

# ORDER ON THE GO

**Project Title:** ORDER ON THE GO

- **Team Members:**

1. M. V S G VARA LAKSHMI DEVI (**SUPPORTER**)
2. G. SIVA GANGA (**BACKENDER**)
3. A. SAI CHANDESH (**FRONTENDER**)
4. D. GANESH (**FRONTENDER**)
5. G. DURGA SRIRAM PRASAD (**BACKENDER**)
6. B. JAYA RAM TEJA (**SUPPORTER**)

## DESCRIPTION

**Convenience:** Allowing users to place orders quickly and efficiently, often through mobile apps or online platforms.

**Efficiency:** Reducing wait times and improving order accuracy by automating the process.

**User Experience:** Providing a seamless and user-friendly interface for customers.

**Business Growth:** Helping businesses attract more customers and manage orders effectively.

## FEATURES

**Self-Service Kiosks:** Customers can browse menus, customize orders, and make payments without staff assistance.

**Mobile Integration:** A user-friendly app or mobile platform for placing orders on the go.

**Point-of-Sale (POS) Integration:** Seamless connection with POS systems for efficient order processing.

**Loyalty Programs:** Features to reward repeat customers and encourage brand loyalty.

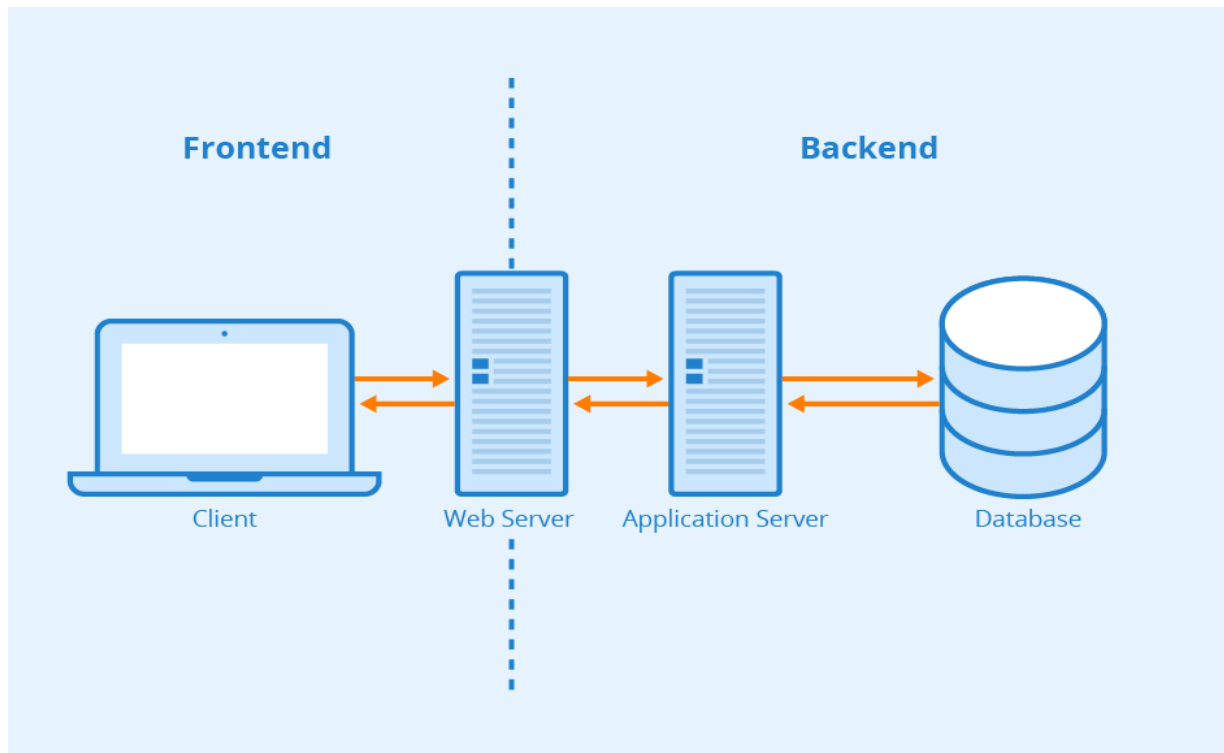
**Upselling and Promotions:** Automated suggestions for add-ons or special offers to increase sales.

**Order Tracking:** Real-time updates on order status for customers.

**Offline Access:** Ability to function even without an internet connection.

**Data Synchronization:** Ensures smooth operations across devices and platforms

## ARCHITECTURE



## FRONTEND

### React:

- **Component-Based Architecture:** React lets you build reusable UI components for better scalability and maintainability.
- **Dynamic Rendering:** React uses a virtual DOM to update and render only the components that change, ensuring efficient performance.
- **State Management:** React's state and props features help manage application data effectively.
- **Flexibility:** Integrates easily with other libraries or frameworks like Axios and Bootstrap.

### Axios:

- **API Integration:** Axios simplifies HTTP requests, making it easier to fetch or send data to a backend server.
- **Promise-Based:** Supports promise chaining for handling API responses and errors cleanly.
- **Cross-Browser Support:** Works reliably across different browsers.
- **Interceptors:** Helps in adding headers like authentication tokens and handling errors globally.

