Parcial - Ingenieria Web

Proyecto Auto CRUD - Scaffolding con Flask y Docker

Estudiante	Escuela	Asignatura
Melany YasminLazo Arana mlazoa@ulasalle.edu.pe	Carrera Profesional de Ingeniería de Software	Ingenieria Web Semestre 2025-I ING-WEB

Estudiante	Escuela	Asignatura
Alvaro Andre Machaca Melendez amachacam@ulasalle.edu.pe	Carrera Profesional de Ingeniería de Software	Ingenieria Web Semestre 2025-I ING-WEB

Estudiante	Escuela	Asignatura
Carla Fernanda Ropa Calizaya cropac@ulasalle.edu.pe	Carrera Profesional de Ingeniería de Software	Ingenieria Web Semestre 2025-I ING-WEB

Índice

1. Introducción

Este documento describe el proyecto de scaffolding desarrollado con Flask y Docker para el parcial de Ingeniería Web. El sistema implementa un CRUD automático para la gestión académica, incluyendo modelos para cursos, estudiantes, profesores, inscripciones y cargas de trabajo.

La Ingeniería Web es una disciplina de la ingeniería de software centrada en el diseño, desarrollo, implementación y mantenimiento de aplicaciones web de calidad. Esta rama combina principios de la ingeniería tradicional con tecnologías modernas de la web para construir soluciones interactivas, eficientes y escalables. En este contexto, Flask, un microframework ligero escrito en Python, se presenta como una herramienta poderosa y flexible para el desarrollo web. Gracias a su simplicidad, modularidad y capacidad de expansión, Flask permite a los desarrolladores construir desde aplicaciones pequeñas hasta sistemas web complejos de forma rápida y ordenada.

1.1. Tecnologías y herramientas utilizadas

- Python: Lenguaje de programación principal.
- Flask: Microframework web para desarrollo rápido.
- SQLAlchemy: ORM para manejo de bases de datos.
- Docker: Para contenerización de la aplicación.
- PostgreSQL: Base de datos relacional (en modo Docker).
- SQLite: Base de datos alternativa (en modo local).
- Flask-Admin: Para generar el panel de administración.
- Alembic: Para realizar migraciones de base de datos.

2. Marco Teórico

2.1. Flask

Flask es un microframework web escrito en Python que permite desarrollar aplicaciones web de forma sencilla, rápida y flexible. Diseñado para ser minimalista, proporciona solo las herramientas esenciales para comenzar, permitiendo al desarrollador agregar únicamente los componentes que necesita. Esta filosofía lo convierte en un framework ideal para proyectos pequeños o medianos, aunque también puede escalarse para aplicaciones más complejas gracias a su arquitectura modular. Su curva de aprendizaje es suave, lo que lo hace perfecto tanto para principiantes como para desarrolladores experimentados que buscan un control total sobre la estructura de su aplicación.

2.2. Docker

Docker es una plataforma de código abierto que automatiza el despliegue, la escala y la gestión de aplicaciones mediante contenedores. Los contenedores son unidades estándar de software que empaquetan el código y todas sus dependencias para que la aplicación se ejecute de manera confiable de un entorno computacional a otro. Docker permite crear entornos aislados y reproducibles, lo que facilita el desarrollo, las pruebas y el despliegue de aplicaciones.

2.3. CRUD

CRUD es un acrónimo para las operaciones básicas que se pueden realizar con datos persistentes:

- Create (Crear): Insertar nuevos registros en la base de datos.
- Read (Leer): Consultar registros existentes.
- Update (Actualizar): Modificar registros existentes.
- Delete (Eliminar): Eliminar registros de la base de datos.

Un sistema CRUD automático permite generar estas operaciones básicas sin necesidad de escribir código repetitivo para cada modelo de datos.

