marp: true
theme: default
paginate: true

 Modularización y MVC

Organizando nuestro código Flask

Python Full Stack - Clase 27

¿Qué aprenderemos hoy?

- 🎯 Entender el patrón MVC y sus beneficios
- 📁 Crear la estructura modular de carpetas
- 🎮 Separar controladores y modelos
- 💡 Aplicar buenas prácticas de organización

🎯 El Patrón MVC

¿Qué es MVC?

MVC significa **Modelo-Vista-Controlador** (Model-View-Controller).

Es un **patrón de diseño** que organiza el código en tres componentes:

```
📦 Aplicación
  ├── 🎮 Controlador → Maneja rutas y solicitudes
  ├── 📊 Modelo → Interactúa con la base de datos
  └── 🎨 Vista → Muestra información al usuario
```

Beneficios:

- ✅ Código organizado y fácil de mantener
- ✅ Separación clara de responsabilidades
- ✅ Trabajo en equipo sin conflictos
- ✅ Escalabilidad y reutilización

Las Tres Partes de MVC

🎮 Controlador (Controller)

- Recibe las solicitudes HTTP
- Llama al modelo para obtener datos
- Decide qué vista mostrar
- Maneja redirecciones

Modelo (Model)

- Interactúa con la base de datos
- Contiene la lógica de datos
- Representa las tablas como clases
- Consultas SQL y operaciones CRUD

🎨 Vista (View)

- Plantillas HTML
- Lo que ve el usuario
- Recibe datos del controlador
- Muestra información formateada

Flujo de MVC en Acción

```
👤 Usuario hace clic en "Ver todos los tacos"
        ↓
🌐 Navegador envía solicitud HTTP GET /tacos
        ↓
🎮 CONTROLADOR recibe la solicitud
        ↓
🎮 CONTROLADOR llama al MODELO: Taco.get_all()
        ↓
📊 MODELO consulta la base de datos
        ↓
📊 MODELO devuelve lista de objetos Taco
        ↓
🎮 CONTROLADOR pasa datos a la VISTA
        ↓
🎨 VISTA renderiza HTML con los datos
        ↓
🌐 Navegador muestra la página al usuario
```

Tabla Comparativa: Responsabilidades

Componente	Responsabilidades	Ejemplo
Modelo	• Generar tablas en BD • Consultas SQL • Lógica de datos	`Taco.get_all()` `Taco.save(datos)`
Vista	• Página HTML • Mostrar datos • Formularios	`templates/index.html` `templates/resultados.html`
Controlador	• Recibir solicitudes • Llamar al modelo • Renderizar vista	`@app.route('/tacos')` `return render_template(...)`

Modularización

Estructura Modularizada

Modularización significa dividir el código en archivos y carpetas organizados.

```
proyecto_tacos/
  ├── server.py              ← Solo ejecuta la app
  └── flask_app/
      ├── __init__.py        ← Inicializa Flask
      ├── controllers/       ← Rutas y controladores
      ├── models/            ← Clases de base de datos
      ├── config/            ← Configuración
      ├── templates/         ← Plantillas HTML
      └── static/            ← CSS, JS, imágenes
```

Componentes principales:

- `__init__.py` : Configura Flask (`app = Flask(__name__)`)
- `server.py` : Solo importa y ejecuta (`from flask_app import app`)
- `controllers/` : Rutas organizadas por recurso
- `models/` : Clases que representan tablas
- `config/` : Archivos de configuración (mysqlconnection.py)

🎮 Controladores

Controladores

Los **controladores** contienen las **rutas** de la aplicación, organizadas por recurso.

Ejemplo:

```
# flask_app/controllers/tacos.py
from flask_app import app
from flask import render_template, redirect, request
from flask_app.models.taco import Taco

@app.route('/tacos')
def tacos():
    tacos = Taco.get_all()
    return render_template("resultados.html", todos_tacos=tacos)
```

Importante: En `server.py` debemos importar los controladores:

```
from flask_app.controllers import tacos  # Registra las rutas
```

Modelos

Modelos

Los **modelos** son clases que representan las **tablas** de la base de datos.

Ejemplo:

```python
# flask_app/models/taco.py
from flask_app.config.mysqlconnection import connectToMySQL

class Taco:
    @classmethod
    def get_all(cls):
        query = "SELECT * FROM tacos;"
        resultados = connectToMySQL('esquema_tacos').query_db(query)
        tacos = []
        for taco in resultados:
            tacos.append(cls(taco))
        return tacos
```

Importaciones:

- En modelos: `from flask_app.config.mysqlconnection import connectToMySQL`
- En controladores: `from flask_app.models.taco import Taco`

Estructura Final Completa

```
proyecto_tacos/
   ├── server.py
   └── flask_app/
       ├── __init__.py
       ├── config/
       │   └── mysqlconnection.py
       ├── controllers/
       │   └── tacos.py
       ├── models/
       │   └── taco.py
       ├── templates/
       │   ├── index.html
       │   ├── resultados.html
       │   ├── detalle.html
       │   └── editar.html
       └── static/
           ├── css/
           ├── js/
           └── img/
```

¡Estructura modular completa! 🎉

División de Responsabilidades

🎮 Controlador (tacos.py)

```
@app.route('/tacos')
def tacos():
    tacos = Taco.get_all()  # Llama al modelo
    return render_template("resultados.html", todos_tacos=tacos)
```

Responsabilidades:

- ✅ Recibe la solicitud HTTP
- ✅ Llama al modelo para obtener datos
- ✅ Pasa datos a la vista
- ✅ Renderiza la plantilla

