1. Introduction

This document provides a comprehensive explanation of the technologies, concepts, and configuration files used in the development of a microservices-based E-Learning Platform. The goal is to make every term and tool clear to a beginner or intermediate developer working in Java and Spring Boot.

2. Technologies Used

- Java 17: Modern object-oriented programming language used to develop backend services.
- Spring Boot: Framework to build stand-alone, production-grade Spring-based applications.
- Spring Cloud: Provides tools for developers to quickly build some of the common patterns in distributed systems (e.g., configuration management, service discovery, circuit breakers, routing).
- Eureka Server: Service discovery tool by Netflix used for locating services.
- Spring Cloud Gateway: Handles routing and filtering of requests at the edge of your system.
- MySQL: Relational database used to persist user data.
- JPA (Java Persistence API): Abstraction over ORM (Hibernate) used for database interaction.
- Maven: Dependency management and build tool.
- Lombok: Reduces boilerplate code (e.g., getter/setter) via annotations.
- Postman: Tool used for testing APIs.
- GitHub: Version control and project hosting platform.

3. Key Concepts

- Microservices: Architecture style that structures an application as a collection of small, independent services.
- Dependency: A library or module required by your project to compile or run.
- Bean: An object managed by the Spring IoC container.

- Autowiring: Automatically injecting dependencies (beans) into other beans.

- Port: Logical endpoint through which the application communicates (e.g., 8080, 8761).

- Service Registration & Discovery: Allows services to register themselves and discover other services (via Eureka).

4. pom.xml

This is the Maven configuration file. It contains:

- Dependencies: Third-party libraries the application needs.

- Plugins: Additional tools Maven should use (e.g., compiler).

- DependencyManagement: Allows centralized dependency version control.

- Parent: Inherits settings from Spring Boot starter-parent.

5. application.properties / application.yml

This file contains configuration details such as:

- server.port: Defines on which port the service will run.

- spring.datasource: Sets up DB connection (URL, username, password).

- spring.application.name: Used for registering with Eureka.

- eureka.client.service-url.defaultZone: Eureka Server address.

- logging.level: Log verbosity levels.

6. Service Structure

Each microservice (e.g., user-service, api-gateway) generally has the following structure:

- controller/: Handles HTTP requests.

- service/: Business logic layer.

- repository/: Database access layer.

- model/: Entity classes (data structures).

- config/: Any custom configuration classes.

7. Project Flow

- Start Eureka Server on port 8761.
- Start API Gateway on port 8080.
- Start User Service on port 8081.
- Use API Gateway URL (e.g., http://localhost:8080/user-service/auth/register) to interact with user service.
- Register and authenticate users using Postman.
- All services register with Eureka for discovery.

8. GitHub Deployment

- `git init`: Initialize a Git repo.
- `git add .`: Add all files.
- `git commit -m "message"`: Save your changes.
- `git remote add origin <URL>`: Link local repo to GitHub.
- `git push -u origin main`: Push changes.

9. Final Thoughts

This architecture allows scaling each microservice independently, improves modularity, and makes the application easier to manage in the long term. Spring Boot and Spring Cloud simplify microservice creation and integration.