Food Ordering Website

Deepika Saiprabha Chilukuri
DeepikaSaiprabha_Chilukuri@student.uml.edu

Introduction

The proposed food-ordering website is a web-based application that enables users to view available foods, modify their orders, and complete secure payments. The basic goals of the website design are achieving high usability and security for users and proper compliance with current web tendencies. It will use a RESTful API consistent across all major browsers while supporting HTTPS connections using Localtunnel for development.

The website is expected to be customer-oriented, but it also aims to provide logging, auditing, and serious safety requirements in the backend for data storage and operations.

Key Features

This food ordering website is intended to serve as a complete food ordering solution where users can:

1. See what meals are being offered and information about the meals such as the ingredients used, the price, and the number of calories in the meal.
2. They should place their orders with the desired quantity and add a memo for special instructions (for example, “No onions” may be included in their order).
3. The particular user should then fill in relevant delivery details such as his/her name, address, and phone number.
4. Payments are made in full utilizing a secure online payment gateway that is linked through REST APIs.

5. It would be better if they could audit the order history in the future by creating a user account.
6. You can be sure their data is processed securely through items including HTTPS, logging, and auditing.

Why This Solution?

The usage of ordering foods online has recently gained a lot of popularity with the enhanced usage of the internet as well as the high number of people who prefer online services. Most present solutions are costly and need collaboration with large delivery channels or networks. This website seeks to:

1. Create an option for small or medium restaurants that can integrate online ordering at a fairly low cost.
2. Improve the ease of ordering, the way & processes of customizing food as well as securely & conveniently paying online.
3. Integration possibilities with future delivery services must be offered with a view to improving the scalability of the platform.

Supported Features

1. **Food Catalog**: A list that changes with time when the various food items available are issued. Every item will have brief description, price, composition, and photo.
2. **Order Customization**: Customers will be allowed to enter additional unique notes such as about diet information and also personalize the quantity of the product on the order.
3. **User Input**: The clients are allowed to enter their delivery address and other means of contacting them.
4. **Secure Payment Gateway**: Continuation from the previous point: Connection to a payment gateway (e. g. Stripe or PayPal) for handling of payments.

5. **Cross-Browser Support**: It works properly with all the primary web browsers including Chrome, Firefox, and MS Edge while responding to the responsive design concepts.
6. **HTTPS Support:** Securing of the transaction procedures and data sharing through HTTPS with the use of Localtunnel for development and the use of Standard SSL in the final stage.
7. **REST API Support:** The back end will present Restful API to incite interaction with the database; basic food items, orders, payment, etc.
8. **Auditing and Logging:** This means that whatever the system does, right from the API calls to the occurrence of errors will at times also be done with different velocities of detail information (info, debug, error, and trace).
9. **Authentication**: Optional customer profiles will enable the reoccurring customer to input and save their payment options as well as history.

