PROYECTO INTEGRADOR
CODING
WITH AI

Contenido

Proyecto integrador

Desarrollar una aplicación móvil (web) que integre los conocimientos adquiridos en cada módulo del curso. El proyecto final será una recopilación de los diferentes aprendizajes adquiridos durante el curso y los casos prácticos.

Descripción general:

Consolidar el desarrollo de una aplicación de tipo e-commerce que refleje un entendimiento profundo de los temas del curso, incluyendo diseño y desarrollo de base de datos, desarrollo backend y frontend impulsado por IA, integración de modelos de IA, y prácticas de seguridad y optimización de

Estructura:

Planificación y diseño:

Tomando como referencia lo aprendido en el Módulo 1, definir la idea de la aplicación, su propósito principal.

GreenHub

Marketplace de productos de segunda mano, reutilizados y upcycled, con un enfoque en la economía circular.

Propósito principal:

- Reducir el desperdicio masivo de recursos fomentando la compra-venta de productos usados en buen estado.
- Ofrecer una alternativa económica y ecológica al consumo tradicional.

Público objetivo

Consumidores eco-conscientes (18-50 años) que buscan reducir su huella ambiental sin gastar mucho (ej.: millennials y Gen Z preocupados por el fast fashion o la obsolescencia programada).

Vendedores independientes: Tiendas de thrift, artesanos de upcycling, dueños de talleres de reparación (ej.: electrónica o muebles).

Empresas con excedentes: Marcas que quieran vender stock no vendido o devoluciones en lugar de destruirlo.

Diseñar un esquema inicial para la base de datos y la arquitectura general de la aplicación usando la IA para generar recomendaciones y/o sugerencias.

Entidades principales:

Tabla	Descripción
users	Almacena los datos de los usuarios
products	Información de productos en venta
orders	Compras realizadas por los usuarios
reviews	Opiniones y puntuaciones de productos
categories	Categorías para clasificar los productos

- Un usuario puede vender **muchos productos** (1 a N).

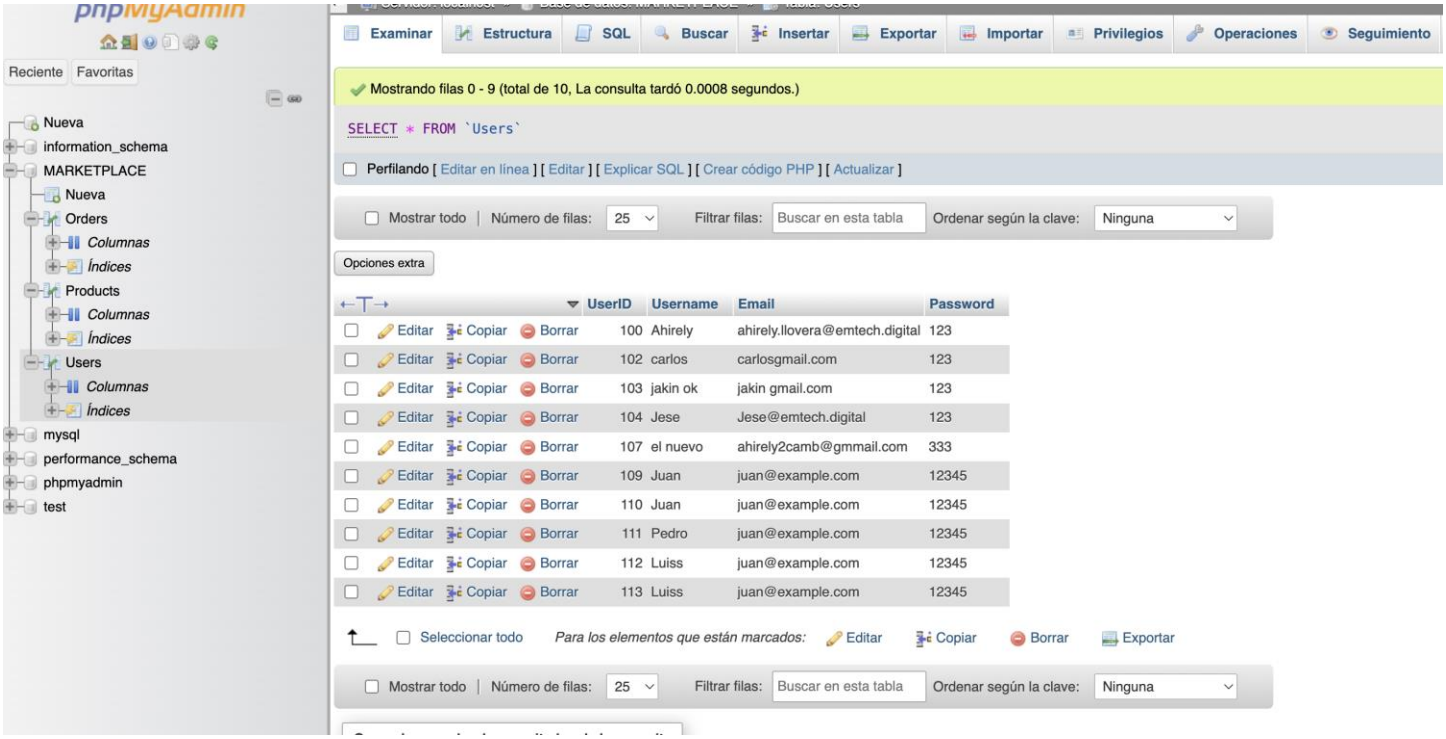
- Un usuario puede hacer **muchas órdenes**.
- Un producto puede tener **muchas reviews**.
- Un producto **pertenece a una categoría**.

