DOCUMENTACIÓN DE PROYECTO DE Clothing Shop

Eder Jaset Jimenez Peralta

Christian Domingo Flores

Diego Loza Romero

Materia:

Institución: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Profesor(a): Aldo Romero Arroyo

Fecha: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Contenido

[Índice 2](#_Toc212383698)

[Resumen Ejecutivo 4](#_Toc212383699)

[Introducción 4](#_Toc212383700)

[Contexto del proyecto 4](#_Toc212383701)

[Objetivo general 4](#_Toc212383702)

[Objetivos específicos 4](#_Toc212383703)

[Alcance del proyecto 5](#_Toc212383704)

[Justificación 5](#_Toc212383705)

[Situacion Actual 7](#_Toc212383706)

[Requisitos del sistema 8](#_Toc212383707)

[Requisitos funcionales 8](#_Toc212383708)

[Requisitos no funcionales 8](#_Toc212383709)

[Diseño del sistema 9](#_Toc212383710)

[Arquitectura del sistema 9](#_Toc212383711)

[Diagramas (casos de uso, clases, ER, etc.) 9](#_Toc212383712)

[Diseño de interfaz 9](#_Toc212383713)

[Desarrollo e implementación 10](#_Toc212383714)

[Tecnologías utilizadas 10](#_Toc212383715)

[Estructura del código 15](#_Toc212383716)

[Configuración del entorno 17](#_Toc212383717)

[Integración del bot de WhatsApp 17](#_Toc212383718)

[Pruebas del sistema 18](#_Toc212383719)

[Tipos de pruebas 18](#_Toc212383720)

[Casos de prueba 18](#_Toc212383721)

[Resultados de las pruebas 18](#_Toc212383722)

[Resultados 19](#_Toc212383723)

[Conclusiones y recomendaciones 20](#_Toc212383724)

[Bibliografía 21](#_Toc212383725)

[Anexos 22](#_Toc212383726)

[Manual de usuario 22](#_Toc212383727)

[Manual técnico 22](#_Toc212383728)

[Código fuente (opcional) 22](#_Toc212383729)

# Resumen Ejecutivo

El presente proyecto consiste en el desarrollo de una aplicación web integrada con un sistema automatizado basado en un bot de WhatsApp. Su objetivo principal es optimizar la comunicación entre los usuarios y la plataforma mediante la automatización de tareas repetitivas, mejorando así la eficiencia operativa y la experiencia del usuario.

La aplicación web permitirá a los usuarios interactuar con el sistema a través de una interfaz intuitiva, mientras que el bot de WhatsApp servirá como un canal alternativo de comunicación, gestionando consultas, notificaciones y respuestas automáticas. Esta integración se desarrollará utilizando tecnologías modernas de desarrollo web y APIs de mensajería, garantizando escalabilidad, seguridad y facilidad de mantenimiento.

El proyecto aborda la necesidad de automatizar procesos de atención o registro dentro de un entorno web, reduciendo la intervención manual y el tiempo de respuesta. De esta manera, se ofrece una solución tecnológica adaptable a distintos escenarios empresariales o académicos.

# Introducción

### Contexto del proyecto

En la actualidad, la automatización de procesos mediante inteligencia artificial y bots de mensajería se ha convertido en una herramienta clave para mejorar la eficiencia en la comunicación entre usuarios y sistemas digitales. WhatsApp, al ser una de las plataformas de mensajería más utilizadas a nivel mundial, representa una excelente oportunidad para integrar soluciones automatizadas en entornos web.

### Objetivo general

Desarrollar una aplicación web con integración a un bot automatizado de WhatsApp que permita la comunicación eficiente entre el usuario y el sistema, automatizando tareas básicas y optimizando los tiempos de respuesta.

### Objetivos específicos

* Diseñar e implementar una interfaz web funcional y amigable para el usuario.
* Configurar un bot de WhatsApp que permita la automatización de consultas o procesos.
* Desarrollar la conexión entre la aplicación web y el bot mediante el uso de APIs.
* Implementar medidas de seguridad y gestión de datos.
* Realizar pruebas de funcionamiento e integración para garantizar la estabilidad del sistema.

### Alcance del proyecto

El proyecto abarcará el desarrollo completo del sistema, desde el diseño de la base de datos hasta la integración del bot de WhatsApp. No se incluirán funciones avanzadas de inteligencia artificial o procesamiento de lenguaje natural en esta versión, aunque el sistema quedará preparado para futuras ampliaciones.

