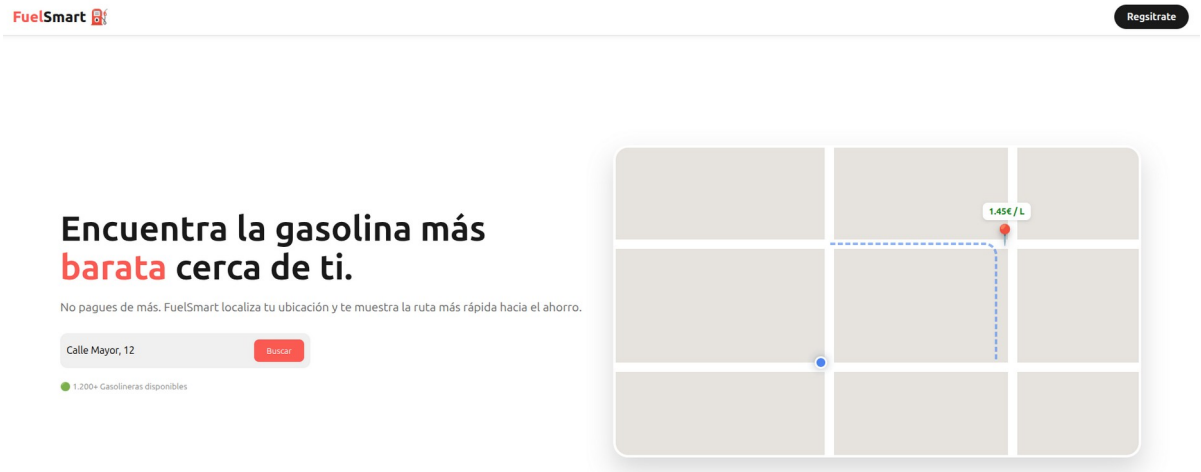
Yulian Radavets Busteroha **2ºDAW**

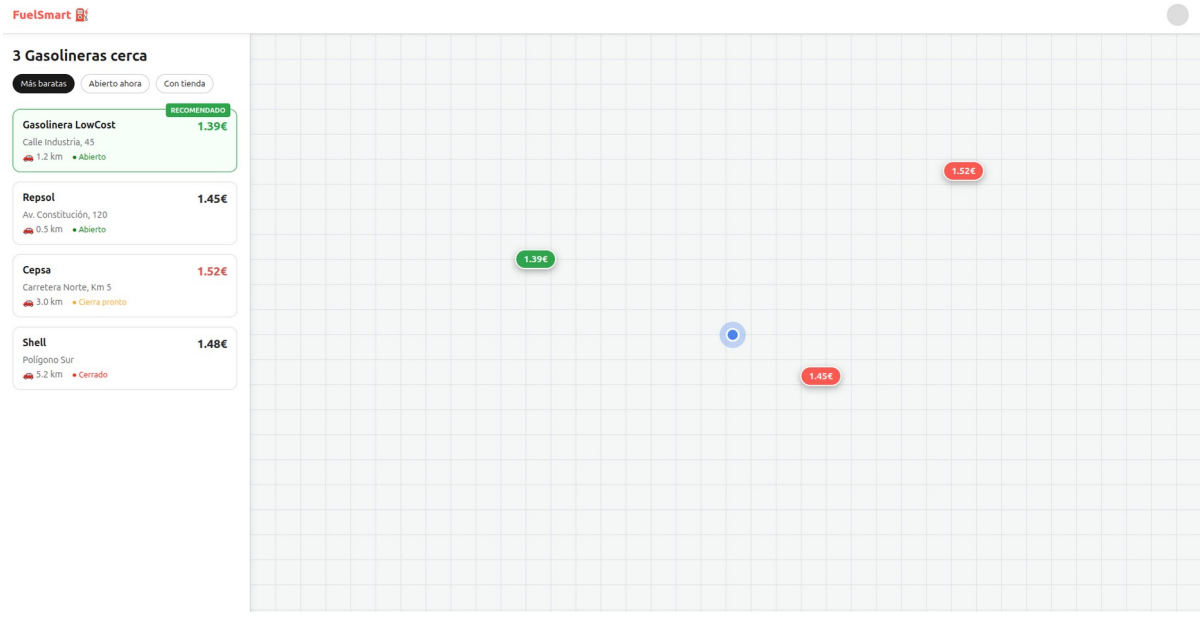
índice

1.Portada

1.1-Landing page



1.2-Pagina a la hora de hacer la búsqueda.



1.3-Nombre del proyecto

FuelSmart

1.4-Ciclo Formativo

Grado Superior en Desarrollo de Aplicaciones Web

1.5-Integrantes

Yulian Radavets Butseroha

1.6-Profesor responsable

Raúl Almarcha Olivares

1.7-Fecha de entrega

18 de febrero de 2026

2. Resumen del proyecto

FuelSmart es una aplicación web Full Stack diseñada para el seguimiento y análisis de precios de combustible. Desarrollada utilizando una arquitectura moderna, separa la lógica de negocio en un backend robusto (Django & PostgreSQL) y la interfaz de usuario en un frontend dinámico (React & TypeScript).

