VietFood

Final Project Technical Report

SE/COM S 3190 – Construction of User Interfaces
Spring 2025

Team Members:

Tam Minh Nguyen - tamminh@iastate.edu

Tat Bach Nguyen - ntbach@iastate.edu

May 11, 2025

1. Introduction

Viet Food is a full-stack web application developed by Tam Minh Nguyen and Bach Nguyen for the COM S 3190 final project at Iowa State University. The platform enables users to explore traditional Vietnamese dishes, place orders, make secure payments, and leave reviews—all through a responsive and user-friendly interface.

Motivated by our cultural roots and technical interests, we set out to combine Vietnamese cuisine with modern web technologies. This project showcases our skills in full-stack development using React, Node.js, and MongoDB, while offering users a convenient way to experience authentic Vietnamese food.

While the idea is original, it builds on a basic food menu app created earlier in the course. The final version includes advanced features such as authentication, cart and order management, Stripe payment integration, and an admin panel—making Viet Food both a technical milestone and a cultural expression.

2. Project Description

Explain major features, user flow, CRUD operations and entities affected.

Viet Food is a full-stack online food ordering platform that allows users to browse Vietnamese dishes, place orders, make payments, and leave reviews. The application is built using React for the frontend, Node.js/Express for the backend, and MongoDB for data storage.

Major Features:

- **User Authentication:** Signup, login, logout, and session verification using JWT.
- **Menu Browsing:** Users can explore dishes by type or ingredients.
- **Food Details:** Each item displays ingredients, price, and dietary labels.
- **Cart System:** Add, update, and remove items before checking out.
- **Checkout & Payment:** Users enter delivery details and pay securely via Stripe.
- **Reviews:** Post and read reviews for each dish.

User Flow:

1. Visitor lands on the homepage and browses the menu.
2. **Signup/Login** to create an account.
3. **Browse & Add to Cart**: Select dishes and add them to the cart.
4. **Checkout**: Fill in delivery info and complete payment.
5. **Confirmation**: View order summary and estimated delivery.
6. **Post-Order**: Users can rate dishes.

CRUD Operations and Entities:

- Users: Create (register), Read (profile, order history), Update (info).
- FoodItems: Read all the dishes and by id.
- Cart: Users can Create (add items), Read (view cart), Update (quantities), Delete (items or entire cart).

- Orders: Create (place order), Read (view orders), handled per user.
- **Reviews**: Users can Create (submit), and Read.

3. File and Folder Architecture

📁 Project Folder Structure

```
Root
├── README.md

backend/
├── .env
├── .gitignore
├── index.js
├── package.json
├── package-lock.json
├── seedReviews.js
│
├── controllers/
│   ├── auth.js
│   ├── cart.js
│   ├── foodItem.js
│   ├── order.js
│   ├── payment.js
│   ├── review.js
│   └── user.js
│
├── models/
│   ├── Cart.js
```

```
│   ├── FoodItem.js
│   ├── Order.js
│   ├── Review.js
│   └── User.js
│
├── routes/
│   ├── authRoutes.js
│   ├── cartRoutes.js
│   ├── foodItemRoutes.js
│   ├── orderRoutes.js
│   ├── paymentRoutes.js
│   ├── reviewRoutes.js
│   └── userRoutes.js
│
└── utils/
    ├── error.js
    └── verifyToken.js

Documents/
├── MN6_Final_Proposal (1).pdf
│
└── Mini-Assignments/
    ├── Mini-Assignments 1/
    │   ├── mini_assignment_1_Bach_Nguyen.png
    │   └── mini_assignment_1_Tam_Minh_Nguyen.png
    │
    └── mini_assigment_2/
```

```
    ├── mini_2_Bach_Nguyen.mp4
    └── mini_2_Tam_Minh_Nguyen.mp4


frontend/
├── .env
├── .gitignore
├── package.json
├── package-lock.json
├── README.md
│
├── public/
│   ├── favicon.ico
│   ├── index.html
│   ├── logo192.png
│   ├── logo512.png
│   ├── manifest.json
│   ├── robots.txt
│   │
│   └── images/
│       ├── bach.jpg
│       └── minh.jpg
│
└── src/
    ├── App.js
    ├── App.css
    ├── App.test.js
    ├── index.js
```

```
├── index.css
├── logo.svg
├── reportWebVitals.js
└── setupTests.js
│
├── components/
│   ├── footer/
│   │   ├── Footer.jsx
│   │   └── footer.css
│   │
│   ├── navbar/
│   │   ├── Navbar.jsx
│   │   └── navbar.css
│   │
│   └── protectedRoute/
│       └── ProtectedRoute.jsx
│
├── context/
│   ├── AuthContext.js
│   └── CartContext.js
│
├── hooks/
│   └── useFetch.js
│
└── pages/
    ├── about/
    │   ├── About.jsx
```

```
│   └── about.css
│
├── aboutus/
│   ├── Aboutus.jsx
│   └── Aboutus.module.css
│
├── checkout/
│   ├── Checkout.jsx
│   └── checkout.css
│
├── checkoutWrapper/
│   └── CheckoutWrapper.jsx
│
├── foodItem/
│   ├── FoodItem.jsx
│   └── FoodItem.module.css
│
├── home/
│   ├── Home.jsx
│   └── home.css
│
├── login/
│   ├── Login.jsx
│   └── login.css
│
├── menu/
│   ├── Menu.jsx
```

```
│   └── menu.css
│
├── myCart/
│   ├── myCart.jsx
│   └── myCart.css
│
├── orderConfirmation/
│   ├── OrderConfirmation.jsx
│   └── OrderConfirmation.module.css
│
└── signUp/
    ├── Signup.jsx
    └── signup.css
```

Project Folder Structure Description

frontend/ – Client-side React Application

This directory contains the React-based frontend of the Viet Food app.

