TPC

Java Spring Boot Interview Preparation Plan

This plan is designed to comprehensively prepare you for backend interviews, covering essential topics in **Java/Spring Boot**, **Data Structures & Algorithms (DSA)**, and **System Design**. By following this plan, you will be equipped with the skills and knowledge necessary to excel in backend development interviews, from language fundamentals and framework expertise to solving algorithmic problems and designing scalable systems.

Day 1: Core Java & Spring Boot Fundamentals

- **Java Fundamentals**:
  - Data types, control flow, and OOP principles (inheritance, encapsulation, polymorphism).
  - Deep dive into Java collections (Lists, Maps, Sets), streams, lambda expressions, functional programming, and method references.
  - Generics and wildcard usage.
- **Spring Boot Basics**:
  - Overview of Spring Boot architecture, its auto-configuration feature, and how Spring Boot simplifies application development.
  - Creating a Spring Boot project from scratch using Spring Initializr.
  - RESTful API creation with basic CRUD operations.
  - **Dependency Injection**: `@Autowired`, constructor injection, field injection, and setter injection.
- **Practice**:
  - Implement OOP principles and explore Java collections in coding problems.
  - Build a simple Spring Boot CRUD application and use dependency injection.
- **Resources**:
  - *Effective Java* by Joshua Bloch
  - Spring Boot Official Documentation: Getting Started

Day 2: Spring Boot Dependency Injection, Application Context & Beans

- **Spring Beans**:
  - Understanding the Spring IoC container, `@Component`, `@Bean`, `@Configuration`, and `@Scope`.
  - Difference between singleton, prototype, request, and session-scoped beans.
- **Application Context**:
  - Life cycle of a Spring bean, bean scopes, and bean factory vs application context.

    - Use of `@PostConstruct` and `@PreDestroy`.
- **Profiles in Spring Boot**:
    - Using `@Profile` annotation for environment-specific beans (development, production).
- **Practice**:
    - Implement beans using `@Component`, `@Bean`, and explore scopes.
    - Create environment-specific beans using `@Profile`.
- **Resources**:
    - Spring Boot Official Documentation: Beans and Scopes

Day 3: Advanced Java & Spring Boot Features

- **Concurrency & Multithreading**:
    - Advanced topics in Java multithreading: `CompletableFuture`, `ExecutorService`, synchronization, thread pools, and concurrent collections.
    - Java memory model, garbage collection tuning, and optimizing performance.
- **Spring Boot Advanced**:
    - Creating **interceptors** to intercept HTTP requests.
    - Use of **asynchronous methods** with `@Async` and handling asynchronous errors.
    - **Spring AOP** (Aspect-Oriented Programming): Using aspects for logging, security, and transaction management with `@Before`, `@After`, and `@Around`.
- **Spring Security**:
    - Setting up Spring Security, JWT authentication, and securing endpoints.
    - Role-based access control.
- **Practice**:
    - Implement asynchronous tasks using `@Async` and interceptors in a Spring Boot application.
    - Secure endpoints with JWT and role-based access control.
- **Resources**:
    - *Java Concurrency in Practice* by Brian Goetz
    - Spring Boot Security

Day 4: Error Handling, File Handling & Design Patterns

- **Global Exception Handling**:
    - Implementing global exception handling in Spring Boot using `@ControllerAdvice` and `@ExceptionHandler`.
    - Custom error messages, HTTP status codes, and validation error handling using `@Valid` and `BindingResult`.

- **File Handling**:
    - Asynchronous file handling using `CompletableFuture`.
    - Serving files from Spring Boot with proper MIME types and error handling.
- **Design Patterns**:
    - Singleton, Factory, Observer, and Strategy patterns in Java.
    - Applying these patterns in Spring Boot, particularly the Singleton and Observer pattern in event-driven architectures.
- **Practice**:
    - Build a global error handling mechanism in Spring Boot.
    - Create a design pattern-based Java program and integrate patterns in Spring Boot.
- **Resources**:
    - *Head First Design Patterns* by Eric Freeman
    - Spring Boot Official Documentation: Exception Handling

