# E-COMMERCE SITE

**CS23333-OBJECT ORIENTED PROGRAMMING USING JAVA**

*Submitted by*

KABIL RS- 231001081

KISHORE JM - 231001095

*Of*

**BACHELOR OF TECHNOLOGY IN**

**INFORMATION TECHNOLOGY**

**RAJALAKSHMI ENGINEERING COLLEGE, THANDALAM**
**(An Autonomous Institution)**


**DEPARTMENT OF INFORMATION TECHNOLOGY**

**RAJALAKSHMI ENGINEERING COLLEGE**

**THANDALAM , CHENNAI - 600025**

**NOVEMBER 2024**

## BONAFIDECERTIFICATE

Certified that this project titled "E-Commerce Site" is the Bonafide work of "**KABIL R S. (231001081)"," KISHORE J M .(231001095)"** who carried out the project work under my supervision.


SIGNATURE

Dr. P. Valarmathie

HEAD OF THE DEPARTMENT

Information Technology

Rajalakshmi Engineering College,
Rajalakshmi Nagar, Thandalam
Chennai – 602105

SIGNATURE

Mrs.T Sangeetha

COURSE INCHARGE

Information Technology

Rajalakshmi Engineering College
Rajalakshmi Nagar, Thandalam
Chennai – 602105


This project is submitted for IT19341 – Introduction to Oops and Java held on

_____

**INTERNAL EXAMINER** **EXTERNAL EXAMINER**

# TABLE OF CONTENTS

# E-COMMERCE SITE

## 1.1 Abstract

This E-Commerce Platform is a comprehensive solution designed to streamline online retail operations using modern technologies such as React, Java, and MongoDB. The system features a user-friendly interface built with React, providing an intuitive shopping experience for customers. The backend, powered by Java, handles all business logic and integrates with MongoDB to efficiently manage and store product data, user information, and order details.

Key features include secure user authentication, allowing customers to create accounts, log in, and manage their profiles. The platform supports a robust catalog management system, enabling administrators to add, edit, and delete products. Additionally, the shopping cart functionality ensures seamless order placement and tracking, enhancing the overall user experience.

## 1.2 Introduction

The E-Commerce Platform is a robust and versatile solution tailored for retail businesses, leveraging cutting-edge technologies such as React, Java, and MongoDB. This platform delivers an exceptional user experience through a responsive and intuitive interface developed with React, ensuring seamless navigation and interaction for customers. Java powers the backend, orchestrating the business logic and facilitating efficient communication with MongoDB to manage vast amounts of product data, user profiles, and order information.

The system prioritizes secure user authentication, enabling customers to register, log in, and maintain their profiles with confidence. Administrators benefit from a comprehensive catalog management system, which allows them to effortlessly add, update, and remove products. The integrated shopping cart functionality ensures smooth and reliable order processing, enhancing the overall satisfaction for both customers and administrators.

## 1.3 Purpose

The purpose of this project is to develop a robust and user-friendly e-commerce platform that benefits both customers and retailers. The system aims to:

- **Streamline Online Retail Operations:** Provide an intuitive and efficient interface for customers to browse, select, and purchase products.
- **Manage Comprehensive Product Information:** Enable retailers to manage detailed product listings, including names, descriptions, prices, categories, and images.
- **Ensure Secure Transactions:** Implement secure authentication and payment processing mechanisms to protect user data and transaction details.
- **Enhance User Experience:** Offer a seamless shopping experience with features like a dynamic shopping cart, order tracking, and personalized recommendations.
- **Support Retailer Operations:** Facilitate easy management of inventory, orders, and customer data to improve operational efficiency and accuracy.
- **Promote Scalability and Reliability:** Utilize modern technologies such as React, Java, and MongoDB to ensure the platform can handle growth and provide a reliable service.

## 1.4 Scope of the Project

The scope of the E-Commerce Platform encompasses a comprehensive range of functionalities designed to enhance the online retail experience for both customers and retailers. The system aims to streamline the processes of browsing, selecting, and purchasing products, while also providing robust tools for managing inventory, orders, and customer data.

A core component of this project is its integration with MongoDB using Java,JDBC ensuring efficient and secure interaction between the application and the database. The platform features a user-friendly interface built with React, which allows for seamless navigation and a smooth shopping experience.

## 1.5 Software Requirement Specification

**1. Database Integration:**

- MongoDB Database: The system uses MongoDB to store product data, user information, and order details. This database is connected using Java Database Connectivity (JDBC), ensuring efficient and secure interaction between the application and the database.

- Schema Design: The database schema includes collections for products, users, and orders with fields for relevant details such as product names, descriptions, prices, user information, and order statuses.

**2. Backend Development:**

- **Server Setup:** The backend server is implemented using Java with Spring Boot, handling various endpoints for different functionalities.
- **Endpoints:**
  - **/add-to-cart:** Handles the insertion of product records.
  - **/update-cart:** Manages updates to existing product records.
  - **/products:** Retrieves and displays product records.
  - **/login:** Manages user authentication for accessing the system.

