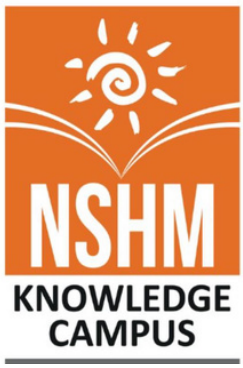
# Food Delivery Website




NSHM COLLEGE OF
MANAGEMENT AND TECHNOLOGY

Mentor : Prof. Sanjay Kumar Pal

| | |
|---|---|
| Name | Mohsin Ansari |
| Course | BCA (6th SEM) |
| Roll No. | 23401221027 |
| Paper | Major Project |
| Paper Code | BCAD681 |

# RealTime Food Delivery Web App

# ACKNOWLEDGEMENT

I would like to express my gratitude to all the individuals who have contributed directly or indirectly to the success of my major project, a real-time food delivery website.

Mentors and Advisors: A heartfelt thank you to Prof. Sanjay Pal for guiding us throughout the development process. Your advice and support have been crucial to the project's success.

Stripe: A special thanks to Stripe for providing the seamless payment processing capabilities that are essential to the functionality of this system.

Open Source Community: Thanks to the entire open-source community for creating and maintaining the various libraries and tools that were instrumental in building this project, including Express.js, Socket.io, Mongoose, and many more.

Contributors: I appreciate the contributions from anyone who has provided feedback, reported issues, or contributed code to this project. Your input has been invaluable.

Testers: Last but not least, thank you to all the users who have tested and used this food delivery website, providing valuable insights and suggestions for improvement.

Team member’s Signature

Mohsin Ansari

# CERTIFICATE

This certificate is to acknowledge that the major project entitled "Food Delivery Website" has been completed by Mohsin Ansari under my guidance. The work presented in this project is original and has not been submitted elsewhere for any purpose. The successful completion of this project reflects the dedication and hard work put forth by Mohsin Ansari.

Professor Sanjay Kumar Pal
[ NSHM Knowledge Campus ]

Date:

# Index

# 1. Introduction

This project is a comprehensive Food Delivery Web Application designed to provide a seamless experience for customers to order food online and for administrators to manage the food menu and orders efficiently. The application incorporates various modern web technologies and best practices to ensure a robust, scalable, and user-friendly platform. It supports real-time updates for order tracking and uses secure payment gateways for transactions. Additionally, it includes an admin panel for managing the food menu, orders, and overall operations.

## Features :

**Authentication and Authorization:**

- Secure sign-up and login using passport.js
- Passwords are hashed for security using Bcrypt.

**User Friendly Design:**

- Compatible with desktops, tablets, and smartphones.
- Minimal User Interface
- Easy to navigate

**User Order Management:**

- Browse Food Items
- Searching & Filtering
- Add To Cart
- Place Order
- Track Order Status

**Admin Order Management:**

- Login as Admin
- View user orders
- Update order status

**Multiple Payment Methods:**

- Stripe payment gateway for secure transactions.
- Cash On Delivery

# 2. Objectives of The Project

The primary objectives of this Food Delivery Web Application are:

- **Provide a Seamless User Experience:**
  - Design an intuitive and user-friendly interface.
  - Ensure responsive design for optimal usability across various devices.
  - Facilitate easy navigation through the menu and ordering process.
- **Secure User Authentication:**
  - Implement robust registration and login processes using Passport.js.
  - Ensure user data protection through hashing and secure session management.
- **Efficient Order Management:**
  - Enable users to place orders easily and track their status in real-time.
  - Provide admins with tools to manage and update order statuses efficiently.
- **Comprehensive Menu Management:**
  - Allow admins to add, update, and delete food items seamlessly.
  - Support image uploads and categorization for better menu organization.
- **Enable Secure Payments:**
  - Integrate Stripe for secure and reliable payment processing.
  - Ensure that all transactions are encrypted and user payment data is protected.
- **Real-time Communication:**
  - Use Socket.io for real-time updates on order statuses.
  - Ensure users receive immediate notifications about their order progress.
- **Maintain Data Integrity and Security:**
  - Use MongoDB for secure and scalable data storage.
  - Implement data validation and error handling to maintain database integrity.
- **Enhance Administrative Control:**
  - Provide an admin dashboard for overseeing platform operations.
  - Enable order and menu management with minimal administrative effort.
- **Scalability and Performance:**
  - Design the application to handle growing user and order volumes.
  - Optimize database queries and server responses for fast performance.

# 3. Software Requirement Specifications

**Purpose**

The purpose of this Software Requirements Specification (SRS) document is to define the requirements for the Food Delivery Web Application. This document provides a detailed description of the system's functionality, features, and constraints to ensure a clear understanding among stakeholders and developers.

**Scope**

This project involves developing a comprehensive Food Delivery Web Application that allows users to order food from various restaurants and vendors. The application will include features such as user registration, menu browsing, order placement, real-time order tracking, and an admin panel for managing food items and orders. Future enhancements will include migrating to Next.js to support multiple stores and shops.

**Product Perspective**

A standalone system aimed at competing with other food delivery platforms, initially supporting single-vendor operations, with plans for multi-vendor support.

**Product Functions**

- User registration and login
- Menu browsing, item search, item filtering
- Order placement and payment processing
- Real-time order tracking
- Admin panel for managing items and orders

# 4. Technical Specifications

## i. Visual Studio Code

- Visual Studio Code (VS Code) is a lightweight, open-source code editor developed by Microsoft. It is known for its flexibility, extensive language support, and a rich set of features that enhance the development experience.
- Its support for various programming languages, extensions, and built-in Git integration makes it a versatile choice for web development projects.
- VS Code provides a clean and user-friendly interface, allowing developers to focus on code without distractions.
- The editor supports a vast array of extensions that can be installed to enhance functionality, ranging from language support to debugging tools.
- VS Code includes a built-in terminal, enabling developers to run commands, scripts, and interact with the development environment without leaving the editor.
- Git integration is seamlessly integrated into VS Code, allowing for version control operations, commit history visualization, and branch management.
- VS Code features intelligent code completion, syntax highlighting, and error checking, providing a smooth coding experience.
- The built-in debugger simplifies the process of identifying and fixing issues in the code by providing step-through and breakpoint functionality.

Visual Studio Code

## ii. EJS (Embedded Javascript)

- EJS is a simple templating language that lets you generate HTML markup with plain JavaScript. It provides a straightforward way to embed dynamic content and logic directly into HTML files.
- **Partial Views:** EJS supports the inclusion of partial views, allowing the reuse of common HTML components across multiple pages.
- EJS is the chosen template engine for the Attendance System, allowing the server to generate dynamic HTML content based on data from the MongoDB database.


