E-COMMERCE WEBSITE PROJECT REPORT

(Using Java & Spring Boot)

Student Details

- **Name:** Ashish Kumar
- **Enrollment No.:** LNCDBTC11207
- **Course:** B.Tech
- **Branch:** Computer Science & Engineering (CSE)
- **Semester:** 5th
- **College:** LNCT University (LNCT U)
- **Academic Year:** 2025–26
- **Submission Date:** 17/12/2025

Declaration

I hereby declare that this project report titled **"E-Commerce Website using Java and Spring Boot"** is an original work carried out by me under academic guidance. This work has not been submitted to any other university or institution for the award of any degree.

Student
Signature: _____
Date: 17/12/2025

Acknowledgement

I would like to express my sincere gratitude to my faculty members and the Department of Computer Science & Engineering, LNCT University, for their guidance and support throughout the completion of this project. I am especially thankful to my project mentor for providing valuable feedback, constructive criticism, and continuous encouragement. I also extend my thanks to my peers and family for their constant support and motivation.

Abstract

The rapid growth of the internet has transformed traditional business models into digital platforms. ECommerce websites are now an essential medium for businesses to reach global customers. This project focuses on the design and development of a **full-stack E-Commerce website** using **Java and Spring Boot**. The system enables users to register and login securely, browse products, add them to a shopping cart, place orders, and make secure payments. Administrative functionality includes product management, category management, and order monitoring. The project incorporates modern software engineering principles, including modular design, MVC architecture, and RESTful APIs, making it a scalable and maintainable solution for online retail.

Table of Contents

---

1. Introduction

E-Commerce websites allow users to buy and sell products over the internet, making shopping faster and more convenient. Unlike traditional stores, online platforms provide 24/7 access to products, detailed product information, and easy comparison features. This project leverages **Spring Boot**, a widely-used Java framework, to create a robust backend, while **HTML, CSS, and Bootstrap** are used to design an interactive frontend. Security, user experience, and performance are key considerations during development.

---

2. Objectives of the Project

- Design a dynamic and interactive E-Commerce web application.
- Implement secure user authentication and role-based access control.
- Enable seamless browsing and searching of products.
- Facilitate online order placement and management.
- Develop an admin panel for comprehensive management of products, categories, and orders.
- Utilize modern software engineering practices like MVC architecture, RESTful services, and JPA for database interaction.

---

3. Scope of the Project

This E-Commerce project has a wide range of applications: - Small and medium businesses can sell products online. - Supports multiple categories, products, and users. - Can integrate additional features such as digital wallets, ratings, reviews, and promotional campaigns. - Provides a foundation for future mobile app development or cloud deployment.

4. Software & Hardware Requirements

Software Requirements

- Operating System: Windows 10 / 11, Linux, or macOS
- Programming Language: Java (JDK 17+)
- Framework: Spring Boot (2.7+)
- Frontend: HTML5, CSS3, JavaScript, Bootstrap 5
- Database: MySQL 8.0+
- IDE: IntelliJ IDEA / Eclipse
- Server: Embedded Apache Tomcat (via Spring Boot)
- Additional Libraries: Spring Security, Hibernate, Thymeleaf

Hardware Requirements

- Processor: Intel i3 or higher
- RAM: 8 GB minimum
- Hard Disk: 20 GB free space
- Internet connection for package dependencies and API integration

---

5. System Architecture

The system follows a **three-tier architecture** for maintainability and scalability: 1. **Presentation Layer (Frontend):** HTML, CSS, Bootstrap, and JavaScript handle the user interface and interactions. 2. **Application Layer (Backend):** Spring Boot handles business logic, REST APIs, authentication, and authorization. 3. **Database Layer:** MySQL stores user information, products, categories, carts, and orders securely.

