A
PROJECT REPORT ON

Online Food Ordering in Canteen
(Reduce waiting time)

By

KRISH MAKADIA (CE-063) (20ECUOG058)
JASH SHAH (CE-046) (20CEUON126)

B.Tech CE Semester-V Subject:
ADVANCED TECHNOLOGIES

Guided by: *Prof.Siddharth P. shah* and Prof. Prashant Jadav

Faculty of Technology
Department of Computer Engineering
Dharmsinh Desai University

Faculty of Technology
Department of Computer Engineering
Dharmsinh Desai University

CERTIFICATE

This is to certify that the practical / **term work** carried out in the subject of

Advanced Technologies and recorded in this journal is the bonafide

work of

KRISH MAKADIA (CE-063) (20ECUOG058)
JASH SHAH (CE-046) (20CEUON126)

of B.Tech semester **V** in the branch of **Computer Engineering**

during the academic year **2022-2023**.

Prof. Siddharth P. Shah
Assistant Professor,
Dept. of Computer Engg.,
Faculty of Technology
Dharmsinh Desai University, Nadiad

Dr. C. K. Bhensdadia,
Head,
Dept. of Computer Engg.,
Faculty of Technology
Dharmsinh Desai University, Nadiad

Contents

Abstract

Several platforms have made their marks and impact on the home food delivery industry, but nothing quite like this has existed before that allows client to order food online at the canteen itself. Our WebApp **online food ordering in canteen** fills this void in a way that is easy to use and effective for users and canteen managers.

Time and tide wait for none, it seems that nowadays time is more valuable than what money is to mankind. Waiting in long queues at the overcrowded canteens don't seem to be the ideal option if all you want to buy is a tiny pack of snack. That's why our application reduces waiting time by allowing the user to order online from his/her device and collect the respective items from the counter when the order is prepared. Customers can choose any item of their choice and enjoy the benefits of paying online.

Admins can improve sales too much extent as they have the complete list of all the orders and can start serving the one that are already available.

Introduction

- **About Project:**

  Online food ordering system in canteen (reduced waiting time) is a web-based canteen management application with food ordering functionality. It connects users and canteen managers in an online community allowing users to browse menu and order food of their choice without having to wait in long queues. Users can pay online and wait for their order to get ready while the management system can handle the menu and orders.

  Our application includes some of the major use cases like user account registration, login/logout, order food online, view order history. Confirming order, manage menu details and categories at owner side.

- **Technology:**

  Our project uses MongoDB Atlas and Node.js to back the interface with strong database functionality and React Framework as frontend. Online food ordering system integrates Cloudinary as a cloud storage for Dynamic image upload. Along with this the use of **Netlify** and **Render** are employed to host the application's frontend and backend parts respectively. This project will target the major web browsers as the initial platform.

- **Tools:**
  - Visual Studio Code (editor)
  - MongoDb compass
  - Github
  - Netlify
  - Render
  - Postman

SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

Online Food Ordering System in Canteen (reduce wait time)

R 1: User side

Description: User can see only view the home page of the application, to access further functionalities user must sign up first or login if already registered.

R 1.1: Authentication

Description: If the user is new to the system, they must register with the application else they can login if already registered previously.

R 1.1.1: Register

Input: User details.
Process: Validate details.
Output: Redirect to menu page.

R 1.1.2: Sign In

Input: User details.
Process: Validate details.
Output: Redirect to menu page.

R 1.1.3: Log Out

Input: User Selection.
Output: Redirect to Home page.

R 1.2: Menu

Description: as per user selection system shows different food categories

R 1.2.1: Menu item details

Input: User selection.
Output: list the food item details.

R 1.2.2: Add/Remove

Input: User selection.

Process: Total quantity of corresponding item gets added or subtracted based on user selection.

Output: Adds or removes the respective food item along with quantity from the food cart.

.

R 1.3: Orders History

Description: it will show list and status of all orders of the respective user.

R 1.4: Cart

Description: Display the contents of the cart based on the selection

R 1.4.1: Place Order

Input: User selection

Process: Payment processing

Output: User redirected to respective order's page

R 1.4.2: Clear Cart

Input: User selection

Process: All the cart items are removed

Output: Cart is cleared

R 2: Admin Side

Description: The first user who registers for the application is assigned the role of admin and the subsequent users are assigned the normal users. After approval the admin will be redirected to admin menu page.

R 2.1: Authentication

Description: If user is new to the system then first he/she must sign up to the system otherwise user can directly login and then enter to the system.

R 2.1.1: Sign Up

Input: User details.

Process: Validate details.

Output: Redirect to admin page.

R 2.1.2: Sign In

Input: User details.

Process: Validate details.
Output: Redirect to admin page

R 2.1.4: Log Out
Input: User Selection.
Output: Redirect to Home page.

R 2.2: Manage Menu
Description: Shows the food categories and respective food items.

R 2.2.1 Manage Categories
Description: Shows the existing food item categories in the menu.

R 2.2.1.1 Add Category
Input: User selection
Output: Category Added

R 2.2.1.2 Edit Category
Input: User selection
Output: Category updated

R 2.2.1.3 Delete Category
Input: User selection
Output: Category deleted

R 2.2.2 Manage Food Items
Description: Lists all the food items corresponding to the category selected.

R 2.2.2.1 Add Food Item
Input: User selection
Output: Food item corresponding to the respective category added

R 2.2.2.2 Edit Food Item
Input: User selection
Output: Food item corresponding to the respective category updated

R 2.2.2.3 Delete Food Item
Input: User selection

Output: Food item corresponding to the respective category deleted

R 2.3: Manage Orders

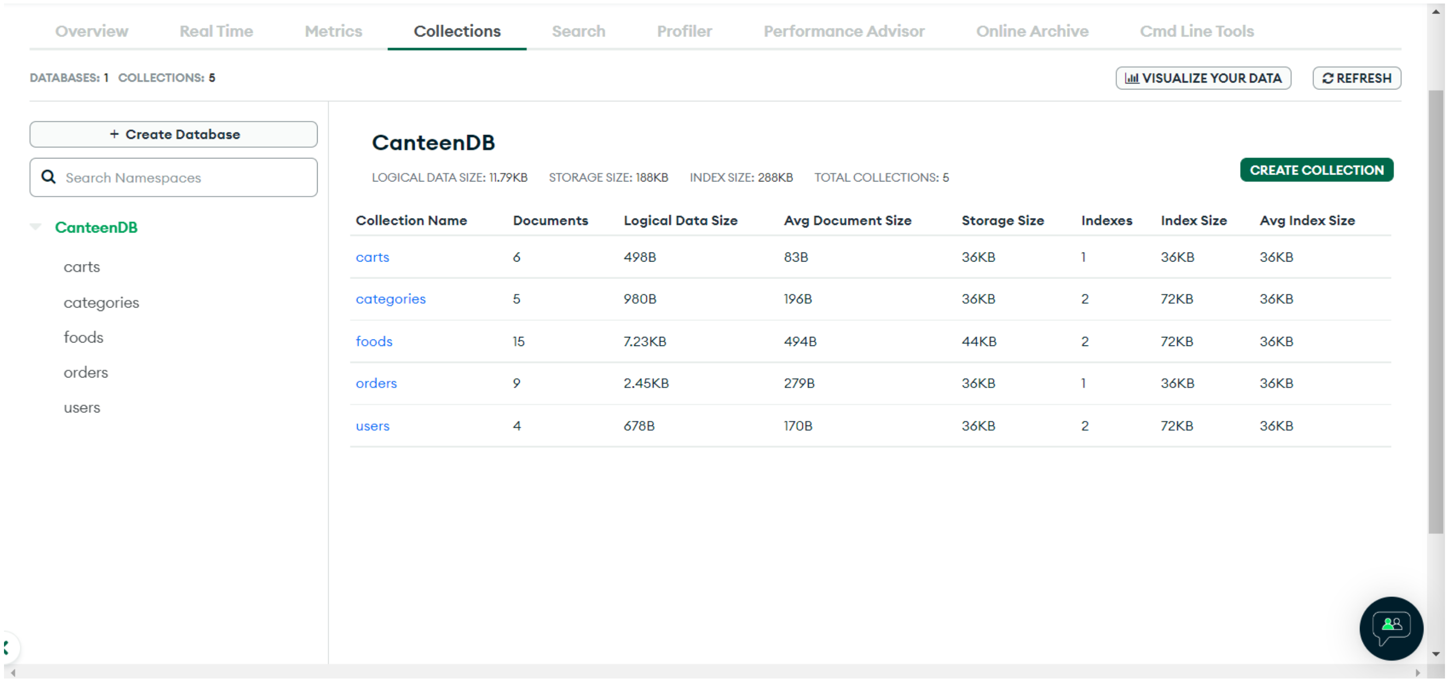
Description: Shows all the pending as well as completed orders.

R 2.3.1 Order Prepared

Input: User Selection.
Process: Order status changes from pending to approved.
Output: Order of the respective customer is finished.

