FoodFly: Design and Implementation of a Full-Stack MERN-Based Online Food Ordering and Delivery Platform

Sunil Dhull, PhD
Department of Mechanical Engineering, Panipat Institute of Engineering and Technology, Panipat, Haryana, India

Stuti Mehla, PhD
Department of Computer Science & Engineering, Panipat Institute of Engineering & Technology, Panipat, Haryana, India

Anju Gandhi, PhD
Department of Computer Science & Engineering, Panipat Institute of Engineering & Technology Panipat, Haryana, India

Rubi Kumari
Department of Computer Science & Engineering, Panipat Institute of Engineering & Technology Panipat, Haryana, India

Simpi Kumari
Department of Computer Science & Engineering Panipat Institute of Engineering & Technology Panipat, Haryana, India,

Love Kumar
Department of Computer Science & Engineering Panipat Institute of Engineering & Technology, Panipat, Haryana, India

Geeta Rani
Department of Computer Science & Engineering, Panipat Institute of Engineering & Technology Panipat, Haryana, India

ABSTRACT
The rapid growth of e-commerce and on-demand services has transformed the food and beverage industry. Consumer expectations regarding fast, convenient, and digitally mediated ordering experiences have changed. This paper presents FoodFly, a full-stack web-based online food ordering and delivery application built on the MERN stack (MongoDB, Express.js, React, and Node.js). The system enables customers to browse a categorized food menu, view item details and ratings, manage the shopping cart, apply discount coupons, complete checkout, and maintain a personal wishlist along with a profile, while a RESTful backend independently handles user authentication and food-catalogue management. The architecture separates concerns between a React and Redux Toolkit-powered single-page client and a Node.js/Express API layer backed by a MongoDB document store, with security implemented through bcrypt password hashing and JSON Web Token (JWT)-based authentication. This paper documents the motivation, related work, problem statement, objectives, proposed methodology, system architecture, entity-relationship and database schema design, module-wise implementation, algorithms, and a discussion of the platform's functional and non-functional characteristics, evaluated against comparable MERN-based food-ordering systems reported in the literature. The evaluation highlights the strengths of the MERN stack for rapid, component-driven development and transparently identifies concrete integration gaps, including the pending wiring of client-side calls to the authentication API and the absence of a payment gateway and real-time order-tracking logic—as the immediate next steps of development.

Keywords
Food Ordering, MERN Stack, React, Node.js, MongoDB, E-commerce, RESTful API, JWT Authentication, Software Architecture

1. INTRODUCTION
Online food ordering systems are among the most common types of consumer web and mobile apps. This is a recent trend because people's lives are changing. Urbanization is increasing and the demand for convenient services is growing. Business platforms usually have certain basic business requirements. Customers want a menu that is categorized and searchable/sortable. Customers also want the same shopping cart every time and accounts that are safe and secure. Online food ordering needs to have a way to give discounts and promotions. They need a checkout process which is simple and easy to use. Building such a system end-to-end offers a practical vehicle for studying full-stack web engineering including client-side state management, RESTful API design, and security.

In this research on FoodFly, a complete full-stack web application for online food ordering that is developed using the MERN stack, which consists of MongoDB for data storage, Express.js and Node.js for the server-side API, and React (Redux Toolkit and Context API) for the client-side interface. The research studies the applicability of the modern JavaScript oriented technology stack for the implementation of the core functional modules of a food-ordering system: user registration and login, food catalog browsing, cart and wishlist management, coupon-based discounting, checkout and order confirmation.

The rest of the paper is organized as follows. Section 2 describes related work and literature survey. Section 3 frames the problem, objectives, methodology, system architecture implementation and related algorithms. The results and their implications are shown in section 4. In sections 5 discusses the future scope and section 6 contains conclusion.

2. LITERATURE REVIEW

Several recent studies have looked at the use of the stack for online food ordering. [1] Wang et al. developed the web based online food ordering system and also described the main components for menu display, order processing and web technologies selection. Patil [2] developed an ordering system with PHP and MySQL and gave a basic comparison to other methods such as MERN which are document and JavaScript based. Hossain [3] created an e-commerce website for restaurants to handle money and manage the available orders, which is essential in many food ordering systems regardless of the technology used. Acharya [4] outlined a complete online food-order system from a requirements-engineering perspective, covering the workflows for menu management and order placement. Darshan et al. [5] presented Quickbites, a MERN-stack food-ordering system that combined menu browsing, cart management, and administrative dashboards in a single-vendor system with a similar scope to the system presented here.

