CARRERA:

Tecnología Superior Universitaria en Desarrollo de Software.

MATERIA:

Programación Orientada a Objetos II

INTEGRANTES:

Emilia Crespo

TEMA:

Proyecto final Python: Conceptos de Aplicaciones en Producción

1. **Contexto del Proyecto Final:**

Propósito del Proyecto:
El objetivo es crear un recorrido virtual interactivo del campus del Instituto Superior Universitario del Azuay para mejorar la orientación vocacional y promover los programas académicos. Los usuarios podrán explorar el campus en un entorno 3D detallado y recibir recomendaciones personalizadas de carreras a través de un chatbot inteligente.

Funcionalidades Principales:
- Entorno 3D: Modelado detallado de las instalaciones utilizando Blender y Three.js para ofrecer una experiencia visual inmersiva.
- Navegación Interactiva: Sistema que permita a los usuarios moverse por el campus de manera intuitiva.
- Chatbot Inteligente: Un chatbot que interactúa con los usuarios, recolecta sus preferencias y les sugiere carreras.
- Gestión de Datos: Sistema para almacenar datos de los usuarios, garantizar la privacidad y generar informes para el Instituto.

Tecnologías Utilizadas:
- Blender: Para el modelado 3D de las instalaciones.
- Three.js: Biblioteca de JavaScript para renderizar gráficos 3D en la web.
- Python: Para la creación del chatbot y el manejo de la inteligencia artificial.
- Bases de datos SQL: Para el almacenamiento de las preferencias de los usuarios y datos de interacción.
- Docker: Para empaquetar y desplegar la aplicación en entornos consistentes.

2. **Propuestas de Mejora**

El documento debe posteriormente proveer una serie de **propuestas de mejora**, utilizando los distintos temas **cubiertos en el curso**, por ejemplo se puede considerar las siguientes secciones (aunque esto es totalmente una decisión del estudiante):

Mejoras en el Código y el Sistema:

2.1 Actualización de versiones de Python

Es fundamental mantener el proyecto actualizado con las últimas versiones estables de Python para beneficiarse de las mejoras en rendimiento, nuevas funcionalidades, y correcciones de seguridad. En este sentido, al actualizar el proyecto a la versión más reciente de Python 3.11, incluye mejoras sustanciales en la gestión de memoria y optimizaciones en la ejecución de código.

La versión de Python 3.11 ha mejorado la velocidad de las funciones, especialmente en procesos intensivos de cálculo, además de ser más eficiente en la gestión de memoria comparado con versiones anteriores. Esto resulta clave en el proyecto, donde el uso de un chatbot y la generación de recomendaciones personalizadas podrían beneficiarse significativamente.

Con esta actualización, los procesos internos del chatbot serán más rápidos, y el uso de memoria durante la navegación por el recorrido virtual será más eficiente, mejorando la experiencia del usuario.

2.2 Refactorización del Código

El proyecto puede beneficiarse de la identificación y refactorización de ciertas partes del código que presentan redundancia o alta complejidad. Se propone dividir funciones largas en componentes más pequeños, reutilizables y fáciles de mantener, lo que contribuirá a la legibilidad y mantenibilidad del sistema.

La refactorización de código no solo facilita su mantenimiento, sino que también reduce la probabilidad de errores y mejora la eficiencia del sistema. Dividir funciones monolíticas permite una depuración más sencilla y hace que el código sea más fácil de extender. Además, al aplicar principios de diseño como "Single Responsibility Principle" (SRP), cada función tendrá un propósito claro.

Ejemplo de Mejora:

Actualmente hay una función que gestiona tanto la entrada del usuario como el proceso de recomendación de carreras. Esta función puede separarse en dos:

```
def recolectar_datos_usuario():
    # Código para recopilar las preferencias del usuario
    pass

def recomendar_carrera(preferencias):
    # Código para procesar y recomendar una carrera
    pass
```

De este modo, se facilita la reutilización de cada función de manera independiente, promoviendo un código más limpio y eficiente.