## iii. CSS + Tailwind CSS

- **Vanilla CSS:** Vanilla CSS refers to traditional CSS, written directly by developers without the use of any frameworks or libraries. It allows for full control over styling by targeting specific HTML elements and applying custom styles.
- **Tailwind CSS:** Tailwind CSS is a utility-first CSS framework that provides a set of pre-built utility classes for styling HTML elements. Instead of writing custom CSS, developers can use these utility classes to apply styles directly in the HTML markup.
- **In this project,** Vanilla CSS handles complex styling needs, offering precise control over design elements. Meanwhile, Tailwind CSS speeds up development by providing ready-to-use styles for common components, ensuring consistency across the interface. This blend allows for both detailed customization and efficient styling, enhancing the project's overall design and development process.

## iv. Node.js

- Node.js is an open-source, cross-platform JavaScript runtime environment built on the V8 JavaScript engine. It allows developers to execute JavaScript code server-side, enabling the development of scalable and high-performance network applications.
- The ability to use JavaScript for both server-side and client-side development streamlines the development process and promotes code reuse.
- Node.js is built on the V8 JavaScript engine, known for its high-performance execution of JavaScript code.
- npm, the Node.js package manager, provides access to a vast ecosystem of open-source libraries and tools that can be easily integrated into Node.js projects.

## v. Express.js

- Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features to develop web and mobile applications.
- It facilitates the creation of server-side logic and simplifies the handling of HTTP requests and responses.
- Express.js serves as the primary web application framework for the Attendance System, enabling the definition of routes, middleware, and the overall structure of the server-side code.
- **Routing:** Express.js allows the definition of routes to handle different HTTP methods and URL patterns, making it easy to structure the application's endpoints.
- **Middleware:** Middleware functions in Express.js provide a way to execute code during the request-response cycle. This is used for tasks such as logging, authentication, and error handling.
- **Template Engine Support:** Express.js supports various template engines like EJS, allowing the dynamic generation of HTML content on the server.

## vi. MongoDB

- MongoDB is a NoSQL database that provides a flexible, schema-less data model, making it suitable for storing and managing large volumes of unstructured data.
- **Document-Oriented**: MongoDB stores data in JSON-like Collections & documents, allowing for a flexible and dynamic schema.
- **Scalability:** MongoDB is designed to scale horizontally, making it suitable for handling large datasets and high read and write loads.
- **Query Language:** MongoDB supports a rich query language, making it easy to retrieve and manipulate data.
- **Usage:** User details, items, sessions and other relevant data are stored in MongoDB collections.

## vii. Mongoose

- Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It provides a schema-based solution to model the application data and simplifies interactions with MongoDB databases.
- **Schema Definition:** Mongoose enables the definition of schemas with typed fields, validation rules, and default values.
- **Query Building:** Mongoose provides a query builder API, making it easier to construct complex MongoDB queries using a fluent syntax.
- Usage

1. Mongoose is initialized with a MongoDB connection URI and is used to define models based on schemas.
2. Models are employed to interact with MongoDB collections, providing methods for CRUD operations.
3. Mongoose is used in the project to define schemas for the user , food items and allowing for structured data storage and retrieval.

- **Schema (Structure):** Users schema which contains credentials like username, email, password.


## viii. Passport JS

- Passport.js is a popular authentication middleware for Node.js applications. Its primary purpose in this project is to handle user authentication and session management. Here's how Passport.js is utilized in the Food Delivery Web Application:
- **User Authentication**: Passport.js streamlines user authentication by offering support for multiple authentication methods, including username/password, OAuth, and JSON Web Tokens (JWT). This flexibility enables developers to choose the most suitable authentication strategy for their project's needs.
- **Session Management**: Passport.js seamlessly integrates with Express.js to provide built-in session management capabilities. This allows users to maintain their authentication status across different requests without the need to re-enter their credentials repeatedly. As a result, users can navigate through the application smoothly while staying securely authenticated.

## ix. Bcrypt

- Bcrypt is a Node.js module for password hashing and salting.
- **Secure Password Storage:** Bcrypt is employed to hash user passwords securely before storing them, enhancing protection against unauthorized access to user accounts.
- **Salted Hashing:** Bcrypt utilizes a salted hashing technique, appending a unique random string (salt) to each password before hashing. This approach fortifies security by preventing the use of precomputed hash tables for password decryption.
- **Authentication Integration:** Bcrypt is seamlessly integrated into the user authentication process, ensuring that passwords are securely hashed and verified during login attempts, thus safeguarding user credentials from potential attacks.


# x. Stripe Payment

- Stripe is a widely-used payment processing platform that facilitates secure online transactions. In this project, Stripe is integrated to handle payment transactions, allowing users to make purchases securely through the application.
- **Easy Integration:** Stripe offers straightforward integration with web applications, providing developers with robust APIs and SDKs. In this project, Stripe's APIs are utilized to seamlessly integrate payment processing functionality, enabling users to complete transactions without friction.
- **Secure Transactions**: With built-in security features such as PCI compliance and advanced fraud detection, Stripe ensures that payment transactions are secure and protected. By leveraging Stripe's secure infrastructure, this project ensures that users' payment information is handled safely during the checkout process.
- **Customizable Checkout**: Stripe provides customizable checkout solutions that can be tailored to match the look and feel of the application. Developers have the flexibility to design a seamless checkout experience for users, enhancing user satisfaction and trust during the payment process.
- **Subscription Management**: Stripe supports subscription-based billing models, allowing businesses to offer recurring payment plans to customers. In this project, Stripe's subscription management features can be leveraged to implement subscription-based services, such as premium memberships or subscription boxes, with ease.
- **International Support**: Stripe supports payments in multiple currencies and languages, making it suitable for businesses operating globally. With Stripe, this project can accept payments from customers worldwide, facilitating international transactions and expanding the reach of the application.
- **Mobile Payments**: Stripe supports mobile payments, enabling users to make purchases seamlessly from mobile devices. With the increasing prevalence of mobile commerce, Stripe's mobile payment capabilities enhance the accessibility and convenience of the application for users on-the-go.

# 5. Software Development Life Cycle

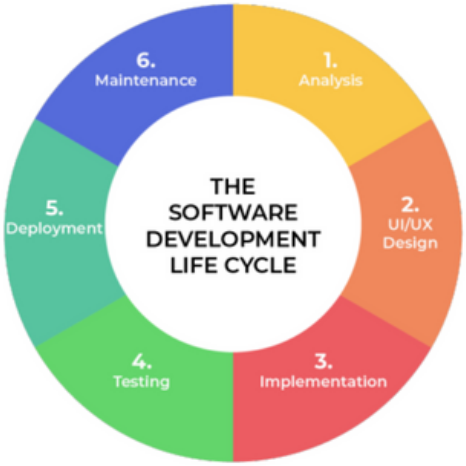
The Software Development Life Cycle (SDLC) is a process followed by software development teams to design, build, test, and deploy high-quality software.

