

# Contents

<b>Abstract .....</b>	<b>4</b>
<b>Introduction.....</b>	<b>5</b>
<b>About Project: .....</b>	<b>5</b>
<b>Technology: .....</b>	<b>5</b>
<b>Tools:.....</b>	<b>5</b>
<b>SOFTWARE REQUIREMENTS SPECIFICATION (SRS).....</b>	<b>6</b>
Functional Requirement : .....	6
Non Functional Requirement :- .....	9
<b>Database Design.....</b>	<b>10</b>
<b>Database Schema :.....</b>	<b>11</b>
<b>Implementation Detail.....</b>	<b>16</b>
<b>Major Functionalities: .....</b>	<b>18</b>
Authentication: .....	18
Payment : .....	19
<b>Testing.....</b>	<b>21</b>
<b>Screen-shots.....</b>	<b>22</b>
<b>Conclusion :.....</b>	<b>28</b>
<b>Limitation and Future Enhancement .....</b>	<b>28</b>
<b>Reference / Bibliography .....</b>	<b>28</b>

# **Abstract**

- Our website is an online food ordering system that enables ease for the customers. It overcomes the disadvantages of the traditional queueing system.
- Our proposed system is a medium to order online food hassle free from restaurants . This system improves the method of taking the order from customer. The online food ordering system sets up a food menu online and customers can easily place the order as per their wish. Also with a food menu.
- This system also provides a review system in which user can rate the food items and the delivery. Also, the proposed system can recommend hotels, food, based on the ratings given by the user, the hotel staff will be informed for the improvements along with the quality. The payment can be made online by upi, card, wallet.

# **Introduction**

## **About Project:**

Online food ordering system is a web-based restaurant management application with order booking functionality. It connects clients, restaurants, and stylists in an online community allowing users to browse restaurants, and food Order. User can see the overview of a restaurants. Users can also give and read reviews of restaurants given by other Users. User can also see menu photos and restaurant photos .

Our application includes Some of the major use cases user account registration, login/logout, food ordering , view Orders ,manage cart, give review of food orders at customer side and payment .

## **Technology:**

My project uses MongoDB Atlas and Express-js to backend the interface is created with React Framework as frontend. Online food ordering system integrates AWS technology as a cloud storage for Dynamic image upload, google map picker for view map , Google cloud console for Google Authentication , razorpay for payment . This project will target the major web browsers as the initial platform.

## **Tools:**

- **Visual Studio Code (editor)**
- **MongoDb compass**
- **Github**

# SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

## Online Food Ordering System

### Functional Requirement :

#### 1. User Side :

##### 1.1 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.

##### 1.1.1: Sign Up

**Input:** User details.

**Process:** Validate details.

**Output:** Redirect to home page.

##### 1.1.2: Sign In

**Input:** User details.

**Process:** Validate details.

**Output:** Redirect to Home page.

##### 1.1.3: Log Out

**Input:** User Selection.

**Output:** Redirect to Home page.

##### 1.2 Restaurant Details :

**Description :** as per user selection system shows restaurant details view .

##### 1.2.1: Overview :

**Input :** User Selection .

**Output :** Show Restaurant details .

##### 1.2.2: Food Items :

**Input :** User selection .

**Output :** List of food Items .

### **1.3 Manage Cart :**

**Description :** The Customer Can View Their Selected Food item And Can Add New food As Well As Delete The food from cart

#### **1.3.1: Add Food :**

**Input :** Select food .

**Output :** Cart Will be updated (added food in cart ) .

#### **1.3.1: Remove Food :**

**Input :** Select food .

**Output :** Cart Will be updated (Remove food )

#### **1.3.1: Place order :**

**Description :** Customer has to Select Address details To Place order

**Input :** Select Address .

**Output :** Direct to the Payment page .

### **1.3 Manage Payment :**

**Description :** The Customer Can Pay With The many Modes Of Payment like Netbanking , through Cards , Upi , wallet etc .

