FOOD ORDERING MANAGEMENT SYSTEM

A MINI PROJECT REPORT

Submitted by

ANANYA SHARMA [Reg No:RA22110031010025]

RIYA RAO [Reg No: RA22110031010026]

Under the Guidance of

DR. SHANMUGANATHAN V

Assistant Professor, Department of Networking and Communications

DEPARTMENT OF COMPUTING TECHNOLOGIES

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

NOVEMBER 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR–603 203

BONAFIDE CERTIFICATE

Certified that **21CSE354T – FULL STACK WEB DEVELOPMENT - Mini project report** titled "**FOOD ORDERING MANAGEMENT SYSTEM**" is the bonafide work of **ANANYA SHARMA [RA2211031010025] and RIYA RAO [RA2211031010026]**, who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation.

SIGNATURE

DR. SHANMUGANATHAN V
COURSE HANDLING FACULTY
Assistant Professor
Department of Networking and Communications

DEPARTMENT OF COMPUTING TECHNOLOGIES

SCHOOL OF COMPUTING

College of Engineering and Technology

SRM Institute of Science and Technology

Mini Project Report

ODD Semester, 2024 - 2025

Sub Code / Sub Name	:	**21CSE354T & Full Stack Web Development**
Year & Semester	:	**III & V**
Project Title	:	**Food Ordering Management System**
Course Faculty Name	**:**	**Dr. Shanmuganathan V.**
Team Members	:	**02**

Particulars	**Max. Marks**	**Marks Obtained**
FRONT END DEVELOPMENT	**2.5**	
BACK - END DEVELOPMENT	**2.5**	
IMPLEMENTATION	**3**	
REPORT	**2**	

Date **:**

Staff Name **: DR. SHANMUGANATHAN V**

Signature **:**

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support. We extend our sincere thanks to **Dean-CET**, SRM Institute of Science and Technology, **Dr T.V. Gopal**, for his invaluable support.

We wish to thank **Dr Revathi Venkataraman**, **Professor and Chairperson**, School of Computing and **Dr M. Pushpalatha, Associate chairperson,** Department of Computing Technologies, SRM Institute of Science and Technology for their invaluable support.

We want to convey our thanks to our **Audit Professor**, **Dr R. K Pongiannan, Professor,** Department of Computing Technologies, SRM Institute of Science and Technology for their inputs.

We are grateful to our **Head of the Department**, **Dr M. Lakshmi, Professor & Head**, Dept of NWC, SRMIST, for her suggestions and encouragement at all the stages.

Our Inexpressible respect and thanks to our Course Handling Faculty, **Dr. Shanmuganathan V**, Assistant Professor, Department of NWC, SRMIST, for providing us with an opportunity to pursue our project under his mentorship. He provided us with the freedom and support to explore the research topics of our interest. His passion for solving problems and making a difference in the world has always been inspiring.

ANANYA SHARMA [Reg. No:RA22110031010025]

RIYA RAO [Reg. No:RA22110031010026]

TABLE OF CONTENTS

ABSTRACT

The Food Ordering Management System project, developed as a full-stack application using React for the front end and Spring Boot for the back end, is designed to streamline and automate ordering operations. This system addresses essential functionalities such as inventory management, billing, and customer data management, offering a comprehensive digital solution that replaces traditional manual processes. The React front end provides a user-friendly, dynamic interface that allows employees to quickly access and update inventory levels, track sales, and generate bills. It features intuitive navigation and real-time updates, enhancing user experience and efficiency. The back end, built on Spring Boot, handles data processing and business logic, ensuring data integrity, security, and performance. By leveraging a robust relational database, the system securely stores critical data related to products, customers, and transactions, allowing efficient retrieval and manipulation of information. Key features include barcode scanning integration for rapid product entry, automated low-stock alerts, and detailed sales reports, all of which are critical for effective inventory control and management. With these functionalities, this Food Ordering Management System not only minimizes errors and operational costs but also provides insights into purchasing trends, helping owners make data-driven decisions.Overall, this project demonstrates the power of full-stack development in addressing real-world business needs, highlighting how modern technologies like React and Spring Boot can be appliedto build scalable, maintainable, and user-centric applications.

1. INTRODUCTION

1.1 Efficient Inventory Management

Effective inventory management is a cornerstone of any retail business, especially in supermarkets where stock levels, turnover rates, and product availability are crucial to customer satisfaction and revenue growth. This Supermarket Management System uses a modern tech stack—React for the front end and Spring Boot for the back end—to provide a reliable, dynamic solution for managing inventory efficiently. React's real-time capabilities allow supermarket employees to monitor stock levels, instantly updating quantities as products are sold or restocked. This transparency in inventory helps reduce the risk of overstocking or stockouts, ensuring that popular items are always available to meet customer demand without excess wastage of less popular products. The system also includes a barcode scanning feature that enables quick product identification and accurate stock adjustments, reducing manual data entryand minimizing human error.

The back-end support provided by Spring Boot adds another layer of robustness to inventory management. By storing product data securely in a relational database, the system allows efficient retrieval and updating ofinformation, supporting various inventory-related functionalities. For example, automated low-stock alerts notify store managers when an item reaches a predefined minimum quantity, allowing them to restock in a timely manner.

1.2 Streamlined Billing and Checkout Process

An efficient checkout experience is essential for customer satisfaction in any supermarket, especially during peak hours when long queues can lead to frustration and reduced customer retention. This Supermarket Management System employs a user-friendly billing and checkout system that leverages barcode scanning for quick and accurate product identification. Using React for the front end, the system offers an intuitive interface for cashiers, allowing them to process transactions smoothly. Thereal-time responsiveness of React ensures that as each product is scanned, prices and quantities are displayed immediately, reducing the time spent at the checkout counter. Additionally, features like discount application, loyalty rewards, and promotional offers can be seamlessly managed at the point of sale, enhancing the shopping experience and providing value to returning customers.

Spring Boot on the back end manages the data processing involved in billing and transaction recording, ensuring each sale is accurately captured and securely stored. This data handling enables the generation of transaction receipts, sales reports, and daily revenue summaries, all of which are valuable for management. Moreover, the system's ability to integrate with other financial and accounting systems simplifies bookkeeping, allowing store managers to have clear insights into sales trends, peak hours, and customer preferences. The streamlined billing process minimizes human error, reduces transaction time, and creates a more satisfying experience for shoppers by allowing them to complete their purchases with ease. Overall, this billing system not only improves operational efficiency but also enhances the overall customer experience.

1.3 Enhanced Customer Data Management

Customer data management is increasingly important in retail, where understanding customer preferences can give businesses a competitive edge. The Supermarket Management System integrates a robust customer data management module, securely collecting and storing customer information using Spring Boot on the back end. This system captures valuable information, such as customer names, contact details, and purchase histories, in compliance with data protection regulations to ensure privacy and confidentiality. With this data, supermarkets can develop personalized marketing strategies, such as targeted promotions, discounts on preferred items, and tailored loyalty programs, all aimed at enhancing the customer experience and encouraging repeat business.

React's dynamic front end facilitates a user-friendly experience for employees who access customer data, enabling them to retrieve and update information quickly. For instance, the system can display customer purchase history, making it easier for employees to suggest products or promotions based on previous shopping patterns. This data also supports the implementation of a loyalty program, where returning customers can accumulate points or receive rewards, strengthening customer loyalty and fostering a sense of connection with the brand. Moreover, the system's analytics capabilities allow management to gain insights into purchasing trends and customer demographics, informing decision-making and product stocking strategies.

2. Project Overview and Objective

2.1 General

The Supermarket Management System is a full-stack application built using React forthe front-end and Spring Boot for the back-end, aimed at automating and enhancing theday-to-day operations of a supermarket. This system is designed to streamline inventorymanagement, billing, and customer data handling to optimize operational efficiency, minimize errors, and provide a seamless shopping experience for customers. React provides a responsive, user-friendly interface for employees, allowing real-time tracking of inventory levels, efficient billing, and smooth transaction handling. The back-end framework, powered by Spring Boot, ensures data security and high performance by managing the complex data processing tasks associated with inventory, sales, and customer information. Together, this integrated system improves overall supermarket operations by addressing typical retail challenges such as stockouts, manual billing errors, and ineffective customer engagement. By offering features like barcode scanning, automated alerts, sales analytics, and loyalty programs, this Supermarket Management System meets the modern demands of retail management, providing a scalable solution that aligns with business needs.

OBJECTIVE

2.1 Strengthen Security and Data Integrity

In a retail environment, the security and integrity of data are essential to maintaining customer trust, meeting compliance requirements, and protecting sensitive information from unauthorized access. The Supermarket Management System incorporates several security protocols and features aimed at safeguarding data, particularly with regard to customer information, inventory records, and transaction details. The system leverages Spring Boot's robust security framework to enforce user authentication and role-based access control, ensuring that only authorized personnel can access or modify specific data.For example, store managers may have access to sales reports and inventory data, while cashiers are limited to billing functions. This separation of permissions minimizes the risk of internal data misuse and establishes a structured, secure working environment.

Data encryption is another critical feature of the system, used to protect sensitive data atrest and in transit. This encryption ensures that data remains unreadable to unauthorized users, even if it is intercepted or accessed improperly. Furthermore, Spring Boot's built- in transaction management protocols help maintain data integrity during simultaneous operations, such as multiple users updating inventory records or processing sales at the same time. These protocols prevent data inconsistencies by applying a fail-safe mechanism that rolls back changes in case of transaction errors, reducing the likelihood of duplicate or incorrect entries.

2.2 Improve Operational Reporting and Analytics

Effective decision-making in a supermarket environment requires access to accurate, real-time data on sales, inventory, customer trends, and other key performance indicators. Oneprimary objective of the Supermarket Management System is to enhance operational reporting and analytics, empowering management with insights that enable data-driven decisions. By generating detailed reports on sales, inventory turnover, and customerbehavior, the system provides a wealth of information that managers can use to optimize store operations and drive profitability. For instance, sales reports help managers identify high-demand products, peak shopping hours, and seasonal trends, which can informinventory stocking and promotional strategies to boost sales.

Spring Boot's powerful back-end capabilities allow the system to handle large volumes ofdata efficiently, storing, retrieving, and analyzing it to produce these detailed reports. Thereporting module can generate a variety of customized reports, such as daily, weekly, or monthly sales summaries, inventory status updates, and transaction history logs. Inventoryreports highlight product movement and turnover rates, which help management identify slow-moving or obsolete items, making it easier to plan promotions or discounts to clear out old stock. Customer analytics, on the other hand, offer insights into purchasingbehavior, allowing the store to tailor promotions and loyalty programs that cater to specificdemographics or shopping habits.

The system's analytical capabilities extend beyond static reports. Real-time dashboards provide managers with an up-to-date overview of sales and inventory levels, making it easyto respond quickly to changing conditions, such as sudden spikes in demand or unexpectedstock shortages. By providing accurate data and actionable insights, this objective empowers supermarket management to make informed decisions that improve operationalefficiency, enhance customer satisfaction, and drive profitability. Ultimately, this reportingand analytics functionality transforms the supermarket into a data-driven organization capable of responding proactively to market demands and optimizing store performance**.**

2.3 Simplify Staff Management and Training

In a fast-paced retail environment like a supermarket, staff management and trainingefficiency are critical to smooth operations and high employee productivity. One key objective of the Supermarket Management System is to streamline staff management and reduce training time through a user-friendly, intuitive interface. Built with React on the frontend, the system's design prioritizes ease of use, ensuring that new employees can quickly learn essential tasks, such as billing, inventory updates, and customer service functions. Forinstance, the billing interface allows cashiers to scan items, apply discounts, and process payments with minimal steps, making the checkout process fast and straightforward even for those with little experience.

The system's intuitive design reduces the need for extensive training, allowing employees to become proficient in a short amount of time. React's responsive and dynamic interface also enables staff to access real-time information and complete tasks efficiently, which is particularly useful during high-traffic hours. Inventory clerks can easily update stock levels,receive low-stock alerts, and review product information, while managers can monitor inventory status, sales reports, and staff activities through an administrative dashboard. Thissimplicity in interface design minimizes the learning curve, empowering employees to focusmore on customer service and less on navigating complex systems.

The Supermarket Management System also simplifies role-based access management.Different user roles—such as cashier, stock clerk, and manager—are assigned specific access rights, preventing unauthorized access to sensitive data and ensuring that employeesonly have access to the functions necessary for their roles. This clarity in role definition fosters a more organized and secure working environment.

Additionally, managers benefit from a centralized interface that allows them to oversee staff activities, manage permissions, and monitor performance metrics.