In this project I have used the Agile model of software development. The **Agile** software development life cycle (SDLC) model is characterized by its iterative and incremental approach, focusing on delivering value to customers through collaboration, flexibility, and continuous improvement.

The key stages in Agile model are as follows:

- **Planning:** Establish project goals and requirements, focusing on high-level objectives rather than detailed plans.
- **Iterative Development:** Break the project into small iterations, delivering incremental value with each iteration.
- **Continuous Feedback:** Gather feedback from stakeholders throughout the development process to validate assumptions and prioritize features.
- **Adaptation:** Embrace change and adjust plans based on feedback and evolving requirements.
- **Continuous Integration and Testing:** Integrate code changes frequently and perform automated testing to ensure product quality.
- **Delivery:** Deliver a potentially shippable product increment at the end of each iteration.
- **Continuous Improvement:** Reflect on processes and performance regularly to identify areas for improvement and implement changes.

Overall, Agile SDLC promotes collaboration, transparency, and customer focus, enabling teams to deliver value early and often while adapting to changing requirements and priorities.



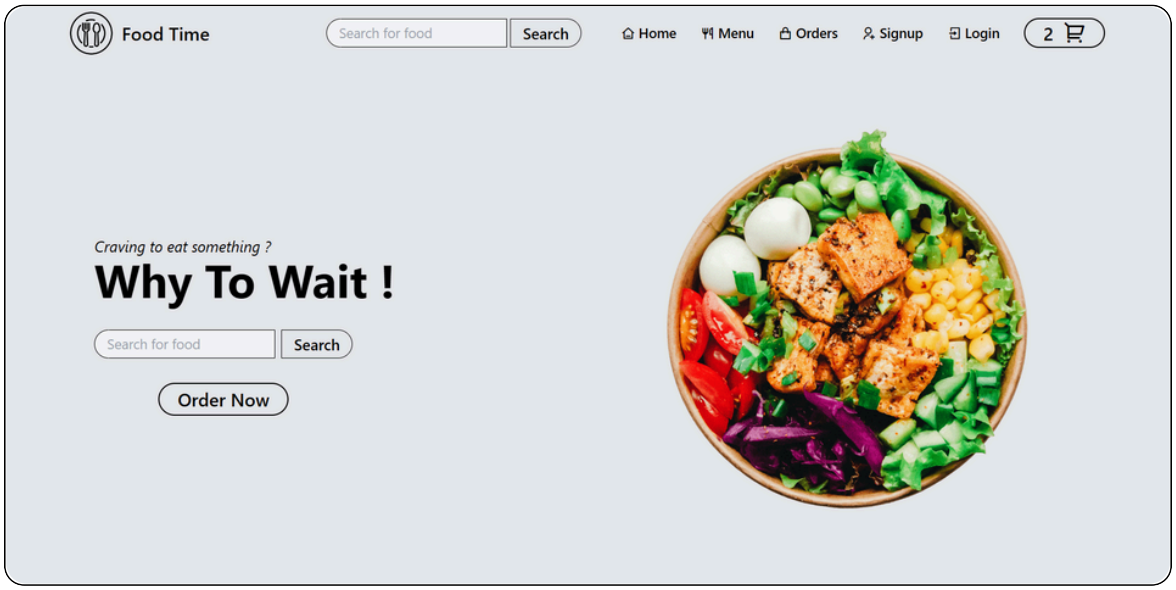
# 6. Testing the Project

Testing is the process of evaluating a system or application to identify any errors, gaps, or missing requirements in contrast to the actual requirements. The goal is to ensure that the software behaves as expected and meets the requirements.

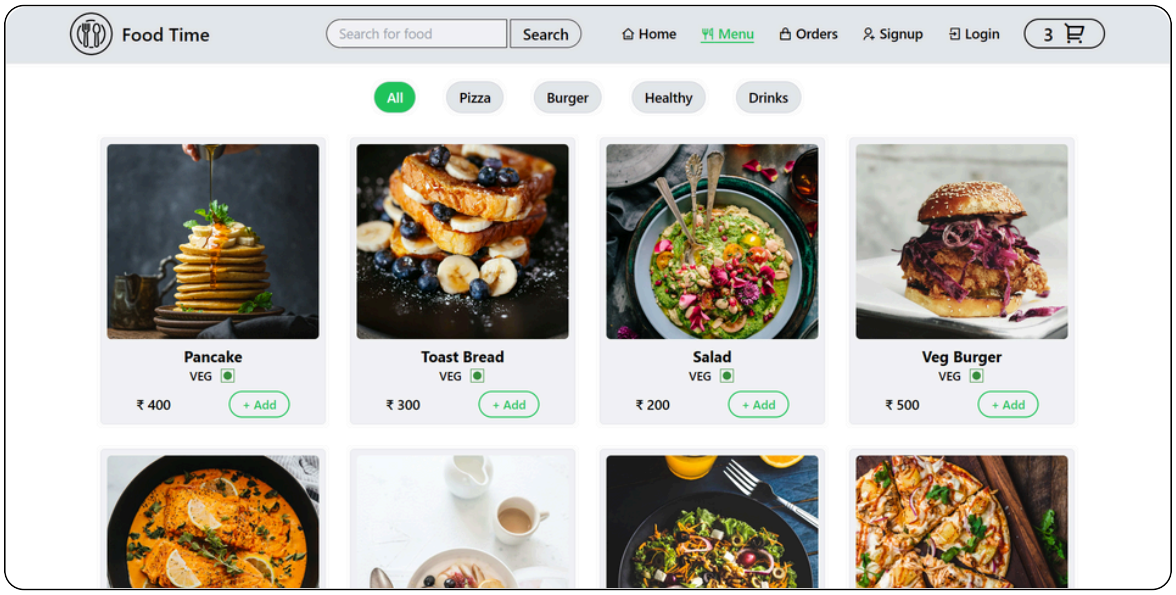
1. **Unit Testing:**
   - We conducted thorough unit testing for individual components of the system, ensuring that functions, modules, and services worked as expected.
2. **Integration Testing:**
   - Integration testing was crucial to verify that different parts of the system, such as the facial recognition module, user authentication, and MongoDB database, worked seamlessly together.
3. **End-to-End Testing:**
   - We performed end-to-end testing to simulate real-world scenarios, checking the entire flow from placing order to data storage and payment.
4. **User Testing:**
   - User testing played a significant role. We invited potential users to interact with the system, providing insights into its user-friendliness and identifying areas for improvement.
5. **Scalability Testing:**
   - To prepare for potential growth, we conducted scalability testing to assess how the system performed under varying loads, ensuring it could handle increased usage.
6. **Security Testing:**
   - Security was a top priority. We performed security testing to identify and address vulnerabilities, safeguarding user data and system integrity.
7. **Performance Testing:**
   - Performance testing helped optimize the system's responsiveness. We measured load times, response times, and resource utilization to ensure a smooth user experience.
8. **User Acceptance Testing (UAT):**
   - Before the official launch, we organized user acceptance testing sessions, allowing users to validate that the system met their requirements and expectations.
9. **Bug Tracking and Resolution:**
   - Bugs identified during testing were promptly logged, tracked, and resolved. This iterative process improved the system's stability and reliability.
10. **Adapting to User Suggestions:**
    - Users' suggestions and observations during testing influenced our development decisions. This iterative approach allowed us to align the system with user needs.

