Design and Implementation of Foodie: a Food Ordering System

by

Md. Mamun Hossen
ID: CSE2103024112

Somi
ID: CSE2103024117

Md. Kawsar Ali
ID: CSE2103024125

Rashidul Islam
ID: CSE2103024171

Supervised by
Arifur Rahaman

Submitted in partial fulfillment of the requirements for the degree of
Bachelor of Science in Computer Science and Engineering



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SONARGAON UNIVERSITY (SU)

May 2025

Design and Implementation of Foodie: a Food Ordering System

by

Md. Mamun Hossen
ID: CSE2103024112

Somi
ID: CSE2103024117

Md. Kawsar Ali
ID: CSE2103024125

Rashidul Islam
ID: CSE2103024171

Supervised by
Arifur Rahaman

Submitted in partial fulfillment of the requirements for the degree of
Bachelor of Science in Computer Science and Engineering



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SONARGAON UNIVERSITY (SU)

May 2025

APPROVAL

The project titled **Design and Implementation of Foodie a Food Ordering System** submitted by Md. Mamun Hossen (CSE2103024112), Somi (CSE2103024117), Md.Kawsar Ali (CSE2103024125), Rashidul Islam (CSE2103024171) to the Department of Computer Science and Engineering, Sonargaon University (SU), has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering and approved as to its style and contents.

Board of Examiners

Arifur Rahaman
Assistant Professor & Coordinator,
Department of Computer Science and Engineering
Sonargaon University (SU)

Supervisor

(Examiner Name and Signature)
Department of Computer Science and Engineering
Sonargaon University (SU)

Examiner 1

(Examiner Name and Signature)
Department of Computer Science and Engineering
Sonargaon University (SU)

Examiner 2

(Examiner Name and Signature)
Department of Computer Science and Engineering
Sonargaon University (SU)

Examiner 3

DECLARATION

We, hereby, declare that the work presented in this report is the outcome of the investigation performed by us under the supervision of **Arifur Rahaman, Assistant Professor & Coordinator,** Department of Computer Science and Engineering, Sonargaon University, Dhaka, Bangladesh. We reaffirm that no part of this project has been or is being submitted elsewhere for the award of any degree or diploma.

Countersigned

(Arifur Rahaman)
Supervisor

Signature

Md. Mamun Hossen
ID: CSE2103024112

Somi
ID: CSE2103024117

Md. Kawsar Ali
ID: CSE2103024125

Rashidul Islam
ID: CSE2103024171

ABSTRACT

This project presents the design and implementation of a modern food website developed using the MERN stack—MongoDB, Express.js, React, and Node.js. The website serves as a dynamic and interactive platform that allows users to explore various food items, view detailed menus, and place orders seamlessly. The backend, powered by Node.js and Express.js, handles API requests, manages user authentication, and interfaces with a MongoDB database to store and retrieve user data, orders, and menu information. The frontend, developed using React, provides a responsive and intuitive user interface, ensuring a smooth browsing experience across devices. Key features include user registration and login, category-based food browsing, cart and checkout functionality, and an admin panel for managing content. The project aims to enhance the digital food ordering experience by offering real-time interactions, scalability, and a clean UI/UX. This book documents the full development lifecycle of the project, from system analysis and design to deployment and testing, offering insights into building full-stack web applications using the MERN architecture.

ACKNOWLEDGMENT

First and foremost, we would like to express my deepest gratitude to Mr. Arifur Rahaman Assistant Professor & Coordinator, Department of Computer Science and Engineering Sonargaon University for his invaluable guidance, support, and encouragement throughout the development of this project. Their insights and expertise were instrumental in shaping the direction and success of this food website.

We also extend our heartfelt thanks to the faculty members of Sonargaon University for providing the knowledge and foundation necessary for this work. Their dedication to teaching and mentorship has been a source of inspiration.

Special thanks to Our friends for their constant support, motivation, and understanding during the entire development process.

Finally, we would like to acknowledge all the online communities, forums, and documentation contributors whose shared knowledge played a significant role in helping me troubleshoot, learn, and improve throughout the journey of building this MERN-based web application.

LIST OF ABBREVIATIONS

API	Application Programming Interface
CDN	Content Delivery Network
CSS	Cascading Style Sheets
CRUD	Create, Read, Update, Delete
DB	Database
JWT	JSON Web Token
HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
JS	JavaScript
JSON	JavaScript Object Notation
MERN	MongoDB, Express.js, React, Node.js
MVC	Model-View-Controller
NPM	Node Package Manager
REST	Representational State Transfer
UI	User Interface
UX	User Experience

TABLE OF CONTENTS

Title Page No.

DECLARATION ……..……. iii

ABSTRACT ……...…… iv

ACKNOWLEDGEMENT ……..…….. v

LIST OF ABBREVIATION ……..…… vi

CHAPTER 1 1-3

INTRODUCTION AND OBJECTIVE

1.1 Introduction ……..…….…..... 1

1.2 Objectives …..……...…….…..... 1

1.3 Related Work……. 2

1.4 Needs of Online Food Order……. 3

1.5 Functionalities……... 3

CHAPTER 2 4 – 6

REQUIREMENT AND ANALYSIS

2.1 Software Requirement Specification (SRS)…….…..…. 4

2.1.1 Data Gathering…….……...…..... 4

2.1.2 Feasibility Study…… 4

2.1.3 Software Process Model…….…..…..……..... 4

2.2 Hardware Requirement…….…..……..... 4

2.3 Software Requirement…….…..……..... 4

2.4 Justification of Selection of Technology…….. 5

2.4.1 Backend.……..…… 5

2.4.2 Frontend.……...……... 5

2.4.3 Database……..……..... 5

2.5 Data Flow Diagram (DFD)……..……. 5 – 6

CHAPTER 3 7 – 14

SYSTEM DESIGN

3.1 ER Diagram ……..……..... 7

3.2 Data Dictionary ……..…….…….. 7

3.3 Database Normalization…….…… 8

3.4 Architecture: Clean Layers…….. 12 – 14

CHAPTER 4 15-16

TESTING

4.1 Testing Approach……..…..... 15

4.1.1 Purpose of Testing ……..……..... 15

4.1.2 Types of Testing Applied ……..……..……. 15

4.1.3 Testing Summary…… ……...…….....……. 16

CHAPTER 5 15 – 23

SYSTEM IMPLEMENTATION

5.1 Output Screen ……..…..……. 17 – 25

CHAPTER 6 26

CONCLUSION AND FUTURE WORKS

6.1 Conclusion …… 26

6.2 Limitation ……...…….. 26

6.3 Future Scope ……...……..……..... 26

CHAPTER 7 27

REFERENCES ……..……. 27

LIST OF TABLES

Table No.	Title	Page No.
Table 3.2	Data Dictionary	8
Table 3.3	Database Normalization	8
Table 4.2.3	Testing Summary	16

LIST OF FIGURES

Figure No.	Title	Page No.
Fig.2.5.1	DFD Level 0	5
Fig.2.5.2	DFD Level 1	6
Fig.3.1	ER Diagram	7
Fig.3.3	Database Normalization	9
Fig.5.1.1	Home Page	17
Fig.5.1.2	Explore Menu	18
Fig.5.1.3	Menu	19
Fig.5.1.4	Footer	19
Fig.5.1.5	Sign Up & Login	20
Fig.5.1.6	Cart Item	21
Fig.5.1.7	Customer Information	22
Fig.5.1.8	Cart Item Page	23
Fig.5.1.9	Admin panel	23
Fig.5.2.0	Add Items	24
Fig.5.2.1	List Items	25

CHAPTER 1

INTRODUCTION AND OBJECTIVE

1.1 Introduction

In today's fast-paced world, convenience is a key factor in everyday choices—especially when it comes to food. The rising popularity of online food ordering platforms reflects the growing need for efficient, user-friendly digital solutions in the food service industry. Foodie is a web-based application designed to meet this demand by offering a seamless food browsing and ordering experience, built entirely using the MERN stack— MongoDB, Express.js, React, and Node.js.

Foodie is more than just a menu display; it is an interactive platform where users can explore a variety of food items, view detailed descriptions and images, add items to their cart, and place orders with ease. The system also includes a robust admin panel that allows restaurant owners or staff to manage menu items, monitor orders, and handle user accounts securely and efficiently.

The decision to use the MERN stack was driven by its flexibility and full-stack JavaScript environment, allowing for faster development and a more integrated architecture. React is utilized for creating a responsive, single-page user interface, while Express.js and Node.js handle the server-side logic and APIs. MongoDB provides a scalable, document-oriented database solution to store and retrieve all relevant data.

This documentation outlines the development lifecycle of Foodie, including planning, design, implementation, testing, and deployment. The goal of the project is to demonstrate how modern web technologies can be effectively used to solve real-world problems and improve user experience in the food ordering sector.

The suggested system is made to prevent users from acting inappropriately and making deadly mistakes. The scope of the suggested system is justifiable because many individuals are moving to different places, allowing a wide range of people to use it. The user will provide input to the system or interface. The key characteristics that will influence the dataset are: name, address, email-Id, mobile no, other personal related values, etc. The output will include user/customer's Order, Bill, Feedback and Payment options. Initially there will be mess services considered inside 2 to 3 areas. The reason why to choose this project is the idea behind project that is to solve problem of people which they are facing when they shift to different city. The system is not only for user but also for provider who provides food service. This system is for making efficient communication between consumer and producer of the food system which will then leads to the ideal and effective system.[1]

1.2 Objectives

The study's objective was to determine whether the application is user-centered and based onuser requirements. This system developed all problems pertaining to every user that it includes. Almost anyone may use the program if they know how to use an Android

smart phone. The various problems with Mess service will be resolved by this system. The implementation of an online food ordering system is done to assist and resolve significant issues for consumers. Based on the application, it can be said that: This system makes placing orders simple; it gives customers the information they need to place orders. Through the program, it is able to receive orders and change their data, and it also aids the administrator in managing all the Food system. [2]

General Objective:

To design and implement an interactive, user-friendly, and responsive food ordering website that allows users to browse menus, place orders, and manage their food choices online.

