

# Building a Robust Dropshipping Website Using MERN STACK

Nishit Shah  
B.Tech. CSE  
MIT World Peace University Pune,  
India  
1032221549@mitwpu.edu.in

Abhas Ajay Jaltare  
B.Tech. CSE  
MIT World Peace University Pune,  
India  
1032221128@mitwpu.edu.in

Om Durdi  
B.Tech. CSE  
MIT World Peace University  
Pune, India  
103221115@mitwpu.edu.in

Dhruv Pandey  
B.Tech. CSE  
MIT World Peace University Pune,  
India  
1032211613@mitwpu.edu.in

Dr. Reshma Sonar  
SCET  
MIT World Peace University  
Pune, India

**Abstract**— In today's digital commerce landscape, the need for secure, scalable, and user-friendly e-commerce platforms continues to rise. This project presents Shoppiyo, a full-stack dropshipping web application built using the MERN stack (MongoDB, Express.js, React.js, and Node.js). The platform provides a complete shopping experience with product browsing, cart management, secure authentication using JWT, real-time order handling, and integrated payment gateway support. A key highlight of this system is the implementation of an Automated Fraud Detection Mechanism designed to enhance platform security without relying on complex machine learning models. The system monitors login activity, tracks repeated failed attempts, and automatically temporarily locks suspicious accounts, enabling proactive fraud prevention while ensuring a seamless user experience. Additionally, an admin dashboard is provided for managing products, orders, customer messages, and monitoring blocked users in real time. The project demonstrates how the MERN stack can be effectively leveraged to build a scalable, maintainable, and secure e-commerce solution suitable for modern retail environments, particularly for dropshipping operations where rapid order processing and account security are essential.

**Keywords**- Dropshipping, E-commerce, Fraud Detection, Node.js, MongoDB, React.js, Automation, Payment Gateway, Online Retail, Supply Chain Management.

## I. INTRODUCTION

With the rapid development and competitive nature of online commerce, businesses are increasingly seeking efficient and secure ways to offer goods and services. While online platforms continue to grow in popularity for buying and trading, ensuring their usability, security, and scalability remains a primary concern. Dropshipping, in particular, demands a robust technical foundation capable of handling dynamic product availability, supplier variations, and high volumes of user interactions.

To address these challenges, we have developed the Shoppiyo platform using the MERN stack (MongoDB, Express.js, React.js, and Node.js), one of the most reliable and scalable technology stacks for full-stack web applications. The platform is designed to provide a seamless shopping experience while simultaneously addressing security vulnerabilities typically associated with e-commerce systems—especially unauthorized access attempts and suspicious login behavior.

### A. User-Centric Frontend Development:

Our platform emphasizes user-centric design to ensure smooth navigation and operational efficiency for both customers and administrators. The initial User Experience (UX) and User Interface (UI) were prototyped and refined in Figma, enabling consistent layout structure across essential pages such as Home, Collection, Product View, Cart, and Checkout.

The frontend is engineered using a modern high-performance setup:

- **React.js:** Serves as the core library for building a dynamic Single-Page Application (SPA) with reusable components and optimized rendering.
- **Vite:** Utilized as a next-generation build tool to ensure fast development refresh cycles and improved performance through Hot Module Replacement (HMR).
- **Tailwind CSS:** Applied as a utility-first CSS framework to deliver responsive, mobile-optimized, and rapidly styled UI components.

This combination ensures a clean, consistent, and highly responsive user interface that aligns with the demands of modern e-commerce platforms.

### *B. Core System Innovation (Fraud Detection):*

A major enhancement in Shoppiyo is the built-in Automated Fraud Detection Layer, designed to strengthen platform security without relying on complex machine learning models. The system continuously monitors user login attempts, tracks consecutive failed logins, and automatically temporarily locks accounts exhibiting suspicious patterns. This prevents brute-force attacks and unauthorized access attempts. Additionally, the admin panel provides real-time visibility into blocked users, enabling quick administrative oversight and ensuring the integrity of the system.

### *C. System Architecture and Design:*

The e-commerce platform is structured according to the MVC (Model–View–Controller) architecture, which is a widely adopted design pattern for scalable web systems.

- The View (client-side UI) is implemented using React.js.
- The Controller (routing and logic) is handled by Express.js and Node.js.
- The Model (data schema and business logic) is managed using MongoDB with Mongoose.

This MVC structure ensures modularity, maintainability, and the ability to scale as the platform grows in traffic and product volume.

The system also integrates key backend functionalities such as secure JWT-based authentication, encrypted password storage, payment gateway handling, order management, and administrative tools. Together, these components form a full-fledged dropshipping platform that supports smooth interactions between customers, suppliers, and administrators, alongside strengthened security measures to counter fraud and unauthorized access.

## **II . LITERATURE REVIEW**

The evolution of e-commerce systems has prompted extensive research into developing scalable, secure, and intelligent platforms capable of meeting the growing demands of online consumers. With the increasing reliance on digital marketplaces, the need for robust frameworks, efficient data handling, and advanced fraud detection mechanisms has become paramount. Recent developments in full-stack web technologies, particularly the MERN stack (MongoDB, Express.js, React.js, and Node.js), combined with Artificial Intelligence (AI) and Behavioral Biometrics, have enabled the design of more resilient and adaptive e-commerce systems.

This section provides a comprehensive review of existing literature that explores these technologies and their applications in the context of secure and scalable e-commerce development.

A. Patil et al. [1] conducted an extensive study on harnessing the MERN stack for the development of scalable and responsive e-commerce platforms. Their research emphasizes the integration of MongoDB for efficient data management, Express.js and Node.js for server-side functionality, and React.js for dynamic client-side rendering. The authors highlight the advantages of a full-stack JavaScript environment, including seamless communication between front-end and back-end components, improved maintainability, and enhanced scalability. The study demonstrates that adopting the MERN stack architecture can significantly optimize performance, reduce development complexity, and facilitate real-time data processing, making it a robust choice for modern e-commerce applications.

S. E. Cebeci et al. [2] focus on the security aspect of e-commerce systems by proposing a secure e-commerce scheme designed to enhance data protection and transaction confidentiality. Their work presents a framework that utilizes advanced encryption techniques and authentication protocols to safeguard user data during online transactions. The authors further analyze vulnerabilities associated with traditional e-commerce platforms and compare the performance of their proposed model against existing security mechanisms. The study concludes that implementing a multi-layered security architecture improves system resilience against cyber threats such as identity theft, data breaches, and unauthorized access, thereby strengthening consumer trust in online marketplaces.

