SAVORY BISTRO

by

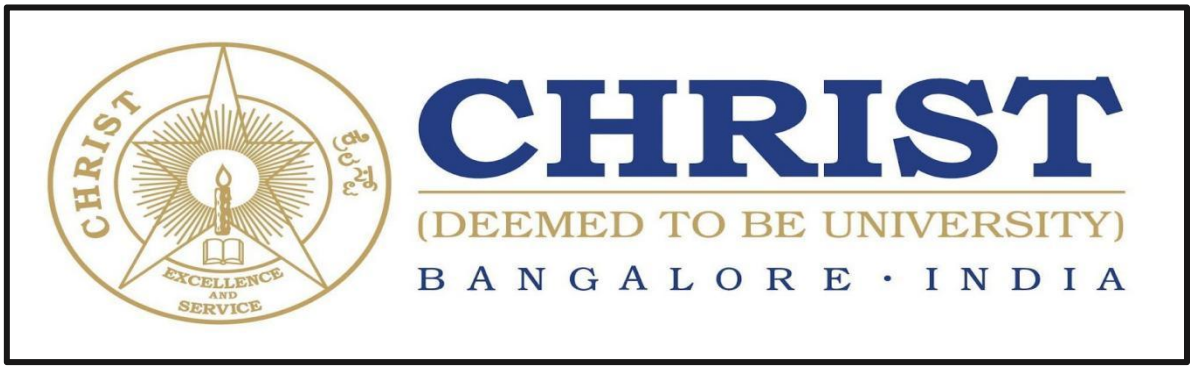
Arun Kumar(2348001)

Sanjay R(2348055)

Likith Kumar(2348072)

Under the guidance of

Dr SATHYA P



A Project report submitted in partial fulfillment of the requirements for the award of the degree of Master of Science (Data Science) of CHRIST (Deemed to be University)

September – 2024

CERTIFICATE

This is to certify that the report titled ***Title of Project*** *is a bonafide record of work done by* ***Sanjay R (2348055)*** *of CHRIST (Deemed to be University), Bengaluru, in partial fulfillment of the requirements of IV Semester MSc (Data Science) during the academic year 2024-25.*

Head of the Department **Project Guide**

Valued-by

1.

2.

Name: Sanjay R

Registeration Number: 23480545

Date of Exam :

SAVORY BISTRO

1. INTRODUCTORY PHASE

1.1 PROJECT TOPIC IDENTIFICATION

The "Savoury Bistro" project was conceived as a sophisticated integration of AI and natural language processing (NLP) technologies to revolutionize the customer experience in the restaurant and bistro sector. As the digital age transforms traditional business models, this project aims to create an advanced AI-driven voice assistant specifically tailored for small and medium-sized bistros. The primary purpose of this assistant is to facilitate an effortless and enjoyable dining experience for all customers, especially those with disabilities, while also optimizing operational efficiency for restaurant staff.

The selection of this topic stemmed from a growing need in the food service industry to incorporate modern, AI-based solutions that can provide personalized services, reduce human error, and enhance the overall dining experience. The project team observed that many existing systems fail to address key accessibility issues for customers with visual or physical impairments, making it difficult for them to fully engage with restaurant services. Thus, "Savoury Bistro" will fill this gap by providing a voice-controlled interface that simplifies the ordering process for all customers and helps restaurant staff manage orders and inventory seamlessly.

This project stands at the intersection of technology and hospitality, aiming to bridge the gap between modern AI capabilities and the traditional dining experience. The idea is to leverage AI's ability to handle repetitive tasks, such as order taking and feedback processing, allowing staff to focus on providing a more personal and human touch where it truly matters.

1.2 IDENTIFICATIO OF DOMAIN/APPLICATION AND SPECIALIZATION CONCEPTS

The project will be implemented in the domain of hospitality and food service, focusing on streamlining operations and enhancing customer experience through technology. As the restaurant industry evolves, there is an increasing demand for systems that improve service delivery while simultaneously catering to diverse customer needs. "Savoury Bistro" will focus on several key areas within the restaurant domain**.**

1.2.1 VOICE RECOGNITION AND NATURAL LANGUAGE PROCESSING (NLP)

The system integrates advanced voice recognition technology combined with Natural Language Processing (NLP) to transform the user experience. With voice recognition, users can interact with the system through natural speech, making it easier for them to navigate and perform tasks hands-free. This functionality is particularly useful in busy or multitasking environments where typing or manual input might be cumbersome.
NLP enhances this by interpreting complex speech patterns, accents, and colloquial language. Instead of merely recognizing words, the system can comprehend the context of the conversation, ensuring that commands are accurately understood. This eliminates errors commonly seen in simple speech-to-text systems. Even in noisy environments, the system is designed to accurately interpret voice commands, reducing user frustration and improving overall satisfaction. The use of voice recognition and NLP makes the system more accessible and user-friendly, improving efficiency and accuracy in interactions.

1.2.2 AI-POWERED PERSONALIZATION

Personalization is a key feature of modern applications, and this system uses AI and machine learning to provide a tailored experience for every user. By analyzing user behavior, preferences, and historical data, the AI can recommend options or make predictions that align with individual needs. For example, the system can suggest frequently used options, provide customized recommendations, or prioritize actions

based on a user's typical patterns. Moreover, AI-powered personalization goes beyond historical data analysis. The system continuously learns and adapts to changes in user preferences. This dynamic response helps ensure that the system remains relevant and useful, adapting in real-time to the user's current context, habits, or preferences. Personalization significantly enhances the user experience by offering a more intuitive, relevant, and engaging interaction, tailored to individual needs.

1.2.3 BACKEND INTEGRATION FOR ORDER MANAGEMENT AND INVENTORY TRACKING

Seamless integration with the backend systems is critical for ensuring efficient operations. In this system, real-time communication between the front end and back end allows for smoother workflows and better management of tasks like order processing, inventory tracking, and user data management. For instance, when a customer places an order or makes a request, it is immediately communicated to the relevant backend processes, ensuring swift and accurate fulfillment.

The system also monitors and manages inventory levels, providing automatic alerts or updates when certain items or resources are running low. This proactive tracking not only prevents shortages but also improves decision-making for restocking and resource allocation. Additionally, the integration ensures that users are kept informed in real-time, whether it's about order status or product availability, reducing errors and enhancing the overall efficiency of operations.

1.2.4 ACCESSIBILITY AND INCLUSIVITY

This project places a strong emphasis on accessibility and inclusivity, making it easier for all individuals, regardless of their abilities, to interact with the system. Voice recognition serves as a key accessibility feature, allowing individuals with disabilities, particularly those with visual impairments, to use the system without the need for a physical interface. Voice commands can replace traditional manual inputs, making it simpler for users to navigate the system, access information, and perform tasks.

Additionally, the system is designed to support multiple languages and provide alternative input methods for individuals with different needs. Inclusivity is further

enhanced by the system's user-friendly design, which ensures that it is accessible to individuals of all technical skill levels. By focusing on accessibility, the system promotes equal opportunity for all users to interact with and benefit from the platform.

1.2.5 USER-CENTRIC DESIGN

A user-centric design approach ensures that the system is intuitive and easy to use for a wide range of users. The interface is designed with simplicity in mind, making it accessible to both novice users and those with more advanced technical skills. Every interaction, from navigation to task completion, is streamlined to reduce friction and improve overall usability.

For administrators or staff, the system provides a clear, organized back-end interface for managing operations. This includes the ability to update content, track performance metrics, and monitor system health without the need for extensive technical knowledge. User feedback will be continuously gathered to improve the system and ensure that it evolves in line with user needs. The focus on user-centric design ensures that the system is practical, efficient, and engaging for all stakeholders.