## Bootstrap:

- **Responsive Design:** Bootstrap's grid system ensures your application looks great on all devices.
- **Pre-Built Components:** Provides buttons, modals, navbars, and more, speeding up development.
- **Customizable:** You can override default Bootstrap styles with your own CSS for a unique look.
- **Ease of Use:** Simplifies styling with classes, saving time compared to writing extensive custom CSS.

## TECHNOLOGIES

**React:** Build the layout and components (e.g., Header, Footer, ProductList, OrderForm).

**Axios:** Fetch data (e.g., available items or user orders) from an API and display them dynamically in the React components.

**Bootstrap:** Style your components, ensuring a clean and consistent design across the application.

## BACKEND

**Node.js:**

**Event-Driven and Non-Blocking I/O:** Node.js processes multiple requests asynchronously, making it highly efficient for handling concurrent operations.

**JavaScript Everywhere:** Use JavaScript on both the frontend and backend, enabling seamless development.

**Scalability:** Suitable for real-time applications and microservices architecture.

**Rich Ecosystem:** With npm (Node Package Manager), you have access to thousands of libraries and tools.

## Express:

**Lightweight and Flexible:** A minimal framework for building robust APIs and web applications.

**Middleware Support:** Use middleware for handling requests, logging, parsing, authentication, and more.

**Routing:** Define custom routes to handle specific HTTP methods and endpoints.

**Template Engines:** Easily integrate template engines like Pug or EJS for dynamic content rendering.

A **Node.js + Express backend** is ideal for projects like "OrderOn the Go," features like **RESTful APIs** for:

**Order Management:** Create, read, update, or delete orders.

**Authentication:** Secure login and token-based user authentication.

**Real-Time Data:** Use WebSockets for live order tracking.

## DATA BASE

### MongoDB:

MongoDB is a NoSQL database that stores data in a JSON-like format (BSON). It is highly flexible, scalable, and is well-suited for modern applications requiring real-time data or large-scale distributed systems.

### Connecting Node.js to MongoDB:

The connection enables Node.js to interact with the MongoDB database to perform CRUD (Create, Read, Update, Delete) operations. To achieve this:

1. **Driver or ORM:** Use a driver or Object Relational Mapping (ORM) library like:
  - **Native MongoDB Driver:** Directly interact with the database.
  - **Mongoose:** An abstraction over MongoDB that simplifies interactions with schemas and models.
2. **Establish the Connection:** The `connect()` method from the library creates a connection between Node.js and MongoDB.
3. **Schemas and Models:** With tools like Mongoose, developers define schemas for structuring data and create models for database operations.
4. **Execute Queries:** After connecting, you can execute queries to add, retrieve, update, or delete data in MongoDB collections.

## SETUP INSTRUCTIONS

### Backend Setup (Node.js, Express, MongoDB)

1. **Initialize the Project:** Begin by setting up a Node.js project and configuring it with essential dependencies like Express for building the backend, Mongoose for connecting to MongoDB, and middleware tools such as CORS and body parsers.
2. **Environment Configuration:** Create an environment file to store sensitive information like the MongoDB connection URI and other configuration variables securely.

3. **Database Connection:** Set up a connection between the backend and a MongoDB database using Mongoose, ensuring the database is ready to store application data.
4. **API Routes:** Define RESTful endpoints in the backend to handle tasks such as retrieving, creating, updating, or deleting data.
5. **Start Server:** Run the backend server on a specified port, ensuring it is ready to handle requests.

## Frontend Setup (React, Axios, Bootstrap)

1. **Create the React Application:** Initialize a new React project to act as the frontend for the application.
2. **Install Required Libraries:** Add packages like Axios for making HTTP requests to the backend and Bootstrap for styling the user interface.
3. **API Integration:** Configure Axios in the frontend to make requests to the backend endpoints, allowing communication between the two layers.
4. **Component Setup:** Build React components to display the data fetched from the backend and enable user interactions like form submissions or updates.
5. **Styling:** Use Bootstrap to style the components and make the frontend responsive and visually appealing.
6. **Start Development Server:** Run the frontend on a local development server to interact with the backend.

## Integration Between Frontend and Backend

1. **Cross-Origin Resource Sharing (CORS):** Ensure that CORS is enabled in the backend so the frontend can make requests without restrictions.
2. **Test Communication:** Verify that the frontend can successfully send requests to the backend and receive appropriate responses.
3. **Handle Data Flow:** Ensure data flows smoothly between the backend, database, and frontend, enabling real-time updates or interactions as needed.

# INSTALLATION

## Step-by-Step Installations for Frontend, Backend, and Database Connections

### Frontend Setup:

1. **Initialize Frontend Project:** Begin by setting up the structure of the frontend application using a framework like React. This creates the base project where all frontend features will be developed.
2. **Install Dependencies:** Install the necessary libraries and packages required for building the user interface, handling HTTP requests, and styling the application (e.g., using Axios and Bootstrap).
3. **Configure API Integration:** Establish a connection between the frontend and backend by setting up a way to make HTTP requests (e.g., defining an API utility for managing backend communication).
4. **Build UI Components:** Develop components that handle user interactions, display data, and provide a seamless user experience.