2.3 Mejora de la Legibilidad y Mantenibilidad

El uso de tipado estático y herramientas de análisis de código estático como mypy, contribuirá a mejorar la robustez y confiabilidad del código ya que el uso de este ayuda a detectar errores antes de la ejecución del código, lo que es crucial en un proyecto con múltiples componentes interactuando entre sí. Además, aumenta la comprensión del código por parte de otros desarrolladores que podrían trabajar en el proyecto a futuro.

- Incorporar tipado explícito a las funciones para hacer el código más claro.
- El uso de mypy permitirá verificar que el código cumpla con los tipos esperados.

2.4 Uso de Herramientas de Calidad de Código

Para garantizar la calidad y uniformidad del código, se propone el uso de herramientas como pylint, flake8 y black. Estas herramientas ayudan a identificar errores y a garantizar que el código sigue estándares de formato, proporcionan validaciones automáticas que evitan errores comunes y aseguran que el código sea consistente en términos de estilo.
Pylint y **flake8** identificarán problemas relacionados con la estructura del código, mientras que **black** garantizará que todo el código esté correctamente formateado de manera automática, mejorando así la legibilidad.

Al ejecutar pylint sobre el código del proyecto puede resaltar mejoras sugeridas en el código, como nombres de variables más descriptivos o la reducción de la complejidad ciclomática, lo que mejora la mantenibilidad del proyecto.

Planes de Pruebas Completos:

2.5 Implementación de Pruebas Unitarias e Integración

Actualmente, el proyecto carece de una cobertura completa de pruebas. Una mejora sería la implementación de un conjunto exhaustivo de pruebas unitarias y pruebas de integración para garantizar la estabilidad y confiabilidad del sistema a medida que crece.

Las pruebas unitarias permiten verificar que cada componente del sistema funcione correctamente de manera aislada, mientras que las pruebas de integración validan la interacción correcta entre múltiples componentes. Esto es fundamental en un sistema complejo que incluye tanto un recorrido virtual como un chatbot interactivo. Herramientas como pytest pueden ser utilizadas para automatizar estas pruebas.

Mejoras en el Pipeline de CI/CD:

2.6 Integración Continua y Despliegue Continuo (CI/CD)

Actualmente, el proyecto no cuenta con un pipeline automatizado de pruebas y despliegue por lo que GitHub Actions para automatizar las pruebas y el despliegue en un entorno de staging

sería lo ideal. Esto garantizará que cada nueva versión sea probada y desplegada de manera confiable.

Automatizar el proceso de pruebas y despliegue mejora significativamente la eficiencia del desarrollo y reduce el riesgo de introducir errores en producción. Configurar un pipeline CI/CD con GitHub Actions asegurará que cada cambio sea probado antes de integrarse en la rama principal.

Ejemplo de Mejora:

```yaml
name: CI/CD Pipeline

on:
  push:
    branches:
      - main

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: '3.11'
      - name: Install dependencies
        run: pip install -r requirements.txt
      - name: Run tests
        run: pytest
```

Acceso a Datos:

El proyecto actual utiliza una base de datos relacional PostgreSQL para almacenar la información relacionada con las carreras universitarias, perfiles de usuarios. Además en el chatbot vocacional, una mejora es que se pueda acceder a fuentes de datos externas a través de APIs para obtener sugerencias de carreras o preferencias del usuario.

2.7 Mejora esperada con SQLAlchemy : Se reduce el riesgo de errores en las consultas SQL y mejora la mantenibilidad del código.

El uso de SQLAlchemy para la gestión de bases de datos relacionales: Actualmente, las consultas SQL manuales pueden resultar propensas a errores y difíciles de mantener. SQLAlchemy facilita la interacción con bases de datos relacionales mediante un ORM

(Object-Relational Mapping) que abstrae las operaciones SQL. Esto ayudará a que el código sea más legible y menos propenso a fallos como la inyección SQL y también permite trabajar con transacciones de forma más controlada.

2.8 Validación de datos con Pydantic: El chatbot vocacional maneja entradas del usuario que deben ser validadas antes de procesarse. Pydantic es una biblioteca que garantiza que los datos de entrada cumplan con los requisitos esperados antes de ser procesados, lo que minimiza errores y garantiza la integridad de los datos.