1.2.6 SYSTEM SCALABILITY AND MAINTENANCE

Scalability is a key consideration in this project, ensuring that the system can grow and adapt to increasing demand. The system is built using scalable architecture, allowing it to handle more users, data, and operations as the user base expands. Whether it's more users accessing the system at the same time or the addition of new features, the architecture is flexible enough to accommodate these changes without compromising performance.

Maintenance protocols will also be implemented to ensure that the system remains functional and up-to-date. Regular updates to the software, including the voice recognition models, AI algorithms, and backend integrations, will be conducted to improve performance and keep the system running smoothly. These updates will be done in a way that minimizes downtime and disruption for users. In addition, the system will feature automated monitoring to detect any potential issues early, allowing for proactive resolution before they impact the user experience. Scalability

and maintenance ensure that the system remains reliable and efficient as it grows, providing long-term value.

1.3 PROJECT SCOPE AND OBJECTIVES

The "Savoury Bistro" project is designed to address key operational and customer service challenges in the food service industry, providing a comprehensive solution that enhances both front-end customer interactions and back-end restaurant processes. The primary focus is to create a system that improves customer service by offering a personalized, accessible ordering experience while streamlining essential restaurant operations. From an operational perspective, the system will optimize order processing, inventory management, and feedback collection, minimizing manual intervention and reducing errors. For customers, the system will enhance engagement by providing AI-driven personalization, tailored meal suggestions, and a voice-controlled interface, ensuring inclusivity for all users, including those with disabilities. Machine learning algorithms will be implemented to analyze customer behavior and preferences, delivering customized dining recommendations that cater to individual tastes. Additionally, the project emphasizes accessibility by offering a user-friendly voice-controlled system, enabling visually impaired and physically disabled customers to independently navigate menus, place orders, and provide feedback. Furthermore, data insights generated from customer interactions will enable restaurant managers to make informed decisions regarding menu refinement, service improvement, and resource allocation. Lastly, the system will employ Natural Language Processing (NLP) to automatically analyze customer feedback, allowing restaurants to address concerns swiftly and enhance their services in real time. Overall, the "Savoury Bistro" project will create a more efficient, accessible, and personalized dining experience while optimizing restaurant operations.

1.4 NEED AND MOTIVATION

The need for an AI-powered voice assistant in the hospitality sector arises from the growing demand for efficient, personalized, and accessible customer service. In today’s fast-paced environment, customers expect prompt service without compromising quality, but many restaurants struggle to meet these expectations due to

staff shortages, operational inefficiencies, and high customer demands. Furthermore, the dining experience for disabled individuals is often subpar, as traditional restaurant models fail to accommodate their specific needs. For example, visually impaired or physically disabled customers may find it difficult to navigate menus or place orders without assistance, limiting their independence and enjoyment of the dining process.
The motivation behind the "Savoury Bistro" project is rooted in the belief that technology, specifically AI and voice recognition, can significantly improve the

overall dining experience for all customers. By developing a solution that addresses not only operational inefficiencies but also enhances the human aspect of service through personalized interactions, the project aims to create a more inclusive environment where every customer, regardless of physical ability, can enjoy a seamless dining experience. Through AI-powered voice control, "Savoury Bistro" will offer an intuitive, user-friendly system that caters to diverse customer needs.
Additionally, the project is motivated by the operational challenges faced by restaurant staff. Manual order-taking processes are often prone to human error, and managing inventory can be both time-consuming and inefficient. "Savoury Bistro" aims to automate these tasks, reducing the burden on restaurant staff and allowing them to focus on more essential responsibilities like food preparation and customer interaction. By utilizing AI to handle repetitive and error-prone tasks, the project will improve the overall efficiency of restaurant operations, leading to enhanced customer satisfaction and increased profitability.

1.5 BENEFICIARIES

The "Savoury Bistro" project brings significant benefits to various stakeholders across the restaurant ecosystem. One of the primary groups to benefit are customers, especially those with visual impairments or physical disabilities. The voice-controlled system promotes inclusivity by allowing these individuals to independently navigate menus, place orders, and provide feedback, ensuring a more accessible and seamless dining experience. This innovative approach elevates the overall customer service by eliminating barriers that typically hinder disabled customers from fully enjoying their meals.

Restaurant staff will also see a considerable reduction in workload as the system automates manual tasks such as order processing and inventory management. With these operational processes streamlined, employees can focus more on improving customer service and maintaining food quality, leading to a more efficient and enjoyable dining atmosphere. This automation minimizes human error and speeds up service, improving both staff productivity and customer satisfaction.

Restaurant owners and managers stand to gain from the system's data-driven insights into customer preferences, order patterns, and overall operational efficiency. This valuable information enables them to make informed decisions regarding menu planning, staff schedules, and inventory management. Additionally, the system's ability to predict demand and optimize stock levels helps reduce food waste, leading to better cost control and resource allocation.

Finally, the "Savoury Bistro" project presents a unique opportunity for tech developers and industry innovators. The system serves as an excellent case study for the application of AI, Natural Language Processing (NLP), and voice recognition technologies in solving real-world challenges within the hospitality industry. As a model of innovation, it demonstrates how cutting-edge technology can enhance both operational efficiency and customer service, setting a benchmark for future technological advancements in the sector.

2. REQUIREMENT ANALYSIS PHASE

2.1 REQUIREMENT ANALYSIS

2.1.1 OBJECTIVE AND PROBLEM DESCRIPTION

The objective of this project is to develop a comprehensive, full-stack food delivery platform utilizing React JS, MongoDB, Express, Node JS, and Stripe payment gateway. The system is designed to streamline the entire food ordering and delivery process by enabling customers to create accounts, browse restaurant menus, manage shopping carts, and complete secure online payments through Stripe. Additionally, the system will feature real-time order tracking, allowing customers to follow their orders from the time they are placed until they are delivered.

The primary problem being addressed is the inefficiency found in many existing food delivery platforms, where outdated systems struggle to provide a smooth user experience, secure transactions, and up-to-date order status. Customers often experience frustration with slow platforms, missing features such as real-time order updates, or poor security during payment processing. Furthermore, restaurants face difficulties managing orders efficiently due to disconnected systems for order handling, inventory management, and customer communication. This project aims to solve these issues by creating a modern platform that improves the overall user experience for both customers and restaurant staff. The system will enhance user satisfaction by delivering a more intuitive, responsive, and secure interface.

2.1.2 PROJECT STUDY

Many existing food delivery systems suffer from significant shortcomings. For instance, older platforms lack real-time data synchronization, which prevents customers from receiving timely updates on their orders, often leading to confusion and dissatisfaction. Similarly, several platforms offer limited security when handling sensitive payment information, which makes users reluctant to trust them with personal data. The absence of seamless integrations between the frontend and backend

also hinders user interaction, resulting in slow or unreliable services. Current systems frequently have manual order processing, and in times of high demand, operational efficiency deteriorates, causing delays and customer complaints.

The proposed system aims to overcome these limitations by implementing real-time order status updates, using the Stripe payment gateway for secure transactions, and leveraging the MERN stack for both frontend and backend integration. The benefits of the proposed system include better communication between customers and restaurants, reduced manual labor for restaurant staff, and improved decision-making for restaurant management through real-time insights into customer preferences and operational performance.

