សាកលវិទ្យាល័យភូមិន្ទភ្នំពេញ

ROYAL UNIVERSITY OF PHNOM PENH

មហាវិទ្យាល័យវិស្វកម្ម

FACULTY OF ENGINEERING

Developing a full-stack E-commerce application with MERN Stack

Student Name:

SOTHUN Darachhat

NAI Chhannut

Supervisor:

CHHIM Bunchhun

Subject:

Project Practicum

Information Technology Engineering

2024

ABSTRACT

Chhim Bunchhun
Royal University of Phnom Penh
Faculty of Engineering
Bachelor in Information Technology Engineering

SOTHUN DARACHHAT || NAI CHHANNUT :
Developing a full-stack E-commerce application with MERN Stack

Project Practicum’s Report 30 pages
December 2024

E-commerce is becoming a strong and important way of making business This report presents the implementation and technology behind an e-commerce application. This application was created to make use of online shopping's advantages, expand the target market, increase sales, and improve the entire consumer experience. The study then explores the application's technology framework, system design, and implementation process. Features of this application are user registration and login system, authentication process, product list, and shopping cart functionalities. The application was developed using technologies such as React, NodeJS, MongoDB, and Express.

At the end of the report will be clearer about the responsiveness of the application.

TABLE OF CONTENTS

Page

1 INTRODUCTION

E-commerce has revolutionized the way businesses interact with their customers, providing new opportunities for growth and success. With the increase in online shopping, particularly after the COVID-19 pandemic, the demand for efficient and scalable eCommerce platforms has risen significantly. Small and local businesses, in particular, can overcome the limitations of physical stores by creating an online presence to reach a wider customer base.

In Cambodia, the rapid adoption of the internet and mobile technologies has spurred the growth of eCommerce platforms. As businesses seek to expand online, the challenge lies in creating a website that is not only user-friendly but also secure and scalable. Understanding the components that make up an eCommerce website and how to integrate them effectively is critical for developers aiming to build successful online stores.

This report presents the development of an eCommerce website using the MERN stack, a powerful combination of MongoDB, Express.js, React.js, and Node.js. By leveraging these technologies, the project aims to build a flexible, dynamic, and responsive eCommerce platform that supports features such as user authentication, product management, order tracking, and secure payments. The focus is on utilizing the strengths of the MERN stack to create a robust platform capable of meeting the demands of modern online shopping.

Through this project, the report will demonstrate how an eCommerce platform can be built using the MERN stack, covering its architecture, technologies, and the implementation of key features such as product catalogs, shopping carts, and secure user logins. The goal is to show how the MERN stack provides a complete solution for building a modern, scalable, and efficient eCommerce website.

2. METHODOLOGY

The development of the eCommerce website using the MERN stack follows a systematic approach, with each component of the stack playing a vital role in the creation of a robust and scalable platform. The methodology involves the integration of MongoDB for data storage, Express.js for server-side operations, React.js for the front-end interface, and Node.js to serve as the back-end framework. This section outlines the key steps involved in the development process and explains the advantages of using the MERN stack in building an efficient eCommerce platform.

2.1 Benefits of using the MERN Stack

The MERN stack is widely adopted for full-stack JavaScript development, consisting of MongoDB, Express.js, React.js, and Node.js. These technologies, when combined, provide several advantages for developers looking to build modern web applications.

Unified Language: One of the primary benefits of the MERN stack is the use of JavaScript for both the client-side and server-side development. This unified language simplifies development as developers need not switch between languages, allowing for faster development and easier maintenance.

Scalability and Flexibility: MongoDB offers a flexible NoSQL database that scales horizontally, enabling the eCommerce platform to grow without performance issues. This scalability is vital for eCommerce websites that expect high traffic and large amounts of data.

React for Dynamic UI: React.js allows developers to build highly interactive and dynamic user interfaces that respond quickly to user interactions. React's component-based structure makes it easier to maintain and reuse code, enhancing the speed of development.

Efficient Backend with Node.js and Express.js: Node.js enables high-performance server-side applications with its non-blocking I/O model, which is especially important for handling multiple simultaneous user requests in an eCommerce environment. Express.js simplifies routing and middleware management, making the development process more efficient.