Tipos de relaciones en el diseño

Relación	Tipo	Comentario
productos.id_vendedor → usuarios.id_usuario	Uno a muchos	Un usuario puede tener muchos productos
productos.id_categoria → categorias.id_categoria	Uno a muchos	Una categoría puede tener muchos productos
pedidos.id_comprador → usuarios.id_usuario	Uno a muchos	Un usuario puede hacer muchos pedidos
pedidos.id_producto → productos.id_producto	Uno a muchos	Un producto puede estar en muchos pedidos
reseñas.id_producto → productos.id_producto	Uno a muchos	Un producto puede tener muchas reseñas
reseñas.id_usuario → usuarios.id_usuario	Uno a muchos	Un usuario puede dejar muchas reseñas

Desarrollo de la base de datos:

Aplicando los conocimientos del Módulo 3, se deberá implementar y configurar la base de datos del proyecto, incluyendo tablas y relaciones necesarias. Se puede tomar como base y/o referencia, la base de datos utilizada en el Caso Práctico 3.



Ejemplo, desde SQL.

```
-- Tabla de usuarios: contiene compradores y vendedores
CREATE TABLE usuarios (
    id_usuario INT AUTO_INCREMENT PRIMARY KEY,
    nombre_usuario VARCHAR(50) UNIQUE NOT NULL,
    correo VARCHAR(100) UNIQUE NOT NULL,
    contraseña VARCHAR(100) NOT NULL
);

-- Tabla de categorías: cada producto puede pertenecer a una categoría
CREATE TABLE categorias (
    id_categoria INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(50) UNIQUE NOT NULL
);

-- Tabla de productos: cada producto tiene un vendedor y una posible categoría
CREATE TABLE productos (
    id_producto INT AUTO_INCREMENT PRIMARY KEY,

    -- RELACIÓN: muchos productos pueden pertenecer a un usuario vendedor
    id_vendedor INT NOT NULL,
```

```
    FOREIGN KEY (id_vendedor) REFERENCES usuarios(id_usuario),

    -- RELACIÓN: muchos productos pueden pertenecer a una categoría
    id_categoria INT,
    FOREIGN KEY (id_categoria) REFERENCES categorias(id_categoria),

    nombre VARCHAR(100) NOT NULL,
    descripcion TEXT,
    precio DECIMAL(10, 2) NOT NULL
);

-- Tabla de pedidos: cada pedido está hecho por un comprador y está asociado a un producto
CREATE TABLE pedidos (
    id_pedido INT AUTO_INCREMENT PRIMARY KEY,

    -- RELACIÓN: muchos pedidos pueden ser hechos por el mismo comprador
    id_comprador INT NOT NULL,
    FOREIGN KEY (id_comprador) REFERENCES usuarios(id_usuario),

    -- RELACIÓN: muchos pedidos pueden referirse al mismo producto
    id_producto INT NOT NULL,
    FOREIGN KEY (id_producto) REFERENCES productos(id_producto),

    cantidad INT NOT NULL,
    fecha_pedido DATE NOT NULL
);
```

- Desarrollo del Backend:

Basado en el Módulo 4, será necesario desarrollar la lógica backend de la

- aplicación, incluyendo la creación de servicios REST API principalmente en Python o JavaScript.
- Será importante garantizar que se integre el backend con la base de datos para poder realizar las distintas operaciones necesarias que se trabajaron en el Caso Práctico 4:
  1. Altas: crear nuevos registros (usuarios, productos, pedidos).
  2. Bajas: eliminar registros existentes.
  3. Modificaciones: actualizar datos de registros existentes.