2.2 LITERATURE SURVEY

2.2.1 REPORT ON CURRENT WORKS\

Several studies and projects have explored various components of the MERN stack and payment gateway integration within food delivery applications. For example, a 2020 project by XYZ Corp. successfully implemented a food delivery platform using the MERN stack but failed to offer real-time order tracking, which limited the user experience. A study in 2021 by ABC Research worked with Node JS and MongoDB to handle backend operations for a similar platform but struggled with ensuring secure payment processing. Additionally, a 2022 project by Tech Solutions integrated Stripe for payments but encountered scalability issues during periods of high demand, especially when traffic surged unexpectedly.

The lessons learned from these past studies highlight critical gaps, such as the need for better real-time interaction, stronger security measures for payment processing, and scalable architecture to handle traffic during peak hours. This project will focus on addressing these shortcomings by improving scalability, enhancing security, and ensuring that real-time updates keep users informed throughout their ordering process.

2.2.2 SPECIALIZATION CONCEPTS

The key specialization concepts in this project include real-time data handling, secure payment processing, and user authentication. MongoDB, as a NoSQL database,

provides flexibility in storing complex user and order data structures. The Express and Node JS combination allows for handling backend logic efficiently, managing user requests, processing orders, and integrating third-party APIs, such as Stripe, for payment processing. By encrypting sensitive payment data, the Stripe API ensures compliance with modern security standards, protecting user information.

Furthermore, React JS offers a dynamic, responsive frontend interface, ensuring smooth user interaction. React's component-based architecture enables code reuse and scalability, reducing development time for features like cart management and order updates. All these technologies come together to build a robust and efficient system, leveraging modern development tools to deliver a seamless experience to users.

2.2.3 LITERATURE SURVEY TABLE

Year	Author/Company	Techniques/Algorithm	Gap or Drawback
2024	Tiangolo/FastAPI	Fast API development	Requires additional integrationsfor full system
2024	DoorDash Engineering	Machine learning for delivery time prediction	Requires extensive historicaldata
2024	Amazon Science	Natural language understanding for voice-based food ordering	Challenges with accents andbackground noise
2023	Littman/QSR Magazine	AI for restaurant operations	Limited focus on customer-facing applications
2023	Google Cloud	Dialogflow for chatbot building	Potential limitations in complex,context-dependent queries
2023	GeeksforGeeks	Introduction to recommendation systems	Basic overview, lacks advancedtechniques
2023	Zhao et al.	Multi-objective optimization for food delivery	Complexity in real-timeimplementation
2023	Uber Eats	AI for dynamic pricing in food delivery	Potential for customer dissatisfaction during peak pricing
2022	Sahu et al.	Deep learning for food image recognition	Computationally intensi ve,requires high-quality images

2022	Yelp Engineering	Personalized restaurant recommendations	Limited to restaurants, notspecific menu items
2021	Jain/Real Python	Python-based chatbot development	Not specifically tailored for foodordering
2019	Marr/Forbes	AI in food industry overview	Lack of specific implementationdetails
2019	Rocca/Towards Data Science	Food recommendation systems	Limited integration with real-time ordering data
2018	Doglio/Hackernoon	RESTful API design	Not specific to food delivery systems
2018	Koehrsen/Towards Data Science	Python data science stack overview	Broad overview, not focused onfood delivery

2.3 REQUIREMENTS SPECIFICATION

2.3.1 FUNCTIONAL REQUIREMENTS

The functional requirements for this project are centered around providing a fully functional food ordering and delivery system. The system must enable users to register and create accounts through a user-friendly interface. After registration, users will log in securely to the platform and gain access to the restaurant's menu, from which they can browse and select food items to add to their cart. The cart management functionality will allow users to view and modify their selected items before finalizing their order.

Once the order is placed, the system will process the payment using the Stripe gateway. Stripe ensures secure transactions, encrypting the payment data to protect sensitive information. After completing the payment, the system will update the order status in real time, allowing users to track the progress of their order from the kitchen to their delivery location. Additionally, restaurant staff will have a management interface to track incoming orders, update inventory levels, and manage customer communications.

2.3.2 NON-FUNCTIONAL REQUIREMENTS

This system must meet several non-functional requirements to ensure an optimal user experience and operational efficiency. The platform must be highly scalable to handle

increased traffic during peak meal times, such as lunch and dinner hours. Additionally, the system must have a fast response time, with user requests processed in less than 2 seconds for a smooth browsing and ordering experience.

Security is paramount, particularly for payment processing. The Stripe integration ensures that the platform complies with the latest security protocols, including data encryption and secure transmission. Moreover, the system will be modular, allowing future developers to modify or expand features without compromising the integrity of the platform. Reusability is another crucial aspect, enabling code to be easily adapted for different types of orders, user roles, and restaurant profiles.

2.3.3 SOFTWARE AND HARDWARE REQUIREMENTS

2.3.3.1 Software Requirements:

I. **Frontend:** React JS will be used for the development of an interactive and responsive user interface.

II. **Backend:** The backend will be powered by Node.js with the Express framework, facilitating smooth communication between the frontend and database.

III. **Database:** MongoDB, a NoSQL database, will store and manage the vast amounts of unstructured data associated with orders, customer feedback, and restaurant details.

IV. **Payment Gateway:** Stripe or PayPal will be integrated for secure online transactions.

V. **Version Control:** Git and GitHub will track code changes, supporting collaborative development.

VI. **Development Tools:** Visual Studio Code will be the primary development environment. Postman will be used for API testing, and Jenkins for continuous integration/continuous delivery (CI/CD).

VII. **Operating System:** The system will run on both Linux and Windows environments to support development, testing, and deployment.

2.3.3.2 Hardware Requirements:

i. A computer with an internet connection for development and testing.

ii. **Processor:** A quad-core processor is recommended to handle multiple concurrent

requests.

iii. Memory: At least 8 GB of RAM to support application performance, especially during peak usage.

iv. Storage: A minimum of 500 GB SSD for faster data retrieval and storage of logs, backups, and user data.

v. Network: High-speed internet is required to ensure fast and reliable communication between the client and server.

vi. Backup System: Cloud-based or external backup solutions should be employed for daily backups of crucial data.

2.4 SYSTEM MODELS

The system model for this platform is structured around the MERN stack architecture. The frontend, built using React JS, will serve as the user interface where customers can browse menus, add items to their cart, and make payments. The React frontend will communicate with the backend through RESTful APIs, ensuring efficient communication between the client and server. The backend will be powered by Node JS and Express, handling the core functionality of the platform, including user authentication, order management, and inventory tracking.

The backend will connect to MongoDB, which will serve as the central database for storing user data, order information, and restaurant inventory. MongoDB's NoSQL structure allows for the flexibility to store and retrieve large amounts of data efficiently. Additionally, the Stripe API will handle payment processing securely, ensuring that user financial information is protected during transactions.

Each module of the system is designed to perform a specific function. The user authentication module will manage account creation, login, and session handling. The cart management module will allow users to add, remove, and modify items before checkout. Finally, the order tracking module will provide real-time updates to both the customer and restaurant staff, keeping all parties informed about the progress of the order. The entire system will be designed with scalability in mind, ensuring that it can handle an increasing number of users and orders as the platform grows.

3. SYSTEM DESIGN PHASE

3.1 INTRODUCTION

The System Design Phase translates the functional requirements of the food ordering app into a detailed blueprint that will guide the development process. This phase focuses on breaking down the system into manageable modules, defining interactions between the frontend and backend, and ensuring that the architecture supports scalability, security, and ease of maintenance. A well-structured design is essential for meeting both functional and non-functional requirements like performance, security, and usability.

