Software Engineering Project Report

by

Amogh K Umesh (670)

Ayush Kumar (683)

FastFleet Food Delivery System

Submitted to

Dr. Oishila Bandyopadhyay

For the partial fulfillment of the course (CSC702)

Under Department of Computer Science and Engineering

Autumn Semester - 2024

Table of Contents

Software Requirements Specification (SRS) for FeastFleet

Introduction

Purpose

This document provides a detailed description of the FeastFleet system, including its objectives, features, and requirements. It is intended for developers, project managers, and stakeholders involved in the project.

Scope

FeastFleet is a web-based application designed to facilitate the ordering and delivery of food items. The system includes three main components:

- Frontend: User-facing interface for browsing and ordering food.
- Admin: Admin interface for managing food items, orders, and users.
- Backend: Server-side logic and database management.

Definitions, Acronyms, and Abbreviations

- API: Application Programming Interface
- JWT: JSON Web Token
- SRS: Software Requirements Specification
- UI: User Interface

References

- MDN Docs

Overall Description

Product Perspective

FeastFleet is a standalone web application that integrates with MongoDB for database management and Stripe for payment processing.

Product Functions

- User registration and login
- Browsing and searching for food items
- Adding items to the cart

- Placing and tracking orders
- Admin management of food items, orders, and users

User Classes and Characteristics

- End Users: Individuals who browse and order food items.
- Admin Users: Individuals who manage the food items, orders, and user data.

Operating Environment

- Frontend: Runs in modern web browsers (Chrome, Firefox, Safari).
- Backend: Runs on Node.js server.
- Database: MongoDB

Design and Implementation Constraints

- Must use React for frontend development.
- Must use Express.js for backend development.
- Must use MongoDB for database management.
- Must use Stripe for payment processing.

Assumptions and Dependencies

- Users have access to the internet.
- Users have modern web browsers installed.
- MongoDB and Stripe services are available and operational.

System Features

User Registration and Login

- Description: Allows users to register and log in to the system.
- Functional Requirements:
    - Users can register with a name, email, and password.
    - Users can log in with their email and password.
    - Passwords are hashed using bcrypt.
    - JWT is used for session management.

Browsing and Searching Food Items

- Description: Allows users to browse and search for food items.
- Functional Requirements:
  - Users can view a list of food items.
  - Users can search for food items by name or category.
  - Food items display name, description, price, and image.

Cart Management

- Description: Allows users to add, remove, and view items in their cart.
- Functional Requirements:
  - Users can add items to their cart.
  - Users can remove items from their cart.
  - Users can view the contents of their cart.

Order Placement and Tracking

- Description: Allows users to place and track orders.
- Functional Requirements:
  - Users can place orders with items in their cart.
  - Users can track the status of their orders.
  - Orders are processed using Stripe for payment.

Admin Management

- Description: Allows admin users to manage food items, orders, and users.
- Functional Requirements:
  - Admins can add, edit, and delete food items.
  - Admins can view and update order statuses.
  - Admins can manage user data.

External Interface Requirements

User Interfaces

- Frontend: User interface for browsing and ordering food.
- Admin: Admin interface for managing the system.

Hardware Interfaces

- No specific hardware interfaces required.

Software Interfaces

- MongoDB: Database management.
- Stripe: Payment processing.

Communications Interfaces

- HTTP/HTTPS: For communication between frontend, backend, and external services.

System Requirements

Functional Requirements

- User Registration and Login: Implemented in userController.js.
- Browsing and Searching Food Items: Implemented in Home.jsx.
- Cart Management: Implemented in cartController.js.
- Order Placement and Tracking: Implemented in orderController.js.
- Admin Management: Implemented in App.jsx.

Nonfunctional Requirements

- Performance: The system should handle up to 1000 concurrent users.
- Security: User data should be encrypted, and secure authentication should be implemented.
- Usability: The UI should be intuitive and easy to navigate.
- Reliability: The system should have a high reliability.

Other Nonfunctional Requirements

Performance Requirements

- The system should respond to user actions quickly.

