

**CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E  
INGENIERÍAS (CUCEI)**

**DIVISIÓN DE ELECTRÓNICA Y COMPUTACIÓN**

**DEPARTAMENTO DE CIENCIAS COMPUTACIONALES**

**Carrera:** Ingeniería en Computación

**Nombre Materia:** Computación Tolerante a Fallas

**Profesor:** López Franco Michel Emanuel

**SECCIÓN:** Do6

**Nombre alumno:** López Arellano Ricardo David

**CODIGO:** 217136143



**Arquitectura monolítica vs Microservicios**

**Fecha de entrega:** 13/11/2023

## Objetivo:

Genera una aplicación utilizando microservicios.

## Desarrollo:

Crear una aplicación basada en microservicios implica dividir la funcionalidad de la aplicación en servicios más pequeños e independientes que se comunican entre sí. Aquí hay un ejemplo simple de cómo estructurar una aplicación utilizando microservicios en Python. En este ejemplo, utilizaremos Flask para crear servicios web y Flask-RESTful para construir una API REST.

Supongamos que estamos construyendo una aplicación de gestión de libros con dos servicios: uno para gestionar la información del libro y otro para gestionar los comentarios sobre los libros.

## Código:

**Código para el servicio de información del libro (book\_service.py):**

```
from flask import Flask, jsonify
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///books.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['SECRET_KEY'] = 'your_secret_key'

db = SQLAlchemy(app)

class Book(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    author = db.Column(db.String(100), nullable=False)

db.create_all()

@app.route('/books', methods=['GET'])
def get_books():
    books = Book.query.all()
    book_list = [{'id': book.id, 'title': book.title, 'author':
book.author} for book in books]
    return jsonify({'books': book_list})

if __name__ == '__main__':
    app.run(port=5001)
```

### Código para el servicio de comentarios del libro (comment\_service.py):

```
from flask import Flask, jsonify
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///comments.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['SECRET_KEY'] = 'your_secret_key'

db = SQLAlchemy(app)

class Comment(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    book_id = db.Column(db.Integer, nullable=False)
    comment = db.Column(db.String(250), nullable=False)

db.create_all()

@app.route('/comments/<int:book_id>', methods=['GET'])
def get_comments(book_id):
    book_comments = Comment.query.filter_by(book_id=book_id).all()
    comment_list = [{'id': comment.id, 'comment': comment.comment} for
comment in book_comments]
    return jsonify({'comments': comment_list})

if __name__ == '__main__':
    app.run(port=5002)
```