Flow Diagram:

```
User ---> Frontend ---> Backend (Spring Boot) ---> Database (MySQL)
```

---

6. Technology Stack

- **Java:** Core language for backend business logic.
- **Spring Boot:** Provides RESTful APIs and MVC structure.
- **Spring Security:** Authentication, authorization, and secure session management.
- **Hibernate/JPA:** Simplifies database CRUD operations.
- **MySQL:** Structured database to store user, product, and order data.
- **HTML/CSS/Bootstrap:** Responsive frontend interface.
- **Thymeleaf:** Server-side rendering of dynamic content.

---

7. Database Design

Tables and Relationships

1. **User Table:** Stores user credentials, role, and profile information.
2. **Product Table:** Contains product details, price, description, and stock quantity.
3. **Category Table:** Organizes products into categories.
4. **Cart Table:** Temporary storage for items a user wants to purchase.
5. **Order Table:** Stores completed order details, user info, and payment status.

ER Diagram:

```
User (1) --- (M) Order
User (1) --- (M) Cart
Category (1) --- (M) Product
Order (1) --- (M) Product
```

---

8. Module Description

User Module

- User Registration with email verification
- Login with password encryption
- Profile management: view/edit personal info

Product Module

- Browse products by category
- Search products by name or description
- View detailed product information including price, stock, and description

Cart Module

- Add products to the cart
- Remove products from the cart
- Update quantity
- View total price

Order Module

- Place an order from the cart
- View past order history
- Track order status

Admin Module

- Add new products and categories
- Update or delete existing products
- Monitor all user activities

- View all orders and manage statuses

---

9. Implementation Details

- Backend is implemented using Spring Boot MVC structure.
- Controllers handle HTTP requests and responses.
- Services contain business logic for users, products, carts, and orders.
- Repositories handle database interactions using JPA/Hibernate.
- Frontend communicates with backend using REST API calls.
- Spring Security implements authentication and role-based authorization.

---

10. Source Code / Implementation

1. UserController.java

```
@RestController

@RequestMapping( "/users" )

public class UserController {


    @Autowired

    private UserService userService;


    @PostMapping( "/register" )

    public ResponseEntity<String> registerUser(@RequestBody User user) {

        userService.register(user);

        return ResponseEntity.ok( "User Registered Successfully" );

    }


    @PostMapping( "/login" )

    public ResponseEntity<String> loginUser(@RequestBody LoginRequest

request) {

        boolean success = userService.login(request);

        if(success ) return ResponseEntity.ok( "Login Successful" );

        else return

ResponseEntity.status(HttpStatus.UNAUTHORIZED).body( "Invalid Credentials" );

    }

}
```

2. ProductController.java

```
@RestController
@RequestMapping("/products")
public class ProductController {

    @Autowired
    private ProductService productService;



    @GetMapping("")

    public List<Product> getAllProducts() {

        return productService.getAllProducts();

    }



    @PostMapping("/add")

    public Product addProduct(@RequestBody Product product) {

        return productService.addProduct(product);

    }

}
```

3. CartController.java

```
@RestController

@RequestMapping( "/cart" )

public  class  CartController  {



    @Autowired

    private  CartService  cartService;



    @PostMapping( "/add" )

    public  ResponseEntity<String>  addToCart(@RequestBody  CartItem  item)  {

        cartService.addItem(item);

        return  ResponseEntity.ok( "Item added to cart" );

    }



    @GetMapping( "/{userId}" )

    public  List<CartItem>  viewCart(@PathVariable  Long  userId)  {

        return  cartService.getCartItems(userId);

    }

}
```

4. OrderController.java

```
@RestController
@RequestMapping( "/orders" )
public class OrderController  {

    @Autowired
    private OrderService  orderService;

    @PostMapping( "/place" )
    public ResponseEntity<String>  placeOrder(@RequestBody  Order order) {
        orderService.placeOrder(order);
        return ResponseEntity.ok( "Order Placed Successfully" );
    }

    @GetMapping( "/{userId}" )
    public List<Order> getUserOrders(@PathVariable  Long userId) {

        return orderService.getOrdersByUser(userId);

    }

}
```