Estructura:

```python
class User(db.Model):
    __tablename__ = 'users'
    UserID = db.Column(db.Integer, primary_key=True)  # Nombre correcto de l
    username = db.Column(db.String(50), unique=True, nullable=False)
    email = db.Column(db.String(50), unique=True, nullable=False)
    password = db.Column(db.String(50), nullable=False)

    products = db.relationship('Product', backref='seller', lazy=True)
    orders = db.relationship('Order', backref='buyer', lazy=True)
```

```python
class User(db.Model):...
@app.route('/')
def home():...
@app.route('/users', methods=['GET'])
def get_users():...
@app.route('/users', methods=['POST'])
def add_user():...
@app.route('/users/<int:user_id>', methods=['PUT'])
def update_user(user_id):...

@app.route('/users/<int:user_id>', methods=['DELETE'])
def delete_user(user_id):...
```

```python
class Product(db.Model):
    __tablename__ = 'products'
    ProductID = db.Column(db.Integer, primary_key=True)
    SellerID = db.Column(db.Integer, db.ForeignKey('users.UserID'), nullable=False)
    ProductName = db.Column(db.String(50), nullable=False)
    Description = db.Column(db.Text, nullable=False)
    Price = db.Column(db.Numeric(10, 2), nullable=False)

    # Relación con Order
    orders = db.relationship('Order', backref='product', lazy=True)
```

```python
class Product(db.Model): ...

@app.route('/products', methods=['GET'])
def get_products(): ...
@app.route('/products', methods=['POST'])
def add_product(): ...
@app.route('/products/<int:product_id>', methods=['PUT'])
def update_product(product_id): ...

@app.route('/products/<int:product_id>', methods=['DELETE'])
def delete_product(product_id): ...
```

```python
#Orders
class Order(db.Model):
    __tablename__ = 'orders'
    OrderID = db.Column(db.Integer, primary_key=True)
    BuyerID = db.Column(db.Integer, db.ForeignKey('users.UserID'), nullable=False)
    ProductID = db.Column(db.Integer, db.ForeignKey('products.ProductID'), nullable=False)
    Quantity = db.Column(db.Integer, nullable=False)
    OrderDate = db.Column(db.DateTime, nullable=False)
```

```python
class Order(db.Model): ...

@app.route('/orders', methods=['GET'])
def get_orders(): ...
@app.route('/orders', methods=['POST'])
def add_order(): ...
@app.route('/orders/<int:order_id>', methods=['PUT'])
def update_order(order_id): ...

@app.route('/orders/<int:order_id>', methods=['DELETE'])
def delete_order(order_id): ...
```

Desarrollo del frontend:

- Siguiendo el Módulo 5, diseñar y desarrollar el frontend de la aplicación, ya sea móvil o web, utilizando Kivy o tecnologías similares.

```
class ProductPanel(BoxLayout):
    def __init__(self, **kwargs):
        super().__init__(orientation='vertical', **kwargs)

        self.add_widget(Label(text='Nombre del Producto:'))
        self.product_name = TextInput(multiline=False)
        self.add_widget(self.product_name)

        self.add_widget(Label(text='Descripción:'))
        self.product_description = TextInput(multiline=False)
        self.add_widget(self.product_description)

        self.add_widget(Label(text='Precio:'))
        self.product_price = TextInput(multiline=False)
        self.add_widget(self.product_price)

        self.add_product_button = Button(text='Guardar Producto')
        self.add_product_button.bind(on_press=self.save_product)
        self.add_widget(self.add_product_button)

        self.view_products_button = Button(text='Ver Productos en
Consola')
        self.view_products_button.bind(on_press=self.show_products)
        self.add_widget(self.view_products_button)

        self.product_list_label = Label(text='Estado: Esperando
acción...')
        self.add_widget(self.product_list_label)
```

- Implementar funcionalidades clave como el inicio de sesión y la visualización de productos.