A. Afeez [3] investigates the application of Artificial Intelligence and Behavioral Biometrics for user authentication in e-commerce platforms. The study explores how AI models can leverage behavioral traits such as keystroke dynamics, mouse movement patterns, and browsing behavior to verify user identity. This approach offers a dynamic and adaptive alternative to static password-based systems, enabling continuous authentication based on real-time user behavior. The author's findings reveal that AI-integrated biometric systems not only enhance authentication accuracy but also provide an effective means of fraud prevention. This research underscores the growing relevance of intelligent biometric technologies in addressing evolving cybersecurity challenges within e-commerce ecosystems.

C. Miao et al. [4] present a novel study on AI-powered fraud detection in Business Intelligence (BI) systems, emphasizing the role of machine learning and behavioral biometrics in identifying fraudulent activities. Their research introduces a model that continuously monitors user interactions and applies

supervised learning algorithms to detect anomalies indicative of fraudulent intent. By employing behavioral profiling and anomaly detection, the system is capable of identifying high-risk transactions in real time. The authors demonstrate that integrating AI-based fraud detection significantly improves accuracy, minimizes false positives, and enhances the reliability of financial systems. This work provides valuable insights into how AI can be leveraged to secure e-commerce transactions and maintain data integrity.

N. Sharma et al. [5] discuss the design and development of an e-commerce website utilizing the MERN stack, with a focus on achieving high performance and scalability. Their research outlines the architectural framework and implementation process of a full-stack web application where MongoDB manages the database, Express.js and Node.js serve as the backend framework, and React.js powers the user interface. The study highlights the efficiency of JavaScript-based full-stack development, demonstrating how uniformity in programming language streamlines communication between different application layers. The authors conclude that MERN-based applications offer significant advantages in terms of speed, scalability, and ease of maintenance, establishing the framework as a preferred choice for next-generation e-commerce solutions.

In summary, the reviewed studies collectively emphasize the convergence of modern web technologies and intelligent security mechanisms in shaping the future of e-commerce development. The integration of the MERN stack provides a scalable and efficient foundation for building responsive web applications, while the incorporation of AI and Behavioral Biometrics enhances system security and user authentication. Together, these advancements contribute to the realization of secure, data-driven, and high-performance e-commerce systems capable of meeting the demands of today's digital economy.

### III. SYSTEM ARCHITECTURE

#### A. *Shoppiyo : System Architecture*

The proposed model outlines the architectural design and functional workflow of the Shoppiyo platform, ensuring scalability, secure data handling, and smooth interaction between customers, administrators, and backend services. The system follows a distributed, three-tier architecture based on the MERN stack, with built-in security features for fraud prevention through login attempt monitoring and account-locking mechanisms.

#### • Frontend (React + Vite)

This layer provides a dynamic Single-Page Application (SPA) developed using React.js and optimized with Vite for faster builds and improved performance, it manages :

- User interface rendering
- Page routing
- State management
- Form handling (login, cart, checkout)
- Real-time feedback (toast alerts, lock warnings)

The React frontend communicates with the backend exclusively through REST API calls, ensuring a clean separation of concerns.

#### • Backend / API Layer (Node.js + Express.js)

Built using Node.js and Express.js, this layer forms the core logic of the system. It is responsible for :

- User authentication using JWT tokens
- Secure password hashing using bcrypt
- Login attempt tracking and automated account locking
- Managing product listings, orders, and contact submissions
- Admin authentication and privileged operations
- Integrating payment processing via Razorpay/Stripe
- Exposing all system functionality through RESTful endpoints

This layer also maintains the fraud detection logic by monitoring failed login attempts and updating user records accordingly.

#### • Database Layer (MongoDB + Mongoose)

MongoDB serves as the backend database for scalable and flexible document storage. The system uses dedicated collections for :

- Users
- Products
- Orders
- Contact messages
- Login attempt logs (for fraud monitoring)

Mongoose is used to define strong schema models and ensure structured data handling.

### • Built-in Fraud Detection Module

Instead of an external ML-based microservice, Shoppiyo incorporates a native fraud detection system embedded within the backend. This component :

- Tracks failed login attempts for each account
- Automatically locks accounts after repeated suspicious activity
- Records login attempts for admin review
- Displays blocked users in the admin panel

This approach provides practical, real-time security while maintaining low system complexity and high reliability.

### • External Integrations

The platform also includes essential third-party service integrations:

- Payment Gateways (Razorpay) for secure handling of online transactions
- Cloudinary for efficient image hosting and product media storage
- Optional shipping or supplier APIs for dropshipping workflows (future scope)

These services ensure the system maintains reliability, performance, and high availability across all operations.

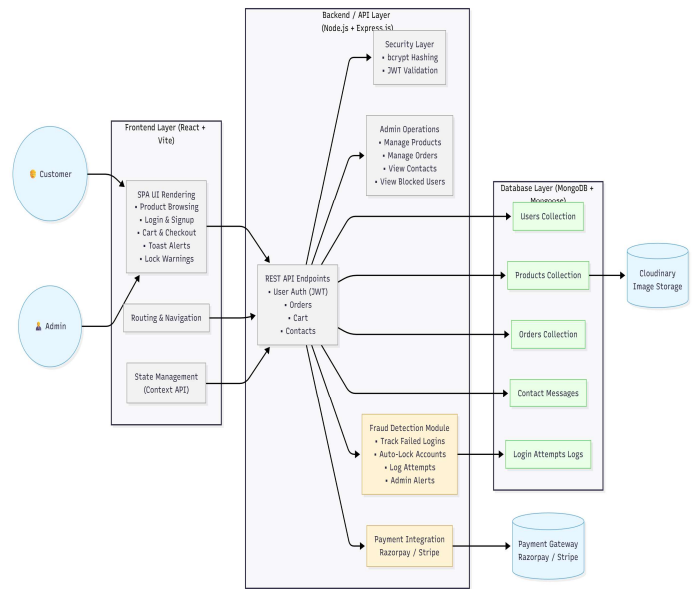


Fig.1. System Architecture

### B. Use Case Diagram (System Functionality)

The Use Case Diagram illustrates the functional scope and the interactions between the Shoppiyo platform and its primary actors. It helps define the complete behavioral requirements of the system while outlining how different users and system components interact within the e-commerce workflow.