Data Dictionary

Tables



DATABASES: 1 COLLECTIONS: 5

CanteenDB
- carts
- categories
- foods
- orders
- users

CanteenDB

LOGICAL DATA SIZE: 11.79KB STORAGE SIZE: 188KB INDEX SIZE: 288KB TOTAL COLLECTIONS: 5

Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
carts	6	498B	83B	36KB	1	36KB	36KB
categories	5	980B	196B	36KB	2	72KB	36KB
foods	15	7.23KB	494B	44KB	2	72KB	36KB
orders	9	2.45KB	279B	36KB	1	36KB	36KB
users	4	678B	170B	36KB	2	72KB	36KB

Cart:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const CartItemsSchema = new Schema({
  _id: {
    type: mongoose.Types.ObjectId,
    ref: 'Food',
    required: [true, 'Please provide Food Item id'],
  },
  name: {
    type: String,
    required: [true, 'Please provide Food Item name'],
  },
  quantity: {
    type: Number,
    required: [true, 'Please provide Food Item quantity'],
  },
  price: {
    type: Number,
    required: [true, 'Please provide Food Item price'],
  },
});
```

```
const CartSchema = new Schema({
  userId: {
    type: mongoose.Types.ObjectId,
    ref: 'User',
    required: [true, 'Please provide a user'],
  },
  cartItems: [CartItemsSchema],
  totalPrice: {
    type: Number,
    default: 0,
  },
});

module.exports = mongoose.model('Cart', CartSchema);
```

Category:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const CategorySchema = new Schema({
  name: {
    type: String,
    required: [true, 'Please provide name of the category'],
    minlength: [3, 'Name cannot be less than 3 characters'],
    maxlength: [40, 'Name cannot be more than 40 characters'],
    unique: true,
  },
  slug: {
    type: String,
  },
  image: {
    type: String,
    default:
      'https://res.cloudinary.com/leantuts/image/upload/v1667803848/canteen-
backend/category/default.jpg',
  },
});

CategorySchema.pre('save', function () {
  let slugName = this.name;
  this.slug = slugName
    .toString()
    .trim()
    .toLowerCase()
    .replace(/\s+/g, '-')
    //eslint-disable-next-line
    .replace(/[^\w\-]+/g, '')
    //eslint-disable-next-line
    .replace(/\-\-+/g, '-')
    .replace(/^-+/, '')
    .replace(/-+$/, '');
});

module.exports = mongoose.model('Category', CategorySchema);
```

Food:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const FoodSchema = new Schema({
  category: {
    type: mongoose.Types.ObjectId,
    ref: 'Category',
    required: [true, 'Please provide id of category'],
  },
  slug: { type: String },
  name: {
    type: String,
    minlength: [3, 'Name cannot be less than 3 characters'],
    maxlength: [40, 'Name cannot be more than 40 characters'],
    unique: true,
    required: [true, 'Please provide name of the food item'],
  },
  price: {
    type: Number,
    required: [true, 'Please provide the price of the food item'],
  },
  inStock: {
    type: Boolean,
    default: true,
  },
  description: {
    type: String,
    required: [true, 'Please provide the description of the food item'],
  },
  image: {
    type: String,
    default:
      'https://res.cloudinary.com/leantuts/image/upload/v1667803945/canteen-
backend/food/default.jpg',
  },
});
```

```
FoodSchema.pre('save', function () {
  let slugName = this.name;
  this.slug = slugName
    .toString()
    .trim()
    .toLowerCase()
    .replace(/\s+/g, '-')
    //eslint-disable-next-line
    .replace(/[^\w\-]+/g, '')
    //eslint-disable-next-line
    .replace(/\-\-+/g, '-')
    .replace(/^-+/, '')
    .replace(/-+$/, '');
});

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

Order:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const CartItemsSchema = new Schema({
  _id: {
    type: mongoose.Types.ObjectId,
    ref: 'Food',
    required: [true, 'Please provide Food Item id'],
  },
  name: {
    type: String,
    required: [true, 'Please provide Food Item name'],
  },
  quantity: {
    type: Number,
    required: [true, 'Please provide Food Item quantity'],
  },
  price: {
    type: Number,
    required: [true, 'Please provide Food Item price'],
  },
});

const OrderSchema = new Schema(
  {
    userId: {
      type: mongoose.Types.ObjectId,
      ref: 'User',
      required: [true, 'Please provide a user'],
    },
    totalPrice: {
      type: Number,
      required: [true, 'Please provide the total price'],
    },
    isPrepared: {
      type: Boolean,
      default: false,
    },
    orderItems: [CartItemsSchema],
  },
  { timestamps: true }
);

module.exports = mongoose.model('Order', OrderSchema);
```

User:

```
const mongoose = require('mongoose');
const validator = require('validator');
const bcrypt = require('bcryptjs');

const Schema = mongoose.Schema;

const UserSchema = new Schema({
  name: {
    type: String,
    required: [true, 'Please provide a name'],
    minlength: [3, 'Name cannot be less than 3 characters'],
    maxlength: [30, 'Name cannot be more than 30 characters'],
  },
  email: {
    type: String,
    required: [true, 'Please provide an email'],
    unique: true,
    validate: {
      validator: validator.isEmail,
      message: 'Please provide valid email',
    },
  },
  password: {
    type: String,
    required: [true, 'Please provide a password'],
    minlength: [6, 'Password should be more than 6 characters long'],
  },
  role: {
    type: String,
    enum: ['admin', 'user'],
    default: 'user',
  },
});

UserSchema.pre('save', async function () {
  if (!this.isModified('password')) return;
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
});

UserSchema.methods.comparePassword = async function (checkPassword) {
  return await bcrypt.compare(checkPassword, this.password);
};

module.exports = mongoose.model('User', UserSchema);
```

Implementation Details

- **Modules created and brief description of each module**

  This Project consists of 4 major modules.

  1) User Module
  2) Menu Module
  3) Cart Module
  4) Orders Module

  Each module consists of several methods to implement the required functionality Implementation is done using MERN.

1) User Module

There are three types of Users: Admin, Owner, Customer. This module is the base for authentication and authorization to ensure the security aspect of the user.

2) Menu Module

This module is responsible for listing various categories of food items and food items associated with them. It also shows makes other information available such as item price or whether item is out of stock or not and along with that lets the user add the corresponding food item to the cart. Admins can customize menu based on the items served and availability.

3) Cart Module

This module lists all the food items selected by the user and shows a brief order summary which includes total bill and order details.

4) Orders Module

It is accountable to list all the previous successful orders placed by the corresponding user. Admins are able to see the orders of all the users but an individual user can only see his/her respective previous orders. Admin can approve the order when prepared and thus changing the status of order from pending to completed.

➢ Function prototypes which implement major functionality

1) User Module

Login

```
const { StatusCodes } = require('http-status-codes');
const UserModel = require('../../models/User');
const Errors = require('../../errors');
const JWT = require('../../utils/jwt');

const login = async (req, res) => {
  const { email, password } = req.body;
  if (!email || !password) {
    throw new Errors.BadRequestError('Email or Password missing');
  }

  const user = await UserModel.findOne({ email });
  if (!user) {
    throw new Errors.UnauthenticatedError('Wrong credentials entered');
  }

  const passwordMatch = await user.comparePassword(password);
  if (!passwordMatch) {
    throw new Errors.UnauthenticatedError('Wrong credentials entered');
  }

  const userPayload = { userId: user._id, name: user.name, role: user.role };
  JWT.createJWT(res, userPayload);

  return res.status(StatusCodes.OK).json({ user: userPayload });
};

module.exports = login;
```

Logut

```
const { StatusCodes } = require('http-status-codes');

const logout = async (req, res) => {
  res.cookie('token', 'logout', {
    httpOnly: true,
    expires: new Date(Date.now() + 10),
    secure: process.env.NODE_ENV === 'production',
    signed: true,
    sameSite: 'none',
  });
  return res.status(StatusCodes.OK).send();
};

module.exports = logout;
```

Register

```
const { StatusCodes } = require('http-status-codes');
const UserModel = require('../../models/User');
const JWT = require('../../utils/jwt');
const createCart = require('../cart/createCart');
const Errors = require('../../errors');

const register = async (req, res) => {
  const isFirstAccount = (await UserModel.countDocuments({})) === 0;
  let user;
  let cart;

  if (isFirstAccount) {
    user = await UserModel.create({ ...req.body, role: 'admin' });
  } else {
    const emailAlreadyExists = await UserModel.findOne({
      email: req.body.email,
    });
    if (emailAlreadyExists) {
      throw new Errors.BadRequestError('Email already exists');
    }

    user = await UserModel.create({ ...req.body, role: 'user' });
    cart = await createCart(req, res, user._id);
  }

  const userPayload = { userId: user._id, name: user.name, role: user.role };
  JWT.createJWT(res, userPayload);

  return res.status(StatusCodes.CREATED).json({ user: userPayload, cart });
};

module.exports = register;
```

2) Menu Module

Get All Categories

```
const CategoryModel = require("../../models/Category");
const { StatusCodes } = require("http-status-codes");

const getAllCategories = async (req, res) => {
  const categories = await CategoryModel.find({});
  res.status(StatusCodes.OK).json({ categories, count: categories.length });
};

module.exports = getAllCategories;
```

Get food corresponding to category

```
const FoodModel = require('../../models/Food');
const { StatusCodes } = require('http-status-codes');

const getAllFood = async (req, res) => {
  const food = await FoodModel.find({});
  res.status(StatusCodes.OK).json({ food, count: food.length });
};

module.exports = getAllFood;
```

3) Cart

Get Cart of the current user

```
const CartModel = require('../../models/Cart');
const { StatusCodes } = require('http-status-codes');
const Errors = require('../../errors');

const getCurrentUserCart = async (req, res) => {
  if (req.user.role === 'admin') {
    return res
      .status(StatusCodes.NOT_FOUND)
      .json({ msg: 'Admin cannot have a cart' });
  }
  const userCart = await CartModel.findOne({ user: req.user.userId });
  if (!userCart) {
    throw new Errors.NotFoundError(`No cart with userId ${id}`);
  }
  res.status(StatusCodes.OK).json({ userCart });
};
```

```
module.exports = getCurrentUserCart;
```

Clear the cart

```
const CartModel = require('../../models/Cart');
const { StatusCodes } = require('http-status-codes');
const Errors = require('../../errors');

const clearCart = async (req, res) => {
  const cart = await CartModel.findOne({ userId: req.user.userId });
  if (!cart) {
    throw new Errors.NotFoundError(`No cart with userId ${id}`);
  }
  cart.totalPrice = 0;
  cart.cartItems = [];
  await cart.save();
  res.status(StatusCodes.OK).json({ cart });
};

module.exports = clearCart;
```

4) Order Module

Place order

```
const OrderModel = require('../../models/Order');
const { StatusCodes } = require('http-status-codes');

const createOrder = async (req, res) => {
  req.body.userId = req.user.userId;
  const order = await OrderModel.create({ ...req.body });
  return res.status(StatusCodes.CREATED).json({ order });
};

module.exports = createOrder;
```

Get all orders

```
const OrderModel = require("../../models/Order");
const { StatusCodes } = require("http-status-codes");

const getAllOrders = async (req, res) => {
  const orders = await OrderModel.find({});
  res.status(StatusCodes.OK).json({ orders, count: orders.length });
};
```