La innovación principal del proyecto reside en su capacidad de almacenamiento histórico: el sistema consume diariamente la API pública del Ministerio para la Transición Ecológica (enfocada inicialmente en la región 30, Murcia, como piloto escalable), guardando los datos para generar gráficas de evolución de precios. Esto permite al usuario visualizar no solo la gasolinera más barata en OpenStreetMap, sino también la tendencia del precio a lo largo del tiempo.

3. Introducción

3.1-Contexto y justificación

En un mercado de hidrocarburos volátil, los consumidores carecen de herramientas que les permitan entender si el precio está subiendo o bajando a medio plazo. Las aplicaciones actuales son "fotos fijas" del momento actual. FuelSmart se justifica por la necesidad de añadir la dimensión "tiempo" a la búsqueda de gasolina, permitiendo un ahorro inteligente basado en datos históricos procesados.

3.2-Objetivos del proyecto

Construir una API REST propia con Django: Que sirva de intermediaria entre los datos del gobierno y nuestra web, añadiendo valor (históricos).

Persistencia de datos en PostgreSQL: Diseñar una base de datos relacional eficiente para almacenar miles de registros de precios diarios sin perder rendimiento.

Frontend interactivo con React y TypeScript: Crear una experiencia de usuario fluida, sin recargas de página (SPA), asegurando el tipado de datos para reducir errores.

Integración de Mapas Open Source: Uso de OpenStreetMap para una visualización geográfica libre de costes de licencia.

3.3-Alcance y limitaciones

Alcance: La aplicación web permite consultar precios actuales y pasados. El sistema de recolección de datos (Scraping/Ingesta) se

centra en la API gubernamental (/FiltroProvincia/30), escalable a nivel nacional.

Limitaciones: La generación de gráficas históricas requiere que el sistema esté operativo y recolectando datos durante un periodo de tiempo (días/semanas) para ser visualmente relevante.

4. Análisis y contextualización de empresa/s del sector

En este apartado se analiza el entorno competitivo y empresarial en el que se enmarca el proyecto FuelSmart. Al tratarse de un proyecto de desarrollo de software, el "sector" analizado es el tecnológico, específicamente el nicho de aplicaciones de servicios al conductor y comparadores de precios.

4.1. Caracterización de empresa/s del sector

Identificación de las empresas representativas del sector: El mercado en el que opera FuelSmart está dominado por empresas tecnológicas de diferente índole que ofrecen servicios de geolocalización y datos.

Podemos clasificarlas en:

Grandes Gigantes Tecnológicos (Big Tech):

Ejemplo: Google (Maps/Waze).

Ubicación: Global (Sede en EEUU, filiales en España).

Tamaño: Multinacional masiva.

Sector: Tecnología y Servicios de Datos.

Portales Web Especializados (Competencia directa):

Ejemplo: Dieselogasolina.com, Komparing, CasaCocheCurro.

Ubicación: Nacional (España).

Tamaño: PyMEs o proyectos unipersonales.

Sector: Medios digitales y comparadores.

Empresas Energéticas (Apps corporativas):

Ejemplo: Waylet (Repsol), Mi BP.

Tipo: Apps de fidelización. Solo muestran sus propios precios.

Análisis del sector económico (tendencias, oportunidades y riesgos):

Tendencias: Existe una clara migración hacia las Web Apps Progresivas (PWA) y el uso de datos en tiempo real. Los usuarios demandan interfaces limpias, rápidas y que funcionen bien en móviles sin necesidad de instalar archivos pesados. El uso de tecnologías como React marca el estándar actual.

Oportunidades: La volatilidad económica y la inflación han creado un consumidor "sensible al precio". Existe una gran oportunidad para herramientas que no solo digan dónde repostar, sino cuándo (análisis histórico), algo que los grandes mapas generalistas no priorizan.

Riesgos: Dependencia de terceros: El funcionamiento depende de la disponibilidad de la API del Gobierno (sedeaplicaciones). Si esta fuente cae, el servicio se detiene.

Justificación de la empresa seleccionada para el proyecto (FuelSmart): Se ha optado por simular la creación de una startup (FuelSmart) en lugar de analizar una gasolinera física, ya que el ciclo formativo es de desarrollo de software. FuelSmart se posiciona como una solución ágil que cubre el hueco entre los mapas (Google) y los listados de texto obsoletos, aportando valor mediante analítica de datos históricos con Django y PostgreSQL.

4.2. Productos y servicios

Identificación de los principales productos y servicios:

Producto Principal: Aplicación Web de consulta de precios de carburantes (FuelSmart Web).

Servicio de Geolocalización: Visualización en mapa (OpenStreetMap) de estaciones de servicio en la ubicación del usuario o en la Región de Murcia (Filtro 30).

Servicio de Analítica de Datos (Valor Agregado): Generación de gráficas de evolución de precios (diario/semanal) gracias al almacenamiento persistente en base de datos propia.

