

Online Food Delivery System

A Project Report for Industrial Training and Internship submitted by

Deep Patra

Chanchala Show

Neha Show

In the partial fulfillment of the award of the degree of

BCA

in the

[DURGAPUR INSTITUTE OF MANAGEMENT & SCIENCE]

Of

[BENGAL COLLEGE OF ENGINEERING & Technology]

At



Ardent Computech Pvt. Ltd





CERTIFICATE FROM SUPERVISOR

This is to certify that **Deep Patra, Chanchala Show, Neha Show, 32442723035, 32442723032, 32442723058** have completed the project titled Online Food Delivery system under my supervision during the period from 26/06/2025 to 10/08/2025 which is in partial fulfillment of requirements for the award of the **BCA** degree and submitted to the Department of DIMS of **Bengal College Of Engineering & Technology**.

Signature of the Supervisor

Date: 10/08/2025

Name of the Project Supervisor: Subhojit Santra





BONAFIDE CERTIFICATE

Certified that this project work was carried out under my supervision

Online Food Delivery System is the bonafide work of

Name of the student : Deep Patra

Signature: D. Patra

Name of the student : Chanchala Show

Signature: C. Show

Name of the student: Neha show

Signature: N. Show

SIGNATURE:

Name :

PROJECT MENTOR

SIGNATURE

Name:

EXAMINERS

Ardent Original Seal



ACKNOWLEDGEMENT

The achievement that is associated with the successful completion of any task would be incomplete without mentioning the names of those people whose endless cooperation made it possible. Their constant guidance and encouragement made all our efforts successful.

We take this opportunity to express our deep gratitude towards our project mentor **Subhojit Santra** for giving such valuable suggestions, guidance and encouragement during the development of this project work.

Last but not the least we are grateful to all the faculty members of **Ardent Computech Pvt. Ltd.** for their support.

CONTENT PAGE

1. COMPANY PROFILE	1
2. INTRODUCTION	2
2A.OBJECTIVE	3
2B.SCOPE	4
1. User Registration & Login :	4
2. Money browsing:	4
3. Order Placement:	4
4. Admin Panel :	4
5. Payment Integration :	4
6. Responsible Design:	
3. SYSTEM ANALYSIS	5
3A.IDENTIFICATION OF NEED	6
1. Access To Food Service:	6
2. Streamline Restaurant Operation	6
3.Tracking & Secure Payment	
3B.FEASIBILITY STUDY	7
3C.WORKFLOW	8
Waterfall Model Design:	8
Iterative Waterfall Design:	8
▪ Advantages:	9
▪ Disadvantages:	9
▪ Applications:	10
3D .STUDY OF THE SYSTEM	11
• Register:	11
• Verify account:	11
• Login	11
• Home Page:	11
• Restaurant & Menu	11
• Orders:	11
• About Page:	11
• Account:	11
2. Admin Interface:	11
3E.INPUT AND OUTPUT	12
INPUT:	12
OUTPUT:	12
3F.SOFTWARE REQUIREMENT SPECIFICATIONS	13
The developer is responsible for:	13
Functional Requirements:	13
A. User Registration & Authentication	13

B. Browse and Search	13
c. Menu & Order Management	13
D. Admin Controls	13
Hardware Requirements:	13
Software Requirements:	13
3G.SOFTWARE ENGINEERING PARADIGM APPLIED	14
4. SYSTEM DESIGN	15
4A. DATA FLOW DIAGRAM	16
DFD Notation:	17
DFD Example:	17
Database Input Output	17
Rules for constructing a Data Flow Diagram	18
• LEVEL 0 DFD OR CONTEXT DIAGRAM:	18
• LEVEL 1 DFD:	19
• LEVEL 1 DFD:	20
4B.SEQUENCE DIAGRAM	21
How to draw Use Case Diagram?	25
4D.SCHEMA DIAGRAM	27
5. UI SNAPSHOT	28
❖ FRONTEND :	28
✓ CODE	28
Verify Account Page:	31
✓ CODE	31
CODE :	32
3) Food Listing PAGE(for admin and user)	34
• components – ProductCard.jsx :	35
4) Dish Detail And Ordering Interface (for user);	37
✓ CODE	37
5) Dashboard PAGE:	39
6) Service Page:	40
✓ CODE:	41
7)ADD FOOD: ADMIN Panel For Listing New Items	42
8) CART PAGE:	46
9) SECURE CHECKOUT:	50
✓ CODE	53
10) PAYMENT GATEWAY:	54
✓ CODE:	60
❖ BACKEND:	61
• Database - db.js:	61
2)FOOD DATA:	62
• model – Food.js :	62

10. CONCLUSION	63
-----------------------	-----------

11. BIBLIOGRAPHY	64
-------------------------	-----------

1. COMPANY PROFILE

ARDENT (Ardent Computech Pvt. Ltd.) is a certified software development and training company operating in India since 2003. Recently, it merged with ARDENT Technologies to expand its global presence.

ARDENT Technologies

ARDENT Technologies offers IT services in the UK, USA, Canada, and India. It specializes in custom software solutions, covering system analysis, design, development, deployment, and training. The company also provides consultancy and electronic security services to clients across education, entertainment, hospitality, telecom, media, and other industries.

ARDENT Collaborations

ARDENT Collaborations is the training and R&D wing of the company. It conducts professional programs for students and professionals from technical and management backgrounds, including B.Tech, M.Tech, MBA, MCA, BCA, and MSc. Training options include summer, winter, and industrial formats. Outstanding performers may receive stipends or scholarships.

Accreditations

ARDENT is affiliated with NCVT under DGET, Ministry of Labour & Employment, Govt. of India. It is ISO 9001:2015 certified and focuses on reducing the gap between academics and industry through practical training.

2. INTRODUCTION

In today's fast-paced digital world, convenience and efficiency have become key drivers of consumer behavior—especially in the food industry. With the growing demand for online services, food delivery platforms have transformed how people order and enjoy meals. This project, the Online Food Delivery App, aims to create a seamless and user-friendly experience for both customers and restaurant partners.

Built using the MERN stack (MongoDB, Express.js, React.js, Node.js), our application provides a robust and scalable solution for managing food orders, browsing menus, tracking deliveries, and handling payments. Customers can explore a variety of cuisines, place orders with ease, and monitor real-time delivery updates. On the other side, restaurants can manage their listings, update menus, and process incoming orders efficiently.

By combining intuitive design with powerful backend functionality, the app bridges the gap between food providers and consumers—offering a reliable, fast, and enjoyable way to satisfy hunger with just a few clicks.

2A.OBJECTIVE

The primary objective of the Online Food Delivery App is to simplify and enhance the food ordering experience by providing a fast, reliable, and user-friendly digital platform. It aims to connect customers with a wide range of restaurants, allowing them to browse menus, place orders, and track deliveries—all from the comfort of their homes.

The app is designed to streamline the entire process for both users and vendors, ensuring smooth communication, efficient order management, and timely service. By leveraging modern web technologies, the platform promotes convenience, supports local businesses, and meets the growing demand for on-demand food delivery.

Ultimately, the project seeks to offer a smart solution that brings delicious meals to users' doorsteps—quickly, safely, and effortlessly.

2B.SCOPE

Our project focuses on building a responsive web-based application that streamlines the food ordering and delivery process for users and restaurant partners.

1. User Registration & Login :

Secure sign-up/login for customers, restaurant owners, and admins.

2. Restaurant & Menu Browsing:

Users can explore restaurants, view menus, and filter items by cuisine, price, or rating.

3. Cart & Order Placement:

Customers can add items to cart, customize orders, and place them with ease.

4. Order Tracking:

Real-time updates on order status—from preparation to delivery.

5. Admin Panel:

Controls for managing users, restaurant listings, orders, and system analytics.

6. Payment Integration:

Support for secure online payments via cards, UPI, or wallets.

7. Responsive Design:

Accessible across desktops, tablets, and mobile browsers.

3. SYSTEM ANALYSIS

3A. IDENTIFICATION OF NEED

System analysis plays a vital role in developing our Online Food Delivery App, focusing on building a smooth, efficient, and user-friendly platform for both customers and restaurant partners.

With the rise of digital services and changing consumer habits, there is a growing demand for convenient, fast, and reliable food ordering solutions. Traditional food ordering methods—like phone calls or walk-ins—can be time-consuming and prone to errors. Our app addresses these challenges by offering a scalable digital solution that simplifies the entire process.

Key Needs Identified:

1. Convenient Access to Food Services

- Users need a platform to browse restaurants and place orders anytime, from anywhere.
- Busy lifestyles demand quick and hassle-free food delivery options.

2. Streamlined Restaurant Operations

- Restaurants require tools to manage menus, track orders, and communicate with customers efficiently.
- Digital dashboards reduce manual errors and improve service speed.

3. Real-Time Order Tracking & Secure Payments

- Customers expect live updates on their order status—from preparation to delivery.
- Secure and flexible payment options are essential for user trust and satisfaction.

3B.FEASIBILITY STUDY

The feasibility study of our Online Food Delivery App confirms that the project is practical, achievable, and valuable across multiple dimensions.

Technically, the app can be developed using the MERN stack—MongoDB, Express.js, React.js, and Node.js—which offers scalability, flexibility, and strong community support. The stack is well-suited for building responsive interfaces and handling real-time data, such as order tracking and user interactions

.Economically, the platform has strong potential for revenue generation through delivery charges, restaurant commissions, and promotional partnerships. It supports a sustainable business model with room for future expansion.

Operationally, the app is designed to be intuitive and efficient for both users and restaurant partners. It simplifies the ordering process, reduces manual errors, and enhances customer satisfaction through real-time updates and secure payments.

Legally, the project is viable with adherence to data protection regulations and food service compliance standards. Security measures are implemented to protect user information and transaction data.

With an estimated development timeline of 6 to 8 months, the project is well within a reasonable scope for completion, making it a promising solution in the fast-growing food tech industry.

3C.WORKFLOW

This document plays a vital role in the Software Development Life Cycle (SDLC) as it outlines the complete requirements of our Online Food Delivery App. It serves as a reference for developers during implementation and testing. Any future changes to the system will follow a formal approval process.

Waterfall Model Design:

The Waterfall Model is a linear and sequential development approach. Each phase must be completed before the next begins, with no overlap. It ensures clarity and discipline, making it suitable for structured projects like ours. In this model, the output of one phase becomes the input for the next.

Iterative Waterfall Design:

The Iterative Waterfall Model is a refined version of the traditional Waterfall approach. It divides development into manageable cycles, allowing for review and improvement before moving forward. This model combines the structured flow of Waterfall with the flexibility of iterative feedback.

Phases in the Iterative Waterfall Model:

- Requirement Gathering and Analysis

All functional and non-functional requirements—such as user login, restaurant listings, cart, order tracking, and payment—were documented.

- System Design

Based on the requirements, we designed the system architecture using the MERN stack. This included defining the database schema, frontend layout, and backend APIs.

- Implementation

Modules like authentication, menu browsing, and cart management were developed and tested individually (unit testing).

- Integration and Testing

All modules were integrated into a single system and tested thoroughly to ensure smooth performance and bug-free operation.

- Deployment

After successful testing, the app was deployed in a live environment, accessible across devices and browsers.

- Maintenance

Post-deployment issues were resolved through patches and updates. New features were added based on user feedback and evolving needs.

