**Documentación de la API del Sistema Cardiovascular**

**Descripción general**

**Propósito y alcance**

Este documento proporciona una descripción general completa del sistema CardioPredict, una plataforma de predicción del riesgo cardiovascular basada en la web diseñada para profesionales de la salud. El sistema permite a los médicos evaluar el riesgo cardiovascular en pacientes utilizando modelos de aprendizaje automático, administrar datos de pacientes y monitorear análisis de predicción a través de una interfaz de panel unificada.

Esta descripción general cubre la arquitectura de alto nivel, los componentes principales y los flujos de trabajo clave del sistema CardioPredict. Para obtener información detallada sobre subsistemas específicos, consulte las siguientes secciones:

* Arquitectura y componentes del frontend: [Sistemas frontend](https://deepwiki.com/JeffCode2022/Tesis/3-frontend-systems)
* API y servicios de backend: [Sistemas backend](https://deepwiki.com/JeffCode2022/Tesis/4-backend-systems)
* Canal de predicción del aprendizaje automático: [Sistema de aprendizaje automático](https://deepwiki.com/JeffCode2022/Tesis/4.2-machine-learning-system)
* Configuración del entorno de desarrollo: [Configuración de desarrollo](https://deepwiki.com/JeffCode2022/Tesis/5-development-setup)

**Descripción general de la arquitectura del sistema**

El sistema CardioPredict sigue una arquitectura full-stack moderna con un frontend Next.js y un backend API REST de Django, integrado con capacidades de aprendizaje automático para la evaluación del riesgo cardiovascular.

Interfaz de usuario gráfica, Diagrama, Aplicación

El contenido generado por IA puede ser incorrecto.

**Componentes principales del sistema**

El sistema está organizado en distintos módulos funcionales, cada uno de los cuales maneja aspectos específicos de la predicción del riesgo cardiovascular y el manejo del paciente.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

**Pila de tecnología**

El sistema aprovecha tecnologías web modernas y marcos de aprendizaje automático para ofrecer una aplicación de atención médica sólida.

| **Capa** | **Tecnología** | **Versión** | **Archivos clave** |
| --- | --- | --- | --- |
| Frontend | Siguiente.js | Último | [frontend/app/dashboard/page.tsx](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/app/dashboard/page.tsx) |
| Tiempo de ejecución del frontend | Reaccionar | Último | [frontend/app/dashboard/page.tsx3-4](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/app/dashboard/page.tsx#L3-L4) |
| Marco de backend | Django | 3.2.24 | [backend/requisitos.txt2](https://github.com/JeffCode2022/Tesis/blob/a84f4817/backend/requirements.txt#L2-L2) |
| Marco API | Marco REST de Django | 3.14.0 | [backend/requisitos.txt3](https://github.com/JeffCode2022/Tesis/blob/a84f4817/backend/requirements.txt#L3-L3) |
| Aprendizaje automático | scikit-learn | 1.3.2 | [backend/requisitos.txt23](https://github.com/JeffCode2022/Tesis/blob/a84f4817/backend/requirements.txt#L23-L23) |
| Procesamiento de datos | pandas | 2.0.3 | [backend/requisitos.txt22](https://github.com/JeffCode2022/Tesis/blob/a84f4817/backend/requirements.txt#L22-L22) |
| Computación numérica | numpy | 1.26.3 | [backend/requisitos.txt21](https://github.com/JeffCode2022/Tesis/blob/a84f4817/backend/requirements.txt#L21-L21) |
| Persistencia del modelo | joblib | 1.3.2 | [backend/requisitos.txt24](https://github.com/JeffCode2022/Tesis/blob/a84f4817/backend/requirements.txt#L24-L24) |
| Autenticación | JWT | 5.3.0 | [backend/requisitos.txt4](https://github.com/JeffCode2022/Tesis/blob/a84f4817/backend/requirements.txt#L4-L4) |
| Base de datos | PostgreSQL | vía psycopg2 | [backend/requisitos.txt9](https://github.com/JeffCode2022/Tesis/blob/a84f4817/backend/requirements.txt#L9-L9) |
| Almacenamiento en caché | Redis | 5.0.1 | [backend/requisitos.txt13](https://github.com/JeffCode2022/Tesis/blob/a84f4817/backend/requirements.txt#L13-L13) |

**Flujos de trabajo clave del sistema**

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.**Flujo de trabajo de gestión de pacientes**

**Flujo de trabajo de generación de predicciones**

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

**Características principales**

**Gestión de pacientes**

* **Registro de pacientes**: Crear y actualizar registros de pacientes a través de **createOrUpdate()** método en [frontend/lib/services/patients.ts240-281](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/lib/services/patients.ts" \l "L240-L281" \t "_blank)
* **Búsqueda basada en DNI**: Localizar pacientes utilizando **search\_by\_dni** acción en [backend/apps/patients/views.py65-72](https://github.com/JeffCode2022/Tesis/blob/a84f4817/backend/apps/patients/views.py" \l "L65-L72" \t "_blank)
* **Historia médica**: Realice un seguimiento de los registros de los pacientes **medical\_history** punto final en [backend/apps/patients/views.py89-94](https://github.com/JeffCode2022/Tesis/blob/a84f4817/backend/apps/patients/views.py" \l "L89-L94" \t "_blank)

**Predicción del riesgo cardiovascular**

* **Predicciones individuales**: Generar evaluaciones de riesgos vía **predict** acción en [backend/aplicaciones/predicciones/vistas.py53-206](https://github.com/JeffCode2022/Tesis/blob/a84f4817/backend/apps/predictions/views.py" \l "L53-L206" \t "_blank)
* **Procesamiento por lotes**: Manejar múltiples predicciones mediante **batch\_predict** acción en [backend/aplicaciones/predicciones/vistas.py207-227](https://github.com/JeffCode2022/Tesis/blob/a84f4817/backend/apps/predictions/views.py" \l "L207-L227" \t "_blank)
* **Categorización de riesgos**: Clasificar a los pacientes en niveles de riesgo (Bajo, Medio, Alto)

**Análisis del panel de control**

* **Métricas en tiempo real**: Mostrar indicadores clave de rendimiento a través de **MetricsCards** componente en [frontend/app/dashboard/page.tsx230-245](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/app/dashboard/page.tsx" \l "L230-L245" \t "_blank)
* **Distribución del riesgo**: Visualice las categorías de riesgo de los pacientes a través de **RealTimeAnalysis** componente
* **Rendimiento del modelo**: Monitorear la precisión de las predicciones y las estadísticas del sistema

**Importación/exportación de datos**

* **Importación de datos médicos**: Procesar archivos CSV y JSON mediante **MedicalDataImport** componente en [frontend/app/dashboard/page.tsx298-304](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/app/dashboard/page.tsx" \l "L298-L304" \t "_blank)
* **Procesamiento de pacientes por lotes**: Manejar varios registros de pacientes simultáneamente
* **Procesamiento de datos de texto**: Analizar datos de texto estructurados a través de **processTextData()** función en [frontend/app/dashboard/page.tsx171-213](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/app/dashboard/page.tsx" \l "L171-L213" \t "_blank)

**Arquitectura del sistema**

Este documento describe la arquitectura general del sistema de la plataforma CardioPredict, un sistema de predicción del riesgo cardiovascular basado en la web. Cubre la estructura de alto nivel, las relaciones de los componentes, los patrones de flujo de datos y las tecnologías clave utilizadas en los sistemas frontend y backend.

Para obtener información detallada sobre componentes frontend específicos, consulte [Sistemas frontend](https://deepwiki.com/JeffCode2022/Tesis/3-frontend-systems). Para conocer los puntos finales de la API de backend y la integración del aprendizaje automático, consulte [Sistemas backend](https://deepwiki.com/JeffCode2022/Tesis/4-backend-systems). Para conocer los patrones de comunicación de API y el flujo de autenticación, consulte [Comunicación API](https://deepwiki.com/JeffCode2022/Tesis/2.3-api-communication).

