



**Instituto Técnico de Grado Superior en Informática**  
**2024/2025**

## **Desarrollo de Aplicaciones Web**

**JUAN MANUEL GONZÁLEZ DÍAZ**

**Tutor del proyecto:**  
**Mariano Martín**

---

<b>1. Introducción</b>	4
<b>2. Objetivos</b>	5
<b>3. Recursos utilizados</b>	6
3.1 Hardware:	6
3.2 Software:	6
<b>4. Tecnologías</b>	7
4.1 Backend.	7
4.2 Frontend.	8
<b>5. Estructura general del proyecto</b>	9
<b>6. Planificación del Tiempo</b>	10
6.1 Fases de la planificación.	11
<b>7. Estudio preliminar del proyecto</b>	12
7.1 Descripción de la aplicación.	12
7.2 Funcionalidades básicas.	13
7.3 Estimación del coste del sistema.	14
7.3.1 Distribución del tiempo:	15
7.3.2 Simulación de coste profesional.	15
7.3.3 Recursos Humanos.	16
<b>8. Estimación del coste del sistema</b>	17
8.1 Datos.	17
8.1.1 Entidades y Atributos.	17
8.1.2 Relaciones.	18
8.1.3 Traslado a un esquema relacional.	18
8.1.3.1 Normalización.	18
8.1.4 Traslado a un esquema relacional.	19
8.1.5 Restricciones no implementables en el modelo de datos.	19
8.2 Funciones.	20
8.2.1 Diagramas de flujo de datos.	20
8.2.2 Diagramas de flujo de datos.	20
8.2.3 Otros.	21
<b>9. Implementación del sistema físico</b>	22
9.1 Elección justificada del hardware en relación calidad/precio.	22
9.1.1 PC.	22
9.1.2 Impresoras.	23
9.1.3 Red (cableado, dispositivos, etc.).	23
9.1.4 Servidor.	23
9.1.5 Coste total (estimado) del hardware.	24
9.2 Elección justificada de los diferentes sistemas operativos.	24
9.2.1 Servidor.	24
9.2.2 Coste total (estimado) de las licencias.	26
9.3 Elección justificada de un paquete ofimático y otro software necesario.	27

9.3.1 Elección justificada de un paquete ofimático y otro software necesario.	28
<b>10. Planes de Prueba</b> 	<b>29</b>
10.1 Plan de pruebas de caja blanca	29
10.1.1 Objetivos de las pruebas	29
10.1.2 Objetivos de las pruebas	30
10.1.3 Informe de pruebas de caja blanca	30
10.2 Plan de pruebas de caja Negra	32
10.2.1 Registro de usuario	33
10.2.2 Login correcto:	35
10.2.3 Búsqueda Vacía:	36
10.2.4 Sin tiendas seleccionadas:	36
10.2.5 Búsqueda real (ej. “iPhone 16”):	37
10.2.6 Búsqueda sin resultados:	39
<b>11. Manuales</b> 	<b>40</b>
11.1 Manual técnico:	40
11.1.1 Descripción General del Sistema	40
11.1.2 Estructura del Código	41
11.1.3 Detalles de Implementación	41
11.1.4 Dependencias y Librerías	42
11.1.5 Consideraciones de Seguridad	42
11.2 Manual de instalación:	43
11.2.1 Requisitos del sistema:	43
11.2.2 Instalación del backend:	43
11.2.3 Instalación del frontend:	43
11.2.4 Observaciones	43
11.3 Manual de usuario:	44
11.3.1 Acceso inicial:	44
11.3.2 Interfaz de búsqueda	45
11.3.3 Visualización de resultados	46
11.3.4 Otras funciones	47
11.4 Manual de administración:	48
11.4.1 Acceso a la base de datos	48
11.4.2 Logs del sistema	49
11.4.3 Reinicio del scraper	50
11.4.4 Mantenimiento del entorno	50
<b>12. Conclusiones finales</b> 	<b>51</b>
12.1 Grado de consecución de objetivos finales:	51
12.2 Posibles mejoras o ampliaciones por implementar en el futuro:	52
12.3 Análisis del Aprendizaje y Experiencia:	53
<b>13. Bibliografía</b> 	<b>54</b>
<b>14. PRESENTACIÓN</b> 	<b>55</b>

# 1. Introducción



## IndexPrice

Tecnología que compara, inteligencia que ahorra.

En un entorno digital cada vez más orientado al comercio electrónico, la comparación de precios y productos entre distintas tiendas online se ha convertido en **una necesidad para los consumidores**. Este Trabajo de Fin de Grado presenta el desarrollo de una plataforma web que permite realizar **búsquedas automatizadas** de productos con precios indexados desde **múltiples sitios web**, utilizando técnicas de recolección de información como *Google Hacking*, *filtrado selectivo* y *parseo de contenido HTML*.

El sistema desarrollado integra un **buscador avanzado que permite aplicar distintos filtros** (por tienda, rango de precios, palabras clave, etc.), con el objetivo de **ofrecer resultados relevantes y personalizados**. Los datos extraídos se presentan al usuario mediante una **interfaz intuitiva** basada en tarjetas visuales, donde se muestra información clave de cada producto: nombre, imagen, precio, tienda de origen, enlace directo y foto.

Además, se ha implementado un sistema de gestión de usuarios con operaciones **CRUD** (crear, leer, actualizar y eliminar).

El proyecto combina aspectos técnicos como el **web scraping no intrusivo** a través de motores de búsqueda (en particular mediante operadores avanzados como site:"amazon.co.uk" ), el análisis estructurado del DOM en HTML, y el diseño de interfaces modernas orientadas a la experiencia de usuario. Todo ello se ha desarrollado siguiendo **buenas prácticas** de ingeniería de software, priorizando la modularidad, escalabilidad y mantenimiento del sistema.

## 2. Objetivos



El objetivo principal de este **Trabajo de Fin de Grado** es desarrollar una aplicación web que permita a los usuarios comparar productos y precios entre diferentes tiendas en línea de **forma automática** y eficiente. A partir de este objetivo general, se definen los siguientes objetivos específicos:

- Implementar un sistema de **búsqueda inteligente** de productos a través de Google utilizando **técnicas** de Google Hacking.
- Diseñar e implementar un sistema de **filtrado y parseo** de resultados HTML para extraer **información útil** de cada producto.
- Desarrollar un sistema de gestión de usuarios (**CRUD**) que permita el registro, autenticación y administración de cuentas.
- Crear una interfaz **web intuitiva** para mostrar los resultados en forma de tarjetas informativas.
- Garantizar la **escalabilidad y facilidad de mantenimiento** del sistema mediante una arquitectura modular.

### 3. Recursos utilizados



#### 3.1 Hardware:

Equipo	CPU	RAM	Espacio
PC	i5-12400 k	32 gb	2 tb
portatil	i3- 11400	8 gb	256 gb

#### Elementos de red:

Red doméstica con router de fibra 600 Mbps

#### 3.2 Software:

- Sistema Operativo: Windows 10
- Entorno de Desarrollo Integrado (IDE): Visual Code Studio
- Gestión de Versiones: Git Bash
- Herramientas de Diseño Gráfico: Canva Pro, watermarkly, LucidChart
- Sistema cliente: Google, Bing, DuckDuckGo
- Base de datos: SQLite 3

#### Criterios de elección:

- **Relación calidad/precio:** herramientas de código abierto y gratuitas.
- **Compatibilidad:** alta interoperabilidad entre tecnologías web modernas.
- **Facilidad de mantenimiento:** arquitectura clara y modular para futuras mejoras.

## 4. Tecnologías

### 4.1 Backend.

**Python + Flask:** Framework ligero y flexible, ideal para construir APIs rápidas y limpias.

**Flask-SQLAlchemy:** ORM que simplifica la interacción con la base de datos.

**Flask-Migrate:** Herramienta de migraciones para mantener la estructura de la base de datos.

**Selenium + WebDriver Manager:** Para el scraping real desde buscadores, con control del navegador.

**BeautifulSoup4:** Para parsear los resultados HTML de manera eficiente.

**SQLite:** Base de datos ligera, ideal para proyectos individuales y prototipos.

## 4.2 Frontend.

- **Astro + HTML + CSS personalizado:** Permite estructurar componentes reutilizables sin complejidad excesiva.
- **JavaScript Vanilla:** Para manejar eventos, llamadas a la API y mostrar resultados dinámicamente.
- **Tailwind CSS (parcial):** Utilizado para estilo rápido de elementos comunes.

### ¿Por qué estas tecnologías?

Se eligieron herramientas que ofrecen equilibrio entre simplicidad, control y velocidad de desarrollo. Python y Flask permiten crear una API robusta con poco código. Astro facilita mantener una interfaz modular. Las herramientas de scraping elegidas permiten emular un comportamiento de usuario real, útil cuando no se dispone de una API oficial.

## 5. Estructura general del proyecto



```
📦 TFG/
  └── backend/
    ├── app.py
    ├── config.py
    ├── extensions.py
    ├── requirements.txt
    ├── modelos/
    ├── rutas/
    ├── utils/
    └── migrations/
  └── frontend/
    ├── public/
    ├── src/
      ├── components/
      ├── pages/
      └── styles/
    └── package.json
  └── instance/
    └── db.sqlite3
  └── logs/
    ├── scraper.log
    ├── ReiniciarIP.txt
    └── run_backend.txt
└── README.md
```

# 6. Planificación del Tiempo



## 1. Análisis de requisitos

Investigación de necesidades, herramientas y alcance funcional.

## 2. Diseño del sistema.

Estructura backend/frontend, base de datos, diseño visual.

## 3. Diseño del sistema

Codificación del backend (API, scraper), luego del frontend (interfaz, lógica JS).

## 4. Pruebas y validación.

Test funcionales, detección de errores, ajustes.

## 5. Documentación y entrega.

Manuales, README, estructura del código comentada.

## 6.1 Fases de la planificación.

### ◆ **Fase 1 – Planificación y diseño (Días 1-8)**

- Definición de requisitos
- Búsqueda de tecnologías adecuadas
- Diseño de la arquitectura del sistema
- Mockups de interfaz

### ◆ **Fase 2 – Desarrollo del Backend (Días 8-30)**

- Configuración del servidor Flask
- Implementación de modelos y rutas
- Lógica del scraper con Selenium
- Manejo de errores y captchas
- Pruebas de endpoints y resultados

### ◆ **Fase 3 – Desarrollo del Frontend (Días 30-40)**

- Estructuración con Astro y componentes
- Integración de Tailwind y estilos personalizados
- Conexión con la API
- Interacción: loader, filtros, respuestas

### ◆ **Fase 4 – Documentación y pruebas (Días 40-45)**

- Documentación técnica y de usuario
- Pruebas funcionales y de flujo completo
- Redacción del README y guía de despliegue
- Ajustes finales y revisión

**Herramientas de planificación:** Google Sheets, Google Calendar.

# 7. Estudio preliminar del proyecto



## 7.1 Descripción de la aplicación.

El proyecto consiste en el desarrollo de una aplicación web denominada **IndexPrice**, cuyo propósito principal es permitir a los usuarios realizar búsquedas de productos en diversas tiendas en línea y comparar sus precios de forma automática y centralizada. Para ello, el sistema emplea técnicas de scraping combinadas con motores de búsqueda públicos como Google, Bing y DuckDuckGo, extrayendo los datos directamente de los resultados que aparecen indexados en dichos motores.

