Online Grocery Website

Project report (CA3) submitted in fulfilment of the requirements for the Degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

RISHABH VARSHNEY

(12207817)

Roll No:-45

SUBJECT

INT219

School of Computer Science and Engineering

Lovely Professional University Phagwara, Punjab

(India)

TABLE OF CONTENTS

1. INTRODUCTION

1.1 PROBLEM STATEMENT

In today's fast-paced world, customers often encounter challenges securing their desired groceries through online platforms due to limited availability and inefficient ordering systems. This leads to frustration for both shoppers and website administrators, resulting in lost sales opportunities and unsatisfactory shopping experiences. To tackle this issue, there is a pressing need for a robust, user-friendly online grocery ordering system that harnesses modern technologies such as Node.js, Express.js, React.js, and MongoDB to streamline the shopping process for both customers and website owners.

Key problems to address:

- User-Friendly Interface: An intuitive web-based interface for customers to browse products, check availability, and place orders seamlessly.
- Real-Time Updates: Integration with a real-time database to provide instant updates on product availability, enabling customers to shop with confidence.
- Order Management: A dashboard for website administrators to efficiently manage orders, including processing, modifying, and canceling orders, as well as tracking customer preferences and special requests.
- Data Analytics: Advanced analytics capabilities to analyze order data, identify trends, and optimize inventory management and delivery operations to enhance overall efficiency of the online grocery website.

1.2 WHY USE EXPRESS?

The choice of Express for the project is driven by several key factors:

- Robustness: Express is a battle-tested web framework known for its reliability and performance, making it a solid choice for building web applications.
- Scalability: Express is designed to be lightweight and flexible, allowing it to scale effectively to handle increasing user demands and growing data.
- Active Developer Community: The Express community is large and active, providing continuous support, updates, and a wealth of resources for developers, ensuring the longevity and sustainability of the application.
- Rapid Development: Express's minimalist and unopinionated design allows developers to work quickly and efficiently, reducing development time and costs while maintaining code quality.

1.3 ARCHITECTURE

The restaurant reservation system follows the architectural pattern, comprising three interconnected components:

- Model: Manages data and database interactions, handling reservations, restaurant information, and user data. It encapsulates business logic related to reservation processing and data management.

- View: Responsible for the presentation layer and user interface, displaying information to users in a visually appealing and intuitive manner. It includes web pages, forms, and interfaces for browsing restaurants, making reservations, and viewing reservation details.

- Controller: Acts as an intermediary between the Model and View, processing user requests and orchestrating interactions between the two. It handles incoming requests, retrieves data from the Model, and renders the appropriate View to the user. Additionally, it validates user input, executes business logic, and updates the Model as need.

2. TECHNOLOGIES USED

2.1 NODE.JS

Node.js is a powerful JavaScript runtime environment built on Chrome's V8 JavaScript engine, designed for server-side applications. It offers several key features and characteristics:

- Asynchronous and Event-Driven: Node.js operates on a non-blocking, event-driven architecture, allowing for efficient handling of concurrent requests without blocking the execution thread.
- JavaScript Everywhere: Node.js enables developers to use JavaScript for both client-

side and server-side development, promoting code reusability and consistency across the stack.

- Scalability: Node.js is well-suited for building scalable applications, with built-in support for clustering and load balancing to handle large volumes of traffic.
- Extensive Package Ecosystem: Node.js boasts a vast ecosystem of npm (Node Package Manager) modules, offering ready-made solutions for various functionalities and integrations.
- Real-Time Communication: Node.js is ideal for real-time applications such as chat applications and streaming services, thanks to its ability to handle WebSocket connections and event-driven architecture.

2.2 EXPRESS.JS

Express.js is a minimalist web application framework for Node.js, renowned for its simplicity and flexibility. It provides a robust set of features for building web applications and APIs:

- Middleware: Express.js utilizes middleware functions to handle HTTP requests, enabling developers to add custom functionality such as authentication, logging, and error handling.
- Routing: Express.js offers a concise and expressive routing system, allowing developers to define routes for different HTTP methods and URL patterns easily.
- Templating Engines: Express.js supports various templating engines such as EJS and Pug, facilitating the dynamic generation of HTML content.
- Error Handling: Express.js provides built-in error handling mechanisms, allowing developers to handle errors gracefully and provide meaningful responses to clients.
- RESTful APIs: Express.js is commonly used for building RESTful APIs, thanks to its simplicity and support for handling HTTP requests and responses efficiently.

2.3 MONGODB

MongoDB is a popular NoSQL database management system known for its flexibility, scalability, and performance:

- Document-Oriented: MongoDB stores data in flexible, JSON-like documents, allowing for a dynamic and schema-less data model.
- Scalability: MongoDB is designed to scale horizontally across multiple servers,

making it suitable for handling large volumes of data and high-traffic applications.

• High Availability: MongoDB supports replication and automated failover, ensuring data availability and reliability in distributed environments.

• Indexing: MongoDB provides flexible indexing options to optimize query performance and enhance data retrieval speed.

• Geospatial Queries: MongoDB offers support for geospatial queries, allowing developers to perform location-based searches and analysis efficiently.

2.4 REACT.JS

React.js is a JavaScript library for building user interfaces, developed by Facebook. It offers several key features and characteristics:

• Component-Based Architecture: React.js promotes a component-based approach to UI development, allowing developers to build encapsulated and reusable UI components.

• Virtual DOM: React.js utilizes a virtual DOM for efficient rendering, minimizing DOM manipulation and improving performance.

• Declarative Syntax: React.js uses a declarative syntax to describe the UI state, making it easier to understand and maintain UI code.

• JSX: React.js introduces JSX, a syntax extension that allows developers to write HTML-like code within JavaScript, enhancing code readability and maintainability.

• Unidirectional Data Flow: React.js follows a unidirectional data flow architecture, ensuring predictable data updates and reducing bugs in complex UIs.

• **HTML**: HTML provides the structure for web pages, defining the elements that make up the content and layout of the website. It serves as the backbone of the web, forming the foundation upon which other technologies build.

