5/5/25

Week 2

Introduction To Spring Boot Web Mvc, Mvc Architecture

REST APIs:-

→ REST (Representational State Transfer) APIs (Application Programming Interfaces) are a set of rules and Conventions for building and interacting with Web services.

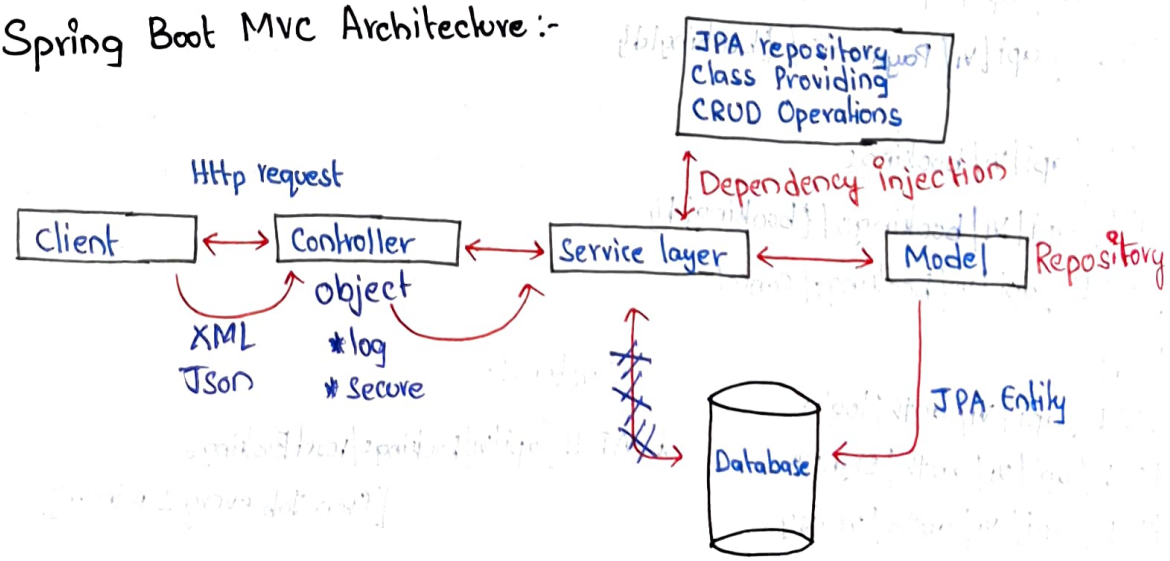
* GET /users :- Retrieve a list of all users.
* GET /users/{id} :- Retrieve a specific user by ID.
* POST /users :- Create a new user.
* PUT /users/{id} :- Update an existing user by ID
* PATCH/users/{id} :- Partially update an existing user by ID.
* DELETE /users/{id} :- Delete a User by ID.

Spring-boot-starter-web:-

→ Spring-boot-starter-web Contains the following dependencies:-

* Spring-boot-starter
* Jackson. → Convert data like Json (or) XML to Java objects
* Spring-Core
* Spring-mvc
* Spring-boot-starter-tomcat

Spring Boot Mvc Architecture :-



1. Spring Boot MVC is a part of Spring Boot used to build web Applications using the Model - View - Controller (MVC) Pattern.
2. Helps Separate business logic (Model), UI (View), and User input handling (Controller).
3. Spring Boot MVC uses Dependency Injection to loosely Connect Components.
4. Makes each class independent and easily changeable without affecting others.
5. Model → Holds data and business logic.
   View → Defines how data is displayed (like HTML Pages).
   Controller → Handles requests and Connects Model with view.
6. This clear Separation makes code Cleaner and easier to Manage.
7. Since Components are independent, they can be reused in other Projects.
8. You can test individual parts (like Controllers (or) Services) without loading the whole app.
9. As your Project grows, you can easily add new features without breaking the old ones.
10. Easier to Maintain large projects because of the Clear Structure.