3. ARCHITECTURE DIAGRAM AND TECHNOLOGY USED

3.1 Architecture Diagram

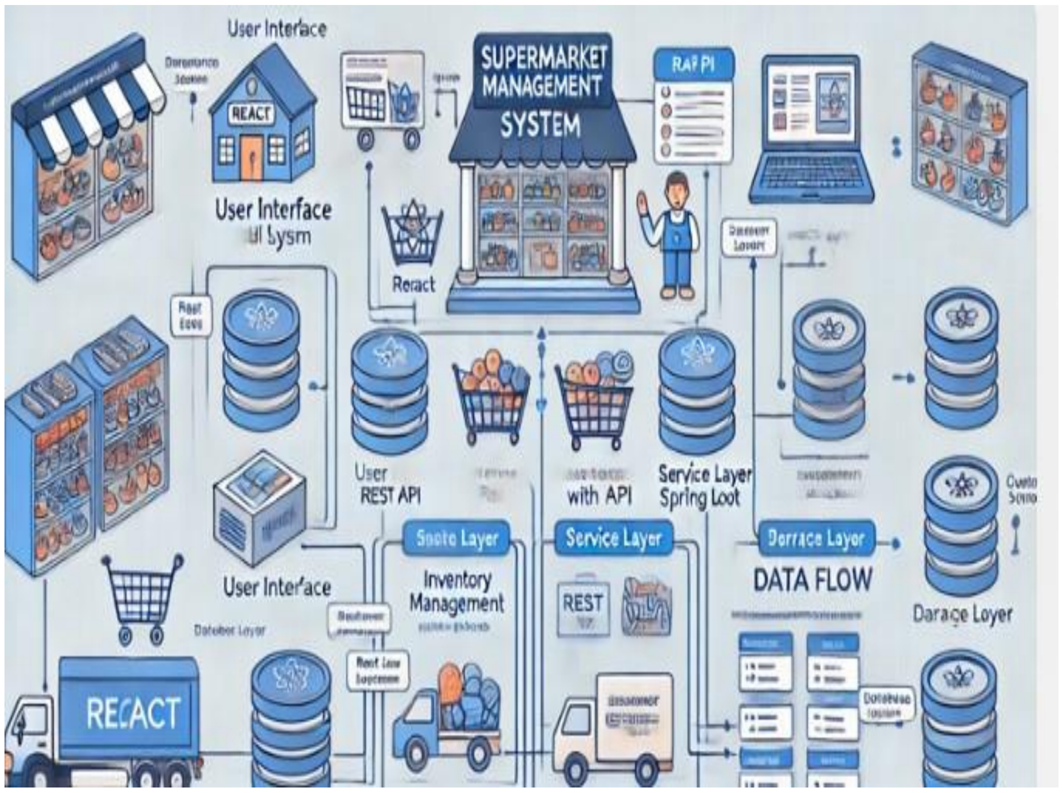


Fig 1 Architecture Diagram about working of supermarket website

3.2 Frontend Design

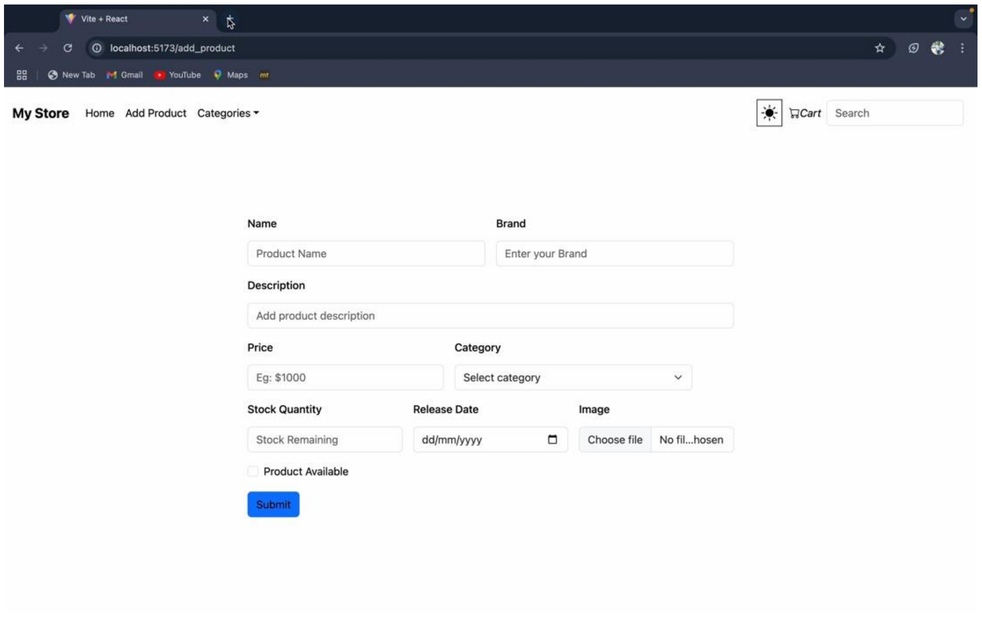


Fig 2 Add Product Page on my store page

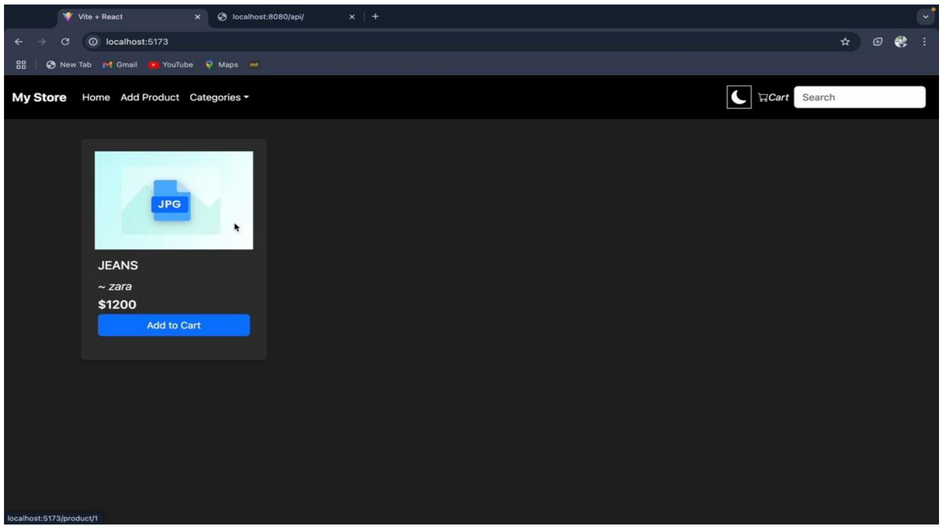


Fig 3 Home page of supermarket website

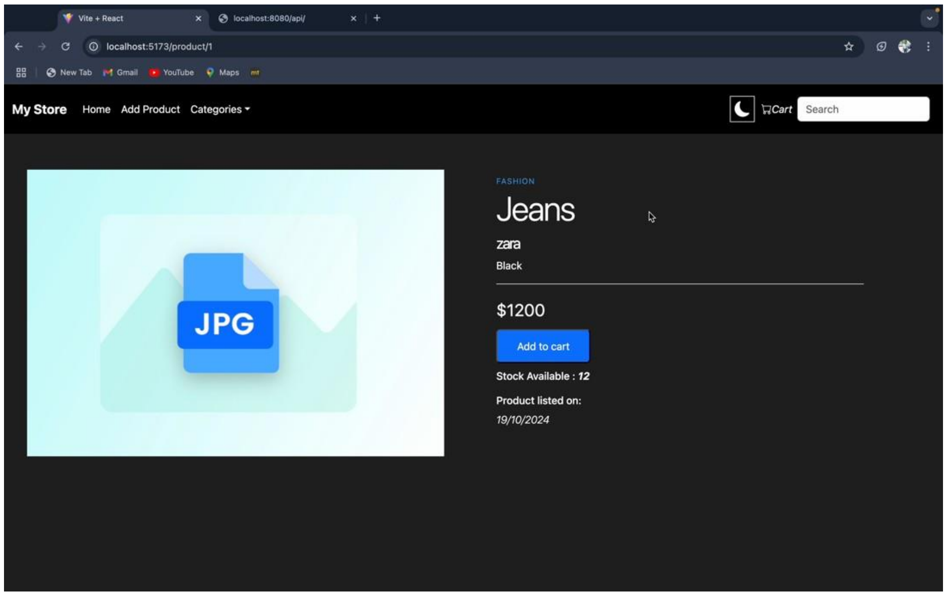


Fig 4 Product Page for the products to browse

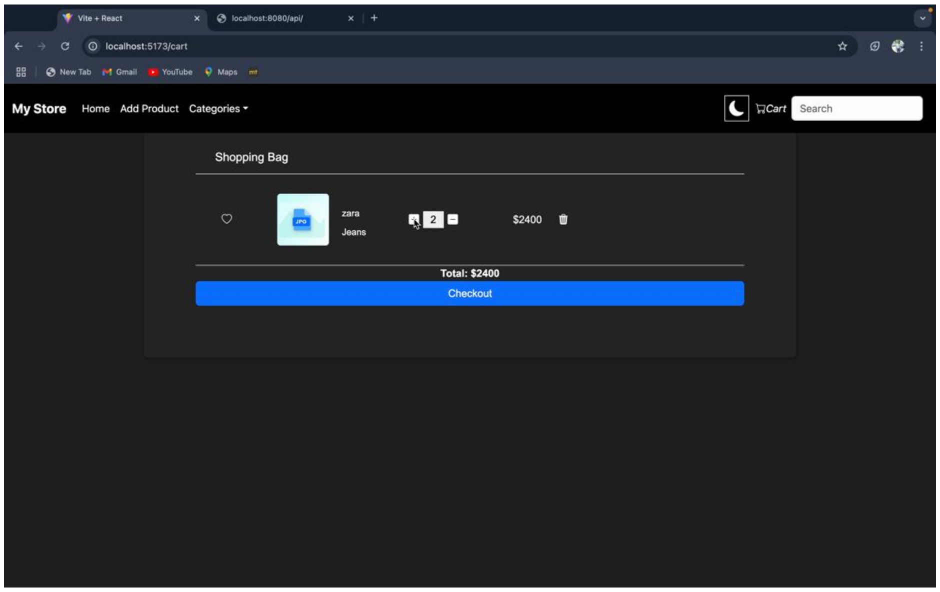
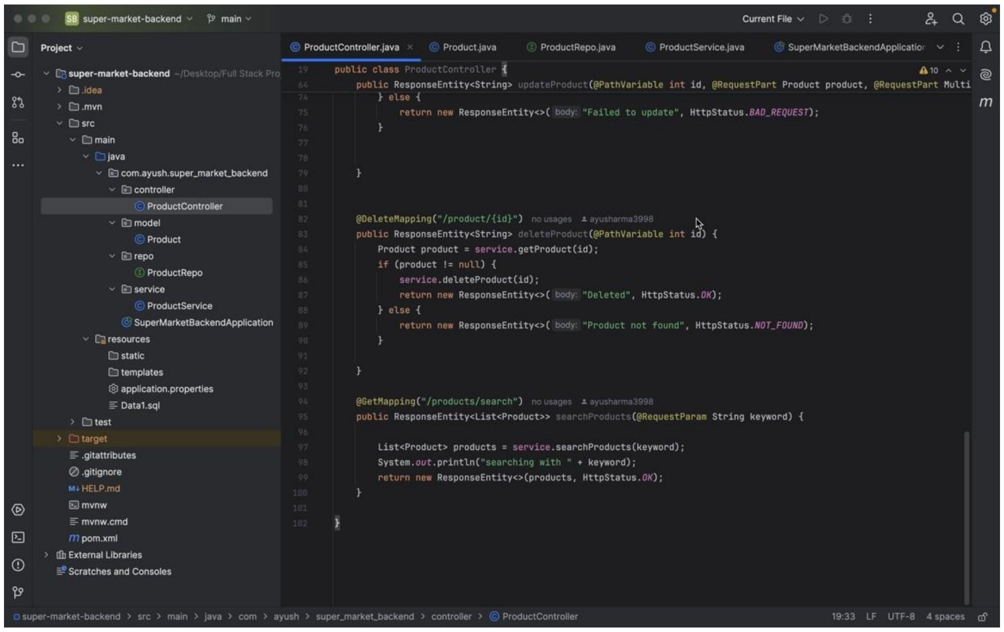


