**Lab-11 Tasks**

**Task 1: Customized Greeting Service with Dependency Injection**

In this task, you will build a RESTful API with a custom greeting service that responds differently based on the time of day (morning, afternoon, evening). Begin by creating a GreetingService class to encapsulate the logic for determining the correct greeting based on the current time. Then, use the @Autowired annotation to inject this service into a GreetingController class. The API endpoint, /greet/{name}, should greet the user with a personalized message based on the time of day. Additionally, add an optional language query parameter to allow the user to request the greeting in different languages, such as English, French, or Spanish. Implement error handling so that if an unsupported language is requested, the API returns an informative error message. This task encourages the use of dependency injection, @RequestMapping, and response customization, helping you understand how Spring Boot controllers can manage more dynamic and user-friendly interactions.

**Task 2: Multi-Action Form Submission Controller**

This task involves creating a multi-functional HTML form that can handle both login and registration actions in a single interface. Begin by designing an HTML form with two separate submission buttons, each directing to a different URL: one for login and one for registration. In the backend, create a UserController with @PostMapping and @GetMapping methods to handle these actions. The registration endpoint should verify if a username is already taken by checking against a predefined list, returning a message if the username is unavailable. The login endpoint should validate user credentials and return an appropriate success or failure message. Use @RequestParam to retrieve and process form data. This task provides experience with multiple HTTP methods within a single controller, as well as simple form validation logic.

**Task 3: Item Management with CRUD Endpoints**

For this task, you’ll create a simple inventory management API that simulates CRUD operations for an in-memory list of items. Implement a ItemController with RESTful endpoints to support adding, updating, deleting, and retrieving items. Use @PostMapping("/items") to create new items, @PutMapping("/items/{id}") to update existing items, @DeleteMapping("/items/{id}") to remove items, and @GetMapping("/items/{id}") to fetch specific items by ID. Ensure that the controller handles errors, such as when an item ID does not exist, by returning a meaningful response. This exercise helps solidify an understanding of REST principles, while using Spring annotations for creating multiple CRUD endpoints in a single class without involving a database.

**Task 4: Path and Query Parameter Combination in URL Routing**

This task is focused on using both path and query parameters in a single endpoint to build a flexible and specific report generator. Create a ReportController class that has an endpoint /report/{year}/{month} to fetch reports based on a given year and month. Use @PathVariable to capture the year and month values directly from the URL and @RequestParam to add an optional type parameter (e.g., summary or detailed). The response should change based on the provided report type, returning a message such as "Displaying a detailed report for August 2024." This exercise emphasizes the power of combining path and query parameters, allowing you to customize responses dynamically based on multiple user-provided inputs.

**Task 5: Thymeleaf-Based Product List and Filter**

In this task, you’ll use Thymeleaf templates to build a basic product listing page with filter functionality. Create a ProductController class with a @GetMapping("/products") endpoint that retrieves a list of products (simulated with a static list) and displays them on a products.html page. Use @RequestParam to add an optional category filter, allowing users to view only the products in a specified category. In the products.html template, display the product list and apply the category filter based on user input. This task introduces you to Thymeleaf for server-side HTML rendering, providing a foundation for developing dynamic and interactive web pages with Spring Boot.