Público objetivo y mercado potencial:

Público General: Conductores particulares que buscan ahorrar en el repostaje diario.

Profesionales del Transporte: Taxistas, repartidores y transportistas autónomos para quienes una diferencia de céntimos supone un gran impacto en sus costes operativos.

Analistas/Curiosos: Usuarios interesados en ver la evolución del mercado energético a través de las gráficas.

Diferenciación con la competencia: Mientras que la competencia (portales web antiguos) suele limitarse a mostrar una tabla estática con el precio del día, FuelSmart se diferencia por su arquitectura moderna y su enfoque histórico:

Tecnología: Uso de React y TypeScript para una experiencia de usuario fluida y sin recargas, frente a webs lentas en PHP antiguo.

Inteligencia de Datos: El Backend en Django guarda el historial. La competencia te dice el precio de hoy; FuelSmart te permite ver si el precio está bajando o subiendo respecto a ayer.

Ética de Datos: Uso de OpenStreetMap (Mapas libres) frente a soluciones privativas.

4.3. Relación con los Objetivos de Desarrollo Sostenible (ODS)

ODS 12: Producción y Consumo Responsables.

ODS 9: Industria, Innovación e Infraestructura.

Justificación de la relación entre los ODS y la actividad de la empresa:

ODS 12 (Meta 12.8): FuelSmart promueve la información y la conciencia sobre los precios de la energía. Al facilitar el acceso transparente a los datos de precios, permite a los usuarios tomar decisiones de consumo más eficientes y responsables, fomentando la competencia leal y el ahorro de recursos económicos.

ODS 9 (Innovación Tecnológica): El proyecto aplica tecnologías modernas de desarrollo web (Stack: Django + React + Postgres) para modernizar el acceso a la información pública gubernamental, creando una infraestructura digital eficiente y accesible.

4.4. Identificación de los riesgos laborales en la empresa

Dado que FuelSmart es una empresa de desarrollo de software, los riesgos laborales no son físicos (como en una gasolinera), sino ergonómicos y psicosociales, típicos del trabajo de oficina y programación.

Análisis de los riesgos asociados a las actividades de la empresa:

Riesgos Ergonómicos:

Trastornos Musculoesqueléticos: Dolores de espalda, cuello (cervicalgia) y muñecas (túnel carpiano) derivados de pasar

largas horas programando en postura estática frente al ordenador.

Medida preventiva: Uso de sillas ergonómicas, reposapiés y pausas activas cada 2 horas.

Riesgos Físicos/Ambientales:

Fatiga Visual: Cansancio ocular, sequedad y visión borrosa por la exposición continua a pantallas (Síndrome Visual Informático).

Medida preventiva: Iluminación adecuada en el área de trabajo y uso de filtros de luz azul o "Modo Oscuro" en el IDE de desarrollo.

Riesgos Psicosociales:

Estrés y Burnout: Presión por los plazos de entrega del software (Deadlines), gestión de errores complejos (bugs) en el código y carga mental elevada.

Medida preventiva: Organización ágil (Trello), respeto de horarios y desconexión digital.

4.5. Conclusiones del análisis

Síntesis de los puntos clave del análisis: El análisis del sector revela que, aunque existe competencia fuerte, hay un nicho desatendido cuanto a Experiencia de Usuario (UX) y Análisis de datos. Las herramientas actuales son funcionales pero visualmente pobres o carecen de profundidad histórica. El proyecto es viable técnicamente gracias a la disponibilidad de APIs públicas y herramientas Open Source.

Propuesta de líneas de acción u oportunidades de mejora detectadas:

Automatización Robusta: Es crucial que el sistema de recolección de datos (Backend Django) sea a prueba de fallos para garantizar que las gráficas históricas no tengan "huecos" vacíos.

Escalabilidad: Se recomienda comenzar con la API filtrada por provincia (Murcia/30) para validar el modelo, pero diseñar la base de datos PostgreSQL preparada para escalar a nivel nacional sin perder rendimiento.

Valor añadido: Enfocarse en la visualización gráfica (Chart.js/Recharts) como principal ventaja competitiva frente a los listados de texto de la competencia.

5. Desarrollo del Proyecto en Trello

5.1. Metodología de trabajo

Para la gestión y desarrollo de FuelSmart, se ha empleado una metodología Ágil basada en Kanban. Este enfoque se ha implementado utilizando la herramienta digital Trello, lo que ha permitido una visualización clara del flujo de trabajo y un control constante del progreso.

El enfoque se ha caracterizado por:

Gestión Visual: Se ha dividido el tablero en cuatro columnas de estado para monitorizar el ciclo de vida de cada tarea:

Backlog (Pila de producto): Repositorio de todas las funcionalidades, historias de usuario y tareas técnicas pendientes de realizar, organizadas por fases.