#### **1.3.1: Payment using Net-banking:**

**Input :** Bank details.

**Process :** Validate details

**Output :** Confirmation Message.

#### **1.3.2: Payment using Card :**

**Input :** Card-No, Expiry-Date, Card-Holder No , CVV.

**Process:** Validate details

**Output :** Confirmation Message.

#### **1.3.3: Payment using Upi :**

**Input :** Enter Upi-id.

**Process:** Validate details

**Output :** Confirmation Message.

#### **1.3.4 Payment using Wallet :**

**Input :** Select a wallet.

**Process:** Validate details

**Output :** Confirmation Message.

#### **1.3.5 Pay Later :**

**Input :** Select a method.

**Process:** Validate details

**Output :** Confirmation Message.

### **1.4. Manage Reviews :**

**Description :** Customers Can View As Well As Give The Reviews And Ratings On The Quality Of Food .

#### **1.4.1 Add Review :**

**Input :** Review And Rating

**Output :** Confirmation

#### **1.4.2 View Review :**

**Output :** View The Review .

## Non Functional Requirement :-

**N-1 Database** :- A database should be there to store the data.

**N-2 Availability** :- our system should be available at all times.

**N-3 System Independence** :- our website should be platform independent. It should work on all the platforms.

**N-4 device independence** :- our website should work on all kinds of smart devices.

**N-4 Privacy** :- Personal data of the customers should not be disclosed to anyone.

**N-5 Maintainability** :- our system should work 24x7 and should be maintained properly.

# Database Design

## Database tables :

<b>foods</b>				
<b>Storage size:</b> 20.48 kB	<b>Documents:</b> 10	<b>Avg. document size:</b> 276.00 B	<b>Indexes:</b> 1	<b>Total index size:</b> 36.86 kB
<b>images</b>				
<b>Storage size:</b> 20.48 kB	<b>Documents:</b> 14	<b>Avg. document size:</b> 369.00 B	<b>Indexes:</b> 1	<b>Total index size:</b> 36.86 kB
<b>menus</b>				
<b>Storage size:</b> 20.48 kB	<b>Documents:</b> 3	<b>Avg. document size:</b> 204.00 B	<b>Indexes:</b> 1	<b>Total index size:</b> 36.86 kB
<b>orders</b>				
<b>Storage size:</b> 20.48 kB	<b>Documents:</b> 1	<b>Avg. document size:</b> 1.58 kB	<b>Indexes:</b> 1	<b>Total index size:</b> 20.48 kB
<b>restaurants</b>				
<b>Storage size:</b> 20.48 kB	<b>Documents:</b> 5	<b>Avg. document size:</b> 559.00 B	<b>Indexes:</b> 1	<b>Total index size:</b> 36.86 kB
<b>reviews</b>				
<b>Storage size:</b> 20.48 kB	<b>Documents:</b> 3	<b>Avg. document size:</b> 172.00 B	<b>Indexes:</b> 1	<b>Total index size:</b> 36.86 kB
<b>users</b>				
<b>Storage size:</b> 20.48 kB	<b>Documents:</b> 4	<b>Avg. document size:</b> 181.00 B	<b>Indexes:</b> 1	<b>Total index size:</b> 36.86 kB



## Database Schema :

### User:

```
const UserSchema = new mongoose.Schema(  
  {  
    fullName: { type: String, required: true },  
    email: { type: String, required: true },  
    password: { type: String, required: true },  
    address: [{ details: { type: String }, for: { type: String } }],  
    phoneNumber: [{ type: Number }],  
  },  
  {  
    timestamps: true,  
  })
```

## Food:

```
const FoodSchema = new mongoose.Schema(  
  {  
    name: { type: String, required: true },  
    descript: { type: String, required: true },  
    isVeg: { type: Boolean, required: true },  
    isContainsEgg: { type: Boolean, required: true },  
    category: { type: String, required: true },  
    photo: {  
      type: mongoose.Types.ObjectId,  
      ref: "images",  
    },  
    prices: { type: Number, required: true },  
    addOnes: [{  
      type: mongoose.Types.ObjectId,  
      ref: "foods",  
    }],  
    restaurant: {  
      type: mongoose.Types.ObjectId,  
      ref: "restaurants",  
      required: true,  
    },  
  },  
  {  
    timestamps: true,  
  });
```

## Image:

```
const ImageSchema = new mongoose.Schema(  
  {  
    images: [  
      {  
        location: { type: String, required: true },  
      },  
    ],  
  },  
  {  
    timestamps: true,  
  },  
);
```