• **CSS**: CSS is used for styling and presentation, allowing developers to customize the appearance of HTML elements, such as colors, fonts, and layouts. It helps maintain consistency across web pages and ensures a visually appealing and user-friendly interface. CSS also facilitates the creation of responsive designs, enabling web pages to adapt seamlessly to different screen sizes and devices. Its ability to target specific elements and apply styles selectively enhances the accessibility and usability of websites. Additionally, CSS preprocessors like Sass and Less extend CSS's capabilities by introducing features such as variables, mixins, and nesting.

• JavaScript:

JavaScript is a scripting language used to add interactivity and dynamic behavior to web pages. It enables features like form validation, animations, and interactive elements. JavaScript is a versatile scripting language used extensively in web development to create interactive and dynamic user experiences. It runs client-side in web browsers, empowering developers to manipulate webpage elements, respond to user actions in real-time, and fetch data from servers asynchronously. JavaScript frameworks and libraries, such as React.js, AngularJS, and Vue.js, extend its capabilities, facilitating the development of complex web applications. Its event-driven nature enables functionalities like event handling, DOM manipulation, and AJAX requests, enhancing the overall interactivity and responsiveness of web pages. JavaScript's widespread adoption and continual evolution contribute to its status as a cornerstone technology in modern web development.

• Bootstrap:

Bootstrap is a popular CSS framework that provides pre-designed components and styles for building responsive and mobile-friendly web interfaces. It offers a grid system, typography, forms, buttons, and other UI components, helping developers create consistent and visually appealing designs efficiently.

Just like React.js, these technologies play crucial roles in developing the frontend of the online grocery website, offering features like structured layout, styling, interactivity, and responsiveness.

Bootstrap revolutionized frontend development by offering a comprehensive set of tools and components that simplify the creation of modern web interfaces. Its grid system provides a flexible and responsive layout structure, ensuring compatibility across various devices and screen sizes. Additionally, Bootstrap's extensive collection of CSS classes and utility functions streamline styling and customization, empowering developers to achieve consistent design patterns with minimal effort. The framework's built-in components, such as navigation bars, carousels, and modals, expedite the development process by providing ready-to-use solutions for common UI elements. Furthermore, Bootstrap's focus on mobile-first design principles enhances accessibility and usability, making it an indispensable tool for building user-friendly and visually appealing websites.

3. KEY FEATURES OF THE WEBSITE

The online grocery ordering system is meticulously crafted to offer an array of features, ensuring a seamless and enjoyable shopping experience for users. These features are designed to enhance accessibility, streamline the ordering process, and foster collaboration between shoppers and website administrators. Here are the core features:

User Authentication: Allow users to create accounts or log in using their credentials,

providing a personalized experience and ensuring data security.

• Product Catalog: Display the online grocery store's inventory in an intuitive and visually appealing format, allowing users to browse available products with ease.

• Order Booking: Enable users to place orders for groceries by selecting their desired items, quantities, and delivery or pickup options. Users can add their details, such as name, contact information, and any special delivery instructions.

• Order Management: Provide website administrators with a dashboard to manage orders efficiently, including viewing, editing, and confirming purchases. This ensures optimal inventory management and minimizes delivery or pickup wait times for shoppers.

• Real-Time Inventory Updates: Implement real-time updates on product availability, allowing users to make informed decisions when placing orders. This enhances user confidence and reduces the likelihood of ordering out-of-stock items.

• Order Confirmation Notifications: Send confirmation notifications to users upon successful order placement, providing reassurance and eliminating ambiguity regarding their purchase.

• User Account: Offer users a profile page where they can view their order history, update personal information, and manage preferences such as delivery address and payment methods.

• Seamless Communication: Facilitate communication between users and store staff through integrated messaging or chat features. This allows for smooth coordination and resolves any queries or concerns promptly.

• Feedback Mechanism: Incorporate a feedback mechanism where users can provide ratings and reviews based on their shopping experience. This valuable input helps the online grocery store improve service quality and customer satisfaction.

• Responsive Design: Develop the application with a responsive design to ensure compatibility across various devices, enabling users to make reservations conveniently from desktops, tablets, or smartphones.

• Accessibility Features: Implement accessibility features to accommodate users with disabilities, ensuring inclusivity and equal access to reservation services.

These features collectively create a user-centric online grocery ordering system, catering to the needs of both shoppers and website administrators. By leveraging technology and user-centered design principles, the system aims to streamline the ordering process, optimize inventory management, and enhance overall shopping experiences for users.

4. MODULES

Project Structure:

Controllers:

- Under the 'backend>Controller' directory, the 'order.js' file manages order-related functionalities. It handles order requests, interacts with the database, and orchestrates the order processes.

Database Connection:

- The 'dbConnection.js' file, located in the 'database' directory, establishes and maintains a connection to the database (e.g., MongoDB). It provides methods for executing database queries and managing data interactions.

Error Handling:

- Within the 'error' directory, the 'error.js' module handles error scenarios gracefully. It encapsulates exception handling, validation error responses, and other error-related functionalities to ensure robust error management.

Models:

- Under the 'models' directory, the 'orderSchema.js' file defines the schema for order data. It specifies the structure of order records, including fields such as order ID, user details, order date, and product information.

Routes:

In the 'routes' directory, the 'orderRoutes.js' file defines routes for order-related endpoints. It maps HTTP requests to corresponding controller methods, facilitating request handling and routing for order functionalities.

server.js:

- The server.js' file serves as the entry point for our Node.js

application. It initializes the server, sets up middleware, defines routes, and handles HTTP requests and responses. Additionally, it establishes the connection with the database and includes error handling middleware to manage any runtime errors gracefully. This file plays a crucial role in orchestrating the backend operations of our restaurant reservation system.

index.js:

-The index.js file is responsible for starting the server by importing the app instance from the app.js file and listening on the specified port. Once the server is started, it logs a message to the console indicating that the server has started successfully along with the port number it's listening on.

This structured approach ensures a clear organization of components within the online grocery ordering system. Controllers handle request processing, models define data structures, routes manage endpoint mappings, and frontend scripts and views facilitate user interactions. Additionally, error handling functionalities are integrated to ensure robustness and reliability in managing error scenarios.