La aplicación está organizada en dos bloques principales:

- **Backend:** desarrollado en Python con Flask, se encarga de gestionar las rutas API, el scraping dinámico con Selenium, la conexión con la base de datos SQLite y la lógica de filtrado por precio o tienda.
- **Frontend:** desarrollado con Astro, HTML y JavaScript, presenta al usuario una interfaz moderna, responsive y clara, desde la cual puede realizar búsquedas, aplicar filtros y visualizar los resultados.

El sistema no depende de APIs comerciales externas de tiendas, lo cual lo hace más flexible, aunque también implica la necesidad de gestionar correctamente posibles errores como cambios de estructura en las webs indexadas.

## 7.2 Funcionalidades básicas.

Las funcionalidades esenciales implementadas en el sistema son las siguientes:

### Usuario:

- Registro de cuenta con nombre, email y contraseña.
- Inicio de sesión persistente mediante almacenamiento local.

### Búsqueda:

- Introducción de un término de búsqueda (producto).
- Selección de una o más tiendas específicas para limitar los resultados.
- Filtros opcionales por **precio mínimo** y **precio máximo**.
- Selección del **motor de búsqueda** preferido (Google, Bing o DuckDuckGo).
- Ejecución del scraping con detección automática de errores.

### Resultados:

- Visualización de productos encontrados con:
  - Título
  - Fragmento descriptivo
  - Precio (si está disponible)
  - Imagen representativa (si se encuentra)
  - Enlace directo a la tienda
- Loader animado durante el proceso de scraping.
- Mensaje de alerta si no se encuentran resultados o si el scraper ha sido bloqueado.

## **Extras:**

- Página de ayuda con preguntas frecuentes (FAQ).
- Interfaz adaptada a fondo oscuro con diseño responsivo.
- Sistema modular que permite futuras ampliaciones (historial, favoritos, más motores).

## **7.3 Estimación del coste del sistema.**

Teniendo en cuenta que el proyecto se ha desarrollado durante **1 mes y medio (6 semanas)** de forma simultánea a las prácticas, se ha llevado un registro estimado del tiempo invertido, distribuyendo las horas reales dedicadas según cada etapa del desarrollo. El cálculo del coste se basa exclusivamente en las **horas trabajadas**, no en el tiempo total calendario.

### **Datos generales:**

- **Días trabajados:** 35
- **Horas por día dedicadas al proyecto:** 4 horas y 15 minutos
- **Total de horas trabajadas:** 148,75 horas (aproximadamente 149 h)

### 7.3.1 Distribución del tiempo:

Actividad	Horas Invertidas
Investigación (Frameworks, APIs, scraping)	10 h
Desarrollo interfaz gráfica (UI/UX, Astro + CSS)	25 h
Creación de recursos visuales (imágenes, loader)	5 h
Periodo sin progreso (pruebas fallidas, bloqueos)	18 h
Desarrollo funcional (backend + scraping estable)	75 h
Corrección de errores y bugs	15 h
Modelado y ajustes de base de datos	2 h
<b>Total estimado</b>	<b>149 h</b>

### 7.3.2 Simulación de coste profesional.

Partiendo del salario medio estimado en España para un programador junior (13 €/hora), se obtiene:

- Coste por hora: 13 €
- Total aproximado para 149 h: 1.937 €

Este valor representa el coste de la primera fase del proyecto (funcionalidad básica, interfaz, scraper y pruebas). Se estima que una segunda fase completa —incluyendo mejora de rendimiento, escalabilidad, seguridad y diseño avanzado— elevaría el coste a unos 3.800 – 4.000 € en su versión final y pulida.

Por tanto, el proyecto es asumible en 1 mes y medio de dedicación intensiva y se encuentra, en esta entrega, en un estado completamente funcional y demostrable.

### 7.3.3 Recursos Humanos.

Para la realización del presente proyecto se ha contado con la participación de un único desarrollador, quien ha asumido todas las fases del desarrollo, desde el análisis y la planificación inicial hasta la programación, pruebas, documentación y presentación final.

Rol del desarrollador:

- Diseño de la arquitectura general del sistema, tanto del backend como del frontend.
- Implementación del backend usando Python, Flask, SQLAlchemy y el scraper con Selenium.
- Diseño e implementación del frontend, incluyendo componentes en Astro, estilos en Tailwind CSS y JavaScript para la interacción.
- Modelado de base de datos, creación de los modelos y migraciones necesarias en SQLite.
- Integración completa entre frontend y backend mediante una API REST.
- Pruebas manuales y funcionales del sistema.
- Documentación técnica del código y elaboración de manuales.
- Redacción del informe final y estructuración para su presentación académica.

Este enfoque integral ha permitido al desarrollador aplicar de forma práctica los conocimientos adquiridos durante el ciclo formativo en Desarrollo de Aplicaciones Multiplataforma, además de desarrollar habilidades transversales como la autogestión, la planificación personal del tiempo, la solución autónoma de errores y la toma de decisiones tecnológicas.

A pesar de tratarse de un proyecto individual, la complejidad de las tecnologías empleadas ha requerido una alta capacidad de

adaptación, investigación y aprendizaje continuo por parte del alumno.

## 8. Estimación del coste del sistema

### 8.1 Datos.

El sistema utiliza una base de datos ligera y local basada en **SQLite 3**, ideal para entornos de desarrollo, pruebas y despliegues simples. En la versión actual del sistema, **únicamente se persisten los datos de los usuarios registrados**. Los resultados de las búsquedas se generan dinámicamente en memoria mediante scraping y **no se almacenan permanentemente**.

#### 8.1.1 Entidades y Atributos.

##### Entidad usuarios (persistente)

Contiene la información básica de autenticación del usuario:

- **id**: Identificador único (clave primaria)
- **nombre**: Nombre del usuario
- **email**: Correo electrónico (único)
- **contraseña**: Contraseña cifrada (hash recomendado)

##### Entidad lógica producto (no persistente)

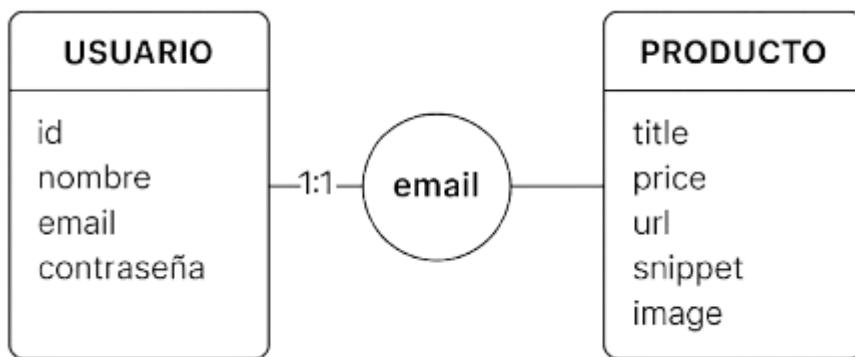
Representa los productos obtenidos dinámicamente a través del scraper, en tiempo de ejecución:

- **title**: Título del producto
- **price**: Precio extraído (formato flotante con símbolo €)
- **url**: Enlace al producto
- **snippet**: Descripción corta del resultado
- **image**: Enlace a la imagen del producto (opcional)

## 8.1.2 Relaciones.

Dado que no se guarda el historial de productos ni las búsquedas realizadas, la única relación directa en la base de datos es la de los usuarios registrados:

- **Relación uno a uno entre email y usuario:** cada dirección de correo pertenece a un único usuario.
- **No existen claves foráneas ni relaciones múltiples,** dado que los productos son volátiles y no persisten.



## 8.1.3 Traslado a un esquema relacional.

- **usuarios (id, nombre, email, contraseña)**

### 8.1.3.1 Normalización.

La tabla **usuarios** se encuentra en **Tercera Forma Normal (3FN)**:

- No hay atributos repetitivos ni multivaluados.
- Cada campo depende exclusivamente de la clave primaria.
- No existen dependencias transitivas.

### **8.1.4 Traslado a un esquema relacional.**

```
CREATE TABLE usuarios (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nombre TEXT NOT NULL,
    email TEXT NOT NULL UNIQUE,
    contraseña TEXT NOT NULL
);
```

### **8.1.5 Restricciones no implementables en el modelo de datos.**

Existen restricciones que no pueden ser implementadas directamente en la base de datos, por lo que son gestionadas desde la lógica de la aplicación:

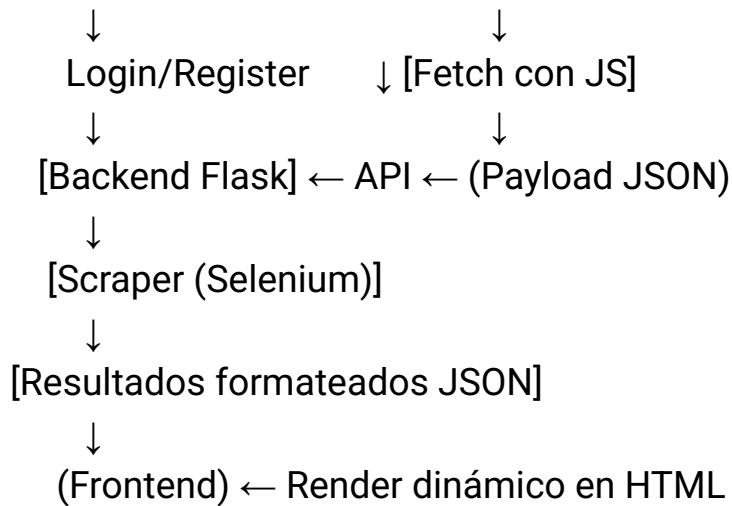
- **Evitar registros duplicados:** se comprueba desde Flask si el correo ya está registrado.
- **Cifrado de contraseñas:** las contraseñas no se almacenan en texto plano.
- **Resultados de búsqueda no persistentes:** por privacidad y simplicidad, los productos obtenidos no se guardan.
- **Acceso restringido:** ciertas funcionalidades del frontend están disponibles solo si el usuario ha iniciado sesión correctamente.