### Justificación

Este proyecto surge de la necesidad de mejorar la atención al usuario y la gestión de información mediante canales automatizados. Al combinar una aplicación web con un bot de mensajería, se facilita la interacción inmediata, reduciendo la carga de trabajo manual y aumentando la disponibilidad del servicio. La solución puede aplicarse en distintos contextos, como soporte técnico, seguimiento de pedidos, atención académica o notificaciones automáticas.

## 

**Analisis del problema**

**Descripción del problema**

En el sector de venta de ropa, muchas pequeñas y medianas tiendas enfrentan dificultades para mantener una comunicación eficiente con sus clientes y gestionar pedidos de forma automatizada.

Generalmente, las consultas sobre disponibilidad, precios, tallas o seguimiento de pedidos se realizan de manera manual por mensajes o llamadas, lo que genera retrasos, saturación de atención y errores en la gestión de información.

Además, la falta de una plataforma web unificada impide ofrecer una experiencia de compra moderna, con catálogos digitales y comunicación en tiempo real.

Por ello, se plantea el desarrollo de una aplicación web de venta de ropa, complementada con un bot automatizado de WhatsApp que permita atender consultas, registrar pedidos y enviar notificaciones de manera inmediata y eficiente.

Actores involucrados

* **Administrador del sistema:** Encargado de gestionar el inventario, subir productos, administrar usuarios y supervisar las interacciones del bot. También recibe notificaciones mediante un componente React en el panel administrativo.
* **Clientes o usuarios finales:** Personas interesadas en adquirir productos de ropa. Pueden navegar por el catálogo, realizar pedidos desde la aplicación web y recibir confirmaciones o respuestas a través del bot de WhatsApp.
* **Bot automatizado de WhatsApp:** Sistema encargado de responder consultas frecuentes, enviar mensajes de confirmación o notificación (por ejemplo, estado del pedido o promociones). Facilita la atención al cliente sin intervención humana.
* **Base de datos (RDS MySQL):** Contiene la información del inventario, usuarios, pedidos y registros del bot. Permite que tanto el backend como el bot accedan a los datos en tiempo real de forma segura y centralizada.

## Situacion Actual

Actualmente, las tiendas de ropa que operan sin una plataforma automatizada dependen de la atención manual mediante redes sociales o aplicaciones de mensajería.

Esto ocasiona que los pedidos se gestionen de forma poco eficiente, con alta posibilidad de error en inventario, pérdida de ventas y demoras en la atención al cliente.

Con la implementación de una aplicación web de venta de ropa y su integración con un bot automatizado de WhatsApp, se busca ofrecer un sistema capaz de:

* Gestionar pedidos y productos de manera organizada.
* Permitir atención 24/7 mediante respuestas automáticas.
* Reducir errores humanos.
* Mejorar la satisfacción y fidelización de los clientes.

# Requisitos del sistema

## Requisitos funcionales

## Requisitos no funcionales

# Diseño del sistema

## Arquitectura del sistema

## Diagramas (casos de uso, clases, ER, etc.)

## Diseño de interfaz

# Desarrollo e implementación

## Tecnologías utilizadas

**Frontend – React**

**¿Qué es?**

**React** es una **biblioteca de JavaScript** creada por Facebook para construir **interfaces de usuario (UI)** interactivas y dinámicas.  
 Permite crear **componentes reutilizables** que actualizan la información en pantalla sin recargar toda la página.

**¿Para qué sirve en tu proyecto?**

En tu sistema, React se usa para:

* Mostrar las pantallas de la aplicación web (formularios, paneles, dashboards, etc.).
* Conectarse al backend Flask mediante peticiones **HTTP o API REST**.
* Mostrar en tiempo real la información que proviene del bot o la base de datos.

**¿Por qué usarlo?**

Alta velocidad de carga (SPA: Single Page Application).  
 Reutilización de componentes (reduce código repetido).  
 Excelente integración con APIs y servicios externos (como el bot de WhatsApp).  
 Comunidad amplia y soporte constante.

**Backend – Python (Flask)**

**¿Qué es?**

**Flask** es un **microframework de Python** para desarrollo web.  
 Sirve para crear **rutas, controladores y APIs RESTful**, que conectan el frontend con la base de datos o con otros servicios.

