tiempo real y actualizaciones dinámicas en la interfaz.

4.4 Configuración y optimización de Nginx

Nginx es un servidor web clave para la arquitectura de la plataforma, utilizado para servir el frontend estático de la aplicación y manejar la comunicación entre el cliente y la API. Su configuración está optimizada para garantizar un rendimiento eficiente, tanto en la entrega de los recursos del frontend como en la administración de las solicitudes hacia el backend.

4.4.1 Configuración básica de Nginx

La configuración de Nginx comienza con la creación de un archivo de configuración específico para la aplicación. Este archivo se encarga de manejar el enrutamiento de las solicitudes HTTP y servir los archivos estáticos, como el HTML, CSS y JavaScript generados por el proyecto Vue.js.

4.4.2 Optimización del rendimiento

Para mejorar el rendimiento del sistema, se implementan varias estrategias de optimización en la configuración de Nginx. Estas optimizaciones incluyen:

- Compresión de archivos: Nginx puede comprimir los archivos estáticos (HTML, CSS, JS) usando gzip para reducir el tamaño de los datos transferidos y mejorar el tiempo de carga.
- Cacheo de recursos estáticos: Se configuran reglas para que los navegadores del cliente puedan almacenar en caché los recursos estáticos, como imágenes, hojas de estilo y scripts, lo que reduce la carga en el servidor y mejora la experiencia del usuario al acceder nuevamente a la plataforma.
- Límites de tamaño de carga: En algunos casos, es importante definir límites de tamaño para las solicitudes. Esto evita que los usuarios suban archivos de tamaño excesivo o hagan peticiones que puedan sobrecargar el servidor.

4.4.3 Seguridad en Nginx

Nginx también es configurado con varias medidas de seguridad para proteger tanto el frontend como el backend:

- HSTS (Strict Transport Security): Asegura que el cliente use conexiones HTTPS seguras. Este encabezado fuerza a los navegadores a usar HTTPS en futuras conexiones con el servidor.
- CORS (Cross-Origin Resource Sharing): Para gestionar la interacción entre el frontend y la API, Nginx puede controlar las políticas de CORS. Aunque FastAPI maneja las respuestas CORS, Nginx también puede definir reglas adicionales.
- Protección contra ataques DDoS: Nginx puede limitar el número de conexiones simultáneas desde una misma IP para mitigar ataques de denegación de servicio.

4.4.4 Balanceo de carga

En entornos más avanzados, Nginx puede configurarse para balancear la carga entre varios servidores de backend. Esto mejora la escalabilidad y la disponibilidad del sistema, permitiendo manejar un mayor volumen de tráfico.

4.4.5 Monitoreo y registro

El monitoreo de las solicitudes HTTP y los errores es fundamental para asegurar la estabilidad y el rendimiento de la aplicación. Nginx genera registros de acceso y errores que permiten detectar problemas y realizar ajustes en la configuración si es necesario.

Los logs de Nginx se pueden personalizar para incluir información adicional, como la duración de las solicitudes y el tamaño de las respuestas.

5 Transmisión de Video

5.1 Implementación técnica

La transmisión de video es un componente esencial de la plataforma, ya que permite a los usuarios visualizar en tiempo real las actividades realizadas por los robots durante los experimentos. Para lograr una transmisión de video eficiente y de alta calidad, se

ha implementado una arquitectura técnica específica que integra múltiples tecnologías y herramientas. A continuación se describe la implementación técnica de este sistema.

5.1.1 Arquitectura del Sistema de Transmisión de Video

El sistema de transmisión de video se basa en una arquitectura que combina ROS (Robot Operating System) para la adquisición y procesamiento de video, junto con Nginx como servidor de streaming. Esta arquitectura permite la transmisión en tiempo real a través de protocolos de red estándar, garantizando una experiencia fluida y receptiva para los usuarios.