## 8.2 Funciones.

El sistema se divide en **backend (API + scraper)** y **frontend (interfaz de búsqueda)**. Las funciones clave se organizan como sigue:

### 8.2.1 Diagramas de flujo de datos.

[Usuario] → (Frontend) → [Formulario de búsqueda]



### 8.2.2 Diagramas de flujo de datos.

Campo	Tipo	Restricciones
nombre	Texto	Requerido, máx. 100 caracteres
email	Texto	Requerido, único, formato email válido
contraseña	Texto	Requerido, cifrado (hash)
query	Texto	Requerido en búsqueda
minPrice	Float	Opcional, $\geq 0$
maxPrice	Float	Opcional, $\geq \text{minPrice}$
site	Texto	Lista de URLs válidas
engine	Texto	google

### 8.2.3 Otros.

- El sistema no persiste productos por privacidad y por evitar conflictos legales con tiendas.
- Se gestiona visualmente el **loader** con animación mientras se realiza el scraping.
- Los resultados son temporales y se muestran en pantalla pero no se guardan.
- El sistema redirige automáticamente a login si no hay sesión activa.

## 9. Implementación del sistema físico

### 9.1 Elección justificada del hardware en relación calidad/precio.

Dado que el proyecto se ha desarrollado de forma individual y en entorno local, no ha requerido inversión en infraestructura especializada. Aun así, se analiza la elección del hardware utilizado bajo criterios de rendimiento, compatibilidad y coste.

#### 9.1.1 PC.

El desarrollo completo del sistema (frontend y backend) se ha llevado a cabo en un único equipo con las siguientes especificaciones:

- **Modelo:** Montado por mi
- **Procesador:** Intel Core i5-12400
- **RAM:** 32 GB DDR4
- **Almacenamiento:** 2 TB M.2
- **Sistema operativo:** Windows 10 Pro
- **Precio aproximado:** 703€

Justificación: Esta configuración ofrece una excelente relación calidad/precio para tareas de desarrollo web, programación y pruebas de scraping, sin necesidad de equipos de gama alta.

### **9.1.2 Impresoras.**

El proyecto no ha requerido impresión de documentación técnica ni resultados físicos, por lo que no se ha utilizado ninguna impresora.

### **9.1.3 Red (cableado, dispositivos, etc.).**

Toda la infraestructura de red ha sido la doméstica habitual:

- **Conexión:** Fibra óptica 300 Mbps
- **Router:** proporcionado por el operador (Vodafone)
- **Cableado:** conexión por Wi-Fi y ocasionalmente Ethernet

Justificación: Suficiente para pruebas con scraping, envío de peticiones al backend y navegación general. No ha habido necesidad de switches, puntos de acceso adicionales o routers avanzados.

### **9.1.4 Servidor.**

No se ha utilizado un servidor físico dedicado. El backend (Flask) ha sido ejecutado localmente mediante localhost en el mismo equipo del desarrollador. Para una futura versión en producción, se podría migrar fácilmente a un VPS básico como DigitalOcean o Railway.

Justificación: El entorno de desarrollo en local es suficiente para validar la funcionalidad del sistema sin necesidad de inversión adicional.

### 9.1.5 Coste total (estimado) del hardware.

Elemento	Precio estimado
PC desarrollo	703 €
Conectividad/red	incluido
Impresoras	No necesarias
<b>Total</b>	<b>703 €</b>

## 9.2 Elección justificada de los diferentes sistemas operativos.

### 9.2.1 Servidor.

Para el backend del sistema (desarrollado en Flask y pensado para ser ejecutado en entorno local o remoto), es fundamental contar con un sistema operativo que sea **estable, seguro y compatible con entornos de desarrollo web**.

Aunque el proyecto se ejecutó en entorno local, a continuación se presentan dos opciones viables en caso de migración a producción:

## Opción A: Ubuntu Server 20.04 LTS (Long Term Support)

### Ventajas:

- **Estabilidad:** versión con soporte a largo plazo (5 años).
- **Costo:** completamente gratuito y de código abierto.
- **Comunidad:** amplia base de usuarios, documentación y foros de soporte.
- **Rendimiento:** consumo de recursos bajo, ideal para servidores ligeros como Flask.
- **Seguridad:** actualizaciones frecuentes y repositorios oficiales mantenidos.

### Justificación:

Ubuntu Server es una opción excelente para entornos de producción de aplicaciones web modernas. Su bajo consumo de recursos y compatibilidad con Python y herramientas como Gunicorn, Nginx o Docker lo convierten en la **opción preferida si se desea desplegar el buscador de precios de forma estable y escalable**. Además, permite aprovechar servicios como Railway, Render o VPS basados en Linux.

## Opción B: Windows Server 2019

### Ventajas:

- **Compatibilidad:** alta con aplicaciones empresariales y entornos Microsoft.
- **Administración:** incluye herramientas gráficas de gestión como Server Manager.
- **Soporte:** soporte oficial de Microsoft y actualizaciones regulares.

### **Justificación:**

Windows Server puede ser útil si se desea integrar el sistema con otros servicios empresariales basados en Microsoft. Su interfaz gráfica es más accesible para usuarios sin experiencia en entornos Linux. No obstante, **su coste y consumo de recursos** pueden representar un inconveniente frente a alternativas libres como Ubuntu.

### **Decisión final:**

Dado que el proyecto se centra en tecnologías open source y no requiere integración con servicios empresariales de Microsoft, **Ubuntu Server 20.04 LTS** es la opción más adecuada para una posible fase de producción.

### **9.2.2 Coste total (estimado) de las licencias.**

Software	Licencia	Precio estimado
Windows 11 (preinstalado en el equipo)	OEM incluida	0 €
Visual Studio Code	Libre	0 €
Python 3.11	Libre	0 €
Astro / TailwindCSS	Libre	0 €
Git y GitHub	Libre	0 €
Navegadores (Chrome, Bing)	Libre	0 €
Flask	Libre	0 €
<b>Total:</b>		<b>0 €</b>

**Justificación:** Gracias a la amplia oferta de herramientas gratuitas, no ha sido necesario realizar ninguna inversión económica para licencias durante el desarrollo del proyecto.

### **9.3 Elección justificada de un paquete ofimático y otro software necesario.**

Paquete ofimático

- **Google Docs / Google Sheets**

Se ha utilizado el paquete de Google Workspace en su versión gratuita para redactar la memoria, estructurar la planificación (cronogramas) y documentar los resultados.

**Ventajas:**

- Gratuito y accesible desde cualquier navegador.
- Colaborativo, ideal para compartir con tutores o compañeros.
- Compatible con formatos Word, Excel y PDF.

#### **Software complementario utilizado**

Herramienta	Uso principal
Canva	Diseño gráfico para portada y presentaciones.
Figma	Bocetos rápidos del frontend y UI.
Lucidchart	Mockups y visualización web
Python pip	Gestión de dependencias del entorno
NPM	Instalación de dependencias Astro

**Justificación:** Se han utilizado herramientas modernas, accesibles y gratuitas, enfocadas a la productividad en entornos de desarrollo web y documentación académica.

### **9.3.1 Elección justificada de un paquete ofimático y otro software necesario.**

<b>Herramienta / Paquete</b>	<b>Tipo de licencia</b>	<b>Coste estimado</b>
Canva	Gratis	0€
Figma	Gratis	0€
Lucidchart	Gratis	0€
Python pip	Gratis	0€
NPM	Gratis	0€
<b>Total estimado</b>		<b>0€</b>

# 10. Planes de Prueba



Durante el desarrollo del sistema se han realizado diversas pruebas con el objetivo de asegurar la **estabilidad, precisión y funcionalidad** de cada componente de la aplicación. Estas pruebas se clasifican en dos enfoques principales:

- **Caja blanca:** centradas en la lógica interna y el código fuente.
- **Caja negra:** enfocadas en el comportamiento externo del sistema desde el punto de vista del usuario.

## 10.1 Plan de pruebas de caja blanca

Las pruebas de caja blanca consisten en revisar el código fuente del sistema, controlando directamente la ejecución de funciones internas, flujos lógicos, condiciones, estructuras de control y el manejo de excepciones.

Estas pruebas se han aplicado principalmente sobre:

- El **backend** desarrollado en Flask.
- El **scraper**, desarrollado con Selenium y BeautifulSoup.

### 10.1.1 Objetivos de las pruebas

- Verificar el correcto funcionamiento del scraping y la estructura de los datos devueltos.
- Comprobar el tratamiento de excepciones y errores.
- Validar la transformación del precio desde texto a número decimal.
- Asegurar que **no se dupliquen productos** en los resultados.
- Verificar que los filtros de precio actúan correctamente sobre los datos obtenidos.
- Comprobar que el login gestiona correctamente emails no registrados.

## 10.1.2 Objetivos de las pruebas

**Función get\_google\_results():** se verificó que devuelva una lista válida de productos o gestione errores correctamente.

**Filtro por precios (minPrice y maxPrice):** se comprobaron distintos escenarios con valores válidos, vacíos y mal formateados.

**Conversión de precios:** se validó la correcta transformación de formatos como "49,99 €" a **float**.

**Validación de login:** se comprobó la respuesta adecuada al intentar iniciar sesión con un correo no registrado.

## 10.1.3 Informe de pruebas de caja blanca

### Prueba 1 – get\_google\_results() devuelve una lista válida

- **Entrada:** búsqueda de "teclado gaming" en *PcComponentes*
  - **Resultado esperado:** lista de productos con **título, precio, URL**
  - **Resultado real:**  correcto, 8 productos devueltos
  - **Observación:** sin duplicados, estructura completa y coherente
- 

### Prueba 2 – Filtro de precios aplicado correctamente

- **Entrada:** **minPrice = 30, maxPrice = 60**
- **Resultado esperado:** productos dentro del rango
- **Resultado real:**  correcto, los valores fuera de rango fueron excluidos

## Prueba 3 – Email no registrado en login

- **Entrada:** POST a `/api/login` con un email inexistente en la base de datos
  - **Resultado esperado:** código **404** y mensaje "**Correo no registrado**"
  - **Resultado real:** correcto, error gestionado y mostrado al usuario
  - **Observación:** la función `Usuario.query.filter_by(email=email).first()` devuelve `None` y es controlada correctamente
- 

## Prueba 4 – Conversión de precio

- **Entrada:** "49,99 €"
- **Transformación esperada:** `float(49.99)`
- **Resultado real:** correcto, soporta `,` y `.` como separadores