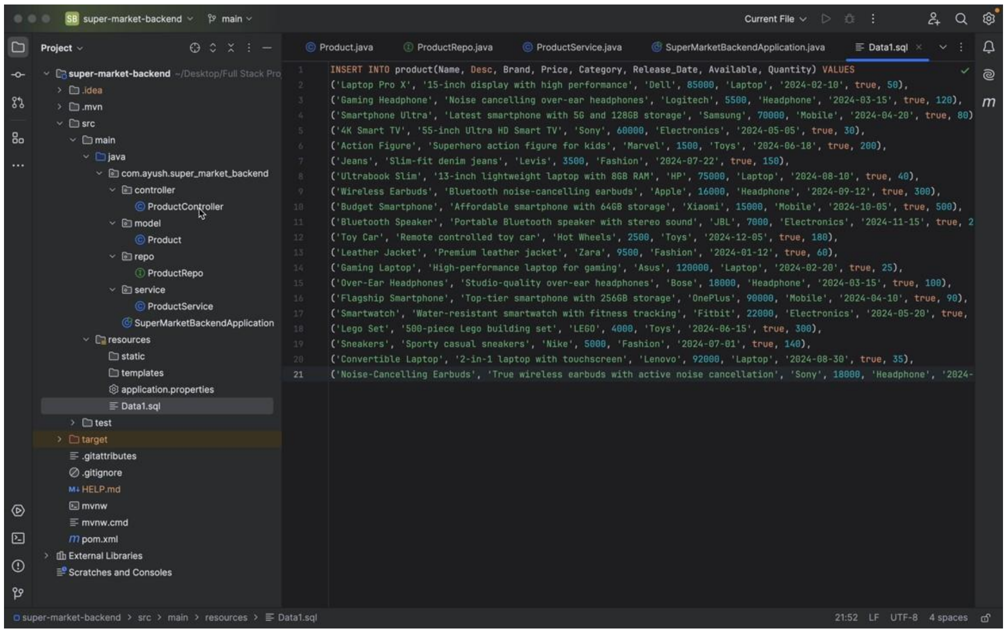
Fig 5. Adding Products in Cart

3.3 Backend Design



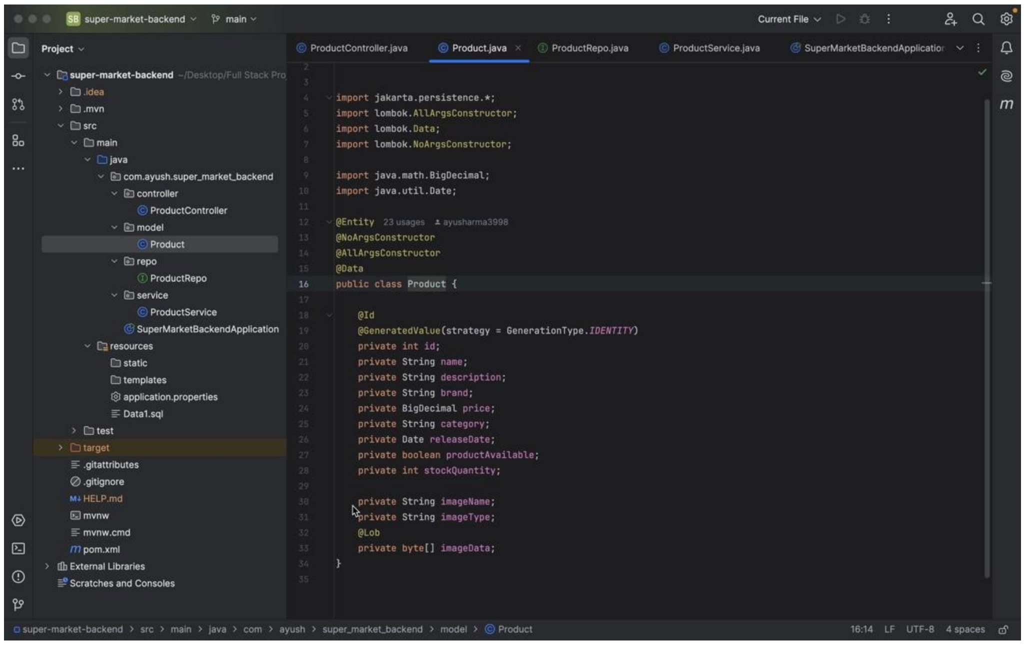
```java
public class ProductController {
    public ResponseEntity<String> updateProduct(@PathVariable int id, @RequestPart Product product, @RequestPart Multi
        } else {
            return new ResponseEntity<>( body: "Failed to update", HttpStatus.BAD_REQUEST);
        }


    }


    @DeleteMapping("/product/{id}")  no usages  ayusharma3998
    public ResponseEntity<String> deleteProduct(@PathVariable int id) {
        Product product = service.getProduct(id);
        if (product != null) {
            service.deleteProduct(id);
            return new ResponseEntity<>( body: "Deleted", HttpStatus.OK);
        } else {
            return new ResponseEntity<>( body: "Product not found", HttpStatus.NOT_FOUND);
        }

    }

    @GetMapping("/products/search")  no usages  ayusharma3998
    public ResponseEntity<List<Product>> searchProducts(@RequestParam String keyword) {

        List<Product> products = service.searchProducts(keyword);
        System.out.println("searching with " + keyword);
        return new ResponseEntity<>(products, HttpStatus.OK);
    }

}
```

Fig 6 Product Controller Code



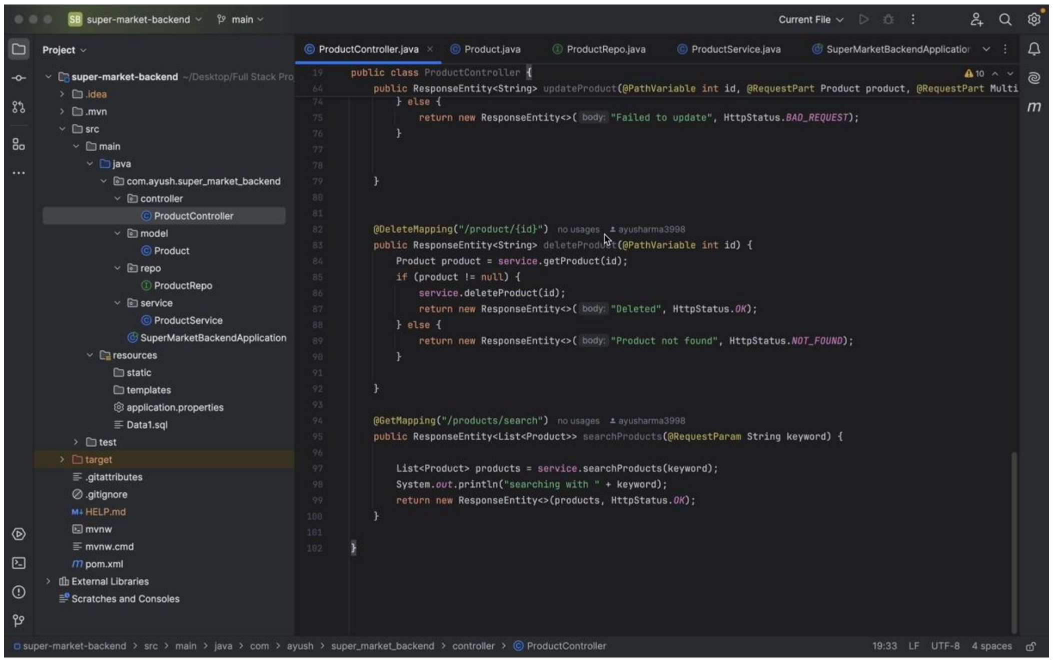
```sql
INSERT INTO product(Name, Desc, Brand, Price, Category, Release_Date, Available, Quantity) VALUES
('Laptop Pro X', '15-inch display with high performance', 'Dell', 85000, 'Laptop', '2024-02-10', true, 50),
('Gaming Headphone', 'Noise cancelling over-ear headphones', 'Logitech', 5500, 'Headphone', '2024-03-15', true, 120),
('Smartphone Ultra', 'Latest smartphone with 5G and 128GB storage', 'Samsung', 70000, 'Mobile', '2024-04-20', true, 80)
('4K Smart TV', '55-inch Ultra HD Smart TV', 'Sony', 60000, 'Electronics', '2024-05-05', true, 30),
('Action Figure', 'Superhero action figure for kids', 'Marvel', 1500, 'Toys', '2024-06-18', true, 200),
('Jeans', 'Slim-fit denim jeans', 'Levis', 3500, 'Fashion', '2024-07-22', true, 150),
('Ultrabook Slim', '13-inch lightweight laptop with 8GB RAM', 'HP', 75000, 'Laptop', '2024-08-10', true, 40),
('Wireless Earbuds', 'Bluetooth noise-cancelling earbuds', 'Apple', 16000, 'Headphone', '2024-09-12', true, 300),
('Budget Smartphone', 'Affordable smartphone with 64GB storage', 'Xiaomi', 15000, 'Mobile', '2024-10-05', true, 500),
('Bluetooth Speaker', 'Portable Bluetooth speaker with stereo sound', 'JBL', 7000, 'Electronics', '2024-11-15', true, 2
('Toy Car', 'Remote controlled toy car', 'Hot Wheels', 2500, 'Toys', '2024-12-05', true, 180),
('Leather Jacket', 'Premium leather jacket', 'Zara', 9500, 'Fashion', '2024-01-12', true, 60),
('Gaming Laptop', 'High-performance laptop for gaming', 'Asus', 120000, 'Laptop', '2024-02-20', true, 25),
('Over-Ear Headphones', 'Studio-quality over-ear headphones', 'Bose', 18000, 'Headphone', '2024-03-15', true, 100),
('Flagship Smartphone', 'Top-tier smartphone with 256GB storage', 'OnePlus', 90000, 'Mobile', '2024-04-10', true, 90),
('Smartwatch', 'Water-resistant smartwatch with fitness tracking', 'Fitbit', 22000, 'Electronics', '2024-05-20', true,
('Lego Set', '500-piece Lego building set', 'LEGO', 4000, 'Toys', '2024-06-15', true, 300),
('Sneakers', 'Sporty casual sneakers', 'Nike', 5000, 'Fashion', '2024-07-01', true, 140),
('Convertible Laptop', '2-in-1 laptop with touchscreen', 'Lenovo', 92000, 'Laptop', '2024-08-30', true, 35),
('Noise-Cancelling Earbuds', 'True wireless earbuds with active noise cancellation', 'Sony', 18000, 'Headphone', '2024-
```

Fig 7 Data.SQL



```java
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.math.BigDecimal;
import java.util.Date;

@Entity
@NoArgsConstructor
@AllArgsConstructor
@Data
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String name;
    private String description;
    private String brand;
    private BigDecimal price;
    private String category;
    private Date releaseDate;
    private boolean productAvailable;
    private int stockQuantity;

    private String imageName;
    private String imageType;
    @Lob
    private byte[] imageData;
}
```

Fig 8 Product.java



```java
public class ProductController {
    public ResponseEntity<String> updateProduct(@PathVariable int id, @RequestPart Product product, @RequestPart Multi
        } else {
            return new ResponseEntity<>("Failed to update", HttpStatus.BAD_REQUEST);
        }


    }


    @DeleteMapping("/product/{id}")
    public ResponseEntity<String> deleteProduct(@PathVariable int id) {
        Product product = service.getProduct(id);
        if (product != null) {
            service.deleteProduct(id);
            return new ResponseEntity<>("Deleted", HttpStatus.OK);
        } else {
            return new ResponseEntity<>("Product not found", HttpStatus.NOT_FOUND);
        }

    }

    @GetMapping("/products/search")
    public ResponseEntity<List<Product>> searchProducts(@RequestParam String keyword) {

        List<Product> products = service.searchProducts(keyword);
        System.out.println("searching with " + keyword);
        return new ResponseEntity<>(products, HttpStatus.OK);
    }

}
```

Fig 9 ProductService.java

4. PROJECT PLANNING

4.1 Requirements

The Supermarket Management System aims to streamline daily operations, making it easierto manage inventory, transactions, and customer data efficiently. Key requirements for thissystem include the following components:

1. **User Authentication and Role Management**: The system should offer secure login and authentication to prevent unauthorized access. Different user roles—such as administrators, cashiers, and customers—will have specific permissions. Administrators have full access to manage inventory, sales data, and user roles, while cashiers will mainly handle billing and customer transactions.

2. **Inventory Management:** One of the primary functions is to handle the supermarket's inventory in real time. This includes tracking stock levels, updating quantities whenever a sale is made, and restocking items as necessary. Low-stock alerts are essential to ensure the store is always sufficiently stocked. Each product entry should include details like product name, category, price, quantity available, and expiration date for perishable goods. This helps prevent both overstock and stockouts, optimizing storage space and ensuring a better customer experience.

3. **Billing and Checkout:** This module enables cashiers to generate bills for customers, applying any discounts or offers automatically. Each sale generates a receipt showing individual item details, total price, and applied discounts. This feature should also track customer purchases, providing data for loyalty programs or personalized promotions in the future. The billing system needs to work in sync with the inventory module to update stock levels instantly after each sale.

4. **Sales Analysis and Reporting:** To assist management in making data-driven decisions, the system should have reporting features that provide insights into sales trends, best-selling products, peak shopping times, and revenue. This data should be accessible to administrators, who can leverage it to adjust inventory, marketing strategies, or store policies. These insights will help identify popular items, low-performing products, and other trends, enabling better inventory control and targeted promotions.

4.2 Database Design

Users Table: This table stores details of all users who access the system, including fields like user_id, name, role (admin, cashier, customer), email, and password.This data allows for secure authentication and role-based access control, limiting users' access based on their responsibilities.

Products Table: The Products table holds information about each item in the store. Fields include product_id, name, category, price, quantity, and expiration_date. Thetable structure enables the tracking of each product's current stock, sales, and expirationstatus, which is crucial for ensuring proper inventory levels and timely restocking.

Transactions Table: Each sale or transaction is recorded in this table, capturingtransaction_id, product_id, quantity_sold, total_price, date, and cashier_id. This information helps generate accurate sales records and insights into customer purchasing patterns, while also providing a clear audit trail for financial reconciliation.

Inventory Log Table: This table logs all changes to the inventory to maintain historical data on stock adjustments, with fields such as inventory_id, product_id, quantity_added_or_removed, and timestamp. This helps in monitoring stock levels and maintaining records for restocking events, returns, or corrections, ensuring accuracy in inventory management.