3. Estructura del Proyecto

La estructura del proyecto se organiza de la siguiente manera:

```
AUTO_CRUD_PROJECT/
|----_pycache_/
     |----app.cpython-311.pyc
     |----config.cpython-311.pyc
     +----seed.cpython-311.pyc
|----migrations/
     |----_pycache_/
     |----versions/
     |----alembic.ini
     |----env.py
     |----README
     +----script.py.mako
|----models/
     |----_pycache_/
     |----course.py
     |----generation_db.py
     |----inscription.py
     |----student.py
     |----teacher.py
     +----workload.py
|----.gitignore
|----app.db
|----app.py
|----config.py
|----docker-compose.yml
|----Dockerfile
|----entrypoint.sh
|----README.md
|----requirements_docker.txt
|----requirements_local.txt
+----seed.py
```

4. Descripción de Componentes Principales

4.1. Modelos de Datos

El proyecto incluye los siguientes modelos de datos para la gestión académica:

- **course.py**: Define el modelo para los cursos académicos.
- **student.py**: Define el modelo para los estudiantes.
- **teacher.py**: Define el modelo para los profesores.
- **inscription.py**: Define el modelo para las inscripciones de estudiantes a cursos.
- **workload.py**: Define el modelo para las cargas de trabajo asignadas a profesores.
- **generation_db.py**: Contiene funcionalidades para la generación automática de la base de datos.

4.2. Configuración

Los archivos de configuración clave incluyen:

- **config.py**: Contiene la configuración de la base de datos y otras variables de entorno.
- **app.py**: Archivo principal que inicializa la aplicación Flask.
- **seed.py**: Script para poblar la base de datos con datos iniciales.

4.3. Docker

Los archivos relacionados con Docker son:

- **docker-compose.yml**: Define los servicios necesarios para ejecutar la aplicación.
- **Dockerfile**: Contiene las instrucciones para construir la imagen Docker.
- **entrypoint.sh**: Script que se ejecuta cuando se inicia el contenedor.

4.4. Migraciones

El directorio `migrations/` contiene los archivos necesarios para las migraciones de la base de datos usando Alembic y Flask-Migrate.

5. Instalación y Ejecución

5.1. Clonar el Repositorio

Para comenzar con el proyecto, primero clone el repositorio:

```
git clone https://github.com/tu-usuario/tu-repo.git
cd tu-repo
```

Reemplace `https://github.com/tu-usuario/tu-repo.git` con la URL real del repositorio en GitHub.

5.2. Ejecución con Docker

Para ejecutar el proyecto utilizando Docker, siga estos pasos:

5.2.1. Construir la Imagen

Para construir la imagen Docker sin utilizar la caché (para evitar restos de compilaciones anteriores):

```
docker-compose build --no-cache web
```

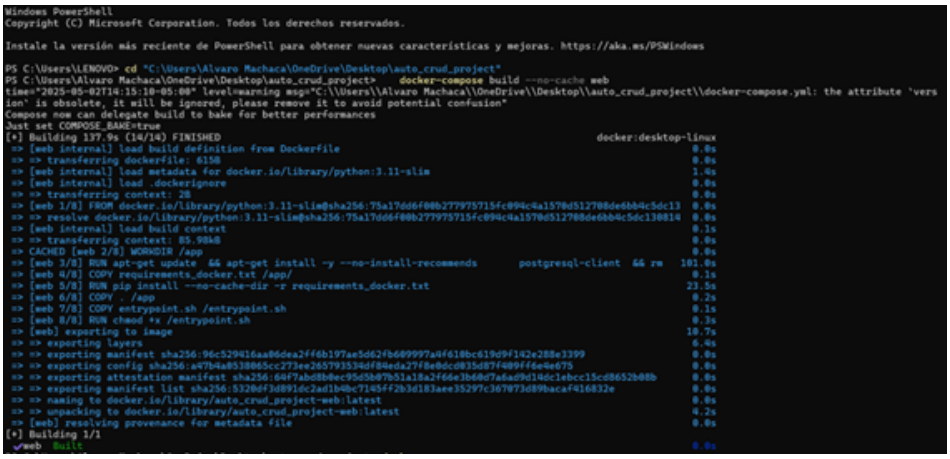

Figura 1: Proceso de construcción de la imagen Docker (docker-compose build)

5.2.2. Levantar los Contenedores

Para iniciar los servicios definidos en el archivo docker-compose.yml:

```
docker-compose up
```

