

Desarrollo Backend de QuetzAI - Informe de Progreso: Mes 1

Fecha: 4 de julio de 2025

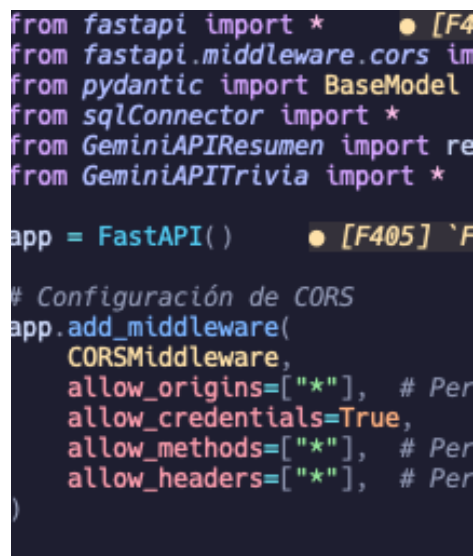
Autor: Eduardo Muñoz González

1. Resumen

Este informe describe el trabajo fundamental completado en el backend de QuetzAI durante el primer mes de desarrollo. El enfoque principal fue establecer una arquitectura de servidor robusta y escalable, configurar la base de datos, definir los modelos de datos iniciales e implementar la funcionalidad central para el registro de visitantes. Los cimientos técnicos establecidos este mes sostendrán las funcionalidades más complejas planeadas para los próximos ciclos de desarrollo.

2. Logros Clave

- **Selección y Configuración del Stack Tecnológico:**
 - **Framework:** Se eligió FastAPI como framework web por su alto rendimiento, capacidades asíncronas y generación automática de documentación.
 - **Base de datos:** Se optó por MySQL como sistema relacional para almacenar datos de usuarios y de la aplicación.
 - **Dependencias:** Se creó un archivo requirements.txt para gestionar todas las dependencias de Python y asegurar un entorno reproducible. Las librerías clave incluyen fastapi, uvicorn, mysql-connector-python y pydantic



```
from fastapi import *
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
from sqlConnector import *
from GeminiAPIResumen import re
from GeminiAPITrivia import *

app = FastAPI()

# Configuración de CORS
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"], # Per
    allow_credentials=True,
    allow_methods=["*"], # Per
    allow_headers=["*"], # Per
)
```

- **Arquitectura Inicial del Servidor:**

- Se desarrolló una aplicación básica de FastAPI (Backend.py), con un endpoint raíz (/) para comprobaciones de estado.
- Se configuró el middleware CORS (Cross-Origin Resource Sharing) para permitir solicitudes desde el frontend en React Native.

```

1  # Configuración de CORS
2  app.add_middleware(
3      CORSMiddleware,
4      allow_origins=["*"], # Permitir todas las solicitudes
5      allow_credentials=True,
6      allow_methods=["*"], # Permitir todos los métodos (GET, POST, etc.)
7      allow_headers=["*"], # Permitir todos los headers
8  )
9
10 # Pydantic model para las peticiones del NFC
11 class ScanRequest(BaseModel):
12     scan_data: str # NFC ID
13     id_visitante: int
14
15 # Pydantic model para los visitantes
16 class Visitante(BaseModel):
17     nombre: str
18     edad: int
19
20 # Modelo para recibir la solicitud

```

• Integración de Base de Datos:

- Se creó un módulo de conector a la base de datos (sqlConnector.py) para centralizar todas las interacciones con MySQL, mejorando la limpieza y mantenibilidad del código.
- Se implementaron funciones para crear y recuperar la información de visitantes.

```

1  # Description: Aquí se encuentran las funciones que se encargan de la conexión con la base de datos
2  #* Status: Complete
3  import os
4  import mysql.connector
5  from mysql.connector import Error
6
7  # Funcion para crear la conexión a la base de datos
8  def create_connection():
9      """Crear una conexión a la base de datos"""
10     try:
11         connection = mysql.connector.connect(
12             host = os.getenv("DB_HOST"),
13             port=int(os.getenv("DB_PORT", 3306)),
14             database = os.getenv("DB_NAME"),
15             user = os.getenv("DB_USER"),
16             password = os.getenv("DB_PASSWORD")
17         )
18         if connection.is_connected():
19             print("Conexión exitosa a la base de datos 'museo'")
20             return connection
21     except Error as e:
22         print(f"Error al conectarse a MySQL: {e}")
23         return None
24
25 # Funcion para crear la conexión a la base de datos
26 def close_connection(connection):
27     """Cerrar la conexión a la base de datos"""
28     if connection.is_connected():
29         connection.close()
30         print("Conexión cerrada")
31
32 # Funcion para crear un nuevo usuario temporal, no retorna nada
33 def crear_usuario_temporal(nombre, edad):
34     """Crear un nuevo usuario temporal al escanear un QR"""
35     connection = create_connection()
36     if connection is not None:
37         try:
38             cursor = connection.cursor()
39             insert_query = "INSERT INTO Visitante (Nombre, Edad, Puntuacion_trivia) VALUES (%s, %s, %s)" # Información dada desde el
40             cursor.execute(insert_query, (nombre, edad, 0))
41             connection.commit()
42             print(f"Usuario temporal '{nombre}' creado con éxito.")
43         except Error as e:
44             print(f"Error al crear usuario temporal: {e}")
45         finally:
46             close_connection(connection)

```

- **Implementación de la Funcionalidad Principal: Registro de Visitantes:**
 - Se implementó con éxito el endpoint /registrar_visitante.
 - Se definió un modelo de Pydantic (Visitante) para validar los datos entrantes (nombre, edad), garantizando la integridad de la información.
 - El endpoint procesa la solicitud, crea un nuevo usuario en la base de datos y devuelve un id_visitante único.
- **Contenerización:**
 - Se crearon un Dockerfile y un docker-compose.yml para contenerizar la aplicación FastAPI. Esto simplifica la configuración del entorno de desarrollo y asegura consistencia con futuros entornos de producción.

```
12 FROM python:3.12.10-slim
11
10 # Establece el directorio de trabajo dentro del contenedor
9 WORKDIR /app
8
7 # Copia los requisitos e instálalos
6 COPY requirements.txt .
5
4 RUN apt update && apt install -y curl
3
2
1 RUN pip install --no-cache-dir -r requirements.txt
13
1 # Copia el resto del código fuente
2 COPY . .
3
4 # Expone el puerto en el que correrá FastAPI
5 EXPOSE 8000
6
7 # Comando por defecto para iniciar el servidor
8 CMD ["uvicorn", "Backend:app", "--host", "0.0.0.0", "--port", "8000"]
```