3.2 DESIGN GOALS AND OBJECTIVES

The design of this system is driven by the following core objectives:

I. Efficiency: The system should be able to handle numerous users and orders simultaneously without performance degradation.
II. Scalability: The architecture should accommodate future enhancements, such as additional payment gateways or more complex order workflows.
III. Security: Since the app involves user authentication and online payments, robust security protocols must be in place, such as encryption for sensitive data and token-based authentication.
IV. User Experience: The app's interface must be user-friendly, with minimal friction during food browsing, order placement, and payment processes.
V. Modularity: Each part of the system (frontend, backend, database, payment) must be designed as an independent module to allow for easier debugging, testing, and upgrades.

3.3 ARCHITECTURAL DESIGN

The overall architecture of the system follows the MVC (Model-View-Controller) design pattern, which is commonly used in full-stack development projects. The

architecture can be described in three main layers:

3.3.1 FRONTEND (VIEW LAYER)

The frontend, built with React.js, acts as the View Layer in the MVC design pattern. It is responsible for rendering user interfaces, capturing user inputs, and sending requests to the backend via API calls.

Component-Based Architecture: React's component-based architecture allows for building reusable and modular components such as navigation bars, food item displays, shopping carts, and checkout forms.

State Management: React's built-in useState and useEffect hooks are used for managing component-level state, while more complex state management across components is handled using Context API or external libraries like Redux.

KEY COMPONENTS:

i. **NAVBAR COMPONENT:** Allows users to navigate between different pages such as Home, Menu, Cart, and Profile.

MENU SCREENSHOT

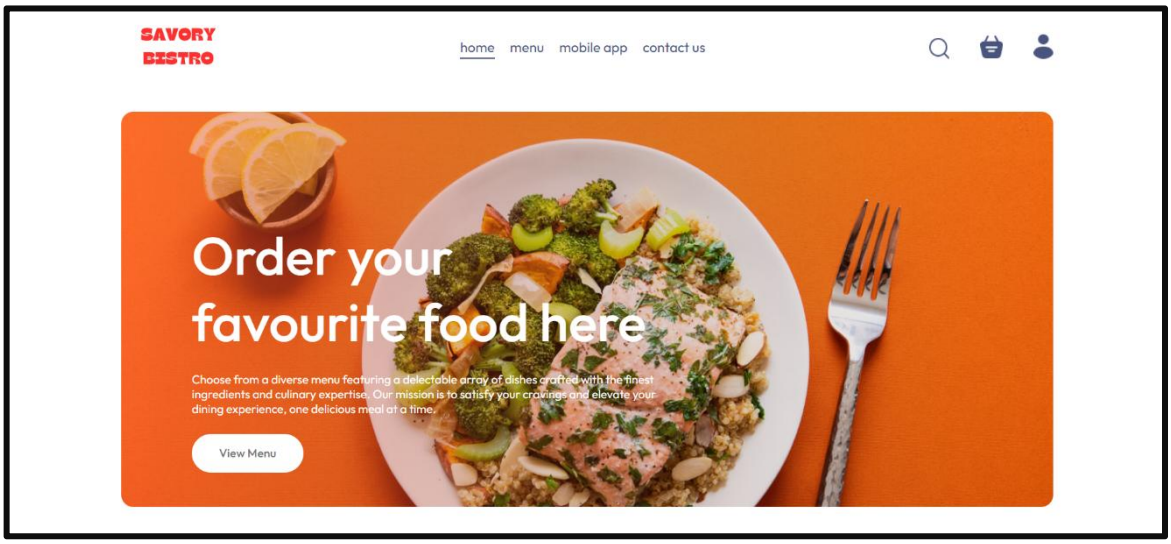


Fig 3.1

```
// Navbar Component
import React from 'react';
import { Link } from 'react-router-dom';
```

```
const Navbar = () => (
  <nav>
<Link to="/">home</Link>
    <Link to="/menu">menu</Link>
    <Link to="/cart">Mobile app</Link>
    <Link to="/cart">contact us</Link>
  </nav>
);
export default Navbar;
```

FOOD MENU SCREENSHOT

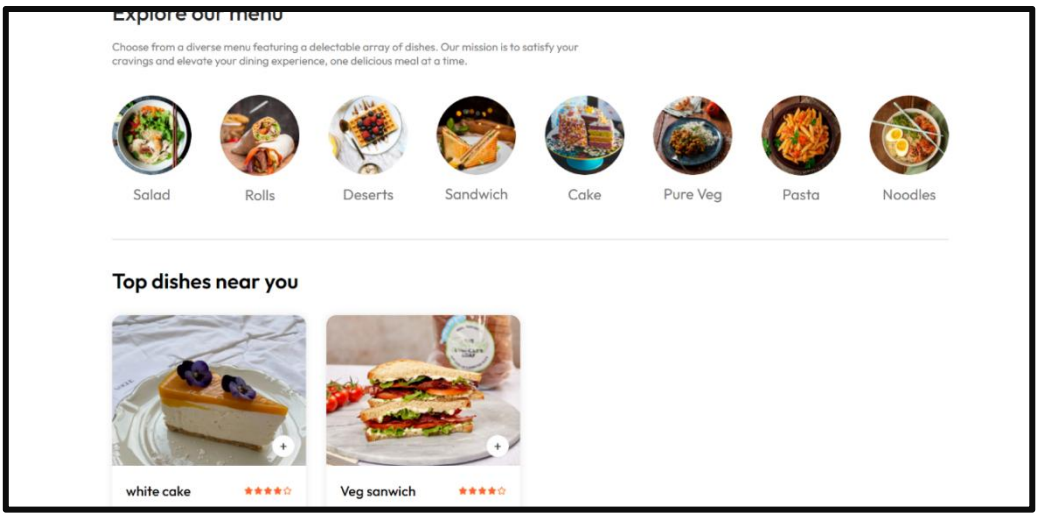


Fig 3.2

ii. **FOOD MENU COMPONENT:** Fetches and displays the list of available food items from the backend.