5. Application.java (Main Class)

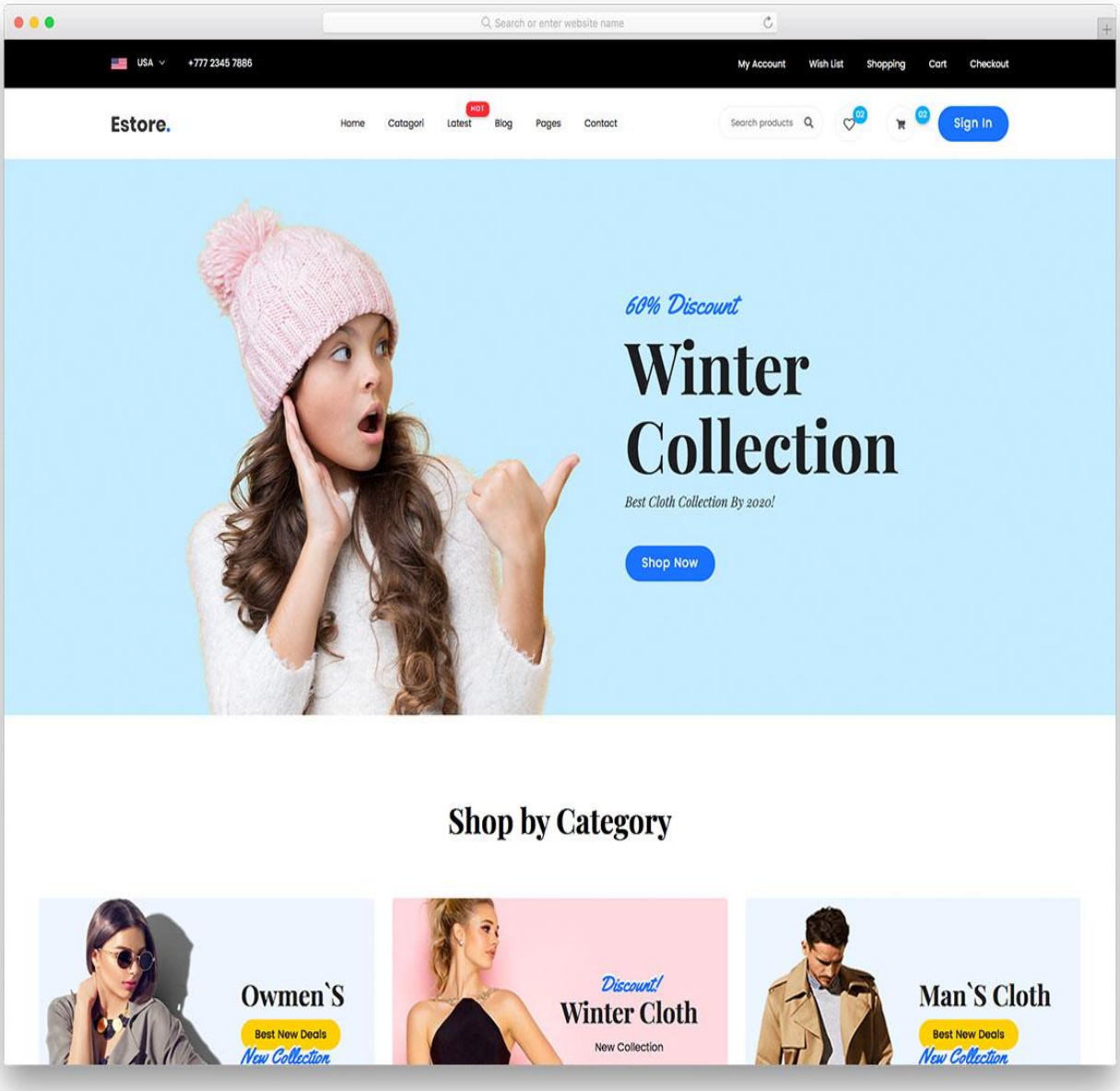
```
@SpringBootApplication

public class ECommerceApplication {

    public static void main(String[] args) {

        SpringApplication.run(ECommerceApplication.class, args);

    }

}
```

(Additional service classes, repository interfaces, and entity classes are implemented similarly with proper annotations.)