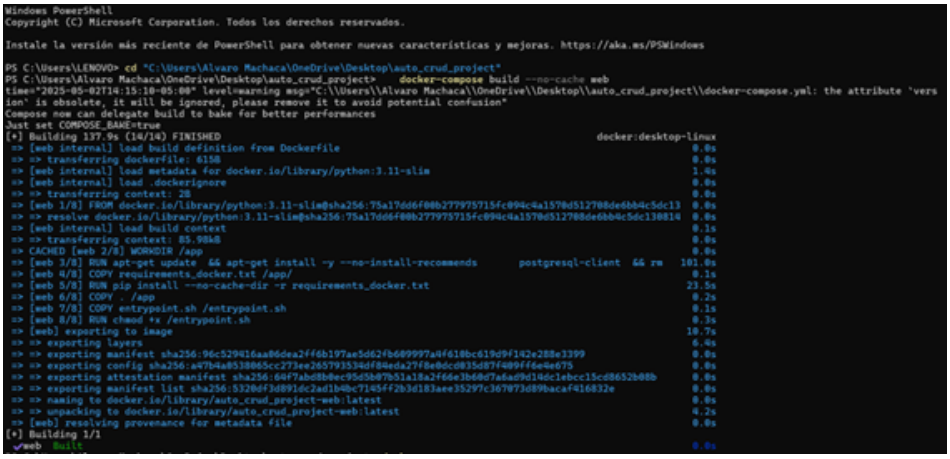

Figura 2: Contenedores Docker en ejecución (docker-compose up)

Una vez que los contenedores estén funcionando, el panel de administración estará disponible en `http://localhost:5000/admin`.

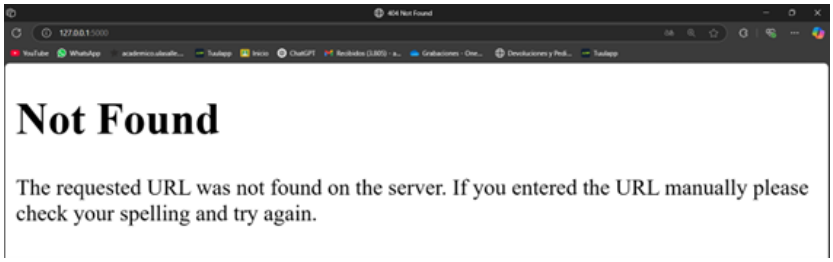

Figura 3: Error al intentar acceder a URL incorrecta

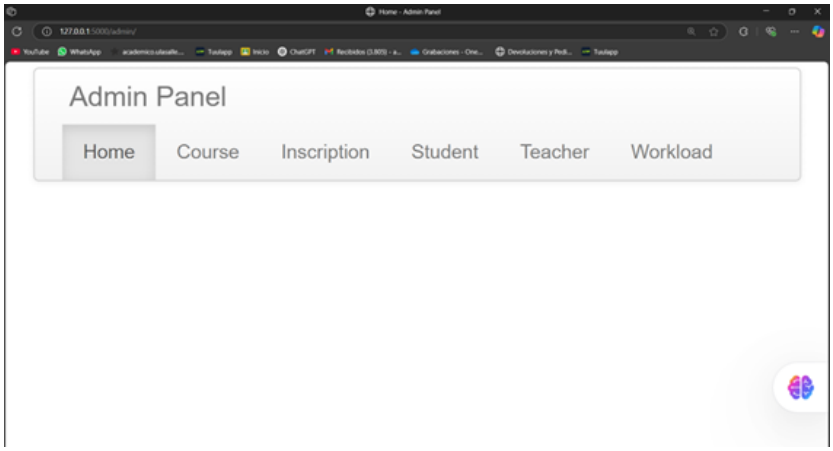

Figura 4: Panel de administración del sistema funcionando correctamente

Como se puede observar en las Figuras 3 y 4, cuando intentamos acceder a una URL incorrecta obtenemos un error, pero al acceder a la ruta correcta del panel de administración (/admin), podemos ver la interfaz funcionando correctamente con todas las opciones disponibles para gestionar los modelos del sistema.

Nota: Los servicios `db` (PostgreSQL) y `web` (aplicación Flask) se inician simultáneamente como se muestra en la Figura 2, donde podemos ver que PostgreSQL se inicializa y la aplicación Flask comienza a escuchar en el puerto 5000.

