ShopEZ: E-commerce Application using MERN Stack

Submitted by

GOWTHAMAN K (Reg No.212821104003)

RAGUL S(Reg No.212821104011)

PREETHI S (Reg No.212821104010)

NATHISHA T (Reg No.212821104009)

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

JAYA COLLEGE OF ENGINEERING AND TECHNOLOGY , POONAMALLEE

ANNA UNIVERSITY :: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report on **"ShopZ-Ecommerce Application"** is the Bonafide record of work done by **GOWTHAMAN K** (Reg No. 212821104003) from the **Department of Computer Science and Engineering**, submitted in partial fulfillment of the requirements for the award of the degree of **Bachelor of Engineering** under **Anna University, Chennai**.

Head of the Department

Internal Guide

Internal Examiner

External Examiner

Abstract

In today's digital landscape, e-commerce has become a cornerstone of modern retail, bridging the gap between consumers and businesses through innovative technology. The **Shopez-Ecommerce application** is designed to provide a seamless shopping experience, offering an intuitive platform that caters to diverse consumer needs. With its user-friendly interface and robust functionality, Shopez ensures a hassle-free journey from product discovery to checkout, making online shopping more accessible and efficient.

At the heart of Shopez lies an advanced product recommendation engine, powered by machine learning algorithms, which personalizes the shopping experience for users based on their preferences and browsing behavior. This feature, combined with a secure and efficient payment gateway, ensures customer satisfaction and trust. Additionally, Shopez supports a wide array of products and vendors, creating a diverse marketplace that empowers small and large-scale sellers alike.

To meet the challenges of modern e-commerce, the application integrates analytics-driven inventory management, real-time order tracking, and streamlined logistics. These functionalities enhance operational efficiency and ensure timely delivery, solidifying Shopez as a reliable partner for both customers and businesses. Furthermore, the platform is equipped with advanced security measures, ensuring the confidentiality and integrity of user data.

Shopez is not just an e-commerce platform; it is a comprehensive solution designed to adapt to the evolving needs of the digital marketplace. By leveraging cutting-edge technology and a customer-centric approach, Shopez aims to redefine the online shopping experience, fostering growth and innovation in the e-commerce industry.

Table of Contents

1. Introduction

The **E-Commerce Application** developed in Naan Madhulvan leverages the MERN stack (MongoDB, Express.js, React, and Node.js) to deliver a dynamic and engaging online shopping experience. This platform is designed to bridge the gap between customers and a variety of sellers, offering an efficient, scalable, and user-friendly digital shopping environment. With features that include secure payments, a robust product catalog, and real-time order tracking, the application aims to provide a comprehensive e-commerce solution suitable for both small businesses and larger online marketplaces.

Team Structure and Roles

Team Member	Role
GOWTHAMAN K	Lead (Oversees and contributes to all MERN stack development.
RAGUL S	Developed Frontend and Backend
PREETHI S	Designed User Interface
NATHISHA T	Test the Interface

2. Project Overview

The main objectives of this e-commerce project are to:

- **Create a Seamless Shopping Experience**: Ensure an intuitive user interface where customers can effortlessly browse, search, and purchase products.
- **Enable Seller Management**: Allow sellers to list products, update details, manage inventory, and track sales.
- **Ensure Secure Transactions**: Integrate secure payment options to protect customer and transaction data.
- **Build Scalability**: Develop an architecture capable of handling increasing numbers of users, products, and transactions.
- **Provide Admin Control**: Enable an admin dashboard for easy management of users, products, orders, and system-wide settings.

Scope:
The application is built to serve a broad range of customers and businesses, providing features such as personalized product recommendations, shopping cart functionality, order management, and an efficient checkout process. It's designed for multi-device compatibility, making it accessible on both desktop and mobile.