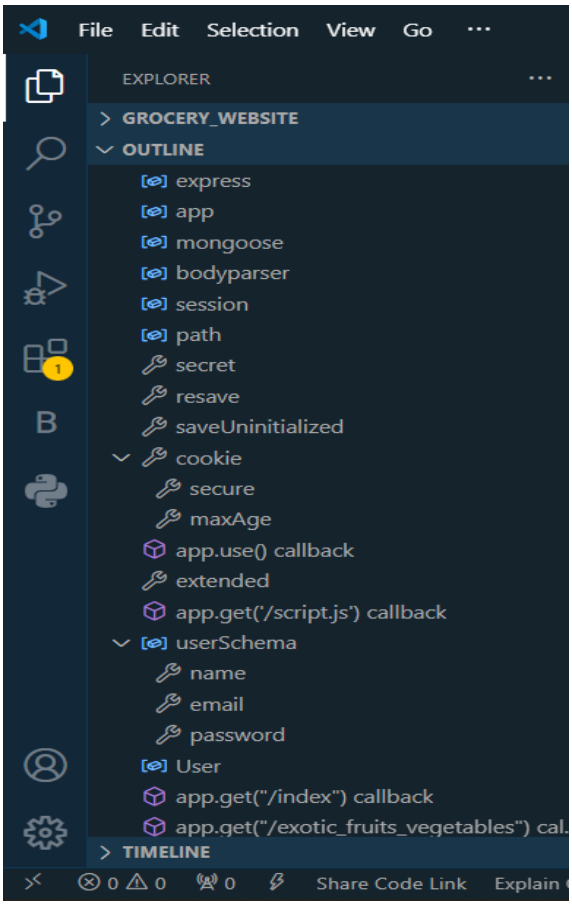
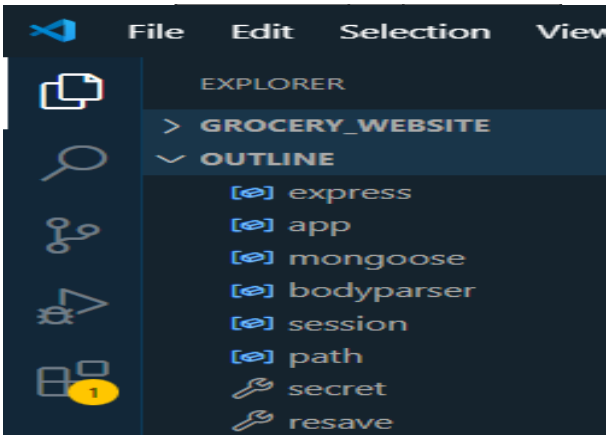
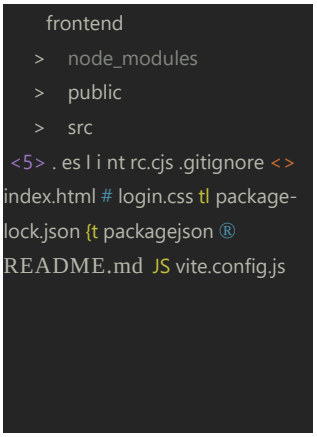


Figure : Project directory structure as seen in VS Code editor





4.1 EXPRESS SERVER CONFIGURATION

```
import express from "express";
importdotenv from "dotenv";
import corsfrom "cors";
import {dbConnection} from "./database/dbConnection.js";
import {errorMiddleware} from "./error/error.js"; import
reservationRouterfrom "./routes/reservationRoute.js";



const app = express();
app.use(cors({credentials: true, origin: true}));
dotenv.config({ path: "./config/config.env"});


app.use(
    cors({
      origin: [process. env.FRONTENDURL],
      methods: ["POST"], credentials: true,
    })
 );
 app. use (express.json ());

  app.use(express.urlencoded({extended: true})); app.
  use("/api/v1/reservation ", reservationRouter);

 dbConnection(); app. use(errorMiddleware); export
default app;
```

Explanation: This module configures an Express application to manage HTTP requests for the restaurant reservation system. It sets up middleware, routes, and database connections to facilitate reservation functionalities while ensuring robust error handling.

4.2 MONGODB DATABASE CONNECTION

```
import mongoose from "mongoose ";


 export const dbConnection = () => {
    mongoose
      .connect(process.env.MONGO URI, {
        dbName: "ORDERMANAGEMENT",
      })
      .then(() => {
        console.log("Connected to database!");
      })
      .catch((err) => {
        console.log(Some error occured while connecing to database: ${err}'); });
```

Explanation:

- This module establishes a connection to the MongoDB database using Mongoose, a MongoDB object modeling tool designed for Nodejs.

- The dbConnection function connects to the MongoDB database specified by the MONGO_URI environment variable. If the connection is successful, it logs a message confirming the connection. If there is an error during the connection process, it logs an error message detailing the issue encountered.

4.3 MANAGEMENT SCHEMA

```
import mongoose from "mongoose";
import validator from "validator";

const reservationSchema = new mongoose.Schema({
firstName: { type: String, required: true,
    minLength: [3, "First name must be of at least 3 Characters."],
  maxLength: [30, "First name cannot exceed 30 Characters."], },
  lastName: {
```

```
  type: String, required: true,
  minLength: [3, "Last name must be of at least 3 Characters. "],
  maxLength: [30, "Last name cannot exceed 30 Characters."],
 },

 email: { type:
  String,
  required: true,
  validate: [validator.isEmail, "Provide a valid email"],
 },
 date: { type:
  String,
  required: true,
 },
 time: { type:
  String,
  required: true,
 },
 phone: { type:
  String,
  required: true,
  minLength: [10, "Phone number must contain 10 Digits."],
  maxLength: [10, "Phone number must contain 10 Digits."],
 },
});

export const Reservation = mongoose.model("Reservation", reservationSchema);
```

Explanation:

- The This module defines the schema for a reservation using Mongoose, a MongoDB object modeling tool for Nodejs.
- The reservationSchema specifies the structure of a reservation record with the following fields:

- firstName: String type, required field with minimum and maximum length constraints for the first name.
- lastName: String type, required field with minimum and maximum length constraints for the last name.
- email: String type, required field with email validation using the validator library.
- date: String type, required field representing the reservation date.