**Descripción general**

CardioPredict es una aplicación web completa creada con una arquitectura frontend Next.js y backend Django. El sistema proporciona a los profesionales de la salud herramientas para el manejo del paciente, la evaluación del riesgo cardiovascular y el análisis de datos médicos a través de una interfaz moderna y responsiva.

Diagrama

El contenido generado por IA puede ser incorrecto.**Arquitectura de alto nivel**

**Arquitectura frontend**

El frontend está construido con Next.js 15.2.4 y sigue una arquitectura React moderna con soporte TypeScript. La aplicación utiliza un enfoque en capas con una clara separación entre los componentes de la interfaz de usuario, los servicios y la gestión del estado.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

**Jerarquía de componentes frontend e integración de servicios**

El frontend utiliza una arquitectura orientada a servicios donde los componentes de la interfaz de usuario se comunican con el backend a través de módulos de servicio dedicados. El **useAuth** hook administra el estado de autenticación globalmente, mientras que los servicios especializados manejan operaciones específicas del dominio.

**Arquitectura del backend**

El backend de Django implementa una arquitectura modular con una clara separación entre los puntos finales de API, la lógica empresarial y los servicios de aprendizaje automático. El sistema utiliza Django REST Framework para el desarrollo de API y servicios personalizados para la integración de ML.

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

**Arquitectura de servicios backend e integración de ML**

El backend sigue las mejores prácticas de Django con ViewSets que manejan solicitudes HTTP, clases de servicio que contienen lógica empresarial y servicios de ML dedicados para la predicción del riesgo cardiovascular. El **PredictionService** actúa como orquestador principal de las operaciones de ML.

**Arquitectura de flujo de datos**

El sistema implementa un patrón de flujo de datos unidireccional con una clara separación entre las interacciones del usuario, la comunicación API y las capas de procesamiento de datos.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

**Flujo de datos de extremo a extremo para la predicción del riesgo cardiovascular**

El flujo de datos demuestra cómo se procesan las entradas de los usuarios a través de los servicios frontend, la capa API y los servicios ML backend para generar predicciones de riesgo cardiovascular. El sistema admite tanto predicciones de modelos ML como lógica alternativa basada en reglas.

**Tecnologías clave y dependencias**

**Pila de tecnología frontend**

| **Tecnología** | **Versión** | **Propósito** |
| --- | --- | --- |
| Siguiente.js | 15.2.4 | Marco React y SSR |
| Reaccionar | 19.1.0 | Biblioteca de componentes de interfaz de usuario |
| TypeScript | 5.8.3 | Comprobación de tipo estático |
| Viento de cola CSS | 3.4.17 | Marco CSS que prioriza las utilidades |
| Axios | 1.10.0 | Cliente HTTP para llamadas API |
| Interfaz de usuario de Radix | Varios | Primitivas de componentes |
| Formulario de gancho de reacción | 7.60.0 | Gestión del estado del formulario |
| Decodificación JWT | 4.0.0 | Análisis de tokens JWT |
| Recharts | 2.15.4 | Biblioteca de gráficos y visualización |

**Pila de tecnología de backend**

| **Componente** | **Tecnología** | **Propósito** |
| --- | --- | --- |
| Marco web | Django | Marco de API de backend |
| Marco API | Marco REST de Django | Desarrollo de API RESTful |
| Base de datos | PostgreSQL | Almacenamiento de datos primarios |
| Biblioteca ML | Aprendizaje de Scikit | Modelos de aprendizaje automático |
| Serialización de modelos | Biblioteca de trabajo | Persistencia del modelo ML |
| Almacenamiento en caché | Caché de Django | Optimización del rendimiento |
| Autenticación | JWT | Autenticación basada en tokens |

**Patrones de comunicación API**

El sistema implementa varios patrones clave para una comunicación API confiable:

1. **Autenticación basada en tokens**: Utiliza tokens JWT almacenados en localStorage con lógica de actualización automática
2. **Solicitar interceptores**: Adjunte automáticamente encabezados de autenticación y tokens CSRF
3. **Interceptores de respuesta**: Manejar la actualización del token y la gestión de errores
4. **Reintentar lógica**: Reintento automático de solicitudes fallidas con retroceso exponencial
5. **Manejo de errores**: Procesamiento centralizado de errores con mensajes fáciles de usar

**Integración de aprendizaje automático**

El sistema integra capacidades de aprendizaje automático a través de un enfoque en capas que admite tanto modelos entrenados como alternativas de predicción basadas en reglas.

Una captura de pantalla de una red social

El contenido generado por IA puede ser incorrecto.

**Arquitectura y gestión de modelos de servicios de aprendizaje automático**

La integración de ML proporciona mecanismos de respaldo sólidos, lo que garantiza que el sistema siga siendo funcional incluso cuando los modelos entrenados no están disponibles. El **PredictionService** organiza la carga de modelos, la preparación de características y la generación de predicciones con un manejo integral de errores.

**Seguridad y autenticación**

El sistema implementa medidas de seguridad integrales en las capas frontend y backend:

**Seguridad del frontend**

* **Protección de ruta**: Comprobaciones de autenticación basadas en middleware en rutas protegidas
* **Gestión de tokens**: Almacenamiento seguro y actualización automática de tokens JWT
* **Protección CSRF**: Adjudicación automática de tokens CSRF para métodos HTTP no seguros
* **Validación de entrada**: Validación de formularios utilizando los esquemas React Hook Form y Zod

**Seguridad del backend**

* **Autenticación**: Autenticación basada en JWT con soporte de token de actualización
* **Autorización**: Control de acceso basado en roles para puntos finales de API
* **Validación de datos**: Validación de entrada integral a nivel de API
* **Manejo de errores**: Mensajes de error seguros que no exponen información confidencial

**Arquitectura frontend**

**Propósito y alcance**

Este documento describe la arquitectura frontend del sistema CardioPredict y cubre la estructura de la aplicación Next.js, la jerarquía de componentes, el sistema de autenticación y la organización de la capa de servicio. La interfaz sirve como interfaz de usuario para que los profesionales de la salud gestionen a los pacientes, realicen predicciones de riesgo cardiovascular y vean paneles analíticos.

Para obtener información sobre los patrones de comunicación de API y la integración del backend, consulte [Comunicación API](https://deepwiki.com/JeffCode2022/Tesis/2.3-api-communication). Para implementaciones detalladas de componentes y sistemas de interfaz de usuario, consulte [Sistemas frontend](https://deepwiki.com/JeffCode2022/Tesis/3-frontend-systems).

**Pila de tecnología**

El frontend está construido utilizando tecnologías web modernas optimizadas para aplicaciones de atención médica:

| **Tecnología** | **Versión** | **Propósito** |
| --- | --- | --- |
| Siguiente.js | 15.2.4 | Marco React con enrutador de aplicaciones |
| Reaccionar | 19.1.0 | Biblioteca de interfaz de usuario con ganchos y componentes |
| TypeScript | 5.8.3 | JavaScript seguro para tipos |
| Viento de cola CSS | 3.4.17 | Marco CSS que prioriza las utilidades |
| Interfaz de usuario de Radix | Varios | Primitivas de componentes accesibles |
| Axios | 1.10.0 | Cliente HTTP para comunicación API |
| Decodificación JWT | 4.0.0 | Análisis y validación de tokens |

Fuentes: [frontend/paquete.json1-75](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/package.json" \l "L1-L75" \t "_blank)

**Estructura de la aplicación**

El frontend sigue las convenciones del enrutador de aplicaciones Next.js con una clara separación de preocupaciones:

Imagen que contiene interior, pastel, mapa, tabla

El contenido generado por IA puede ser incorrecto.