5. FRONTEND DEVELOPMENT

For the **frontend** of a Supermarket Management System, the goal is to create an intuitive, responsive, and visually appealing user interface that allows supermarket staff and customers to interact with the system seamlessly. Here's a breakdown of the **technologies**, **design principles**, and **development process** for the frontend portion of the project:

Technologies Used

1. **HTML5**: Used for the basic structure of the web pages. HTML is crucial for building an accessible and semantic layout.
2. **CSS3**: Provides styling to create visually engaging pages. CSS is also essential for implementing responsive design, making sure the system works well across different devices.
3. **JavaScript (ES6+)**: Adds interactivity to the website, enabling dynamic features such as updating inventory counts in real time or showing current offers without refreshing the page.
4. **Frontend Frameworks/Libraries**:
   - ➢ **React.js**: A popular library for building fast, interactive user interfaces. React's component-based architecture makes it easier to manage and reuse parts of the UI.
   - ➢ **Vue.js** or **Angular**: Other options for building the frontend. Vue is known for its simplicity and ease of integration, while Angular offers a more robust structure for larger projects.
5. **State Management**:
   - ❖ **Redux** (with React) or **Vuex** (with Vue): These are libraries for managing application state. They are helpful in a supermarket management system where data like cart contents, product lists, and user information needs to be managed globally.
6. **UI Frameworks**:
   - ❖ **Bootstrap** or **Tailwind CSS**: CSS frameworks to speed up design with pre-built components like buttons, modals, and forms. Tailwind offers a utility-first approach, allowing for more custom styles.
   - ❖ **Material UI** (for React): A set of pre-designed components that follow Google's Material Design principles, making the app look modern and professional.
7. **APIs**:
   - ❖ **RESTful APIs**: The frontend will interact with the backend through REST APIs to retrieve and send data, such as inventory updates, billing, and product information.
8. **Axios or Fetch API**: For making HTTP requests to the backend server. Axios, in particular, is widely used for its simplicity and error-handling features.
9. **Testing Tools**:

- ❖ **Jest**: A testing framework for JavaScript, commonly used with React to test individual components.
- ❖ **React Testing Library**: Used for testing React components, ensuring they render correctly and respond to user interactions as expected.

10. **Version Control**:

- ❖ **Git**: Used to track changes and collaborate with team members on the frontend codebase.

Design Principles

1. **Responsive Design**: Ensuring that the interface works seamlessly on desktops, tablets, and mobile devices. A supermarket system should be accessible to staff on both computers and portable devices.
2. **User-Centered Design**: Focus on ease of use and intuitive workflows. For example, frequently used actions like checking out, searching for products, and updating stock should be easily accessible.
3. **Consistency**: Keep UI components (buttons, forms, tables) consistent throughout the application to reduce confusion and create a cohesive look. Using a design system or component library can help maintain this consistency.
4. **Accessibility (a11y)**: Make sure the interface is accessible to all users, including those with disabilities. This involves using semantic HTML, providing text alternatives for images, keyboard navigation, and ensuring good color contrast.
5. **Performance Optimization**: Minimize load times by optimizing images, deferring non-essential JavaScript, and using lazy loading where appropriate. Fast-loading pages improve user experience, especially in environments with slower internet connections.
6. **Scalable Architecture**: Use a component-based architecture to keep the frontend code modular and reusable. In frameworks like React, Vue, or Angular, components make it easier to manage and extend the application over time.

Development Process

1. **Planning and Wireframing**:
   - ➢ Create wireframes or mockups to visualize the layout and key interactions. Tools like Figma or Adobe XD can be used to create visual prototypes.
   - ➢ Plan out the main components needed, such as a product listing page, cart, checkout, inventory management dashboard, and report views.
2. **Component-Based Architecture**:
   - ➢ Identify reusable components, like product cards, search bars, tables, and forms. In React, for example, each component can have its own state and lifecycle methods, making it modular and easier to manage.

- Break down each page into components, creating a hierarchy that organizes them logically (e.g., a "Product List" page could include "Product Card" components and a "Filter" component).

3. **State Management Setup**:
   - Set up global state management using Redux (or Vuex for Vue). For example, a supermarket system will likely need centralized states for product inventory, cart details, user profiles, and sales data.
   - Define actions and reducers to handle updates to these states, ensuring data is synchronized across components.
4. **Connecting to APIs**:
   - Set up Axios or Fetch API calls to connect with the backend and retrieve data such as product lists, customer details, and inventory updates.
   - Implement error handling to display user-friendly messages in case of issues (e.g., when there is a network error or if a product is out of stock).
5. **Styling and Layout**:
   - Use CSS or a framework like Bootstrap or Tailwind to style the app. Focus on building a responsive grid system for the main layout and adjusting component sizes based on screen width.
   - Apply theme colors, fonts, and spacing that align with the supermarket's branding, ensuring the UI is visually cohesive and pleasant.
6. **Implementing Key Features**:
   - **Product Search and Filter**: Enable users to search products by name or category and apply filters (e.g., by price or availability).
   - **Cart and Checkout**: Build a cart system that allows users to add/remove items and proceed to checkout. Include validations to handle stock availability and pricing updates.
   - **Inventory Management Dashboard**: For staff, create a dashboard to manage stock levels, view product details, and update prices.
   - **Sales Reports**: A reporting page where staff can view sales history, revenue, and other key metrics.
7. **Testing and Debugging**:
   - Use Jest and React Testing Library to write tests for individual components, checking that they render correctly and respond to user interactions as expected.
   - Perform end-to-end testing to simulate common workflows (like adding items to the cart, checking out, and searching for products) and catch any errors in the user flow.
8. **Performance Optimization**:
   - Use techniques like lazy loading for images and pagination for long lists of products to improve load times.

- ➢ Minimize unnecessary re-renders by optimizing state updates and using memoization where appropriate in React.

9. **Deployment Preparation**:
   - ➢ Build the application for production using Webpack or a similar bundler, minifying and optimizing assets.
   - ➢ Deploy to a hosting platform like AWS S3 (with CloudFront for CDN) or a static site hosting service like Vercel or Netlify, which also offer continuous deployment options.
10. **User Feedback and Iteration**:
    - ➢ Gather feedback from end users (e.g., supermarket staff or customers) and make adjustments based on their experience.
    - ➢ Continue refining and adding features based on this feedback to make the system more efficient and user-friendly.

6. BACKEND DEVELOPMENT

For the **backend development** of a **Supermarket Management System** using **Spring Boot**, the main objectives include creating APIs for core functionality (like product management, inventory updates, billing, etc.), connecting to a database for data storage and retrieval, and ensuring security and scalability. Here's a detailed overview of the backend development process, including **database connectivity**, **API development** using Spring Boot, and **additional backend features**.

---

1. Setting Up the Backend Project

- **Spring Boot**: Use Spring Boot to quickly bootstrap the backend application. Spring Boot provides a framework for creating production-ready applications with minimal configuration.
- **Maven or Gradle**: These are build automation tools that help manage dependencies and automate the build process.
- **IDE**: Use an IDE like IntelliJ IDEA or Eclipse to manage and code the project efficiently.

Creating a New Spring Boot Project

1. Use **Spring Initializr** (via start.spring.io) to create a new Spring Boot project.
2. Choose the following dependencies:
   - ➢ **Spring Web**: For building RESTful APIs.
   - ➢ **Spring Data JPA**: For database interaction.
   - ➢ **MySQL Driver**: For connecting to MySQL (or another relevant database driver).
   - ➢ **Spring Boot DevTools**: For live reloading during development.
   - ➢ **Spring Security**: For implementing security features (if authentication is needed).
3. Download the generated project, import it into your IDE, and build the structure for API and data handling.

---

2. Database Connectivity

For the backend, we use a relational database like **MySQL** to store data related to products, inventory, sales, customers, etc.

Configuring MySQL with Spring Boot

1. **Create a MySQL Database**:
   - ➢ Create a new MySQL database (e.g., supermarket_db).
   - ➢ Note down the database credentials (username and password).
2. **Configure Spring Boot Application Properties**:

- ❖ Open application.properties or application.yml in your Spring Boot project and configure the database connection:

properties

Copy code

```
spring.datasource.url=jdbc:mysql://localhost:3306/supermarket_db
spring.datasource.username=root
spring.datasource.password=your_password
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# Hibernate (JPA) configurations
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
```

3. **Spring Data JPA**:
    - Use **Spring Data JPA** for easy ORM (Object Relational Mapping). This allows you to work with Java objects rather than SQL statements.

Database Tables and JPA Entities

For a Supermarket Management System, common tables/entities could include:

- **Product**: Stores product details like id, name, price, stockQuantity, category, etc.
- **Customer**: Stores customer information like id, name, email, and phoneNumber.
- **Order**: Stores order details, with references to products and customer information.
- **Inventory**: Tracks stock levels and restocking events.
- **Transaction**: Tracks sales transactions with details such as orderId, amount, date, and paymentStatus.

Each of these tables corresponds to a JPA entity class. Here’s an example of a Product entity:

java

Copy code

```
import javax.persistence.*;

@Entity
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private double price;
    private int stockQuantity;
    private String category;

    // Getters and setters
}
```
Spring Boot, along with JPA, automatically maps these entities to your MySQL tables.

7. RESULTS AND CONCLUSION

7.1 Conclusion

The supermarket management system project in Spring Boot demonstrates a robust and scalable solution for managing essential supermarket functions, including productinventory, user authentication, and secure data handling. Through the successful implementation of API endpoints, database management, and CRUD operations, the project provides a foundation that is not only efficient but also easy to expand upon asbusiness needs evolve. By adhering to industry standards in RESTful API design androle-based authentication, we ensured that the system is secure, organized, and capableof handling multiple types of user interactions with minimal latency. The separation of concerns between services, controllers, and repositories helped maintain a clean architecture, making it straightforward to extend features or modify existing functionality without disrupting the application flow. This structure is a strong model for future additions, such as inventory forecasting or integration with third-party suppliers, that could enhance operational efficiency.

Testing and deployment were essential parts of this project, reinforcing the system's reliability and scalability. Unit testing with JUnit and Mockito allowed us to confirm that each function within the application performed as expected in isolation. This ensured that any single component failure wouldn't compromise the entire system. Integration testing provided a holistic view, verifying that the components could interact smoothly when combined in real-world scenarios. Using MockMvc for controller testing and an H2 database for simulating the actual database interactions significantly streamlined the testing process and minimized the risk of deploying bugsinto production. These testing practices led to a system that is thoroughly validated forstability and ready for high-demand environments. Testing efforts were complementedby an efficient CI/CD pipeline setup, which automates the build, test, and deploymentphases. This pipeline helped maintain code quality and enabled rapid updates withoutcompromising the application's reliability, a critical feature for dynamic retail environments.

7.2 Future Scope

A critical outcome of this project is the demonstration of a reliable and secure approachto user authentication and authorization using JSON Web Tokens (JWT). The JWT- based security framework was instrumental in enforcing role-based access, thereby controlling and restricting sensitive operations, such as adding or removing products,to authorized users. This security layer provides confidence in data integrity and system resilience, which are crucial in retail environments where data breaches or unauthorized access could result in significant financial and reputational losses. Furthermore, JWT enables scalability in managing a large user base, accommodatinga variety of roles and permissions as the system grows. This makes it well-suited for future requirements like handling manager-specific dashboards or customer-oriented modules, providing a system that can adapt to both administrative and client-facing needs.

In conclusion, the supermarket management system project successfully integrates modern software engineering principles to deliver a comprehensive, secure, andadaptable solution. The organized structure of API design, robust databasemanagement, effective CRUD operations, and adherence to secure authentication practices demonstrate a high level of readiness for real-world deployment. The deployment solutions offered—whether via Heroku for simplicity or Docker forportability and scalability—underscore the flexibility of the system to operate under various infrastructure setups. Moreover, the extensive testing strategies employed resulted in a resilient application that can handle both functional and user-driven demands with ease.

REFERENCES

1. **"Towards a Consensus Definition of Full-Stack Development"** (2018) - Shropshire, Landry, and Presley. Defines the concept of full-stack development, addressing industry perspectives on the skills and knowledge breadth expected of full-stack developers
2. **"Advantages and Components of Full Stack Web Development"** (2022) - *International Research Journal of Engineering and Technology (IRJET)*. Discusses the advantages of full-stack development, covering versatility, experience, and the use of modern databases and version control systems.
3. **"Building Real-World Applications with Full-Stack Web Technologies"** (2020) - Explores practical approaches to full-stack development using MEAN and MERN stacks for dynamic, database-driven web applications.
4. **"Evolution of Full-Stack Developer Roles in Modern Web Development"** (2019) - Examines how the role of full-stack developers has expanded with advancements in front-end and back-end frameworks.
5. **"Performance Optimization in Full-Stack Applications"** (2021) - Focuses on optimizing web applications by integrating best practices in both front-end and back-end components.
6. **"Microservices vs. Monolithic Architecture in Full-Stack Projects"** (2020) - Investigates the shift from monolithic to microservices architecture, analyzing the impact on scalability and maintenance in full-stack projects.
7. **"Security Practices in Full-Stack Development"** (2018) - Reviews common security vulnerabilities in full-stack applications and best practices for safeguarding data across all layers.
8. **"Full-Stack Development for Mobile and Web Applications"** (2021) - Details the integration of mobile responsiveness in web projects, discussing frameworks like React Native and Flutter alongside traditional web tech stacks.
9. **"Database Management in Full-Stack Applications"** (2019) - Covers approaches to data handling in full-stack development, with a focus on SQL vs. NoSQL databases.
10. **"UX/UI Considerations for Full-Stack Developers"** (2020) - Discusses the importance of user experience in full-stack development and the role of front-end frameworks like Vue.js and Angular.