Two architectural trends keep recurring across these studies. One, data stores like MongoDB have been increasingly favoured over relational schemas for catalogue-type data food items with different attributes like ratings, categories, availability, and so forth—due to the flexible, schema-less document model and its natural JSON compatibility with JavaScript-based backend [6], [7]. Second, the use of hashed passwords and JSON Web Tokens for stateless authentication has become the de facto pattern in systems based on Node.js, avoiding the need for server-side session storage while still allowing token verification on each request [8], [9]. Table 1 summarizes the reviewed studies on food-ordering and their methodology and contribution.

Table 1. Comparative study of related online food-ordering systems

Ref.	Focus Area	Stack/Method	Key Contribution
[1] Wang et al.,2021	Web-based ordering	Generic web tech.	Architecture & workflow for ordering
[2] Jadhav & Patil, 2020	Food ordering system	PHP + MySQL	Relational-DB baseline
[3] Hossain & Hossain, 2022	Restaurant e-commerce	Web application	Transaction & catalogue handling
[4] Acharya, 2024	Food order system	Requirement driven	End-to-end workflow specification
[5] Darshan et al., 2025	Quickbites platform	MERN stack	Cart, menu, admin dashboard

In terms of this research, FoodFly is similar to the projects that use the MERN system, like the one in reference [5]. This paper contributes to the discussion by clearly defining the completed and the ongoing part of the project. For example, the Express and MongoDB authentication and the food catalog API are fully integrated. On the other side, the menu browsing, cart and checkout features are still in prototype and not fully integrated yet. Such detail is often missing from reports.

A lot of mid-sized food vendors take orders the old-fashioned way (by hand or phone) or use third party services. These methods do not give vendors control over the look of their brand, the customer information they can collect and how they present their menus. They have to pay a fee to use these services. These vendors need a system that they can use themselves to take orders. It should be built with technology that is cheap to develop, host and maintain, but that also works well and is easy to add new features. The FoodFly has to be an easy system to use and economical to run. The FoodFly system should be something that vendors can use to take orders in a way that is easy for them to manage. The paper deals with the concrete problem of how to design and implement the core modules of such a platform – secure account management, browsable and searchable food catalog, cart and wishlist management, coupon-based discounting and checkout – using a full-stack JavaScript technology stack (MERN) in a maintainable, modular architecture suitable for incremental development by a small team?

- Create a three tier client server setup for a website where people can order food online using React, Express.js, Node.js and MongoDB.
- Make sure users can safely sign up and log in using a method where passwords are turned into a code and a special type of token is used to keep them logged in without saving any information. Model Food, User and Cart entities as MongoDB document schemas appropriate for catalogue-style, variable structure data.
- Implement a RESTful API for authentication and food-catalogue CRUD operations.
- Develop a component-based client interface that is responsive for menu browsing, cart/wishlist management, coupon application, and checkout.
- Publicly publish the status of all modules and what is needed to wire them into the client-server.

3. PROPOSED METHODOLOGY

The system has been built in an incremental way, the backend API, the database and the frontend have been built separately. Each piece of the system was then built and tested individually and then integrated. People do front and back end JavaScript projects. JavaScript is used by developers on the client side and on the server. Then they can work at the same time. The methodology consists of the following steps.

Step 1: Look at the requirements Normal Food-Ordering Platform: Major use cases identified (Browse menu, Register and login, Manage cart, Apply coupon, Checkout, Manage wishlist and Manage profile).

Step 2: Technology Selection: The technologies that have been selected are MongoDB, Node.js, React and Express.js. It had a JavaScript runtime, a flexible document modeling, and a mature ecosystem (Redux Toolkit, Tailwind CSS, JWT, and bcrypt).

Step 3: Database Schema Design: It is used to define the User, Food, and Cart schemas in mongoose. It also has application layer validation rules.

Step 4. Backend API Development: Built Express routes/controllers for user registration/login (bcrypt hashing, JWT issuance) and full CRUD operations on the food catalogue.

Step 5. Front End Section: It built the page views, and the React components. React Router took care of those components and how they worked.

Step 6. Implementation of cart for the client: It will implement wishlist and user profile for the client side state management. It has done this job with React Context providers and the Redux Toolkit store.

Step 7. Interaction Design and Styling: It used Tailwind CSS utility classes and Framer Motion transitions for interaction design and styling to make sure it would look and work great on devices.