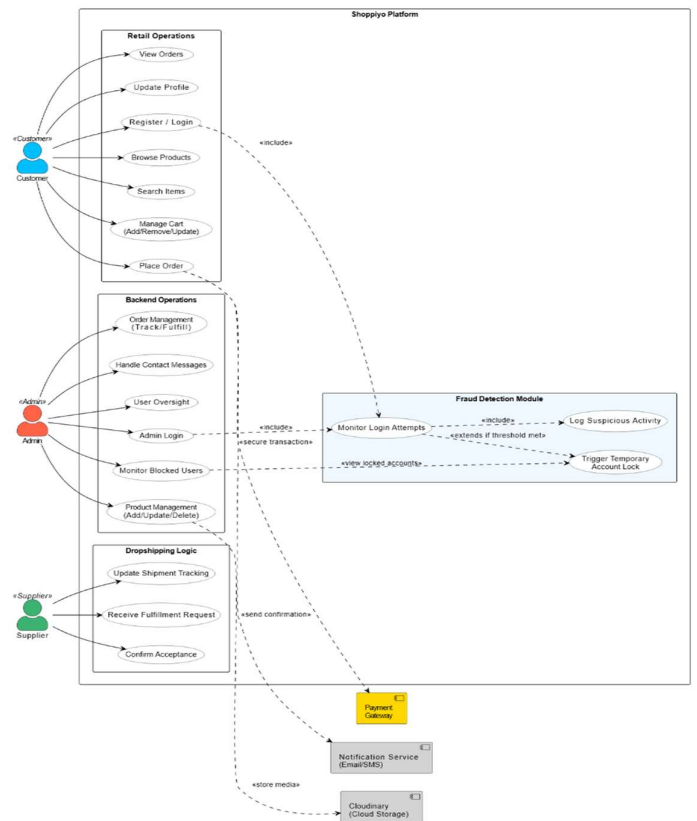


Fig.2. Use Case Diagram

## 1. Customer

The Customer represents the core end-user of the platform and performs the essential retail operations. Their primary use cases include:

- Register / Login: Create an account or authenticate using secure credentials.
- Browse Products: View product collections, categories, and detailed product pages.
- Search Items: Filter and search items based on user needs.
- Manage Cart: Add items, update quantities, or remove products.
- Place Order: Proceed with checkout and finalize a purchase using integrated payment methods.
- View Orders: Review previous order history and track purchase status.
- Update Profile: Manage personal account details.

These actions represent the typical customer journey expected in a modern e-commerce platform.

## 2. Admin

The Admin oversees backend operations, platform maintenance, and retail management. Their key use cases include:

- Admin Login: Secure authentication to access administrative controls.
- Product Management: Add new products, update existing listings, or delete outdated items.
- Order Management: View, track, and update customer orders.
- Contact Message Handling: View and respond to user-submitted contact queries.
- Blocked User Monitoring: View accounts that are temporarily locked due to repeated failed login attempts.
- User Oversight: Access general user information as needed for support and troubleshooting. The Admin actor ensures that the business functions smoothly and securely.

## 3. Supplier (Dropshipping Workflow)

Although suppliers do not interact directly with the platform's interface, their role is logically represented in the system design.

Their use cases include:

- Receive Order Fulfillment Request: The platform forwards order details after successful payment.
- Confirm Acceptance: Supplier confirms inventory availability.
- Update Shipment Tracking: Provide updates for delivery progress (future enhancement scope).

This actor reflects the external logistics layer inherent to dropshipping.

## 4. System Integrations & Infrastructure

Shoppiyo communicates with several external modules to ensure reliable functional flow. These include:

- Payment Gateway: Handles secure transaction processing during checkout.
- Cloud Storage (Cloudinary): Manages product images and media assets.
- Notification Services (Email/SMS) (future scope): For sending order confirmations or updates.

These integrations are essential for delivering a complete e-commerce experience.

## 5. Fraud Detection Module

The platform incorporates a native Fraud Detection system that enhances account security by monitoring suspicious login activity.

Its use cases include:

- Monitor Login Attempts: Track consecutive failed login attempts for each user.
- Trigger Temporary Account Lock: Automatically lock accounts after a threshold is reached.

- **Log Suspicious Activity:** Store login attempt data for admin review.
- **Show Blocked Users in Admin Panel:** Allow the administrator to identify and manage locked accounts.

This module strengthens authentication security without the need for complex machine learning models.

### C. Activity Diagram (Transactional Flow)

The Activity Diagram illustrates the sequential flow of operations during the two most critical processes of the Shoppiyo platform: User Authentication Security and Order Placement. This diagram helps visualize how user actions and backend logic interact continuously to ensure both a smooth shopping experience and protection against unauthorized access.

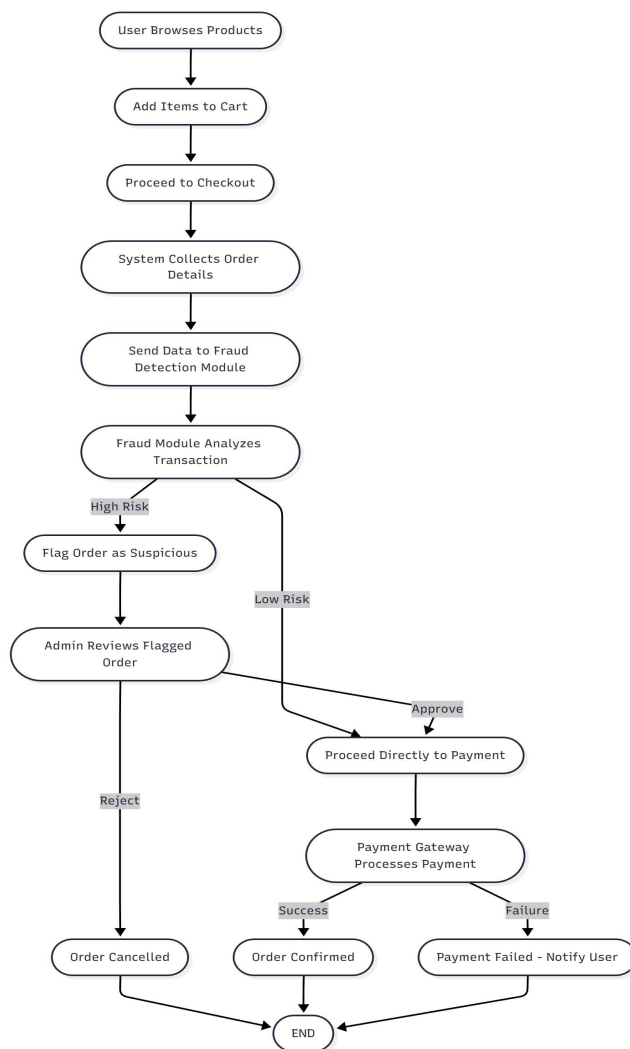


Fig.3. Activity Diagram

## 1. Login & Fraud Monitoring Flow

This part of the activity flow focuses on the platform's built-in fraud detection mechanism, which prevents unauthorized access by monitoring repeated failed login attempts.