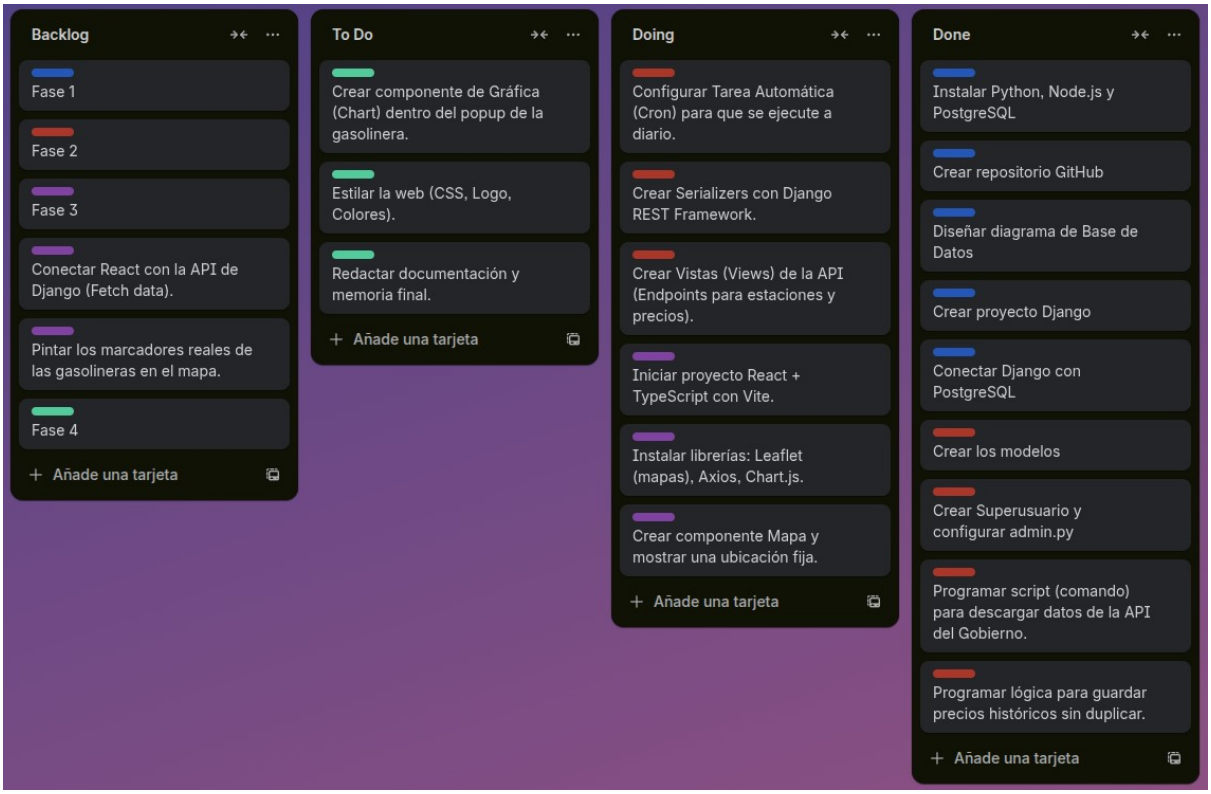
To Do (Por hacer): Tareas seleccionadas para ejecutar en el ciclo actual.

Doing (En proceso): La tarea activa en la que se está trabajando en ese momento (limitando el trabajo en curso para mantener el foco).
Done (Hecho): Tareas finalizadas, revisadas y funcionales.

Organización por Fases (Etiquetas): Se han utilizado etiquetas de colores para identificar visualmente a qué parte de la arquitectura pertenece cada tarea:
Fase 1 (Configuración): Entorno y diseño.
Fase 2 (Backend): Lógica de servidor y Base de Datos.
Fase 3 (Frontend): Interfaz Web y Mapas.
Fase 4 (Finalización): Gráficas y Estilos.

5.2. Temporalización del proyecto

El proyecto se ha planificado con una duración total de 8 semanas, divididas en bloques de trabajo (Sprints) de dos semanas por fase.



5.3. Actividades realizadas

Relación de las principales acciones técnicas ejecutadas durante el desarrollo:

Configuración del Entorno: Instalación y configuración de Python, Node.js y PostgreSQL. Creación del repositorio en GitHub para el control de versiones.

Desarrollo del Backend (Django):

Definición de modelos de datos en models.py.

Programación de un comando de gestión (script) personalizado para descargar, procesar y guardar el JSON de la API del Ministerio.

Implementación de lógica para evitar duplicados y solo añadir nuevos registros de precios históricos.

Creación de Serializadores y Vistas (Views) para exponer los datos en formato JSON.

Desarrollo del Frontend (React):

Creación de la estructura de componentes (Mapa, Buscador, Modal de detalles).

Integración de la librería Leaflet para renderizar el mapa interactivo.

Uso de Axios para realizar peticiones asíncronas al servidor y obtener los datos.

Visualización de Datos:

Implementación de componentes gráficos con Chart.js que renderizan la evolución del precio (Eje X: Fechas, Eje Y: Precio) basándose en el historial almacenado.

5.4. Recursos y tecnologías empleadas

A. Stack Tecnológico (Software):

Backend:

Lenguaje: Python 3.x.