Not Supported Features

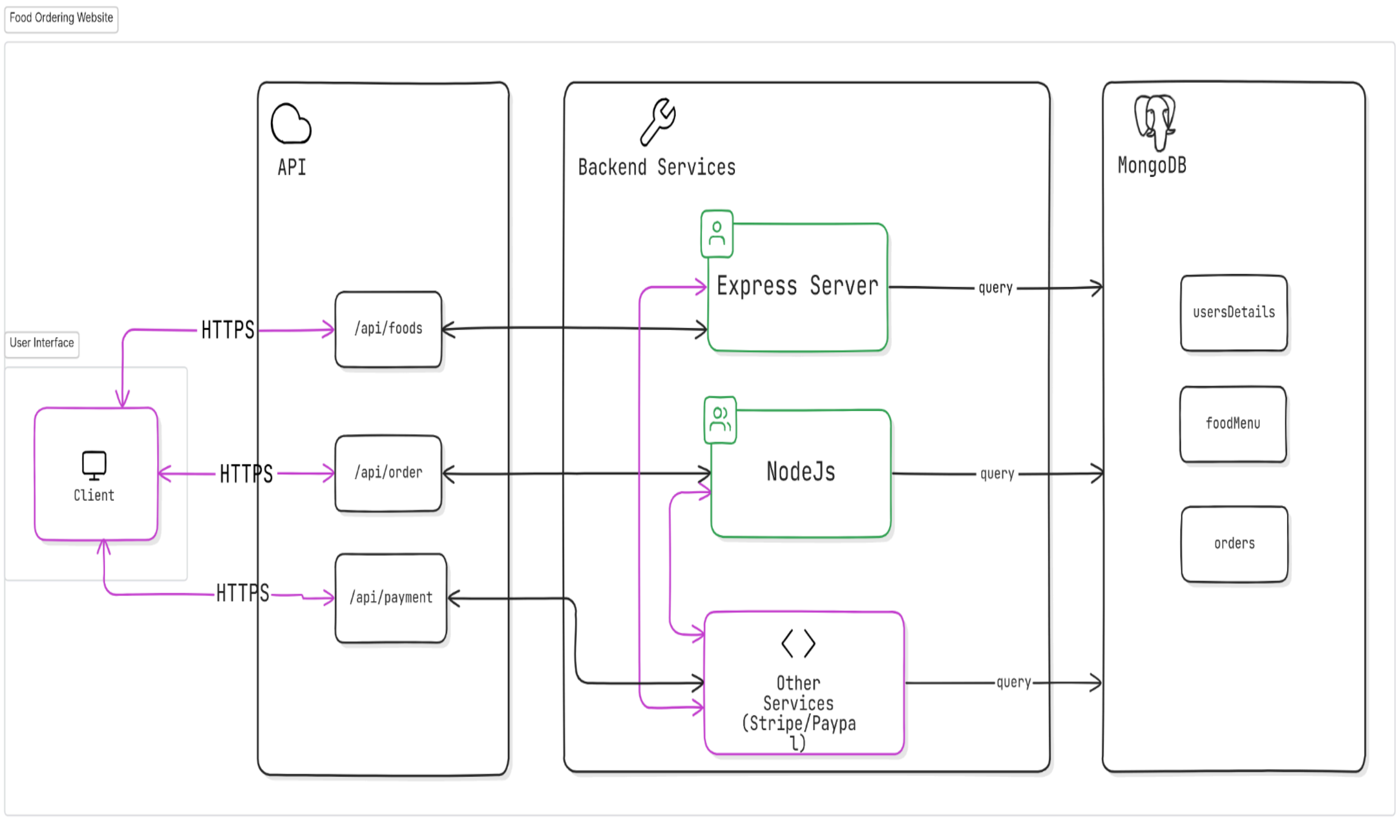
1. **Real-Time Delivery Tracking**: Real-time order delivery tracking with the help of third-party delivery services will not be available in this version.
2. **Offline Mode**: Offline usage will not be possible on the website so the users will require an active internet connection for placing the orders.
3. **Multiple Currencies**: Payments will only be processed in one default currency for simplicity to avoid complications.
4. **Multi-Language Support:** During the first phase, the interface will be given in only English but will be translated into other languages in subsequent phases.

Future Planned Features

1. **User Authentication & Account System**: It allow users to create an account, log in, view previous orders, and even save addresses for easy order repeats.
2. **Order History & Reorder:** Customers will be allowed to perform product and order reviews and repeat their previous orders with a single click.
3. **Loyalty Program & Discounts**: A points-based system, in which clients accumulate points that can be exchanged into particular discounts.
4. **Push Notifications/Email Integration**: To inform the customer once they have placed their order when the order has been processed, and finally when the order has been shipped, integration with services like Twilio and SendGrid.
5. **Integration with Delivery Services**: Efficient delivery tracking through integration with other delivery platforms such as UberEats

How to achieve the Website

Architecture Diagram



Components and Modules

1. Frontend (React. js):

Components: Navigation bar, Food catalog page, Cart, Checkout page, Payment form, and Order confirmation.