### 1. User Initiates Login:

The flow begins when the user enters their email and password.

### 2. Credential Verification:

The backend compares the submitted password with the securely hashed password stored in the database.

### 3. Decision Node — Password Match:

- **If the Password is Incorrect:**
  - The system increments the user's failed login counter.
  - A warning is shown to the user, including attempts left.
  - If the number of failed attempts reaches the threshold (three attempts):
    - The system temporarily locks the account for a fixed duration (5 minutes).
    - The user is shown a lockout message and cannot log in until the lock expires.
    - The admin can view this locked user in the admin panel.
- **If the Password is Correct:**
  - Failed login counter is reset.
  - A JWT token is generated.
  - The user is granted access to their account.

This flow ensures that login activity is continuously monitored and that suspicious behavior is mitigated through automated account locking.

## 2. Add to Cart & Checkout Flow

Once the user is authenticated, they can proceed with the shopping and order placement process.

### 1. Product Interaction :

The user selects a product and triggers Add to Cart.

### 2. Proceed to Checkout:

The user reviews their cart and confirms that they want to place the order.

### 3. Order Validation:

The system verifies product availability and validates the user session and order details.

### 4. Payment Initialization:

The backend establishes a secure connection with the payment gateway (Stripe or Razorpay) to process the transaction.

## 3. Decision Node — Payment Status

A decision point determines the result of the payment operation.

### • If Payment Is Successful:

- The system records the order in MongoDB.
- The user's cart is cleared.
- An order confirmation is displayed.
- The admin panel updates the order list.

### • If Payment Fails:

- The user receives an error message.
- The cart remains unchanged.
- The user can retry the payment.

## IV. METHODOLOGY

In the development of an e-commerce website, we used the MERN stack because of its efficient development workflow, scalability, and performance. Figure shows MERN Stack architecture, which comprises the open-source elements, namely MongoDB, Express.JS, React.JS/Redux, and Node.JS. In this the frontend is assigned by React.JS + Vite, HTML, and CSS. For the backend runtime environment, Node.JS and web framework Express.JS, which gives and provides various libraries. MongoDB is used as a data-based system in which data is stored in documents, and a group of documents is known as collections. We used Windows operating system, along with Git, GitHub, and VS Code tools. The MERN stack significantly boosts the e-commerce platform's usability and overall functionality. Its ability to scale ensures the platform can handle growth without compromising performance. Node.js powers fast server-side processing, resulting in quick response times, while React.js creates a smooth, interactive user interface with rapid load speeds. MongoDB's flexible data management efficiently supports large product inventories, and Express.js strengthens backend security. Crucially, this core MERN architecture is enhanced by the integration of a specialized AI Fraud Detection Microservice. This dedicated security module operates as a checkpoint within the transactional flow, analyzing transaction payloads and behavioral data in real-time. This proactive measure elevates the platform's security beyond standard payment gateway checks, directly mitigating the unique fraud risks inherent in the dropshipping model. Together, these technologies create a responsive, secure, and high-performing platform, offering an outstanding shopping experience that evolves with user demands.

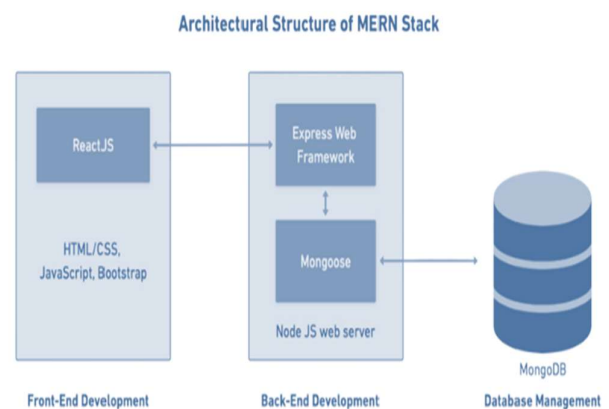


Fig.4. MERN stack architecture

## MONGODB

For our project, we employed MongoDB, a document-oriented database. In this database, data is stored in documents, and a group of documents is known as a collection. MongoDB, the most well-known NoSQL database, offers a substitute for relational databases, which store all of their data in tables with rows and columns. JSON data is actually transformed by MongoDB into a binary version on the server, which is essentially stored and queried more effectively. BSON is the query language used by MongoDB. We cannot conceive of MongoDB as a JSON database, however, as it saves BSON format both internally and over the network. Any data can be represented in JSON format and saved natively in MongoDB. It can also be simply accessed in JSON format. Thus, from our study and implementation of MongoDB, we can conclude that it is flexible and enables users to design schema, databases, tables, and other objects. We had the choice to use Mongo Shell after installing MongoDB because it provides a JavaScript interface for users to communicate and do any query-related activities. Since MongoDB is a document-oriented database, indexing documents is simple. and for that reason, it processes responses more quickly. Scalability Characterizes MongoDB. The enormous data is divided in the MongoDB database into a nested described structure. Multiple databases can be operated on MongoDB, a database server.

## NODEJS

C++ is used in the creation of Node.js, a JavaScript operating system. For optimal performance, it makes use of the Google Chrome V8 engine. Node.js is built with a single-thread architecture and employs event-driven, asynchronous programming callback functions. Our ability to create websites using node.js for back-end development has been aided by the abundance of event-based and asynchronous APIs that are made possible by the essential core notion of node.js design: event-drivenness. In line with our web business logic, Node.js employed the appropriate callback function. Using a single-threaded architecture is the primary benefit of event-driven and asynchronous programming. Without waiting for a specific code to finish, the callback function code is executed, and the limited resources were used for other tasks that were part of our web business logic. This design was appropriate for our back-end development, which was also our system's intended purpose. Managing synchronous requests was a significant task in server development, because blocking resulted in either inefficient or wasteful use of the resources. The resource usage is increased and enhanced website speed with single-threaded architecture and asynchronous callback routines, which also produced the expected testing outcomes. It is evident from the Node.js supported module that many

functions, including file operations, are performed asynchronously, in contrast to other programming languages. Node.js uses particularly big network components, such as HTTP, DNS, NET, UDP, HTTPS, TLS, etc., to make server building easier. These network modules allow developers to set up a Web server.

