AMD201 (2024/25)	**Advancde Microservice Development and Deployment**	**Contribution: 100% of course**
Author: Ngo Tra (TraNLT2@fe.edu.vn)	**FastAPI Microservice**	**Deadline Date: 13/03/2025**

Build a Ride Sharing System with FastAPI

Overview

A ride-sharing system allows users to request rides and be matched with the nearest available riders. Similar to services like a ride-sharing platform, this system will handle user registration, rider management, ride booking, fare calculation, and ride status updates.

In this assignment, you will design and implement a backend-only microservices system using FastAPI. You will learn how to apply system design and API development principles to build a scalable, performant, and efficient ride-hailing application.

Scenario

You are a team of **software engineers** (3 - 4 **members**) working for a startup launching a new **ride sharing service**. Your task is to develop the **backend system and API** for the service.

The system should:

- Use the **C4 Model** to document the software architecture
- Allow **users** to register and book rides.
- Allow **riders** to register and accept ride requests.
- Assign the **nearest available rider** to a user based on a **fake distance matrix**.
- **Calculate ride fares** based on a tiered distance pricing model.
- Track **ride status** updates (Pending, In Progress, Completed, Canceled).
- Store all data in a **PostgreSQL database**.
- Addition points if you can add Kafka

Requirements

Your ride sharing system should include:

1. Software Architecture Design (C4 Model)

- Use the **C4 Model** to describe and document the architecture.
- Include at least the following **C4 diagrams**:

- **Context Diagram**: Overview of the system and its interactions with users and external components.
- **Container Diagram**: High-level components such as FastAPI services, database, and external systems.
- **Component Diagram**: Breakdown of microservices and their roles.
- **Code Diagram (Optional)**: A detailed view of class/module relationships within the system.

2. User & Rider Management

- **Users**: Store **name, phone number, password**.
- **Riders**: Store all **user details + rating + vehicle information (type, license plate)**.
- **Rider status**: Track availability (Available / Busy).

3. Finding the Closest Free Rider

- No **real GPS** integration; use a **predefined table** of **5 users mapped to 5 riders** with **random distances** between them.

	Rider_1	**Rider_2**	**Rider_3**	**Rider_4**	**Rider_5**
User_1	8 km	5 km	6 km	2 km	7 km
User_2	3 km	9 km	4 km	6 km	1 km
User_3	5 km	2 km	8 km	7 km	4 km
User_4	6 km	10 km	3 km	1 km	9 km
User_5	7 km	4 km	2 km	9 km	5 km

- Assign the **nearest available rider**. If multiple riders are available, **choose randomly**.

4. Fare Calculation

- Random distance 1 to 10 km
- Distance-based **pricing model**:
  - **1 km → 10k/km**
  - **2 - 4 km → 15k/km**
  - **More than 4 km → 12k/km**
- **No** integration with **Google Maps API** or real-world distance calculation.

5. Booking & Ride Status Management

- **Instant ride requests only** (no future bookings).
- Ride statuses: **Pending, In Progress, Completed, Canceled**.

6. Technology Stack & Features

- **Backend-only** application (no frontend required).

- **FastAPI** for API development.
- **PostgreSQL** as the database.
- **No payment integration** (use fake payments).

For this **Ride Sharing App coursework**, a **microservices-based architecture** would typically have the following core services:

1. User Service

- Manages user authentication and profile (CRUD for users).
- Handles login, registration, and password management.
- JWT-based authentication and role-based access control.

2. Rider Service

- Manages rider profiles, including availability status (Available/Busy).
- CRUD operations for riders.
- Stores vehicle details (type, license plate) and ratings.

3. Booking Service

- Manages ride requests and assigns the closest available rider.
- Stores ride history, ride statuses (Pending, In Progress, Completed, Canceled).
- Handles fare calculation based on distance.

4. Ride Matching Service

- Implements the logic for finding the nearest available rider.
- Uses a **predefined table of users mapped to riders with random distances**.
- Ensures fair distribution if multiple riders are closest.

5. API Gateway Service

- Central entry point for clients (users & riders).
- Manages request routing, load balancing, and authentication.

Deliverables

You will submit the following deliverables:

- **Group Presentation (70%):** You will present your system design and implementation to the class. You will explain the architecture, technologies, and trade-offs of your solution. You will also demonstrate the functionality of your web application. Your presentation should include **slides** and **source code (git link)**.
- **Individual Report (30%):** You will write a **self-evaluation report** of the project. You will reflect on your contribution, challenges, and learning outcomes. You will also provide feedback on your team members and the assignment. Your report should include **source code (git link with your branch)**.

Assessment

To pass this course, you must achieve **at least 5 on average.**
You will be assessed based on the following criteria:

Group Presentation

MARK	CRITERIAS
9 - 10	Exceptional presentation with a thorough understanding, highly organized structure, and seamless transitions. Technical depth is exceptional, showcasing mastery. The demonstration of functionality is flawless, showcasing all aspects effectively. The source code exhibits excellent practices, being clean, well-documented, and following best practices.
7 – 8.5	The presentation reflects a solid understanding with clear organization and logical flow. Technical depth is evident with clear explanations, and the demonstration of functionality is successful with minor issues. The source code adheres to coding standards with few minor bugs.
5 – 6.5	The presentation demonstrates a basic understanding but with significant gaps. The structure is somewhat organized, but clarity issues persist. Technical details are basic, with inconsistencies. The demonstration of functionality is limited, and the source code shows adherence to basic standards but with some issues.
< 5	The presentation lacks understanding, coherence, and technical depth. Content is superficial, structure is disorganized, and technical explanations are flawed. The demonstration of functionality is ineffective, and the source code quality is poor.

Individual Report

MARK	CRITERIAS
9 - 10	An exceptional report with a thorough and insightful reflection on personal contribution, challenges, and learning outcomes. Feedback on team members and the assignment is detailed, constructive, and specific, displaying a high level of self-awareness and critical evaluation.
7 – 8.5	The report reflects a solid understanding with a reasonable reflection on personal contribution, challenges, and learning outcomes. Feedback on team members and the assignment is constructive and provides some specificity.
5 – 6.5	The report shows a basic understanding but with significant gaps. Reflection on personal contribution, challenges, and learning outcomes is limited and lacks depth. Feedback on team members and the assignment is basic and lacks specificity.
< 5	The individual report lacks depth and understanding. Reflection on personal contribution, challenges, and learning outcomes is superficial. Limited or no feedback is provided on team members and the assignment.