Step 8. Manual functional testing of each module: This step made sure that each module was performing the required functionalities (including error handling for the authentication endpoints).

Step 9. Done Gap Analysis Integration: Performed integration gap analysis and tested the client-server wiring of all modules to determine overall functionality, still a mockup on client side.

Step 10. Documentation: It recorded the schema, the API surface and the algorithms. This is also to help in research report and easy to maintain the system.

The various steps used in this research are shown in figure 1.

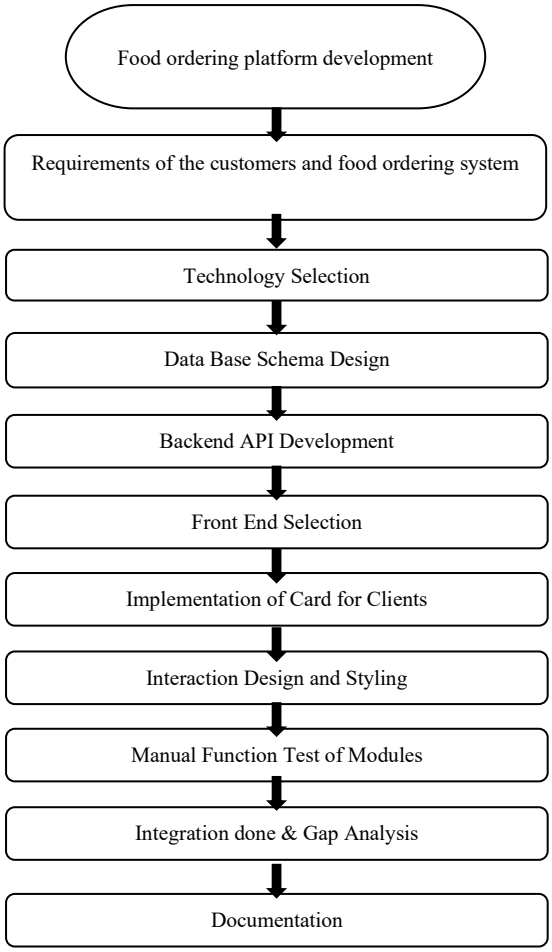

Figure 1 Steps in online food development platform

3.1 SYSTEM ARCHITECTURE

The FoodFly system is implemented using a three-tier client–server architecture that includes a presentation tier (the React single-page application), an application tier (the Express.js REST API running on Node.js), and a data tier (MongoDB accessed through Mongoose) and follows the REST architectural style [10]. Fig. 2 shows the architecture including client-side and server-side module groupings.

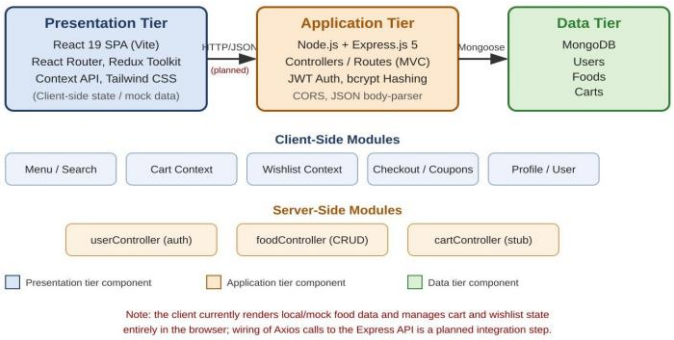

Fig 2. Illustrates the architecture, including client-side and server-side module groupings

Figure 3 shows the FoodFly system. It has three parts. The FoodFly system is made up of the presentation part the application part and the data part. These are the modules of FoodFly. The presentation part, the application part, and the data part of FoodFly work to make FoodFly work properly is shown below.

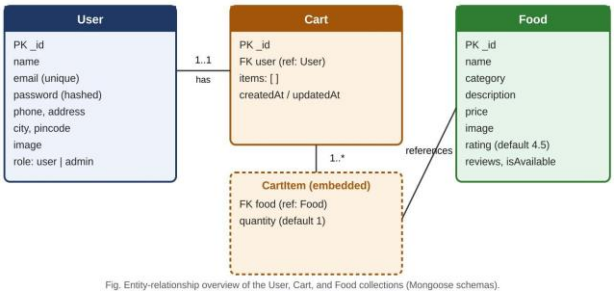

Fig. 3. Entity–relationship diagram of the User, Cart, and Food collections

A. Client (Presentation) Tier

Client is a React 19 single page application bundled with Vite. React Router is used to route between the views - Home, Menu, Food Details, Cart, Checkout, Order Success, Wishlist, Coupons, Login, Register and Profile. Cross-cutting client state (cart, applied coupon, authenticated user, wishlist) is managed via React Context providers (CartContext, UserContext, WishlistContext) and a Redux Toolkit store. The interface is built with Tailwind CSS, Framer Motion and Socket. IO-client is included as a dependency for real-time features to come.

