

# BATCH-ID

## LTVIP2025TMID59612

### 1.Introduction

- **Project Title:**

**OrderOnTheGo**: Your On-Demand Food  
Ordering Solution

- **Team Members:**

- ❖ Shaik Ziaur Rahaman – Team Leader,
- ❖ Rebba Gopi– Team Member,
- ❖ Porumalla Sai Muneesh– Team Member,
- ❖ Puli Abhishek– Team Member.

## 2. Application Flow

### • Purpose:

SB Foods is an innovative digital food ordering platform designed to transform the online dining experience. Built for maximum \*convenience and efficiency\*, SB Foods caters to both casual diners and passionate food enthusiasts by offering a seamless and user-friendly web application.

With SB Foods, users can easily \*explore a wide variety of dishes\*, view detailed descriptions, customer reviews, prices, and promotions — all in one place. This transparency empowers users to make confident and well-informed food choices without any confusion or guesswork.

The platform's \*streamlined ordering process\* allows customers to quickly place orders by simply entering their name, delivery address, payment method, and chosen dishes. Once the order is submitted, an instant confirmation ensures peace of mind and eliminates the need to wait in lines or navigate complex systems. SB Foods stands out by combining \*simplicity, clarity, and speed\*, redefining how users browse and order food online.

### • Features:

- **User Registration & Login:** Users can sign up or log in securely using email and password. Authentication ensures personalized experiences and secure access.
- **Browser Menu & Dishes:** Explore a wide range of dishes categorized by cuisine, type, and popularity. Each dish includes photos, ingredients, and price.
- **Smart Search & Filters:** Quickly find meals using keyword search or filters like cuisine, price, or ratings. Enhances the user's ability to find preferred dishes.

- **Quick Checkout Process:** Users can enter their name, delivery address, and payment info in a few steps. The simplified form minimizes ordering time.
- **Payment Integration:** Supports multiple payment methods like cards, wallets, or UPI. Transactions are processed securely and instantly.
- **Order Confirmation:** Once an order is placed, users receive an instant confirmation message. It ensures peace of mind and verifies successful placement.
- **Order tracking:** Users will be able to track their order status in real-time. This adds transparency and reduces uncertainty.
- **Secure Authentication :** User data is protected with secure login and session handling. Maintains platform integrity and user trust.