**¿Para qué sirve en tu proyecto?**

Flask es el **núcleo lógico del sistema**, encargado de:

* Procesar las solicitudes del frontend React.
* Ejecutar la lógica de negocio (por ejemplo, registro, autenticación, manejo del bot).
* Conectarse a la base de datos MySQL.
* Integrar el **bot automatizado de WhatsApp** (usando la API de WhatsApp Cloud o Twilio).
* Gestionar respuestas automáticas, notificaciones o mensajes enviados desde la web.

**¿Por qué usarlo?**

Ligero, rápido y fácil de configurar.  
 Compatible con librerías de IA, análisis de datos y APIs.  
 Permite una arquitectura limpia (backend separado del frontend).  
 Ideal para microservicios e integraciones (como bots o APIs externas).

**Base de Datos – Amazon RDS (MySQL)**

**¿Qué es?**

**Amazon RDS (Relational Database Service)** es un servicio de base de datos relacional en la nube de **AWS (Amazon Web Services)**.  
 Permite ejecutar bases de datos como **MySQL** sin tener que instalar ni mantener servidores manualmente.

**¿Para qué sirve en tu proyecto?**

En tu sistema, **RDS MySQL** se usa para:

* Almacenar todos los datos del sistema web y del bot.
* Guardar usuarios, mensajes, registros de interacción, logs o configuraciones.
* Permitir acceso remoto y seguro desde Flask sin preocuparte por el hardware.

**¿Por qué usarlo?**

Alta disponibilidad y seguridad administrada por AWS.  
Escalabilidad: se adapta Frontend – React

**¿Qué es?**

React es una biblioteca de JavaScript creada por Facebook para construir interfaces de usuario (UI) interactivas y dinámicas.

Permite crear componentes reutilizables que actualizan la información en pantalla sin recargar toda la página.

**¿Para qué sirve en tu proyecto?**

En tu sistema, React se usa para:

* Mostrar las pantallas de la aplicación web (formularios, paneles, dashboards, etc.).
* Conectarse al backend Flask mediante peticiones HTTP o API REST.
* Mostrar en tiempo real la información que proviene del bot o la base de datos.

**¿Por qué usarlo?**

Alta velocidad de carga (SPA: Single Page Application).

Reutilización de componentes (reduce código repetido).

Excelente integración con APIs y servicios externos (como el bot de WhatsApp).

Comunidad amplia y soporte constante.

**Backend – Python (Flask)**

**¿Qué es?**

Flask es un microframework de Python para desarrollo web.

Sirve para crear rutas, controladores y APIs RESTful, que conectan el frontend con la base de datos o con otros servicios.

**¿Para qué sirve en tu proyecto?**

Flask es el núcleo lógico del sistema, encargado de:

* Procesar las solicitudes del frontend React.
* Ejecutar la lógica de negocio (por ejemplo, registro, autenticación, manejo del bot).
* Conectarse a la base de datos MySQL.
* Integrar el bot automatizado de WhatsApp (usando la API de WhatsApp).
* Gestionar respuestas automáticas, notificaciones o mensajes enviados desde la web.

**¿Por qué usarlo?**

Ligero, rápido y fácil de configurar.

Compatible con librerías de IA, análisis de datos y APIs.

Permite una arquitectura limpia (backend separado del frontend).

Ideal para microservicios e integraciones (como bots o APIs externas).

**Base de Datos – Amazon RDS (MySQL)**

**¿Qué es?**

Amazon RDS (Relational Database Service) es un servicio de base de datos relacional en la nube de AWS (Amazon Web Services).

Permite ejecutar bases de datos como MySQL sin tener que instalar ni mantener servidores manualmente.

**¿Para qué sirve en tu proyecto?**

En tu sistema, RDS MySQL se usa para:

* Almacenar todos los datos del sistema web y del bot.
* Guardar usuarios, mensajes, registros de interacción, logs o configuraciones.
* Permitir acceso remoto y seguro desde Flask sin preocuparte por el hardware.

**¿Por qué usarlo?**

Alta disponibilidad y seguridad administrada por AWS.

Escalabilidad: se adapta si tu sistema crece.

Copias automáticas de respaldo (backups).

Ideal para integrarse con aplicaciones distribuidas (como la tuya).

Mantiene conexión directa con Flask a través de un endpoint RDS (URL de conexión MySQL). si tu sistema crece.  
 Copias automáticas de respaldo (backups).  
 Ideal para integrarse con aplicaciones distribuidas (como la tuya).  
Mantiene conexión directa con Flask a través de un **endpoint RDS** (URL de conexión MySQL).