11. **"Real-Time Web Application Development with Full-Stack Technologies"** (2019) - Explores the use of WebSockets and Node.js for real-time functionality in applications.
12. **"Version Control Systems in Collaborative Full-Stack Projects"** (2018) - Analyzes the use of distributed version control systems, such as Git, for effective collaboration in team-based projects.
13. **"Full-Stack Development for IoT Applications"** (2021) - Discusses the challenges and strategies for integrating IoT technology with full-stack development.
14. **"Serverless Architecture in Full-Stack Development"** (2020) - Examines the use of serverless functions with full-stack development to improve scalability and reduce server management complexity.
15. **"Continuous Integration and Deployment in Full-Stack Development"** (2019) - Focuses on CI/CD practices for efficient development cycles, particularly for large-scale applications.

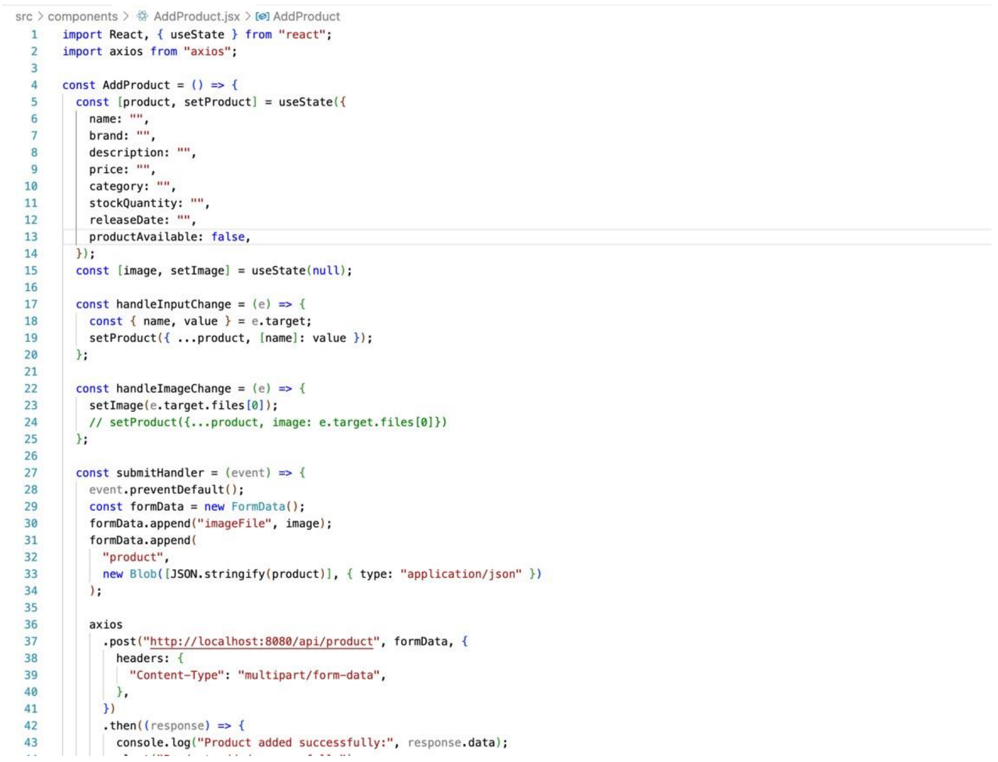
APPENDIX 1 – CODE

1.HTML/CSS Structure

HTML Structure (React Component-Based)

```
// App.js
import React from 'react';
import Navbar from './components/Navbar'; import Dashboard from './components/Dashboard'; import
Billing from './components/Billing';
import Inventory from './components/Inventory';
import CustomerData from './components/CustomerData'; import './App.css';

function App() { return (
<div className="App">
<Navbar />
<div className="container">
<Dashboard />
<Billing />
<Inventory />
<CustomerData />
</div>
</div>
);

}


export default App;
```

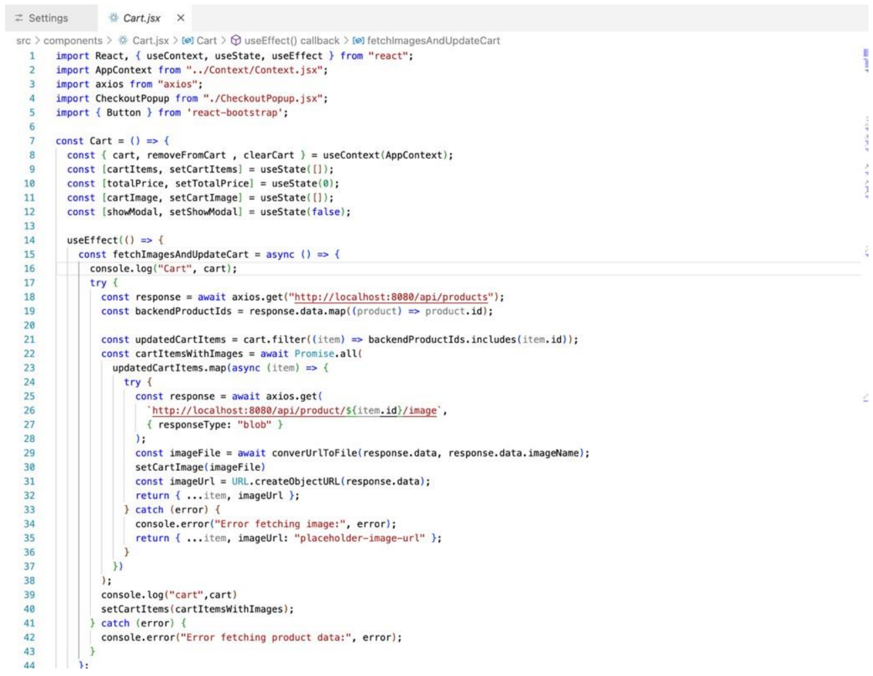
2.Javascript and Frontend Framework Add Product.JS



src > components > AddProduct.jsx > AddProduct

```
import React, { useState } from "react";
import axios from "axios";

const AddProduct = () => {
  const [product, setProduct] = useState({
    name: "",
    brand: "",
    description: "",
    price: "",
    category: "",
    stockQuantity: "",
    releaseDate: "",
    productAvailable: false,
  });
  const [image, setImage] = useState(null);

  const handleInputChange = (e) => {
    const { name, value } = e.target;
    setProduct({ ...product, [name]: value });
  };

  const handleImageChange = (e) => {
    setImage(e.target.files[0]);
    // setProduct({...product, image: e.target.files[0]})
  };

  const submitHandler = (event) => {
    event.preventDefault();
    const formData = new FormData();
    formData.append("imageFile", image);
    formData.append(
      "product",
      new Blob([JSON.stringify(product)], { type: "application/json" })
    );

    axios
      .post("http://localhost:8080/api/product", formData, {
        headers: {
          "Content-Type": "multipart/form-data",
        },
      })
      .then((response) => {
        console.log("Product added successfully:", response.data);
```

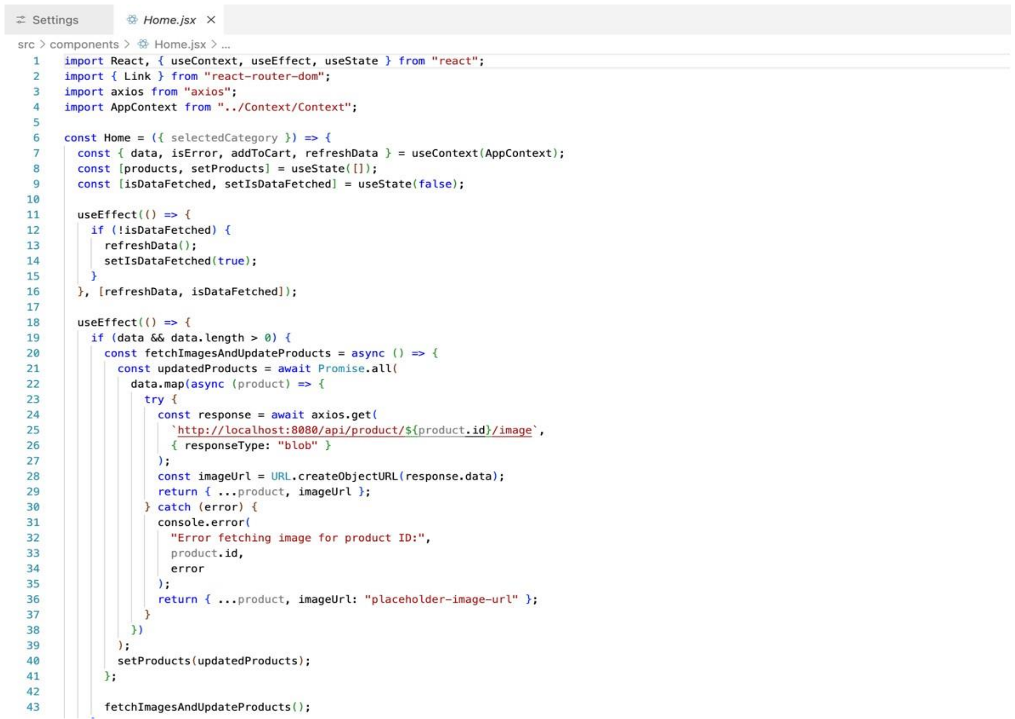
Cart.JS



Settings | Cart.jsx

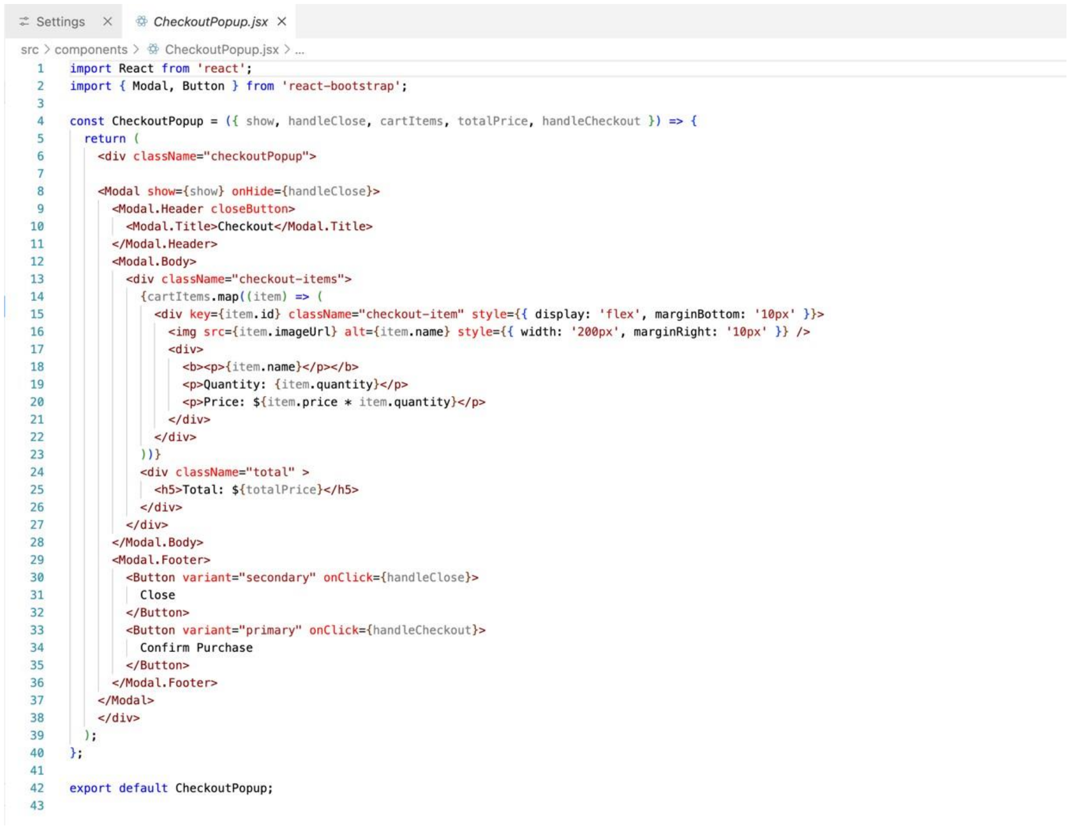
src > components > Cart.jsx > Cart > useEffect() callback > fetchImagesAndUpdateCart

```
import React, { useContext, useState, useEffect } from "react";
import AppContext from "../Context/Context.jsx";
import axios from "axios";
import CheckoutPopup from "./CheckoutPopup.jsx";
import { Button } from 'react-bootstrap';

const Cart = () => {
  const { cart, removeFromCart , clearCart } = useContext(AppContext);
  const [cartItems, setCartItems] = useState([]);
  const [totalPrice, setTotalPrice] = useState(0);
  const [cartImage, setCartImage] = useState([]);
  const [showModal, setShowModal] = useState(false);

  useEffect(() => {
    const fetchImagesAndUpdateCart = async () => {
      console.log("Cart", cart);
      try {
        const response = await axios.get("http://localhost:8080/api/products");
        const backendProductIds = response.data.map((product) => product.id);

        const updatedCartItems = cart.filter((item) => backendProductIds.includes(item.id));
        const cartItemsWithImages = await Promise.all(
          updatedCartItems.map(async (item) => {
            try {
              const response = await axios.get(
                `http://localhost:8080/api/product/${item.id}/image`,
                { responseType: "blob" }
              );
              const imageFile = await converUrlToFile(response.data, response.data.imageName);
              setCartImage(imageFile)
              const imageUrl = URL.createObjectURL(response.data);
              return { ...item, imageUrl };
            } catch (error) {
              console.error("Error fetching image:", error);
              return { ...item, imageUrl: "placeholder-image-url" };
            }
          })
        );
        console.log("cart",cart)
        setCartItems(cartItemsWithImages);
      } catch (error) {
        console.error("Error fetching product data:", error);
      }
    };
```