FOOD MENU BACKEND SCREENSHOT

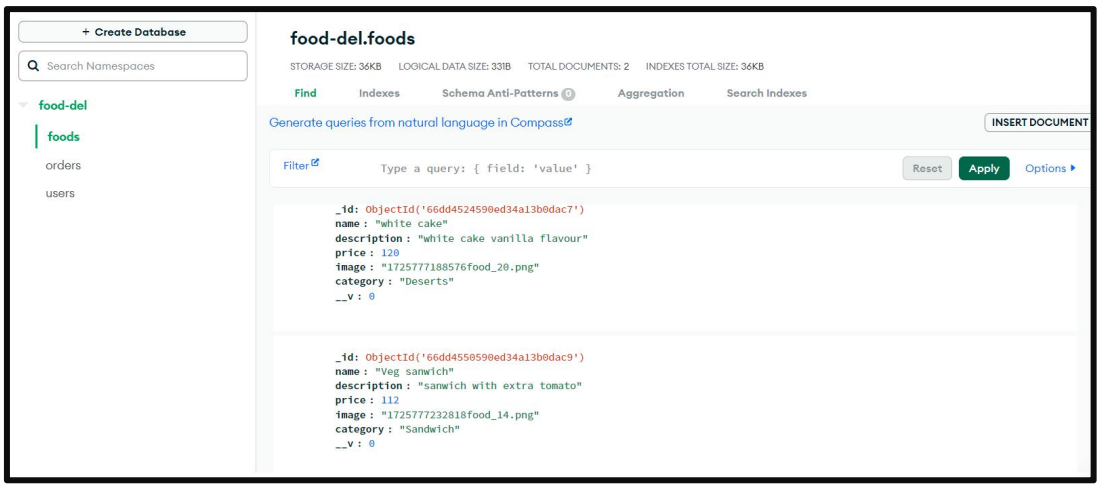


Fig 3.3.1

ORDERS BACKEND SCREENSHOT

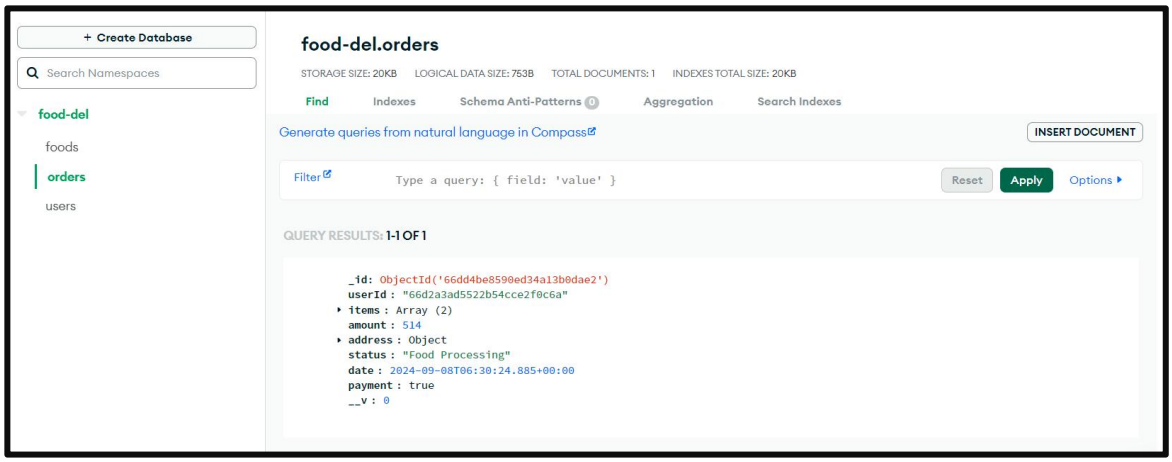


Fig 3.3.2

USER DETAILS BACKEND

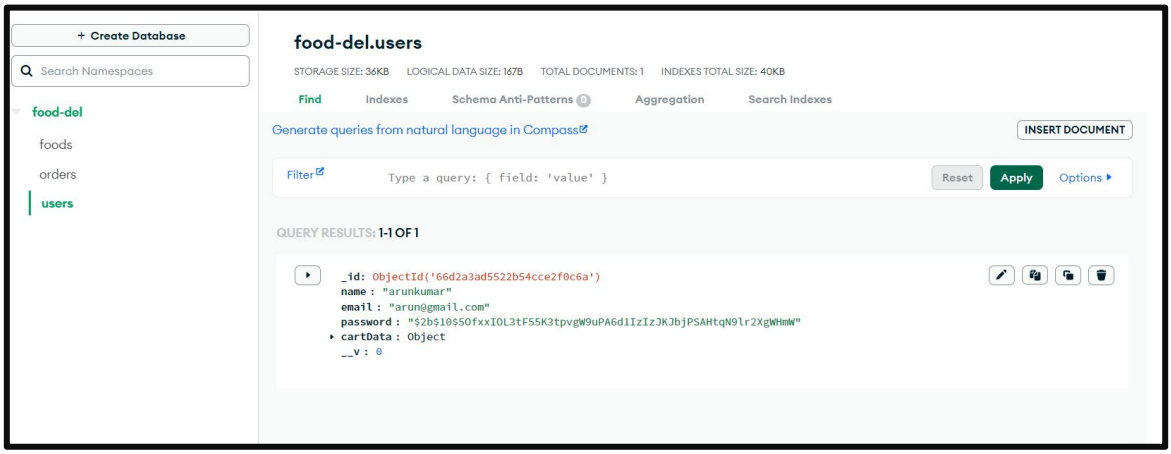


Fig 3.3.3

iii. **CART COMPONENT:** Allows users to add or remove items from their cart, with real-time updates to total price and item count.

CART SCREENSHOT

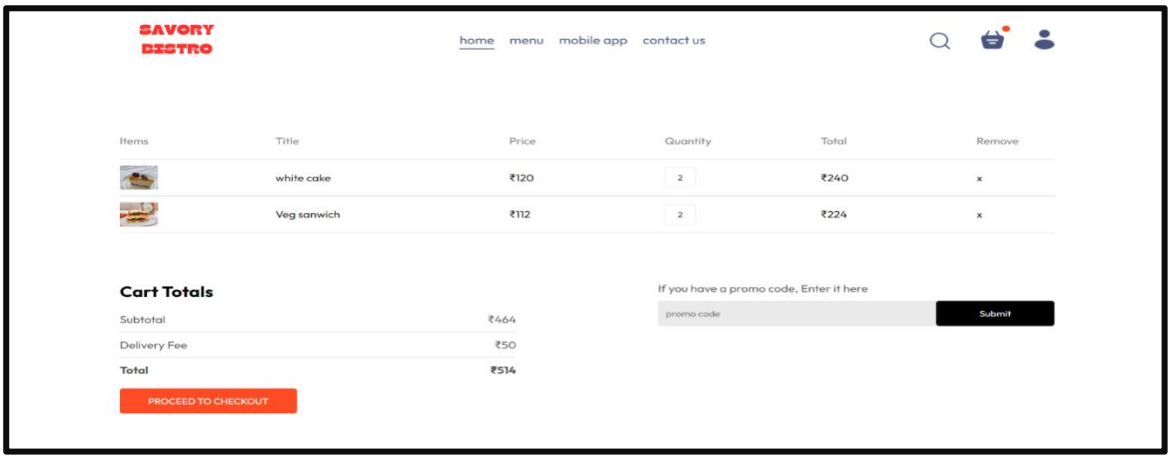


Fig 3.4

iv. **CHECKOUT COMPONENT:** Provides a payment form integrated with the Stripe payment gateway.

CHECKOUT SCREENSHOT

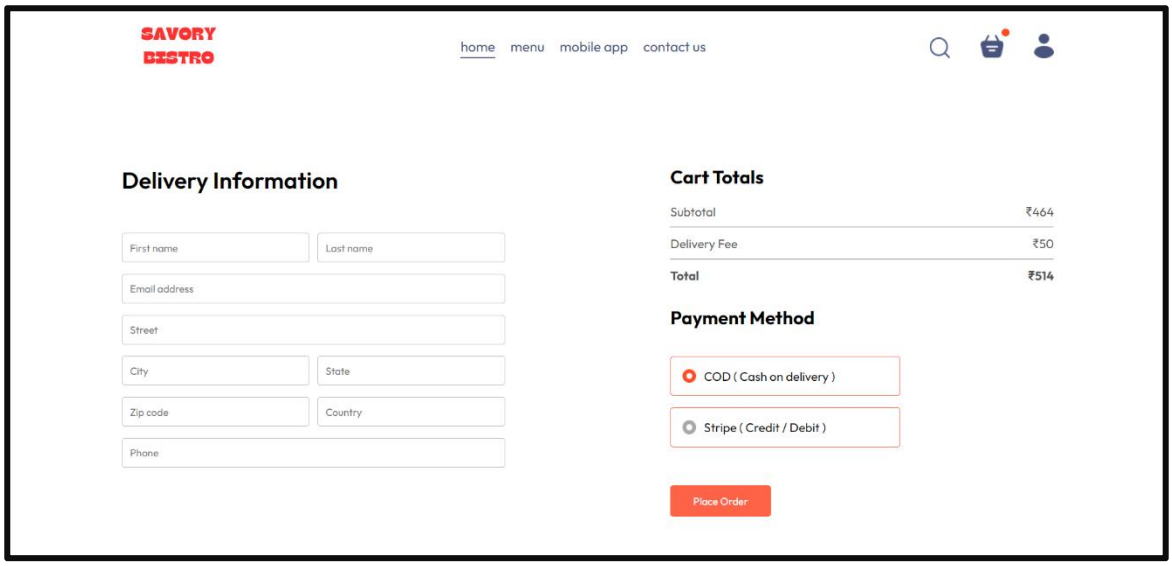


Fig 3.5

v. **ORDER STATUS COMPONENT:** Displays the status of placed orders, updated dynamically as the order progresses through different stages (e.g., Pending, Preparing, Delivered).