```
module.exports = getAllOrders;
```

Get current user order

```
const OrderModel = require('../../models/Order');
const { StatusCodes } = require('http-status-codes');
const Errors = require('../../errors');

const getCurrentUserOrders = async (req, res) => {
  if (req.user.role === 'admin') {
    return res
      .status(StatusCodes.NOT_FOUND)
      .json({ msg: 'Admin cannot have orders' });
  }
  const userOrders = await OrderModel.find({ userId: req.user.userId });
  if (!userOrders) {
    throw new Errors.NotFoundError(`No orders with userId ${id}`);
  }
  res
    .status(StatusCodes.OK)
    .json({ orders: userOrders, count: userOrders.length });
};

module.exports = getCurrentUserOrders;
```

Update order

```
const OrderModel = require('../../models/Order');
const { StatusCodes } = require('http-status-codes');
const Errors = require('../../errors');

const updateOrder = async (req, res) => {
  const { id: orderId } = req.params;
  const order = await OrderModel.findOneAndUpdate(
    { _id: orderId },
    { ...req.body },
    { new: true, runValidators: true }
  );
  if (!order) {
    throw new Errors.NotFoundError(`No order with id ${orderId}`);
  }
  res.status(StatusCodes.OK).json({ order });
};

module.exports = updateOrder;
```

Testing

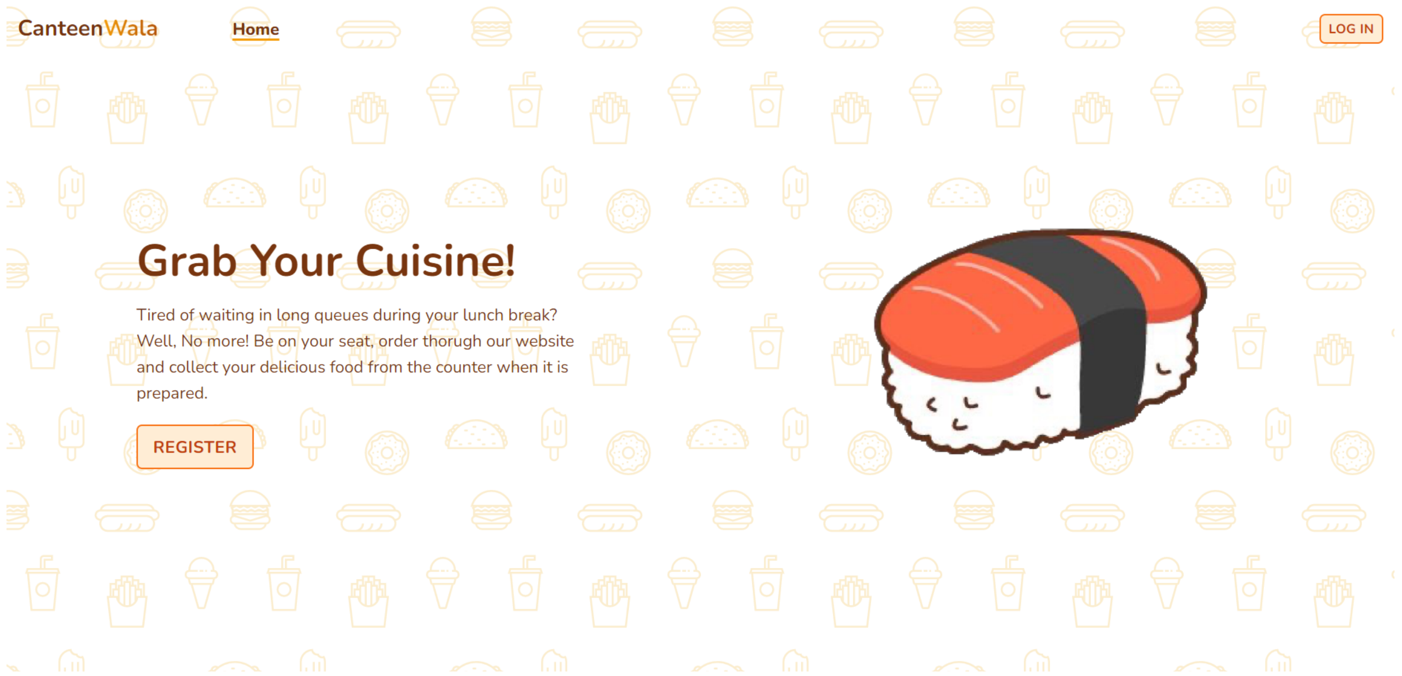
Test Cases

Sr No	Test Case Objective	Input Data	Expected Output	Actual Output	Status
1	Authentication of customer and owner	Credential	Success or error message	Success or error message	Pass
