



# Implementation of an Organic Food E-commerce Website for Bangladeshi Consumers

ARIFUR RAHAMAN

BACHELOR'S THESIS  
June 2024

Software Engineering

## ABSTRACT

Tampereen ammattikorkeakoulu  
Tampere University of Applied Sciences  
Software Engineering

RAHAMAN, ARIFUR:

Implementation of an Organic Food E-commerce Website for Bangladeshi Consumers

Bachelor's thesis 41 pages

June 2024

---

Food adulteration is a major health concern in Bangladesh, creating a growing demand for reliable supplies of fresh, unadulterated food. This thesis describes the development of an e-commerce website that addresses this urgent need by providing a trustworthy platform for acquiring high-quality organic and unadulterated food items.

The objective of this study was to develop a reliable e-commerce platform focused on delivering organic and unadulterated food products. The website employs the latest web technologies to offer a powerful, user-friendly, and efficient online shopping experience. The front end was created using React.js and designed with Material UI to ensure a responsive user interface. The back end was built with Node.js and Express, providing a robust server-side architecture. MongoDB was used for database management, offering reliable and adaptive data storage.

The main features implemented in the project include a user registration and login system, authentication, user profiles, dynamic product listings, add-to-cart functionality, cart management, and checkout processes. These features enable users to easily sign up, log in, explore products, add items to their cart, manage their cart, and update their profiles.

---

Key words: e-commerce, bangladesh, react, organic food, web development

## CONTENTS

1	INTRODUCTION .....	5
2	RESEARCH ON FOOD ADULTERATION AND ITS IMPACT .....	6
	2.1 What is Food Adulteration .....	6
	2.2 Food Adulteration Worldwide .....	6
	2.3 Health Effects of Food Adulteration.....	7
	2.4 The Issue of Food Adulteration in Bangladesh .....	8
3	E-COMMERCE .....	9
	3.1 Overview of E-commerce.....	9
	3.2 E-commerce Growth .....	9
	3.3 E-commerce Business Models.....	10
4	PROJECT OVERVIEW.....	11
	4.1 Overview of Technologies.....	11
	4.1.1 React.js .....	11
	4.1.2 Node.js .....	11
	4.1.3 Express.js.....	11
	4.1.4 React Router Dom.....	12
	4.1.5 JSON Web Token (JWT).....	12
	4.1.6 Node package manager (npm).....	12
	4.1.7 MongoDB .....	13
	4.1.8 Material UI .....	13
	4.1.9 Redux .....	14
	4.2 Key Features Of Organic Food E-Commerce Platform .....	14
	4.2.1 Features For Admin.....	14
	4.2.2 Features For Users.....	15
	4.3 Graphical User Interface of Organic Food E-commerce Platform	15
5	SYSTEM DESIGN .....	17
	5.1 Client-Side Application Architecture .....	17
	5.2 Server-Side Application Architecture.....	19
	5.3 Database Design.....	23
6	TECHNICAL IMPLEMENTATION.....	24
	6.1 Server-Side Configuration.....	24
	6.2 Client-Side Configuration .....	24
	6.3 Backend Implementation.....	25
	6.3.1 Basic Setup .....	25
	6.3.2 Authentication and Authorization .....	27
	6.4 Frontend Implementation .....	31

7	FUTURE DEVELOPMENT .....	36
7.1	Ongoing Improvement.....	36
7.2	Implementation Challenges.....	36
7.3	Deployment.....	37
8	DISCUSSION .....	38
	REFERENCES .....	40

## 1 INTRODUCTION

This thesis project planned to develop an e-commerce website dedicated to offering fresh, unadulterated food to Bangladeshi consumers because finding fresh, pure food has become extremely difficult in Bangladesh. Food adulteration causes major health risks and harms public trust in food quality and safety. As consumers become more aware of these issues, the desire for trustworthy sources of fresh, unadulterated food is increasing, and e-commerce offers a possible solution to this problem. We can connect customers with trusted vendors through online platforms, making high-quality, unadulterated food easily available.

The primary goal is to build a user-friendly and efficient online platform called Organic Food that offers a wide range of high-quality organic and beneficial food products. The website will offer fresh items and natural health products acquired from suppliers who maintain the highest food quality and safety requirements.

For the website, frontend was built with React.js to ensure a responsive and dynamic user experience. The backend, managed by Node.js and Express.js, handles user authentication, routing, server-side functionality, and APIs. MongoDB stores product data, user data, order details, and reviews, offering effective data management. Material UI was used for consistent and responsive styling with pre-designed components. Redux manages complex interactions like user authentication and the shopping cart, maintaining the application's state.

## **2 RESEARCH ON FOOD ADULTERATION AND ITS IMPACT**

### **2.1 What is Food Adulteration**

Food adulterant is any substance that is added to food to reduce its quality and increase its quantity. Low-quality chemical compounds that are toxic in food are called food adulterants, these can be added intentionally, to reduce costs and increase profits, or unintentionally, although some non-nutritive ingredients may be added in small amounts to improve food flavour, texture, or storage life, these intentional additions are largely profit-driven, often compromising consumer health (GeeksforGeeks 2024).

Common adulterants include water, salt, sugar, and spices. Colors, preservatives, and stabilizers are often used, and can even be adulterated with basic food items such as milk, eggs, oil, and flour. Hazardous chemicals like calcium carbide, sodium cyclamate, cyanide, and formalin are used to artificially ripen, preserve, and extend the shelf life of fruits (Amin et al 2004; The Daily Prothom Alo 2005). Textile dyes are sometimes deliberately used to color vegetables, fruits, sweets, and beverages, which can be harmful to health. Fish are sometimes preserved with formalin to maintain their appearance and prevent decomposition (National Library of Medicine 2023).

### **2.2 Food Adulteration Worldwide**

Food adulteration is a significant global issue with serious public health and economic implications. The World Health Organization (WHO) estimates that 22% of food is adulterated, affecting 57% of the global population and costing between US\$10 billion to \$40 billion annually. Developing countries are particularly impacted, with the Food and Agriculture Organization (FAO) reporting adulteration rates as high as 50% in some areas.

In the United States, the Food and Drug Administration (FDA) found that 33% of imported food samples were rejected for adulteration in 2021, with lead, arsenic,

mercury, pesticides, and undeclared allergens being the main contaminants. The European Food Safety Authority (EFSA) Highlighted in 2022 that 1.2% of food samples in the EU did not comply with food safety regulations, Major problems identified were pesticide residues that surpassed safe thresholds, the presence of unauthorized additives, and incorrect labeling.

The International Food Safety Network (INFOSAN) reported a 40% rise in worldwide food safety incidents from 2017 to 2021. Most of the increase was associated with food fraud and adulteration, revealing a serious worldwide issue (Global Indian Network n.d.).