### Backend Setup:

1. **Initialize Backend Project:** Create the backend framework using tools like Node.js and Express to handle server-side logic and API creation.
2. **Install Required Modules:** Add packages for routing, database integration, request parsing, cross-origin resource sharing (CORS), and environment management.
3. **Setup Middleware:** Enable middleware to handle requests, responses, and ensure secure communication with the frontend.
4. **Define API Routes:** Create endpoints for handling various tasks like data retrieval, updates, and deletion.

### Database Setup and Integration:

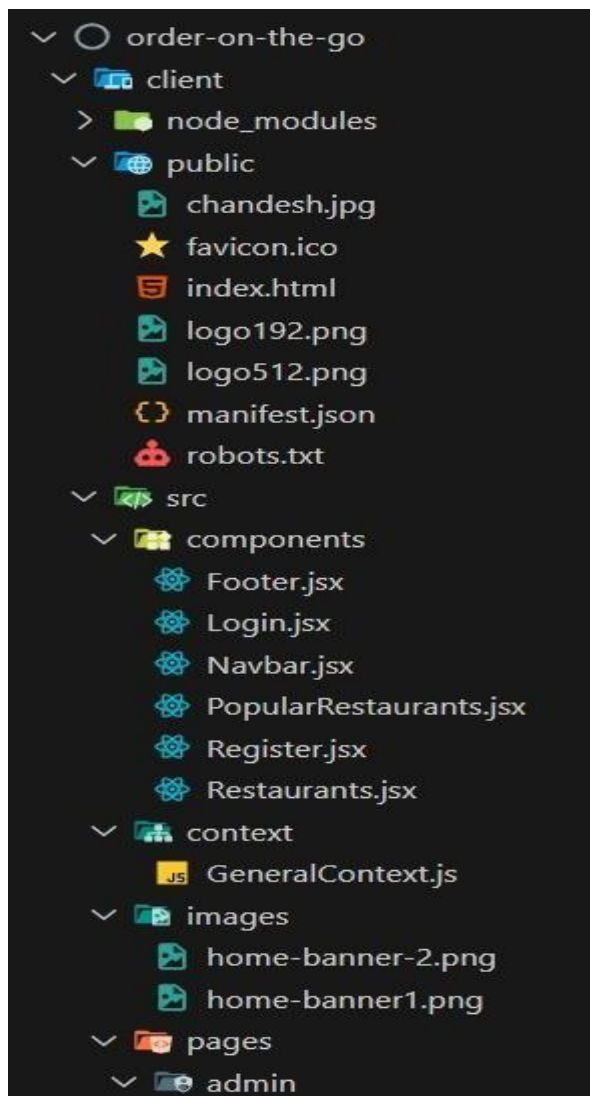
1. **Database Creation:** Set up a MongoDB database to store application data securely, either locally or using a cloud service like MongoDB Atlas.
2. **Install Database Drivers/ORM:** Use tools like Mongoose (or the native MongoDB driver) to connect the backend to the database.
3. **Define Schemas and Models:** Create structured schemas for storing and retrieving specific types of data consistently.
4. **Test Connectivity:** Ensure that the database and backend can communicate effectively to perform CRUD (Create, Read, Update, Delete) operations.

## Integration of Frontend, Backend, and Database:

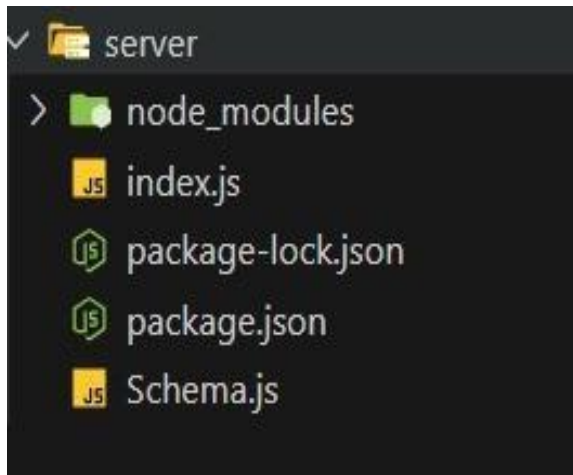
1. **Link Frontend to Backend:** Configure the frontend to interact with backend API endpoints for tasks such as fetching or submitting data.
2. **Enable Cross-Origin Communication:** Use middleware like CORS to ensure requests from the frontend (hosted on one port) can access backend APIs (hosted on another port).
3. **Verify Data Flow:** Test the full workflow to ensure smooth communication between the frontend, backend, and database.

## FOLDER STRUCTURE

### Client:



## Server:



## RUNNING THE APPLICATIONS:

### Frontend Server

To start the React-based frontend server:

- Navigate to the frontend directory in your terminal.
- Use the command to run the development server, which will typically launch on port 3000.

### Backend Server

To start the Node.js backend server:

- Navigate to the backend directory in your terminal.
- Use the command to run the backend server, which will usually run on a specified port (e.g., 5000).