## EXPRESSJS

Express is a Node.js framework that we used. While constructing the website, we learned that Express made it simpler and easier to write the back-end code and implement it in an organized fashion, as opposed to writing the code with NodeJS and producing a ton of node modules. Express's support for numerous middlewares, which results in shorter and easier-to-write code, assisted us in the creation of the web and APIs needed for the project. Our design benefits greatly from Express's single-threaded architecture and asynchronous programming. Robust API is essential for our design. In order to begin our Express project, we made a new folder. The first step involved adding a command to the command prompt in order to populate the package.json file. Following that, we had to proceed with accepting the default settings. The command to begin is npm init.

## REACTJS

Programming in JavaScript uses the front-end package React.JS. We built our web design user interface using React.JS because of its fast rendering of dynamically changing data. With React, developers can write JS code and produce User Interface elements. The development environment was optimized for speed and efficiency. The entire frontend was initialized and bundled using Vite, which ensures exceptionally fast development server startup and Hot Module Replacement (HMR). Furthermore, the application's clean, reusable component architecture was styled using Tailwind CSS, a utility-first framework that allows for rapid, responsive design directly within the code. We looked at the virtual DOM objects in React.JS, the framework we used for our project. The entire user interface would re-render the virtual DOM whenever we made modifications to our e-commerce design. This lets us assess how the DOM object and virtual DOM might differ from one another. We employed JSX (JavaScript XML), which made writing our code in the React application more straightforward and easier. The components with React.JS are used while each component has a logic connected to our e-commerce design and adds to the overall user interface of the project. Components are the building blocks of the user interface. It was easier to understand our site design code and improve overall efficiency when components could be reused.

## MVC architectural pattern

The design of the application adheres to the Model-View-Controller (MVC) architectural pattern, which breaks down complexity into three logical components:

- **View (Frontend):** The web design user interface is built using React.js. The entire frontend was initialized and bundled using Vite, which ensures exceptionally fast development server startup and Hot Module Replacement (HMR). Furthermore, the application's clean, reusable component architecture was styled using Tailwind CSS, a utility-first framework that allows for rapid, responsive design directly within the code. React's use of JSX and the Virtual DOM ensures fast rendering of dynamically changing data and superior efficiency through component reusability.
- **Model:** The persistence layer is handled by Mongoose (the server-side Object Data Modeling layer) and the MongoDB database.
- **Controller:** The functional logic is handled by the Node.js and Express.js environment.

This comprehensive approach ensures that together, these technologies create a responsive, secure, and high-performing platform, offering an outstanding shopping experience that evolves with user demands.

## Fraud Detection Methodology

To strengthen the platform's security and protect against unauthorized access, the Shoppiyo system integrates an automated fraud-mitigation layer. This module does not rely on complex Machine Learning models but instead implements lightweight behavioral security techniques, login attempt monitoring, and account-locking mechanisms that significantly reduce the risk of misuse. The methodology ensures rapid processing, high accuracy, and seamless integration within the MERN architecture.

### 1. Data Monitoring and Feature Extraction

Instead of traditional static checks (e.g., password-only verification), the system evaluates dynamic behavioral and session-based patterns. During the authentication process, the backend quietly monitors:

#### a. Behavioral Indicators

- Number of consecutive failed login attempts.

- Frequency and pattern of incorrect submissions.

#### b. Session & Device Features

- IP address consistency across login attempts.
- Device or browser changes (via User-Agent fingerprint).
- Timing patterns such as extremely rapid repeated login attempts.

#### c. Account Activity Patterns

- Sudden spike in login attempts.
- Attempts occurring at unusual times or from new devices.

These lightweight indicators are sufficient to detect suspicious access behavior without requiring a full ML model.

## 2. Rule-Based Risk Evaluation & Thresholding

The fraud detection engine uses a **threshold-based classification method** to determine whether a login attempt should be allowed, delayed, or blocked.

### Core Logic Implemented:

- **Failed Attempt Counter**

Every incorrect login increases the failed count.

- **Threshold Check**

Upon reaching 3 failed attempts, the system automatically triggers the security protocol.

- **Temporary Account Locking**

At the threshold, the system assigns a lockUntil timestamp which prevents the user from logging in for a fixed period (5 minutes).

This prevents:

- brute-force attempts,
- bot-generated login storms,
- credential-stuffing attacks.

This method ensures high precision in identifying suspicious activity while avoiding false positives that negatively affect legitimate users.

### 3. Real-Time Backend Decision Flow

The fraud detection logic is embedded within the authentication pipeline of the Node.js/Express backend.

#### Process Flow:

1. **User submits login request.**
2. The system retrieves the user's failed attempt count and lock status.
3. **If account is locked:**
  - A lockout message is displayed.
  - User cannot attempt login until the lock expires.
4. **If password is wrong:**
  - Failed counter increases.
  - Lock is applied if threshold is reached.
  - User receives appropriate warnings.
5. **If password is correct:**
  - All counters and lock flags reset.
  - JWT token is issued.

This approach ensures that every authentication attempt undergoes immediate risk evaluation before authorization.

### 4. Admin-Side Monitoring and Response

To maintain transparency and oversight, the system exposes login-related risk signals to administrators.

#### Admin Panel Displays:

- Users who are currently locked.
- Timestamp of lock expiration.
- Failed login attempt count.

This empowers the admin to:

- reset or unblock users if necessary,
- monitor suspicious activity patterns,
- identify potential targeted attacks.

## V. PROPOSED MODEL

The proposed model outlines the architecture and operational framework for Shoppiyo, a MERN-based dropshipping e-commerce platform equipped with an integrated fraud detection and account-protection mechanism. The system is designed to deliver a seamless online shopping experience while maintaining robust security controls, scalability, and efficient data management. The model focuses on three major components: the platform's MERN architecture, real-time authentication security, and administrative operational automation.

### A. System Architecture (MERN-Based Platform)

The core of the Shoppiyo system is built upon the MERN architecture . MongoDB, Express.js, React.js, and Node.js , which enables modularity, scalability, and consistent API-driven communication.

1. **Frontend Layer (React + Vite + Tailwind CSS):**

The interface is implemented as a reactive Single-Page Application (SPA), capable of dynamic rendering without full page reloads.