**3. Frontend Development:**

- **React:** The frontend is developed using React, providing a responsive and interactive interface for managing product information and facilitating customer interactions.
- **CSS:** Utilizes Tailwind-CSS for styling, ensuring an attractive and user-friendly interface.
- **JavaScript:** Enhances the frontend with dynamic updates and interactive elements, handling form submissions and user interactions.

**4. Functionality:**

- **Data Input:** Administrators can input product details through web forms.
- **Data Management:** The system supports adding new products, editing existing information, and viewing product details.
- **User Authentication:** Secure login mechanism ensures that only authorized personnel can access the system.
- **Shopping Cart:** Customers can add, remove, and update products in their shopping cart before checkout.
- **Order Processing:** Efficient order management, tracking, and history features for both customers and administrators.

**5. Technology Stack:**

- **Frontend:** React, CSS, JavaScript
- **Backend:** Java, Spring Boot, JDBC
- **Database:** MongoDB

- **Tools and Libraries:** JSON handling libraries for data exchange, Spring Boot for handling HTTP requests and responses.

## Overall Description

### Product Perspective

The system is built using a client/server architecture, compatible with various operating systems. The frontend is developed with React, CSS, and JavaScript, providing a dynamic and responsive user interface. The backend is powered by Java with Spring Boot for handling HTTP requests, and MongoDB for data storage.

### Product Functionality

a) **User Registration and Authentication:**

- **Sign-Up:** Allows new users to create an account by providing their email and password, securely storing hashed passwords in the database.
- **Login:** Enables users to log in with their credentials, verifying them against stored records to ensure secure access.

b) **Product Management:**

- **Insert Product Data:** Administrators can add new product records, including details such as name, description, price, category, and image URL.
- **Update Product Data:** Facilitates the modification of existing product records to keep the information accurate and up-to-date.
- **View Product Data:** Provides a mechanism to retrieve and display detailed product information for browsing and purchasing.

c) **Shopping Cart Management:**

- **Add to Cart:** Users can add products to their shopping cart by specifying the product ID and quantity, with the system handling duplicates and updating quantities as needed.
- **Remove from Cart:** Enables users to remove specific products from their shopping cart.
- **Update Cart:** Allows users to change the quantity of products in their cart, ensuring they can manage their selections before checkout.

d) **Order Management:**

- **Retrieve Cart:** Users can view the current contents of their shopping cart, including product details and total cost.

e) **Security and Data Integrity:**

- **Password Encryption:** Utilizes BCrypt to securely hash user passwords, ensuring data security and protection against unauthorized access.
- **Validation and Error Handling:** Ensures robust validation and error handling to maintain data integrity and provide meaningful feedback to users.

## User and Characteristics

**Qualification:**

- Users should have a basic understanding of wensites and ecommerce sites.

- **Experience:**

  - Familiarity with ecommerce sites and basic data handling is beneficial.

**Technical Skills:**

- Users are expected to have elementary knowledge of computers and the ability to interact with web-based applications.

**Operating Environment:**

**Hardware Requirements:**

- **Processor**: Any processor over i3
- **Operating System**: Windows 8, 10, 11
- **Processor Speed**: 2.0 GHz
- **RAM**: 4GB
- **Hard Disk**: 500GB

**Software Requirements:**

- **Database**: MongoDB
- **Frontend**: React
- **Backend**: Java

## Constraints

- **Authorized Access:** System access is limited to authorized users, such as administrators and registered customers.
- **Delete Operation:** The delete operation for products is restricted to administrators, ensuring data integrity and security.
- **Data Consistency:** Administrators must exercise caution during deletion to ensure data consistency and prevent unintended data loss.

## Assumptions and Dependencies

- **System Administrators:** Responsible for creating and securely communicating login IDs and passwords to users.

- **User Knowledge:** Users are assumed to have basic knowledge of using web-based applications.

## Specific Requirements

### User Interface Features for the User Management System of Students Are:

- **Login**: Secure login for authorized users to maintain data security.
- **Insert Product Data**: Input and store new product information accurately and efficiently.
- **View Product Data**: Display and manage detailed product information seamlessly.
- **Update Product Data**: Modify existing product records to keep data up to date.
- **Search Product Data**: Retrieve specific product details quickly based on search criteria.
- **User Authentication**: Ensure secure management of user access to protect sensitive information.
- **Shopping Cart Management**: Add, update, and remove products in the shopping cart.
- **Order Management**: Manage orders efficiently, including tracking and history.
- **Payment Processing**: Securely handle transactions and ensure data security.

### Hardware Interface:

- **Screen Resolution**: Minimum screen resolution of 1024 x 768 or above.
- **Compatibility:** Compatible with modern web browsers on various operating systems, including Windows, macOS, and Linux.

**Software Interface for Student Mark Analysis**

- **Operating System**: Compatible with modern web browsers on various operating systems, including Windows (8, 10, 11), macOS, and Linux.
- **Frontend Development:** React, CSS, JavaScript.
- **Backend Development**: Java with Spring Boot.
- **Database**: MongoDB.

## Functional Requirements for User Management System of Students

- **Login Module (LM):**

  - Users (customers and admins) can access the Login Module via a secure login page.
  - The system supports login using an email and password, with passwords encrypted for security.
  - Only authorized users, whose credentials match those in the database, are granted access.