## API DOCUMENTATION

### 1. Fetch All Items

- **Endpoint:** Retrieves all items available in the database, typically used to display a list of products or data.
- **Method:** A request type designed to retrieve data from the server.
- **Parameters:** None required, making it a simple endpoint for fetching all records.



- **Response:** A structured list containing details of all items, such as name, price, or identifiers.

## 2. Fetch a Specific Item

- **Endpoint:** Provides detailed information about a single item by its unique identifier.
- **Method:** A type of request that fetches data based on a specified ID in the URL.
- **Parameters:** Requires a unique identifier as part of the URL path.
- **Response:** Detailed data for the requested item, like name, description, price, or related attributes.

## 3. Add a New Item

- **Endpoint:** Used to create a new entry in the database, adding a new product, record, or entity.
- **Method:** A request type for submitting new data to the server.
- **Parameters:** Requires specific fields in the body, such as name, price, or other details.
- **Response:** A confirmation that the new item has been successfully created, along with its assigned identifier.

## 4. Update an Existing Item

- **Endpoint:** Allows modification of an existing item in the database using its unique identifier.
- **Method:** A request type that updates specified fields of an item.
- **Parameters:** Requires a unique identifier in the URL and fields to be updated in the body.
- **Response:** A confirmation that the item has been updated, with the new details of the modified record.

## 5. Delete an Item

- **Endpoint:** Removes an item from the database by its unique identifier.
- **Method:** A request type for deleting a resource from the server.
- **Parameters:** Requires a unique identifier in the URL path to specify which item to delete.
- **Response:** A confirmation that the item has been successfully deleted.

## 6. Place an Order

- **Endpoint:** Creates a new order in the system, linking one or more items with a customer or session.
- **Method:** A request type for submitting new data related to an order.
- **Parameters:** Requires fields like item details and the total amount in the request body.
- **Response:** A confirmation of the order being placed, with details such as order ID and total price.

## 7. Fetch All Orders

- **Endpoint:** Retrieves all placed orders, typically for administrative purposes or user order history.
- **Method:** A request type for retrieving multiple records.
- **Parameters:** None required, providing access to all orders in the database.
- **Response:** A list of all orders with details like items, quantities, total prices, and identifiers.

## 8. Delete an Order

- **Endpoint:** Removes an order from the system using its unique identifier.
- **Method:** A request type designed to delete a specified resource.
- **Parameters:** Requires a unique order identifier in the URL path.
- **Response:** A confirmation that the order has been successfully removed.

## AUTHENTICATION

### Authentication:

- **Token-Based Authentication:** Users log in with their credentials, and the server generates a token (e.g., JSON Web Token or JWT). This token is sent with each request to verify the user's identity.
- **Session-Based Authentication:** The server creates a session for the user and stores session data on the server. A session ID is sent to the user's browser as a cookie, which is used to authenticate subsequent requests.

## **Authorization:**

- After authentication, authorization determines what resources or actions the user is allowed to access. This is often role-based (e.g., admin, user, guest) or permission-based.

## **Security Enhancements:**

- **Encryption:** Tokens or session data are encrypted to prevent unauthorized access.
- **Expiration:** Tokens or sessions have expiration times to reduce the risk of misuse.
- **Refresh Tokens:** Used to obtain new tokens without requiring the user to log in again.

## **USER INTERFACE**

**Web Interface:** A responsive web application is often created using frameworks like React, Angular, or Vue.js. These interfaces are designed to work seamlessly across devices, including desktops, tablets, and smartphones.

**Mobile Interface:** Dedicated mobile apps for Android and iOS are developed using native technologies like Kotlin/Swift or cross-platform frameworks like Flutter or React Native. These apps focus on providing a smooth and intuitive user experience.

**Admin Dashboard:** A separate interface for administrators to manage orders, users, and other backend operations. This is usually web-based and includes features like data visualization, analytics, and role-based access.

**Customer-Facing Features:** Interfaces for customers typically include functionalities like browsing menus, placing orders, tracking deliveries, and making payments. These are designed to be user-friendly and visually appealing.

**Accessibility:** Ensuring the interface is accessible to users with disabilities by adhering to standards like WCAG (Web Content Accessibility Guidelines).

# TESTING

## Unit Testing:

- Focuses on testing individual components or modules of the application to ensure they function as expected.
- Often automated and performed using frameworks like JUnit, NUnit, or Jest.

## Integration Testing:

- Verifies that different modules or services within the application work together correctly.
- Ensures seamless communication between APIs, databases, and other components.

## System Testing:

- Tests the entire application as a whole to ensure it meets the specified requirements.
- Includes end-to-end testing of workflows, such as placing an order, processing payments, and tracking deliveries.

## Performance Testing:

- Evaluates the application's responsiveness, stability, and scalability under various conditions.
- Tools like JMeter or LoadRunner are commonly used.

## User Acceptance Testing (UAT):

- Conducted by end-users to ensure the application meets their needs and expectations.
- Focuses on usability, functionality, and overall user experience.

