ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

VICERRECTORADO ACADÉMICO

DIRECCIÓN DE DESARROLLO ACADÉMICO

FACULTAD:

INFORMATICA Y ELECTRONICA

CARRERA:

SOFTWARE

NOMBRE DEL ESTUDIANTE:

KATHERINE PAULINA PELÁEZ ROBLES

TEMA:

DISEÑO ARQUITECTÓNICO DE LA APLICACIÓN

ASIGNATURA:

APLICACIONES INFORMÁTICAS II

NIVEL: OCTAVO "A"

RIOBAMBA- ECUADOR

2025 – 2026

SELECCIONAR LA ARQUITECTURA PARA LA APLICACIÓN, CON BASE A LOS REQUISITOS ESPECIFICADOS.

Proyecto: Desarrollo de un sistema web para la gestión de sensores IoT incorporando algoritmos de machine learning para la predicción de variables objetivo.

Arquitectura seleccionada:

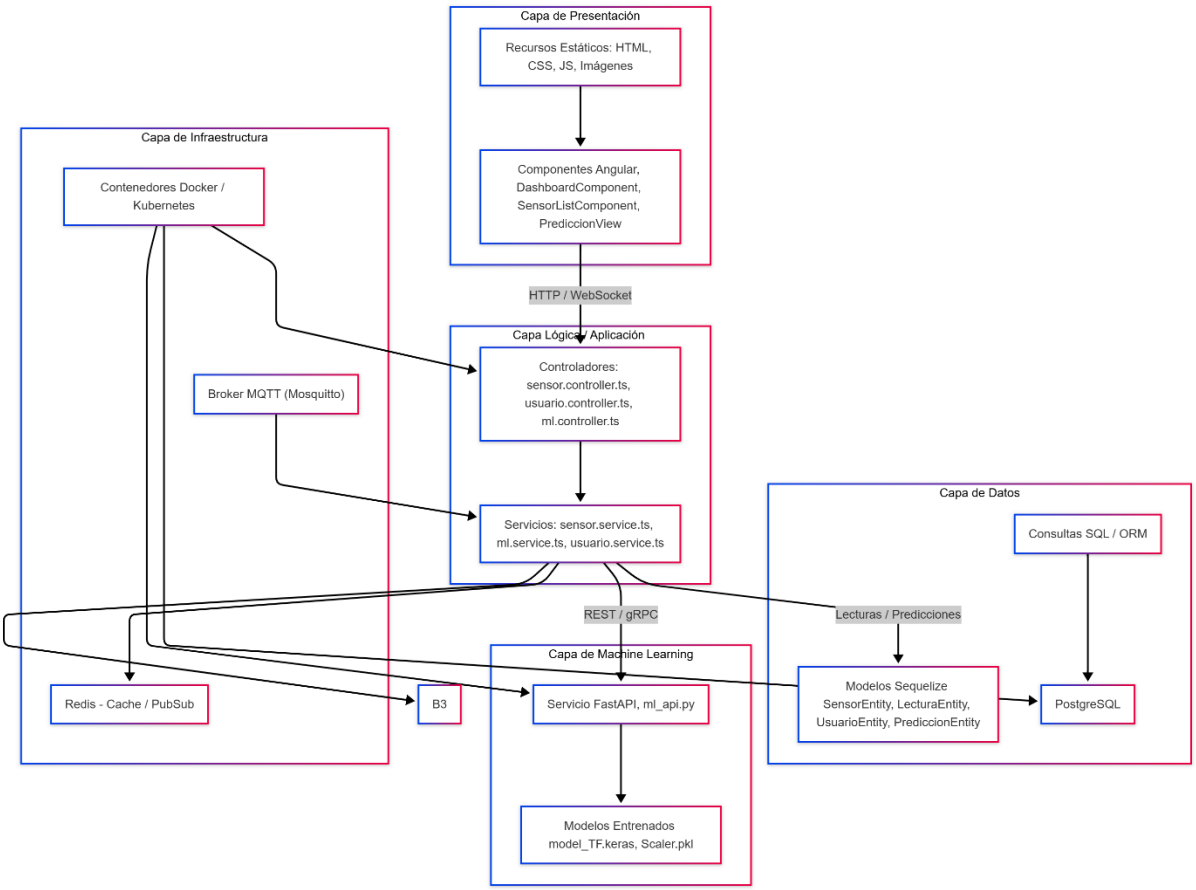
La arquitectura propuesta se fundamenta en un enfoque modular basado en microservicios, integrando el estilo arquitectónico orientado a eventos (Event-Driven Architecture) y un modelo cliente-servidor con procesamiento distribuido.

Esta arquitectura se diseñó para procesar grandes volúmenes de datos generados en tiempo real por múltiples sensores distribuidos, garantizando al mismo tiempo escalabilidad, modularidad, y facilidad de mantenimiento.

El backend principal se desarrolla con NestJS, un framework progresivo basado en Node.js y TypeScript, que implementa el patrón Modelo–Vista–Controlador (MVC) y promueve la arquitectura modular mediante el uso de módulos, controladores y servicios.