Specific Objectives:

1. To develop a dynamic frontend using React.js that ensures smooth user navigation, real-time updates, and a modern UI/UX experience.
2. To build a secure backend using Node.js and Express.js that handles user authentication, order processing, and communication with the database.
3. To integrate MongoDB as the database to store user data, food items, orders, and admin records in a scalable and efficient manner.
4. To implement user functionalities such as registration, login, food item browsing, cart management, and order placement.
5. To provide admin functionalities including adding, updating, or deleting food items, managing orders, and monitoring user activity.

1.3 Related Work

In [3] here an application of integration of hotel management systems by web services technology is presented. Digital Hotel Management integrates lots of systems of hotel industry such as Ordering System Kitchen Order Ticket (KOT), Billing System, Customer Relationship Management system (CRM) together. This integration solution can add or expand hotel software system in any size of hotel chains environment. This system increases quality and speed of service. This system also increases attraction of place for large range of customers. Implementing this system gives a cost-eficient opportunity to give your customers a personalized service experience where they are in control choosing what they want, when they want it – from dining to ordering to payment and feedback.Also in [4] paper they done a research work aims to design and develop a wireless food ordering system in the restaurant. The project presents in-depth on the technical operation of the Wireless Ordering System (WOS) including systems architecture, function, limitations and recommendations. It is believed that with the increasing use of handheld device e.g. PDAs in restaurants, pervasive application will become an important tool for restaurants to improve the management aspect by utilizing PDAs to coordinate food ordering could increase eficiency for restaurants and caterers by saving time, reducing human errors and by providing higher quality customer service. With the combination of simple design and readily available emerging communications technologies, it can be concluded that this system is an attractive solution for the hospitality industry

1.4 Needs of Online Food Order

Helping customers in placing meal orders whenever they want. Customers will be able to order their preferred foods at any time, but as we've already mentioned, this is only a limited option. As a result, restaurants need to have a specific system in place that will allow them to serve a large number of customers while streamlining operations. One of the best platforms is ordering, which offers all of these services in addition to a host of cutting-edge features that have helped countless small and large enterprises establish themselves as market leaders

1.5 Functionalities

- Provides search options based on a variety of criteria. like Food Item, Customer, Order, and Order Confirmation.
- Online food ordering systems also manage payment information for order details, order confirmation details, and food items online.
- It keeps track of all the data regarding Categories, Payments, Orders, etc.
- Displays the food item's information and description for the customer
- It focuses on keeping track of order's data and transactions.
- Manage the food item's information.
- Improvements in editing, adding, and updating records lead to proper resource management of food item data.
- Manage the order's information by combining all Confirm Order data.[5]

CHAPTER 2

REQUIREMENT AND ANALYSIS

2.1 Software Requirement Specification

Abilities will lead to better performance (Goodhue and Thompson, 1965). In the FVM however the construct is modified from the pereceived fit in the TTF model to use more objective Mobile Technology in Bussiness.[6]That is fit only considers the nature of technologu and the requirement of task itself.

2.1.1 Data Gathering

Data was collected through online surveys, user interviews, and analysis of existing food delivery platforms to identify key features expected in a modern food ordering system, such as ease of navigation, order tracking, and secure payments.

2.1.2 Feasibility Study

• Technical Feasibility: The MERN stack (MongoDB, Express.js, React, Node.js) provides an efficient and scalable environment suitable for building the application.

• Operational Feasibility: The system is user-friendly, meeting customer needs for online food ordering.

• Economic Feasibility: Open-source technologies minimize cost, making development affordable for individuals or small teams.

2.1.3 Software Process Model

A waterfall model under the software development life cycle (SDLC) is the methodology used to produce the food ordering system and the customer self-ordering system. It is used by system developers to produce or alter information systems or software. It divides the development process into several stages or processes. After the completion of one stage, it will logically move to another stage. Sometimes moving back to the previous stage is necessary due to failure that occurs in current stage.

2.2 Hardware Requirements

- Processor: Intel Core i5 or equivalent (minimum)
- Disk space: 8 GB (minimum)
- Hard Disk: 256 GB SSD or higher
- Graphics – DirectX 9- Capable Video Card
- Display – 1024 x 768 or Higher Resolution
- Internet Connection: Required for deployment and external libraries

2.3 Software Requirement

- Operating System: Windows / Linux / macOS
- Code Editor: Visual Studio Code
- Backend: Node.js, Express.js
- Frontend: React.js
- Database: MongoDB
- Browser: Chrome, Firefox (latest versions)
- Package Manager: NPM
- Version Control: Git & GitHub

2.4 Justification of Selection of Technology

The MERN stack was chosen for its efficiency, full JavaScript environment, and flexibility in both frontend and backend development.

2.4.1 Backend: Node.js & Express.js

- Provides scalable API services.
- Handles asynchronous tasks effectively.

2.4.2 Frontend: React.js

- Enables a responsive and dynamic user interface.
- Component-based architecture simplifies development.

2.4.3 Database: MongoDB

- NoSQL structure fits well with evolving data models.
- Easily integrates with Node.js using Mongoose.

2.5 Data Flow Diagram (DFD)

DFD Level 0 (Context Diagram)

Purpose:

Level 0 provides a high-level overview of the system. It shows how the entire system interacts with external entities such as users and admins.

Components:

- **External Entities**:
  - **User**: Can register, login, view menu, place order.
  - **Admin**: Manages menu, views orders.
- **Process**:
  - Foodie Website System (Main system where all logic resides)
- **Data Stores**:
  - **Database** (Holds users, food items, and orders)
- **Data Flows**:
  - User → Order Info, Login Data → System
  - Admin → Menu Updates, View Orders → System
  - System ↔ Database (for CRUD operations)
  - System → Order Confirmation, Menu → User/Admin

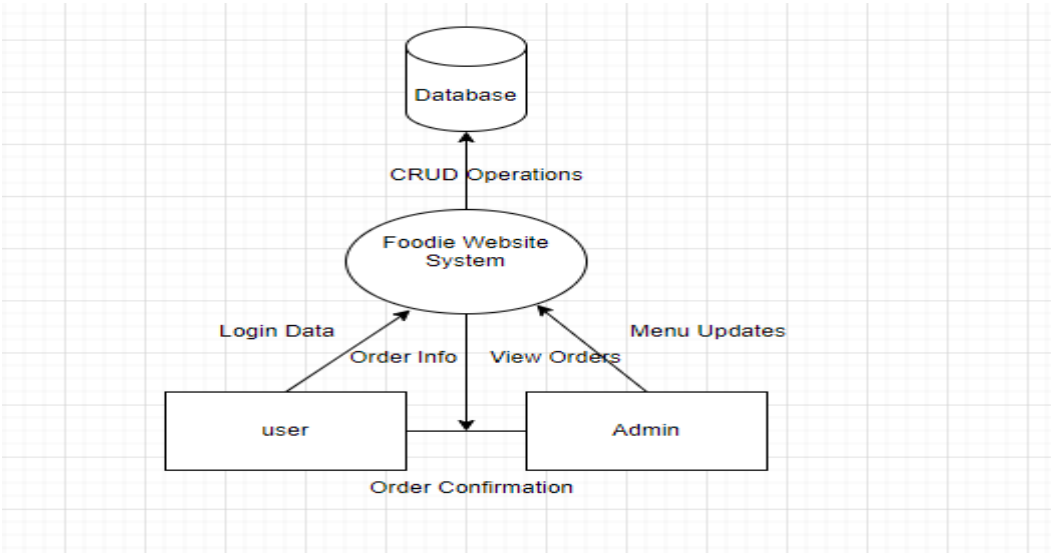


Fig.2.5.1: DFD Level 0

DFD Level 1 (Detailed Functional Flow)
Purpose:
Breaks down the main system into sub-processes to show how data flows between them in more detail.

Example Breakdown for Foodie:

1. **User Registration/Login**
    - Inputs: User credentials
    - Outputs: Success/failure response
2. **Browse Menu**
    - Inputs: Request from user
    - Outputs: List of food items
3. **Add to Cart / Place Order**
    - Inputs: Selected food items
    - Outputs: Order confirmation, stored in database
4. **Admin Menu Management**
    - Inputs: New items, updates, or deletions
    - Outputs: Confirmation, changes stored in database
5. **View Orders (Admin)**
    - Inputs: Admin request
    - Outputs: List of current and past orders

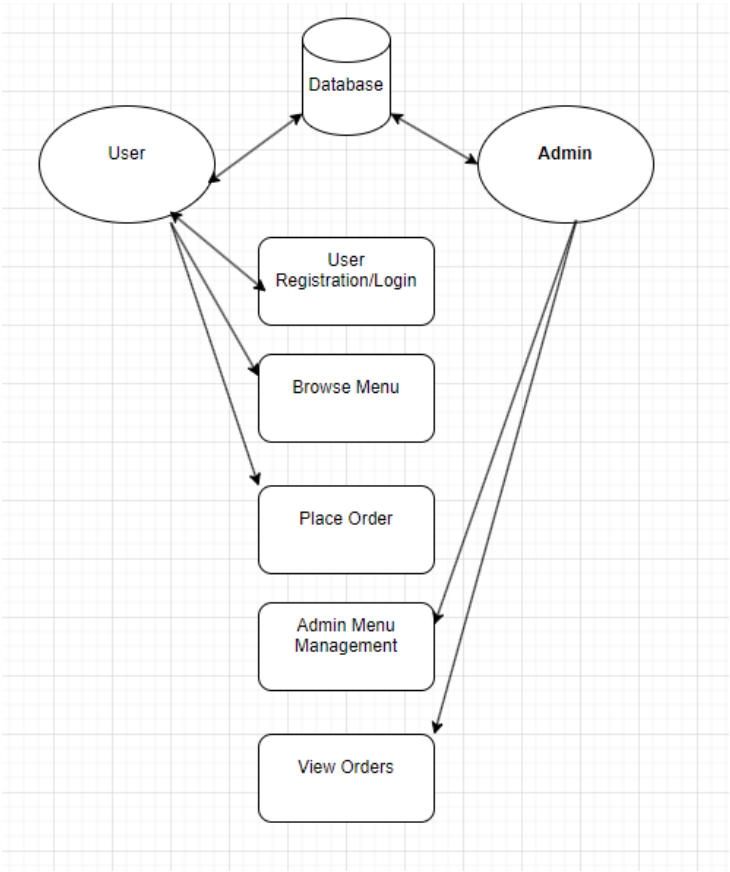


Fig.2.5.2: DFD Level 1

CHAPTER 3

SYSTEM DESIGN

3.1 ER Diagram

The Entity Relationship Diagram (ERD) visually represents how entities (like users, orders, food items) relate to each other in your database.