State Management: Application of the useState and useEffect hooks for React to deal with dynamic values (food items, order status).

Form Validation: Inclusive of form validation, for instance, user input data such as details and credit card information using frameworks such as Formik or native HTML5 form validation.

2. Backend API (Node. js with Express)

Order Controller: Responsible for order management – creation, cancellation, and even the retrieval of the orders.

Payment Controller: Responsible for the communication protocols between the system and the payment gateway.

Authentication (Optional): JWT to be used for token-based authentication for future features.

Audit Service: Records some key system events (order creation, payment success/failure, etc.) which later help in auditing as well as security purposes.

3. Database (MongoDB):

Food Items Collection: Food information such as name, ingredients, price, and availability are stored here.

Orders Collection: Records user orders including items that are purchased, the quantities, remarks, and delivery preferences.

User Collection (Optional): In case there will be user accounts in the future where users' data will be stored on the site.

Audit Logs Collection: Collects and stores system-level events for audit trail functions.

4. Logging & Auditing:

Winston/Bunyan: Third-party libraries that hosted the more structured logs.

Log Levels: Different log levels will be created by the system including info, debug, error and trace.

Languages for the modules

1. **Frontend**: JavaScript (React.js), HTML5, CSS3.
2. **Backend**: JavaScript (Node.js with Express).
3. **Database**: MongoDB (NoSQL).
4. **Security/Authentication**: JWT (JSON Web Tokens).

3rd Party/Open-source modules

1. **Express.js (Backend Framework)**: Node.js framework for building the backend RESTful API.
2. **Mongoose (ODM for MongoDB)**: Handles schema definitions and interactions with MongoDB.
3. **Localtunnel**: For exposing the local server with an HTTPS endpoint during development.
4. **Stripe/PayPal SDK**: To handle online payments securely.
5. **Winston/Bunyan (Logging)**: Open-source logging library for generating structured logs.

3rd Party Services/APIs

1. **Localtunnel (Free)**: Used during development to create HTTPS endpoints for local development servers.
2. **Stripe (Paid/Free Tier)**: Secure payment processing service.

3. **SendGrid (Free Tier)**: To send order confirmation emails.