Layered Architecture

① Presentation Layer → * Authentication, * JSON Translation → Controllers

↕

② Business Layer → * Business Logic, * Validation, * Authorisation → Service Classes

↕

③ Persistence Layer → * Storage Logic → JPA (or) JDBC (or) H2 (JPQL, Hibernate → SQL)

↕ CRUD Operations are Performed.

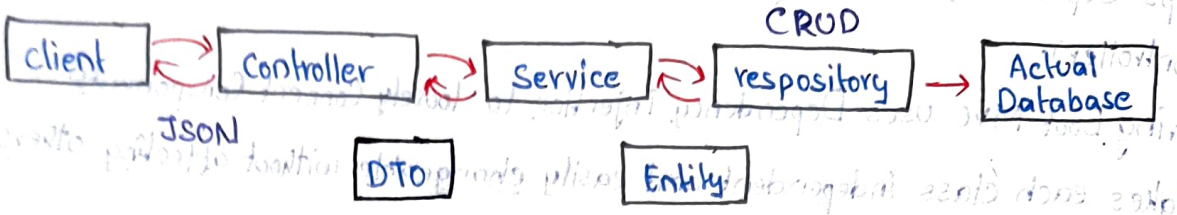
④ Database → * Actual Database → Actual database (SQL)

*Presentation Layer:-
→ it handles the HTTP requests, translates the JSON Parameter to object, and Authenticates the request and transfer it to the business layer.

*Business Layer:-
→ This layer handles all the businesslogic. it Consists of service classes and uses services Provided by data Access layer. it also Performs Authorization and validation

*Persistance layer:-
→ This layer contains all the storage logic and Translates business objects from and to database rows.

Spring Boot Web Project Structure



1. Client → Controller (HTTP Request with JSON)

The client (like a web browser (or) Postman) sends an HTTP request to the Controller in JSON format.

2. Controller → Service (DTO Transfer)

The Controller receives the request, converts it into a DTO (DataTransferObject) and Passes it to service layer for Processing.

3. Service → Repository (Entity ← Business Logic)

The service layer contains business logic and works with Entity objects. it Calls the Repository to interact with the database.

4. Repository → Database (CRUD Operations)

The repository layer Performs CRUD operations directly on the actual database using Entity Classes.

5. Response Flow (Back to client)

The Processed data flows back in reverse – from repository → Service → Controller → client, Usually as a JSON response.

6/5/25

The Presentation Layer, DTO and Controllers:-

Annotated Controllers:-

→ Spring MVC Provides an annotation-based Programming model where @Controller and @RestController Components use annotations to express request mappings, request input, exception handling, and move.

→ The @RestController annotation is a shorthand for @Controller and @ResponseBody, Meaning all methods in the Controller will return JSON/XML directly to the response body.

@Rest Controller Vs @Controller:-

1. @Controller Annotation is used to represent classes as Spring MVC Framework to implement server side logic [Business logic].
2. @Rest Controller is a Specialized version of @Controller, it Contains 2 Annotation
   i) @Controller
   ii) @ResponseBody
   → it will Convert the response in plaintext / JSON / XML

Dis Advantages:-

1. @RestController doesn't work with view Technology (JSP/JSTL) that Means methods Can't return ModelAndView object.
2. This Annotation will be used at Class level, whereas @Controller will return ModelAndView object but if you want to display the response directly on the browser as a Message, we have to use @ResponseBody Annotation at either Class level (or) Method level.

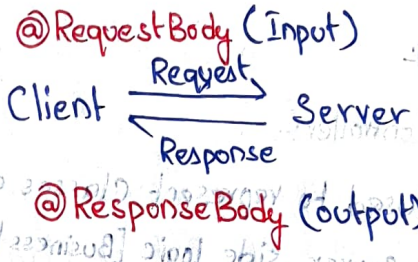
@RequestBody Vs @ResponseBody.

1. These Annotations are used to find the HttpRequest (or) HttpResponse as a Method Parameter (or) Method return value Respectively.

2. @RequestBody will Find the HttpRequest body to the Method Parameter

Eg:- @RequestBody Employee emp → Parameter JSON.

3. @ResponseBody will return the HttpResponse body to the browser.