Framework: Django y Django REST Framework (DRF).

Base de Datos: PostgreSQL (por su robustez para series temporales y datos geoespaciales).

Frontend:

Librería: React (versión 18+).

Lenguaje: TypeScript (para tipado estático y reducción de errores).

Empaquetador: Vite.

Mapas: OpenStreetMap y librería React-Leaflet.

Gráficas: Chart.js y react-chartjs-2.

Peticiones HTTP: Axios.

B. Herramientas de Gestión y Desarrollo:

Gestión de Proyecto: Trello (Tablero Kanban).

Control de Versiones: Git y GitHub.

IDE (Entorno de Desarrollo): Visual Studio Code (con extensiones para Python y React).

Cliente de Base de Datos: pgAdmin 4 (para visualizar lastablas).

Pruebas de API: Postman o navegador web.

C. Fuentes de Datos:

API Pública: Servicio REST de Precios de Carburantes (Ministerio para la Transición Ecológica y el Reto Demográfico), endpoint de "Filtro Provincia".

6. Resultados y Análisis

Análisis de los resultados obtenidos y su impacto

Al finalizar el desarrollo del prototipo de FuelSmart, se ha comprobado que la aplicación cumple con lo prometido: descargar datos reales, guardarlos y mostrarlos en un mapa. Al tratarse de un proyecto simulado, los resultados se han evaluado en un entorno de desarrollo, obteniendo las siguientes conclusiones:

6.1. Funcionamiento Técnico (El Backend)

La parte más crítica del proyecto, que era "el robot" (script de Django) encargado de guardar los precios, ha funcionado correctamente en las simulaciones:

Captura de datos: El sistema se conecta con éxito a la API del Gobierno y descarga la lista de gasolineras.

Histórico simulado: Se ha verificado que la base de datos es capaz de diferenciar entre "gasolinera nueva" y "precio nuevo". Al ejecutar el script varios días seguidos (o simulando fechas distintas), la base de datos ha guardado el historial sin duplicar las estaciones, creando correctamente la estructura necesaria para las gráficas.

6.2. Experiencia Visual (El Frontend)

La web desarrollada con React es rápida y fluida.

Mapa Interactivo: A diferencia de otras webs que tardan en cargar, el mapa de OpenStreetMap se mueve con soltura. Los marcadores aparecen en su ubicación exacta gracias a las coordenadas importadas.

Las Gráficas: La librería Chart.js dibuja correctamente la línea de precios al recibir los datos del Backend. Esto confirma que la conexión entre Django (servidor) y React (cliente) funciona bien.

6.3. Impacto Potencial (Si fuera real)

Aunque es una simulación, el análisis de los datos recolectados muestra la utilidad real de la herramienta:

Ahorro inteligente: En las pruebas realizadas con datos de la Región de Murcia, se observan diferencias de precio notables en distancias cortas.

El valor de la gráfica: La función principal del proyecto (ver la evolución histórica) permite al usuario saber si el precio está subiendo o bajando. En un escenario real, esto ayudaría al conductor a decidir si repostar hoy o esperar a mañana, algo que Google Maps no ofrece actualmente.

7.Conclusiones y Recomendaciones

7.1. Conclusiones

El desarrollo del proyecto FuelSmart ha permitido consolidar los conocimientos adquiridos durante el ciclo formativo, integrando diversas tecnologías en una solución Full Stack funcional. Tras la finalización del prototipo simulado, se extraen las siguientes conclusiones principales:

Viabilidad de la Arquitectura Moderna: La separación del proyecto en dos partes diferenciadas (Backend en Django y Frontend

en React) resultado ser una decisión acertada. Ha permitido trabajar la lógica datos y la interfaz de usuario de forma independiente, facilitando la detección de errores y mejorando la escalabilidad del código.

El Valor del Dato Histórico: A diferencia de los comparadores convencionales, se ha comprobado que el verdadero valor añadido reside en la persistencia de datos. La implementación del script de recolección diaria en PostgreSQL ha demostrado que, con una lógica sencilla de automatización, se puede transformar información pública efímera en una herramienta analítica potente (gráficas de tendencias).

Integración de Servicios Externos: El consumo de la API REST del Ministerio ha puesto de manifiesto la importancia de los Datos Abiertos (Open Data). Sin embargo, también ha revelado la necesidad de crear sistemas robustos (nuestro Backend) que "protejan" a la aplicación ante posibles caídas del servidor gubernamental, garantizando que el usuario siempre vea información.

Aprendizaje Técnico: Se ha superado la curva de aprendizaje que supone pasar de lenguajes básicos a frameworks profesionales como Django REST Framework y librerías de tipado estático como TypeScript, acercando la experiencia de desarrollo a un entorno laboral real.

7.2. Recomendaciones para futuros proyectos

Basándonos en la experiencia obtenida durante el desarrollo y las limitaciones detectadas en esta simulación, se proponen las siguientes líneas de mejora para futuras iteraciones o proyectos similares:

Implementación de Docker (Contenerización):

Problema detectado: Configurar Python, Node.js y PostgreSQL en diferentes ordenadores puede dar problemas de versiones.