### 3. Architecture

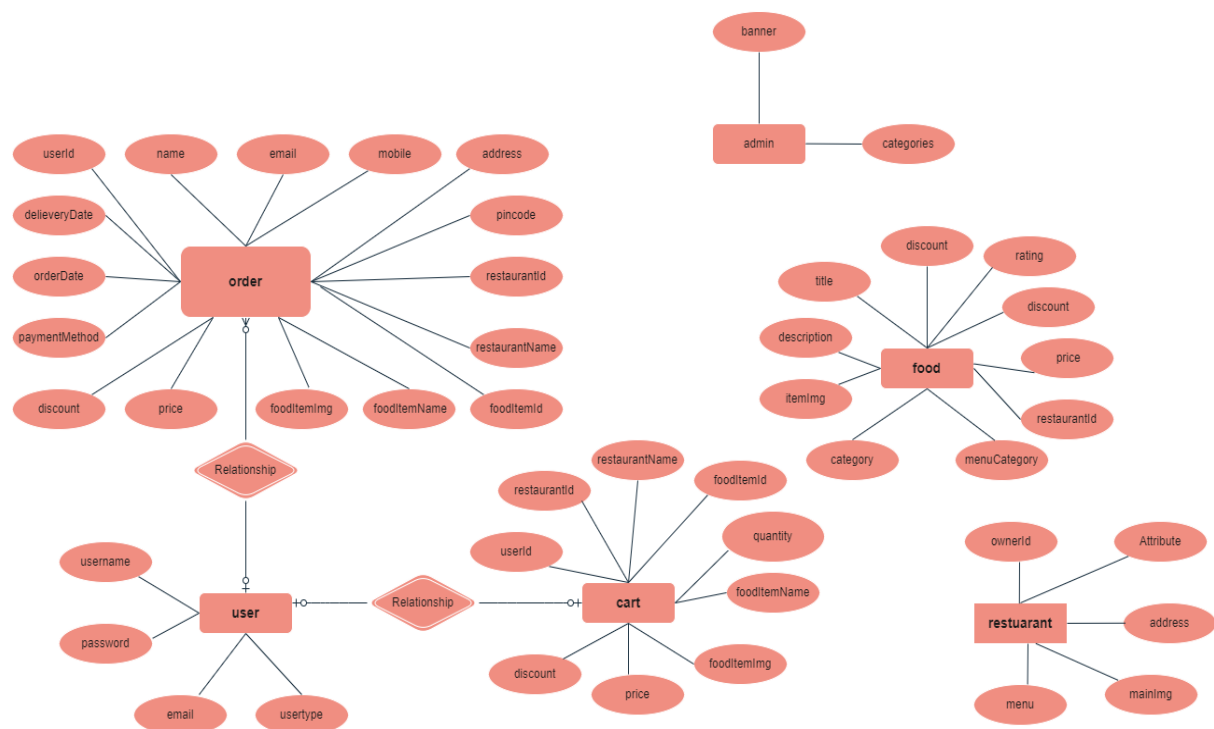


Fig: - Architecture

#### Frontend (React.js)

The frontend is responsible for the visual representation and interaction of the system. It would include:

- **User Authentication:** Forms for login and registration with role-based access (customer, admin, restaurant owner).
- **Dashboard:** Role-specific dashboards showing order history (user), food/menu management (owner), and analytics (admin).
- **Food Browsing:** Interface to view food items with filters like category, rating, discount, and price.
- **Cart Management:** Option to add/remove items, update quantity, and view total price before ordering.
- **Order placement:** A checkout page with options for delivery details, payment method, and discount application.

## Backend (Express.js & Node.js)

The backend handles the server-side operations, including:

- **Authentication:** Verifies user credentials and grants access.
- **Authorization:** Determines the permissions for each user role.
- **Order Management:** Interacts Handles order creation, delivery date tracking, payment method processing, and status updates details.
- **Business Logic:** Implements logic for price calculation (with discounts), food availability, and order timings.
- **Food & Restaurant Management:** Admin and owners can create, update, or delete food items and manage restaurant details.
- **Data Validation & Security:** Ensures clean, secure data handling through input validation, hashing passwords, and secure API communication.

## Database (MongoDB)

The database stores the persistent data of the system. It would typically include:

- **Users:** Collection to store user information including name, email, password, and user role (e.g., customer, admin, restaurant owner).
- **Orders:** Collection to store order details such as order date, delivery date, payment method, price, discount, food item details, and user address.
- **Cart:** Collection to temporarily store food items selected by users before placing an order. Includes item quantity, price, and associated restaurant.
- **Food:** Collection to store food item details including title, description, price, discount, rating, timing, menu category, and restaurant ID.

- **Restaurants:** Collection to store restaurant information like name, address, operating hours, menu, and owner ID.
- **Admins:** Collection to manage admin-related content such as promotional banners and food categories.

## 4. Setup Instructions

### • Prerequisites:

- **Node.js and npm:** Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the server side. It provides a scalable and efficient platform for building network applications.
- **Express.js:** Express.js is a fast and minimalist web application framework for Node.js. It simplifies the process of creating robust APIs and web applications, offering features like routing, middleware support, and modular architecture.
- **MongoDB:** MongoDB is a flexible and scalable NoSQL database that stores data in a JSON-like format. It provides high performance, horizontal scalability, and seamless integration with Node.js, making it ideal for handling large amounts of structured and unstructured data.
- **Moment.js:** Moment.js is a JavaScript package that makes it simple to parse, validate, manipulate, and display date/time in JavaScript. Moment.js allows you to display dates in a human-readable format based on your location. Install React.js, a JavaScript library for building user interfaces
- **React.js:** React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications.

### • Installation:

- React
- Bootstrap
- Material UI
- Axios
- react-bootstrap

- bcryptjs
- express
- dotenv
- mongoose
- Nodemon
- Jsonwebtoken

## 5. Folder Structure

### •Client:

- **public:** contains all the npm packages installed for the React app.
- **src:** Main source folder.
  - **index.html:** Main HTML file where your React app mounts.
  - **favicon.ico:** Browser tab icon.
  - **logo192.png& logo512.png:** App icons used in PWA or splash screens.
  - **services:** API calls and other services.
  - **manifest.json:** Metadata for Progressive Web App (name, icons, colors).
  - **robots.txt:** Instructions for web crawlers (SEO-related).
- **package.json:** Lists dependencies, scripts (like start, build), and metadata for the React app.
- **package-lock.json:** Exact version tree of installed dependencies—used to ensure consistent installs across machines.
- **README.md:** Documentation for your React project—how to install, run, or understand the app.

