Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, Vacaciones de junio 2024

PROYECTO 2 IPC2 – IPCMARKET ONLINE

202300434 – Cerezo Paredes, Fátima Florisel

202300512 – Hernandez Flores, Daniel Andreé

Resumen

IPCMarket es una aplicación de comercio electrónico que consta de un servicio frontend web y un backend API. El sistema permite a los usuarios iniciar sesión como administradores o compradores, manejar productos, realizar compras y ver estadísticas. Los administradores pueden cargar datos masivamente, gestionar productos y generar reportes, mientras que los compradores pueden navegar por productos, agregarlos al carrito y completar compras. La aplicación utiliza archivos XML para almacenar y procesar datos de usuarios, productos, empleados y actividades.

El backend procesa las solicitudes del frontend, gestiona la persistencia de datos y proporciona endpoints para diversas funcionalidades. IPCMarket también ofrece características estadísticas y reportes, todo ello implementado utilizando una arquitectura cliente-servidor con Python, Flask para el backend y Django para el frontend.

Palabras clave
XML, POO, Python, Backend, Frontend.

Abstract

IPCMarket is an e-commerce application consisting of a frontend web service and a backend API. The system allows users to log in as admins or buyers, manage products, make purchases and view statistics. Administrators can bulk upload data, manage products, and generate reports, while shoppers can browse products, add them to the cart, and complete purchases. The application uses XML files to store and process user, product, employee and activity data.

The backend processes frontend requests, manages data persistence and provides endpoints for various features. IPCMarket also offers statistical features and reports, all implemented using a client-server architecture with Python, Flask for backend and Django for frontend.

Keywords
XML, POO, Python, TDA, Backend, Frontend.

Introducción

IPCmarket, inicialmente concebida como una aplicación de escritorio destinada a facilitar la compra diaria de productos de consumo, se ha transformado en una plataforma más accesible y dinámica con su nueva iteración como aplicación web. Esta transición responde a la necesidad de expandir el alcance del servicio, permitiendo un acceso más amplio y flexible para los usuarios a través de la web.

La aplicación ha sido diseñada para ofrecer una experiencia de usuario interactiva y comprensible, destacando la visualización de estadísticas relacionadas con productos, usuarios y transacciones en forma de gráficos dinámicos. Además, IPCmarket busca mejorar la interacción con los usuarios al permitirles gestionar compras, ver productos disponibles, y acceder a reportes y estadísticas detalladas sobre las actividades de la plataforma.

Al migrar de un modelo local a uno basado en web, IPCmarket no solo amplía su funcionalidad, sino que también refuerza su compromiso con la accesibilidad y la mejora continua de la experiencia de compra en línea para todos sus usuarios.

Desarrollo del tema

La Programación Orientada a Objetos (POO) es fundamental para el desarrollo de aplicaciones modernas porque puede modelar elementos del mundo real de manera intuitiva. La POO de IPCmarket facilita la reutilización y la modularidad del código, lo que es esencial para gestionar el complejo sistema de compra de productos. Cada entidad del dominio, como los usuarios, los productos y las compras, se representa como un objeto con estados y comportamientos particulares. Al mejorar la legibilidad del código y reducir la probabilidad de errores, esto mejora la escalabilidad y el mantenimiento del sistema.

Backend

El backend de IPCmarket está desarrollado utilizando Flask, un microframework de Python que facilita la creación de aplicaciones web. Al inicio del código, se configura Flask junto con CORS (Cross-Origin Resource Sharing), que permite a la aplicación aceptar solicitudes desde diferentes dominios, lo cual es esencial para aplicaciones web que interactúan con clientes en diferentes ubicaciones o en desarrollo local.

Para manejar la persistencia de los datos, se utilizan varios archivos XML, cada uno destinado a almacenar diferentes tipos de datos como usuarios, productos, empleados, actividades, etc. Al arrancar la aplicación, se asegura que exista un directorio data/uploads donde estos archivos residen. Si algún archivo no existe al momento de iniciar la aplicación, se crea un archivo XML vacío correspondiente. Esto asegura que la aplicación pueda funcionar correctamente desde el inicio sin necesidad de estructuras de datos preexistentes.

El backend proporciona endpoints como /carga_masiva_usuarios y /carga_masiva_productos para permitir la carga masiva de datos desde archivos XML. Estos endpoints incluyen validaciones para asegurar la unicidad de los identificadores y la corrección de los formatos de datos como emails y números de teléfono. Además, se ofrecen endpoints para recuperar la información almacenada, como /get_users y /get_products, que facilitan la visualización y gestión de los datos en la aplicación.

Este backend es un componente crucial de IPCmarket, permitiendo la interacción dinámica entre la interfaz de usuario y los datos almacenados. Utiliza tecnologías eficientes para el manejo de datos y proporciona una base sólida para la expansión de la aplicación, garantizando al mismo tiempo la seguridad y la integridad de los datos.

Tabla 1

Figura 1. Tabla de endpoints.