Safety Requirements

- The system should ensure data integrity and prevent unauthorized access.

Security Requirements

- Use HTTPS for all communications.
- Store passwords securely using bcrypt.

- Use JWT for secure session management.

Software Quality Attributes

- Maintainability: The codebase should be modular and well-documented.
- Scalability: The system should be able to scale horizontally to handle increased load.

Business Rules

- Users must be registered and logged in to place orders.
- Admin users have additional privileges for managing the system.

Software Design Document (SDD) for FeastFleet

Introduction

Purpose

This document provides a detailed design of the FeastFleet system, including its architecture, components, and interactions. It is intended for developers and system architects.

Scope

The design covers the frontend, backend, and database components of the FeastFleet system.

Definitions, Acronyms, and Abbreviations

- API: Application Programming Interface
- JWT: JSON Web Token
- UI: User Interface

References

- MDN

System Architecture

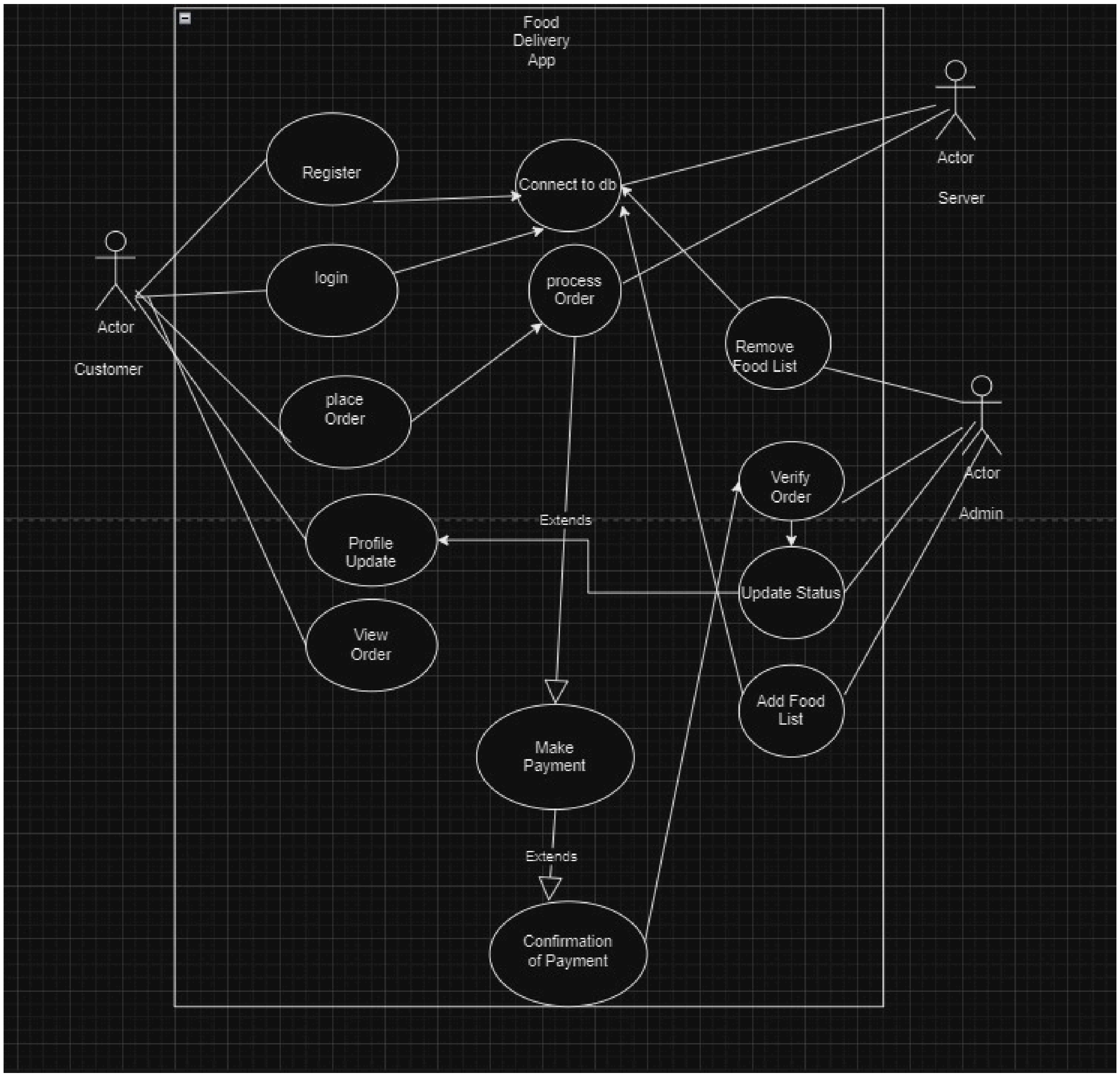
Overview

The FeastFleet system follows a three-tier architecture:

1. Frontend: Built with React, it provides the user interface for customers and admins.
2. Backend: Built with Node.js and Express.js, it handles business logic and API endpoints.
3. Database: MongoDB is used for data storage.

Component Diagram

```
+-----------------+       +-----------------+
+-----------------+
|    Frontend     | <---> |     Backend     | <---> |     Database
|
|     (React)     |       |(Node.js/Express)|       |    (MongoDB)
|
+-----------------+       +-----------------+
+-----------------+
```



The Software Development Life Cycle (SDLC) model used for the FeastFleet system is the Agile methodology. Agile was chosen for flexibility, and rapid development.

1. Planning:
   - Requirements were gathered through discussions among teammates.

2. Design:
   - High-level architecture and detailed design documents (SRS and SDD) were created.
   - Design reviews were conducted to ensure alignment with requirements.

3. Development:
   - Development was carried out in iterative sprints, typically lasting 3-4 days.
   - Features were developed incrementally, with continuous integration.

