



Dr. K.N. Modi Institute of Engineering & Technology
N.H. -58 ,Modinagar,Ghaziabad (U.P.) ,201204

Affiliated to Dr. A.P.J. Abdul Kalam Technical University, Lucknow



Training- Assessment

Report

E-Commerce Product App

Session:

2025-2026

WDR-1 Major Project

SUBMITTED TO :

Mr. ANUJ KUMAR
Trainer (ANUDIP FOUNDATION)

Submitted by:

SHIKHAR HUDDA

B. Tech (C.S.E)

Roll No.: 2200770100080 ,

ARJIT TANK

B. Tech (C.S.E)

Roll No.: 2200770100026

Project Title:
E-Commerce Product App

Submitted by:	ARJIT TANK
Course:	WDR-1
Date:	29 November 2025

Submitted by:	SHIKHAR HUDDA
Course:	WDR-1
Date:	29 November 2025

1. Executive Summary

Project Overview The E-Commerce Product App is a full-stack software solution designed to demonstrate a scalable, production-ready application architecture. It integrates a responsive frontend with a robust backend to manage product displays, filtering, and shopping cart functionalities. The project leverages React and Redux for the client side, Spring Boot for the server side, and MySQL for database management.

Technical Architecture The system is built upon a clean "Service + Repository" architecture, ensuring efficient data flow and API integration:

- **Frontend:** Built with React and Bootstrap for a responsive user interface. It utilizes Redux for state management (specifically for the "Add to Cart" feature) and implements skeleton loading UIs for enhanced user experience.
- **Backend:** Powered by a Spring Boot REST API with CORS enabled. It manages data retrieval via specific endpoints (e.g., fetching all products, specific IDs, or filtering by category).
- **Database:** A MySQL database serves as the source of truth for product inventory.

Operational Workflow The application follows a seamless four-step operational flow:

1. **Request:** The React frontend initiates data requests.
2. **Processing:** The Spring Boot backend processes these requests and queries the MySQL database.
3. **Display:** The frontend renders the retrieved product data and recommendation logic (similar products).
4. **Action:** Users interact with the interface to add items to their dynamic cart.

2. Research Questions Explored (Exploratory Approach)

Question We Answered Visually

1. Research Questions Explored (Exploratory Approach)

#	Question	We Answered Visually
1	How is full-stack integration achieved?	Code Snippets: showing useEffect in React calling Spring Boot endpoints. ¹¹¹¹
2	How is the backend data structured?	Database Schema: Screenshot of the MySQL desc products table. ²
3	How are API endpoints exposed?	Controller Code: The ApplicationController class defining @GetMapping routes. ³³
4	Can the UI handle dynamic content?	UI Screenshots: The "Latest Products" grid and responsive design. ⁴
5	How is the shopping cart state managed?	Cart Interface: The "Your Cart is Empty" / dynamic cart screen (Redux). ⁵
6	How does data flow between layers?	Architecture Logic: The connection between Service, Repository, and Controller. ⁶⁶⁶⁶

3. Data Sources & Methodology

Data Sources

The application relies on a structured relational database system as its primary source of truth.

- Database System: MySQL.
- Data Structure: The data is organized in a Products table containing the following schema fields: id, category, description, image, price, rating, and title.
- Data Access Point: Data is exposed to the frontend via a custom-built Spring Boot REST API , serving JSON responses through specific endpoints like /products/all and /products/{id}.

Methodology

The project follows a full-stack development methodology utilizing a clean separation of concerns:

- Architectural Design:
 - Backend: Implements a Service + Repository architecture. The ApplicationController handles HTTP requests, the AppService contains business logic, and the AppRepository interfaces directly with the MySQL database.
 - Frontend: Built with React and Bootstrap for a responsive UI.
- Implementation Workflow: The application follows a four-step data flow methodology:
 1. Fetch: The React frontend initiates asynchronous calls using useEffect to the Spring Boot API.
 2. Query: Spring Boot processes the request and retrieves data from MySQL.
 3. Render: The frontend displays the product details and automatically filters "Similar Products" based on categories.
 4. State Management: User interactions (like adding items to the cart) are managed globally using Redux.