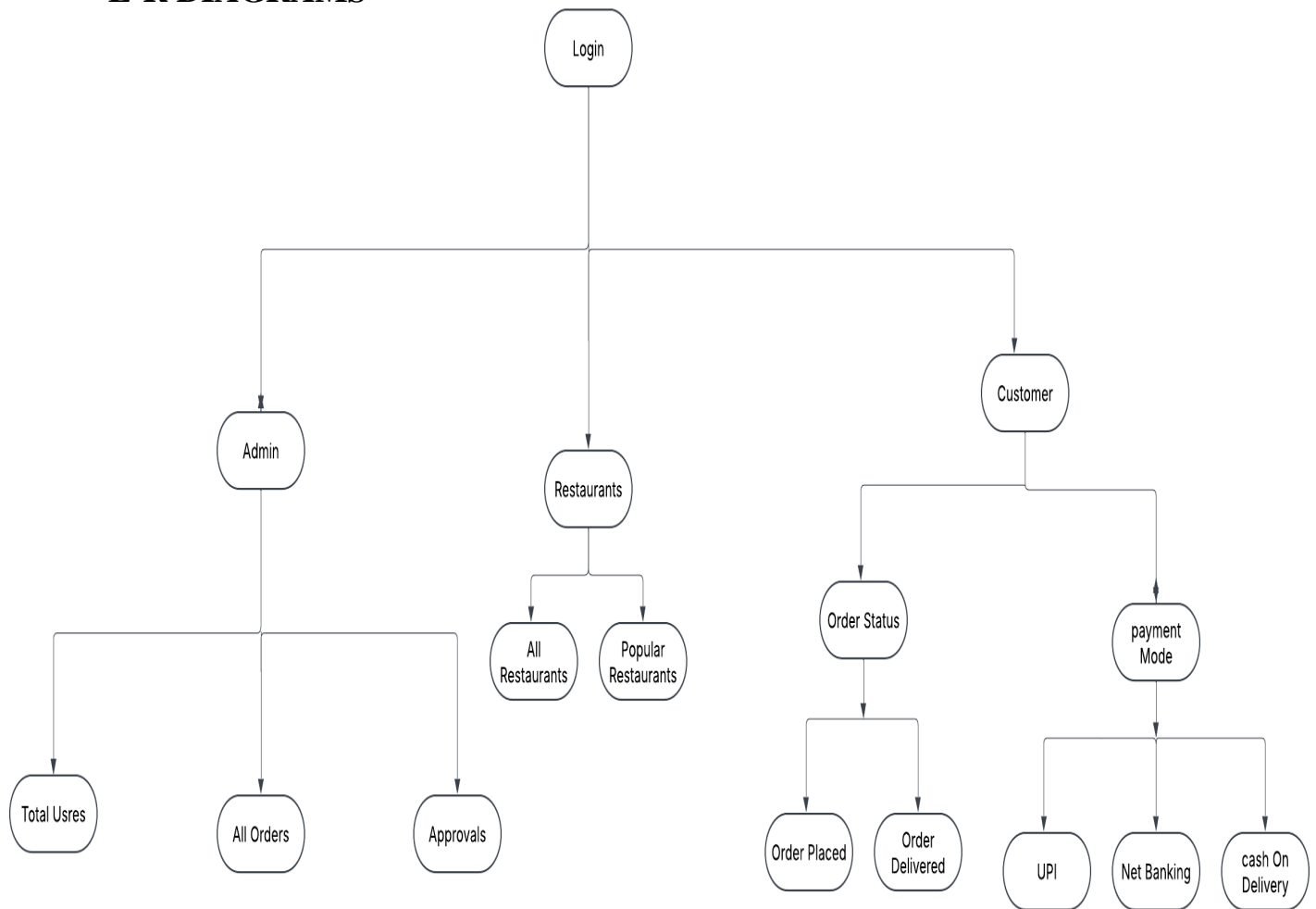
## Regression Testing:

- Ensures that new changes or updates to the application do not negatively impact existing functionality.

## Security Testing:

- Identifies vulnerabilities and ensures the application is secure against potential threats.
- Includes penetration testing and vulnerability assessments.

## E-R DIAGRAMS



A customer care registry ER-diagram represents the entities and relationships involved in a customer care system. It helps visualize how customers, inquiries, complaints, feedback, requests, agents, and administrators interact within the system. Here's a breakdown of the entities and their relationships:

**CUSTOMER:** Represents the individuals or entities that interact with the customer care system. They can submit inquiries, complaints, feedback, and requests.

**INQUIRY, COMPLAINT, FEEDBACK, REQUEST:** These entities represent different types of interactions or requests made by the customers. Each of these entities is associated with the CUSTOMER entity, indicating that a customer can submit multiple inquiries, complaints, feedback, or requests.

**AGENT:** Represents the agents or customer service representatives who handle the inquiries, complaints, feedback, and requests. Each of these entities is associated with the INQUIRY, COMPLAINT, FEEDBACK, or REQUEST entity, indicating that an agent can handle multiple customer interactions.

