IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.154170

TradeSphere: A Full Stack Web-based Stock Trading Platform Using MERN Stack

Aaditya Gupta[1], Akshay Dhiman[2], Aman Kumar[3], Harsh Sharma[4], Dr.Uruj Jaleel[5], Dr.Satish Kumar Soni [6]

Student, MCA, Meerut Institute of Engineering and Technology, U.P. India[1]

Student, MCA, Meerut Institute of Engineering and Technology, U.P. India[2]

Student, MCA, Meerut Institute of Engineering and Technology, U.P. India[3]

Student, MCA, Meerut Institute of Engineering and Technology, U.P. India[4]

Professor, MCA, Meerut Institute of Engineering and Technology, U.P. India[5]

Associate Professor, MCA. Meerut Institute of Engineering and Technology. U.P. India[6]

Abstract: The rapid expansion of financial markets and the increasing participation of students, young professionals, and first-time investors in stock trading have created a strong demand for educational tools that simplify the understanding of market operations. In recent years, digital trading platforms have become highly popular due to the availability of smartphones, internet access, and low-cost investment options. However, most existing trading applications are designed for professional traders and real investors, making them complex and difficult for beginners to understand. These platforms generally include advanced trading charts, multiple financial indicators, margin calculations, technical analysis tools, and live market data, which may overwhelm new learners. As a result, there is a growing requirement for a simplified educational trading system that can help users understand the core concepts of stock market trading without involving real financial risk.

This paper presents TradeSphere, a simulation-based full-stack web application developed to provide an easy, structured, and accessible environment for learning the basic functionalities of a stock trading platform. The system has been designed mainly for students, academic demonstrations, and beginner-level users who want to understand how modern online trading systems work. Instead of interacting with actual stock exchanges or real money transactions, TradeSphere simulates the major components of a stock trading dashboard, including stock holdings, market positions, order placement, and portfolio management.

Keywords: TradeSphere, MERN Stack, Stock Trading Platform, React.js, Node.js, Express.js, MongoDB, REST API, Portfolio Management, Trading Dashboard.

INTRODUCTION

The stock market plays a central role in the modern global economy, enabling companies to raise capital and providing investors with opportunities to grow their wealth. Despite its importance, a large proportion of students and fresh graduates remain unfamiliar with even the basic operational mechanics of equity trading — including what a stock holding means, how open positions differ from settled ones, or how a buy or sell order is processed through a brokerage system [1].

This knowledge gap is largely attributed to the complexity and commercial orientation of existing trading platforms, which are designed for experienced investors rather than learners. Educational institutions have begun to recognize the importance of financial literacy within technical curricula, particularly for students of computer science and management, but most lack dedicated practical tools to teach trading concepts in a hands-on manner [2].

Advances in full-stack web development — particularly the emergence of JavaScript-based stacks such as MERN — have made it possible for student developers to build functional, production-ready web applications that closely simulate real-world systems [3]. The MERN stack, comprising MongoDB, Express.js, React.js, and Node.js, provides a consistent JavaScript environment across both the frontend and backend, reducing the language-switching overhead that traditionally complicated full-stack development.

TradeSphere addresses the challenge of trading education by providing a simulation-based stock trading platform built entirely on the MERN stack. The system allows users to view simulated stock holdings and positions, and to place new orders through a modern, responsive web interface. The application is deployed on cloud infrastructure, requires no

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026
DOI: 10.17148/IJARCCE.2026.154170

external financial API subscriptions, and stores all data in a cloud-hosted MongoDB database. This makes it ideally suited for academic use, project demonstrations, and self-directed learning.

The purpose of TradeSphere is three-fold: first, to demonstrate how a modern full-stack web application is architected and implemented using the MERN stack; second, to simulate the core data flows that underpin a real stock trading platform; and third, to serve as a foundation for progressive feature development in subsequent academic or professional projects.

The remainder of this paper is organized as follows: Section II defines the objectives of the project. Section III describes the scope. Section IV provides a system overview. Section V explains the architecture. Section VI details the technology stack. Section VII describes the step-by-step working of the system. Section VIII covers the modules. Section IX presents the API design. Section X covers database design. Section XI discusses results. Section XII lists advantages. Section XIII concludes the report.

