Problem Statement: eCommerce Website with Admin Panel

Objective:

To develop a comprehensive eCommerce web portal that allows users to browse, search, and purchase products. Additionally, an admin panel is required for managing products, images, pricing, discounts, and order processing.

Scope:

1. User Portal:

- User Registration and Authentication:

  Users should be able to sign up, log in, and log out.

  Password recovery and account verification functionalities.

- Product Browsing:

  Users should be able to browse products by categories.

  Products should have detailed descriptions, images, and prices.

- Search and Filter:

  Users should be able to search for products using keywords.

  Filters for price, category, ratings, etc.

- Product Details:

  Detailed product page with images, description, price, and user reviews.

- Shopping Cart:

  Users can add, update, or remove products from the cart.

- Checkout Process:

  Users can enter shipping information and select payment methods.

  Order summary and confirmation page.Order History:

  Users can view past orders and track current orders.

- User Reviews and Ratings:

  Users can leave reviews and ratings for purchased products.

2. Admin Panel:

- Product Management:

  Admins can add, edit, and delete products.

  Upload multiple images for products.

  Set product pricing and stock availability.

- Category Management:

  Create, edit, and delete product categories.

- Discount Management:

  Define and manage discount codes and special offers.

- Order Management:

  View and process incoming orders.

  Update order status (e.g., shipped, delivered, cancelled).

- User Management:

  View and manage user accounts.

  Reset user passwords and handle user queries.

- Analytics and Reports:

  Sales reports, product performance, and user activity.

- Settings:

  Configure website settings such as payment gateways, shipping options, and tax rates.

What are functional requirements in product development?

A functional requirement is a statement of what a product (system, subsystem, system component, device or software program) must do.

Types of functional requirements include prescriptions of (rules for):

- Operations and workflows the product must perform (i.e., the functional details of the product's features)
- Formats and validity of data to be input and output by the product
- User interface behavior
- Data integrity and data security requirements
- What the product must do to meet safety and other regulatory requirements
- How the system validates user access/authorization for use and modification of the system

Functional Requirements:

- Authentication and Authorization:

  Secure user and admin authentication with role based access control.

- Responsive Design:

  The website should be fully responsive and work seamlessly on desktop, tablet, and mobile devices.

- Payment Gateway Integration:

  Integration with popular payment gateways (e.g., Razor pay).

- Security:

  Implement SSL for secure data transmission.

  Data encryption and protection against common vulnerabilities (e.g., SQL injection, XSS).

- Performance:

  Ensure fast loading times and efficient database queries.

- Scalability:

The architecture should support future scalability to handle a growing number of users and products.

Non-Functional Requirements:

- User Experience:

  Intuitive and user-friendly interface.

- Documentation:

  Comprehensive documentation for users and admin functionalities.

- Maintenance and Support:

  Plan for regular updates, bug fixes, and customer support.

Wireframes/UI Elements for User Portal

1. Home Page:

- Header:

  Logo on the left.

  Search bar in the center.

  User account and cart icons on the right.

- Main Navigation:

  Links to different categories (e.g., Home, Shop, About Us, Contact Us).

- Banner:

  Large banner with promotional images.

- Featured Products:

  Grid layout showcasing top products.

- Footer:

  Links to Privacy Policy, Terms of Service, etc.

2. Product Listing Page:

- Header:

  Same as the home page.

- Sidebar:

  Filters for categories, price range, ratings, etc.

- Product Grid:

  Display products in a grid format with images, names, prices, and ratings.

- Pagination:

  Controls to navigate between pages of products.

3. Product Detail Page:

- Product Image:

  Large image of the product.

- Product Info:

  Product name, price, description, and available sizes/colors.

- Add to Cart:

  Button to add the product to the cart.

- Reviews:

  Section for user reviews and ratings.

4. Shopping Cart Page:

- Cart Items:

  List of products added to the cart with images, names, prices, and quantity controls.

- Order Summary:

  Total price, applied discounts, and shipping costs.