Key Entities:

- **User** (user_id, name, email, password, role)
- **FoodItem** is (item_id, name, category, price, image, availability)
- **Order** (order_id, user_id, items, total_amount, status, timestamp)
- **Cart** (user_id, items)

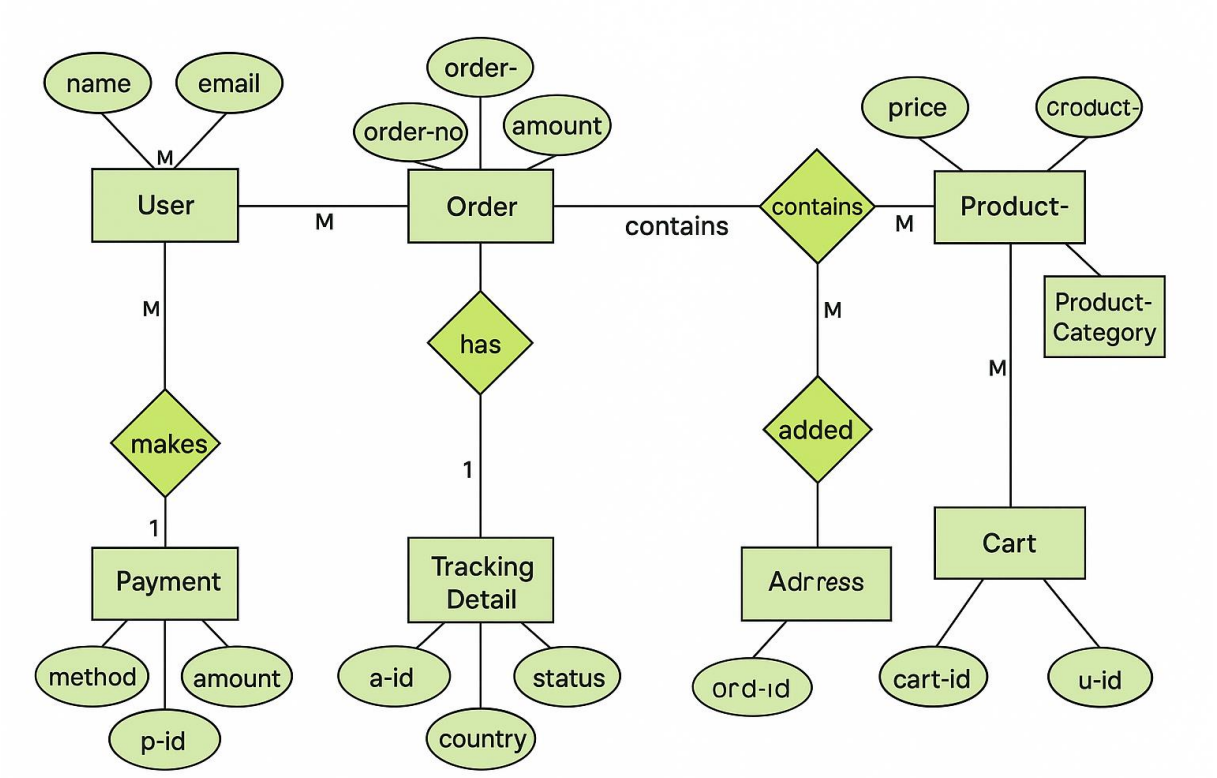


Fig.3.1: ER Diagram

3.2 Data Dictionary

A data dictionary, or metadata repository, as defined in the IBM Dictionary of Computing, is a "centralized repository of information about data such as meaning, relationships to other data, origin, usage, and format". Oracle defines it as a collection of tables with metadata.

Table Name	Field	Data Type	Description
users	user_id	ObjectId	Primary key
	name	String	User full name
	email	String	Login email
	password	String	Hashed password
food_items	item_id	ObjectId	Primary key
	name	String	Food item name
	price	Number	Price of the item
orders	order_id	ObjectId	Primary key
	user_id	ObjectId	Reference to users table
	total_amount	Number	Total price of the order
	status	String	e.g., pending, delivered

3.3 Database Normalization

Database normalization is the process of organizing data to reduce redundancy and improve data integrity. Below is how your Foodie project's database can be normalized through the first three normal forms

First Normal Form (1NF)

- Ensures that each column contains atomic values (no repeating groups or arrays).
- **Example Fix**: Instead of storing multiple food items in one column of an order, create a separate table to store them individually.
- **Before (Not 1NF):**

order_id	user_id	food_items	total_amount
001	U01	"Pizza, Burger"	500

- **After (1NF):**

order_id	user_id	food_item	total_amount
001	U01	Pizza	500

Second Normal Form (2NF)

- Achieved when it is in 1NF and all non-key attributes are fully dependent on the primary key.
- Split data into multiple related tables to eliminate partial dependencies.

Fix:

- Move food item details into a separate Food Item table.
- Only store food item references in the order.

Third Normal Form (3NF)

- Achieved when it is in 2NF and there are no transitive dependencies.
- Non-key attributes must only depend on the primary key — not on other non-key attributes.

Fix:

- If you have a field like category_name in the FoodItem table, move it to a separate Category table and reference it with category_id.

Final Tables After Normalization:

- **User** (user_id, name, email, password)
- **FoodItem** (item_id, name, price, category_id)
- **Category** (category_id, name)
- **Order** (order_id, user_id, total_amount, timestamp)
- **OrderItem** (order_item_id, order_id, item_id, quanti

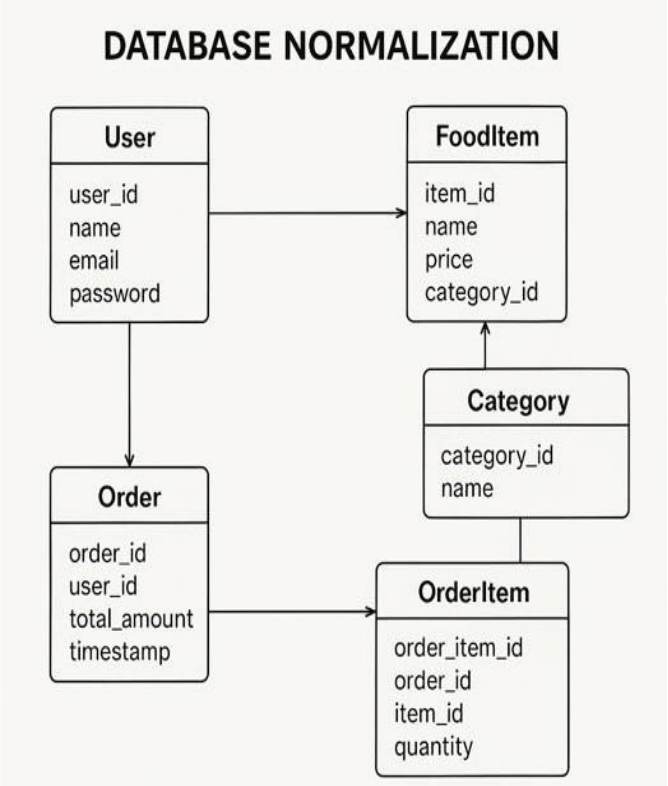


Fig.3.3: Database Normalization

High-Level Overview

The app is a basic **food ordering system** that includes:

- User-facing UI for browsing food items.
- Add to cart functionality.
- Checkout.
- Possibly admin panel for adding/editing products.

3.2 System Architecture

Users <--> Frontend (React) <--> Backend (Node.js / Express) <--> Database (MongoDB)
|
|-- Cloud Storage (for images - e.g., Cloudinary, S3)
|
|-- Payment Gateway (Stripe, Razorpay)
|
|-- Notification Service (Email/SMS)

Functional Requirements
These are the core features the app should support:
User Side

- Browse food items (menu)
- View food details
- Search & filter items (by category, price, etc.)
- Add to cart
- Checkout (with payment)
- View past orders

Admin Side

- Add/edit/delete food items
- Manage orders
- View sales stats

Optional Features

- User registration/login
- Coupons and discounts
- Real-time order tracking
- Email notifications

Frontend (React App)
Framework:

- **React.js** (or **Next.js** if SEO and SSR is needed)

Libraries:

- **React Router** – for routing between pages
- **Axios** – for API calls
- **Tailwind CSS / Bootstrap** – for styling
- **Redux / Zustand / Context API** – for state management

Components to build:

- Navbar: Home, Cart, Login
- FoodList: Shows all items
- FoodCard: Each food item preview
- FoodDetail: Detailed view on click
- Cart: Item list + quantities + totals
- Checkout: Payment page
- Login/Register: Auth forms
- AdminDashboard: Manage products & orders

Backend (API Server)
Framework:

- **Node.js + Express**

Structure:
bash
CopyEdit
/routes
/auth.js // login, register
/foods.js // list, get, add, edit, delete foods
/orders.js // place order, get orders
/cart.js // cart logic
/controllers

Key APIs:
Auth:

- POST /api/register – Create user
- POST /api/login – Login + return JWT

Foods:

- GET /api/foods – List all foods
- GET /api/foods/:id – Get one food
- POST /api/foods – (Admin) Add food
- PUT /api/foods/:id – (Admin) Edit food
- DELETE /api/foods/:id – (Admin) Delete food

Orders:

- POST /api/orders – Place an order
- GET /api/orders/:userId – Get past orders
- GET /api/admin/orders – (Admin) All orders

Authentication
Use JWT:

- On login/register, issue a token
- Store token in localStorage or httpOnly cookie
- Use middleware to protect admin and user routes

Payment Integration
Use Stripe or Razorpay:
Client-side:

- Use Stripe JS SDK
- Create a payment session
- Redirect to Stripe checkout

Backend:

- Use Stripe SDK to create checkout sessions
- Handle webhook for payment confirmation
- Update order status to paid

Image/File Storage
Food images need hosting:

- **Cloudinary**: Easy image upload, free plan
- **AWS S3**: More control, good for production
- Upload image from Admin Panel → get URL → save in DB

Deployment
Frontend:

- **Netlify**, **Vercel**, **Firebase Hosting**

Backend:

- **Render**, **Railway**, **Heroku**, **AWS EC2**

Database:

- **MongoDB Atlas** (cloud)
- **PlanetScale** or **Supabase** (for SQL)

DevOps & Monitoring

- **GitHub/GitLab CI** – for auto deployment
- **Sentry** – crash/error tracking
- **Postman** – test APIs

Optional Advanced Features

- **PWA**: Make the app installable on mobile
- **Admin analytics**: Graphs of sales/orders
- **Live status tracking**: Using Socket.IO
- **Push notifications**: For order status

User Experience (UX) Workflow
Let's define **how users interact** with your system.
Regular User Workflow:

1. **Visit the homepage** – see food items, banners, etc.
2. **Browse food** – search by name, filter by category, price.
3. **Click on an item** – view its details.
4. **Add to cart** – choose quantity.
5. **View cart** – modify or remove items.
6. **Checkout** – enter delivery info, make payment.
7. **Receive confirmation** – order placed, receive email/SMS

Admin Workflow:

1. **Login as admin**
2. **View dashboard** – orders, revenue, top products.
3. **Add/edit/delete products**
4. **Mark orders as dispatched/deliver**

3.4 Architecture: Clean Layers

1. Frontend (React) → UI/UX rendering
2. Backend (Node.js) → Business logic and APIs
3. Database (Mongo/Postgres) → Data persistence
4. External Services → Stripe, Cloud-nary, Email, etc.

Frontend
Tools:

- **React** + **React Router**
- **Tailwind CSS** (modern utility styling)
- **Axios** – for API calls
- **Context API / Redux Toolkit** – for global state
- **React Hook Form / Formik** – for managing forms

Component Tree:
<App>
├── Navbar
├── Home (FoodList)
├── FoodDetail
├── Cart
├── Checkout
├── Login/Register
└── AdminDashboard

Cart Logic:

- Store in localStorage or global state
- Maintain: [{ foodId, name, price, quantity }]
- Compute total = sum(qty × price)

Backend

/server

```
├── /routes       ← API endpoints
├── /controllers ← Business logic
├── /models       ← Mongoose/Postgres schemas
├── /middleware   ← Auth, error handlers
└── server.js     ← App entry point
```

Controllers:

- authController: register/login
- foodController: CRUD food
- orderController: create/view/update orders
- adminController: sales stats

Middleware:

- authMiddleware – verify JWT
- adminMiddleware – check user role
- errorMiddleware – catch errors globally

External Services

1. Cloudinary (for image uploads)
   - Upload images via admin panel
   - Get URL and save in DB
   - Serve optimized images

2. Email (Nodemailer + Mailgun/SendGrid)

- Send order confirmations
- Admin alerts for new orders

3. Twilio

- SMS for order updates

4. Socket.IO

- Real-time order tracking

Security Best Practices

- Hash passwords with bcrypt
- Use helmet, rate limiter, cors
- Protect API routes with JWT
- Validate all incoming data (express-validator)
- Sanitize image uploads

Usability Testing

Focus on user experience:

- Is the interface intuitive? Can someone use it without instructions?
- Are button labels and form fields clear?
- Does feedback appear after actions (like “Item Deleted” or “Item Added”)?

Accessibility Testing

Make the app usable for all users:

- Use Wave or Lighthouse Accessibility audit.
- Are form fields labeled?
- Is contrast between text and background sufficient?
- Can you navigate the app using just the keyboard?

Error & Edge Case Testing
Try things that might break the app:

- Add duplicate food items.
- Enter invalid characters or extremely long names.
- Submit the form with no input.
- Try deleting all items — what happens?

API & Backend Integration Testing
In the Network tab (DevTools), inspect API calls:

- Are GET/POST/DELETE requests working?
- Are you handling failed requests (e.g., show error if server is down)?
- Are there loading spinners or messages during fetch?

Security Checks (Basic)

- Are any secrets (API keys, credentials) exposed in the front-end code?
- Are form submissions validated before sending to backend?
- Is there any console log leaking sensitive data?

Cross-Browser Testing
Test on multiple browsers:

- Google Chrome
- Mozilla Firefox
- Microsoft Edge
- Safari (if possible)

CHAPTER 4

TESTING

4.1 Testing Approach

4.1.1 Purpose of Testing

The goal of testing in the Foodie project is to ensure that all features—user registration, food item browsing, cart functionality, order placement, and admin operations—work as expected, with minimal bugs and maximum reliability.

4.1.2 Types of Testing Applied

1. Unit Testing
    - What it tests: Individual components (e.g., React components like Menu, Cart, or backend routes such as /API/orders).
    - Tools used: Jest, React Testing Library, Mocha (optional for backend).
    - Example: Testing whether the Add to Cart button correctly updates the cart state.
2. Integration Testing
    - What it tests: How multiple modules work together (e.g., placing an order involves frontend → backend → database).
    - Focus Areas:
        - User Login & Registration
        - Order processing
        - Admin item updates
3. System Testing
    - What it tests: The complete system as a whole.
    - Scenario: Simulating real-world user flow from login to checkout and verifying the backend logic and database updates.
4. User Acceptance Testing (UAT)
    - What it tests: Verifies if the system meets user requirements.
    - Participants: Developers and testers simulate end-users (both customers and admin).
    - Outcome: Feedback collected was used to improve user interface and feature clarity.

4.1.3 Testing Summary

Feature	Test Type	Status
User Registration/Login	Unit + System	Passed
Menu Viewing	Integration	Passed
Cart & Order Placement	Integration	Passed
Admin Management	System	Passed
Data Consistency	Manual + API	Passed

User Interface (UI) Testing
Check the appearance and layout of each component:
Fonts, colors, buttons, and spacing are consistent.
Navigation menu is styled properly and highlights the current page.
All images load properly (food items, profile avatar, etc.).
Table elements (like List Items) align properly on all screen sizes.

Functionality Testing
Verify that core features work as expected:
Add Items Page

- Can you add a new food item with all required fields filled?
- What happens if you leave a field blank or enter invalid data.
- Are errors shown clearly.

List Items Page

- Do all existing food items display correctly?
- Can you delete an item by clicking the "X" button?
- Is the deleted item removed from the database/state and UI updated?

Orders Page

- Are order entries visible (if applicable)?
- Can you change or update order status?
- Are payment or total fields accurate.

Responsiveness Testing
Test on multiple devices and screen sizes:

- Desktop: Resize browser window.
- Tablet/Mobile: Use Chrome DevTools > Toggle Device Toolbar.
- UI adjusts properly for all sizes (sidebar collapses, buttons realign, tables resize).
- Test orientation changes (portrait ↔ landscape).

Performance Testing
Use tools like **Lighthouse** (built into Chrome DevTools):

- **Page load speed**: Should ideally load under 3 seconds.
- **Image optimization**: Are images properly compressed?
- **JavaScript performance**: Are bundles too large?

CHAPTER 5

SYSTEM IMPLEMENTATION

5.1 Output Screen

Home Page:

The Home Page of the Foodie food ordering system serves as the entry point for users, providing a clean, intuitive, and visually appealing interface. It is designed to immediately capture the user's attention with high-quality images, engaging text, and a user-friendly navigation bar.
Key elements featured on the Home Page include:

- Navigation Bar: Positioned at the top, it provides quick access to major sections like Home, Explore Menu, Cart, Login/Signup, and Order.
- Hero Section: Showcases a featured banner image or promotional content, often highlighting popular dishes or current deals.
- Explore Button: Encourages users to browse the available menu and start the ordering process.
- Responsiveness: The layout adjusts seamlessly across different screen sizes (desktop, tablet, and mobile), enhancing accessibility and user experience.

The Home Page acts as a dynamic gateway, setting the tone for the rest of the platform by combining functionality with modern design aesthetics.