Objectives of the Project

- The following objectives guided the design, development, and evaluation of TradeSphere:
- .To design and implement a full-stack web application using the MERN stack that simulates a core stock trading environment for educational purposes.
- To provide a clean, responsive, and intuitive user interface using React.js and Bootstrap that allows users to navigate between different sections of the trading dashboard with ease.
- To implement a RESTful backend API using Node.js and Express.js that handles client requests for fetching stock holdings, fetching open positions, and recording new simulated orders.
- To design a structured MongoDB database schema using Mongoose that stores and manages stock holdings, positions, and order records in an organized and scalable manner.
- To demonstrate the seamless integration of frontend and backend components through HTTP-based communication, illustrating the full request-response lifecycle of a MERN stack application.
- To simulate the core data presentation features of a stock portfolio dashboard, including stock names, quantities, average buy prices, and current simulated values.
- To establish a modular and well-organized project structure that is easy to understand, extend, and maintain by developers with intermediate-level skills.
- To deploy the application on cloud infrastructure so it is accessible from any device with a web browser, without requiring local installation.

Scope of the Project

What the System Covers

TradeSphere covers the following functional and technical areas within its defined scope:

- Design and implementation of a fully functional MERN stack web application with a React.js frontend and a Node.js and Express.js backend, deployed on cloud hosting services.
- Display of simulated stock holdings, showing stock names, quantity owned, average purchase price, and net value per holding.
- Display of open market positions, representing stocks that are currently active in a trade and have not yet been squared off.
- Order placement functionality through which users can submit new simulated buy or sell orders that are recorded in the MongoDB database.
- Three RESTful API endpoints: GET /allHoldings, GET /allPositions, and POST /newOrder, each serving a specific data function.
- A responsive frontend with a Navbar, Footer, Home page with multiple informational sections, a Signup page (UI), and dedicated pages for Holdings, Positions, and Order placement.
- Cloud deployment of the frontend and backend with data stored in a cloud-hosted MongoDB Atlas database, making the application accessible over the internet.

Proposed System

System Overview

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.154170

TradeSphere is a web-based, simulation-driven stock trading platform that replicates the user experience and data flow of a basic trading dashboard. The system provides users with a multipage interface through which they can view their stock holdings, check their open market positions, and place new simulated orders.

The system is divided into two primary components: the frontend client application, built with React.js, and the backend server application, built with Node.js and Express.js. These two components are deployed as independent services and communicate over HTTP using REST APIs. All data is stored in a MongoDB Atlas cloud database.

Since TradeSphere is a simulation platform, the data displayed — including stock names, prices, and quantities — is pre-populated in the database and does not reflect live market conditions. This design keeps the system stable and appropriate for academic and learning use cases.

System Architecture

- The architecture of TradeSphere consists of five principal modules that operate in a sequential pipeline:
- User Interface Module: A React.js-based web interface that manages user interaction, displays pages, accepts inputs, and renders API responses. Styled with Bootstrap for responsiveness.
- Routing Module: React Router handles client-side navigation between the Home, Holdings, Positions, Orders, and Signup pages without causing full browser reloads.
- API Communication Module: Axios or the Fetch API is used within React components to send HTTP requests to the Express.js backend and receive JSON responses asynchronously.
- Backend API Module: Express.js route handlers receive requests, process them, interact with the database using Mongoose models, and return structured JSON responses.
- Database Module: MongoDB Atlas stores all application data in three collections — Holdings, Positions, and Orders — with Mongoose schemas enforcing structure and validation.

System Flow

The operational flow of TradeSphere proceeds as follows: The user accesses the deployed web application through a browser. The React.js frontend loads and renders the Home page. The user navigates to Holdings, Positions, or Orders using the Navbar. Each page, upon mounting, sends the appropriate API request to the backend. The backend processes the request, queries MongoDB, and returns a JSON response. The frontend renders the received data in the appropriate component. For order placement, the user fills and submits a form, which triggers a POST request to record the order in the database, and a confirmation message is displayed.

