Employee Management System – Interview Q&A

---

1. What is the Employee Management System project about?

Answer:
It is a Spring Boot application that provides CRUD operations for managing employees. It exposes REST APIs to create, read, update, and delete employee records, with database persistence using Spring Data JPA.

2. Which technologies and tools are used in this project?

Answer:

- **Java 15/17+** (backend language)
- **Spring Boot** (application framework)
- **Spring Data JPA + Hibernate** (ORM and persistence)
- **Maven** (build automation)
- **MySQL/PostgreSQL** (database, depending on config)
- **JUnit/Mockito** (for testing)
- **IntelliJ IDEA / Eclipse** (IDE)

3. Explain the project architecture.

Answer:

- **Controller Layer** → Handles API requests (EmployeeController).
- **Repository Layer** → Database interaction (EmployeeRepository).
- **Entity/Model Layer** → Defines employee schema (Employee).
- **Exception Handling Layer** → Manages errors (ResourceNotFoundException).
- Uses **MVC design pattern** with **REST architecture**.

4. Which dependencies are included in the pom.xml?

Answer:

- **spring-boot-starter-web** → REST API development
- **spring-boot-starter-data-jpa** → ORM & database interaction

- **mysql-connector-java** (or postgresql driver) → Database connectivity
- **spring-boot-starter-test** → Testing framework

5. What packages are present in this project?

Answer:

- com.ems.controller → REST controllers
- com.ems.model → Entity classes
- com.ems.repository → JPA repositories
- com.ems.exception → Custom exceptions
- com.ems → Main application class

6. How is the main class EmployeeManagementSystemApplication defined?

Answer:
It is annotated with @SpringBootApplication, which is a combination of:

- @Configuration (Java-based config)
- @EnableAutoConfiguration (auto-configures beans)
- @ComponentScan (scans for Spring beans in com.ems)

7. How do you run this project?

Answer:

- Import into IDE → Build with Maven → Run EmployeeManagementSystemApplication.java.
- Or run in terminal:

```
mvn spring-boot:run
```

- Application starts at http://localhost:8080.

---

8. Which database operations are supported?

Answer:

- **Create** (Add Employee)
- **Read** (Get all / Get by ID)

- **Update** (Update employee by ID)
- **Delete** (Delete employee by ID)

9. Explain the flow of a POST request to create an employee.

Answer:

1. Client sends POST /api/v1/employees with JSON body.
2. EmployeeController.saveEmployee() receives request.
3. Controller calls EmployeeRepository.save(employee).
4. Hibernate generates SQL INSERT query.
5. Employee record stored in DB.
6. Response returned with saved employee details.

10. How is exception handling implemented?

Answer:

- Custom exception: ResourceNotFoundException.
- If an employee is not found, this exception is thrown.
- Spring Boot maps it to **HTTP 404 Not Found**.
- Can be extended using @ControllerAdvice and @ExceptionHandler for global handling.

11. What is the purpose of EmployeeRepository?

Answer:
It extends JpaRepository<Employee, Long> which provides:

- save() → Insert/Update
- findById() → Retrieve employee by ID
- findAll() → Retrieve all employees
- deleteById() → Delete employee
- Custom queries can be added with @Query

12. What annotations are used in Employee entity?

Answer:

- @Entity → Marks class as JPA entity
- @Table(name = "employees") → Maps to DB table
- @Id → Primary key
- @GeneratedValue(strategy = GenerationType.IDENTITY) → Auto-increment ID
- @Column(name = "...") → Maps fields to DB columns

---

13. How do you configure database connection?

Answer:
In application.properties:

```
spring.datasource.url=jdbc:mysql://localhost:3306/ems

spring.datasource.username=root

spring.datasource.password=1234

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true
```

---

14. How are APIs tested in this project?

Answer:

- Using **Postman** or **cURL**.
- Example:

```
GET http://localhost:8080/api/v1/employees
```

- Unit tests: EmployeeManagementSystemApplicationTests.java with JUnit.

15. How is dependency injection used in this project?

Answer:

- Spring injects EmployeeRepository into EmployeeController via @Autowired.
- Eliminates manual object creation.
- Promotes loose coupling & testability.

16. How is data validation handled?

Answer:
Currently not added, but can use:

- @NotNull, @Size, @Email, @Pattern in Employee entity.
- @Valid in controller methods to validate request body.

17. How does Hibernate generate SQL queries?

Answer:

- Hibernate maps Java entity (Employee) to DB table (employees).
- save() → INSERT SQL
- findById() → SELECT ... WHERE id=?
- deleteById() → DELETE FROM employees WHERE id=?
- SQL logs shown if spring.jpa.show-sql=true.

18. What HTTP status codes are returned by the APIs?

Answer:

- 200 OK → Successful fetch/update
- 201 Created → New employee added
- 404 Not Found → Employee not found
- 400 Bad Request → Invalid request
- 500 Internal Server Error → Unexpected server failure

19. How do you ensure data consistency?

Answer:

- Spring Data JPA manages transactions automatically.
- Each repository method runs inside a transaction (@Transactional).
- If an exception occurs → rollback is triggered.

20. How do you secure the application?

Answer:
(Currently not implemented)

- Can use **Spring Security** with JWT.
- Roles: Admin (CRUD) & User (Read-only).
- Encrypt passwords using **BCrypt**.

21. How would you add a Service Layer?

Answer:

- Create EmployeeService interface + EmployeeServiceImpl.
- Business logic goes in service, not controller.
- Controller → Service → Repository flow.
- Improves testability & separation of concerns.

22. How would you implement pagination & sorting?

Answer:

- Extend JpaRepository with methods like:

```
Page<Employee> findAll(Pageable pageable);
```

- Example request:

```
GET /employees?page=0&size=5&sort=lastName,asc
```

23. How would you deploy this project?

Answer:

- **Jar Deployment**: mvn clean package → run .jar with java -jar.
- **Docker**: Create Dockerfile, build image, deploy container.
- **Cloud Deployment**: Use AWS Elastic Beanstalk, Heroku, or Kubernetes.

24. How would you improve performance?

Answer:

- Enable **caching (Redis/EhCache)**.

- Use **batch inserts/updates** for bulk operations.
- Optimize queries with indexes.
- Use **connection pooling** (HikariCP).

25. What logging is used?

Answer:

- By default → **Spring Boot uses SLF4J with Logback**.
- Logs requests, SQL queries, errors.
- Can be customized in application.properties.

26. How do you test repository layer without hitting DB?

Answer:

- Use **Mockito** to mock EmployeeRepository.
- Use **@DataJpaTest** with H2 in-memory DB.
- Ensures unit tests don't depend on actual DB.

27. How would you add Swagger to document APIs?

Answer:

- Add dependency: springdoc-openapi-ui.
- Access UI at http://localhost:8080/swagger-ui.html.
- Generates interactive API documentation automatically.

28. What are some challenges you faced while building this project?

Answer:

- Managing database schema changes.
- Handling exceptions gracefully.
- Designing REST APIs with proper status codes.
- Ensuring code modularity & clean architecture.

29. How would you extend this system with new features?

Answer:

- Add **Departments, Roles, Salaries** tables.
- Implement **Authentication/Authorization**.
- Add **Reports/Analytics (PDF/Excel export)**.
- Enable **Email notifications**.

30. If asked in an interview: "Why should we hire you based on this project?"

Answer:

- I have hands-on experience with **Spring Boot + JPA + REST APIs**.
- I understand **enterprise architecture** and best practices.
- I can extend this project to production-ready systems with **security, scaling, and testing**.
- Demonstrates my ability to build **end-to-end backend systems**.