MarvellousPortal – Interview Questions & Answers

---

1. What is the MarvellousPortal project?

A Spring Boot web application that manages batch entries with CRUD operations using Spring Data JPA, REST APIs, and a layered architecture.

2. Explain the architecture of this project.

- Controller Layer → Handles HTTP requests (BatchEntryController).
- Service Layer → Contains business logic (BatchEntryService).
- Repository Layer → Handles DB operations (BatchEntryRepository).
- Entity Layer → Maps DB tables (BatchEntry).
- Config Layer → application.properties for DB setup.

3. What technologies are used in this project?

- Spring Boot (Core Framework)
- Spring Data JPA (Database interaction)
- MySQL (Database)
- Maven (Build tool)
- Postman (API testing)
- JUnit/Spring Boot Test (Unit testing)

4. What is the role of the BatchEntry entity?

Represents a database table as a Java class with fields mapped to columns using JPA annotations like @Entity, @Id, and @GeneratedValue.

5. Explain the function of BatchEntryRepository.

- Extends JpaRepository<BatchEntry, Long>.
- Provides built-in CRUD operations (save, findAll, deleteById).
- Supports custom queries without writing SQL.

6. What does BatchEntryService do?

- Implements business logic (e.g., validating data before saving).
- Interacts with repository.
- Keeps controller lightweight.

7. Explain the role of BatchEntryController.

- Maps REST endpoints:
  - POST /batch → create batch
  - GET /batch → fetch all batches
  - DELETE /batch/{id} → delete batch
- Returns JSON responses to clients.

8. What is the purpose of the HealthCheck class?

Provides a /health endpoint to check if the application is running → helps in monitoring & debugging.

9. What is the role of application.properties?

Configures database connection, JPA, and server properties. Example:

spring.datasource.url=jdbc:mysql://localhost:3306/marvellousdb

spring.datasource.username=root

spring.datasource.password=****

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

10. What happens when you run mvn spring-boot:run?

- Builds project → loads Spring context → starts embedded Tomcat → registers beans → runs REST APIs on port 8080.

11. Why use Spring Boot instead of traditional Spring?

- Auto-configuration reduces boilerplate.
- Embedded server (Tomcat).
- Starter dependencies.
- Production-ready tools (Actuator, Health checks).

12. What is JPA and why did you use it?

Java Persistence API → Maps Java objects to DB tables.
Used for:

- Reducing SQL queries.
- Simplifying CRUD.
- Database independence.

13. How does Spring Data JPA generate queries?

- By method name → findByBatchName.
- By JPQL with @Query.
- Native SQL queries if required.

14. What is Dependency Injection in this project?

Spring injects dependencies using @Autowired.
Example: BatchEntryController injects BatchEntryService.

15. What is the difference between @RestController and @Controller?

- @Controller → returns views (JSP/HTML).
- @RestController → returns JSON/XML directly (used in this project).

16. How are exceptions handled in this project?

- Basic exception handling via try-catch.
- Can be improved with @ControllerAdvice and @ExceptionHandler for global handling.

17. How do you test the project?

- Unit testing with JUnit & Spring Boot Test.
- API testing with Postman.
- DB verification via MySQL queries.

18. How does Hibernate work in this project?

- Spring Boot uses Hibernate as the default JPA provider.
- Converts Entity classes into SQL queries.
- Handles ORM (Object Relational Mapping).

19. What is the role of pom.xml?

- Defines project dependencies (Spring Boot Starter Web, JPA, MySQL).
- Specifies build plugins.
- Manages versions.

20. How is data validated before saving?

- Service layer checks data.
- Can use annotations like @NotNull, @Size, and @Valid.

21. What is the default server in Spring Boot?

Embedded Apache Tomcat (port 8080 by default).

22. How do you deploy this project?

- Package as a JAR → mvn clean package.
- Run with java -jar target/MarvellousPortal-0.0.1-SNAPSHOT.jar.
- Deployable on cloud (AWS, Docker, Heroku).

23. How does the Repository-Service-Controller pattern help?

- Loose coupling.
- Separation of concerns.
- Easier testing & maintenance.
- Code reusability.

24. What tools were used for version control?

- Git & GitHub (.gitignore and .gitattributes present in project).

25. How do you monitor the application health?

- HealthCheck endpoint.
- Can integrate Spring Boot Actuator for metrics and monitoring.

26. What are advantages of Spring Boot in microservices?

- Lightweight.
- Embedded server.
- Easy REST API development.
- Actuator for monitoring.
- Works well with Docker & Kubernetes.

27. What are common annotations used in this project?

- @SpringBootApplication → Main entry point.
- @Entity → Maps class to table.
- @Id, @GeneratedValue → Primary key.
- @Repository → Data access layer.
- @Service → Business logic.
- @RestController, @RequestMapping → API layer.

28. How did you ensure database schema updates?

- spring.jpa.hibernate.ddl-auto=update → Auto-updates schema.
- Can also use Flyway/Liquibase for versioned migrations.

29. If asked to improve the project, what would you do?

- Add JWT Authentication for security.
- Implement DTOs for cleaner responses.
- Add Swagger for API documentation.
- Introduce logging & auditing.
- Implement global exception handling.

30. What challenges did you face while building this project?

- Configuring MySQL connectivity.
- Managing dependencies in pom.xml.
- Debugging errors in JPA queries.
- Structuring layers (Controller-Service-Repository).