METHODOLOGY

Technology Stack

TradeSphere is implemented using the following technology stack, with each component selected for specific and well-justified reasons:

1. React.js: React.js is an open-source JavaScript library maintained by Meta for building dynamic, component-based user interfaces. React's virtual DOM ensures efficient rendering, and its component model allows the TradeSphere dashboard to be built as independent, reusable pieces — Navbar, Footer, Holdings, Positions, and Orders are each self-contained components. React Router enables seamless client-side navigation between pages, creating a Single Page Application (SPA) experience [4].
2. Node.js: Node.js is a JavaScript runtime built on Google Chrome's V8 engine that enables server-side JavaScript execution. Its non-blocking, event-driven I/O model makes it highly efficient for handling concurrent API requests. In TradeSphere, Node.js serves as the runtime for the entire backend server [5].
3. Express.js: Express.js is a minimal and flexible Node.js web framework that simplifies the creation of REST APIs and web servers. It is used in TradeSphere to define the three API routes, configure CORS middleware to allow cross-origin requests from the React frontend, and manage JSON body parsing for incoming POST requests [6].
4. MongoDB: MongoDB is a document-oriented NoSQL database that stores data in BSON format. Its schema-flexible document model is well-suited for stock trading data where structures may evolve. TradeSphere uses a MongoDB Atlas cloud-hosted instance, eliminating the need for local database installation and enabling access from any deployed environment [7].
5. Mongoose: Mongoose is an Object Data Modeling library for MongoDB and Node.js. It adds schema enforcement, data validation, and a clean model-based API to MongoDB's flexible documents. TradeSphere uses Mongoose to define and interact with the Holdings, Positions, and Orders collections [8].

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.154170

6. Bootstrap: Bootstrap is an open-source CSS framework developed by Twitter that provides pre-built responsive components, grid systems, buttons, and navigation elements. TradeSphere uses Bootstrap to ensure the interface is visually consistent and responsive across different devices and screen sizes [9].
7. REST API Architecture: Representational State Transfer (REST) is an architectural style for designing networked APIs. TradeSphere's backend exposes three RESTful endpoints — GET /allHoldings, GET /allPositions, and POST /newOrder — following HTTP conventions. The decoupled nature of REST allows the frontend and backend to be developed, deployed, and scaled independently [10].

Development Phases

The development of TradeSphere was structured into eight phases: Requirement Analysis, System Design and Architecture Planning, Database Schema Design, Backend API Development, Frontend Component Development, Frontend-Backend Integration, Testing and Debugging, and Cloud Deployment and Final Review. Each phase was completed sequentially with validation checkpoints to ensure quality at every stage.

Implementation

Project Folder Structure

TradeSphere is organized as two separate Node.js projects within a single repository: one for the frontend and one for the backend. This mirrors real-world full-stack project organization:

```
tradesphere/
├── frontend/              ← React.js Frontend Application
│   ├── public/
│   │   └── index.html
│   ├── src/
│   │   ├── components/
│   │   │   ├── Navbar.js
│   │   │   └── Footer.js
│   │   ├── pages/
│   │   │   ├── Home.js
│   │   │   ├── Holdings.js
│   │   │   ├── Positions.js
│   │   │   ├── Orders.js
│   │   │   └── Signup.js
│   │   ├── App.js
│   │   └── index.js
│   └── package.json
│
└── backend/               ← Node.js + Express.js Backend
    ├── model/
    │   ├── HoldingsModel.js
    │   ├── PositionsModel.js
    │   └── OrdersModel.js
    ├── index.js
    └── package.json
```

Frontend and Backend Connection

The frontend and backend are connected through HTTP API calls. In each React component, the Fetch API or Axios library is used to send asynchronous requests to the deployed backend server. For example, in the Holdings component, a useEffect hook executes when the component mounts and calls the GET /allHoldings endpoint. The JSON response is stored in the component's state using useState, which triggers a re-render that displays the holdings table.