3. Architecture

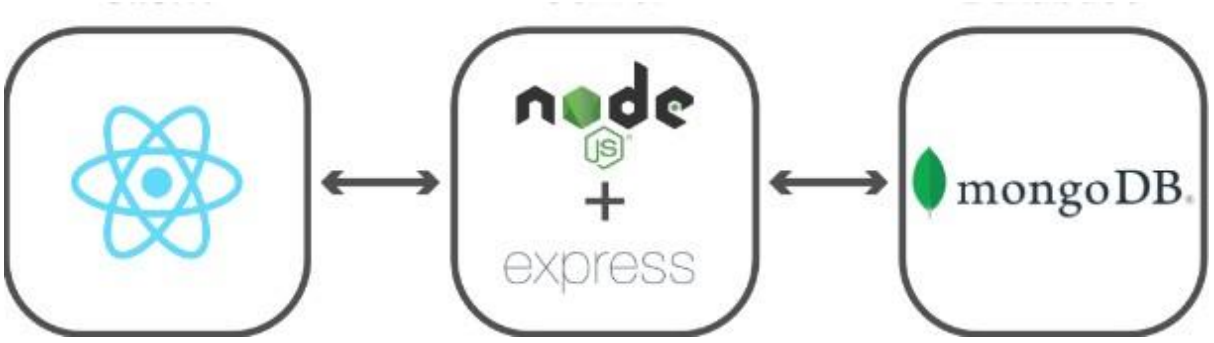
The **MERN Stack** was chosen for this project due to its powerful combination of MongoDB, Express.js, React, and Node.js, each of which contributes to the application's speed, scalability, and performance:

- **MongoDB**: A NoSQL database used to store user, product, order, and transaction data. MongoDB's flexible schema design is ideal for handling diverse and evolving e-commerce data.

- **Express.js**: A back-end framework that provides a lightweight, flexible setup for handling API requests, data routing, and middleware integration.

- **React**: A front-end library that enables the creation of a dynamic and responsive user interface, ensuring an intuitive and engaging experience across devices.

- **Node.js**: A server-side platform that allows for scalable networking applications, efficiently handling concurrent requests and enabling real-time operations.



Together, these technologies ensure a high-performance application with a fast, interactive UI and a reliable back-end infrastructure.

4. Setup Instructions

1. Prerequisites

Ensure you have the following installed:

- **Node.js (v16 or above) – Download**
- **MongoDB (local or cloud) – Download**

2. Clone the Repository

git clone <repository-url>

cd <project-folder-name>

3. Install Dependencies

Run the following commands in the respective directories:

Back End:

cd server

npm install

Front End:

```
cd client

npm install
```

5. Folder Structure

1. Client

```
client/
├── public/
│   ├── index.html       -- Main HTML file for the React app
│   ├── favicon.ico      -- App favicon
│   └── assets/          -- Static assets like images and fonts
├── src/
│   ├── components       -- Reusable React components
│   ├── pages/           -- Main page components
│   ├── context/         -- Context API files for state management
│   ├── hooks/           -- Custom React hooks
│   ├── utils/           -- Utility functions
│   ├── App.js           -- Main app component
│   ├── index.js         -- Entry point of the React app
│   ├── styles/          -- Global and component-specific styles
│   └── config/          -- Configuration files
```

2. Server

server/

```
├── config/
│   ├── db.js                -- MongoDB connection setup
│   ├── cloudinary.js        -- Cloudinary setup for image uploads
│   └── payment.js           -- Payment gateway configuration
├── controllers/             -- Handles request logic for various routes
│   ├── authController.js    -- User authentication logic
│   ├── productController.js -- Product CRUD operations
│   ├── orderController.js   -- Order management
├── models/                  -- MongoDB schemas and models
│   ├── User.js              -- User schema
│   ├── Product.js           -- Product schema
│   └── Order.js             -- Order schema
├── routes/                  -- Express route definitions
│   ├── authRoutes.js        -- Routes for authentication
│   ├── productRoutes.js     -- Routes for product management
│   ├── orderRoutes.js       -- Routes for orders
├── middleware/              -- Custom middleware
│   ├── authMiddleware.js    -- Protects private routes
```

```
│   ├── errorHandler.js        -- Handles errors globally
├── utils/                     -- Utility functions
│   ├── sendEmail.js           -- Helper for email notifications
│   └── jwtHelper.js           -- Token generation and validation
├── .env                       -- Environment variables
├── server.js                  -- Entry point of the Node.js server
├── package.json               -- Dependencies and scripts
```

6. Running The Application

Back End:

```
cd server
npm start
```

Front End:

```
cd client
npm start
```

Access the Application

- **Frontend:** http://localhost:3000
- **Backend API:** http://localhost:5000

7. API Documentation

1. User Registration – Endpoint (POST / api / auth / register)

Request Body - { "name": "John", "email": "john@example.com", "password": "123456" }