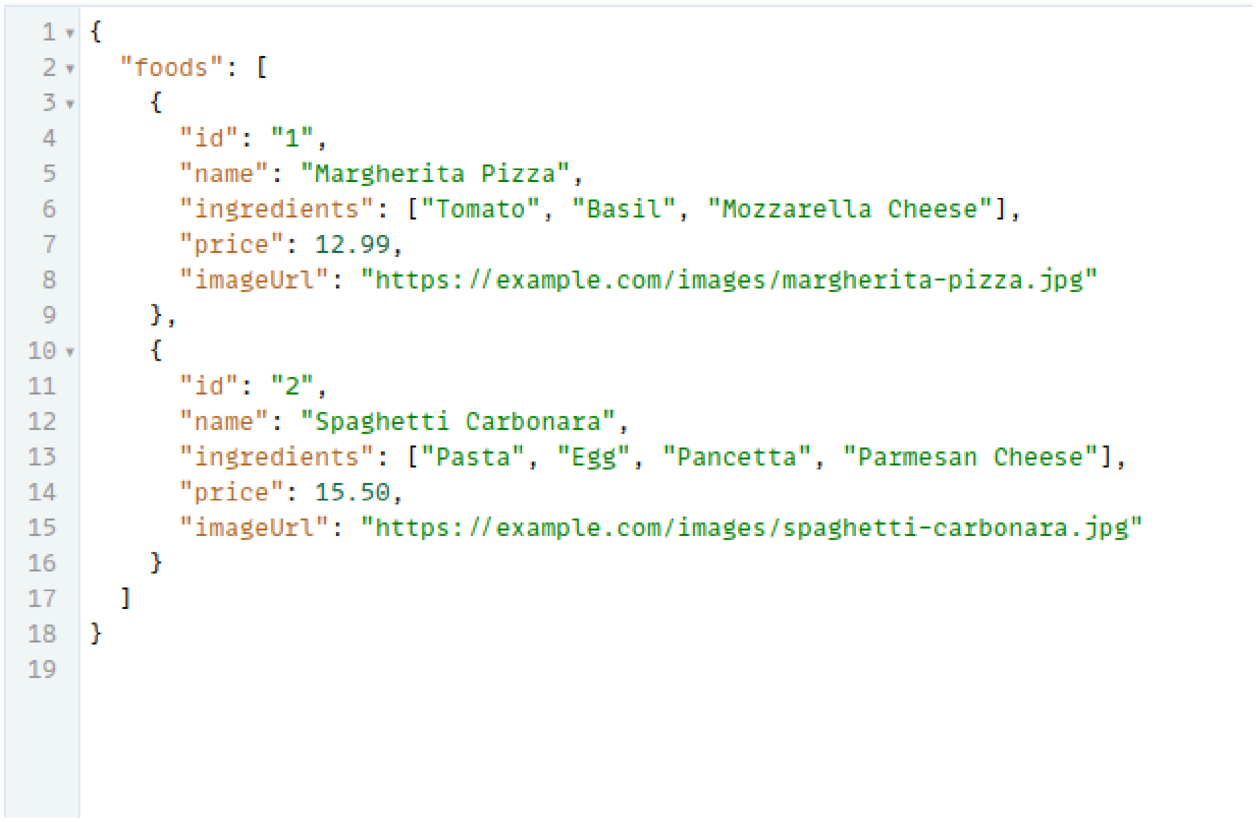
REST APIs Endpoint

1. GET METHOD: /api/foods

This API is used to obtain a list of available food items from the database which the user can browse and select the foods they want. It involves details on the food, the image, a short description, name, ingredient, price, and id

Method: GET
URL: /api/foods
Request Parameters: None
Response: Returns a JSON object containing an array of food items.

Sample Response:



```json
{
  "foods": [
    {
      "id": "1",
      "name": "Margherita Pizza",
      "ingredients": ["Tomato", "Basil", "Mozzarella Cheese"],
      "price": 12.99,
      "imageUrl": "https://example.com/images/margherita-pizza.jpg"
    },
    {
      "id": "2",
      "name": "Spaghetti Carbonara",
      "ingredients": ["Pasta", "Egg", "Pancetta", "Parmesan Cheese"],
      "price": 15.50,
      "imageUrl": "https://example.com/images/spaghetti-carbonara.jpg"
    }
  ]
}
```

2. POST METHOD /api/order

This endpoint is required to create a new order. It receives details from the customer then passes them to the server to acknowledge the order and confirm it.

Method: POST

URL: /api/order

Request Body:

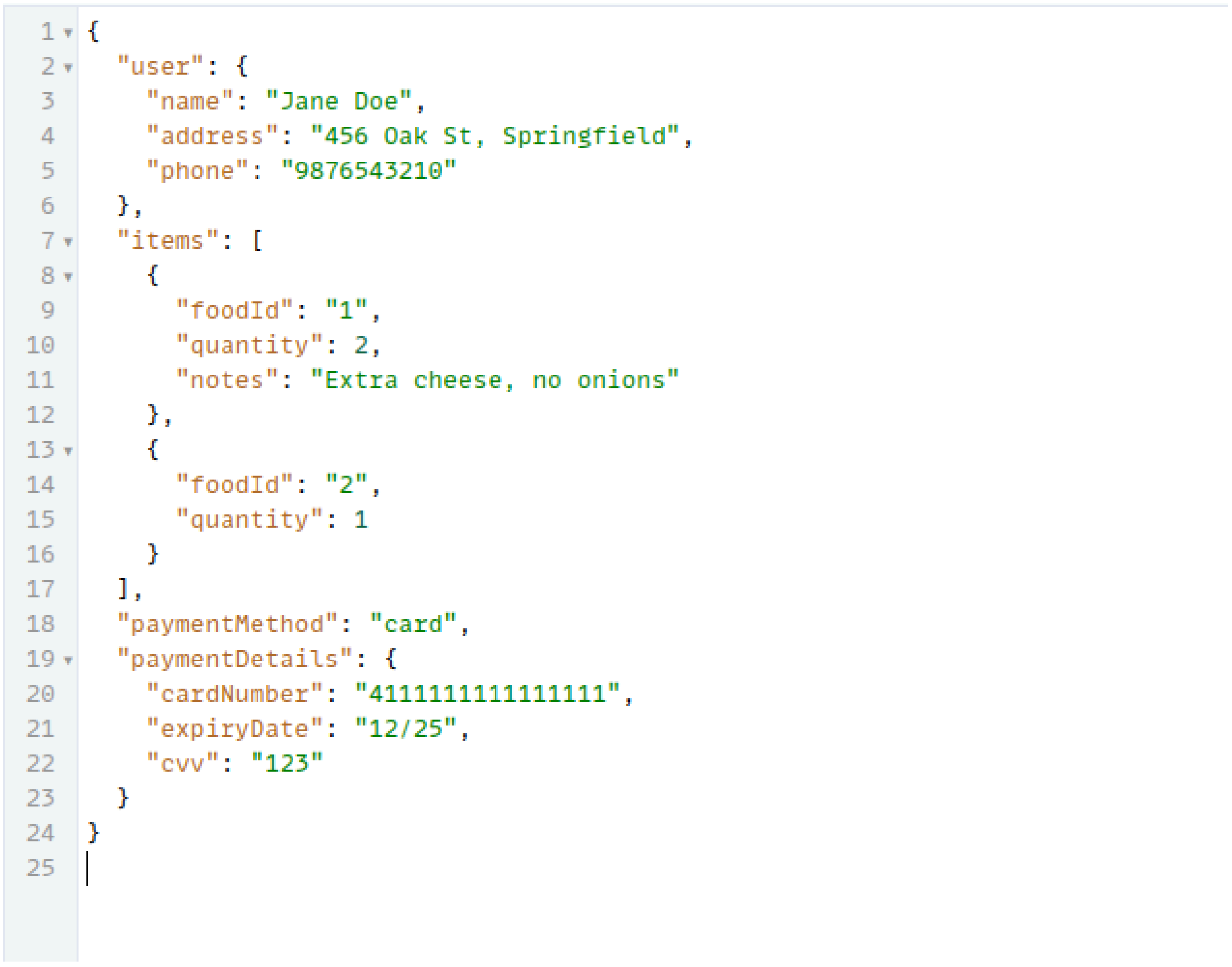
I. **user**: Object containing customer details like name, address, and phone number.

II. **items**: Array of food items, each with an ID, quantity, and optional note.

III. **paymentMethod**: Payment method (e.g., "card", "paypal") chosen by the user.

IV. **paymentDetails**: Card or payment details (if not stored in a wallet for authenticated users).

Sample Request Body

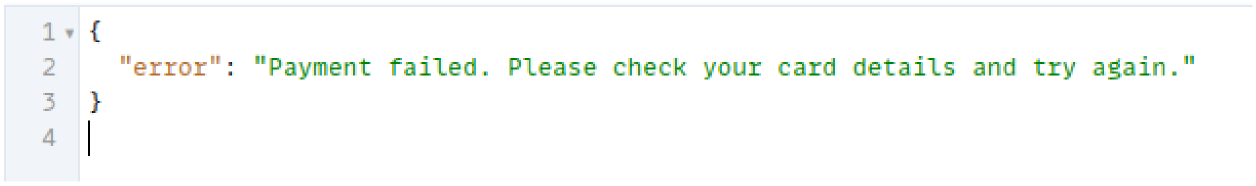


```json
{
  "user": {
    "name": "Jane Doe",
    "address": "456 Oak St, Springfield",
    "phone": "9876543210"
  },
  "items": [
    {
      "foodId": "1",
      "quantity": 2,
      "notes": "Extra cheese, no onions"
    },
    {
      "foodId": "2",
      "quantity": 1
    }
  ],
  "paymentMethod": "card",
  "paymentDetails": {
    "cardNumber": "4111111111111111",
    "expiryDate": "12/25",
    "cvv": "123"
  }
}
```