To prevent CORS errors when the frontend and backend are hosted on different domains, the CORS npm middleware is configured in the Express server to include the appropriate Access-Control-Allow-Origin headers in all responses, permitting the deployed frontend to access backend resources.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.154170

Deployment

TradeSphere is designed for cloud deployment. The React.js frontend is deployed on Vercel or Netlify, both of which provide automatic CI/CD integration with GitHub repositories and serve the compiled React build as a static website with global CDN distribution. The Node.js and Express.js backend is deployed on Render or Railway, which provide free-tier Node.js hosting with automatic restarts and environment variable management. The MongoDB database is hosted on MongoDB Atlas, the official cloud database service from MongoDB, Inc., which provides a managed, globally distributed database cluster accessible from the deployed backend through a secure connection string.

Modules Description

Navbar Module

The Navbar module is a persistent navigation component rendered at the top of every page. It is built as a React component and uses React Router's NavLink elements to provide client-side navigation between the Home, Holdings, Positions, and Orders pages. The TradeSphere brand name is displayed on the left while navigation links are aligned to the right. The Navbar is fully responsive, collapsing into a hamburger menu on smaller screens using Bootstrap's responsive classes.

Home Page Module

The Home page serves as the landing page and is structured into multiple informational sections. The Hero section introduces TradeSphere with a headline and a brief description. A Features section highlights the core capabilities using card-style UI components. A Call-to-Action section invites the user to explore the dashboard. The Home page is entirely static and does not require API calls. It communicates the purpose of the platform clearly to first-time visitors.

Holdings Module

The Holdings module displays the user's current stock portfolio — the list of stocks purchased and currently held. This page renders a data table populated with information retrieved from the backend via the GET /allHoldings API. Each row represents one stock holding and displays the stock's ticker symbol, quantity owned, average purchase price, current simulated price, and net value. The component uses useEffect to fetch data on mount and useState to store and render the holdings array.

Positions Module

The Positions module displays open market positions — trades that have been initiated but not yet completed or squared off. This page fetches data from GET /allPositions and presents it in a tabular format. Fields include stock name, net quantity, day's buy or sell value, and the current profit or loss. This module helps users understand the operational difference between a settled holding and an active open position — a concept central to trading literacy.

Orders Module

The Orders module provides a form-based interface through which users can place new simulated stock orders. The form collects the stock name, quantity, price, and order type (buy or sell), and submits the data to the backend via POST /newOrder. On successful submission, a confirmation message is displayed. The form is managed using React's controlled component pattern, where each input field is bound to a corresponding state variable.

Signup Page Module

The Signup page provides a basic user registration interface with fields for name, email, and password. In the current version, this page is a static UI placeholder and does not connect to any backend authentication logic. It serves as the foundation for the authentication module planned in future development.

Backend API Module

The API module is the core of the backend application. It defines the three REST API routes using Express.js, with each handler interacting with MongoDB through Mongoose models. The module configures the JSON body parser middleware to parse incoming POST request bodies, and the CORS middleware to allow requests from the deployed frontend. The API module is cleanly separated from the database schema definitions, following the principle of separation of concerns.

API Design

TradeSphere's backend exposes three RESTful API endpoints. Each endpoint maps to a specific resource, uses the appropriate HTTP method, and returns data in JSON format.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.154170

Endpoint	Method	Function	Response
/allHoldings	GET	Fetch all stock holdings	JSON array of Holdings documents
/allPositions	GET	Fetch all open positions	JSON array of Positions documents
/newOrder	POST	Record a new simulated order	Saved Order document confirmation

GET /allHoldings

This endpoint retrieves all documents from the Holdings collection. When the frontend Holdings page mounts, it sends a GET request to the /allHoldings route on the deployed backend. The Express route handler calls HoldingsModel.find({}), instructing Mongoose to retrieve all documents from the Holdings collection without any filters. The complete array of holding documents is returned as a JSON response with HTTP status 200. No request body is required for this endpoint.

GET /allPositions

This endpoint functions identically to GET /allHoldings but targets the Positions collection. When the Positions page mounts, a GET request is sent to /allPositions. The handler uses PositionsModel.find({}) to fetch all documents and returns them as a JSON array. The frontend parses this array and renders each position as a row in the positions table, allowing users to see all currently open trade positions stored in the simulation database.