B. API Layer

The backend is an Express.js app organized with Mongoose models, controller modules, and route modules that map HTTP verbs to controller functions – an MVC-style organization adapted for a headless API. CORS middleware, JSON body-parsing and a cookie-parser all at once used in API. The API is structured into two mounted namespaces, /api/users and /api/foods. A Cart data model and controller are also implemented, but not yet mounted as an active route (Section X).

C. Data Tier

Persistent data is stored in MongoDB, with access via Mongoose schemas which define validation rules at the application layer. This approach is in line with the flexible, document-oriented approach favored for catalogue-style e-commerce data over strictly normalized relational schemas [6], [11]. It has three collections: Users, Foods, and Carts detailed in Section VII.

Table 2. Technology Stack by Architectural Tier

Tier	Technology	Responsibility
Presentation	React 19, Vite, Router, Redux Toolkit, Tailwind, Framer Motion	Renders UI, manages navigation & client state
Application	Node.js, Express 5, JWT, bcrypt, Multer, Cloudinary, Socket.IO	Exposes REST endpoints, enforces auth
Data	MongoDB, Mongoose ODM	Persists users, food items, cart documents

Table II shows how each part of the system is built using technologies and what each part is responsible, for.

3.2 DATABASE AND ENTITY-RELATIONSHIP DESIGN

FoodFly uses three Mongoose schemas to represent the main elements of the application. Each of these schemas takes advantage of MongoDB's native capability to track when a document is created and updated. Figure 2 presents an overview of the connections between the entities.

The user schema contains all the information about a person's account like profile and customer's information like name, email address (all in lowercase letters), and a secret password. This will help in protecting their personal information's. They can also add a picture to their profile, if they want. There is a field that tells if they are a normal user or an admin which helps to decide what they can and cannot do.

The food schema is the list of all the food items that are there in the menu. Each food item has a name, type of food, description, price, picture and rating. The rating is usually 4.5. It also tells you how many people have rated the food and if it's available for purchase.

The Cart schema is similar to a shopping cart. It is linked to a user and contains a list of all the food items the user wants to purchase. Each food item in the cart specifies the type and quantity of food the user wishes. This way when users look at the cart they can see everything that is in the cart without having to look at a lot of other information. That makes it more convenient to use.

Table 3. MongoDB Collections and Key Fields

Collection	Key Fields	Notes
User	name, email(unique), password(hashed), phone, address, city, pincode, image, role	role: user|admin
Food	name, category, description, price, image, rating, reviews, is Available	rating default 4.5
Cart	user(ref), items[{food(ref), quantity}]	items embedded

3.3 MODULE IMPLEMENTATION AND ALGORITHMS

A. Authentication and Security

A separate controller implements user registration and login. When registering the server determines if there is an existing account with the same email, hashes the submitted password with bcrypt (salt-round factor 10) using the adaptive-hashing approach from [8] and saves the user document. On login, the password sent is compared with the stored hash using bcrypt's compare function. If successful, the server generates a signed JWT [9] including the user identifier with an expiration date that can be configured, and returns it along with a sanitized user object excluding the password hash. It's stateless, so you don't have to keep sessions on the server, which helps with horizontal scaling.

A. Cart Aggregation Algorithm (On Client Side)

Algorithm 1 shows the client-side logic for updating the cart which is coded in CartContext: when a food item is added, the cart array is scanned for an item identifier which matches the added item, if found, its quantity is incremented, otherwise a new entry is appended with quantity one. Removal filters the matching entry out of the array. The CartContext also stores the currently applied coupon object used by the checkout total computation described below.

Algorithm 1: AddToCart(cart, food)

1. Match ← find item in cart where item.id = food.id
2. If match exists then match.quantity ← match.quantity + 1
3. Else append {food, quantity: 1} to cart
4. Return updated cart

A. Checkout Total Computation Algorithm

The Checkout component computes the order total from the cart, a delivery fee, a flat 5% goods-and-services charge, and any applied coupon discount, as shown in Algorithm 2.