La aplicación utiliza una arquitectura en capas donde **RootLayout** proporciona contexto temático, **middleware.ts** se encarga de la protección de la ruta y **useAuth** gestiona el estado de autenticación en todos los componentes.

**Arquitectura de componentes**

El sistema de panel implementa una jerarquía de componentes modular con un flujo de datos claro:

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

El **Dashboard** componente en [frontend/app/dashboard/page.tsx66-387](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/app/dashboard/page.tsx" \l "L66-L387" \t "_blank) sirve como orquestador principal, administrando el estado de múltiples secciones **activeSection** y **setActiveSection**. El **renderContent()** La función representa dinámicamente diferentes vistas según la navegación del usuario.

**Sistema de autenticación**

El sistema de autenticación proporciona una gestión integral de sesiones con tokens JWT:

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

El **useAuth** gancho [interfaz/ganchos/useAuth.ts6-187](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/hooks/useAuth.ts#L6-L187) gestiona el estado de autenticación con actualización automática del token cada 4 minutos. El **middleware** función en [frontend/middleware.ts3-24](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/middleware.ts" \l "L3-L24" \t "_blank) protege rutas comprobando **auth\_token** cookies y redireccionamiento de usuarios no autorizados a **/login**.

**Gestión estatal**

La aplicación utiliza la gestión de estado incorporada de React con ganchos para diferentes dominios de datos:

| **Dominio estatal** | **Gancho/Servicio** | **Propósito** |
| --- | --- | --- |
| Autenticación | **useAuth** | Sesión de usuario y gestión de tokens |
| Métricas del panel de control | **useState<DashboardMetrics>** | KPI y datos analíticos |
| Datos del paciente | **useState<Patient[]>** | Registros de pacientes y datos importados |
| Predicciones | **useState<PredictionResult>** | Resultados de la predicción de ML |
| Datos del formulario | **useState<FormData>** | Entradas del formulario de predicción |
| Estado de la interfaz de usuario | **useState<string>** | Navegación por secciones activas |

El estado del panel principal se administra en [frontend/app/dashboard/page.tsx72-122](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/app/dashboard/page.tsx" \l "L72-L122" \t "_blank) con objetos de estado separados para diferentes preocupaciones. El **metrics** El estado utiliza el **DashboardMetrics** tipo, mientras **patients** y **importedPatients** mantener matrices separadas para diferentes fuentes de datos.

**Capa de servicio**

La capa de servicio proporciona una abstracción limpia para la comunicación API y la lógica empresarial:

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

El **api.ts** servicio en [frontend/lib/services/api.ts58-201](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/lib/services/api.ts" \l "L58-L201" \t "_blank)crea una instancia de axios configurada con administración automática de tokens, protección CSRF y manejo de errores. El interceptor de solicitudes en [frontend/lib/services/api.ts69-136](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/lib/services/api.ts" \l "L69-L136" \t "_blank)agrega encabezados de autenticación y tokens CSRF, mientras que el interceptor de respuesta en [frontend/lib/services/api.ts139-198](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/lib/services/api.ts" \l "L139-L198" \t "_blank)maneja la actualización del token y las respuestas de error.

**Enrutamiento y navegación**

La aplicación utiliza Next.js App Router con navegación del lado del cliente para el panel:

| **Ruta** | **Componente** | **Propósito** |
| --- | --- | --- |
| **/** | Redirección raíz | Redirecciona al panel de control o inicia sesión |
| **/login** | Página de inicio de sesión | Autenticación de usuarios |
| **/dashboard** | Panel de control | Interfaz principal de la aplicación |
| **/dashboard?section=patients** | Lista de pacientes | Gestión de pacientes |
| **/dashboard?section=predictions** | Formulario de predicción | Evaluación de riesgos |
| **/dashboard?section=import** | Importación de datos médicos | Importación de datos |

El panel implementa una navegación basada en secciones a través del **activeSection** estado en [frontend/app/dashboard/page.tsx69](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/app/dashboard/page.tsx" \l "L69-L69" \t "_blank) y **menuItems** configuración en [frontend/app/dashboard/page.tsx215-223](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/app/dashboard/page.tsx#L215-L223) El **renderContent()** función en [frontend/app/dashboard/page.tsx225-314](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/app/dashboard/page.tsx" \l "L225-L314" \t "_blank)representa dinámicamente diferentes secciones según la selección activa.

La protección de ruta está a cargo del middleware en [frontend/middleware.ts9-21](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/middleware.ts" \l "L9-L21" \t "_blank) que comprueba tokens de autenticación y redirige a usuarios no autorizados. El **publicPaths** matriz en [frontend/middleware.ts5](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/middleware.ts" \l "L5-L5" \t "_blank)define rutas que no requieren autenticación.

**Arquitectura del backend**

Este documento explica la arquitectura backend de Django del sistema CardioPredict, incluido el diseño de API, el enrutamiento de URL, la integración del aprendizaje automático y las relaciones del modelo de datos. El backend sirve como motor de procesamiento central que maneja la gestión de datos de pacientes, predicciones de riesgo cardiovascular y análisis a través de puntos finales de API REST.

Para obtener información sobre los patrones de comunicación frontend-backend, consulte [Comunicación API](https://deepwiki.com/JeffCode2022/Tesis/2.3-api-communication). Para obtener documentación detallada del punto final de la API, consulte [Puntos finales de la API](https://deepwiki.com/JeffCode2022/Tesis/4.1-api-endpoints).

**Estructura de la aplicación y enrutamiento de URL**

El backend sigue la arquitectura basada en aplicaciones de Django con múltiples aplicaciones especializadas que manejan diferentes dominios de funcionalidad.

**Organización de aplicaciones Django**

El sistema está organizado en seis aplicaciones principales de Django:

| **Aplicación** | **Propósito** | **Componentes clave** |
| --- | --- | --- |
| **patients** | Gestión de pacientes y registros médicos | **Patient**, **MedicalRecord** modelos, API CRUD de pacientes |
| **predictions** | Motor de predicción de riesgos cardiovasculares | **Prediction** modelo, **PredictionService**, Integración de ML |
| **medical\_data** | Procesamiento ampliado de datos médicos | **MedicalData**modelo, procesamiento por lotes |
| **analytics** | Métricas y estadísticas del panel | Puntos finales de análisis, servicios de agregación |
| **integration** | Integración de sistemas externos | Importación/exportación de datos, API de terceros |
| **authentication** | Autenticación y autorización de usuarios | Gestión de tokens JWT, sesiones de usuario |

**Arquitectura de enrutamiento de URL**

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

**Diseño de API y conjuntos de vistas**

El backend implementa una arquitectura API RESTful utilizando Django REST Framework con puntos finales basados en ViewSet que proporcionan operaciones CRUD integrales y acciones especializadas.

**Conjuntos de vistas de API principales**

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

**Puntos finales clave de la API**

El **PredictionViewSet** proporciona la funcionalidad principal de predicción:

* **predict()**- Punto final de predicción integral que maneja la creación/actualización de pacientes, la creación de registros médicos y la generación de predicciones de ML
* **batch\_predict()**- Procesa múltiples predicciones en una sola solicitud
* **performance\_metrics()**- Devuelve estadísticas de rendimiento del modelo ML
* **statistics()**- Proporciona análisis de predicción y datos de distribución de riesgos

El **PatientViewSet** se encarga del manejo del paciente:

* **for\_prediction()**- Punto final optimizado que devuelve a los pacientes con los últimos registros médicos para el procesamiento por lotes
* **search\_by\_dni()**- Búsqueda de pacientes por identificación nacional con manejo duplicado
* **add\_medical\_record()**- Crea nuevos registros médicos para pacientes existentes

**Integración de aprendizaje automático**

El backend integra capacidades de aprendizaje automático a través de una arquitectura de servicio en capas que admite tanto modelos de aprendizaje automático entrenados como sistemas de predicción basados en reglas.