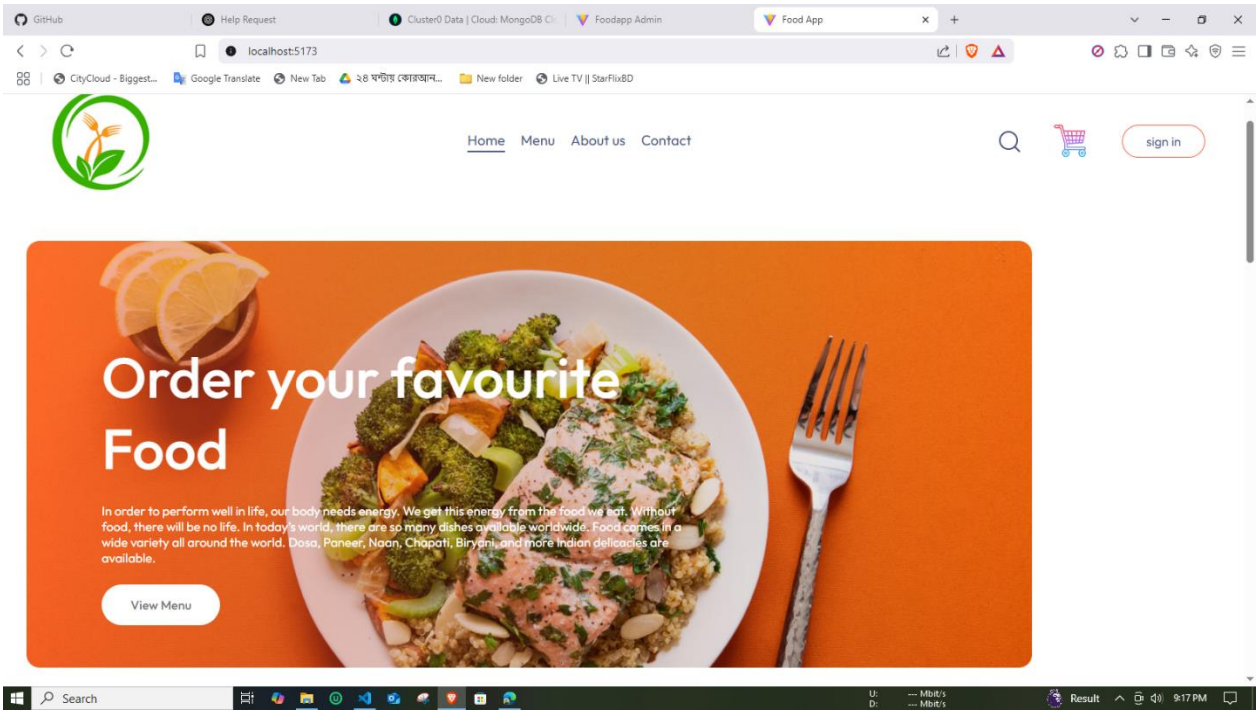


Fig.5.1.1: Home Page

Explore menu:

The Explore Menu section of the Foodie web application is designed to offer users a seamless and engaging way to browse available food items. It plays a central role in the user experience, acting as the digital menu interface for the platform.

Key Features:

- Category-Based Filtering: Users can easily switch between categories such as pizza, burger, drinks, and more, allowing for quick discovery of preferred food types.
- Visual Food Cards: Each item is displayed in a card layout, featuring:
    - A high-quality image of the food
    - The item names
    - Price
    - “Add to Cart” button for instant ordering.

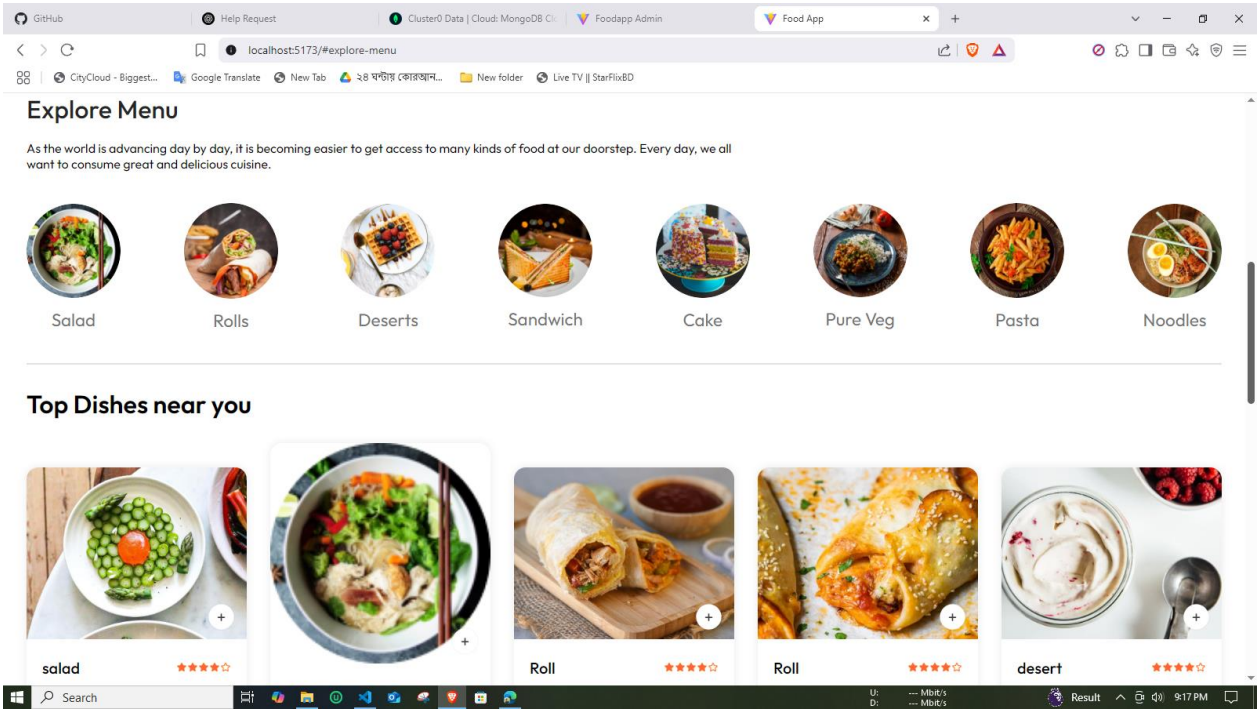


Fig.5.1.2: Explore Menu

Menu:

The Menu section of the Foodie application serves as the core component where users can view the complete list of available food items offered by the platform. It is a structured and user-friendly interface that displays all dishes in an organized format, allowing users to make quick and informed ordering decisions.

Comprehensive Display: The menu showcases all food items regardless of category, providing users with an overview of everything offered on the platform.

Visual Layout:

- Each food item includes an image, title, price, and an "Add to Cart" button.
- Items are arranged in a grid layout, making it easy to scan through multiple options at once.

Interactive Features:

- Users can add or remove items directly from this page.
- Quantities update in real time as users interact with the cart.

Database Connectivity:

- All menu items are pulled dynamically from the backend database (MongoDB), ensuring any admin-side updates (additions, edits, deletions) are reflected in real time.

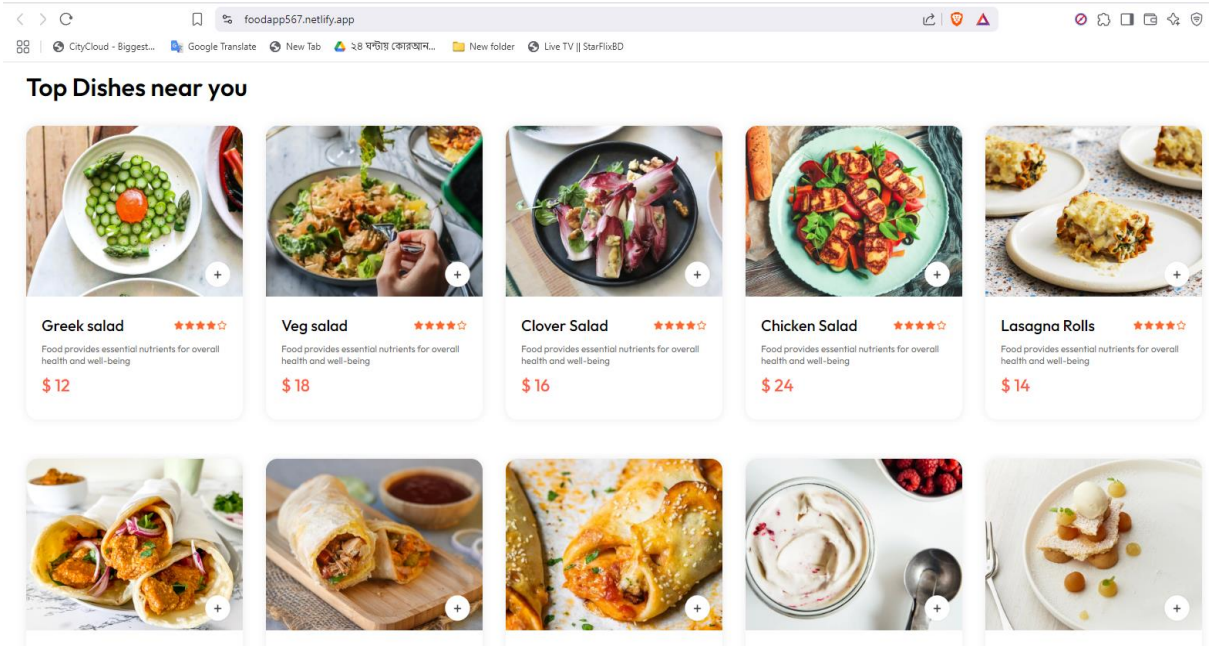


Fig.5.1.3: Menu

Footer:

The Footer section of the Foodie web application appears consistently at the bottom of every page, providing users with quick access to essential information and enhancing the website's overall structure and usability.

Brand Logo: Reinforces the identity of the Foodie platform and ensures brand visibility pages.

Quick Links: Includes navigation links to key sections such as Home, Explore Menu, Cart, and Contact (if available).

Social Media Icons: May include icons that link to the brand's social profiles or placeholders for future integration.