Recomendación: "Dockerizar" el proyecto. Crear contenedores aislados para el Backend, Frontend y Base de Datos permitiría desplegar la aplicación en cualquier servidor o nube en cuestión de minutos.

Sistema de Caché (Redis):

Problema potencial: A medida que la base de datos crezca con años de precios históricos, las consultas pueden volverse lentas.

Recomendación: Integrar una caché intermedia como Redis. Esto permitiría guardar en la memoria RAM los precios más consultados para no tener que leer el disco duro de la base de datos cada vez que un usuario entra.

Alertas Personalizadas:

Oportunidad: Ya que tenemos los datos, el sistema podría ser proactivo.

Recomendación: Añadir un sistema de usuarios y notificaciones por correo electrónico que avise automáticamente: "La gasolina en tu estación favorita ha bajado de 1.50€ hoy".

Optimización para Móviles (PWA):

Recomendación: Convertir la web actual de React en una Progressive Web App (PWA). Esto permitiría a los usuarios instalarla en su móvil como si fuera una app nativa, e incluso consultar los últimos precios descargados sin conexión a internet.

8. Bibliografía y Webgrafía

A continuación, se detallan las fuentes documentales, recursos técnicos y referencias bibliográficas consultadas para el desarrollo, implementación y análisis del proyecto FuelSmart.

8.1. Fuentes de Datos (Open Data)

Ministerio para la Transición Ecológica y el Reto Demográfico. (2025). Geoportal de Hidrocarburos: Documentación de Servicios REST. Fuente oficial de los datos de precios.

Recuperado de:

https://sedeaplicaciones.minetur.gob.es/ServiciosRESTCarburantes/PreciosCarburantes/help

OpenStreetMap Foundation. (2025). Copyright and License. Datos geográficos y mapas base.
Recuperado de: https://www.openstreetmap.org/copyright

8.2. Documentación Técnica del Backend (Servidor)

Django Software Foundation. (2024). Django Documentation (Versión 5.x). Manual de referencia oficial del framework web.
Recuperado de: https://docs.djangoproject.com/en/stable/

Django REST Framework. (2024). API Guide & Tutorial. Documentación para la construcción de la API RESTful.
Recuperado de: https://www.django-rest-framework.org/

PostgreSQL Global Development Group. (2024). PostgreSQL 16 Documentation. Manual de referencia del sistema de gestión de bases de datos.
Recuperado de: https://www.postgresql.org/docs/

Python Software Foundation. (2024). Python 3.12 Documentation. Referencia del lenguaje de programación.

Recuperado de: https://docs.python.org/3/

8.3. Documentación Técnica del Frontend (Cliente Web)

Meta Platforms, Inc. (2024). React: The library for web and native user interfaces. Documentación oficial y guías de hooks.
Recuperado de: https://react.dev/
Microsoft. (2024). TypeScript Handbook. Guía oficial del lenguaje y tipado estático.
Recuperado de: https://www.typescriptlang.org/docs/
Vite. (2024). Vite Guide: Next Generation Frontend Tooling. Documentación del empaquetador web.
Recuperado de: https://vitejs.dev/guide/
Leaflet. (2023). Leaflet: an open-source JavaScript library for mobile-friendly interactive maps. Documentación de la API de mapas.
Recuperado de: https://leafletjs.com/reference.html
React Leaflet. (2023). React components for Leaflet maps. Integración de mapas en componentes React.
Recuperado de: https://react-leaflet.js.org/
Chart.js. (2024). Chart.js Docs. Librería de visualización de datos y gráficas.
Recuperado de: https://www.chartjs.org/docs/latest/
Axios. (2024). Axios HTTP Client Documentation. Librería para peticiones asíncronas.
Recuperado de: https://axios-http.com/docs/intro

8.4. Metodología y Herramientas de Gestión

Atlassian. (2024). Guía de Kanban: Qué es y cómo usarlo en el desarrollo de software. Referencia metodológica para la gestión del tablero Trello.
Recuperado de: https://www.atlassian.com/es/agile/kanban
Git SCM. (2024). Git Reference Manual. Documentación del sistema de control de versiones.

