Medicare+

A Medical Store Front

Professor: Ramsey Muvva

INF651: Front End Web Development I

Dec 9, 2025

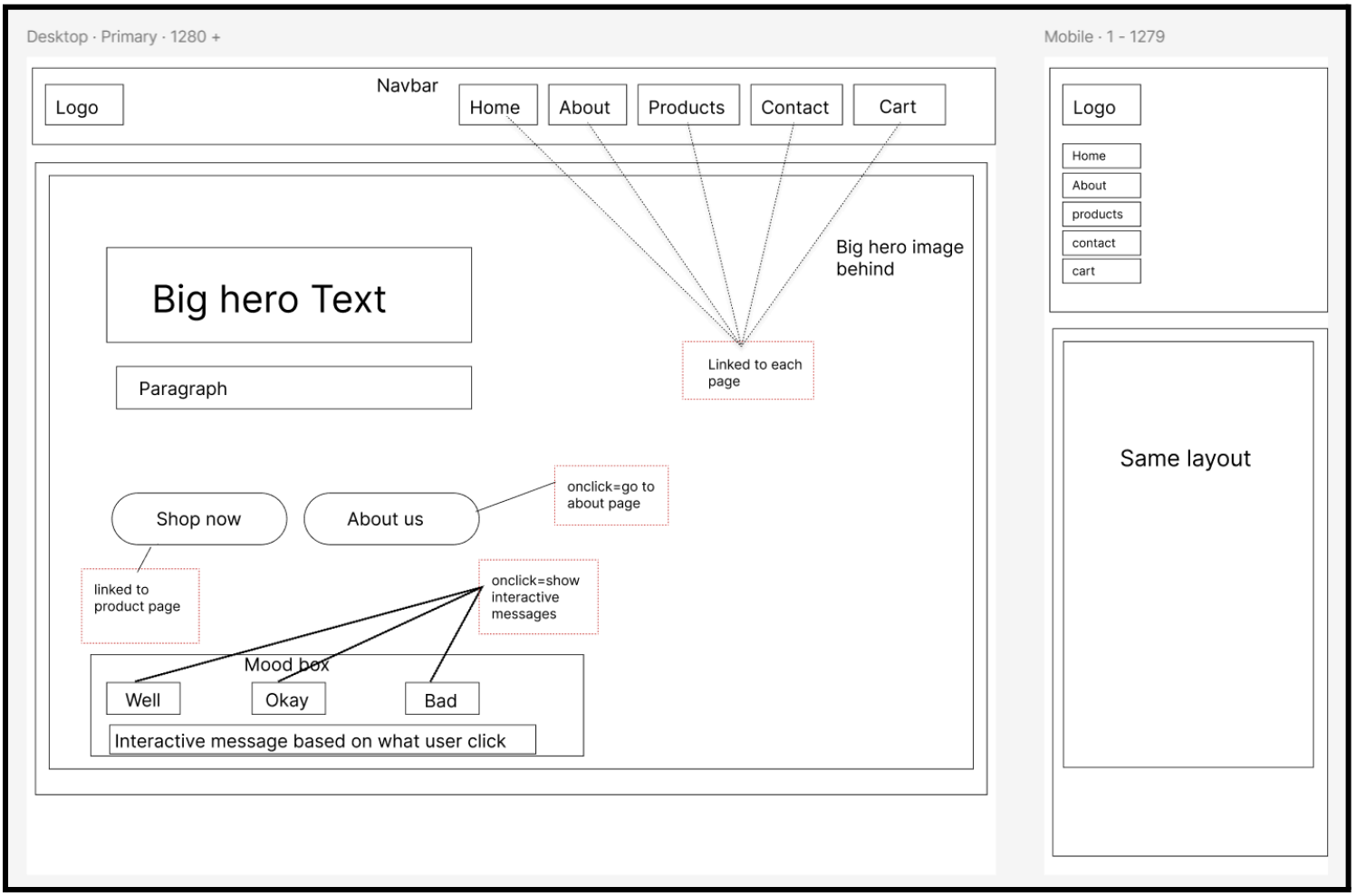
Student: Chivithrokal Chan