Request Mappings:-

→ You can use the @RequestMapping Annotation to map requests to Controllers methods.

→ It has various attributes to Match by URL, HTTP method, request Parameters, headers, and media types.

→ There are also HTTP method Specific shortcut variants of @RequestMapping.

* @GetMapping → Path, Params
* @PostMapping
* @PutMapping
* @DeleteMapping
* @PatchMapping

Dynamic URLs Paths

@PathVariable

/employees/123

→ it gets the value directly from the URL Path

→ Use path variables when the Parameter is an essential Part of the URL Path that identifies a resource.

→ Value Comes from the Path → /user/{id}

@RequestParam

/employees?id=123

→ it gets the value from the URL after ?

→ Use query Parameters when the Parameter is optional and used for filtering, sorting, (or) other Modifications to the request.

→ where we Open the Amazon website

→ value Comes from the query → ?key=value

RequestBody:-

1. @RequestBody is used to bind the HTTP request body to a Java object.
2. When a Client sends data in the body of a request (e.g., JSON or XML), @RequestBody maps this data to a Java objects.

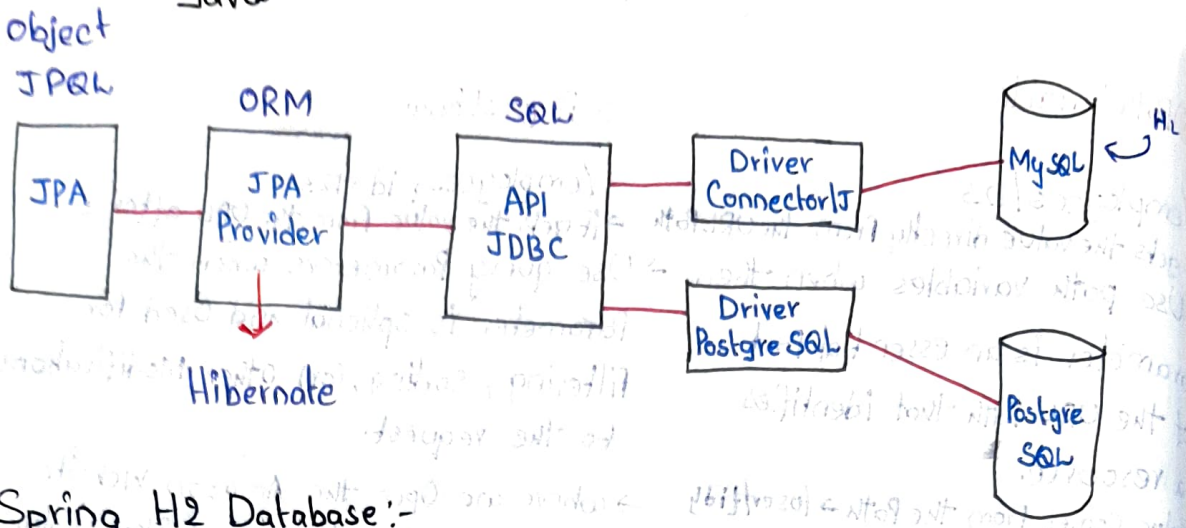
7/5/25

Persistence Layer & JPA:-

Persistence Layer:-

1. it's the part of your Application that handles Saving, Updating, deleting and fetching data from the database.
2. it works directly with the database Using JPA (or) JDBC.
3. it works with Entity Classes that represent database tables.
   Ex:- a User Java class maps to a User table in the DB.
4. Spring Data JPA Provides JpaRepository (or) CrudRepository interfaces.
5. You just define a repository interface (e.g., UserRepository), and Spring gives you built-in CRUD methods (like Save(), findById()).
6. You don't write SQL manually.
7. JPA (with Hibernate) Converts your Java Code into SQL beind the Scenes.
8. You Can Switch from H2 → MySQL → PostgreSQL just by changing a few settings - The Persistence Layer Code remains the Same.

Java Persistence API - JPA



Spring H2 Database :-

→ Spring Boot makes it very easy to set up an in-memory H2 database for development and testing Purposes.