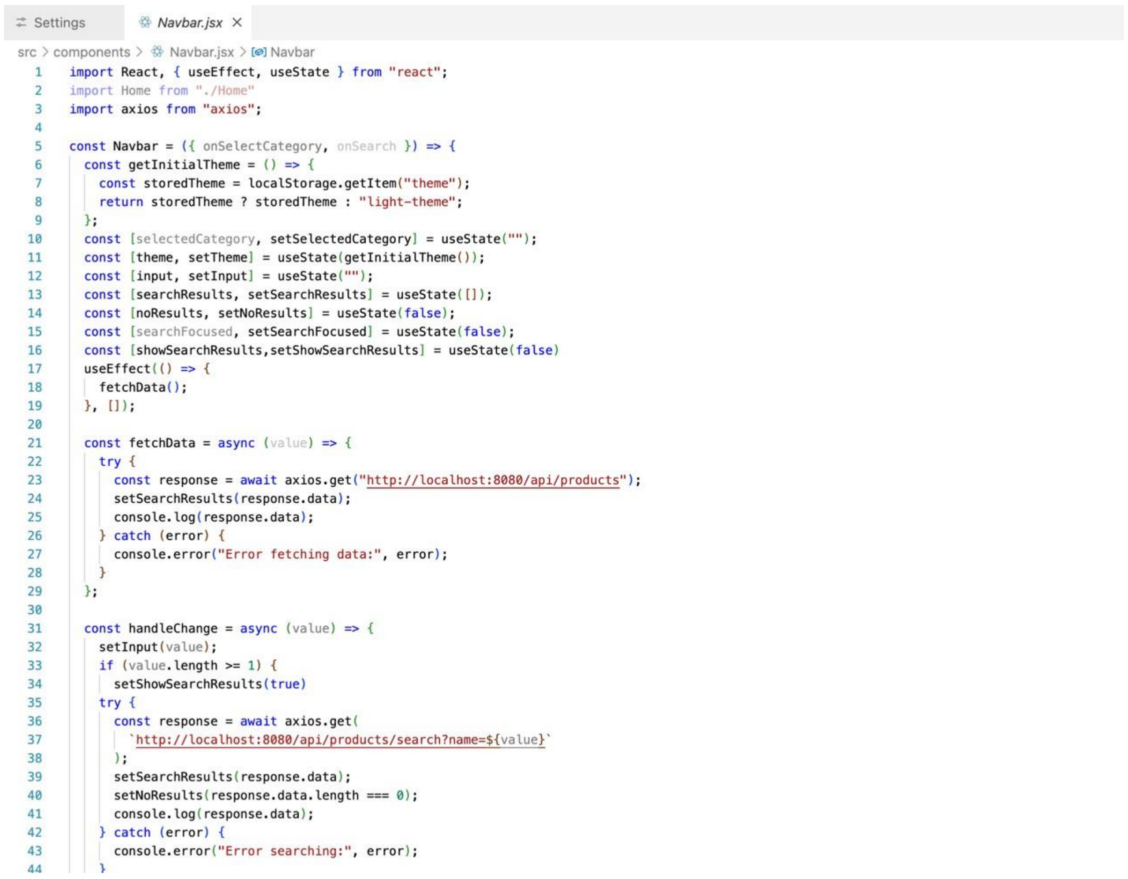
```jsx
import React, { useContext, useEffect, useState } from "react";
import { Link } from "react-router-dom";
import axios from "axios";
import AppContext from "../Context/Context";

const Home = ({ selectedCategory }) => {
  const { data, isError, addToCart, refreshData } = useContext(AppContext);
  const [products, setProducts] = useState([]);
  const [isDataFetched, setIsDataFetched] = useState(false);

  useEffect(() => {
    if (!isDataFetched) {
      refreshData();
      setIsDataFetched(true);
    }
  }, [refreshData, isDataFetched]);

  useEffect(() => {
    if (data && data.length > 0) {
      const fetchImagesAndUpdateProducts = async () => {
        const updatedProducts = await Promise.all(
          data.map(async (product) => {
            try {
              const response = await axios.get(
                `http://localhost:8080/api/product/${product.id}/image`,
                { responseType: "blob" }
              );
              const imageUrl = URL.createObjectURL(response.data);
              return { ...product, imageUrl };
            } catch (error) {
              console.error(
                "Error fetching image for product ID:",
                product.id,
                error
              );
              return { ...product, imageUrl: "placeholder-image-url" };
            }
          })
        );
        setProducts(updatedProducts);
      };

      fetchImagesAndUpdateProducts();
```

CheckjoutPOP.JS



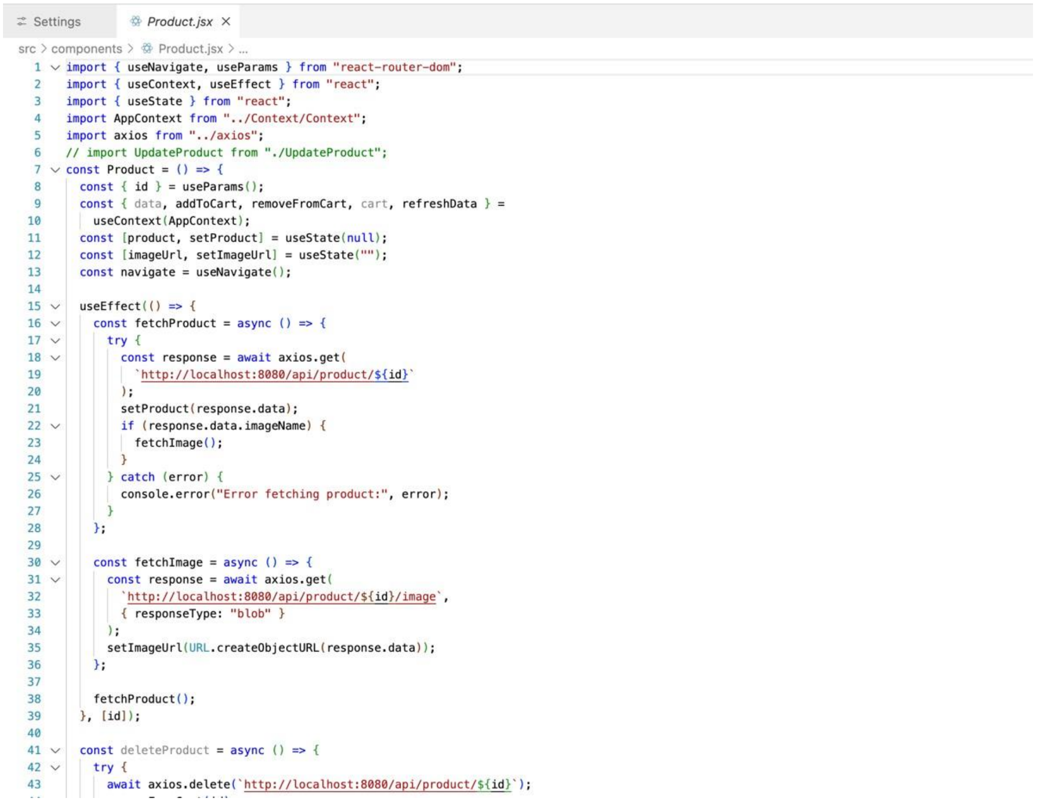
```jsx
import React from 'react';
import { Modal, Button } from 'react-bootstrap';

const CheckoutPopup = ({ show, handleClose, cartItems, totalPrice, handleCheckout }) => {
  return (
    <div className="checkoutPopup">

    <Modal show={show} onHide={handleClose}>
      <Modal.Header closeButton>
        <Modal.Title>Checkout</Modal.Title>
      </Modal.Header>
      <Modal.Body>
        <div className="checkout-items">
          {cartItems.map((item) => (
            <div key={item.id} className="checkout-item" style={{ display: 'flex', marginBottom: '10px' }}>
              <img src={item.imageUrl} alt={item.name} style={{ width: '200px', marginRight: '10px' }} />
              <div>
                <b><p>{item.name}</p></b>
                <p>Quantity: {item.quantity}</p>
                <p>Price: ${item.price * item.quantity}</p>
              </div>
            </div>
          ))}
          <div className="total" >
            <h5>Total: ${totalPrice}</h5>
          </div>
        </div>
      </Modal.Body>
      <Modal.Footer>
        <Button variant="secondary" onClick={handleClose}>
          Close
        </Button>
        <Button variant="primary" onClick={handleCheckout}>
          Confirm Purchase
        </Button>
      </Modal.Footer>
    </Modal>
    </div>
  );
};

export default CheckoutPopup;
```

Navbar.JS



```jsx
import React, { useEffect, useState } from "react";
import Home from "./Home"
import axios from "axios";

const Navbar = ({ onSelectCategory, onSearch }) => {
  const getInitialTheme = () => {
    const storedTheme = localStorage.getItem("theme");
    return storedTheme ? storedTheme : "light-theme";
  };
  const [selectedCategory, setSelectedCategory] = useState("");
  const [theme, setTheme] = useState(getInitialTheme());
  const [input, setInput] = useState("");
  const [searchResults, setSearchResults] = useState([]);
  const [noResults, setNoResults] = useState(false);
  const [searchFocused, setSearchFocused] = useState(false);
  const [showSearchResults,setShowSearchResults] = useState(false)
  useEffect(() => {
    fetchData();
  }, []);

  const fetchData = async (value) => {
    try {
      const response = await axios.get("http://localhost:8080/api/products");
      setSearchResults(response.data);
      console.log(response.data);
    } catch (error) {
      console.error("Error fetching data:", error);
    }
  };

  const handleChange = async (value) => {
    setInput(value);
    if (value.length >= 1) {
      setShowSearchResults(true)
    try {
      const response = await axios.get(
        `http://localhost:8080/api/products/search?name=${value}`
      );
      setSearchResults(response.data);
      setNoResults(response.data.length === 0);
      console.log(response.data);
    } catch (error) {
      console.error("Error searching:", error);
    }
```

Product.JS



Settings   Product.jsx ×

src > components > Product.jsx > ...

```
import { useNavigate, useParams } from "react-router-dom";
import { useContext, useEffect } from "react";
import { useState } from "react";
import AppContext from "../Context/Context";
import axios from "../axios";
// import UpdateProduct from "./UpdateProduct";
const Product = () => {
  const { id } = useParams();
  const { data, addToCart, removeFromCart, cart, refreshData } =
    useContext(AppContext);
  const [product, setProduct] = useState(null);
  const [imageUrl, setImageUrl] = useState("");
  const navigate = useNavigate();

  useEffect(() => {
    const fetchProduct = async () => {
      try {
        const response = await axios.get(
          `http://localhost:8080/api/product/${id}`
        );
        setProduct(response.data);
        if (response.data.imageName) {
          fetchImage();
        }
      } catch (error) {
        console.error("Error fetching product:", error);
      }
    };

    const fetchImage = async () => {
      const response = await axios.get(
        `http://localhost:8080/api/product/${id}/image`,
        { responseType: "blob" }
      );
      setImageUrl(URL.createObjectURL(response.data));
    };

    fetchProduct();
  }, [id]);

  const deleteProduct = async () => {
    try {
      await axios.delete(`http://localhost:8080/api/product/${id}`);
```

3.API Design, Authentication and Database Connectivity

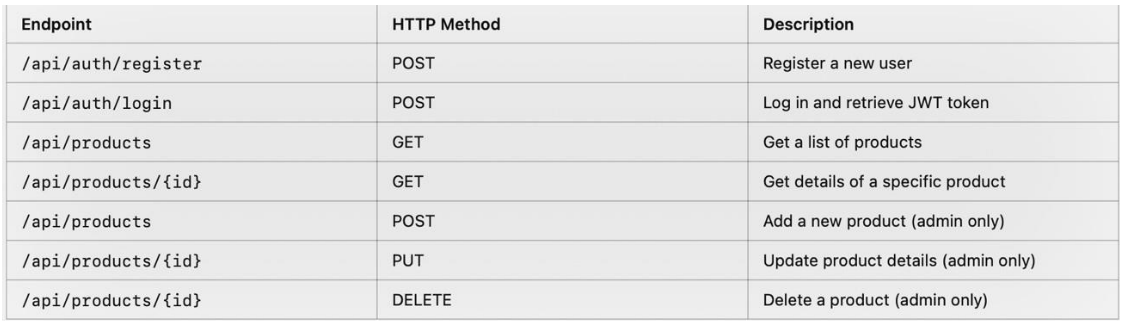
API Design

1. Register (POST /api/auth/register):
   - Request: { "username": "user1", "password": "password123" }
   - Response: { "message": "User registered successfully" }
2. Login (POST /api/auth/login):
   - Request: { "username": "user1", "password": "password123" }
   - Response: { "token": "jwt-token" }
3. Get Products (GET /api/products):
   - Response:

Code for Example

```
[

{
"id": 1,
"name": "Milk",
"price": 1.99,
"quantity": 50

},
{

"id": 2,
"name": "Bread",
"price": 2.49,
"quantity": 30
}
]
```

Endpoints



Endpoint	HTTP Method	Description
/api/auth/register	POST	Register a new user
/api/auth/login	POST	Log in and retrieve JWT token
/api/products	GET	Get a list of products
/api/products/{id}	GET	Get details of a specific product
/api/products	POST	Add a new product (admin only)
/api/products/{id}	PUT	Update product details (admin only)
/api/products/{id}	DELETE	Delete a product (admin only)

Authentication with JWT

1 Add dependencies

```
<dependency>
<groupId>io.jsonwebtoken</groupId>
<artifactId>jjwt</artifactId>
<version>0.9.1</version>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

3 Configure JWT Token Utility:

3.1 Create a utility class to generate, validate, and parse JWT tokens.

3 Configure Security Filter:

- Implement a filter that intercepts requests and checks for the JWT token’s presence and validity.

4 Define Roles:

Use @PreAuthorize or security configurations to restrict specific actions to admins.

Security Configuration

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {



@Autowired
private JwtFilter jwtFilter;



@Override
protected void configure(HttpSecurity http) throws Exception { http.csrf().disable()
.authorizeRequests()
.antMatchers("/api/auth/**").permitAll()
.antMatchers(HttpMethod.GET, "/api/products").hasAnyRole("USER", "ADMIN")
.antMatchers(HttpMethod.POST, "/api/products").hasRole("ADMIN")
.antMatchers(HttpMethod.PUT, "/api/products/{id}").hasRole("ADMIN")
.antMatchers(HttpMethod.DELETE, "/api/products/{id}").hasRole("ADMIN")
.anyRequest().authenticated()
.and()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);



http.addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);
}



@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
// Define in-memory or JDBC-based authentication here

}
```

Database Connectivity

Configure Database spring.datasource.url=jdbc:mysql://localhost:3306/supermarket spring.datasource.username=root spring.datasource.password=your_password spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver spring.jpa.hibernate.ddl-auto=update spring.jpa.show-sql=true

Define Entities

import jakarta.persistence.*;
import lombok.AllArgsConstructor; import lombok.Data;
import lombok.NoArgsConstructor; import java.math.BigDecimal; import java.util.Date;

@Entity
23 usages & ayusharma3998 @NoArgsConstructor @AllArgsConstructor @Data public class

```
Product { @Id
@GeneratedValue (strategy = GenerationType. IDENTITY) private int id;
private String name;
private String description; private String brand; private
BigDecimal price; private String category; private
Date releaseDate; private
boolean productAvailable; private int stockQuantity; Aprivate string imageType:

@Lob
private bytell imageData;
```

APPENDIX 2

5.Database Management - Spring Boot concepts and CRUD operation

Creting a Repository Interface

```
import org.springframework.data.jpa.repository.JpaRepository;


public interface ProductRepository extends JpaRepository<Product, Long> {
// Custom queries can be added here if needed

}
```

CRUD Operations

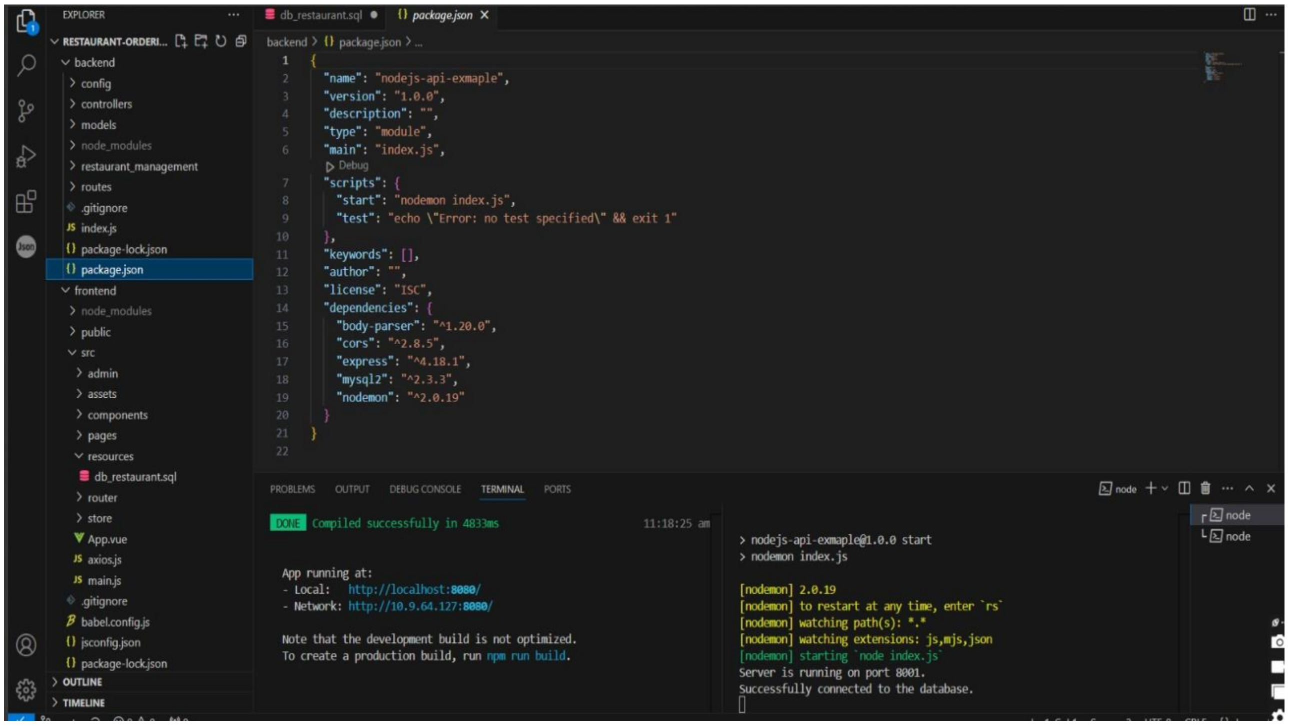
Create a Service Class

```
import org.springframework.beans.factory.annotation.Autowired; import
org.springframework.stereotype.Service;
import java.util.List; import java.util.Optional;

@Service
public class ProductService {


@Autowired
private ProductRepository productRepository;


// Create
public Product createProduct(Product product) { return productRepository.save(product);
}
```

CRUD Endpoints and Controller

```
import org.springframework.beans.factory.annotation.Autowired; import
org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;


import java.util.List;


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

@Autowired
private ProductService productService;


// Create a new product @PostMapping
public Product createProduct(@RequestBody Product product) { return
productService.createProduct(product);
}


// Get all products @GetMapping
public List<Product> getAllProducts() { return productService.getAllProducts();
}

// Get a product by ID @GetMapping("/{id}")
public ResponseEntity<Product> getProductById(@PathVariable Long id) { return
productService.getProductById(id)
.map(ResponseEntity::ok)
.orElse(ResponseEntity.notFound().build());

}



// Update a product @PutMapping("/{id}")
public  ResponseEntity<Product>      updateProduct(@PathVariable          Long    id, @RequestBody Product
updatedProduct) {
try {
Product product = productService.updateProduct(id, updatedProduct); return
ResponseEntity.ok(product);
} catch (RuntimeException e) {
return ResponseEntity.notFound().build();

}
}




// Delete a product @DeleteMapping("/{id}")
public ResponseEntity<Void> deleteProduct(@PathVariable Long id) {
productService.deleteProduct(id);
return ResponseEntity.noContent().build();

}
}
```

FINAL OUTPUT

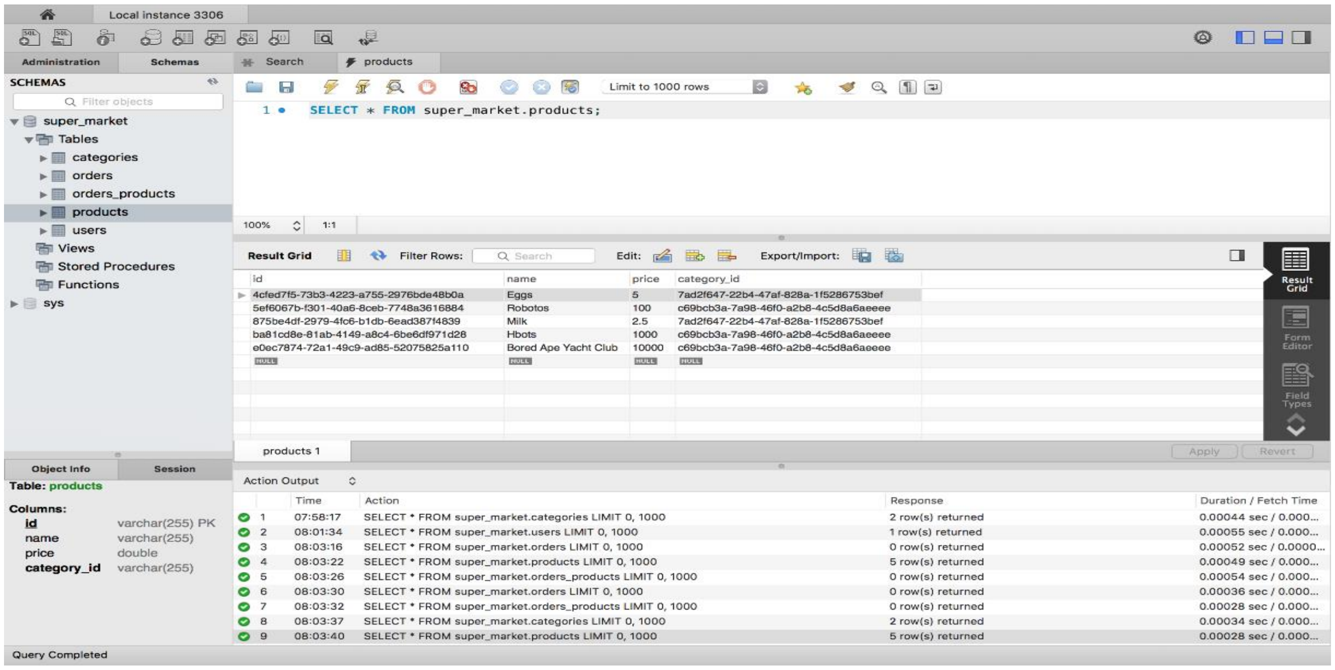


```json
{
  "name": "nodejs-api-exmaple",
  "version": "1.0.0",
  "description": "",
  "type": "module",
  "main": "index.js",
  "scripts": {
    "start": "nodemon index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.20.0",
    "cors": "^2.8.5",
    "express": "^4.18.1",
    "mysql2": "^2.3.3",
    "nodemon": "^2.0.19"
  }
}
```

```
DONE  Compiled successfully in 4833ms                    11:18:25 am

  App running at:
  - Local:   http://localhost:8080/
  - Network: http://10.9.64.127:8080/

  Note that the development build is not optimized.
  To create a production build, run npm run build.
```

```
> nodejs-api-exmaple@1.0.0 start
> nodemon index.js

[nodemon] 2.0.19
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Server is running on port 8001.
Successfully connected to the database.
```

Fig 11 Package.json for adding products in cart



```sql
SELECT * FROM super_market.products;
```

id	name	price	category_id
4cfed7f5-73b3-4223-a755-2976bde48b0a	Eggs	5	7ad2f647-22b4-47af-828a-1f5286753bef
5ef6067b-f301-40a6-8ceb-7748a3616884	Robotos	100	c69bcb3a-7a98-46f0-a2b8-4c5d8a6aeeee
875be4df-2979-4fc6-b1db-6ead387f4839	Milk	2.5	7ad2f647-22b4-47af-828a-1f5286753bef
ba81cd8e-81ab-4149-a8c4-6be6df971d28	Hbots	1000	c69bcb3a-7a98-46f0-a2b8-4c5d8a6aeeee
e0ec7874-72a1-49c9-ad85-52075825a110	Bored Ape Yacht Club	10000	c69bcb3a-7a98-46f0-a2b8-4c5d8a6aeeee
NULL	NULL	NULL	NULL

Table: products

Columns:

