

# Examen Final

---

Curso: Administración de Redes  
Semestre: 2024-II  
Ciencias de la Computación - UNI

**Apellidos y Nombres:** Pacheco Taboada André Joaquín

**Código:** 20222189G

## Desarrollo del Examen: Sistema de APIs con Flask

Para este examen, desarrollé un sistema de tres APIs interconectadas usando Flask y TinyDB. Separé cada API en su propio archivo para mantener el código organizado y fácil de mantener.

### 1. Configuración de Base de Datos (`db_config.py`)

Primero, implementé la configuración central de la base de datos usando TinyDB. Aquí definí las bases de datos compartidas para usuarios y mensajes, junto con funciones de utilidad.

```
practices > FinalExam > api_project > apis > db_config.py > ...
16 from tinydb import TinyDB, Query
15 import os
14
13 # Inicializar directorio de base de datos
12 db_path = 'db'
11 if not os.path.exists(db_path):
10 |     os.makedirs(db_path)
9
8 # Bases de datos compartidas
7 users_db = TinyDB(os.path.join(db_path, 'users.json'))
6 messages_db = TinyDB(os.path.join(db_path, 'messages.json'))
5
4 # Función de utilidad para verificar existencia de usuario
3 def check_user_exists(username):
2 |     User = Query()
1 |     return users_db.search(User.usuario == username)
17
```

### 2. API de Registro (`registro_api.py`)

Desarrollé la primera API para manejar el registro de usuarios. Esta API recibe solicitudes POST con un nombre de usuario y lo almacena en la base de datos si no existe.

```

practices > FinalExam > api_project > apis > registro_api.py > ...
25 from flask import Blueprint, request, jsonify
24 from .db_config import users_db, Query
23
22 # Crear Blueprint para la API de registro
21 registro_bp = Blueprint('registro', __name__)
20
19 @registro_bp.route('/register', methods=['POST'])
18 def register():
17     """
16     API de Registro: Recibe un usuario y lo registra en la base de datos
15     """
14     data = request.get_json()
13     if not data or 'usuario' not in data:
12         return jsonify({'error': 'Usuario no proporcionado'}), 400
11
10     user = data['usuario']
9     User = Query()
8
7     # Verificar si el usuario ya existe
6     if users_db.search(User.usuario == user):
5         return jsonify({'error': 'Usuario ya existe'}), 400
4
3     # Registrar nuevo usuario
2     users_db.insert({'usuario': user})
1     return jsonify({'message': 'Usuario registrado exitosamente'}), 201
26

```

### 3. API de Autenticación (autenticacion\_api.py)

La segunda API verifica si un usuario está registrado. Retorna "Ok" si el usuario existe o "Usuario no Registrado" si no existe.

```

practices > FinalExam > api_project > apis > autenticacion_api.py > ...
21 from flask import Blueprint, request, jsonify
20 from .db_config import check_user_exists
19
18 # Crear Blueprint para la API de autenticación
17 autenticacion_bp = Blueprint('autenticacion', __name__)
16
15 @autenticacion_bp.route('/authenticate', methods=['POST'])
14 def authenticate():
13     """
12     API de Autenticación: Verifica si un usuario está registrado
11     """
10     data = request.get_json()
9     if not data or 'usuario' not in data:
8         return jsonify({'error': 'Usuario no proporcionado'}), 400
7
6     user = data['usuario']
5
4     # Verificar si el usuario existe
3     if check_user_exists(user):
2         return jsonify({'message': 'Ok'}), 200
1     return jsonify({'message': 'Usuario no Registrado'}), 404
22

```

#### 4. API de Mensajes (`mensajes_api.py`)

La tercera API maneja la escritura y lectura de mensajes. Implementé dos endpoints:

- Escritura: Guarda mensajes de usuarios registrados
- Lectura: Retorna todos los mensajes almacenados