Together, these components provide a highly efficient stack for building scalable, responsive, and maintainable eCommerce websites.

2.2 MongoDB Database

MongoDB is a NoSQL database that uses a flexible schema and stores data in JSON-like format, which makes it ideal for handling unstructured or semi-structured data. This is particularly beneficial for eCommerce platforms, where product information, user data, and transaction history can vary greatly in structure.

Dynamic Schemas: MongoDB's flexibility allows developers to adapt the database schema to the evolving needs of the business without downtime. For example, the product data model can evolve over time to include new attributes without the need for database migration scripts.

Scalability: MongoDB can scale horizontally by sharding data across multiple servers, which is crucial for handling large volumes of traffic and data as the eCommerce website grows.

Integration with Mongoose: The use of Mongoose, an Object Data Modeling (ODM) library for MongoDB, makes it easier to interact with the database by providing a straightforward way to define models, query data, and perform operations like CRUD (Create, Read, Update, Delete).

MongoDB plays a crucial role in this eCommerce platform by managing products, users, orders, and other critical data.

2.3 Basic Description of Express

Express.js is a web application framework for Node.js that provides a robust set of features to build web and mobile applications. It simplifies the creation of server-side applications and APIs by handling requests and responses, routing, and middleware management.

Routing: Express enables developers to define routes for various HTTP methods (GET, POST, PUT, DELETE), making it easy to handle requests from the front-end. This is critical in an eCommerce website where users perform various actions like viewing products, adding items to the cart, and placing orders.

Middleware: Express supports middleware, which allows developers to insert custom functions that process requests before they reach the route handlers. This is useful for tasks such as validating user input, checking for authentication, and logging requests.

Express’s simplicity and flexibility make it an ideal choice for building the server-side of the eCommerce platform, enabling efficient handling of HTTP requests and responses.

2.4 Basic Description of React

React is a JavaScript library used to build user interfaces, especially for single-page applications. React is based on the concept of components, which are reusable, self-contained units of code that render UI elements based on input data.

Component-Based Architecture: React’s component-based architecture allows for the modular development of the front-end. Components like product listings, shopping carts, and user profiles can be built and reused across the website.

Virtual DOM: React uses a virtual DOM to optimize performance. Rather than updating the entire DOM every time there is a change, React creates an in-memory representation and updates only the parts of the UI that need to be changed, making the website more responsive and efficient.

React Router: React Router is used for navigation between pages on the front-end without reloading the page. This is critical for providing a smooth user experience on an eCommerce platform where users may browse products, manage their cart, and make payments without page reloads.

React’s ability to build dynamic, interactive user interfaces is a key component in the success of the eCommerce website.

2.5 Basic Description of Node

Node.js is a JavaScript runtime built on Chrome's V8 engine that allows developers to build fast, scalable server-side applications. It is well-suited for real-time applications and APIs, making it an ideal choice for building the backend of an eCommerce platform.

Asynchronous and Non-Blocking: Node.js uses a non-blocking I/O model, meaning it can handle multiple requests concurrently without waiting for one operation to complete before starting another. This is essential for eCommerce websites that handle high volumes of user requests simultaneously.

Real-Time Capabilities: Node.js can handle real-time features like order tracking and live notifications, which are crucial in modern eCommerce platforms.

Large Ecosystem of Modules: Node.js has a vast ecosystem of modules available via npm (Node Package Manager), allowing developers to quickly integrate third-party functionality into their applications, such as payment gateways and user authentication.

Node.js enables the back-end of the eCommerce platform to be both scalable and efficient, making it a critical component in handling large amounts of data and traffic.

2.6 Explanation of Backend Tools

The backend of the eCommerce website is supported by a variety of tools and libraries that enhance functionality, security, and performance. These tools provide solutions for managing user authentication, handling file uploads, securing sensitive data, and more. Below are the key backend tools used in the development of this eCommerce platform:

2.6.1 CORS (Cross-Origin Resource Sharing)

CORS is a security feature implemented by browsers that restricts web pages from making requests to a domain different from the one that served the web page. In the context of an eCommerce website, enabling CORS allows the frontend (React.js) running on a different server to make API requests to the backend (Node.js/Express.js) without running into security issues.

