Homework #2 - Development of a basic REST-based application with Token authentication

Overview

In this homework, your task is to create a small microservice-based application in Python using FastAPI. The goal is to get solid knowledge in building a simple application, as a set of such applications would further form a solution.

Requirements

1. **Microservice Decomposition**

    - **Business Logic Service**
        - Exposes a single core endpoint (e.g., /process) to perform the main "longer-running" logic – for instance, ML model inference, data transformations, or any processing that might take a few seconds.
            1. Possible, but not mandatory possibility is to call API of some LLM and get its response (you can check out https://openrouter.ai/, which has free models that you can call).
        - Includes a /health (health check) endpoint for basic status reporting.
        - If the user calls the root (/) of the service, it should returna short description.

    - **Database Service**
        - A service that handles reading and writing data.
        - You can simulate a database by storing data in a Python in-memory structure (lists, dictionaries, etc.).
        - It should have at least two endpoints:
            1. One for writing/saving data (e.g., /write or /save),
            2. One for reading/retrieving data (e.g., /read or /get).
        - Includes a /health endpoint.
        - If the user calls the root (/) of the service, it should return a short description.

- **Client Service**
  - The only service that external clients (users) can directly call.
  - Orchestrates calls to both the Database Service and the Business Logic Service. For instance:
    1. Reads some data from the Database Service.
    2. Calls the Business Logic Service to process or transform the data.
    3. Saves the result back to the Database Service.
    4. Returns the final response to the user.
  - It requires **simple token-based access** so that users can provide a token to call it (e.g., via a request header).
  - Includes its own /health endpoint.
  - If the user calls the root (/) of the service, it should return a short description.

2. **Security Constraint**
   - The **Client Service** is the **only publicly accessible endpoint** (apart from its /health); users should **not** have direct access to the Database or Business Logic services.
   - A minimal token-based mechanism is sufficient. For example, you can define a fixed token in an environment variable or in code and require that token in the "Authorization" header.
3. **Health Check Endpoints**
   - Each service must provide a /health endpoint that returns a simple JSON status (e.g., {"status": "ok"}) so that you can quickly verify whether each service is running.
4. **Optional Extensions**
   - You can split the Business Logic Service into multiple sub-services (for example, one for data preprocessing, one for the actual "ML model," etc.). This is optional.
   - You can demonstrate asynchronous calls, add Docker support, or create a simple Docker Compose file to run all services together—although not strictly required, it is an excellent opportunity to learn.

Deliverables

1. **Source Code**
   - Separate your microservices into clearly distinguished files/modules (for example, client_service.py, business_service.py, db_service.py, or similar).
   - Each service must be a small FastAPI application with its respective endpoints.
2. **README/Documentation**
   - Provide clear instructions on how to run each service or start them all together
   - Explain how the token-based authentication works for the Client Service.
   - Summarize the request flow: **Client → (Client Service) → Database Service → Business Logic Service → Database Service → Client**.

3. **Example Usage**
    - Show a brief example of the HTTP requests (e.g., using curl, Postman, or a Python requests script) that demonstrates how a user can interact with the Client Service endpoint, triggers the orchestration flow by using client-server, and gets a result. It would also be beneficial to provide a script or scripts to start the app with one command.

Implementation Hints

FastAPI Quick Start (From our lecture example)

from fastapi import FastAPI, Request

```
app = FastAPI()

@app.get("/health")
def health_check():
    return {"status": "ok"}
```

Token Check

```
from fastapi import FastAPI, Header, HTTPException

APP_TOKEN = "YourSuperSecretToken" # That should not be hardcoded or exposed

@app.get("/some-protected-route")
def protected_route(authorization: str = Header(None)):
    if authorization != f"Bearer {APP_TOKEN}":
        raise HTTPException(status_code=401, detail="Unauthorized")
    return {"message": "You are authorized!"}
```

Processing Endpoint

```
# business_service.py
import time
from fastapi import FastAPI

app = FastAPI()

@app.post("/process")
def process_data(payload: dict):
    # Simulate a CPU-intensive or ML-like operation:
    time.sleep(2)  # mock long processing
    processed_result = {"original": payload, "processed": True}
    return processed_result
```