2	List of categories	Selection of category	List of food items based on selected category	List of food items of category	Pass
3	Add food item to cart	Item selection	Item Added to cart with quantity	Item successfully added to cart with quantity	Pass
4	Clear the cart	User selection	Contents of cart are deleted	Cart is cleared successfully	Pass
5	Place order	User selection	Order placed successfully and user redirected to orders page	Order placed and status is pending	Pass
6	Manage categories	Adding, updating and deleting of categories with correct credentials	Categories successfully added, updated or deleted respectively	List of categories is successfully modified with the changes	Pass
7	Manage food items	Adding, updating and deleting of food items with correct credentials	Food items successfully added, updated or deleted respectively	List of food items are successfully modified with the changes	Pass
8	Manage orders	Owner selection	Status of order changes from “pending” to “completed” on customer side	Status of order changed successfully	Pass
9	Logout	Customer/owner selection	Logged out of the system and redirected to home page	User successfully logged out of the system	Pass

<u>Screenshots</u>

Customer Interface: -

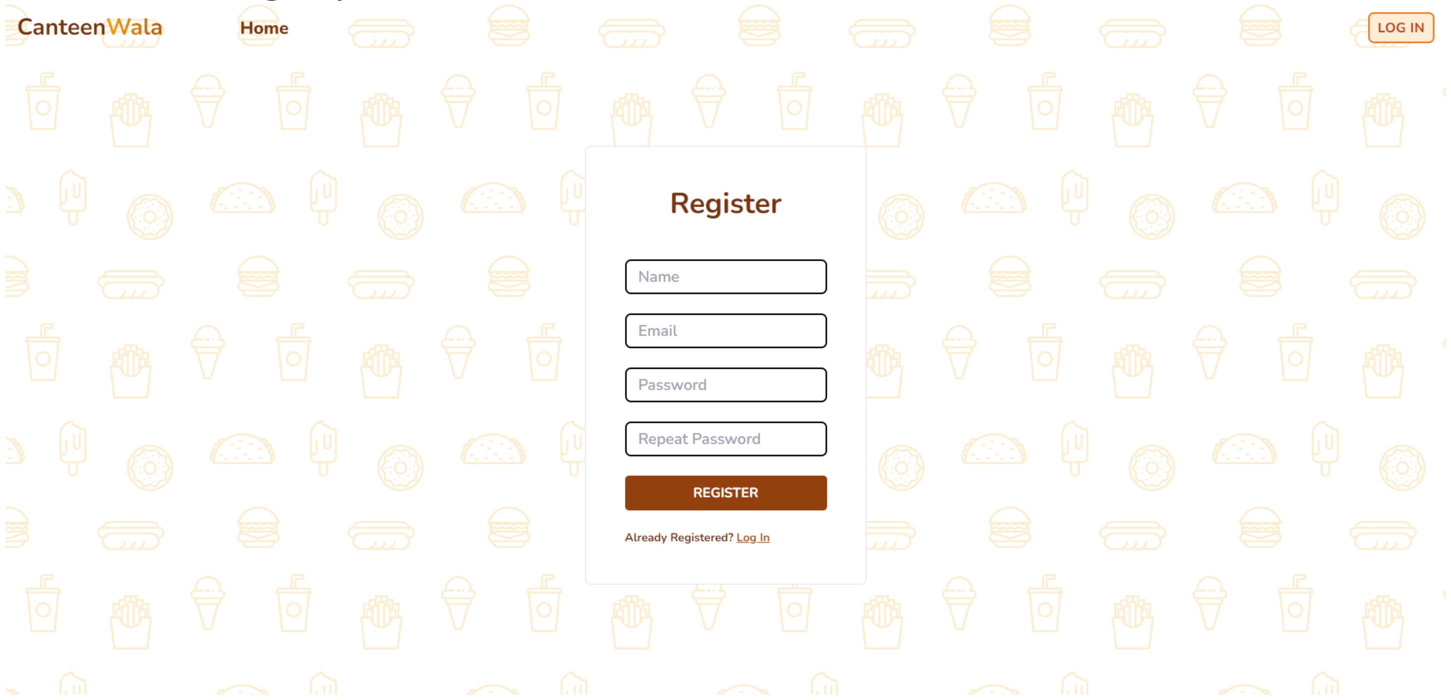
- Home page



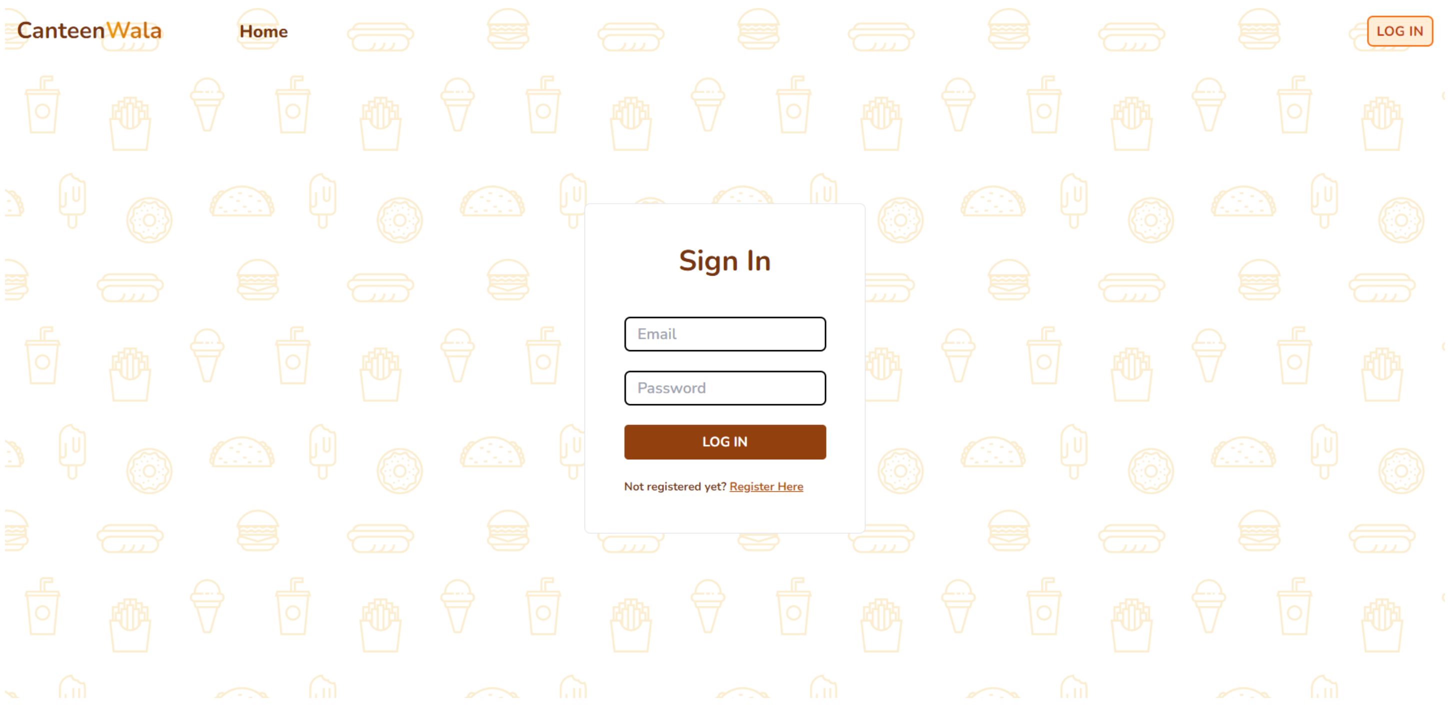
Grab Your Cuisine!

Tired of waiting in long queues during your lunch break? Well, No more! Be on your seat, order thorugh our website and collect your delicious food from the counter when it is prepared.