### 2.3 Health Effects of Food Adulteration

Adulterants	Health Hazards
Calcium carbide, sodium cyclamate, cyanide, formalin, low-cost textile dyes.	liver and kidney failure, autism, metabolic dysfunctions, and cancer.
Malachite green, oxytocin saccharin, wax, calcium carbide, and copper sulphate.	Stomach disorders, Vomiting.
Chalk, urea, caustic soda, hydrogen peroxide.	Food poisoning, vomiting, nausea, and Heart Problems.
Soapstone, papaya seeds, brick powder, talc powder, artificial colors, metanil yellow, red oxide of lead, and colored dried tendrils of maize cob are often used as adulterants.	Severe allergic reactions, including stomach and skin irritations, cancer, and stomach disorders, can occur.
Dust, pebbles, stones, straw, and weed seeds are common contaminants.	Liver disorders and toxicity in the body can occur.

TABLE 1. Food adulterants and their effects on the human body (GeeksforGeeks n.d.).

## 2.4 The Issue of Food Adulteration in Bangladesh

In Bangladesh, harmful chemicals and substances are purposely added to food products to increase their weight or appearance. These practices include the use of artificial colors, artificial flavors, chemical preservatives, pesticides, fertilizers, antibiotics, and hormones.

In 2019, a survey conducted by the Bangladesh Food Safety Authority (BFSA) found that 52% of food samples across the country were contaminated. A study by the Bangladesh Institute of Development Studies found that foodborne illness affects an estimated 26 million people annually. Young children especially those under the age of five, are at high risk of being affected by contaminated food, The study indicates that foodborne illness costs Bangladesh 2% of its overall GDP.

Food adulteration is widespread in Bangladesh, affecting almost all types of food products, including milk, fish, fruits, vegetables, spices, and oils. Common adulterants such as formalin and pesticides are frequently found in these products. Formalin, commonly used to preserve corpses, is often found in fish, fruits, and vegetables, which can cause health problems such as cancer and kidney failure if consumed over time. Calcium carbide is used to artificially ripen fruits, causing nausea, diarrhea, and other health problems. Pesticides are often used in agriculture to protect crops from harmful insects and increase their yield in Bangladesh. However unscrupulous sellers usually use excess amounts of pesticides. Excessive pesticide intake can cause cancer, birth defects, and other health problems.

Milk is an essential source of nutrition for people worldwide. However, the quality of milk in Bangladesh is greatly affected by adulteration as unscrupulous sellers often add water, detergents, and other chemicals to increase milk volume, which significantly compromises its nutritional value and safety. Oils are similarly adulterated by mixing high-quality oils with low-quality oils and adding harmful substances like palm oil and argemone oil, leading to health problems like heart disease and stroke (Md Muzakkerrul 2023).



## **3 E-COMMERCE**

### **3.1 Overview of E-commerce**

E-commerce, short for electronic commerce, describes companies and individuals who purchase and sell products and services online (Andrew, B 2024). The use of technology and digital platforms has made e-commerce a reality, where websites, mobile applications, and social media are used as channels for buying and selling (Christiana 2024).

### **3.2 E-commerce Growth**

E-commerce has grown rapidly worldwide, with a global market value of \$18.98 trillion in 2022 and expected to reach \$47.73 trillion by 2030, representing a compound annual growth rate of 12.22% (Marketing Report 2024). This growth is due to the increasing availability of the internet, usage of mobile devices, and the preference of consumers to buy products online. Online retail shopping is projected to account for 20.1% of total retail sales in 2024, with digital orders steadily increasing. Large companies like Amazon lead the market, contributing 37.6% of total e-commerce sales. Furthermore, social media commerce is on the rise, with \$992 billion spent on social media shopping in 2022 and an estimated \$8.5 trillion by 2030.

Mobile shopping also plays an important role, with smartphones making up 91% of internet shopping. The global fresh food e-commerce market is anticipated to experience substantial growth. In 2019, the global food e-commerce market was valued at USD 138,833 million and is projected to grow to USD 1,116,201.88 million by 2031, with a compound annual growth rate (CAGR) of 18.97% during the forecast period (business research insights 2024). However, growth brings concerns, such as e-commerce fraud, which could lead to \$41 billion in losses by 2022. Overall, the e-commerce industry is dynamic, providing significant opportunities for companies to expand their reach and increase global sales.

### 3.3 E-commerce Business Models

Companies use a variety of business models, each presenting specific challenges. Many companies employ a combination of models to achieve their objectives. Here are the most common types of e-commerce business models:

- B2C (business-to-consumer) In this model companies sell their products directly to customers.
- B2B (business-to-business) This model describes a situation where companies sell products or services to other companies, which then sell to consumers (Mary 2024).
- B2B2C (business-to-business-to-consumer) This occurs when one company works with another company to sell products to customers while maintaining transparency about the product's origin.
- B2G (business-to-government) describes a business model where companies sell products or services to government agencies such as city councils.
- C2B (consumer-to-business) This business model allows individuals to sell products or services to companies (McFarland 2022).
- D2C (direct-to-consumer) companies use this model to sell their items directly to customers, cutting out third-party sellers such as retailers.
- C2C (consumer-to-consumer) In this model, platforms enable regular people to buy and sell items to each other (Big Commerce n.d.).

## **4 PROJECT OVERVIEW**

### **4.1 Overview of Technologies**

#### **4.1.1 React.js**

React is a JavaScript library for creating user interfaces, mostly for single-page web applications. React, created by Facebook, enables developers to create responsive and dynamic UI elements that effectively reload and render changes when data or state changes. React has a component-based architecture, which divides UI elements into reusable and modular components, making it easier to manage and maintain complicated UI structures. React virtual DOM updates the browser's DOM faster, which improves performance. For this reason, React has become extremely popular in the software development community (React n.d.).

#### **4.1.2 Node.js**

Node.js is a runtime environment and platform that allows developers to execute JavaScript code on the server. Ryan Dahl built it in 2009, and it has since become a popular platform for developing robust and effective web applications. Node.js is well-known for its asynchronous, event-driven architecture, which allows numerous tasks to be handled simultaneously without disrupting the performance of other tasks. It is commonly used in web development to build server-side apps, APIs, and real-time applications (Simplilearn n.d.).

#### **4.1.3 Express.js**

Express.js is a web application framework for Node.js, that provides extensive features for building web and mobile applications. Express.js allows developers to build multiple types of web applications such as single-page, multi-page, and hybrid web applications (Simplilearn n.d.).

Express makes the process of developing web-based applications easier by offering a set of features and tools including routing, middleware, and HTTP requests and responses (Kinsta 2022).

#### **4.1.4 React Router Dom**

React Router Dom is a npm package that enables dynamic routing in web applications, specifically React-based single-page applications. It enables developers to create routes and move between different pages or components without the need for page refresh. This improves the user experience by making navigating smoother and faster than in typical page-based applications (GeeksforGeeks 2024).