- time: String type, required field representing the reservation time.
- phone: String type, required field representing the phone number with length constraints.
- The Reservation model is created using mongoose.model(), which defines a MongoDB collection named "Reservation" based on the specified schema.

4.4 PRODUCT ORDER MANAGEMENT HANDLER

```
importErrorHandlerfrom "../error/error.js";
import {Reservation} from "../models/reservationSchemajs";


export const sendReservation = async (req, res, next) => { const
{firstName,lastName,email,date,time,phone} = req.body; if (!firstName ||
!lastName || !email || !date || !time || !phone) { return next(new
ErrorHandler("Please Fill Full Reservation Form!", 400));
}

   try {
    await Reservation. create ({firstName, lastName, email, date, time,phone }) ;
    res.status(200).json({ success: true,
      message: "Reservation Sent Successfully!",
    });
} catch (error) {
    // Handle Mongoose validation errors if (error.name === 'ValidationError') { const
    validationErrors = Object.values(error.errors).map(err => err.message); return
    next(new ErrorHandler(validationErrors.join(', '), 400));
    }



    // Handle other errors
    return next(error);
}
};


export default sendReservation;
```

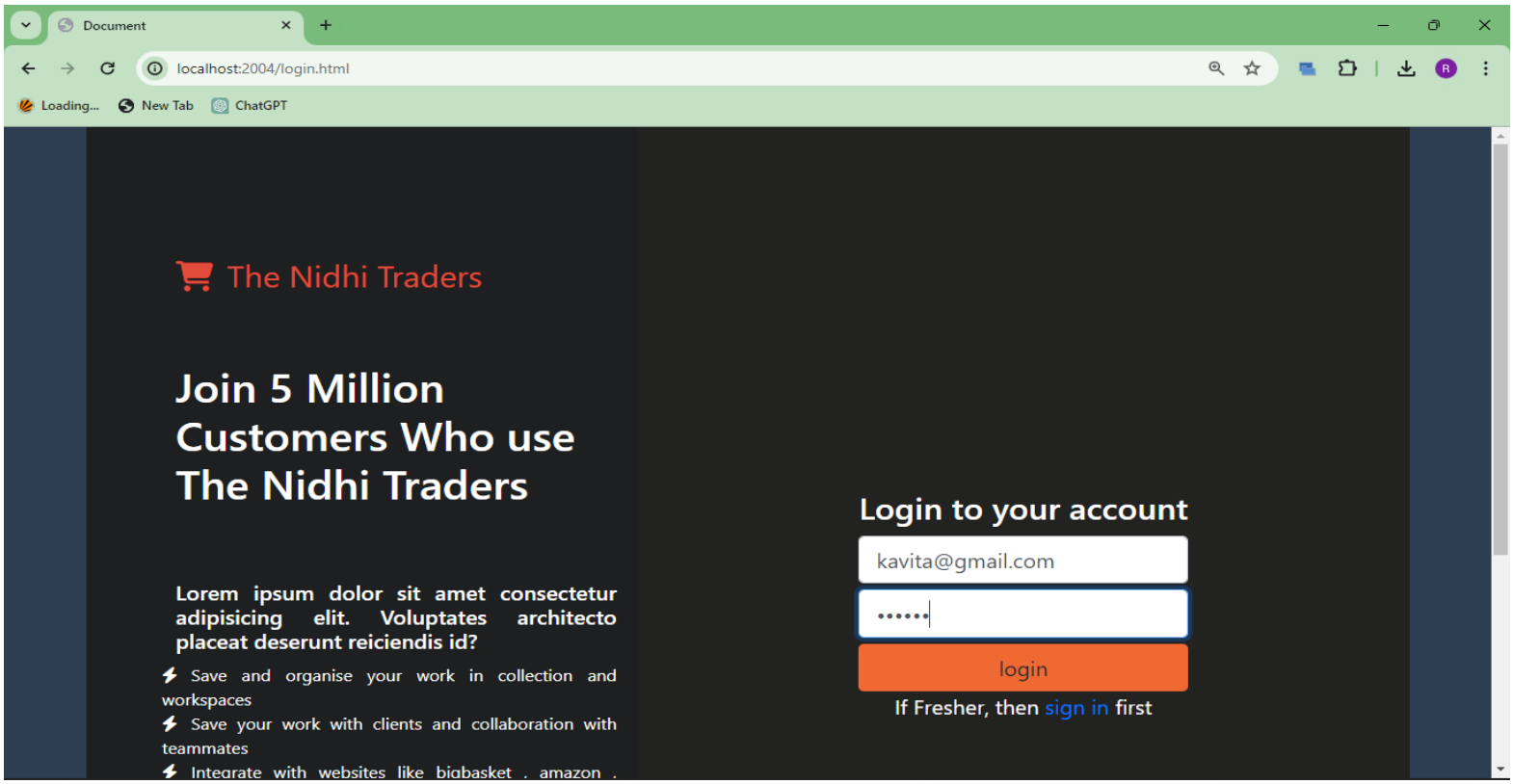
Explanation:

- The 'sendReservation' module handles reservation submissions. It

extracts reservation details like firstName, lastName, email, date, time, and phone from the request body. If any required field is missing, it triggers an error using the ErrorHandler class, forwarding it with a status code of 400 (Bad Request).

- Upon ensuring all required fields are present, it tries to create a new reservation in the database using Reservation.create(). If successful, it sends a response with status code 200 and a success message JSON object.In case of validation errors during reservation creation (e.g., invalid email format), it formats and sends the validation error messages to the next middleware with a status code of 400.
- Unexpected errors are passed to the next middleware function for centralized errorN handling.