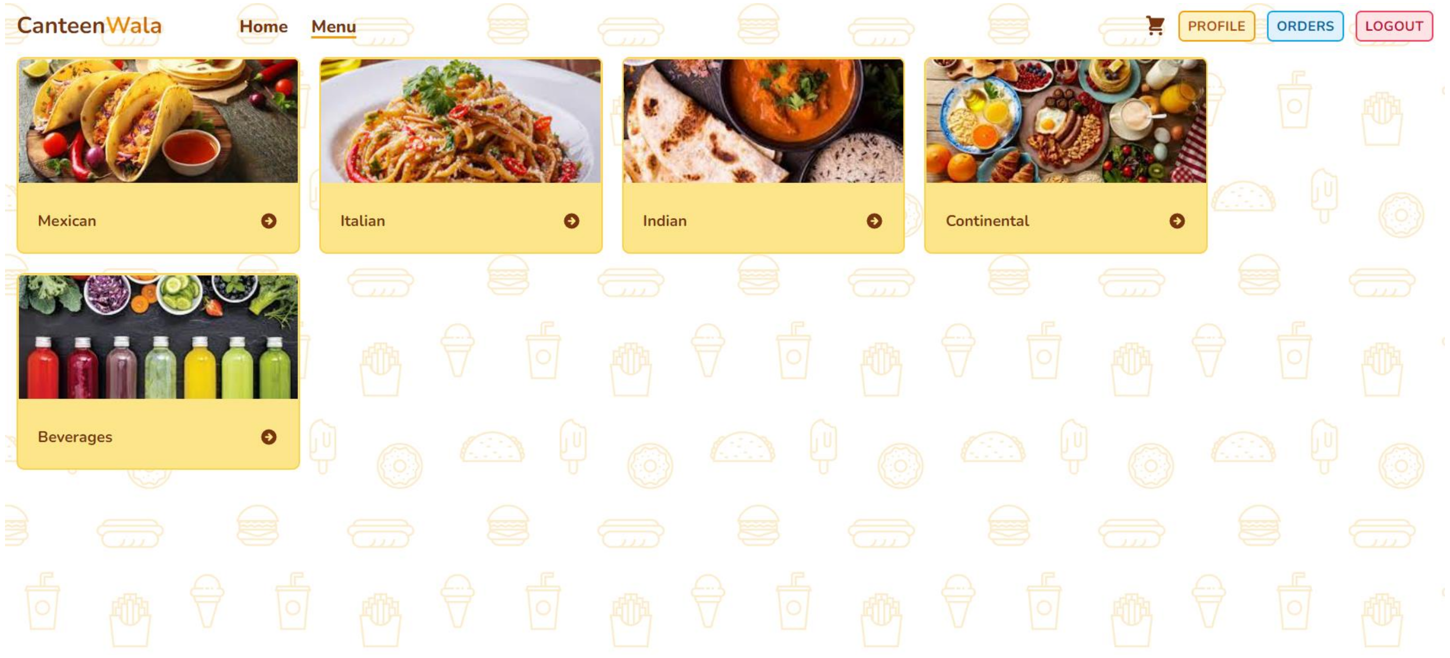
- User signup



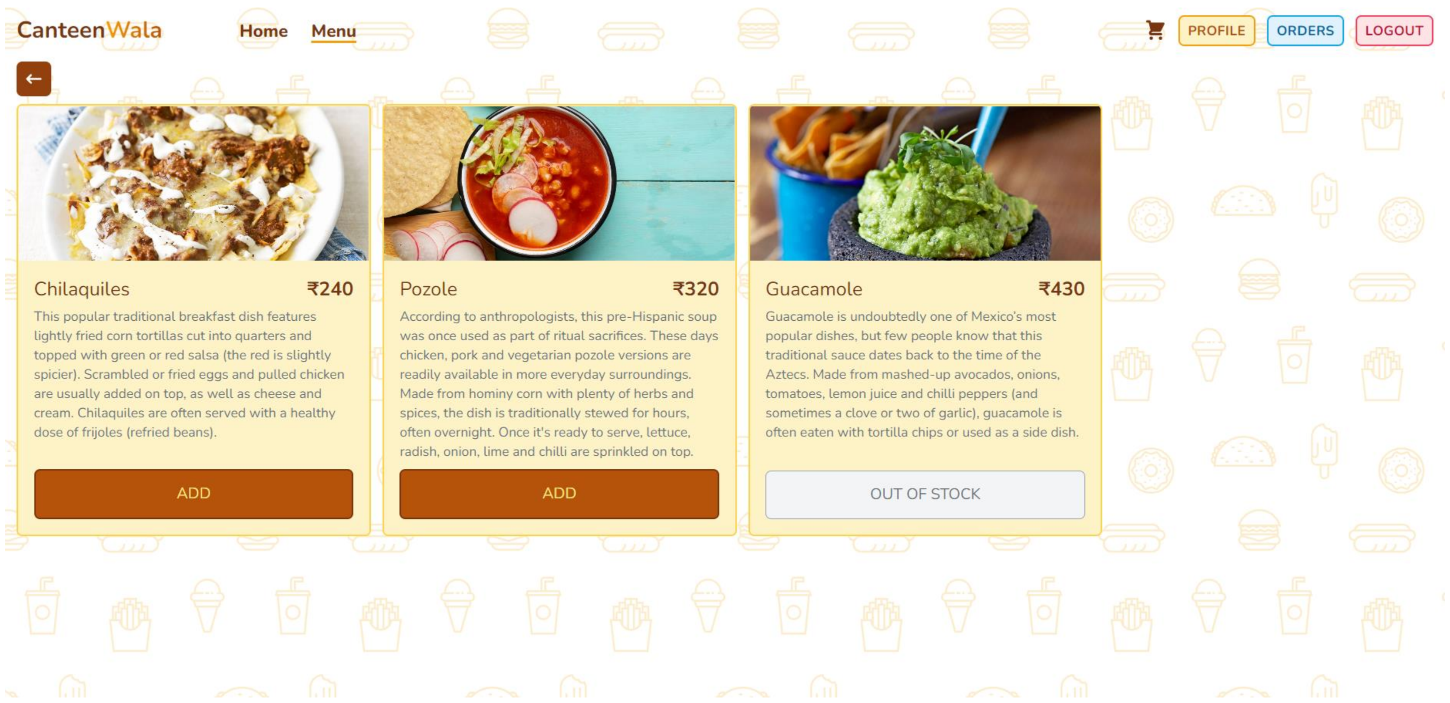
- User Login



- User Menu



- Menu of a category



CanteenWala Home Menu PROFILE ORDERS LOGOUT

Chilaquiles ₹240

This popular traditional breakfast dish features lightly fried corn tortillas cut into quarters and topped with green or red salsa (the red is slightly spicier). Scrambled or fried eggs and pulled chicken are usually added on top, as well as cheese and cream. Chilaquiles are often served with a healthy dose of frijoles (refried beans).

ADD

Pozole ₹320

According to anthropologists, this pre-Hispanic soup was once used as part of ritual sacrifices. These days chicken, pork and vegetarian pozole versions are readily available in more everyday surroundings. Made from hominy corn with plenty of herbs and spices, the dish is traditionally stewed for hours, often overnight. Once it's ready to serve, lettuce, radish, onion, lime and chilli are sprinkled on top.

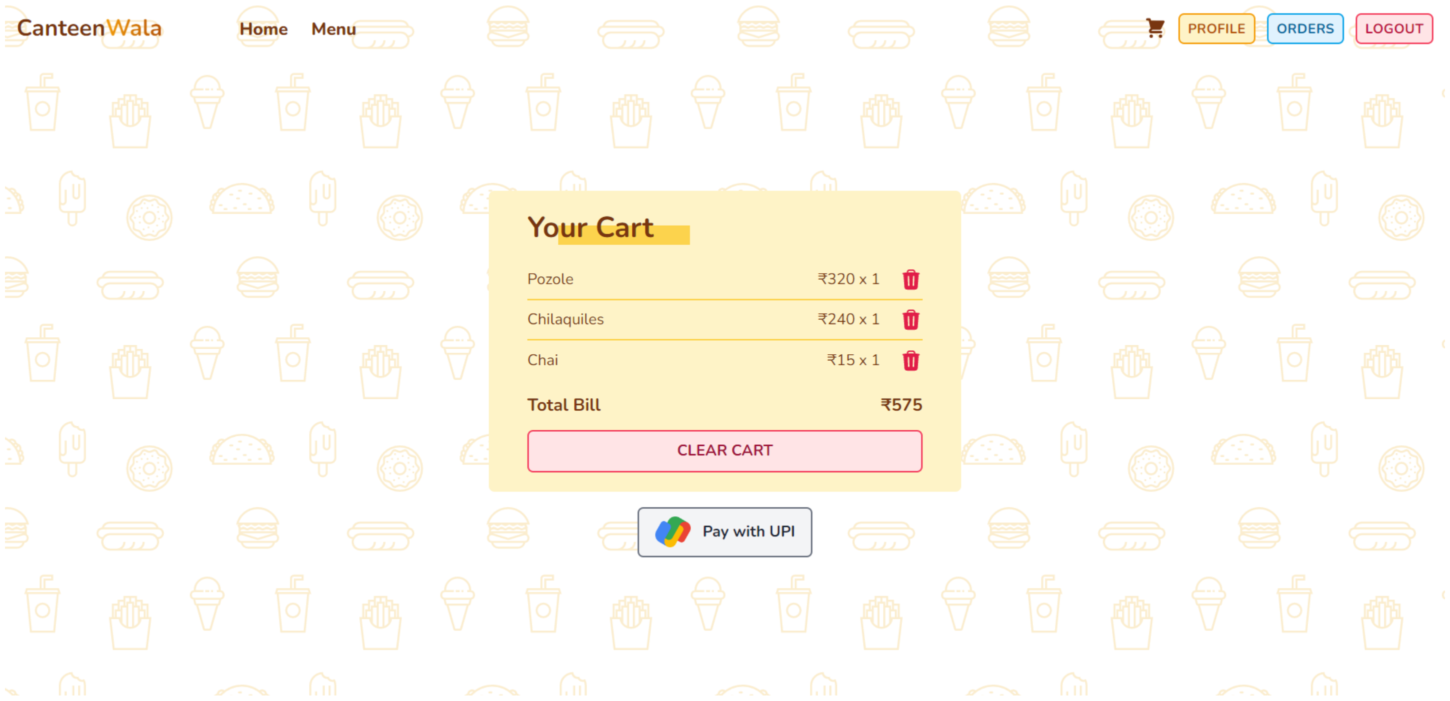
ADD

Guacamole ₹430

Guacamole is undoubtedly one of Mexico's most popular dishes, but few people know that this traditional sauce dates back to the time of the Aztecs. Made from mashed-up avocados, onions, tomatoes, lemon juice and chilli peppers (and sometimes a clove or two of garlic), guacamole is often eaten with tortilla chips or used as a side dish.