## Restaurant:

```

✓ const RestaurantSchema = new mongoose.Schema({
  name: { type: String, required: true },
  city: { type: String, required: true },
  address: { type: String, required: true },
  maplocation: { type: String, required: true },
  cuisine: [String],
  resturentTimings: String,
  contactNumber: Number,
  website: Number,
  popularDishes: [String],
  avrageCost: Number,
  amenties: [String],      //amenties = facilities
  menuImages: {
    type: mongoose.Types.ObjectId,
    ref: "images",
  },
  menu: {
    type: mongoose.Types.ObjectId,
    ref: "menus",
  },
  reviews: [{
    type: mongoose.Types.ObjectId,
    ref: "reviews",
  }],
  photos: { type: mongoose.Types.ObjectId, ref: "images" },
}, {
  timestamps: true,
})

✓ RestaurantSchema.methods.get_id = function () {
  return this._id.toString();
}

```

## Menu:

```
const MenuSchema = new mongoose.Schema(  
  {  
    menus: [  
      {  
        name: { type: String, required: true },  
        items: [{  
          type: mongoose.Types.ObjectId,  
          ref: "foods",  
        }]  
      },  
    ],  
    recomanded: [{  
      type: mongoose.Types.ObjectId,  
      ref: "foods"  
    }]  
  }, {  
    timestamps: true,  
  }  
)
```

## Order:

```

const OrderSchema = new mongoose.Schema({
  user: {
    type: mongoose.Types.ObjectId,
    ref: "users",
  },
  orderDetails: [{
    food: [{ details: { type: mongoose.Types.ObjectId, ref: "foods" }, quantity: { type: Number, required: true }, }],
    paymode: { type: String, required: true },
    status: { type: String, default: "placed" },
    paymentDetails: {
      totalItem: { type: Number, required: true },
      promo: { type: Number, required: true },
      tax: { type: Number, required: true },
      razorpay_id: { type: String, required: true },
    }
  }]
}, {
  timestamps: true,
})

```

## Review:

```

const ReviewSchema = new mongoose.Schema(
  {
    food: { type: mongoose.Types.ObjectId, ref: "foods" },
    restaurant: { type: mongoose.Types.ObjectId, ref: "restaurants" },
    user: { type: mongoose.Types.ObjectId, ref: "users" },
    rating: { type: Number, required: true },
    reviewText: { type: String, required: true },
    isResturantReview: Boolean,
    isFoodReview: Boolean,
    photos: {
      type: mongoose.Types.ObjectId,
      ref: "images"
    }
  },
  {
    timestamps: true,
  }
)

```

## Implementation Detail

### ➤ **Modules created and brief description of each module**

This Project consists of 3 major modules.

- 1) User Module
- 2) Cart Module
- 3) Review and Rating Module
- 4) Login module

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

#### **1) User Module**

There is one type of User: Customer. This module is the base for authentication and authorization to ensure the security aspect of the user.

#### **2) Cart Module**

In this module customer can add or delete food items. Customer have to according the chosen item's prize and quantity.

### **3) Review and Rating Module**

In this module customer can give review and rating to their delivered orders which is also reflected in restaurant's overview review.

### **4) Login Module**

Login Module: The current state of the user must be persisted in the system. When the registered user sign-in into the system, its user profile and cart status is retrieved from the database and all the activities performed by the user will be stored in the system unless the user logs out from the system. User is also allowed to change its credentials and personal information and the changes will immediately reflect in the output.

# Major Functionalities:

## Authentication:

User must be authenticated before continuing with Website or the payment. This is performed by using the JWT (JSON Web Tokens). Each user is given a unique token after his successful registration. That token gets verified each time the user proceeds with the payment. This helps to fulfil the safety requirement of the system.

- For Generating token :

```
UserSchema.methods.generateJsonWebTokens = function () {  
  return jwt.sign({ user: this._id.toString() }, process.env.SECRETORKEY);  
}
```

- For salting a password :

```
UserSchema.pre("save", function (next) {  
  const user = this;  
  
  // password is modified  
  if (!user.isModified("password")) return next();  
  
  // generate bcrypt salt  
  bcrypt.genSalt(8, (error, salt) => {  
    if (error) return next(error);  
  
    // hash the password  
    bcrypt.hash(user.password, salt, (error, hash) => {  
      if (error) return next(error);  
  
      // assigning hashed password  
      user.password = hash;  
      return next();  
    });  
  });  
});
```