- Checkout Button:

  Proceed to checkout button.

5. Checkout Page:

- Shipping Information:

  Form to enter shipping address.

- Payment Information:

  Form to enter payment details.

- Order Summary:

  Review of items and total cost.

- Place Order Button:

  Button to complete the purchase.

Wireframes/UI Elements for Admin Panel

1. Dashboard:

- Header:

  Logo and admin profile dropdown.

- Sidebar:

  Links to different sections (e.g., Dashboard, Products, Orders, Users, Reports, Settings).

- Main Content:

  Overview of sales, recent orders, and user activity.

2. Product Management:

- Product List:

  Table listing all products with columns for name, category, price, stock, and actions (edit, delete).

- Add/Edit Product Form:

Form to enter product details (name, description, price, stock, images, category).

3. Order Management:

- Order List:

  Table listing all orders with columns for order ID, customer name, date, total amount, status, and actions (view, update status).

4. User Management:

- User List:

  Table listing all users with columns for name, email, role, and actions (edit, delete).

- Add/Edit User Form:

  Form to enter user details (name, email, password, role).

5. Reports:

- Sales Report:

  Graphical representation of sales data over time.

- Product Performance:

  Report showing top selling products, low stock products, etc.

Tools and Technologies:

- Frontend:

  HTML, CSS, JavaScript, ReactJS – Bootstrap / MUI Framework

- Backend:

  Java – Spring Boot Framework

- Database:

  MySQL

- Hosting:

  CloudJiffy

Timeline and Milestones:

- Phase 1: Planning and Design

  Requirements gathering, wireframes, and design prototypes.

- Phase 2: Development

  Setting up the development environment, frontend and backend development.

- Phase 3: Testing

  Unit testing, integration testing, and user acceptance testing.