**Arquitectura de servicios ML**

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

**Canalización de ingeniería de características**

El **PredictionService** implementa ingeniería de funciones sofisticada para transformar datos médicos sin procesar en funciones listas para ML:

Texto

El contenido generado por IA puede ser incorrecto.

El servicio incluye mecanismos alternativos cuando los modelos ML no están disponibles, implementando algoritmos de predicción basados en reglas que siguen pautas médicas.

**Modelos de datos y relaciones**

El backend implementa una arquitectura de modelo de datos integral diseñada para la evaluación del riesgo cardiovascular y el manejo del paciente.

**Estructura del modelo de datos básicos**

Un mapa de una computadora

El contenido generado por IA puede ser incorrecto.

**Propiedades del modelo y campos calculados**

El **Patient** El modelo incluye propiedades calculadas para cálculos médicos:

* **imc**- Cálculo del índice de masa corporal: **peso / (altura\_m^2)**
* **nombre\_completo**- Concatenación de nombres completos

El **MedicalRecord** El modelo proporciona indicadores de riesgo cardiovascular:

* **presion\_arterial**- Formato de la presión arterial: **"systolic/diastolic"**
* **indice\_paquetes\_ano**- Índice de tabaquismo: **(cigarettes\_per\_day / 20) \* years\_smoking**
* **riesgo\_diabetes**- Evaluación del riesgo de diabetes basada en los niveles de glucosa y HbA1c

**Capa de servicios y lógica empresarial**

El backend implementa una capa de servicios integral que encapsula una lógica empresarial compleja para la generación de predicciones, análisis de riesgos y sistemas de recomendación.

**Arquitectura de servicios**

| **Servicio** | **Propósito** | **Métodos clave** |
| --- | --- | --- |
| **PredictionService** | Orquestación de predicción de ML | **get\_prediction()**, **batch\_predict()**, **\_prepare\_features()** |
| **CardiovascularPredictor** | ML avanzado y predicción basada en reglas | **predict\_cardiovascular\_risk()**, **\_ml\_prediction()**, **\_rule\_based\_prediction()** |
| **CardiovascularMLService** | Inferencia de modelos ML e ingeniería de características | **predict()**, **\_prepare\_features()**, **\_ml\_prediction()** |

**Análisis de riesgos y recomendaciones**

La capa de servicios implementa sofisticados algoritmos de análisis de riesgos:

*# Risk level determination based on multiclass prediction*

**def** **\_determine\_risk\_level\_multiclass**(predicted\_class):

**if** predicted\_class == 0:

**return** 'Bajo'

**elif** predicted\_class == 1:

**return** 'Medio'

**else**:

**return** 'Alto'

El motor de recomendaciones genera asesoramiento médico personalizado en función de los factores de riesgo:

* **Alto riesgo**: Consulta cardiológica inmediata, evaluación cardíaca integral
* **Riesgo medio**: Monitoreo médico regular, modificaciones en el estilo de vida
* **Bajo riesgo**: Atención preventiva, mantenimiento de un estilo de vida saludable

**Almacenamiento en caché y optimización del rendimiento**

El backend implementa múltiples estrategias de optimización del rendimiento para garantizar una operación eficiente bajo alta carga.

**Integración de Redis**

El sistema utiliza Redis para múltiples escenarios de almacenamiento en caché:

* **Almacenamiento en caché de predicciones** - Almacena resultados de predicción para evitar cálculos de ML redundantes
* **Métricas de rendimiento del modelo** - Almacena en caché estadísticas de modelos a las que se accede con frecuencia
* **Gestión de sesiones** - Maneja el estado de autenticación del usuario

**Optimización de bases de datos**

Los modelos incluyen índices de bases de datos estratégicas:

**class** **Meta**:

indexes = [

models.Index(fields=['numero\_historia']),

models.Index(fields=['external\_patient\_id']),

models.Index(fields=['created\_at']),

]

La optimización de consultas incluye:

* **Optimización de subconsultas** en **PatientViewSet.for\_prediction()** para un procesamiento eficiente por lotes
* **Estrategias de precarga** para registros médicos relacionados y predicciones
* **Paginación** para grandes conjuntos de resultados

**Pila de tecnología**

El backend aprovecha una moderna pila de tecnología basada en Python optimizada para aprendizaje automático y API web de alto rendimiento:

| **Componente** | **Tecnología** | **Versión** | **Propósito** |
| --- | --- | --- | --- |
| **Marco** | Django | 3.2.24 | Marco web central |
| **API** | Marco REST de Django | 3.14.0 | Implementación de API REST |
| **Autenticación** | djangorestframework-simplejwt | 5.3.0 | Gestión de tokens JWT |
| **Base de datos** | PostgreSQL | vía psycopg2-binary 2.9.9 | Almacenamiento de datos primarios |
| **Almacenamiento en caché** | Redis | 5.0.1 | Optimización del rendimiento |
| **Marco ML** | scikit-learn | 1.3.2 | Canalización de aprendizaje automático |
| **Procesamiento de datos** | pandas, numpy | 2.0.3, 1.26.3 | Manipulación de datos |
| **Cola de tareas** | Apio | 5.3.6 | Procesamiento de fondo |
| **Almacenamiento** | AWS S3 | vía boto3 1.34.14 | Almacenamiento de archivos |

**Comunicación API**

Este documento cubre la capa de comunicación API del sistema CardioPredict, incluida la configuración del cliente HTTP, los flujos de autenticación, el manejo de errores y los interceptores de solicitudes/respuestas. La capa de comunicación API sirve como puente entre el frontend Next.js y el backend Django, administrando tokens de autenticación, lógica de reintento de solicitudes y manejo de errores.

Para obtener información sobre puntos finales de API específicos y sus estructuras de datos, consulte [Puntos finales de la API](https://deepwiki.com/JeffCode2022/Tesis/4.1-api-endpoints). Para obtener detalles del sistema de autenticación, consulte [Sistema de autenticación](https://deepwiki.com/JeffCode2022/Tesis/3.1-authentication-system).

**Configuración del cliente API**

El sistema utiliza dos configuraciones principales de cliente API: un cliente principal en **api.ts** y un cliente especializado en **services/api.ts**. Ambos están construidos sobre el **axios** biblioteca y proporcionar diferentes niveles de funcionalidad.

**Cliente API principal**

Diagrama

El contenido generado por IA puede ser incorrecto.El cliente API principal está configurado en [frontend/lib/services/api.ts58-66](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/lib/services/api.ts" \l "L58-L66" \t "_blank)con configuración base:

El cliente determina automáticamente la URL de la API utilizando **process.env.NEXT\_PUBLIC\_API\_URL** con una alternativa a **http://localhost:8000** para el desarrollo local.

**Solicitar interceptores**

Los interceptores de solicitudes manejan tokens de autenticación y protección CSRF:

Diagrama

El contenido generado por IA puede ser incorrecto.

El interceptor excluye los puntos finales de autenticación de la conexión automática de tokens y maneja tokens CSRF para métodos HTTP no seguros.

**Flujo de autenticación**

El sistema de autenticación administra tokens JWT con capacidades de actualización automática:

Diagrama

El contenido generado por IA puede ser incorrecto.

**Mecanismo de actualización de tokens**

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.El sistema implementa la actualización automática de tokens cuando los tokens están cerca de expirar:

**Manejo de errores**

La capa de comunicación API implementa un manejo integral de errores para diferentes escenarios de falla:

Interfaz de usuario gráfica, Diagrama

El contenido generado por IA puede ser incorrecto.

