Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 1er. Semestre 2024

PROYECTO 2. IPC2 MARKET

201701010– Bryant Herrera Rubio

Resumen

El programa desarrollado permite a los usuarios acceder mediante autenticación, ofreciendo diferentes roles como administrador. Una vez autenticado, el administrador tiene acceso a varias funciones clave. Una de las más destacadas es la capacidad de realizar carga masiva de datos desde archivos XML, facilitando la incorporación eficiente de usuarios, productos, actividades y empleados al sistema. Esto no solo simplifica el proceso de gestión de datos, sino que también garantiza la integridad y precisión de la información introducida.

Finalmente, para los usuarios finales, se ha implementado la capacidad de realizar compras directamente desde la interfaz de usuario, facilitando una experiencia intuitiva y eficiente para la adquisición de productos disponibles. En conjunto, mejorando significativamente la eficiencia y la experiencia de usuario en diversas operaciones administrativas y comerciales.

Palabras clave

Django

Manejo de datos

Flask

Frontend

Backend

Abstract

The application is designed to manage a login and administration system. The admin.py file defines the AdminWindow class, which sets up an interface for administrative tasks, including CRUD operations for products. The login.py file contains the LoginWindow class, which establishes a login window where users enter their credentials. If authenticated as an administrator or normal user, a success message is displayed, and the login window closes. In case of incorrect credentials, an error message is shown. Finally, main.py initiates the application by creating an instance of LoginWindow when executed as the main program, facilitating user interaction with the login interface. This modular design enables efficient management of users and administrators within the application.

Keywords

Django

Data Handling

Flask

Frontend

Backend.

Introducción

En el ámbito de la programación y la informática, el manejo eficiente de listas y colas es fundamental para la organización y procesamiento de datos. Las listas, estructuras de datos flexibles que permiten almacenar elementos secuenciales, facilitan operaciones como la inserción, eliminación y búsqueda. Por otro lado, las colas, estructuras FIFO (primero en entrar, primero en salir), son esenciales para la gestión ordenada de elementos, especialmente en situaciones donde se requiere un orden estricto de procesamiento. Además, herramientas como Graphviz ofrecen recursos poderosos para visualizar relaciones y estructuras de datos complejas, facilitando así el análisis y la comprensión de información interrelacionada. Este ensayo explora la importancia de estas herramientas y técnicas en la programación, destacando su papel en la optimización y eficiencia de los algoritmos y aplicaciones informáticas.

Desarrollo del tema

Desarrollo Detallado

1. Uso de clases para almacenar los archivos

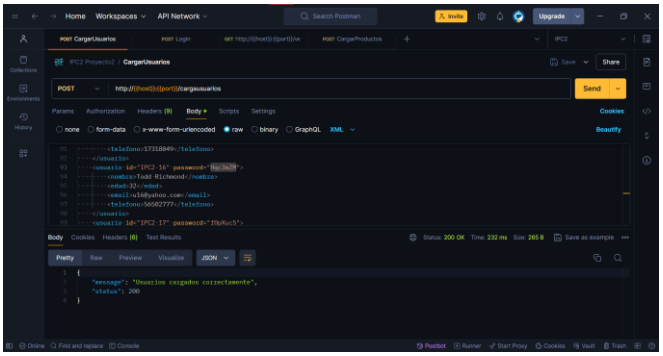
En el proyecto IPC 2 MARKET, se emplean estructuras de datos avanzadas como las listas circulares y ortogonales para gestionar eficientemente la información. Se utilizaron varias clases en donde se irán pasando información todo en formato xml, algunas ocaciones lo convierte en json, como las conexiones entre usuarios y sus preferencias de compra, optimizando consultas y actualizaciones cruzadas. Cada nodo en estas listas puede estar vinculado a múltiples nodos en otras listas, lo que facilita la gestión de relaciones complejas de datos.

Clases y Estructuras de Datos Utilizadas

BackendFrontend

Aquí se usaron varias clases en donde se puede almacenar información, todo esto se puede comprobar mediante el uso de POSTMAN para poder comprobar la funcionalidad.

Se hizo uso de FLASK para poder trabajar en el proyecto en este caso para el backend, se tiene que seguir una serie de pasos para poder trabajar, usando esta herramienta.



Frontend

En esta parte se puede visualizar lo que puede ver el usuario, en este proceso se utilizó el FRAMEWORK Django.

Para poder hacer uso de esta herramienta se es necesario seguir una serie de varios pasos, este nos crea automáticamente una carpeta donde se encuentran varias carpetas y archivos que es para el manejo de la página web

Clases que se usaron:

Para poder llegar hacer que el programa funcione se es necesario hacer uso de varias clases que permiten el funcionamiento del programa.

Actividadescontroller.py

Blueprint_actividad: Define un Blueprint en Flask para manejar las rutas relacionadas con actividades.

cargarActividades(): Maneja la carga de actividades desde un XML recibido por una solicitud POST. Lee los datos XML, crea objetos Actividad, los añade a una lista actividades, y actualiza un archivo XML existente o crea uno nuevo si no existe.

precargar_actividades(): Lee actividades previamente almacenadas desde un archivo XML al iniciar la aplicación, las guarda en una lista actividades y devuelve esta lista.