Endpoint de escritura:

```
practices > FinalExam > api_project > apis > mensajes_api.py > ...
53 from flask import Blueprint, request, jsonify
52 from .db_config import messages_db, check_user_exists
51
50 # Crear Blueprint para la API de mensajes
49 mensajes_bp = Blueprint('mensajes', __name__)
48
47 @mensajes_bp.route('/messages/write', methods=['POST'])
46 def write_message():
45     """
44     API de Mensajes - Escritura: Guarda un mensaje de un usuario registrado
43     """
42     data = request.get_json()
41     if not data or 'usuario' not in data or 'mensaje' not in data:
40         return jsonify({'error': 'Usuario y mensaje son requeridos'}), 400
39
38     user = data['usuario']
37     message = data['mensaje']
36
35     # Primero autenticar al usuario
34     if not check_user_exists(user):
33         return jsonify({'error': 'Usuario no Registrado'}), 404
32
31     # Guardar el mensaje
30     messages_db.insert({
29         'usuario': user,
28         'mensaje': message
27     })
26     return jsonify({'message': 'Mensaje guardado exitosamente'}), 201
```

Endpoint de lectura:

```
practices > FinalExam > api_project > apis > mensajes_api.py > write_message
5  def write_message():
15      return jsonify({'message': 'Mensaje guardado exitosamente'}), 201
16
17  @mensajes_bp.route('/messages/read', methods=['POST'])
18  def read_messages():
19      """
20      API de Mensajes - Lectura: Retorna todos los mensajes para usuarios registrados
21      """
22      data = request.get_json()
23      if not data or 'usuario' not in data:
24          return jsonify({'error': 'Usuario no proporcionado'}), 400
25
26      user = data['usuario']
27
28      # Primero autenticar al usuario
29      if not check_user_exists(user):
30          return jsonify({'error': 'Usuario no Registrado'}), 404
31
32      # Obtener todos los mensajes
33      messages = messages_db.all()
34
35      # Formatear la respuesta según el ejemplo
36      formatted_messages = []
37      for msg in messages:
38          formatted_messages.append([msg['usuario'], msg['mensaje']])
39
40      return jsonify(formatted_messages), 200
41
```

## 5. Programa Principal (main.py)

Finalmente, creé un programa principal que integra las tres APIs y ejecuta una demostración automática de todas las funcionalidades. El ejemplo de ejecución que usé es el mismo que se muestra en el enunciado del examen.

```
Cliente: Registrar("uni")
Cliente: Autenticar("uni")
Respuesta: "Ok"
Cliente: Autenticar("fc")
Respuesta: "Usuario no Registrado"
Cliente: Registrar("fc")
Cliente: Escribir("uni","Este es un mensaje")
Cliente: Escribir("uni","Este es un segundo mensaje")
Cliente: Escribir("fc","Tercer mensaje")
Cliente: Escribir("faua","Otro mensaje")
Cliente: LeerTodo("uni")
Respuesta: {
  "uni","Este es un mensaje",
  "uni","Este es un segundo mensaje",
  "fc","Tercer mensaje"
}
```