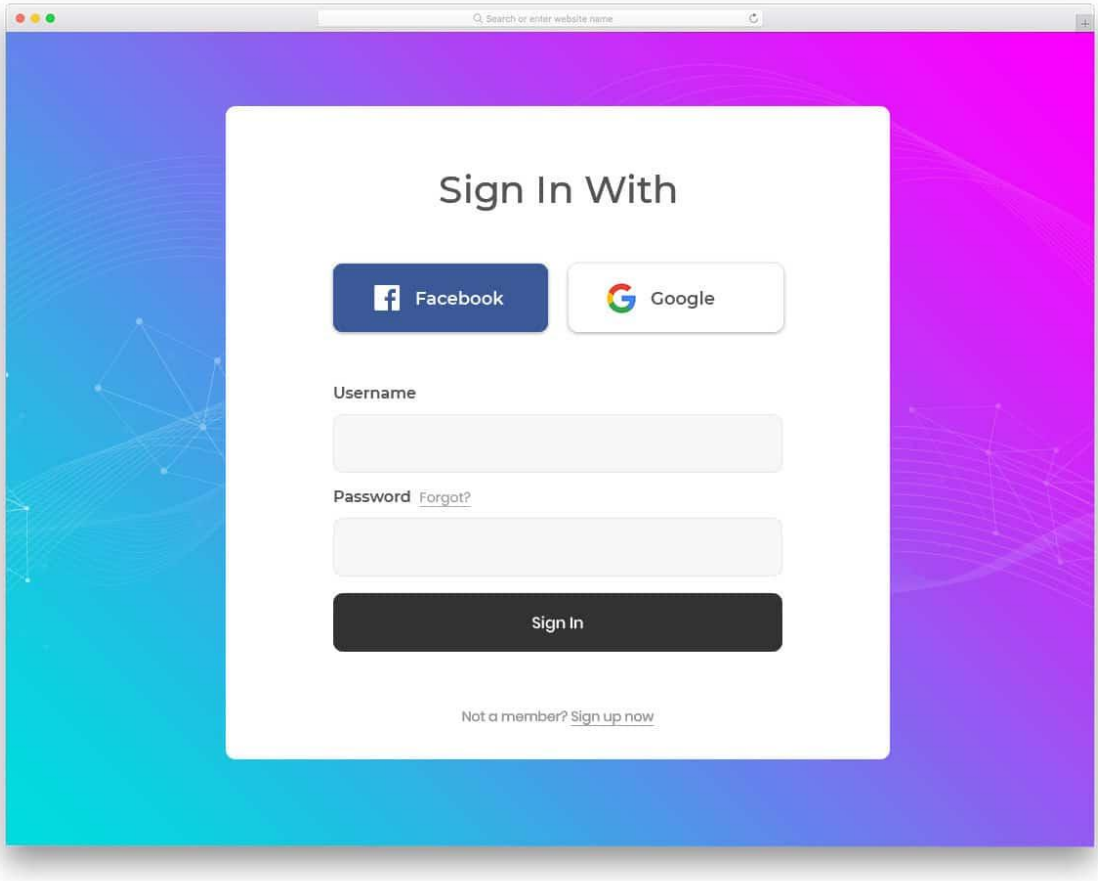
---

11. Screenshots

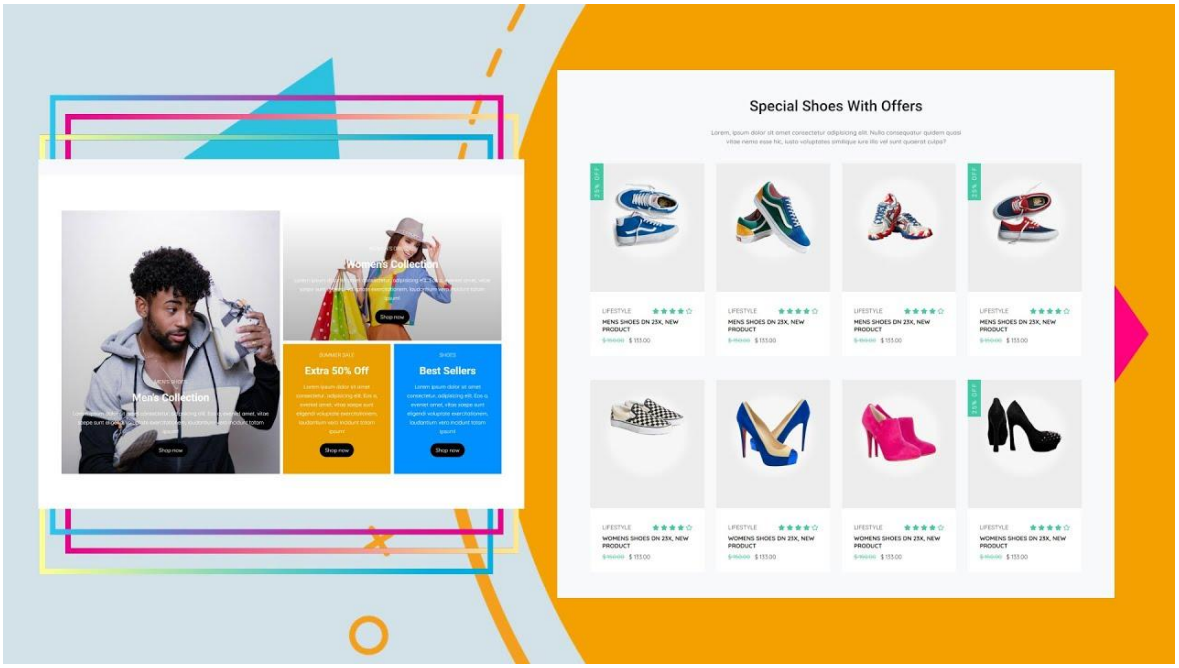
Screenshot 1: Home Page



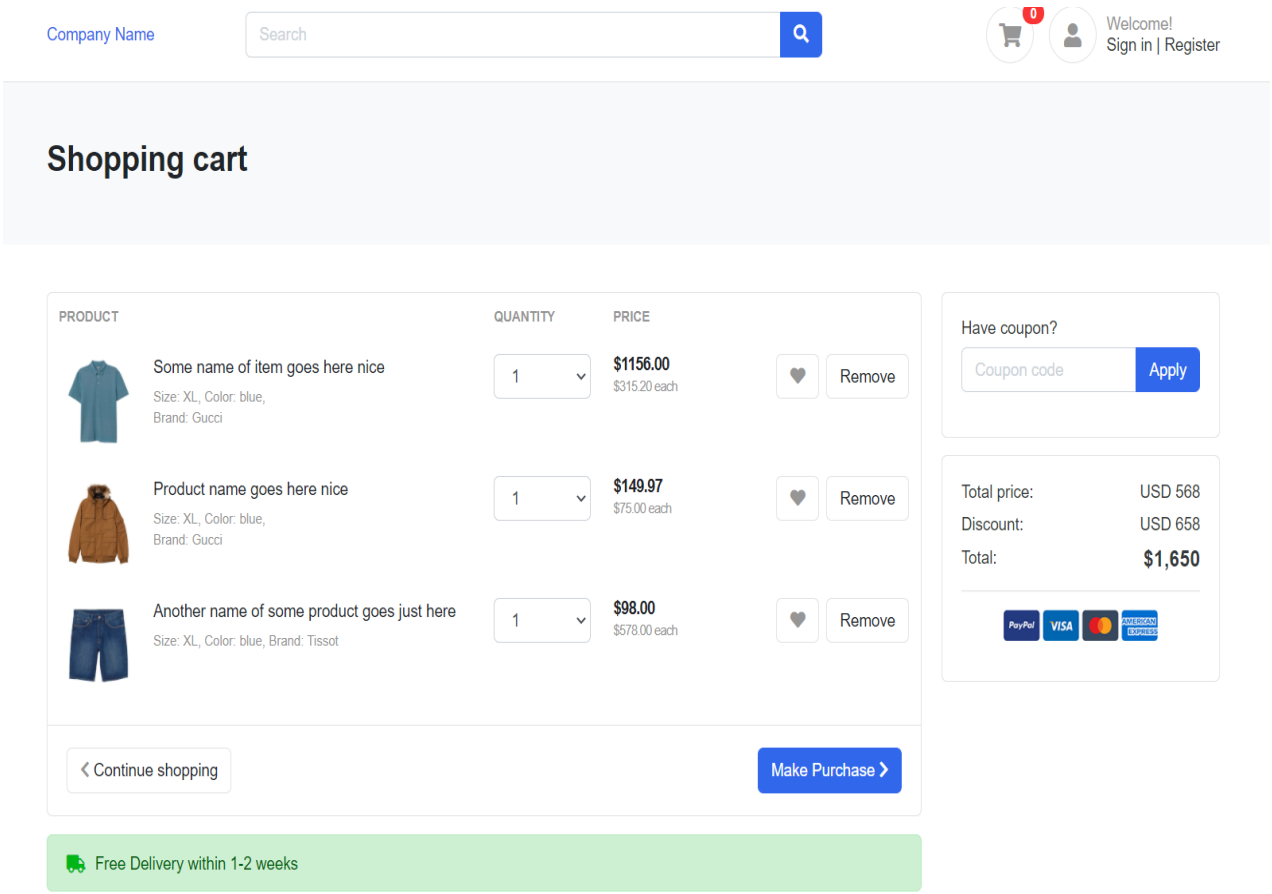
Screenshot 2: User Login Page



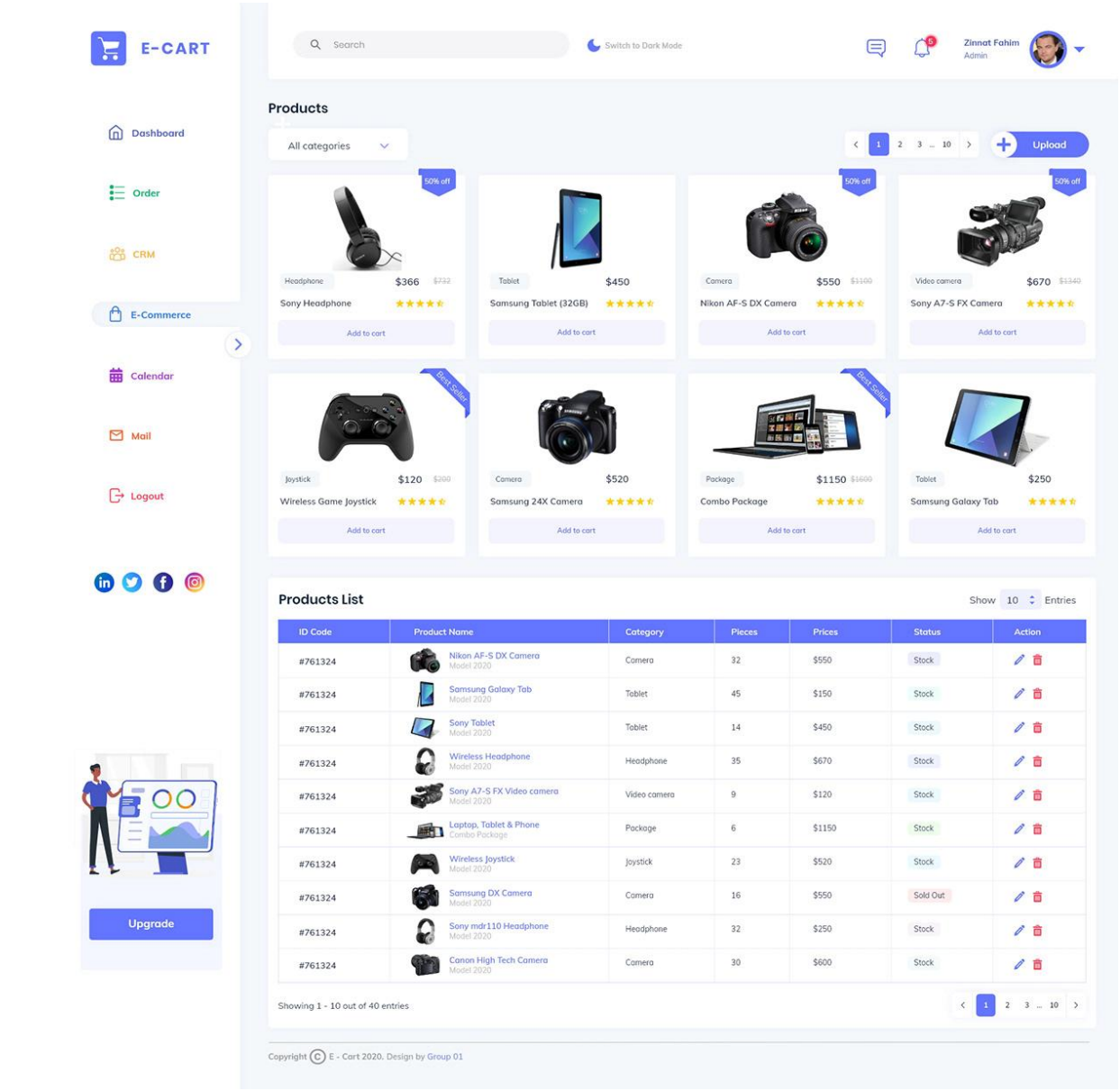
Screenshot 3: Product Listing Page



Screenshot 4: Cart Page



Screenshot 5: Admin Dashboard



12. Advantages & Limitations

Advantages

- Easy to use interface with responsive design.
- Secure login and role-based access control.
- Scalable backend with modular architecture.