Key Files & Folders:

- .env: Stores environment variables (e.g., API base URLs).
- package.json: Manages project dependencies and scripts.
- public/: Static files served directly (like index.html, favicon.ico, and images).
- src/: All source code, organized into:
  - components/: Reusable UI components like Navbar, Footer, ProtectedRoute.
  - pages/: Page-level views for routes like Home, Login, Menu, Checkout, etc.
  - context/: React Contexts for global state (e.g., AuthContext, CartContext).

- hooks/: Custom hooks like useFetch for data fetching.

- Other core files: App.js for routing/layout and index.js as the entry point.

backend/ – Server-side Express Application

This directory holds the Node.js/Express API server, responsible for database access, authentication, and order management.

Key Files & Folders:

- .env: Contains sensitive credentials (MongoDB URI, Stripe keys, JWT secret).

- index.js: Entry point of the backend app. Sets up routes, middleware, DB connection.

- controllers/: Business logic for each route (auth, cart, food, order, etc.).

- models/: Mongoose schemas for MongoDB collections (User, Order, Cart, etc.).

- routes/: Express route definitions that map endpoints to controller functions.

- utils/: Helper functions like error.js and token verification middleware.

Documents/ – Supporting Materials

Contains proposal and mini-assignment deliverables related to the class project.

Key Files & Folders:

- MN6_Final_Proposal (1).pdf: Final project proposal document.

- Mini-Assignments/:

  - Mini-Assignments 1/: Images for team members' first mini-assignments.

  - mini_assigment_2/: Video presentations (.mp4) of the second mini-assignments.

4. Code Explanation and Logic Flow

4.1. Frontend–Backend Communication

The frontend (built with React) interacts with the backend (Node.js/Express) using **HTTP requests** via **fetch** and **axios**. These requests target RESTful API endpoints hosted at **http://localhost:8800/api/**. Authentication tokens are managed via cookies with credentials: 'include' to support secure, session-based operations.

Client-Side (Frontend) API requests:

- User Authentication **(/api/auth/login, /verifyToken, /logout)**: Handled using **fetch** inside **AuthContext.js**. JWT tokens are set in HTTP-only cookies to maintain sessions.
- Cart Operations **(/api/cart/:userId/cart, etc.)**: **axios** is used inside **CartContext.js** to add, update, delete, and clear cart items, with optimistic UI updates and localStorage caching.
- Food Item Data **(/api/fooditems, /api/fooditems/:id)**: **useFetch.js** and **axios** retrieve menu data and dish details for display.
- Order Processing **(/api/orders)**: On successful Stripe payment, the order is sent to the backend using **axios.post** from **Checkout.jsx**.
- Stripe Payment **(/api/payments/create-payment-intent)**: **CheckoutWrapper.jsx** sends a POST request to generate a **clientSecret** for Stripe's Payment Element.

Server-Side (Backend) Handling:

- The Express server uses routers **(authRoutes.js, cartRoutes.js,** and **orderRoutes.js)** to match incoming requests to controller functions.
- Controllers such as **auth.js**, **cart.js**, **order.js**, etc., handle logic, interact with MongoDB via Mongoose models (User, Cart, Order, etc.), and return JSON responses.
- Middleware such as **verifyToken.js** secures protected routes.
- Error responses are standardized through a custom **createError** utility and a central Express error handler.

4.2. React Component Structure

AuthContext.js

- **State:**
  - user: Object storing user info (e.g., name, email).
  - userId: Used globally (especially by CartContext).
  - loading, error: Status for async auth operations.
- **Functions provided via context:**
  - login(email, password)
  - logout()
- **Props:**

- children: The nested app components that consume auth state.

CartContext.js

- **State:**
  - cart: Includes an array of foodItems with quantity.
  - total: Calculated from cart items.
  - loading, debounceTimeout: For UI responsiveness and API optimization.
- **Functions provided via context:**
  - addToCart(foodItem)
  - updateItemQuantity(foodItemId, quantity)
  - removeFromCart(foodItemId)
  - clearCart()
  - getTotal()
- **Props:**
  - children: Wrapped component tree under the CartProvider.

State Dependencies:

- CartContext depends on AuthContext's userId.
- Route-level components rely on context to display cart contents, protect routes, and handle authentication-based logic.

4.3. Database Interaction

The Viet Food application uses MongoDB as its NoSQL database, with Mongoose for schema modeling and object-document mapping. The system manages five main entities: **User, FoodItem, Cart, Order,** and **Review**, each represented by a Mongoose schema.

1. **User**:
   - Stores personal information, login credentials, address, and admin status.
   - Linked to **Order** through the **orderHistory** array.
   - Fields: **firstName, lastName, email, password, address, orderHistory, isAdmin.**
2. **FoodItem**:
   - Represents each menu item.