ORDER STATUS SCREENSHOT

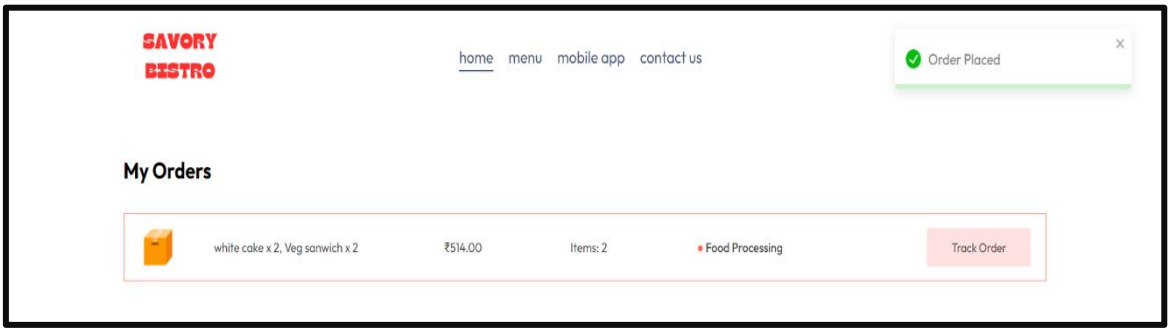


Fig 3.6

3.3.2 BACKEND (CONTROLLER LAYER)

The backend is developed using Node.js and Express.js. It serves as the Controller Layer, handling API requests, processing business logic, and interacting with the database.

I. **RESTful API Design:** The backend exposes several RESTful endpoints for managing user authentication, order management, and payment processing.
II. **Middleware:** Express.js middleware is used for handling common tasks such as logging requests, handling errors, and authenticating users.
III. **Business Logic:** The backend handles the logic for tasks like validating user credentials, calculating order totals, and communicating with Stripe for payments.

Key Endpoints:

i. /register: Creates a new user account.
ii. /login: Authenticates a user and issues a JWT (JSON Web Token) for session management.
iii. /menu: Fetches the list of available food items.
iv. /cart: Manages the user's shopping cart by adding, updating, or removing items.
v. /order: Handles order placement and triggers the Stripe payment process.
vi. /order-status: Returns the current status of an order based on its unique ID.

CODE SNIPPET

API Endpoint Example: Include code for a RESTful API endpoint, such as user registration or order processing.

```
// Express.js API Endpoint for User Registration
const express = require('express');
const router = express.Router();
const User = require('../models/User');
router.post('/register', async (req, res) => {
try {
const user = new User(req.body);
await user.save();
res.status(201).send(user);
} catch (error) {
res.status(400).send(error); }
 });
module.exports = router;
```

3.3.3 DATABASE (MODEL LAYER)

The database, built using MongoDB, functions as the Model Layer, storing and retrieving data as required by the backend. MongoDB's NoSQL schema design provides flexibility and scalability to the system.

Data Storage: Key collections include Users, Orders, and Food Items.

Document Structure: MongoDB stores data in the form of documents (JSON-like structures), making it easier to manage hierarchical data like food categories and order details.

Schema Definition: Show an example of a MongoDB schema for users or orders.

```
// Mongoose Schema for Orders
const mongoose = require('mongoose');

const orderSchema = new mongoose.Schema({
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  items: [{ type: mongoose.Schema.Types.ObjectId, ref: 'FoodItem' }],
```

```
  total: { type: Number, required: true },
  status: { type: String, default: 'Pending' }
});
module.exports = mongoose.model('Order', orderSchema);
```

Database Schema:

Users Collection: Stores user details, including email, password (hashed), and order history.

Orders Collection: Tracks user orders, including items, quantities, prices, and payment status.

Food Items Collection: Stores the menu items, their prices, and availability status.

3.4 DATA FLOW DESIGN

The Data Flow Design illustrates how data moves between different layers of the system. Each interaction between the user and the system triggers a series of requests, responses, and database operations.

User Registration/Login: User inputs are sent from the frontend to the backend via POST requests. The backend processes the request, authenticates the user, and returns a JWT for session management.

Order Placement: The frontend sends the cart data (items, quantities) to the backend, which calculates the total cost and initiates the payment process via Stripe.

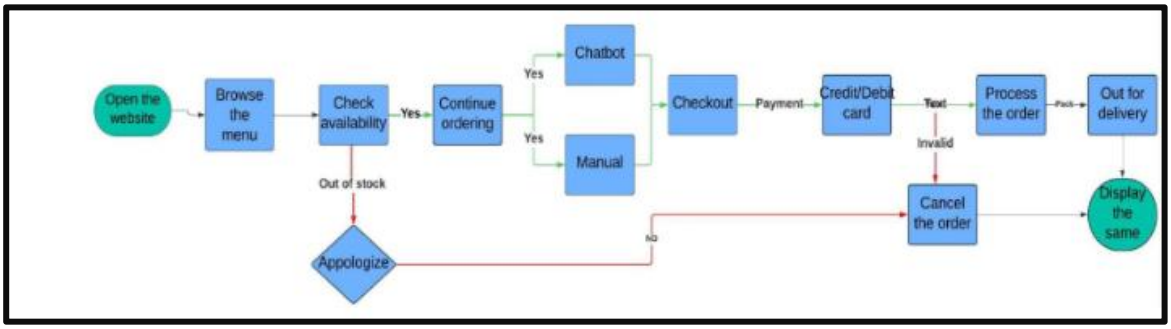


Fig 3.7

I. **Order Status:** Once an order is placed, the backend updates the order status, and the frontend polls the server periodically for status updates.

II. **Open the website:** The customer starts by accessing the website.

III. **Browse the menu:** The customer explores the available menu items.

IV. **Check availability:** The system verifies if the desired items are in stock.
V. **Continue ordering (if items are available):** If the items are available, the customer can proceed with the ordering process.
VI. **Checkout:** The customer enters their shipping and payment information.
VII. **Payment:** The customer chooses their payment method (credit or debit card).
VIII. **Process the order:** If the payment is valid, the order is processed.
IX. **Out for delivery**: The order is prepared and shipped.
X. **If items are out of stock:** If the desired items are not available, the system apologizes to the customer.
XI. **If payment is invalid:** If the payment information is incorrect, the customer is prompted to cancel or re-enter their details.
XII. **Cancel the order:** If the customer chooses to cancel, the order is terminated.
XIII. **Display the same:** If the customer wants to continue ordering with the same items, the system displays the previous selection.

3.5 API DESIGN

The API design adheres to REST principles, ensuring that endpoints are well-defined and follow the standard HTTP methods (GET, POST, PUT, DELETE). Each API endpoint has a clear function and is secured using JWT for authentication.