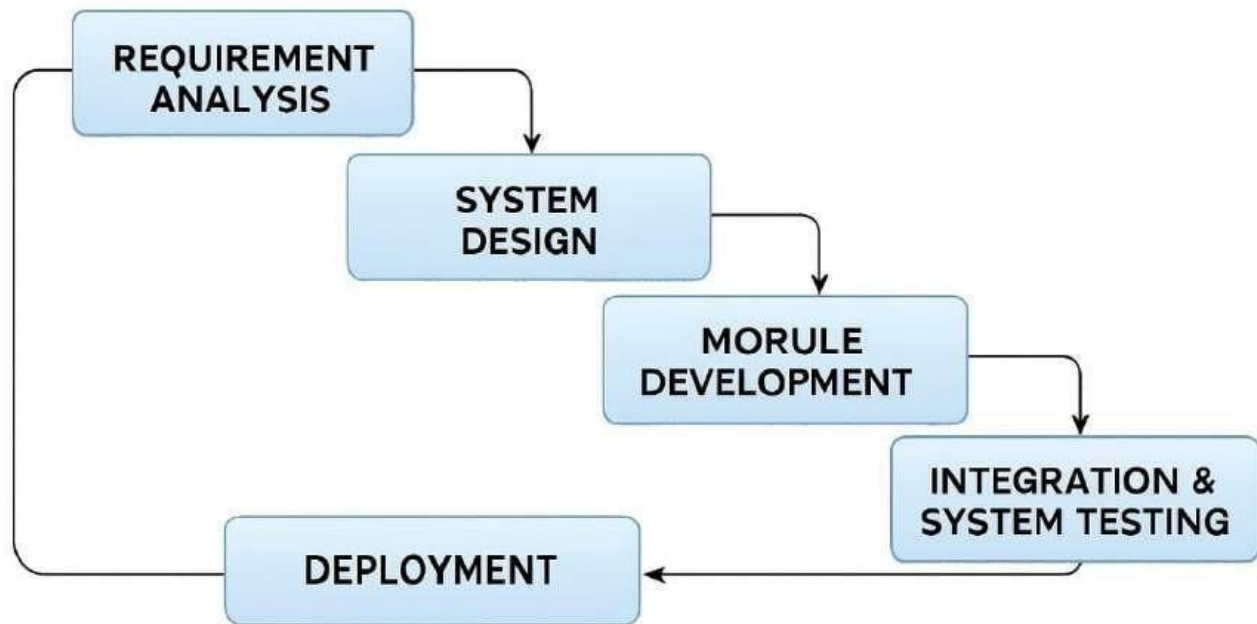
Advantages:

1. Flexibility – Allows improvements based on feedback
2. Early Delivery – Enables partial rollout of working modules
3. Risk Management – Identifies and resolves issues early

Disadvantages:

1. Increased Complexity – Iterations can complicate planning
2. Scope Creep – Frequent changes may expand project scope
3. Resource Intensive – Revisiting phases demands more effort

Iterative Waterfall Model



Applications:

The Iterative Waterfall Model is well-suited for projects like our Online Food Delivery App, where requirements may evolve during development. It allows for structured planning while supporting feedback-driven improvements.

This model is especially useful when partial system delivery is beneficial—such as releasing core features like user registration, restaurant listings, or order tracking early—so stakeholders can evaluate progress and suggest refinements.

Its flexibility makes it ideal for real-world applications where usability, performance, and customer experience are continuously enhanced.

3D .STUDY OF THE SYSTEM

The system is divided into several functional modules that work together to deliver a seamless food ordering and delivery experience. Each module has distinct roles for users and administrators.

Modules Overview:

- **Register:**
 1. User Registration: Customers can sign up to browse restaurants and place orders.
 2. Admin Registration: Admins register to manage restaurant data, orders, and user roles.
- **Verify Account:**
 1. An OTP is sent to the user's email during login to verify identity and ensure secure access.
- **Login:**
 1. User Login: Enables users to access their dashboard, browse menus, and place orders.
 2. Admin Login: Allows admins to manage restaurant listings, orders, and system settings.

- **Home Page:**

Displays featured restaurants, offers, and user reviews.

- **Restaurants & Menus:**

User Interface: Users can browse restaurants, view menus, and search for specific dishes.
Admin Interface: Admins can add, update, or remove restaurant profiles and menu items.

- **Orders:**

User Interface: Users can place orders, track delivery status, and view order history.
Admin Interface: Admins can monitor incoming orders, assign deliveries, and update statuses.

- **About Page:**

Provides information about the app, its purpose, and the development team.

- **Account:**
 1. **User Interface:**
 2. **My Profile:** Displays user details and saved addresses.
 3. **Dashboard:** Shows current orders, past orders, and payment history.
- **Admin Interface:**
 1. **Admin Profile:** Displays admin credentials and access level.
 2. **Admin Dashboard:** Allows management of users, restaurants, orders, and system analytics.

3E.INPUT AND OUTPUT

This section outlines the key inputs and outputs of the system, along with their primary functions:

□ INPUT:

1. Users enter their login credentials (email and password) to access the app.
2. Customers input delivery details, select food items, and place orders.
3. Admins input restaurant data, menu items, and manage order statuses.

□ OUTPUT:

1. Users can view restaurant listings, browse menus, track orders, and receive delivery updates.
2. Admins can access a centralized dashboard to manage users, restaurants, orders, and system analytics.
3. The system generates order confirmations, delivery status updates, and payment receipts.

3F.SOFTWARE REQUIREMENT SPECIFICATIONS

The Software Requirements Specification (SRS) outlines the complete scope of the Online Food Delivery App, including both functional and non-functional requirements. It serves as a blueprint for developers and ensures that the system meets user expectations and business goals.

This document was prepared after thorough discussions with the project team and stakeholders to ensure clarity and completeness.

Developer Responsibilities:

- a. Build a system that fulfills all requirements listed in the SRS.
- b. Demonstrate and deploy the system after successful testing.
- c. Provide user manuals and documentation for system usage and maintenance.

Functional Requirements:

A. User Registration and Authentication:

1. Users must be able to create secure accounts.
2. The system should verify credentials and manage login sessions.

B. Browse and Search Restaurants:

1. Users should be able to browse restaurants and search for specific dishes or cuisines.

C. Menu and Order Management:

1. Users should view detailed menus with prices and availability.
2. Users must be able to place orders and track delivery status.

D. Admin Controls:

1. Admins should manage restaurant profiles, menu items, and user roles.
2. Admins must be able to view, update, or delete entries from the system.

Hardware Requirements:

Intel i3 Processor or higher

8 GB RAM

SSD Storage

Software Requirements:

Windows 11 OS

Visual Studio Code

MongoDB Atlas

Node.js and Express.js

React.js (for frontend development)

3G.SOFTWARE ENGINEERING PARADIGM APPLIED

Software engineering paradigms define the structured methods used during software development. These paradigms exist in a hierarchical form: the programming paradigm is a subset of the software design paradigm, which in turn is part of the broader software development paradigm.



System reliability is addressed at two levels:

1. Ensuring correct requirements through thorough system analysis.
2. Delivering reliable performance to users during actual operation.

Three approaches to reliability are applied:

1. Error Avoidance: Preventing errors during development.
2. Error Detection and Correction: Identifying and fixing errors.
3. Error Tolerance: Allowing the system to continue functioning despite errors.

4. SYSTEM DESIGN

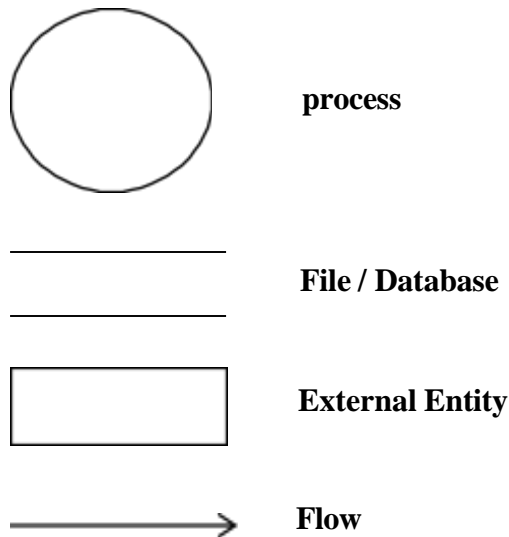
4A. DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) is a graphical tool used to represent the movement of data within an information system. It models the system's functional aspects by illustrating how data is input, processed, stored, and output across various modules.

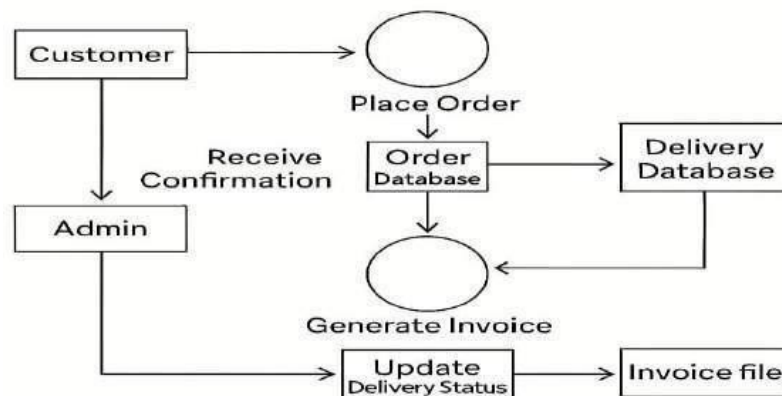
In the context of the Online Food Delivery App, the DFD provides a high-level overview of how user requests, restaurant data, and delivery operations interact within the system. It serves as a foundational step in structured design, enabling developers and stakeholders to visualize the system's behavior before implementation.

. In the development of an online food delivery system, a data flow diagram (DFD) serves as a key tool for visualizing how data moves throughout the application. As leveled DFDs are created, the developer must determine how the platform can be divided into functional modules—such as user registration, order placement, and delivery tracking—and identify the transactional data exchanged between them. DFDs are applicable during both the Analysis and Design stages of the Software Development Life Cycle (SDLC). Different notational styles are used to represent system components, including processes (e.g., placing an order), data stores (e.g., customer profiles), data flows (e.g., payment details), and external entities (e.g., users, restaurants, delivery partners).

DFD Notation:



DFD EXAMPLE:



Steps to Construct Data Flow Diagram:

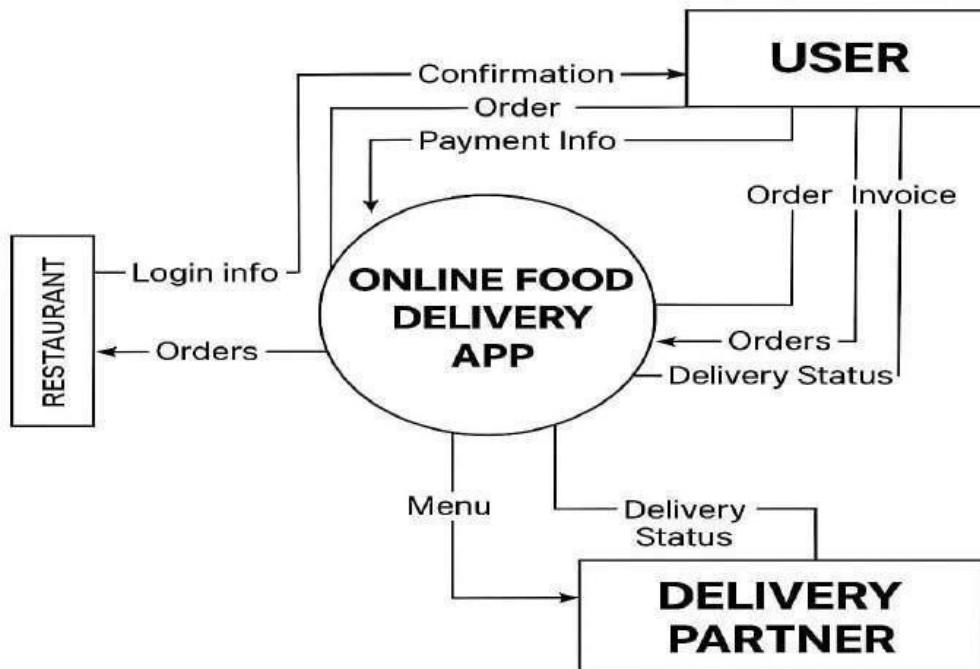
Four steps are generally used to construct a DFD for the Online Food Delivery System:

- Each process (e.g., Place Order, Assign Delivery) should be named clearly and referred for easy identification. The name must represent the actual function in the system.
- The direction of data flow in the diagram should be maintained from top to bottom and from left to right, ensuring logical readability.
- When a process is broken down into lower-level details (e.g., Order Management into 1.0 Validate Order, 1.1 Confirm Payment), they should be numbered accordingly.
- Names of data stores, sources, and destinations (e.g., CUSTOMER, RESTAURANT DATABASE, DELIVERY PARTNER) must be written in capital letters for consistency.