#### **4.1.5 JSON Web Token (JWT)**

JSON Web Token (JWT) is an open standard (RFC 7519) that enables secure transmission of information as a digitally signed JSON object. Security and data integrity are ensured by signing the JWT using either the HMAC algorithm or a public/private key pair (RSA or ECDSA) (JWT.IO 2017).

JWTs are widely used for user authorization due to their simplicity and strong security measures. After a user logs in, each subsequent request contains a JWT, which is validated to grant authorization. Upon successful validation, the server decodes the Base64Url-encoded payload to access user information, thereby allowing authorized users to access specific resources, services, and routes within the application. Furthermore, JWTs have an expiration time, requiring periodic token updates to maintain security (JWT.IO 2017).

#### **4.1.6 Node package manager (npm)**

NPM, or Node Package Manager, is the default tool for managing and installing third-party packages and dependencies in Node.js applications. It can be

accessed through [www.npmjs.com](http://www.npmjs.com), which is the largest software registry globally. Developers of popular stacks like MERN often use NPM to install packages like Express.js, MongoDB, and React libraries. NPM's command-line interface simplifies the process of installing and uninstalling packages (Kinsta 2022).

#### 4.1.7 MongoDB

MongoDB is an open-source document-based database that can efficiently store large amounts of data. It is a type of NoSQL database that uses a flexible and scalable design. Unlike traditional SQL databases that store data in tables with rows and columns, MongoDB stores each piece of data as a BSON (a binary format) document, which can be easily converted to JSON for use. MongoDB was first released in February 2009 (GeeksforGeeks 2021).

```
{
  "_id": 1,
  "name": {
    "first": "Charles",
    "last": "Babbage"
  },
  "title": "Father of the Computer",
  "interests": ["engineering", "computing"]
}
```

PICTURE 1. Example of a JSON document describing a historical figure.

MongoDB's architecture allows it to efficiently manage large amounts of data across multiple servers, making it fast and efficient. MongoDB is popular among developers for a wide range of applications (MongoDB n.d.).

#### 4.1.8 Material UI

Material-UI is a React component library that implements Google's Material Design guidelines, offering pre-designed components, styles, and theming options. Developers can quickly create modern and visually appealing web interfaces

using its wide range of components such as buttons, forms, and menus, all following Material Design standards. Its theming system enables customization of colors, typography, and spacing for consistent design across applications. Prioritizing accessibility, overall, Material-UI streamlines the development of attractive and responsive applications (Material UI n.d.).

#### **4.1.9 Redux**

Redux is an open-source JavaScript library used to manage application state. It provides a predictable state container for JavaScript applications, making it easier to manage and debug state changes. Redux was created in 2015 by Dan Abramov and Andrew Clark and is a widely used state management library in modern web development (Javatpoint n.d.).

## **4.2 Key Features Of Organic Food E-Commerce Platform**

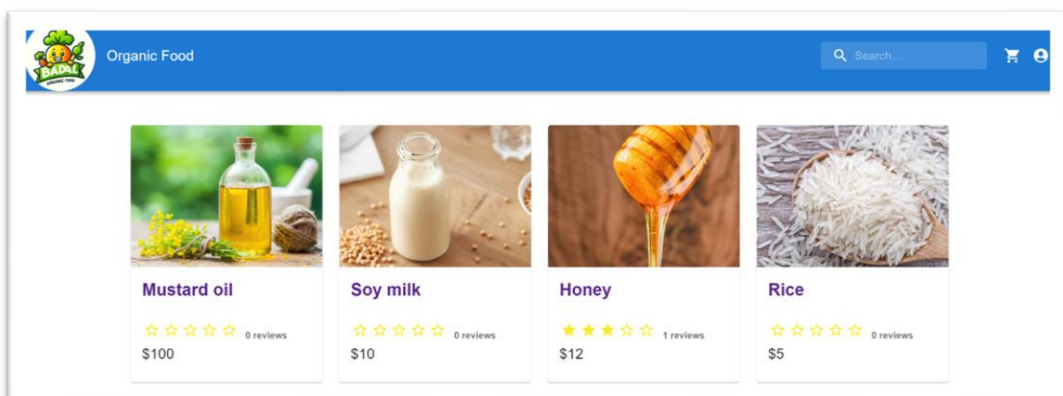
### **4.2.1 Features For Admin**

- **Product Management:** Admin can control product listings, add, edit, or remove items, and manage items to keep items in stock.
- **Order Management:** Admin can track orders, and update order status.
- **User Management:** The admin can manage, create, edit, or delete user accounts and assign various permissions.
- **Feedback and Review Management:** Admin can monitor and respond to user feedback and reviews, ensuring that customer concerns are addressed promptly and efficiently.
- **Financial Oversight:** Admin can manage financial transactions, process refunds, and manage payment gateways with financial and expense reports.

### 4.2.2 Features For Users

- Finding Products and Managing Cart: Users can easily find products and manage their shopping carts, adding, removing, and adjusting quantities effortlessly.
- Secure Payments and Order Tracking: Ensure safe transactions and protect user data while allowing users to track orders from confirmation to delivery.
- Product Reviews and Special Offers: Users can leave reviews, and write comments, and admin can manage them. Additionally, offer discounts and promotions to attract customers.
- User Accounts and Search Filters: Users can easily create and manage accounts and search for products with convenient filters.
- Mobile Compatibility: Users can enjoy a seamless shopping experience on mobile devices.
- Account Management: Users can update their profile information, and manage payment methods.

### 4.3 Graphical User Interface of Organic Food E-commerce Platform



PICTURE 2. Main homepage.

The homepage welcomes users warmly and guides them with clear buttons. Highlighted items grab their attention, while the search bar helps them find specific items.

Organic Food

Search...

SIGN IN

### Sign In

Email Address \*

Password \*

SIGN IN

[New Customer? Register](#)


PICTURE 3. Sing up page.

The sign-up page enables users to create an account, allowing them to save favorite items, track orders, and purchase products.

Organic Food

Search...

GO BACK



### Rice

☆☆☆☆ 0 reviews

Price: \$5

Status: In Stock

Qty: 1

ADD TO CART

### Reviews

No Reviews

### Write a Customer Review

Please sign in to write a review

PICTURE 4. Add to cart and review page.

This page is like a shopping basket where users keep all the things they want to buy, and the review page allows users to share their feedback and opinions about specific products.



## 5 SYSTEM DESIGN

### 5.1 Client-Side Application Architecture

In this e-commerce application, the App component serves as the central hub for routing, branching out into key domains, each serving a specific function. The HomeScreen component is the default landing page and handles search functionality and pagination. The ProductScreen component displays detailed information about specific products. The CartScreen displays the items added by the user for purchase. The LoginScreen and RegisterScreen allow users to log in or register.

