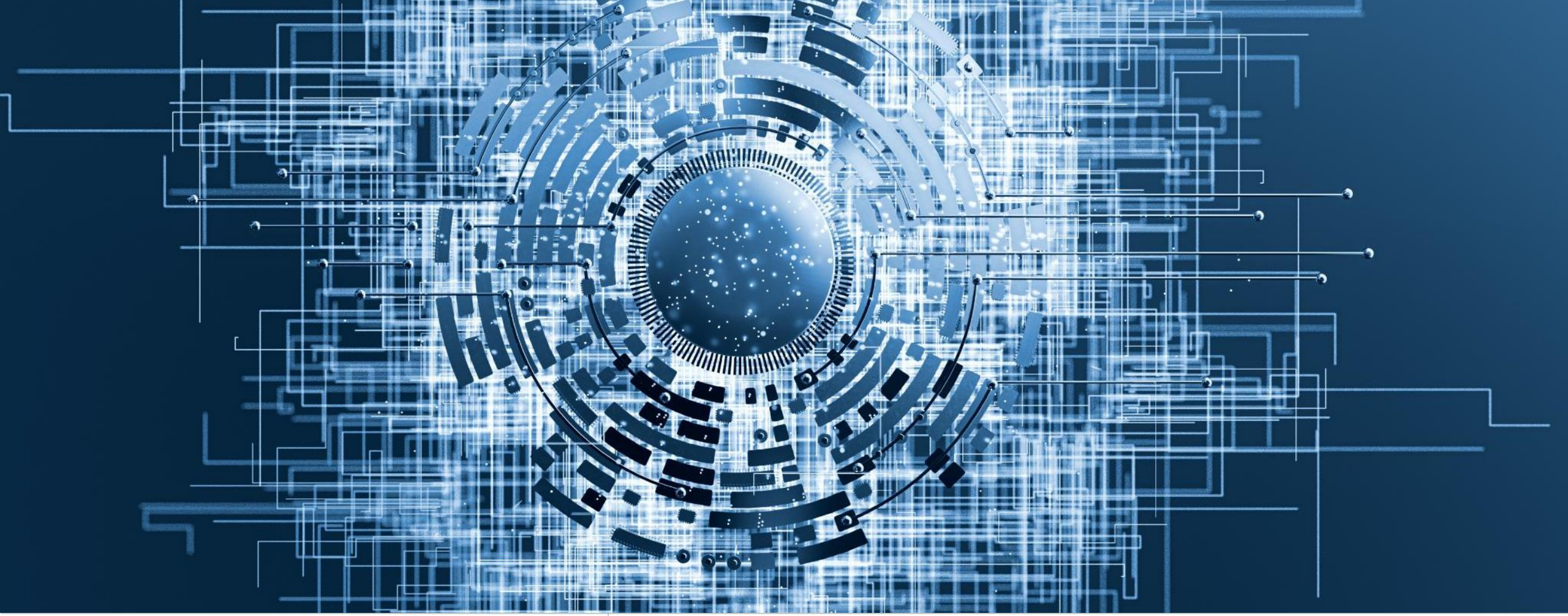
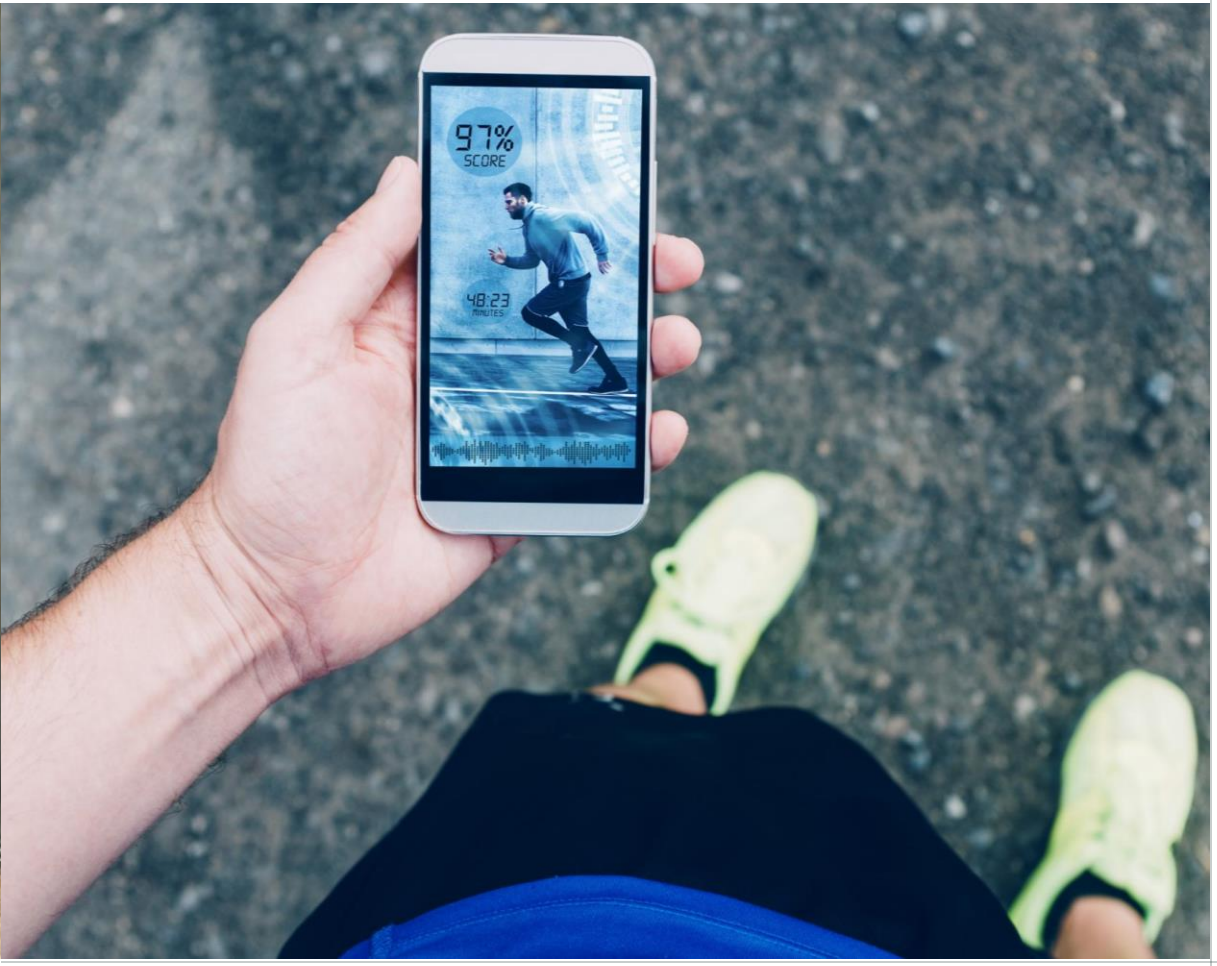