DESARROLLO DE DIAGRAMAS ARQUITECTÓNICOS

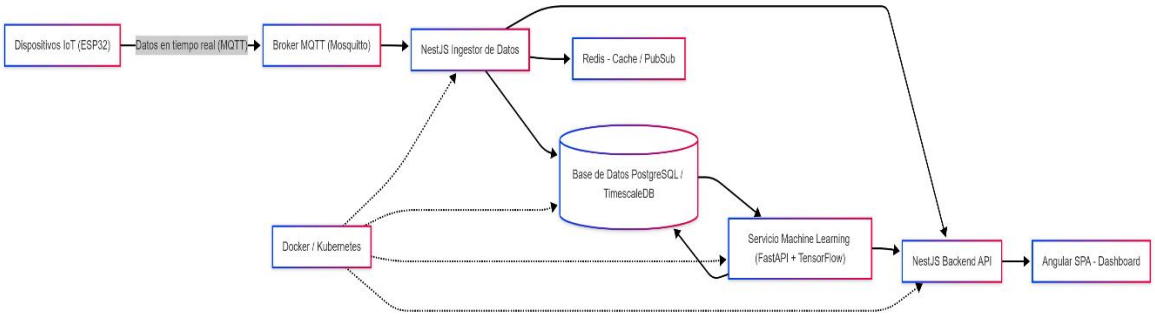
Diagrama de Capas



El diagrama de capas muestra la organización estructural del sistema web para la gestión de sensores IoT con Machine Learning, dividiendo los componentes según su función dentro de la arquitectura. La capa de presentación incluye la interfaz desarrollada en Angular, responsable de la visualización y la interacción con el usuario. La capa de aplicación, implementada con NestJS, gestiona la lógica de negocio, controladores y servicios. La capa de datos almacena y administra la información en PostgreSQL y Redis. Además, la capa IoT conecta los dispositivos con el sistema mediante MQTT, mientras que la capa de Machine Learning procesa predicciones con FastAPI y TensorFlow. Esta estructura mejora la escalabilidad, el mantenimiento y la seguridad del sistema.

Capa	Descripción	Tecnologías
Capa de Presentación	Interfaz del usuario creada con Angular. Consume datos del backend NestJS mediante HTTP o WebSocket.	Angular, TypeScript
Capa Lógica / Aplicación	Backend modular de NestJS. Contiene controladores, servicios y middleware para validar usuarios, sensores y predicciones.	NestJS, JWT, Express
Capa de Datos	Almacenamiento en PostgreSQL o TimescaleDB. ORM con Sequelize.	PostgreSQL, Sequelize
Capa de Machine Learning	Servicio FastAPI en Python que procesa y predice variables objetivo.	FastAPI, TensorFlow
Capa de Infraestructura	Comunicación, despliegue y cache. MQTT, Redis, Docker y Kubernetes.	Mosquitto, Redis, Docker

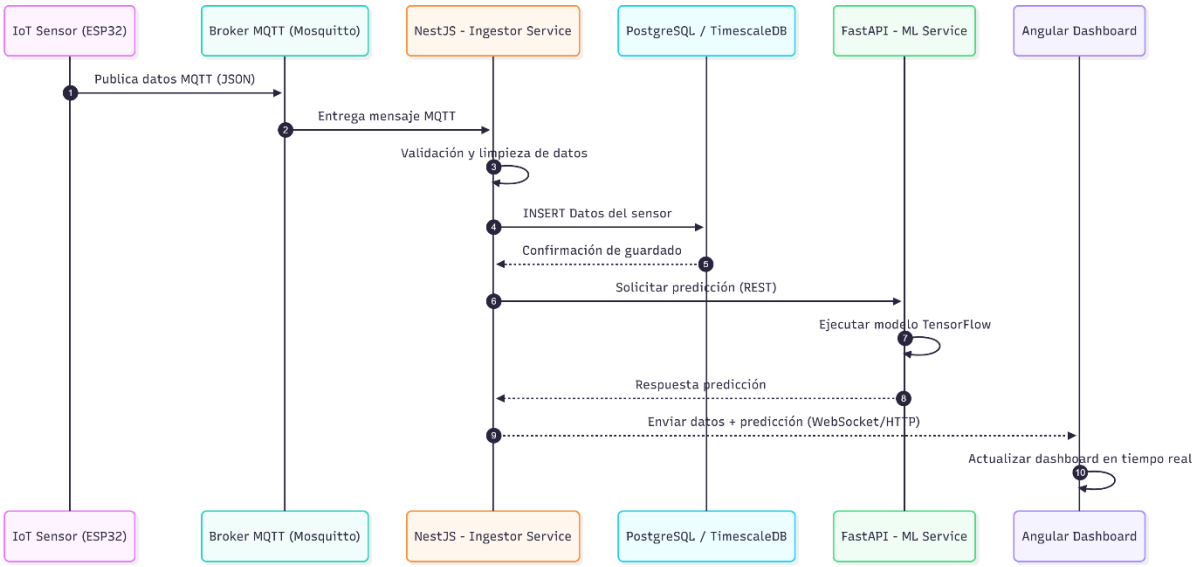
Diagrama de Flujo de Datos



1. Dispositivos IoT envían lecturas mediante MQTT al broker Mosquitto.
2. El Ingestor NestJS recibe los datos, los valida y los almacena en la base de datos y en Redis para cacheo.
3. Los datos también se envían al servicio de Machine Learning para generar predicciones.