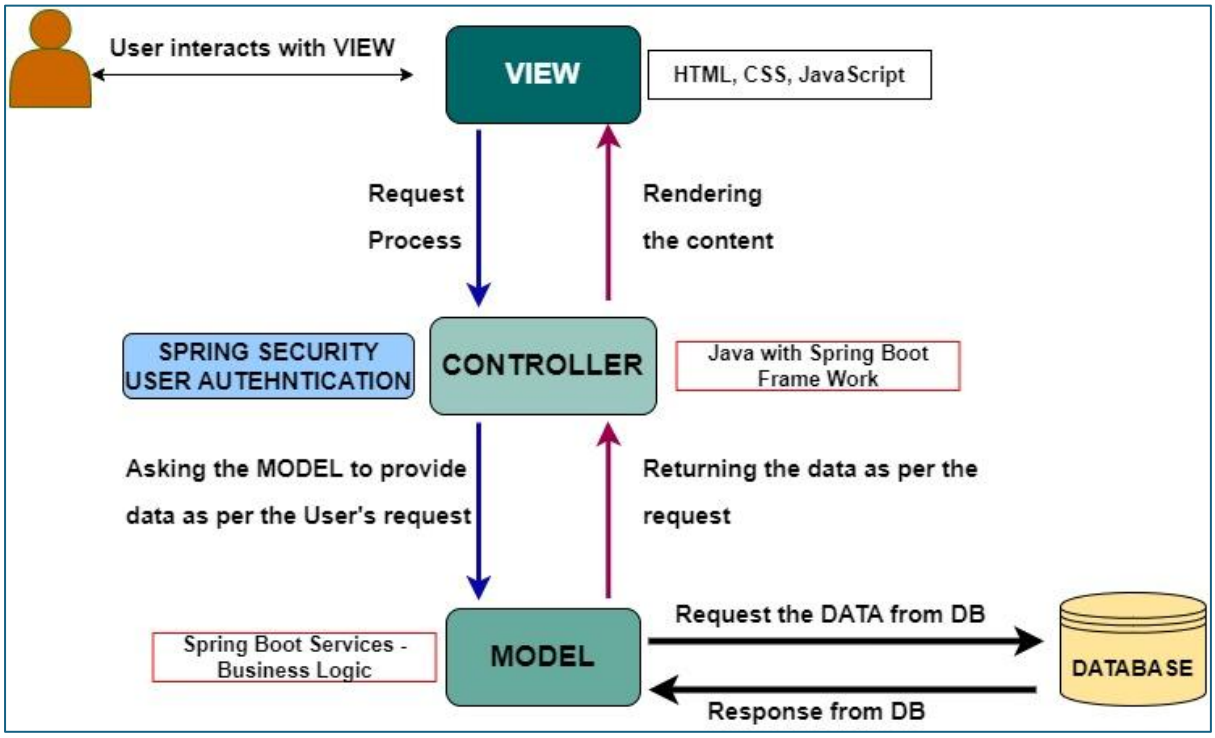
- Phase 4: Deployment

  Deploying the website to a live server.

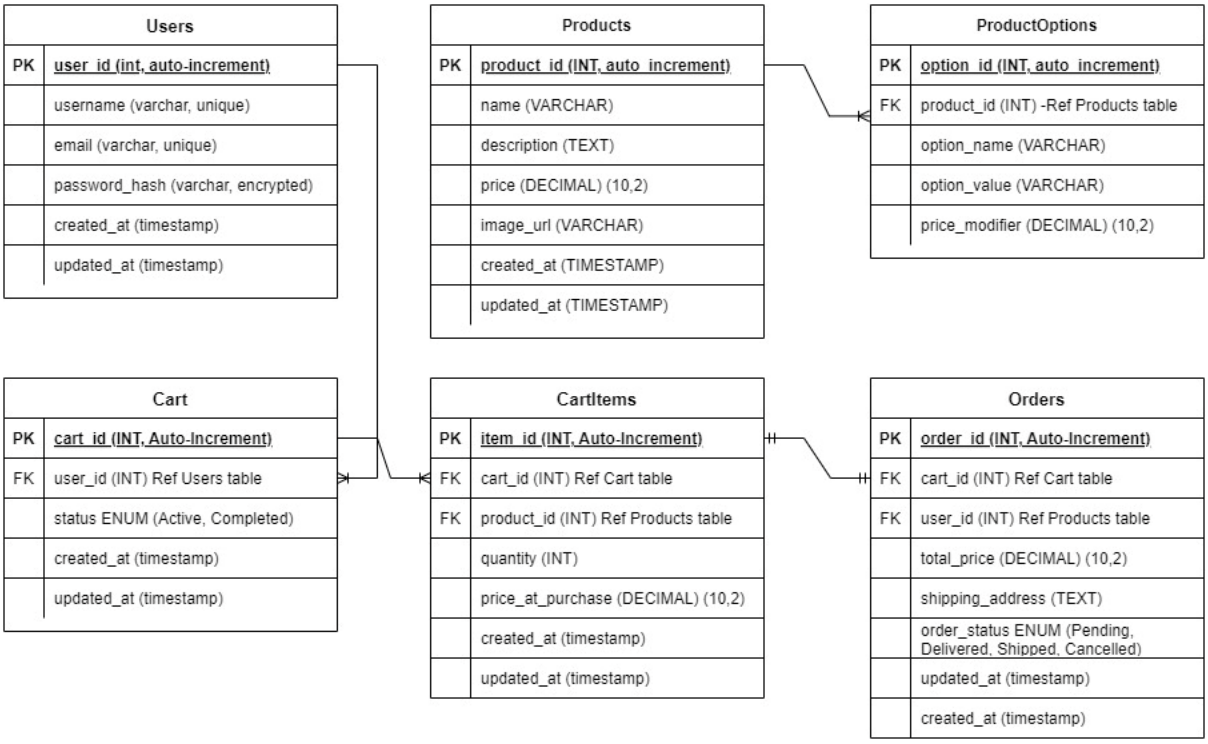
- Phase 5: Post-Deployment

  Monitoring, maintenance, and user support.