5. WEBSITE SNAPSHOTS



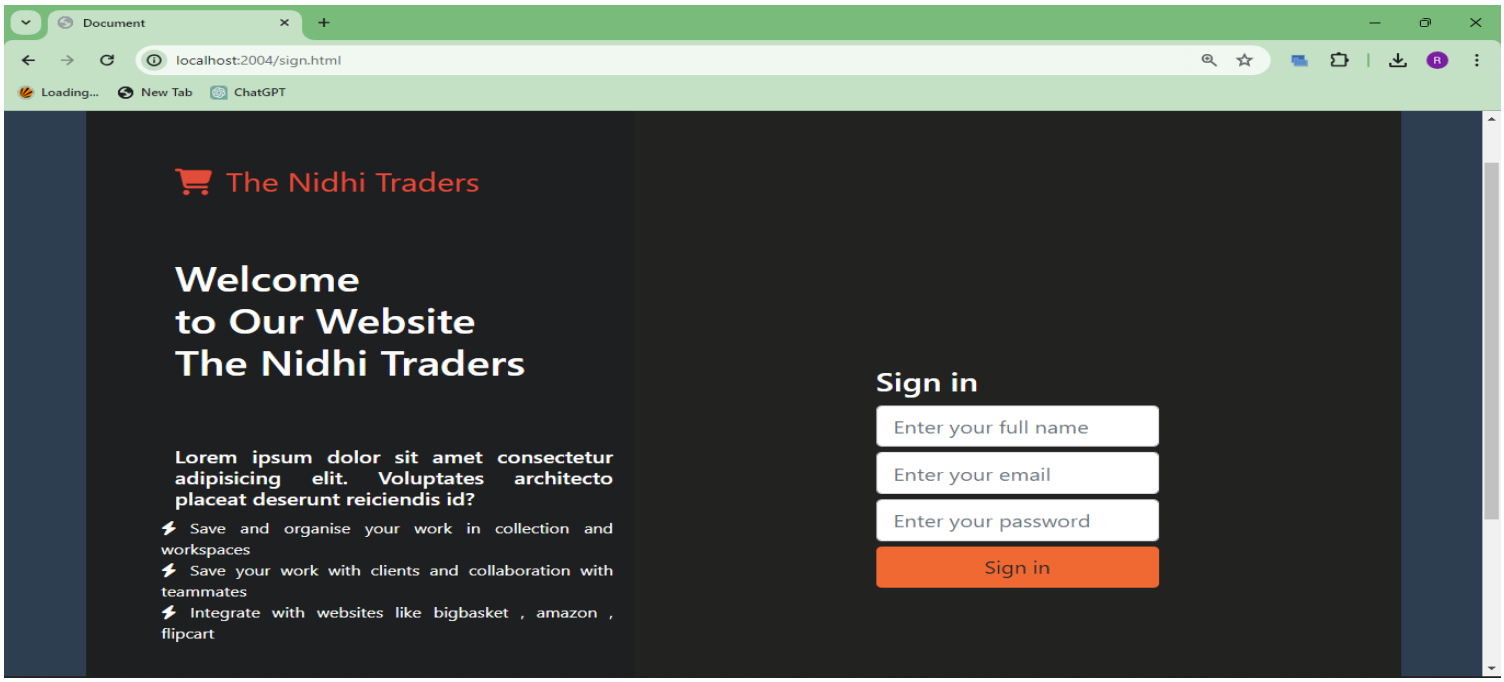


Figure 4: Registration Page