Table of Content

Wireframe design

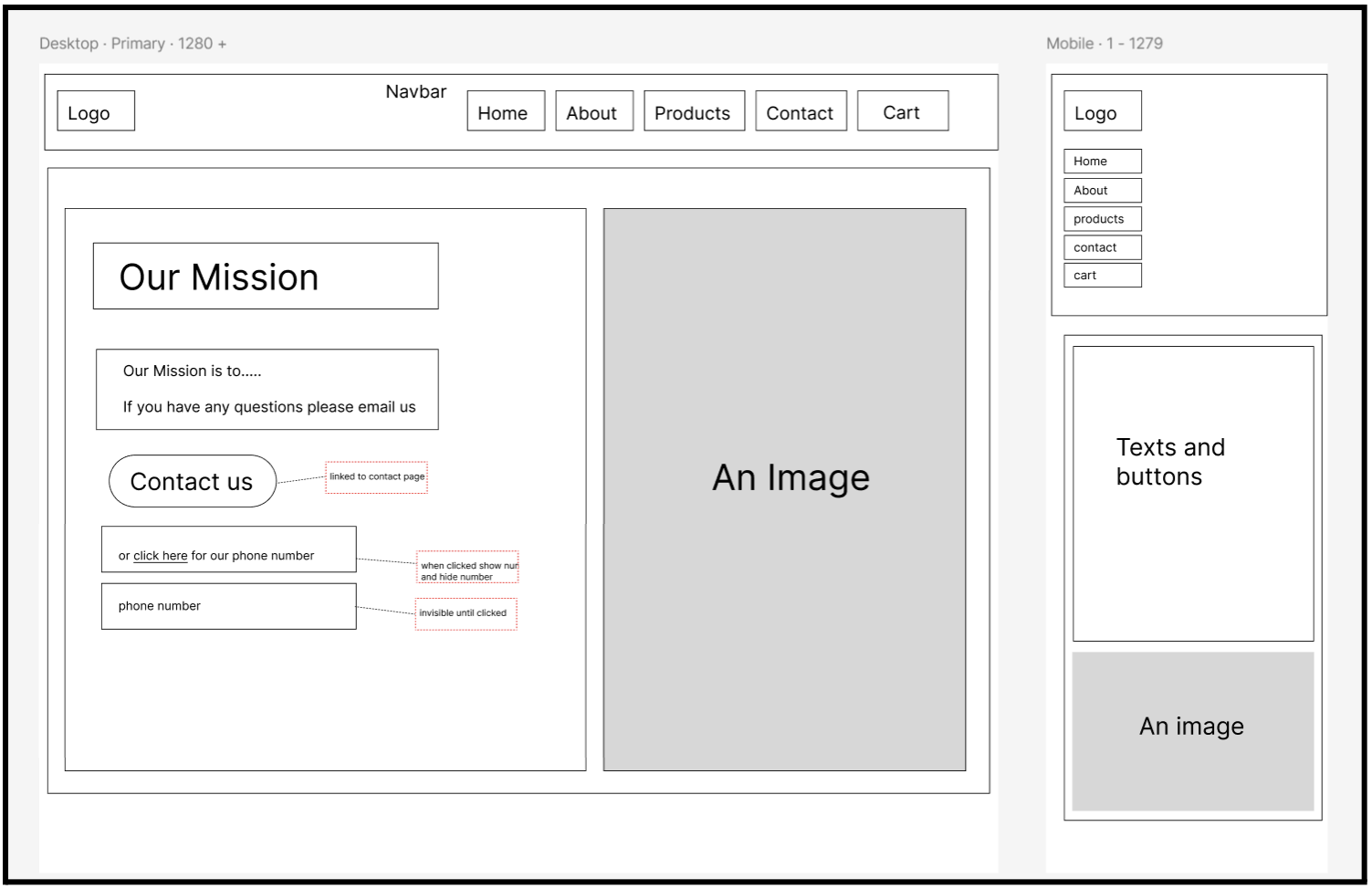
Index/Home page

On the left is the desktop view and the right is the mobile/small screen's view.