Architecture:



Database Design:



Coding Standards:

Spring Boot Coding Standards

1. Code Structure and Organization:

- Use a modular folder structure (e.g., `src/main/java`, `src/main/resources`).
- Separate business logic from configuration and utility code.
- Use meaningful package names (e.g., `com.example.myapp`).

2. Naming Conventions:

- Use camelCase for variable and method names (e.g., `customerName`, `calculateTotal`).
- Use PascalCase for class names (e.g., `CustomerService`, `OrderController`).
- Use UPPERCASE for constants (e.g., `MAX_ITEMS`).

3. Error Handling:

- Use `@ExceptionHandler` to handle exceptions globally.
- Log errors using a logging framework (e.g., Logback, SLF4J).

4. Code Style:

- Use consistent indentation (e.g., 4 spaces).
- Use braces `{}` for blocks, even for single statements.
- Use spaces around operators (e.g., `int sum = a + b;`).

5. Documentation and Comments:

- Add Javadoc comments for public methods and classes.
- Use inline comments to explain complex logic.

6. Security:

- Validate and sanitize user inputs to prevent SQL injection and XSS attacks.
- Use environment variables for sensitive information (e.g., database credentials).

Bootstrap Coding Standards

1. Code Structure and Organization:

- Use a consistent folder structure for CSS, JavaScript, and HTML files.
- Separate stylesheets, scripts, and templates.

2. Naming Conventions:

- Use camelCase for variable and function names (e.g., `myFunction`, `myVariable`).
- Use PascalCase for class names (e.g., `MyComponent`).

3. Code Style:

- Use consistent indentation (e.g., 2 spaces or 4 spaces).
- Use semicolons to terminate statements.
- Use single quotes for strings (`'example'`).

4. Documentation and Comments:

- Add comments to explain the purpose of complex functions and logic.
- Use inline comments to clarify code blocks.

5. Security:

- Validate and sanitize user inputs to prevent XSS attacks.
- Use HTTPS for secure communication.

Java Coding Standards

Code Structure and Organization:

- **Package Structure:** Organize classes into meaningful packages.

  E.g., `com.example.myapp.controllers`, `com.example.myapp.models`.

- **Class Organization:** Follow a consistent order within a class: class variables, constructors, methods.

Naming Conventions:

- **Classes and Interfaces:** Use PascalCase for class and interface names. E.g., `CustomerService`, `OrderController`.
- **Methods and Variables:** Use camelCase for method and variable names. E.g., `calculateTotal`, `orderDate`.
- **Constants:** Use UPPERCASE with underscores for constants. E.g., `MAX_ITEMS`, `DEFAULT_TIMEOUT`.

Code Style:

- **Indentation:** Use 4 spaces for indentation. Avoid using tabs.
- **Braces:** Use braces `{}` for all control structures, even if they contain a single statement.
- **Whitespace:** Use single spaces around operators (e.g., `int sum = a + b;`).

Documentation and Comments:

- **Javadoc Comments:** Use Javadoc comments for all public methods and classes.
- **Inline Comments:** Use inline comments to explain complex logic or important code sections.
- **TODO Comments:** Use `// TODO` comments to mark unfinished code that needs attention.

Error Handling:

- **Exceptions:** Use specific exception types and provide meaningful error messages.
- **Logging:** Log exceptions using a logging framework (e.g., SLF4J, Logback).