Day 5: Databases (SQL/NoSQL) & Spring Boot Data Access

- **SQL vs NoSQL**:
    - Differences between relational and non-relational databases.
    - When to use SQL (ACID) vs NoSQL (BASE), common use cases.
    - Database indexing and optimization techniques for SQL queries.
    - **Indexing**: Understanding when and how to use indexes effectively.
    - **Optimizing Joins**: How to write efficient joins (inner/outer joins) that minimize performance issues.
    - **Subqueries**: Writing optimized subqueries and replacing them with joins when needed.
    - **Aggregate Queries**: Practice querying aggregate data, filtering based on conditions, and ensuring efficient `GROUP BY`, `HAVING`, and `ORDER BY` clauses.
    - **Analytical Queries**: Understanding window functions such as `ROW_NUMBER()`, `RANK()`, and `DENSE_RANK()` for ranking and sorting data.
    - Foreign key constraints and ensuring proper usage of relationships to maintain data integrity.
- **Spring Data JPA**:
    - Working with Spring Data JPA for SQL databases: Entity relationships (One-to-One, One-to-Many, Many-to-Many).
    - Writing complex queries, using `@Query`, pagination, and sorting.
- **NoSQL Integration**:
    - Using Spring Boot with NoSQL databases like MongoDB.
    - CRUD operations, aggregations, and pagination in MongoDB.
- **Practice**:
    - Create a Spring Boot application integrated with SQL (MySQL or PostgreSQL) and NoSQL (MongoDB).
    - Write complex queries and implement pagination in both databases.
- **Resources**:

    - *Spring in Action* by Craig Walls
    - Spring Data JPA Documentation

Day 6: Microservices Architecture & Spring Boot

- **Microservices Fundamentals**:
    - Introduction to microservices, how Spring Boot facilitates microservice creation.
    - **Service discovery** with Netflix Eureka, API Gateway using Spring Cloud Gateway.
    - **Inter-service communication** using REST and messaging queues like RabbitMQ or Kafka.
- **Resilience & Fault Tolerance**:
    - **Circuit Breakers** with Resilience4J or Hystrix.
    - Handling service failures, retries, and timeouts.
- **Configuration Management**:
    - Centralized configuration using Spring Cloud Config and Spring Boot Actuator for monitoring microservices.
- **Practice**:
    - Build a small microservices application using Spring Boot, Eureka, and Spring Cloud Gateway.
    - Implement circuit breakers and distributed tracing.
- **Resources**:
    - *Building Microservices* by Sam Newman
    - Spring Cloud Documentation

Day 7: Testing & Optimization in Spring Boot

- **Testing Spring Boot Applications**:
    - Unit testing using JUnit 5 and Mockito.
    - Writing integration tests with `@SpringBootTest`, `@MockBean`, and `TestRestTemplate`.
    - Testing repositories with embedded databases like H2.
- **Performance Optimization**:
    - Profiling and optimizing Spring Boot applications using **JVisualVM**, Spring Boot Actuator, and Micrometer.
    - Caching with Spring Boot using `@Cacheable` and Redis for performance improvement.
- **Practice**:
    - Write unit and integration tests for your Spring Boot application.
    - Use Actuator for monitoring and JVisualVM for profiling.
- **Resources**:
    - *Pragmatic Unit Testing in Java* by Jeff Langr
    - Spring Boot Testing Documentation

Day 8: Distributed Systems & Security

- **Distributed Systems**:
  - Understanding distributed system patterns: CAP theorem, data consistency models, and eventual consistency.
  - Implementing distributed tracing with **Zipkin** or **Jaeger**.
  - **Event-Driven Architecture**: Implementing event-driven microservices using **Kafka** or **RabbitMQ**.
- **Security in Microservices**:
  - OAuth2 authorization server, API security, and token management in a microservices architecture.
  - Securing inter-service communication and ensuring data integrity.
- **Practice**:
  - Implement distributed tracing in a Spring Boot microservice.
  - Set up security between microservices using OAuth2 and JWT tokens.
- **Resources**:
  - *Designing Data-Intensive Applications* by Martin Kleppmann
  - Spring Security OAuth2 Documentation

With a solid understanding of **Java/Spring Boot** fundamentals, it's time to shift focus to **System Design**. This section will prepare you to design scalable, efficient, and maintainable systems, a crucial skill for backend interviews. You'll cover key aspects like high-level and low-level design, optimization, and real-world case studies.