# 7. Project Views

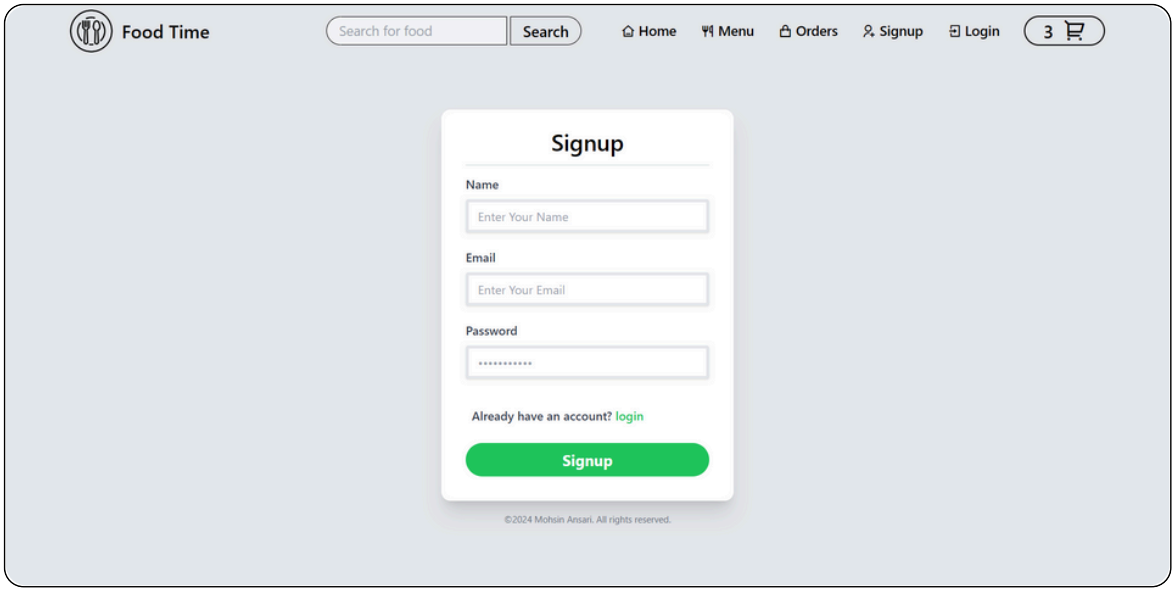
## Landing Page



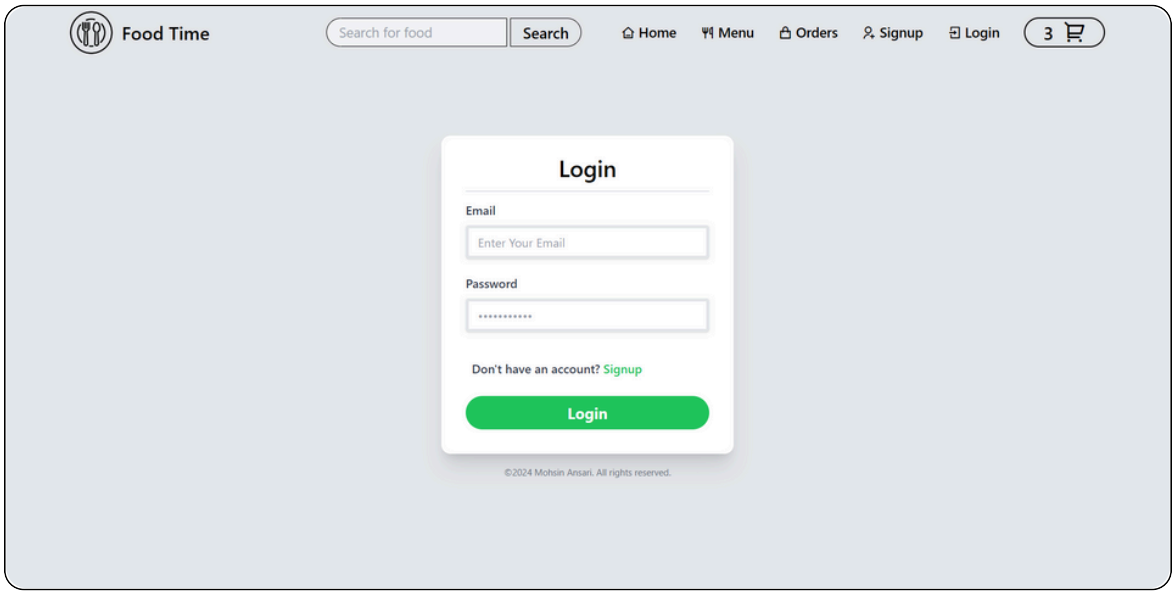

## Food Menu



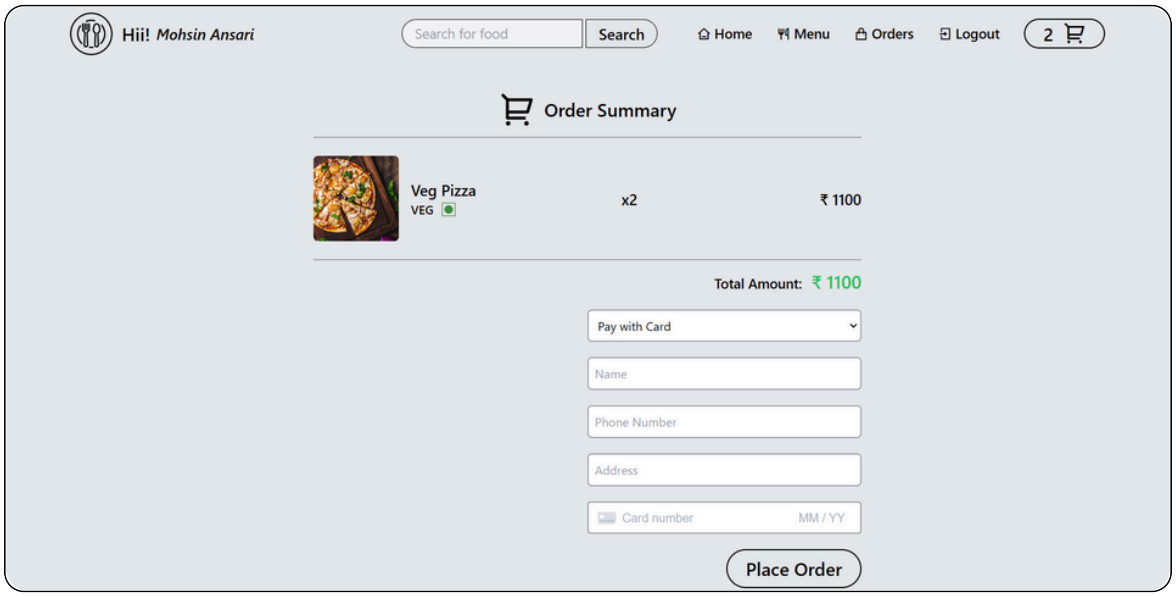
# Signup Page



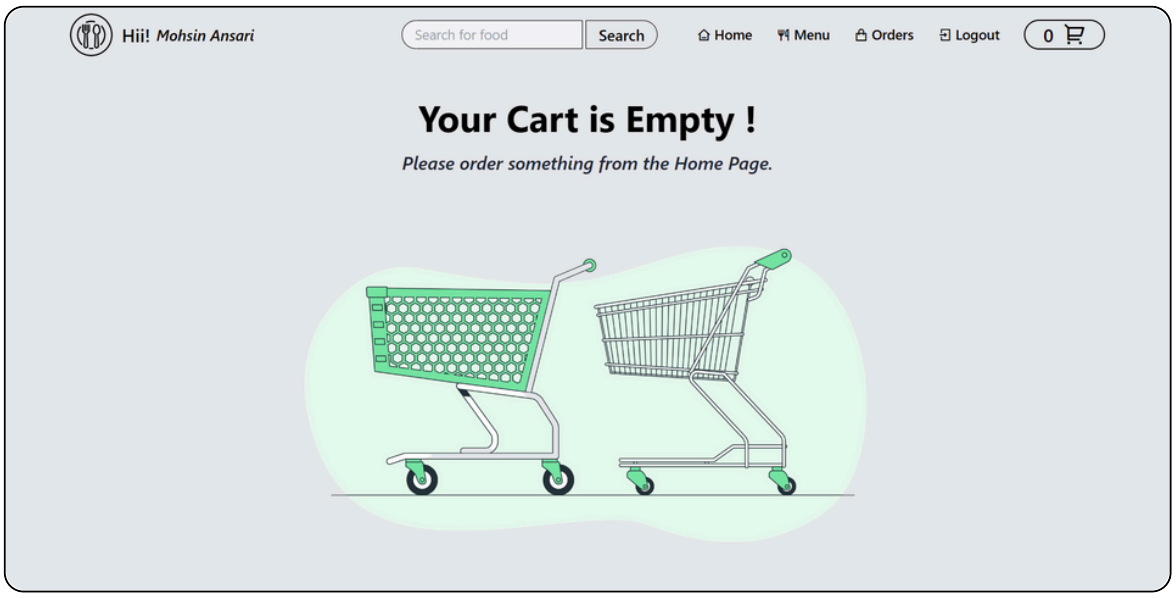

# Login Page



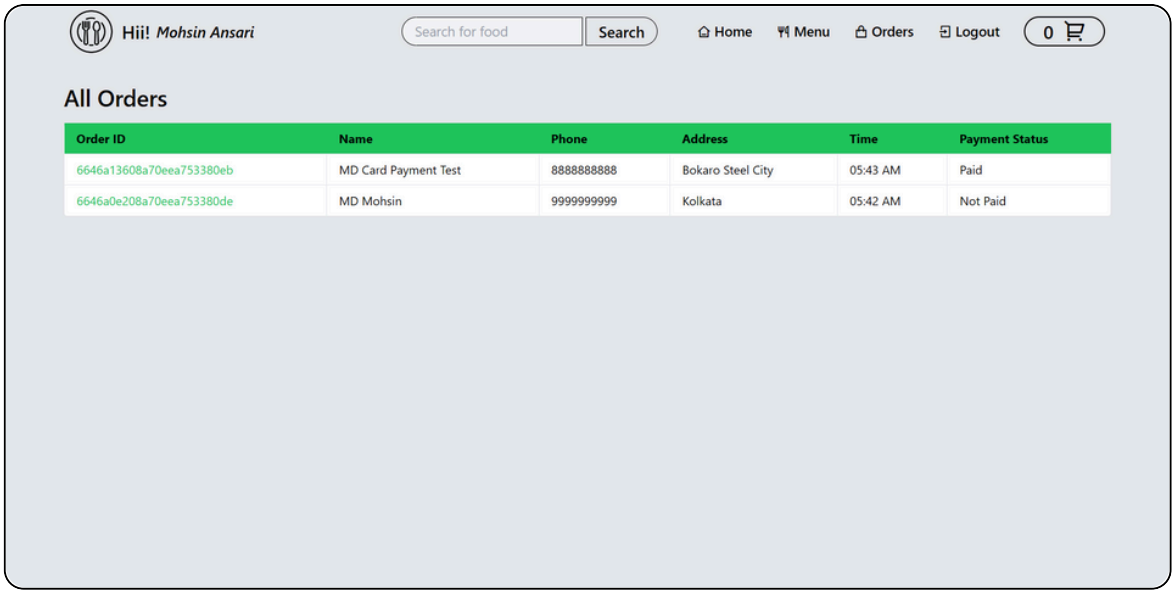
# Cart Page



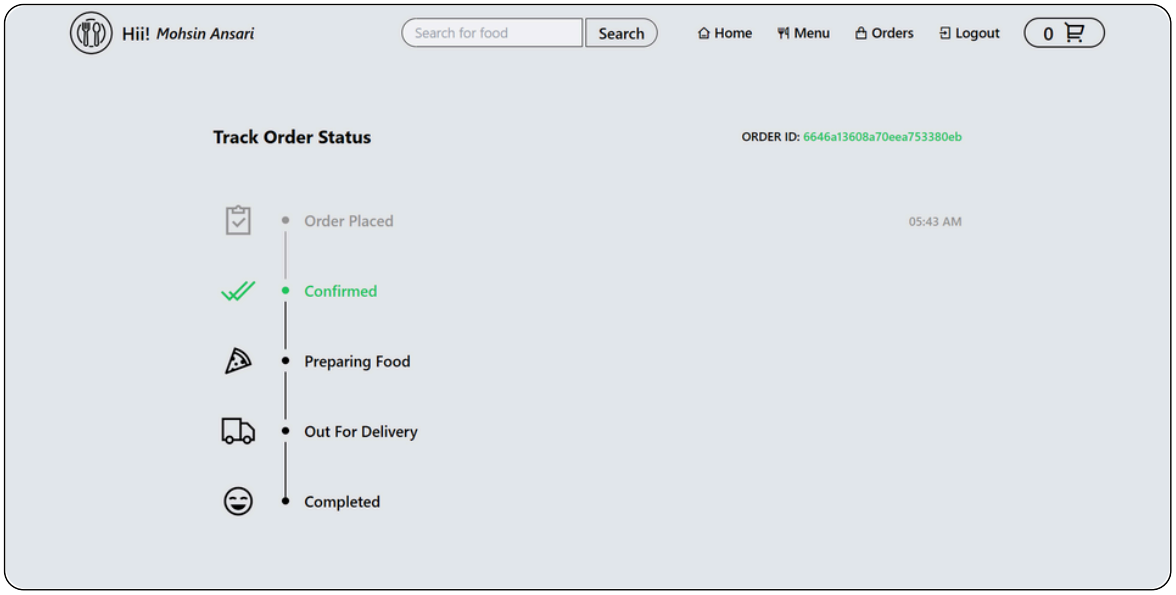

# Empty Cart Page



# Orders Page




# Order Status Page



# 8. Code Snippets

## package.json

```json
{
  "name": "food-delivery-website",
  "version": "1.0.0",
  "description": "Realtime food delivery website",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "dev": "nodemon app.js",
    "start": "node app.js",
    "tailwind": "npx tailwindcss -i ./public/css/input.css -o ./public/css/output.css --watch"
  },
  "author": "Mohsin Ansari",
  "license": "MIT",
  "dependencies": {
    "axios": "^1.6.8",
    "bcrypt": "^5.1.1",
    "connect-mongo": "^5.1.0",
    "dotenv": "^16.4.5",
    "ejs": "^3.1.9",
    "express": "^4.18.3",
    "express-ejs-layouts": "^2.5.1",
    "express-flash": "^0.0.2",
    "express-session": "^1.18.0",
    "moment": "^2.30.1",
    "mongoose": "^8.2.3",
    "noty": "^3.2.0-beta-deprecated",
    "passport": "0.5.0",
    "passport-local": "^1.0.0",
    "stripe": "^15.6.0"
  },
  "devDependencies": {
    "nodemon": "^3.1.0",
    "tailwindcss": "^3.4.1"
  }
}

```

## tailwind.config.js

```js
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: ["./**/*.{html,js,ejs}"],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

# app.js (Main Server)

```
//! Food Delivery Main Server
require('dotenv').config()
const express = require('express')
const path = require('path')
const app = express()
const flash = require('express-flash');
const mongoose = require('mongoose');
const session = require('express-session');
const MongoStore = require('connect-mongo');
const passport = require('passport');    // passport.js for login

//? Database Connection
mongoose.connect(process.env.MONGO_URI);
const DB = mongoose.connection;
DB.on('open', () => console.log("Connected To FoodWebsite-DB"));