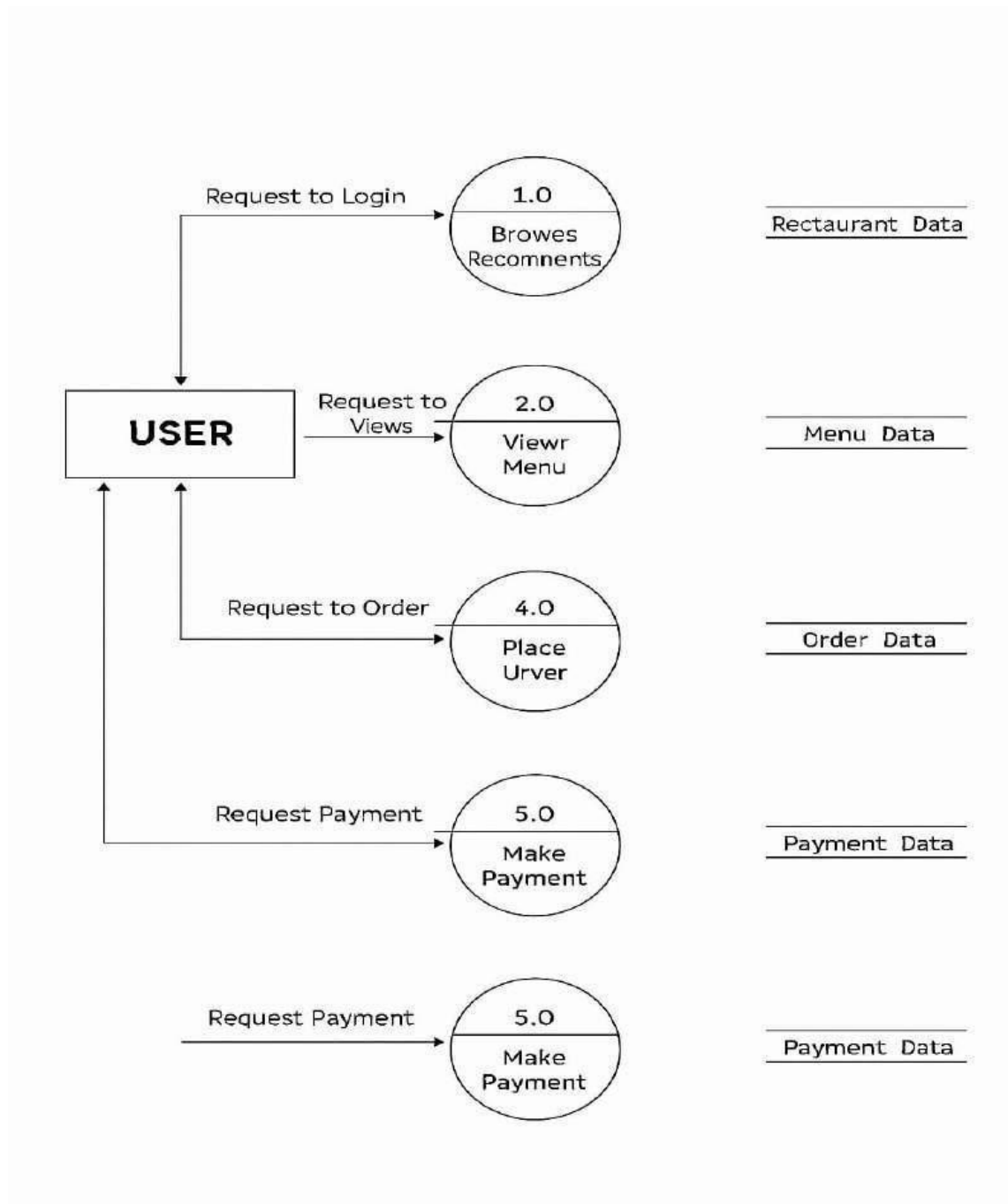
Rules for constructing a Data Flow Diagram:

- Avoid crossing arrows to keep the diagram clear.
- All symbols (squares, circles, files) must be labeled.
- Decomposed processes can reuse names if context is clear.
- Place data flows around the outer edges of the diagram.

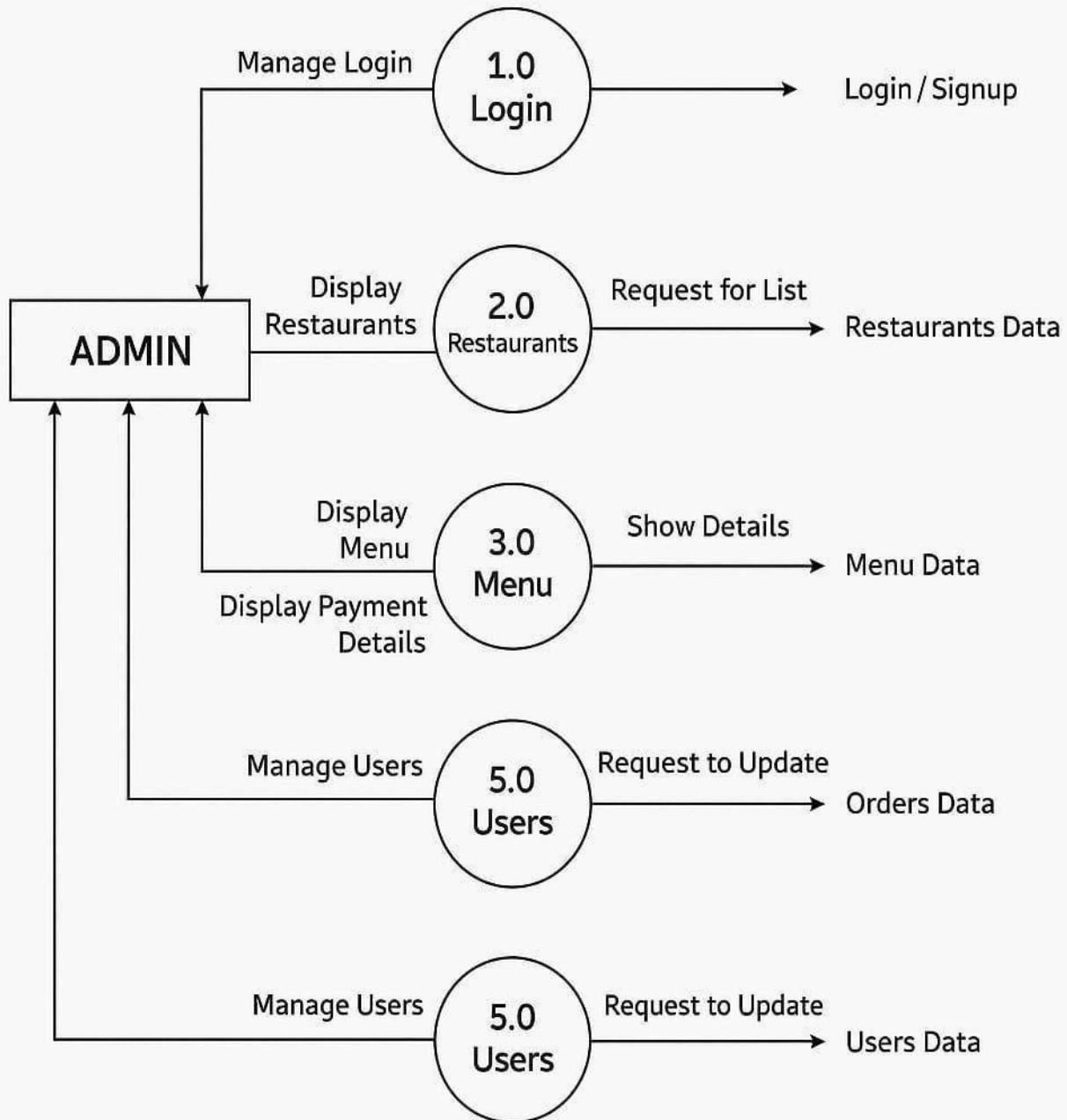
LEVEL 0 DFD OR CONTEXT DIAGRAM:



LEVEL 1 DFD:



LEVEL 1 DFD:



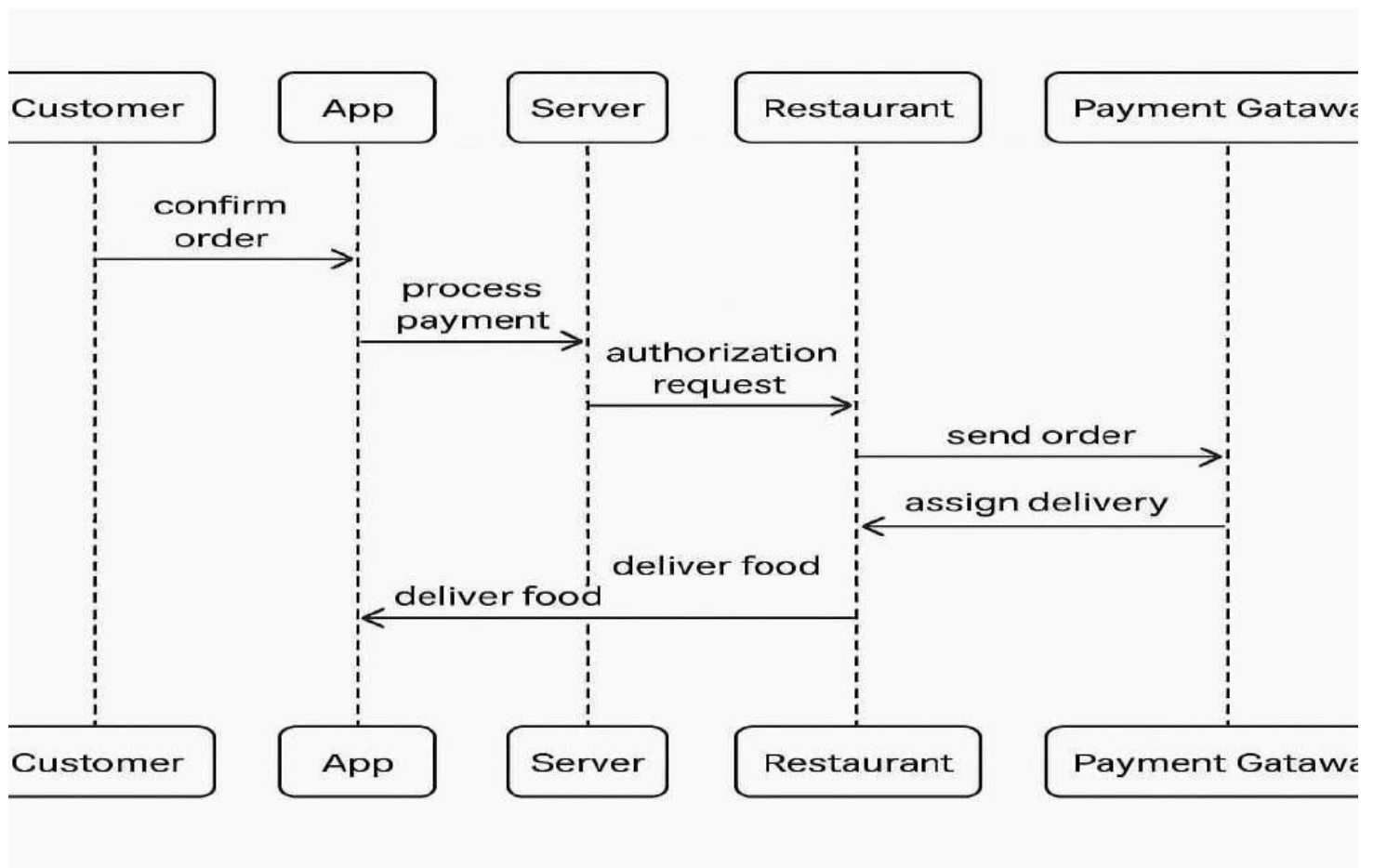
4B.SEQUENCE DIAGRAM

A Sequence Diagram for an online food delivery application illustrates how various components-such as Customer, App, Server, Restaurant, Delivery Partner, and Payment Gateway-interact in a time-ordered sequence to complete a specific process.