  - React components manage user interactions such as browsing products, managing cart items, and initiating checkout.
  - Vite optimizes development and bundling speed, ensuring fast loading and enhanced runtime performance.
  - Tailwind CSS provides a mobile-responsive and customizable design system.
2. **Backend API Layer (Node.js + Express.js):**

The backend implements all functional logic, including:

  - Secure JWT-based user authentication
  - API routing for products, orders, user accounts, payments, and messages
  - Fraud detection logic for login attempts
  - Integration with third-party payment services such as Razorpay or Stripe

The backend also ensures secure data validation and error handling for all client-initiated operations.

3. **Database Layer (MongoDB + Mongoose):**  
MongoDB is used as a flexible NoSQL document store for:

- User profiles
- Product catalog
- Orders
- Contact messages
- Login-attempt logs

Mongoose schemas enforce structure, constraints, and relationships between data entities.

4. **External Integrations:**

The model incorporates essential external services:

- **Cloudinary** for product media storage
- **Payment Gateway** for secure transaction handling
- **IP-based login monitoring** to assist fraud detection

B. Real-Time Authentication Security and Fraud Prevention System

Unlike typical e-commerce platforms that rely solely on payment gateway fraud controls, Shoppiyo integrates a real-time login-attempt monitoring system to detect and halt unauthorized access early in the user journey.

The fraud control mechanism works as follows:

1. **Failed Login Attempt Tracking:**

Each incorrect password increases a failed login count for the account. This counter is stored in the user's MongoDB record.

2. **Risk Threshold Decision:**

When the number of failed attempts reaches a threshold (three consecutive failures), the system automatically triggers the security workflow.

3. **Automatic Temporary Account Locking:**  
A lockUntil timestamp is generated, preventing the user from logging in for a fixed duration (e.g., 5

minutes).

This mechanism mitigates:

- Brute-force attacks
- Credential-stuffing attempts
- Bot-driven login storms

4. **Admin Fraud Dashboard:**

Locked users appear automatically in the admin panel under "Blocked Users," enabling administrators to monitor suspicious activity and manually intervene when required.

This security subsystem protects both the user and the platform by preventing unauthorized access before transactional activity occurs.

C. Order Processing and Checkout System

Once authenticated, users can browse products, manage their cart, and proceed to checkout. The checkout subsystem follows a streamlined workflow:

1. **Cart Validation:**

Ensures product availability and pricing accuracy.

2. **Payment Initialization:**

A request is sent to the integrated payment gateway (RazorPay) Users complete payment using secure gateway pages.

3. **Order Confirmation:**

Upon successful payment:

- The order is stored in MongoDB
- The user's cart is cleared
- The admin receives real-time updates in the dashboard

4. **Supplier Fulfillment (Dropshipping Model):**  
The system prepares structured order details for supplier integration or future automation.

This model ensures high throughput and low latency during ordering workflows.

## VI . RESULT ANALYSIS

Graphical results play a crucial role in enhancing the reader’s understanding of the system’s functionality and usability. This section presents the visual and functional outputs of the Shoppiyo platform by showcasing key user interface screens, admin management modules, and integrated system responses. Each figure demonstrates how the MERN-based architecture effectively delivers a smooth, secure, and responsive e-commerce experience.

- **Customer Interface Results**

### 1. Login Page

Before accessing personalized features, users authenticate through the secure login interface. The Login Page validates credentials, handles incorrect attempts, and ensures safe access using JWT-based authentication. The interface is designed for clarity, responsiveness, and ease of use.

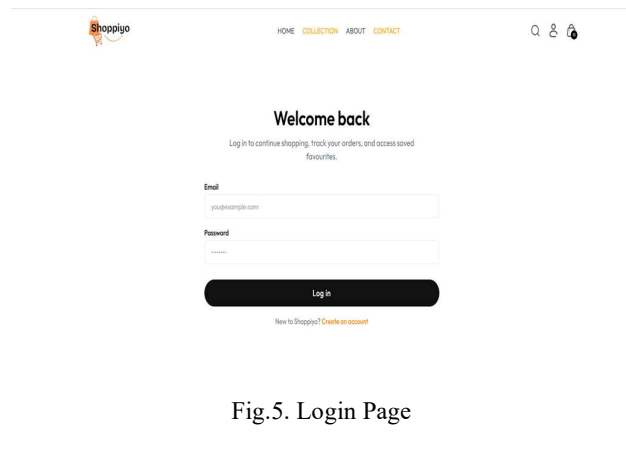


Fig.5. Login Page

### 2. Home Page

The home page serves as the central entry point for the user. It displays featured items, promotional banners, and smooth product navigation. The visually appealing layout is designed using React.js and Tailwind, ensuring a responsive and dynamic user experience.

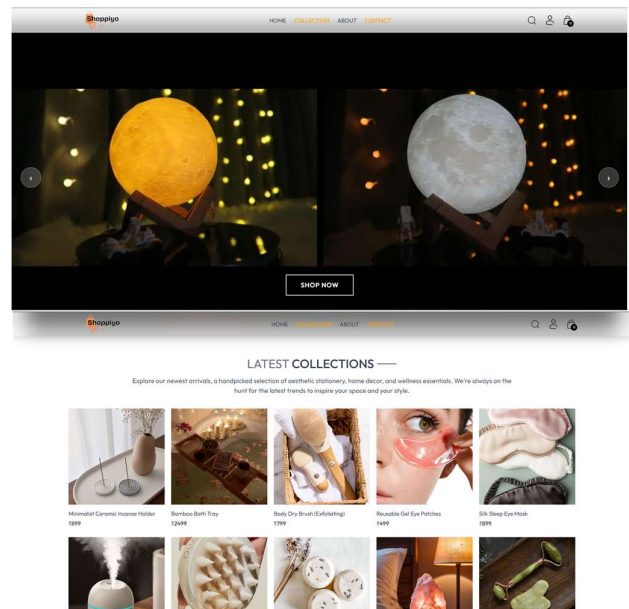


Fig.6. Shoppiyo Home Page

### 3. Collections Page

The Collections page showcases the organized product catalog, enabling users to browse categories, explore new arrivals, and filter items. This interface allows customers to seamlessly navigate through the dropshipping inventory.