The PrivateRoute component restricts access to certain pages to authorized users only. Within the restricted area, the ShippingScreen collects shipping details, the PaymentScreen handles transactions, the PlaceOrderScreen allows users to review and confirm their orders, the OrderScreen displays past orders, and the ProfileScreen lets users view and edit their profile.

The AdminRoute component restricts access to administrative pages to authorized users only. Within the administrative area, the OrderListScreen displays all transactions, the ProductListScreen handles product management, the UserListScreen manages user accounts, the ProductEditScreen allows for product editing, and the UserEditScreen allows for user account editing.



FIGURE 1. Client-side architecture of the react application.

Here is a clear and detailed explanation of the diagram as follows

- The **App** component is the main entry point of the application, connecting to different screens through defined routes to direct users to the appropriate screens based on their interactions.
- The **HomeScreen** component is linked to the **App** component through multiple routes, allowing users to view the home page, search for products using keywords, and navigate through different pages of search results.

- For detailed product information, users can access the ProductScreen component via the route `/product/:id`, and the CartScreen component for managing the user's shopping cart is accessible through the `/cart` route.
- User authentication is handled by the LoginScreen and RegisterScreen components, accessible through the `/login` and `/register` routes.
- Protected routes ensure that only authenticated users can access certain screens, including user-specific and admin-specific screens. The PrivateRoute component verifies user authentication before granting access to specific routes.
- Administrative functionalities are managed through the AdminRoute component, which ensures that only users with administrative privileges can access certain screens. This includes screens for managing orders, products, and user accounts.

## 5.2 Server-Side Application Architecture

In this e-commerce application, the backend architecture is well-organized to handle various functionalities, ensuring seamless operation and user experience. The central hub of this backend system is the server file, which manages routing and CRUD operations across different domains such as products, users, orders, and file uploads.

Starting with the product routes, the application provides a series of endpoints to manage products. The root path for product-related operations is `/api/products`. Here, the application handles retrieving a list of products through a GET request. When an admin user wants to create a new product, they send a POST request to the same endpoint, but this action is protected, meaning only authenticated admins can perform it. For adding product reviews, a POST request is made to `/api/products/reviews`, where the ID of the product is validated before proceeding. Top-rated products are fetched through a GET request to `/api/products/top`. Additionally, specific product details can be accessed by sending a GET request to

`/api/products/`, with ID validation to ensure accuracy. Similarly, updating and deleting a product are achieved through PUT and DELETE requests to the same path, with both operations restricted to authenticated admins.

User-related operations are managed under `/api/users`. User registration is facilitated through a POST request to the root of this endpoint, while the list of users, accessible only by admins, is fetched via a GET request. For authentication purposes, users send a POST request to `/api/users/auth`, and to log out, they use a POST request to `/api/users/logout`. The user profile can be viewed and updated by making GET and PUT requests, respectively, to `/api/users/profile`, with both actions restricted to authenticated users. Admins have additional capabilities, such as deleting a user through a DELETE request to `/api/users/`, retrieving user details with a GET request, and updating user information with a PUT request, all of which require appropriate admin permissions.

Order management is handled through the `/api/orders` endpoint. Users can add new orders by sending a POST request to the root of this endpoint, and admins can retrieve all orders via a GET request. Individual users can access their orders by making a GET request to `/api/orders/mine`. To fetch specific order details, a GET request is sent to `/api/orders/`. Furthermore, updating the payment status of an order is done through a PUT request to `/api/orders/pay`, while marking an order as delivered requires a PUT request to `/api/orders/deliver`, with the latter restricted to admins.

File uploads are managed through the `/api/upload` endpoint. The system uses `multer` for handling file uploads, with specific configurations for storage and filtering to ensure only image files (JPEG, PNG, WEBP) are accepted. When an image is uploaded, a POST request is sent to the root of this endpoint, and the file is processed, saved to the designated directory, and a success message is returned.

The application also integrates PayPal for payment processing. The PayPal client ID is retrieved by sending a GET request to `/api/config/paypal`, allowing the frontend to configure and utilize PayPal services. Note that this payment integration is not fully functional yet and should be explicitly noted as a work in progress.

For serving static files and handling deployment configurations, the application distinguishes between production and development environments. In production, static files are served from the frontend/build directory, and uploaded files are accessible from /var/data/uploads. In development, the application serves uploaded files from the local uploads directory and provides a simple response for the root path to indicate that the API is running.

To ensure robustness, the backend includes custom middleware for error handling. The notFound middleware catches 404 errors for undefined routes, while the errorHandler middleware manages other server errors, providing a structured response to the client.