- Route for Sign-in, Sign-up, Google-Authentication



```

Router.post("/signup", async (req, res) => {
  try {
    await SignupValidation(req.body.credentials);
    await UserModel.findByEmailAndPhone(req.body.credentials);
    const newUser = await UserModel.create(req.body.credentials);
    const tokens = newUser.generateJsonWebTokens();
    return res.status(200).json({ tokens, status: "success" });
  } catch (error) {
    return res.status(500).json({ error: error.message });
  }
})

Router.post("/signin", async (req, res) => {
  try {
    await SigninValidation(req.body.credentials);
    const user = await UserModel.findByEmailAndPassword(req.body.credentials);
    const tokens = user.generateJsonWebTokens();
    return res.status(200).json({ tokens, status: "success" });
  } catch (error) {
    return res.status(500).json({ error: error.message });
  }
})

Router.get('/google', passport.authenticate('google', {
  scope: [
    "https://www.googleapis.com/auth/userinfo.profile",
    "https://www.googleapis.com/auth/userinfo.email",
  ]
}))

Router.get('/google/callback', passport.authenticate('google', { failureRedirect: "/" }), (req, res) => {
  return res.redirect(
    `http://localhost:3000/google/${req.session.passport.user.token}`
  );
})

```

Payment :

For payment We use razorpay . In rezorpay we use checkout.js library for the payment .

- In ./client/public/index.html

```

<body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div>      You, 6 days ago • new created
  <script src="https://checkout.razorpay.com/v1/checkout.js"></script>
</body>

```

- In checkout page

```
const payNow = () => {
  let options = {
    key: "rzp_test_ngl3xispaI3qN8",
    amount:
      | cart.reduce((total, current) => total + parseInt(current.totalPrice), 0) * 100,
    currency: "INR",
    name: "Zomato Master",
    description: "Fast Delivery Service",
    handler: (data) => {
      alert("Payment Successful");
      console.log(data);
    },
    prefill: {
      name: user.name,
      email: user.email,
    },
    theme: {
      color: "#e23744",
    },
  };

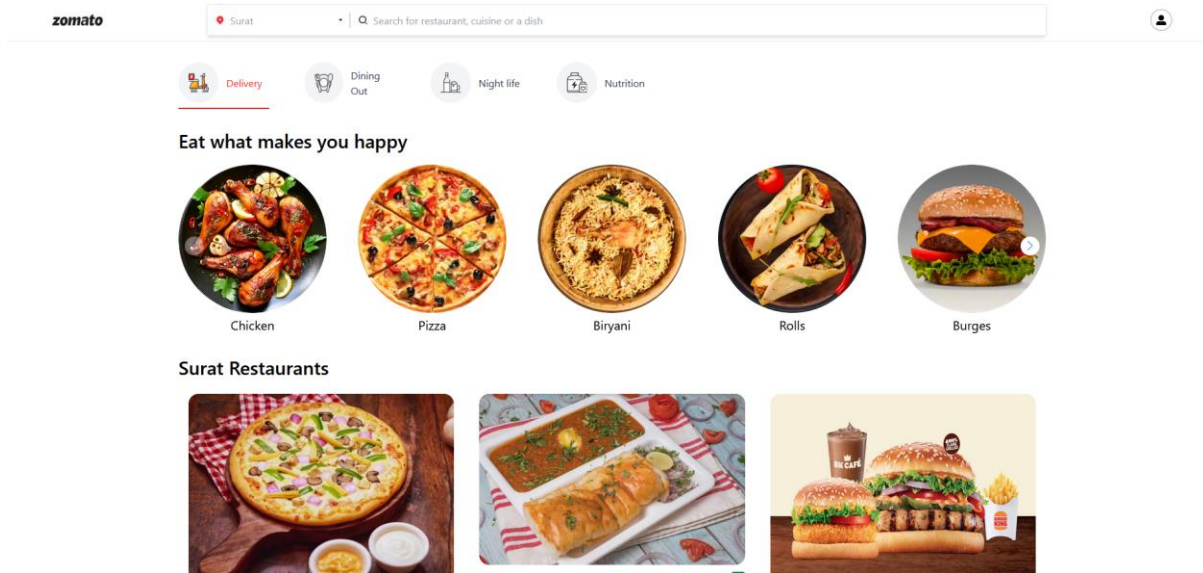
  let razorpay = new window.Razorpay(options);
  razorpay.open();
};
```