# Estructura del código

**Estructura del Backend (Flask)**

El backend del sistema fue desarrollado con **Flask (Python)**, un microframework que permite crear APIs RESTful ligeras y modulares.  
A continuación se describe la estructura general del código y la funcionalidad de los principales archivos:

**Archivo principal: app.py**

Este archivo contiene la configuración y el arranque del servidor Flask, además de las rutas principales de la aplicación.  
Se encarga de conectar con la base de datos, inicializar las extensiones de seguridad y exponer los endpoints para el registro e inicio de sesión.

**Funciones principales:**

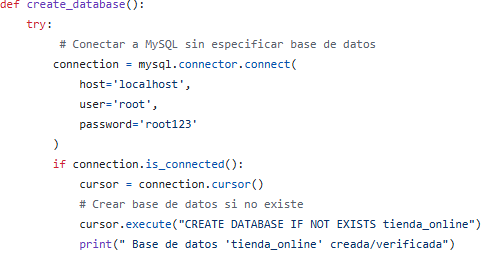
|  |  |  |
| --- | --- | --- |
| Función | Descripción | Archivo |
| create\_database() | Crea la base de datos tienda\_online si no existe. | app.py |
| signup() | Registra nuevos usuarios en la base de datos, cifrando su contraseña con bcrypt. | app.py |
| Signup\_Admin() | Registra cuentas administrativas con rol asignado. | app.py |
| login() | Autentica usuarios o administradores, valida credenciales y crea sesión. | app.py |

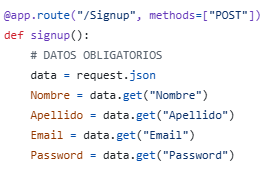
**Tecnologías implementadas:**

* Flask para manejo de rutas y lógica de servidor.
* SQLAlchemy para la gestión de datos con MySQL.
* Bcrypt para la encriptación de contraseñas.
* CORS para permitir la comunicación entre el backend y el frontend React.

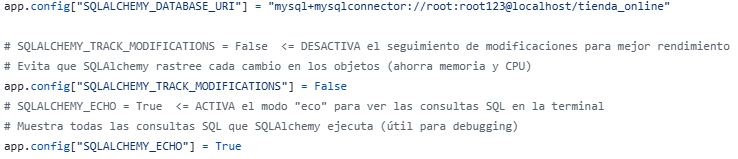
**Configuración del servidor y conexión con la base de datos**

El siguiente fragmento muestra cómo se inicializa Flask y se configura la conexión a la base de datos **MySQL**.  
Se utiliza mysql.connector para verificar la existencia de la base de datos tienda\_online, y en caso de no existir, se crea automáticamente.





Esta función garantiza que la base de datos esté disponible antes de iniciar el sistema.  
Posteriormente, se configura SQLAlchemy para establecer la conexión ORM (Object Relational Mapping):

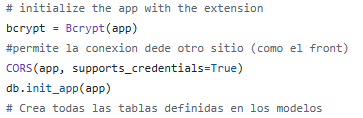


**Inicialización de extensiones**

Flask permite integrar extensiones que amplían su funcionalidad.

En este proyecto se utilizaron las siguientes:

* Flask-Bcrypt: para cifrar contraseñas.
* Flask-CORS: para permitir la comunicación con el frontend React.
* SQLAlchemy: para gestionar los modelos de base de datos.

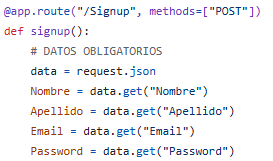
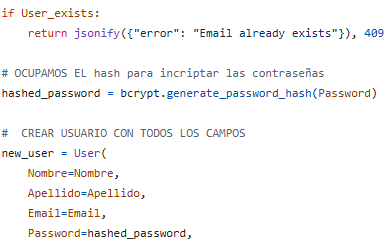


Al iniciar la aplicación, se ejecuta el siguiente bloque que genera todas las tablas definidas en models.py:

De esta forma, el sistema garantiza que la estructura de la base de datos esté sincronizada con los modelos del proyecto sin necesidad de ejecutar scripts SQL manualmente.

**Endpoint de registro de usuario**

El endpoint /Signup permite registrar nuevos usuarios en la base de datos.