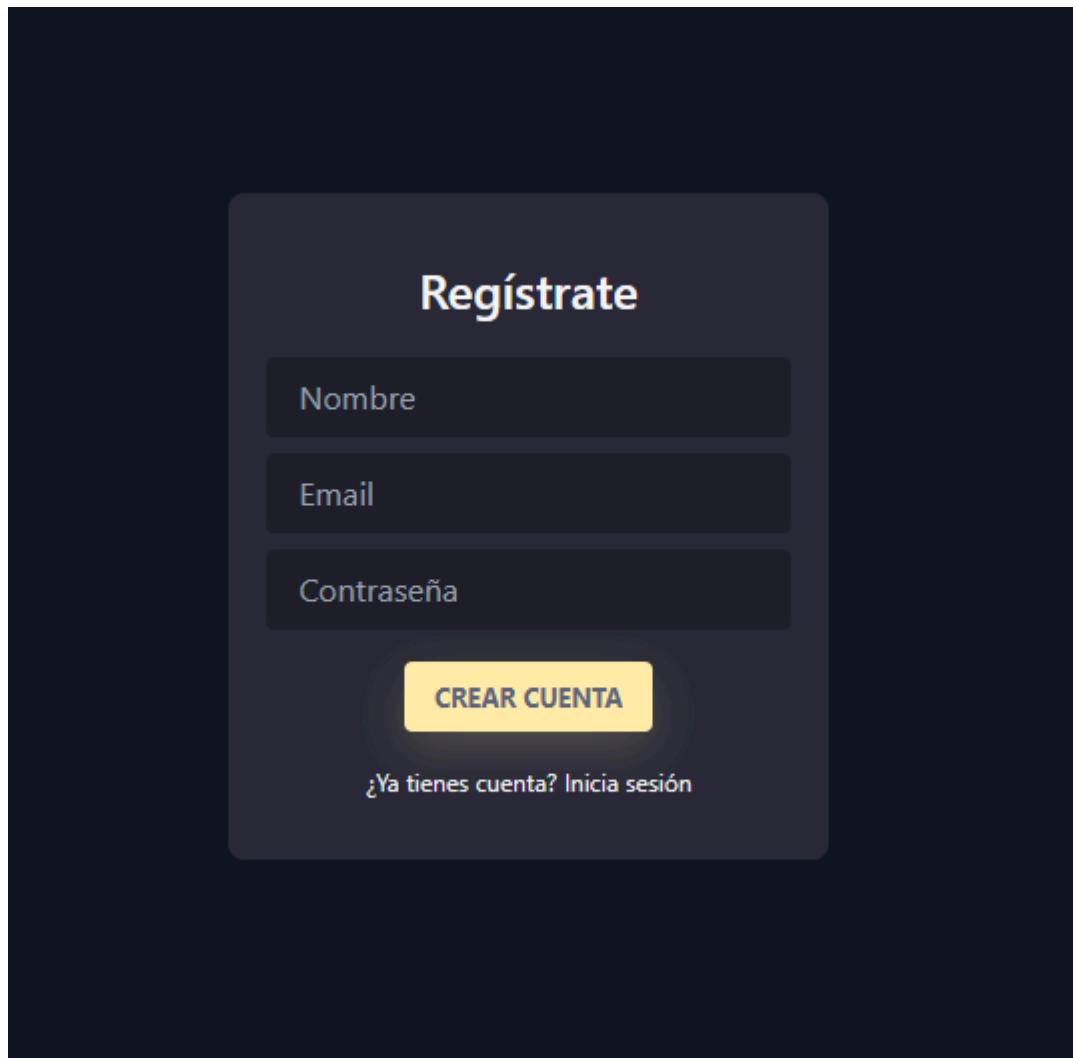
## 10.2 Plan de pruebas de caja Negra

Las pruebas de caja negra se realizaron desde el punto de vista del usuario, sin conocimiento del código interno. Se verificó el comportamiento esperado en distintos escenarios de uso.

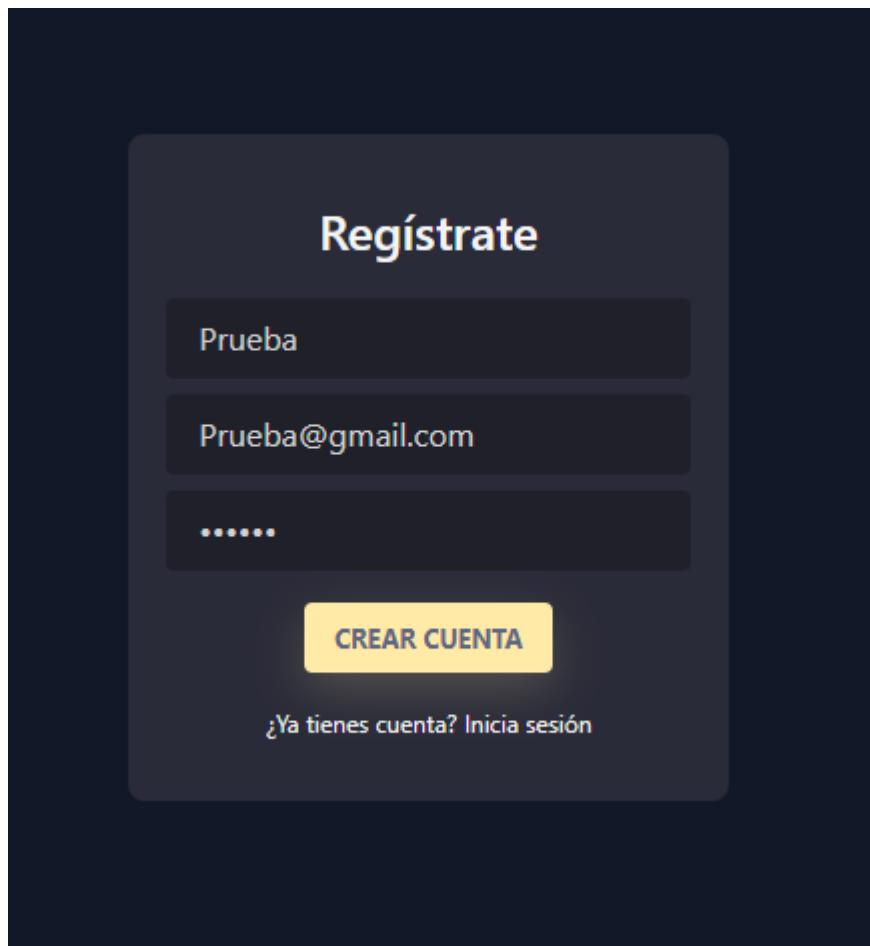
### Casos de prueba principales:

Prueba	Entrada esperada	Resultado esperado
Registro de usuario	Nombre, email válido, contraseña	Usuario creado y redireccionado a login
Login correcto	Email existente	Acceso a la interfaz de búsqueda
Búsqueda vacía	Sin escribir producto	Alerta: "Escribe un producto para buscar."
Sin tiendas seleccionadas	Producto escrito, sin tiendas	Alerta: "Selecciona al menos una tienda."
Búsqueda real (ej. "iPhone 16")	Producto + tienda + motor Google	Muestra tarjetas con resultados indexados
Búsqueda sin resultados	Producto existente no	Muestra mensaje: "No se encontraron resultados."

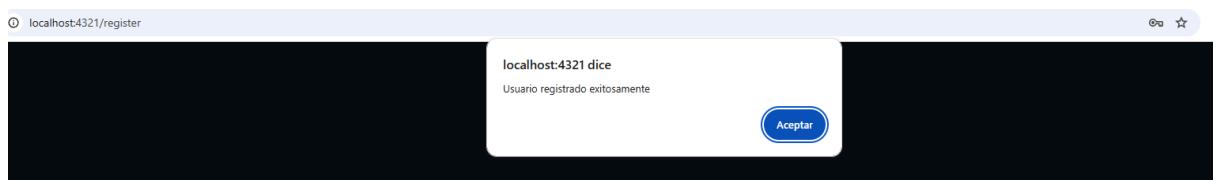
## 10.2.1 Registro de usuario



Colocamos los datos

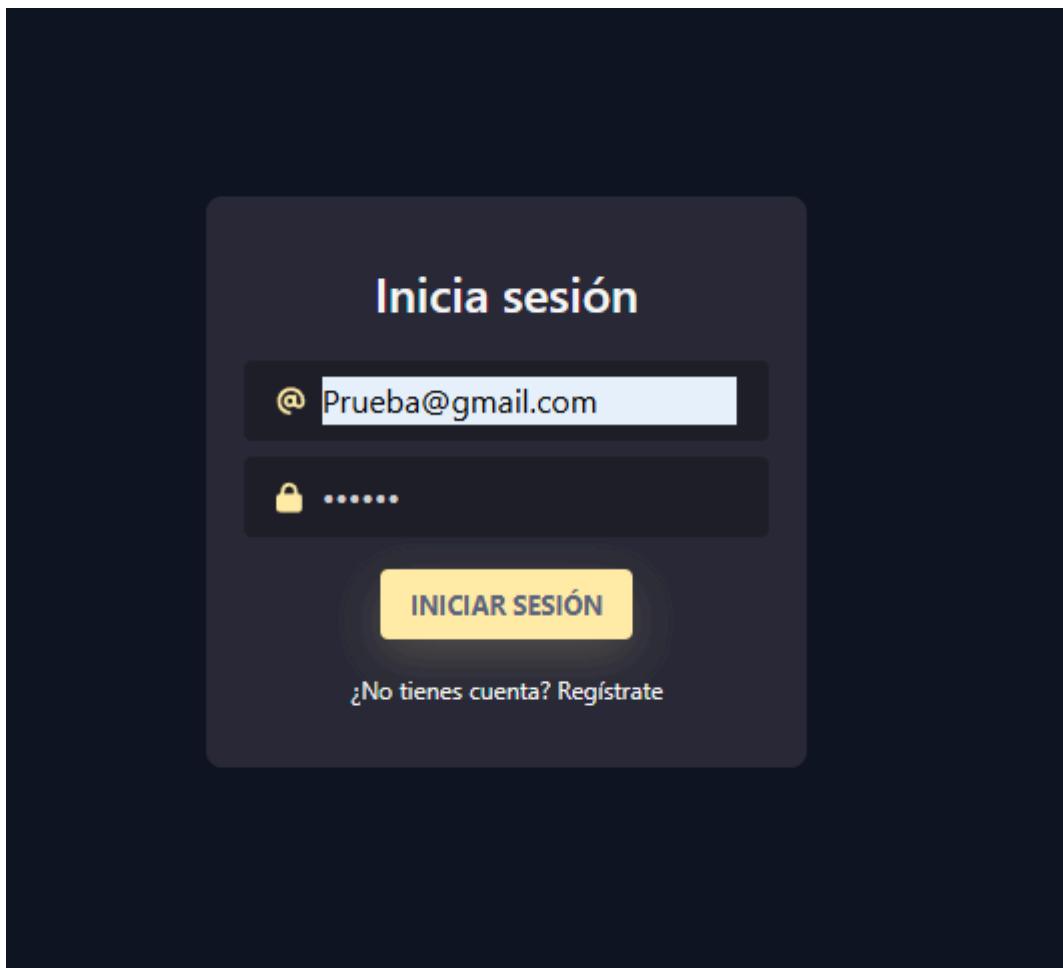


Y le damos a Crear Cuenta



Usuario creado y redireccionado a login

## 10.2.2 Login correcto:



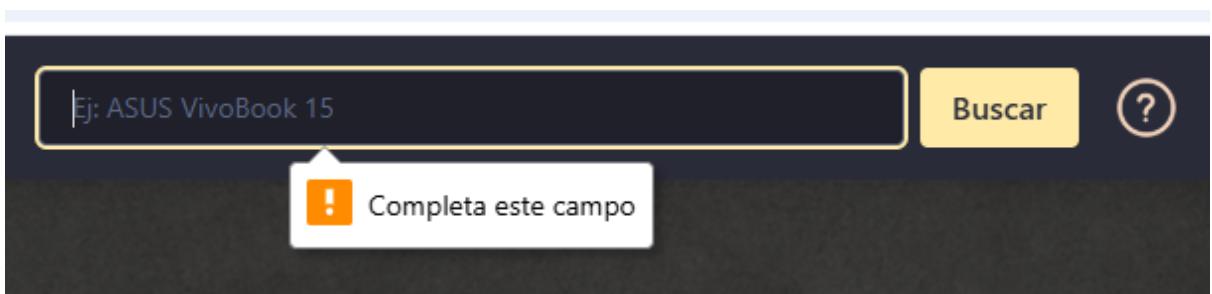
Le damos al botón de Iniciar Sesión

A screenshot of a web application titled "Buscador de Precios". The top navigation bar includes a search bar with the query "Ej: ASUS VivoBook 15", a "Buscar" button, and a user profile "Hola, Prueba". A sidebar on the left contains a "Motor de búsqueda" dropdown set to "Google", a "Filtros" section with price range inputs ("Precio mínimo (€) Ej: 100" and "Precio máximo (€) Ej: 1500"), and a "Tiendas" section listing various retailers like Amazon, AliExpress, Zalando, and eBay.