/login	POST	Autenticación de usuarios.
/carga_masiva_usuarios	POST	Carga masiva de datos de usuarios desde un archivo XML.
/get_users	GET	Recupera todos los usuarios registrados.
/carga_masiva_productos	POST	Carga masiva de datos de productos desde un archivo XML.
/get_products	GET	Recupera todos los productos registrados.
/carga_masiva_empleados	POST	Carga masiva de datos de empleados desde un archivo XML.
/get_employees	GET	Recupera todos los empleados registrados.
/carga_masiva_actividades	POST	Carga masiva de actividades desde un archivo XML.
/get_activities	GET	Recupera todas las actividades registradas.
/add_cart	POST	Agrega productos al carrito de compras.
/descarga_carrito	GET	Descarga el archivo del carrito de compras.
/comprar	POST	Realiza la compra de los productos en el carrito.
/descarga_compras	GET	Descarga el archivo de compras realizadas.
/descarga_actividades_hoy	GET	Descarga las actividades del día actual.
/admin	GET	Acceso protegido para administradores.
/categorias_estadisticas	GET	Estadísticas de categorías más populares.
/productos_con_mas_cantidad	GET	Productos con mayor cantidad disponible.

El programa podrá ser totalmente con el tipo de archivo XML utilizados en la aplicación de escritorio, por lo que únicamente se detallará las validaciones del programa:

a) Usuarios:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<usuarios>
    <usuario id="1" password="12345">
        <nombre>Juan Perez</nombre>
        <edad>30</edad>
        <email>juan.perez@example.com</email>
        <telefono>12345678</telefono>
    </usuario>
```

Figura 2. Estructura del archivo XML de entrada.

Fuente: elaboración propia, 2024.

Donde:

“id” no se puede repetir entre usuarios. “email” contiene la validación de cumplir con una estructura valida de un correo electrónico.

“telefono” debe cumplir con el requisito de tener 8 dígitos.

b) Empleados:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<empleados>
    <empleado codigo="001">
        <nombre>Ana Garcia</nombre>
        <puesto>Gerente</puesto>
    </empleado>
```

Figura 3. Estructura del archivo XML de entrada.

Fuente: elaboración propia, 2024.

Donde:

“codigo” no se puede repetir entre empleados.

c) Productos

```xml
<productos>
    <producto id="3">
        <nombre>Laptop DEF</nombre>
        <precio>750.00</precio>
        <descripcion>Laptop DEF de alta gama</descripcion>
        <categoria>Electrónica</categoria>
        <cantidad>8</cantidad>
        <imagen>imagen3.jpg</imagen>
    </producto>
```

Figura 4. Estructura del archivo XML de entrada.

Fuente: elaboración propia, 2024.

Donde:

“id” no se puede repetir entre productos.

“precio” de cumplir con tener decimales.

“categoria” se utiliza para la creación de las gráficas.

“cantidad” es la etiqueta que nos indica a cuantas unidades del producto hay en existencia en el sistema, y debe ser un entero.