### • Server:

- **config:** Configuration files, such as database and application settings.

- **controllers**: Logic for handling requests and responses.
- **models**: Database schemas and models.
- **routes**: Route definitions for the API endpoints.
- **middlewares**: Custom middleware functions.
- **schema**: Utility Contains Mongoose schema definitions for MongoDB collections (e.g., User, Book, etc.).
- **seed.js**: Script to prepopulate the database with test data or default values.
- **package.json**: Lists dependencies and scripts.

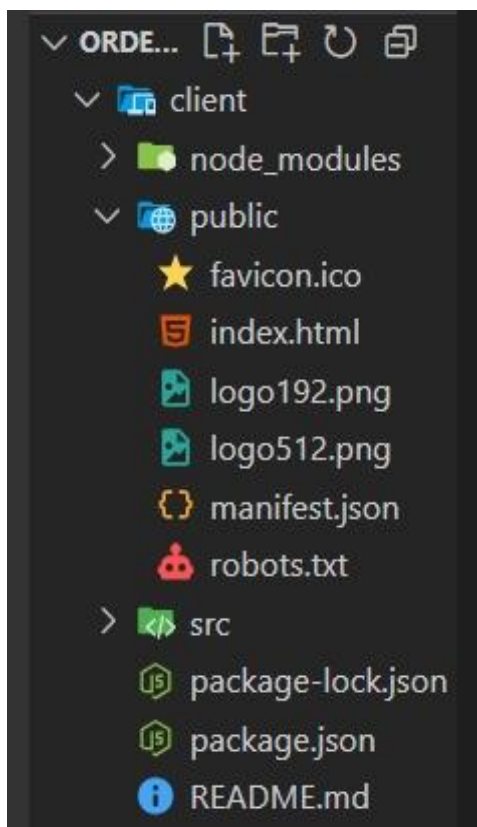


Fig: - Frontend  
Backend

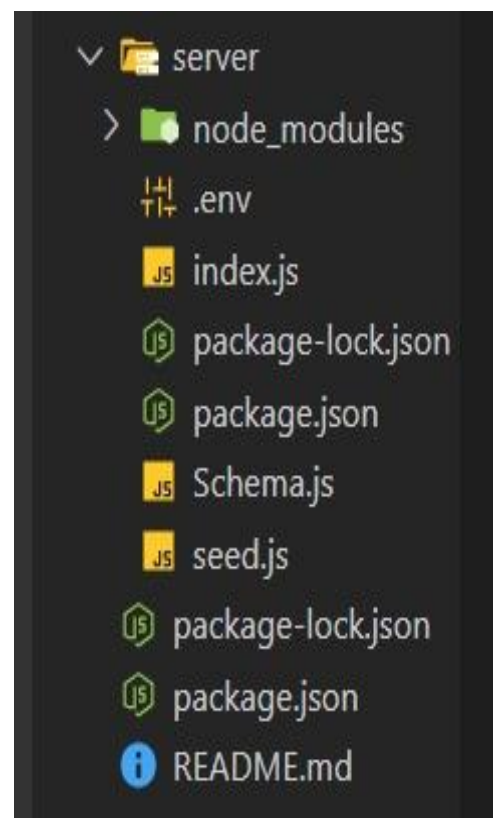


Fig: -

## 6. Running the Application



**Frontend:** npm start -in the client directory

**Backend:** npm run debug- in the server directory.

## 7. Application Flow

```
import dotenv from 'dotenv';
import path from 'path';
import { fileURLToPath } from 'url';
import mongoose from 'mongoose';
import { Restaurant, Admin } from './Schema.js'; // Import Admin model
```

```
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);
```

```
dotenv.config({ path: path.resolve(__dirname, '.env') });
```

```
const MONGO_URL = process.env.MONGO_URL;
```

```
if (!MONGO_URL) {
  console.error(' ✖ MONGO_URL is undefined. ');
  process.exit(1);
}
```

```
const seedRestaurants = async () => {
  try {
```

```

await mongoose.connect(MONGO_URL, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

await Restaurant.deleteMany({});
await Admin.deleteMany({}); // Clear admin data too

const sampleRestaurants = await Restaurant.insertMany([
  {
    ownerId: "seed1",
    title: "McDonald's",
    address: "123 Main Street, New York",
    mainImg:
"https://pngimg.com/uploads/mcdonalds/mcdonalds_PNG9.png",
    menu: []
  },
  {
    ownerId: "seed2",
    title: "Paradise Biryani",
    address: "Hyderabad, India",
    mainImg:
"https://imgmedia.lbb.in/media/2020/05/5ec76dbedb54da5c766f2bf3_15
90128062246.jpg",
    menu: []
  },
  {