Enabling CORS: Express.js supports CORS through the cors package, which is configured to allow requests from specified domains. This is essential for the eCommerce platform, as the frontend and backend may be hosted on different servers or ports during development and production.

2.6.2 Doteny

Dotenv is a zero-dependency module that loads environment variables from a .env file into process.env. This is crucial for securely managing configuration variables such as API keys, database URLs, and authentication secrets.

Environment Variable Management: By storing sensitive data in environment variables, the eCommerce platform ensures that credentials are not exposed in the source code. The use of Dotenv also allows for easy switching between different environments (development, production) by simply changing the .env file.

2.6.3 JSON Web Tokens (JWT)

JWT is a popular method for securely transmitting information between parties as a JSON object. It is widely used for handling user authentication in modern web applications.

Authentication and Authorization: In the eCommerce platform, JWT is used to verify users during login and authorize access to protected routes. After a user logs in, the server generates a JWT, which is sent to the client. This token is stored in the client (typically in localStorage) and is sent with each subsequent request to access protected resources, ensuring secure communication between the client and backend.

2.6.4 Mongoose

Mongoose is an ODM (Object Data Modeling) library for MongoDB and Node.js. It simplifies data interaction by providing a schema-based solution to model application data.

Schema Definitions: Mongoose allows the creation of schemas for each collection in MongoDB, which helps enforce data integrity and validation rules. For example, the Product schema defines the structure of product data, including fields like name, price, and description, ensuring that the data stored in the database adheres to a consistent format.

CRUD Operations: Mongoose provides built-in methods to interact with the MongoDB database, such as .find(), .save(), and .update(), making database operations easier and more efficient.

2.6.5 Multer

Multer is a middleware for handling multipart/form-data, primarily used for uploading files. In the context of the eCommerce website, Multer is used to handle the uploading of product images.

File Uploads: The eCommerce platform allows users to upload images when adding or updating product listings. Multer handles the process of saving images to the server and managing metadata like file size and type.

Storage Options: Multer supports multiple storage options, such as storing files locally or integrating with cloud storage providers like Cloudinary.

2.6.6 Nodemon

Nodemon is a tool that helps with the development of Node.js applications by automatically restarting the server when file changes are detected. This speeds up the development process, as developers don't have to manually restart the server every time they make changes to the backend code.

Development Efficiency: By using Nodemon, the backend of the eCommerce platform automatically reloads, reflecting code changes immediately without the need for manual intervention.

2.6.7 ABA PayWay

ABA PayWay is a payment gateway solution in Cambodia, used for processing online payments. It provides a secure and easy way for customers to pay for products through various payment methods, including credit cards and bank transfers.

Payment Integration: The eCommerce platform integrates ABA PayWay for seamless payment processing. Customers can securely complete their transactions, and the platform ensures that payments are handled safely and efficiently.

2.6.8 Validator

Validator is a library that provides a set of functions for validating and sanitizing user input. It is used in the eCommerce platform to ensure that data entered by users, such as email addresses and phone numbers, is in the correct format.

Data Validation: Validator helps ensure that only valid and sanitized data is processed by the backend, preventing issues related to incorrect or malicious user input.

2.6.9 Cloudinary

Cloudinary is a cloud-based image and video management service. It allows developers to upload, store, and manipulate media files with ease. In the eCommerce platform, Cloudinary is used for hosting product images, ensuring high availability and fast loading times.

Image Optimization: Cloudinary provides tools for automatically optimizing images for the web, reducing file sizes without compromising quality. This is important for improving the website's performance and user experience.

2.6.10 Bcrypt

Bcrypt is a hashing library used to hash passwords before storing them in the database. This ensures that user passwords are securely stored and protected from unauthorized access.

Password Security: Bcrypt uses a one-way hashing algorithm to convert plain-text passwords into hashed values. Even if the database is compromised, the original passwords cannot be retrieved, ensuring the security of user data.