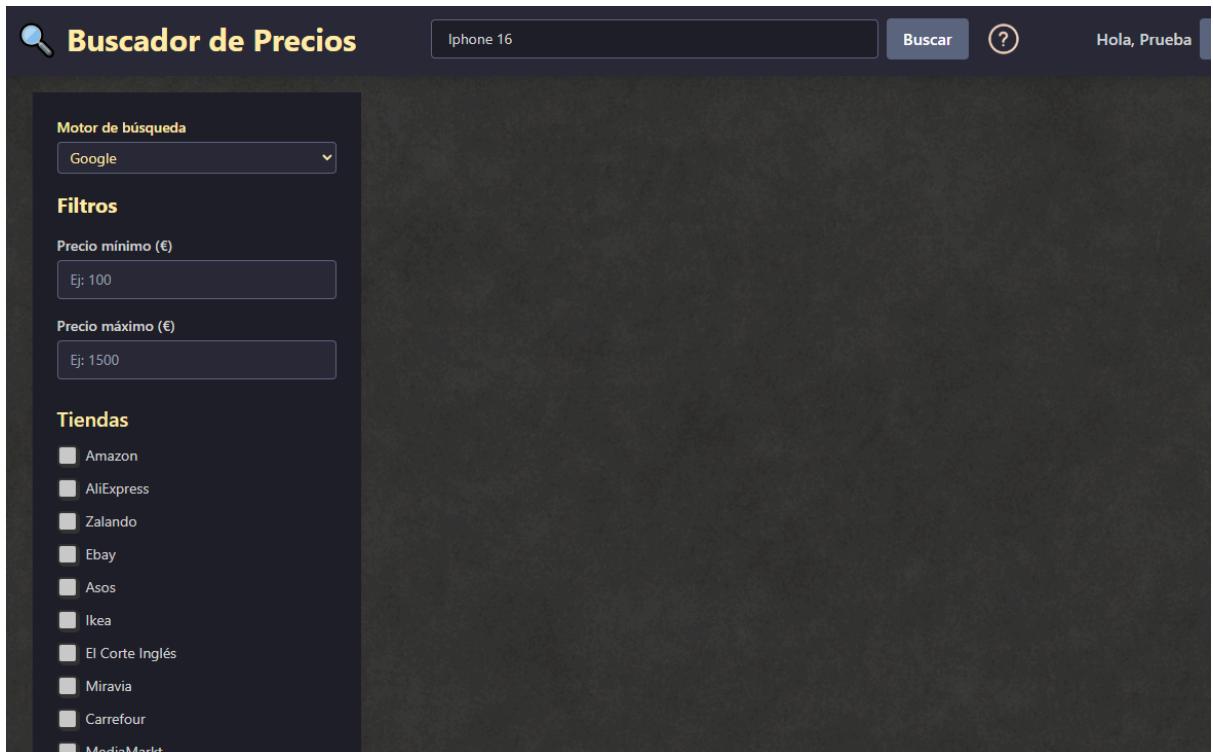
### 10.2.3 Búsqueda Vacía:



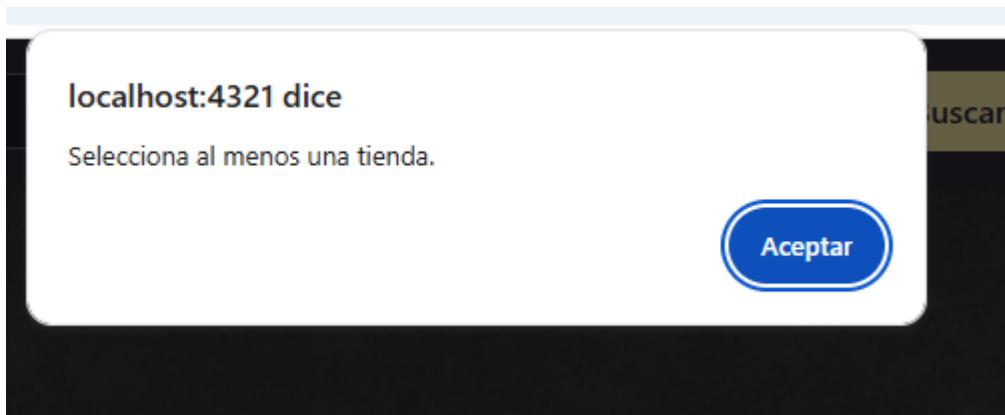
Presionamos el botón de Buscar



### 10.2.4 Sin tiendas seleccionadas:



Colocamos el prompt sin seleccionar ninguna tienda

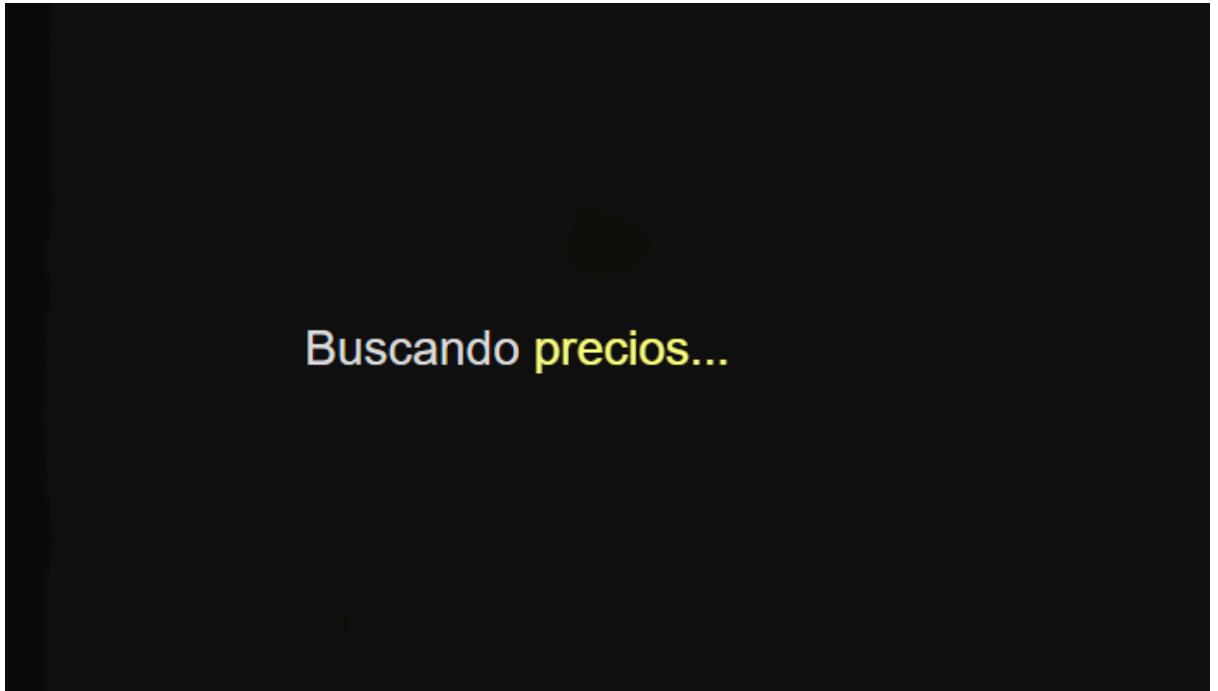


Nos devuelve la alerta

### 10.2.5 Búsqueda real (ej. "iPhone 16"):

The screenshot shows the 'Buscador de Precios' website. At the top, there's a search bar with the text 'Iphone 16'. Below it, a dropdown menu is set to 'Bing'. On the right side of the header, there are buttons for 'Buscar', a help icon, and user information ('Hola, Prueba' and 'Cerrar sesión'). The main content area has a sidebar on the left. The sidebar contains a section for 'Filtros' with input fields for 'Precio mínimo (€)' (with 'Ej: 100') and 'Precio máximo (€)' (with 'Ej: 1500'). Below that is a section for 'Tiendas' with a list of checkboxes. Several checkboxes are checked: 'El Corte Inglés' and 'MediaMarkt'. Other unselected options include Amazon, AliExpress, Zalando, Ebay, Asos, Ikea, Miravia, Carrefour, Nike, Adidas, and Puma.

Seleccionamos las tiendas y le damos al botón de buscar



Buscando precios...

Nos aparece la pantalla de carga

Screenshot of the 'Buscador de Precios' (Price Finder) website interface. The search bar at the top contains the text 'Iphone 16'. Below the search bar, there is a 'Motor de búsqueda' (Search engine) dropdown set to 'Bing', and a 'Buscar' (Search) button. To the right of the search bar are links for 'Hola, Prueba' (Hello, Test) and 'Cerrar sesión' (Close session).