Algorithm 2: ComputeGrandTotal(cart, coupon)

1. subtotal ← Σ (price(i) × quantity(i)) for i in cart
2. delivery ← 0 if subtotal ≥ 500 else 40
3. gst ← round(subtotal × 0.05)
4. discount ← numeric value parsed from coupon.discount, or 0 if no coupon
5. grandTotal ← subtotal + delivery + GST − discount
6. return grandTotal

B. Menu Search and Filter Algorithm

The menu component filters the food catalogue by a case-insensitive substring match of the search query against the item name or category, conjoined with an equality check against the selected category (or "All"), so that only items satisfying both predicates are rendered.

C. Wishlist Toggle Algorithm

The WishlistContext toggles membership by checking whether an item identifier already exists in the wishlist array.

The WishlistContext toggles membership by checking whether an item identifier already exists in the wishlist array, removing it if present and appending it if not, which provides constant-time UI feedback for favoring or unfavoring an item.

D. API Design

The API follows conventional REST semantics [10]. The

The /api/users namespace exposes POST /register and POST /login. The /api/foods namespace exposes a full CRUD surface, supporting an administrative workflow for menu management alongside customer-facing browsing, as summarized in Table IV.

Table 4. Core REST API Endpoints

Method	Endpoint	Description
POST	/api/users/register	Registers a new user account
POST	/api/users/login	Authenticates a user, issues JWT
GET	/api/foods	Retrieves all food items

GET	/api/foods/:id	Retrieves one food item
POST	/api/foods	Creates a food item
PUT	/api/foods/:id	Updates a food item
DELETE	/api/foods/:id	Deletes a food item

4. RESULTS AND DISCUSSION

Functional verification of the API layer was carried out through manual endpoint testing to confirm correct status codes and payload shapes for registration, login, and food CRUD operations, including negative-path cases such as duplicate-email registration and invalid-password submission, both returning descriptive JSON errors with appropriate HTTP status codes. Table V summarises the current implementation and integration status of each module, reported transparently as required for an accurate account of the system's readiness.

Table 5. Module Implementation and Integration Status

Module	Backend Status	Frontend Status
User auth	Implemented (bcrypt+JWT)	UI present; not yet wired to API
Food catalogue	Full CRUD implemented	Renders local/mock data
Cart	Model + stub controller only	Fully functional via Context
Wishlist	Not implemented server-side	Fully functional via Context
Coupons	Not implemented server-side	Client-side static lookup
Checkout	No order persistence	Computes total; confirms order

The audit reveals that the authentication and food-catalogue REST endpoints are functionally complete on the server-side, independently verified through direct HTTP requests, while the customer-facing client currently operates against local component state and a static, in-memory food list instead of the live API—the Axios dependency listed in the project is not yet invoked from any component. This is a typical and sensible intermediate state during incremental full-stack development: it checks UI/UX and backend logic separately, before the higher-risk step of connecting them. Qualitative evaluation showed that the schema-flexible, document-orientated data tier and modular component-based frontend facilitated iterative feature addition, consistent with the productivity benefits observed for MERN-based systems in [5],[7].

4.1 BENEFITS

- A single language (JavaScript/JSX) for both client and server reduces context switching and allows for shared tooling.
- MongoDB's flexible document schema supports catalog items with variable or evolving attributes without migrations.
- JWT authentication is stateless, so the server doesn't need to store session data, which helps with horizontal scalability.
- A React frontend that is component-based encourages reuse and independent development of UI modules.
- The presentation, application and data tiers are separated cleanly and each can be tested and deployed independently.

4.2. LIMITATIONS

- The client does not yet invoke the authentication or food-catalogue API. The Login and Register pages are UI scaffolds without submit handlers.
- Cart persistence is only in the browser-side context state. The server-side cart model and controller are implemented, but not mounted as a route.
- No payment processing integration; checkout ends at order confirmation step without live payment gateway.
- Order tracking and live status updates are not implemented, despite Socket. IO is a dependency.
- The schema declares role-based route protection (role field) but the authorization middleware does not yet enforce it.
- Automated unit/integration testing has not yet been implemented; verification has been manual so far.