**Estructura de respuesta a errores**

El sistema estandariza las respuestas a errores con mensajes estructurados:

| **Tipo de error** | **Código de estado** | **Acción** | **Redirigir** |
| --- | --- | --- | --- |
| Autenticación | 401 | Actualizar token o borrar autenticación | **/login** |
| Solicitud incorrecta | 400 | Mostrar mensaje de error | Ninguno |
| Error del servidor | 500+ | Mostrar mensaje genérico | Ninguno |
| Error de red | N/A | Mostrar mensaje de conectividad | Ninguno |
| Tiempo de espera agotado | N/A | Mostrar mensaje de tiempo de espera | Ninguno |

**Ciclo de solicitud/respuesta**

El ciclo completo de solicitud/respuesta incluye múltiples capas de procesamiento:



**Integración de la capa de servicio**

El cliente API se integra con capas de servicio específicas para diferentes áreas funcionales:

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

**Arquitectura de integración de servicios**

**Configuración y entorno**

La capa de comunicación API se adapta a diferentes entornos mediante configuración:

| **Configuración** | **Variable de entorno** | **Valor predeterminado** | **Propósito** |
| --- | --- | --- | --- |
| URL base de la API | **NEXT\_PUBLIC\_API\_URL** | **http://localhost:8000** | Punto final de la API de backend |
| Solicitar tiempo de espera | Hardcoded | **10000ms** | Tiempo de espera de la solicitud HTTP |
| Intentos de reintento | Hardcoded | **3** | Número de intentos de reintento |
| Límite de velocidad | Hardcoded | **100 req/min** | Limitación de la velocidad de solicitud |

**Desarrollo vs Producción**

El sistema incluye diferentes comportamientos para entornos de desarrollo y producción:

* Desarrollo: Registro detallado de errores y registro de solicitudes/respuestas
* Producción: Registro mínimo con mensajes de error fáciles de usar
* Las cookies CSRF y de autenticación utilizan indicadores seguros en producción

**Sistemas frontend**

**Propósito y alcance**

Este documento proporciona documentación completa de los sistemas frontend en la aplicación CardioPredict. Cubre la arquitectura de interfaz de usuario basada en Next.js, la organización de componentes, la gestión de estados y los patrones de integración de servicios frontend.

Para obtener información sobre los puntos finales de la API de backend y los modelos de datos, consulte [Sistemas backend](https://deepwiki.com/JeffCode2022/Tesis/4-backend-systems). Para obtener detalles sobre los protocolos de comunicación frontend-backend, consulte [Comunicación API](https://deepwiki.com/JeffCode2022/Tesis/2.3-api-communication). Para obtener detalles de implementación específicos de subsistemas frontend individuales, consulte [Sistema de autenticación](https://deepwiki.com/JeffCode2022/Tesis/3.1-authentication-system), [Sistema de panel de control](https://deepwiki.com/JeffCode2022/Tesis/3.2-dashboard-system), [Sistema de gestión de pacientes](https://deepwiki.com/JeffCode2022/Tesis/3.3-patient-management-system), [Sistema de predicción](https://deepwiki.com/JeffCode2022/Tesis/3.4-prediction-system), y [Sistema de importación de datos](https://deepwiki.com/JeffCode2022/Tesis/3.5-data-import-system).

**Descripción general de la arquitectura frontend**

El frontend se construye utilizando Next.js 14 con TypeScript y sigue una arquitectura basada en componentes con gestión de estado centralizada a través de ganchos React y proveedores de contexto.

**Componentes básicos de la arquitectura**

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

**Sistema de panel de control principal**

El **Dashboard** El componente sirve como orquestador central para toda la aplicación frontend, administrando múltiples subsistemas y sus interacciones.

**Gestión del estado del panel de control**

El panel mantiene varios objetos de estado clave:

| **Variable de estado** | **Tipo** | **Propósito** |
| --- | --- | --- |
| **activeSection** | **string** | Controla qué sección se muestra actualmente |
| **metrics** | **DashboardMetrics** | Almacena análisis y datos de rendimiento |
| **formData** | **FormData** | Gestiona los datos de entrada del formulario de predicción |
| **patients** | **Patient[]** | Almacena la lista actual de pacientes |
| **importedPatients** | **Patient[]** | Gestiona datos de pacientes importados |
| **prediction** | **PredictionResult** | Almacena los últimos resultados de predicción |

**Sistema de navegación de secciones**

Diagrama

El contenido generado por IA puede ser incorrecto.El panel utiliza un sistema de navegación basado en secciones controlado por el **activeSection** state:

**Sistema de componentes de encabezado**

El **Header** El componente gestiona la visualización global del estado de la aplicación y las interacciones del usuario:

**interface** HeaderProps {

modelAccuracy: number

notifications?: number

userName?: string

userRole?: string

}

El encabezado muestra la precisión del modelo con indicadores codificados por colores:

* Verde (**CheckCircle**): ≥90% de precisión
* Amarillo (**AlertTriangle**): 75-89% de precisión
* Rojo (**XCircle**): <75% de precisión

**Sistemas de componentes de interfaz de usuario**

**Arquitectura de componentes de lista de pacientes**

El **PatientsList** El componente implementa una interfaz integral de gestión de pacientes con paginación, búsqueda y análisis de riesgos.

Interfaz de usuario gráfica, Diagrama

El contenido generado por IA puede ser incorrecto.

**Sistema de componentes MetricsCards**

El **MetricsCards** El componente muestra indicadores clave de rendimiento con elementos interactivos:

**interface** MetricsData {

total\_patients: number

total\_predictions: number

high\_risk\_count: number

monthly\_growth: number

patients\_history?: number[]

high\_risk\_history?: number[]

accuracy\_history?: number[]

}

El componente representa tres métricas principales:

1. **PACIENTES HOY**: Total de pacientes con **Users** icono
2. **RIESGO ALTO**: Recuento de pacientes de alto riesgo con **Activity** icono
3. **PRECISIÓN IA**: Precisión del modelo con **BarChart3** icono

**Capa de integración de servicios**

**Arquitectura de servicios frontend**

Diagrama

El contenido generado por IA puede ser incorrecto.Los servicios frontend actúan como una capa de abstracción entre los componentes de la interfaz de usuario y las API backend:

**Patrones de gestión estatal**

El panel implementa varios patrones de gestión de estados:

1. **Estado del componente local**: Los componentes individuales administran su propio estado de interfaz de usuario
2. **Perforación de utilería**: Los datos fluyen hacia abajo a través de la jerarquía de componentes
3. **Proveedores de contexto**: Estado de autenticación compartido vía **useAuth** gancho
4. **Estado de la capa de servicio**: Lógica empresarial encapsulada en módulos de servicio

**Arquitectura de flujo de datos**

Diagrama

El contenido generado por IA puede ser incorrecto.

**Sistemas backend**

**Propósito y alcance**

Este documento cubre los sistemas backend de Django que impulsan la plataforma de evaluación de riesgos cardiovasculares CardioPredict. Detalla los puntos finales de la API REST, el proceso de predicción del aprendizaje automático, los modelos de datos y la infraestructura de soporte que permiten al sistema procesar datos médicos y generar evaluaciones de riesgos.

Para obtener información sobre los componentes frontend que consumen estos servicios backend, consulte [Sistemas frontend](https://deepwiki.com/JeffCode2022/Tesis/3-frontend-systems). Para obtener detalles sobre los patrones de comunicación de API y los flujos de autenticación, consulte [Comunicación API](https://deepwiki.com/JeffCode2022/Tesis/2.3-api-communication).