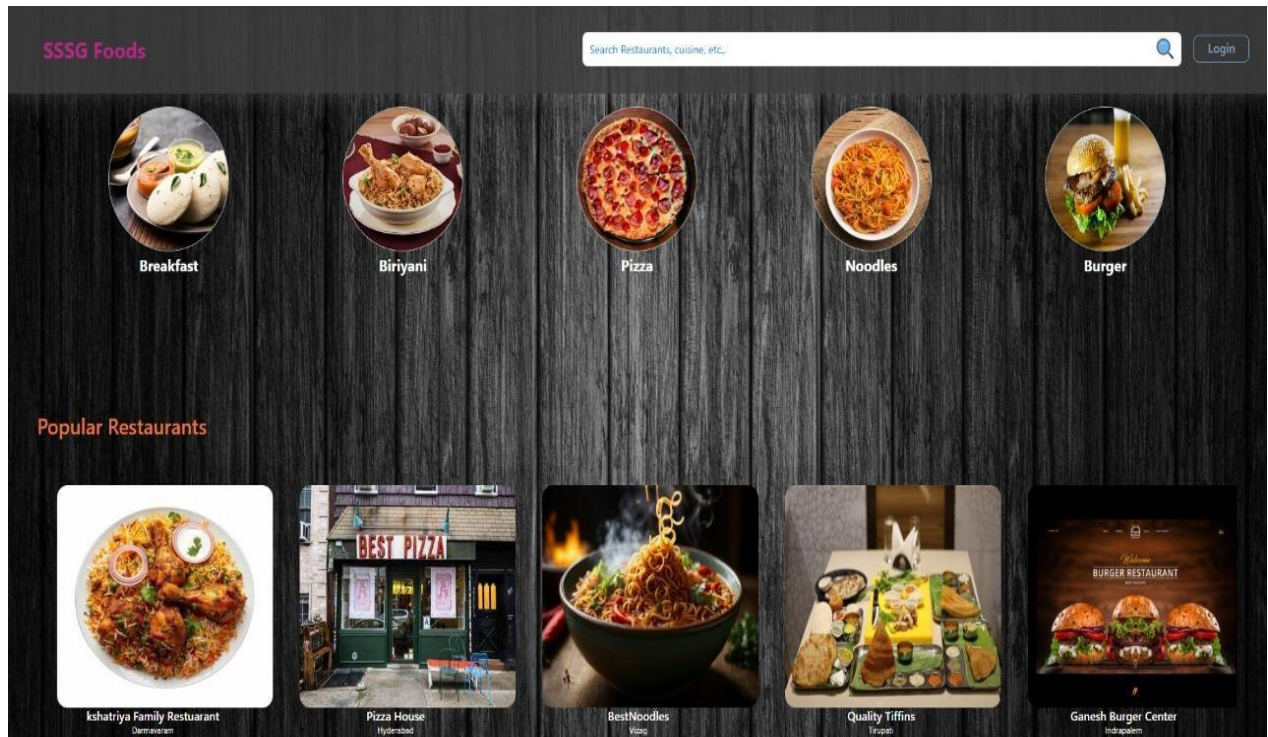
**ADMIN:** Represents the administrators or supervisors who manage the customer care system. The ADMIN entity is associated with the AGENT entity, indicating that an admin can manage multiple agents.

## Relationships

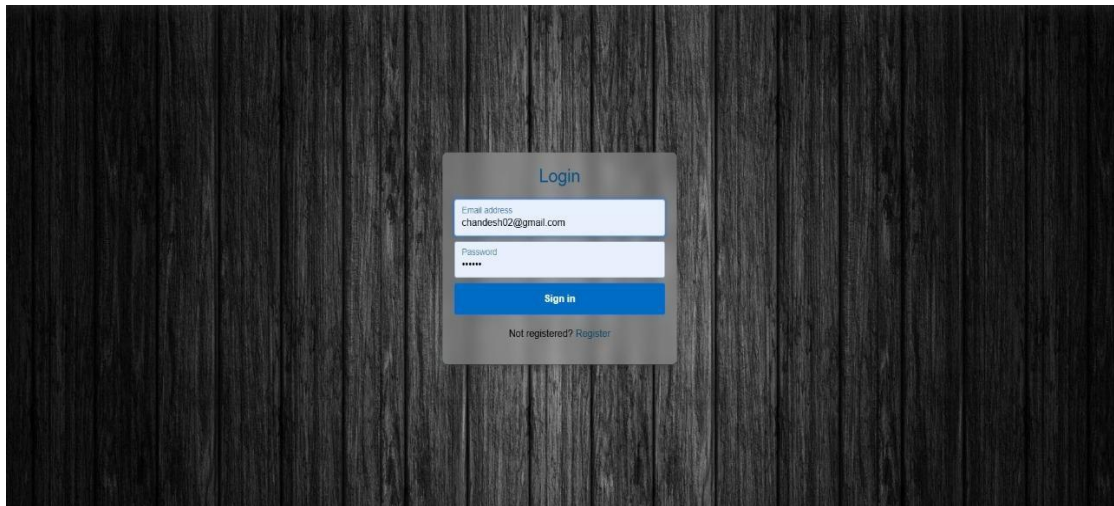
The relationships between the entities are represented using different notations. The "`||--o{`" notation indicates a one-to-many relationship, where a customer can submit multiple inquiries, complaints, feedback, or requests. The "`}|--|{`" notation represents a many-to-many relationship, indicating that an agent can handle multiple inquiries, complaints, feedback, or requests, and each of these interactions can be handled by multiple agents. The "`--o|`" notation shows a one-to-one relationship, where an agent assists a single customer. The "`}|--|{`" notation also represents a many-to-many relationship between the ADMIN and AGENT entities, indicating that an admin can manage multiple agents, and each agent can be managed by multiple admins.