DB.on('error', (error) => console.error(error));

//? Middlewares
app.use(express.urlencoded({ extended: true }));     // to parse form data
app.set('views', path.join(__dirname, '/views'));    // views path
app.set('view engine', 'ejs');                       // view engine setup
app.use(express.static('public'));                   // static folder path
app.use(flash());                                    // using flash message
app.use(express.json());
```

```
//? Session Config (to store a session to the DB)
app.use(session({
    secret: process.env.COOKIE_SECRET,
    resave: false,
    saveUninitialized: false,
    cookie: { maxAge: 1000 * 60 * 60 * 24 },     // 24hrs cookies validation
    store: MongoStore.create({
        mongoUrl: process.env.MONGO_URI,         // MongoDB connection URL
        collectionName: 'sessions',              // Collection name in the DB
    })
}))

//? Passport config
const passportConfig = require('./app/config/passport');
passportConfig(passport);
app.use(passport.initialize());
app.use(passport.session());

//? Getting all web routes
const webRouter = require('./routes/web')
// calling web.js to execute the function inside and passing app instance
webRouter(app);

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
    console.log(`Server listening on port ${PORT}`);
})
```

# Routes & Controllers

```
//! All Routes Function
// All the routes are here
// Logic of routes are inside /controllers

//* Controllers
const homeController = require('../app/http/controllers/homeController')
const authController = require('../app/http/controllers/authController')
const cartController = require('../app/http/controllers/customers/cartController')
const orderController = require('../app/http/controllers/customers/orderController')
const adminOrderController = require('../app/http/controllers/admin/orderController')
const statusOrderController = require('../app/http/controllers/admin/statusController')
const errorController = require('../app/http/controllers/errorController')