4. Testing:
   - Each sprint included unit testing and acceptance testing.
   - We also tested the whole code at the end.

5. Deployment:
   - Continuous deployment practices were followed, with regular releases to a staging environment.
   - Final deployment to production was done after thorough testing and stakeholder approval.

6. Maintenance:
   - Post-deployment, the system was monitored for any issues.
   - Bug fixes and minor enhancements were handled in subsequent sprints.

Detailed System Design

Frontend

- Technology: React
- Components:
  - Home: Displays food items.
  - Cart: Manages items added by the user.
  - Order: Handles order placement and tracking.
  - Admin: Manages food items, orders, and users.

Backend

- Technology: Node.js, Express.js
- Controllers:
    - userController.js: Manages user registration, login, and profile.
    - cartController.js: Manages cart operations.
    - orderController.js: Manages order placement and tracking.
    - adminController.js: Manages admin operations.

Database

- Technology: MongoDB
- Collections:
    - Users: Stores user information.
    - FoodItems: Stores food item details.
    - Orders: Stores order details.
    - Cart: Stores cart details.

Database Design

Users Collection

```
{
  "_id": "ObjectId",
  "name": "string",
  "email": "string",
  "password": "string"
}
```

FoodItems Collection

```
{
  "_id": "ObjectId",
  "name": "string",
  "description": "string",
  "price": "number",
  "image": "string"
}
```

Orders Collection

```
{
  "_id": "ObjectId",
  "userId": "ObjectId",
  "items": [
    {
      "foodItemId": "ObjectId",
      "quantity": "number"
    }
  ],
  "status": "string",
  "totalPrice": "number"
}
```

Cart Collection

```
{
  "_id": "ObjectId",
  "userId": "ObjectId",
  "items": [
    {
      "foodItemId": "ObjectId",
      "quantity": "number"
    }
  ]
}
```

User Interface Design

Frontend Components

- Home Page: Displays a list of food items with search functionality.
- Cart Page: Displays items added to the cart with options to update quantities or remove items.
- Order Page: Displays order details and tracking information.
- Admin Page: Provides interfaces for managing food items, orders, and users.

Security Design

Authentication

- JWT: Used for user authentication and session management.