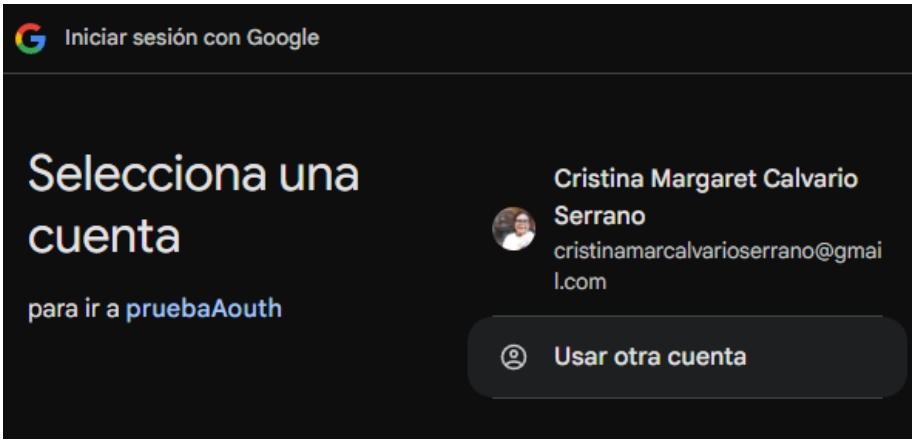
Use la librería sqlite3, pero si quisiera manejar los datos desde un servidor, debo isa SQLAlchemy + PyMySQL. Tiene interfaz grafica Kivy. Contiene Machine Leraning, pandas y scikit-learn.

Inicio de sesión

google_oauth.py > ...

```python
from requests_oauthlib import OAuth2Session

client_id = "906555750937-mef9o1b0tfu6hinvs0moip4gj90evehs.apps.googleusercontent.com"
client_secret = "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"
redirect_uri = "http://localhost"

# Definir los alcances (permisos) necesarios
scope = ["https://www.googleapis.com/auth/userinfo.email", "https://www.googleapis.com/auth/userinfo.profile"]

# Crear sesión OAuth con el scope correcto
google = OAuth2Session(client_id, redirect_uri=redirect_uri, scope=scope)

authorization_url, state = google.authorization_url(
    "https://accounts.google.com/o/oauth2/auth",
    access_type="offline",
    prompt="consent"
)

print(f"Inicia sesión en: {authorization_url}")
```

```
Inicia sesión en: https://accounts.google.com/o/oauth2/auth?response_type=code&client_id=906555750937-mef9o1b0tfu6hinv
s0moip4gj90evehs.apps.googleusercontent.com&redirect_uri=http%3A%2F%2Flocalhost&scope=https%3A%2F%2Fwww.googleapis.com
%2Fauth%2Fuserinfo.email+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fuserinfo.profile&state=MZbacSipd5gHirQjKshNK7LfThbq
Gx&access_type=offline&prompt=consent
```



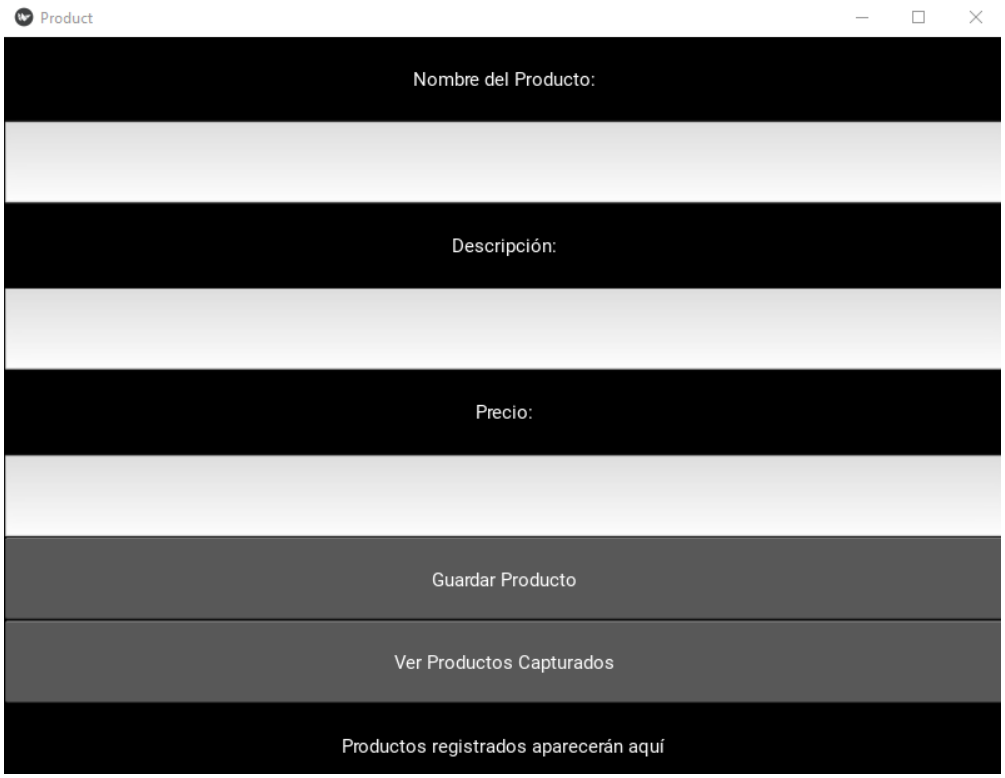
- Se deberán utilizar como mínimo las tres interfaces trabajadas en el Caso Práctico 5, donde la primera consiste en crear un inicio de sesión con OAUTH de Google,

