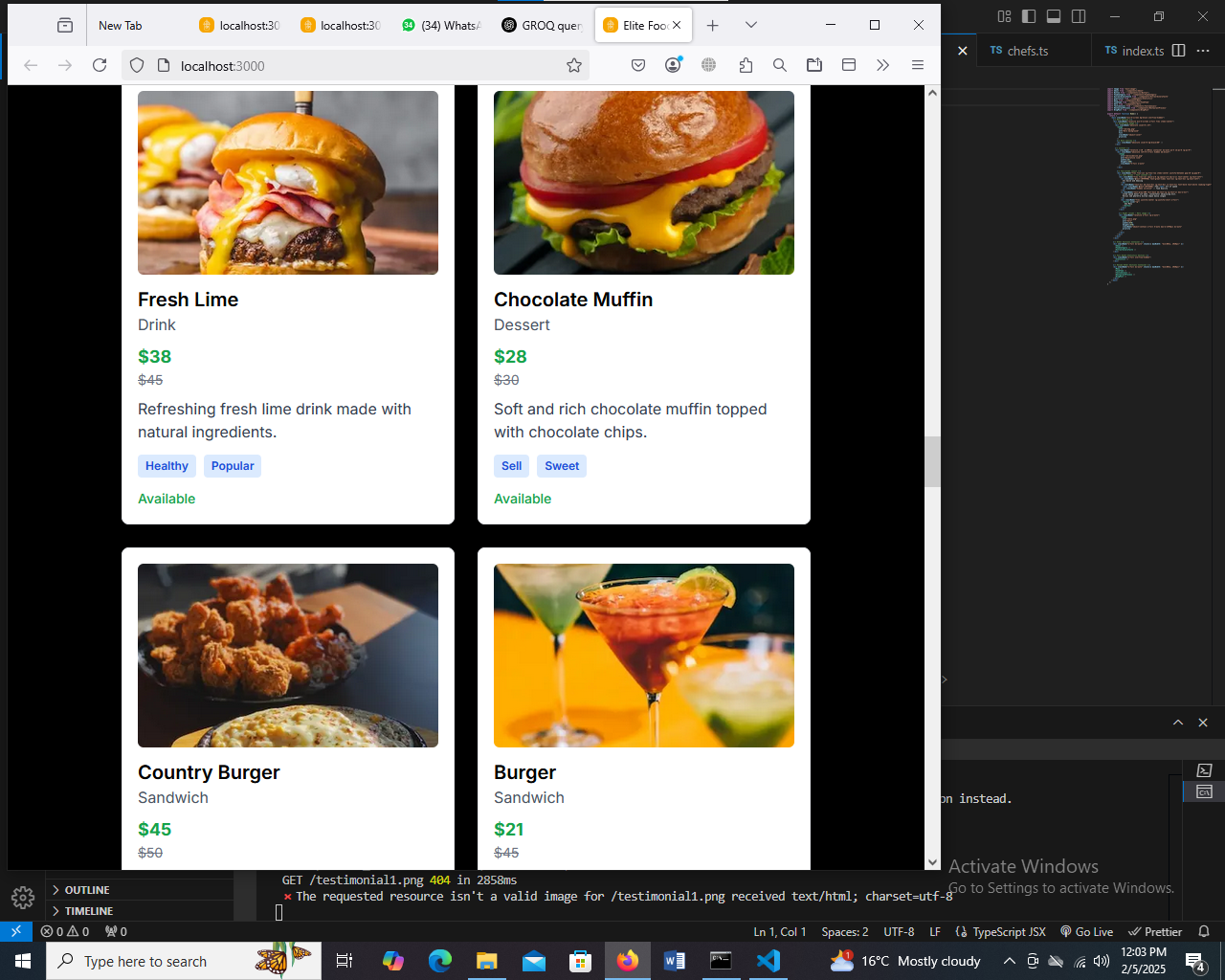
**CDay:4 Dynamic Frontend Component-FoodTuck**