- bcrypt: Used for hashing passwords.

Authorization

- Role-based Access Control: Admin users have additional privileges.

Data Protection

- HTTPS: All communications are encrypted.
- Data Encryption: Sensitive data is encrypted in the database.

Error Handling and Logging

Error Handling

- Frontend: Displays user-friendly error messages.
- Backend: Returns appropriate HTTP status codes and error messages.

Logging

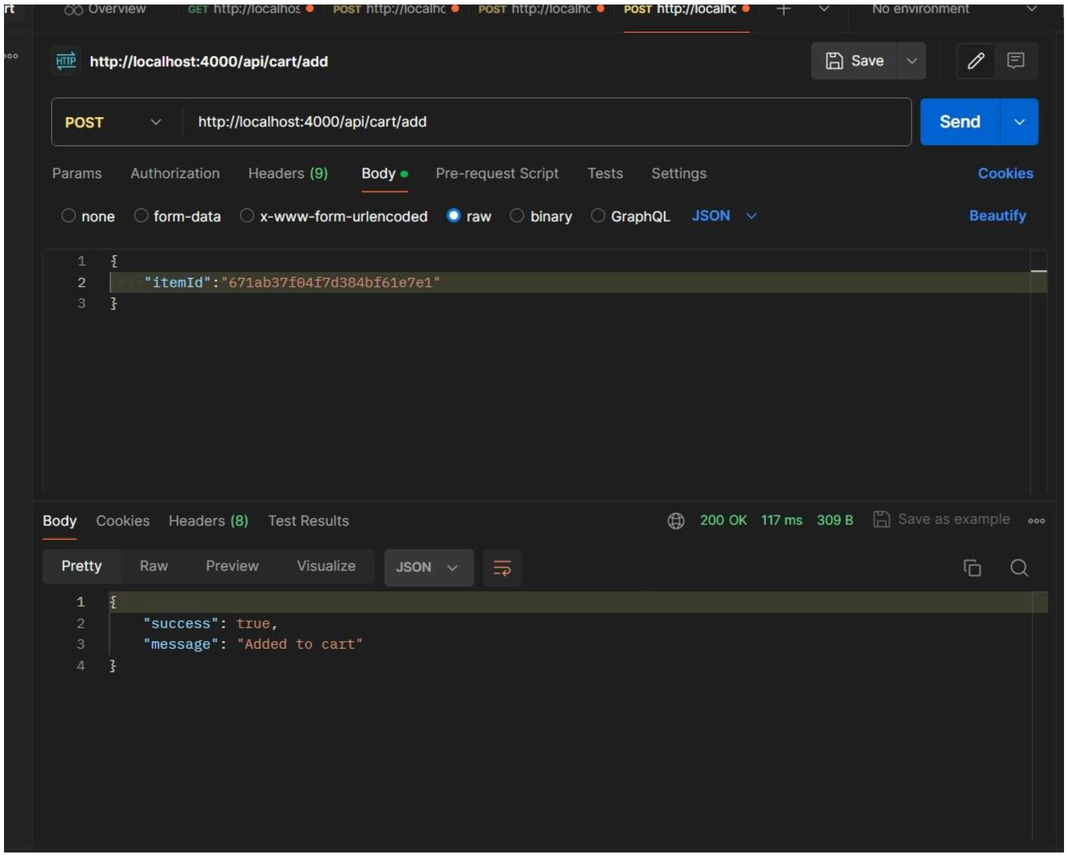
- Backend: Logs errors and important events using a logging library (e.g., Winston).

Testing

Unit testing was done with PyTest and Postman.
Integration Test was done via Big Bang method where we tested the whole system as a whole.