The main content area shows search results for 'Iphone 16' across various categories:

- Telefonía - El Corte Inglés**:  
El Corte Inglés  
<https://www.elcorteingles.es/electronica/telefonia> - El Corte Inglés Si eres fiel a los icónicos iPhones, descubre los modelos del nuevo iPhone 16, como el iPhone 16, 16 Plus, 16 Pro, y 16 Pro Max, cada uno con su propio *Precio no disponible*.
- Apple iPhone 16e móvil libre - Apple - El Corte Inglés**:  
El Corte Inglés  
<https://www.elcorteingles.es/electronica/apple/iphone-16e-movil-libre> - Apple - El Corte Inglés Apple iPhone 16e móvil libre. Blanco . Elige capacidad *Precio no disponible*.
- Apple - Móviles y Smartphones - Electrónica - El Corte Inglés**:  
El Corte Inglés  
<https://www.elcorteingles.es/electronica/moviles-y-smartphones>... Apple - Móviles y Smartphones - Electrónica - El Corte Inglés El iPhone 16 se presenta en tres versiones para adaptarse a cada usuario: iPhone 16, 16 Pro y 16 Pr *Precio no disponible*.
- Apple - Electrónica - El Corte Inglés**:  
El Corte Inglés  
<https://www.elcorteingles.es/electronica/apple-electronica> - El Corte Inglés Compra tu iPhone 16 ahora y consigue un descuento de hasta 800 € entregando tu antiguo iPhone. ¡Descúbrelo! *Desde 800 €*
- Recompra dispositivo Apple - Electrónica - El Corte Inglés**:  
El Corte Inglés  
<https://www.elcorteingles.es/electronica/recompra-apple> Recompra dispositivo Apple - Electrónica - El Corte Inglés Rápido y sencillo, la mejor manera de conseguir tu nuevo iPhone, portátil Mac, iPad o Watch al mejor precio. *Precio no disponible*
- Apple iPhone 16 Pro móvil libre - El Corte Inglés**:  
El Corte Inglés  
<https://www.elcorteingles.es/electronica/apple/iphone-16-pro-movil-libre> - El Corte Inglés Hasta 33 horas de reproducción de video en el iPhone 16 Pro Max y hasta 27 horas en el iPhone 16 Pro. Consulta los avisos legales *Precio no disponible*

## 10.2.6 Búsqueda sin resultados:

The screenshot shows the 'Buscador de Precios' web application. At the top, there's a search bar with 'Iphone 16', a 'Buscar' button, and a user session indicator 'Hola, Prueba'. Below the search bar are sections for 'Motor de búsqueda' (Google), 'Filtros' (with 'Precio mínimo (€)' set to 4000 and 'Precio máximo (€)' set to Ej: 1500), and 'Tiendas' (listing various retailers like Amazon, AliExpress, Zalando, Ebay, Asos, Ikea, El Corte Inglés, Miravia, Carrefour, MediaMarkt, Nike, Adidas, Puma, Fnac, and Bicomponentes). The main content area is dark and empty, indicating no search results.

Forzamos la búsqueda para que no nos devuelva ningún resultado.

This screenshot shows the same 'Buscador de Precios' interface as the previous one, but with a different outcome. The search term 'Iphone 16' is present in the search bar. The 'Filtros' section has 'Precio mínimo (€)' set to 4000 and 'Precio máximo (€)' set to Ej: 1500. The 'Tiendas' section includes 'Amazon' and 'AliExpress'. On the right side of the interface, the text 'No se encontraron resultados.' is displayed, indicating that the search query did not yield any results.

# 11. Manuales

## 11.1 Manual técnico:

### 11.1.1 Descripción General del Sistema

El sistema desarrollado es una aplicación web denominada Buscador de Precios Indexados, que permite al usuario buscar un producto en varias tiendas online a través de motores de búsqueda (Google, Bing, DuckDuckGo), extraer los precios mediante técnicas de scraping y visualizar los resultados en una interfaz clara y filtrable.

El sistema se divide en dos partes:

- Backend (Flask): expone una API que recibe las peticiones de búsqueda, ejecuta el scraper y devuelve los resultados en formato JSON.
- Frontend (Astro + HTML/JS): permite al usuario registrarse, iniciar sesión, realizar búsquedas y ver los resultados de forma ordenada.

## 11.1.2 Estructura del Código

### Backend:

backend/	
app.py	Punto de entrada del servidor Flask
config.py	Configuración general
extensions.py	Inicialización de SQLAlchemy y
migraciones	
modelos/	
models.py	Modelo de Usuario
rutas/	
usuarios.py	Registro, login, gestión de usuario
search.py	Ruta principal de búsqueda
utils/	
scraper.py	Scraper principal con Selenium
logger.py	Logger configurado para el scraper

### Frontend:

frontend/src/	
components/	Header, filtros y elementos comunes
pages/	index.astro, login.astro etc...
styles/	global.css y clases personalizadas

## 11.1.3 Detalles de Implementación

- El backend usa Flask para definir las rutas API (</api/registro>, </api/login>, </api/search>).
- El scraper está encapsulado en la clase **GoogleScraper**, que emplea Selenium y BeautifulSoup para extraer resultados desde los motores de búsqueda.
- El frontend hace peticiones **fetch()** en JavaScript para comunicar con el backend y renderiza los resultados dinámicamente.
- Los resultados no se almacenan en la base de datos; solo los usuarios están persistidos.

## 11.1.4 Dependencias y Librerías

### Python (backend):

- Flask, Flask-SQLAlchemy, Flask-Migrate
- Selenium, webdriver-manager, beautifulsoup4
- requests, python-dotenv

### ASTRO (frontend):

- HTML, CSS, JS
- Tailwind CSS
- npm

### Otros:

- SQLite 3 como **base de datos**
- VS Code, Postman, Git, ChromeDriver

## 11.1.5 Consideraciones de Seguridad

- Las contraseñas de usuario se almacenan cifradas.
- No se guarda ningún dato del producto, lo que mejora la privacidad.
- Se detectan comportamientos sospechosos y se advierte al usuario.
- Se valida que no se pueda registrar con correos duplicados ni realizar búsquedas vacías o sin tiendas seleccionadas.
- Las peticiones al backend están limitadas a POST y se filtra el contenido JSON recibido.

## 11.2 Manual de instalación:

### 11.2.1 Requisitos del sistema:

- **Sistema operativo:** Windows 10/11 o Linux
- **Python:** versión 3.11 o superior
- **Navegador web:** Google Chrome
- **Conexión a internet**

### 11.2.2 Instalación del backend:

```
cd backend  
python -m venv venv  
venv\Scripts\activate  
pip install -r requirements.txt  
cd ..  
set FLASK_APP=backend.app:create_app  
set FLASK_ENV=development  
flask run
```

### 11.2.3 Instalación del frontend:

```
cd frontend  
npm install  
npm run dev
```

### 11.2.4 Observaciones

- El scraper requiere tener instalado ChromeDriver compatible con la versión del navegador.
- La carpeta **instance/** se genera automáticamente con el archivo **db.sqlite3** la primera vez que se ejecuta el backend.

## 11.3 Manual de usuario:

Este manual está destinado al usuario final que accede a la aplicación web desde el navegador.

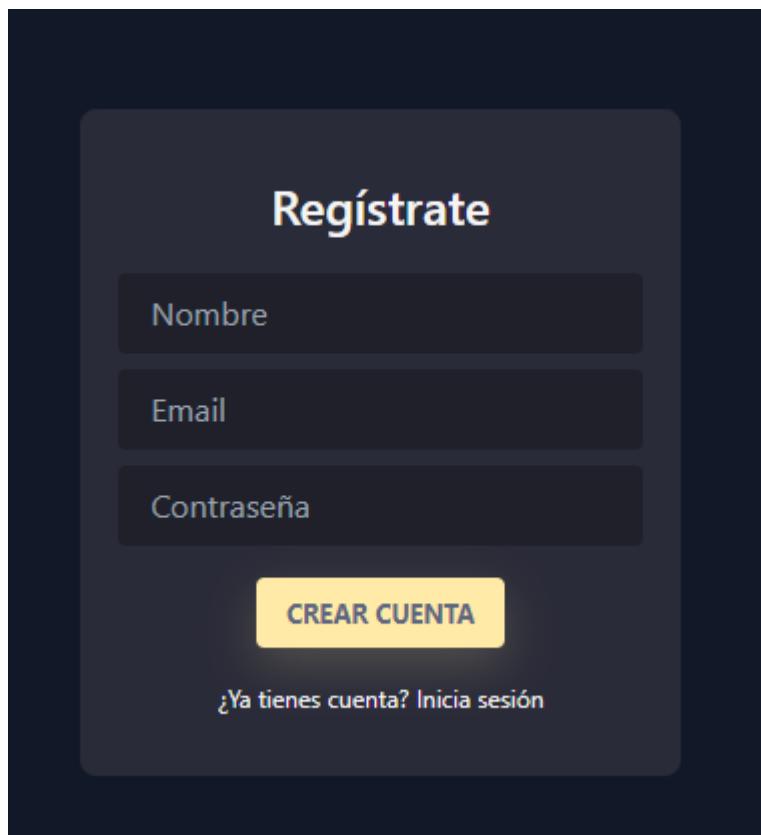
### 11.3.1 Acceso inicial:

#### 1. Abre el navegador y accede a la URL:

<http://localhost:4321/login>

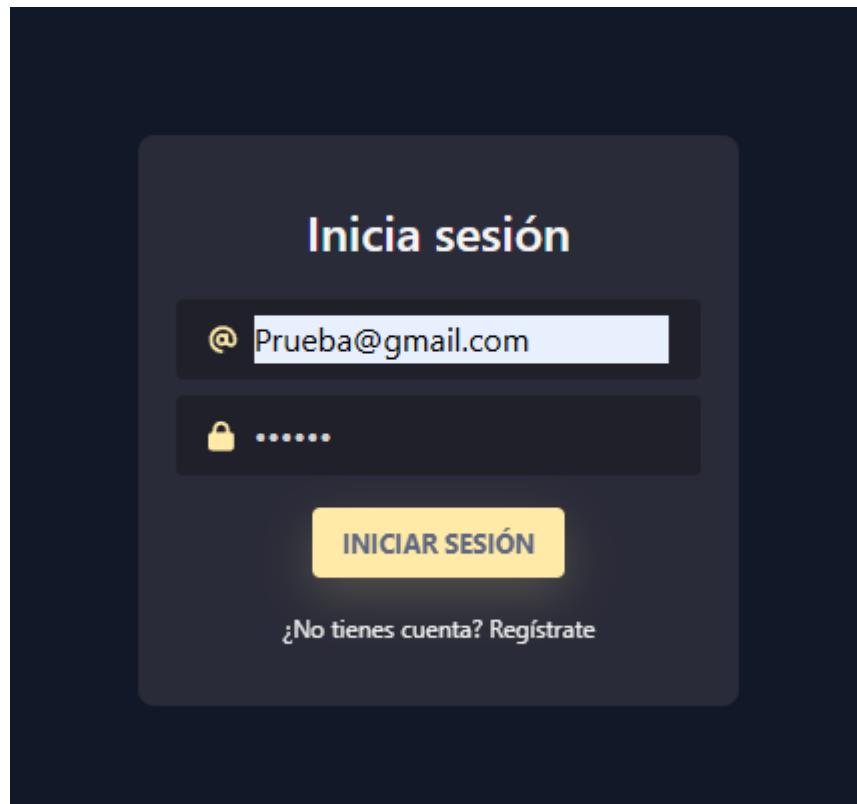
#### 2. Si no tienes cuenta:

- Haz clic en “**Regístrate**”
- Introduce tu **nombre, email y contraseña**
- Al registrarte, serás redirigido automáticamente al login