### Código para el servicio principal (app.py):

```
from flask import Flask, jsonify
from flask_restful import Api, Resource
from flask_sqlalchemy import SQLAlchemy
from flask_jwt import JWT, jwt_required

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///app.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['SECRET_KEY'] = 'your_secret_key'

db = SQLAlchemy(app)
api = Api(app)

# Definir modelos de la base de datos
class Book(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    author = db.Column(db.String(100), nullable=False)
```

```

class Comment(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    book_id = db.Column(db.Integer, nullable=False)
    comment = db.Column(db.String(250), nullable=False)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    password = db.Column(db.String(80), nullable=False)

# Funciones para autenticación JWT
def authenticate(username, password):
    user = User.query.filter_by(username=username).first()
    if user and user.password == password:
        return user

def identity(payload):
    user_id = payload['identity']
    return User.query.filter_by(id=user_id).first()

jwt = JWT(app, authenticate, identity)

# Recurso para obtener la lista de libros
class BookListResource(Resource):
    @jwt_required()
    def get(self):
        books = Book.query.all()
        book_list = [{'id': book.id, 'title': book.title, 'author':
book.author} for book in books]
        return jsonify({'books': book_list})

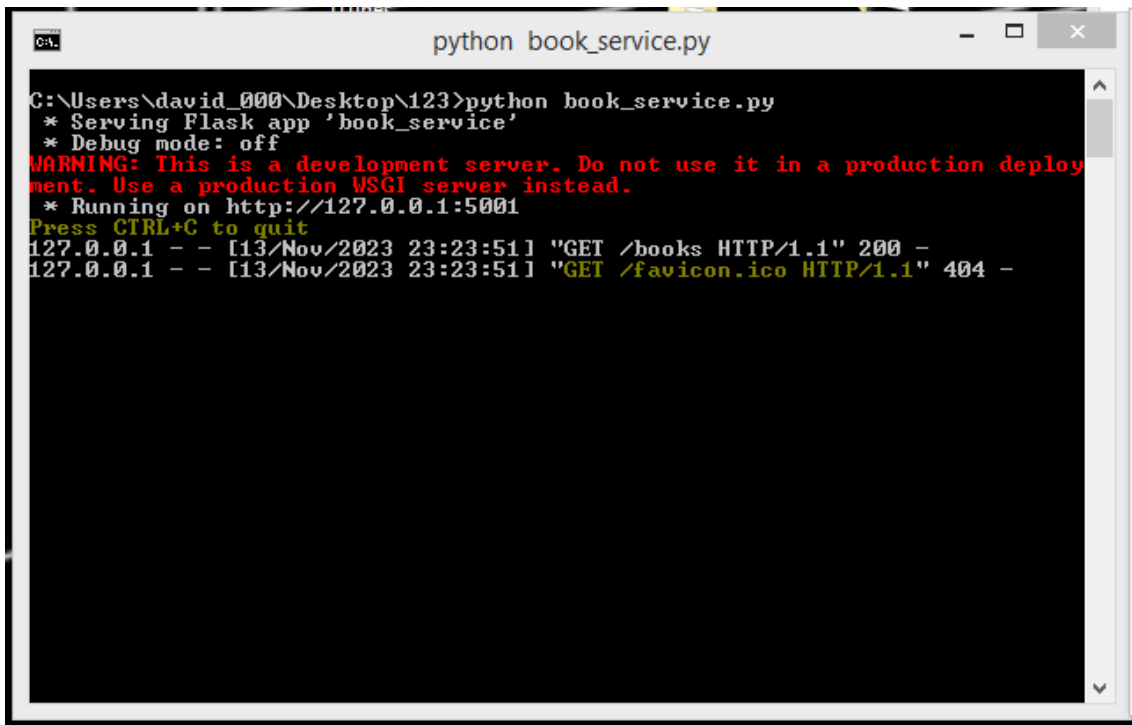
# Agregar el recurso a la API
api.add_resource(BookListResource, '/books')

if __name__ == '__main__':
    # Crear las tablas en la base de datos antes de ejecutar la
    aplicación
    db.create_all()
    app.run(port=5000)

```

## Resultados:

### Book\_service:

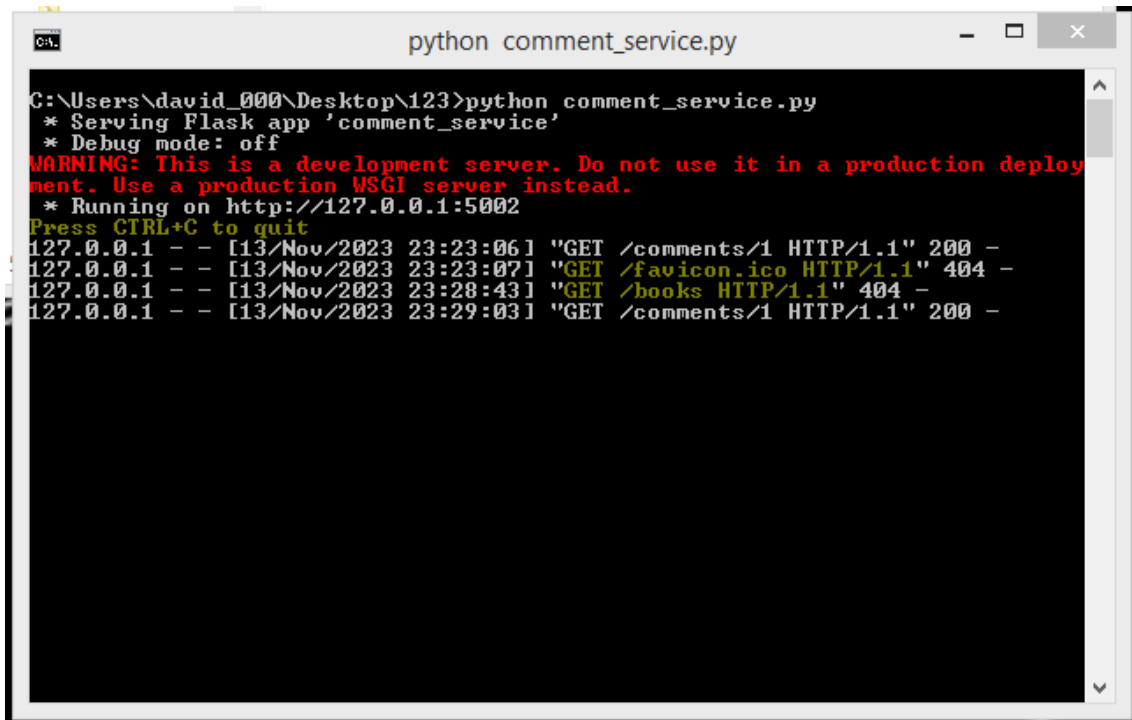


```
python book_service.py

C:\Users\david_000\Desktop\123>python book_service.py
* Serving Flask app 'book_service'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5001
Press CTRL+C to quit
127.0.0.1 - - [13/Nov/2023 23:23:51] "GET /books HTTP/1.1" 200 -
127.0.0.1 - - [13/Nov/2023 23:23:51] "GET /favicon.ico HTTP/1.1" 404 -
```