It uses vertical lifelines to represent each role and horizontal arrows to depict messages exchanged between them, arranged from top to bottom to indicate the flow of time.

Example Scenario – Place Order Flow:

The Customer confirms an order in the app → the Server processes payment via the Payment Gateway → forwards the order to the Restaurant → assigns a Delivery Partner → who then delivers the food to the Customer.



A Use Case Diagram for an online food delivery system represents how different users interact with the application and the various functionalities they access. It visually maps the relationship between actors (users or external systems) and use cases (specific operations or services provided by the app).

This diagram captures the dynamic behavior of the system by illustrating external and internal interactions. Each actor—such as Customer, Restaurant, Admin, and Delivery Partner—is connected to relevant use cases like Place Order, Manage Menu, Assign Delivery, or Track Order.

Unlike static diagrams, a use case diagram focuses on functional requirements and helps identify the roles and responsibilities of each actor. It provides an outside view of the system, making it easier to understand how users engage with different features.

Key Purposes of the Use Case Diagram:

- Identify and organize system requirements
- Define internal and external actors interacting with the app
- Visualize core functionalities from a user's perspective
- Serve as a foundation for further design and development

Each use case diagram typically models one specific functionality. To represent the entire system, multiple diagrams may be used, each focusing on a distinct module or user flow.

How to draw Use Case Diagram?

Use Case Diagrams are used during high-level requirement analysis to represent system functionalities and user interactions. Each use case reflects a specific feature or service, while actors represent users or external systems interacting with those features.

To create an effective use case diagram, first identify:

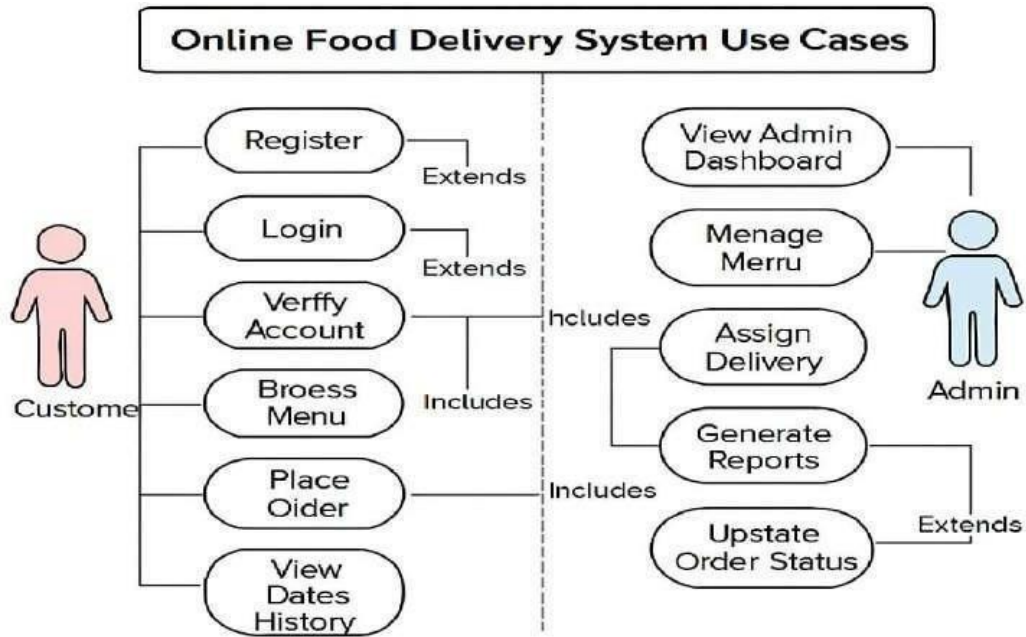
- Functionalities to be shown as use cases
- Actors (e.g., Customer, Admin, Restaurant, Delivery Partner)
- Relationships between actors and use cases

Guidelines for Drawing:

- Choose clear, descriptive names for use cases and actors
- Show relationships and dependencies accurately
- Focus on capturing core requirements—avoid overcomplicating
- Add notes if needed to clarify specific interactions

This diagram helps visualize how users engage with the system and supports further design and development

- **USE CASE DIAGRAM:**



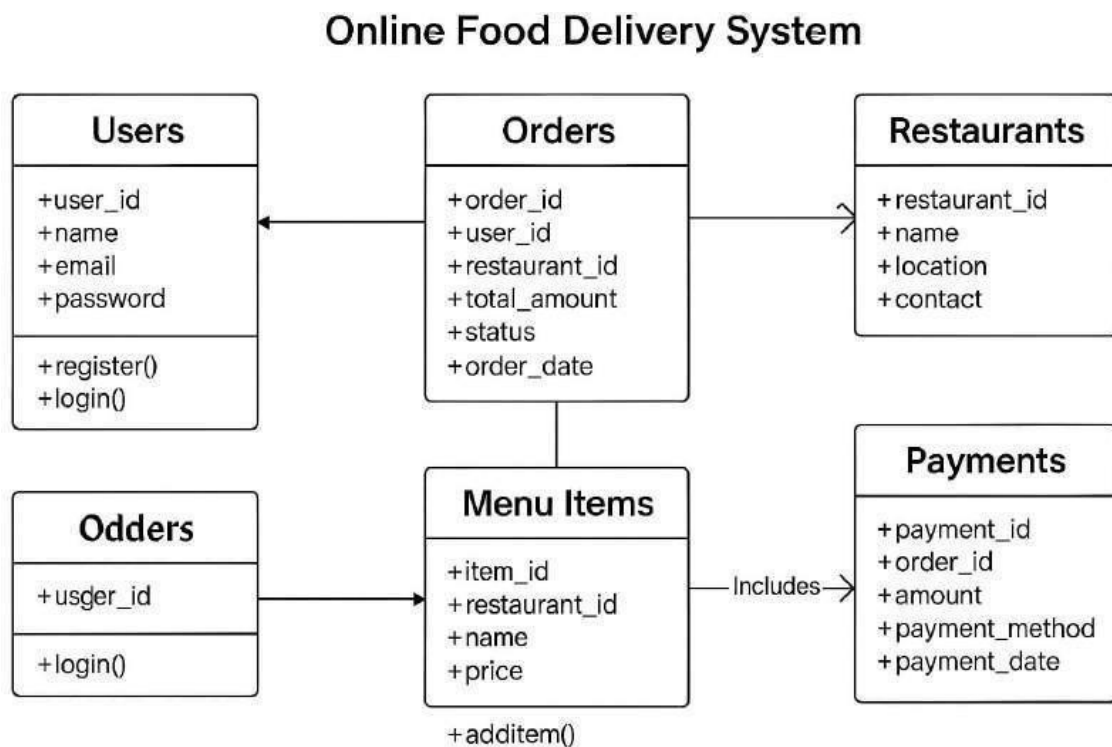
4D.SCHEMA DIAGRAM

The schema represents the logical structure of the online food delivery system's database. It defines the main data categories—such as Users, Restaurants, Menu Items, Orders, and Payments—and the relationships between them.

By organizing data into these entities and linking them appropriately, the schema supports essential functions like browsing restaurants, adding items to the cart, placing orders, processing payments, and tracking deliveries.

A well-designed schema ensures that information is easy to store, retrieve, and manage, enabling the system to operate efficiently and reliably.

- **SCHEMA DESIGN:**



1. UI SNAPSHOT

❖ FRONTEND :-

1) Login Page:



✓ CODE

```
import React, { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';

const Login = () => {
  const [form, setForm] = useState({ email: '', password: '' });
  const Navigate = useNavigate();

  const hc = e => setForm({ ...form, [e.target.name]: e.target.value });

  const hs = async (e) => {
    e.preventDefault();
    const res = await axios.post('http://localhost:3000/api/auth/login', form);
    localStorage.setItem('token', res.data.token);
    Navigate('/home');
```

```

};

return (
  <>
    <form onSubmit={hs} className="register-form">
      <div className='form-group'>
        <label>Email:</label>
        <input
          type='email'
          className='form-control custom-input'
          name='email'
          onChange={hc}
          required
        />
      </div>
      <div className='form-group'>
        <label>Password:</label>
        <input
          type='password'
          className='form-control custom-input'
          name='password'
          onChange={hc}
          required
        />
      </div>
      <button type='submit' className='btn btn-success mt-2 custom-button'>
        🚪 Login
      </button>
    </form>
  </>
);
};

export default Login;

```

Register Page:

Ψ foodyyy Register Login

Name:
deep

Email:
deepattra29@gmail.com

Password:
...

Register

```
import React, { useState } from 'react';
import axios from 'axios';
import './Register.css';

const Register = () => {
  const [form, setForm] = useState({ name: "", email: "", password: "" });

  const hc = e => setForm({ ...form, [e.target.name]: e.target.value });

  const hs = async (e) => {
    e.preventDefault();
    await axios.post('http://localhost:3000/api/auth/register', form);
    alert("Registration successfully");
  };

  return (
    <>
    <form onSubmit={hs} className="register-form">
```

```
<div className="form-group">
  <label>Name:</label>
  <input
    type="text"
    className="custom-input"
    name="name"
    onChange={hc}
    required
  />
</div>
<div className="form-group">
  <label>Email:</label>
  <input
    type="email"
    className="custom-input"
    name="email"
    onChange={hc}
    required
  />
</div>
<div className="form-group">
  <label>Password:</label>
  <input
    type="password"
    className="custom-input"
    name="password"
    onChange={hc}
    required
  />
</div>
<button type="submit" className="custom-button">Register</button>
</form>

</>
```

```
);  
};
```

```
export default Register;
```

Reset Password Page:

Code:

```
const User = require('../models/User');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');

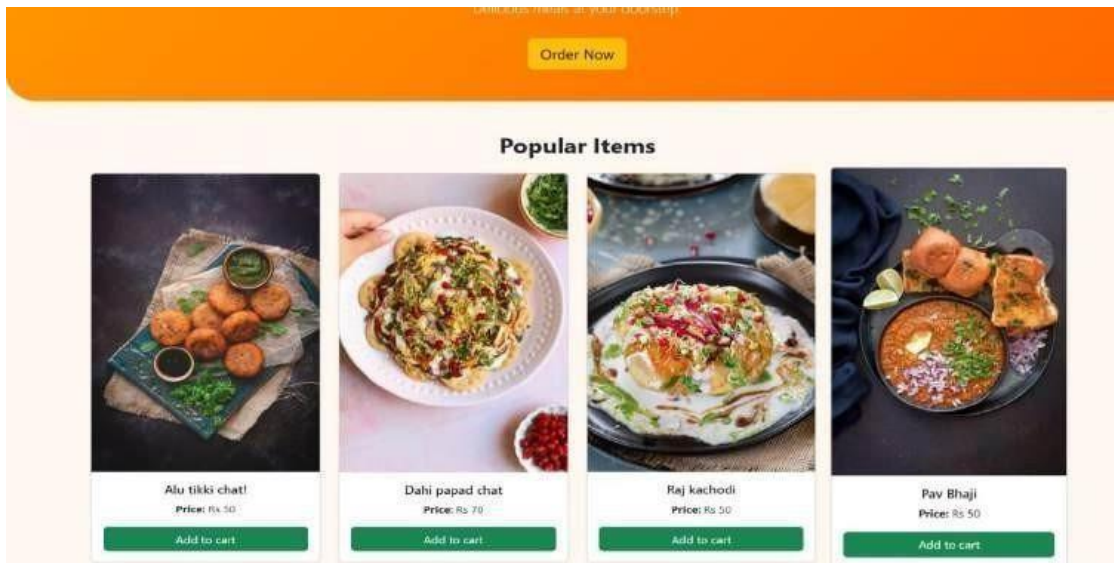
//register
exports.register = async(req,res)=>{
  const {name,email,password} = req.body;
  try {
    const hashedPassword = await bcrypt.hash(password,10);
    const user = await User.create({name,email,password:hashedPassword});
    res.status(201).json({message:'user register sucessfully'});
  } catch(err) {
    res.status(400).json({error:'user already exists'});
  }
};

//login
exports.login = async(req,res)=>{
  const {email,password} = req.body;
  try {
    const user = await User.findOne({email});
    if(!user) return res.status(400).json({error:'invalid credentials'});
    const match = await bcrypt.compare(password,user.password);
    if(!match) return res.status(400).json({error:'invalid credentials'});

    const token =
    jwt.sign({userId:user._id},process.env.JWT_SECRET,{expiresIn:'30d'});
    res.json({token});
  } catch(err) {
    res.status(500).json({error:'login failed'});
  }
}
```

```
};  
//dashboard  
exports.dashboard = (req,res)=>{  
  res.json({message:'welcome to dashboard'});  
}
```