3 SYSTEM DESIGN

System design involves structuring the eCommerce platform to meet the requirements of users while ensuring efficiency, scalability, and maintainability. This section covers the user requirements, system architecture, and database design, which form the backbone of the eCommerce website.

3.1 User Requirements

The eCommerce platform is designed to cater to the needs of both customers and administrators. Key user requirements include:

User Registration and Authentication: Customers must be able to register for an account, log in, and securely access their profiles.

Product Listing and Search: Users should be able to view and search for products based on various criteria such as name, category, and price.

Shopping Cart and Checkout: Customers should be able to add products to their shopping cart and proceed to checkout for payment.

Order Management: Users should be able to track their orders, while administrators must be able to manage order status and product availability.

Admin Panel: Admins should be able to manage product listings, update product details, and monitor transactions and customer activity.

3.1.1 Application Event Flow

Figure 1 illustrates the use case diagram of the e-commerce system, representing the interactions between various actors and the system. It captures the primary functionalities offered to users, administrators, and external systems.

Actors and Roles:

Guest:

- Can browse products, search collections, view product details, and add items to the cart.
- Has limited access and cannot proceed to checkout or place orders without registering.

User:

- Has all the privileges of a guest.
- Can register, log in, view cart contents, remove items from the cart, proceed to checkout, place orders, view order history, and track orders.

Admin:

- Manages products, orders, and their statuses.
- Updates the status of orders and tracks customer orders.

External Payment System (e.g., ABA):

- Handles the payment process for user orders.

Functionalities:

Guest:

- Browse products, search collections, and view details without logging in.

User:

- Add items to the cart, view and manage cart, proceed to checkout, and make payments.
- View order history and track placed orders.

Admin:

- Manage product listings and orders, update order statuses, and oversee the platform's operational activities.

Payment System:

- Provides payment processing integration.

Purpose:

The use case diagram ensures clear documentation of system capabilities, facilitating better understanding and development of application features.