OUT OF STOCK

- User cart



CanteenWala Home Menu PROFILE ORDERS LOGOUT

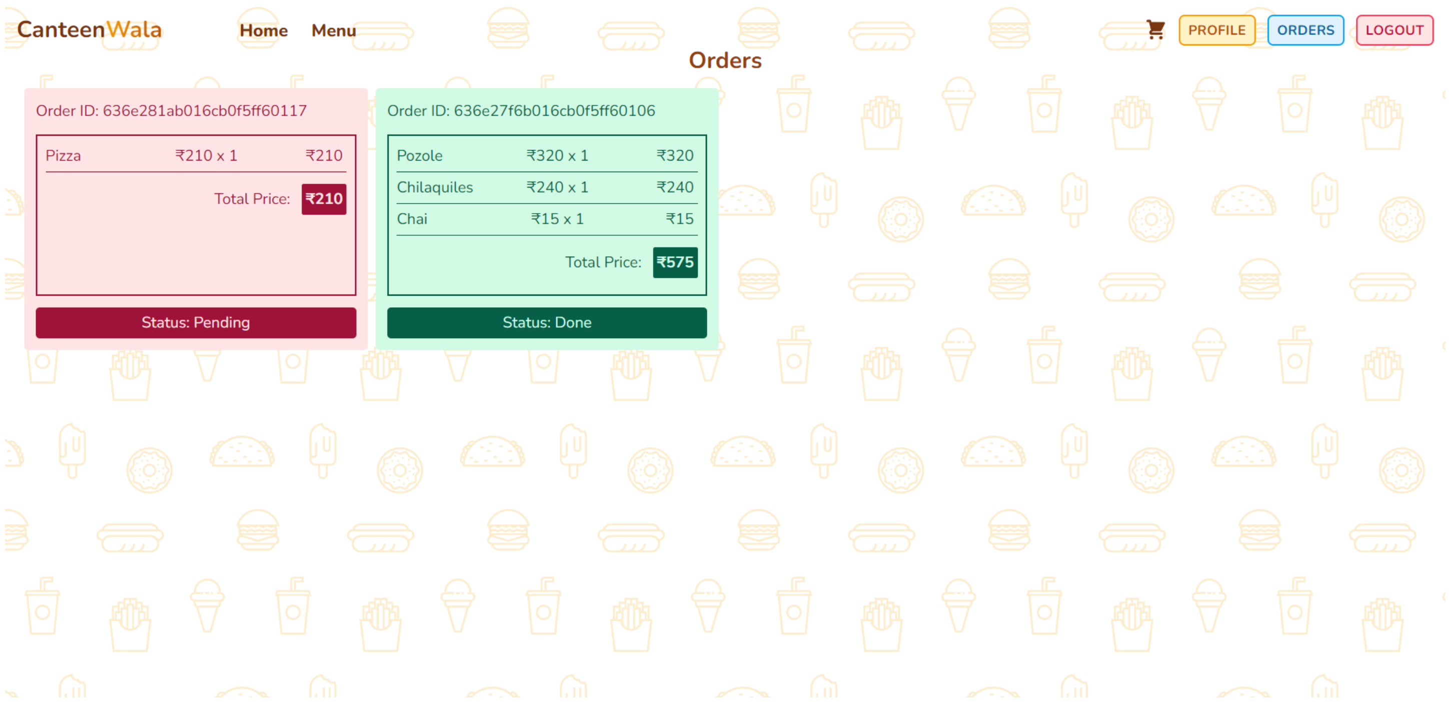
Your Cart

Pozole	₹320 x 1
Chilaquiles	₹240 x 1
Chai	₹15 x 1
Total Bill	**₹575**

CLEAR CART

Pay with UPI

- Order history of a user



Orders

Order ID: 636e281ab016cb0f5ff60117

Pizza	₹210 x 1	₹210

Total Price: ₹210

Status: Pending

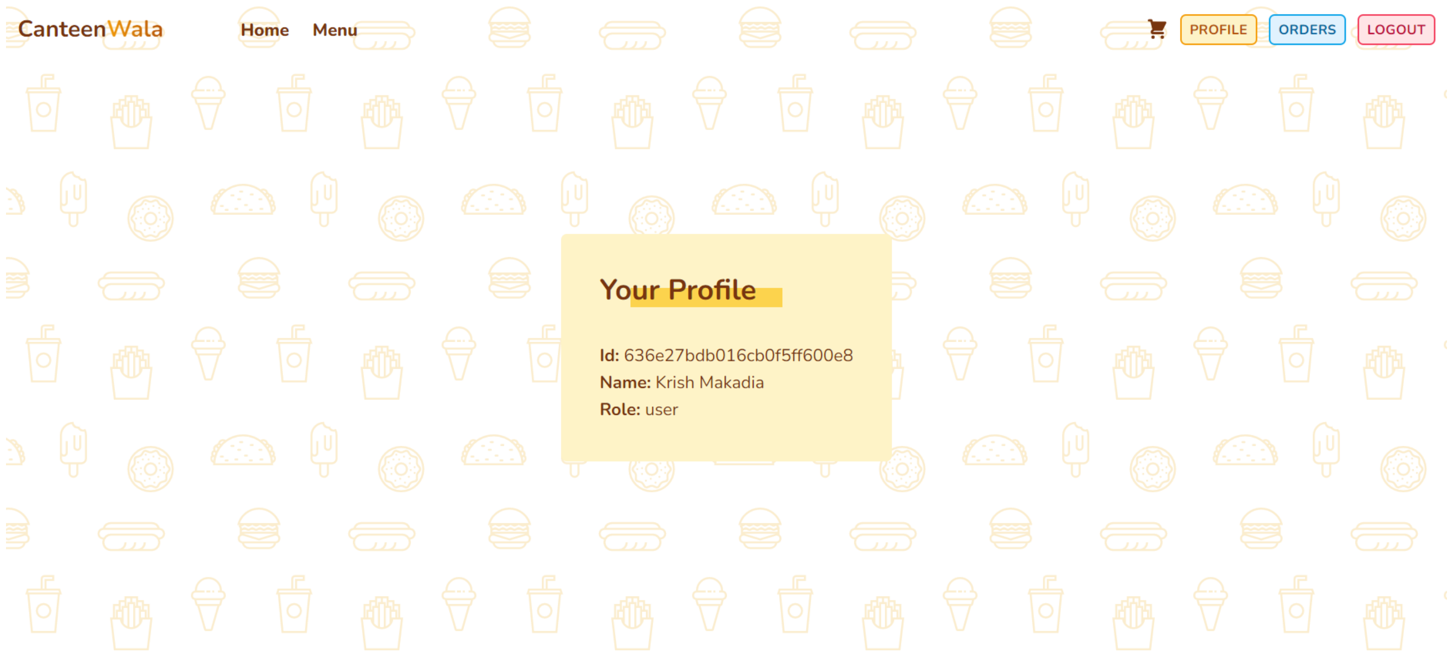
Order ID: 636e27f6b016cb0f5ff60106

Pozole	₹320 x 1	₹320
Chilaquiles	₹240 x 1	₹240
Chai	₹15 x 1	₹15

Total Price: ₹575

Status: Done

- User profile page

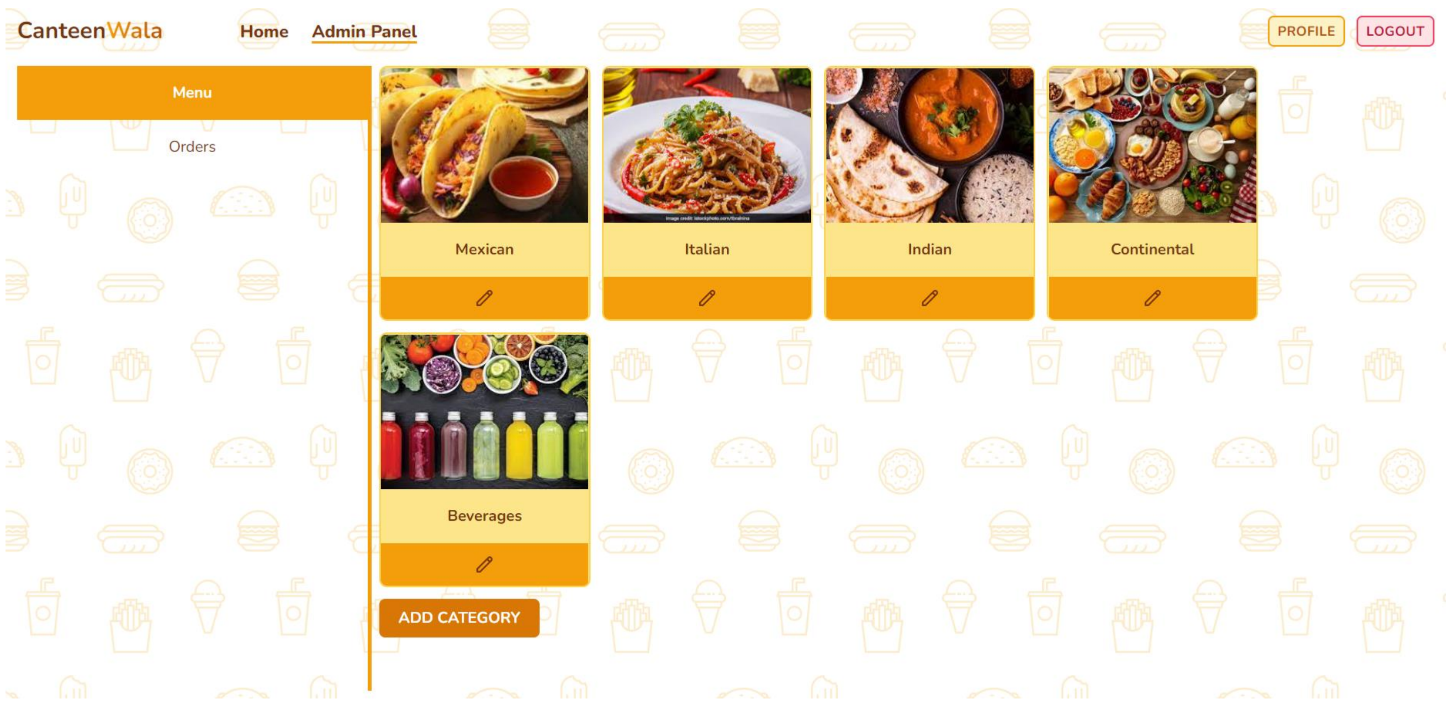


Your Profile

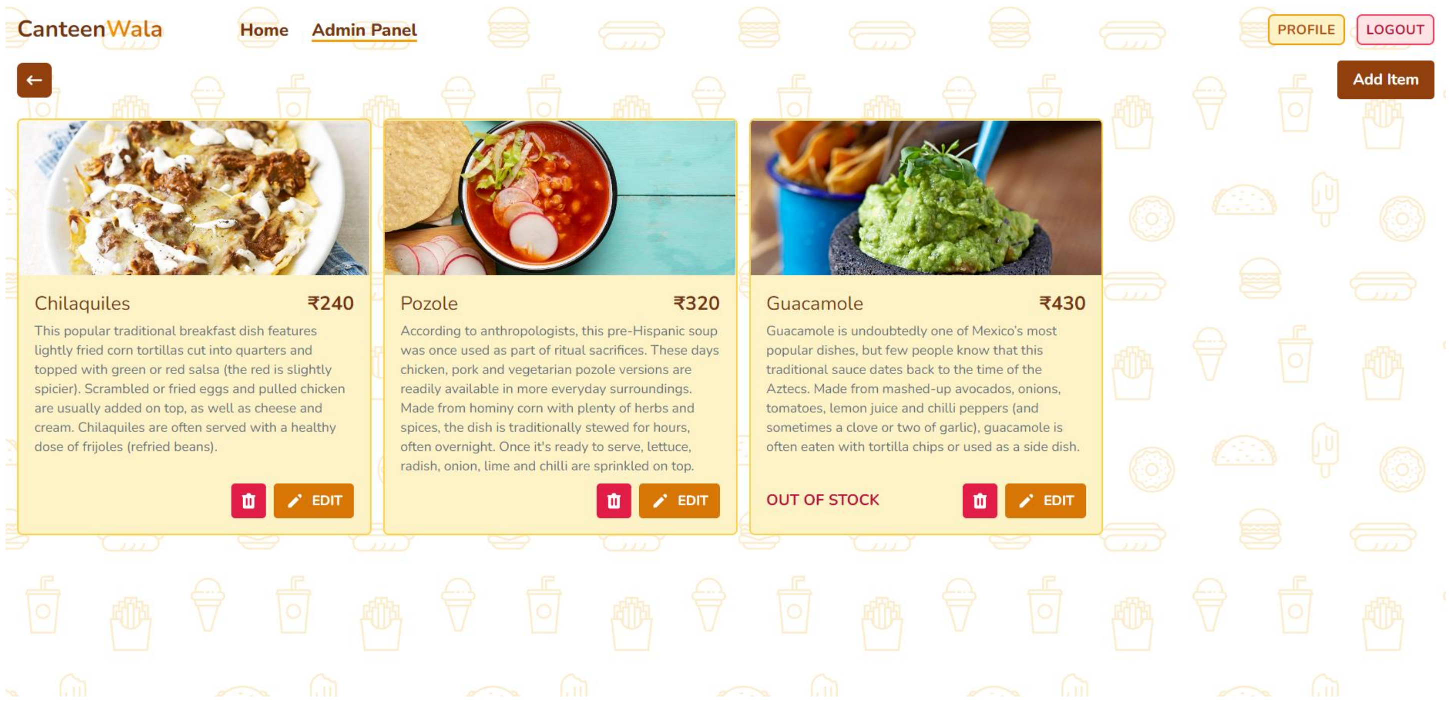
Id: 636e27bdb016cb0f5ff600e8
Name: Krish Makadia
Role: user

Admin Interface: -

- Menu view



- Food items view of a category

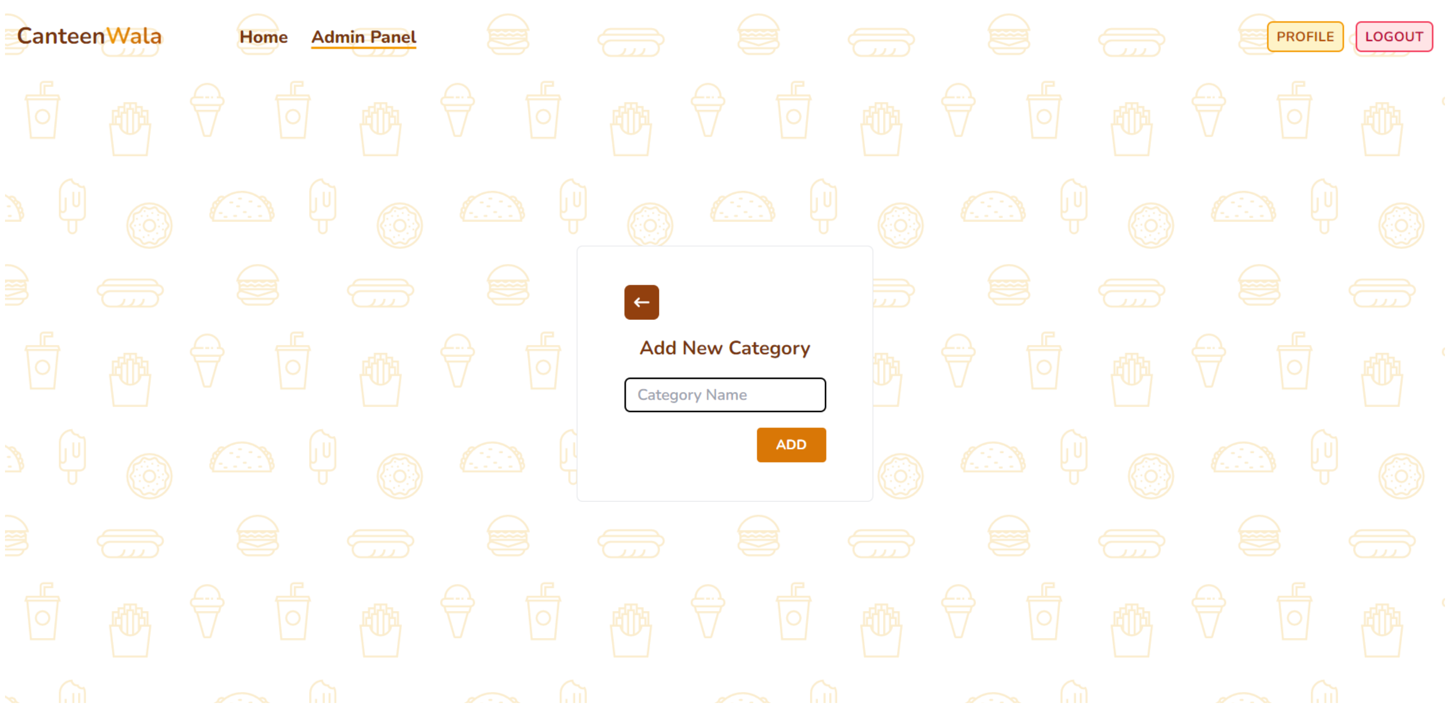


Chilaquiles ₹240
This popular traditional breakfast dish features lightly fried corn tortillas cut into quarters and topped with green or red salsa (the red is slightly spicier). Scrambled or fried eggs and pulled chicken are usually added on top, as well as cheese and cream. Chilaquiles are often served with a healthy dose of frijoles (refried beans).

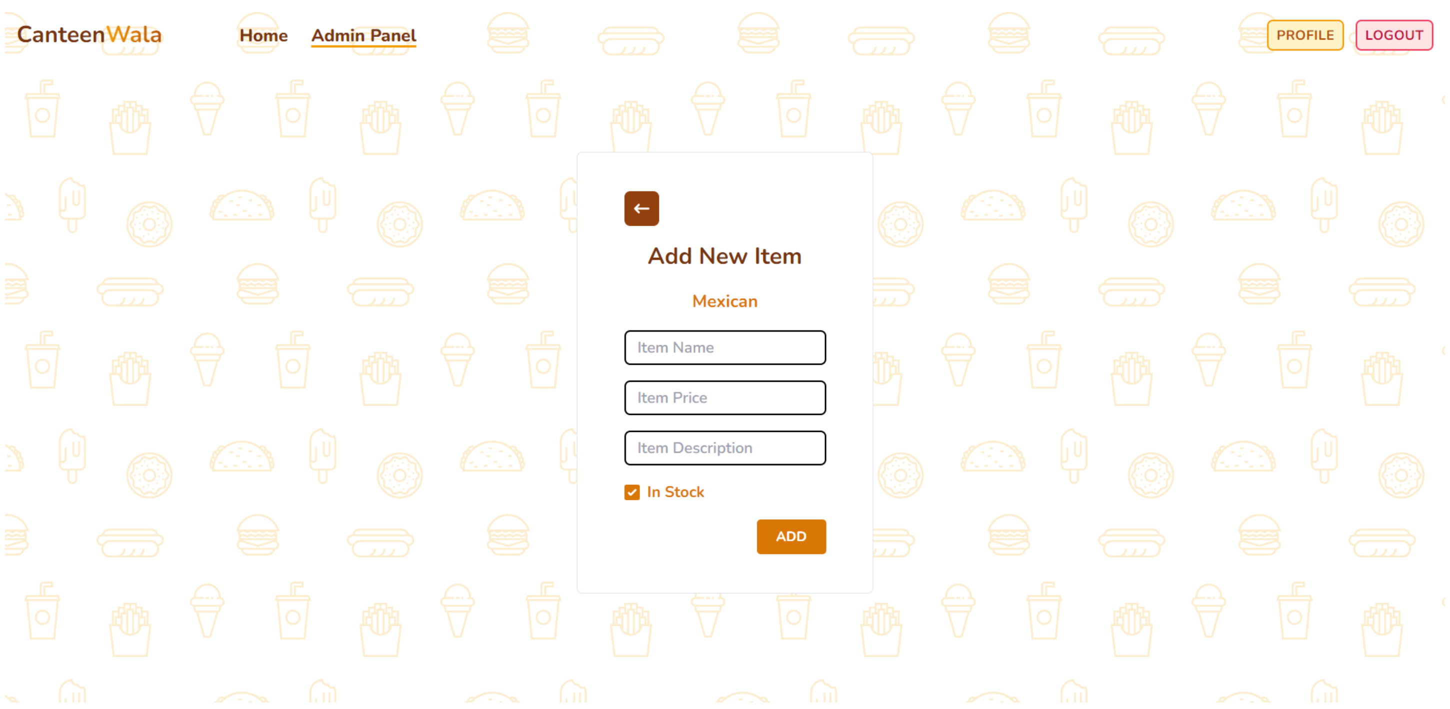
Pozole ₹320
According to anthropologists, this pre-Hispanic soup was once used as part of ritual sacrifices. These days chicken, pork and vegetarian pozole versions are readily available in more everyday surroundings. Made from hominy corn with plenty of herbs and spices, the dish is traditionally stewed for hours, often overnight. Once it's ready to serve, lettuce, radish, onion, lime and chilli are sprinkled on top.