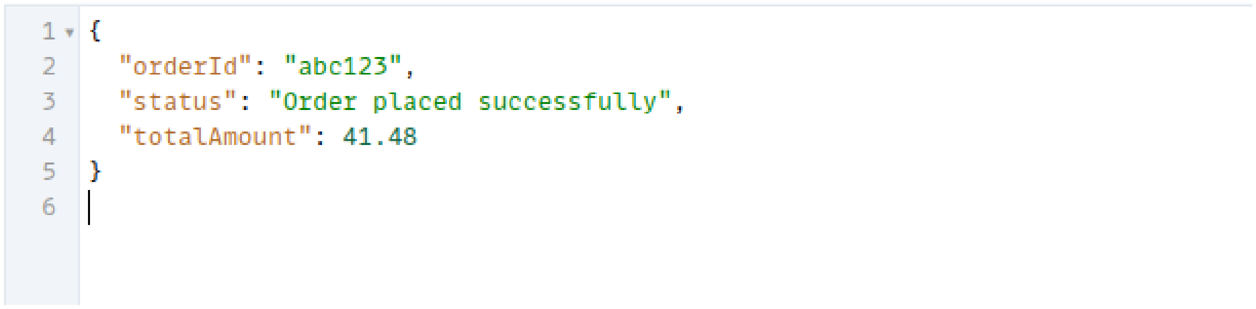
Response:

If the order is confirmed and placed successfully, a confirmation message is returned along with the total amount.

Sample Response:



```json
{
  "error": "Payment failed. Please check your card details and try again."
}
```



```json
{
  "orderId": "abc123",
  "status": "Order placed successfully",
  "totalAmount": 41.48
}
```

Error Response:

If the payment fails or the required fields are missing, an error message is returned.

3. POST METHOD /api/payments

This endpoint handles all the payments related to the orders. It connects with third-party payment solutions (for example; Stripe or PayPal) and guarantees that the payment information sent is safe. Upon confirmation of payment, the status of the particular order's payment changes in the system.

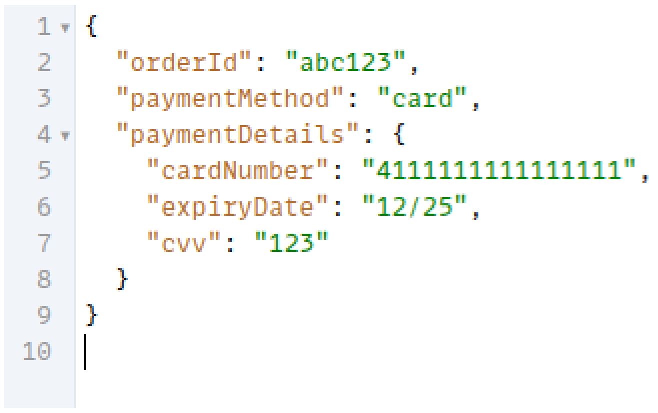
Method: POST

URL: /api/payments

Request Body:

I. **orderId**: Unique identifier for the order.
II. **paymentMethod**: The method chosen for payment (e.g., credit card, PayPal).
III. **paymentDetails**: Object containing payment-related data like card information or PayPal credentials.