Test Case	Description	Data	Expected Result	Actual Result
Login User - Successful	User logs in with valid email and password	{"email": "email", "password": "password"}	Token is generated and returned in response	Matches Expected
Login User - Invalid Email	User attempts to log in with an email that does not exist	{"email": "email_that_doesnt_exist", "password": "password"}	{"success": False, "message": "User doesn't exist."}	Matches Expected
Login User - Invalid Password	User logs in with incorrect password	{"email": "email", "password": "wrong__password"}	{"success": False, "message": "Invalid credentials"}	Matches Expected
Register User - Successful	User registers with valid details	{"name": "name", "email": "email", "password":	{"success": True, "token": "generated_token"}	Matches Expected

		"password"}		
Register User - Missing Fields	User tries to register without providing all required fields	{"name": "", "email": "", "password": ""}	{"success": False, "message": "Please enter all fields."}	Matches Expected
Register User - User Already Exists	User tries to register with an already registered email	{"name": "name", "email": "email", "password": "password"}	{"success": False, "message": "User already exists."}	Matches Expected
Add Food - Successful	Admin adds a food item with valid details	{"name": name, "description": desc, "price": price, "category": cat, "image": image_path}	{"success": True, "message": "Food Added"}	Matches Expected
List Food - Successful	Retrieve the list of all food items	{ }	{"success": True, "data": [food_items]}	Matches Expected
Remove Food - Successful	Admin removes a food item	{"itemId": itemid}	{"success": True, "message": "Food Removed"}	Matches Expected
Add to Cart - Successful	Add a valid item to the user's cart	{header{token: token}, "itemId": itemId}	{"success": True, "message": "Added to cart"}	Matches Expected
Add to Cart - Item Already Exists	Add an already existing item to increase quantity	{header{token: token}, "itemId": itemId}	{"success": True, "message": "Added to cart"}	Matches Expected
Add to Cart - Invalid Token	Attempt to add to cart with an invalid token	{header{token: wrong_token}, "itemId": itemId}	{"success": False, "message": "Unauthorized"}	Matches Expected
Remove from Cart - Successful	Successfully remove an item from the cart	{header{token: token}, "itemId": itemId}	{"success": True, "message": "Removed from cart"}	Matches Expected