Response - { "success": true, "message": "User registered successfully" }

2. User Login – Endpoint (POST / api / auth / login)

Request Body - { "email": "john@example.com", "password": "123456" }

Response - { "success": true, "token": "JWT-TOKEN" }

3. Getting all Products – Endpoint (GET / api / products)

Response - [

{ "id": "p1", "name": "Product A", "price": 100 },

{ "id": "p2", "name": "Product B", "price": 150 }

]

4. Getting Products by ID – Endpoint (GET / api / products / :id)

Response - { "id": "p1", "name": "Product A", "price": 100, "stock": 50 }

5. Adding new Products – Endpoint (POST / api / products)

Request Body - { "name": "Product C", "price": 200, "stock": 30 }

Response - { "success": true, "message": "Product added successfully" }

6. Cart – Endpoint (POST / api / cart)

Request Body - { "productId": "p1", "quantity": 2 }

Response - { "success": true, "message": "Added to cart" }

7. Orders – Endpoint (GET / api / orders)

Request Body - { "address": "123 Main St", "paymentMethod": "Credit Card" }

Response - { "success": true, "orderId": "o1", "status": "Pending" }

8. Payment - (POST / api / payments /process)

Request Body - { "orderId":"o1", "paymentDetails": { "cardNumber": "4242424242424242" } }

Response - { "success": true, "message": "Payment successful" }

8. Authentication

To provide a secure shopping experience, the application incorporates **User Authentication and Authorization**:

- **User Registration and Login**: New users can register with their email address and password, while existing users can log in securely. Passwords are hashed and stored securely to protect user information.

- **Role-Based Access Control**: Implemented to distinguish between customers, sellers, and administrators, each with different levels of access. Customers can browse and make purchases, sellers can manage their products, and admins have full control over the platform's content and users.

- **Session Management**: Secure sessions are maintained to keep users logged in without compromising security. Sessions automatically expire after a set period of inactivity.

- **Third-Party Authentication (optional)**: Integration with social media logins (e.g., Google, Facebook) for a seamless sign-in experience if required.

The **JWT (JSON Web Token)** is used to manage user sessions securely, enabling stateless authentication across the application.

9. User Interface Design

The **User Interface (UI) Design** of the e-commerce application focuses on delivering an intuitive and visually appealing shopping experience that caters to users of all technical backgrounds. The design prioritizes accessibility, ease of navigation, and responsiveness across different devices (desktop, tablet, and mobile). Key elements include:

- **Homepage**: Displays featured products, categories, and promotional banners to engage users from the start.
- **Product Pages**: Offers detailed product descriptions, images, reviews, and options for adding items to the cart or wishlist.
- **Search and Filter Options**: Allows users to search for products by keywords and filter results based on categories, price, rating, and other criteria.
- **User Dashboard**: Provides a personalized space where users can view their order history, update profile information, and manage saved items in their wishlist.

- **Responsive Design**: Ensures that the platform adjusts seamlessly to various screen sizes, offering a consistent experience on mobile and desktop.

The UI was designed using **React components** to create reusable and modular code, enabling faster development and easier maintenance of the user interface.

10. Product Management

The **Product Management** module allows sellers and administrators to manage products efficiently:

- **Add, Update, and Delete Products**: Sellers can add new products, modify existing listings, and remove items that are out of stock or discontinued.

- **Inventory Management**: Enables sellers to track stock levels, update quantities, and receive alerts when inventory is low.

- **Product Categorization**: Allows products to be grouped by categories, tags, and filters, making it easier for users to browse and find items.

- **Product Details**: Each product entry includes attributes such as name, description, images, price, and specifications. The system also supports uploading multiple images for each product.

The product management functionality is handled on the back-end using **Express and MongoDB**, while the front-end uses React to provide sellers and admins with a user-friendly interface.

11. Shopping Cart and Wishlist Functionality

The **Shopping Cart and Wishlist** features enhance user convenience and engagement:

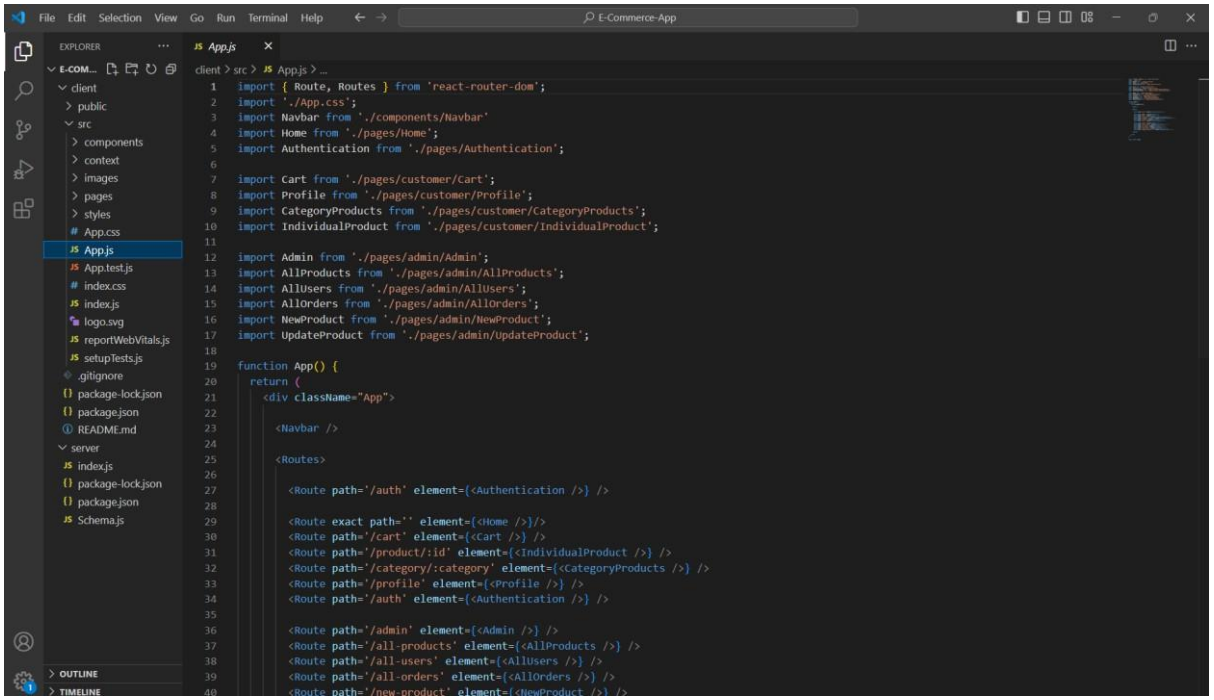
- **Shopping Cart**: Allows users to add items they wish to purchase immediately. Users can adjust quantities, view the total price, and proceed to checkout from the cart page.

- **Wishlist**: Enables users to save products they are interested in but may not be ready to purchase immediately. This list is stored in the user's profile and can be accessed anytime.

- **Persistent Cart and Wishlist**: Both cart and wishlist contents are saved in the database and persist across sessions, allowing users to access their saved items even after logging out.

- **Real-Time Updates**: The cart and wishlist are updated in real time to reflect any changes made by the user.

React handles the dynamic updates to the cart and wishlist on the front-end, while MongoDB stores the data on the back-end, providing a seamless user experience.



12. Search and Filtering Options

The **Search and Filtering Options** are designed to help users find products quickly and efficiently:

- **Search Bar**: A prominently placed search bar allows users to search for products by keywords, such as product names, categories, or specific attributes.

- **Advanced Filters**: Users can filter search results by various criteria, including category, price range, brand, rating, and availability. These filters enhance the shopping experience by narrowing down the product selection to items that match the user's preferences.

- **Sorting Options**: Users can sort search results by popularity, price (low to high or high to low), and customer ratings.

- **Real-Time Search Suggestions**: As users type, search suggestions dynamically populate, helping them find products faster.

These search and filtering functionalities are powered by MongoDB's flexible querying capabilities, and React manages the user interface for smooth, real-time updates.

13. Payment Gateway Integration

The **Payment Gateway Integration** allows users to securely complete transactions on the platform:

- **Supported Payment Methods**: Integration with popular and secure payment gateways such as Stripe, PayPal, or Razorpay (depending on project needs) provides multiple payment options, including credit cards, debit cards, and digital wallets.

- **Secure Transactions**: User payment information is securely processed through encrypted connections, ensuring compliance with PCI-DSS (Payment Card Industry Data Security Standard) requirements.

- **Order Confirmation**: After a successful payment, users receive an order confirmation, and the order details are saved in the database.

- **Refund and Cancellation Support**: The platform supports refunds and cancellations if needed, with automatic updates to the payment gateway.

This integration is handled on the back-end using Node.js and Express, while the front-end uses React to provide a smooth checkout experience for users.