5. SCOPE FOR FUTURE

The next immediate step is to wire up the existing Axios dependency to the authentication and food endpoints, replacing local mock data and static forms with live API calls, and to mount the existing Cart controller behind an authenticated route. Besides this connection work, future iterations may include payment-gateway integration, Socket. The project will include IO-based real-time order tracking, admin-only route protection with the existing role field, automated testing with Jest/Vitest and Supertest, multi-vendor (restaurant-level) support, geolocation-based delivery estimation, and analytics dashboards for order and revenue trends.

6. CONCLUSION

This paper presented FoodFly, a MERN stack based full-stack web application for ordering and delivering food online, and compared it with similar systems reported in the literature [1]–[5]. The system demonstrates a real-world three-tier architecture with a functionally complete, independently verified authentication and food-catalogue REST API, a flexible MongoDB document schema, and a component-based React frontend implementing cart, wishlist, coupon, and checkout interactions on the client side. The paper has honestly reported the current integration status of each module rather than assuming full end-to-end wiring, in line for accurate technical account. The project shows the practical benefits of using a unified JavaScript stack for full-stack development, and the identified integration and feature gaps define a concrete roadmap to complete the platform.

7. REFERENCES

[1] Y. Wang, J. Zhang, and X. Wang, "Design and Implementation of an Online Food Ordering System Based on Web Technology," in Proc. IEEE Int. Conf. on Computer and Communications (ICCC), 2021, pp. 1285–1290, doi: 10.1109/ICCC54389.2021.9674726.

[2] R. R. Jadhav and S. M. Patil, "Web-Based Food Ordering System Using PHP and MySQL," Int. J. Scientific and Research Publications, vol. 10, no. 4, 2020, doi: 10.29322/IJSRP.10.04.2020.p10043.

[3] M. A. Hossain and M. S. Hossain, "E-commerce Web Application for Restaurant Ordering System," in Proc. IEEE Int. Conf. on Innovations in Science, Engineering and Technology (ICISET), 2022, doi: 10.1109/ICISET54378.2022.9747865.

[4] K. Acharya, "Online Food Order System," Research Proposal, ResearchGate, May 2024.

[5] Darshan R., A. Chabukswar, J. Ranjith Vivek, and S. Pahuja, "A Web-Based Food Ordering Platform Quickbites Using MERN Stack," Conference Paper, Sep. 2025.

[6] K. Banker, P. Bakkum, S. Verch, D. Garrett, and T. Hawkins, MongoDB in Action, 2nd ed. Shelter Island, NY, USA: Manning Publications, 2016.

[7] S. Holmes, Getting MEAN with Mongo, Express, Angular, and Node, 2nd ed. Shelter Island, NY, USA: Manning Publications, 2020.

[8] N. Provos and D. Mazières, "A Future-Adaptable Password Scheme," in Proc. USENIX Annual Technical Conf., 1999, pp. 81–91.

[9] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," IETF RFC 7519, May 2015.

[10] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, Univ. of California, Irvine, CA, USA, 2000.

[11] M. Stonebraker, "SQL Databases v. NoSQL Databases," Commun. ACM, vol. 53, no. 4, pp. 10–11, 2010.

[12] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," Commun. ACM, vol. 13, no. 6, pp. 377–387, 1970.

[13] E. Brewer, "CAP Twelve Years Later: How the \"Rules\" Have Changed," Computer, vol. 45, no. 2, pp. 23–29, 2012.

[14] T. Bray, "The JavaScript Object Notation (JSON) Data Interchange Format," IETF RFC 8259, Dec. 2017.

[15] R. Fielding, M. Nottingham, and J. Reschke, Eds., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content," IETF RFC 7231, Jun. 2014.

[16] Meta Open Source, "React — A JavaScript Library for Building User Interfaces," React documentation, Meta Platforms, Inc., 2024.

[17] OpenJS Foundation, "Express.js — Fast, Unopinionated, Minimalist Web Framework for Node.js," Express.js documentation, 2024.

[18] OpenJS Foundation, "Node.js Documentation," OpenJS Foundation, 2024.

[19] MongoDB, Inc., "Mongoose ODM Documentation," 2024.

[20] Okta, Inc., "JSON Web Tokens Introduction," Auth0 Documentation, 2023.

[21] S. Hoque, Full-Stack React Projects: Learn MERN Stack Development by Building Modern Web Apps, 2nd ed. Birmingham, UK: Packt Publishing, 2021.

[22] Redux Toolkit Maintainers, "Redux Toolkit Documentation," Redux.js, 2024.

[23] Tailwind Labs, "Tailwind CSS Documentation," 2024.