**Registered Users Module (RUM):**

- After successful login, users (customers) are granted access to the main features.
- Users can view detailed information about products and their orders.
- Customers can add products to their cart, place orders, and manage their profiles.

**Administrator Module (AM):**

- The system displays administrative functions after a successful login for admins.
- Admins can manage product data: adding new products, updating existing product records, and deleting products.

- The "Add" function allows admins to input new product details, while the "Update" function allows modifications to existing information.
- All add, update, or delete actions trigger communication with the backend (via the Server Module) to make necessary changes in the database.

**Shopping Cart Module (SCM):**

- Allows users to add products to their cart, update quantities, and remove items.
- The system calculates the total cost of the items in the cart.
- Users can proceed to checkout to place their orders.

**Order Management Module (OMM):**

- Manages order processing, tracking, and history.
- Admins can view, update, and manage orders placed by customers.

**Payment Processing Module (PPM):**

- Integrates with secure payment gateways to handle transactions.
- Ensures secure handling of payment data and transaction details.

**Server Module (SM):**

- Acts as an intermediary between the frontend modules and the database.
- Receives and processes requests from various modules, ensures proper data formatting, and manages the system's functionality.
- Handles communication with the database to validate and execute requests, ensuring data consistency and integrity**.**

## Non-functional Requirements

### Performance:

- **Efficient Data Handling:** The system must efficiently handle product data and user requests, ensuring that actions like adding to cart, updating product details, and processing orders are completed in under 2 seconds.
- **Concurrency Management:** The system should handle a large number of concurrent requests for browsing products, adding items to the cart,

and completing transactions without significant delays, ensuring high responsiveness for users.

Reliability:

- **Robust System:** The system must be robust and capable of recovering gracefully from failures. In case of data corruption or abnormal shutdown, the system should provide mechanisms for data recovery and **ensure minimal data loss.**
- **Thorough Testing:** The system should be thoroughly tested to handle edge cases such as incorrect or missing data entries, ensuring smooth operation under various conditions.

Availability:

- **24/7 Availability**: The system should be available 24/7 for users to browse products, place orders, and manage their accounts.
- **High Availability:** It should maintain high availability and perform critical operations, like processing payments and updating orders, with minimal downtime or service interruptions.

Security:

- **High Maintainability:** The system should be designed to be easily maintainable, with clear documentation and modular code to allow for efficient updates and troubleshooting.