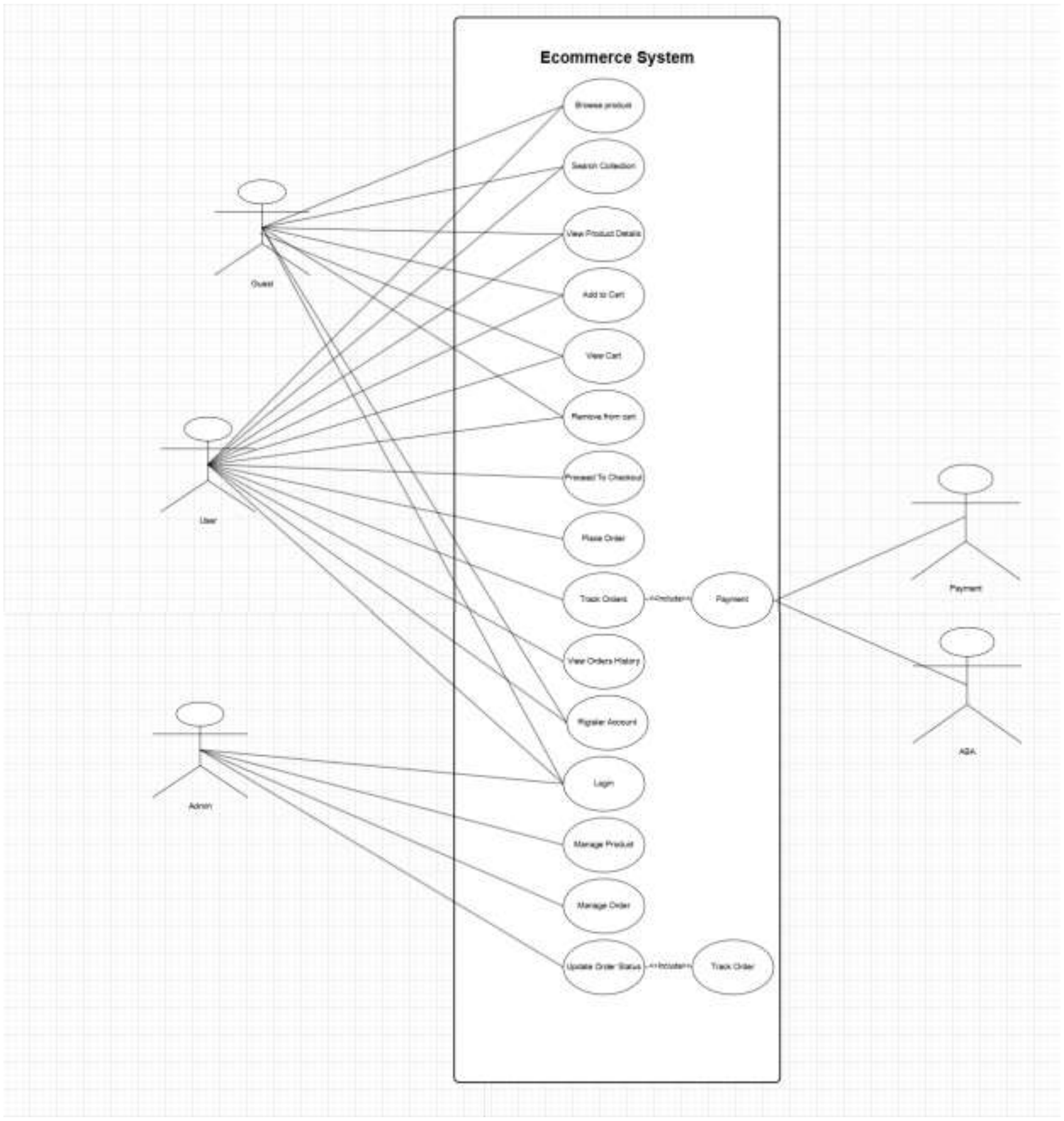

Figure 2: Use Case Diagram for E-commerce Application

3.2 System Architecture

The eCommerce platform follows a client-server architecture with a clear separation of concerns between the frontend, backend, and database. The overall structure can be described as follows:

Frontend: React.js is used to build the user interface, communicating with the backend via API calls. The frontend is responsible for displaying product information, handling user interactions, and rendering dynamic content based on user actions.

Backend: Node.js and Express.js provide the backend services, handling API requests from the frontend, interacting with the database, and processing user authentication and payments.

Database: MongoDB is used for data storage, with collections for products, users, orders, and reviews. It offers flexibility for storing and retrieving large amounts of data with ease.

3.2.1 Database Design

The database for this e-commerce application is designed to support the core functionalities of an online store, including user management, product cataloging, order processing, and more. It uses a relational database structure to efficiently manage data relationships and ensure data integrity. Below is an overview of the database design:

1. **Core Entities:**
    - **Users**: Stores information about customers, including their name, email, and password.
    - **Products**: Contains product details such as name, description, price, and associated categories.
    - **Orders**: Tracks customer orders, including the total amount, status, and payment method.
    - **Order Items**: Represents individual items in an order, including their quantity and size.
    - **Categories**: Organizes products into categories and subcategories.
    - **Cart Items**: Manages the contents of each user's shopping cart.

2. **Support Entities:**
    - **Product Images**: Stores URLs for multiple images associated with a product.
    - **Product Sizes**: Defines available sizes for each product.
    - **Order Addresses**: Stores customer shipping information, such as name, address, and contact details.
    - **Payment Methods**: Specifies payment options used in orders.
    - **Order Status**: Tracks the current state of an order (e.g., pending, shipped, delivered).

3. **Relationships:**
    - A **User** can have multiple **Orders**.

- An **Order** can contain multiple **Order Items**.
- **Products** are associated with **Categories** and may have multiple sizes and images.
- **Order Addresses** are linked to **Orders** to manage shipping details.

3.2.2 Database Entity Relationship Diagram

The Entity Relationship Diagram (Figure 1) visually represents the database structure and its relationships:

Users and Orders:

- The Users table has a one-to-many relationship with the Orders table, allowing each user to place multiple orders.
- The Orders table references Users via the userId field.

