🛠️ Section 1: Spring Boot Fundamentals

1️⃣ What is Spring Boot, and how does it differ from Spring Framework?

💡 **Answer:**
Spring Boot is a **framework built on top of Spring** that simplifies application development by providing **auto-configuration, embedded servers, and reduced boilerplate code**.

Key Differences:
✅ **Spring Framework** → Requires manual configuration (`XML` or `Java-based`).
✅ **Spring Boot** → Provides **auto-configuration**, so we don't need to set up everything manually.

2️⃣ What are the key dependencies in a Spring Boot application?

💡 **Answer:**
The main dependencies in `pom.xml` (for a Maven project) are:
✅ **Spring Boot Starter Web** → `spring-boot-starter-web` (For REST APIs)
✅ **Spring Boot Starter Data JPA** → `spring-boot-starter-data-jpa` (For database integration)
✅ **Spring Boot Starter Security** → `spring-boot-starter-security` (For authentication & authorization)
✅ **Spring Boot Starter Validation** → `spring-boot-starter-validation` (For input validation)
✅ **JWT Dependencies** → `jjwt` (For token-based authentication)

3️⃣ How does Spring Boot simplify application development?

💡 **Answer:**
Spring Boot makes development easier by:
✅ **Auto-Configuration** → Automatically configures beans based on classpath dependencies.
✅ **Embedded Servers** → Comes with Tomcat, Jetty, etc., so we don't need external servers.
✅ **Production-Ready Features** → Includes monitoring, metrics, and logging (`Spring Actuator`).

4️⃣ What is the role of `application.properties` or `application.yml`?

💡 **Answer:**
This file **stores application configurations** like database settings, server ports, and logging levels.

✅ **Example (application.properties):**

properties Copy Edit

```
server.port=8080 spring.datasource.url=jdbc:mysql://localhost:3306/expense_tracker
spring.datasource.username=root spring.datasource.password=yourpassword
```

✅ **Example (application.yml):**

yaml Copy Edit

```
server: port: 8080 spring: datasource: url: jdbc:mysql://localhost:3306/expense_tracker
username: root password: yourpassword
```

💡 **This makes it easy to change configurations without modifying the code.**

5️⃣ How does dependency injection work in Spring Boot?

💡 **Answer:**
Dependency Injection (DI) is a design pattern where Spring **automatically injects objects** into components instead of manually creating them.

✅ **Example in Spring Boot:**

java Copy Edit

```
@Service public class ExpenseService { private final ExpenseRepository expenseRepository;
@Autowired // Injects dependency public ExpenseService(ExpenseRepository
expenseRepository) { this.expenseRepository = expenseRepository; } }
```

💡 **Spring Boot manages the `ExpenseRepository` instance and injects it into `ExpenseService` automatically.**

📡 Section 2: REST API & Controller Layer

6️⃣ How are RESTful APIs structured in Spring Boot?

💡 **Answer:**

Spring Boot follows a **layered architecture**:

✅ **Controller Layer** (`@RestController`) → Handles HTTP requests.
✅ **Service Layer** (`@Service`) → Implements business logic.
✅ **Repository Layer** (`@Repository`) → Interacts with the database using JPA.

✅ **Example API** (`ExpenseController.java`):

java Copy Edit

```
@RestController @RequestMapping("/expenses") public class ExpenseController { @Autowired private ExpenseService expenseService; @GetMapping("/get/all") public List<Expense> getAllExpenses() { return expenseService.getAllExpenses(); } }
```

---

7 What is the role of `@RestController` vs `@Controller`?

💡 **Answer:**

✅ `@RestController` → Used for **REST APIs** (`@ResponseBody` is automatically applied).
✅ `@Controller` → Used for **handling views (Thymeleaf, JSP, etc.)**.

---

8 How do you handle GET, POST, PUT, and DELETE requests in Spring Boot?

💡 **Answer:**

✅ **GET** → Fetch data
✅ **POST** → Create new data
✅ **PUT** → Update existing data
✅ **DELETE** → Remove data

✅ **Example API** (`ExpenseController.java`):

java Copy Edit

```
@RestController @RequestMapping("/expenses") public class ExpenseController { @PostMapping("/add") public Expense addExpense(@RequestBody Expense expense) { return expenseService.addExpense(expense); } @PutMapping("/update/{id}") public Expense updateExpense(@PathVariable Long id, @RequestBody Expense expense) { return expenseService.updateExpense(id, expense); } @DeleteMapping("/delete/{id}") public void deleteExpense(@PathVariable Long id) { expenseService.deleteExpense(id); } }
```

9️⃣ What is the purpose of `@RequestParam`, `@PathVariable`, and `@RequestBody`?

💡 **Answer:**
✅ `@RequestParam` → Extracts query parameters.
✅ `@PathVariable` → Extracts values from the URL path.
✅ `@RequestBody` → Maps JSON data to Java objects.

✅ **Example:**

java Copy Edit

```java
@GetMapping("/filter") public List<Expense> filterExpenses(@RequestParam String category)
{ return expenseService.getExpensesByCategory(category); }
```

🔐 Section 4: Security (JWT, Role-Based Access, CSRF)

1️⃣6️⃣ How does authentication work in your Spring Boot application?

💡 **Answer:**
1️⃣ User logs in (`/auth/login`).
2️⃣ Backend **validates credentials** and generates a JWT token.
3️⃣ Token is sent to the frontend and stored in `sessionStorage` .
4️⃣ The frontend includes the token in API requests (`Authorization: Bearer <token>`).

1️⃣7️⃣ How is JWT (JSON Web Token) used for authentication?

💡 **Answer:**

- JWT is a **self-contained token** that contains user details and an expiration time.
- The backend **verifies the token** on each request using a filter.

✅ **Example JWT Filter in Spring Boot:**

java Copy Edit

```
public class JwtFilter extends OncePerRequestFilter { @Override protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException { String token = request.getHeader("Authorization").replace("Bearer ", ""); if (token != null && JwtUtil.validateToken(token)) { UsernamePasswordAuthenticationToken authentication = JwtUtil.getAuthentication(token); SecurityContextHolder.getContext().setAuthentication(authentication); } filterChain.doFilter(request, response); } }
```

---

1 8 How do you implement role-based access control (RBAC) in Spring Boot?

💡 Answer:

- Use `@PreAuthorize` to restrict access based on roles.

java Copy Edit

```
@PreAuthorize("hasRole('ADMIN')") @GetMapping("/admin-only") public String adminEndpoint() { return "This is only accessible to admins"; }
```

⚡ Section 5: Performance Optimization & Scalability

2 1 How would you optimize API performance in your Spring Boot backend?

✅ Enable Caching (`@Cacheable`)
✅ Use Pagination (`Pageable` in Spring Data JPA)
✅ Use Asynchronous Processing (`@Async`)
✅ Optimize Database Queries (Indexing, Joins, Query Caching)