🧩 Proyecto: Sistema Publisher/Subscriber con FastAPI y MQTT

Autores: Andrés David Pérez Cely, Daniel Fernando González Cortés, Juan Diego Reyes Rodríguez

🧠 Descripción del Proyecto

Este proyecto implementa el **patrón de arquitectura Publisher–Subscriber (Pub/Sub)** utilizando **FastAPI**, **MySQL**, **MQTT (Mosquitto/HiveMQ)** y un frontend **Vue.js**.

El sistema está dividido en dos servicios principales:

- **Publisher:** publica mensajes en un *topic MQTT* y los guarda en una base de datos MySQL.
- **Subscriber:** escucha el mismo *topic MQTT*, recibe los mensajes, los almacena en su propia base de datos y notifica en tiempo real mediante **Server-Sent Events (SSE)**.

⚙️ Stack Tecnológico

Componente	Tecnología	Función Principal
Frontend	Vue.js	Interfaz web dinámica y reactiva
Backend	FastAPI	API REST y lógica de publicación/suscripción
Base de datos	MySQL	Almacenamiento de publicaciones
Broker MQTT	HiveMQ / Mosquitto	Canal de comunicación Pub/Sub
Patrón arquitectónico	Publisher / Subscriber	Comunicación asíncrona y desacoplada

🚀 Instalación y Ejecución

1️⃣ Crear entorno virtual

```
python -m venv venv
```

2️⃣ Activar entorno

```
# En Windows
venv\Scripts\activate
# En Linux / Mac
source venv/bin/activate
```

3 Instalar dependencias

```
pip install -r requirements.txt
```

4 Guardar dependencias (si agregas nuevas)

```
pip freeze > requirements.txt
```

5 Ejecutar el servidor

```
uvicorn main:app --reload
```

Por defecto, el servidor se ejecuta en:

```
http://127.0.0.1:8000
```

Endpoints Principales

Publisher

Método	Endpoint	Descripción
`POST`	`/publication`	Crea una publicación, la guarda en MySQL y la envía al broker MQTT
`GET`	`/publications`	Devuelve todas las publicaciones almacenadas

Subscriber

Método	Endpoint	Descripción
`GET`	`/`	Retorna todas las publicaciones almacenadas
`GET`	`/stream`	Endpoint de **Server-Sent Events (SSE)** para notificaciones en tiempo real

---

Comunicación MQTT

- **Broker:** `broker.hivemq.com`
- **Puerto:** `1883`
- **Topic:** `mi/topico/de/prueba`

Calidad de Servicio (QoS)

Nivel	Descripción	Uso recomendado
QoS 0	Envío sin confirmación ("como máximo una vez")	Datos no críticos
QoS 1	Confirmación de recepción ("al menos una vez")	Comunicación general
QoS 2	Entrega garantizada ("exactamente una vez")	Datos críticos o transacciones

Arquitectura

El patrón **Publisher–Subscriber** se caracteriza por:

- **Desacoplamiento:** publicadores y suscriptores son independientes.
- **Asincronía:** comunicación no bloqueante.
- **Basado en eventos:** el flujo depende de la llegada de mensajes MQTT.
- **Escalabilidad:** múltiples instancias pueden conectarse al mismo topic.
- **Fiabilidad:** control de entrega con niveles QoS.

Esquema General

```
Publisher → Broker MQTT → Subscriber → SSE → Frontend
```

Flujo de Ejecución

1. El **Publisher** crea una publicación (`POST /publication`).
2. El mensaje se guarda en la base de datos y se envía al **broker MQTT**.
3. El **Subscriber** escucha el topic, recibe el mensaje, lo almacena y lo reenvía mediante **SSE**.
4. El **Frontend** muestra la publicación en tiempo real sin necesidad de recargar.

Casos de Uso

- **Notificaciones en tiempo real**
- **Mensajería distribuida**
- **Monitoreo IoT**
- **Automatización industrial**
- **Actualización de sistemas descentralizados**

Principios SOLID y Calidad del Software

Principio	Implementación
S (Responsabilidad Única)	Publisher y Subscriber tienen responsabilidades separadas
O (Abierto/Cerrado)	Nuevos endpoints o topics pueden agregarse sin alterar los existentes

Principio	Implementación
L (Sustitución de Liskov)	Servicios intercambiables mientras cumplan la interfaz
I (Segregación de Interfaces)	Inyección de dependencias para aislar responsabilidades
D (Inversión de Dependencias)	FastAPI desacoplado de la base de datos y del broker MQTT

Atributos de Calidad (ISO/IEC 25010)

- **Adecuación funcional:** cumple los requerimientos de comunicación asíncrona.
- **Eficiencia:** bajo tiempo de respuesta gracias a FastAPI y MQTT.
- **Compatibilidad:** integrable con frontend Vue.js y servicios externos.
- **Fiabilidad:** manejo robusto de errores y persistencia con SQLAlchemy.

📦 Dockerización (opcional)

Puedes ejecutar los servicios con `docker-compose` para aislar los contenedores del **publisher**, **subscriber** y **MySQL**:

```
version: '3.9'
services:
  publisher_db:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: publisher
    ports:
      - "3307:3306"

  subscriber_db:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: subscriber
    ports:
      - "3308:3306"

  publisher:
    build: ./publisher
    depends_on:
      - publisher_db
    ports:
      - "8000:8000"

  subscriber:
    build: ./subscriber
    depends_on:
      - subscriber_db
```

```
    ports:
      - "8001:8000"
```

🌍 Enlaces Relevantes

- **Repositorio:** https://github.com/talleres-arqui/presentacion-1
- **Release oficial:** https://github.com/talleres-arqui/presentacion-1/releases/tag/v1.0

👨‍💻 Comandos útiles

```
# Crear entorno virtual
python -m venv venv

# Activar entorno
venv\Scripts\activate  # (Windows)
source venv/bin/activate  # (Linux/Mac)

# Instalar dependencias
pip install -r requirements.txt

# Guardar dependencias
pip freeze > requirements.txt

# Ejecutar
```