const searchController = require('../app/http/controllers/searchController')

//* Middlewares
const guest = require('../app/http/middlewares/guest');            // LoggedIn user cannot access /login & /signup only Guests can
const isLoggedIn = require('../app/http/middlewares/isLoggedIn');  // Protect Routes to avoid access without login
const admin = require('../app/http/middlewares/admin');            // Protect Routes from other users(only admins can access the route)
```

```
//* Routes
const allRoutes = (app) => {
    //? Home Route
    app.get('/', homeController().home); // using controller as a callback function

    //? Cart Route
    app.get('/cart', cartController().index)
    app.post('/update-cart', cartController().update)

    //? Orders/Customer Route
    app.post('/orders', isLoggedIn, orderController().store)
    app.get('/customer/orders', isLoggedIn, orderController().index)
    app.get('/customer/orders/:id', isLoggedIn, orderController().showStatus) // order status user

    //? Admin Routes (Can only be accessed by Admins)
    app.get('/admin/orders', admin, adminOrderController().index);
    app.post('/admin/order/status', admin, statusOrderController().update);

    //? Login Route
    app.get('/login', guest, authController().login)
    app.post('/login', authController().postLogin)

    //? Signup Route
    app.get('/signup', guest, authController().signup)
    app.post('/signup', authController().postSignup)

    //? Logout Route
    app.get('/logout', authController().logout)

    //? Search Route
    app.get('/search', searchController().search)

    //? Error Route
    app.get('*', errorController().err)
}

module.exports = allRoutes;
```

## Home Controller

```
//! Home Controller
const menuModel = require('../../models/menu');

function homeController() {
    return {
        async home(req, res) {
            const foodItems = await menuModel.find();
            return res.render('homePage', { foodItems: foodItems });
            // console.log(foodItems);
        }
    }
}