4. Technical Implementation – What Makes This Project Advanced

The following table outlines the advanced engineering standards applied to transform a basic web concept into a scalable, full-stack application:

Feature	Implementation Details	Why It Matters
Decoupled Architecture	React (Frontend) \rightarrow Spring Boot (Backend) via REST API	Allows independent scaling and development of client and server sides ¹ .
Clean Backend Pattern	Service + Repository Architecture (Controller \rightarrow Service \rightarrow Repository)	Ensures separation of concerns, making the code modular and testable ²²² .
Global State Management	Redux integration for Shopping Cart functionality	Manages complex state changes across the application without "prop drilling" ³³³ .
Intelligent Data Fetching	Async Chained Requests (Fetch ID \rightarrow Extract Category \rightarrow Fetch Similar)	Provides dynamic, context-aware product recommendations automatically ⁴⁴⁴ .
Security & Config	CORS Enabled (@CrossOrigin("*")) on Controllers	Enables secure cross-origin resource sharing between the frontend and backend servers ⁵⁵⁵ .
ORM Integration	JPA / Hibernate (@Entity, @Table)	Eliminates raw SQL, preventing injection attacks and streamlining database operations ⁶ .

Spring Boot fetches data from Mysql

```
9 @Table(name = "Products")
10 @Entity
11 public class Products {
12
13     @Id
14     @GeneratedValue(strategy = GenerationType.IDENTITY)
15     private Integer id;
16     private String title;
17     private String description;
18     private String image;
19     private Double price;
20     private String category;
21     private Double rating;
22
23     // Constructor
24     public Products() {
25     }
26
27     // Setter and Getter for the given above fields
28     public Integer getId() {
29         return id;
30     }
31     public void setId(Integer id) {
32         this.id = id;
33     }
34     public String getTitle() {
35         return title;
36     }
37     public void setTitle(String title) {
38         this.title = title;
39     }
40     public String getDescription() {
41         return description;
42     }
43     public void setDescription(String description) {
44         this.description = description;
45     }
46 }
```

Working(React Fetches Data from Spring Boot)

```
useEffect(() => {
    const getProduct = async () => {
        setLoading(true);
        setLoading2(true);
        const response = await fetch(`http://localhost:8080/products/cricket`);
        const data = await response.json();
        setProduct(data);
        setLoading(false);
        const response2 = await fetch(`http://localhost:8080/products/category/${data.category}`);
        const data2 = await response2.json();
        setSimilarProducts(data2);
        setLoading2(false);
    };
    getProduct();
}, [id]);
```

```
@CrossOrigin("*")
@RestController
@RequestMapping("/products")
public class AppController {

    private final AppRepository appRepository;

    @Autowired
    private AppService service;

    AppController(AppRepository appRepository) {
        this.appRepository = appRepository;
    }

    @GetMapping("/all")
    public List<Products> getall()
    {
        return appRepository.findAll();
    }

    // GET product by ID
    @GetMapping("/{id}")
    public Products getProduct(@PathVariable Integer id) {
        return service.getProduct(id);
    }

    // GET products by category
    @GetMapping("/category/{category}")
    public List<Products> getByCategory(@PathVariable String category) {
        return service.getProductByCategory(category);
    }

    // ADD new product
    @PostMapping("/addnew")
    public String addProduct(@RequestBody Products product) {
        service.addProduct(product);
        return "Product Added";
    }
}
```

Entity FILE

```
19 @Table(name = "Products")
20 @Entity
21 public class Products {
22
23     @Id
24     @GeneratedValue(strategy = GenerationType.IDENTITY)
25     private Integer id;
26     private String title;
27     private String description;
28     private String image;
29     private Double price;
30     private String category;
31     private Double rating;
32
33     // Constructor
34     public Products() {
35     }
36
37     // Setter and Getter for the given above fields
38     public Integer getId() {
39         return id;
40     }
41     public void setId(Integer id) {
42         this.id = id;
43     }
44     public String getTitle() {
45         return title;
46     }
47     public void setTitle(String title) {
48         this.title = title;
49     }
50     public String getDescription() {
51         return description;
52     }
53     public void setDescription(String description) {
54         this.description = description;
55     }
56 }
```

5. The 8 Exploratory Visualizations (Story Flow)

The following table outlines the visual narrative of the project, tracing the journey from the User Interface down to the Database Schema:

Panel	Visualization / Screen	Key Insight Revealed
1	Home Page UI ¹	User Experience: Displays a responsive "Latest Products" grid with category filters (Men's, Women's, Electronics).
2	React Logic (Code Snippet) ²	Frontend Logic: Shows the useEffect hook initiating asynchronous fetch calls to the API.
3	Cart Interface ³	State Management: Visualizes the "Your Cart is Empty" state, powered by Redux for dynamic updates.
4	Java Entity Class (Code) ⁴⁴	Data Modeling: Defines the Products object with JPA annotations (@Entity, @Id) for ORM mapping.
5	Spring Boot Controller (Code) ⁵	API Architecture: Highlights the REST endpoints (/all, /{id}) and CORS configuration (@CrossOrigin).
6	MySQL Terminal ⁶⁶	Data Structure: Confirms the backend schema (desc products), showing columns like id, price, and image.

6. Final Output