Blueprint_empleado: Define un Blueprint en Flask para manejar las rutas relacionadas con empleados.

cargarEmpleados(): Gestiona la carga de empleados desde un XML recibido por una solicitud POST. Lee los datos XML, crea objetos Empleado, los añade a una lista empleados, y actualiza un archivo XML existente o crea uno nuevo si no existe.

precargar_empleados(): Lee empleados previamente almacenados desde un archivo XML al iniciar la aplicación, los guarda en una lista empleados y devuelve esta lista.

Blueprint_producto: Define un Blueprint en Flask para manejar las rutas relacionadas con productos.

cargarProductos(): Gestiona la carga de productos desde un XML recibido por una solicitud POST. Lee los datos XML, crea objetos Producto, los añade a una lista productos, y actualiza un archivo XML existente o crea uno nuevo si no existe.

ver_productos(): Maneja una solicitud GET para obtener la lista de productos almacenados en memoria. Convierte los objetos Producto en formato JSON y los devuelve como respuesta junto con un mensaje de éxito o error.

precargar_productos(): Lee productos previamente almacenados desde un archivo XML al iniciar la aplicación, los guarda en una lista productos y devuelve esta lista.

Blueprint_user: Define un Blueprint en Flask para manejar las rutas relacionadas con usuarios.

cargarUsuarios(): Gestiona la carga de usuarios desde un XML recibido por una solicitud POST. Lee los datos XML, crea objetos User, los añade a una lista users, y actualiza un archivo XML existente o crea uno nuevo si no existe.

obtenerUsuario(id): Maneja una solicitud GET para obtener un usuario específico basado en su ID. Lee la lista de usuarios almacenada, busca el usuario por ID y devuelve sus detalles en formato JSON si se encuentra, o un mensaje de error si no se encuentra.

Login(): Maneja una solicitud POST para autenticar usuarios. Compara el ID y la contraseña proporcionados con los almacenados, permitiendo el acceso como administrador (role 0) si se utilizan credenciales específicas, o como usuario normal (role 1) si se encuentran coincidencias en la lista de usuarios.

obtenerLogueado(): Maneja una solicitud GET para devolver el usuario actualmente logueado almacenado en la variable global user_logueado.

precargar_usuarios(): Lee usuarios previamente almacenados desde un archivo XML al iniciar la aplicación, los guarda en una lista usuarios y devuelve esta lista.

Cada clase está diseñada para encapsular datos relacionados y funcionalidades específicas, promoviendo así un diseño orientado a objetos que facilita la mantenibilidad y extensibilidad del sistema.

3. Implementación del Inicio de Sesión

El proceso de inicio de sesión se gestiona en la clase **LoginWindow**. Cuando un usuario ingresa sus credenciales, el sistema verifica la combinación de nombre de usuario y contraseña utilizando estructuras de datos como diccionarios o bases de datos. Si las credenciales son correctas, se muestra un mensaje de éxito y se abre la ventana correspondiente (ya sea para administrador o usuario normal).

Este proceso garantiza la seguridad y autenticación adecuada, utilizando técnicas como hashing para almacenar contraseñas de forma segura y proteger la información sensible de los usuarios.

4. Integración

La integración efectiva de listas circulares, ortogonales y visualización gráfica con Graphviz en IPC 2 MARKET no solo mejora la eficiencia operativa, sino que también prepara el sistema para escalar según las demandas del mercado y los usuarios. La combinación de estas técnicas permite una gestión robusta de datos, asegurando que el sistema pueda crecer y adaptarse dinámicamente a medida que se agregan más productos, usuarios y funcionalidades.