// Exporting the controllers
// To use, import in the routes
module.exports = homeController
```

## Search Controller

```
//! Search Food Controller
const menuModel = require('../../models/menu');

function searchController() {
    return {
        async search(req, res) {
            try {
                // Retrieve search term from query parameters
                const search = req.query.search ? req.query.search.replace(/[.*+?^${}()|[\]\\]/g, '\\$&') : '';

                // Find items matching the search term
                const itemData = await menuModel.find({ "name": { $regex: new RegExp(search, 'i') } });

                if (itemData.length > 0) {
                    // Send the found items as response to the frontend
                    // res.status(200).send({ data: itemData });
                    res.render('searchPage', { foodItems : itemData })
                } else {
                    // res.status(404).send({ message: 'No items found matching the search criteria.' });
                    res.redirect('/#items');
                }
            } catch (err) {
                console.error(err);
                res.status(500).send({ error: 'An internal server error occurred.' });
            }
        }
    }
}

module.exports = searchController;
```

# Auth Controller (Login, Signup)

```
//! Login & Signup controllers
const userModel = require('../../models/user');
const bcrypt = require('bcrypt');
const passport = require('passport');   // passport.js for login

function authController() {
    // Function to redirect 'user' OR 'admin'
    const _getRedirectUrl = (req) => {
        return req.user.role === 'admin' ? '/admin/orders' : '/customer/orders';
    }

    return {
        //? Login
        login(req, res) {
            res.render('loginPage');
        },
        postLogin(req, res, next) {
            const { email, password } = req.body
            //* Validations and display flash error
            if (!email || !password) {
                req.flash('error', 'Fields are empty!');
                // storing the input data before refreshing the page
                return res.redirect('/login');
            }

            passport.authenticate('local', (err, user, info) => {
                if (err) {
                    req.flash('error', info.message)
                    return next(err);
                }

                if (!user) {
                    req.flash('error', info.message)
                    return res.redirect('/login');
                }

                req.logIn(user, (err) => {
                    if (err) {
                        req.flash('error', info.message)
                        return next(err);
                    }
                    // Redirect after login is successful
                    // For customer  -> /customer/orders
                    // For admin     -> /admin/orders
                    return res.redirect(_getRedirectUrl(req));
                })
            })(req, res, next);
        },
```

```
        //? Signup
        signup(req, res) {
            res.render('signupPage');
        },
        async postSignup(req, res) {
            const { name, email, password } = req.body
            //* Validations and display flash error
            if (!name || !email || !password) {
                req.flash('error', 'Fields are empty!');
                // storing the input data before refreshing the page
                req.flash('name', name);
                req.flash('email', email);
                return res.redirect('/signup');
            }

            //* Check if email already exists
            const existingUser = await userModel.findOne({ email: email });
            if (existingUser) {
                req.flash('error', 'Email Already Exists');
                req.flash('name', name);
                req.flash('email', email);
                return res.redirect('/signup');
            }

            //* Hash Password using Bcrypt before storing it in the DB
            const hashedPassword = await bcrypt.hash(password, 10);

            //* Create user using the model and data from the form input
            const user = new userModel({
                name: name,
                email: email,
                password: hashedPassword
            })

            //* Save to DB
            user.save().then((user) => {
                // if user is created - redirect to login page
                return res.redirect('/login');
            }).catch(err => {
                req.flash('error', 'Failed to signup!');
                return res.redirect('/signup');
            })
            // console.log(req.body);
        },

        //? Logout
        logout(req, res) {
            req.logout(); // function by Passport to logout the user
            req.flash('success_msg', 'You have been logged out successfully'); // flash a success message
            res.redirect('/'); // Redirect the user to Home page
        }
    }
}

module.exports = authController;
```

# Cart Controller

```
//! Cart controller
function cartController() {
    return {
        index(req, res) {
            res.render('cartPage');
        },

        update(req, res) {
            //* Structure of data to be stored in session
            // let cart = {
            //     items : {
            //         itemdId: { item: itemObject, qty:0 },
            //     },
            //     totalQty: 0,
            //     totalPrice: 0,
            // }

            // Check if cart does not exists
            if (!req.session.cart) {
                req.session.cart = {
                    items: {},
                    totalQty: 0,
                    totalPrice: 0,
                }
            }
            let cart = req.session.cart

            // Check if item does not exists in cart
            if (!cart.items[req.body._id]) {
                cart.items[req.body._id] = {
                    item: req.body,
                    qty: 1,
                },
                    cart.totalQty = cart.totalQty + 1,
                    cart.totalPrice = cart.totalPrice + req.body.price
            } else {
                // if already exists increase the quantity
                cart.items[req.body._id].qty = cart.items[req.body._id].qty + 1,
                    cart.totalQty = cart.totalQty + 1,
                    cart.totalPrice = cart.totalPrice + req.body.price
            }

            return res.json({ totalQty: req.session.cart.totalQty })
        }
    }
}

module.exports = cartController;
```

# Order Controller

```
//! Order Controller
const orderModel = require('../../../models/order');    // Orders Model
const moment = require('moment');   // For formatting Date & Time
const stripe = require('stripe')(process.env.STRIPE_SECRET_KEY)

function orderController() {
    return {
        store(req, res) {
            // console.log(req.body);
            //* Validate request
            const { name, phone, address, stripeToken, paymentType } = req.body;
            if (!name || !phone || !address) {
                return res.status(422).json({ message: 'All fields are required' })
                // req.flash('error', 'All fields are required')
                // return res.redirect('/cart');
            }

            //* Creating an order
            const order = new orderModel({
                customerId: req.user._id,
                items: req.session.cart.items,
                name: name,
                phone: phone,
                address: address,
                // Payment default from schema
                // Status default from schema
            });

            //* Save order to DB
            order.save().then((placedOrder) => {
                //* Stripe Payment
                if (paymentType === 'card') {
                    return stripe.charges.create({
                        amount: req.session.cart.totalPrice * 100,
                        source: stripeToken,
                        currency: 'inr',
                        description: `Your Order: ${placedOrder._id}`
                    }).then(() => {
                        placedOrder.paymentStatus = true;
                        placedOrder.paymentType = paymentType;
                        // Save the updated order with payment status
                        return placedOrder.save();
                    }).then((updatedOrder) => {
                        // Empty the cart from session of the user when order is placed
                        delete req.session.cart;
                        return res.json({ message: 'Payment successful, Your order is placed' });
                    }).catch((err) => {
                        console.error(err);
                        delete req.session.cart; // Clear cart even if there's an error
                        return res.json({ message: 'Payment Failed, Order Placed As COD' });
                    });
                } else {
                    // Empty the cart from session of the user when order is placed
                    delete req.session.cart;
                    return res.json({ message: 'Order placed successfully' });
                }
            }).catch((err) => {
                console.error(err);
                return res.status(500).json({ message: 'Something Went Wrong' });
            });
        },
```