  la segunda se basa en crear un panel donde se registran los productos

La tercera servirá para presentar los productos para una compra y adición al carrito.

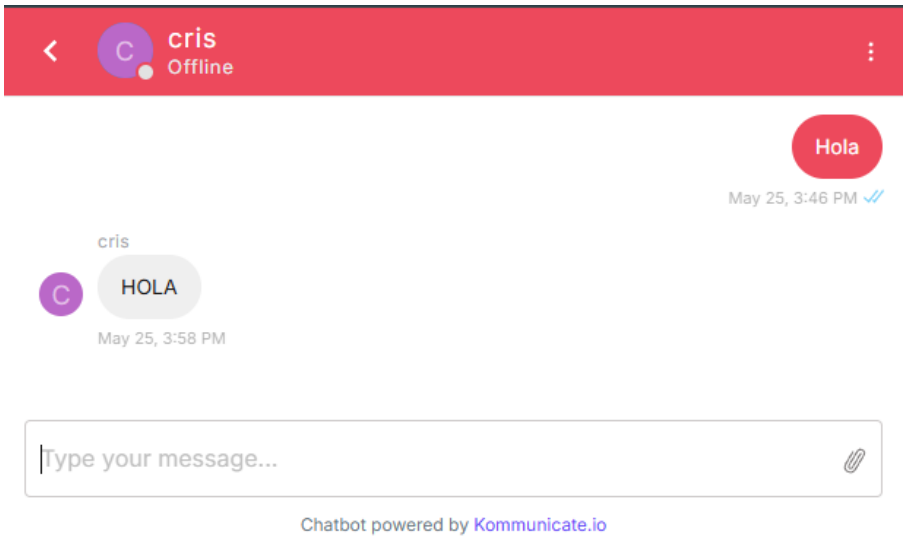
Integración de modelos de IA:

Registro de productos



Chatbot

```python
import webbrowser
from kivy.app import App
from kivy.uix.button import Button
from kivy.uix.boxlayout import BoxLayout

class ChatbotApp(App):
    def build(self):
        layout = BoxLayout(orientation='vertical')

# Botón para abrir el chatbot en el navegador
        open_button = Button(text="Abrir Chatbot", size_hint=(1, 0.1))
        open_button.bind(on_press=self.open_chatbot)
        layout.add_widget(open_button)
        return layout

    def open_chatbot(self, instance):


# Abre el chatbot en el navegador predeterminado
# webbrowser.open("https://widget.kommunicate.io/chat")
        webbrowser.open("https://widget.kommunicate.io/chat?appId=1ef0018ec23321811284bcdb1b4194c87")


if __name__ == '__main__':
    ChatbotApp().run()
```

