Universidad Nacional de Colombia - sede Bogotá
Facultad de Ingeniería
Departamento de Sistemas e Industrial
Curso: Ingeniería de Software 1 (2016701)
ROSALES OBANDO PABLO-QUIÑONES MEJIA OMAR ANDRES-QUIÑONES SEGURA JAIME JAVIER

Tutorial: Construcción de una aplicación Hola Mundo con Python + Django + PostgreSQL

Introducción

Vamos a construir una pequeña aplicación web que mostrará un mensaje almacenado en una base de datos. Usaremos Python, el framework Django, y una base de datos PostgreSQL.

El objetivo es recorrer todas las capas de una aplicación moderna:

- Capa de dominio (modelo)
- Repositorio (acceso a datos)
- Servicio (lógica de negocio)
- Controlador (vista Django)
- Interfaz de usuario (HTML + JS)

1. Preparación del entorno

Requisitos:

- Python 3.10+
- Docker y Docker Compose
- Git (opcional)

```
python -m venv venv
source venv/bin/activate
pip install django psycopg2-binary dj-database-url python-dotenv
```

Para la base de datos, usaremos Docker. Creamos un archivo docker-compose.yml:

docker-compose.yml

```yaml
version: '3.8'

services:
  db:
    image: postgres:15-alpine
    container_name: hello_django_db
    environment:
      POSTGRES_USER: django_user
      POSTGRES_PASSWORD: django_pass
      POSTGRES_DB: hello_django_db
    ports:
      - "5432:5432"
    volumes:
      - postgres-data:/var/lib/postgresql/data
    restart: unless-stopped
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U django_user -d hello_django_db"]
      interval: 10s
      timeout: 5s
      retries: 5

volumes:
  postgres-data:
```

Y la levantamos con:

docker-compose up -d

Esto nos deja una base de datos lista para conectarse desde Django.

2. Creando el proyecto Django

Creamos el proyecto y una aplicación:

```
django-admin startproject holamundo_project
cd holamundo_project
python manage.py startapp helloapp
```

Ahora registramos nuestra app en holamundo_project/settings.py:

```python
INSTALLED_APPS: list[str] = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'helloapp.apps.HelloappConfig',
]
```

Y configuramos la base de datos usando variables de entorno:

```python
import dj_database_url
DATABASES = {
    'default': dj_database_url.config(conn_max_age=600, ssl_require=False)
}
```

3. el modelo

Vamos a definir una clase que represente el mensaje que queremos mostrar.

models.py > ...

```python
from django.db import models

class Message(models.Model):
    content: CharField[str] = models.CharField(max_length=255)

    def __str__(self) -> str:
        return self.content
```

Este modelo crea una tabla helloapp_message en la base de datos, con una columna content.
Ejecutamos las migraciones:

python manage.py makemigrations
python manage.py migrate

4. Sembrando datos iniciales (Singleton)

Queremos asegurarnos de que exista al menos un registro con el mensaje “Hola Mundo”.
Para no duplicarlo cada vez que inicia el servidor, aplicaremos el patrón Singleton:
solo se ejecutará una vez, sin importar cuántas veces Django arranque.

Creamos helloapp/initial_seeder.py:

```python
import threading
from django.db import IntegrityError

class InitialDataSeeder:
    _instance = None
    _lock = threading.Lock()
    _seeded = False

    def __new__(cls) -> Self:
        if cls._instance is None:
            with cls._lock:
                if cls._instance is None:
                    cls._instance: Self = super().__new__(cls)
        return cls._instance

    def seed_initial_data(self) -> Any:
        if not self._seeded:
            with self._lock:
                if not self._seeded:
                    from .models import Message
                    try:
                        Message.objects.get(pk=1)
                    except Message.DoesNotExist:
                        Message.objects.create(pk=1, content="Hola Mundo desde Django + BD! 🎮")
                    self._seeded = True

initial_seeder = InitialDataSeeder()
```

Y lo activamos al inicio de la app en helloapp/apps.py:

```python
from django.apps import AppConfig

class HelloappConfig(AppConfig):
    default_auto_field: str = 'django.db.models.BigAutoField'
    name = 'helloapp'

    def ready(self) -> None:
        from .initial_seeder import initial_seeder
        initial_seeder.seed_initial_data()
```

Cuando Django carga la app, llama a ready().
Ese método crea (o reutiliza) una única instancia de InitialDataSeeder,
que inserta un registro en la base de datos si no existe.

5. Acceso a datos

Ahora creamos una función encargada **únicamente** de hablar con la base de datos.

repositories.py > ...

```python
from .models import Message

def get_message_by_id(message_id: int) -> Message:
    return Message.objects.get(pk=message_id)

```

Si el registro con id=1 existe, lo devuelve.
Si no, Django lanza una excepción Message.DoesNotExist.

6. Lógica de negocio (Servicio)

La capa de servicio se encarga de decidir qué hacer si el mensaje no existe, o si necesitamos procesar los datos.

helloapp/services.py:

```python
from . import repositories
from .models import Message

def get_hello_message() -> Message:
    try:
        message: Message = repositories.get_message_by_id(message_id=1)
        return message
    except Message.DoesNotExist:
        return Message(content="Error: Mensaje ID 1 no encontrado.")

```

Aquí se abstrae la “regla de negocio”:
“el mensaje principal es el registro con ID 1”.

7. Controlador (Vistas Django)

En helloapp/views.py escribimos dos funciones:

- una que entrega el HTML inicial
- otra que sirve el mensaje como JSON para el frontend

views.py > ...

```python
from django.shortcuts import render
from django.http import JsonResponse, HttpRequest, HttpResponse
from . import services

def index_view(request: HttpRequest) -> HttpResponse:
    return render(request, 'helloapp/index.html')

def hello_api_view(request: HttpRequest) -> JsonResponse:
    message_obj: Message = services.get_hello_message()
    data: dict[str, Any] = {
        'id': message_obj.id,
        'content': message_obj.content
    }
    return JsonResponse(data)

```

La primera muestra la interfaz;
la segunda expone el endpoint /hello-data/ que usará nuestro botón.

8. Interfaz de usuario (Frontend)

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Hola Mundo Django</title>
</head>
<body>
    <h1>Aplicación Hola Mundo Django</h1>
    <button id="loadButton">Mostrar mensaje</button>
    <p id="messageDisplay"></p>

    <script>
        document.getElementById('loadButton').onclick = async () => {
            const res = await fetch('/hello-data/');
            const data = await res.json();
            document.getElementById('messageDisplay').innerText = data.content;
        };
    </script>
</body>
</html>
```

Cada vez que haces clic en el botón:

1. El script hace un fetch('/hello-data/').
2. Django ejecuta hello_api_view.
3. hello_api_view llama al servicio → repositorio → base de datos.
4. Devuelve el mensaje en JSON.
5. El navegador lo muestra en pantalla.

9. Probando la aplicación

Levantamos el servidor:

```
python manage.py runserver
```