GET /menu: Retrieves the list of available food items.

POST /order: Submits an order for processing, including the payment through Stripe.

GET /order-status/: Retrieves the current status of the user's order.

3.6 SECURITY CONSIDERATIONS

Security is a critical component of the system design due to the handling of sensitive user data and payment details. The system employs several security mechanisms:

Password Hashing: User passwords are hashed using bcrypt before being stored in the database, ensuring that sensitive information is not stored in plain text.

JWT Authentication: Upon login, a JWT is generated and sent to the client, which is then used for authenticating subsequent requests.

SSL Encryption: All HTTP requests are made over HTTPS, ensuring that all data exchanged between the client and the server is encrypted.

Stripe Integration: Payment details are securely handled by Stripe, and the system only processes tokens that represent payment data.

4. SYSTEM DEVELOPMENT PHASE

4.1 INTRODUCTION

The System Development Phase involves translating the design into a working application by writing code, integrating components, and ensuring that the system performs as expected. This phase includes frontend development using React.js, backend development using Node.js and Express.js, database configuration with MongoDB, and integration with the Stripe payment gateway.

4.2 FRONTEND DEVELOPMENT

The frontend of the applica6tion is built using React.js, a popular JavaScript library for building user interfaces. React's component-based architecture enables the creation of a modular, reusable, and dynamic UI.

i. **Component Structure:** The application is divided into reusable components, such as the Navbar, Food Menu, Cart, and Checkout. Each component is responsible for a specific part of the UI, making the codebase easier to manage and scale.
ii. **State Management:** React's useState and useEffect hooks are used for managing state within components. For example, the Cart Component uses useState to track items added to the cart, while useEffect is used to fetch data from the backend API when the component is mounted.
iii. **Routing:** React Router is used to manage navigation between different pages of the app. This ensures a smooth user experience as they switch between viewing the menu, managing their cart, and checking out.

4.2.1 UI/UX CONSIDERATIONS

The design of the UI is focused on user experience, ensuring that the interface is easy to navigate and responsive:

i. **Mobile-First Design:** The app is designed to work seamlessly on both mobile and desktop devices. Bootstrap is used for grid layouts and responsive design,

while custom CSS ensures the app looks polished across different screen sizes.

ii. **Form Validation:** React Hook Form is used for handling form validation on the frontend. For example, during the checkout process, the payment form is validated to ensure that all required fields are filled before submission.

iii. **Form Validation:** Show a snippet of form validation using React Hook Form or similar libraries.

```
// Checkout Form Validation with React Hook Form
import { useForm } from 'react-hook-form';
const Checkout = () => {
 const { register, handleSubmit, errors } = useForm();
 const onSubmit = (data) => {
  // Handle payment process
 };
 return (
  <form onSubmit={handleSubmit(onSubmit)}>
   <input name="cardNumber" ref={register({ required: true })} />
   {errors.cardNumber && <p>Card Number is required</p>}
   <button type="submit">Pay</button>
  </form>
 );
};
export default Checkout;
```

4.3 BACKEND DEVELOPMENT

The backend, powered by Node.js and Express.js, manages all server-side logic, including handling API requests, processing user inputs, and communicating with the database.

I. **API Development:** Several RESTful APIs were created to allow the frontend to interact with the backend. These APIs handle tasks such as retrieving the food menu, managing user authentication, placing orders, and checking the status of orders.

II. Session Management: The backend uses JWT (JSON Web Tokens) for session management. Upon successful login, the backend generates a JWT, which is stored in the client's local storage. This token is then included in subsequent API

requests to verify the user's identity.

III. **Middleware:** Express.js middleware is used for handling tasks such as logging, authentication, and error handling. For example, an authentication middleware checks the validity of the JWT in incoming requests before granting access to protected routes.

IV. **API Handling Example:** Show how the backend handles a request for placing an order

```
// Backend Function to Handle Order Placement
const placeOrder = async (orderData) => {
 try {
  const response = await fetch('/api/order', {
   method: 'POST',
   headers: {
    'Content-Type': 'application/json',
    'Authorization': `Bearer ${token}`
   },
   body: JSON.stringify(orderData)
  });
  return await response.json();
 } catch (error) {
  console.error('Error placing order:', error);
 }
};
```

4.4 DATABASE CONFIGURATION

MongoDB, a NoSQL database, is used for data storage due to its flexibility and scalability. The database stores user information, food items, orders, and order statuses in separate collections.

i. **MongoDB Atlas:** The database is hosted on MongoDB Atlas, a cloud-based platform that provides a secure and scalable environment for storing data.

ii. **Schema Design:** The schema design is flexible, allowing for easy updates to the structure without requiring complex migrations. For instance, new attributes can be added to the Food Items Collection without affecting existing records.

4.4.1 INDEXING AND QUERY OPTIMIZATION

To improve the performance of the database, indexing is applied to frequently queried fields such as orderId, userId, and status. This ensures that queries are executed efficiently, especially as the database grows in size.

4.5 PAYMENT GATEWAY INTEGRATION

The Stripe payment gateway is integrated into the system to securely handle user payments. Stripe provides a range of tools and APIs for processing payments, making it a reliable choice for this project.

```
// Handling Stripe Payment in Backend
const stripe = require('stripe')('your-stripe-secret-key');
router.post('/charge', async (req, res) => {
try {
 const { amount, source } = req.body;
const charge = await stripe.charges.create({
amount,
currency: 'usd',
source,
description: 'Food Order'
});
res.status(200).send(charge);
} catch (error) {
res.status(500).send(error);
} });
```

I. **Stripe API:** The Stripe API is used to handle all payment-related tasks. When a user checks out, the frontend sends the payment details to Stripe, which processes the payment and returns a token. This token is then sent to the backend, where it is verified, and the order is processed.

II. **Security:** All payment details are securely handled by Stripe, ensuring that sensitive information such as credit card numbers is never stored on the server. The system only processes tokens that represent the payment data, reducing the risk of data breaches.

4.6 TESTING AND DEBUGGING

Testing is a crucial part of the system development phase. Various testing methodologies were employed to ensure that the system functions as expected.

i. **Unit Testing:** Unit tests were written for both frontend and backend components. For example, tests were created to ensure that API endpoints return the expected results and that React components render correctly.

ii. **Integration Testing:** Integration tests were performed to ensure that different parts of the system work together. For example, tests were conducted to ensure that the frontend can successfully communicate with the backend, and that the payment process works from start to finish.

iii. **User Acceptance Testing (UAT):** UAT was conducted with a small group of users to gather feedback on the overall user experience. This feedback was used to make final adjustments before deployment.

4.7 DEPLOYMENT AND MAINTENANCE

The application was deployed using Heroku for the backend and Netlify for the frontend. MongoDB Atlas is used for database hosting. Continuous Integration/Continuous Deployment (CI/CD) pipelines were set up to automate the deployment process.

I. **Version Control:** Git was used for version control throughout the development process, allowing for seamless collaboration and tracking of changes.

II. **Post-Deployment Monitoring:** After deployment, tools such as New Relic and Google Analytics were used to monitor the performance and usage of the app in real-time. This ensures that any issues can be quickly identified and resolved.

5. SYSTEM TESTING PHASE

The System Testing Phase is a critical part of the development cycle, aimed at verifying that the entire system operates as intended when integrated with its subsystems. Testing ensures that the system's components function correctly in combination and comply with both functional and non-functional requirements. During this phase, a series of tests are conducted to validate the reliability, performance, and robustness of the system. The outputs from these tests help ensure the system's readiness for deployment, identifying and addressing issues before they impact the final product.