2) FOOD LISTING PAGE (for admin and user):



- **components-pages-Productcard.js.jsx**

```
import React from 'react';
import { Card, Button } from 'react-bootstrap';
import { useCart } from './Cartcontext'; // ✓ adjust path if
needed
```

```
const Productcard = ({ item }) => {
  const { addToCart } = useCart();

  return (
    <Card className='shadow-sm h-100'>
      <Card.Img variant='top' src={item.image} />
      <Card.Body>
        <Card.Title>{item.name}</Card.Title>
        <Card.Text>
          <strong>Price:</strong> {item.price}
        </Card.Text>
        <Button variant='success' onClick={() => addToCart(item)}>
          Add to cart
        </Button>
      </Card.Body>
    </Card>
  );
};
```

```
};
```

```
export default Productcard;
```

Components – Home.jsx :-

```
import React from 'react';
import Carouselslider from './Carouselslider';
import { productdata } from '../data';
import Productcard from './Productcard';
import './Home.css';

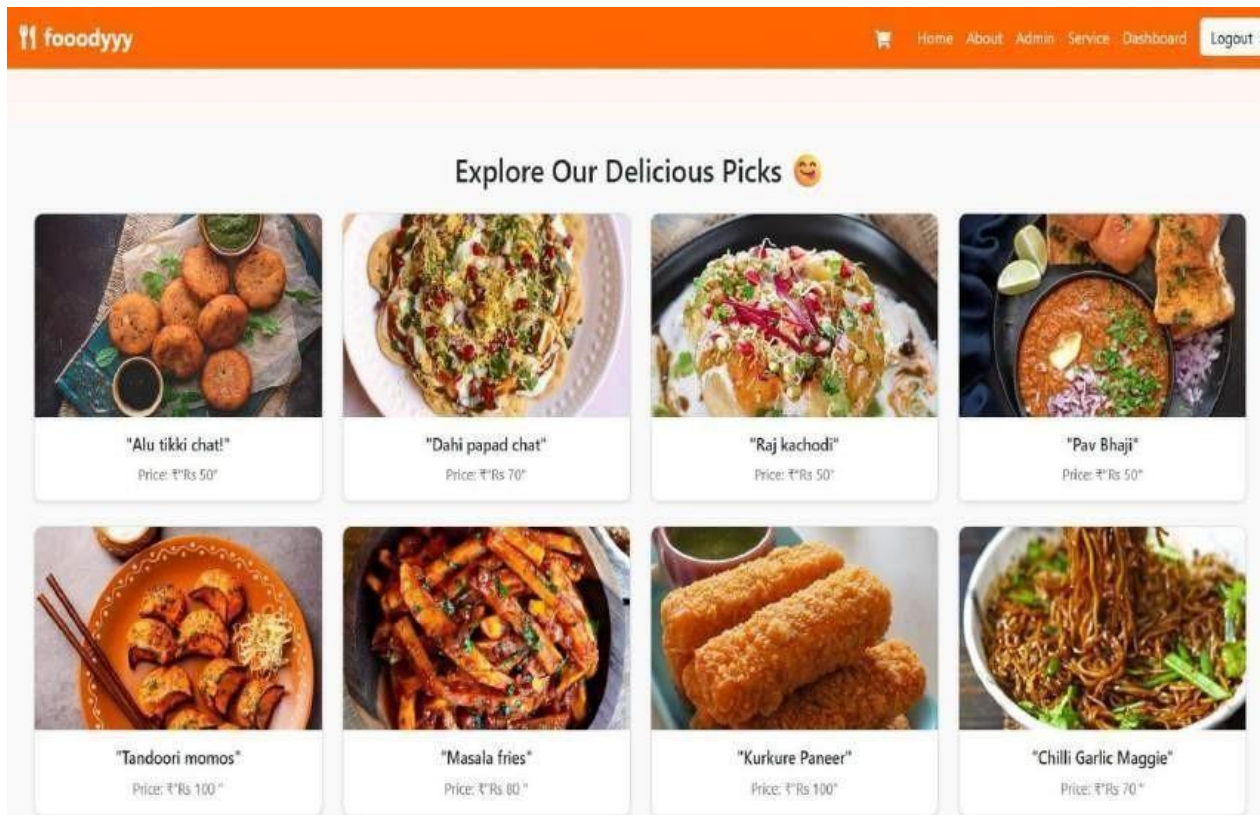
const Home = () => {
  return (
    <>
    <Carouselslider />

    {/* Hero Section */}
    <section className="hero-section text-center text-white py-5">
    <h1 className="display-4 fw-bold">Welcome to Foodyyy!</h1>
    <p className="lead">Delicious meals at your doorstep.</p>
    <a href="#menu" className="btn btn-warning btn-lg mt-3">Order Now</a>
    </section>

    {/* Product Cards Section */}
    <div className='container mt-5' id="menu">
    <h2 className='mb-4 text-center fw-bold">Popular Items</h2>
    <div className='row">
    {productdata.map((item, index) => (
    <div className='col-md-3 mb-4' key={index}>
    <Productcard item={item} />
    </div>
    ))}
    </div>
    </div>
    </>
  );
};

export default Home;
```

3) DISH DETAIL AND ORDERING INTERFACE (for user);



✓ CODE

```
// Import React and required components
import React from 'react';
import { Card, Button } from 'react-bootstrap';
import CarouselSlider from './CarouselSlider';
import { useCart } from './Cartcontext'; // Cart context for managing orders
import { productdata } from '../data'; // Sample product list
import './Home.css';

// Product Card Component

// Shows a single dish's details and lets the user add it to cart
const ProductCard = ({ item }) => {
  const { addToCart } = useCart(); // Get addToCart function from context

  return (
    <Card className='shadow-sm h-100'>
      { /* Dish Image */ }
      <Card.Img variant='top' src={item.image} alt={item.name} />
```

```

{/* Dish Details */}
<Card.Body>
<Card.Title>{item.name}</Card.Title>
<Card.Text>
<strong>Price:</strong> ₹{item.price}
</Card.Text>

{/* Ordering Button */}
<Button variant='success' onClick={() => addToCart(item)}>
Add to Cart
</Button>
</Card.Body>
</Card>
);
};

```

// Home Component

```

// Displays the hero section + list of dishes (menu)
const Home = () => {
return (
<>
{/* Carousel for promotions or featured dishes */}
<CarouselSlider />

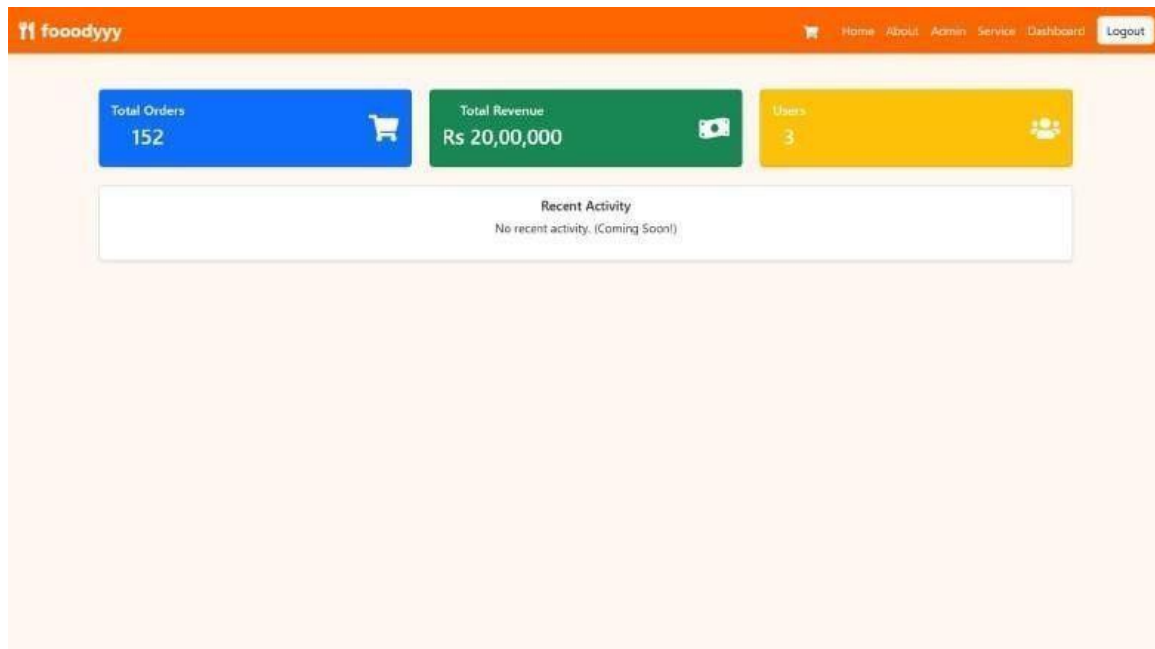
{/* Hero Section */}
<section className="hero-section text-center text-white py-5">
<h1 className="display-4 fw-bold">Welcome to Foodyyy!</h1>
<p className="lead">Delicious meals at your doorstep.</p>
<a href="#menu" className="btn btn-warning btn-lg mt-3">
Order Now
</a>
</section>

{/* Menu Section (Dish Details + Order Buttons) */}
<div className='container mt-5' id="menu">
<h2 className='mb-4 text-center fw-bold'>Popular Items</h2>
<div className='row'>
{productdata.map((item, index) => (
<div className='col-md-3 mb-4' key={index}>
{/* Pass dish data to ProductCard */}
<ProductCard item={item} />
</div>
)))}
</div>
</div>
</>
);
};

export default Home;

```

DASHBOARD PAGE:



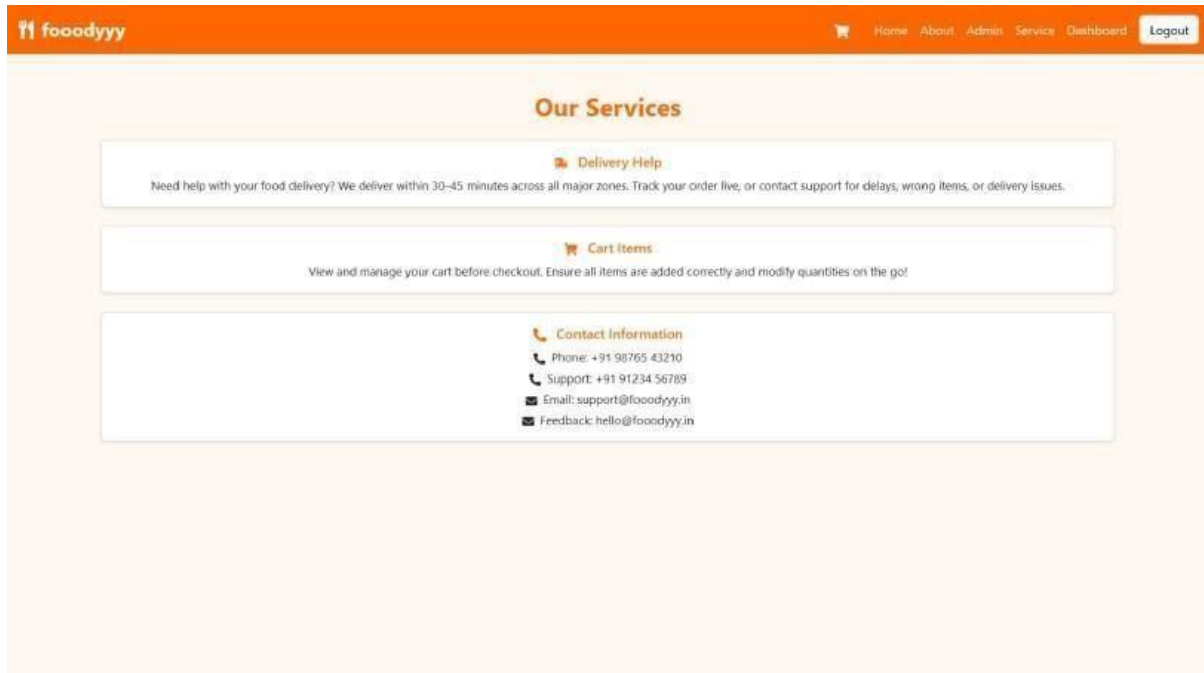
CODE:

```
import React from 'react'
import PaymentPage from './Paymentpage'

const Dashboard = () => {
  return <>
    <PaymentPage/>
  </>
}

export default Dashboard
```

SERVICEPAGE:



✓ CODE:

```
import React from 'react'
import { FaShippingFast, FaShoppingCart, FaEnvelope, FaPhoneAlt } from 'react-icons/fa'
```

```
const ServicePage = () => {
  return (
    <div className="container mt-5">
      <h2 className="text-center mb-4 fw-bold" style={{ color: '#FF6B00' }}>
        Our Services
      </h2>
```

```
    { /* Delivery Help Section */ }
    <div className="card mb-4 shadow-sm">
      <div className="card-body">
        <h4 className="card-title text-orange" style={{ color: '#FF6B00' }}>
          <FaShippingFast className="me-2" /> Delivery Help
        </h4>
        <p className="card-text">
```

```
          Need help with your food delivery? We deliver within 30-45 minutes across all major zones.
```

Track your order live, or contact support for delays, wrong items, or delivery issues.