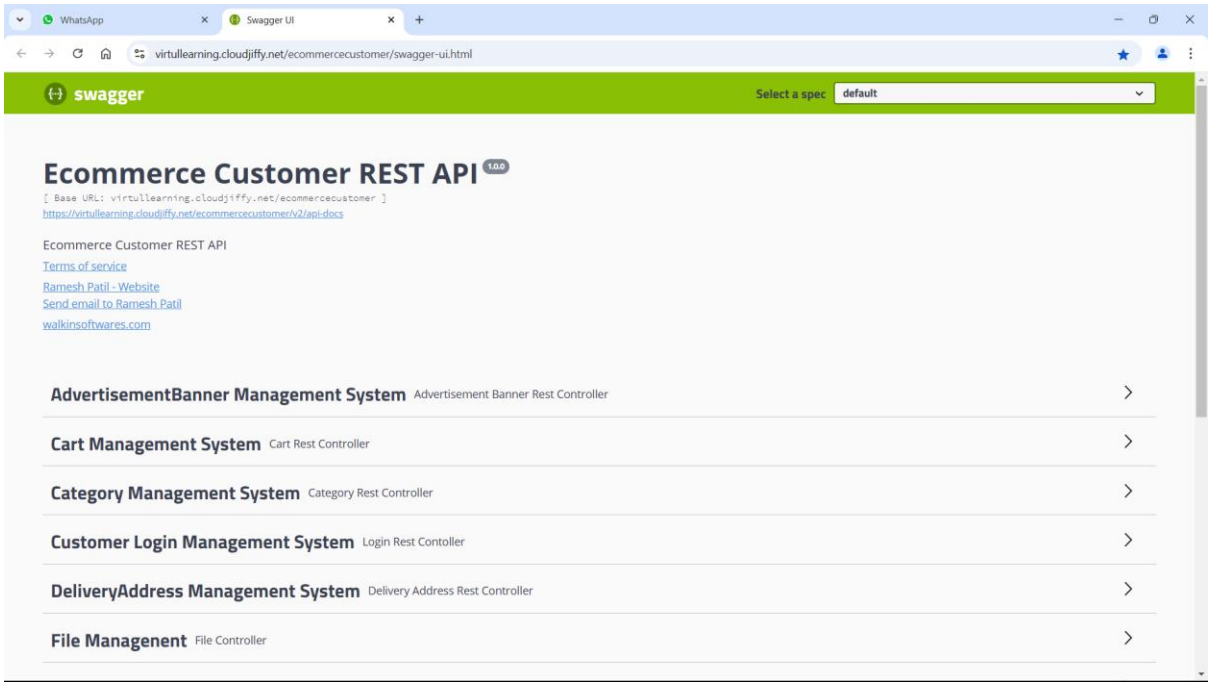
Security:

- **Input Validation:** Validate and sanitize all user inputs to prevent security vulnerabilities.
- **Sensitive Data:** Avoid hardcoding sensitive information (e.g., passwords, keys). Use environment variables or secure vaults.

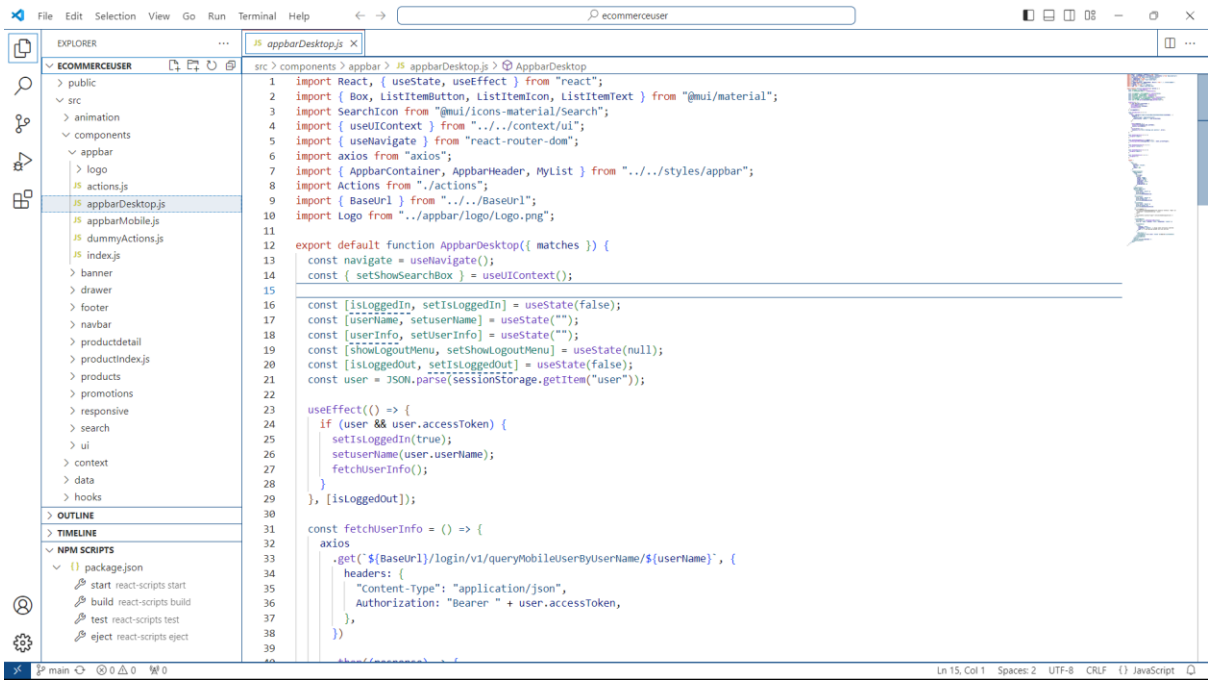
Performance:

- **Efficient Data Structures:** Use appropriate data structures for optimal performance.
- **Minimize Object Creation:** Reuse objects where possible to reduce memory overhead.

API Documentation:



Version Control:



Testing:

Brief About Testing

Testing is the process of evaluating a system or its components to check whether it meets the specified requirements. It involves executing a system in order to identify any gaps, errors, or missing requirements in contrast to the actual requirements. The primary objective of testing is to ensure the software quality by finding and fixing bugs.

Why Do We Write Test Cases?

1. To have better test coverage.
2. To have better consistency in test execution.
3. To avoid training new testers on the product or requirement.
4. To avoid missing scenarios and defects.
5. Test cases show developers and customers that we have tested the application for all possible scenarios.
6. Test cases are the basis for automation.
7. Test cases can help to do testing in an organized way

Writing Test Cases

A **test case** is a set of conditions or variables under which a tester determines whether a system or one of its features is working as it was originally established for it to do. Writing test cases involves several steps:

1. **Understand Requirements:** Fully understand the requirements and functionality of the system to be tested.
2. **Define Test Objective:** Clearly outline the purpose of the test and what it aims to achieve.
3. **Identify Test Conditions:** Determine the conditions under which the test is to be performed.
4. **Specify Input Data:** Define the data that will be input during the test.
5. **Describe the Steps:** Document the steps to be followed during the test execution.
6. **Expected Result:** Clearly state the expected outcome of the test.

Test Case Template with Example Test Case

Test Case ID:	TC001
Test Case Title:	Verify LOGIN functionality
Description:	This Test case verifies that the user can successfully login with valid Username and Password
Pre-conditions:	User should have valid Username and Password
Test Steps:	
1.	Go to the LOGIN page
2.	Enter valid Username & Password in respective fields
3.	Click on LOGIN button
Test Data:	Valid Username and Password
Expected Result:	User should be logged in to the application & redirected to dashboard page
Actual Result:	Login successful and redirected to User's dashboard
Status (Pass/Fail):	Pass
Comments:	*Username and Password had different fonts.*

User Documentation

User Manuals: Guide users on how to use the software.

Installation Guides: Provide instructions for installing the software.

FAQ: Include a list of frequently asked questions and answers.

Maintenance Documentation

Deployment Guides: Detail the deployment process and environment setup.

Maintenance Plans: Outline the procedures for maintaining and updating the software.

Troubleshooting Guides: Provide solutions to common problems.

Release Notes

Summarize new features, bug fixes, and any known issues in each release.

Legal and Compliance Documentation

- Include licenses, terms of use, and compliance with relevant regulations.