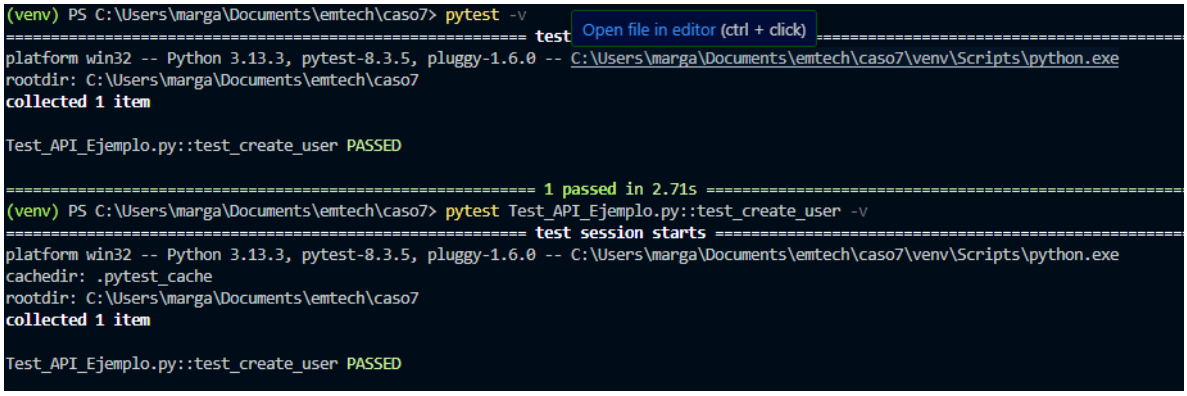


Carrito de producto

Test

Test_API_Ejemplo.py > client

```python
import pytest
from app import app, db
from models import User

@pytest.fixture
def client():
    app.config['TESTING'] = True
    app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///:memory:'
    app.config['WTF_CSRF_ENABLED'] = False  # Deshabilitar CSRF para testing

    with app.test_client() as client:
        with app.app_context():
            db.create_all()
        yield client
        with app.app_context():
            db.drop_all()

def test_create_user(client):
    response = client.post('/users', json={
        'username': 'testuser',
        'email': 'test@example.com',
        'password': 'testpass'
    })
    assert response.status_code == 201
    assert response.json['message'] == 'Usuario creado con éxito'
```



Optimización y seguridad:

Aplicar las técnicas aprendidas en los Módulos 8 y 9 para optimizar el código usando los asistentes trabajados en el curso y asegurar que la aplicación esté protegida contra vulnerabilidades básicas y comunes.

Del Modulo **8.Monitoreo**…

```
# Importamos las librerías necesarias
import requests # Para hacer solicitudes HTTP a una URL
import time # Para medir el tiempo y pausar el monitoreo

def get_response_time(url):
"""
Realiza una solicitud HTTP GET a la URL proporcionada y calcula
cuánto tiempo tardó en obtener la respuesta.
"""
# requests.get(url) envía una solicitud GET, HTTP GET a la URL
response = requests.get(url)

# Obtenemos el tiempo de respuesta en segundos desde que se envió hasta que se recibió
# .elapsed es un objeto timedelta, y .total_seconds() lo convierte a segundos flotantes
response_time = response.elapsed.total_seconds()

# Retornamos el tiempo de respuesta
return response_time

def monitor_performance(url, threshold):
"""
Verifica el rendimiento de una URL comparando el tiempo de respuesta
con un umbral definido por el usuario.
"""
# Llama a la función que obtiene el tiempo de respuesta actual
response_time = get_response_time(url)
```

```
# Compara el tiempo de respuesta con el umbral definido
if response_time > threshold:
# Si se excede el umbral, se imprime una alerta con la información relevante
print(
"ALERTA: El tiempo de respuesta de la URL '{}' es de {:.2f} segundos. "
"Esto supera el umbral de {:.2f} segundos.".format(url, response_time, threshold)
)
else:
# Si está dentro del rango normal, se muestra un mensaje informativo
print(
"OK: El tiempo de respuesta de '{}' es {:.2f} segundos, dentro del umbral.".format(url,
response_time)
)

if __name__ == "__main__":
"""
Bloque principal del programa: define la URL a monitorear y el umbral
y ejecuta el monitoreo en un bucle continuo.
"""
# Definimos la URL de la aplicación web a monitorear
#url = "https://www.google.com"
url = "https://www.example.com"
# Definimos el umbral de tiempo de respuesta permitido (en segundos)
threshold = 2.0

# Inicia un bucle infinito que monitorea la URL cada segundo
while True:
# Llama a la función que evalúa el rendimiento
monitor_performance(url, threshold)

# Espera 1 segundo antes de volver a comprobar
```
- time.sleep(1.0)

Estructura:

Primero se planifico el proyecto, se obtuvo un carrito de productos. Se hizo el diseño de la base de datos y se creó. Posteriormente se hizo el código del backend, usando apis, luego se realizó el frontend usando la tecnología de Kivy. También se Outh para el inicio de sesión y un chatbot para la asistencia técnica del carrito. Por ultimo se hicieron pruebas de monitoreo y test de su funcionalidad y para la seguridad.

Entregables a considerar:

Aplicación móvil o web completamente funcional en una carpeta .ZIP con todos los assets y códigos necesarios.
Documentación técnica del desarrollo del proyecto.