→ H2 is a lightweight, fast and Open-source relational database engine that can run in both in-memory and Persistent modes.

Dependency → <groupId> Com.h2database </groupId>

@Entity Annotation :-

1. The @Entity annotation in Spring and Java Persistence API (JPA) is used to mark a class as a Persistent entity, Meaning it represents a table in a relational database.
2. This is a fundamental part of the ORM (Object-Relational Mapping) Paradigm, where Java Objects are mapped to database tables.

Key Points of @Entity :-

* Class-Level Annotation.
* Primary key.
* Automatic Table Mapping.

@Entity is a Java class ~~which~~ That you need to Convert into a Table in Database.

@Table → To mention the Table name.

@Id → it is used to indicate one of the attributes in the class as Primary key Column (Primary key = Unique + not null)
→ it is a Field Level Annotation.

@Column → it is used to provide additional information like Column name, length, nullable (true/false), Unique (true/false), Precision and Scale.

Note:-

1. if you don't mention the Table name then the table name will be Same as Class name.
2. if you don't mention the Column name then the Column name will be Same as attribute/Property name.
3. decimal(P,S) where P means Precision and S means Scale.
   Example:- decimal(4,2)
   → 123.45 (P means 123 and S means 45)
4. By default size of integer attribute in the table is number(38).
5. if you make any Column as not null then nullable = false and if you make any Column as Unique then Unique = true.

JPARepository Interface

→ The JPARepository interface in Spring Data JPA provides a set of CRUD (Create, Read, Update, Delete) Operations and query methods for interacting with the database.

⇒ Key points of CrudRepository:-

* Generic Interface. ✓
* Predefined Methods. ✓
* Custom Queries. ✓

lombok → which eliminates getter and setters in POJO class.

Ex:- @Getter, @Setter, @AllArgsConstructor, @NoArgsConstructor

8/5/25

Service Layer:-

1. Acts as a Middleman:-
   → it sits between the Controller (Presentation layer) and the Repository (data access layer) to Separate Concerns and manage f
2. Handles Business Logic:-
   → it contains all the Core business rules and Calculations, keeping them Separate from the Controller and database logic.
3. Coordinates Components:-
   → it Manages interactions b/w different services, repositories and external APIs to fulfill a request.
4. Provides Clean interfaces:-
   → it offers a Simple, Consistent API for the Controller (or) other Services to use, hiding internal Complexities.
5. Improves Maintainability:-
   → By Separating logic, it makes the Application easier to debug, test, scale and update.

→ The Service layer Acts as a bridge b/w the Persistence layer (responsible for data access) and the Presentation layer (handling user interaction).

→ It encapsulates the business logic of the application, Orchestrates interactions b/w different Components, and Provides a clean interface for external clients to interact with the System.

→ By abstracting away the Complexities of data access and business Operations, the Service layer Promotes Modularity, Maintainability, and Scalability.

1. Controller Layer (Entry Point):-

→ The Controller handles incoming API requests (like GET/POST).
→ it doesn't directly interact with the database.
→ it sends the request to the Service Layer.

2. Service Layer (Business Logic):-

→ This layer Contains the main business logic.
→ it acts as a bridge b/w Controller and Repository.
→ Ensures clean Separation of Concerns and code reuse (DRY principle).

3. Repository Layer (Database Access):-

→ Handles all database Operations like CRUD.
→ Only the Service Layer should talk to it, not the Controller directly.

4. No Direct Controller ↔ Repository/DTO Communication

→ Direct Communication from Controller To Repository (or) Controller to DTO breaks layered Architecture.
→ That's why we want Centralized logic in Service Layer.

5. Benefits :- DRY, Logging, Security

→ Keeping logic in one layer avoids duplication
→ Easier to apply logging, Security, Validation Centrally in service Layer.

ModelMapper:-

1. ModelMapper is a Java library used to Automatically map objects from One type to another.
2. Mostly used to Convert b/w Entity Classes and DTOs (Data Transfer Objects).
3. Commonly used in REST APIs to Avoid exposing internal database entities directly.