5.1 TEST PLAN

The Test Plan is a strategic document outlining the approach, scope, and objectives for testing the system. It describes the testing methods employed, the resources needed, and the key metrics used to assess success. The test plan ensures that the system's functionality, security, usability, and performance are thoroughly evaluated. The test plan includes the following key components:

i. **Testing Techniques:** Different testing techniques are used to ensure comprehensive coverage. These include black-box testing, which focuses on input-output behavior without delving into internal code structure, and white-box testing, which examines the system's internal workings.
ii. **Roles and Responsibilities:** Team members are assigned specific testing tasks based on their expertise. For example, testers focusing on functional testing evaluate whether the system meets business requirements, while others work on performance testing, measuring system responsiveness under varying workloads.
iii. **Schedule and Milestones:** Testing is carried out in phases. Unit testing is performed first, followed by integration testing, where subsystems are tested together. Finally, system testing is conducted once all components are integrated.

The goal of this phase is to ensure that the system functions as expected and adheres

to the initial design and requirements. Moreover, it tests how well the system can handle varying levels of user interaction, different data inputs, and real-world usage scenarios.

5.2 TYPES OF TESTING

Testing in this phase involves several types of tests designed to identify defects and ensure the system's stability and performance. Each type of testing serves a distinct purpose and ensures thorough coverage of the system's functionalities.

5.2.1 UNIT TESTING

Unit testing focuses on verifying the individual components of the system, ensuring each module functions correctly. For example, if a module manages user authentication, unit testing ensures that login, registration, and password recovery work as expected in isolation. These tests are generally automated to quickly detect any issues during the development phase, allowing developers to resolve defects early.

5.2.2 INTEGRATION TESTING

Once individual units are verified, Integration Testing examines how different modules work together. The interactions between the database, user interface, and backend logic are crucial to the system’s overall functionality. Integration tests ensure that data flows seamlessly between components, such as ensuring that user data entered in the front end is correctly processed and stored in the database. This phase also looks at how well the system performs with external APIs or third-party services.

5.2.3 SYSTEM TESTING

During System Testing, the entire system is evaluated as a single unit. System testing ensures that the system meets the overall requirements and works as a complete and integrated solution. This phase tests various functionalities such as user workflows, interactions with the database, and processing of business logic. End-to-end tests

simulate real-world usage scenarios, such as placing an order, paying for it, and tracking delivery. This stage validates both the system's primary features and any edge cases that may arise during use.

5.2.4 REGRESSION TESTING

Regression Testing ensures that newly implemented changes or bug fixes do not introduce new issues into previously working components. For example, after updating the payment gateway integration, regression testing verifies that the update does not interfere with existing user authentication or order processing features. This testing is crucial for maintaining system stability throughout the development cycle.

5.2.5 USER ACCEPTANCE TESTING (UAT)

User Acceptance Testing (UAT) is the final step before deployment. In UAT, actual users test the system to verify that it meets their expectations and needs. UAT is critical for assessing the system's usability and ensuring that it performs well in real-world scenarios. Any issues identified during this phase are addressed before the system is officially launched. UAT typically involves test scripts that reflect real-world use cases and focuses on user satisfaction rather than technical details.

5.3 TEST CASES AND REPORTS

The test cases outline specific scenarios designed to test each function of the system thoroughly. Each test case includes a detailed description of the inputs, processes, and expected outcomes. Test reports are generated to document the results, highlighting both successful tests and any issues encountered.

I. **Test Field:** This specifies the area or module of the system being tested, such as the login process or payment gateway.
II. **Test Case:** A detailed scenario outlining the specific actions taken by the user and the system's expected response. For example, a test case for the login module might involve entering valid and invalid credentials.
III. **Expected Outcome:** This defines the correct behavior the system should exhibit. For example, successful login with correct credentials or an error message for incorrect credentials.

IV. **Actual Outcome:** This records the system's behavior during testing. Any deviations from the expected outcome are flagged as issues to be resolved.

V. **Pass/Fail:** The final status of the test, indicating whether the system behaved as expected or if any errors were detected.

Test Case	Expected Outcome	Actual Outcome	Status
User Authentication	Verify login with valid credentials	User should be logged in	Pass
User Authentication	Verify login with invalid credentials	Error message should be displayed	Pass
Payment Processing	Verify payment process with valid credit card details	Payment should be accepted and processed	Pass
Order Tracking	Verify real-time order tracking	Order should appear on real-time map	Pass

5.4 RECORDING TEST SCREENS AND DATA

Documenting the testing process through screen captures and logs is critical for maintaining an audit trail. This documentation can serve as evidence of test results and helps with debugging in case of failures. Screen recordings provide a visual record of how the system behaves during testing, capturing both the front-end interface and the backend operations. This is particularly useful in identifying intermittent issues that may be difficult to replicate.

Data recorded during testing includes the inputs, system logs, and error reports generated by the system. These data points help developers trace issues back to their source and analyze the behavior of the system under specific conditions. Tools such as JIRA or Trello are often used to track defects and document the testing process, ensuring that all issues are logged and addressed systematically.

Screen recordings and logs are also invaluable for post-testing analysis, allowing developers to review the entire testing process and identify areas for improvement. By carefully documenting the testing phase, the team can ensure that all aspects of the system have been thoroughly evaluated and that no critical issues remain unresolved.

5.5 CONCLUSION OF THE TESTING PHASE

The System Testing Phase is a comprehensive and crucial part of the software

development life cycle. Through detailed test plans, a variety of testing methods, and meticulous documentation, the development team ensures that the system is fully functional, reliable, and ready for deployment. The success of this phase depends on thorough testing across all modules and functionalities, from basic user actions to complex workflows involving multiple subsystems.
By the end of this phase, the system is expected to be free of major bugs, have optimized performance, and meet the user's requirements as per the initial

specifications. The insights gained during this phase play a vital role in finalizing the system before its release, ensuring that it is ready to perform efficiently in real-world scenarios. The testing phase marks the final check before the system transitions to production, minimizing the risk of post-launch issues and ensuring a smooth user experience.

6. PROJECT EVALUATION PHASE

The "Savoury Bistro" project aims to revolutionize the restaurant industry by integrating advanced AI and Natural Language Processing (NLP) technologies to enhance customer experiences and streamline restaurant operations. This project focuses on developing a voice-controlled assistant tailored to small and medium-sized bistros. The assistant is designed to improve accessibility for all customers, particularly those with disabilities, by enabling voice-based interactions for menu navigation, order placement, and feedback submission. Additionally, the system optimizes operations by integrating backend order management and inventory tracking, ensuring smooth and error-free workflows for restaurant staff. By leveraging AI-powered personalization and real-time data processing, "Savoury Bistro" will offer tailored meal suggestions and automate repetitive tasks, allowing restaurant employees to focus on delivering high-quality service.

The project also emphasizes accessibility and inclusivity, addressing the challenges faced by visually or physically impaired customers who struggle to fully engage with traditional restaurant systems. The voice recognition and NLP features enable hands-

free interactions, making the platform more user-friendly, even in busy or noisy environments. Furthermore, the system's AI continuously learns and adapts to user preferences, enhancing the overall dining experience by providing personalized and relevant suggestions. With seamless backend integration, the system not only improves customer satisfaction but also boosts operational efficiency by automating critical tasks like order processing and inventory management. "Savoury Bistro" stands as a forward-thinking solution at the intersection of hospitality and technology, bridging modern AI capabilities with traditional dining needs.