MODULE 9

Java – Spring Webservices

- ## Introduction to Web Services

  QUES: What are Web Services?

  What are Web Services?

  Web services are standardized ways for applications to communicate over the internet. They allow different applications, regardless of platform or language, to exchange data and perform functions.

  Types of Web Services:

  - **SOAP (Simple Object Access Protocol):** A protocol for exchanging structured information in web services using XML.
  - **REST (Representational State Transfer):** A set of principles for designing networked applications, using HTTP and standard operations (GET, POST, PUT, DELETE).

  Advantages of Web Services:

  - **Platform and language independence:** Web services can work across different platforms and programming languages.
  - **Integration across diverse systems:** They enable seamless communication between various systems and applications.
  - **Enables microservices architecture:** Facilitates the development of scalable and independent microservices.

- ## Basics of REST APIs

  QUES: What is REST (Representational State Transfer)?

  REST is an architectural style for designing networked applications. It is based on stateless communication and uses standard HTTP methods to perform operations on resources.

- **Statelessness:** Each request from a client to the server must include all the information needed for processing; no session state is stored.

- **Resource-based URLs:** Resources are identified by URIs (Uniform Resource Identifiers).
- **HTTP Methods:**
    - **GET:** Retrieve data from the server.
    - **POST:** Submit data to the server.
    - **PUT:** Update existing data on the server.
    - **DELETE:** Remove data from the server.
- **Status Codes:** Used to indicate the outcome of HTTP requests (e.g., 200 for success, 404 for not found).

Key REST Concepts:

- **Resources:** Everything in REST is considered a resource, such as data or services, that can be accessed or manipulated.
- **URI:** A unique identifier (Uniform Resource Identifier) for each resource.
- **Stateless Communication:** Each request is independent, meaning the server does not store any session data between requests.

- ## Spring MVC (Model-View-Controller)

QUES: Spring MVC Overview:

Explanation of the MVC design pattern: Model, View, and Controller.

MVC Design Pattern:

Model: Represents the application's data and business logic.

View: The user interface that displays the data from the model (e.g., HTML pages).

Controller: Handles user input, processes it (often involving the model), and returns a view.

How Spring MVC handles incoming web requests and maps them to the correct controller.

- When a request is received, Spring MVC routes it to the appropriate **Controller** based on the URL pattern.
- The controller processes the request, interacts with the model if necessary, and returns the name of the **View** (e.g., a JSP or Thymeleaf template) to be rendered.

QUES: Controller and View:

Creating a controller to handle user requests.

A controller is a Java class annotated with @Controller, and it contains methods annotated with @RequestMapping or more specific annotations like @GetMapping to handle requests.

```java
@Controller
public class MyController {
   @GetMapping("/hello")
   public String hello(Model model) {
      model.addAttribute("message", "Hello, World!");
      return "helloView"; // View name (Thymeleaf template or JSP)
   }
}
```

Using a view template engine (e.g., Thymeleaf) to render dynamic data.

Thymeleaf is commonly used for rendering views in Spring MVC.
 It processes the dynamic content within templates.

```html
<!-- helloView.html (Thymeleaf template) -->
<!DOCTYPE html>
<html>
<head>
   <title>Greeting</title>
</head>
<body>
   <h1>${message}</h1>
</body>
</html>
```

- **Aspect-Oriented Programming (AOP)**

  QUES: What is AOP (Aspect-Oriented Programming)?

  AOP is a programming paradigm that allows you to separate cross-cutting concerns (e.g., logging, security, transaction management) from the main business logic. It helps improve code modularity by isolating concerns that affect multiple parts of the application.

  Overview of AOP and how it helps in separating cross-cutting concerns (e.g., logging, security, transaction management)

  AOP provides a way to add functionality to existing code without modifying it directly. It focuses on concerns that are spread across multiple classes and typically involve code that cuts across multiple methods (like logging or security checks). With AOP, you can define reusable code (aspects) and apply it where needed, making the code cleaner and easier to maintain.

Key AOP Terms:

- **Aspect:** A module that encapsulates a cross-cutting concern, such as logging, transaction management, or security. An aspect can be thought of as the code that defines the behavior to be applied across different points in the application.
- **Advice:** The action that is taken by an aspect at a specific joinpoint. It can be applied in various ways:
  - **Before:** Code that runs before the joinpoint.
  - **After:** Code that runs after the joinpoint.
  - **Around:** Code that runs both before and after the joinpoint, and can even prevent the execution of the joinpoint.
- **Joinpoint:** A point in the execution of the program where an aspect can be applied. Examples include method calls, field accesses, or object instantiations.
- **Pointcut:** An expression that defines where and when advice should be applied. It determines which joinpoints the aspect will target. For example, it could specify that the advice should be applied before a method in a certain package is called.

● Spring REST (CRUD API, Pagination, Fetching from Multiple Tables, Image Upload/Download)

QUES: Spring REST Overview:

Introduction to creating RESTful services in Spring Boot.