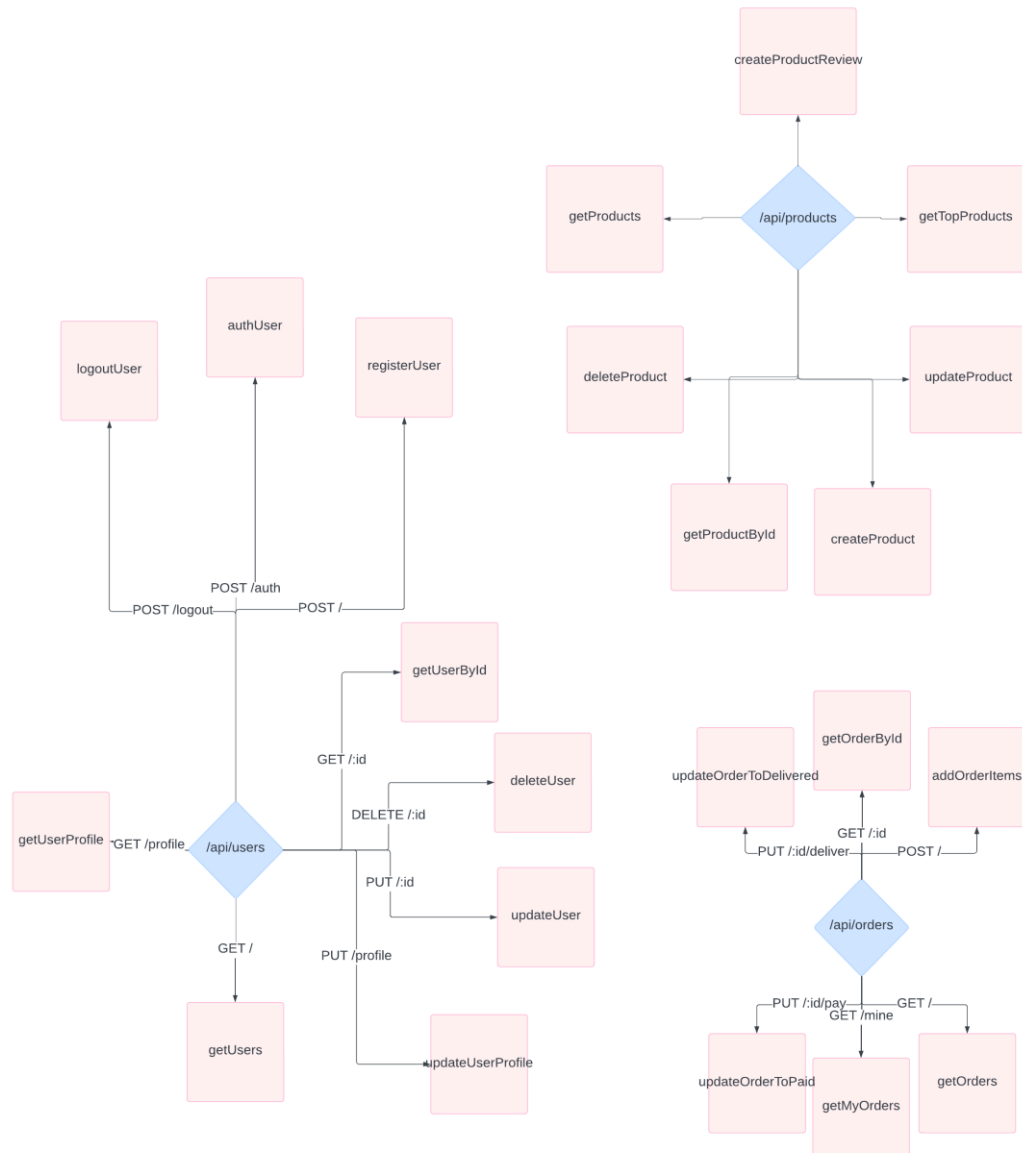


FIGURE 2. Server-side application architecture.

Here is the detailed explanation of the diagram –

- To manage products, use the `/api/products` endpoint. Users can view, add, update, and delete products using GET, POST, PUT, and DELETE requests.
- For user management, the `/api/users` endpoint handles user registration, authentication, profile viewing, and updating. Admins can access user information and delete users using GET, PUT, and DELETE requests.

- Order management is facilitated through the `/api/orders` endpoint, allowing users to add items, view orders, and update order status.

### 5.3 Database Design

The database is crucial for data storage. Its design defines the system's structure and ensures clarity, outlining the e-commerce platform's operations such as user management, product management, inventory management, shopping cart management, and payment processing. These functions are vital for the daily operations of an e-commerce platform. To ensure data is stored efficiently, accurately, and securely, the database design requires careful and detailed planning (Lim 2020).

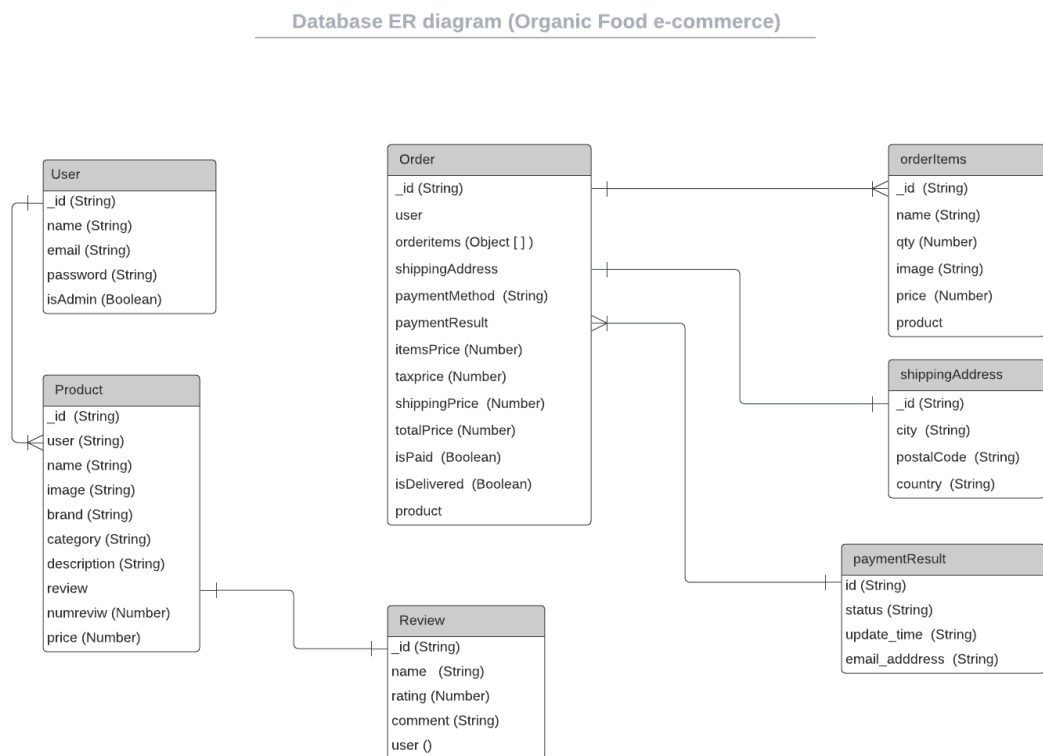


FIGURE 3. Schema design diagram.

In this schema, relationships are depicted with lines connecting entities. For example, the Order entity is connected to the User, OrderItems, ShippingAddress, and PaymentResult entities, illustrating that an order is placed by a user, includes multiple order items, are shipped to an address, and has a payment result. Similarly, the Product entity is connected to the Review entity, indicating that products can have multiple reviews.

## 6 TECHNICAL IMPLEMENTATION

### 6.1 Server-Side Configuration

To establish the server-side configuration of the e-commerce application, the focus is on setting up the backend built with Node.js and Express, along with integrating MongoDB.

We begin by defining the main entry point in the `package.json` file, where `server.js` serves as the starting point for the backend server. This file orchestrates the backend operations, handling routing, controllers, and database interactions.

For smoother development, npm scripts are employed. The `start` script launches the backend server using Node.js, while the `server` script harnesses the power of nodemon to automatically restart the server upon file modifications, enhancing the development experience.

MongoDB, our database solution, plays a pivotal role in storing application data. We set up a MongoDB Atlas account, a cloud-based database service, and created a new cluster to obtain a unique connection string. This connection string is utilized in `server.js` to establish a connection to the MongoDB Atlas cluster using the Mongoose library, enabling seamless interaction with the database.

Within `server.js`, routes and controllers are defined to encapsulate API functionality. For instance, routes such as `/api/users/register` handle user registration, while `/api/users` manage user data retrieval. These routes are designed to handle incoming requests, process them accordingly, and send back appropriate responses.

### 6.2 Client-Side Configuration



On the client side, which encompasses the frontend developed using React, the focus shifts to setting up dependencies, scripts, and configuring the development environment.