```

```
        ownerId: "seed3",
        title: "Minerva Coffee Shop",
        address: "Banjara Hills, Hyderabad",
        mainImg:
        "https://pix10.agoda.net/hotelImages/446/446359/446359_15080415040
        033619151.jpg?s=1024x768",
        menu: []
    }
});
```

```
console.log('✅ Restaurants Seeded Successfully!');
```

```
// Pick McD & Paradise as popular
```

```
const promotedIds = [sampleRestaurants[0]._id,
sampleRestaurants[1]._id];
```

```
const adminDoc = new Admin({
    categories: [],
    promotedRestaurants: promotedIds
});
```

```
await adminDoc.save();
```

```
console.log('✅ Popular Restaurants added to Admin collection!');
```

```
process.exit();
```

```
} catch (error) {
```

```
        console.error(' ✖ Seeding Error:', error);
        process.exit(1);
    }
};
seedRestaurants();
seed,js
```

## API Request/Response Formats

- Request: JSON
- Response: JSON

## API Error Handling

- Error Codes: 400, 404, 500
- Error Messages: Detailed error messages

## 8. Authentication

Explain how authentication and authorization are handled in the project. Include details about tokens, sessions, or any other methods used.

### • Authentication:

- User Registration: Users sign up with username, email, password, and role.
- Passwords: passwords are hashed and stored in MongoDB.
- Login: On valid credentials, a JWT token is generated and sent to the client.
- Token storage: Upon Token is saved on the frontend (e.g., localStorage) for future secure requests.

### • Authorization:

- Role Embedding: Users are assigned roles (admin, doctor, patient).

- Access Control: Routes/features are restricted based on role (e.g., only Admins manage menus).
- Middleware Check: Role-based checks are performed before processing protected actions..

- **Token Management:**

- Token Generation: JWT is generated using user ID, role, and expiration time.
- Token Storage: JWT is stored in local storage on the client-side.
- Token Validation: JWT is validated on each request using middleware.

## 9. User Interface

- Provide screenshots showcasing login page and registration form.

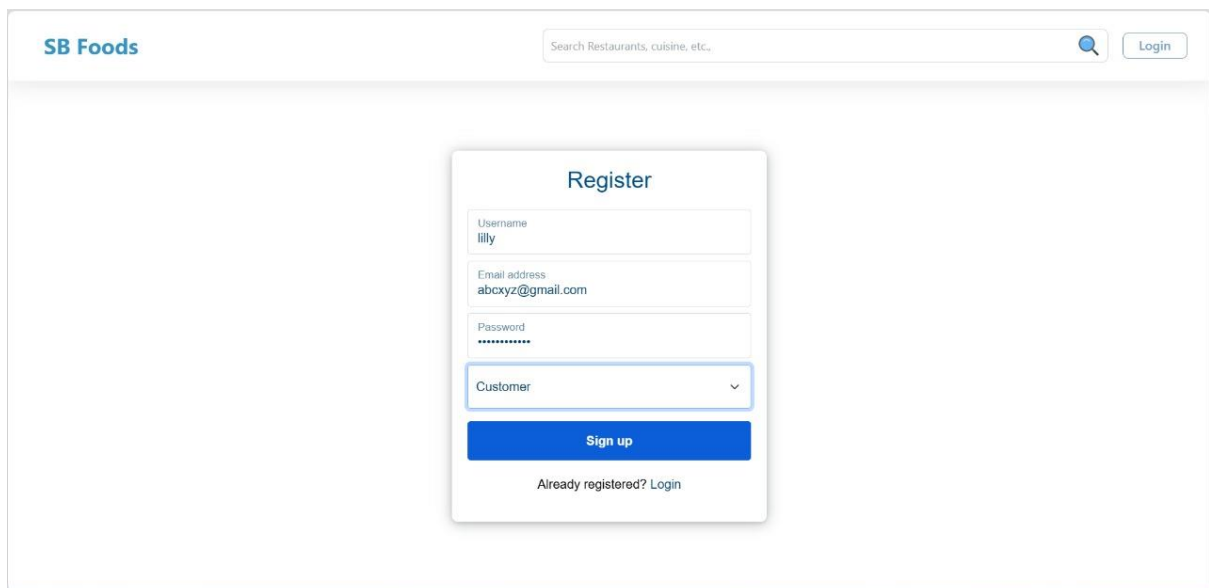
The screenshot shows a web browser window with the 'SB Foods' logo in the top left. A search bar with the placeholder text 'Search Restaurants, cuisine, etc.' and a magnifying glass icon is in the top right, next to a 'Login' button. In the center of the page is a 'Register' form. The form has four input fields: 'Username' with the value 'lily', 'Email address' with the value 'abcxyz@gmail.com', 'Password' with masked characters '\*\*\*\*\*', and a dropdown menu for 'Customer' currently showing 'Customer'. Below these fields is a blue 'Sign up' button. At the bottom of the form, there is a link that says 'Already registered? Login'.