Recuperado de: https://git-scm.com/doc

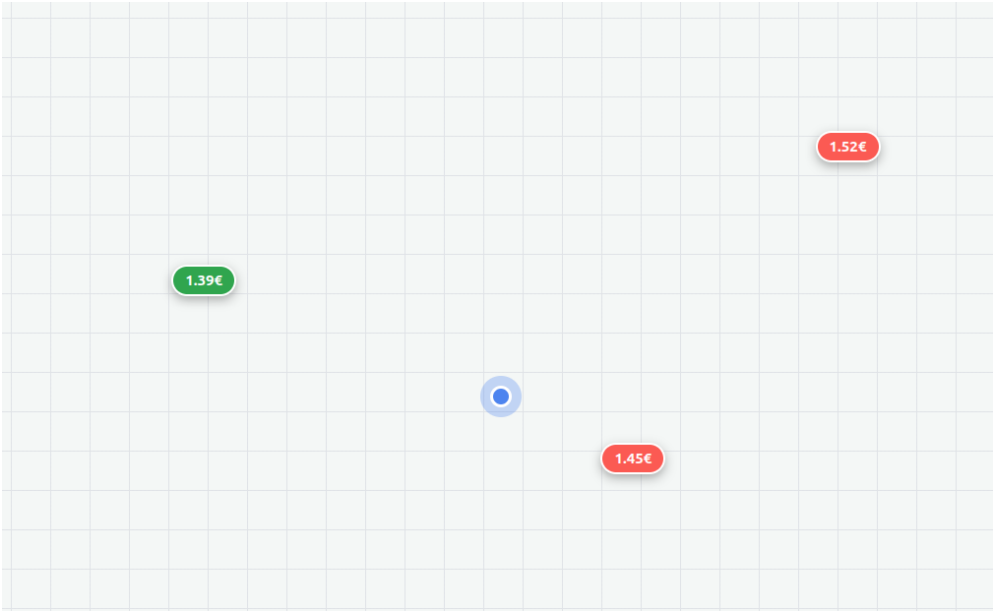
Naciones Unidas. (2015). Objetivos de Desarrollo Sostenible (ODS). Referencia para el análisis de impacto social (ODS 9 y 12). Recuperado de: https://www.un.org/sustainabledevelopment/es/

9. Anexos

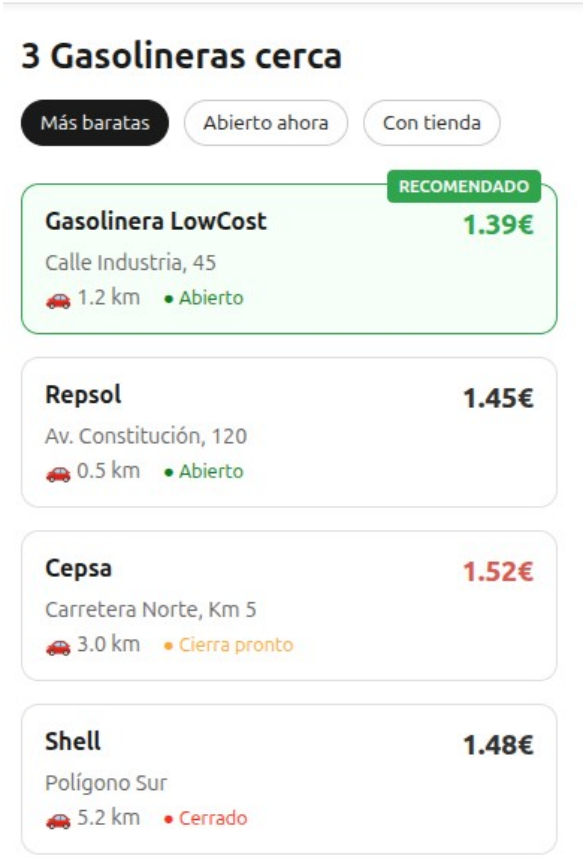
9.1 Evidencias fonográficas y multimedia

Buscador: En esta sección introducimos nuestra ubicación actual para localizar las gasolineras cercanas.

Mapa: Esta parte del mapa indica la ubicación exacta de cada gasolinera.

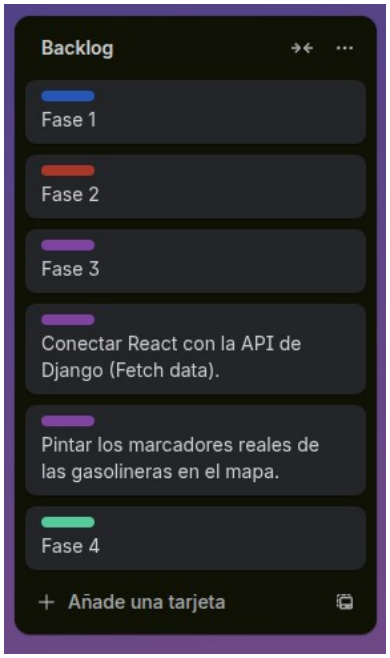


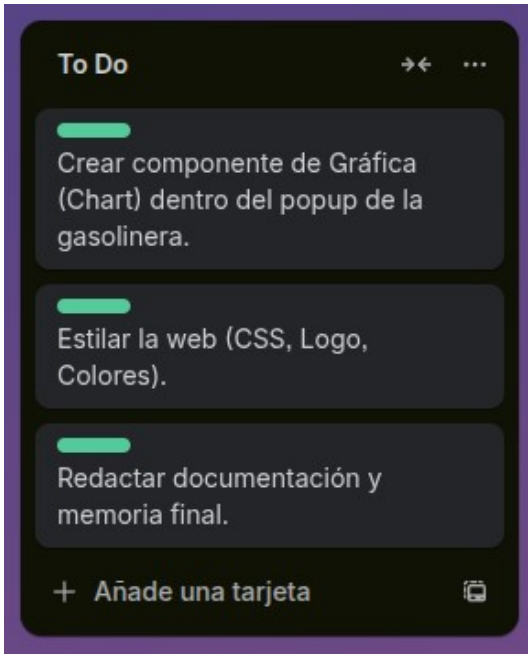
Resultados: Aquí se muestran los establecimientos encontrados, con los precios ordenados de menor a mayor (ascendente).