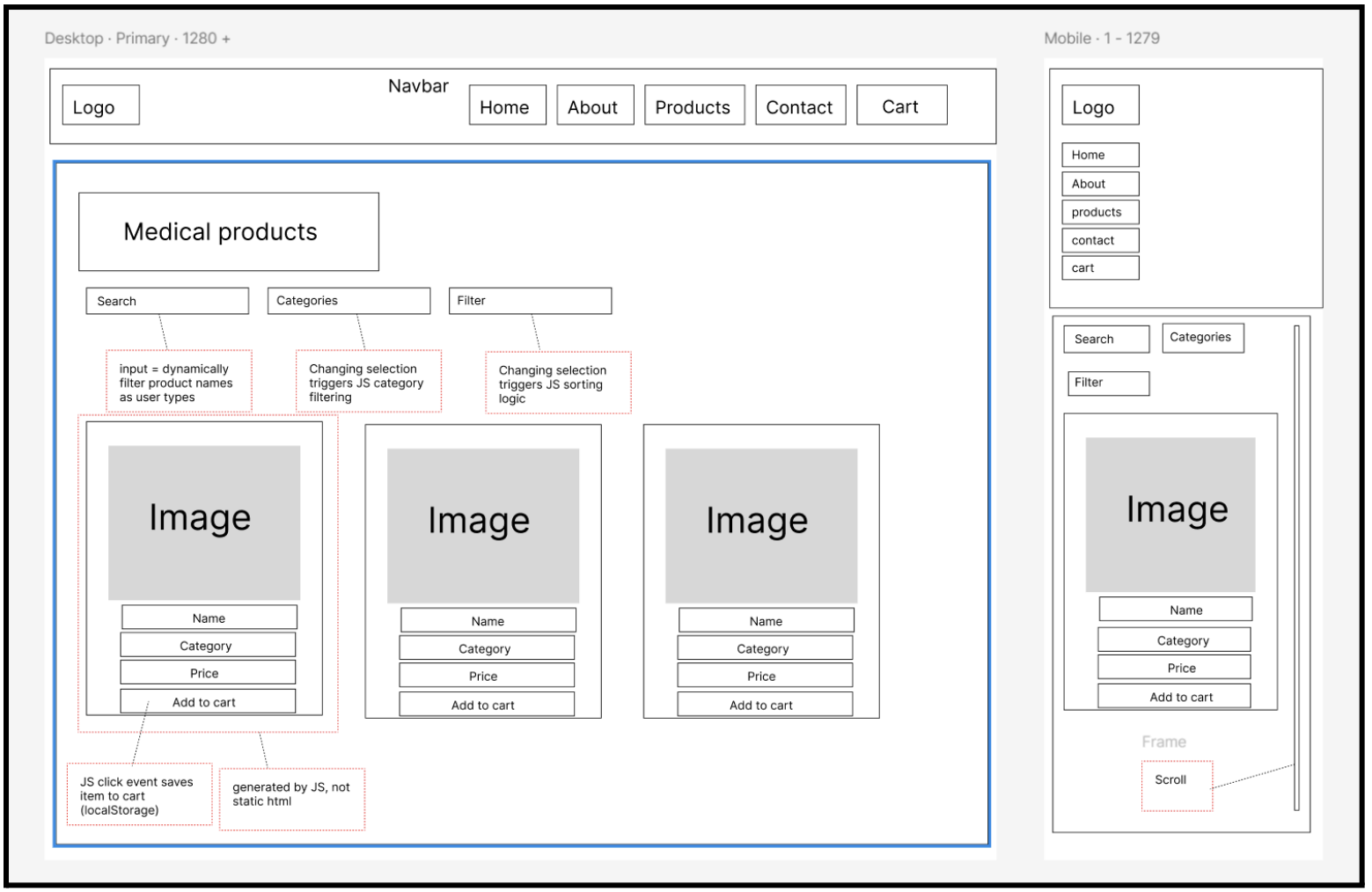
About Page

On the left is the desktop view and the right is the mobile/small screen's view.