Dependencies essential for the React application are outlined in the `package.json` file within the frontend directory. These dependencies include libraries and packages necessary for building the frontend components and managing the state.

The React app is initialized using npm scripts. The start script launches the development server, allowing developers to preview changes in real-time. The build script compiles the React application, optimizing it for production deployment.

To enable simultaneous development of frontend and backend, the `concurrently` package is utilized. The dev script runs both the backend server and the frontend development server concurrently, streamlining the development process and facilitating rapid iteration.

Additionally, the frontend's `package.json` includes a proxy configuration directing API requests to the backend server running on `http://localhost:5000`. This proxy configuration ensures seamless communication between the frontend and backend during development, enabling frontend components to interact with backend APIs without encountering cross-origin resource sharing (CORS) issues.

## **6.3 Backend Implementation**

### **6.3.1 Basic Setup**

In this e-commerce application, the backend serves as the foundation for managing data, handling requests, and ensuring secure operations. The implementation begins by setting up the necessary libraries and configuring the Express server. The server setup begins by creating routes in the `server.js` file.

First, we import essential modules. Express, a minimal and flexible Node.js web application framework, facilitates server creation and route management. The dotenv module loads environment variables from a .env file, enhancing configuration management by keeping sensitive data like database URLs and API keys out of the codebase. cookieParser is included to parse cookies attached to the client requests, which is crucial for handling JWTs for authentication. Next, an instance of the Express application is created using the app variable.

```
import express from 'express';
import dotenv from 'dotenv';
import cookieParser from 'cookie-parser';
dotenv.config();
const app = express();
```

To handle incoming data, middleware functions are used. The express.json() middleware parses incoming JSON payloads, allowing the server to handle JSON data from client requests. Similarly, express.urlencoded({ extended: true }) is used to parse URL-encoded payloads, making it easier to handle form submissions. cookieParser() middleware is also used to parse cookies, enabling the server to manage session data and JWTs effectively. The server's port is then defined, with a fallback to port 5000 if the environment variable PORT is not set.

```
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(cookieParser());
const port = process.env.PORT || 5000;
```

For database connectivity, the MongoDB URL is fetched from environment variables. Using Mongoose, a MongoDB object data modeling (ODM) library, we connect to the MongoDB database. Mongoose simplifies interactions with the database by providing a straightforward schema-based solution to model application data. A successful connection logs a confirmation message to the console.

```
const mongoURL = process.env.DB_URL;
mongoose.connect(mongoURL).then(() => console.log('Database Connected!'));
```

Finally, the server is configured to listen on the specified port. When the server starts, it logs a message indicating the mode (development or production) and the port it is running on:

```
app.listen(port, () =>
  console.log(`Server running in ${process.env.NODE_ENV} mode on port
  ${port}`)
);
```

This basic setup forms the backbone of the backend, enabling the server to process incoming requests, handle data through a connected database, and manage user sessions securely. By leveraging environment variables, it also ensures that sensitive information is not hardcoded, promoting better security practices.

### 6.3.2 Authentication and Authorization

In this e-commerce application, authentication and authorization play a vital role in securing user data and controlling access to various functionalities based on user roles. The implementation of these features is woven into the backend server setup, middleware functions, route protection, and user model definitions, ensuring a comprehensive security framework. This section outlines the implementation of these security measures, focusing on JWT tokens for authentication and middleware for route protection.

For handling user-related routes, we create an instance of `express.Router()`. This modular router enables us to manage user-related endpoints more efficiently. Within the `userRouter`, we define various routes to handle user registration, authentication, profile management, and administrative functions. Each route is associated with specific controllers and middleware functions.

In this setup, the `registerUser`, `authUser`, and `logoutUser` functions are the controllers that handle respective requests. These controllers interact with the database to perform actions like registering new users, logging in, and logout of the application.

```
const userRouter = express.Router();

userRouter.route('/api/user/')
  .post(registerUser)
  .get(protect, admin, getUsers);
userRouter.post('/api/user/auth', authUser);
userRouter.post('/api/user/logout', logoutUser);
```

FIGURE 4. User authentication and authorization routes.

To secure the routes, we use two middleware functions: `protect` and `admin`. These functions ensure that only authenticated users can access certain routes and that only administrators have access to specific endpoints.

The `protect` middleware function checks if a JWT token is present in the cookies of the incoming request. If a valid token is found, the user is authenticated and allowed to proceed. The `admin` middleware function checks if the authenticated user has administrative privileges.

```

const protect = asyncHandler(async (req, res, next) => {
  let token;
  token = req.cookies.jwt;
  if (token) {
    try {
      const decoded = jwt.verify(token, process.env.JWT_SECRET);
      req.user = await User.findById(decoded.userId).select('-password');
      next();
    } catch (error) {
      console.error(error);
      res.status(401);
      throw new Error('Not authorized, token failed');
    }
  } else {
    res.status(401);
    throw new Error('Not authorized, no token');
  }
});

const admin = (req, res, next) => {
  if (req.user && req.user.isAdmin) {
    next();
  } else {
    res.status(401);
    throw new Error('Not authorized as an admin');
  }
};

```

FIGURE 5. The necessary middleware's for route protection.

The user model defines the structure of user documents in the MongoDB database. It includes fields for the user's name, email, password, and admin status. The model also includes methods for password encryption and verification.

```

import mongoose from 'mongoose';
const userSchema = mongoose.Schema(
  {
    name: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true },
    isAdmin: { type: Boolean, required: true, default: false },
  }
);
export const User = mongoose.model('User', userSchema);

```

FIGURE 6. Example user model.

The controller functions to handle the core logic for user authentication, registration, and profile management. The `authUser` function authenticates a user by checking their email and password. If valid, a JWT token is generated and sent as an HTTP-only cookie.

```
const authUser = asyncHandler(async (req, res) => {
  const { email, password } = req.body;

  const user = await User.findOne({ email });

  if (user && (await user.matchPassword(password))) {
    generateToken(res, user._id);

    res.json({
      _id: user._id,
      name: user.name,
      email: user.email,
      isAdmin: user.isAdmin,
    });
  } else {
    res.status(401);
    throw new Error('Invalid email or password');
  }
});
```

FIGURE 7. Auth login controller.

The `registerUser` function registers a new user. If the user does not already exist, it creates a new user and generates a JWT token for them. The `logoutUser` function clears the JWT cookie, effectively logging the user out.

```

const registerUser = asyncHandler(async (req, res) => {
  const { name, email, password } = req.body;
  const userExists = await User.findOne({ email });
  if (userExists) {
    res.status(400);
    throw new Error('User already exists');
  }
  const user = await User.create({ name, email, password });
  if (user) {
    generateToken(res, user._id);
    res.status(201).json({
      _id: user._id,
      name: user.name,
      email: user.email,
      isAdmin: user.isAdmin,
    });
  } else {
    res.status(400);
    throw new Error('Invalid user data');
  }
});

```

FIGURE 8. Register user controller.

```

const logoutUser = (req, res) => {
  res.clearCookie('jwt');
  res.status(200).json({ message: 'Logged out successfully' });
};

```

FIGURE 9. User logout controller.