# SYSTEM FLOW DIAGRAMS

## 2.1 Use Case Diagrams :

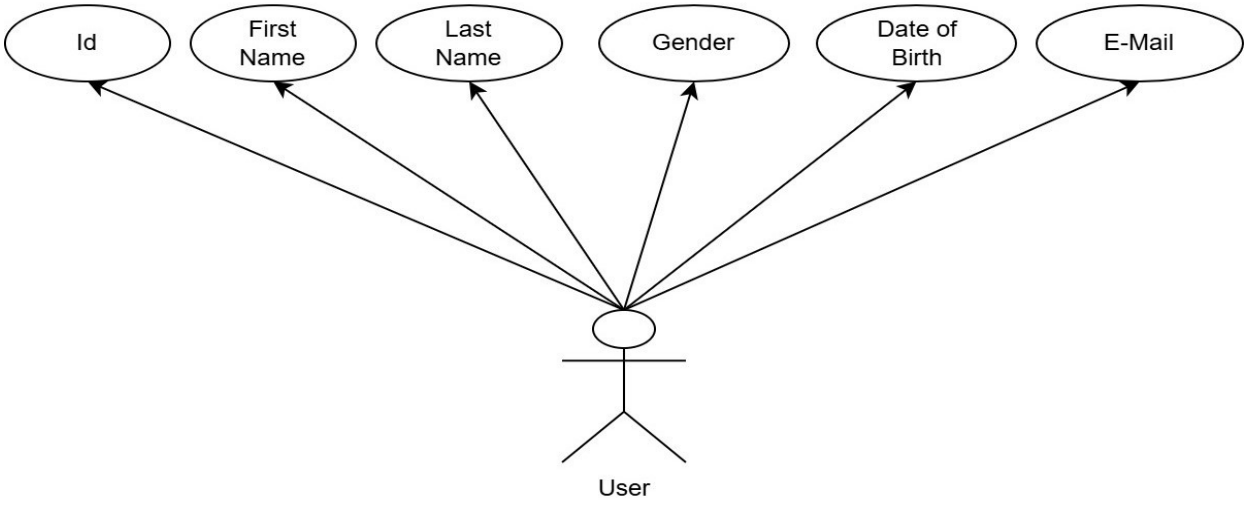


FIG 2.1 CASE DIAGRAM

## 2.2 Entity-relationship diagram:

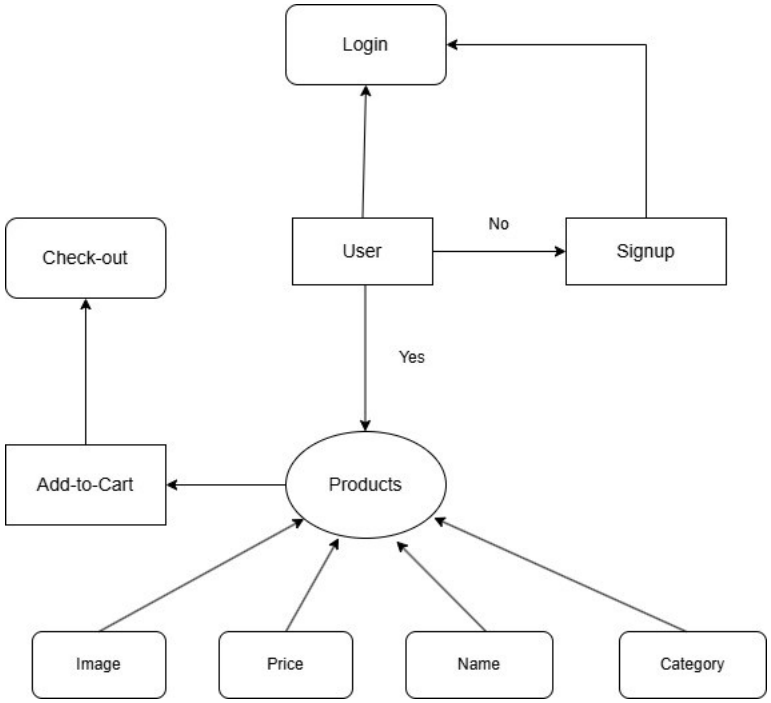


Fig 2.2 ER DIAGRAM OF UMS

# MODULE DESCRIPTION

**Register:**

- **Admin Registration:** Admins can register an account by providing a unique email and secure password. The system securely stores these credentials, ensuring only authorized administrators can access the backend.

**Login:**

- **Admin Login:** Admins log in using their email and password. The system verifies credentials against the database, granting access to administrative functions upon successful login.
- **User Login:** Customers log in using their email and password to access their accounts and manage orders.

**After Login:**

- **Add Product Data:** Admins can input product details, including names, descriptions, prices, categories, and image URLs. The system saves this data for product management and display.
- **View Product Data**: Admins can view detailed product information in a user-friendly format, with options to update or modify records as needed.
- **Update Product Data:** Admins can modify product details, allowing for necessary corrections or updates to keep information accurate.
- **Delete Product Data**: Admins can delete product records when no longer needed, with restrictions to prevent accidental deletion of important data.
- **Manage Shopping Cart**: Customers can add products to their cart, view cart details, update quantities, and remove items as needed.
- **Place Orders:** Customers can proceed to checkout, place orders, and manage order history.
- **Order Management**: Admins can view, update, and manage customer orders, ensuring efficient order processing.
- **Generate Sales Reports**: Admins can request sales reports and analytics, including total sales, top-selling products, and revenue metrics, aiding in business decision-making.
- **Remove Admin**: Admins can remove other administrators from the system, ensuring only authorized personnel have access to sensitive operations.

**Key Functional Modules:**

1. **Register Module (RM):**

- Admins register with unique email and secure password, securely stored in the database.

2. **Login Module (LM):**
    - Admins and users log in securely, with passwords encrypted and verified against the database.
3. **Product Management Module (PMM):**
    - Admins manage product data, including adding, updating, viewing, and deleting products.
4. **Shopping Cart Module (SCM):**
    - Customers manage their shopping cart, including adding, updating, and removing products.
5. **Order Management Module (OMM):**
    - Efficiently process, track, and manage customer orders and order history.
6. **Report Generation Module (RGM):**
    - Admins generate sales and performance reports for business insights.
7. **Admin Management Module (AMM):**
    - Manage admin accounts, including adding and removing admins.

**Security and Reliability:**

- Secure Authentication: Ensure strong authentication mechanisms for both admins and users.
- Data Encryption: Encrypt sensitive user data in transit and at rest.
- Data Integrity: Maintain data consistency and prevent unauthorized modifications