StrideSense – Presentación del Trabajo Fin de Máster

Innovación tecnológica aplicada a soluciones avanzadas

FUNDAMENTOS DEL PROYECTO STRIDESENSE

Contexto y motivación del sistema StrideSense



Crecimiento del running recreativo

El aumento del running recreativo ha generado mayor incidencia de lesiones por sobrecarga y falta de control.

Democratización de predicción de lesiones

StrideSense busca ofrecer predicción accesible de lesiones sin necesidad de hardware especializado.

Ecosistema tecnológico unificado

Uso de TypeScript, Angular, NestJS, Prisma y TensorFlow.js para un desarrollo escalable y coherente.

Impacto académico y práctico

StrideSense une ingeniería full-stack, machine learning y diseño centrado en usuario para empoderar corredores.

DEFINICIÓN DE OBJETIVOS DEL TFM STRIDESENSE

Objetivo general del proyecto

Construir un sistema que prediga el riesgo de lesión en corredores amateurs con un modelo de IA eficiente en TensorFlow.js.

Análisis de requisitos exhaustivo

Definir necesidades funcionales y no funcionales, como gestión de sesiones, seguridad, rendimiento y usabilidad.

Diseño arquitectónico modular

Diseñar arquitectura moderna y modular usando C4 y separación clara entre backend y frontend.

Interfaz de usuario accesible

Desarrollar una interfaz clara y fácil de usar que comunique riesgos y recomendaciones efectivamente.

Validación rigurosa del sistema

Realizar pruebas unitarias, de integración y seguridad para garantizar la estabilidad del sistema.

Arquitectura general del sistema StrideSense

Frontend Angular moderno

Angular 17+ gestiona la interfaz con componentes modulares y patrones de estado para una experiencia clara y moderna.

API backend con NestJS

API modular en NestJS con autenticación, seguridad robusta y separación clara de responsabilidades.