- Includes name, description, ingredients, price, category (enum), image URL, and a featured flag.
- This is a central reference in **Cart, Order,** and **Review** models.

3. **Cart**:
   - Each user has one unique cart.
   - Stores an array of food items with their respective quantities.
   - References both **User** and **FoodItem** via **ObjectId**.
   - Enforced by **unique: true** constraint on the **user** field.
4. **Order**:
   - Captures user-placed orders, including selected food items, delivery address, total cost, payment method, and status.
   - References **User** and **FoodItem**.
   - Supports status tracking **(Pending, Completed, Canceled)**.
5. **Review**:
   - Stores feedback from users for specific dishes.
   - Linked to both **User** and **FoodItem**.
   - Includes **rating, comment,** and **timestamps** for sorting and display.

Usage:

- Relationships are handled via **ObjectId** references and populated using Mongoose's **.populate()** method.
- CRUD operations are executed via API calls in the controllers, which then interact with these models for data creation, reading, updates, or deletions.
- This schema setup allows efficient joins (via population), scalable data structure, and clear separation of concerns between users, menu items, orders, and feedback.

4.4. Code Snippets

Include 2–3 meaningful snippets (React component, backend route, DB logic).

Frontend:

```
// Add an item to the cart
const addToCart = async (foodItem) => {
  try {
    const response = await axios.post(`http://localhost:8800/api/cart/${userId}/cart`, {
      foodItemId: foodItem._id,
      quantity: 1,  // Adjust this as needed
    });
    setCart(response.data);
    localStorage.setItem(`cart_${userId}`, JSON.stringify(response.data)); // Update localStorage
  } catch (err) {
    console.error('Error adding item to cart:', err);
  }
};
```

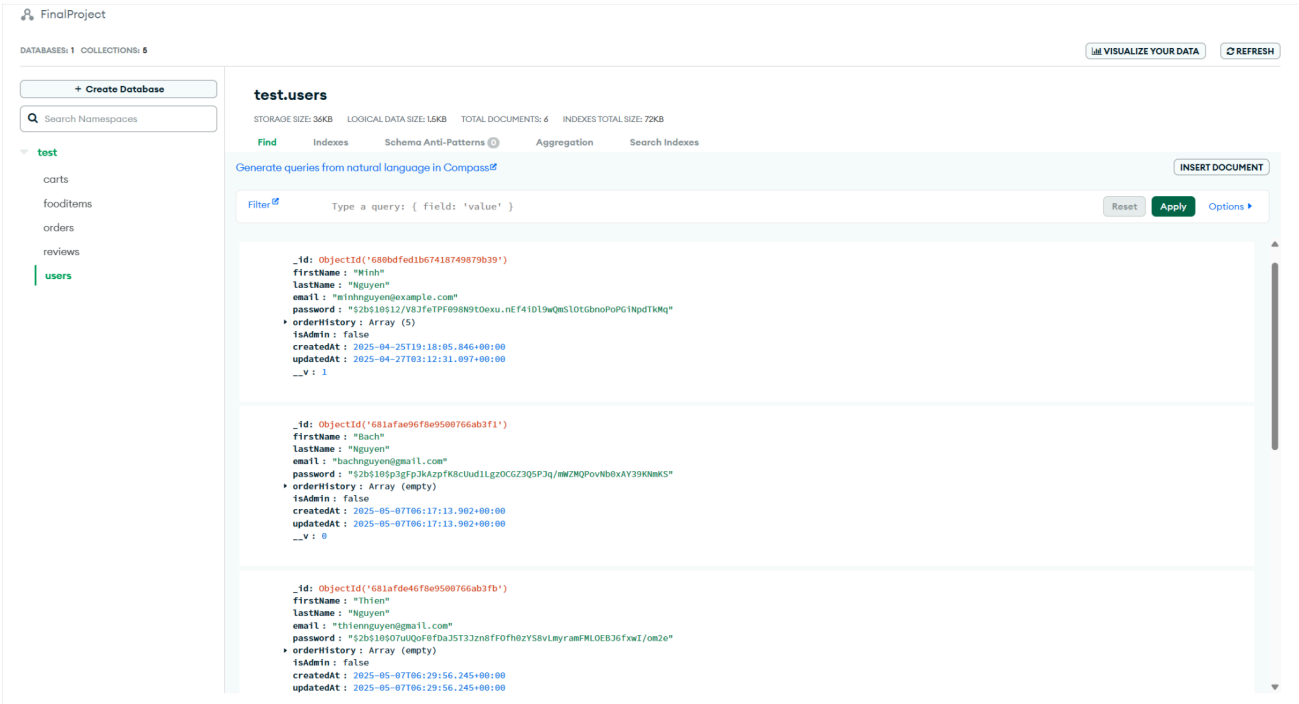
```
// Remove an item from the cart
const removeFromCart = async (foodItemId) => {
  try {
    const response = await axios.delete(`http://localhost:8800/api/cart/${userId}/cart/${foodItemId}`);
    setCart(response.data); // Update the cart after removal
    localStorage.setItem(`cart_${userId}`, JSON.stringify(response.data)); // Update localStorage
  } catch (err) {
    console.error('Error removing item from cart:', err);
  }
};
```