Through the integration of JWT tokens, middleware for route protection, and controller functions, our e-commerce application securely manages user authentication and authorization. These mechanisms ensure that only authenticated users can access specific resources and that administrative functionalities are restricted to users with the appropriate privileges.

## 6.4 Frontend Implementation

The frontend of our e-commerce application is built using React, a popular JavaScript library for building user interfaces. React enables us to create a dynamic and responsive user experience, essential for modern web applications. In this section, we will explore the key components and features of the front-end implementation.

To set up the React application, we begin with a standard React project setup, which provides a solid foundation and a well-organized project structure. The `package.json` file in the frontend directory lists all necessary dependencies and scripts for running, building, and testing the application. This ensures that the development environment is consistent and that all required libraries and tools are readily available.

The key dependencies include React for building the UI, Redux for state management, React Router for navigation, Axios for making HTTP requests, and Material UI for styling. We can use `npm` to install these packages.

User authentication on the frontend involves several components and Redux actions to manage the user state and interact with the backend API. The `LoginScreen` component handles user login. It utilizes Redux actions to dispatch login requests and update the application state accordingly.



```

const LoginScreen = () => {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const dispatch = useDispatch();
  const navigate = useNavigate();

  const { userInfo } = useSelector((state) => state.userLogin);
  const { search } = useLocation();
  const sp = new URLSearchParams(search);
  const redirect = sp.get('redirect') || '/';
  useEffect(() => {
    if (userInfo) {
      navigate(redirect)
    }, [navigate, redirect, userInfo];
  }, [navigate, redirect, userInfo]);
  const submitHandler = (e) => {
    e.preventDefault();
    dispatch(login(email, password));
  };
  return (
    <div>
      <h1>Sign In</h1>
      <form onSubmit={submitHandler}>
        <input
          type="email" placeholder="Enter email" value={email}
          onChange={(e) => setEmail(e.target.value)} required />
        <input
          type="password" placeholder="Enter password" value={password}
          onChange={(e) => setPassword(e.target.value)} required />
        <button type="submit">Sign In</button>
      </form>
      <p>New Customer?
        <RouterLink to={redirect ? `~/register?redirect=${redirect}` : `~/register`} >Register</RouterLink>
      </p>
    </div>
  );
};

```

FIGURE 10. Example login component.

The RegisterScreen component handles user registration by providing a form for users to enter their name, email, password, and password confirmation. When the form is submitted, the submitHandler dispatches the register action to create a new user account. The Redux actions and reducers manage the state of user authentication and registration.

```

const RegisterScreen = () => {
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [confirmPassword, setConfirmPassword] = useState('');
  const dispatch = useDispatch();
  const navigate = useNavigate();
  const { userInfo } = useSelector((state) => state.userRegister);
  const { search } = useLocation();
  const sp = new URLSearchParams(search);
  const redirect = sp.get('redirect') || '/';
  useEffect(() => {
    if (userInfo) navigate(redirect);
  }, [navigate, redirect, userInfo]);

  const submitHandler = (e) => {
    e.preventDefault();
    if (password !== confirmPassword) alert('Passwords do not match');
    dispatch(register(name, email, password));
  };

  return (
    <div>
      <h1>Register</h1>
      <form onSubmit={submitHandler}>
        <input
          type="text" placeholder="Enter name" value={name}
          onChange={(e) => setName(e.target.value)} required />
        <input
          type="email" placeholder="Enter email" value={email}
          onChange={(e) => setEmail(e.target.value)} required />
        <input
          type="password" placeholder="Enter password" value={password}
          onChange={(e) => setPassword(e.target.value)} required />
        <input
          type="password" placeholder="Confirm password" value={confirmPassword}
          onChange={(e) => setConfirmPassword(e.target.value)} required />
        <button type="submit">Register</button>
      </form>
      <p>Already have an account? <RouterLink to={redirect} ?
        \ /login?redirect=${redirect}` : \ /login'\>Login</RouterLink></p>
    </div>
  );
};

```

FIGURE 11. Example sign-up component.

```
export const login = (email, password) => async (dispatch) => {
  try {
    dispatch({ type: USER_LOGIN_REQUEST });
    const config = { headers: { 'Content-Type': 'application/json' } };
    const { data } = await axios.post('/api/users/auth', { email, password }, config);
    dispatch({ type: USER_LOGIN_SUCCESS, payload: data });
    localStorage.setItem('userInfo', JSON.stringify(data));
  } catch (error) {
    dispatch({
      type: USER_LOGIN_FAIL,
      payload: error.response && error.response.data.message
        ? error.response.data.message
        : error.message,
    });
  }
};

export const register = (name, email, password) => async (dispatch) => {
  try {
    dispatch({ type: USER_REGISTER_REQUEST });
    const config = { headers: { 'Content-Type': 'application/json' } };
    const { data } = await axios.post('/api/users', { name, email, password }, config);
    dispatch({ type: USER_REGISTER_SUCCESS, payload: data });
    dispatch({ type: USER_LOGIN_SUCCESS, payload: data });
    localStorage.setItem('userInfo', JSON.stringify(data));
  } catch (error) {
    dispatch({
      type: USER_REGISTER_FAIL,
      payload: error.response && error.response.data.message
        ? error.response.data.message
        : error.message,
    });
  }
};
```

FIGURE 12. Redux action for login and register.

## 7 FUTURE DEVELOPMENT

### 7.1 Ongoing Improvement

Although the project has made good progress, much remains to be done to reach its full potential. In summary, improving user experience includes making the site easily navigable and user-friendly on all platforms, but especially on mobile. Protecting confidential user data will be another imperative task, and optimizing performance to ensure a fast system load. Reliable payment methods will be implemented to ensure smooth transactions. Having strong customer support is also important. We will implement customer support features so users will be able to seek assistance via live chat and email support systems.

### 7.2 Implementation Challenges

Here are a few challenges we faced during the implementation:

- Frontend and backend integration: There were issues with the front-end login page not working properly, even though the back-end login was working.
- Database Integration: While the backend successfully handles user authentication and login, these operations are not properly reflected in the front end.
- Visual code: Another challenge was that Visual Studio Code did not support JSX syntax properly. It took me hours to solve this problem.
- Responsive Design: Ensuring that the website was fully responsive across various devices and screen sizes was challenging. We had to frequently test and adjust the layout and components to provide a seamless experience on desktops, tablets, and mobile devices.
- State Management Complexity: Managing the global state with Redux posed difficulties, especially when handling asynchronous actions and ensuring state consistency across different components.