5.1.2 Componentes Clave de la Implementación

- Cámara USB: Se utiliza una cámara USB para capturar el video en tiempo real. La cámara está conectada al dispositivo que ejecuta ROS, lo que permite la adquisición directa del video.
- ROS:ROS se encarga de gestionar el flujo de datos de video. Se configura un nodo de ROS que utiliza gscam o cv_camera para acceder a la cámara USB y publicar el video en un tópico específico de ROS. Este nodo también se encarga de procesar y codificar el video para su transmisión.
- Servidor de Streaming: Nginx se utiliza como servidor para manejar las solicitudes de video. Se configura para servir el flujo de video en un formato compatible con los navegadores web, como MJPEG o WebRTC, según las necesidades de la aplicación.
- Cliente Web (Vue.js): En el frontend, se desarrolla un componente de Vue.js que consume el flujo de video. Este componente utiliza elementos HTML5 como ¡img¿ o ¡video¿ para mostrar el contenido de video en tiempo real. Se implementan controles adicionales para iniciar o detener la transmisión según sea necesario.

5.1.3 Flujo de Datos en la Transmisión de Video

El flujo de datos en la transmisión de video se realiza de la siguiente manera:

1. La cámara USB captura el video y lo envía al nodo de ROS.
2. El nodo de ROS procesa el video y lo publica en un tópico de ROS.
3. Nginx actúa como un proxy para servir el flujo de video a través de HTTP.
4. El cliente web solicita el video a Nginx y lo muestra en tiempo real utilizando un componente de Vue.js.

Este flujo de datos permite una transmisión de video eficiente y de baja latencia, esencial para las aplicaciones de robótica en tiempo real.

5.1.4 Consideraciones de Rendimiento

Para garantizar un rendimiento óptimo de la transmisión de video, se implementan las siguientes consideraciones:

- Compresión: El video se comprime para reducir el ancho de banda necesario para la transmisión, manteniendo una calidad aceptable.
- Buffering: Se implementan técnicas de buffering en el cliente para minimizar las interrupciones durante la transmisión.
- Escalabilidad: La arquitectura se diseña teniendo en cuenta la escalabilidad, permitiendo que más usuarios se conecten sin afectar la calidad del video.

5.2 Gestión de Experimentos

La gestión eficiente de los experimentos es crucial para garantizar el uso óptimo de los recursos del laboratorio y proporcionar una experiencia fluida a los usuarios. Para abordar esta necesidad, se ha implementado un sistema de cola que organiza y gestiona las solicitudes de ejecución de experimentos. Este sistema permite a los usuarios enviar, programar y supervisar experimentos de manera ordenada y eficiente.

5.2.1 Objetivo del Sistema de Cola

El objetivo principal del sistema de cola es facilitar la gestión de experimentos, evitando conflictos y asegurando que cada solicitud se procese en el orden en que se recibe. Esto es especialmente importante en entornos donde múltiples usuarios pueden intentar ejecutar experimentos simultáneamente, garantizando así un uso eficiente de los robots y otros recursos.

5.2.2 Arquitectura del Sistema de Cola

El sistema de cola se basa en una arquitectura que integra componentes del backend y frontend, utilizando tecnologías como FastAPI para la API y una base de datos para el almacenamiento de datos relacionados con la cola. A continuación se presenta una visión general de la arquitectura:

5.2.3 Funcionamiento del Sistema de Cola

1. Recepción de Solicitudes: Los usuarios envían solicitudes para ejecutar experimentos a través de la interfaz de usuario. Cada solicitud se registra con un identificador único y se añade a la cola de experimentos.
2. Gestión de la Cola: La API de FastAPI recibe las solicitudes y las almacena en una base de datos. Se mantiene un registro del estado de cada solicitud (pendiente, en ejecución, completada, fallida).
3. Ejecutores de Experimentos: Se implementan uno o más ejecutores que procesan las solicitudes en la cola de manera secuencial. Cada ejecutor toma la primera solicitud de la cola, inicia el experimento correspondiente y actualiza el estado en la base de datos.