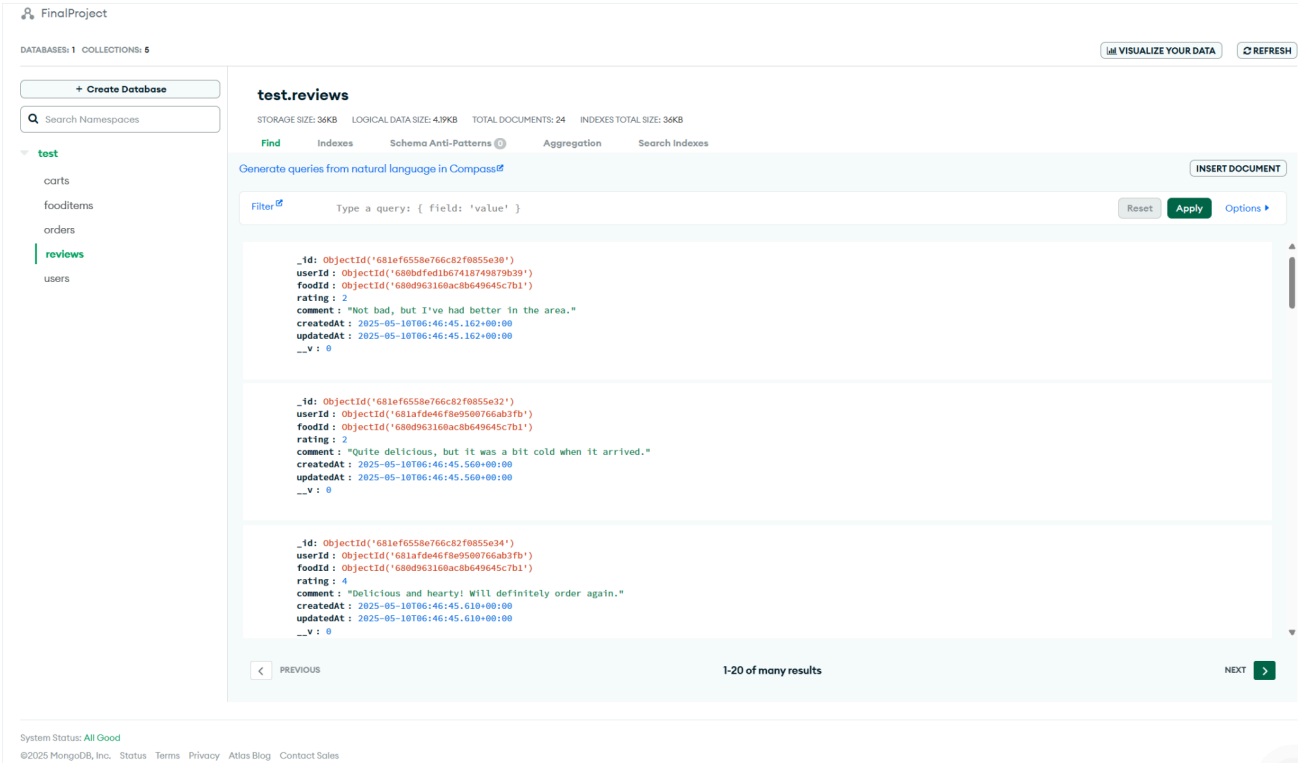
Backend:

```
18  // Add a food item to the cart
19  export const addToCart = async (req, res) => {
20    const { userId } = req.params;
21    const { foodItemId, quantity } = req.body;
22
23    try {
24      let cart = await Cart.findOne({ user: userId });
25
26      if (!cart) {
27        cart = new Cart({
28          user: userId,
29          foodItems: [{ foodItem: foodItemId, quantity }],
30        });
31        await cart.save();
32      } else {
33        const itemIndex = cart.foodItems.findIndex(
34          (item) => item.foodItem.toString() === foodItemId
35        );
36
37        if (itemIndex === -1) {
38          cart.foodItems.push({ foodItem: foodItemId, quantity });
39        } else {
40          cart.foodItems[itemIndex].quantity += quantity;
41        }
42
43        await cart.save();
44      }
45
46      const populatedCart = await Cart.findOne({ user: userId }).populate('foodItems.foodItem');
47      res.status(200).json(populatedCart);
48    } catch (error) {
49      res.status(500).json({ message: 'Server error', error: error.message });
50    }
51  };
```

```
// Remove a food item from the cart
export const removeCartItem = async (req, res) => {
  const { userId, foodItemId } = req.params;

  try {
    const cart = await Cart.findOne({ user: userId });

    if (!cart) {
      return res.status(404).json({ message: 'Cart not found for this user' });
    }

    cart.foodItems = cart.foodItems.filter(
      (item) => item.foodItem.toString() !== foodItemId
    );

    await cart.save();

    const populatedCart = await Cart.findOne({ user: userId }).populate('foodItems.foodItem');
    res.status(200).json(populatedCart);
  } catch (error) {
    res.status(500).json({ message: 'Server error', error: error.message });
  }
};
```

Database:



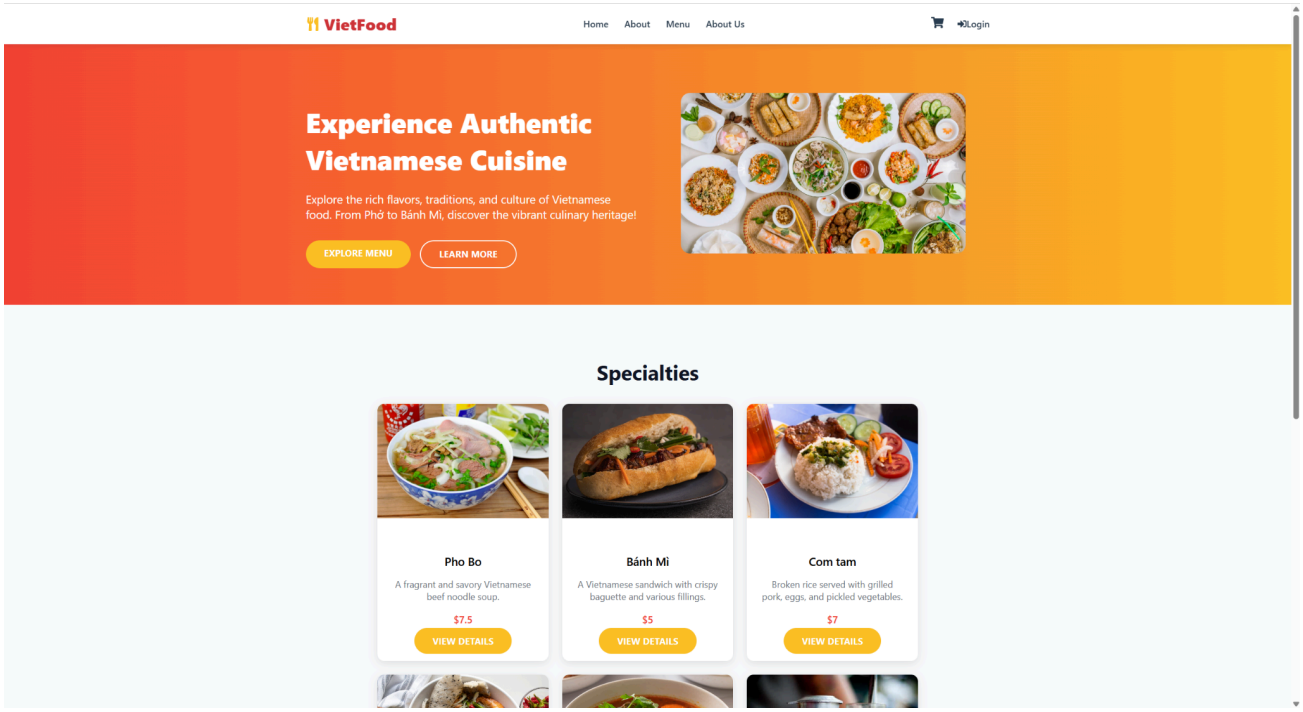

FinalProject
DATABASES: 1 COLLECTIONS: 5
VISUALIZE YOUR DATA
REFRESH
+ Create Database
Search Namespaces
test
carts
fooditems
orders
reviews
users
test.reviews
STORAGE SIZE: 36KB LOGICAL DATA SIZE: 4.19KB TOTAL DOCUMENTS: 24 INDEXES TOTAL SIZE: 36KB
Find Indexes Schema Anti-Patterns 0 Aggregation Search Indexes
Generate queries from natural language in Compass
INSERT DOCUMENT
Filter
Type a query: { field: 'value' }
Reset Apply Options
_id: ObjectId('681ef6558e766c82f0855e30')
userId : ObjectId('680bdfed1b67418749879b39')
foodId : ObjectId('680d963160ac8b649645c7b1')
rating : 2
comment : "Not bad, but I've had better in the area."
createdAt : 2025-05-10T06:46:45.162+00:00
updatedAt : 2025-05-10T06:46:45.162+00:00
__v : 0
_id: ObjectId('681ef6558e766c82f0855e32')
userId : ObjectId('681afde46f8e9500766ab3fb')
foodId : ObjectId('680d963160ac8b649645c7b1')
rating : 2
comment : "Quite delicious, but it was a bit cold when it arrived."
createdAt : 2025-05-10T06:46:45.560+00:00
updatedAt : 2025-05-10T06:46:45.560+00:00
__v : 0
_id: ObjectId('681ef6558e766c82f0855e34')
userId : ObjectId('681afde46f8e9500766ab3fb')
foodId : ObjectId('680d963160ac8b649645c7b1')
rating : 4
comment : "Delicious and hearty! Will definitely order again."
createdAt : 2025-05-10T06:46:45.610+00:00
updatedAt : 2025-05-10T06:46:45.610+00:00
__v : 0
PREVIOUS
1-20 of many results
NEXT
System Status: All Good
©2025 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales

5. Web View Screenshots and Annotations

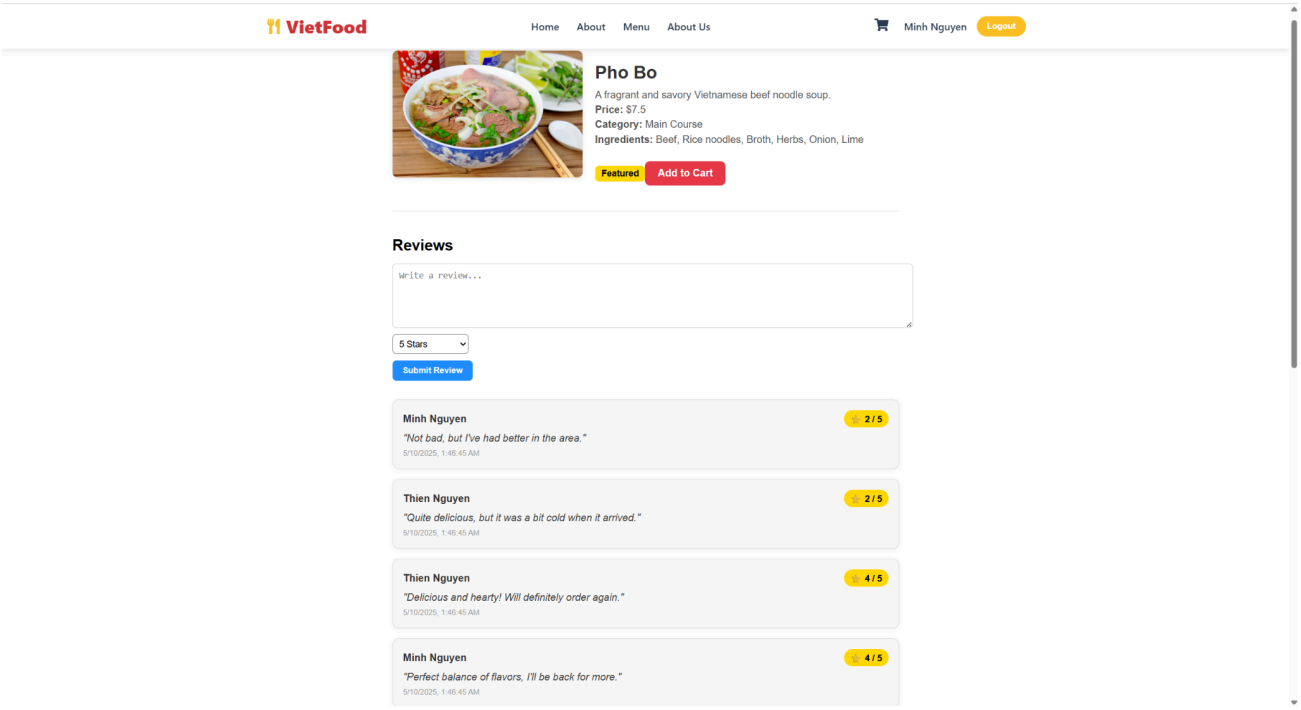
Insert and annotate full-page screenshots for 4 features (2 for each team member)

Description: What this page does and what the user can do.

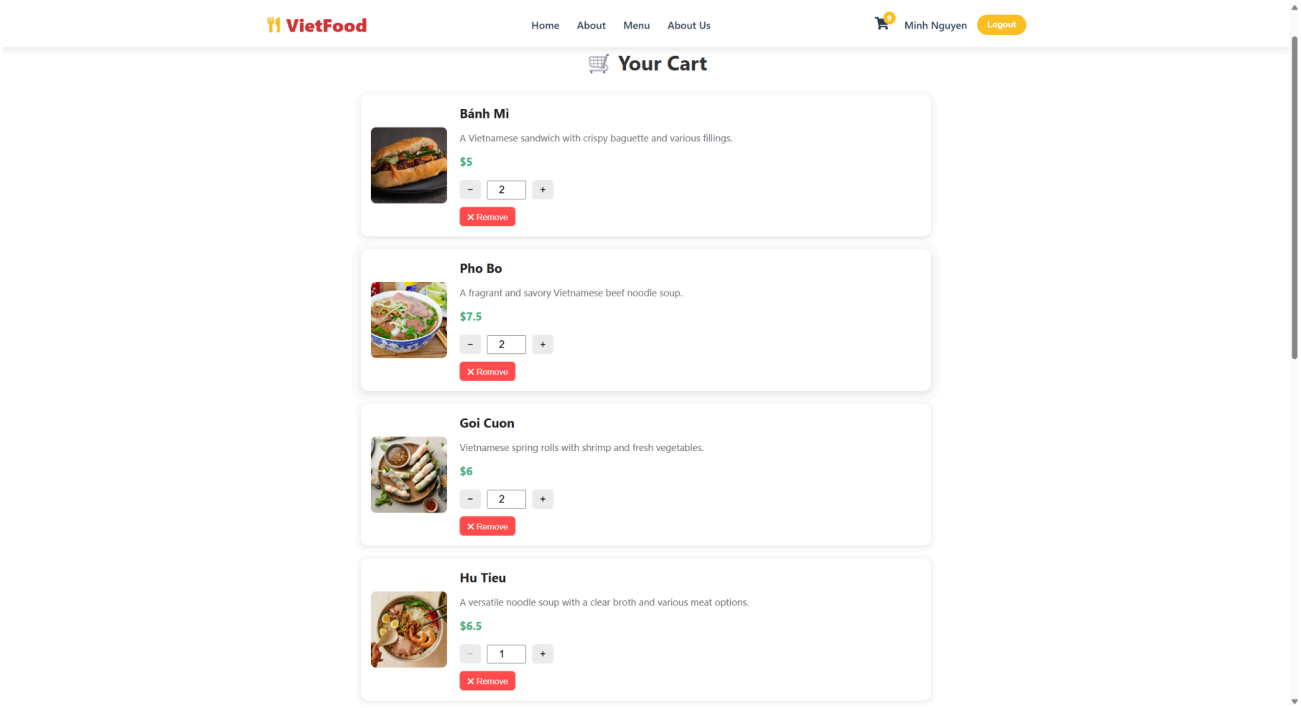
Home page: User can view all the special Vietnamese dishes in the home page