# Testing

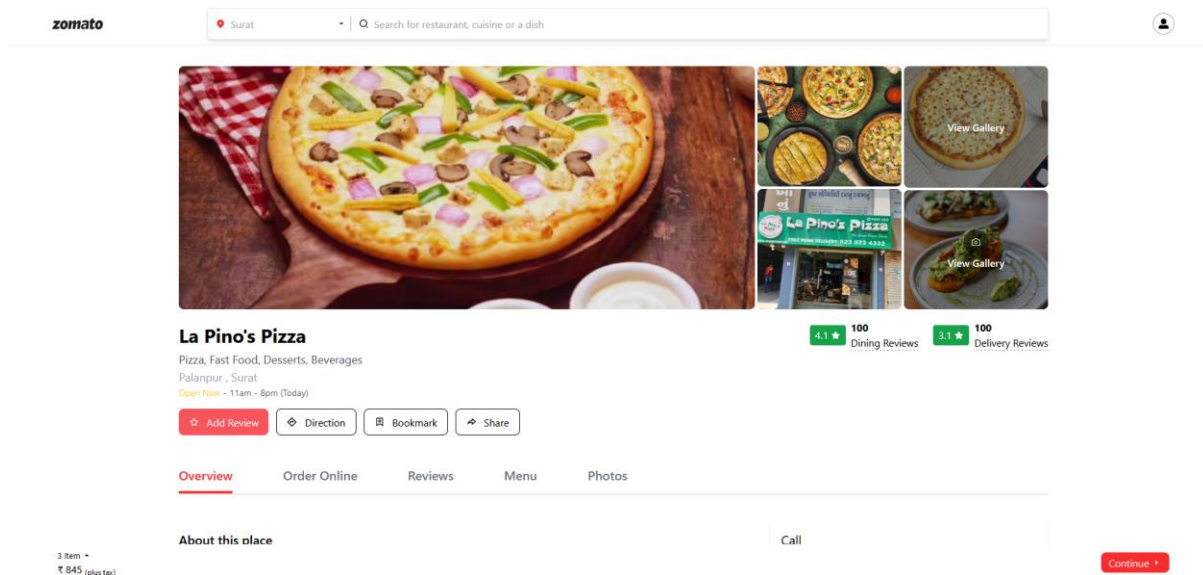
Test Case Objective	Input Data	Expected Output	Actual Output	Status
<b>Authentication of User</b>	credentials	Success or error message	Success or error message	Pass
<b>Manage Cart</b>	Selection of Food	Details of Selected Food items	Details of Selected Food items	Pass
<b>Restaurant Details</b>	Selection of restaurant	Details of Selected Restaurant	Details of Selected Restaurant	Pass
<b>List of Food</b>	Selection of restaurant	Details of all available foods	Details of all available foods	Pass
<b>Add reviews</b>	reviewData	Success or error message	Success or error message	Pass
<b>Manage reviews</b>	Selection of restaurant	List of reviews of selected restaurant	List of reviews of selected restaurant	Pass
<b>Manage Photos</b>	Selection of restaurant	All photos of selected restaurant and their menu photo	All photos of selected restaurant and their menu photo	Pass
<b>Manage payment</b>	Selection of different types of payment methods	Confirmation message (after completing payment)	Confirmation message (after completing payment)	Pass