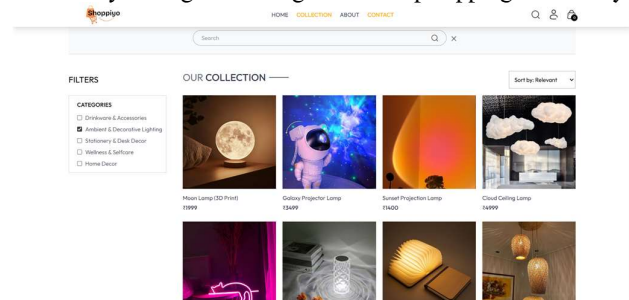


Fig.7. Collections Page

### 4. Contact Page

The contact page allows users to send inquiries or feedback directly to the system administrator. Messages submitted here are captured through the backend API and stored in MongoDB for admin review .

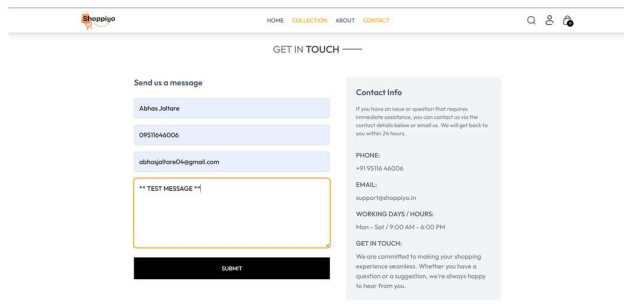


Fig.8. Contact page

## 5. User Profile & My Orders

The user profile page enables users to update their personal information, manage password settings, and view their complete order history. The “My Orders” section retrieves all previous purchases, displaying order status, payment confirmation, and timestamps.

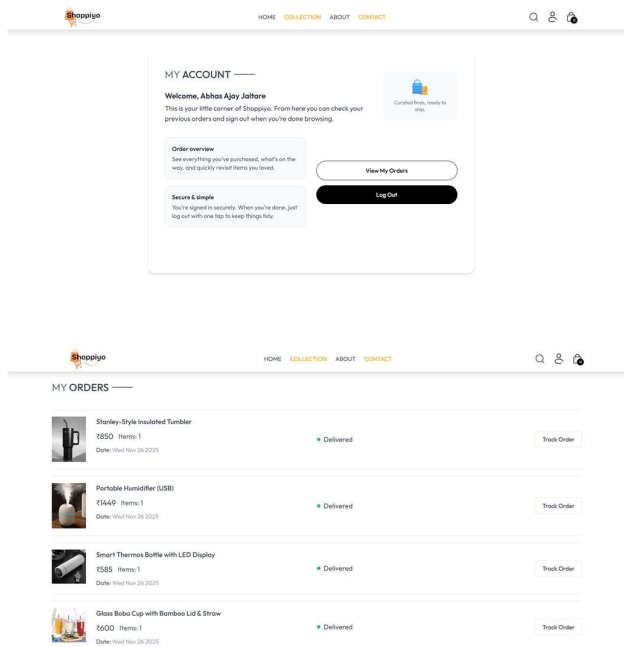


Fig.9. User profile/ Orders page

## 6. Checkout Page

The checkout interface consolidates cart items, shipping information, and payment options. Integrated with Razorpay, it ensures secure and real-time payment processing, providing a smooth flow for order completion.

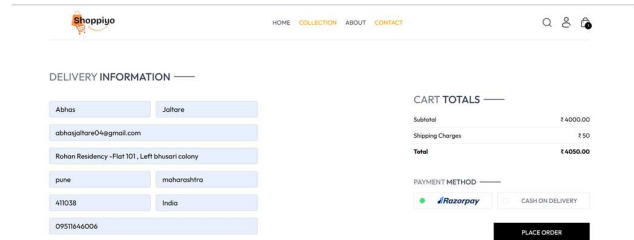
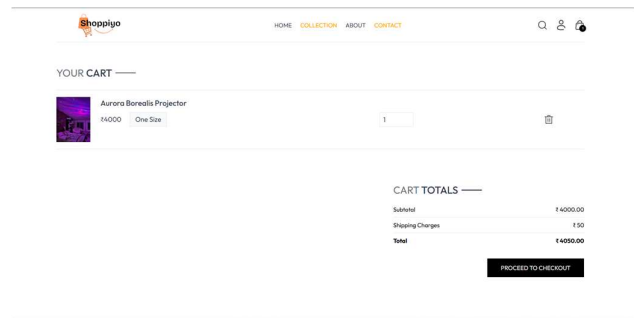


Fig.10. Checkout page

## 7. Order Successful Page

Upon successful payment, the system generates an Order Success page confirming the transaction. The page displays the order ID and provides navigation back to the home page or order tracking.

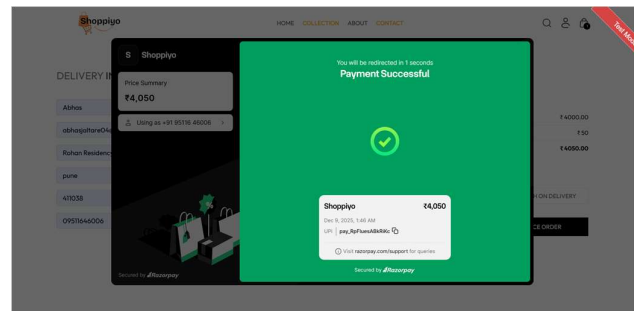


Fig.11. Order placed Page

## • Admin Interface Results

### 1. Admin Login Page

The Admin Login interface ensures controlled access to administrative tools using secure authentication and backend validation.

## 2. Admin Dashboard Overview

The dashboard provides real-time insights into orders, users, messages, and system activities to assist in efficient platform management.

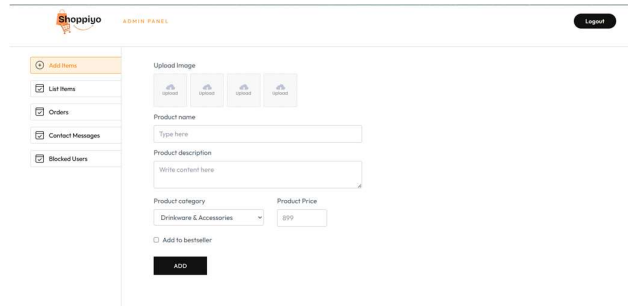


Fig.12. Admin dashboard

## 3. Product Management Page

Admins can add, update, or delete products. The page integrates Cloudinary for seamless image upload and storage.

