**Problem Statement: Employee Record Transmission and Storage System**

**Objective**

Develop a high-performance client-server application that processes employee records from a .csv file and stores them in a **MySQL database**. The solution must leverage **Python advanced coding techniques**, including **function decorators, data classes, and concurrency**, for an efficient and scalable design.

**Requirements**

**1. Data Input (Client-Side)**

* The client application will **read a .csv file** containing **1000 employee records**.
* Each record consists of fields such as:
  + employee\_id
  + name
  + email
  + department
  + designation
  + salary
  + date\_of\_joining

**2. Client Implementation**

* The client should process the .csv file and send records **individually** to the server.
* Use **one of the following communication mechanisms** for sending data:
  + **HTTP API** (RESTful API with POST requests)
  + **Kafka (Pub/Sub model)**
  + **WebSockets (Real-time transmission)**
* Utilize **asynchronous programming** (asyncio, aiohttp, aiokafka) for efficient data transmission.

**3. Server Implementation**

* The server must receive, validate, and store records in a **MySQL database**.
* Use **function decorators** for:
  + Logging request processing.
  + Validating incoming records.
* Use **data classes (dataclass)** to represent employee records.
* Implement **bulk inserts** to optimize database performance.
* Implement **error handling** (e.g., handling duplicate records, database failures).

**4. Advanced Coding Expectations**

* **Function Decorators**:
  + Decorators for logging execution time, error handling, and input validation.
* **Data Classes (dataclass)**:
  + Represent employee records using Python dataclass with type hints.
* **Concurrency**:
  + Use **async/await** with asyncio for non-blocking requests.
  + Optionally use **ThreadPoolExecutor** for parallel processing.

**Expected Deliverables**

1. **Client Application**
   * Reads .csv and transmits records efficiently.
   * Uses advanced Python techniques (async, decorators).
2. **Server Application**
   * Receives, validates, and stores records in MySQL.
   * Uses dataclass and decorators for optimized processing.
3. **Database Schema**
   * MySQL table schema for storing employee records.
4. **Configuration & Instructions**
   * README with setup instructions and dependency requirements.

**Evaluation Criteria**

✅ **Code Quality & Structure**  
✅ **Usage of Advanced Python Concepts** (decorators, data classes, async programming)  
✅ **Performance Optimization** (batch inserts, non-blocking requests)  
✅ **Error Handling & Logging**  
✅ **Scalability & Fault Tolerance** (handling failures, retries, deduplication)

**Bonus Points**

⭐ Implement **JWT-based authentication** for secured API requests.  
⭐ Use **Docker** for containerization.  
⭐ Provide **unit tests** using pytest.  
⭐ Implement **data validation** using pydantic.