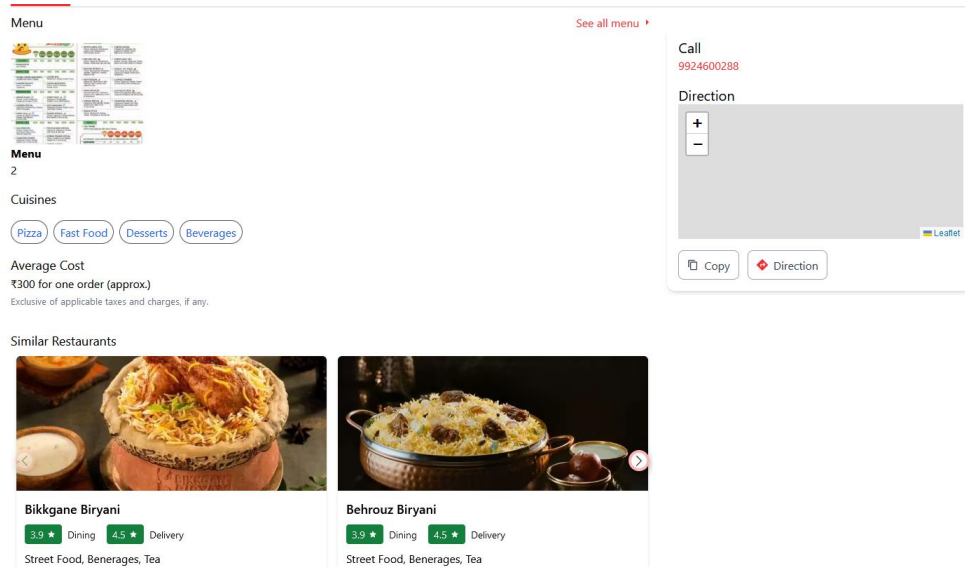
# Screen-shots

## Home-screen :

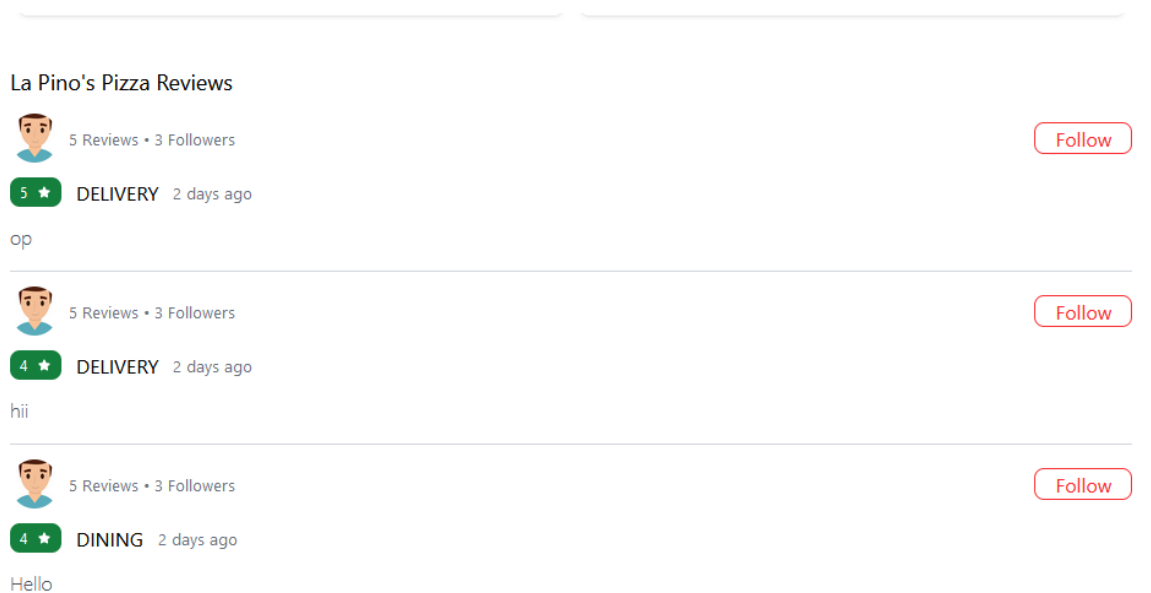