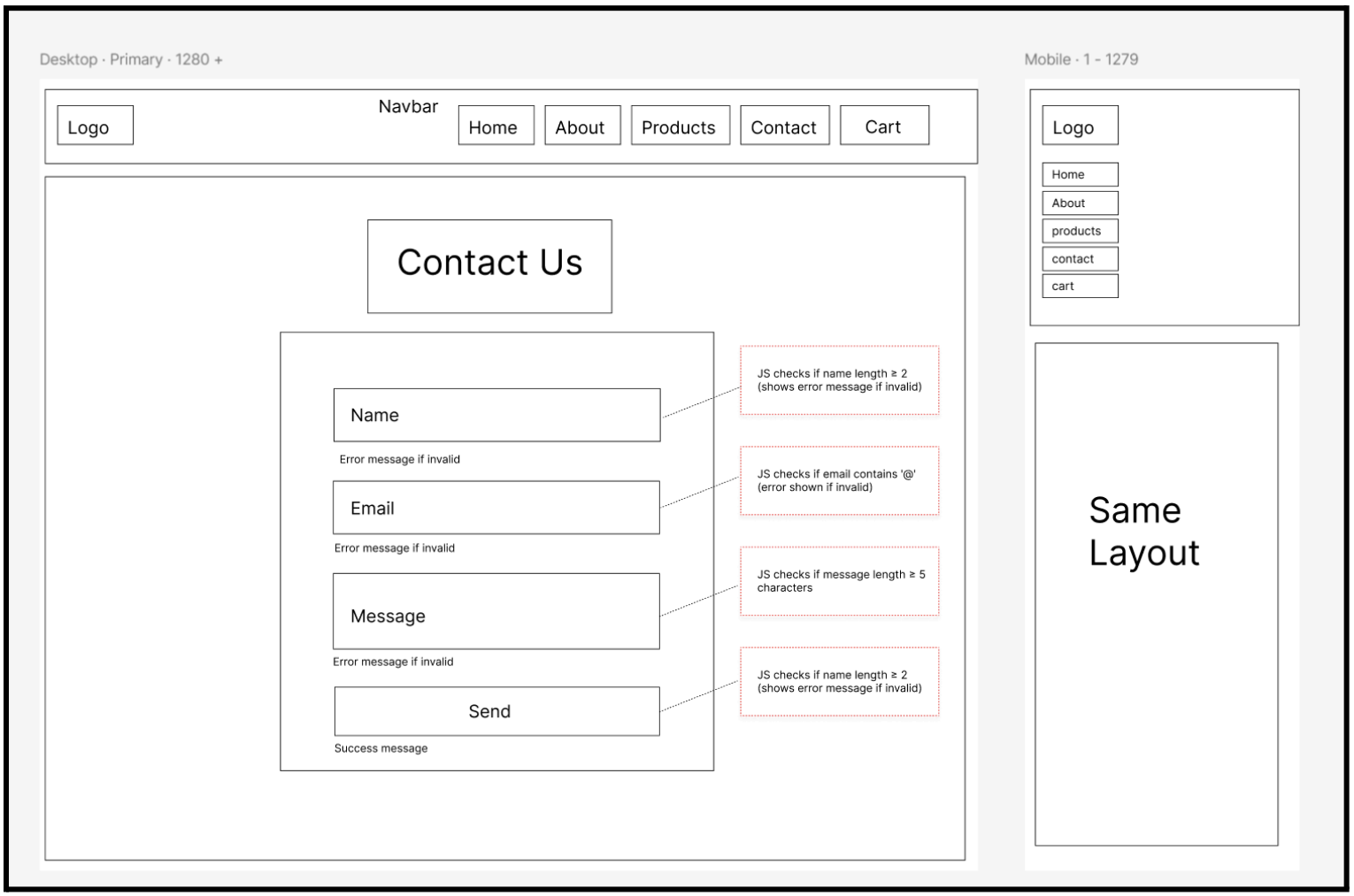
Products Page

On the left is the desktop view and the right is the mobile/small screen's view.