### 3. Si ya tienes cuenta:

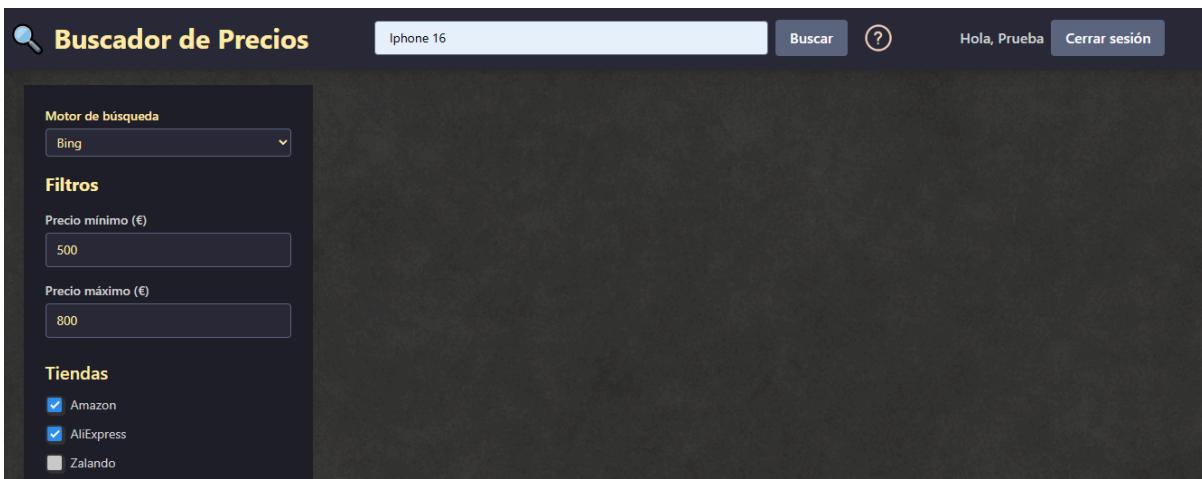
- Introduce tu **email**
- Pulsa “**Iniciar sesión**”



#### 11.3.2 Interfaz de búsqueda

Una vez logueado, se mostrará la interfaz principal:

- Campo de búsqueda: escribe el nombre del producto (ej. *iPhone 16*)
- Selecciona una o varias **tiendas** (checkboxes)
- Opcional: define un **precio mínimo y máximo**
- Elige el **motor de búsqueda** (Google, Bing, DuckDuckGo)
- Pulsa “**Buscar**”



### 11.3.3 Visualización de resultados

- Los productos se muestran en tarjetas con:

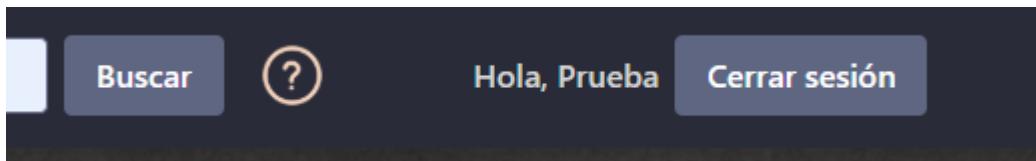
- Título
- Descripción breve
- Precio (si se encuentra)
- Imagen (si está disponible)
- Botón para visitar la tienda

The screenshot shows the search results for 'Iphone 16'. The sidebar remains the same. The results are displayed in a grid of cards:

- Apple iPhone 16 de 128 GB: Smartphone 5G con ...**  
Amazon https://www.amazon.es › dp  
Apple iPhone 16 de 128 GB: Smartphone 5G con ... Compra online Apple iPhone 16 de 128 GB: Smartphone 5G con Control de Cámara, Chip A18 y un subidón en autonomía. Compatible con los AirPods: ... Reseña: 714 temu iPhone Precio no disponible
- Apple iPhone 16 de 128 GB: Smartphone 5G con ...**  
Amazon https://www.amazon.es › dp  
Apple iPhone 16 de 128 GB: Smartphone 5G con ... Compra online Apple iPhone 16 de 128 GB: Smartphone 5G con Control de Cámara, Chip A18 y un subidón en autonomía. Compatible con los AirPods: ... Reseña: 714 temu iPhone Precio no disponible
- Apple iPhone 16 (128 GB) - Azul Ultramar - Amazon.es**  
Amazon https://www.amazon.es › dp  
Apple iPhone 16 (128 GB) - Azul Ultramar - Amazon.es Compra online Apple iPhone 16 (128 GB) - Azul Ultramar + Funda de Silicona con MagSafe - Denim. Envío en 1 día GRATIS con Amazon Prime. 4,4/5 (167)  
Precio no disponible
- Apple iPhone 16 de 128 GB: Smartphone 5G con ...**  
Amazon https://www.amazon.es › dp  
Apple iPhone 16 de 128 GB: Smartphone 5G con ... Compra online Apple iPhone 16 de 128 GB: Smartphone 5G con Control de Cámara, Chip A18 y un subidón en autonomía. Compatible con los AirPods: ... Reseña: 714 temu iPhone Precio no disponible
- Apple iPhone 16 Pro (128 GB) - Titánio Color Deserto ...**  
Amazon https://www.amazon.es › dp  
Apple iPhone 16 Pro (128 GB) - Titánio Color Deserto ... IMPRESIONANTE ACABADO EN TITANIO — El iPhone 16 Pro tiene un diseño de titanio robusto y ligero con una pantalla Super Retina XDR de 6,3 pulgadas ... 4,5/5 (505)  
Precio no disponible
- temu iPhone 16 pro max seguro - Amazon.es**  
Amazon https://www.amazon.es temu iPhone 16 pro max seguro - Amazon.es Métodos abreviados de teclado Buscar: alt + /  
Precio no disponible

### 11.3.4 Otras funciones

- En la parte superior se muestra el saludo: “Hola, [tu nombre]”
- Puedes cerrar sesión en cualquier momento haciendo clic en Cerrar sesión
- En el ícono de interrogación hay un acceso rápido a la página de Ayuda, donde se resuelven dudas comunes



### 11.3.5 Recomendaciones

- Evita hacer muchas búsquedas seguidas para no ser bloqueado por Google.
- Si no se muestran resultados, prueba con un nombre más específico o cambia de tienda/motor.

## 11.4 Manual de administración:

Dado que el sistema está orientado a una estructura mínima, las tareas administrativas se limitan a la gestión del backend y de usuarios (en base de datos).

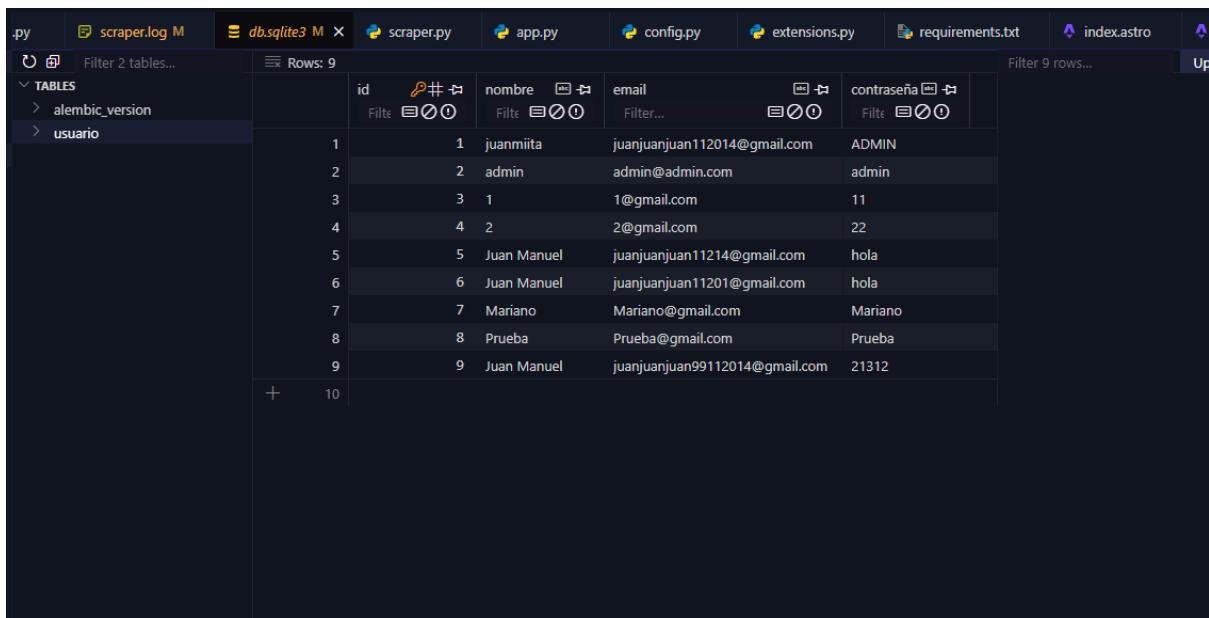
### 11.4.1 Acceso a la base de datos

La base de datos se encuentra en:

**instance/db.sqlite3**

Puedes abrirla con herramientas como DB Browser for SQLite para:

- Ver usuarios registrados
- Editar o eliminar usuarios manualmente si fuera necesario



The screenshot shows a dark-themed interface of DB Browser for SQLite. At the top, there's a toolbar with icons for file operations and a status bar showing 'Rows: 9'. Below the toolbar, a navigation bar lists several files: .py, scraper.log, db.sqlite3 (the current database), scraper.py, app.py, config.py, extensions.py, requirements.txt, and index.astro. Under the 'db.sqlite3' tab, a tree view shows two tables: 'alembic\_version' and 'usuario'. The 'usuario' table is expanded, showing its columns: id, nombre, email, and contraseña. Below the columns, a grid displays 9 rows of data. The first row has id=1, nombre='juanmiita', email='juanjuanjuan112014@gmail.com', and contraseña='ADMIN'. The last row has id=9, nombre='Juan Manuel', email='juanjuanjuan99112014@gmail.com', and contraseña='21312'. A footer at the bottom indicates there are 10 rows in total.

		id	nombre	email	contraseña
1		1	juanmiita	juanjuanjuan112014@gmail.com	ADMIN
2		2	admin	admin@admin.com	admin
3		3	1	1@gmail.com	11
4		4	2	2@gmail.com	22
5		5	Juan Manuel	juanjuanjuan11214@gmail.com	hola
6		6	Juan Manuel	juanjuanjuan11201@gmail.com	hola
7		7	Mariano	Mariano@gmail.com	Mariano
8		8	Prueba	Prueba@gmail.com	Prueba
9		9	Juan Manuel	juanjuanjuan99112014@gmail.com	21312