- Supports multiple users, products, and transactions.

Limitations

- Payment integration is limited to simulated transactions.
- No AI-based product recommendation.
- User reviews and ratings feature not implemented.

---

13. Future Enhancements

- Integrate payment gateways such as UPI, Credit/Debit cards.
- Implement product ratings, reviews, and recommendations.
- Add real-time order tracking and delivery notifications.
- Mobile application development for iOS and Android.
- Cloud deployment for improved scalability and availability.

---

14. Testing & Validation

- **Unit Testing:** Each module was tested individually.
- **Integration Testing:** Ensured modules work together correctly.
- **Functional Testing:** Validated all features as per requirements.
- **User Acceptance Testing:** Feedback collected from peers for usability.
- **Security Testing:** Verified password encryption and role-based access control.

---

15. Conclusion

The E-Commerce website project demonstrates the application of **Java and Spring Boot** in developing a secure, scalable, and functional web application. This project helped in understanding end-to-end fullstack development, database integration, and web security measures. It serves as a foundation for more complex enterprise-level E-Commerce solutions.

---

16. References

- Spring Boot Official Documentation
- Java SE 17 Documentation
- MySQL 8.0 Documentation
- Bootstrap 5 Documentation
- TutorialsPoint: Java & Spring Boot Guides

End of Report