# IMPLEMENTATION

## 4.1 Design:

### Login:

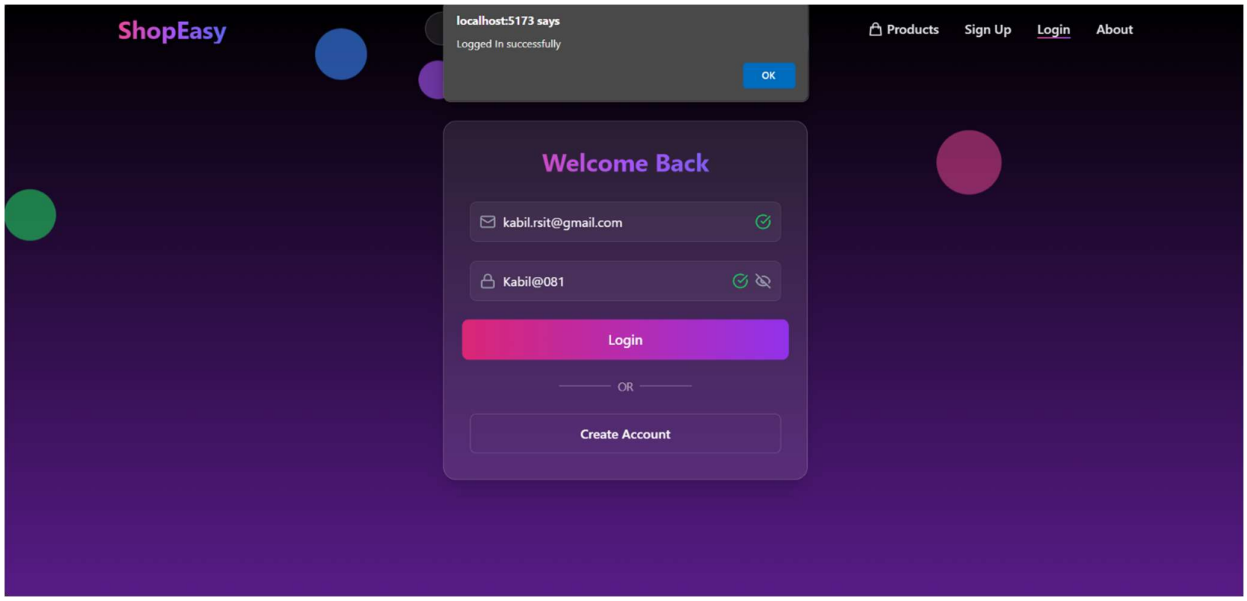


FIG 4.1 LOGIN

### Products:

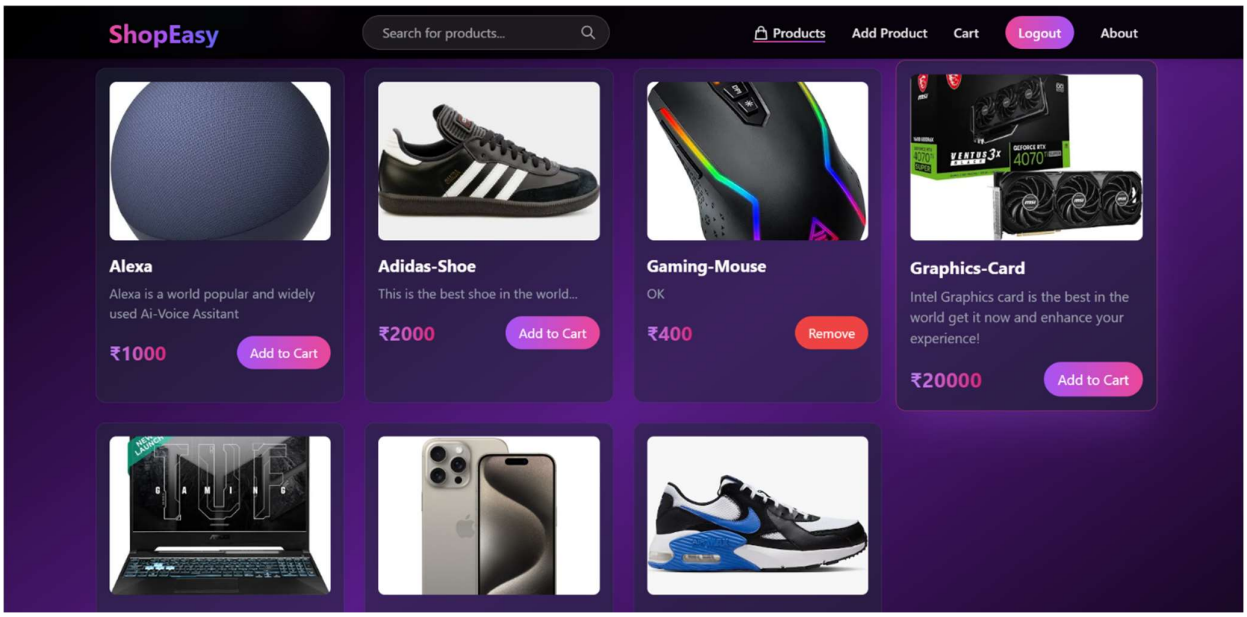


FIG 4.2 PRODUCTS

## Add Product:

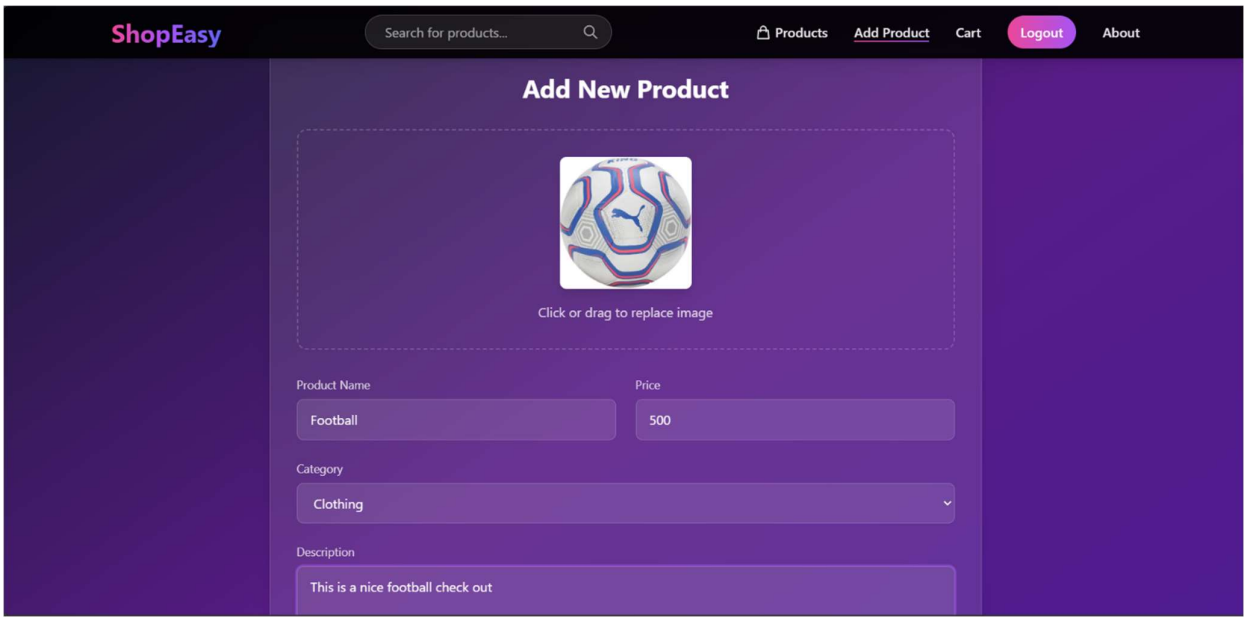


FIG 4.3 ADD PRODUCT

## Checkout:

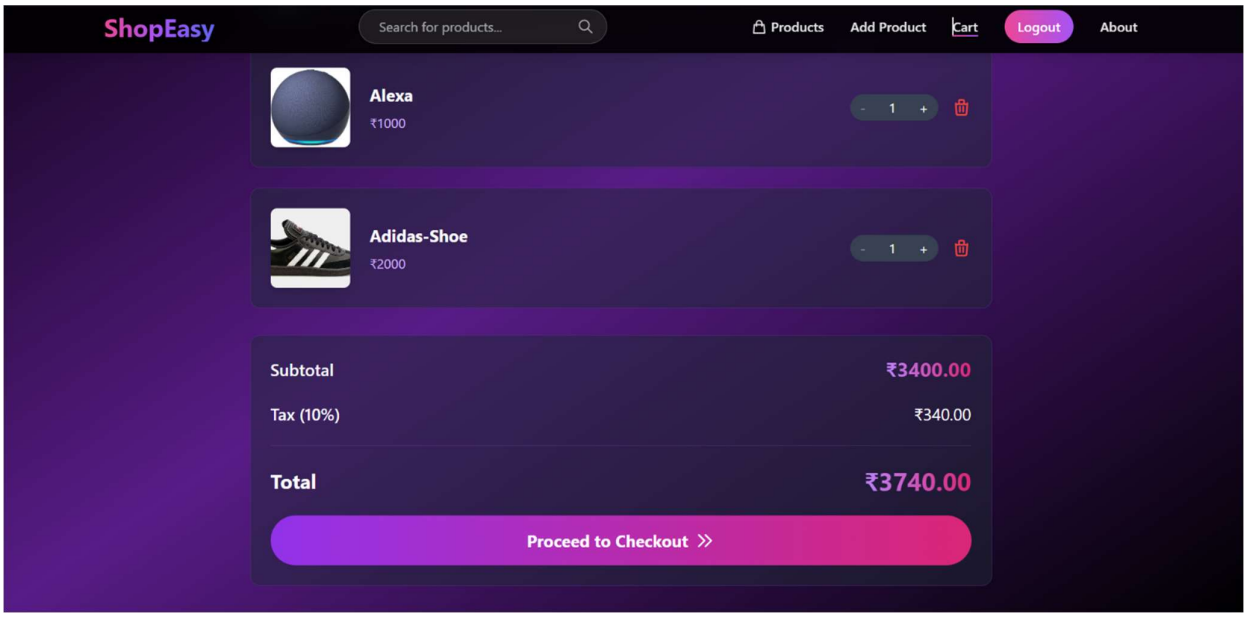


FIG 4.4 ADD PRODUCT

## 4.2 Database Design:

For the E-Commerce Platform, the data is stored and retrieved from a MongoDB database, which is chosen for its flexibility in handling unstructured data efficiently. The system uses a database to store product data, user information, and orders, ensuring smooth operations and data integrity.

**Database Design**

**Data Elements and Structures:**

- **Products:** Each product document includes fields such as product name, description, price, category, and image URL.
- **Users:** User documents contain information such as user ID, email, hashed password, and roles (admin or customer).
- **Orders:** Order documents include fields like order ID, user ID, product details, quantities, total price, and order status.
- **Cart:** Cart documents consist of user ID, product IDs, quantities, and total price.

**Normalization:**

- **Document Structure:** MongoDB's flexible schema allows for nesting documents, reducing the need for traditional normalization. This approach minimizes redundancy and improves performance by reducing the number of joins needed.
- **Embedded Documents:** Frequently accessed related data (e.g., product details within orders) is embedded to enhance read performance and maintain data consistency.