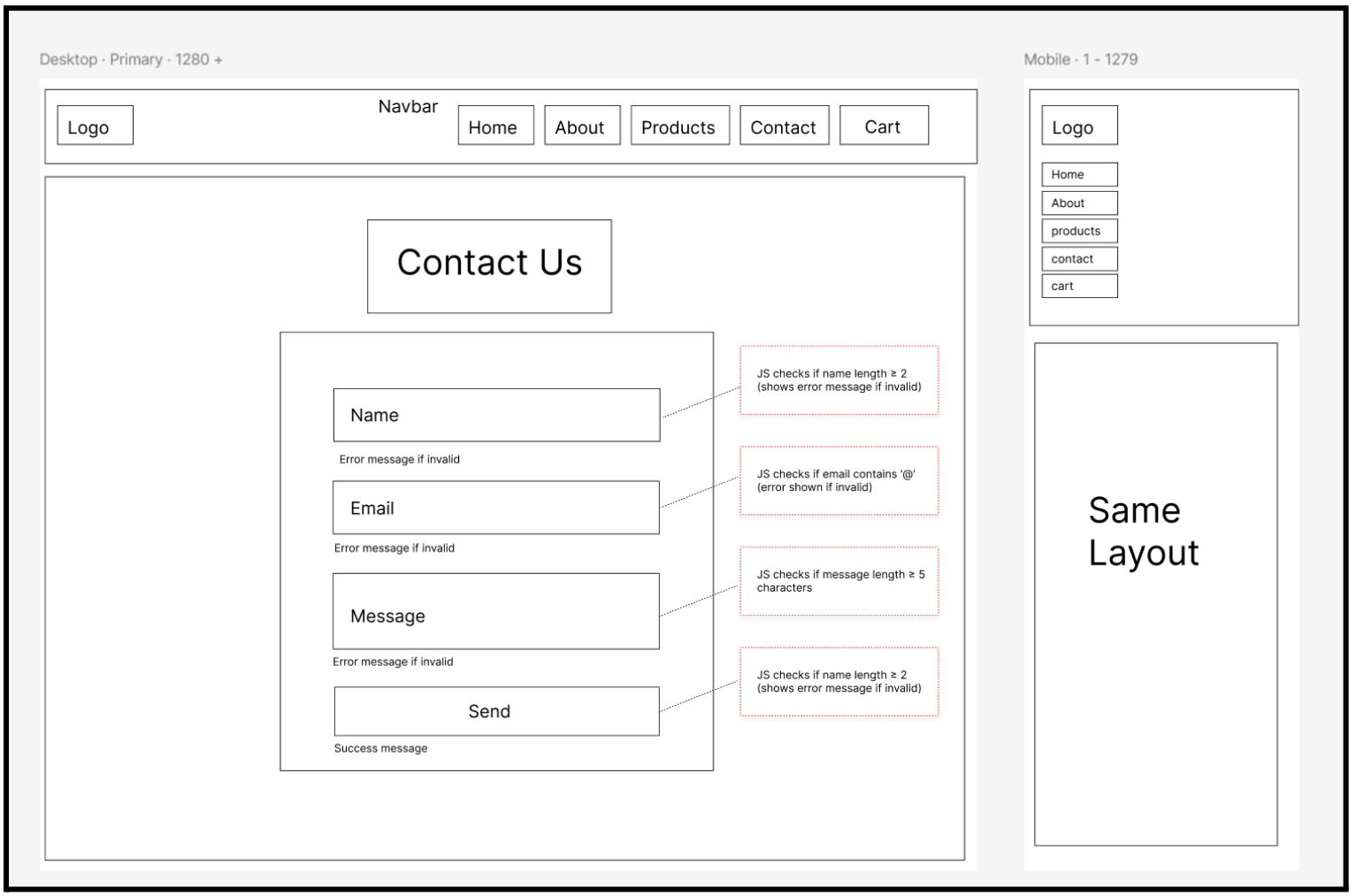
Contact Page

On the left is the desktop view and the right is the mobile/small screen's view.



Cart Page

On the left is the desktop view and the right is the mobile/small screen's view.



1. Features and Functionality

Medicare+ is a multi-page JavaScript-powered web application of an online storefront for medical supplies, wellness products, and small medical devices. The project combines minimalistic UI design with dynamic, interactive functionality. Here, users can browse products, filter and sort items, manage a shopping cart, read about the company, and submit inquiries through a validated contact form. Currently, the website has five pages.

Unlike a static website, Medicare+ enhances user experience by incorporating JavaScript-driven interactions, dynamic rendering, localStorage-based cart persistence, and real-time validation.

1.1. Home Page (index.html)

The homepage introduces users to the platform through a clean hero section and interactive mood selector.