Spring Boot simplifies creating REST APIs by providing embedded servers and auto-configurations.

It allows easy development of services that can be consumed by clients via HTTP using methods like GET, POST, PUT, and DELETE.

Use of @RestController to create REST APIs.

@RestController is a specialized annotation of @Controller for RESTful services. It automatically serializes return objects into JSON (or XML).

Example

```
@RestController
public class MyRestController {
  @GetMapping("/api/hello")
  public String sayHello() {
    return "Hello, World!";
  }
}
```

Handling HTTP requests and returning JSON or XML responses.

By default, Spring Boot returns JSON, but XML can also be configured.
Spring uses Jackson (for JSON) and JAXB (for XML) for serializing data.
Example for JSON response:

```
{ "firstName": "John", "lastName": "loe" }
```

To return XML, include spring-boot-starter-xml and use JAXB annotations like @XmlRootElement.

QUES: Pagination:
Introduction to pagination in REST APIs to handle large datasets.

Pagination is essential for handling large datasets efficiently in REST APIs by splitting data into smaller chunks.

Use of Pageable and Page interfaces from Spring Data JPA for pagination support.

Pageable Interface: Represents pagination information like page number and size.

Page Interface: Contains paginated data and metadata (total pages, total elements).

Example:

Repository Layer:

```
public interface UserRepository extends JpaRepository<User, Long> {

   Page<User> findByLastName(String lastName, Pageable pageable);

}
```

Service Layer:

```
public Page<User> getUsersByLastName(String lastName, int page, int size) {

   Pageable pageable = PageRequest.of(page, size);

   return userRepository.findByLastName(lastName, pageable);

}
```

Controller Layer:

```
@GetMapping("/api/users")
```

```
public Page<User> getUsersByLastName(@RequestParam String lastName,
@RequestParam int page, @RequestParam int size) {

    return userService.getUsersByLastName(lastName, page, size);

}
```

QUES: CRUD Operations

Spring Data JPA simplifies the implementation of CRUD (Create, Read, Update, Delete) operations by providing built-in methods in the JpaRepository interface. Here's a quick overview of how to implement these operations using Spring Data JPA.

Create, Read, Update, Delete (CRUD) operations using Spring Data JPA.

1. Create Operation: Save a new entity to the database.

Use the save() method provided by JpaRepository to create or update an entity.

2. Read Operation: Retrieve an entity or list of entities.

Find by ID: Use findById() to fetch a single entity.

Find All: Use findAll() to fetch all entities.

3. Update Operation: Modify an existing entity.

Use save() to update an entity. Spring Data JPA will automatically update the existing record if the entity's ID matches one in the database.

4. Delete Operation: Delete an entity by its ID.

Use deleteById() to delete a record by its ID. Alternatively, use delete() if you have the entity object.

QUES: Fetching Data from Multiple Tables
In a relational database, data is often spread across multiple tables, and **JPA** (Java Persistence API) provides several annotations to define relationships between entities. These relationships allow you to fetch related data from multiple tables efficiently.

Use of JPA relationships (@OneToOne, @OneToMany, @ManyToOne, and @ManyToMany) to retrieve related data from multiple tables.

1. **@OneToOne**: Defines a one-to-one relationship between two entities.

A one-to-one relationship means that each record in one table is associated with exactly one record in another table.

2. **@OneToMany**: Defines a one-to-many relationship between two entities.

A one-to-many relationship means that each record in one table can have multiple related records in another table.

3. **@ManyToOne**: Defines a many-to-one relationship between two entities.

A many-to-one relationship means that multiple records in one table can be associated with a single record in another table.

4. **@ManyToMany**: Defines a many-to-many relationship between two entities.

A many-to-many relationship means that multiple records in one table can be associated with multiple records in another table.

QUES: Image Upload/Download

Handling file upload and download in a Spring REST API.

To handle file upload and download in a Spring REST API:

1. **File Upload**:
   - Use @PostMapping and @RequestParam for uploading files.
   - Use MultipartFile to receive the file.

   Example:

   ```
   @PostMapping("/upload")
   public ResponseEntity<String> uploadFile(@RequestParam("file") MultipartFile
   file) {
     // Process file
     return ResponseEntity.ok("File uploaded successfully");
   }
   ```

2. **File Download**:
   - Use @GetMapping and ResponseEntity to send the file.

- Set the appropriate headers (e.g., Content-Disposition) for file download.

Example:

```
@GetMapping("/download/{filename}")
public ResponseEntity<Resource> downloadFile(@PathVariable String filename)
throws MalformedURLException {
   Path filePath = Paths.get("path/to/files").resolve(filename);
   Resource resource = new UrlResource(filePath.toUri());
   return ResponseEntity.ok()
         .header(HttpHeaders.CONTENT_DISPOSITION, "attachment; filename=\""
+ resource.getFilename() + "\"")
         .body(resource);
}
```