Guacamole ₹430
Guacamole is undoubtedly one of Mexico's most popular dishes, but few people know that this traditional sauce dates back to the time of the Aztecs. Made from mashed-up avocados, onions, tomatoes, lemon juice and chilli peppers (and sometimes a clove or two of garlic), guacamole is often eaten with tortilla chips or used as a side dish.

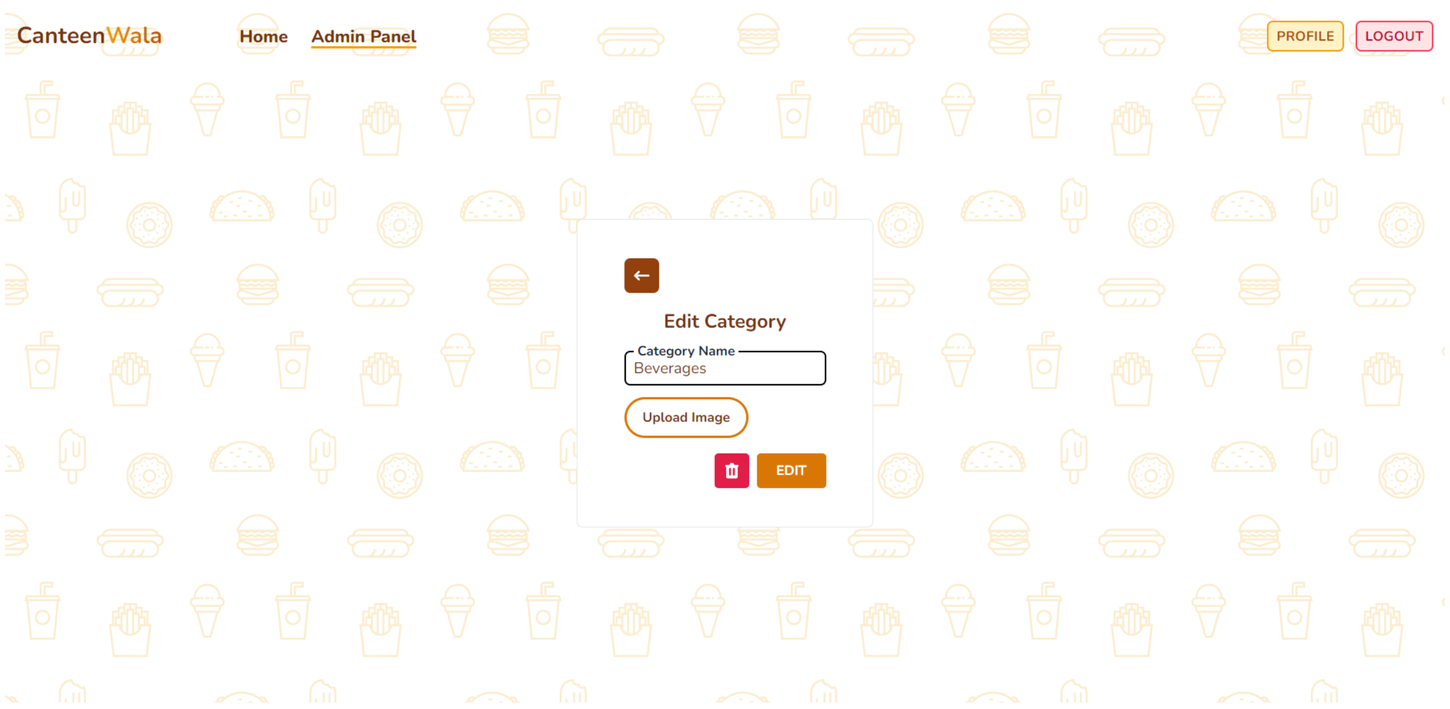
- Add new category



- Add new item

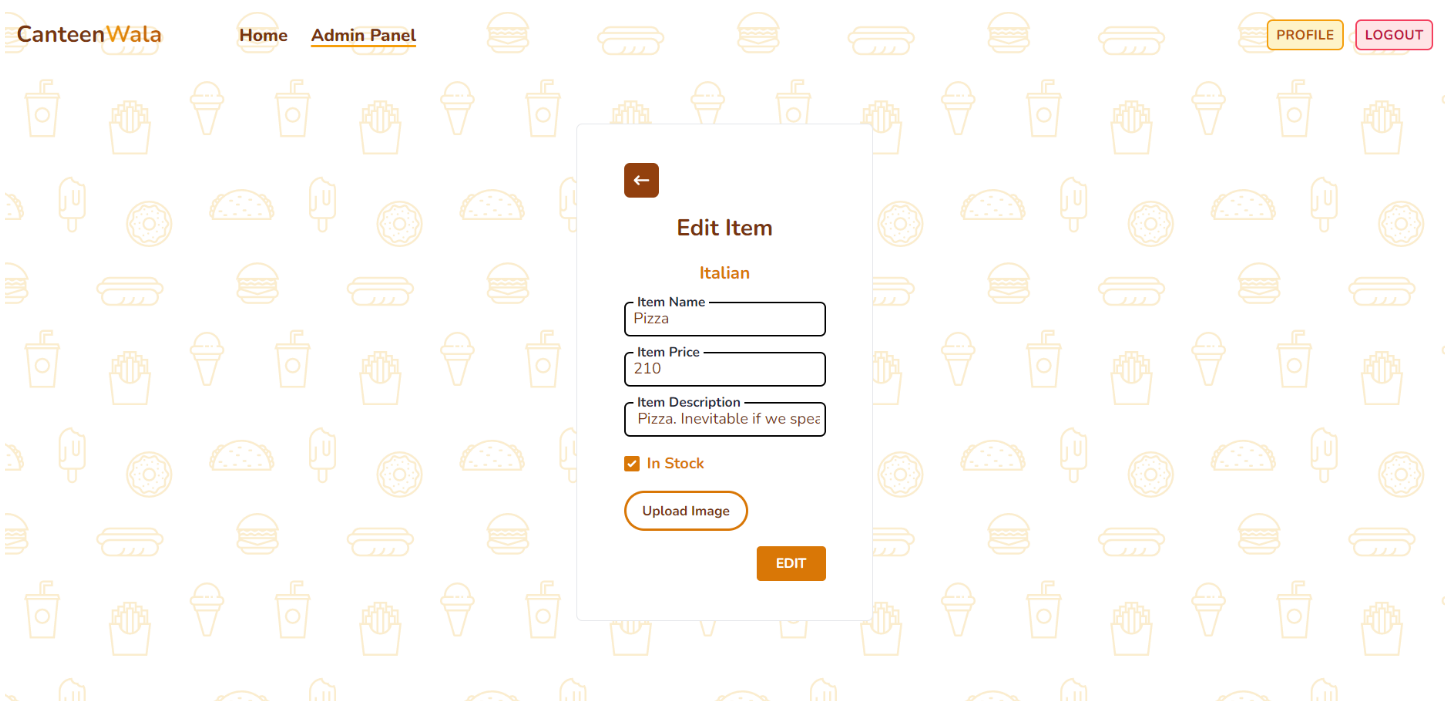


- Edit category



Category Name: Beverages

- Edit item



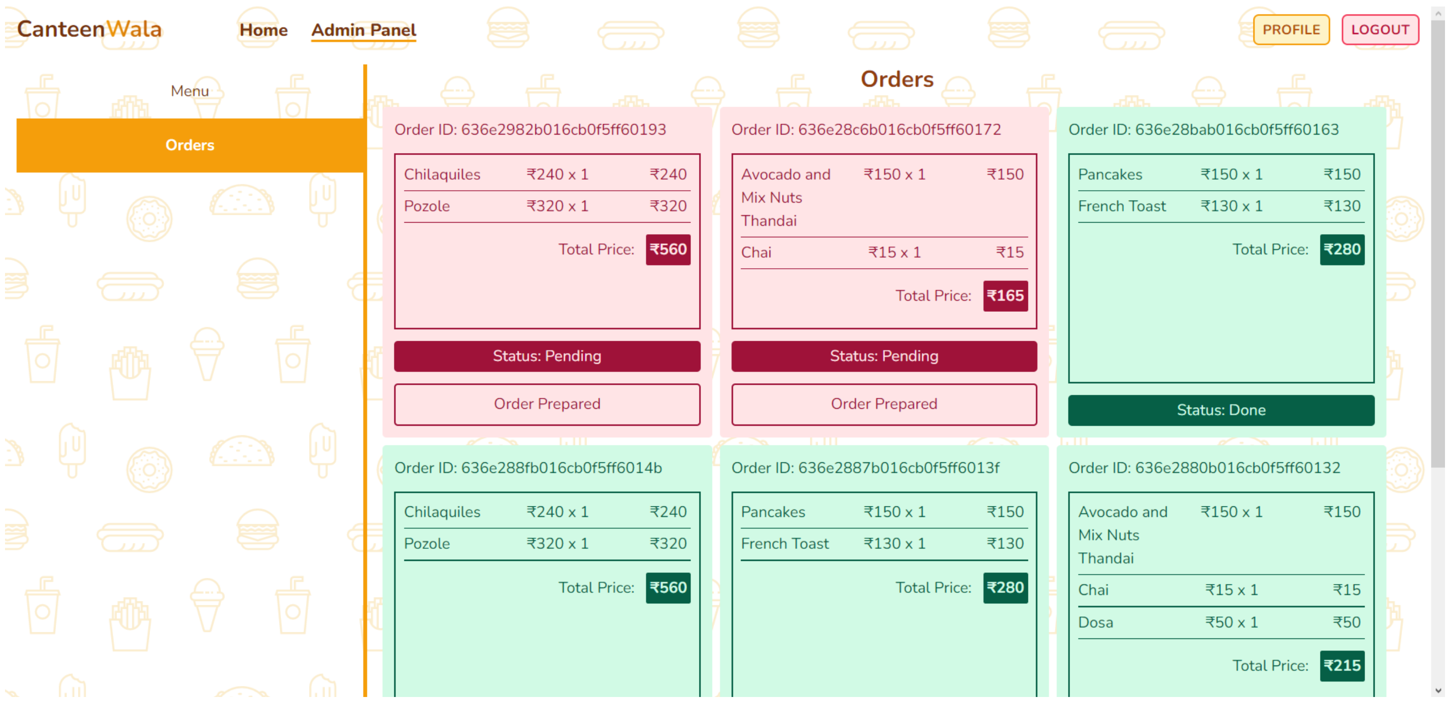
Item Name: Pizza

Item Price: 210

Item Description: Pizza. Inevitable if we spea

✓ In Stock

- Admin order view



Orders

Order ID: 636e2982b016cb0f5ff60193

Chilaquiles	₹240 x 1	₹240
Pozole	₹320 x 1	₹320

Total Price: ₹560

Status: Pending

Order Prepared

Order ID: 636e28c6b016cb0f5ff60172

Avocado and Mix Nuts Thandai	₹150 x 1	₹150
Chai	₹15 x 1	₹15

Total Price: ₹165

Status: Pending

Order Prepared

Order ID: 636e28bab016cb0f5ff60163

Pancakes	₹150 x 1	₹150
French Toast	₹130 x 1	₹130

Total Price: ₹280

Status: Done

Order ID: 636e288fb016cb0f5ff6014b

Chilaquiles	₹240 x 1	₹240
Pozole	₹320 x 1	₹320

Total Price: ₹560

Order ID: 636e2887b016cb0f5ff6013f

Pancakes	₹150 x 1	₹150
French Toast	₹130 x 1	₹130

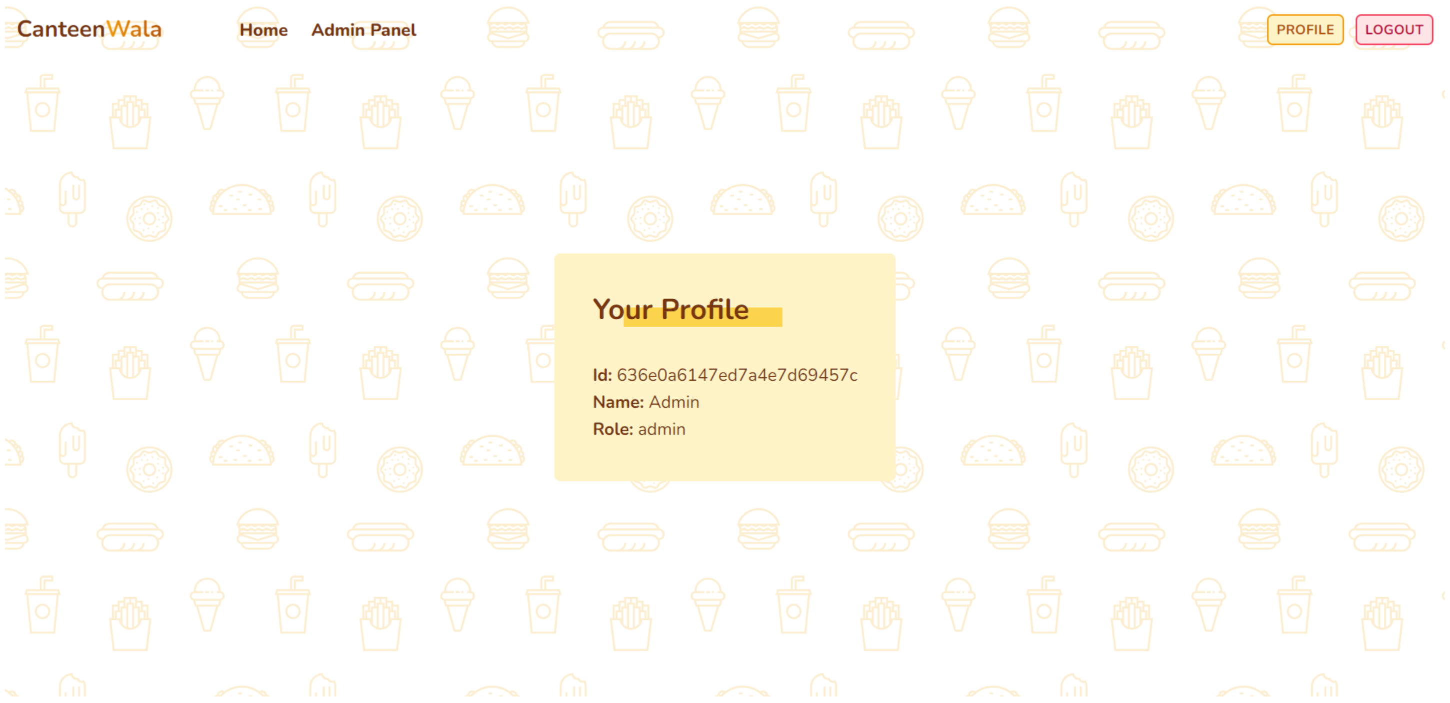