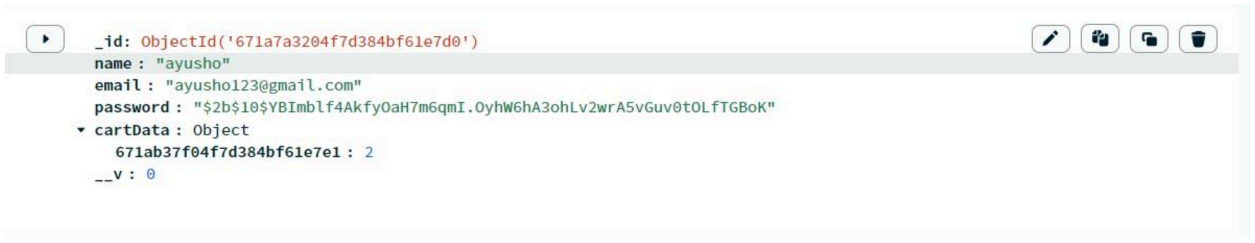


POST http://localhost:4000/api/cart/add

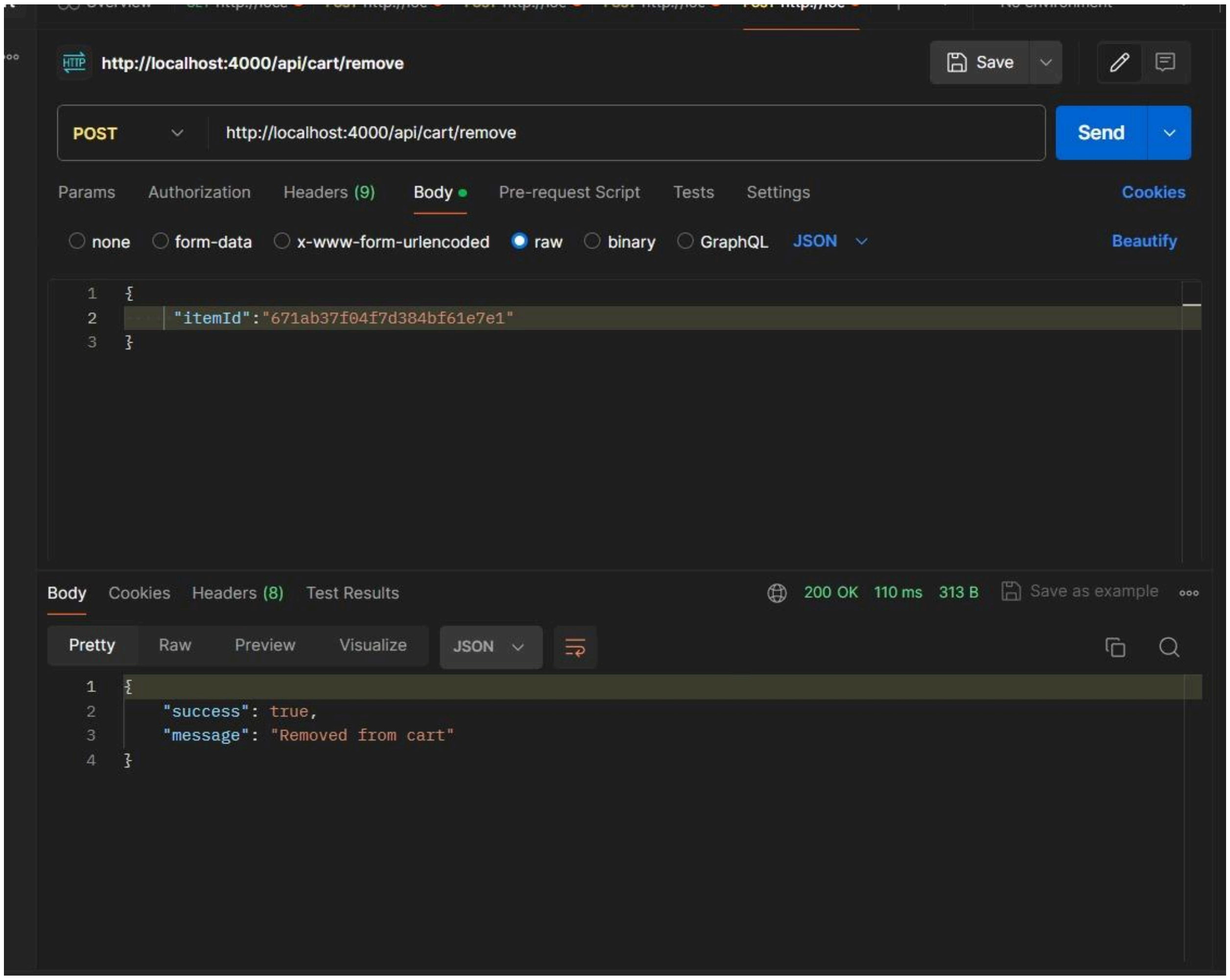
```
{
    "itemId":"671ab37f04f7d384bf61e7e1"
}
```

Body — 200 OK 117 ms 309 B

```
{
    "success": true,
    "message": "Added to cart"
}
```



```
_id: ObjectId('671a7a3204f7d384bf61e7d0')
name : "ayusho"
email : "ayusho123@gmail.com"
password : "$2b$10$YBImblf4AkfyOaH7m6qmI.OyhW6hA3ohLv2wrA5vGuv0tOLfTGBoK"
▾ cartData : Object
    671ab37f04f7d384bf61e7e1 : 2
__v : 0
```