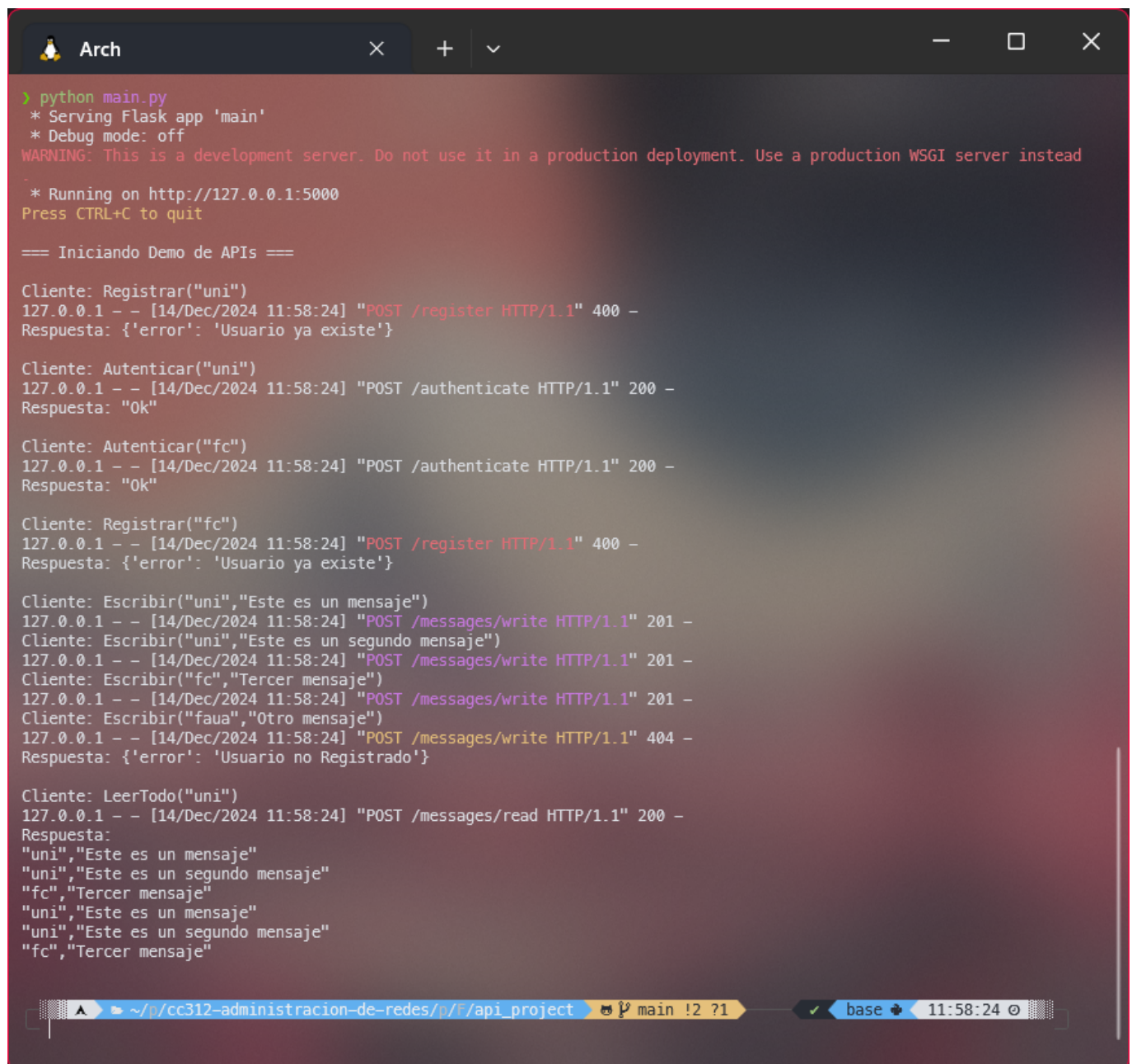
Agregué un poco de logging para que se pueda visualizar las peticiones (métodos POST) que se hacen a los endpoints de las APIs.

