FinTech Lab 3 Report

Name: Krishang Jain
R. No.: 73
Reg. No.: 240958280

Overview

In these lab sessions, we were introduced to the fundamentals of backend application development using Spring Boot, focusing on MVC architecture, REST APIs, layered design, and security concepts relevant to fintech systems. The lectures emphasized industry-standard practices used to build scalable, maintainable, and secure backend applications.

MVC and Layered Architecture in Spring Boot

The lecture began with an overview of the MVC (Model–View–Controller) pattern and how it is implemented in Spring Boot using a layered architecture. This architecture improves code readability, maintainability, and team collaboration.

The commonly used layers are:

- Controller Layer
- Service Layer
- Service Implementation Layer
- Repository / DAO Layer
- Database

This separation of concerns ensures that each layer has a specific responsibility and prevents tight coupling between components.

Controller Layer

The controller acts as the entry point of the application.

Handles incoming HTTP requests from clients (browser, frontend apps, Postman).

Defines API endpoints using annotations such as @GetMapping, @PostMapping, etc.

Delegates processing to the service layer.

Contains no business logic.

Service and Service Implementation Layer

The service layer is defined as an interface, while the service implementation layer contains the actual business logic.

Handles validations and core application logic.

Acts as the “brain” of the application.

Communicates with the repository layer for database operations.

Keeping logic here makes applications easier to scale and maintain.

Repository / DAO Layer

The repository (DAO – Data Access Object) layer:

Interacts directly with the database.

Performs CRUD operations (Create, Read, Update, Delete).

Uses entity classes to map Java objects to database tables.

Does not contain business logic.

Spring Boot Annotations

Several important annotations were discussed:

@Autowired – Enables dependency injection, allowing Spring to create and manage objects automatically at runtime.

@Entity – Marks a class as a database entity representing a table.

@Table(name = "...") – Specifies the table name associated with an entity.

@GetMapping, @PostMapping, etc. – Maps HTTP requests to controller methods.

@SpringBootApplication – Marks the main class and enables auto-configuration, component scanning, and Spring context initialization.

Entity, and DTO

Entity:
Represents the database schema where class attributes map to table columns.

DTO (Data Transfer Object):
Used to transfer data between layers. DTOs do not contain core logic but may include lightweight validation such as checking mobile numbers or required fields.

REST API Fundamentals

Modern applications communicate through REST APIs (Representational State Transfer).

Resources and URIs

A resource represents meaningful data such as:

Customer

Account

Transaction

Product

URIs identify these resources:

GET /customer – Retrieve customers

POST /customer – Create a customer

HTTP Methods

REST APIs use HTTP methods to perform actions:

GET – Retrieve data

POST – Create new data

PUT – Update existing data completely

PATCH – Partial update

DELETE – Remove data

These methods form the basis of CRUD operations.

Statelessness in REST APIs

REST APIs are stateless, meaning:

Each request contains all required information.

The server does not store client session data.

This improves scalability and performance, which is crucial for fintech systems handling millions of users.

HTTP Status Codes

Proper status codes help clients understand responses:

200 OK – Successful request

201 Created – Resource created

204 No Content – Successful update

400 Bad Request – Client error

401 Unauthorized – Authentication failed

403 Forbidden – Permission denied

404 Not Found – Resource missing

500 Internal Server Error – Server failure

503 Service Unavailable – Temporary outage

Filtering, Sorting, and Pagination

To handle large datasets efficiently:

Filtering:
GET /customer?name=John

Pagination:
GET /customer?page=2&limit=10

Sorting:
GET /customer?sort_by=name&order=asc

These techniques improve performance and reduce server load.

Security and JWT

Security is critical in fintech applications.

JWT (JSON Web Token)

JWT is used for authentication without storing session data on the server.

Consists of Header, Payload, and Signature.

Header defines token type and encryption algorithm.

Payload stores user information.

Signature ensures integrity using a secret key.
JWTs can be decoded easily if intercepted, so HTTPS and proper encryption are mandatory.

Dependency Injection and Bean Scopes

Spring Boot supports different bean scopes:

Singleton – One object per application (default)

Request

Session

Application

Dependency Injection connects all layers using @Autowired.

API Design Order

Recommended order for building APIs:

Entity

Repository

Controller

Service

Service Interface

Spring Boot Request Flow

Client → Controller → Service → Repository → Database

Each layer performs a specific role, ensuring clean architecture and scalability.

Documentation, Testing, and Error Handling

APIs should be documented using tools like Postman or Swagger.

Proper error handling and meaningful messages improve debugging.

Thorough testing ensures reliability and correctness.