Universidad Camilo José Cela

Facultad de Tecnología y Ciencia
Grado en Ingeniería Informática

Entregable 3: Migración a MVC

Desarrollo de front-end

Código: 76042

Autor: César Andrés Sánchez
Versión: v2.0.0
Año: 2025-2026

Contenidos

Capítulo 3

Práctica 3. Migración a MVC

3.1 Introducción

La evolución del desarrollo software requiere la adopción de arquitecturas que promuevan la mantenibilidad, escalabilidad y separación de responsabilidades. En las prácticas anteriores se implementó un sistema de login para e-commerce (Práctica 1) y se añadió dinamismo mediante JavaScript (Práctica 2). Sin embargo, estas soluciones presentaban un enfoque monolítico donde la lógica de presentación, negocio y control se encontraban estrechamente acopladas.

Esta tercera práctica tiene como objetivo fundamental realizar una migración arquitectural hacia el patrón Modelo-Vista-Controlador (**MVC**), reestructurando completamente el código existente para lograr una separación clara de responsabilidades que facilite el mantenimiento, la testing y la evolución futura del sistema.

3.2 Objetivos de aprendizaje

- Comprender los principios fundamentales del patrón **MVC** y sus beneficios en el desarrollo software.
- Aplicar la separación de responsabilidades entre Modelo (lógica de dominio y datos), Vista (presentación e interfaz de usuario) y Controlador (orquestación y gestión de flujo).
- Implementar un sistema donde la lógica de negocio sea independiente de la interfaz de usuario.
- Desarrollar competencias en refactorización de código existente hacia arquitecturas más sostenibles.
- Garantizar la trazabilidad y reproducibilidad del proceso de migración.

3.2.1 Entendiendo el framework

3.3 Resumen

Aplicación **Flask** con patrón **Modelo–Vista–Controlador (MVC)**, autenticación con **Flask-Login**, persistencia en **SQLite** mediante **SQLAlchemy**, plantillas **Jinja2**, y de-

spliegue con **Docker/docker-compose**. Incluye CLI para inicialización y siembra y un catálogo de productos con imágenes estáticas.

3.4 Estructura de directorios

```
flask_mvc_login/
|-- app/
|   |-- __init__.py           # fábrica de la app, config, CLI, bootstrap DB
|   |-- models.py              # Modelos SQLAlchemy: User, Product
|   |-- controllers/
|   |   |-- auth.py            # Blueprint auth (login/logout)
|   |   `-- main.py            # Blueprint principal (menú, usuario, etc.)
|   |-- templates/
|   |   |-- base.html
|   |   |-- login.html
|   |   |-- menu.html
|   |   |-- usuario.html
|   |   |-- preferencias.html
|   |   |-- nuevo.html
|   |   `-- productos.html
|   `-- static/
|       `-- products/          # SVG demo (p1.svg ... p4.svg)
|-- wsgi.py                    # WSGI: app = create_app()
|-- run.py                     # arranque desarrollo
|-- requirements.txt
|-- Dockerfile
|-- docker-compose.yml
|-- .env.example
`-- README.md
```

3.5 Patrón MVC

- **Modelos (M)** (`app/models.py`):
  - `User`: id, email, password_hash (Flask-Login).
  - `Product`: id, name, price, image_path.
- **Vistas (V)**: plantillas Jinja2 en `app/templates`.
- **Controladores (C)**: Blueprints en `app/controllers`.

3.6 Ciclo de vida

1. `wsgi.py` expone `app = create_app()`.
2. La fábrica configura SECRET_KEY, SQLALCHEMY_DATABASE_URI, inicializa `db` y `LoginManager`.

3. Registra `auth_bp` y `main_bp`.
4. Bootstrap: crea tablas y siembra admin y productos demo si faltan.

3.7 Rutas principales

Ruta	Método	Descripción
`/`	GET	Redirige a `/auth/login` si no autenticado; si autenticado, muestra `menu.html`.
`/menu`	GET	Menú principal tras login.
`/usuario`	GET	Datos del usuario autenticado.
`/preferencias`	GET/POST	Cambiar email y/o contraseña.
`/nuevo`	GET/POST	Alta de productos (name, price, image_path).
`/productos`	GET	Listado con imágenes.
`/auth/login`	GET/POST	Formulario de acceso; en éxito redirige a `/menu`.
`/auth/logout`	GET	Cierra sesión y vuelve al login.