Copyright Notice: Displays the current year and credits the development team or institution

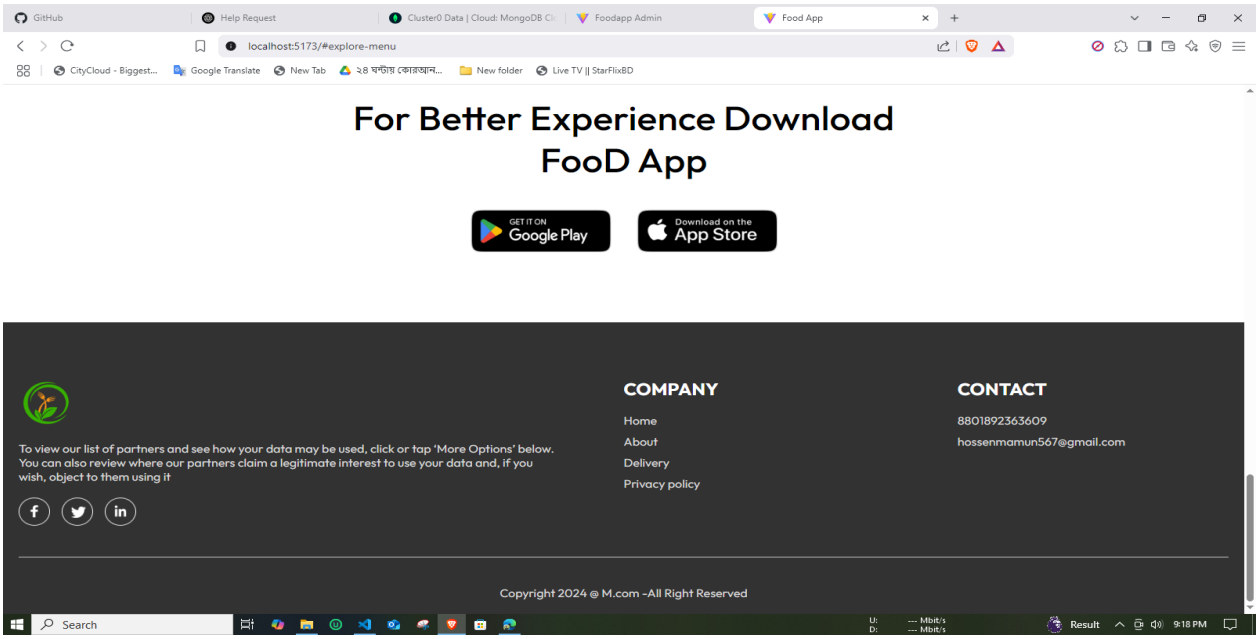


Fig.5.1.4: Footer

Sign Up and LogIn:

This figure illustrates the sign up and login interfaces of the Foodie food ordering system. Both forms are designed with simplicity and responsiveness, ensuring a smooth onboarding process for users. The sign-up form collects the user's name, email, and password to create an account, while the Login form authenticates existing users using their email and password.

Key UI elements shown in the figure include:

- Clean, centered layout with input fields and action buttons
- Real-time validation feedback for missing or incorrect inputs
- Navigation links between the Sign Up and Login forms
- A responsive design that adapts to both desktop and mobile views

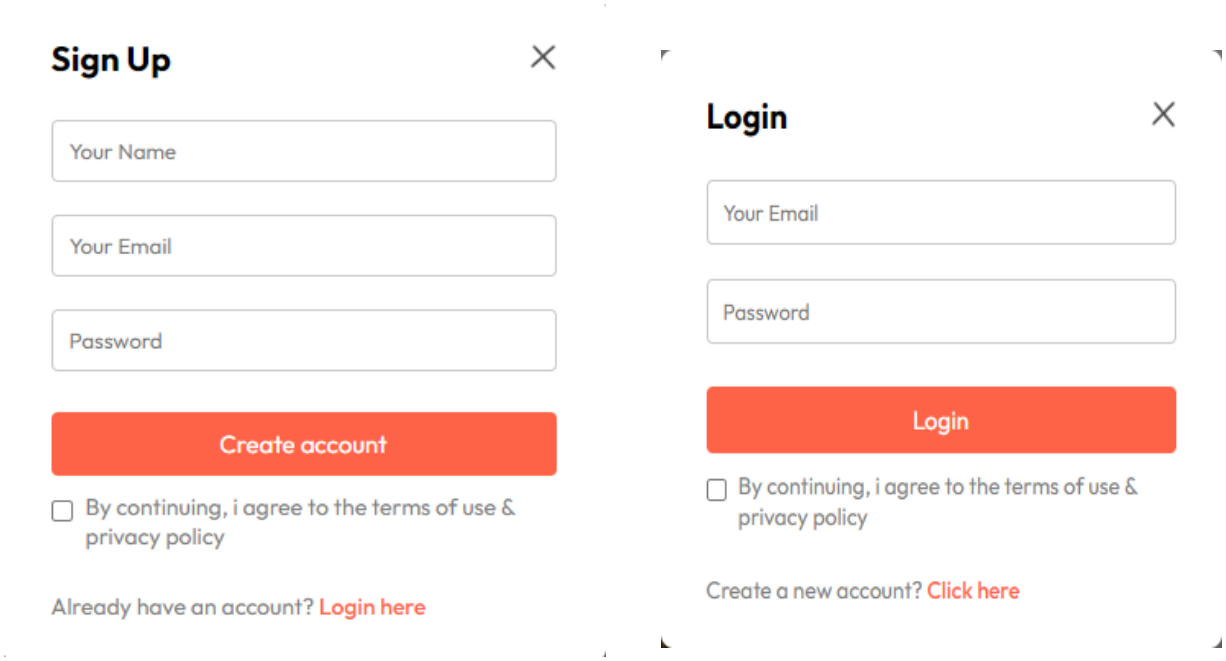


Fig.5.1.5: Sign Up & Login

Cart Item:

The Cart Item section in the Foodie web application serves as the intermediate stage between browsing food items and placing an order. It allows users to review, modify, and finalize their selections before checkout.

Item Overview: Displays all food items added to the cart, including:

- Food image
- Item name
- Unit price
- Quantity selected
- Subtotal per item

Interactive Controls:

- Users can remove items directly from the cart
- Quantity is shown clearly and dynamically reflects changes

Cart Totals:

- Automatically calculates subtotal, delivery fee, and total cost
- Delivery fee is conditionally applied.

Proceed to Checkout Button:

- Redirects users to the Place Order page to complete the transaction

Promo Code Field:

- Includes an input for users to enter promotional codes.

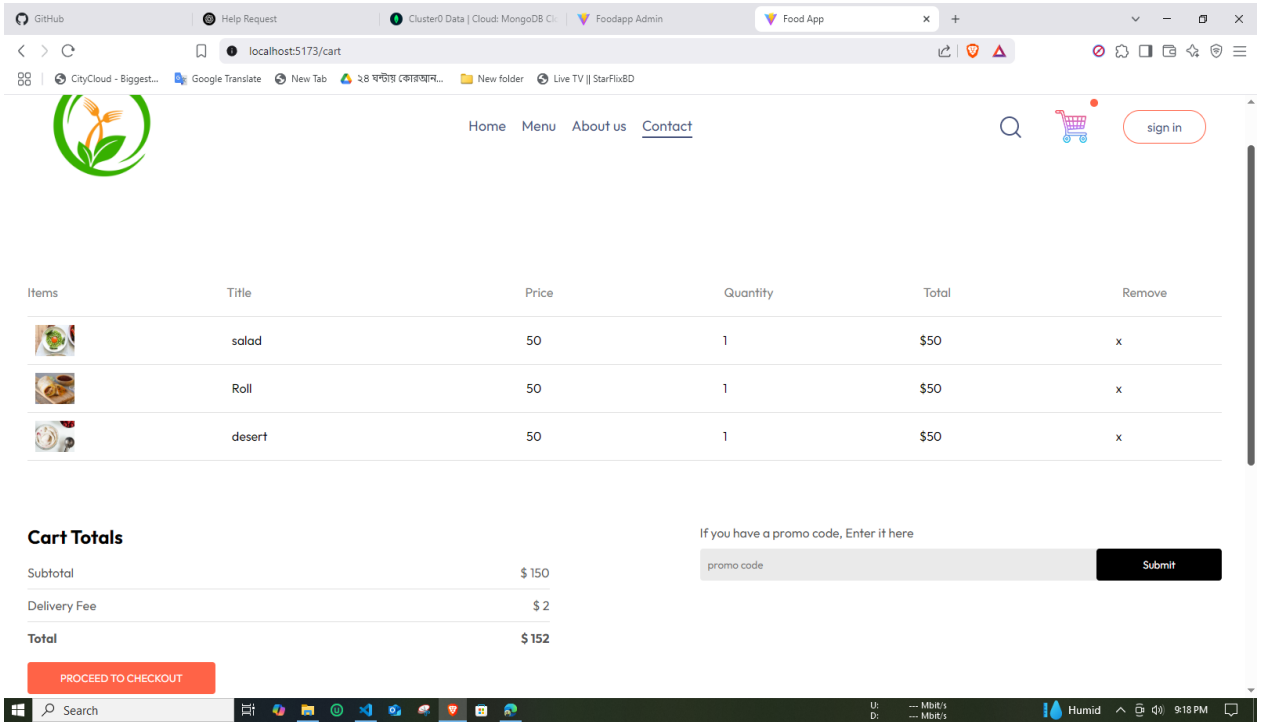


Fig.5.1.6: Cart Item

Customer Information:

Before finalizing an order in the Foodie web application, users are required to provide essential information to ensure accurate and timely delivery. This step occurs on the Place Order page and acts as a crucial checkpoint between the cart and order confirmation.