## Restaurant-screen :

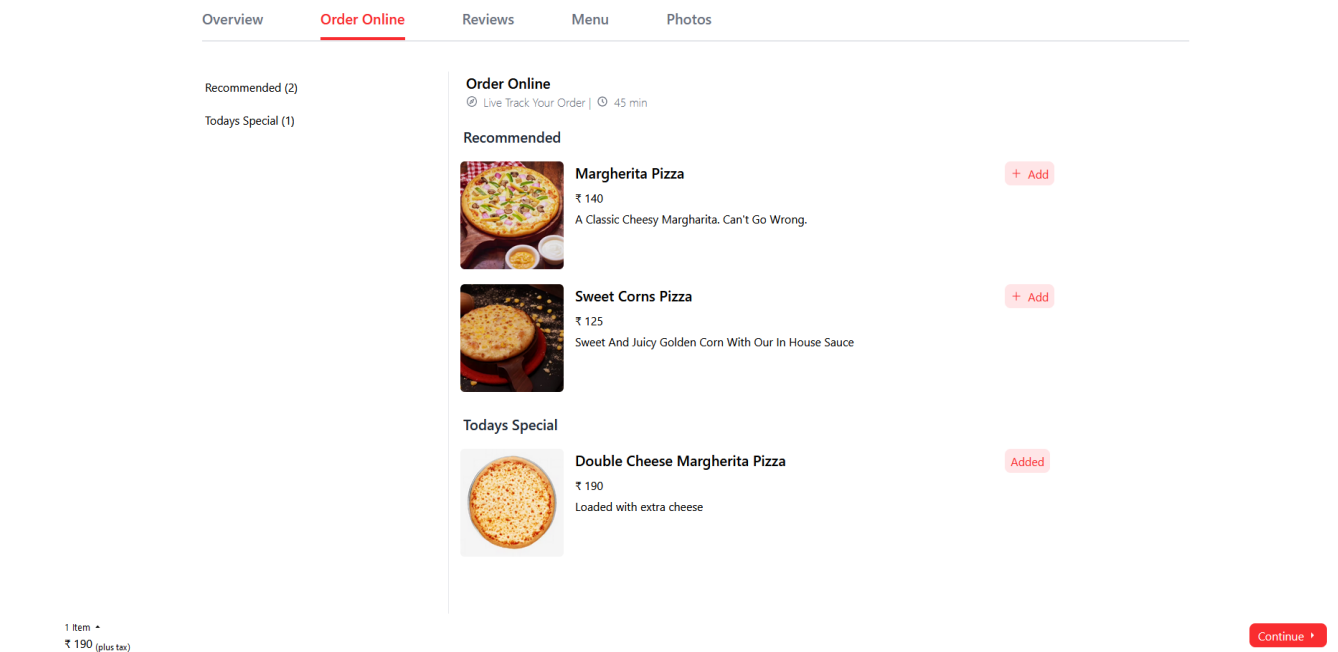




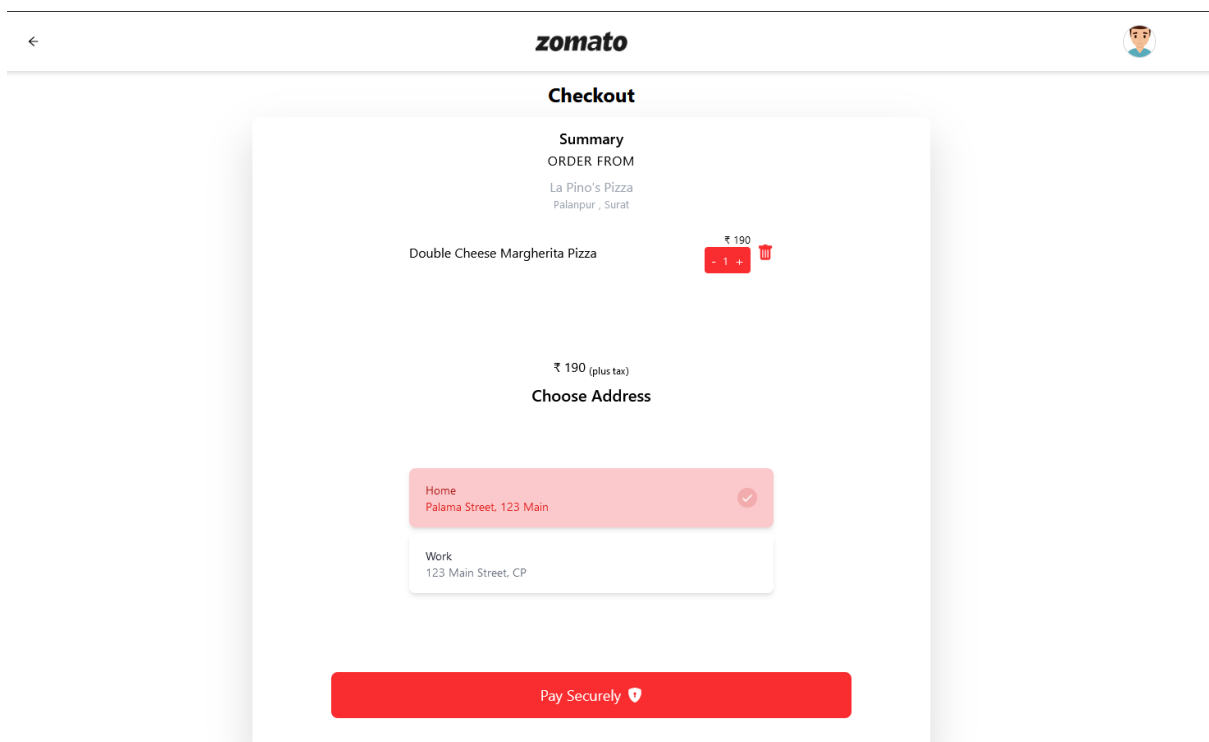
## Review screen :



