Diseño y Desarrollo de servicios web - proyecto GA7-220501096-AA5-EV03

Fase: Ejecución

Actividad No. 5

Evidencia No. 3

Por:
Yamit Alexis Blanco Ávila

Servicio Nacional de Aprendizaje

Tecnología en Análisis
y Desarrollo de
Software

Ficha: 2853234
Instructor: Milton Iván Barbosa
31 de mayo de 2025

Tabla de contenido

Introducción

A través de este documento se presenta el diseño y desarrollo de una API REST implementada en Python utilizando el microframework Flask y la base de datos SQLite. La API corresponde al sistema "Stock.Plus”, una aplicación para gestionar inventarios de productos y usuarios. A lo largo del proyecto se implementaron rutas que permiten realizar operaciones Crud (crear, leer, actualizar y eliminar) sobre la base de datos, utilizando como referencia equivalencia de los lineamientos del componente formativo “Construcción de API”. Se documenta a continuación la arquitectura empleada, el manejo de datos, el control de versiones y los resultados obtenidos.

Justificación

El diseño de una API REST permite separar las responsabilidades entre frontend y backend, facilitando la escalabilidad y el mantenimiento del sistema. Aunque la guía Construcción de API propone el uso de Node.js y MongoDB, el enfoque adoptado con Flask y SQLite se justifica por su rapidez de desarrollo, menor curva de aprendizaje, y facilidad de despliegue. Además, este enfoque es coherente con las características del software, una aplicación de inventario enfocada en la eficiencia operativa para pequeñas y medianas empresas. Esta adaptación demuestra el dominio de conceptos fundamentales sin depender de un stack tecnológico específico. Esta es una solución ideal para un sistema modular, accesible y eficiente para la gestión de inventarios. Flask fue seleccionado por su estructura ligera y rápida de implementar, lo que facilita el desarrollo de servicios web sin una curva de aprendizaje elevada. SQLite, por su parte, se ajusta perfectamente a este tipo de proyectos, ya que es una base de datos embebida, simple y robusta. La API desarrollada cumple con los principios REST, lo que asegura su reutilización e integración futura con aplicaciones móviles o web.

Objetivo general:

Diseñar y desarrollar una API RESTful en Flask con SQLite que permita administrar eficientemente los recursos de inventario en el sistema Stock.Plus.

Objetivos específicos:

1. Crear rutas RESTful que permitan operaciones CRUD sobre usuarios y productos.

2. Documentar los servicios desarrollados con sus respectivas rutas, métodos HTTP y estructura de datos.

3. Versionar el proyecto utilizando Git, asegurando un desarrollo controlado y organizado.

Diseño e implementación de la API REST

1. Estructura del proyecto

El proyecto se encuentra organizado en varios archivos y módulos que cumplen funciones específicas:

- **app.py:** archivo principal donde se configuran las rutas, formularios y lógica de negocio.
- **crear_db.py:** script utilizado para crear la base de datos stock_plus.db y sus tablas (usuarios, productos, etc.).
- **leer_base_datos.py:** permite consultar la base de datos directamente desde consola.
- **static/ y templates/:** contienen los archivos estáticos (CSS, JS) y las vistas HTML para la interfaz visual del sistema.

2. Configuración de la base de datos

La base de datos utilizada es SQLite y se maneja mediante el módulo sqlite3. En crear_db.py, se definen y crean las siguientes tablas:

- **usuarios:** para registrar personas con sus datos personales.
- **productos:** incluye información como nombre, precio, cantidad, fecha de registro.
- **comentarios y contactos:** para gestionar formularios de contacto desde la web.

El archivo leer_base_datos.py permite leer el contenido de las tablas y verificar su correcta estructura.

3. Funcionalidad de la API

Las rutas y lógica de la API se encuentran en app.py. Aunque la aplicación también presenta vistas HTML para la parte visual, se puede acceder directamente a rutas que siguen el modelo REST para gestionar recursos.

Rutas clave (estilo REST)

- **POST /registrar_producto:** crea un nuevo producto.
- **GET /consulta_productos:** muestra todos los productos registrados.
- **GET /editar_producto/<id>:** muestra el formulario para editar un producto.
- **POST /actualizar_producto/<id>:** actualiza los datos de un producto existente.
- **GET /eliminar_producto/<id>:** elimina un producto de la base de datos.
- **POST /contacto:** registra un comentario o mensaje desde el formulario de contacto.

Estas rutas están gestionadas con funciones que abren la base de datos, ejecutan sentencias SQL seguras y luego devuelven una respuesta visual o mensaje.

4. Manejo de datos y seguridad

El proyecto se encuentra organizado en varios archivos y módulos que cumplen funciones específicas:

Se utiliza request.form y request.get_json() (cuando es necesario) para recibir datos del cliente. Además, se protege el acceso a ciertas rutas mediante session y flash, lo que permite controlar sesiones de usuario de forma sencilla.

Los datos son almacenados en la base de datos stock_plus.db, y se realiza validación básica en los formularios para evitar errores de entrada.

5. Documentación de rutas principales

Método	Ruta	Función	Parámetros esperados
POST	**/registrar_producto**	**Crea un nuevo producto**	**Formulario con nombre, precio, cantidad**
GET	**/consulta_productos**	**Lista los productos registrados**	**Nueva ventana lista producto**
POST	**/actualizar_producto/<id>**	**Modifica producto por ID**	**Formulario con campos actualizados**
GET	**/eliminar_producto/<id>**	**Elimina un producto por ID**	**ID por URL**
POST	**/contacto**	**Registra mensaje de contacto**	**Nombre, correo, mensaje**

Control de versiones

Todo el proyecto fue versionado usando GitHub. Los archivos fuente y recursos fueron organizados en un repositorio que puede ser compartido y clonado fácilmente para pruebas o mejoras. El README.md incluye instrucciones de ejecución, estructura del proyecto y una breve explicación de su propósito.

Conclusiones

El desarrollo de esta API REST me permitió aplicar los conceptos aprendidos en el programa formativo, especialmente en el componente de construcción de APIs. Se logró una aplicación funcional, modular y ordenada que permite gestionar productos y usuarios de un inventario mediante operaciones básicas, cumpliendo con el estilo REST. Además de ser útil en escenarios reales, la experiencia permitió reforzar habilidades prácticas como control de versiones, manejo de base de datos y diseño de rutas eficientes. Stock.Plus está listo para ser integrado a futuras interfaces móviles o aplicaciones de escritorio.

Bibliografía

https://zajuna.sena.edu.co/Repositorio/Titulada/institution/SENA/Tecnologia/228118/Contenido/DocArtic/GUI7/Guia_aprendizaje_7.pdf

https://zajuna.sena.edu.co/zajuna/mod/page/view.php?id=654044

https://zajuna.sena.edu.co/Repositorio/Titulada/institution/SENA/Tecnologia/228118/Contenido/OVA/CF33/index.html#/