🔷General Project Questions

1. Can you briefly explain your E-commerce project architecture?

The project follows a layered architecture: - **Frontend (React)**: Handles user interface and sends HTTP requests to the backend. - **Backend (Spring Boot)**: Processes business logic, authentication, and database interactions. - **Database (MySQL/PostgreSQL)**: Stores user, product, and order data. - **Security (JWT + Spring Security)**: Secures endpoints with token-based authentication.

2. Why did you choose Spring Boot and React for this project?

Spring Boot offers a robust backend with minimal setup and integrates well with databases and security mechanisms. React provides a dynamic and efficient frontend with reusable components, allowing fast updates and great user experience.

3. How do the frontend and backend communicate?

They communicate through REST APIs. React makes HTTP requests using `fetch` or `axios` to Spring Boot endpoints. Authentication tokens are passed in the headers.

4. What are the main modules in your application?

- **User Module**: Registration, login, authentication
- **Product Module**: CRUD for products
- **Cart Module**: Add/remove/view items
- **Order Module**: Place orders and view history
- **Admin Module**: Manage users and products

5. How did you manage the build and deployment process?

Used Maven for the backend and npm scripts for the frontend. Deployment was done using platforms like Netlify (frontend) and Render (backend), with CI/CD configured via GitHub.

🔷Frontend (React) Questions

6. How did you structure your React components?

Component structure was modular: - Pages (e.g., Home, Cart, Login) - Components (e.g., ProductCard, Navbar) - Context Providers (e.g., AuthContext, CartContext)

7. How did you manage state across components?

Used React's Context API for global state (auth, cart) and `useState`/`useEffect` for local component state.

8. How did you handle API calls in React?

Used `axios` to send HTTP requests. Created a utility file (`api.js`) to centralize API logic.

9. What libraries did you use for form validation, routing, or UI components?

- **React Router** for navigation
- **Formik + Yup** for form validation
- **Bootstrap** and **Material UI** for styling

10. How is authentication handled on the frontend?

JWT tokens are stored in local storage. Protected routes check for token presence before rendering.

11. How did you secure protected routes in React?

Used a `PrivateRoute` component that checks token existence and redirects unauthorized users to the login page.

🔷Backend (Spring Boot) Questions

12. How is the backend structured (controllers, services, repositories)?

- **Controller Layer**: Handles incoming HTTP requests
- **Service Layer**: Contains business logic
- **Repository Layer**: Interfaces with the database using Spring Data JPA

13. What kind of REST APIs did you expose?

APIs for authentication, product listing, cart operations, order placement, and admin operations.

14. How did you handle user authentication and authorization?

Used Spring Security with JWT for authentication. User roles (USER/ADMIN) were used to control access.

15. How does JWT work in your project?

JWT is generated on successful login. The token is sent with requests in headers. A filter validates the token and sets the security context.

16. What design patterns did you use?

- **Service Layer pattern**
- **DAO pattern with Repository interfaces**
- **DTO pattern for request/response bodies**

17. How did you handle exceptions globally?

Used `@ControllerAdvice` with `@ExceptionHandler` to return custom error responses.

18. How did you validate incoming requests?

Used Hibernate Validator (`@NotNull`, `@Email`, etc.) in DTOs and `@Valid` annotation in controllers.

19. Can you explain how role-based access works?

Defined roles (USER, ADMIN) and secured endpoints using `@PreAuthorize` and Spring Security configurations.

20. How do you manage sessions or stateless authentication?

JWT makes the app stateless. No session data is stored on the server. All state is managed via tokens.

🔷Database & Entity Layer

21. What database did you use and why?

Used **MySQL** for its reliability, support for ACID properties, and strong community.

22. How did you model relationships between entities?

- One-to-Many: User to Order
- Many-to-Many: Order to Product (via a CartItem join table)

23. How did you perform CRUD operations?

Used Spring Data JPA repositories (`save`, `findById`, `delete`, `findAll`).

24. How did you ensure data consistency?

Used JPA constraints, foreign keys, and `@Transactional` annotation in services.

25. Did you use JPA or JDBC Template? Why?

Used **JPA** for its abstraction and reduced boilerplate code.

🔷Security

26. How does JWT authentication flow work?

- User logs in
- Token is issued and sent to client
- Token is included in future requests
- Filter validates token and processes request

27. Where and how do you validate JWT?

A custom JWT filter checks token validity using secret keys before the request hits the controller.

28. How did you protect APIs using Spring Security?

Used configuration classes and annotations like `@PreAuthorize` and URL matchers in security config.

29. How did you manage password encryption?

Used **BCryptPasswordEncoder** to hash passwords before storing them in the database.

🔷API Testing

30. How did you test your APIs?

Used **Postman** for manual testing. Auth headers were added to test protected endpoints.

31. Can you explain how to use your Postman collection?

Import the collection, start the backend, hit the login endpoint to get a token, and use it for other requests.

32. Did you write unit/integration tests?

Yes, used JUnit and Mockito for service and controller layer testing.

🔷Deployment / DevOps

33. How did you run this project locally?

Ran React with `npm start` and Spring Boot using `mvn spring-boot:run`. Configured CORS and APIs to work on different ports.

34. Have you deployed this project online?

Yes. Frontend was deployed on Netlify; backend on Render. Environment variables were used for DB and JWT secrets.

35. What challenges did you face while deploying?

- CORS issues
- Environment variable setup
- API endpoint configuration differences (prod vs dev)

36. How do you manage environment variables?

Used `.env` files in React and application.properties in Spring Boot. Secrets were kept out of source control.

🔷Advanced / Analytical Questions

37. What would you improve with more time?

- Add payment integration (Stripe)
- Write more tests (unit + integration)
- Use Redis for caching frequent queries

38. How would you handle scalability?

- Move to microservices
- Use load balancers
- Use caching (Redis)
- Use asynchronous message queues (Kafka/RabbitMQ)

39. How would you add real-time features?

Use **WebSockets** or **Socket.IO** with a Node.js service or Spring WebSocket for live chat or order status updates.

40. How would you secure payment gateway?

- Use HTTPS
- Never store card details
- Use tokenization
- Integrate with verified payment providers like Stripe or Razorpay

41. What would you optimize for high user load?

- Optimize DB queries
- Add caching (Redis, CDN)

• Use async processing
• Load testing with JMeter or Locust

🔷Behavioral Questions

42. What challenges did you face?

• Token expiration handling
• Conditional rendering based on roles
• Managing state between frontend and backend securely

43. How did you ensure code quality?

• Used ESLint/Prettier for frontend
• Followed clean architecture in backend
• Modular and readable code with comments and documentation

44. Was this a solo or team project?

It was a solo project, where I handled all aspects from database design to deployment.

45. What part are you most proud of?

• JWT-based secure login flow
• Cart/order lifecycle with real-world logic
• Clean separation of concerns across layers