### **7.3 Deployment**

Deployment is a critical step in making a website accessible on the internet. For our project, we chose to deploy using Netlify, a platform known for its ease of use and robust features. Netlify offers two primary methods for deployment. Firstly, users can upload the project folder directly to Netlify via the dashboard, simplifying the process with a drag-and-drop interface. Alternatively, automated deployment with GitLab or GitHub integration is available, streamlining the deployment process and enabling seamless updates (Netlify n.d.).

## 8 DISCUSSION

This thesis has been dedicated to the development of an e-commerce website that focuses on providing fresh, unadulterated organic food in Bangladesh. The primary objective of this project is to address the critical issue of food adulteration by offering consumers a reliable source of high-quality organic products. To accomplish this, we have made use of modern web technologies such as React.js for the frontend, Node.js and Express.js for the backend, MongoDB for the database, Material UI for design, and Redux for state management. These technologies have played a key role in creating a robust, user-friendly, and efficient platform.

The e-commerce platform's development process was challenging but informative. Integrating various technologies required careful planning and execution to ensure seamless functionality. Initially, there were difficulties with user authentication processes, which were resolved through debugging and refining communication between the client and server, resulting in a smoother user experience. Integrating the database posed a major challenge. It was crucial to ensure that backend operations, like user authentication and login, were accurately reflected in the frontend. This required careful attention to detail and extensive testing. The process underscored the importance of having a cohesive development strategy that addresses both backend logic and frontend presentation.

The ethical considerations of this project are extremely important, given the delicate nature of food safety and consumer trust. Offering an untainted organic food source is not only a business endeavor but also a dedication to public health and well-being. The platform aims to address the widespread problem of food adulteration in Bangladesh by providing consumers with a reliable alternative. This commitment also encompasses data security, ensuring that user information is safeguarded by strong authentication mechanisms using JWT, thus ensuring the security of personal and payment data.

The platform's features, including user registration, login, product browsing, cart management, and secure checkout, establish a strong foundation for a complete e-commerce solution. Each feature has been carefully developed to guarantee

user-friendly, secure, and reliable functionality. The capability for users to leave reviews and comments, along with the administration's ability to manage these interactions, promotes a community-driven approach that encourages transparency and trust.

The project has made significant progress, but ongoing improvements are needed for future development. Priority areas include enhancing user experience, protecting user data, optimizing performance, and implementing robust customer support. This thesis addresses the issue of food adulteration in Bangladesh through innovative online technology, highlighting the importance of combining technical expertise with ethical responsibility.

## REFERENCES

- Anubhav, S. 2023. Express JS Tutorial. Published 10.05.2023. Read 9.05.2024.  
[https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js#what\\_is\\_express\\_js](https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js#what_is_express_js)
- Andrew, B. 2024. E-commerce Defined: Types, History, and Examples. Published 29.03.2024. Read 10.05.2024.  
<https://www.investopedia.com/terms/e/ecommerce.asp>
- Bob, K. 2024. Global e-commerce market to reach. Published 18.01.2024. Read 05.05.2024.  
<https://marketingreport.one/retail/global-e-commerce-market-to-reach-47.7-trillion-by-2030-report.html>
- Christiana, J. 2024. What Is E-Commerce? Definition, Types & Getting Started. Published 31.04.2024. Read 10.05.2024.  
<https://www.forbes.com/advisor/business/what-is-ecommerce/>
- GeeksforGeeks. 2024. Food Adulteration: Definition Published 12.01.2024. Read 06.05.2024.  
<https://www.geeksforgeeks.org/food-adulteration/>
- GeeksforGeeks. 2024. What is react-router-dom? Published 22.04.2024. Read 10.05.2024.  
<https://www.geeksforgeeks.org/what-is-react-router-dom/>
- Javatpoint. n.d. ReactJS Tutorial, React Redux. Read 15.05.2024.  
<https://www.javatpoint.com/react-redux>
- Joel, V. 2020. What is Netlify and What are its Benefits? Published 04.08.2020. Read 20.05. 2024.  
<https://agilitycms.com/resources/posts/what-is-netlify-and-why-should-you-care-as-an-editor>
- Kinsta. 2022. What Is Express.js Everything You Should Know. Published 19.04.2022 Read 11.05.2024.  
<https://kinsta.com/knowledgebase/what-is-express-js/>
- Kinsta. 2022. What Is npm? An Introduction to Node's Package Manager. Published 15.07.2022. Read 12.05.2024.  
<https://kinsta.com/knowledgebase/what-is-npm/#what-is-npm-node-package-manager>
- Komala, R. 2024. Food Adulteration: A Global Threat. Published 14.03.2024. Read 7.05.2024.  
<https://globalindiannetwork.com/food-adulteration-a-global-threat/#:~:text=The%20International%20Food%20Safety%20Network,challenge%20on%20a%20global%20scale.>
- Lim, D. 2020. What's an Example of a Good E-Commerce Database Design? Published 19.11.2020. Read 10.05.2024.



<https://fabric.inc/blog/commerce/ecommerce-database-design-example>

Md Muzakkerrul, H. 2023. Adulterated-food culture in Bangladesh: a new form of epidemic. Published 08.06.2023. Read 05.05.2024.

<https://www.dhakatribune.com/bangladesh/284997/adulterated-food-culture-in-bangladesh-a-new-form>

MongoDB. n.d. Release date of MongoDB. Read 12.05.2024.

<https://en.wikipedia.org/wiki/MongoDB>

Material UI. n.d. Intro, Installation, and Component Tutorials. Read 1.05.2024.

<https://react.school/material-ui>

National Library of Medicine. 2023. Mechanisms and Health Aspects of Food Adulteration: A Comprehensive Review. Published 02.01.2023. Read 05.05.2024.

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9818512/>

Postman. n.d. Send your first API request. Read 01.05.2024.

<https://learning.postman.com/docs/getting-started/first-steps/sending-the-first-request/>

Precision Reports. 2023. Worldwide Market Research Report, Analysis & Consulting. Published 12.10.2023. Read 07.05.2024.

<https://www.linkedin.com/pulse/fresh-food-e-commerce-market-foreseen-grow-impressive-0k9qf>

Pragim Technologies. n.d. React and NodeJS installation and setup. Read 05.05.2024.

<https://www.pragimtech.com/blog/reactjs/install-reactjs/>

React app. n.d. How to create a react application. Read 13.05.2023.

<https://create-react-app.dev/>

ReactJS. n.d. Read on 10.05.2024.

<https://legacy.reactjs.org/docs/getting-started.html>

Taha, S. 2023. What is Node.js? A Comprehensive Guide. Published 16.05.2023. Read 09.05.2024.

[https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs#what\\_is\\_nodejs](https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs#what_is_nodejs)