**Descripción general de la arquitectura**

El backend está estructurado como una aplicación Django REST Framework con módulos especializados para el manejo de pacientes, predicción de riesgo cardiovascular y análisis. El sistema admite tanto la inferencia de modelos de aprendizaje automático como alternativas de predicción basadas en reglas.

Una captura de pantalla de una red social

El contenido generado por IA puede ser incorrecto.

**Sistema de puntos finales API**

**Puntos finales de predicción**

El **PredictionViewSet** proporciona la funcionalidad principal de predicción a través de varios puntos finales especializados:

| **Punto final** | **Método** | **Descripción** | **Manejador** |
| --- | --- | --- | --- |
| **/api/predictions/predict/** | POST | Predicción única | **predict()** |
| **/api/predictions/batch\_predict/** | POST | Predicciones por lotes | **batch\_predict()** |
| **/api/predictions/performance\_metrics/** | GET | Métricas del modelo | **performance\_metrics()** |
| **/api/predictions/statistics/** | GET | Estadísticas de predicción | **statistics()** |
| **/api/predictions/clear\_cache/** | POST | Borrar caché | **clear\_cache()** |

Interfaz de usuario gráfica, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

**Puntos finales de la gestión del paciente**

El **PatientViewSet** y **MedicalRecordViewSet** manejar operaciones de datos de pacientes:

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

**Configuración de URL**

El sistema de enrutamiento de URL conecta los puntos finales a los controladores de vista:

Diagrama

El contenido generado por IA puede ser incorrecto.

**Sistema de aprendizaje automático**

**Arquitectura del servicio de predicción**

El sistema ML utiliza una arquitectura en capas con múltiples estrategias de predicción:

Interfaz de usuario gráfica, Aplicación, Sitio web

El contenido generado por IA puede ser incorrecto.

**Canalización de ingeniería de características**

El sistema transforma datos médicos sin procesar en funciones listas para ML:

Captura de pantalla de un mapa

El contenido generado por IA puede ser incorrecto.

**Sistema de predicción cardiovascular**

El sistema de predicción integral admite tanto modelos de aprendizaje automático como alternativas basadas en reglas:

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

**Modelos de datos y serialización**

**Jerarquía del serializador**

El sistema de serialización proporciona diferentes vistas de los mismos datos para varios casos de uso:

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

**Validación y procesamiento de datos**

El sistema implementa una validación y procesamiento integral de datos:

| **Tipo de validación** | **Implementación** | **Ubicación** |
| --- | --- | --- |
| Unicidad del DNI | Restricción de base de datos + lógica de aplicación | [backend/aplicaciones/predicciones/vistas.py81-86](https://github.com/JeffCode2022/Tesis/blob/a84f4817/backend/apps/predictions/views.py#L81-L86) |
| Validación del formato de fecha | **datetime.strptime()** | [backend/aplicaciones/predicciones/vistas.py69-72](https://github.com/JeffCode2022/Tesis/blob/a84f4817/backend/apps/predictions/views.py#L69-L72) |
| Validación de campo requerida | Validación del serializador | [backend/aplicaciones/predicciones/vistas.py111-114](https://github.com/JeffCode2022/Tesis/blob/a84f4817/backend/apps/predictions/views.py#L111-L114) |
| Validación de características | Comprobaciones de NaN/infinito | [backend/aplicaciones/predicciones/servicios.py137-139](https://github.com/JeffCode2022/Tesis/blob/a84f4817/backend/apps/predictions/services.py#L137-L139) |
| Cálculo de edad | Aritmética de fechas | [backend/aplicaciones/predicciones/vistas.py128-136](https://github.com/JeffCode2022/Tesis/blob/a84f4817/backend/apps/predictions/views.py#L128-L136) |

**Relaciones con bases de datos**

Imagen de la pantalla de un computador

El contenido generado por IA puede ser incorrecto.El modelo de datos admite relaciones complejas entre pacientes, registros médicos y predicciones:

**Dependencias y configuración**

**Dependencias de Python**

El backend se basa en varias categorías de dependencia clave:

| **Categoría** | **Dependencias clave** | **Propósito** |
| --- | --- | --- |
| Marco web | **Django==3.2.24**, **djangorestframework==3.14.0** | Marco web central |
| Autenticación | **djangorestframework-simplejwt==5.3.0** | Autenticación de tokens JWT |
| Base de datos | **psycopg2-binary==2.9.9**, **dj-database-url==2.1.0** | Integración con PostgreSQL |
| Almacenamiento en caché | **django-redis==5.4.0**, **redis==5.0.1** | Almacenamiento en caché de Redis |
| Bibliotecas ML | **scikit-learn==1.3.2**, **numpy==1.26.3**, **pandas==2.0.3**, **joblib==1.3.2** | Aprendizaje automático |
| Cola de tareas | **celery==5.3.6** | Procesamiento asincrónico de tareas |
| Almacenamiento en la nube | **boto3==1.34.14**, **django-storages==1.14.2** | Integración de AWS S3 |

**Configuración de carga del modelo**

El sistema ML carga dinámicamente modelos según la configuración:

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

**Puntos finales de la API**

Este documento proporciona documentación completa de todos los puntos finales de la API REST en el sistema CardioPredict. Estos puntos finales manejan el manejo del paciente, predicciones de riesgo cardiovascular, registros médicos y datos analíticos. Para obtener información sobre los sistemas frontend que consumen estas API, consulte [Sistemas frontend](https://deepwiki.com/JeffCode2022/Tesis/3-frontend-systems). Para obtener detalles sobre los modelos de aprendizaje automático subyacentes, consulte [Sistema de aprendizaje automático](https://deepwiki.com/JeffCode2022/Tesis/4.2-machine-learning-system).

**Descripción general**

El backend de CardioPredict expone una API REST creada con Django REST Framework que proporciona puntos finales para:

* **Gestión de pacientes**: Operaciones CRUD para datos de pacientes y registros médicos
* **Servicios de predicción**: Evaluación de riesgos cardiovasculares y predicciones por lotes
* **Analítica**: Métricas del panel y análisis estadístico
* **Rendimiento del modelo**: Monitoreo del modelo ML y métricas de rendimiento

Todos los puntos finales requieren autenticación a través de tokens JWT e implementan controles adecuados de manejo de errores, registro y permisos.

**Arquitectura de API**

Una captura de pantalla de una red social

El contenido generado por IA puede ser incorrecto.

**Puntos finales de la API de predicción**

**Conjunto de vistas de predicción**

El **PredictionViewSet** Proporciona la funcionalidad principal de predicción de aprendizaje automático y admite operaciones CRUD estándar más acciones personalizadas.

**Operaciones base**

| **Método** | **Punto final** | **Descripción** |
| --- | --- | --- |
| GET | **/api/predictions/** | Enumere todas las predicciones con filtrado y paginación |
| POST | **/api/predictions/** | Crea un nuevo registro de predicción |
| GET | **/api/predictions/{id}/** | Recuperar una predicción específica |
| PUT | **/api/predictions/{id}/** | Actualizar una predicción |
| ELIMINAR | **/api/predictions/{id}/** | Eliminar una predicción |

**Acciones personalizadas**

| **Método** | **Punto final** | **Descripción** |
| --- | --- | --- |
| POST | **/api/predictions/predict/** | Generar predicción del riesgo cardiovascular |
| POST | **/api/predictions/batch\_predict/** | Realizar predicciones por lotes |
| GET | **/api/predictions/performance\_metrics/** | Obtenga métricas de rendimiento del modelo |
| POST | **/api/predictions/clear\_cache/** | Borrar caché de predicción (solo administrador) |
| GET | **/api/predictions/statistics/** | Obtenga estadísticas de predicción |

**Solicitud de predicción/flujo de respuesta**

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

**Formato de solicitud de predicción**

El **/predict/** El punto final acepta datos médicos y de pacientes completos:

Imagen que contiene Tabla

El contenido generado por IA puede ser incorrecto.