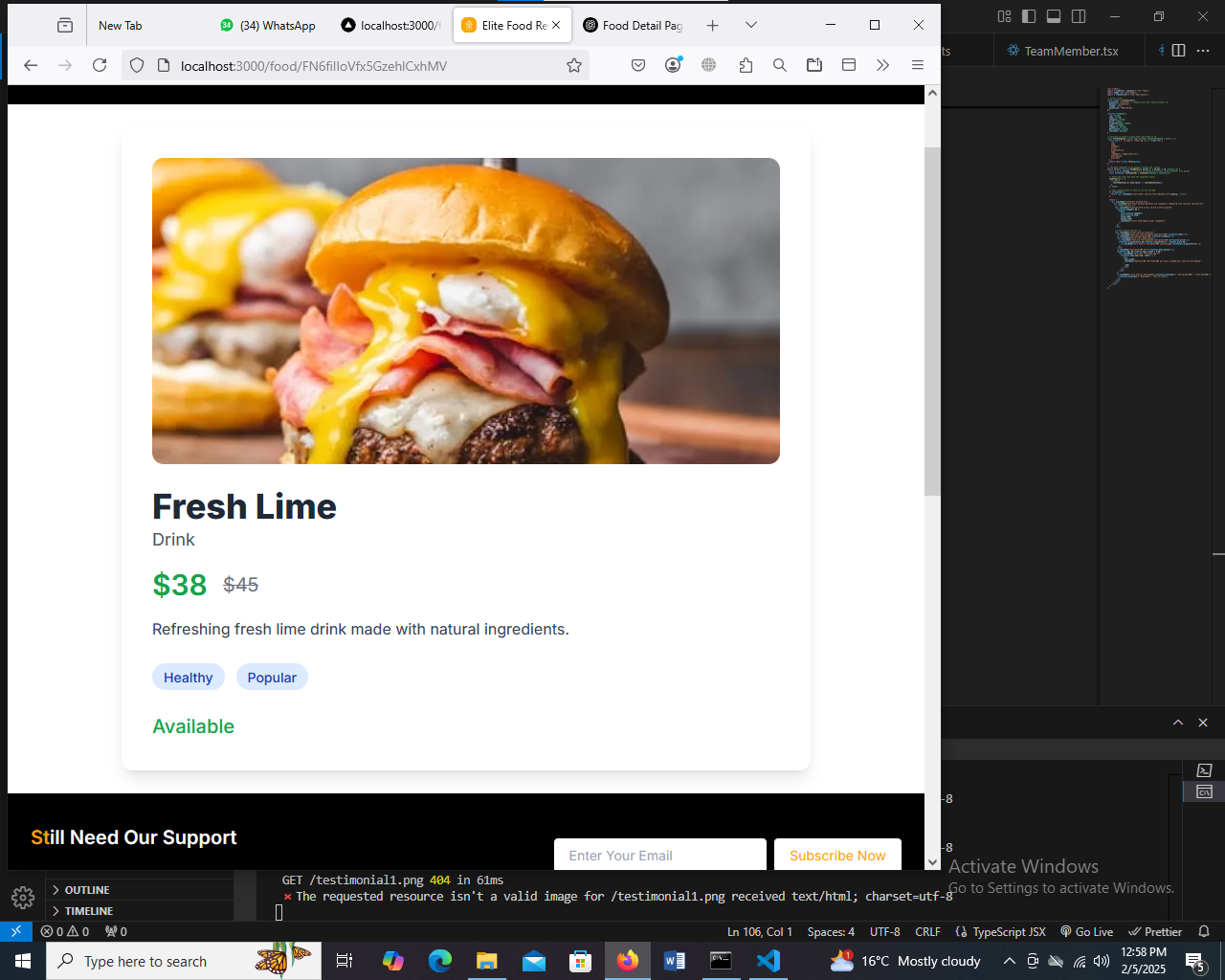
**Made by: Muhammad Bilal Qureshi**

**ID: 00075879 Sunday 2:00 to 5:00**

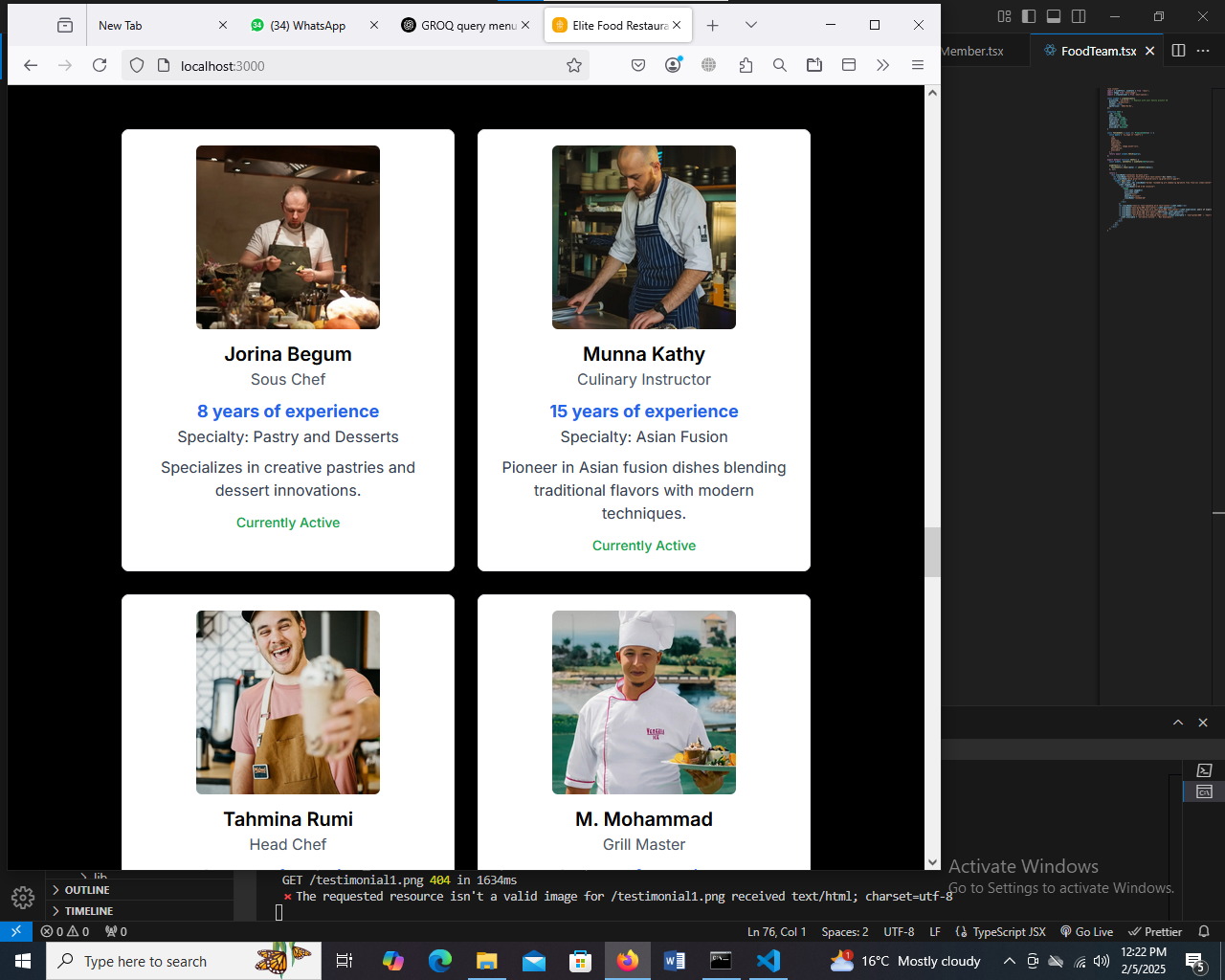
**Product Listing Component:**

****

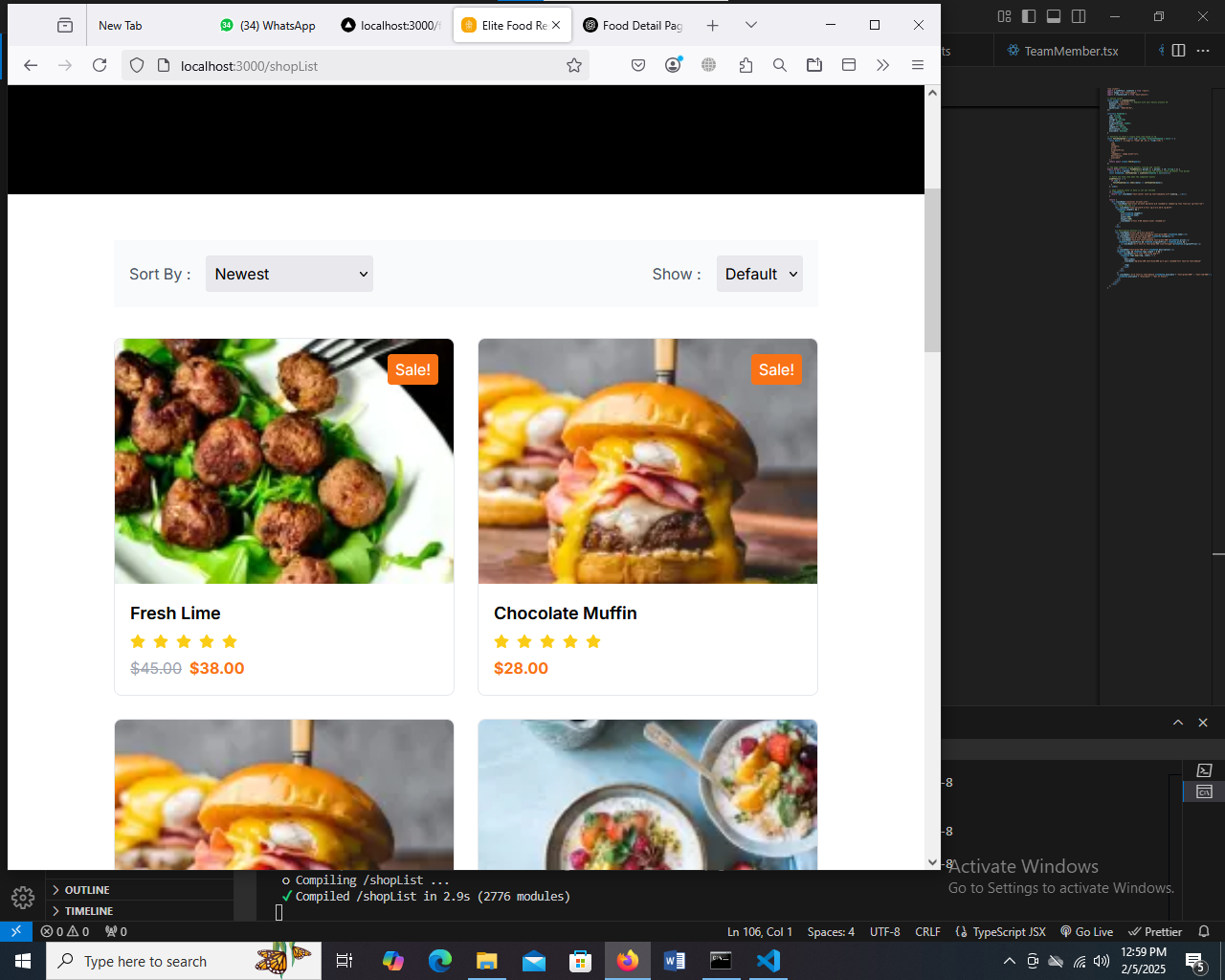
**Product Detail:**

****

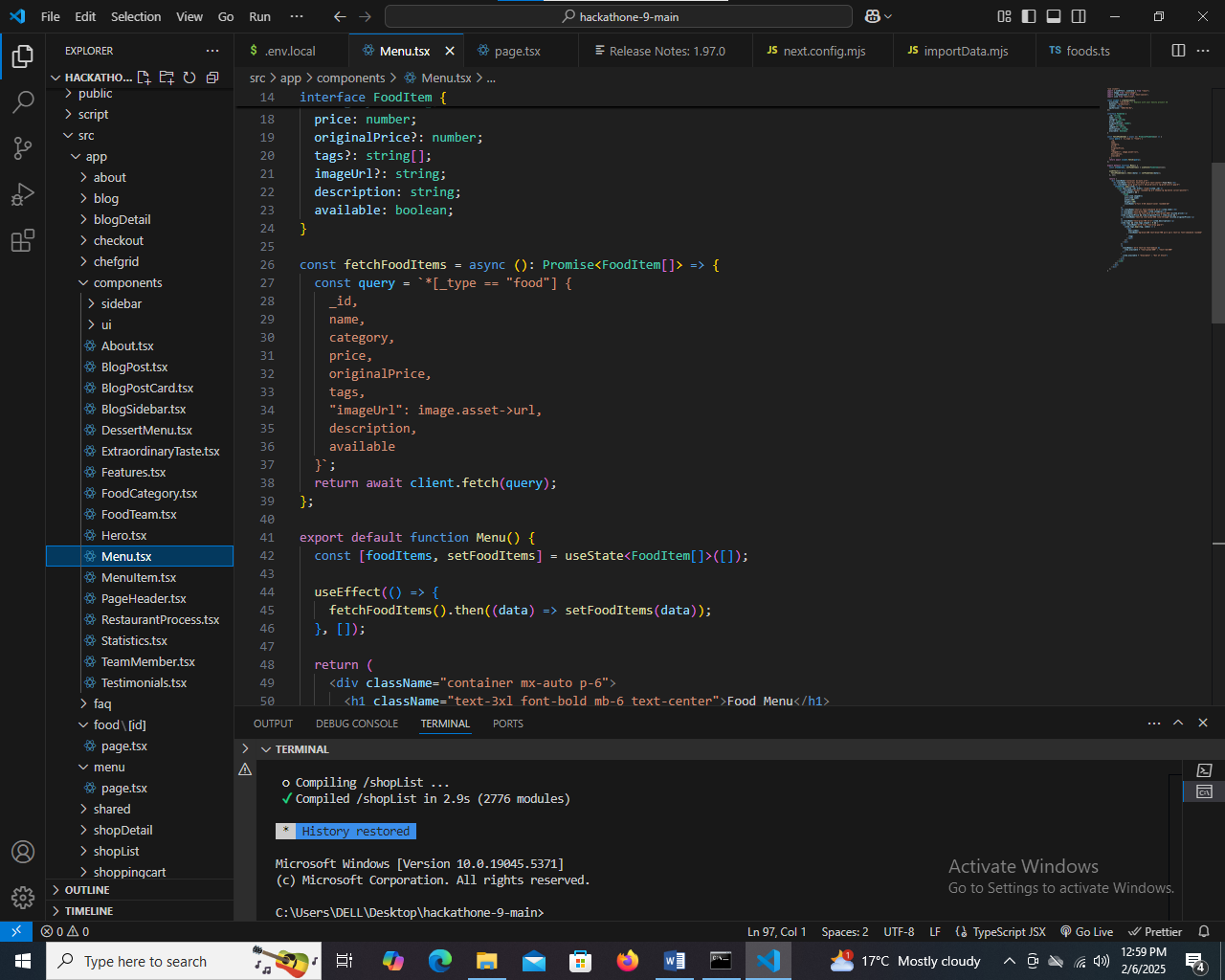
**Chef Component:**

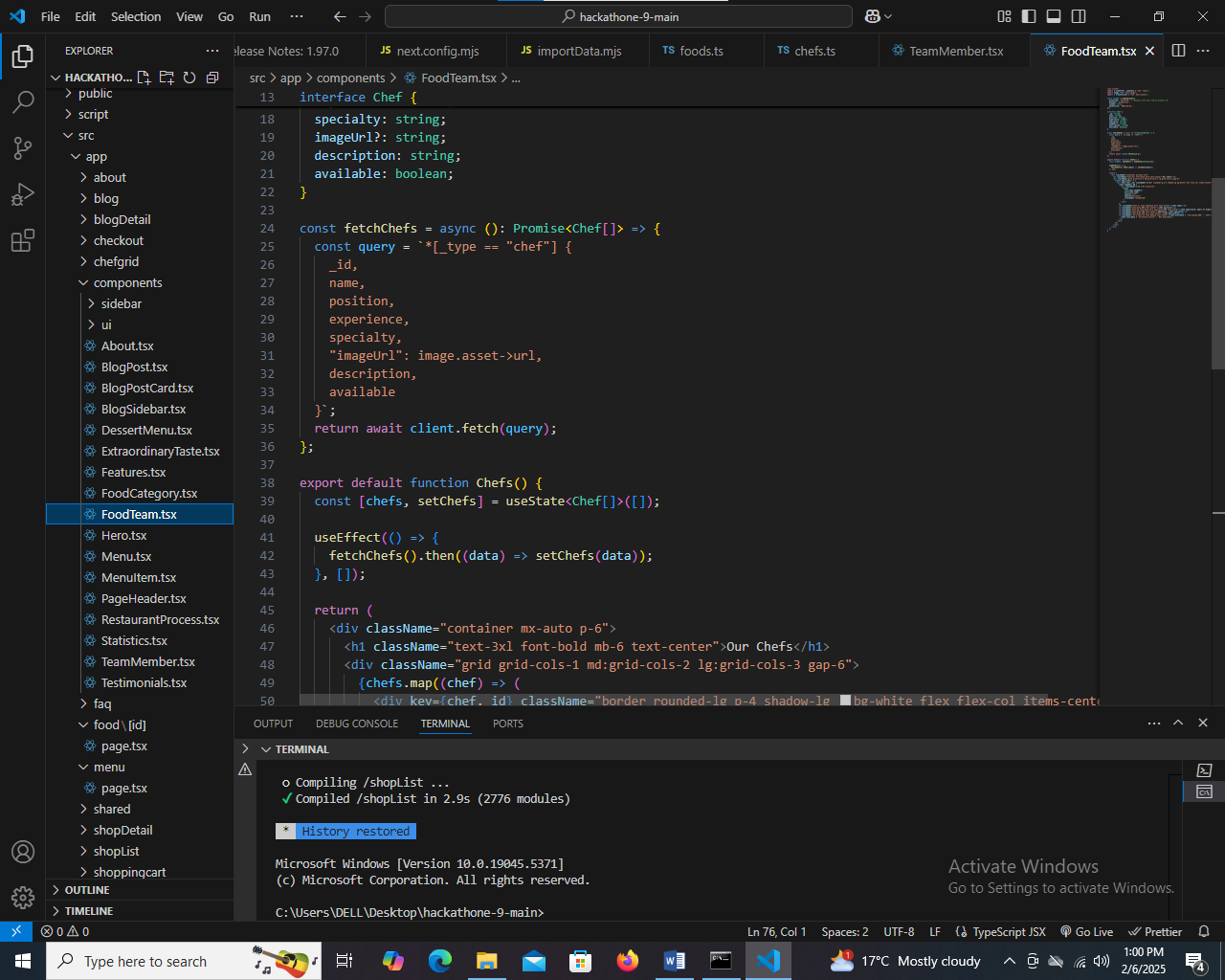
****

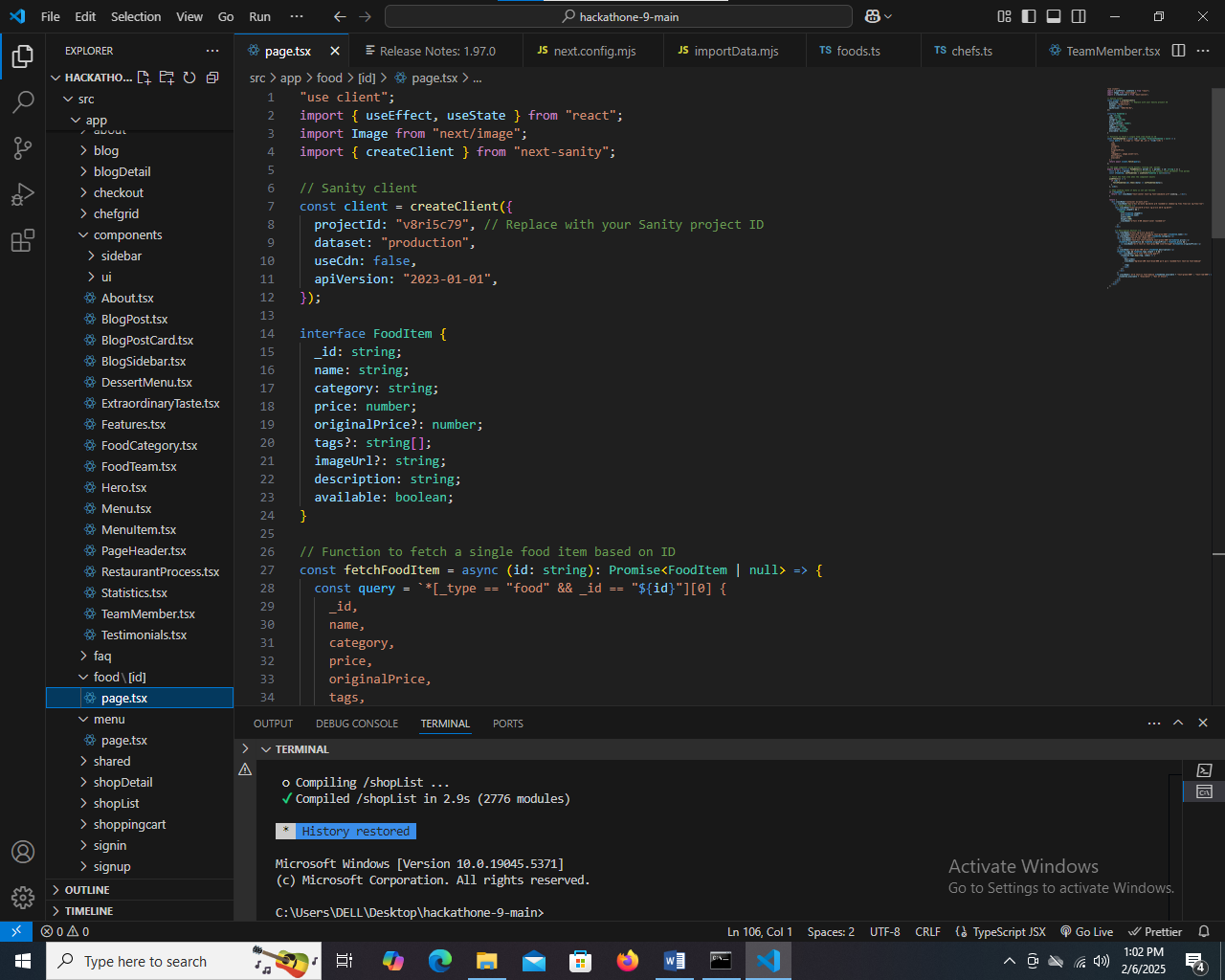