Products and Categories:

- The Products table has many-to-one relationships with Category Products and Subcategory Products, enabling hierarchical product categorization.
- Each product can have multiple entries in the Product Images and Product Sizes tables.

Orders and Order Items:

- The Orders table is linked to the Order Items table, which details the specific products, sizes, and quantities in each order.

Shipping and Payment:

- The Order Addresses table stores shipping information for each order.
- The Payment Method table identifies how the customer paid for their order.

Additional Enhancements:

Reviews: Users can leave feedback for products, with a one-to-many relationship between Products and Reviews.

Wishlist: A new table for managing users' saved items, with many-to-many relationships between Users and Products.

Coupons: A new table to apply promotional offers to orders, linked via the orderId field.

This design ensures scalability, clarity, and efficient management of the data required for an e-commerce platform.

Figure 1: Database Entity Relationship Diagram for the E-Commerce Application

4 IMPLEMENTATION

4.1 Environment Setup

To develop the eCommerce website, we used the MERN stack (MongoDB, Express.js, React.js, Node.js). The backend was set up with Node.js and Express.js to handle API requests, while MongoDB served as the database to store user, product, and order data. The frontend was built using React.js, allowing for dynamic and responsive user interfaces. We also utilized Stripe and ABA PayWay for online payment processing and integrated a Cash on Delivery (COD) option to provide multiple payment methods.

Backend Setup:

The backend server was initialized with Node.js and Express.js to handle HTTP requests and define routes for product listings, user authentication, order management, and payments.

MongoDB was connected to store data using Mongoose, an ODM library that simplifies data interaction.

Frontend Setup:

The React app was set up using create-react-app, a command-line tool that sets up everything needed to begin React development.

Redux was integrated for state management across different components of the eCommerce platform.

Payment Integration:

Stripe API was used to process credit/debit card payments.

ABA PayWay was integrated for processing local bank transfers, specifically catering to Cambodian customers.

Cash on Delivery (COD) was also included as a payment option, giving customers the flexibility to pay when the product is delivered.

4.2 Installing Dependencies

To build and run the eCommerce platform, several dependencies were installed, including libraries for the frontend and backend:

Backend Dependencies:

express: A web framework for building the backend API.

mongoose: A MongoDB ODM to interact with the database.

cors: A middleware to enable cross-origin requests between the frontend and backend.

jsonwebtoken (JWT): Used for user authentication and maintaining secure sessions.

bcryptjs: A library for hashing passwords before storing them in the database.

stripe: For integrating Stripe payments.

dotenv: To manage environment variables.

multer: Used for file uploads, such as product images.

Frontend Dependencies:

react: A JavaScript library for building the user interface.

redux: For managing the application's state.

react-router-dom: For routing between different pages of the app.

axios: For making HTTP requests to the backend.

react-toastify: For showing user notifications like success and error messages.

4.3 Setting up Middleware and Tools

To streamline development and ensure smooth operation, several middleware tools and utilities were integrated:

CORS (Cross-Origin Resource Sharing): This middleware allows the frontend and backend, which run on different servers during development, to communicate seamlessly. It's essential for the client-side to send requests to the backend API without encountering security restrictions.

Dotenv: This library was used to store sensitive information like API keys, database URLs, and authentication tokens in an .env file, ensuring that they are not hardcoded into the source code.

Nodemon: To enhance the development experience, Nodemon was used to automatically restart the server whenever changes are made to the codebase. This helps streamline the development process and makes it more efficient.

JSON Web Token (JWT): For authentication, we used JWT to securely transmit information between the frontend and backend. Upon successful login, the server sends a JWT token that must be included in the header for subsequent requests to protected routes.

Multer: For managing product image uploads, Multer was used. It processes incoming form-data, allowing the upload of images related to products listed on the site.

4.4 API Development

The eCommerce website's backend includes multiple APIs to handle various tasks such as user authentication, product management, order processing, and payment integration. Key APIs are listed below:

User Authentication:

POST /api/user/register: Registers a new user by capturing details such as name, email, and password.