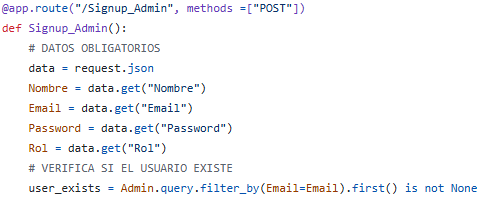
Este bloque de código recibe los datos desde el frontend en formato JSON, valida si el correo ya existe, cifra la contraseña con bcrypt y almacena el registro:

**Flujo funcional:**

1. El usuario completa el formulario de registro desde la aplicación web.
2. React envía los datos al backend Flask por medio de una API REST.
3. Flask valida la información y encripta la contraseña.
4. Se crea el registro en la base de datos MySQL.
5. El sistema devuelve una respuesta JSON confirmando la creación del usuario.

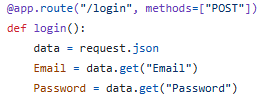
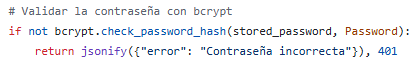
Registro de administrador

El endpoint /Signup\_Admin realiza la misma función que /Signup, pero con el modelo Admin.

Permite crear cuentas con rol administrativo, necesarias para gestionar productos, usuarios y pedidos.

**Inicio de sesión de usuarios y administradores**

La autenticación se gestiona a través del endpoint /login.

Este método permite que tanto clientes como administradores puedan iniciar sesión utilizando sus credenciales.

El flujo de inicio de sesión es el siguiente:

1. El sistema recibe las credenciales del usuario.
2. Verifica si el correo pertenece a un cliente o administrador.
3. Compara la contraseña encriptada.
4. Crea una sesión activa utilizando Flask-Session.
5. Devuelve una respuesta JSON con los datos del usuario y su rol.

**Ejecución del servidor**

Finalmente, el servidor Flask se ejecuta en modo de desarrollo (debug activado):



Esto permite realizar pruebas locales y detectar errores en tiempo real.

**Comunicación con el frontend**

Para permitir la comunicación entre el frontend (React) y el backend (Flask), se habilitó **CORS** con:



**Conclusión del módulo**

El archivo app.py constituye el corazón del backend de la aplicación.

Su diseño modular permite:

* Escalar fácilmente el sistema agregando nuevas rutas.
* Mantener la seguridad de contraseñas mediante encriptación.
* Garantizar compatibilidad con el frontend gracias a CORS.
* Facilitar la conexión y gestión de datos a través de SQLAlchemy.

Este enfoque asegura una arquitectura clara, robusta y fácil de mantener.

**Estructura y explicación del archivo models.py**

El archivo models.py define las tablas y modelos de datos utilizados por la aplicación web de venta de ropa.

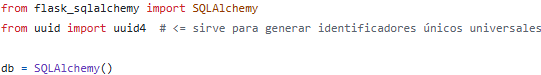
Su propósito es estructurar la información que se almacena en la base de datos MySQL, tanto para los usuarios (clientes) como para los administradores del sistema.

Se utiliza la librería Flask-SQLAlchemy, que facilita el manejo de bases de datos relacionales mediante el uso del patrón ORM (Object Relational Mapping).

**Inicialización de la base de datos**

En primer lugar, se crea una instancia de SQLAlchemy() que será inicializada posteriormente dentro del archivo app.py.

Esto permite que el modelo esté desacoplado del archivo principal, facilitando la modularidad y mantenimiento del código.



**Generación de identificadores únicos (UUID)**

Para los campos de ID, se utiliza la función uuid4() proveniente de la librería uuid, que genera un **identificador único universal**.  
Esto evita conflictos de claves primarias y mejora la trazabilidad de registros.



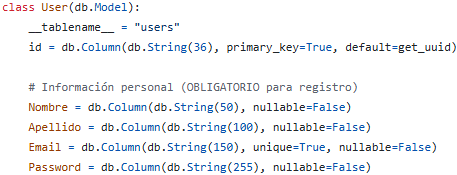
Ventajas del uso de UUID:

* Garantiza unicidad global en todos los registros.
* Aumenta la seguridad al no exponer IDs secuenciales.
* Facilita integraciones con APIs externas (como el bot de WhatsApp).

**Modelo de tabla: User**

La clase User representa a los clientes registrados en la plataforma.

Contiene campos obligatorios y opcionales, organizados por categoría (información personal, contacto, dirección y auditoría).