- **Mejora esperada**: Validación automática de datos, lo que previene errores por entrada de datos incorrectos y asegura que la aplicación funcione de manera consistente.

2.9 Análisis de datos con Pandas: Para hacer análisis de datos, como el comportamiento de las preferencias de carreras y las estadísticas de las mismas elecciones, Pandas facilita la manipulación de grandes volúmenes de datos, así podemos procesar datos tabulares y proporcionar herramientas para análisis estadístico, agrupamiento y visualización.

- **Mejora esperada:** Mayor facilidad para realizar análisis complejos de datos de usuarios o rendimiento de la plataforma.

Empaquetado y Despliegue:

2.10 Multi-stage Builds: Aunque el proyecto ya utiliza Docker para el empaquetado y despliegue, implementar **Docker Multi-stage Builds** sería una mejora significativa debido a que este enfoque permite optimizar la construcción de las imágenes al dividir el proceso en etapas, lo que ofrece múltiples ventajas tanto en desarrollo como en producción.

- **Mejora esperada**: Esto reduce considerablemente el tamaño de la imagen final y asegura que solo los archivos y dependencias esenciales estén presentes en producción.

2.11Gestión de Dependencias con Poetry: Si bien Docker ya se encarga de empaquetar el proyecto en en un contenedor, Poetry puede mejorar la gestión de dependencias y el empaquetado a nivel de código.

Poetry gestiona versiones y dependencias de forma más precisa que un simple requirements.txt. Además, genera un archivo pyproject.toml para la configuración del proyecto y un poetry.lock que asegura que las versiones exactas de las dependencias se instalen, haciendo el entorno más reproducible.

3. Tecnologías y Prácticas No Utilizadas

A medida que desarrollé mi proyecto, exploré diversas tecnologías y enfoques, y tras un cuidadoso análisis, tomé la decisión de no implementar algunas de ellas. Cada elección fue analizada, basada en el deseo de mantener el proyecto simple y enfocado. Aquí comparto

algunas de las tecnologías que no utilicé, junto con las razones que me llevaron a esas decisiones.

- Framework Django: Desde el principio, consideré utilizar este framework debido a su popularidad y robustez en el desarrollo de aplicaciones web. Sin embargo, a medida que profundizaba en los requisitos de mi proyecto, me di cuenta de que la complejidad de Django no era lo que necesitaba. La aplicación no requería características avanzadas, como un sistema de autenticación o una gestión de usuarios complicada. Opté por **Flask**, un microframework más ligero, que me brindó la flexibilidad necesaria para construir una API REST y un recorrido virtual. Esta elección no solo facilitó la personalización de los componentes que realmente importaban, sino que también me permitió centrarme en los aspectos fundamentales del proyecto sin perder tiempo lidiando con una estructura más complicada.

- GraphQL: También consideré **GraphQL**, especialmente por su capacidad para manejar datos de forma más eficiente y por su popularidad en la comunidad de desarrolladores. Sin embargo, tras analizar el flujo de datos en mi proyecto, me di cuenta de que no era necesario complicar las cosas. El uso de RESTful APIs se ajustaba perfectamente a mis necesidades, ya que el intercambio de datos entre el chatbot y el backend era bastante sencillo. Al elegir REST, evité la complejidad adicional de definir esquemas detallados y ajustes en el servidor, lo que hizo que el desarrollo fuera más ágil y menos propenso a errores.

- MongoDB: Aunque también consideré la posibilidad de utilizar MongoDB, que es conocido por su capacidad para manejar datos no estructurados de manera flexible, llegué a la conclusión de que mi base de datos se componía principalmente de información estructurada. Tenía que gestionar datos específicos sobre carreras universitarias y perfiles de usuarios, lo que se ajustaba mejor a un sistema de base de datos relacional. Por esta razón, opté por **PostgreSQL**. Esta base de datos no solo es excelente para realizar consultas SQL complejas, sino que también se integra de manera efectiva con SQLAlchemy para mejoras del proyecto más adelante.

