**1. Creación y estructura de un proyecto backend con Flask, Swagger, SQLite y el patrón MVC - RO**

Contexto general del proyecto (app de estacionamiento), más lo siguiente **nombrado** “por encima”:

### **🧱 Tecnologías utilizadas:**

* **Flask**: Microframework web de Python.
* **Flask-RESTX / Flask-RESTful**: Extensión para construir APIs REST con documentación Swagger integrada.
* **Swagger-UI**: Interfaz visual para consumir las rutas de la API.
* **SQLite**: Base de datos liviana para persistencia local.
* **SQLAlchemy**: ORM para mapear modelos Python a tablas SQL.
* **python-dotenv**: Para variables de entorno.
* **CORS**: Permitir comunicación entre front-end y backend.
* **Prettier / Linter (flake8, black)**: Estilo y calidad de código.

🗂️ . Estructura del proyecto

Estructura tipo MVC (Modelo - Vista - Controlador), con una capa extra de servicios y repositorios:  
  
**🧠 Modelo**

* patente\_model.py (dentro de models/):  
  + Define cómo luce una patente en la DB.
  + Usa **SQLAlchemy** para declarar las columnas.

### **🕹️ Controlador**

* patente\_controller.py:  
  + Define los endpoints de la API REST.
  + Usa @api.route, @api.doc, @api.expect de **Flask-RESTX** para documentar y validar.
  + Se comunica con los **services**.

### **🔄 Servicio**

* patente\_service.py:  
  + Implementa la lógica de negocio, ej: “registrar patente si no existe”, “formatear respuesta”.
  + Evita que el controlador esté cargado de lógica compleja.

### **🗃️ Repositorio**

* patente\_repository.py:  
  + Abstracción para interactuar con la base de datos (selects, inserts).
  + Evita mezclar lógica de negocio con consultas directas a la DB.

### **🔧 Utilidades**

* validators.py: Validaciones manuales.
* serialize\_patente.py: Serialización (pasar de modelo(objeto-db) a dict, por ejemplo).
* model.py define los modelos de datos que se usan para documentar y validar las solicitudes y respuestas de la API en Swagger.
* register\_namespace.py: Registra los namespaces para Swagger y crea endpoints.
* controller\_loader.py: Carga dinámica de controladores/respuesta de los endpoints.

## **Swagger y documentación**

* Flask-RESTX genera documentación **automática** para la API.

## **Base de Datos - SQLite**

* Archivo patentes.db creado en el raíz si no existe, en caso de existir se lee la DB.
* La configuración y conexión a la base de datos database.py.
* ORM: **SQLAlchemy**
  + Base.metadata.create\_all(engine) inicializa las tablas [database.py](http://database.py).