Fig: - Register Form

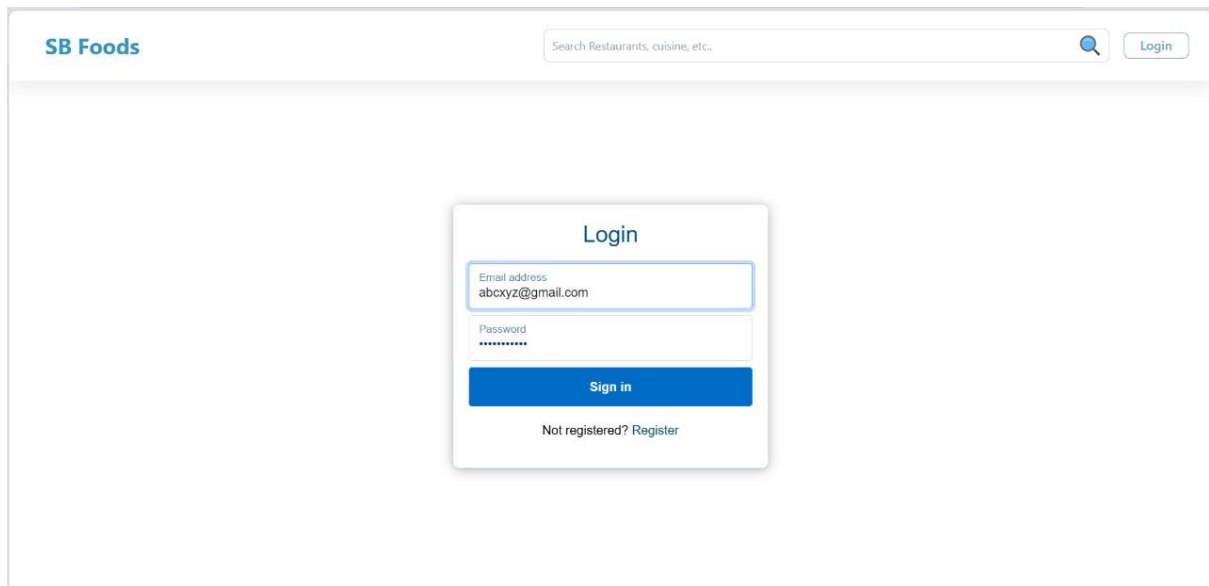


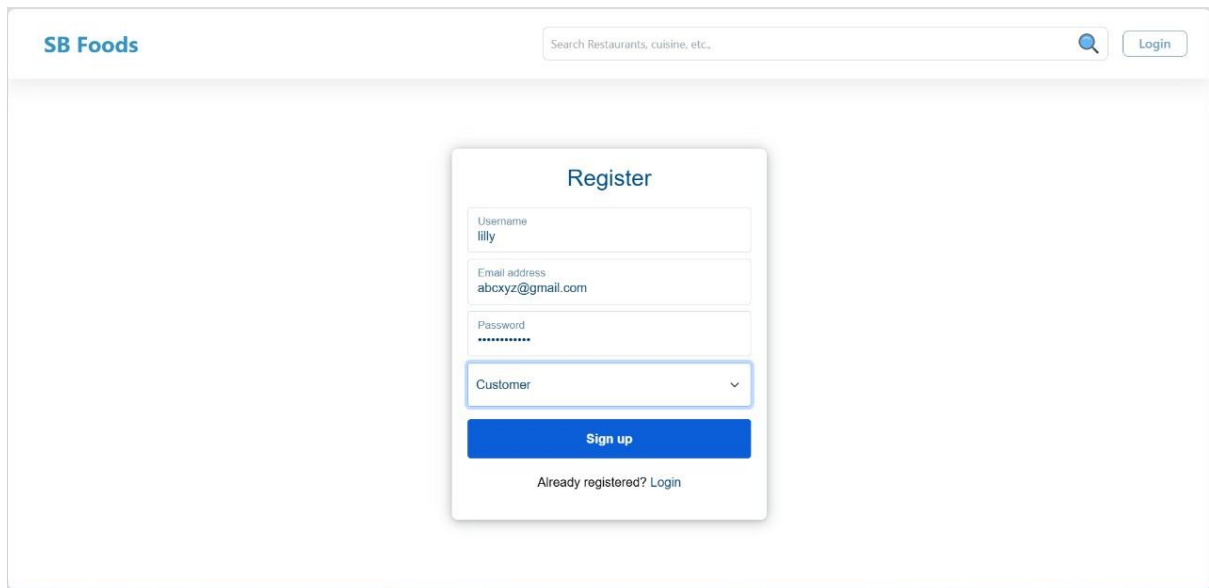
Fig: - Login Page

## 10. Testing: Describe the testing strategy and tools used.

- **API Testing:** Use tools like Postman or Insomnia to test authentication (/register, /login) and protected routes with and without JWT tokens.
- **Unit Testing:** Use Jest or Mocha to write test cases for individual functions like password hashing, token generation, and role checks.
- **Integration Testing:** Test complete flows such as user registration to order placement using tools like Supertest with Express.
- **Frontend Testing:** Use React Testing Library or Jest to test form validations, component rendering, and user interactions.

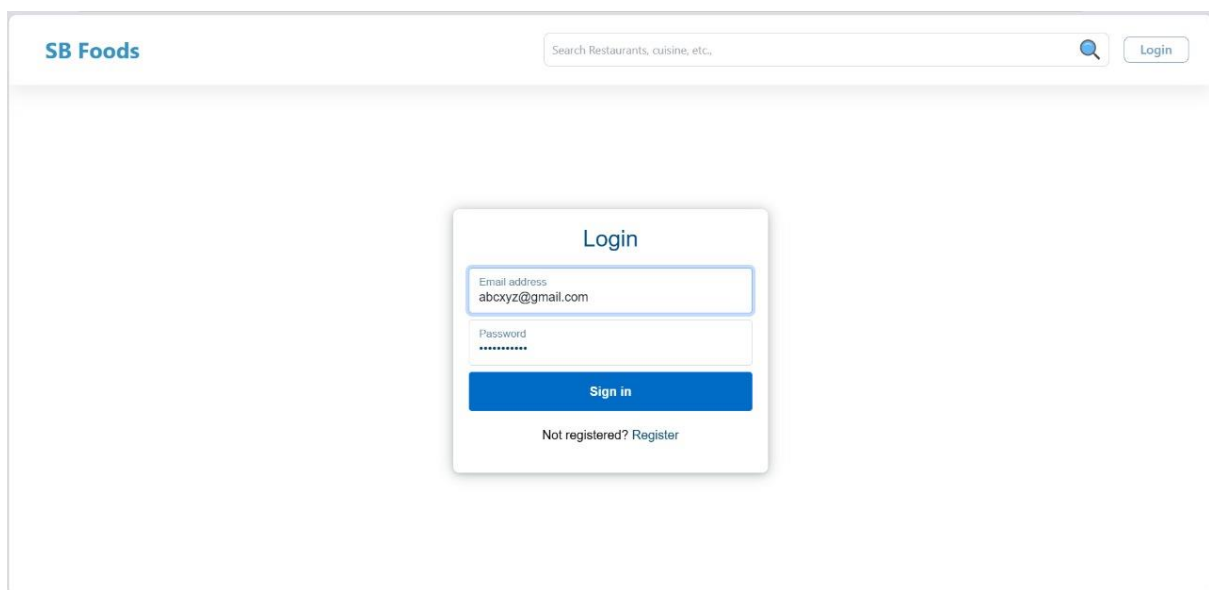
## 11. Screenshots or Demo

### Project Structure



The image shows a web browser window with the "SB Foods" logo in the top left corner. A search bar in the top right contains the placeholder text "Search Restaurants, cuisine, etc." and a magnifying glass icon. To the right of the search bar is a "Login" button. In the center of the page is a "Register" form. The form has a title "Register" and four input fields: "Username" with the value "lily", "Email address" with the value "abcxyz@gmail.com", "Password" with masked characters "\*\*\*\*\*", and a dropdown menu for "Customer" with "Customer" selected. Below the input fields is a blue "Sign up" button. At the bottom of the form is a link that says "Already registered? Login".

