Lab: Breaking a Monolith – Microservices in Python

Learning Objectives

By the end of this lab, you will:

- Understand how to refactor a monolithic application into microservices.
- Develop, run, and test independent services in Python.
- Use Docker and Docker Compose to containerize and orchestrate services.
- Understand basic inter-service communication.

Part 0 – Preparation

Prerequisites:

- Python 3.8+
- Docker and Docker Compose installed
- Basic Flask knowledge

Part 1 – Explore the Monolith (Read-Only)

1. **Review the `monolith/app.py` file** provided by the instructor.
2. Identify how user registration, product management, and order placement are handled.
3. Answer:
    - Which parts of the app are logically separate?
    - What could go wrong if one-part crashes?

Part 2 – Build Microservices

You are given starter code for 3 services:

- `user_service` – handles registration and login
- `product_service` – handles product listing and creation
- `order_service` – handles placing orders and talks to `product_service`

Tasks:

1. **Read and understand** the `app.py` in each service folder.

2. **Build the Docker images**:
3. `docker-compose build`
4. **Run the services**:
5. `docker-compose up`
6. **Test functionality** using Postman or curl:
    - **Register user**:

    -
      ```
      curl -X POST http://localhost:5001/register -H "Content-Type:
      application/json" -d '{"username":"bob", "password":"123"}'
      ```
    - **Add a product**:

    -
      ```
      curl -X POST http://localhost:5002/products -H "Content-Type:
      application/json" -d '{"id": 1, "name":"Pen"}'
      ```
    - **Place an order**:

    -
      ```
      curl -X POST http://localhost:5003/order -H "Content-Type:
      application/json" -d '{"product_id": 1}'
      ```
7. Check logs for communication across services (e.g., order talking to product).