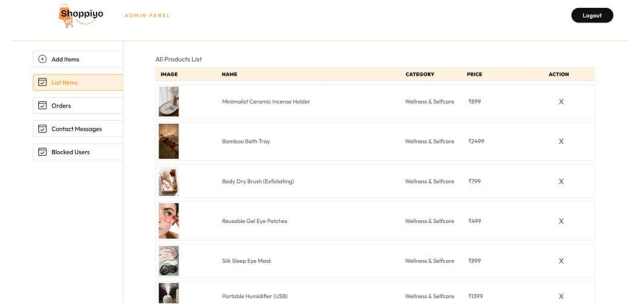


Fig.13. Product management page

## 4. Order Management Page

This view allows administrators to track all orders, update their statuses, and monitor buyer transactions.

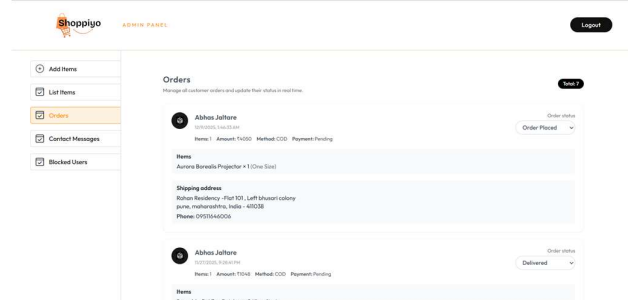


Fig.14. Order Management page

## 5. Contact Messages Page

All customer-submitted contact messages appear here. Admins can review, manage, and respond to users efficiently.

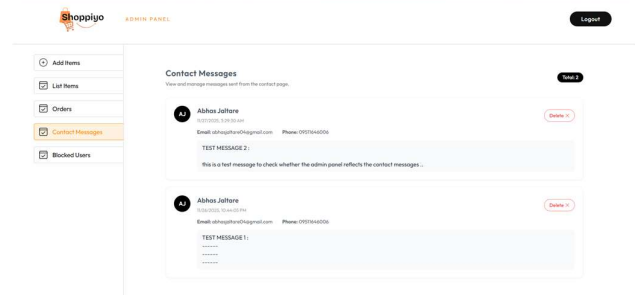


Fig.15. Contact messages page

## 6. Blocked Users Page

This section displays accounts temporarily locked due to repeated failed login attempts. It demonstrates the platform's built-in security mechanism for preventing brute-force login behavior.



Fig.16. Blocked users page

## VII . CONCLUSION

The Shoppiyo platform demonstrates how the MERN stack can be used to build a secure, scalable, and user-friendly dropshipping system suitable for real-world deployment. By combining React.js, Node.js, Express.js, and MongoDB, the platform ensures smooth navigation, efficient order handling, and secure payment processing through Razorpay. The inclusion of automated login-monitoring and account-locking mechanisms further strengthens authentication security without requiring complex machine-learning models. Beyond academic implementation, this project provides a practical foundation that can be extended into a functional, real-world dropshipping business—allowing for streamlined product sourcing, automated order flow, and personalized customer experiences. With future enhancements such as ML-based fraud detection, supplier integration, and advanced analytics, Shoppiyo can evolve into a fully operational e-commerce venture, supporting both technological innovation and personal entrepreneurial growth.

## VI. ACKNOWLEDGEMENT

For her helpful assistance and support during the production of this paper, we would like to sincerely thank our guidance teacher, Dr. Reshma Sonar. Their knowledge, support, and helpful criticism have been crucial in forming the structure and substance of this document.

Additionally, we would want to thank everyone who has helped us learn and acknowledge the support that MIT World Peace University has given us

## VII. REFERENCES

- [1] A. Patil et al., "Harnessing the MERN stack for scalable E-commerce website design: A full-stack approach with MongoDB, Node.js, Express.js, and React.js," *J. Integr. Sci. Technol.*, vol. 13, no. 5, p. 1116, Apr. 2025.
- [2] S. E. Cebeci, K. Nari, and E. Ozdemir, "Secure E-Commerce Scheme," *IEEE Access*, vol. 10, pp. 10359–10370, 2022.
- [3] A. Afeez, "Behavioral Biometrics and AI for User Authentication in E-Commerce," *Res. Gate*, 2025.
- [4] C. Miao et al., "AI-Powered Fraud Detection in BI Systems Using Machine Learning: A Behavioral Biometric Approach," *Int. Res. J. Adv. Eng. Hub (IRJAEH)*, 2025.
- [5] N. Sharma et al., "E-Commerce Website Using MERN Stack," *Int. J. Res. Appl. Sci. Eng. Technol. (IJRASET)*, vol. 10, no. V, pp. 2419–2420, May 2022, doi: 10.22214/ijraset.2022.42808.
- [6] S. E. Cebeci, K. Nari, and E. Ozdemir, "Secure E-Commerce Scheme," *IEEE Access*, vol. 10, pp. 10359–10370, 2022.
- [7] A. Das, "High-Traffic Node.js: Strategies for Success," *Medium*, Mar. 3, 2025.
- [8] M. Maguire, "A Review of Usability Guidelines for E-commerce Website Design," in *Design, User Experience, and Usability*, vol. 14032, Springer, Cham, 2023.
- [9] J. S. Palese and A. Usai, "The relative importance of service quality dimensions in E-commerce experiences," *Int. J. Inf. Manage.*, vol. 40, pp. 132–140, 2018.
- [10] T. T. Shama and H. Hoque, *Full-Stack React Projects: Learn MERN stack development by building modern web apps using MongoDB, Express, React, and Node.JS*. Packt Publishing Ltd, 2020.
- [11] V. May et al., "The MERN Stack's Payment Security Analysis," *Int. J. Eng. Res. Technol. (IJERT)*, vol. 11, no. 06, pp. 1–5, 2023.
- [12] S. K. Shukla, "Application using MERN Stack," *Int. J. Mod. Trends Sci. Technol.*, vol. 8, pp. 102–105, 2022.
- Understood. Here are the three additional dropshipping-focused references, correctly formatted, without the "Relevance" explanation.
- [13] I. Zennaro, "Implementing E-commerce from logistic perspective: Literature review and methodological framework," *Sustainability*, vol. 14, no. 2, p. 911, 2022.
- [14] S. H. Liu, R. T. Chen, and M. W. Cheng, "Inventory Synchronization and Dynamic Pricing Optimization in Dropshipping E-commerce," in *Proc. Int. Conf. Supply Chain Manag. (ICSCM)*, Shanghai, China, 2024, pp. 315–320.
- [15] M. T. Soleimani, "Buyers' trust and mistrust in e-commerce platforms: a synthesizing literature review," *Inf. Syst. E-Business Manage.*, vol. 20, no. 1, pp. 57–78, 2022.
-