**Campos principales:**

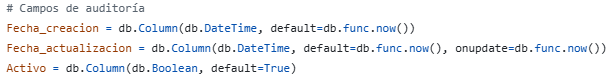
|  |  |  |
| --- | --- | --- |
| Campo | Tipo de dato | Descripción |
| id | String (UUID) | Identificador único del usuario. |
| Nombre, Apellido | String | Información personal obligatoria. |
| Email | String (único) | Correo electrónico utilizado para el inicio de sesión. |
| Password | String | Contraseña cifrada mediante bcrypt. |

**Campos opcionales de contacto y dirección:**

* Telefono, Fecha\_nacimiento, Genero
* Direccion, Ciudad, Estado\_provincia, Codigo\_postal, Pais, Tipo\_direccion

Estos campos permiten personalizar el perfil del cliente y facilitar la gestión de envíos o notificaciones.  
También se incluye un campo Foto\_perfil para almacenar la URL de imagen asociada al usuario.

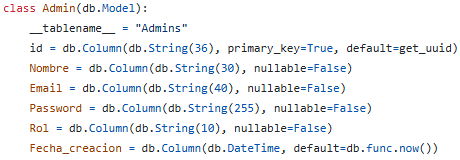
**Campos de Auditoria**

****

Estos campos registran automáticamente la fecha y hora de creación o actualización del usuario, y permiten activar o desactivar cuentas sin eliminarlas físicamente.

**Modelo de tabla: Admin**

La clase Admin define la estructura para los administradores del sistema, responsables de gestionar el catálogo de productos, pedidos y usuarios.



**Integración con el sistema**

Ambos modelos (User y Admin) están relacionados indirectamente con el archivo app.py, donde se realizan las operaciones CRUD (crear, leer, actualizar, eliminar).

Cuando el sistema se inicializa, Flask ejecuta automáticamente:

**Conclusión del módulo**

El archivo models.py cumple una función clave dentro de la arquitectura del sistema:

* Define de manera centralizada la estructura de los datos.
* Separa la lógica del modelo respecto al controlador (app.py).
* Garantiza la integridad y consistencia de la información.
* Facilita la migración a entornos remotos como Amazon RDS.

Este diseño modular permite que tanto los usuarios finales como los administradores interactúen con la base de datos de forma segura y eficiente.

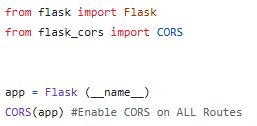
**Archivo de prueba y ejecución del servidor (main.py)**

El archivo main.py tiene la función principal de verificar la correcta inicialización del servidor Flask y validar la respuesta de las rutas base.

Este módulo se empleó en la primera fase de desarrollo del sistema para probar la comunicación entre el servidor backend y el frontend.

**Configuración inicial del servidor**

El siguiente bloque crea una instancia del framework Flask y habilita CORS para permitir la comunicación con clientes web, como la aplicación en React.

**Explicación:**

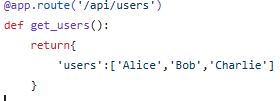
Flask(\_\_name\_\_): inicializa la aplicación.

CORS(app): permite que el servidor acepte solicitudes desde dominios externos, requisito necesario para que el frontend React pueda consumir las APIs.

**Definición de rutas de prueba**

Este archivo define tres rutas principales para realizar pruebas básicas de conexión y respuesta JSON.

Esta ruta responde con un texto simple, confirmando que el servidor Flask está corriendo correctamente.



Esta ruta devuelve una lista en formato JSON simulando un conjunto de usuarios.  
Su propósito es comprobar que las respuestas se devuelven correctamente y que el frontend puede interpretarlas.



**Ejecución del servidor**

Finalmente, el servidor se ejecuta en modo desarrollo para pruebas locales:

Conclusión del módulo

El archivo main.py cumple la función de prototipo de servidor dentro del ciclo de desarrollo.

Fue utilizado para:

* Verificar que Flask estaba correctamente instalado y funcionando.
* Probar la comunicación HTTP con el frontend (CORS).
* Asegurar que las rutas devuelven respuestas en formato JSON.

Aunque no forma parte del backend final, sirvió como base para validar la estructura del entorno antes de integrar los módulos más complejos (app.py y models.py).

## Configuración del entorno

## Integración del bot de WhatsApp

# Pruebas del sistema

## Tipos de pruebas

## Casos de prueba

## Resultados de las pruebas

# Resultados

# Conclusiones y recomendaciones

# Bibliografía

# Anexos

## Manual de usuario

## Manual técnico

## Código fuente (opcional)