Required User Inputs:

- Full Name: Identifies the recipient of the order
- Email Address: Used for order confirmation and potential future communication
- Phone Number: Allows delivery agents or admins to contact the customer if needed
- Delivery Address: Specifies the location where the order will be delivered
- Payment Method: Users can select their preferred method (UI placeholder for future integration)

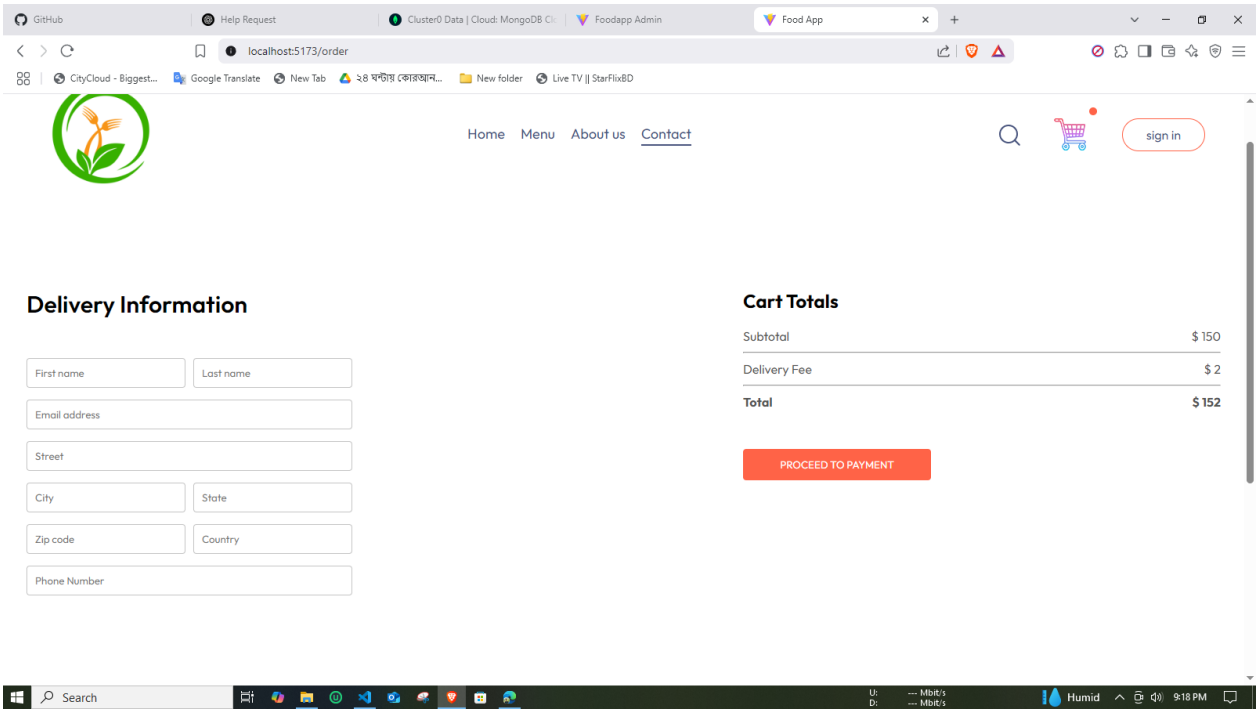


Fig.5.1.7: Customer Information

Cart Item Page:

This figure shows the Cart Page of the Foodie application, where users can review and manage their selected food items before placing an order.

Cart Table:

Items: Thumbnail image of the selected food item
Title: The name of the food item ("Greek salad")
Price: The price per unit ($12)
Quantity: Number of items added to the cart (1)
Total: Computed total cost per item ($12)
Remove Option: A clickable “x” button to remove the item

Cart Totals Section:

Subtotal: Automatically calculated as the sum of item prices
Delivery Fee: A flat fee ($2) added if the cart is not empty
Final Total: Combined total cost ($14)

Promo Code Input: A field where users can enter a promotional discount code (placeholder, currently non-functional)
Proceed to Checkout: A prominent button that navigates the user to the Place Order page.

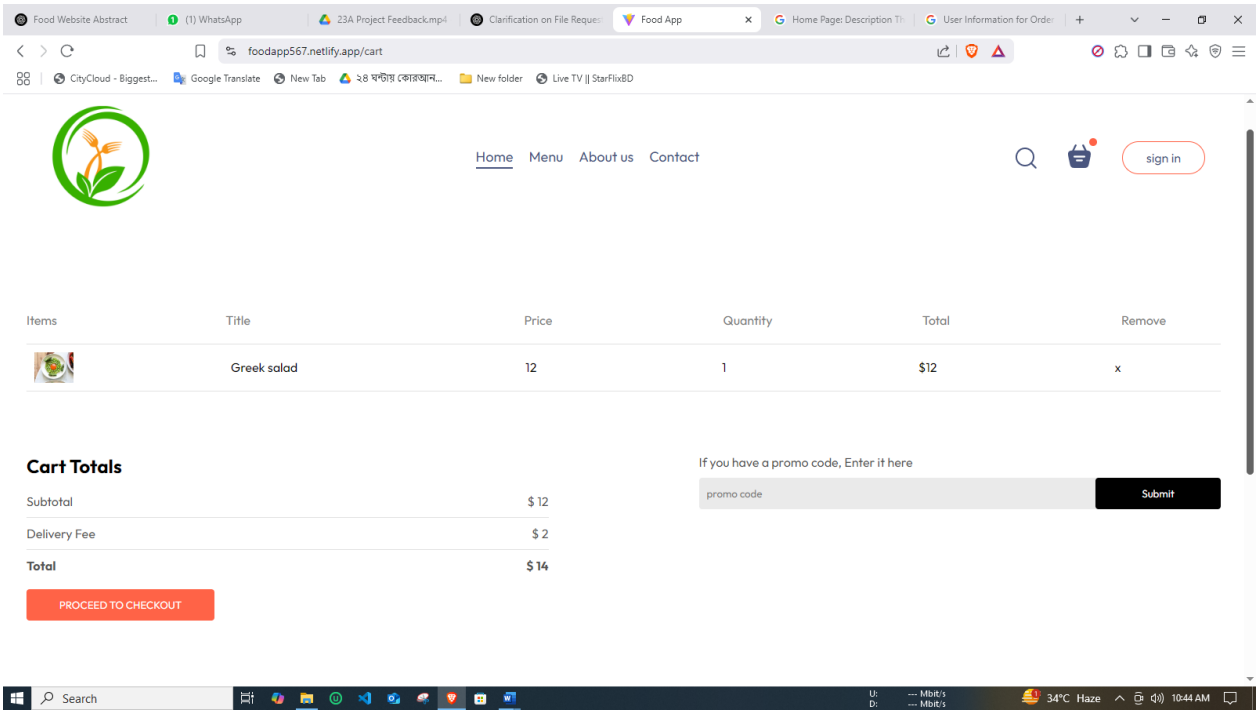


Fig.5.1.8: Cart Item Page

Admin Panel:

The Orders page provides administrators with a centralized interface to view, manage, and update customer orders. This panel is designed for efficiency and clarity, offering a comprehensive table of all order records with the following key features:

Order List Table: Displays all orders with columns such as Order ID, Customer Name, Order Date, Status, Total Amount, and Payment Method.

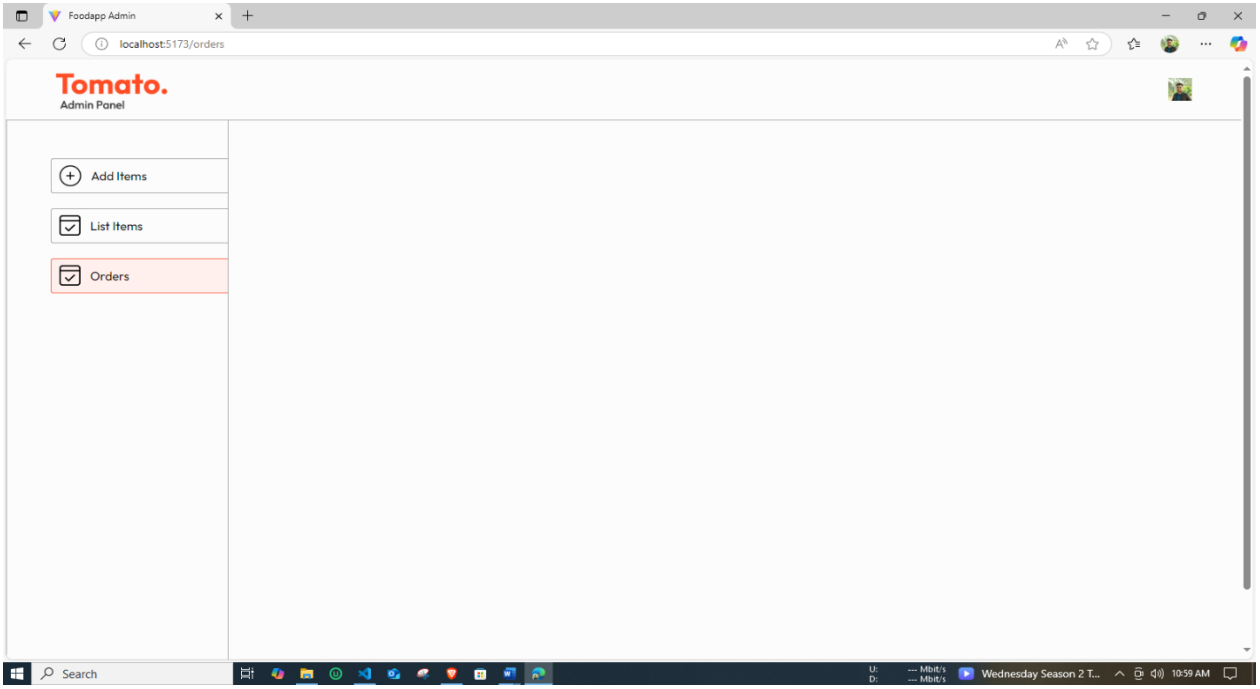


Fig.5.1.9: Admin pane

Add Item:

The Add Item section allows administrators or staff to manually add products or services to a customer's order. It typically includes input fields for the item name, quantity, and unit price.Typically, in food ordering applications, the "Add Item" feature allows users to select food items from a menu and add them to their cart for ordering. This process usually involves choosing the desired item, specifying quantity, and sometimes selecting additional options like size or toppings.

Item Name: Text input for the product or service name.
Quantity: Numeric input to specify how many units are being ordered.
Price: Numeric input for the unit price or cost of the item.
Add Button: A button to confirm and append the item to the current order.