</p>

</div>

</div>

{/* Cart Items Section */}

<div className="card mb-4 shadow-sm">

<div className="card-body">

<h4 className="card-title text-orange" style={{ color: '#FF6B00' }}>

<FaShoppingCart className="me-2" /> Cart Items

</h4>

<p className="card-text">

View and manage your cart before checkout. Ensure all items are added correctly and modify quantities on the go!

</p>

</div>

</div>

{/* Contact Info Section */}

<div className="card shadow-sm">

<div className="card-body">

<h4 className="card-title text-orange" style={{ color: '#FF6B00' }}>

<FaPhoneAlt className="me-2" /> Contact Information

</h4>

<p className="card-text mb-1">

<FaPhoneAlt className="me-2" />

Phone: +91 98765 43210

</p>

<p className="card-text mb-1">

<FaPhoneAlt className="me-2" />

Support: +91 91234 56789

</p>

<p className="card-text mb-1">

<FaEnvelope className="me-2" />

Email: support@foodyyy.in

</p>

<p className="card-text">

<FaEnvelope className="me-2" />

Feedback: hello@foodyyy.in

</p>

</div>

</div>

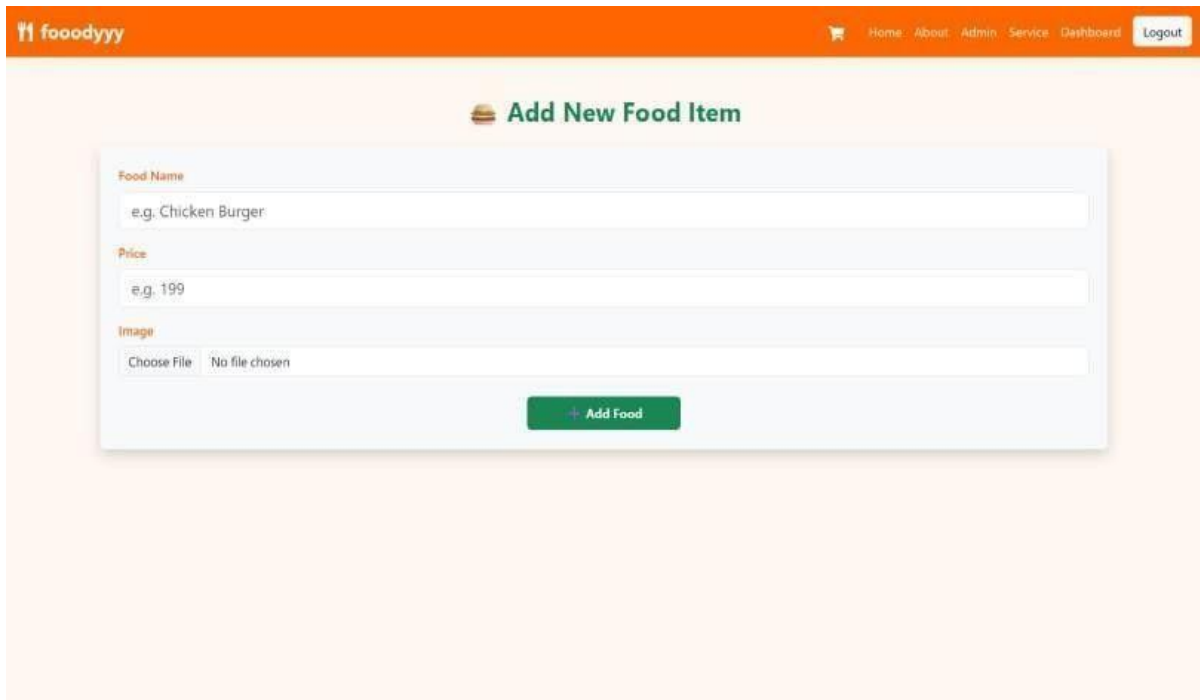
</div>

)

}

export default ServicePage

ADD FOOD: Admin Panel For Listing New Items



The screenshot shows a web application interface for adding a new food item. At the top, there is an orange header bar with the logo 'foodyyy' on the left and navigation links 'Home', 'About', 'Admin', 'Service', 'Dashboard', and a 'Logout' button on the right. Below the header, the main content area has a light orange background. Centered at the top of this area is a green header with a burger icon and the text 'Add New Food Item'. Below this is a white form box with a light gray border. The form contains three sections: 'Food Name' with a text input field containing 'e.g. Chicken Burger', 'Price' with a text input field containing 'e.g. 199', and 'Image' with a file upload area showing 'Choose File' and 'No file chosen'. At the bottom right of the form box is a green button with a plus icon and the text 'Add Food'.

✓ CODE

```
import React, { useState } from 'react';  
import axios from 'axios';
```

```
const FoodForm = ({ onareaaadd }) => {  
  const [form, setForm] = useState({  
    name: "",  
    price: "",  
    image: null,  
  });
```

```
  const hc = (e) => {  
    const { name, value, files } = e.target;  
    setForm(prev => ({  
      ...prev,  
      [name]: files ? files[0] : value
```



```
));  
};
```

```
const hs = async (e) => {  
  e.preventDefault();  
  try {  
    const data = new FormData();  
    data.append('name', form.name);  
    data.append('price', form.price);  
    data.append('image', form.image);
```

```
    const res = await axios.post('http://localhost:3000/api/food', data);  
    if (res.status === 201) {  
      alert('Food added successfully!');  
      onareaadd();  
      setForm({  
        name: "",  
        price: "",  
        image: null,  
      });  
    }  
  } catch (error) {  
    alert('Failed to add food. Please try again.');
```

```
    console.error(error);  
  }  
};
```

```
return (  
  <div className='container mt-5'>  
    <h2 className='text-center mb-4 fw-bold text-success'>Add New Food Item</h2>
```

```
<form onSubmit={hs} encType='multipart/form-data' className='shadow p-4 rounded bg-light'>
```

```
<div className='mb-3'>
```

```
<label className='form-label fw-semibold'>Food Name</label>
```

```
<input
```

```
type='text'
```

```
className='form-control form-control-lg'
```

```
name='name'
```

```
placeholder='e.g. Chicken Burger'
```

```
value={form.name}
```

```
onChange={hc}
```

```
required
```

```
/>
```

```
</div>
```

```
<div className='mb-3'>
```

```
<label className='form-label fw-semibold'>Price</label>
```

```
<input
```

```
type='number'
```

```
className='form-control form-control-lg'
```

```
name='price'
```

```
placeholder='e.g. 199'
```

```
value={form.price}
```

```
onChange={hc}
```

```
required
```

```
/>
```

```
</div>
```

```
<div className='mb-4'>
```

```
<label className='form-label fw-semibold'>Image</label>
```

```
<input
```

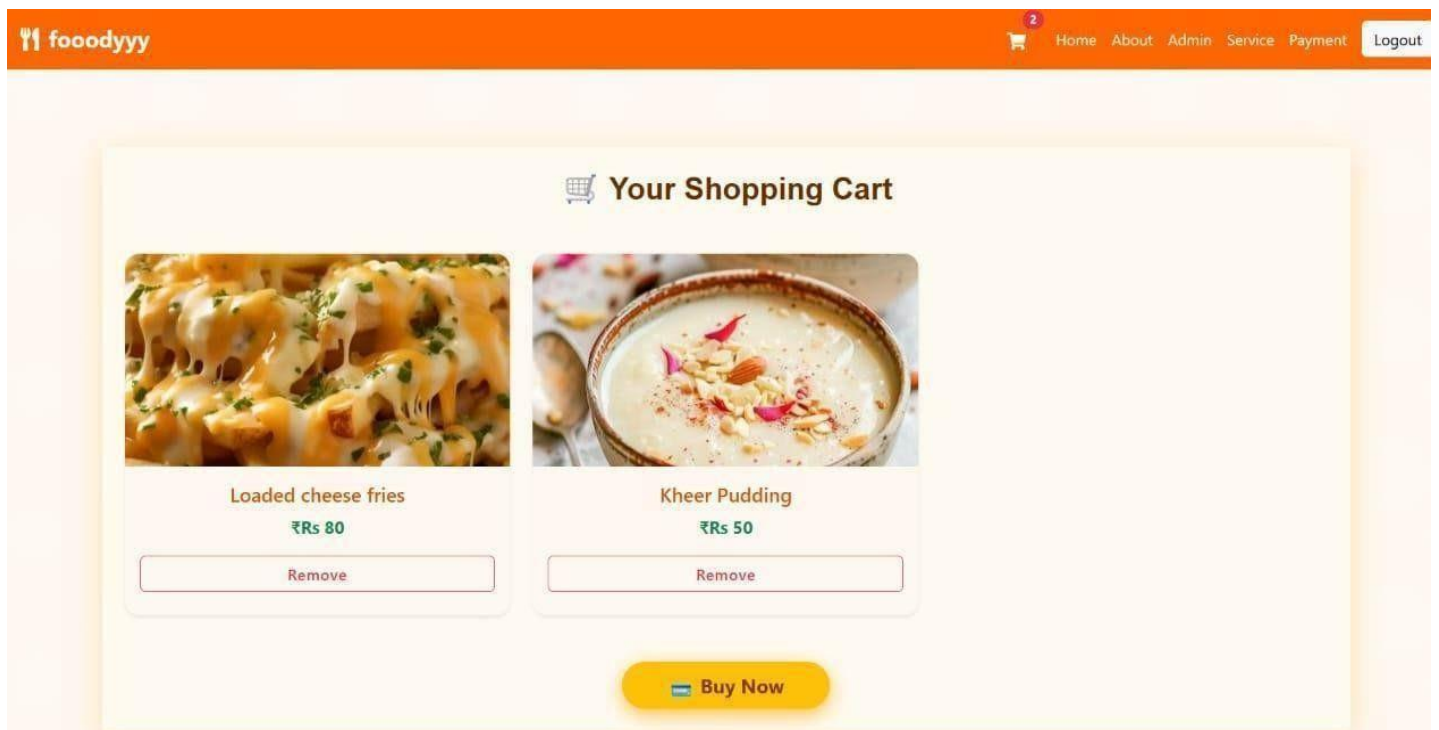
```
type='file'
```

```
className='form-control'  
name='image'  
onChange={hc}  
required  
</>  
</div>
```

```
<div className='text-center'>  
  <button type='submit' className='btn btn-success px-5 py-2 fw-bold'>  
    Add Food  
  </button>  
</div>  
</form>  
</div>  
);  
};
```

```
export default FoodForm;
```

