**Expense Tracker**

**Course** : J2EE Business Components - ITE-5432-0NA

**Professor** : Vitalii Bohudskyi

**Semester :** 4th Semester

**Project Title** : Expense Tracker - Phase III

**Team members:** Urja Tej Bhemat N01648980

Sameeha Shaik N01649346

Kalwala Siddhartha Reddy N01649317

Kledy Danlees N01652920

**Document for Phase III: Deployment and Finalization**

This document details the final deployment of the "Expense Tracker" application and presents the results of the performance and API testing. The project has been successfully containerized and deployed using :

**Podman**, a container management tool.

**1. Deployment Architecture**

The deployment strategy for the "Expense Tracker" is based on containerization to ensure consistency and portability across various environments.

* **Containerization**: The application and its relational database are deployed as separate, isolated containers. A **Dockerfile** is used to automate the creation of the application's container.
* **Java 17+** runtime, all necessary project dependencies, and the packaged application itself. The database (either **MySQL** or **PostgreSQL**) runs in its own dedicated container, simplifying management and upgrades.
* **Configuration Management**: The application utilizes **Spring Config Externalization** to manage environment-specific variables, such as database credentials and port numbers. This approach keeps sensitive configuration data separate from the application's code, allowing for flexible deployment without the need to modify the application's source code.
* **Podman Deployment**: **Podman** is used to orchestrate and run both the application and database containers. This ensures that the two containers can communicate securely and function as a complete, integrated system. The deployment process involves using Podman to build and run the containers, effectively bringing the entire application stack online.

**2. Testing and Final Report**

The final phase includes comprehensive testing and a summary of the project's development lifecycle.

* **API Testing Results**: The RESTful APIs were thoroughly tested on the deployed environment. Using **Postman**, all API endpoints were validated for functionality, security, and data integrity. The results confirm that all functionalities—including user authentication and CRUD operations—are working as expected. The API was designed following an API-first design approach using the

**OpenAPI Specification**.

* **Performance Testing Results**:

Performance tests were conducted on the deployed application to evaluate its stability and responsiveness. The primary tools used for this were Postman for API testing and WebFlux for testing reactive components.

For key API calls, such as user login, adding an expense, and retrieving a list of expenses, the average response time was measured. The application demonstrated its ability to handle multiple concurrent users without significant degradation in performance.

During testing, no major bottlenecks were observed in the core functionalities. However, areas identified for potential optimization in future development include database query performance and fine-tuning the reactive components.

* **Final Project Report**:
  + **Summary**: The "Expense Tracker" project successfully delivered a fully functional MVP. The application meets all core requirements and is a stable, secure platform for multi-currency expense management.
  + **Challenges and Solutions**:

Significant technical challenges arose during the development and deployment of the "Expense Tracker" application, particularly with container orchestration, dependency management, and security configuration.

* **Container Orchestration**: One key challenge was ensuring seamless communication between the application and database containers using **Podman**. Initially, the containers were unable to connect, leading to database connection errors. This was resolved by creating a Podman network and running both containers within it, allowing them to communicate securely using their service names instead of IP addresses.
* **Dependency Management**: Managing the various **Spring Framework** dependencies, especially with different versions of **Spring Boot**, **Spring Security**, and **Spring Data JPA**, proved to be a hurdle. Version conflicts often caused the application to fail at startup. The issue was addressed by carefully reviewing the dependency tree and standardizing versions using a single, consistent parent pom.xml file, which is a common practice in Spring projects.
* **Security Configuration**: Implementing **Spring Security** for user authentication and authorization presented a challenge, specifically when configuring role-based access control and protecting API endpoints. The initial configuration was too restrictive, blocking legitimate user access. The solution involved fine-tuning the security configuration to use JWT (JSON Web Tokens) for session management and creating a custom security filter chain to correctly authorize users based on their roles while allowing public access to login and registration endpoints.
  + **Future Enhancements**: Some potential next steps for the project. These could include implementing advanced features like **microservices** with Spring Boot, integrating an event-driven architecture with **Kafka**, or designing reactive applications with **Spring WebFlux**.