Persistencia con Prisma ORM

Prisma ORM gestiona la base de datos con esquemas tipados y migraciones, usando SQLite y PostgreSQL.

Inteligencia Artificial integrada

TensorFlow.js ejecuta un modelo neuronal eficiente en Node.js para inferencias rápidas y precisas.

Diseño e integración del modelo de inteligencia artificial

Arquitectura del Modelo

Red neuronal multicapa con dos capas ocultas y activaciones ReLU para clasificación de riesgos.

Variables y Normalización

Variables clave normalizadas con técnicas como z-score y escalado min-max para análisis preciso.

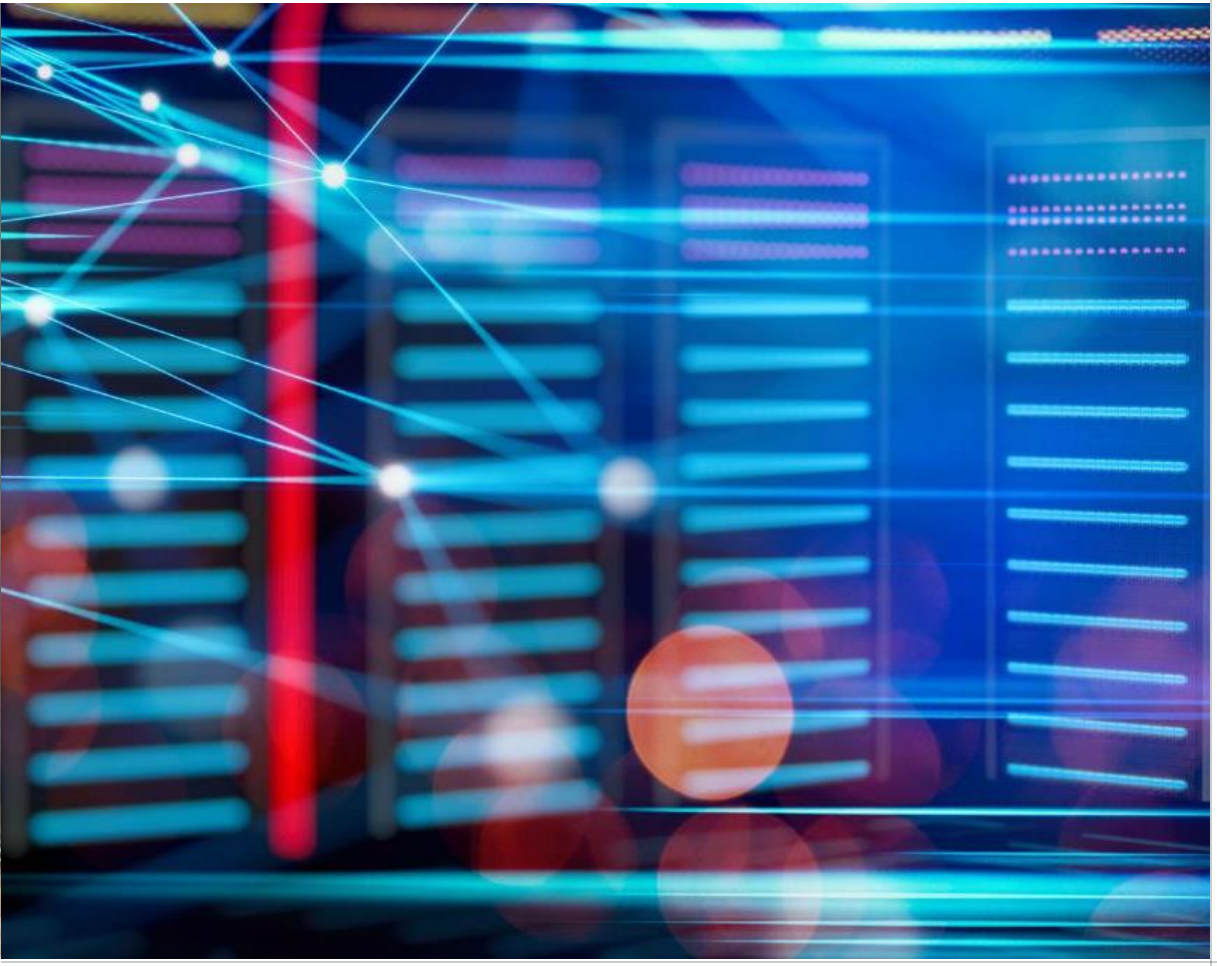
Entrenamiento con Datos Sintéticos

Modelo entrenado con datos sintéticos que simulan patrones realistas de carga y riesgo de lesión.