Total Price: ₹280

Order ID: 636e2880b016cb0f5ff60132

Avocado and Mix Nuts Thandai	₹150 x 1	₹150
Chai	₹15 x 1	₹15
Dosa	₹50 x 1	₹50

Total Price: ₹215

- Admin profile page



Your Profile

Id: 636e0a6147ed7a4e7d69457c
Name: Admin
Role: admin

Conclusion

The functionalities are implemented in system after understanding all the system modules according to the requirements. Functionalities that are successfully implemented in the system are:

- Customer Registration / login (with all validation)
- List of Categories
- Listing of food items corresponding to selected category
- Managing Cart
- Ordering Food
- Order history
- Customer logout
- Owner Registration / login (with all validation)
- Manage menu and categories
- Handling incoming orders
- List of all the orders
- Admin logout

Limitation and Future Enhancement

- Our system doesn’t allow the user to edit his or her profile, which will be implemented in the upcoming versions.
- The owner has no feature of accepting or rejecting the order, it is assumed that the owner will deliver every order or mark food items as out of stock. In the upcoming version, the owner will have the freedom to accept or reject incoming order.
- Customers will be allowed to rate the service and provide valuable feedback in the next versions.

Reference / Bibliography

Following links and websites were referred during the development of this project:

- Stackoverflow
- Npmjs
- React
- MDBReact
- Cloudinary
- Render