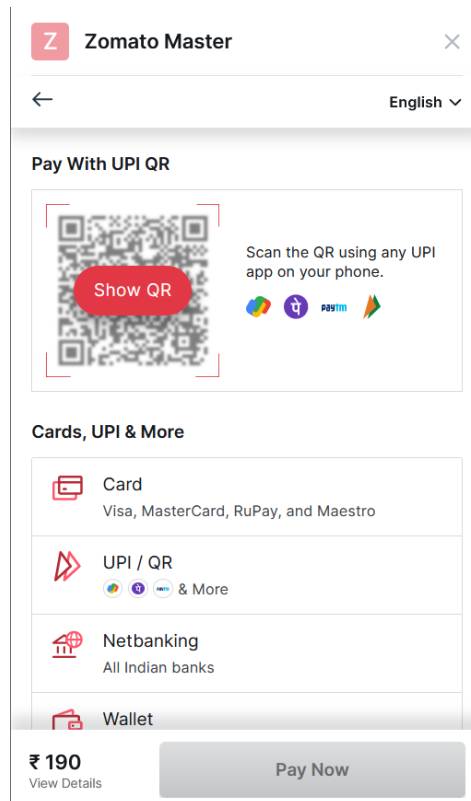
## Order screen :



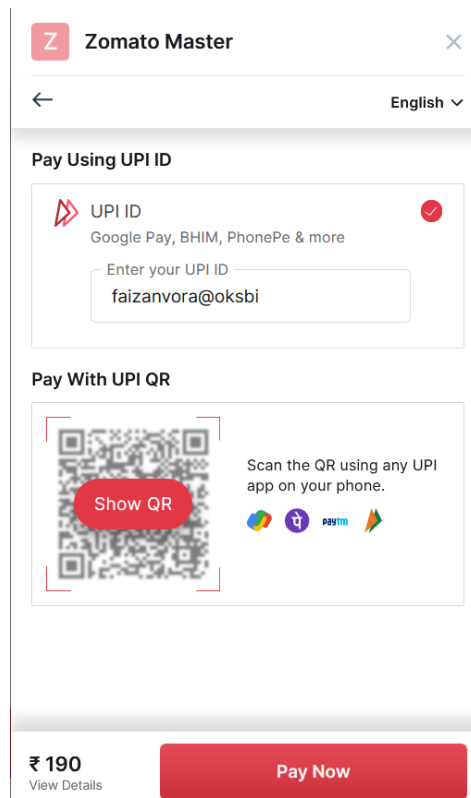
## Cart screen :




## Payment screen :



UPI payment screen :



### Login page:

Sign In With Google 

Email


user@email.com

Password

\*\*\*\*\*

Sign In

### Sign-up page :

Sign Up With Google 

Full Name

John Doe

Email

user@email.com

Password

\*\*\*\*\*

Sign Up

### Add review screen :



### Add Review

☐ Dining ☒ Delivery



Subject

Adding a Review

Comment

The food was amazing and the delivery was quick too.

Add

## 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)
- Order food
- Rate and Review order
- Manage cart

## Limitation and Future Enhancement

- The application does not have a variety of products. It is limited to fixed products in stock. The handling of this problem can be done by adding an admin module which is not yet implemented.
- The application is missing some important features like user interaction etc.
- We can turn static section to dynamic section .
- Live delivery tracking.
- User information page with update and delete facility.
- Implementation of Feedback section can increase the interaction between user and website.
- We can add Billing Page .

## Reference / Bibliography

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

- NodeJS <https://nodejs.org/en/docs/guides/>
- Express.js <https://expressjs.com/en/guide/routing.html>
- MongoDB <https://www.mongodb.com/docs/manual/introduction/>
- React <https://reactjs.org/docs/getting-started.html>
- MERN Stack <https://www.mongodb.com/mern-stack>

➤ Stackoverflow <https://stackoverflow.com/>