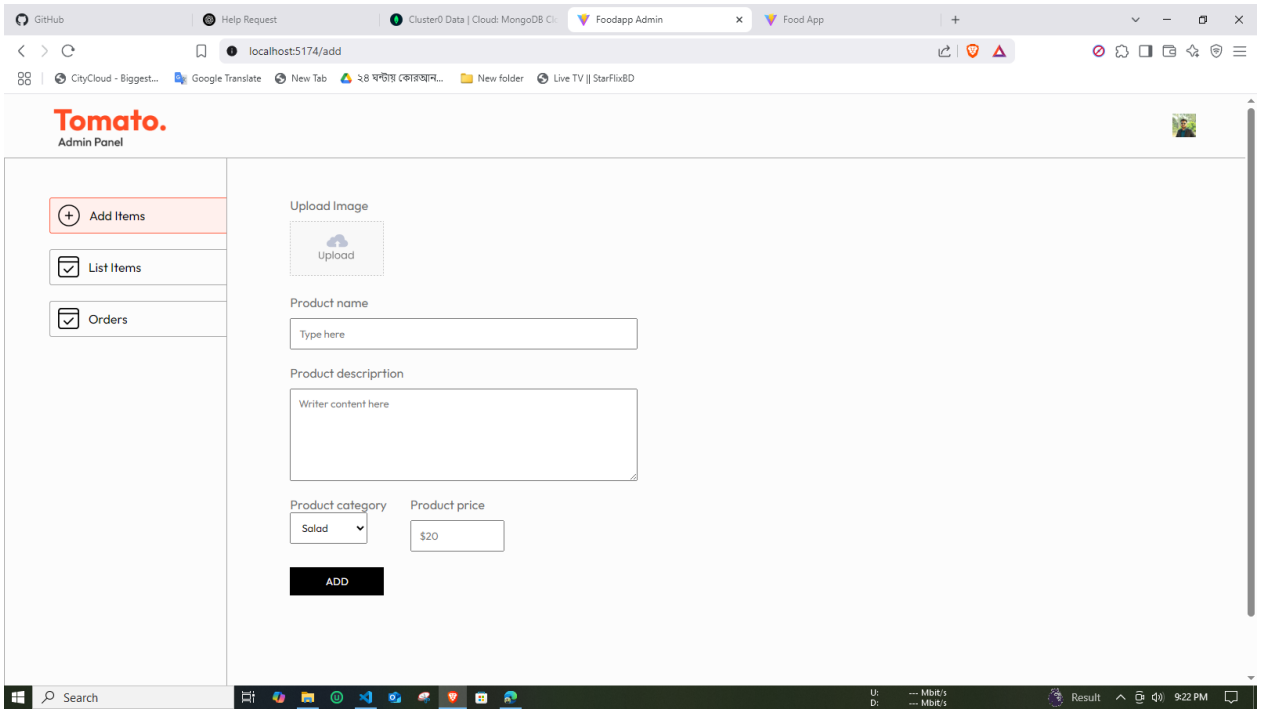


Fig.5.2.0: Add Items

List Item:

The List Items page displays all food items currently available in the system. It is designed to help administrators easily view, manage, and remove items from the menu. This interface is structured as a clean and simple table with the following columns:

- Image: A thumbnail preview of the food item for visual identification.
- Name: The name of the food item (e.g., Salad, Roll, Cake).

- Category: The category to which the item belongs (currently all are listed under "Salad"—may need updating).
- Price: The price of the item.
- Action: A delete button (X) allowing the admin to remove the item from the list.

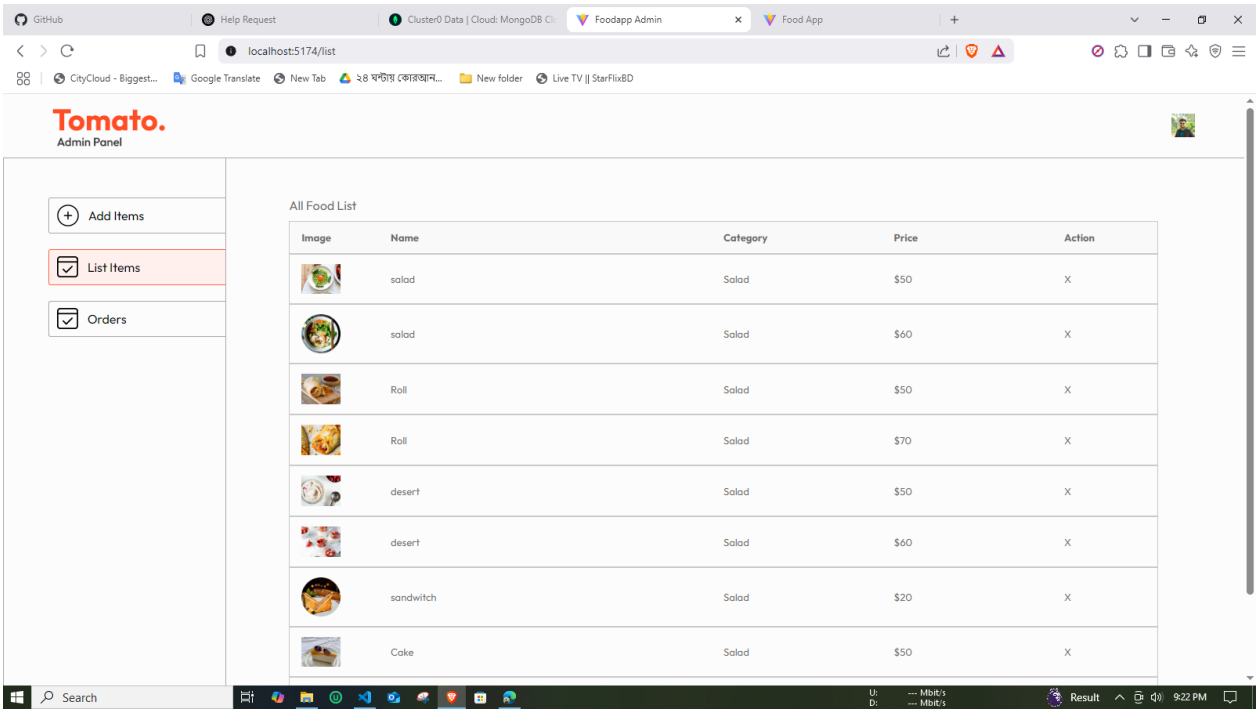


Fig.5.2.1: List Items

CHAPTER 6

CONCLUSION AND FUTURE WORKS

6.1 Conclusion

The Foodie food ordering system has been successfully developed and tested. The platform enables users to browse a menu, add food items to the cart, place orders, and allows administrators to manage menu items and view customer orders. The system was built using React.js, Node.js, Express.js, and MongoDB, making it scalable, responsive, and suitable for modern web deployment. All core functionalities have been tested through unit, integration, and system-level testing. The application is user-friendly, easy to navigate, and efficiently performs the required operations.

6.2 Limitation

Despite successful implementation, the system currently has the following limitations:

- No Email Verification during registration.
- No SEO Optimization for better discoverability.
- No SMS Alert functionality for order updates.
- Security Risks like lack of advanced role-based access control (RBAC) or encryption.

6.3 Future Scope

The Foodie system can be enhanced further by adding new features and expanding its use in broader domains: Email & SMS Notifications to keep users updated with real-time order status. SEO Enhancements to improve visibility on search engines. Mobile App Integration using React Native for better user reach. Delivery System Integration to manage delivery agents and tracking. Enhanced Security through JWT refresh tokens, encryption, and RBAC.

CHAPTER 7

REFERENCES

[1] Ijraset. (n.d.). *Online food ordering system*. IJRASET. https://www.ijraset.com/research-paper/online-food-ordering-system.

[2] Khairunnisa K., Ayob J., Mohd. Helmy A. Wahab, M. Erdi Ayob, M. Izwan Ayob, M. Afif Ayob, "The Application of Wireless Food Ordering System" MASAUM Journal of Computing 2009.

[3] DigitalOrdering System for Restaurant UsingAndroid Ashutosh Bhargave, Niranjan Jadhav, Apurva Joshi, Prachi Oke, Prof. Mr. S. R Lahane Department of Computer Engineering, GES's RHSCOE, International Journal of Scientiic and Research Publications, Volume 3, Issue 4, April 2013.

[4] The Application of Wireless Food Ordering System, Khairunnisa K., Ayob J., Mohd. Helmy A. Wahab, M. Erdi Ayob, M. Izwan Ayob, M. Aif Ayob, MASAUM Journal of Computing, Volume 1 Issue 2, September 2009

[5] Noor Azah Samsudin, Shamsul Kamal Ahmad Khalid, Mohd Fikry Akmal Mohd Kohar,Zulkifli Senin, Mohd Nor Ihkasan," A customizable wireless food ordering system with real time customer feedback", IEEE Symposium on Wireless Technology and Applications(ISWTA) 2011

[6] T.-P. Liang, et al., "Adoption of mobile technology in business: a fitviability model," Industrial Management & Data Systems, vol. 107, pp. 1154-1169, 2007

[7] D. Zhao, et al., "Generating mobile device user interfaces for diagrambased modelling tools," in Seventh Australasian User Interface Conference, Hobart, Australia, 2006, pp. 91-98.

[8] S. N. Cheong, et al., "Design and Development of Multi-Touchable ERestaurant Management System," in IEEE Proceeding of 2010 International Conference on Science & Social Research, 2010, pp. 686- 691.

[9] M. Li, J. Zhang and W. Wang, "Task Selection and Scheduling for Food Delivery: A Game-Theoretic Approach," 2018 IEEE Global Communications Conference (GLOBECOM), pp. 1-6.

[10] Z. Cahyani, R. Nurcahyo and Farizal, "Popularity Analysis of Mobile Food Ordering Apps InIndonesia," 2020 IEEE 7th International Conference on Industrial Engineering and Applications(ICIEA), Bangkok, Thailand, 2020, pp. 1000-1004.

[11] Leong, W. H. (2016). Food Ordering System Using Mobile Phone (Doctoral dissertation, UTAR)

[12] Vincent Cheow Sern Yeo, See-Kwong Goh, Sajad Rezaei, "Consumer experience, attitude and behavioral intention towards online food delivery (OFD) service" Journal of retailing and consumer services, Vol 35, (2017) 150-162