POST /newOrder

This endpoint accepts a new order submission from the frontend Orders form. When the user submits the form, the frontend collects the order data — stock name, quantity, price, and order type — and sends it as a JSON object in the POST request body. The Express handler receives the data through req.body (enabled by the express.json() middleware), creates a new OrdersModel instance, and calls .save() to persist the document to MongoDB Atlas. A success response confirms the order has been recorded, and the frontend displays a confirmation message.

IX. Database Design

TradeSphere uses MongoDB Atlas as its cloud database, with Mongoose providing schema-based data modeling. The database consists of three collections, each corresponding to a distinct category of financial data within the simulation.

Holdings Collection

The Holdings collection stores documents representing stocks currently owned by the simulated user. Each document contains: name (String) — the ticker symbol of the stock; qty (Number) — the number of shares held; avg (Number) — the average price per share at purchase; price (Number) — the current simulated market price; net (Number) — the net profit or loss for the holding; and day (Number) — the price change for the current trading session.

Positions Collection

The Positions collection stores documents representing active, open trades. Each document contains: product (String) — the product type such as CNC or MIS; name (String) — the stock ticker; qty (Number) — the net quantity in the position; avg (Number) — the average price; price (Number) — the current price; net (Number) — the net value change; day (Number) — the day's price movement; and isLoss (Boolean) — a flag indicating whether the position is currently at a loss.

Orders Collection

The Orders collection records every order placed through the Orders module. Each document contains: name (String) — the stock ticker for the order; qty (Number) — the number of shares in the order; price (Number) — the price at which the order was placed; and mode (String) — indicating whether the order is a 'buy' or 'sell' order. A new document is added to this collection each time the POST /newOrder endpoint is successfully called.

RESULTS AND DISCUSSION

System Output

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026
DOI: 10.17148/IJARCCE.2026.154170

To illustrate the system's behavior, consider a typical user interaction. When a user navigates to the Holdings page, the frontend sends a GET request to /allHoldings. The backend queries the MongoDB Holdings collection and returns a JSON array. For example, a response may include a holding in INFY stock with a quantity of 10 shares, an average purchase price of Rs. 1,450, and a current simulated price of Rs. 1,510, resulting in a net gain of approximately 4.1%.

The Positions page similarly fetches and displays open trade records. On the Orders page, when a user places a buy order for 5 shares of TCS at Rs. 3,800, the POST /newOrder API records the order in MongoDB and returns a success confirmation, which the frontend displays as a green confirmation message.

System Performance

The system was tested across multiple scenarios including simultaneous display of holdings and positions, repeated order placement, and navigation between all pages. In all test cases, the application responded correctly with properly structured JSON data and accurate UI rendering. API response times for the GET endpoints were consistently under 300 milliseconds in the deployed environment, demonstrating that the Express.js and MongoDB Atlas combination provides adequate performance for an educational simulation platform with a moderate dataset size. The application was also tested on multiple devices — desktop browsers, tablets, and mobile phones — confirming that the Bootstrap-based responsive layout adapts correctly to different screen sizes without UI breakage.

DISCUSSION

The results demonstrate that TradeSphere successfully models the core data interactions of a stock trading dashboard in a clean, functional, and educationally meaningful manner. The three-tier MERN architecture proved well suited to the requirements of the project, with clear separation between the user interface, business logic, and data layers. The project also confirmed that the MERN stack enables rapid full-stack development with a single programming language across all layers, significantly reducing development complexity and making the codebase accessible for students at the MCA level. The REST API design ensures that the system is extensible — additional endpoints and frontend pages can be added without restructuring the existing architecture.