Sample Request Body:



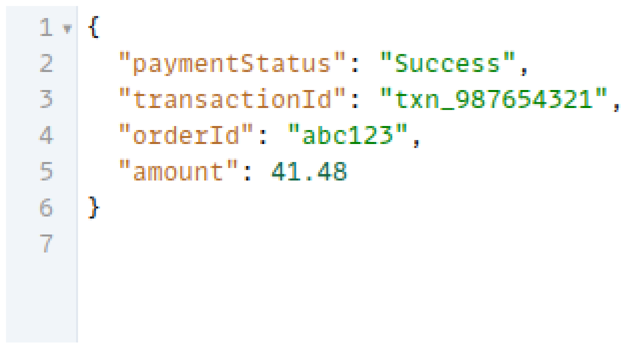
```
{
  "orderId": "abc123",
  "paymentMethod": "card",
  "paymentDetails": {
    "cardNumber": "4111111111111111",
    "expiryDate": "12/25",
    "cvv": "123"
  }
}

```

Response:

I. On successful payment, a confirmation is sent back.
II. Additionally, this endpoint will notify the order controller to update the order's status to "Paid".

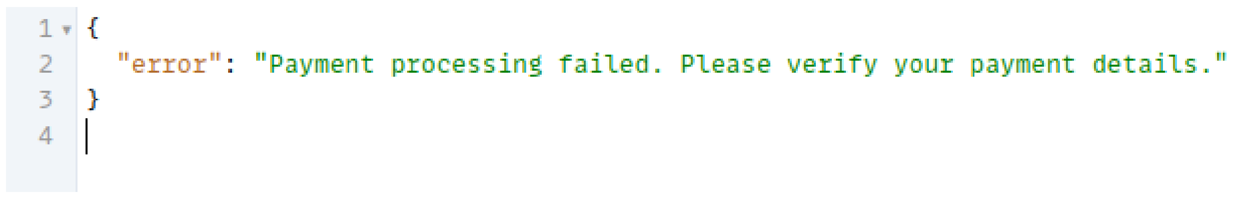
Sample Response:



```json
{
  "paymentStatus": "Success",
  "transactionId": "txn_987654321",
  "orderId": "abc123",
  "amount": 41.48
}
```

Error Response:

I. If payment processing fails due to invalid card information or gateway issues.



```json
{
  "error": "Payment processing failed. Please verify your payment details."
}
```

Building steps / Scripts

Frontend:

Install dependencies: npm install

Build the frontend: npm run build

Backend:

Install dependencies: npm install

Start the server: npm start

Install Steps / Scripts

Step 1: Installing Node.js and NVM

```
node -v
npm -v
```

Step 2: Create a new React Application

```
npx create-react-app food-ordering-frontend
cd food-ordering-frontend
```

Step 3: Install Required Dependencies

```
npm install axios react-router-dom @stripe/stripe-js @stripe/react-stripe-js
```

Step 4: Creating a new Node.js Backend

```
mkdir food-ordering-backend
cd food-ordering-backend
npm init -y
```

Step 5: Install 3rd party dependencies

```
npm install express mongoose cors body-parser dotenv stripe winston
```

Step 6: Setting up MongoDB

1. Create an account on MongoDB Atlas.
2. Create a new cluster.
3. Generate a connection string and store it in a .env file in your backend directory.

Step 7: Install Localtunnel

```
npm install -g localtunnel
```

Step 8: Run the localtunnel

```
lt --port 3001
```

Step 9: Run the frontend and backend

```
npm start
node server.js
```

Github Information:

Link: https://github.com/deepikachilukuri/comp5130/tree/week3

References

MongoDB documentation. (n.d.). MongoDB Documentation. https://docs.mongodb.com/

Express - Node.js web application framework. (n.d.). https://expressjs.com/

Stripe API reference. (n.d.). https://stripe.com/docs/api

Index | Node.js v22.9.0 Documentation. (n.d.). https://nodejs.org/en/docs/