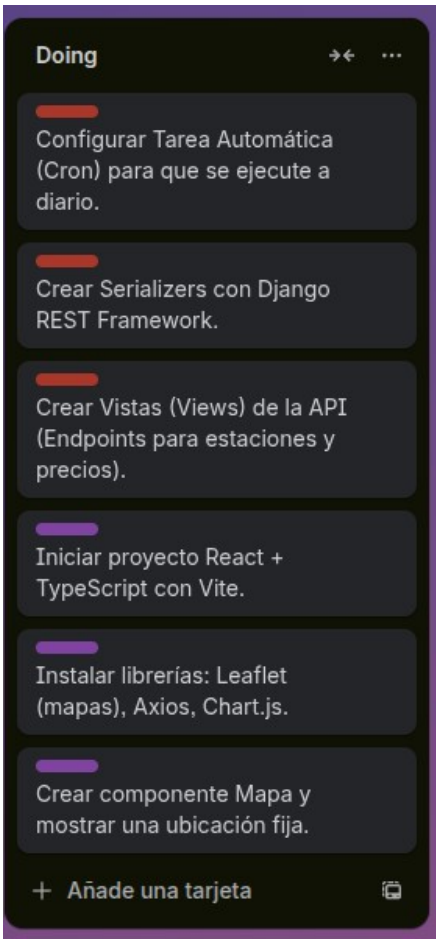


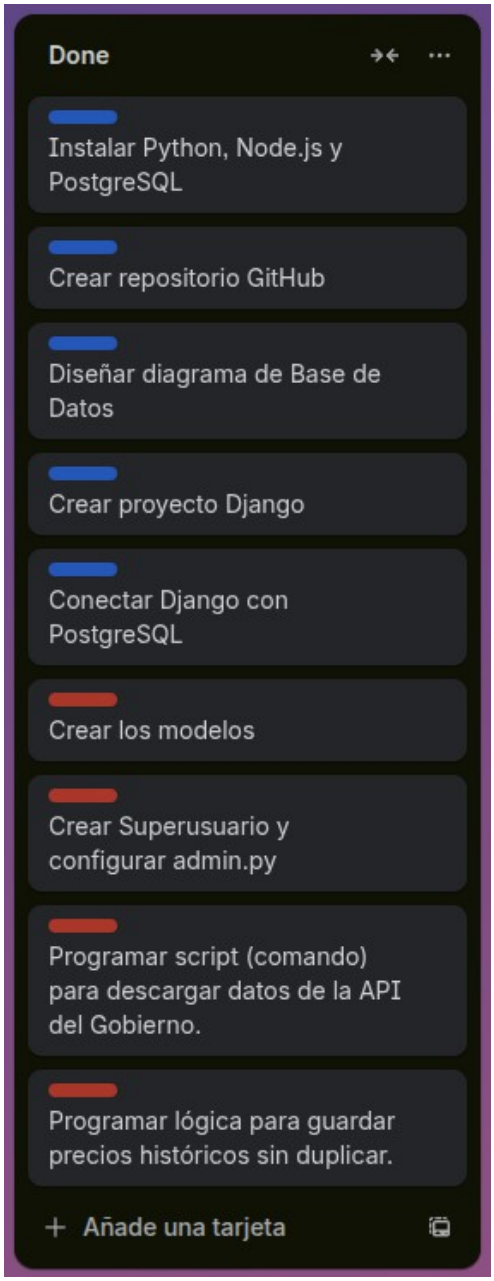
Vista Detallada: Al hacer clic en una gasolinera, se despliega una vista detallada con horarios, métodos de pago, el precio de cada combustible y una gráfica con su evolución histórica.

Organización: Por último, adjunto las capturas de Trello para mostrar la organización de mis tareas.






Doing
Configurar Tarea Automática (Cron) para que se ejecute a diario.
Crear Serializers con Django REST Framework.
Crear Vistas (Views) de la API (Endpoints para estaciones y precios).
Iniciar proyecto React + TypeScript con Vite.
Instalar librerías: Leaflet (mapas), Axios, Chart.js.
Crear componente Mapa y mostrar una ubicación fija.
Añade una tarjeta


Done
Instalar Python, Node.js y PostgreSQL
Crear repositorio GitHub
Diseñar diagrama de Base de Datos
Crear proyecto Django
Conectar Django con PostgreSQL
Crear los modelos
Crear Superusuario y configurar admin.py
Programar script (comando) para descargar datos de la API del Gobierno.
Programar lógica para guardar precios históricos sin duplicar.
Añade una tarjeta