Java Spring Programming

Section 1 - spring

- **Dependancy, Dependancy Injection**
- **Coupling : Tight and Loose**
- **Inversion of Control**
- **Spring Framework Introduction and Architecture**
- **Spring Beans**
- **React Lifecycle(Mounting)**
- **Dependency Injection with Spring**
- **Autowiring**
- **Bean Inheritance**
- **Collection Merging**
- **Bean Aliasing**
- **Bean Scopes**
- **Bean Lifecycle**

*1. Create a web-based employee management system using the Spring Framework. Implement CRUD (Create, Read, Update, and Delete) operations for managing employee records. Use Spring Beans for dependency injection and configure bean scopes appropriately. Use Spring MVC for handling requests and displaying data.

*2. Develop a product Catalog application using Spring Framework. Implement functionality to add, view, update, and delete products. Use Spring Beans and dependency injection to manage dependencies between components. Utilize spring's bean inheritance feature to define common properties for different types of products.

*3. Build a shopping cart application using the Spring Framework. Implement features such as adding products to the cart, updating quantities, and checking out. Utilize spring's auto wiring capability to inject dependencies and manage the lifecycle of the shopping cart.

#1. Develop a library management system using the Spring Framework. Implement functionality to manage books, borrowers, and loans. Use Spring Beans for dependency injection and configure bean scopes appropriately. Implement bean aliasing for different services or components.

#2. Build a social media dashboard application using the Spring Framework. Implement features to retrieve and display social media posts, handle user interactions, and manage user sessions. Utilize spring's bean lifecycle management to control the creation and destruction of beans.

#3. Create an event registration system using the Spring Framework. Implement features for users to register for events, view event details, and manage registrations. Utilize spring's collection merging feature to consolidate data from multiple sources, such as event information and user details.

Section 2 – spring MVC

- **Introduction To Spring MVC**
- **Handler Mapping**
- **Controllers**
- **Services**
- **Models**

*1. Build a user registration system using Spring MVC. Implement a registration form with fields like name, email, and password. Use Spring's handler mapping to map incoming requests to the appropriate controller. Create controllers to handle registration requests, validate user input, and store user details in a database using services and models.

*2. Develop a product inventory management system using Spring MVC. Implement controllers to handle CRUD operations for managing products. Use services to perform business logic, such as adding new products, updating quantities, and retrieving product information. Utilize models to represent the product data.

#1. Create a blogging platform using Spring MVC. Implement controllers to handle requests for creating blog posts, viewing blog posts, and managing user comments. Use services to process and store blog-related data. Implement models to represent blog posts and comments.

#2. Build an e-commerce website using Spring MVC. Implement controllers to handle requests for browsing products, adding items to the cart, and processing orders. Use services to handle business logic, such as calculating total prices, applying discounts, and managing inventory. Utilize models to represent products, carts, and orders.

#3. Develop an event management system using Spring MVC. Implement controllers to handle requests for creating events, viewing event details, and managing attendee registrations. Use services to process event-related data, handle registrations, and perform validations. Utilize models to represent events and attendees.

Section 3 – spring Boot

- **Spring Boot Configuration**
- **Controller and RestController**
- **RequestBody and ResponseBody**
- **JPARepository and CRUDRepository**
- **CRUD Operations**
- **Logging API**
- **Swagger API**
- **Spring Security**
- **Spring Authentication and Authorization**
- **JWT Token**
- **MongoDB with Spring**
- **Microservices**
- **Services Discovery**
- **Euerka Server and Client**

*1. Build a user management system using Spring Boot. Implement RESTful APIs using controllers and RestControllers. Use RequestBody and ResponseBody annotations for handling JSON data. Utilize JPARepository or CRUDRepository to perform CRUD operations on user entities. Implement logging using the Spring Boot logging API.

*2. Enhance an existing Spring Boot application by adding Swagger API documentation. Configure Swagger to generate API documentation for the RESTful endpoints in your application. Document your endpoints with meaningful descriptions, request/response examples, and annotations. Test and verify the generated API documentation using Swagger UI.

#1. Secure a RESTful API using Spring Security. Implement authentication and authorization mechanisms using Spring Security features such as login endpoints, user roles, and access control rules. Explore different authentication methods like JWT token-based authentication. Ensure that only authenticated and authorized users can access protected resources.

#2. Develop a microservices-based application using Spring Boot. Divide your application into multiple microservices, each responsible for a specific functionality. Implement inter-service communication using RESTFUL APIs or messaging frameworks like RabbitMQ. Utilize spring Cloud components for service discovery, load balancing, and fault tolerance.

#3. Integrate MongoDB with your Spring Boot application. Use spring Data MongoDB to perform CRUD operations on MongoDB collections. Configure connection properties, define MongoDB entities, and leverage Spring Data features like query methods and repository interfaces. Implement logging and exception handling for MongoDB operations.