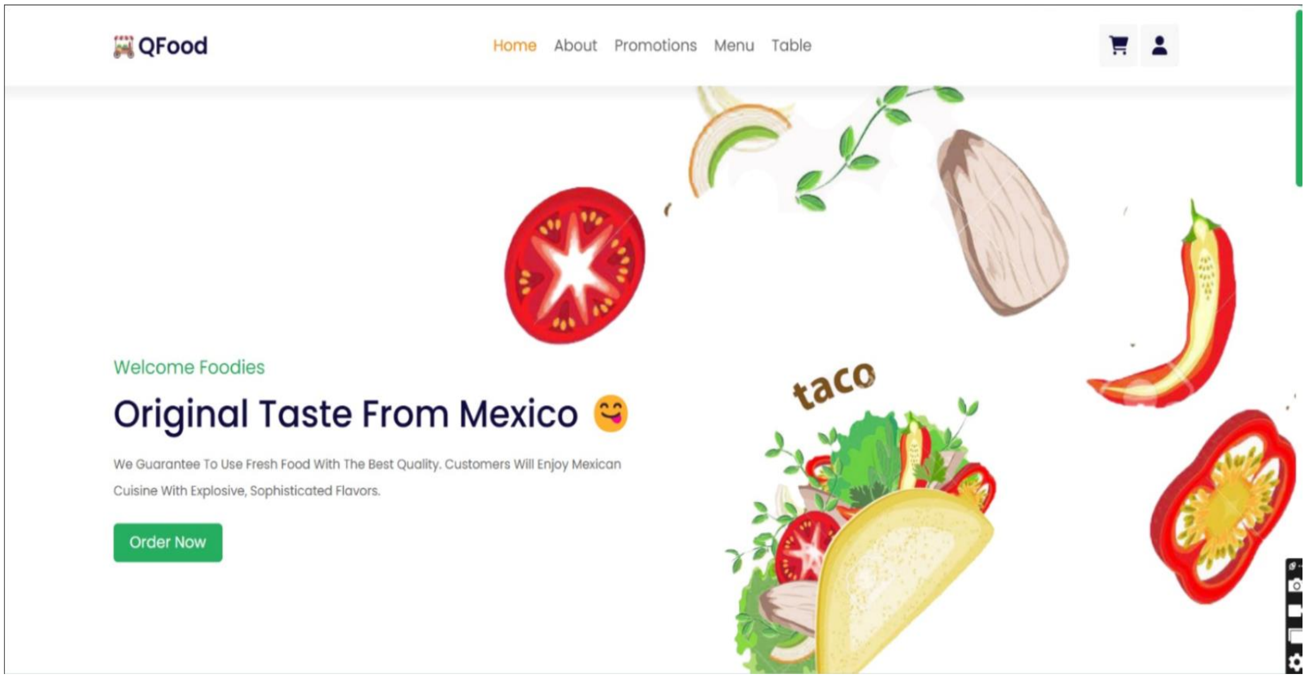
id	varchar(255) PK
name	varchar(255)
price	double
category_id	varchar(255)

Action Output

	Time	Action	Response	Duration / Fetch Time
1	07:58:17	SELECT * FROM super_market.categories LIMIT 0, 1000	2 row(s) returned	0.00044 sec / 0.000...
2	08:01:34	SELECT * FROM super_market.users LIMIT 0, 1000	1 row(s) returned	0.00055 sec / 0.000...
3	08:03:16	SELECT * FROM super_market.orders LIMIT 0, 1000	0 row(s) returned	0.00052 sec / 0.0000...
4	08:03:22	SELECT * FROM super_market.products LIMIT 0, 1000	5 row(s) returned	0.00049 sec / 0.000...
5	08:03:26	SELECT * FROM super_market.orders_products LIMIT 0, 1000	0 row(s) returned	0.00054 sec / 0.000...
6	08:03:30	SELECT * FROM super_market.orders LIMIT 0, 1000	0 row(s) returned	0.00036 sec / 0.000...
7	08:03:32	SELECT * FROM super_market.orders_products LIMIT 0, 1000	0 row(s) returned	0.00028 sec / 0.000...
8	08:03:37	SELECT * FROM super_market.categories LIMIT 0, 1000	2 row(s) returned	0.00034 sec / 0.000...
9	08:03:40	SELECT * FROM super_market.products LIMIT 0, 1000	5 row(s) returned	0.00028 sec / 0.000...

Query Completed

Fig 12 Database for all products



Welcome Foodies

Original Taste From Mexico

We Guarantee To Use Fresh Food With The Best Quality. Customers Will Enjoy Mexican Cuisine With Explosive, Sophisticated Flavors.

Fig 13 Home page for website

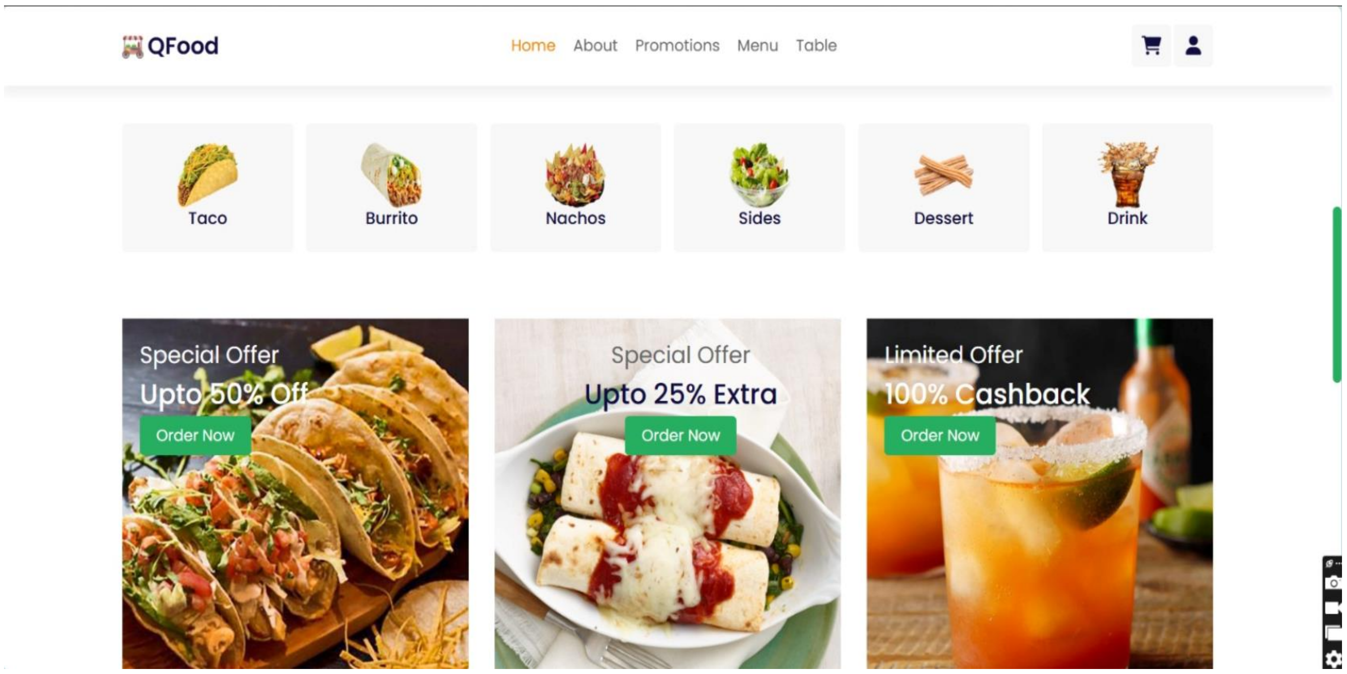
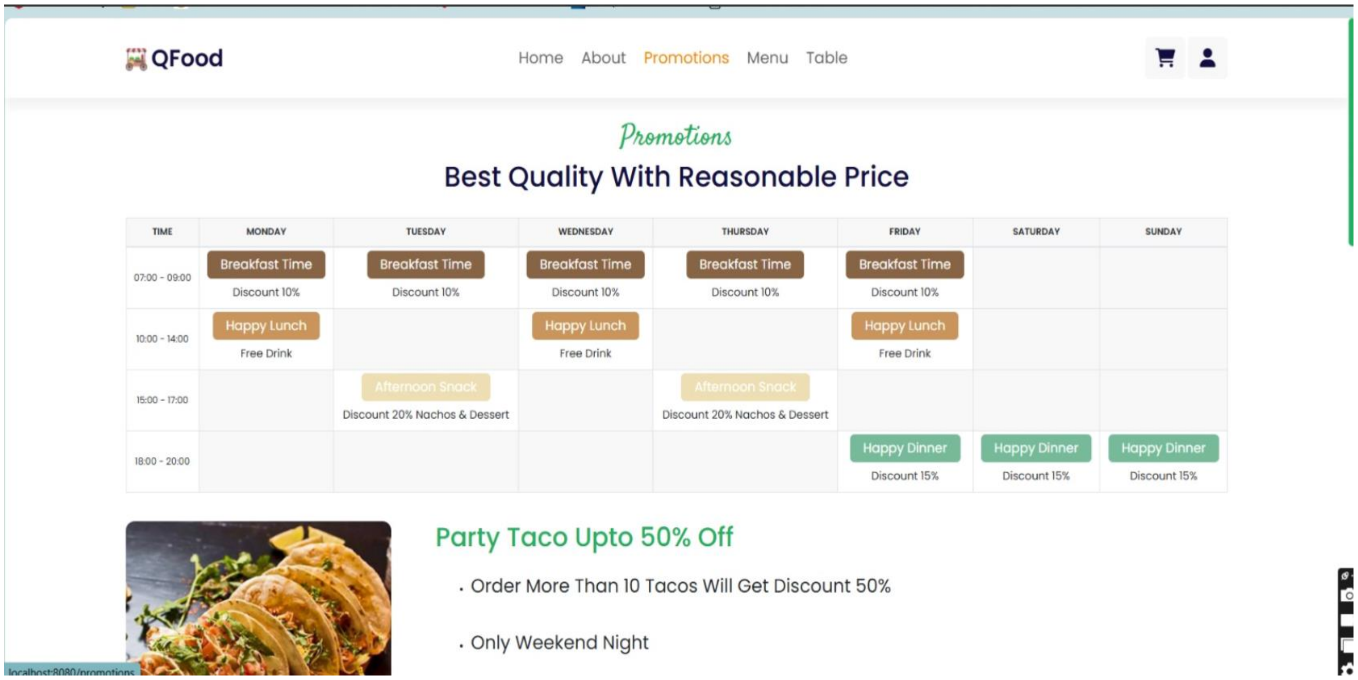


Fig 14 Dashboard for the website



Best Quality With Reasonable Price

TIME	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY	SUNDAY
07:00 - 09:00	Breakfast Time Discount 10%	Breakfast Time Discount 10%	Breakfast Time Discount 10%	Breakfast Time Discount 10%	Breakfast Time Discount 10%		
10:00 - 14:00	Happy Lunch Free Drink		Happy Lunch Free Drink		Happy Lunch Free Drink		
15:00 - 17:00		Afternoon Snack Discount 20% Nachos & Dessert		Afternoon Snack Discount 20% Nachos & Dessert			
18:00 - 20:00					Happy Dinner Discount 15%	Happy Dinner Discount 15%	Happy Dinner Discount 15%

Party Taco Upto 50% Off

- Order More Than 10 Tacos Will Get Discount 50%
- Only Weekend Night

Fig 15 Billing, Customer and timetable of products on website

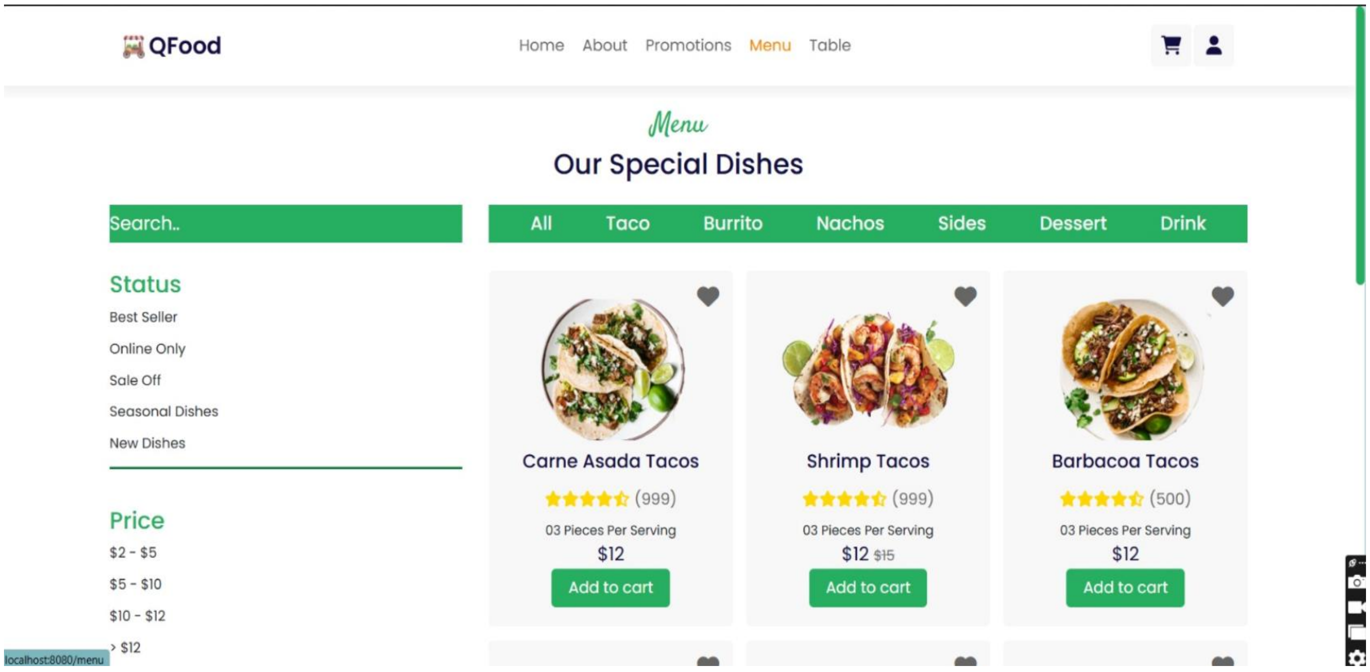


Fig 16 Our special dishes and menu items on the website