POST http://localhost:4000/api/cart/remove

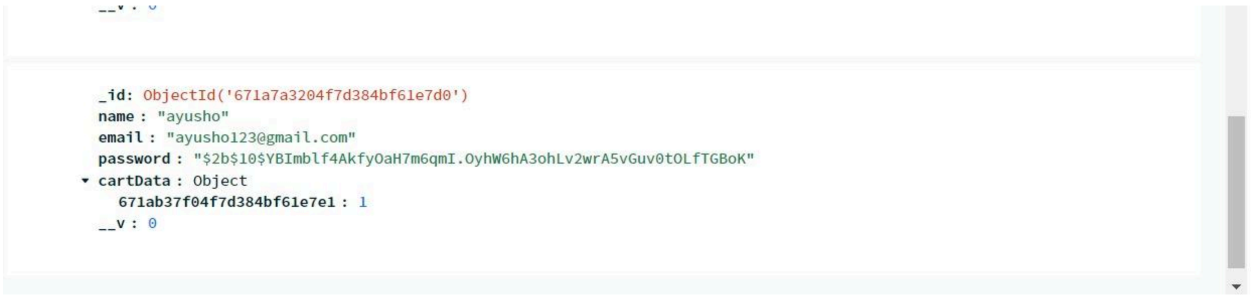
raw JSON

```
{
    "itemId":"671ab37f04f7d384bf61e7e1"
}
```

200 OK 110 ms 313 B

JSON

```
{
    "success": true,
    "message": "Removed from cart"
}
```



```
_id: ObjectId('671a7a3204f7d384bf61e7d0')
name : "ayusho"
email : "ayusho123@gmail.com"
password : "$2b$10$YBImblf4AkfyOaH7m6qmI.OyhW6hA3ohLv2wrA5vGuv0tOLfTGBoK"
▾ cartData : Object
    671ab37f04f7d384bf61e7e1 : 1
__v : 0
```

Deployment

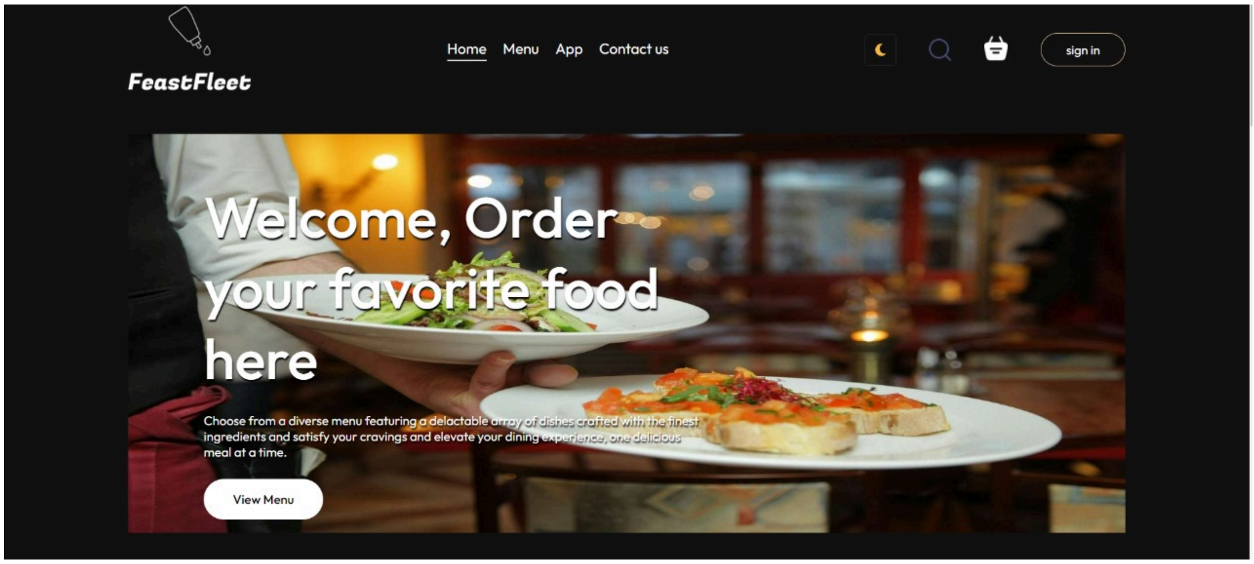
Environment

- Development: Local environment with hot-reloading.
- Production: Deployed on a cloud platform (Vercel).