5.3. Ejecución en Local (SQLite)

Para ejecutar el proyecto localmente utilizando SQLite como base de datos:

5.3.1. Configurar config.py

Modifique el archivo `config.py` para utilizar SQLite. Descomente la sección correspondiente:

```
import os
class Config:
    SQLALCHEMY_DATABASE_URI = os.getenv(
        'DATABASE_URL',
        f"sqlite:///{os.path.abspath(os.path.dirname(__file__))}/app.db"
    )
    SQLALCHEMY_TRACK_MODIFICATIONS = False
```

5.3.2. Instalar Dependencias

Cree un entorno virtual e instale las dependencias necesarias:

```
python -m venv env
source env/bin/activate   # macOS/Linux
env\Scripts\activate      # Windows PowerShell
pip install -r requirements_local.txt
```

5.3.3. Inicializar y Aplicar Migraciones

Inicialice la base de datos y aplique las migraciones:

```
flask db init
flask db migrate -m "Migrando modelos"
flask db upgrade
```

5.3.4. Ejecutar la Aplicación

Inicie la aplicación Flask:

```
python app.py
```

Luego, abra `http://127.0.0.1:5000/admin` en su navegador para acceder al panel de administración.

6. Descripción Detallada de los Modelos

6.1. Modelo Estudiante (student.py)

Este modelo representa a los estudiantes registrados en el sistema. Contiene información como:

- ID del estudiante
- Nombre y apellido
- Correo electrónico
- Fecha de nacimiento
- Otra información relevante

6.2. Modelo Profesor (teacher.py)

Este modelo representa a los profesores en el sistema. Contiene información como:

- ID del profesor
- Nombre y apellido
- Especialidad o departamento
- Información de contacto

6.3. Modelo Curso (course.py)

Define los cursos disponibles en el sistema académico:

- ID del curso
- Nombre del curso
- Descripción
- Créditos o unidades
- Horario o periodo

6.4. Modelo Inscripción (inscription.py)

Representa la relación entre estudiantes y cursos:

- ID de inscripción
- ID del estudiante (relación)
- ID del curso (relación)
- Fecha de inscripción
- Estado (activo, cancelado, etc.)

6.5. Modelo Carga de Trabajo (workload.py)

Define la asignación de cursos a profesores:

- ID de carga de trabajo
- ID del profesor (relación)
- ID del curso (relación)
- Periodo académico
- Horas asignadas

7. Panel de Administración

El sistema incluye un panel de administración accesible desde `http://localhost:5000/admin` o `http://127.0.0.1:5000/admin` dependiendo del método de ejecución. Este panel proporciona una interfaz para gestionar:

- Estudiantes
- Profesores
- Cursos
- Inscripciones
- Cargas de trabajo

8. Requisitos del Sistema

Para ejecutar este proyecto necesita:

8.1. Para Ejecución con Docker

- Docker
- Docker Compose

8.2. Para Ejecución Local

- Python 3.11+
- pip
- Virtualenv (recomendado)
- Dependencias listadas en requirements_local.txt

9. Conclusiones

Este proyecto proporciona un sistema base de scaffolding para desarrollar aplicaciones web con Flask y Docker, enfocado en la gestión académica. La estructura modular permite extender fácilmente las funcionalidades y adaptar el sistema a necesidades específicas.

La combinación de Flask y Docker ofrece varias ventajas:

- **Portabilidad**: El entorno de desarrollo es consistente en diferentes plataformas.
- **Aislamiento**: Las dependencias del proyecto están contenidas y no afectan al sistema host.
- **Reproducibilidad**: Cualquier desarrollador puede ejecutar el proyecto con resultados idénticos.
- **Escalabilidad**: La aplicación puede desplegarse fácilmente en entornos de producción.

El sistema CRUD automático permite ahorrar tiempo en la implementación de operaciones básicas, permitiendo a los desarrolladores enfocarse en la lógica de negocio específica.

10. Referencias

- Flask Documentation. (2023). `https://flask.palletsprojects.com/`
- Docker Documentation. (2023). `https://docs.docker.com/`
- SQLAlchemy Documentation. (2023). `https://docs.sqlalchemy.org/`
- Flask-Admin Documentation. (2023). `https://flask-admin.readthedocs.io/`
- Flask-Migrate Documentation. (2023). `https://flask-migrate.readthedocs.io/`