Fig: - Registration Form



The image shows a web browser window with the "SB Foods" logo in the top left corner. A search bar in the top right contains the placeholder text "Search Restaurants, cuisine, etc." and a magnifying glass icon. To the right of the search bar is a "Login" button. In the center of the page is a "Login" form. The form has a title "Login" and two input fields: "Email address" with the value "abcxyz@gmail.com" and "Password" with masked characters "\*\*\*\*\*". Below the input fields is a blue "Sign in" button. At the bottom of the form is a link that says "Not registered? Register".

Fig: - Login Page

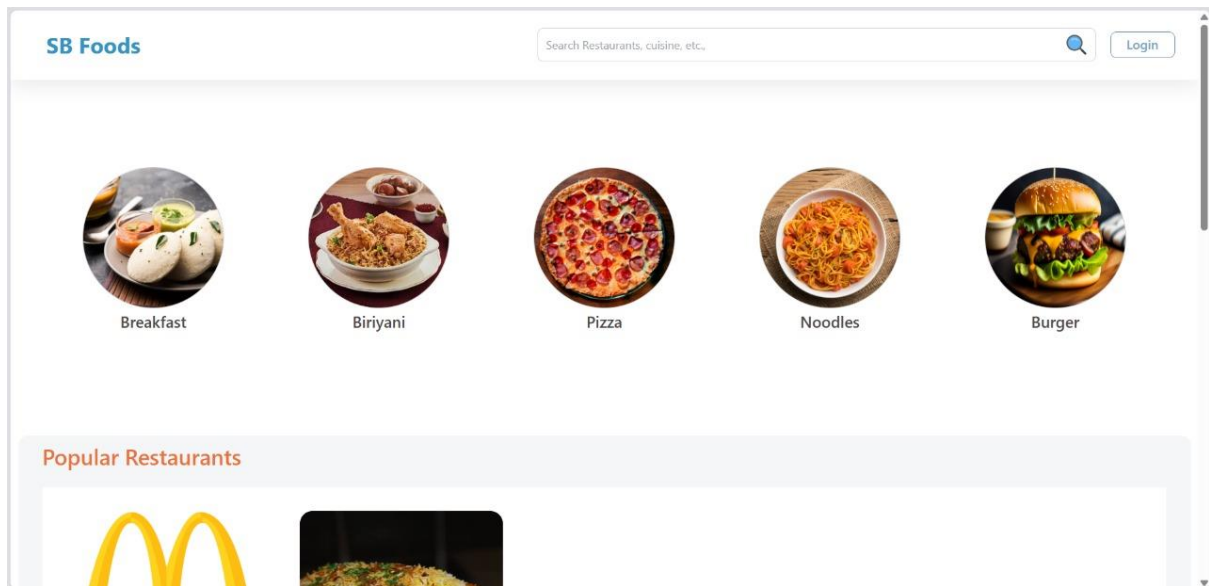


Fig: - User Interface

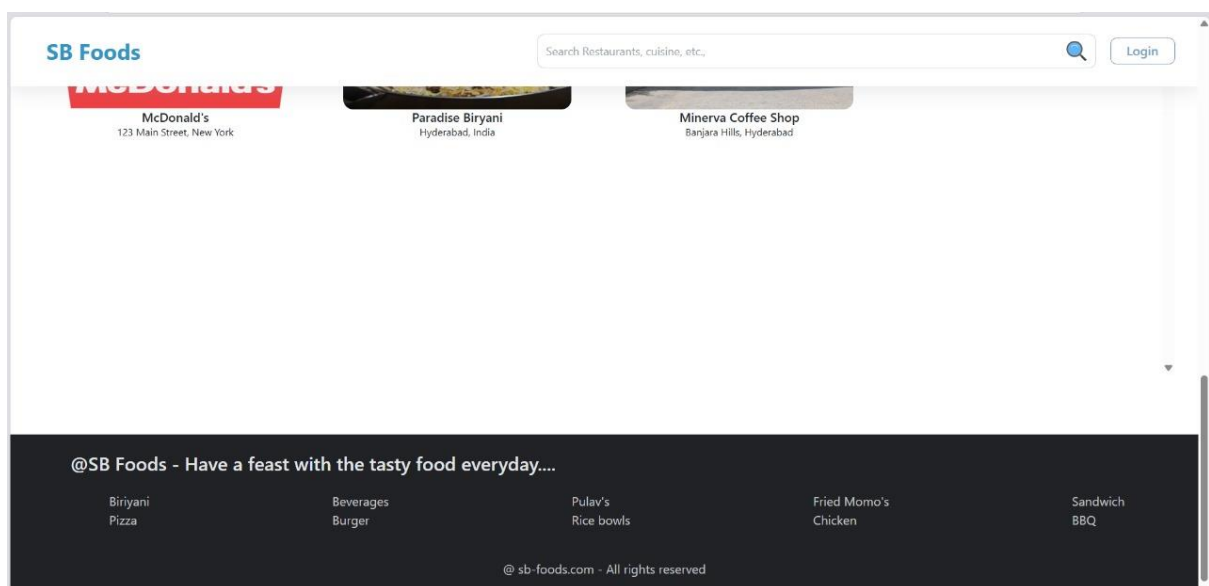


Fig: - Food Browsing



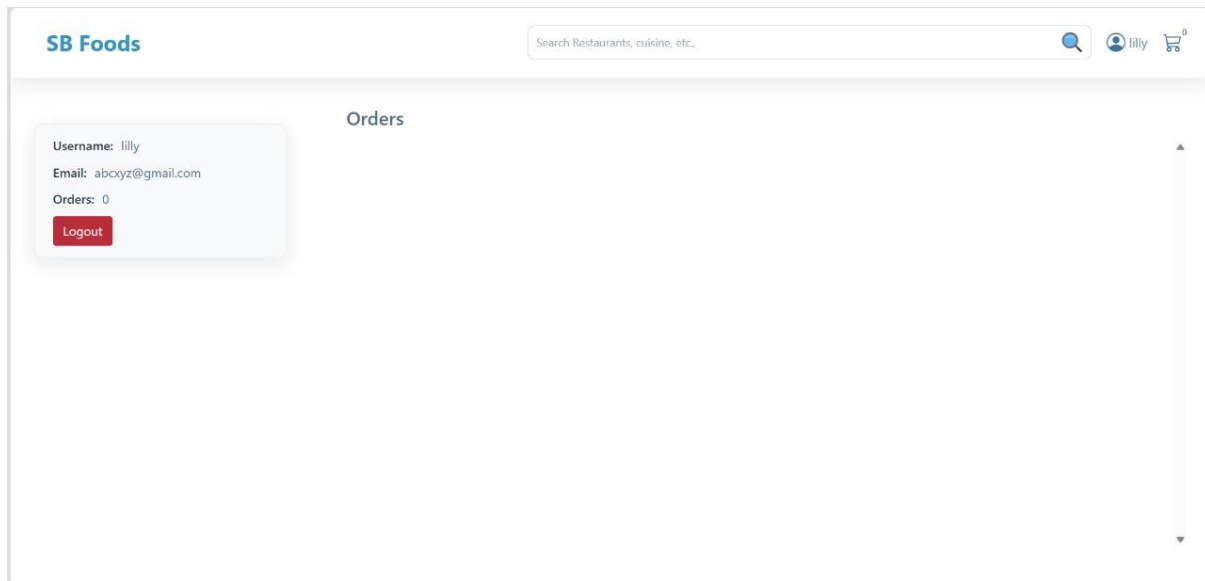


Fig: - Orders

## 12. Known Issues

Document any known bugs or issues that users or developers should be aware of.

- **Backend (Node.js):**

- Error handling: Uncaught exceptions may cause server crashes.
- MongoDB connection issues: Occasional disconnections require manual restart.
- API rate limiting: No rate limiting implemented, vulnerable to abuse.

- **Frontend (React):**

- Browser compatibility: Issues with older browser versions (IE, Safari).
- Mobile responsiveness: Layout issues on smaller screens.
- State management: Complex state updates may cause rendering errors.

- **Database (MongoDB):**

- Data consistency: Lack of transactions may lead to inconsistent data.
- Query optimization: Slow queries impact performance.

- Indexing: Missing indexes cause slow query execution.

- **Security:**

- Authentication: Weak password hashing (BCrypt).
- Authorization: Incomplete role-based access control.
- XSS protection: Insufficient input sanitization.

## **13. Future Enhancements**

Outline potential future features or improvements that could be made to the project.

- Extend schema to include Orders, Cart, and User Roles (Admin, Delivery).
- Implement pagination and lazy loading to improve frontend and backend performance.
- Use React Query or SWR for efficient API calls and caching in the frontend.
- Secure inputs and hash passwords using bcrypt and input sanitization.
- Modularize codebase, set up Docker and CI/CD pipelines for smooth deployment.