## DEMO SCREEN SHOTS

### FRONTEND:



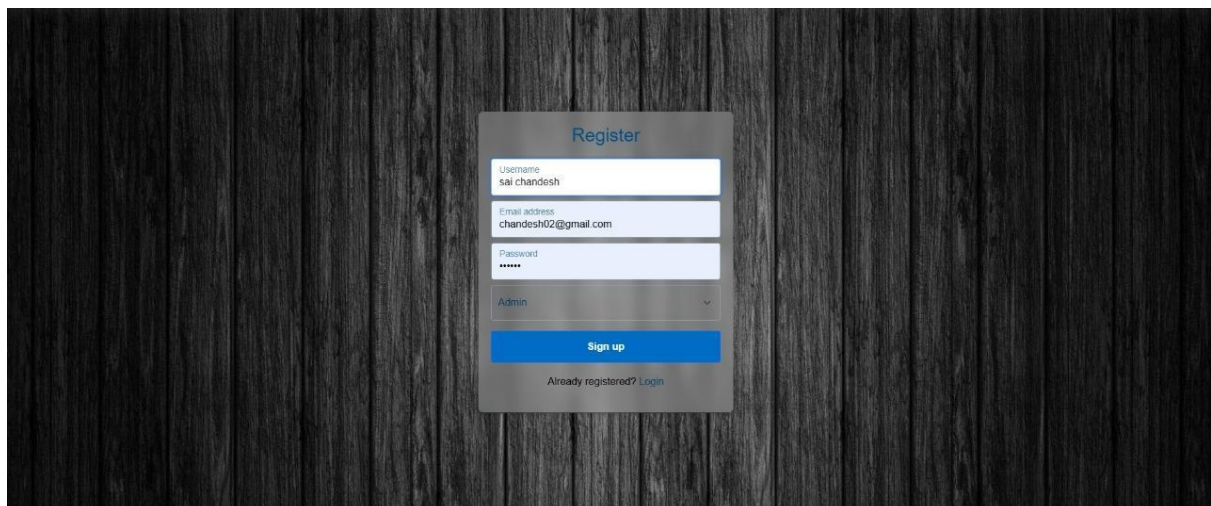
## LOGIN PAGE:



A login form titled "Login" is centered on a dark wood-textured background. The form has a light gray border and contains the following elements:

- Email address:** A text input field containing "chandesh02@gmail.com".
- Password:** A text input field with masked characters "\*\*\*\*\*".
- Sign in:** A blue button with white text.
- Not registered? Register:** A link at the bottom of the form.

## REGISTER PAGE:



A register form titled "Register" is centered on a dark wood-textured background. The form has a light gray border and contains the following elements:



- Username:** A text input field containing "sai chandesh".
- Email address:** A text input field containing "chandesh02@gmail.com".
- Password:** A text input field with masked characters "\*\*\*\*\*".
- Admin:** A dropdown menu with "Admin" selected and a downward arrow.
- Sign up:** A blue button with white text.
- Already registered? Login:** A link at the bottom of the form.




ALL USERS:

All Users			
User Id	User Name	Email Address	User Type
67d162fd36b56e6e83ab636f	kshatriya Family Restuarant	kshatriya@143.com	restaurant
User Id	User Name	Email Address	User Type
67d1954aee994f1997338593	Ganesh	ganesh14@gmail.com	restaurant
User Id	User Name	Email Address	User Type
67d19f0eee994f1997338820	sriram Tiffin And Restaurant	sriram21@.com	restaurant
User Id	User Name	Email Address	User Type
67d1a2aee994f199733890b	Abhiram	abhiram02@sb.com	restaurant
User Id	User Name	Email Address	User Type
67d1a33fee994f1997338925	Teja	teja02@sb.com	restaurant


All Orders

Orders	
	<div><div>Hawaiian</div><div>Pizza House</div><div>Userid: 67d287d7e729045a1224f290    Name: Akash    Mobile: 9776557787    Email: akash@gmail.com</div><div>Quantity: 3    Total Price: ₹ 432 ₹ 540    Payment mode: UPI</div><div>Address: Goa    Pincode: 10111    Ordered on: 2025-03-13 Time: 07:26</div><div>status: Order Placed</div></div>
	<div><div>Fry Piece Biryani</div><div>kshatriya Family Restaurant</div><div>Userid: 67d287d7e729045a1224f290    Name: Akash    Mobile: 9776557787    Email: akash@gmail.com</div><div>Quantity: 1    Total Price: ₹ 127 ₹ 150    Payment mode: UPI</div><div>Address: Goa    Pincode: 10111    Ordered on: 2025-03-13 Time: 07:26</div><div>status: Order Placed</div></div>

