PROJECT (CO3103)

Lecturer: **Nguyễn Minh Tâm**

Email: **tam.nguyen272@hcmut.edu.vn**

The project for this course consists of five tasks.

Task 1: Requirement elicitation

Student needs to perform the following tasks:

- Task 1.1. Identify the motivation of your project. What is expected to be done? What is the scope of the project?
- Task 1.2. Perform general analysis of related works. Describe all functional and non-functional requirements of the desired system. Draw a use-case diagram for the whole system.
- Task 1.3. Choose one specific main feature. Draw its use-case diagram and describe the use-case using a table format.

Task 2: System modeling

Student needs to perform the following tasks:

- Task 2.1. Draw an activity diagram to capture major (not all) functional requirements of the desired system.
- Task 2.2. Draw a sequence diagram for use-case in Task 1.3.
- Task 2.3. Draw a class diagram.

Task 3: Architectural design

Student needs to perform the following tasks:

- Task 3.1. Design the database for your system, including ER diagram, and relational schema.
- Task 3.2. Describe an architectural approach you will use to implement the desired system.

Task 4: Implementation - Sprint 1

Student needs to perform the following tasks:

- Task 4.1. Implement a Minimum Viable Product (MVP) and demonstrate the result. MVP means that you do the least to be able to demonstrate.

Task 5: Implementation - Sprint 2

Student needs to perform the following tasks:

- Task 5.1. Complete the report and demonstrate the whole project.

ADVANCED SECTION (OPTIONAL)

6a. Cloud Deployment and Services: In this task, students will be required to deploy their software system to a cloud platform such as AWS, Google Cloud, or Microsoft Azure.

- **Cloud Platform Setup**: Choose one of the leading cloud providers (AWS, Google Cloud, or Azure) for deployment, ensure that the cloud environment supports the scalability needs of the system.
- **Database Management**: Deploy the database (MySQL, PostgreSQL, MongoDB, etc.) on the cloud platform, back up the database regularly using cloud-native backup tools, and test restoring from a backup.
- **Continuous Deployment**: Implement a CI/CD pipeline that automatically deploys the latest changes from the code repository to the cloud by using tools such as Jenkins, Github Actions, or Travis CI, demonstrate continuous delivery by automatically pushing updates to the staging or production environment when new code is committed.

6b. AI and Machine Learning Integration: In this task, students will be required to include a module where they integrate a simple machine learning model (e.g., predictive analytics or recommendation system, chatbot or NLP system (similar to LangChain)

- **Integrating a Machine Learning/Deep Learning Model**: Implement a basic machine learning model such as predictive model or recommendation engine, within the software system or integrate existing model like conversational AI chatbot.
- **Model Deployment**: The deployed model should be integrated into the software's architecture (e.g., as a RESTful service, microservice, or module within a monolithic system).
- **Model Training and Retraining**: allow the system to handle model retraining as new data comes in (e.g., through scheduled retraining or real-time learning), also ensure the system can handle training on large datasets, possibly requiring

distributed computing or leveraging cloud platforms like AWS SageMaker, Google AI, or Microsoft Azure ML.

6c. API Integration: In this task, students will be tasked with developing and integrating APIs into their software system. There are many real-world use cases for this task such as payments, social logins, and data sharing between microservices. The task also emphasizes the importance of proper documentation and testing of APIs.

- **Multi-service Collaboration via APIs**: Design and implement APIs to allow multiple services to communicate and collaborate seamlessly.
  Examples:
  + ***Payment Gateway Integration***: Allow users to make secure payments through popular platforms such as PayPal, Stripe, or other gateways.
  + ***Social Login***: Integrate social media authentication systems (e.g. Google, Facebook, Github) using OAuth or OpenID Connect for user login.
  + ***Internal Microservices***: For instance, one microservice may handle user authentication, another may manage payments, and another could handle notifications or messaging. These microservices must communicate effectively through APIs.
- **RESTful vs. GraphQL API Design**: Choose between RESTful and GraphQL API design, depending on your system's needs.
  + ***RESTful API***:
    - Implement the key principles of REST: statelessness, client-server architecture, uniform interface, and resource-based URLs.
    - Design CRUD (Create, Read, Update, Delete) operations, using appropriate HTTP methods (GET, POST, PUT, DELETE).
    - Example: A payments microservice could expose endpoints like /api/v1/payments, allowing users to initiate, track, or cancel payments.
  + ***GraphQL API***:

    - Create a schema with queries and mutations for managing data interactions.
    - Example: In a social login system, students could create a GraphQL query for fetching user details or a mutation for user authentication.

- **Testing and Validation**: Ensure that the APIs are rigorously tested and validated before deployment.
  - Use Postman or Swagger to validate that the API endpoints are working as expected. Postman allows students to create test suites to verify request/response pairs and simulate different user behaviors.
  - Ensure that both unit tests (for individual services) and integration tests (for cross-service communication) are written.
  - Testing should cover typical use cases, edge cases, and failure scenarios. Students should handle errors and response codes appropriately (e.g., 400 Bad Request, 401 Unauthorized, 500 Internal Server Error).

DEADLINE: **23:55 - 13/12/2024**

Note: Students need to submit the report (PDF), which must contain a link to the system demonstration.