d) Actividades:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<actividades>
    <actividad id="101">
        <nombre>Reunión de Ventas</nombre>
        <descripcion>Reunión mensual del equipo de ventas</descripcion>
        <empleado>001</empleado>
        <dia hora="7">1</dia>
    </actividad>
```

Figura 5. Estructura del archivo XML de entrada.

Fuente: elaboración propia, 2024.

Donde:

“id” no se puede repetir entre actividades.

Frontend:

El código proporcionado utiliza Django, unframework de alto nivel para Python que fomenta el desarrollo rápido de aplicaciones web seguras y mantenibles. Las vistas son funciones o clases en Django que toman una solicitud web y devuelven una respuesta. Este conjunto de vistas maneja la interacción con un backend definido previamente, utilizando el framework Flask para gestionar datos mediante APIs RESTful.

Gracias al framework es mucho más fácil obtener las repuesta del *backend* y manejarlas en la interfaz, implementando la librería *request*. Además que el código se encuentra optimizado implementando el desarrollo por bloques y la herencia entre pestañas.

Tabla 2

Figura 6. Tabla de endpoints.

Función de Vista	**Descripción**	**Método HTTP**
login	Autentica usuarios y redirige según el rol.	POST
protected	Vista protegida accesible solo por el administrador.	GET
carga_usuarios	Carga masiva de usuarios desde un archivo.	POST, GET
productos_view	Muestra los productos disponibles para usuarios.	GET
productos_view_admin	Muestra los productos disponibles para el administrador.	GET
producto_detalle_view	Detalles y acciones de un producto específico.	POST, GET
producto_detalle_view_admin	Detalles de un producto para el administrador.	GET
descarga_carrito	Redirige para la descarga del archivo del carrito.	GET
descarga_compras	Redirige para la descarga del archivo de compras.	GET
carga_productos	Carga masiva de productos desde un archivo.	POST, GET

actividades	Carga masiva de actividades desde un archivo.	POST, GET
carga_empleados	Carga masiva de empleados desde un archivo.	POST, GET
estadisticas	Muestra estadísticas (función implementada pero no descrita).	GET
reportes	Gestiona la descarga de reportes de compras o actividades.	POST, GET
colaboradores	Muestra información sobre colaboradores.	GET
docu	Proporciona acceso a la documentación	GET

Conclusiones

El proyecto IPCmarket, evolucionado de una aplicación de escritorio a una plataforma web, demuestra ser una solución integral para el comercio electrónico, utilizando Flask y Django para una gestión eficaz de usuarios y productos. Este sistema no solo facilita operaciones complejas como la gestión de carritos y la autenticación de usuarios, sino que también proporciona herramientas poderosas para la administración y análisis de datos, garantizando seguridad y eficiencia en un entorno de comercio electrónico moderno y accesible.

Referencias bibliográficas

Django documentation | Django documentation. (s. f.). Django Project. https://docs.djangoproject.com/en/5.0/

Cross-Origin Resource Sharing (CORS) - HTTP | MDN. (2024, 21 junio). MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

xml.etree.ElementTree — The ElementTree XML API. (s. f.). Python Documentation. https://docs.python.org/3/library/xml.etree.elementtree.html

USAC, T.d (26 de Junio de 2024), Proyecto 2, UEDi. Obtenido de: https://uedi.ingenieria.usac.edu.gt/campus/course/view.php?id=17518

Anexos

Diagrama de clases:

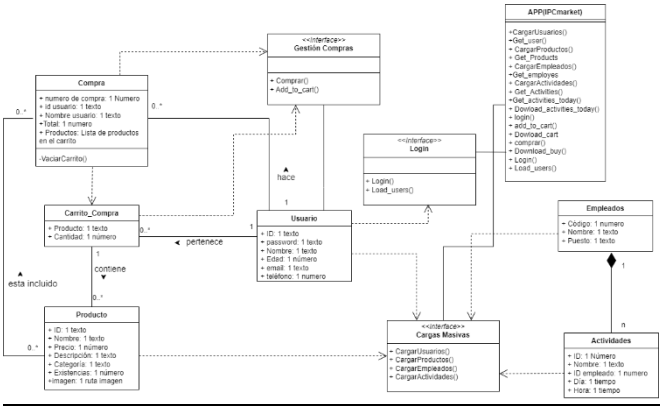

Figura 7. Diagrama de clases

Fuente: elaboración propia, 2024.

Diagrama de componentes:

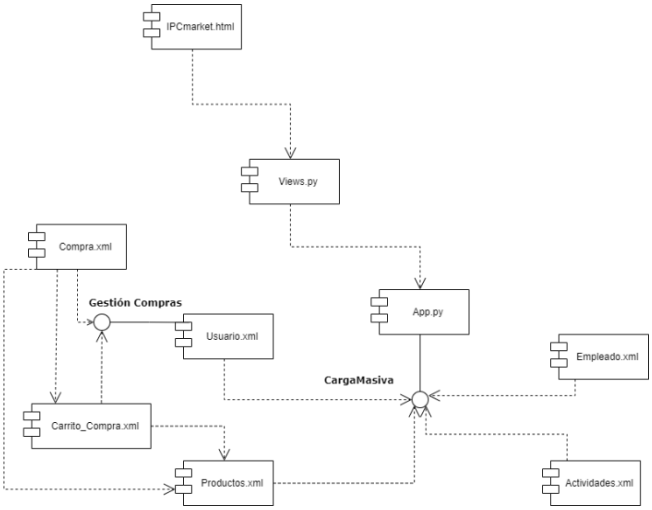

Figura 8. Diagrama de componentes

Fuente: elaboración propia, 2024.

Diagrama de despliegue:

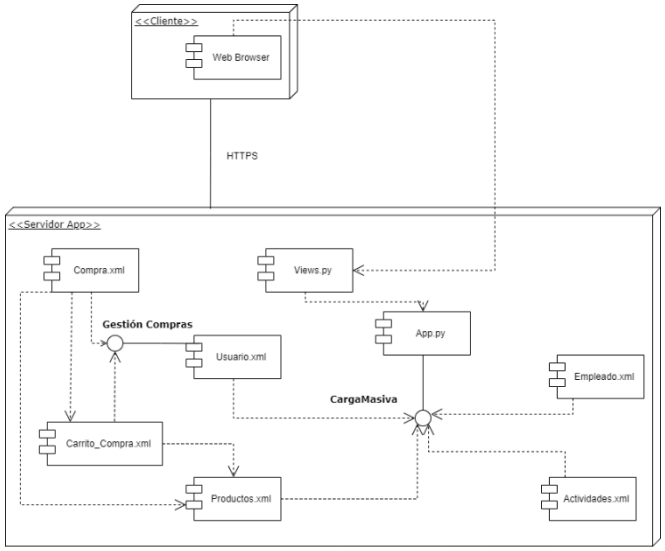

Figura 9. Diagrama de despligue

Fuente: elaboración propia, 2024.