Integración y Ejecución

Modelo guardado en archivos JSON y BIN, integrado con backend NestJS para inferencia en producción.

Proceso de testing, seguridad y validación del sistema



Pruebas Unitarias y Funcionales

Vitest realiza pruebas unitarias para validar sanitización, cálculo de métricas y la inferencia del modelo softmax.

Pruebas de Integración de API

SuperTest verifica endpoints críticos simulando peticiones HTTP válidas, inválidas y adversariales para asegurar consistencia.

Medidas de Seguridad Implementadas

Se aplican autenticación JWT, cifrado bcrypt, cabeceras Helmet, limitación de tasa y validación estricta para proteger el sistema.

Evaluación de Rendimiento y Robustez

Se usan datasets sintéticos para simular cargas variables, asegurando estabilidad con latencias y cobertura de pruebas óptimas.

DESARROLLO TÉCNICO Y DISEÑO FUNCIONAL

Arquitectura frontend basada en Angular 17+

Estructura Modular del Proyecto

El frontend está organizado en módulos funcionales como dashboard, sesiones y autenticación para mayor escalabilidad y mantenimiento.

Gestión del Estado y Comunicación API

Se utilizan NGXS o NGRX para sincronizar datos y servicios Angular para integrar API REST con autenticación y manejo de errores.

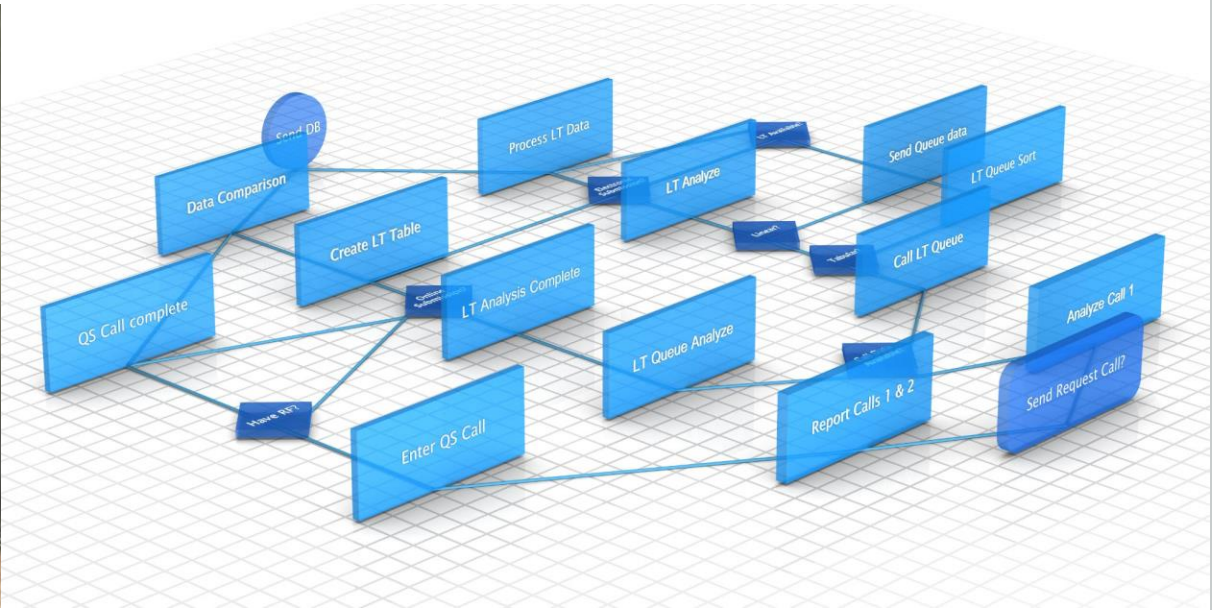
Experiencia de Usuario Moderna

Interfaz inspirada en Apple Fitness con colores suaves, tarjetas informativas y diseño responsive que prioriza accesibilidad.

Visualización y Optimización de Datos

Uso de directivas, pipes y librerías como Chart.js para crear gráficos intuitivos como el donut del riesgo.

Arquitectura backend con NestJS, Prisma y seguridad integrada



Estructura modular con NestJS

NestJS provee una arquitectura clara con módulos, controladores y servicios para escalabilidad y organización del backend.

