

Parte 2

# API REST con Flask y MySQL

*Migraciones con Alembic*

# ¿Por qué Flask para APIs?

## Ventajas de Flask

- Microframework ligero y flexible
- Fácil de aprender
- Excelente para APIs RESTful
- SQLAlchemy ORM potente
- Alembic para migraciones
- Perfecto para microservicios

# Setup Inicial del Proyecto

```
# Instalar dependencias
pip install flask flask-sqlalchemy flask-migrate pymysql
pip install python-dotenv flask-cors
```

```
# Estructura
api/
├── app.py
├── models.py
├── routes.py
├── config.py
├── .env
└── migrations/
```

# Configuración de la aplicación

```
# .env
DATABASE_URL=mysql+pymysql://user:pass@localhost/base_datos
SECRET_KEY=clave_diferente
FLASK_ENV=development

# config.py
import os
from dotenv import load_dotenv

load_dotenv()

class Config:
    SQLALCHEMY_DATABASE_URI = os.getenv('DATABASE_URL')
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    SECRET_KEY = os.getenv('SECRET_KEY')
```

# Aplicación Flask Básica

```
# app.py
from flask import Flask
from flask_migrate import Migrate
from flask_cors import CORS
from config import Config
from models import db

app = Flask(__name__)
app.config.from_object(Config)

db.init_app(app)
migrate = Migrate(app, db)
CORS(app)

import models
import routes
routes.register_routes(app)

if __name__ == '__main__':
    app.run(debug=True)
```

- Crear la aplicación Flask
- Cargar la configuración desde la clase Config
- Conectar la base de datos (SQLAlchemy)
- Activar las migraciones con Flask-Migrate (Alembic)
- Habilitar CORS para permitir peticiones desde otros dominios
- Registrar modelos y rutas
- Ejecutar el servidor en modo desarrollo

# Definiendo Modelos con SQLAlchemy

Los modelos representan tablas en la base de datos.

```
# models.py
from flask_sqlalchemy import SQLAlchemy
db = SQLAlchemy()

class User(db.Model):
    __tablename__ = 'users'

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    posts = db.relationship('Post', backref='author', lazy='dynamic')
```

# Modelo con relaciones

```
# models.py (continuación)
class Post(db.Model):
    __tablename__ = 'posts'

    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(200), nullable=False)
    content = db.Column(db.Text, nullable=False)
    published = db.Column(db.Boolean, default=False)
    user_id = db.Column(db.Integer, db.ForeignKey('users.id'))

    def to_dict(self):
        return {'id': self.id, 'title': self.title}
```

# Configurar migraciones con Alembic

Alembic gestiona cambios en la estructura de la BD.

```
# Inicializar migraciones (sólo una vez)
flask db init

# Crea migrations/ con configuración de Alembic
```

- Migraciones: Scripts que modifican la BD
- Alembic: Genera y ejecuta scripts
- Flask-Migrate: Integración con Flask

# Crear una Migración

```
# Crear migración basada en modelos
flask db migrate -m "Crear tablas de usuarios y posts"
```

Alembic detecta cambios y genera archivo en migrations/versions/

**¡Ejemplo!**

```
# migrations/versions/abc123_create_users.py
def upgrade():
    op.create_table('users',
        sa.Column('id', sa.Integer(), nullable=False),
        sa.Column('name', sa.String(100), nullable=False),
        sa.Column('email', sa.String(120), nullable=False),
        sa.PrimaryKeyConstraint('id')
    )
```

# Aplicar Migraciones

```
# Ejecutar migraciones pendientes  
flask db upgrade
```

```
# Revertir última migración  
flask db downgrade
```

Flujo típico:

1. Modificar models.py
2. flask db migrate -m "descripción"
3. Revisar archivo generado
4. flask db upgrade

# Crear Endpoints REST - GET

```
# routes.py
from flask import jsonify, request, current_app
from models import User, Post, db

def register_routes(app):

    @app.route('/api/users', methods=['GET'])
    def get_users():
        users = User.query.all()
        return jsonify([{'id': u.id, 'name': u.name, 'email': u.email}
                        for u in users])

    @app.route('/api/users/<int:id>', methods=['GET'])
    def get_user(id):
        user = User.query.get_or_404(id)
        return jsonify({'id': user.id, 'name': user.name})
```

# Crear Endpoints REST - POST

```
@app.route('/api/users', methods=['POST'])
def create_user():
    data = request.get_json()

    if not data or 'name' not in data or 'email' not in data:
        return jsonify({'error': 'Missing fields'}), 400

    new_user = User(name=data['name'], email=data['email'])
    db.session.add(new_user)
    db.session.commit()

    return jsonify({'id': new_user.id, 'name': new_user.name}), 201
```

# Endpoints REST - PUT y DELETE

```
@app.route('/api/users/<int:id>', methods=['PUT'])
def update_user(id):
    user = User.query.get_or_404(id)
    data = request.get_json()
    user.name = data.get('name', user.name)
    user.email = data.get('email', user.email)
    db.session.commit()
    return jsonify({'message': 'User updated'})
```

```
@app.route('/api/users/<int:id>', methods=['DELETE'])
def delete_user(id):
    user = User.query.get_or_404(id)
    db.session.delete(user)
    db.session.commit()
    return jsonify({'message': 'User deleted'}), 204
```