División de Responsabilidades

📊 Modelo (taco.py)

```
@classmethod
def get_all(cls):
    query = "SELECT * FROM tacos;"
    tacos_en_bd = connectToMySQL('esquema_tacos').query_db(query)
    tacos = []
    for taco in tacos_en_bd:
        tacos.append(cls(taco))
    return tacos
```

Responsabilidades:

- ✅ Consulta la base de datos
- ✅ Transforma datos en objetos
- ✅ Contiene la lógica de datos
- ✅ Devuelve objetos al controlador

Ejemplo Completo: Crear un Taco

🎮 Controlador

```
@app.route('/crear', methods=['POST'])
def crear():
    datos = {
        "tortilla": request.form['tortilla'],
        "guiso": request.form['guiso'],
        "salsa": request.form['salsa']
    }
    Taco.save(datos)  # Llama al modelo
    return redirect('/tacos')
```

El controlador:

- Recibe los datos del formulario
- Llama al método `save()` del modelo
- Redirige después de guardar

Ejemplo Completo: Crear un Taco

📊 Modelo

```python
@classmethod
def save(cls, datos):
    query = "INSERT INTO tacos (tortilla, guiso, salsa) VALUES(%(tortilla)s, %(guiso)s, %(salsa)s);"
    return connectToMySQL('esquema_tacos').query_db(query, datos)
```

El modelo:

- Recibe el diccionario con los datos
- Ejecuta la consulta SQL
- Guarda en la base de datos

Regla de Oro: Separación de Responsabilidades

🎮 **Controlador:**

- ✅ Maneja rutas y solicitudes HTTP
- ✅ Renderiza plantillas
- ✅ Redirecciones
- ❌ NO hace consultas SQL directamente
- ❌ NO contiene lógica de base de datos

📊 Modelo:

- ✅ Consultas SQL
- ✅ Lógica de datos
- ✅ Transformación de datos
- ❌ NO renderiza plantillas
- ❌ NO maneja rutas

📝 Resumen de Conceptos Clave

Conceptos Clave de MVC

1. **Modelo**: Interactúa con la base de datos
2. **Vista**: Muestra información al usuario (plantillas HTML)
3. **Controlador**: Maneja rutas y conecta modelo con vista

Flujo:

```
Usuario → Controlador → Modelo → Base de Datos
Usuario ← Vista ← Controlador ← Modelo
```

Conceptos Clave de Modularización

1. **flask_app/**: Carpeta principal con todo el código
2. **init.py**: Inicializa la aplicación Flask
3. **controllers/**: Rutas organizadas por recurso
4. **models/**: Clases que representan tablas
5. **config/**: Archivos de configuración
6. **templates/**: Plantillas HTML
7. **static/**: Archivos estáticos (CSS, JS, imágenes)

Ventajas de MVC y Modularización

✅ **Organización**: Código fácil de encontrar

✅ **Mantenibilidad**: Cambios aislados en un solo lugar

✅ **Trabajo en equipo**: Múltiples personas pueden trabajar sin conflictos

✅ **Escalabilidad**: Fácil agregar nuevas funcionalidades

✅ **Separación de responsabilidades**: Cada archivo tiene un propósito claro

✅ **Reutilización**: Modelos y funciones pueden reutilizarse

Estructura de Importaciones

En server.py:

```
from flask_app import app
from flask_app.controllers import tacos
```

En controladores:

```
from flask_app import app
from flask_app.models.taco import Taco
```

En modelos:

```
from flask_app.config.mysqlconnection import connectToMySQL
```

Regla: Siempre usar rutas completas desde `flask_app`

Lo Más Importante

🎯 **MVC separa responsabilidades**: Modelo, Vista, Controlador

📁 **Modularización organiza código**: Cada cosa en su lugar

🔗 **Importaciones correctas**: Usar rutas completas desde `flask_app`

⚙️ **server.py mínimo**: Solo importa y ejecuta la app

🎮 **Controladores manejan rutas**: Una carpeta para organizarlos

📊 **Modelos manejan datos**: Consultas SQL en los modelos

Recursos para Seguir Aprendiendo

📚 **Documentación oficial:**

- Flask: https://flask.palletsprojects.com/
- Python: https://docs.python.org/

🎓 **Conceptos relacionados:**

- Patrones de diseño
- Arquitectura de software
- Clean Code

💡 **Práctica recomendada:**

- Modulariza proyectos anteriores
- Crea nuevos proyectos con estructura MVC desde el inicio

Próximos Pasos

En la siguiente clase aprenderemos sobre:

- 🔐 Autenticación y sesiones
- 🛡️ Validaciones de formularios
- 📧 Mensajes flash
- 🔒 Protección de rutas

¡Sigue practicando la estructura MVC!

Práctica para Casa

Proyecto sugerido:

Modulariza un proyecto anterior que tengas con Flask:

1. Crea la estructura `flask_app/`
2. Separa controladores por recurso
3. Organiza modelos en su carpeta
4. Mueve configuración a `config/`
5. Verifica que todo funciona correctamente

Objetivo: Familiarizarte con la estructura MVC

Consejos Finales

- 💡 **Nombres descriptivos**: Usa nombres claros para archivos y carpetas
- 💡 **Una responsabilidad**: Cada archivo debe tener un propósito específico
- 💡 **Importaciones consistentes**: Siempre desde `flask_app`
- 💡 **Comentarios útiles**: Documenta funciones complejas
- 💡 **Estructura desde el inicio**: Mejor crear MVC desde el principio que refactorizar después

🎉 ¡Felicidades!

Ya sabes modularizar aplicaciones Flask con MVC

Has aprendido a organizar código de manera profesional

🙌 ¡Excelente Trabajo!