Spring MVC Learning Roadmap

1. Spring MVC Basics

- Overview of Spring MVC and its role in web applications.
- Understanding the Model-View-Controller (MVC) pattern and how Spring MVC implements it.
- Role of the DispatcherServlet as the front controller that handles HTTP requests and delegates them to appropriate handlers.
- Separation of concerns between Controllers, Models, and Views in Spring MVC.
- Basic structure of a Spring MVC web application (web.xml or Spring Boot).

2. Setting Up Spring MVC Development Environment

- Installing and configuring Spring MVC with Maven/Gradle.
- Setting up an IDE (Eclipse/IntelliJ) for Spring MVC development.
- Using Spring Initializr for generating Spring Boot projects for web applications.
- Configuring Spring MVC with web.xml (or equivalent in Spring Boot) for web application setup.

3. Controllers & Request Mapping

- Understanding the role of `@Controller` and `@RestController` annotations.
- Mapping HTTP requests to handler methods using `@RequestMapping`, `@GetMapping`, `@PostMapping`, `@PutMapping`, `@DeleteMapping`.
- Handling request parameters with `@RequestParam` and path variables with `@PathVariable`.
- Using request method-specific annotations like `@GetMapping`, `@PostMapping`, etc., for better readability.

4. Model and View

- Binding data between views and controllers using `@ModelAttribute` to bind form fields to model objects.
- Understanding View Resolvers (e.g., `InternalResourceViewResolver` for JSP, `ThymeleafViewResolver` for Thymeleaf).

- Passing model data to views (Thymeleaf or JSP).
- Using Spring Form Tags (`<form:form>`, `<form:input>`, `<form:select>`) to bind form data to model objects in views.

5. Form Handling and Validation

- Creating forms using `<form:form>` and binding form fields to Java objects.
- Server-side validation using **JSR-303 annotations** like `@NotNull`, `@Size`, `@Email`, etc., on model objects.
- Handling validation errors with `BindingResult` and displaying error messages in views (Thymeleaf or JSP).
- Integrating Hibernate Validator for advanced validation techniques (optional but useful).

6. Exception Handling

- Handling exceptions globally using `@ControllerAdvice` for centralized exception management.
- Using `@ExceptionHandler` to handle specific exceptions within controllers.
- Configuring custom error pages for HTTP errors like 404 (Page Not Found) and 500 (Server Error).
- Returning user-friendly error pages based on different error conditions.

7. RESTful Services with Spring MVC

- Building REST APIs with `@RestController` for exposing HTTP endpoints.
- Handling CRUD operations (Create, Read, Update, Delete) using HTTP methods (GET, POST, PUT, DELETE).
- Returning JSON or XML data using `@ResponseBody`.
- Content negotiation to support multiple content types (JSON, XML) depending on client request (`Accept` header).

8. Spring Security (Basic Integration)

- Setting up basic authentication using Spring Security.
- Implementing role-based authorization with Spring Security to restrict access to specific URLs or methods (`@PreAuthorize`, `@Secured`).
- Enabling and managing CSRF Protection that is enabled by default in Spring Security to prevent Cross-Site Request Forgery attacks.

9. View Technologies

- Using Thymeleaf for rendering dynamic HTML views in Spring MVC.
- Thymeleaf syntax basics like `th:text`, `th:each`, `th:if` for dynamic content rendering.
- Integrating Spring Form Tags with Thymeleaf for form handling and data binding in views.

10. Testing Spring MVC Applications

- Unit Testing Controllers using **MockMvc** for simulating HTTP requests and validating responses (status codes, views, JSON data).
- Integration Testing using `@SpringBootTest` to load the full application context and test end-to-end functionality.
- Testing custom exception handlers and global error handling to ensure proper error response behavior.

11. Best Practices for Spring MVC Development

- Layered architecture: Separating concerns between Controller, Service, and Repository layers for maintainability.
- Using DTOs (Data Transfer Objects) to decouple web layer from business logic and database entities.
- Avoiding business logic in views (Thymeleaf/JSP) to maintain a clean separation of concerns.
- Centralized exception handling using `@ControllerAdvice` to keep controllers clean and focused on business logic.

Summary of Core Topics to Focus On:

1. **Spring MVC Architecture**: DispatcherServlet, HandlerMapping, MVC design pattern.
2. **Controllers and Request Mappings**: `@Controller`, `@RestController`, `@RequestMapping`, and handling HTTP methods.
3. **Form Handling & Validation**: Binding forms to model objects, JSR-303 annotations for validation.
4. **RESTful Services**: Exposing REST APIs with Spring MVC.

5. **Spring Security**: Basic authentication, role-based authorization, and CSRF protection.
6. **Testing**: Unit and integration testing using **MockMvc** and **@SpringBootTest**.
7. **Best Practices**: Layered architecture, DTOs, and centralized exception handling.

Full-Stack Java Development

1. **Frontend Integration (Optional for now):**
    - **Basics of Frontend**: You can learn basic HTML, CSS, and JavaScript (or a JS framework like React, Vue, or Angular if you're interested in the full-stack path).
    - Integrating JavaScript with Spring Boot via REST APIs (consuming APIs in frontend frameworks).
2. **APIs & JSON:**
    - Learn how to design and consume RESTful APIs in Spring Boot.
    - Work with **JSON** for frontend-backend communication.
    - Understand **HATEOAS** if needed in future for more complex API designs.
3. **Docker and Deployment (Optional for now):**
    - Once you understand Spring Boot, look into containerizing your application using **Docker**.
    - Learn to deploy your Spring Boot application on platforms like Heroku, AWS, or DigitalOcean.