**Data Integrity:**

- **Relationships:** Relationships between data items are managed using references and embedding, ensuring data integrity. For example, orders reference user IDs and embed product details to maintain consistency.
- **Atomic Operations:** MongoDB supports atomic operations at the document level, ensuring that updates to documents are consistent and reliable.

**MongoDB Database:**

- **Selection:** MongoDB is selected due to its flexibility, scalability, and ability to handle large volumes of unstructured data efficiently. It supports quick, reliable, and flexible access to the data for various users (customers and administrators) while minimizing data inconsistencies.
- **Scalability:** MongoDB's sharding and replication features ensure the database can scale horizontally, handling increased load and providing high availability.

## USERS DATA:

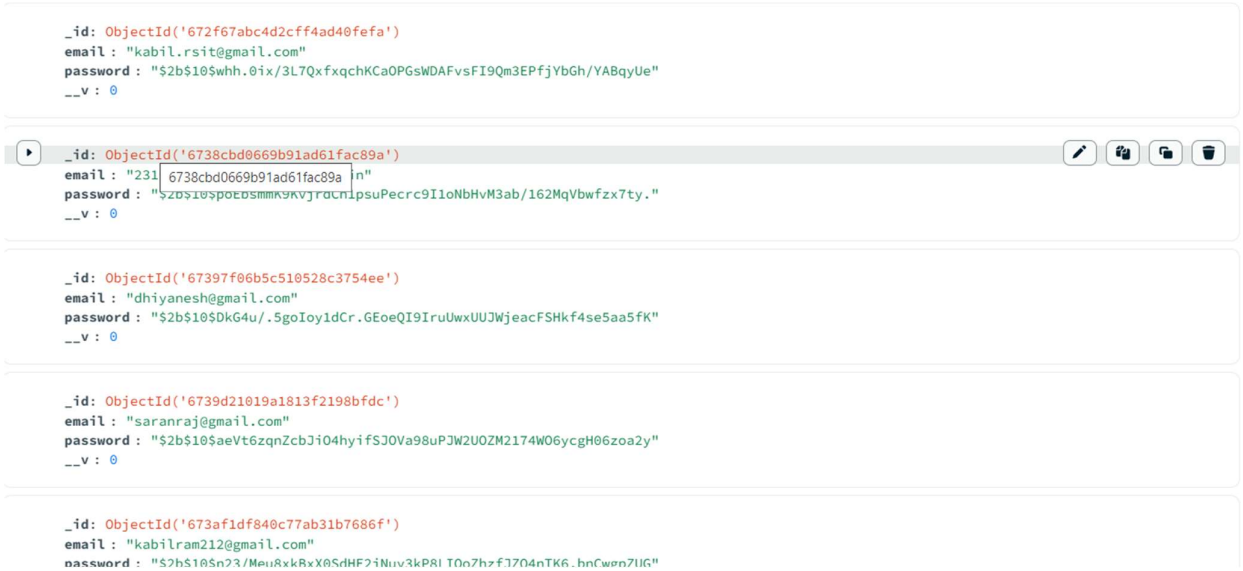


Fig 4.4 USERS DATA

**PRODUCTS DATA:**


Fig 4.5 Products Data

## 4.3 Code:

**appilication.properties:**

spring.data.mongodb.uri=mongodb://localhost:27017/E-commerce
jwt.secret=2d689b4ec754718cf74c47b03aa461718cb1f47a90a6b8b43e1e56f9eb839add
cloud.name=dul6vsdcf
cloudinary.api.secret=x19_704bYUjvht5EvpustGQxGKY
cloudinary.api.key=684924827497183