14. Order Processing and Tracking

The **Order Processing and Tracking** feature manages orders from placement to delivery:

- **Order Placement**: Once payment is confirmed, the order details, including products, quantities, prices, and user information, are saved in the database.

- **Order Status Updates**: The system updates the status of each order (e.g., “Processing,” “Shipped,” “Delivered”) and sends notifications to the user.

- **Order Tracking**: Users can view their order history and track the status of current orders from their profile dashboard.

- **Inventory Adjustment**: After an order is placed, the system automatically updates inventory quantities, ensuring accurate stock levels.

This functionality is built using MongoDB to store order data, and Express and Node.js handle order processing logic, with React providing real-time order updates on the front end.

15. Admin Dashboard Features

The **Admin Dashboard** provides administrators with tools to manage and monitor platform activities:

- **User Management**: Admins can view, edit, or deactivate user accounts as needed.
- **Product Management**: Admins can add new products, update product details, and manage inventory directly from the dashboard.
- **Order Management**: The dashboard provides an overview of all orders, allowing admins to update order statuses, process refunds, and resolve customer issues.
- **Analytics and Reports**: Key metrics such as total sales, number of active users, and inventory levels are displayed, helping admins make informed decisions.
- **Security and System Settings**: Admins have access to security settings and platform configurations, allowing them to manage access controls and other system parameters.

This dashboard is built using React for an interactive interface and is backed by Node.js and Express for managing data and user interactions.

16. Security Measures and Data Protection

Ensuring the security of user data and protecting against threats is a top priority. The application incorporates several **Security Measures and Data Protection** practices:

- **User Data Encryption**: Sensitive data, such as passwords and payment information, is encrypted before being stored in the database.

- **Secure Authentication**: JWT (JSON Web Token) authentication and strong password policies are implemented to prevent unauthorized access.

- **Regular Audits and Vulnerability Checks**: The application undergoes regular security audits to detect and fix vulnerabilities.

- **Data Privacy Compliance**: The platform adheres to data protection standards like GDPR, ensuring that user data is handled responsibly and transparently.

- **Role-Based Access Control (RBAC)**: Different levels of access are assigned to users, sellers, and admins, minimizing the risk of data misuse.

These security features are managed through Express.js and Node.js, while MongoDB's access control settings help protect the data at the database level.

17. Performance Optimization

Performance optimization is crucial for ensuring a fast and seamless user experience on the e-commerce platform. The following techniques were implemented to enhance application speed and responsiveness:

- **Efficient Database Queries**: MongoDB indexes were used to speed up search queries, and redundant data fetching was minimized to improve response times.

- **Caching**: Frequently accessed data, such as product lists and user information, is cached using Redis or in-memory storage, reducing load on the database and speeding up page loading times.

- **Lazy Loading and Code Splitting**: Only essential components are loaded initially, with additional components loaded as needed. This technique, applied through React, significantly reduces initial load time.

- **Image Optimization**: Product images were compressed and resized to minimize file size without compromising quality, reducing the bandwidth required to load product pages.

- **Content Delivery Network (CDN)**: Static files and images are served from a CDN to decrease latency and improve loading times for users in different geographical locations.

These optimizations ensure the application is responsive and performs well even under high user loads.

18. Testing and Validation

To ensure a high-quality and bug-free application, comprehensive **Testing and Validation** strategies were implemented:

- **Unit Testing**: Key functions and components were tested individually to confirm that each part of the application behaves as expected.

- **Integration Testing**: Testing was performed on modules that interact with each other, such as the checkout process, to ensure smooth workflows and data flow between components.

- **End-to-End (E2E) Testing**: Full user journeys, from product browsing to order completion, were tested to simulate real user interactions and validate the platform's functionality.

- **Performance Testing**: Load and stress tests were conducted to measure the application's responsiveness and stability under heavy traffic conditions.

- **Security Testing**: Tests were conducted to identify potential vulnerabilities, such as SQL injection, cross-site scripting (XSS), and data exposure risks.

Testing was conducted using tools like Jest, Mocha, and Cypress, with automated tests integrated into the CI/CD pipeline to catch issues early in the development process.

19. Deployment and Scalability Considerations

The e-commerce application was designed with scalability and reliable deployment in mind:

- **Containerization**: The application is containerized using Docker, enabling consistent and reliable deployment across different environments.

- **Cloud Hosting**: Hosted on cloud platforms such as AWS or Heroku, which offer scalable infrastructure and allow resources to be easily adjusted based on user demand.