3.8 Autenticación

- **Flask-Login** para sesión y `@login_required`.
- Carga de usuario con `user_loader`.
- Hash de contraseñas con Werkzeug.

3.9 Datos y configuración

- SQLite por defecto: `sqlite:///app.db`.
- Variables en `.env`: `SECRET_KEY`, `DATABASE_URL`, `ADMIN_EMAIL`, `ADMIN_PASSWORD`.

3.10 CLI

`init-db` Borra y crea tablas; siembra admin y productos demo.
Uso: `flask --app wsgi.py init-db`

`create-user email password` Crea un usuario.
Uso: `flask --app wsgi.py create-user you@example.com secret`

3.11 Docker

- `Dockerfile`: `python:3.12-slim`, instala `requirements.txt`, expone 5000, arranca `gunicorn` con `wsgi:app`.
- `docker-compose.yml`: servicio `web` con `env_file`, volumen del proyecto y puerto `5000:5000`.

3.12 Alcance y requisitos funcionales

La reestructuración debe mantener toda la funcionalidad implementada en las prácticas anteriores mientras introduce la separación arquitectural del patrón MVC:

3.12.1 Componentes MVC a implementar

Modelo (Model)

- Gestionar los datos de usuarios, credenciales y sesiones
- Contener la lógica de validación de credenciales
- Manejar el estado de la aplicación (sesión activa, datos de usuario)
- Ser independiente completamente de la capa de presentación

Vista (View)

- Mantener toda la interfaz de usuario desarrollada en la Práctica 1 (HTML, CSS responsive)
- Separar las vistas de la lógica de control y negocio
- Recibir datos del controlador para actualizar la presentación
- Capturar interacciones del usuario y delegarlas al controlador

Controlador (Controller)

- Actuar como intermediario entre Modelo y Vista
- Gestionar eventos de la interfaz de usuario
- Invocar métodos del modelo para procesar datos
- Actualizar la vista con los resultados del modelo
- Orquestar el flujo de la aplicación (redirecciones, cambios de estado)

3.12.2 Requisitos técnicos específicos

1. **Separación estricta de capas**: Ningún componente debe tener dependencias directas innecesarias entre Modelo, Vista y Controlador.
2. **Mantenimiento de funcionalidad**: Todas las características de las prácticas 1 y 2 deben conservarse:
    - Formulario de login responsive y accesible
    - Validaciones de frontend existentes
    - Dinamismo JavaScript (manipulación DOM, eventos)
    - Compatibilidad y rendimiento requeridos

3. **Refactorización del JavaScript**: Reestructurar todo el código JavaScript según el patrón MVC, eliminando el acoplamiento actual.
4. **Documentación del proceso**: Incluir en el README una explicación detallada de:
    - Cómo se implementó cada componente MVC
    - Decisiones arquitectónicas tomadas
    - Beneficios observados tras la migración
    - Dificultades encontradas y soluciones aplicadas

3.13 Criterios de evaluación

La evaluación considerará los siguientes aspectos:

- **Correcta implementación del patrón MVC** (40%): Separación clara de responsabilidades y ausencia de acoplamiento indebido.
- **Conservación de funcionalidad** (25%): Todas las características de prácticas anteriores funcionan correctamente.
- **Calidad del código** (20%): Legibilidad, estructuración, comentarios y seguimiento de buenas prácticas.
- **Documentación y justificación** (15%): Claridad en la explicación del proceso de migración y decisiones tomadas.

3.14 Entregables

1. **Código fuente** completamente reestructurado según arquitectura MVC.
2. **README.md** detallado (máximo 3 páginas) que incluya:
    - Diagrama arquitectural mostrando la separación MVC
    - Explicación de cómo cada componente cumple su responsabilidad
    - Evidencias de la separación lograda
    - Reflexión sobre beneficios y desafíos del patrón MVC
3. **Demostración funcional** que muestre que toda la funcionalidad previa se mantiene.

3.15 Conclusión

La migración a MVC representa un paso fundamental en la evolución de cualquier aplicación web, transformando código monolítico en una arquitectura mantenible y escalable. Esta práctica no solo evalúa la capacidad técnica de implementar el patrón, sino también la comprensión conceptual de los principios de diseño que hacen que el software sea sostenible a largo plazo.

La correcta implementación demostrará la madurez en el desarrollo software, mostrando cómo separar adecuadamente las preocupaciones para facilitar testing, mantenimiento y evolución futura del sistema.