## CART:



**Falafel Burger**  
Ganesh Burger Center  
Quantity:  Price: ₹ 80 ₹65  
[Remove](#)



**PEPPER BARBECUE & ONION**  
Pizza House  
Quantity:  Price: ₹ 375 ₹590  
[Remove](#)

**Price Details**

Total MRP:	₹ 585
Discount on MRP:	- ₹ 125
Delivery Charges:	+ ₹ 50
<b>Final Price: ₹ 506</b>	

[Place order](#)

Categories

- ☐ Lunch
- ☐ Breakfast
- ☐ Biryani
- ☐ Biryani
- ☐ Pizza
- ☐ Noodles
- ☐ Breakfast
- ☐ Burger



**Egg Chow Mein**  
A popular street food wit...  
₹ 104 ₹110  
[Add item](#)



**Prawn Noodles**  
Juicy prawns tossed with ...  
₹ 153 ₹180  
[Add item](#)



**Veg Noodles**  
This dish is highly versa...  
₹ 150 ₹200  
[Add item](#)



**Black Bean And Chicken Noodles**  
The dish has a deep umami...  
₹ 210 ₹300  
[Add item](#)





Categories

- ☐ Lunch
- ☐ Breakfast
- ☐ Biryani
- ☐ Biryani
- ☐ Pizza
- ☐ Noodles
- ☐ Breakfast
- ☐ Burger



**Chapati**  
Tiffin chapatis are soft...

₹ 39 46

[Add Item](#)



**Vada**  
Tiffin vada is a crispy...

₹ 30 46

[Add Item](#)



**Paratha**  
It's a popular choice for...

₹ 49 66

[Add Item](#)




**Mysore Bonda**  
Mysore Bonda, also known...

₹ 25 50

[Add Item](#)

Categories


- ☐ Lunch
- ☐ Breakfast
- ☐ Biryani
- ☐ Biryani
- ☐ Pizza
- ☐ Noodles
- ☐ Breakfast
- ☐ Burger



**Pot Biryani**  
The flavorful, aromatic d...

₹ 562 750


[Add Item](#)



**Prawn Biryani**  
Prawn biryani is a treat...

₹ 450 500


[Add Item](#)



**Zafrani Biryani**  
Ah, Zafrani Biryani, a tr...

₹ 360 600

[Add Item](#)



**Veg Biryani**  
This dish features layers...

₹ 250 500

[Add Item](#)

## Ganesh Burger Center

Indrapalem

Filters

Sort By

- ☐ Popularity
- ☐ low-price
- ☐ high-price
- ☐ Discount
- ☐ Rating


Food Type

- ☐ Veg
- ☐ Non Veg
- ☐ Beverages

Categories

- ☐ Lunch
- ☐ Breakfast
- ☐ Biryani


All Items



**Veggie Patty Burger**  
Made with a mix of lentil...

₹ 76 69


[Add Item](#)



**Paneer Tikka Burger**  
Grilled paneer cubes with...

₹ 72 69


[Add Item](#)



**Falafel Burger**  
Crispy falafel patty with...

₹ 80 85

[Add Item](#)



**Aloo Tikki Burger**  
Spiced potato patty with...

₹ 85 100

[Add Item](#)



## KNOW ISSUES

### Authentication Problems:

- Token expiration or invalidation issues.
- Session timeouts not handled gracefully.

### Performance Bottlenecks:

- Slow response times during peak usage.
- Inefficient database queries causing delays.

### UI/UX Challenges:

- Inconsistent design across platforms.
- Accessibility issues for users with disabilities.

**Payment Gateway Errors:**

- Failed transactions due to integration issues.
- Lack of proper error messages for payment failures.

**Order Management Glitches:**

- Incorrect order status updates.
- Delays in syncing order data between systems.

**FUTURE ENHANCEMENTS****Personalized Recommendations:**

- Use AI and machine learning to suggest items based on user preferences and past orders.

**Loyalty Programs:**

- Introduce reward points, discounts, or special offers for frequent customers.

**Multi-Language Support:**

- Add support for multiple languages to cater to a more diverse user base.

**Real-Time Order Tracking:**

- Implement live updates on order preparation, dispatch, and delivery.

**Voice Command Integration:**

- Allow users to place orders using voice commands with smart assistants like Alexa, Google Assistant, or Siri.

**Offline Functionality:**

- Enable users to browse menus and prepare orders even without an internet connection, syncing once online.

**Advanced Analytics for Admins:**

- Offer detailed reports and insights on sales trends, customer behavior, and operational performance.

**Feedback System:**

- Create an intuitive feedback loop where customers can rate and review orders or services.

**Augmented Reality (AR) Features:**

- Let users visualize food items or menu setups through AR before placing an order.

**Sustainability Features:**

- Highlight eco-friendly packaging or allow users to opt-out of unnecessary utensils for delivery.

**Multi-Cart Feature:**

- Support ordering from multiple vendors in a single transaction.

**Enhanced Payment Options:**

- Integrate multiple payment gateways and support modern methods like cryptocurrency or buy-now-pay-later (BNPL) systems.

**Emergency Order Assistance:**

- Provide an option to quickly modify or cancel orders in case of errors or urgent situations.

**Community Feature:**

- Introduce social or community aspects, such as sharing favorite dishes or creating group orders with friends.

**Customizable Interface:**

- Allow users to personalize their app interface for a more tailored experience.