Day 9: Introduction to System Design

Topics to Cover:

- **System Design Basics:**
  - Understand the purpose of system design, the key concepts such as scalability, availability, reliability, and maintainability.
- **Design Process:**
  - Learn the steps in the system design process—requirements gathering, high-level design, detailed design, and trade-offs.
- **Functional vs Non-Functional Requirements:**
  - Clarify and emphasize non-functional requirements during design, including scale, traffic handling, and capacity planning.

Practice:

- Analyze a simple system, identify key components, and discuss non-functional requirements.

Resources:

- *Designing Data-Intensive Applications* by Martin Kleppmann
- *System Design Primer GitHub Repository*
- *High Scalability Blog*

Day 10: High-Level Design (Architecture)

Topics to Cover:

- **Architectural Patterns:**
  - Study common architectural patterns such as microservices, monolithic, and serverless architectures.
- **Scalability and Load Balancing:**
  - Understand how to design systems that scale horizontally and vertically.
  - Learn about load balancing techniques and strategies for distributing traffic.

Practice:

- Design a high-level architecture for a common application, such as an e-commerce site or social media platform.
- Draw diagrams of different architectural patterns and their components.

Resources:

- *Microservices Patterns* by Chris Richardson
- *AWS Well-Architected Framework*
- *System Design Interview – An insider's guide*

Day 11: Low-Level Design (Component Design)

Topics to Cover:

- **Component Design Principles:**
  - Learn about the SOLID principles (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion).
- **Data Models and Storage:**
  - Understand different types of databases (SQL vs NoSQL) and their use cases.
  - Study how to design data schemas and models.
- **Handling Long-Running Jobs:**
  - Focus on job queues, worker nodes, and task tracking.

    - Learn to design APIs for long-running tasks and asynchronous responses.

Practice:

- Design components for a system and API endpoints for long-running tasks (e.g., scraping services or job dispatch systems).

Resources:

- *Clean Architecture* by Robert C. Martin
- *SQL vs NoSQL Guide*
- *Database System Concepts* by Silberschatz, Korth, and Sudarshan

---

Day 12: High-Level System Design Case Study

Topics to Cover:

- **Case Studies:**
    - Study detailed case studies of real-world systems like Google Search, Netflix, or Twitter.
- **Trade-Offs and Decisions:**
    - Learn how to evaluate trade-offs, especially between consistency, availability, and partition tolerance.
- **Edge Cases:**
    - Consider pagination, infinite scrolling, domain redirection, and robots.txt restrictions in system design.

Practice:

- Review and analyze a case study. Identify design decisions and trade-offs made.
- Create a similar high-level design for a system of your choice, justifying your design choices.

Resources:

- *Designing Data-Intensive Applications* by Martin Kleppmann
- *Netflix Technology Blog*

Day 13: Low-Level System Design Case Study

Topics to Cover:

- **Detailed Component Design:**
    - Learn about class diagrams, sequence diagrams, and component interactions.
- **Handling Concurrency:**

    - Study techniques for concurrency and parallelism in system design.
- **Job Queues and Worker Systems:**
    - Focus on retry mechanisms, message queues, and handling failures in worker nodes.

Practice:

- Design detailed components, focusing on concurrency and worker-node systems.
- Implement a component that demonstrates task distribution using job queues.

Resources:

- *Design Patterns: Elements of Reusable Object-Oriented Software* by Gamma, Helm, Johnson, and Vlissides
- *Java Concurrency in Practice* by Brian Goetz

Day 14: Performance and Optimization

Topics to Cover:

- **Performance Metrics:**
    - Understand how to measure performance, including latency, throughput, and resource utilization.
- **Optimization Techniques:**
    - Learn techniques for optimizing performance, including caching, indexing, and database optimization.
- **Cache Management and Expiry:**
    - Understand cache invalidation policies, eventual consistency, and high availability strategies.

Practice:

- Analyze performance bottlenecks in the system designed on Day 2.
- Implement optimization techniques to improve performance.

Resources:

- *High Performance Browser Networking* by Ilya Grigorik
- *System Design Interview – An insider's guide*
- *Google Cloud Platform Performance Optimization Guide*

Day 15: Challenge Day

Take The Proven Club challenge to get your Skill Assessment index!

By taking **The Proven Club Challenge** you will assess your skills and measure your readiness. This will provide valuable insights into your strengths and areas for improvement as you prepare to ace your backend interviews.