POST /api/user/login: Authenticates the user and returns a JWT token for authorized access.

Product Management:

GET /api/products: Retrieves a list of all products available in the store.

POST /api/products: Allows the admin to add new products to the store (only accessible by admins).

GET /api/products/:id: Fetches details of a specific product.

Order Management:

POST /api/orders: When a user places an order, this endpoint stores the order in the database. The payment method selected (either ABA PayWay or COD) is stored with the order.

GET /api/orders/:userId: Retrieves all orders placed by a specific user.

Payment Methods:

ABA PayWay:

POST /api/payment/abapay: This endpoint handles local bank payments through ABA PayWay.

Cash on Delivery (COD):

POST /api/orders/cod: This endpoint processes Cash on Delivery orders. The order is initially marked as "Pending Payment" and, once delivered and paid, is updated to "Paid."

5 CONCLUSION

This project focuses on the development of a fully functional eCommerce platform using the MERN stack (MongoDB, Express.js, React.js, Node.js), enhanced by integrating multiple payment methods, including **Stripe**, **ABA PayWay**, and **Cash on Delivery (COD)**. Through this approach, we demonstrated how modern web technologies can be used to create a scalable, secure, and flexible solution for online businesses.

The use of **MongoDB** allows for easy storage and retrieval of data, with its NoSQL structure making it adaptable to future changes. **Express.js** and **Node.js** form a robust backend that efficiently handles API requests, ensuring smooth interaction with the frontend and database. The **React.js** frontend ensures a dynamic and user-friendly experience for customers, providing a responsive interface that is essential for an engaging shopping experience.

The integration of **ABA PayWay** ensures that the website supports global and local payment methods, while the **Cash on Delivery (COD)** option provides customers with more flexibility, particularly in regions where online payment adoption may be limited. This gives customers the confidence to shop online without the immediate financial commitment.

By utilizing the MERN stack, we have built a cohesive system where the frontend and backend work seamlessly together. The project not only meets the current needs of eCommerce businesses but also has the flexibility to scale as the business grows. The ability to add new features, such as additional payment methods or expanded product management tools, will allow the platform to evolve and stay competitive in the digital marketplace.

In conclusion, the eCommerce website developed in this project is a modern, flexible, and robust solution that demonstrates the power of the MERN stack and the importance of offering diverse payment options to cater to the needs of a global customer base. Future improvements could include the addition of advanced search functionalities, AI-driven product recommendations, and enhanced security features to further improve user experience and operational efficiency.

REFERENCE

MongoDB Documentation

MongoDB, Inc. (2024). MongoDB Manual. Retrieved from https://www.mongodb.com/docs/

Express.js Documentation

Express, Inc. (2024). Express.js Guide. Retrieved from https://expressjs.com/

React Documentation

React, Inc. (2024). React Official Documentation. Retrieved from https://reactjs.org/

Node.js Documentation

OpenJS Foundation. (2024). Node.js Documentation. Retrieved from https://nodejs.org/

JWT (JSON Web Tokens) Documentation

Auth0. (2024). JWT Introduction. Retrieved from https://jwt.io/

Cloudinary Documentation

Cloudinary, Inc. (2024). Cloudinary API Documentation. Retrieved from https://cloudinary.com/documentation

ABA PayWay API Documentation

ABA Bank. (2024). ABA PayWay API Integration Guide. Retrieved from https://www.ababank.com/

Mongoose Documentation

Mongoose, Inc. (2024). Mongoose ODM Documentation. Retrieved from https://mongoosejs.com/docs/

CORS Documentation

MDN Web Docs. (2024). Cross-Origin Resource Sharing (CORS). Retrieved from https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

Bcrypt Documentation

bcrypt.js. (2024). Bcrypt Node.js Documentation. Retrieved from https://www.npmjs.com/package/bcrypt

Multer Documentation

Multer, Inc. (2024). Multer Documentation. Retrieved from https://www.npmjs.com/package/multer

Dotenv Documentation

Dotenv, Inc. (2024). Dotenv Documentation. Retrieved from https://www.npmjs.com/package/dotenv