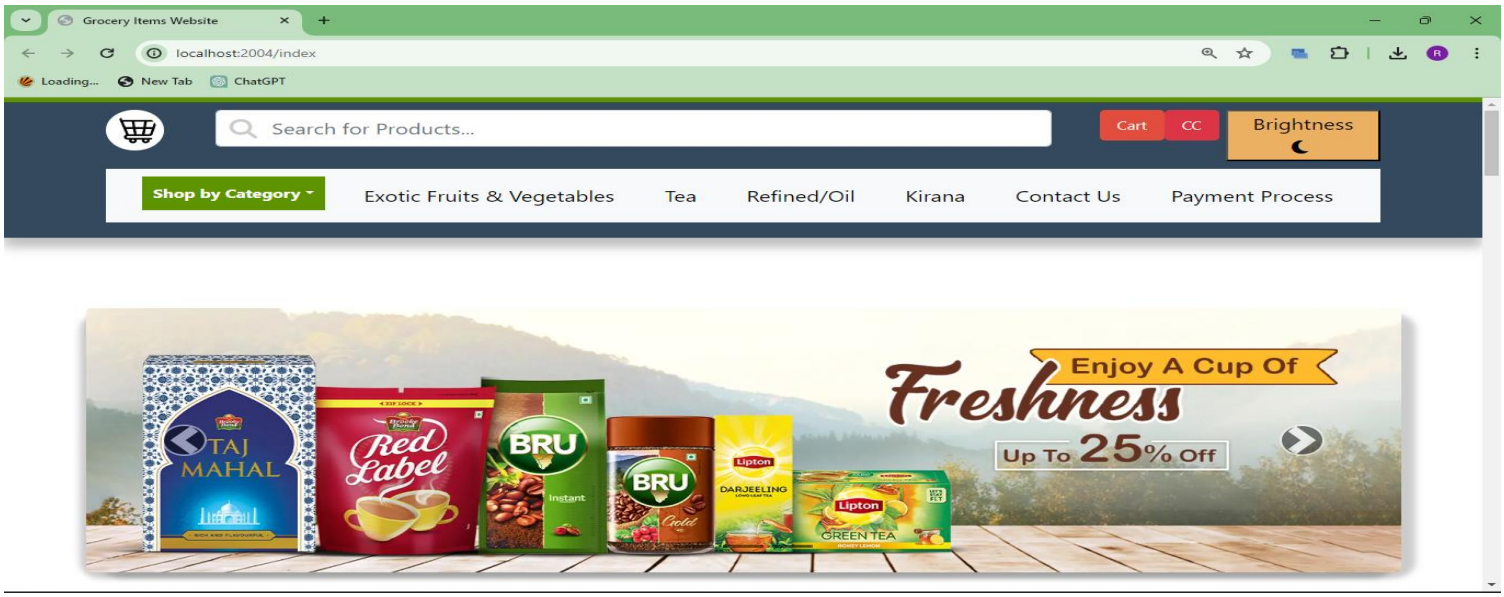
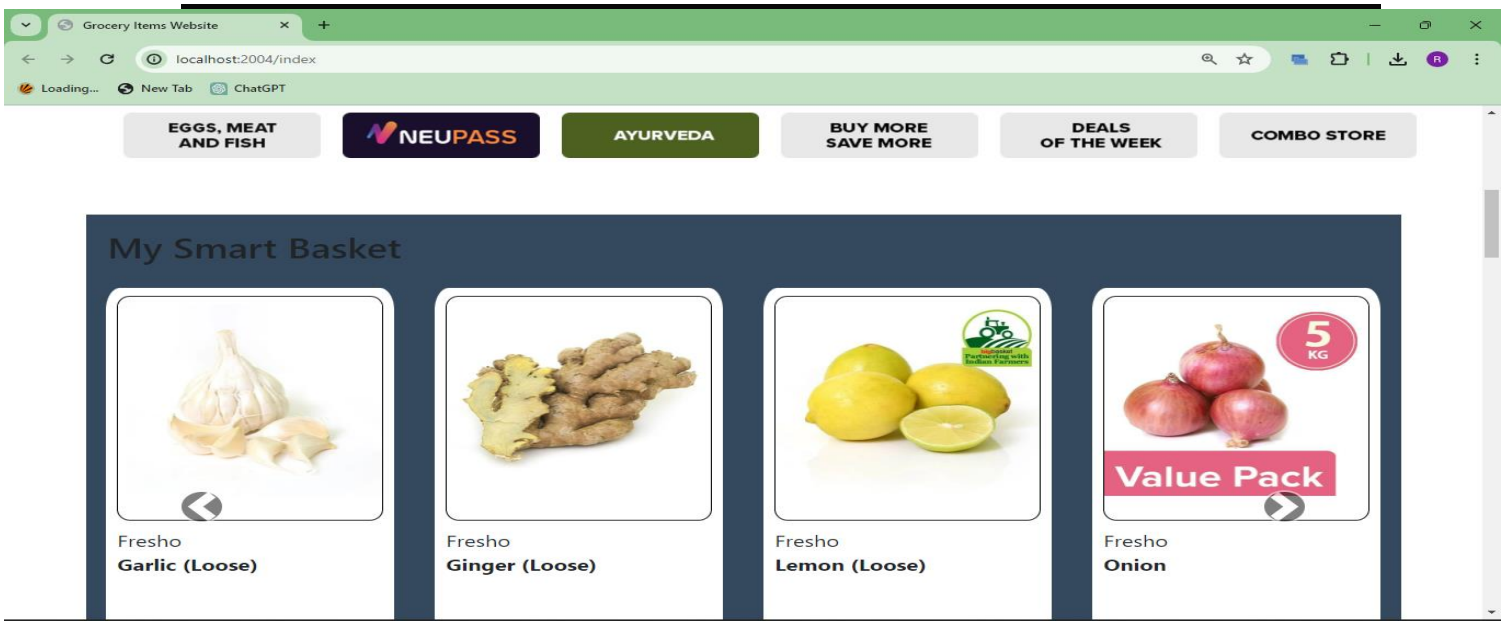