4. Notificaciones a Usuarios: Una vez que un experimento se completa o falla, el sistema notifica al usuario correspondiente a través de la interfaz. Esta notificación puede incluir detalles sobre el resultado del experimento y cualquier mensaje relevante.

5.2.4 Beneficios del Sistema de Cola

- Organización: Permite gestionar múltiples solicitudes de manera ordenada, evitando conflictos entre usuarios.
- Transparencia: Los usuarios pueden ver el estado de sus experimentos, lo que mejora la comunicación y la satisfacción del usuario.
- Escalabilidad: El sistema puede ampliarse fácilmente para manejar un mayor número de usuarios y solicitudes a medida que crece la demanda.

5.2.5 Ejecución de Scripts

La ejecución de scripts es una parte fundamental de la gestión de experimentos en la plataforma, ya que permite a los usuarios ejecutar procedimientos automatizados y personalizados en los robots disponibles. Esta funcionalidad no solo facilita la realización de experimentos de manera eficiente, sino que también proporciona flexibilidad a los usuarios para adaptar sus experimentos a sus necesidades específicas.

5.2.6 Objetivo de la Ejecución de Scripts

El objetivo principal de la ejecución de scripts es permitir a los usuarios cargar y ejecutar scripts personalizados que controlen el comportamiento de los robots en función de sus requerimientos experimentales. Esto proporciona una mayor personalización y control en la realización de experimentos, optimizando así el uso de los recursos del laboratorio.

5.2.7 Flujo de Ejecución de Scripts

El flujo de ejecución de scripts en la plataforma se desarrolla en varias etapas clave:

1. Carga del Script: Los usuarios pueden cargar sus scripts en la interfaz de usuario. Estos scripts se almacenan en el servidor y se preparan para su ejecución.
2. Validación del Script: Antes de ejecutar un script, se realiza una validación para asegurar que cumple con los formatos y requisitos establecidos. Esto ayuda a prevenir errores durante la ejecución.
3. Ejecutar el Script: Una vez validado, el script se envía al servidor para su ejecución en el robot correspondiente. La API se encarga de iniciar el proceso, gestionando la comunicación entre el servidor y el robot.
4. Monitoreo y Notificaciones: Durante la ejecución del script, el sistema monitorea su progreso. Los usuarios reciben notificaciones en tiempo real sobre el estado de la ejecución (en curso, completado, fallido), lo que permite realizar ajustes si es necesario.
5. Resultados y Registro: Al finalizar la ejecución, los resultados del experimento se registran y se almacenan en la base de datos. Esto permite a los usuarios revisar y analizar los resultados posteriormente.

5.2.8 Consideraciones de Seguridad

La ejecución de scripts presenta desafíos de seguridad, especialmente al permitir que los usuarios carguen y ejecuten código. Por ello, se implementan las siguientes medidas:

- Validación de Scripts: Se realiza una validación exhaustiva de los scripts antes de su ejecución para evitar la inclusión de código malicioso.
- Entorno Aislado: Los scripts se ejecutan en un entorno controlado y aislado, minimizando el riesgo de interferir con otros procesos o sistemas.
- Control de Acceso: Se implementan mecanismos de autenticación y autorización para asegurar que solo usuarios autorizados puedan cargar y ejecutar scripts.

5.2.9 Beneficios de la Ejecución de Scripts

- Flexibilidad: Permite a los usuarios personalizar sus experimentos mediante scripts adaptados a sus necesidades específicas.
- Eficiencia: La automatización de procesos reduce el tiempo necesario para llevar a cabo experimentos complejos.
- Registro de Resultados: Almacena los resultados de los experimentos, facilitando el análisis posterior y la mejora continua de los procedimientos experimentales.

6 Anexos

6.1 Código Fuente

Se incluye el código fuente relevante, con control de versiones en el repositorio de github aqui: `https://github.com/Bura-hub/Atriz_web_server.git`