Se corre el programa con el siguiente comando:

```
python main.py
```

## Resultados de la Ejecución

Al ejecutar el programa, se realizan automáticamente todas las operaciones solicitadas en el ejemplo:



```
Arch
python main.py
* Serving Flask app 'main'
* 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://127.0.0.1:5000
Press CTRL+C to quit

=== Iniciando Demo de APIs ===

Cliente: Registrar("uni")
127.0.0.1 - - [14/Dec/2024 11:58:24] "POST /register HTTP/1.1" 400 -
Respuesta: {'error': 'Usuario ya existe'}

Cliente: Autenticar("uni")
127.0.0.1 - - [14/Dec/2024 11:58:24] "POST /authenticate HTTP/1.1" 200 -
Respuesta: "Ok"

Cliente: Autenticar("fc")
127.0.0.1 - - [14/Dec/2024 11:58:24] "POST /authenticate HTTP/1.1" 200 -
Respuesta: "Ok"

Cliente: Registrar("fc")
127.0.0.1 - - [14/Dec/2024 11:58:24] "POST /register HTTP/1.1" 400 -
Respuesta: {'error': 'Usuario ya existe'}

Cliente: Escribir("uni","Este es un mensaje")
127.0.0.1 - - [14/Dec/2024 11:58:24] "POST /messages/write HTTP/1.1" 201 -
Cliente: Escribir("uni","Este es un segundo mensaje")
127.0.0.1 - - [14/Dec/2024 11:58:24] "POST /messages/write HTTP/1.1" 201 -
Cliente: Escribir("fc","Tercer mensaje")
127.0.0.1 - - [14/Dec/2024 11:58:24] "POST /messages/write HTTP/1.1" 201 -
Cliente: Escribir("faua","Otro mensaje")
127.0.0.1 - - [14/Dec/2024 11:58:24] "POST /messages/write HTTP/1.1" 404 -
Respuesta: {'error': 'Usuario no Registrado'}

Cliente: LeerTodo("uni")
127.0.0.1 - - [14/Dec/2024 11:58:24] "POST /messages/read HTTP/1.1" 200 -
Respuesta:
"uni","Este es un mensaje"
"uni","Este es un segundo mensaje"
"fc","Tercer mensaje"
"uni","Este es un mensaje"
"uni","Este es un segundo mensaje"
"fc","Tercer mensaje"

~/cc312-administracion-de-redes/p/f/api_project  main !2 ?1  base 11:58:24
```

## En texto

```
> python main.py
* Serving Flask app 'main'
* 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://127.0.0.1:5000
Press CTRL+C to quit
```

```
=== Iniciando Demo de APIs ===
```

```
Cliente: Registrar("uni")
127.0.0.1 - - [14/Dec/2024 11:58:24] "POST /register HTTP/1.1" 400 -
Respuesta: {'error': 'Usuario ya existe'}
```

```
Cliente: Autenticar("uni")
127.0.0.1 - - [14/Dec/2024 11:58:24] "POST /authenticate HTTP/1.1" 200 -
Respuesta: "Ok"
```

```
Cliente: Autenticar("fc")
127.0.0.1 - - [14/Dec/2024 11:58:24] "POST /authenticate HTTP/1.1" 200 -
Respuesta: "Ok"
```

```
Cliente: Registrar("fc")
127.0.0.1 - - [14/Dec/2024 11:58:24] "POST /register HTTP/1.1" 400 -
Respuesta: {'error': 'Usuario ya existe'}
```

```
Cliente: Escribir("uni","Este es un mensaje")
127.0.0.1 - - [14/Dec/2024 11:58:24] "POST /messages/write HTTP/1.1" 201 -
Cliente: Escribir("uni","Este es un segundo mensaje")
127.0.0.1 - - [14/Dec/2024 11:58:24] "POST /messages/write HTTP/1.1" 201 -
Cliente: Escribir("fc","Tercer mensaje")
127.0.0.1 - - [14/Dec/2024 11:58:24] "POST /messages/write HTTP/1.1" 201 -
Cliente: Escribir("faua","Otro mensaje")
127.0.0.1 - - [14/Dec/2024 11:58:24] "POST /messages/write HTTP/1.1" 404 -
Respuesta: {'error': 'Usuario no Registrado'}
```

```
Cliente: LeerTodo("uni")
127.0.0.1 - - [14/Dec/2024 11:58:24] "POST /messages/read HTTP/1.1" 200 -
Respuesta:
"uni","Este es un mensaje"
"uni","Este es un segundo mensaje"
"fc","Tercer mensaje"
"uni","Este es un mensaje"
"uni","Este es un segundo mensaje"
"fc","Tercer mensaje"
```