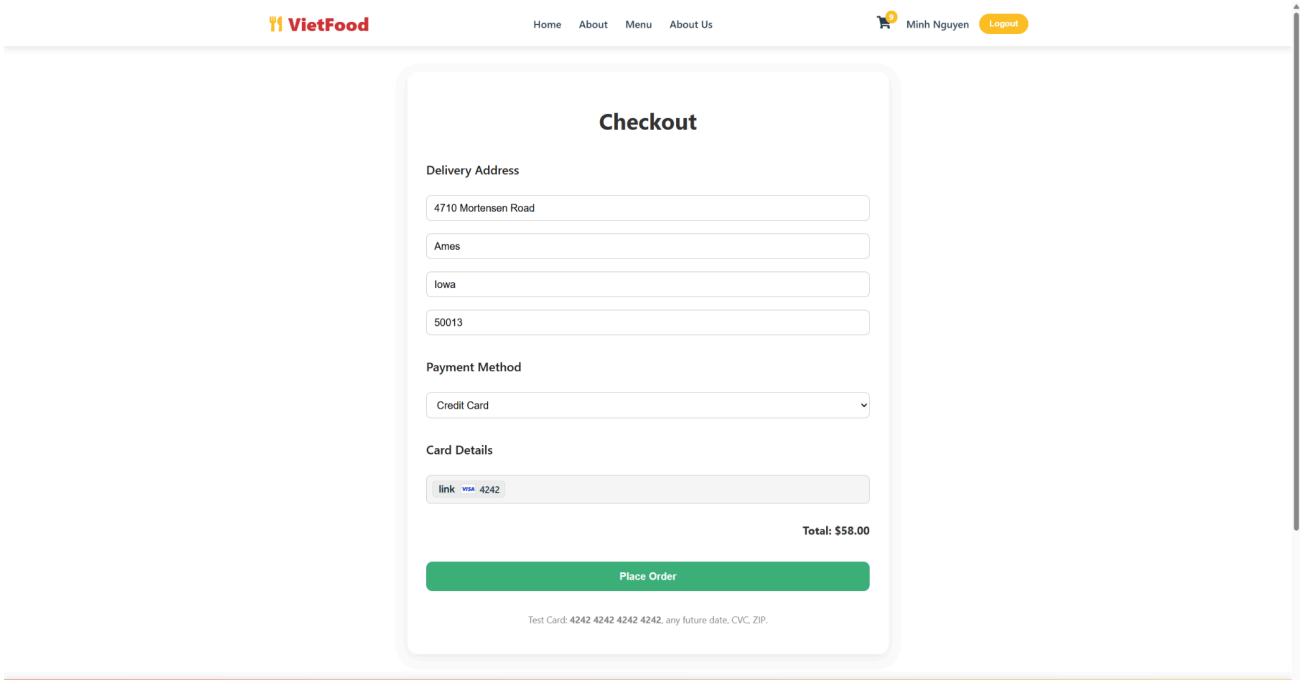
Food detail page: Users can view the details of a dish, including its description and reviews. They also have the option to write their own reviews.