Gestión segura de usuarios

El módulo Auth usa JWT para autenticar y bcrypt para asegurar contraseñas, garantizando seguridad en accesos.

Integración de IA y análisis de sesiones

El módulo Risk procesa modelos TensorFlow.js y Sessions registra métricas para análisis de entrenamiento avanzado.

Seguridad y gestión de base de datos

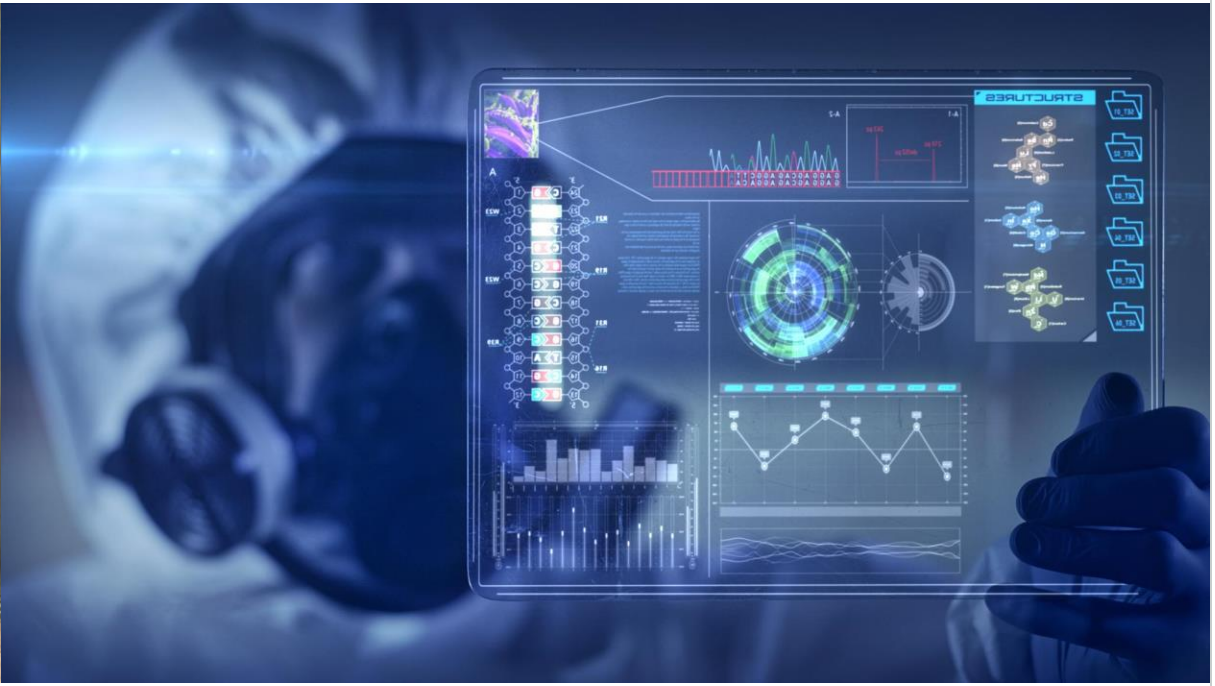
Helmet, rate limiting y validación de datos garantizan seguridad, mientras Prisma ORM gestiona eficientemente la base de datos.

ENTIDAD	ATRIBUTOS PRINCIPALES	RELACIONES
User	email, passwordHash, createdAt	1-N con TrainingSession y Prediction
TrainingSession	distanceKm, durationMin, avgHeartRate, cadence, elevationGain, rpe, sleepHours	Pertenece a User; 1-N con Prediction
Prediction	riskLevel (0/1/2), scoreVector, createdAt	Pertenece a User y TrainingSession

DISEÑO DE LA BASE DE DATOS Y MODELO ENTIDAD-RELACIÓN

EJECUCIÓN, PRUEBAS Y CONCLUSIONES

Evaluación del rendimiento y resultados obtenidos



Rendimiento del backend

El backend mostró tiempos de respuesta entre 120 y 180 ms en operaciones clave, demostrando eficiencia con CPU.

Inferencia con TensorFlow.js

La integración de TensorFlow.js en Node.js alcanzó inferencias rápidas de 70–80 ms sin GPU especializada.

Resultados con datos sintéticos

El modelo correlacionó correctamente cargas y riesgos, mostrando precisión entre 80 y 82 % en datos simulados.

Validación funcional

La interfaz mostró comunicación clara de riesgos y funcionamiento fluido sin errores en el flujo de usuario.