4. El Backend NestJS consulta la base de datos y el ML para responder a las peticiones del frontend Angular.
5. Toda la infraestructura corre dentro de Docker/Kubernetes para asegurar portabilidad y escalabilidad.

Diagrama de secuencia



El diagrama de secuencia representa el flujo de interacción entre los diferentes componentes del sistema web basado en IoT y Machine Learning. En primer lugar, el sensor IoT captura datos físicos del entorno y los transmite en tiempo real al broker Mosquitto mediante el protocolo MQTT. Posteriormente, el servicio Ingestor desarrollado en NestJS se encarga de recibir estos mensajes, validarlos y almacenarlos en la base de datos PostgreSQL/TimescaleDB para su gestión histórica.

Una vez registrados los datos, el mismo servicio realiza una solicitud al microservicio de Machine Learning implementado en FastAPI para ejecutar un modelo predictivo basado en redes neuronales. El servicio ML procesa la información en TensorFlow y devuelve la predicción al backend NestJS.

Finalmente, el backend publica tanto los datos actuales como la predicción hacia el frontend Angular mediante WebSockets o HTTP, donde el usuario puede visualizar la información actualizada en el dashboard en tiempo real. Este proceso garantiza una comunicación eficiente y continua entre los sensores, el sistema de análisis y el usuario final, permitiendo un monitoreo inteligente y predictivo del entorno.

JUSTIFICACIÓN

La arquitectura seleccionada es un modelo de microservicios modular con estilo orientado a eventos (Event-Driven Architecture), combinado con una arquitectura en capas y modularidad proporcionada por NestJS. Esta elección responde directamente a los requisitos funcionales y no funcionales especificados en el SRS, tales como:

1. Escalabilidad: Al separar cada funcionalidad en microservicios independientes (Ingestor, Backend, ML, Frontend), el sistema puede escalar horizontalmente cada componente según la demanda de usuarios y sensores.

2. Modularidad y mantenibilidad: NestJS permite organizar el backend en módulos, servicios y controladores, lo que facilita agregar nuevas funcionalidades o actualizar modelos de Machine Learning sin afectar la estabilidad del sistema.

3. Procesamiento en tiempo real: La integración de MQTT y Redis permite un flujo de datos rápido y eficiente entre dispositivos IoT, backend y frontend, cumpliendo con los requerimientos de monitoreo y predicción en tiempo real.

4. Interoperabilidad y compatibilidad: La comunicación mediante REST/gRPC y WebSockets asegura que los distintos servicios (NestJS, FastAPI, Angular) puedan interactuar sin problemas.

5. Seguridad: Se implementan patrones de autenticación y autorización con JWT, middleware en NestJS y comunicación segura TLS para MQTT, asegurando confidencialidad e integridad de datos.

Estilos y patrones adoptados

- Event-Driven Architecture (EDA): Facilita la comunicación asíncrona entre servicios y la notificación inmediata de cambios de estado en sensores.

- Arquitectura en capas (Layered Architecture): Divide el sistema en presentación, lógica, datos, ML e infraestructura, asegurando claridad en responsabilidades y separación de preocupaciones.

Ventajas de la arquitectura adoptada

- Escalabilidad horizontal: Cada microservicio puede crecer de manera independiente según la carga.

- Mantenibilidad y evolución: Nuevas funcionalidades, sensores o algoritmos de ML pueden integrarse sin afectar la arquitectura base.

- Procesamiento en tiempo real: Ideal para IoT, con Redis y MQTT asegurando entrega de datos rápida y confiable.
- Seguridad integrada: Uso de JWT, TLS y control de acceso modular.
- Flexibilidad tecnológica: Permite integrar servicios escritos en distintos lenguajes o frameworks (Python para ML, Angular para frontend, NestJS para backend).

Conclusiones

1. La elección de una arquitectura modular y orientada a eventos garantiza que el sistema pueda manejar datos IoT en tiempo real y ejecutar predicciones de Machine Learning con eficiencia.
2. El uso de NestJS como framework backend proporciona estructura, consistencia y facilidad de integración con otros servicios, mejorando la mantenibilidad y escalabilidad del sistema.
3. La separación en capas y microservicios facilita la evolución del proyecto y permite que los componentes críticos, como ML o base de datos, se optimicen o escalen de manera independiente.
4. En conjunto, esta arquitectura asegura que el sistema sea robusto, flexible y preparado para el crecimiento futuro, cumpliendo con los objetivos de predicción, monitoreo y gestión de sensores IoT de manera eficiente y segura.