**Formato de respuesta de predicción**

Texto

El contenido generado por IA puede ser incorrecto.

**Puntos finales de la API de gestión de pacientes**

**Conjunto de vistas del paciente**

El **PatientViewSet** Maneja la gestión de datos de pacientes con operaciones CRUD integrales y acciones especializadas.

**Operaciones base**

| **Método** | **Punto final** | **Descripción** |
| --- | --- | --- |
| GET | **/api/patients/** | Enumere a los pacientes con búsqueda y filtrado |
| POST | **/api/patients/** | Crea un nuevo paciente |
| GET | **/api/patients/{id}/** | Recuperar detalles del paciente |
| PUT | **/api/patients/{id}/** | Actualizar la información del paciente |
| ELIMINAR | **/api/patients/{id}/** | Eliminar paciente de forma suave |

**Acciones personalizadas**

| **Método** | **Punto final** | **Descripción** |
| --- | --- | --- |
| GET | **/api/patients/for-prediction/** | Optimice a los pacientes para la predicción por lotes |
| POST | **/api/patients/search\_by\_dni/** | Buscar paciente por DNI |
| POST | **/api/patients/{id}/add\_medical\_record/** | Agregar historial médico al paciente |
| GET | **/api/patients/{id}/medical\_history/** | Obtenga el historial médico del paciente |
| GET | **/api/patients/statistics/** | Obtenga estadísticas de pacientes |

**Estructura de datos del paciente**

Diagrama

El contenido generado por IA puede ser incorrecto.

**Serializador de predicción de pacientes para pacientes**

El optimizado **/for-prediction/** El punto final devuelve datos del paciente formateados específicamente para predicciones por lotes:

Imagen que contiene Tabla

El contenido generado por IA puede ser incorrecto.

**Puntos finales de la API de registros médicos**

**Conjunto de vistas de registros médicos**

El **MedicalRecordViewSet** gestiona datos de registros médicos con operaciones CRUD completas y capacidades de filtrado.

| **Método** | **Punto final** | **Descripción** |
| --- | --- | --- |
| GET | **/api/medical-records/** | Enumere los registros médicos con filtrado |
| POST | **/api/medical-records/** | Crear nueva historia clínica |
| GET | **/api/medical-records/{id}/** | Recuperar historial médico específico |
| PUT | **/api/medical-records/{id}/** | Actualizar historial médico |
| ELIMINAR | **/api/medical-records/{id}/** | Eliminar historial médico |

**Estructura de la historia clínica**

Texto

El contenido generado por IA puede ser incorrecto.

**Puntos finales de la API de análisis**

**Conjunto de vistas de análisis**

El **AnalyticsViewSet** Proporciona métricas del panel y análisis estadístico.

| **Método** | **Punto final** | **Descripción** |
| --- | --- | --- |
| GET | **/api/analytics/dashboard\_metrics/** | Obtenga métricas completas del panel |

**Respuesta de las métricas del panel**

Texto

El contenido generado por IA puede ser incorrecto.

**Autenticación y permisos**

Todos los puntos finales de API requieren autenticación mediante tokens JWT. El flujo de autenticación sigue este patrón:

Diagrama

El contenido generado por IA puede ser incorrecto.

**Niveles de permiso**

* **Está autenticado**: Requerido para todos los puntos finales
* **Solo personal**: **/performance\_metrics/**, **/clear\_cache/** puntos finales
* **Alcance del usuario**: Los usuarios no pertenecientes al personal solo ven los datos de sus propios pacientes

**Manejo de errores**

Todos los puntos finales implementan un manejo de errores consistente con códigos de estado HTTP apropiados:

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

**Códigos de estado HTTP comunes**

| **Código** | **Descripción** | **Uso** |
| --- | --- | --- |
| 200 | OK | Solicitudes GET, PUT exitosas |
| 201 | Creado | Solicitudes POST exitosas |
| 400 | Solicitud incorrecta | Datos de solicitud no válidos |
| 401 | No autorizado | Autenticación faltante o no válida |
| 403 | Prohibido | Permisos insuficientes |
| 404 | No encontrado | Recurso no encontrado |
| 500 | Error interno del servidor | Errores del lado del servidor |

**Filtrado y paginación**

**Soporte de filtrado**

La mayoría de los puntos finales admiten el filtrado mediante parámetros de consulta:

* **Predicciones**: **riesgo\_nivel**, **patient**, **model\_version**
* **Pacientes**: **sexo**, **hospital**, **medico\_tratante**
* **Registros médicos**: **patient**, **actividad\_fisica**, **antecedentes\_cardiacos**

**Funcionalidad de búsqueda**

Los puntos finales de los pacientes admiten la búsqueda de texto completo en:

* **nombre**, **apellidos**, **numero\_historia**, **email**, **dni**

**Paginación**

La paginación DRF estándar se implementa con desactivación opcional a través de **no\_pagination=true** parámetro de consulta.

**Sistema de aprendizaje automático**

Este documento cubre el sistema de aprendizaje automático de predicción del riesgo cardiovascular, incluida la carga de modelos, el procesamiento de características, la generación de predicciones y los mecanismos de respaldo. El sistema proporciona capacidades inteligentes de evaluación de riesgos para los registros médicos de los pacientes a través de modelos de aprendizaje automático capacitados y pautas clínicas basadas en reglas.

Para obtener información sobre los puntos finales de API que exponen estos servicios de ML, consulte [Puntos finales de la API](https://deepwiki.com/JeffCode2022/Tesis/4.1-api-endpoints). Para obtener detalles sobre los modelos de datos que alimentan este sistema, consulte [Modelos de datos y administración](https://deepwiki.com/JeffCode2022/Tesis/4.3-data-models-and-administration).

**Arquitectura del sistema**

El sistema ML consta de múltiples servicios interconectados que brindan capacidades de predicción del riesgo cardiovascular con sólidos mecanismos de respaldo.

**Carga y gestión de modelos**

El sistema implementa una estrategia de carga de modelos robusta con mecanismos de respaldo cuando los modelos entrenados no están disponibles.

**Tubería de preparación de funciones**

El proceso de preparación de características transforma los datos del paciente y del historial médico en características listas para ML con codificación y normalización adecuadas.

| **Característica** | **Fuente** | **Codificación** | **Descripción** |
| --- | --- | --- | --- |
| **edad** | **patient.fecha\_nacimiento** | Edad calculada | Edad actual en años |
| **imc** | **patient.peso**, **patient.altura** | Cálculo del IMC | Índice de masa corporal |
| **presion\_sistolica** | **medical\_record.presion\_sistolica** | Valor directo | Presión arterial sistólica |
| **presion\_diastolica** | **medical\_record.presion\_diastolica** | Valor directo | Presión arterial diastólica |
| **colesterol** | **medical\_record.colesterol** | Valor directo | Colesterol total |
| **glucosa** | **medical\_record.glucosa** | Valor directo | Nivel de glucosa en sangre |
| **indice\_paquetes** | **medical\_record.cigarrillos\_dia**, **medical\_record.anos\_tabaquismo** | Calculado | Índice de tabaquismo por paquetes-año |
| **actividad\_fisica\_encoded** | **medical\_record.actividad\_fisica** | Mapeo ordinal | Nivel de actividad física (0-3) |
| **sexo\_encoded** | **patient.sexo** | Codificación binaria | Masculino=1, Femenino=0 |
| **antecedentes\_encoded** | **medical\_record.antecedentes\_cardiacos** | Mapeo categórico | Antecedentes familiares (0-1) |

