

### **Section 1: Spring Boot Fundamentals**

#### 1 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:**

- $\checkmark$  Spring Framework  $\rightarrow$  Requires manual configuration ( XML or Java-based ).
- $\checkmark$  Spring Boot  $\rightarrow$  Provides auto-configuration, so we don't need to set up everything manually.

#### 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)

#### 3 How does Spring Boot simplify application development?

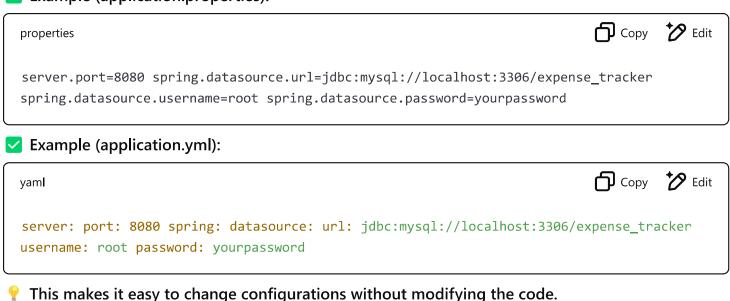
#### Answer:

Spring Boot makes development easier by:

- $\checkmark$  Auto-Configuration  $\rightarrow$  Automatically configures beans based on classpath dependencies.
- ightharpoonup Embedded Servers ightharpoonup Comes with Tomcat, Jetty, etc., so we don't need external servers.
- $\bigvee$  Production-Ready Features  $\rightarrow$  Includes monitoring, metrics, and logging (Spring Actuator).
- 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):



- 5 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:

```
@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.



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 )  $\rightarrow$  Interacts with the database using JPA.
- 🔽 Example API ( ExpenseController.java ):

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

- What is the role of @RestController vs @Controller?
- Answer:
- $\bigcirc$  @RestController  $\rightarrow$  Used for **REST APIs** (@ResponseBody is automatically applied).
- $\bigcirc$  @Controller  $\rightarrow$  Used for handling views (Thymeleaf, JSP, etc.).
- B 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 ):

```
@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); } }
```

What is the purpose of @RequestParam, @PathVariable,

#### and @RequestBody?

- Answer:
- $\bigvee$  @RequestParam  $\rightarrow$  Extracts query parameters.
- lacksquare @PathVariable ightarrow Extracts values from the URL path.
- igspace @RequestBody ightarrow Maps JSON data to Java objects.
- Example:

java

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

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

- 1 6 How does authentication work in your Spring Boot application?
- Answer:
- 1 User logs in ( /auth/login ).
- Backend validates credentials and generates a JWT token.
- Token is sent to the frontend and stored in sessionStorage.
- lacktriangle The frontend includes the token in API requests ( lacktriangle Authorization: lacktriangle Bearer lacktriangle Applies to the same lacktriangle
- 1 7 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:

```
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

@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)