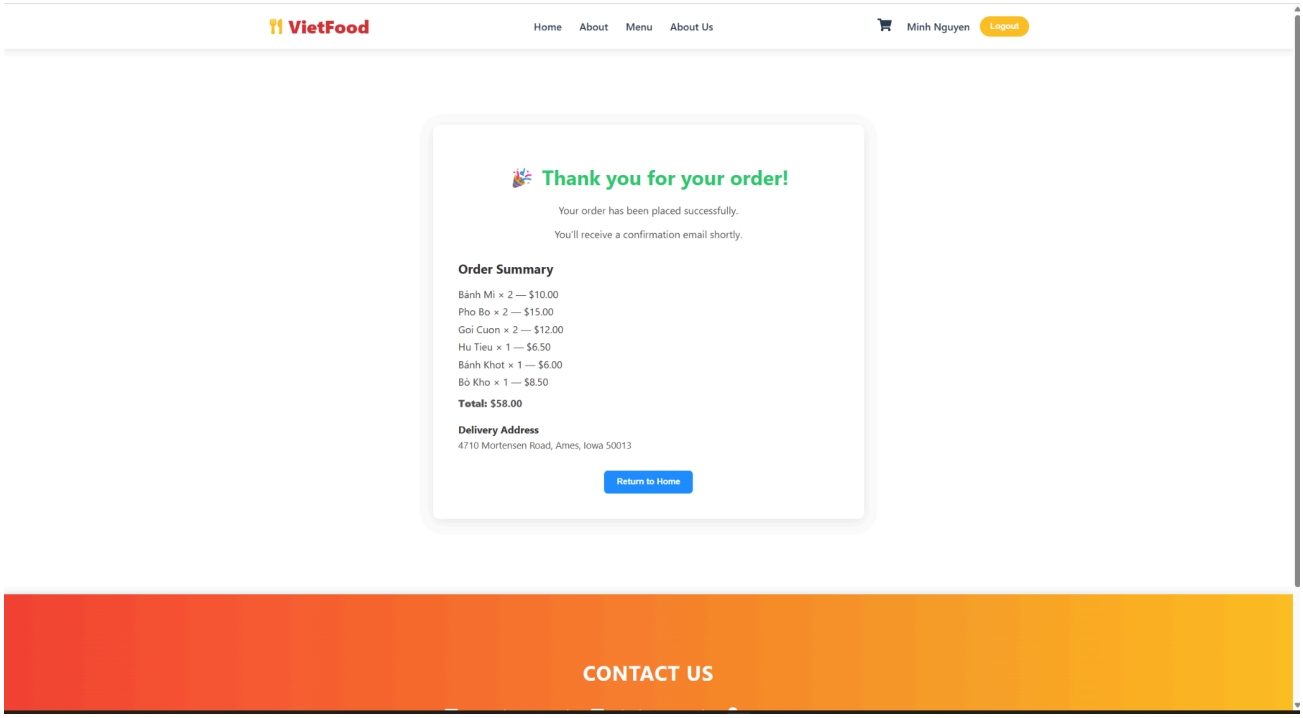
Cart page: Users can view all the dishes in their cart, adjust the quantity of each item, or remove them as needed.



Checkout page: Users enter their delivery address and provide credit card details to complete the payment.



Confirmation page: Users can view a summary of their order along with their delivery address.



About page: Users can explore and learn about Vietnamese food culture.

6. Installation and Setup Instructions

1. Prerequisites

Make sure you have the following installed:

- **Node.js** (v18 or above recommended)
- **MongoDB Atlas account**
- **Stripe account** (for payment integration)

2. Clone the Repository

git clone https://github.com/minhtnguyen2309/FinalProject319.git
cd FinalProject319

Backend Setup (/backend):

Step 1: Navigate to the backend folder
cd backend

Step 2: Install dependencies

npm install

Step 3: Create .env file

MONGO=mongodb+srv://minhnguyentam2k5:dKB7y6RkRixyBKCL@finalproject.ryns9uj.mongodb.net/?retryWrites=true&w=majority&appName=FinalProject

JWT_SECRET=298a58f004850f58f4b08b8a8dee65b006f930fe1a88728acd71148286612c695e4ef1eb737401eec9d0cbd6daec5e2915b576e3ad3c0a6f095c7a7e34b40133

STRIPE_SECRET_KEY=sk_test_51RKueq4DpXguk5ssumODAvPm4phFRnahZCGEOQPfadO3Zr4g6xn5eemn7MRgZqGk4IZyUPDaGQFqJApwwDejtnGJ00QZabAvRv

Step 4: Start the backend server

node index.js

Frontend Setup (/frontend):

Step 1: Navigate to the frontend folder

cd ../frontend

Step 2: Install dependencies

npm install

Step 3: Create .env file

REACT_APP_STRIPE_PUBLIC_KEY=pk_test_51RKueq4DpXguk5ssFSH03ucjFJmycv15jHID0UwQLu3UT1qi8GivFQW3ejjS1VAIKML1DkpWeNI4o6DJoE7cYc6W001X1W7yjH

Step 4: Start the React development server

npm start

7. Contribution Overview

Contribution Overview

Feature	Responsible Team Member

Signup	Bach
Login	Bach
MyCart	Bach
About Us	Bach
Home	Bach
Checkout	Minh
Confirmation	Minh
AuthContext	Minh
CartContext	Minh
FoodItem	Minh

8. Challenges Faced

1. **Cart Synchronization**
   We faced issues with inconsistent cart data between the frontend and backend. To fix this, we cached cart data in localStorage and synchronized it with the backend using **axios** when the user logged in.
2. **JWT Authentication**
   The login system didn't persist sessions across refreshes. We resolved this by storing the JWT in **httpOnly** cookies and using a **/verifyToken** route to ensure secure, persistent sessions.
3. **Stripe Integration**
   We ran into errors with payment amounts and **clientSecret** handling. By adjusting the backend to convert amounts to cents and using a **/create-payment-intent** route, we fixed the issues and enabled seamless payments.

9. Final Reflections

This project provided invaluable hands-on experience with full-stack development, particularly in integrating React, Node.js, and MongoDB. I learned how to manage authentication securely with JWT, handle state synchronization between the frontend and backend, and integrate payment systems like Stripe effectively. Additionally, working with Mongoose for database queries and population was an essential part of handling complex data relationships.

While the project was successful, one area for improvement would be enhancing the error handling across both the frontend and backend. Implementing better feedback mechanisms for users during payment or cart operations could improve the user experience. Additionally, optimizing the database queries to handle larger datasets efficiently would further improve scalability.