*\*Sale cada una de las veces y la hora en que entramos en el local host.*

### comment\_service:

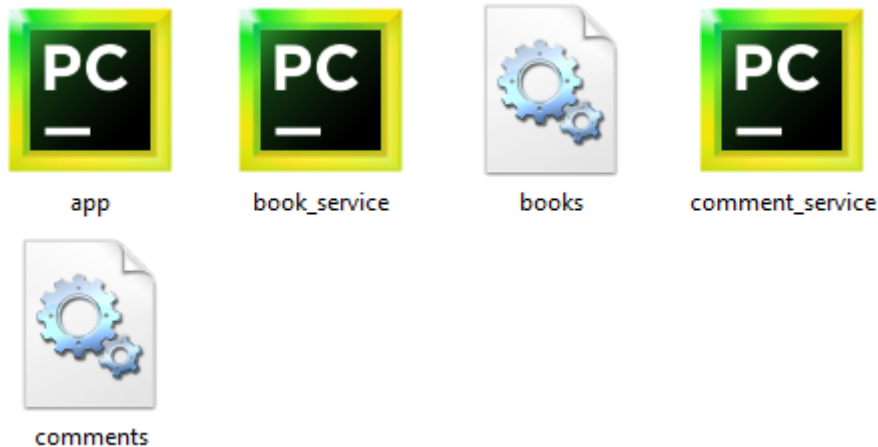


```
python comment_service.py

C:\Users\david_000\Desktop\123>python comment_service.py
* Serving Flask app 'comment_service'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5002
Press CTRL+C to quit
127.0.0.1 - - [13/Nov/2023 23:23:06] "GET /comments/1 HTTP/1.1" 200 -
127.0.0.1 - - [13/Nov/2023 23:23:07] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [13/Nov/2023 23:28:43] "GET /books HTTP/1.1" 404 -
127.0.0.1 - - [13/Nov/2023 23:29:03] "GET /comments/1 HTTP/1.1" 200 -
```

*\*Sale cada una de las veces y la hora en que entramos en el local host.*

### Creación de archivos:



*\*Al tener todos los archivos en la misma carpeta, nos crea esos “.db” a la hora de la ejecución en la ventana de comandos.*

*\*En el LocalHost solo aparece lo mismo que está en las salidas de las líneas de los códigos solo en html (en el navegador).*

## Conclusión:

Este ejemplo proporciona una introducción a la arquitectura de microservicios y cómo se pueden implementar utilizando herramientas y bibliotecas populares en el ecosistema de Python. Es importante destacar que este es un modelo simplificado y que en entornos de producción, se deben considerar aspectos adicionales como seguridad, escalabilidad, gestión de errores, y más.

Además, al trabajar con microservicios, es crucial diseñar una comunicación eficiente entre ellos. En este ejemplo, utilizamos solicitudes HTTP para la comunicación entre los microservicios, pero en aplicaciones más complejas, podrías considerar el uso de protocolos y tecnologías como gRPC, RabbitMQ o Kafka.

En resumen, este trabajo proporciona una base para comprender la implementación de microservicios en Python y puede servir como punto de partida para proyectos más complejos y aplicaciones del mundo real.