**Documento de Documentación de API**Proyecto: Backend de Prueba Técnica  
Fecha: 20/08/2025  
Autor: Adolfo Kaiser

# Resumen

Este proyecto consiste en una API desarrollada con FastAPI para la gestión de precios de combustibles. Incluye autenticación segura con JWT, gestión de perfiles, CRUD de precios con validaciones y filtros, así como carga de archivos PDF/XLSX para extracción de datos. El servicio está documentado en Swagger y desplegado públicamente, garantizando facilidad de integración, seguridad básica y un flujo de uso mínimo verificable desde la propia documentación.

# Problemática planteada

En la gestión diaria de precios de combustibles (Regular, Premium y Diésel), las empresas deben registrar y actualizar información por estación y fecha para cumplir con reportes internos y regulatorios. Actualmente, este proceso suele realizarse de manera manual o mediante hojas de cálculo y documentos PDF enviados por correo.

Este enfoque presenta múltiples limitaciones:

* **Propensión a errores y duplicidades** por la captura manual.
* **Falta de trazabilidad** sobre qué usuario registró o modificó la información.
* **Dificultad en el control de accesos**, lo que expone la información a usos indebidos.
* **Procesos lentos y poco escalables**, que retrasan la consolidación de datos.
* **Escasa integración** con sistemas de análisis, visualización o automatización.

Estas limitaciones reducen la eficiencia operativa, incrementan riesgos de incumplimiento y dificultan la toma de decisiones basada en datos confiables.

# Objetivo general

Desarrollar e implementar una **API segura, escalable y documentada** para la gestión centralizada de precios de gasolina, que garantice el acceso controlado de usuarios autenticados y facilite la administración de información por estación, fecha y producto.

**Objetivos específicos:**

-**Implementar autenticación y autorización de usuarios**, asegurando que solo perfiles autorizados puedan acceder o modificar la información.

-**Diseñar y exponer endpoints CRUD** (crear, consultar, actualizar y eliminar) para la gestión de precios por estación/fecha/producto.

-**Habilitar funciones de búsqueda y filtrado** con soporte de paginación para optimizar la consulta de grandes volúmenes de datos.

-**Procesar archivos de entrada (PDF/XLSX)** para extraer identificadores clave (como códigos de estación o RFC), reduciendo errores y acelerando la captura.

-**Proporcionar documentación interactiva** mediante Swagger/OpenAPI, que facilite la integración de la API con otros sistemas analíticos o de reporte.

# Requerimientos funcionales y no funcionales

**Requerimientos funcionales**

* **Autenticación y autorización**
  + Registro de usuarios (POST /auth/register) con email, nombre y contraseña.
  + Inicio de sesión (POST /auth/login) con generación de token JWT (access token).
  + Protección de endpoints sensibles (perfil, precios, carga de archivos) mediante token.
* **Gestión de perfil**
  + Consulta de información del usuario autenticado (GET /profile).
  + Actualización de nombre y/o email del perfil (PUT /profile).
* **Gestión de precios de gasolina**
  + Registro de precios (POST /prices) con validaciones de producto (Regular, Premium, Diesel) y valor positivo.
  + Consulta de lista de precios (GET /prices) con filtros por estación, rangos de fecha y búsqueda por texto, además de paginación.
  + Consulta de un precio específico (GET /prices/{id}).
  + Actualización de un precio existente (PUT /prices/{id}).
  + Eliminación de un precio (DELETE /prices/{id}).
* **Carga de archivos**
  + Subida de archivos en formato Excel o PDF (POST /upload).
  + Extracción automática de identificadores relevantes (RFC o códigos de estación) mediante expresiones regulares.
* **Monitoreo básico**
  + Endpoint de salud (GET /health) para verificar el estado del servicio.

**Requerimientos no funcionales**

**-Seguridad**

* Hash de contraseñas con bcrypt.
* Autenticación mediante JWT HS256 con expiración configurable.

**-Disponibilidad y rendimiento**

* API desplegado públicamente en Render.
* Tiempo de respuesta promedio menor a 300 ms en entorno de prueba.

**-Mantenibilidad**

* Código modularizado en routers (auth, profile, prices, upload).
* Validaciones de entrada y salida con Pydantic.

**-Usabilidad**

* Documentación autogenerada con Swagger/OpenAPI, accesible en /docs.

**-Calidad**

* Pruebas de integración básicas (test\_auth.py) que validan el flujo de registro, login y perfil.

**-Compatibilidad**

* Soporte de archivos PDF y XLSX como entradas válidas.
* Exposición de servicios mediante estándares REST para facilitar la integración con otros sistemas.

# Criterios de éxito

**-Disponibilidad y despliegue**

* La API está desplegada públicamente y responde en GET /health con {"status":"ok"}.
* La documentación generada por Swagger/OpenAPI es accesible en /docs sin errores.

**-Seguridad**

* Registro e inicio de sesión funcionales (POST /auth/register y POST /auth/login) devuelven las respuestas esperadas.
* Los endpoints protegidos rechazan solicitudes sin token (401) y aceptan con token Bearer válido.
* Las contraseñas se almacenan en la base de datos con hash bcrypt, nunca en texto plano.

**-Gestión de perfil**

* GET /profile devuelve los datos del usuario autenticado.
* PUT /profile permite actualizar nombre y/o email, validando duplicados y formatos.

**-Gestión de precios**

* **Creación de precios con POST /prices valida:**
  + station\_id
  + date
  + product ∈ {Regular, Premium, Diesel}
  + price > 0
  + currency
* Listado de precios (GET /prices) admite filtros (station\_id, date\_from, date\_to, q) y paginación (page, limit), devolviendo: items, page, limit, total.
* Operaciones de lectura, actualización y eliminación (GET | PUT | DELETE /prices/{price\_id}) responden con códigos correctos (200/201/204/404).
* Cada registro incluye metadatos: created\_by y created\_at.

**-Carga y parsing de archivos**

* POST /upload acepta archivos PDF/XLSX y devuelve un objeto UploadResult con: filename, kind, extracted, candidates.
* Archivos no soportados devuelven 415 Unsupported Media Type.

**-Calidad**

* La prueba integrada test\_auth.py pasa completa (flujo register → login → profile).
* Tiempo de respuesta promedio menor a 300 ms en entorno de prueba para operaciones típicas.
* Código modular (routers: auth, profile, prices, upload) con validaciones en Pydantic.

# Alcance y limitaciones

**Alcance:**

**-Persistencia en memoria**, adecuada para fines de demostración o prueba técnica.

-**Seguridad mediante JWT**, con expiración configurable para el control de sesiones.

-**Validaciones de datos con Pydantic**, utilizando **FastAPI** como framework principal.

-**Procesamiento de archivos PDF/XLSX**, con extracción de patrones comunes (RFC, códigos de estación, etc.).

**Limitaciones:**  
-No se incluye **persistencia en bases de datos reales** (SQLite, PostgreSQL) ni gestión de migraciones.

-No se implementan **roles o permisos avanzados** (ej. administrador vs. operador).

-No se contemplan **procesos en background** ni el **versionado histórico de precios**.