Advantages of the System

1. Educational Value: TradeSphere provides a practical, hands-on learning environment for students to understand stock market concepts such as holdings, positions, and order placement, entirely free of financial risk.
2. Full-Stack Integration: The project demonstrates end-to-end integration of a modern JavaScript frontend with a Node.js backend and a cloud MongoDB database, giving students a complete picture of how MERN stack applications operate.
3. Cloud Deployment: Unlike locally-run academic projects, TradeSphere is deployed on cloud infrastructure, making it accessible from any device over the internet and reflecting real-world deployment practices.
4. Clean REST API Architecture: Well-defined REST endpoints separate the frontend and backend into independent, interchangeable layers. New features can be added to either layer without disrupting the other.
5. Responsive and Modern UI: The use of React.js and Bootstrap ensures the interface is responsive, component-based, and visually consistent across all device sizes.
6. Scalable Database Design: MongoDB Atlas's document-based model allows the Holdings, Positions, and Orders collections to scale with growing data volumes without requiring schema migrations.
7. Reusable Component Architecture: The React component model reduces code duplication and makes the frontend easy to maintain, extend, and test independently.
8. Strong Foundation for Future Development: The clean architecture of TradeSphere is designed to support future additions including user authentication, real-time data feeds, and advanced analytics without requiring a fundamental redesign.

CONCLUSION

This paper presented TradeSphere, a web-based stock trading simulation platform developed using the MERN stack as a final year academic project. The system successfully demonstrates how the key components of a stock trading dashboard — portfolio holdings, open market positions, and order placement — can be modeled and implemented as a fully functional, cloud-deployed full-stack web application. TradeSphere fulfills all of its defined objectives. A responsive, multi-page React.js frontend communicates seamlessly with a Node.js and Express.js backend through clearly defined RESTful API endpoints. Data is stored and retrieved efficiently from a MongoDB Atlas cloud database using Mongoose schemas, and the entire application is deployed on cloud infrastructure, accessible to users from any device with a web browser.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026
DOI: 10.17148/IJARCCE.2026.154170

From a technical perspective, the project provides a comprehensive practical exercise in frontend component design using React.js, REST API development using Express.js, cloud database management using MongoDB Atlas, and the full-stack integration of multiple application layers using the MERN stack. Each technology is used purposefully, and the architecture follows established best practices for separation of concerns, modularity, and scalability. From an educational perspective, TradeSphere achieves its core mission by providing students and beginners with an accessible, safe, and functional environment in which to understand how trading platforms work at a systems level — without financial risk, account setup complexity, or dependency on live market data.

The development of TradeSphere produced significant learning outcomes for the project team. The process of designing a three-tier architecture, implementing RESTful APIs, managing asynchronous data flow between React components and a deployed backend, and modeling financial data in MongoDB collectively provided deep, hands-on experience in full-stack web development that complements the theoretical knowledge acquired through academic study. Future development will focus on user authentication, real-time data integration, interactive charting, and mobile application development — enhancements that will progressively evolve TradeSphere into a fully featured financial education platform.

REFERENCES

[1]. Meta Open Source. "React — A JavaScript Library for Building User Interfaces." Meta Platforms, Inc., 2024. Available: https://react.dev. Accessed: April 2026.

[2]. OpenJS Foundation. "Node.js Official Documentation." OpenJS Foundation, 2024. Available: https://nodejs.org/en/docs. Accessed: April 2026.

[3]. T.J. Holowaychuk et al. "Express.js — Fast, Unopinionated, Minimalist Web Framework for Node.js." Express.js Official Documentation, 2024. Available: https://expressjs.com. Accessed: April 2026.

[4]. MongoDB. "MongoDB Atlas — Cloud Database Service." MongoDB, Inc., 2024. Available: https://www.mongodb.com/atlas. Accessed: April 2026.

[5]. Mongoose. "Mongoose ODM v8 — Elegant MongoDB Object Modeling for Node.js." Mongoose Official Documentation, 2024. Available: https://mongoosejs.com. Accessed: April 2026.

[6]. JSON Web Token Contributors. "JSON Web Token Introduction and Documentation." 2024. Available: https://jwt.io. Accessed: April 2026.

[7]. GitHub. "GitHub Documentation." 2024. Available: https://docs.github.com. Accessed: April 2026.

[8]. Cloudinary. "Cloudinary Documentation for Image Upload and Storage." 2024. Available: https://cloudinary.com/documentation. Accessed: April 2026.

[9]. REST API Tutorial. "RESTful Web Services and API Design." 2024. Available: https://restfulapi.net. Accessed: April 2026.