```
        async index(req, res) {
            // Fetching orders using customerId
            const orders = await orderModel.find({ customerId: req.user._id }, null, { sort: { createdAt: -1 } });
            res.header('Cache-Control', 'no-cache, private, no-store, must-revalidate, max-stale=0, post-check=0, pre-check=0');
            res.render('ordersPage', { orders: orders, moment: moment });
            // console.log(orders);
        },

        async showStatus(req, res) {
            const order = await orderModel.findById(req.params.id)

            // check if user is authorized with the order
            // if userId === orderId
            if (req.user._id.toString() === order.customerId.toString()) {
                return res.render('customers/orderStatusPage', { order: order });
            }
            // redirect to homePage if user was not allowed
            return res.redirect('/');
        }
    }
}

module.exports = orderController;
```

# Admin Status Controller

```
//! Admin Status Controller
const orderModel = require('../../../models/order');

function statusController() {
    return {
        // Update order status in the DB
        async update(req, res) {
            // Replace status of order with whatever the admin click on the admin page.
            try {
                await orderModel.updateOne({ _id: req.body.orderId }, { status: req.body.status })
                return res.redirect('/admin/orders');
            } catch (err) {
                console.error(err);
                return res.redirect('/admin/orders');
            };
        }
    }
}

module.exports = statusController;
```

# Middlewares

```
//! isLoggedIn Middleware
// This middleware will be used to protect the routes
// (only loggedin users can access)

function auth(req, res, next) {
    if (req.isAuthenticated()) {
        return next()
    } else {
        res.redirect('/login')
    }
}

module.exports = auth;
```

```
//! Middleware to Make sure only admin can access the route
function admin(req, res, next) {
    // If loggedIn & Role is 'admin'
    if (req.isAuthenticated() && req.user.role === 'admin') {
        return next();
        // success
    } else {
        return res.redirect('/');
        // Else the user will be redirected to home page
    }
}

module.exports = admin;
```

```
// Login and register route can only be accessed if user is not logged in
function guest(req, res, next) {
    if (!req.isAuthenticated()) {
        return next();
    }
    return res.redirect('/');
}

module.exports = guest;
```

# Database Models (Schema)

## Food Menu Model

```
//! Food Menu Model
const mongoose = require('mongoose');

const menuSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    image: {
        type: String,
        required: true
    },
    price: {
        type: Number,
        required: true
    },
    category: {
        type: String,
        default: "all"
    }
})

module.exports = mongoose.model('Menu', menuSchema);
```

# Orders Model

```
//! Orders Model
const mongoose = require('mongoose');

const orderSchema = mongoose.Schema({
    // Here customerId is the data association of 'user'
    customerId: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'user',
        required: true
    },
    items: {
        type: Object,
        required: true
    },
    name: {
        type: String,
        required: true
    },
    phone: {
        type: String,
        required: true
    },
    address: {
        type: String,
        required: true
    },

    paymentType: {
        type: String,
        default: 'COD'
    },
    paymentStatus: {
        type: Boolean,
        default: false,
    },
    status: {
        type: String,
        default: 'order_placed'
    }
}, { timestamps: true });

module.exports = mongoose.model('order', orderSchema);
```

# User Model

```
//! User Model
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
    },
    email: {
        type: String,
        required: true,
        unique: true
    },
    password: {
        type: String,
        required: true,
    },
    role: {     // customer OR admin
        type: String,
        default: 'customer'
    }
}, { timestamps: true })

module.exports = mongoose.model('user', userSchema);
```

# 9. Deployment & Source Code

**Deployment:**
The Food delivery web application has been successfully deployed on Render as well as AWS Ubuntu VPS, a platform for continuous deployment. This deployment method ensures that the latest changes to the project are automatically reflected in the live application, providing a seamless and up-to-date experience for users. The deployment on Render simplifies the deployment process, making it efficient and allowing for quick updates.
Deployed Link : ClickHere

**Source Code:**
The complete source code for the Food delivery web application is available on the GitHub repository with License. This repository serves as a centralized location for the project files, including server-side scripts, frontend code, Installation and configuration. Users and developers can access the source code, contribute to the project, or explore the implementation details. The GitHub repository enhances collaboration and transparency in the development process.
Source Code Link: ClickHere

# 10. Future Scope

- **Enhanced User Experience**:
  - Continuously improve the user interface and experience based on user feedback and emerging design trends, ensuring a seamless and enjoyable experience for customers.

- **Integration of Additional Services**:
  - Explore partnerships with other service providers, such as grocery delivery, meal kit subscriptions, or restaurant reservations, to offer a comprehensive solution for users' food-related needs.

- **Customer Loyalty Programs:**
  - Implement loyalty programs, discounts, and promotional offers to incentivize repeat orders and foster customer loyalty.

- **Recruitment and Team Building:**
  - As the project grows, recruit a dedicated team of developers, designers, marketers, and customer support personnel to scale operations and drive growth effectively.

- **Migration to Next.js:**
  - Consider migrating the codebase to Next.js, a React framework for building server-rendered web applications, to leverage its benefits such as improved performance, SEO, and developer experience, resulting in enhanced user experience.

- **Integration of Machine Learning:**
  - Explore the integration of machine learning algorithms for personalized recommendations, predictive analytics, and fraud detection, enhancing the platform's efficiency and user satisfaction.

- **Voice and Chatbot Integration:**
  - Integrate voice-based interfaces and chatbots to provide users with more natural and conversational interactions, improving accessibility and customer support capabilities.

# 11. Conclusion

In conclusion, the Food Delivery website embodies innovation and convenience in the culinary landscape. By harnessing the power of modern web technologies such as Node.js, Express.js, and MongoDB, coupled with intuitive user interfaces, it offers a seamless and delightful experience for both customers and restaurant partners.

The incorporation of features like real-time order tracking, secure payment gateways powered by Stripe, and dynamic menus ensures a satisfying and efficient food ordering process. Leveraging agile methodologies throughout development has allowed for continuous enhancements and adaptations to meet evolving user needs.

Deployed on robust platforms like Render, the website ensures reliability and scalability, while GitHub hosts the project's source code, fostering collaboration and transparency within the development community.

In essence, the Food Delivery website represents a harmonious fusion of technology and gastronomy, redefining the way people experience and enjoy food delivery services.

Thank you.