## 11.4.2 Logs del sistema

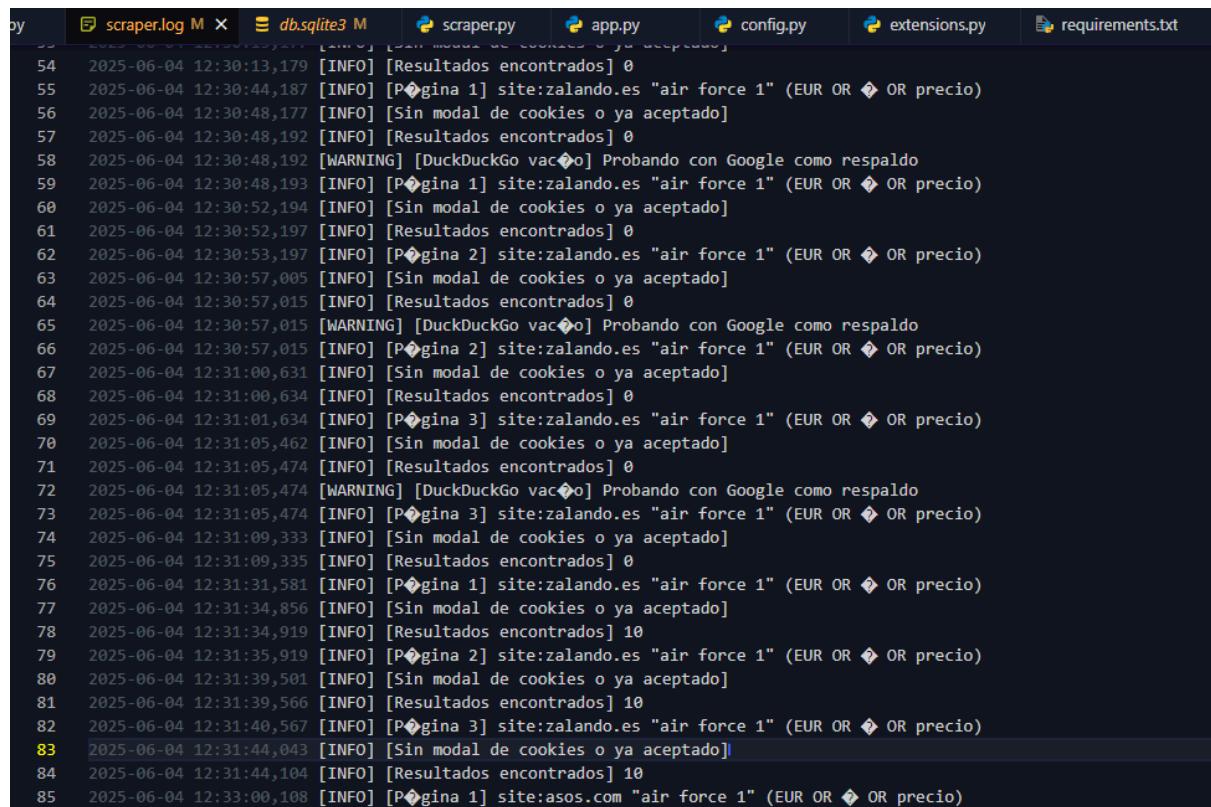
El scraper genera logs automáticos en la carpeta:

**logs/scraping.log**

En ellos se registra:

- Qué términos se buscan
- En qué tienda y motor
- Errores de scraping

Esto permite al administrador revisar el comportamiento del sistema y depurar errores.



```
by  scraper.log M X  db.sqlite3 M  scraper.py  app.py  config.py  extensions.py  requirements.txt
54 2025-06-04 12:30:13,179 [INFO] [Resultados encontrados] 0
55 2025-06-04 12:30:44,187 [INFO] [Página 1] site:zalando.es "air force 1" (EUR OR ⚡ OR precio)
56 2025-06-04 12:30:48,177 [INFO] [Sin modal de cookies o ya aceptado]
57 2025-06-04 12:30:48,192 [INFO] [Resultados encontrados] 0
58 2025-06-04 12:30:48,192 [WARNING] [DuckDuckGo vacío] Probando con Google como respaldo
59 2025-06-04 12:30:48,193 [INFO] [Página 1] site:zalando.es "air force 1" (EUR OR ⚡ OR precio)
60 2025-06-04 12:30:52,194 [INFO] [Sin modal de cookies o ya aceptado]
61 2025-06-04 12:30:52,197 [INFO] [Resultados encontrados] 0
62 2025-06-04 12:30:53,197 [INFO] [Página 2] site:zalando.es "air force 1" (EUR OR ⚡ OR precio)
63 2025-06-04 12:30:57,005 [INFO] [Sin modal de cookies o ya aceptado]
64 2025-06-04 12:30:57,015 [INFO] [Resultados encontrados] 0
65 2025-06-04 12:30:57,015 [WARNING] [DuckDuckGo vacío] Probando con Google como respaldo
66 2025-06-04 12:30:57,015 [INFO] [Página 2] site:zalando.es "air force 1" (EUR OR ⚡ OR precio)
67 2025-06-04 12:31:00,631 [INFO] [Sin modal de cookies o ya aceptado]
68 2025-06-04 12:31:00,634 [INFO] [Resultados encontrados] 0
69 2025-06-04 12:31:01,634 [INFO] [Página 3] site:zalando.es "air force 1" (EUR OR ⚡ OR precio)
70 2025-06-04 12:31:05,462 [INFO] [Sin modal de cookies o ya aceptado]
71 2025-06-04 12:31:05,474 [INFO] [Resultados encontrados] 0
72 2025-06-04 12:31:05,474 [WARNING] [DuckDuckGo vacío] Probando con Google como respaldo
73 2025-06-04 12:31:05,474 [INFO] [Página 3] site:zalando.es "air force 1" (EUR OR ⚡ OR precio)
74 2025-06-04 12:31:09,333 [INFO] [Sin modal de cookies o ya aceptado]
75 2025-06-04 12:31:09,335 [INFO] [Resultados encontrados] 0
76 2025-06-04 12:31:31,581 [INFO] [Página 1] site:zalando.es "air force 1" (EUR OR ⚡ OR precio)
77 2025-06-04 12:31:34,856 [INFO] [Sin modal de cookies o ya aceptado]
78 2025-06-04 12:31:34,919 [INFO] [Resultados encontrados] 10
79 2025-06-04 12:31:35,919 [INFO] [Página 2] site:zalando.es "air force 1" (EUR OR ⚡ OR precio)
80 2025-06-04 12:31:39,501 [INFO] [Sin modal de cookies o ya aceptado]
81 2025-06-04 12:31:39,566 [INFO] [Resultados encontrados] 10
82 2025-06-04 12:31:40,567 [INFO] [Página 3] site:zalando.es "air force 1" (EUR OR ⚡ OR precio)
83 2025-06-04 12:31:44,043 [INFO] [Sin modal de cookies o ya aceptado]
84 2025-06-04 12:31:44,104 [INFO] [Resultados encontrados] 10
85 2025-06-04 12:33:00,108 [INFO] [Página 1] site:asos.com "air force 1" (EUR OR ⚡ OR precio)
```

### 11.4.3 Reinicio del scraper

Si el scraper queda bloqueado (por ejemplo, si Google lanza CAPTCHA de forma permanente), se recomienda:

- **Reiniciar el router (para cambiar la IP) o ejecutar en cmd:**

```
ipconfig /release
```

```
ipconfig /renew
```

```
ipconfig /flushdns
```

- **Cerrar el navegador Chrome**
- **Borrar cookies desde Selenium en el código (opcional)**
- **Volver a lanzar el backend**

### 11.4.4 Mantenimiento del entorno

- Revisa que chromedriver esté actualizado
- Actualiza las dependencias con:

```
pip install -r requirements.txt
```

```
npm install
```

## 12. Conclusiones finales



### 12.1 Grado de consecución de objetivos finales:

Los objetivos establecidos al inicio del proyecto han sido cumplidos en su mayoría con éxito:

- ✓ Se ha desarrollado una aplicación web funcional que permite al usuario buscar productos en múltiples tiendas y comparar precios de manera automatizada.
- ✓ El sistema cuenta con registro, login y gestión básica de usuarios.
- ✓ Se ha implementado un scraper dinámico con Selenium capaz de extraer información desde buscadores como Google, Bing y DuckDuckGo.
- ✓ El usuario puede aplicar filtros personalizados por tienda y rango de precios.
- ✓ Se ha entregado una documentación clara, incluyendo manual técnico, de usuario y plan de pruebas.

En conjunto, el sistema entregado cumple los requisitos funcionales previstos y demuestra un alto grado de madurez técnica en su primera fase.

## 12.2 Posibles mejoras o ampliaciones por implementar en el futuro:

Aunque el sistema está operativo y estable, existen múltiples líneas de mejora que podrían abordarse en fases posteriores:

-  Persistencia de resultados: guardar historial de búsquedas y resultados en base de datos para reutilización o comparación.
-  Añadir más motores de búsqueda: como Brave Search o Yahoo para ampliar cobertura.
-  Sistema de favoritos o listas de seguimiento para productos de interés.
-  Visualización de precios históricos para analizar la evolución de un producto.
-  Autenticación avanzada: incorporar cifrado real de contraseñas y control de sesiones (JWT).
-  Exportación de resultados (CSV, PDF) para comparar offline.
-  Sistema modular de plugins de tiendas: adaptar el scraping directamente a webs específicas.

## 12.3 Análisis del Aprendizaje y Experiencia:

Este proyecto ha supuesto un reto técnico y organizativo real, similar a lo que implicaría el desarrollo de un producto funcional en el entorno profesional.

Durante su desarrollo se han reforzado y aplicado múltiples competencias:

- Diseño y arquitectura web full stack
- Uso avanzado de herramientas como Flask, Selenium, Astro y Tailwind CSS
- Planificación personal y gestión del tiempo mientras se combinaba el proyecto con las prácticas
- Capacidad de autodiagnóstico, resolución de errores y búsqueda activa de soluciones
- Elaboración de documentación técnica y de usuario, profesional y estructurada

La experiencia ha sido muy enriquecedora tanto a nivel técnico como personal. Supone un claro ejemplo de cómo un desarrollador junior puede integrar conocimientos de múltiples áreas para crear una solución útil, bien estructurada y con posibilidades reales de evolución.

## 13. Bibliografía

1. Flask Documentation. *Flask web development framework.* <https://flask.palletsprojects.com/>
2. Selenium Documentation. *Browser Automation with Selenium.* <https://www.selenium.dev/documentation/>
3. BeautifulSoup Documentation. *Web scraping with BeautifulSoup.* <https://www.crummy.com/software/BeautifulSoup/>
4. Astro.build. *Modern static site builder.* <https://docs.astro.build/>
5. TailwindCSS. *Utility-first CSS framework.* <https://tailwindcss.com/docs>
6. SQLite Documentation. *Self-contained, serverless SQL database engine.* <https://www.sqlite.org/docs.html>
7. GitHub. *Repositorio personal de desarrollo del TFG* [https://github.com/juanmiitaklk/TFG-Buscador\\_de\\_precios\\_inexados](https://github.com/juanmiitaklk/TFG-Buscador_de_precios_inexados).
8. Stack Overflow. *Preguntas y respuestas técnicas utilizadas durante el desarrollo.* <https://stackoverflow.com/>

## 14. PRESENTACIÓN

<https://view.genially.com/6845e6479ac5b8c8009942c1/presentation-tfg-juan-manuel-gonzalez-diaz>