**Análisis de factores de riesgo clínicos**

El sistema analiza múltiples dominios clínicos para identificar factores de riesgo específicos:

| **Dominio** | **Método de análisis** | **Umbrales de riesgo** |
| --- | --- | --- |
| Edad | Puntuación basada en la edad | >65 años (alto), 45-65 años (medio) |
| IMC | Clasificación del estado de peso | ≥30 (obesidad), 25-30 (sobrepeso) |
| Presión arterial | Estadificación de la hipertensión | ≥140/90 (hipertensión), 130-139/80-89 (elevado) |
| Colesterol | Evaluación del perfil lipídico | ≥240 mg/dL (alto), 200-240 mg/dL (límite) |
| Glucosa | Detección de diabetes | ≥126 mg/dL (diabetes), 100-126 mg/dL (prediabetes) |
| Fumar | Cálculo de paquetes-año | >20 paquetes-año (alto), 0-20 paquetes-año (moderado) |
| Actividad física | Codificación del nivel de actividad | Sedentario (alto riesgo), ligero-intenso (bajo riesgo) |
| Historia familiar | Evaluación de riesgos genéticos | Historial positivo (mayor riesgo) |

**Manejo de errores y mecanismos alternativos**

El sistema ML implementa un manejo integral de errores y estrategias alternativas para garantizar la confiabilidad del sistema.

**Estrategia de manejo de errores**

| **Tipo de error** | **Detección** | **Acción de respaldo** |
| --- | --- | --- |
| Error de carga del modelo | Archivo no encontrado o error de joblib | Set **self.model = None**, utilice predicción basada en reglas |
| Error de carga del escalador | Archivo no encontrado o error de joblib | Set **self.scaler = None**, utilice funciones no escaladas |
| Error de preparación de funciones | Valores de NaN/Inf o datos faltantes | Raise **ValueError** con mensaje detallado |
| Error de predicción de ML | Fallo en la predicción del modelo | Retroceda a la predicción basada en reglas |
| Error completo del sistema | Todos los métodos de predicción fallan | Devuelve la predicción predeterminada con un riesgo del 50% |

**Configuración de desarrollo**

Este documento cubre la configuración completa del entorno de desarrollo para el sistema de predicción del riesgo cardiovascular CardioPredict. Incluye instrucciones detalladas para configurar componentes frontend y backend, administrar dependencias y establecer el flujo de trabajo de desarrollo.

Para obtener información sobre la arquitectura general del sistema, consulte [Arquitectura del sistema](https://deepwiki.com/JeffCode2022/Tesis/2-system-architecture). Para configuraciones de implementación específicas, consulte la documentación de implementación futura.

**Prerrequisitos**

El entorno de desarrollo requiere las siguientes dependencias principales:

| **Componente** | **Versión** | **Propósito** |
| --- | --- | --- |
| Nodo.js | 18.x o posterior | Tiempo de ejecución del frontend |
| Bollo | Último | Administrador de paquetes y herramienta de compilación |
| Python | 3.12 | Tiempo de ejecución del backend |
| Django | 5.x | Marco de backend |
| PostgreSQL | 13.x o posterior | Sistema de base de datos |

**Entorno de desarrollo frontend**

**Gestión de paquetes y dependencias**

El frontend utiliza **Bollo** como administrador de paquetes principal, configurado en [frontend/paquete.json1-76](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/package.json" \l "L1-L76" \t "_blank) El proyecto se basa en **Siguiente.js 15.2.4** con **Reaccionar 19.1.0** y **TypeScript 5.8.3**.

Las categorías de dependencia clave incluyen:

**Dependencias del marco central:**

* **next**: 15.2.4 - Marco React
* **react**: 19.1.0 - Biblioteca de interfaz de usuario
* **typescript**: 5.8.3 - Sistema de tipos

**Sistema de componentes de interfaz de usuario:**

* Múltiple **@radix-ui/\*** Paquetes para componentes de interfaz de usuario accesibles
* **tailwindcss**: 3.4.17 - Marco CSS que prioriza las utilidades
* **lucide-react**: 0.454.0 - Sistema de iconos

**Comunicación API:**

* **axios**: 1.10.0 - Cliente HTTP
* **jwt-decode**: 4.0.0 - Manejo de tokens JWT

**Guiones de desarrollo**

El flujo de trabajo de desarrollo utiliza estos scripts npm definidos en

Texto

El contenido generado por IA puede ser incorrecto.

**Configuración de API**

El cliente API frontend está configurado en [frontend/lib/services/api.ts1-201](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/lib/services/api.ts" \l "L1-L201" \t "_blank)con la siguiente configuración:

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

**Entorno de desarrollo de backend**

**Configuración de Django**

El backend se ejecuta en **Django** con la recarga automática del servidor de desarrollo habilitada, como se muestra en los registros en El sistema monitorea los cambios de archivos y se reinicia automáticamente cuando se modifica el código.

**Configuración de la base de datos**

Según la arquitectura del sistema, el backend utiliza PostgreSQL para la persistencia de los datos. La conexión se configura a través de la configuración de la base de datos de Django.

**Dependencias del aprendizaje automático**

El backend integra scikit-learn y bibliotecas de ML relacionadas para la predicción del riesgo cardiovascular, como lo demuestran las extensas importaciones de scipy, sklearn y numpy

**Configuración del entorno**

**Variables de entorno del frontend**

El frontend requiere estas variables de entorno:



**Puertos del servidor de desarrollo**

La configuración de desarrollo utiliza la siguiente asignación de puertos:

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

**Flujo de trabajo de desarrollo**

**Proceso de configuración inicial**

Diagrama

El contenido generado por IA puede ser incorrecto.

**Estructura y seguimiento de archivos**

El entorno de desarrollo monitorea estos directorios clave:

**Monitoreo del frontend:**

* **frontend/app/**- Directorio de aplicaciones Next.js
* **frontend/lib/**- Bibliotecas y servicios de servicios públicos
* **frontend/components/**- Componentes de React

**Monitoreo de backend:**

* El cargador automático de Django observa los directorios de plantillas y los archivos de configuración regional como se muestra en [backend/cardiovascular.log4-17](https://github.com/JeffCode2022/Tesis/blob/a84f4817/backend/cardiovascular.log" \l "L4-L17" \t "_blank)

**Construcción y gestión de activos**

**Proceso de compilación del frontend**

El proceso de compilación del frontend utiliza Next.js con la siguiente configuración:

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

**Configuración CSS**

El proyecto utiliza Tailwind CSS con variables CSS personalizadas para la temática, configuradas en [frontend/app/globals.css1-95](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/app/globals.css" \l "L1-L95" \t "_blank)La configuración incluye:

* Directivas básicas de Tailwind
* Variables CSS personalizadas para temas claros/oscuros
* Variables de estilo específicas de la barra lateral

**Autenticación y configuración de seguridad**

**Gestión de tokens JWT**

El cliente API maneja la autenticación JWT con lógica de actualización automática de tokens [frontend/lib/services/api.ts69-136](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/lib/services/api.ts" \l "L69-L136" \t "_blank)La configuración incluye:

* **Almacenamiento de tokens**: Almacenamiento local del navegador/Almacenamiento de sesión
* **Validación de tokens**: Comprobación automática de caducidad
* **Actualización de tokens**: Actualización automática en respuestas 401
* **Protección CSRF**: Encabezado X-CSRFToken para solicitudes que no son GET

**Encabezados de seguridad**

La instancia de axios está configurada con encabezados centrados en la seguridad:

* **Content-Type: application/json**
* **Accept: application/json**
* **withCredentials: true**para autenticación basada en cookies

Fuentes: [frontend/lib/services/api.ts58-66](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/lib/services/api.ts#L58-L66) [frontend/lib/services/api.ts139-198](https://github.com/JeffCode2022/Tesis/blob/a84f4817/frontend/lib/services/api.ts#L139-L198)