Deployment Steps

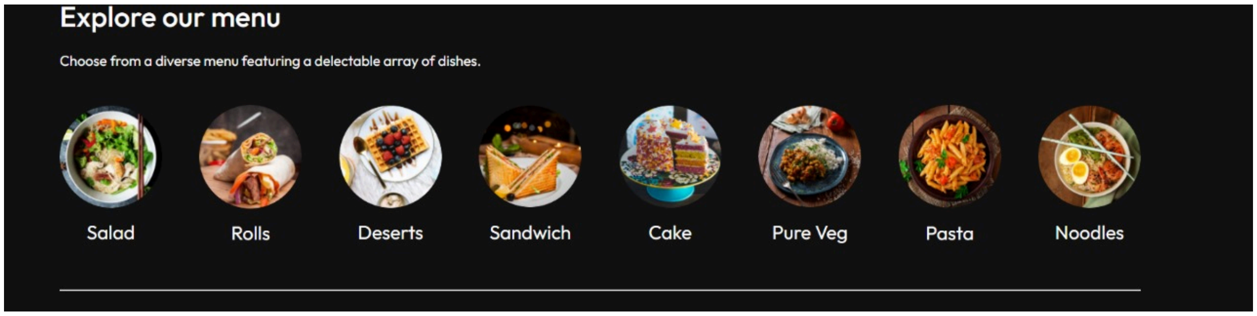
1. Frontend: Build the React application and deploy to a static hosting service (Netlify).
2. Backend: Deploy the Node.js application to a cloud platform.
3. Database: Set up MongoDB on a cloud database service (e.g., MongoDB Atlas).

Overview



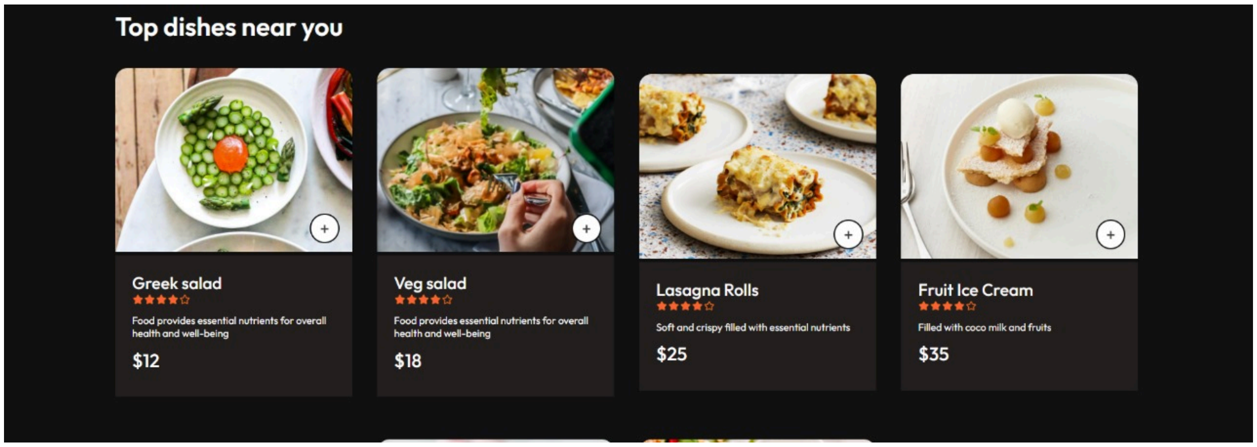
Welcome, Order your favorite food here

Choose from a diverse menu featuring a delactable array of dishes crafted with the finest ingredients and satisfy your cravings and elevate your dining experience, one delicious meal at a time.



Explore our menu

Choose from a diverse menu featuring a delectable array of dishes.


Top dishes near you
Greek salad
Food provides essential nutrients for overall health and well-being
$12
Veg salad
Food provides essential nutrients for overall health and well-being
$18
Lasagna Rolls
Soft and crispy filled with essential nutrients
$25
Fruit Ice Cream
Filled with coco milk and fruits
$35

Future Scope

1. Sophisticated payment gateway using Stripe.
2. Proper automated delivery tracking system.
3. Taking feedback from our customers to improve our service.