- **CI/CD Pipeline**: A continuous integration and deployment pipeline was set up to automate testing and deployment, ensuring that updates reach users quickly and reliably.

- **Horizontal and Vertical Scaling**: The architecture supports both horizontal scaling (adding more servers) and vertical scaling (increasing resources of existing servers), allowing the application to handle growing user numbers and traffic without sacrificing performance.

- **Database Scalability**: MongoDB's sharding and replication features were utilized to ensure the database can handle large volumes of data and high query loads.

These deployment and scalability measures ensure that the application can handle both current and future traffic demands effectively.

20. Future Enhancements and Recommendations

Several **Future Enhancements and Recommendations** have been identified to further improve the application and provide additional value to users:

- **AI-Powered Recommendations**: Implement machine learning algorithms to provide personalized product recommendations based on user behavior and purchase history.

- **Enhanced Analytics Dashboard**: Expand the admin dashboard with more detailed analytics, such as user demographics, peak shopping times, and sales trends.

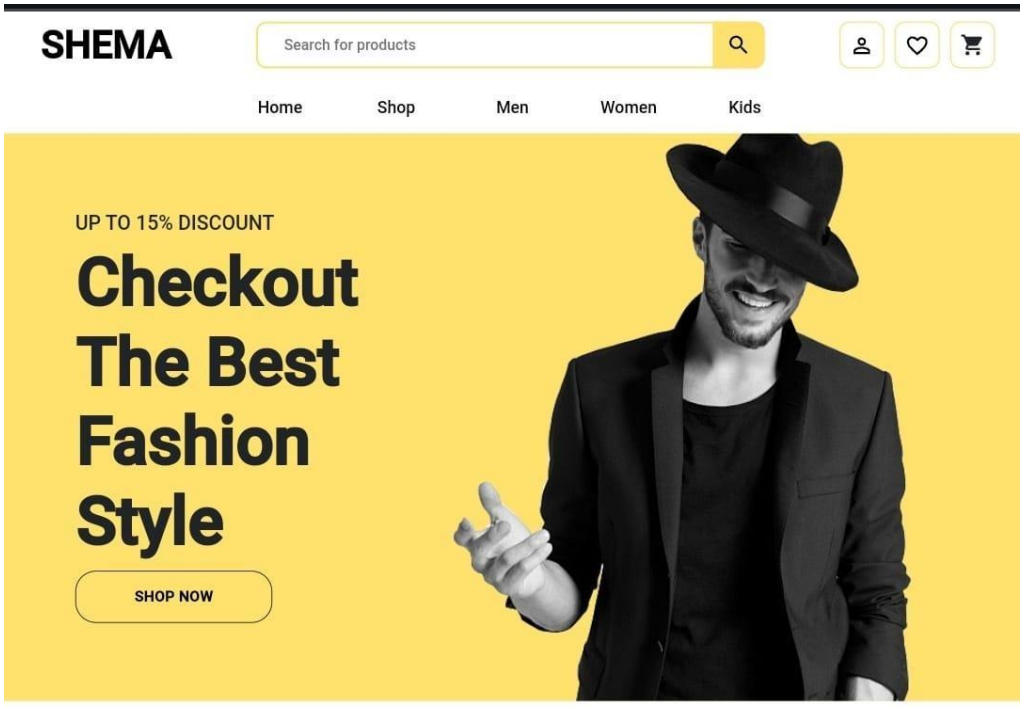
- **Chatbot for Customer Support**: Integrate a chatbot to provide instant assistance to users, improving customer support efficiency and user satisfaction.

- **Mobile App Development**: Create a dedicated mobile app for the platform, offering users a more optimized shopping experience on smartphones and tablets.

- **Loyalty Program**: Implement a rewards and loyalty program to incentivize repeat purchases, boosting customer retention and engagement.

These enhancements will help the platform stay competitive, improve user experience, and expand its functionality to meet evolving market demands.

21. APPENDIX



Featured Categories

Featured Items Show all →

Versace
Odissea chunky leather trainers
$699

Versace
Medusa bootcut jeans
$503

Amina Muadi
Sita leather sandals
$937

Bobo Choses
Crew-neck sweatshirt
$67

Stella Kids
Drawstring cargo track pants
$878

NIKE
Air Force Sneakers
$161

Off-white
Intarsia pencil skirt
$689

VEJA Kids
Nautico suede sneakers
$96