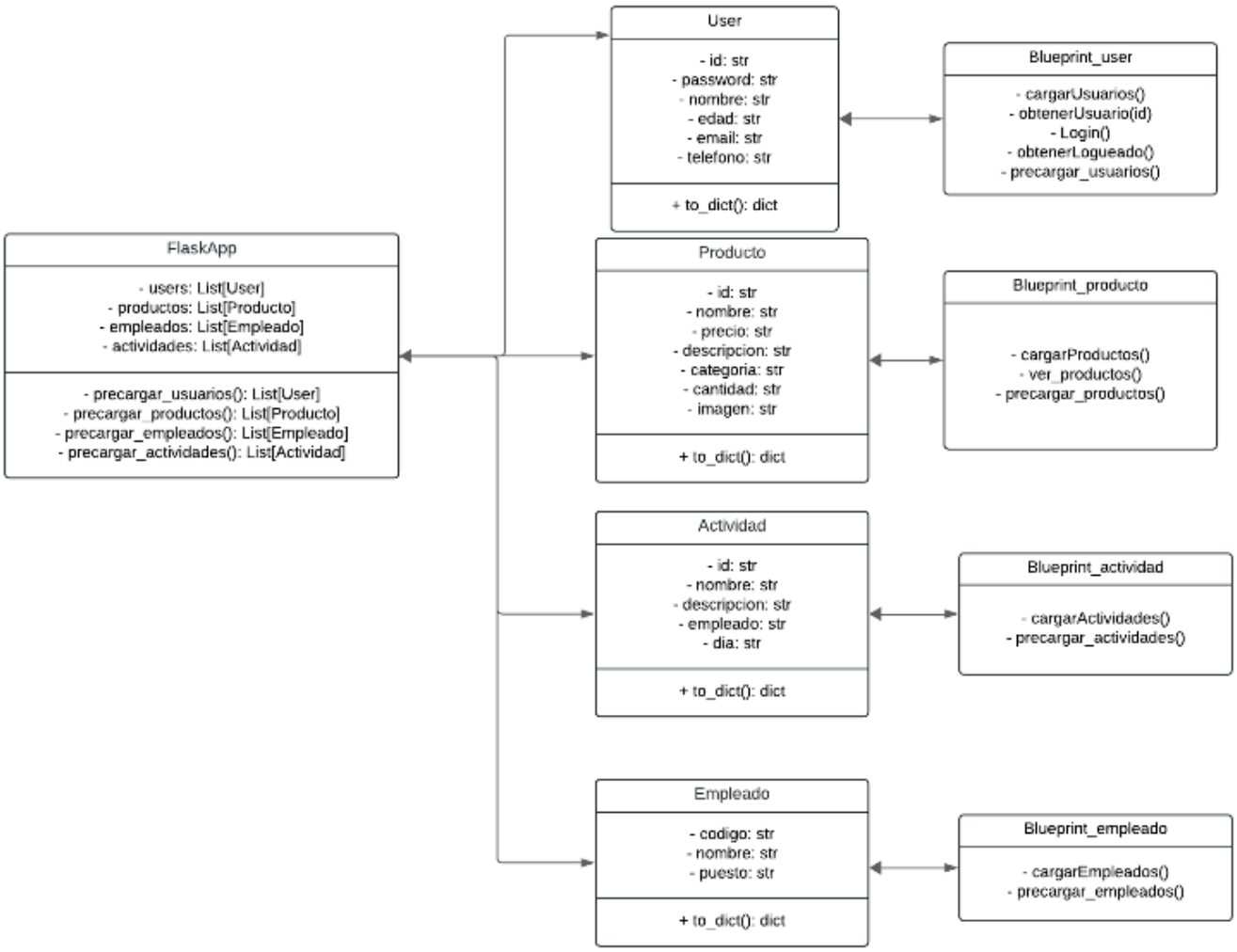
Conclusiones

- **Gestión de datos mediante XML:** El proyecto utiliza archivos XML para almacenar y gestionar datos de usuarios, empleados, productos y actividades. Esto permite una estructura de datos flexible y legible.
- **Uso de Flask y APIs:** Flask se emplea eficazmente para crear una API RESTful que maneja operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre los datos almacenados en XML. Cada entidad (usuarios, empleados, productos, actividades) tiene rutas dedicadas para cargar, ver y autenticar datos.
- **Autenticación y roles**: Implementa un sistema básico de autenticación con diferentes roles (administrador y usuario normal) mediante el uso de contraseñas y credenciales específicas para el administrador.
- **Persistencia y precarga de datos:** Los datos se persisten en archivos XML y se precargan al inicio de la aplicación, lo que asegura que la aplicación pueda manejar datos almacenados previamente y nuevos datos cargados durante la ejecución.
- **Manejo de errores y mensajes:** El proyecto implementa manejo de errores para situaciones como XML vacíos o errores en la carga de datos, proporcionando respuestas JSON claras con mensajes de error adecuados.

Referencias bibliográficas

Python Crash Course
Matthes, E. (2019). Python Crash Course. *No Starch Press.*
Pro Git
Chacon, S., & Straub, B. (2014). Pro Git. *Apress.*

Extensión


User
- id: str
- password: str
- nombre: str
- edad: str
- email: str
- telefono: str
+ to_dict(): dict
Blueprint_user
- cargarUsuarios()
- obtenerUsuario(id)
- Login()
- obtenerLogueado()
- precargar_usuarios()
FlaskApp
- users: List[User]
- productos: List[Producto]
- empleados: List[Empleado]
- actividades: List[Actividad]
- precargar_usuarios(): List[User]
- precargar_productos(): List[Producto]
- precargar_empleados(): List[Empleado]
- precargar_actividades(): List[Actividad]
Producto
- id: str
- nombre: str
- precio: str
- descripcion: str
- categoria: str
- cantidad: str
- imagen: str
+ to_dict(): dict
Blueprint_producto
- cargarProductos()
- ver_productos()
- precargar_productos()
Actividad
- id: str
- nombre: str
- descripcion: str
- empleado: str
- dia: str
+ to_dict(): dict
Blueprint_actividad
- cargarActividades()
- precargar_actividades()
Empleado
- codigo: str
- nombre: str
- puesto: str
+ to_dict(): dict
Blueprint_empleado
- cargarEmpleados()
- precargar_empleados()