**ECommerceApplication.java:**

```
package com.example.ecommerce;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.web.bind.annotation.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.http.HttpStatus;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.authentication.AuthenticationManager;
```

```
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import org.springframework.data.mongodb.core.mapping.DBRef;
import org.springframework.data.annotation.Transient;
import org.springframework.stereotype.Repository;

import org.springframework.beans.factory.annotation.Value;
import io.jsonwebtoken.*;
import org.springframework.web.filter.OncePerRequestFilter;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.validation.constraints.NotNull;
import java.io.IOException;
import java.util.*;
import java.util.stream.Collectors;

@SpringBootApplication
public class EcommerceApplication {
    public static void main(String[] args) {
        SpringApplication.run(EcommerceApplication.class, args);
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
@Document(collection = "users")
class User {
    @Id
    private String id;
    @NotNull
    private String email;
    @NotNull
    private String password;
}

@Document(collection = "products")
class Product {
```

```
    @Id
    private String id;
    private String name;
    private String description;
    private double price;
    private String category;
    private String imageUrl;
}

@Document(collection = "carts")
class Cart {
    @Id
    private String id;
    @DBRef
    private User user;
    private List<CartItem> items = new ArrayList<>();
    private double total;
}

class CartItem {
    @DBRef
    private Product product;
    private int quantity;
}

@Repository
interface UserRepository extends MongoRepository<User, String> {
    Optional<User> findByEmail(String email);
}

@Repository
interface ProductRepository extends MongoRepository<Product, String> {
}

@Repository
interface CartRepository extends MongoRepository<Cart, String> {
    Optional<Cart> findByUserId(String userId);
}

// Services

@RestController
@RequestMapping("/api")
class ApiController {
    @Autowired
    private UserRepository userRepository;
```

```
    @Autowired
    private ProductRepository productRepository;

    @Autowired
    private CartRepository cartRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Value("${jwt.secret}")
    private String jwtSecret;

    @PostMapping("/signup")
    public ResponseEntity<?> signUp(@RequestBody Map<String, String> request) {
        String email = request.get("email");
        String password = request.get("password");

        if (userRepository.findByEmail(email).isPresent()) {
                return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("User already
exists!");
        }

        User user = new User();
        user.setEmail(email);
        user.setPassword(passwordEncoder.encode(password));
        userRepository.save(user);

        return ResponseEntity.status(HttpStatus.CREATED).body("Sign-Up Successful!");
    }
    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody Map<String, String> request) {
        String email = request.get("email");
        String password = request.get("password");
                User user = userRepository.findByEmail(email).orElseThrow(() -> new
UsernameNotFoundException("User Not Found!"));
        if (!passwordEncoder.matches(password, user.getPassword())){
          return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Invalid Password!");
        }
        String token = Jwts.builder()
                .setSubject(user.getId())
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + 3600000))
                .signWith(SignatureAlgorithm.HS256, jwtSecret)
                .compact();
```

```
        Map<String, String> response = new HashMap<>();
        response.put("token", token);
        response.put("userId", user.getId());

        return ResponseEntity.ok(response);
    }

    @PostMapping("/add-product")
    public ResponseEntity<?> addProduct(@RequestBody Product product) {
        productRepository.save(product);
                return ResponseEntity.status(HttpStatus.CREATED).body("Product Added
Successfully!");
    }

    @GetMapping("/products")
    public ResponseEntity<List<Product>> getProducts() {
        return ResponseEntity.ok(productRepository.findAll());
    }

    @PostMapping("/add-to-cart")
    public ResponseEntity<?> addToCart(@RequestBody Map<String, Object> request) {
        String userId = (String) request.get("userId");
        String productId = (String) request.get("productId");
        int quantity = (int) request.get("quantity");

        Cart cart = cartRepository.findByUserId(userId).orElseGet(() -> {
            Cart newCart = new Cart();
                    newCart.setUser(userRepository.findById(userId).orElseThrow(() -> new
RuntimeException("User Not Found")));
            return newCart;
        });

         Product product = productRepository.findById(productId).orElseThrow(() -> new
RuntimeException("Product Not Found"));
                Optional<CartItem> existingItem = cart.getItems().stream().filter(item ->
item.getProduct().getId().equals(productId)).findFirst();

        if (existingItem.isPresent()) {
            existingItem.get().setQuantity(existingItem.get().getQuantity() + quantity);
        } else {
            CartItem newItem = new CartItem();
            newItem.setProduct(product);
            newItem.setQuantity(quantity);
            cart.getItems().add(newItem);
        }
```

```
        cart.setTotal(cart.getItems().stream().mapToDouble(item -> item.getProduct().getPrice()
* item.getQuantity()).sum());
        cartRepository.save(cart);

        return ResponseEntity.ok(cart);
    }

    @PostMapping("/remove-from-cart")
    public ResponseEntity<?> removeFromCart(@RequestBody Map<String, String> request)
{
        String userId = request.get("userId");
        String productId = request.get("productId");

                Cart cart = cartRepository.findByUserId(userId).orElseThrow(() -> new
RuntimeException("Cart Not Found"));
        cart.getItems().removeIf(item -> item.getProduct().getId().equals(productId));
        cart.setTotal(cart.getItems().stream().mapToDouble(item -> item.getProduct().getPrice()
* item.getQuantity()).sum());

        cartRepository.save(cart);
        return ResponseEntity.ok("Product removed from cart");
    }
}
```

# CONCLUSION

The "E-Commerce Platform" project, developed with meticulous attention to detail, emphasizes a seamless user experience while ensuring efficient functionality for both customers and administrators. Key features include adding, viewing, and managing product information, with robust security measures for sensitive operations like modifying and removing product records to maintain data integrity and protect against unauthorized access. This system is designed to meet the current needs of retail businesses, offering long-term effectiveness and adaptability for future improvements, ensuring efficient management and reliability of e-commerce operations. The platform's integration of modern technologies such as React, Java, and MongoDB guarantees high performance, scalability, and reliability, making it a dependable solution for online retail.

## Reference links


- **Spring Boot - Official Documentation**: Spring Boot Documentation
- **React - Official Documentation**: React Documentation
- **MongoDB - Official Documentation**: MongoDB Documentation
- **Spring Data MongoDB - Official Documentation**: Spring Data MongoDB
- **JWT (JSON Web Tokens) - Official Documentation**: JWT Documentation
- **Cloudinary - Official Documentation**: Cloudinary Documentation
- **React Router - Official Documentation**: React Router Documentation
- **Spring Security - Official Documentation**: Spring Security Documentation