**Filter**

****

**Code**

****

****

****

**Steps Taken to Build the Components**

**1. Product Listing Component (Food Data Display)**

**Steps:**

* Created a React component to fetch and display a list of food items.
* Used useEffect to fetch data from an API or local JSON file.
* Implemented useState to manage food data state.
* Mapped over the data to render each food item in a grid or list format.
* Styled the component using Tailwind CSS for responsiveness.
* Added navigation links for users to view product details.

**2. Chef Data Component**

**Steps:**

* Created a separate React component for fetching and displaying chef details.
* Used API calls or a static JSON file to get chef information.
* Implemented a card-based UI with images, names, experience, and specialties.
* Ensured proper component reusability for different sections.

**3. Product Detail Page Component**

**Steps:**

* Created a dynamic route using React Router (/product/:id).
* Extracted the product ID using useParams().
* Fetched the specific product data based on the ID.
* Displayed product details including name, image, description, price, and reviews.
* Added an "Add to Cart" button and integrated state management.
* Ensured responsiveness and user-friendly design.

**Challenges Faced & Solutions Implemented**

| **Challenges** | **Solutions** |
| --- | --- |
| Handling API loading states and errors | Used useState for loading/error handling and displayed messages accordingly. |
|  |  |
| Ensuring dynamic routing works properly | Used useParams() from React Router and verified data fetching logic. |
| Styling consistency across components | Used Tailwind CSS utility classes for a unified design. |
|  |  |
| Handling missing or incorrect data | Implemented default fallback values and error boundaries. |

**Best Practices Followed**

✅ **Component Reusability**: Broke down UI into reusable components (e.g., FoodCard, ChefCard).  
✅ **State Management**: Used useState, useEffect, and context API for better data flow.  
✅ **Code Optimization**: Minimized unnecessary re-renders with React.memo().  
✅ **Error Handling**: Implemented proper error handling for API calls.  
✅ **Responsive UI**: Ensured mobile-first design with Tailwind CSS.  
✅ **SEO & Accessibility**: Used proper semantic HTML tags and alt attributes for images.