CART PAGE: Managing Selected Food Items Before Checkout



✓ CODE:

```
import React from 'react';
import { useCart } from './Cartcontext';
import { Container, Row, Col, Card, Button } from 'react-bootstrap';
import { useNavigate } from 'react-router-dom';
```

```
const CartPage = () => {
  const { cartItems, removeFromCart } = useCart();
  const navigate = useNavigate();
```

```
  const handleBuyNow = () => {
    if (cartItems.length === 0) {
      alert("Your cart is empty!");
      return;
    }
    navigate('/dashboard');
  };
};
```

```
  return (
    <div style={styles.pageBackground}>
      <Container
        className="mt-5 p-4 rounded"
```

```

style={styles.containerBackdrop}
>
<h2
className="mb-5 fw-bold text-center"
style={{ fontFamily: "Poppins", color: "#6b2e00" }}
>
🛒 Your Shopping Cart
</h2>
{cartItems.length === 0 ? (
<p
className="text-center text-muted"
style={{ fontSize: '1.2rem', fontWeight: '500' }}
>
Your cart is empty.
</p>
) : (
<Row>
{cartItems.map((item, index) => (
<Col md={4} sm={6} xs={12} key={index} className="mb-4">
<Card
className="shadow-sm border-0 h-100"
style={{
borderRadius: '15px',
transition: 'transform 0.3s ease',
backgroundColor: '#fff8f0',
boxShadow: '0 6px 10px rgba(255, 135, 77, 0.3)',
}}
onMouseEnter={e => (e.currentTarget.style.transform = 'scale(1.03)')}
onMouseLeave={e => (e.currentTarget.style.transform = 'scale(1)')}
>
<Card.Img
variant="top"
src={item.image}
style={{
height: "220px",
objectFit: "cover",
borderTopLeftRadius: '15px',
borderTopRightRadius: '15px',
}}
/>
<Card.Body className="d-flex flex-column">
<Card.Title
className="fw-semibold"

```


```

style={{ color: '#d35400', fontSize: '1.3rem' }}
>
{item.name}
</Card.Title>
<Card.Text
className="text-success fw-bold"
style={{ fontSize: '1.15rem' }}
>
₹{item.price}
</Card.Text>
<div className="mt-auto">
<Button
variant="outline-danger"
className="w-100 mb-2"
style={{ fontWeight: '600', letterSpacing: '0.05em' }}
onClick={() => removeFromCart(index)}
>
Remove
</Button>
</div>
</Card.Body>
</Card>
</Col>
))}
</Row>

<div className="d-flex justify-content-center mt-4">
<Button
variant="warning"
size="lg"
style={{
paddingLeft: '3rem',
paddingRight: '3rem',
fontWeight: '700',
fontSize: '1.3rem',
borderRadius: '50px',
boxShadow: '0 6px 20px rgba(255, 165, 0, 0.7)',
transition: 'background-color 0.3s ease, box-shadow 0.3s ease',
color: '#7f3700',
}}
onClick={handleBuyNow}
onMouseEnter={e => {
e.currentTarget.style.backgroundColor = '#ff9e1b';
e.currentTarget.style.boxShadow = '0 8px 30px rgba(255, 160, 0, 0.9)';

```

```

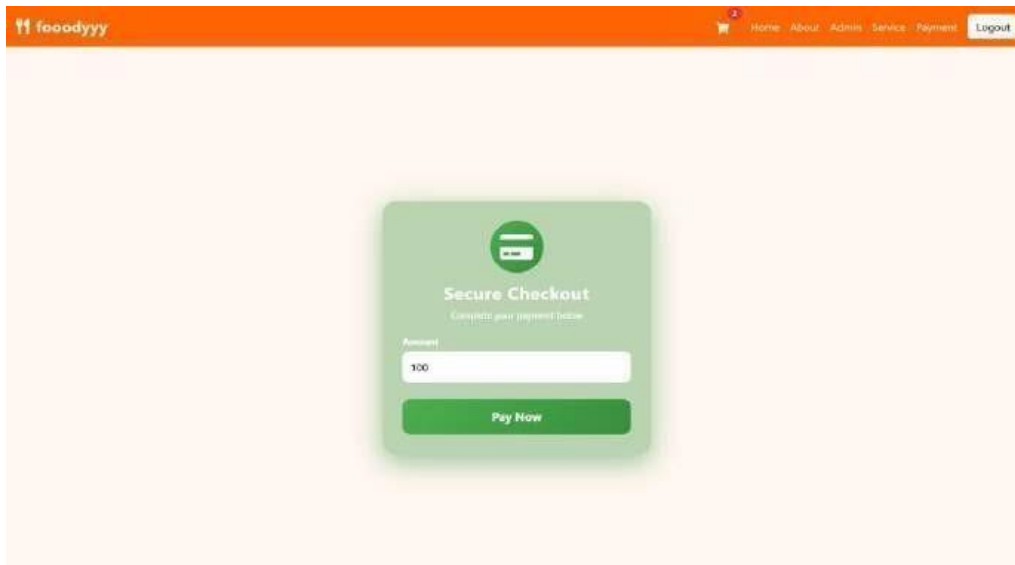
e.currentTarget.style.color = '#4b2200';
}}
onMouseLeave={e => {
e.currentTarget.style.backgroundColor = "";
e.currentTarget.style.boxShadow = '0 6px 20px rgba(255, 165, 0, 0.7)';
e.currentTarget.style.color = '#7f3700';
}}
>
 Buy Now
</Button>
</div>
</>
)}
</Container>
</div>
);
};

const styles = {
pageBackground: {
minHeight: '100vh',
background: linear-gradient(135deg, #fceabb 0%, #f8b500 100%),
backgroundImage: `
radial-gradient(circle at 20% 20%, rgba(255, 255, 255, 0.15) 10%, transparent 20%),
radial-gradient(circle at 80% 80%, rgba(255, 255, 255, 0.15) 15%, transparent 25%)`,
backgroundRepeat: 'repeat',
backgroundSize: '150px 150px',
padding: '2rem 0',
},
containerBackdrop: {
backgroundColor: 'rgba(255, 250, 240, 0.85)',
boxShadow: '0 8px 30px rgba(255, 155, 0, 0.3)',
},
};

export default CartPage;

```

Secure Checkout: Final Payment Confirmation Interface



✓ CODE:

```
import React, { useState } from 'react'
```

```
import axios from 'axios'
```

```
const Foodfrom = ({ onareaadd }) => {
```

```
  const [form, setForm] = useState({
```

```
    name: "",
```

```
    price: "",
```

```
    image: null,
```

```
  });
```

```
  const hc = (e) => {
```

```
    const { name, value, files } = e.target;
```

```
    setForm(prev => ({
```

```
      ...prev,
```

```
      [name]: files ? files[0] : value
```

```
    }));
```

```
  };
```



```

const hs = async (e) => {
  e.preventDefault();
  const data = new FormData();
  data.append('name', form.name);
  data.append('price', form.price);
  data.append('image', form.image);

  const res = await axios.post('http://localhost:3000/api/food', data);
  if (res.status === 201) {
    alert('Food added successfully!');
    onareaadd();
    setForm({
      name: "",
      price: "",
      image: null,
    });
  }
};

return (
  <div className='container mt-5'>
    <h2 className='text-center mb-4 fw-bold text-success'>🍔 Add New Food Item</h2>

    <form onSubmit={hs} encType='multipart/form-data' className='shadow p-4 rounded bg-light'>
      <div className='mb-3'>
        <label className='form-label fw-semibold'>Food Name</label>
        <input
          type='text'
          className='form-control form-control-lg'
          name='name'
          placeholder='e.g. Chicken Burger'
          value={form.name}
          onChange={hc}
          required

```

```
/>
```

```
</div>
```

```
<div className='mb-3'>
```

```
<label className='form-label fw-semibold'>Price</label>
```

```
<input
```

```
type='text'
```

```
className='form-control form-control-lg'
```

```
name='price'
```

```
placeholder='e.g. 199'
```

```
value={form.price}
```

```
onChange={hc}
```

```
required
```

```
/>
```

```
</div>
```

```
<div className='mb-4'>
```

```
<label className='form-label fw-semibold'>Image</label>
```

```
<input
```

```
type='file'
```

```
className='form-control'
```

```
name='image'
```

```
onChange={hc}
```

```
required
```

```
/>
```

```
</div>
```

```
<div className='text-center'>
```

```
<button type='submit' className='btn btn-success px-5 py-2 fw-bold'>
```

```
+ Add Food
```

```
</button>
```

```
</div>
```

```
</form>
```

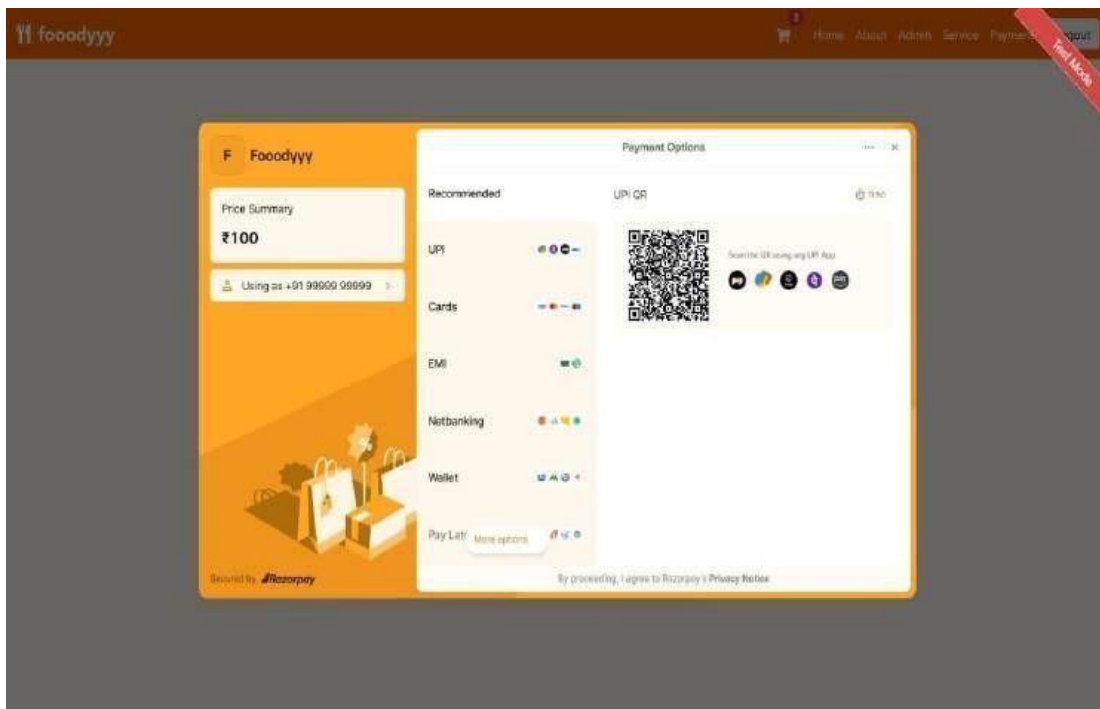
```
</div>
```

```
);
```

```
};
```

```
export default Foodfrom;
```

Payment Gateway: Razorpay Integration with UPI and Card



✓ CODE:

```
import React, { useState, useEffect } from 'react';

import axios from 'axios';

import { useNavigate, useLocation } from 'react-router-dom';
import { FaCreditCard } from 'react-icons/fa';

const PaymentPage = () => {

  const location = useLocation();

  const navigate = useNavigate();

  const [amount, setAmount] = useState(location.state?.price || 100);

  const [loading, setLoading] = useState(false);
```

```

useEffect(() => {
  if (!document.querySelector('#razorpay-script')) {
    const script = document.createElement('script');
    script.id = 'razorpay-script';
    script.src = 'https://checkout.razorpay.com/v1/checkout.js';
    document.body.appendChild(script);
  }
}, []);

const handlePayment = async () => {
  if (amount <= 0) {
    alert('Please enter a valid amount');
    return;
  }
  try {
    setLoading(true);

    const { data } = await
    axios.post('http://localhost:3000/api/payment/create-order', {
      amount });

    const { orderId, currency } = data;

    const options = {
      key: 'rzp_test_3zZ0qPuS56g86S',
      amount: amount * 100,
      currency,
      name: 'Foodyyy',
      description: 'Service Payment',
      order_id: orderId,