- Unity: Al considerar las herramientas para el desarrollo de un recorrido virtual, decidí no utilizarlo. Aunque es una plataforma poderosa para crear experiencias interactivas y juegos, su complejidad y los altos requisitos de hardware que exige podrían limitar el acceso de los usuarios. En su lugar, elegí Three.js como la tecnología principal para el recorrido virtual. Esta elección no solo me permitió crear una experiencia visual atractiva e interactiva, sino que también garantizó que el recorrido fuera accesible desde cualquier navegador, lo que es esencial para que un mayor número de usuarios pueda disfrutar de la experiencia sin importar el dispositivo que estén usando.

- Serverless como AWS Lambda: En un principio, consideré implementar una arquitectura serverless, que es muy popular en el desarrollo moderno por su escalabilidad y eficiencia en costos. Sin embargo, decidí optar por Google Cloud como mi plataforma de despliegue. Utilicé instancias de Compute Engine y almacenamiento en Cloud Storage, lo que me brindó un mayor

control sobre la infraestructura y los recursos. Esta elección fue crucial, ya que el proyecto maneja interacciones en tiempo real y necesitaba un entorno que ofreciera un rendimiento consistente y una gestión efectiva. Google Cloud proporciona herramientas que facilitan la monitorización del rendimiento, lo que me permite ajustar y optimizar la aplicación según sea necesario.

4. Conclusión

En resumen, las propuestas de mejora para el recorrido virtual del campus del Instituto Superior Universitario del Azuay, están diseñadas para que cada una de ellas fortalezca el proyecto de manera significativa. Desde mejorar la calidad del código hasta garantizar la robustez del sistema, cada ajuste está diseñado para hacer que la experiencia del usuario sea más fluida y efectiva.

Por ejemplo, actualizar las versiones de Python y refactorizar partes del código no solo ayudará a que sea más legible, sino que también facilitará futuras colaboraciones. Esto es crucial en un entorno donde las necesidades pueden cambiar rápidamente. Además, descomponer funciones complejas en componentes más pequeños hará que sea mucho más sencillo identificar errores y realizar mejoras.

La importancia de las pruebas no puede ser subestimada. Implementar pruebas unitarias y de integración asegurará que cada parte del sistema funcione correctamente antes de llegar al usuario final. Esto no solo aumentará la confianza en el producto, sino que también reducirá la frustración del usuario al disminuir la posibilidad de fallos inesperados.

Optimizar el pipeline de CI/CD mediante herramientas como GitHub Actions será otro gran paso. Automatizar el proceso de pruebas y despliegue permitirá que los desarrolladores se concentren más en crear nuevas funcionalidades en lugar de preocuparse por el mantenimiento del ciclo de vida del software. Esto es algo que realmente puede hacer la diferencia en un entorno de desarrollo ágil.

También es importante considerar cómo manejamos los datos. Utilizar bibliotecas como SQLAlchemy y Pydantic mejorará la manera en que interactuamos con la base de datos y garantizará que la información del usuario esté bien gestionada y protegida. Esto es fundamental para ofrecer una experiencia personalizada, que es lo que buscamos.

Finalmente, al emplear herramientas como Docker y Poetry para empaquetar y desplegar, se logrará que el proceso sea mucho más consistente y manejable. Esto no solo facilitará la vida del equipo de desarrollo, sino que también asegurará que el software funcione sin problemas en diferentes entornos.

Y reflexionando sobre todo lo aprendido en el curso, se hace evidente cómo estos principios han influido en la manera en que he abordado el desarrollo de este proyecto. La importancia de mantener un código limpio y de alta calidad ha resonado en mí, al igual que la necesidad de

integrar pruebas exhaustivas. Además, los métodos ágiles me han enseñado a ser más adaptable y a valorar la colaboración en equipo.

En resumen, estas mejoras no solo buscan optimizar el recorrido virtual, sino también asegurarse de que el proyecto sea sostenible y escalable en el futuro. Aplicar lo aprendido me ha permitido tener un enfoque más integral y consciente sobre cómo desarrollar software que realmente haga la diferencia para los usuarios.