Department of Computer Engineering

University of Peradeniya

Workshop 2: Web oriented Architecture

- Lab Activity: Extending Web Applications with Event Driven Architecture (EDA)
- Complete this lab with the group you formed during the last workshop session.
- Due: 27th January 2026,before 3.30 p.m.

1.Objectives:

Traditional REST based systems often struggle with scalability and resilience as they grow. This lab focuses on the architectural evolution of a synchronous system into an asynchronous one using an **Event Bus**.

1.1 ILO's:

- Transform synchronous API flows into asynchronous event driven workflows.
- Apply principles of Loose Coupling and Resilience.
- Design and implement Producer Consumer patterns.
- Evaluate architectural trade-offs in modern web systems.

2. System Architecture

- Students will transition from a "Request Response" model to a "Publish Subscribe" model.

2.1 Existing Architecture (Synchronous)

- The current system relies on direct, blocking calls between the UI, Backend, and Database

2.2 Target Architecture (Asynchronous)

- The evolved system introduces an **Event Bus** to decouple the primary API from side effects (consumers)

Before starting, ensure you have the following:

- A working web application (e.g., Hospital Management) with CRUD REST APIs.
- A basic frontend connected to these APIs.

- Knowledge of HTTP and backend development.

3. Lab Tasks

- **Task 1:** Identify a Domain Event

  - **Example:**Let's assume your application is a Hospital Management System.
  - Select one existing API operation (e.g., Create Patient, Schedule Appointment, or Generate Bill). Identify a meaningful Domain Event that occurs after this action, such as PatientRegistered or InvoiceGenerated.

You need to do this task for your own application.

- **Task 2:** Implement the Event Producer

  - Modify your API so that:

1. The database operation completes successfully.
2. A domain event is published to the Event Bus after the DB commit.
3. The API returns a response immediately to the user without waiting for downstream processing.

  - Event Payload Requirement: Include the Event Name, Entity ID, Timestamp, and relevant metadata

- **Task 3:** Set Up the Event Bus

  - Integrate a message broker such as RabbitMQ, Kafka, or a simulated in memory event emitter. The goal is architectural behavior, not deployment complexity.

- **Task 4:** Implement Consumer Service(s)
  - Develop at least one independent service (e.g., Notification Service or Analytics Service) that subscribes to your event.

- Constraint: The consumer must run as a separate process and must not block the main API if it fails.

- **You must verify the following if it is correct:**

- **Decoupling:** Stop the consumer service and show that the main API still functions perfectly.
- **Asynchronicity:** Show via logs that the API response is sent before the consumer finishes processing.

4. Evaluation Criteria

To successfully complete this lab, your work will be assessed based on the following criteria:

- You must select an appropriate domain event that represents a meaningful occurrence in your system.
- The event must be published correctly and only after the API execution is successful.
- You must demonstrate proper asynchronous communication through the use of a message broker or simulated bus.
- You must implement at least one functional, independent consumer service that processes the event.
- You must prove that the producer is not blocked by the consumer and that the API remains functional even if the consumer is stopped.
- You must provide a clear architecture diagram and a technical justification for your design choices

5. Submission

For the final submission, students are required to provide a comprehensive package that includes the full source code for both the producer and consumer services. Along with the code, you must include evidence of functionality in the form of screenshots or logs that clearly demonstrate the asynchronous flow of events between components. Additionally, you must submit a clear architecture diagram accompanied by a short written explanation (150–200 words) that describes your chosen event, the roles of the producer and consumer, and the specific benefits of using an Event-Driven Architecture in your application.