```

```

handler: async (response) => {
  await axios.post('http://localhost:3000/api/payment/save-payment',
  {
    orderId,
    paymentId: response.razorpay_payment_id,
    amount,
    currency,
    status: 'success',
  });
  alert('✔ Payment Successful!');
  navigate('/about');
},
prefill: {
  name: 'User',
  email: 'user@example.com',
  contact: '9999999999',
},
theme: { color: '#FFA726' }, // lighter orange
};

const rzp = new window.Razorpay(options);
rzp.open();
} catch (error) {
  console.error('Payment Error', error);
  alert('✖ Payment failed. Please try again.');
```

```

} finally {
  setLoading(false);

```

```
}
```

```
};
```

```
return (
```

```
<div style={styles.page}>
```

```
<div style={styles.card}>
```

```
<div style={styles.iconWrapper}>
```

```
<FaCreditCard size={50} color="#fff" />
```

```
</div>
```

```
<h2 style={styles.heading}>Secure Checkout</h2>
```

```
<p style={styles.subText}>Complete your payment below</p>
```

```
<div style={styles.amountBox}>
```

```
<span style={styles.amountLabel}>Amount</span>
```

```
<input
```

```
type="number"
```

```
min="1"
```

```
value={amount}
```

```
onChange={(e) => setAmount(Number(e.target.value))}
```

```
style={styles.input}
```

```
/>
```

```
</div>
```

```
<button
```

```
style={{ ...styles.payButton, opacity: loading ? 0.7 : 1 }}
```

```
onClick={handlePayment}
```

```
disabled={loading}
```

```

>
{loading ? 'Processing...' : 'Pay Now'}
</button>
</div>
</div>
);
};
const styles = {
page: {
minHeight: '100vh',

//background: 'linear-gradient(135deg, #a8e6cf, #56ab2f)', // soft
green gradient

display: 'flex',
justifyContent: 'center',
alignItems: 'center',
padding: 20,
},
card: {
width: '100%',
maxWidth: 400,
padding: 30,
borderRadius: 20,
background: 'rgba(56, 142, 60, 0.35)', // green with transparency
backdropFilter: 'blur(20px)',
boxShadow: '0 12px 40px rgba(56, 142, 60, 0.5)', // green shadow
textAlign: 'center',
color: 'white',
},

```



```

iconWrapper: {

background: 'linear-gradient(135deg, #4caf50, #388e3c)', // green
gradient

padding: 15,

borderRadius: '50%',

display: 'inline-block',

marginBottom: 15,

},

heading: {

fontSize: 26,

marginBottom: 8,

fontWeight: '700',

letterSpacing: '1px',

},

subText: {

fontSize: 14,

opacity: 0.85,

marginBottom: 20,

fontWeight: '500',

},

amountBox: {

textAlign: 'left',

marginBottom: 25,

},

amountLabel: {

fontSize: 14,

fontWeight: 'bold',

display: 'block',

```

```

marginBottom: 5,

},

input: {

width: '100%',

padding: 12,

borderRadius: 10,

border: 'none',

outline: 'none',

fontSize: 16,

},

payButton: {

width: '100%',

padding: '14px 0',

fontSize: 18,

fontWeight: 'bold',

color: 'white',

background: 'linear-gradient(135deg, #4caf50, #388e3c)',

border: 'none',

borderRadius: 12,

cursor: 'pointer',

transition: 'transform 0.2s ease',

},

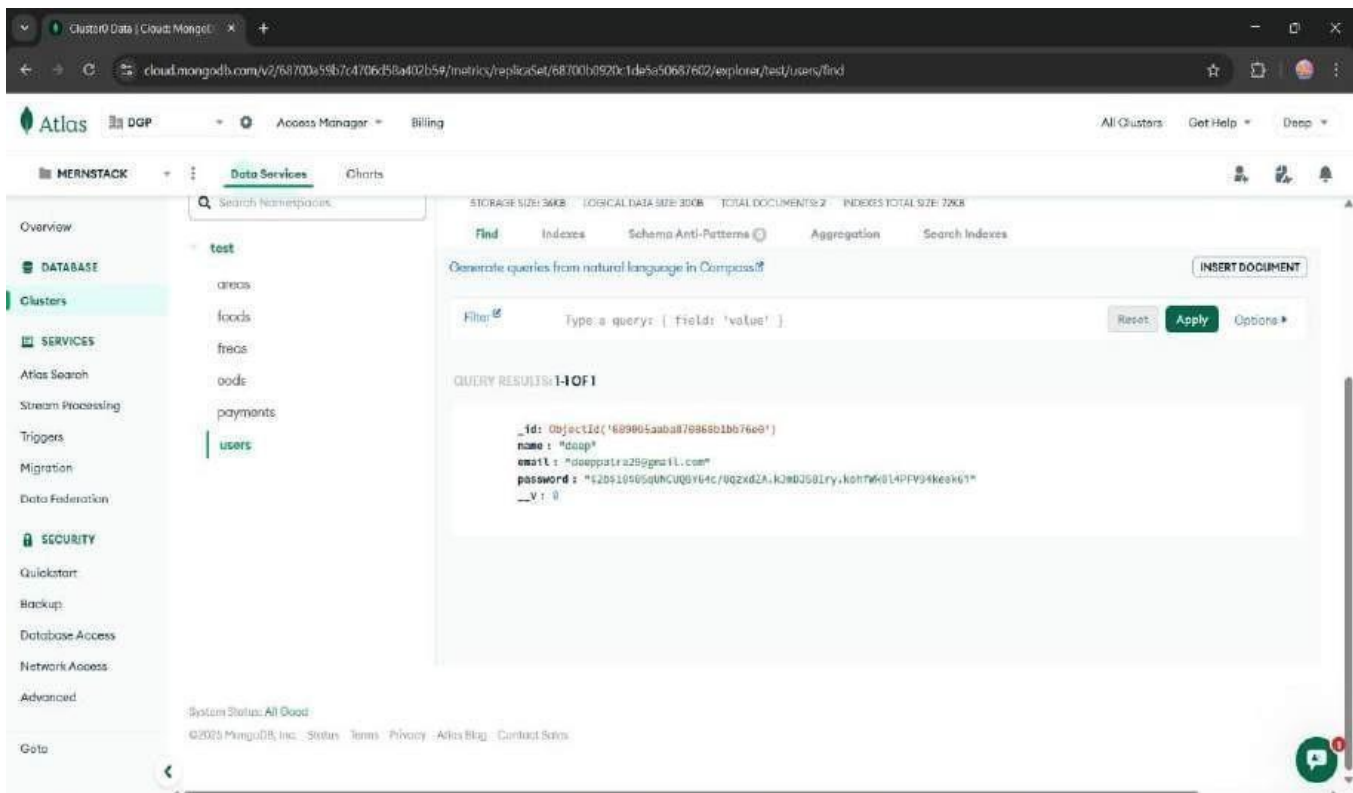
};

```

```
export default PaymentPage;
```

❖ BACKEND:-

1) USER AND ADMIN DATA:



Database - db.js:

```
const mongoose = require("mongoose");

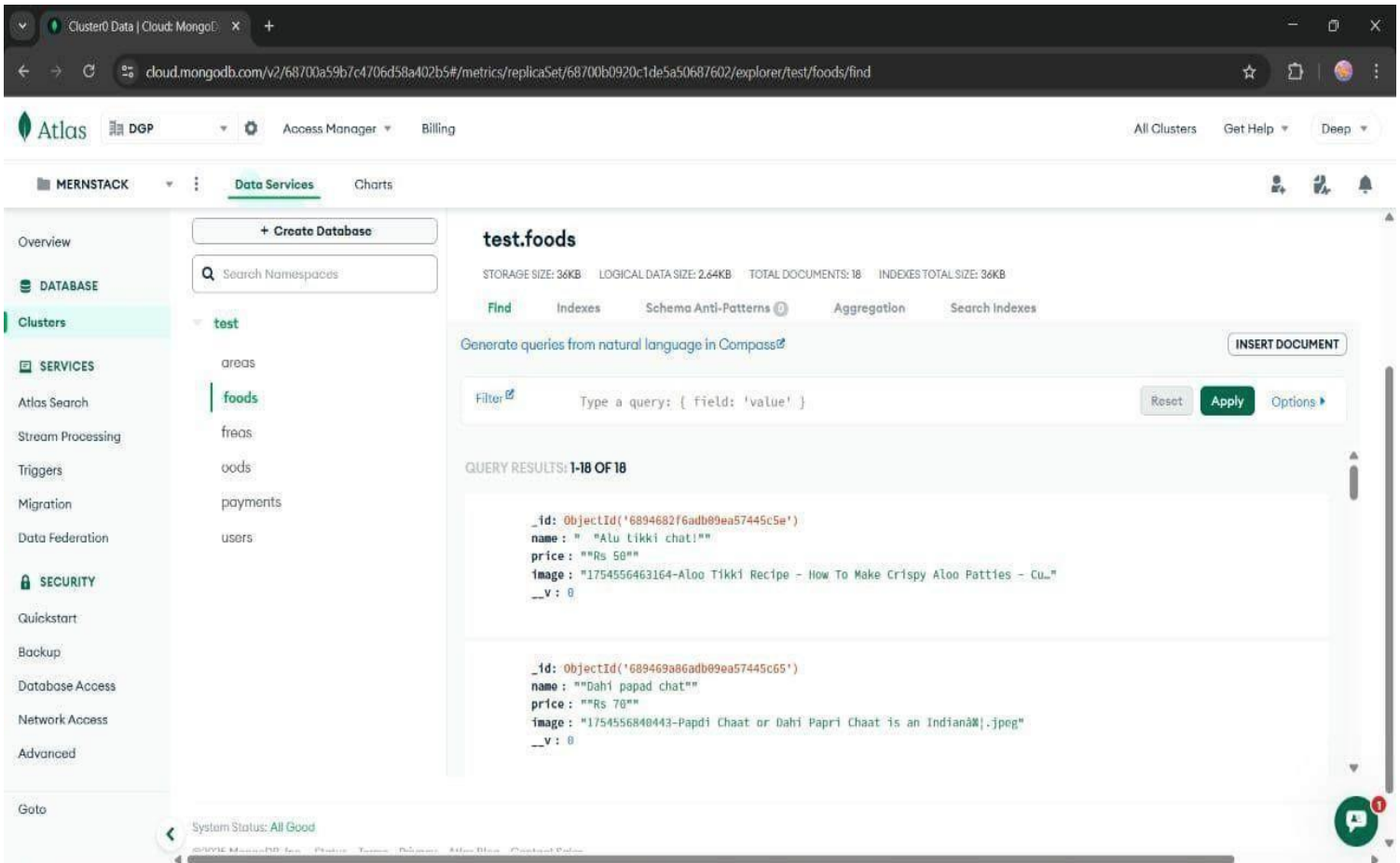
const connectdb = async () => {

  try {
    await mongoose.connect(process.env.MONGO_URI);
    console.log("mongodb connected");
  }
  catch (err) {
    console.error(err);
  }

}

module.exports = connectdb;
```

2) Food DATA:



- model – Food.js :-

```
const mongoose = require("mongoose");

const foodSchema = new mongoose.Schema({
  name:{type:String,required:true},
  price:{type:String,required:true},
  image:{type:String}
});

module.exports = mongoose.model('Food', foodSchema);
```

2. CONCLUSION

Foodyyy is more than just an online food delivery platform — it's a bridge between customers and their favorite meals, delivering convenience, speed, and variety at their fingertips. By integrating an intuitive interface, secure payment options, and efficient order tracking, Foodyyy ensures a seamless experience from browsing the menu to enjoying the food. Its scalable design supports a wide range of cuisines and restaurants, catering to diverse customer preferences. As the demand for quick and reliable food delivery continues to grow, Foodyyy stands as a promising solution to meet the evolving needs of customers while empowering restaurants to reach a broader audience.

BIBLIOGRAPHY

- 1) www.w3schools.com
- 2) www.youtube.com
- 3) www.pexels.com
- 4) www.codepen.io
- 5) www.google.com
- 6) www.fonts.google.com
- 7) www.reactjs.org
- 8) www.codingworld.com