Figure 5: Home page



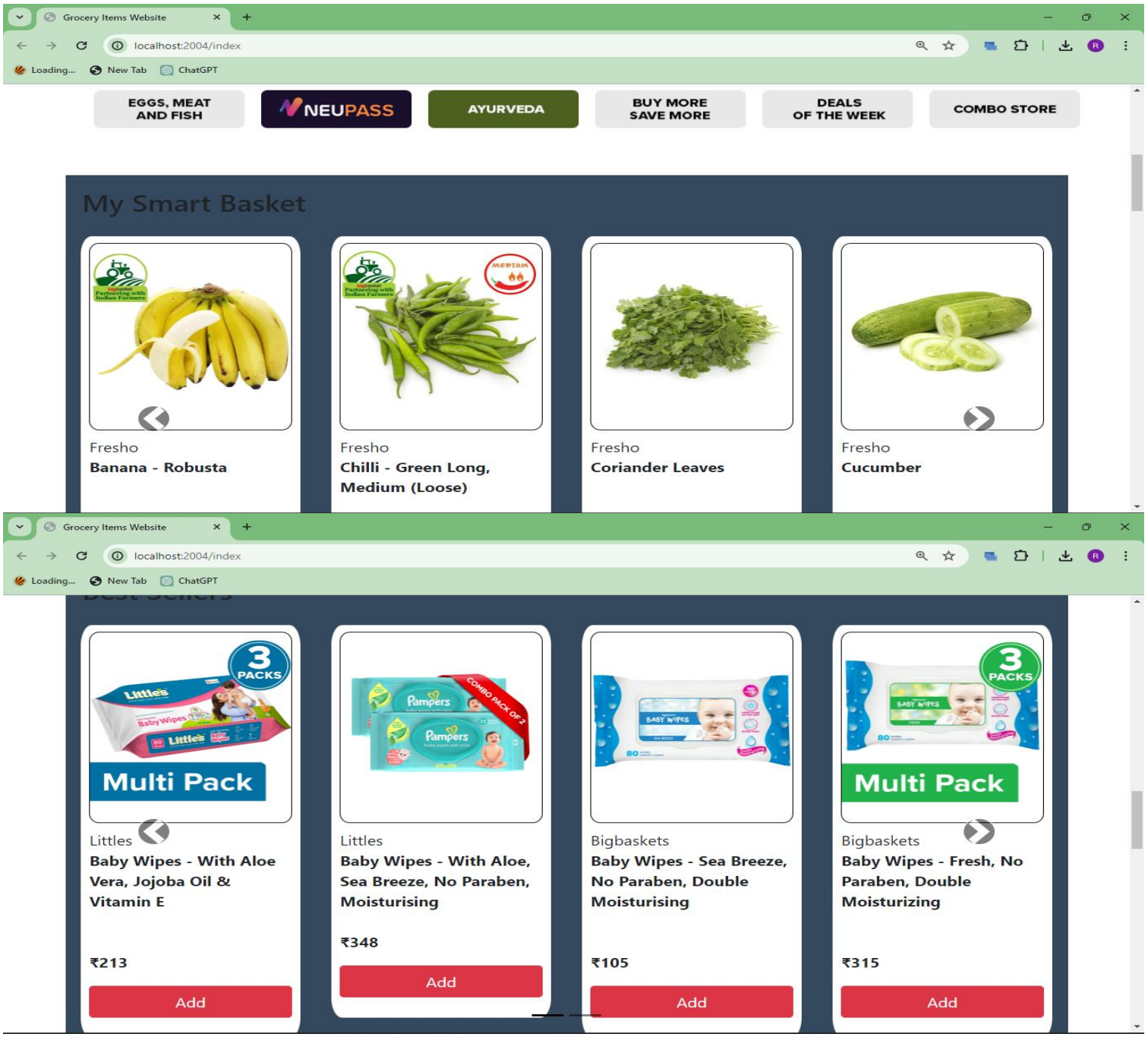


Figure 6: Menu page

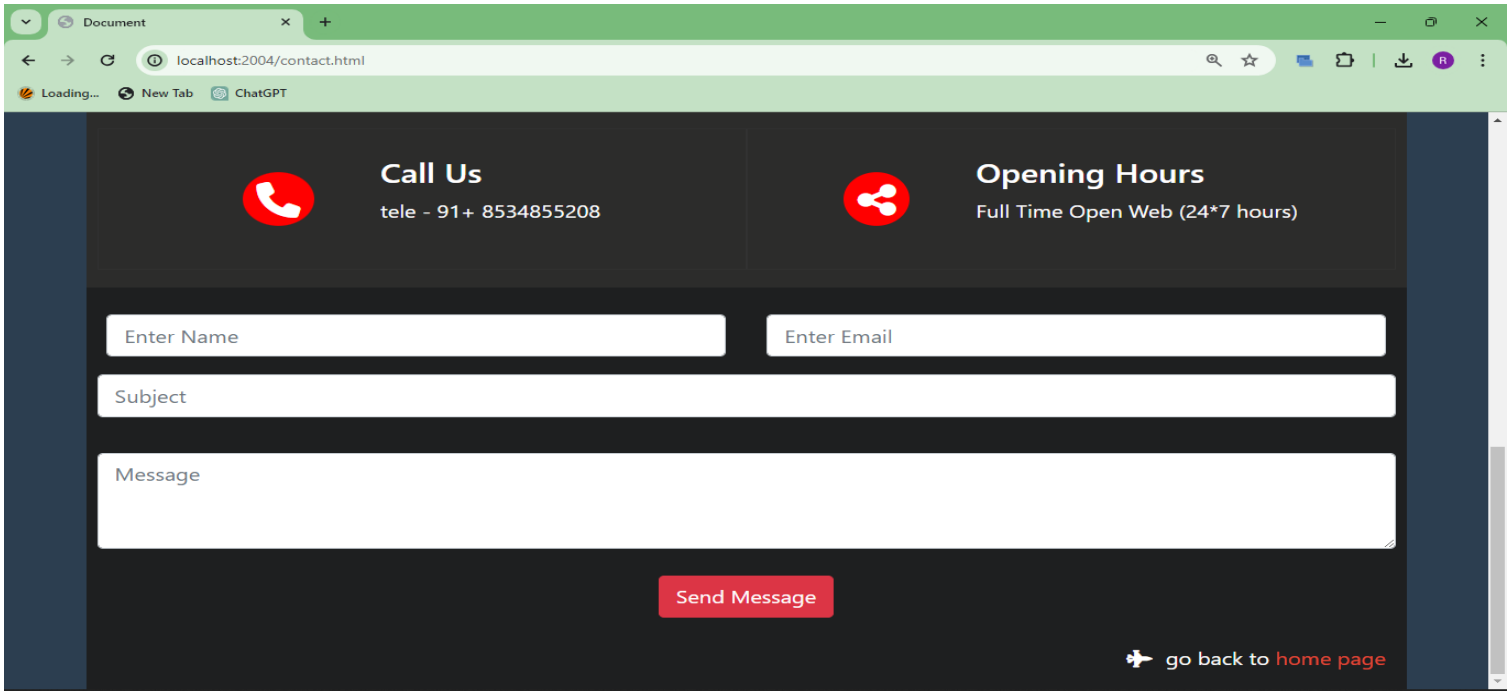


Figure 7: Service page

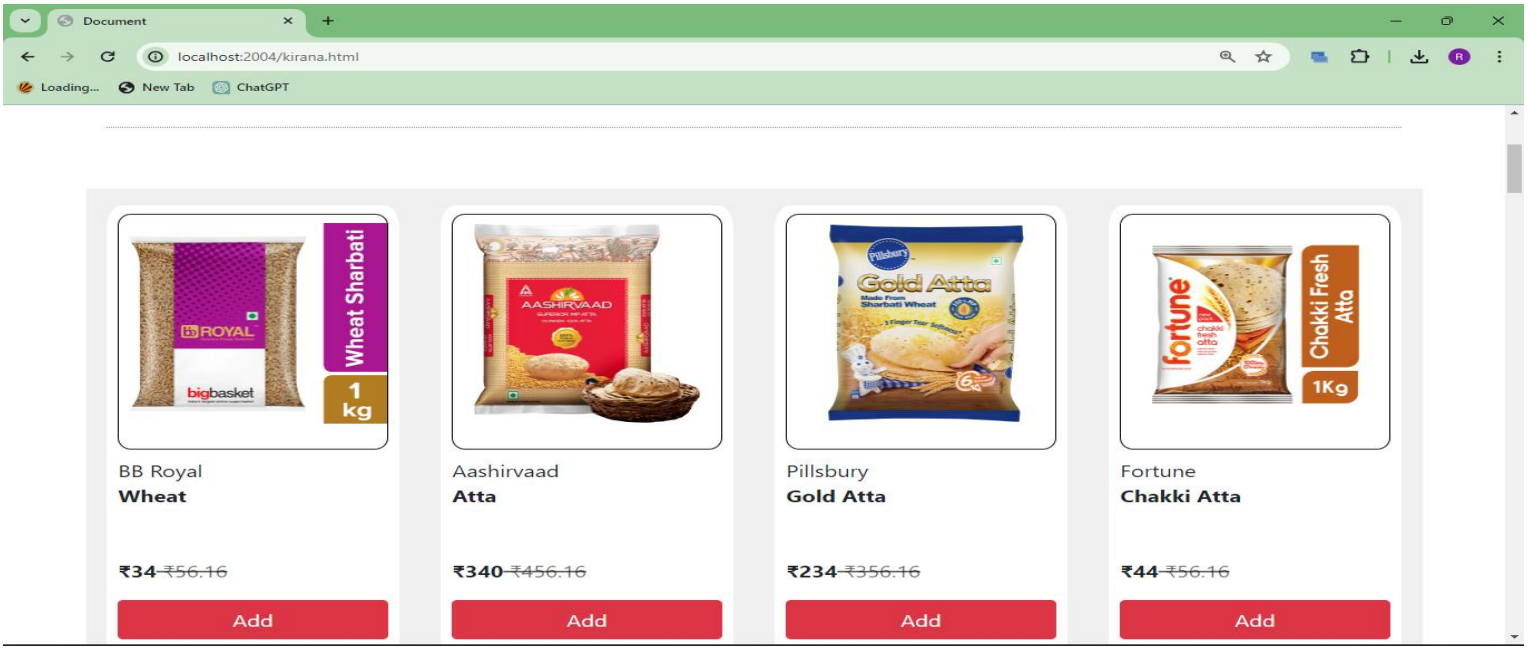
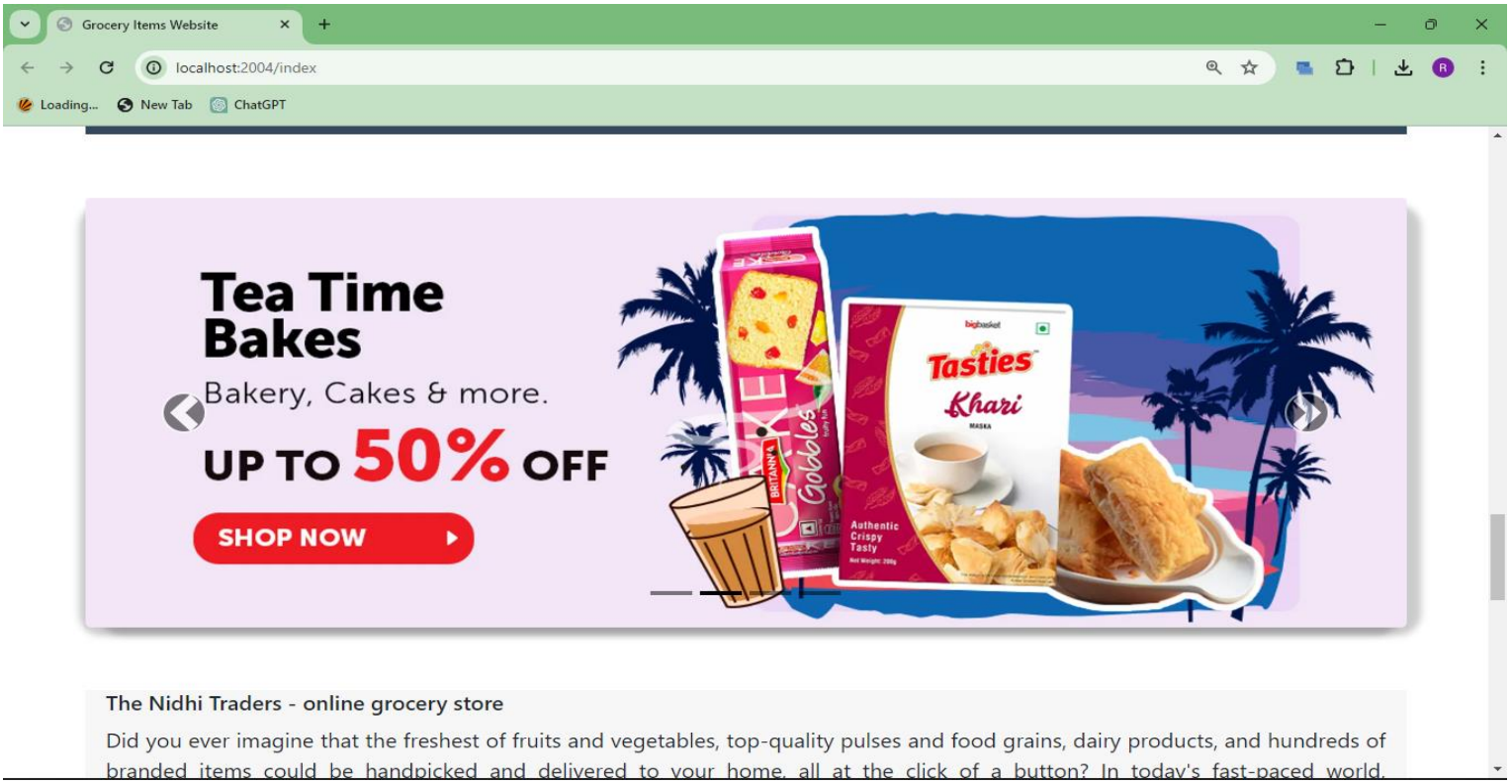


Figure 8: Our Team page



6. GITHUB LINK

- The link to the GitHub repository

https://github.com/RishabhVarshney34/Online-Grocery-Website-

7. LIST OF REFERENCES

- • Node.js Documentation: https://nodejs.org/en/docs/
- • Express.js Documentation: https://expressjs.com/
- • MongoDB Documentation: https://docs.mongodb.com/
- • React.js Documentation: https://reactjs.org/docs/getting-started.html