Key Features:

- Hero banner with background overlay
- “SHOP NOW” button linking to product listings
- “ABOUT US” button redirect handled by JavaScript
- Mood Interaction Widget, allowing users to select how they feel and receive a personalized message.
- JavaScript dynamically updates the mood message using event listeners bound to three buttons. (“Good”, “Okay”, “Not Well”)

1.2. About Page (about.html)

The About page provides an overview of the company’s mission, a contact button, and a phone number that is only revealed when clicked on. This can also be hidden back by the user.

Key Features:

- A toggle button that reveals or hides the company phone number

- Clicking “click here” shows or hides the support phone number, and the link text updates accordingly. The reveal/hide action is handled via JS class toggling and text replacement.
- “Contact Us” button is linked to the contact page.

1.3. Products Page (products.html)

This is the most dynamic page in the application. All product cards are created and updated through JavaScript.

Key Features:

Dynamic Product Rendering

Products are defined in a JavaScript array of objects (script.js) with:

- name
- category
- price
- image path

Cards are generated entirely with JS using DOM manipulation.

Interactive Filters

Users can refine product results using:

- Search bar (real-time text filtering)
- Category filter (Pain Relief, Supplements, Allergy, etc.)
- Sort options (price ↑, price ↓, A–Z)

Each change triggers a full UI re-render using the renderProducts() function.

Add to Cart System

Each product card includes an "Add to Cart" button. JavaScript:

- Stores products in localStorage
- Increments quantity when the same item is added twice
- Displays confirmation via alert()

Cart updates persist even after refreshing the browser.

1.4. Contact Page (contact.html)

The Contact page includes a JavaScript-validated form.

Key Features:

Contains a user input form with fields for:

- Name
- Email
- Message

The form includes real-time JavaScript validation and error messages. If all fields are valid:

- A success message displays
- Form resets automatically

Validation checks include:

- Minimum name length
- Email containing "@"
- Message length requirements

1.5. Cart Page (cart.html)

The cart system uses **localStorage** to save items so they persist across sessions.

Key Features:

The cart page retrieves items from localStorage and dynamically displays:

- Product Name
- Price
- Quantity
- Remove Button
- Auto-calculated total price

Key functionality:

- Removing an item updates localStorage and re-renders the page
- Empty cart state includes a clean message ("Your cart is empty.")

The cart system functions completely without a backend, using the browser as storage.

2. How JavaScript Enhances the User Experience

JavaScript provides the core interactivity across all pages, transforming static HTML into a dynamic shopping experience.

2.1. Dynamic Content Rendering

The Products page uses JavaScript to build product cards entirely in the browser. This enables:

- Instant re-rendering when users filter or sort
- No duplicate HTML code
- Faster and smoother browsing
- Easy scalability by adding items to the product array

2.2. Real-Time Search, Filtering, and Sorting

Every user action including typing in search, changing category, adjusting sort all updates the product grid instantly. This creates a fast, intuitive browsing experience similar to modern online stores.

2.3. Client-Side Cart Persistence

JavaScript's use of localStorage enables:

- Cart saving across page reloads
- Cart restoration when user returns
- Consistent experience without needing a backend
- This improves usability and makes the site feel more professional.

2.4. Form Validation and User Feedback

JavaScript enhances the Contact page with:

- Real-time validation
- Friendly inline error messages
- Preventing incomplete or incorrect submissions

- Success confirmation and automatic form reset

Users are guided clearly, reducing frustration.

2.5. Event-Driven User Interactions

JavaScript listens to various events such as:

- Button clicks
- Input changes
- Form submissions
- Page load events

This allows seamless, responsive interactivity throughout the website.

3. Challenges Faced and Solutions

One of the biggest challenges I faced was figuring out how to store the users' cart data. I don't have much experience working with databases yet, so I needed a way for the cart to "remember" items even after the user switched pages or refreshed the browser. At first, I wasn't sure how to approach this because everything in my project is entirely frontend, no backend, no database.

Solution:

I decided to use localStorage as a simple, temporary storage system for the cart. Instead of relying on a full database, I stored all cart items inside an array and saved that array as a JSON string in localStorage.

Here's how it works now:

- When a user adds a product, JavaScript checks if that product already exists in the saved cart.
- If it does, the quantity increases, if not, a new entry is added.
- After every update, the cart array is saved back into localStorage.
- When the cart page loads, the app reads from localStorage to rebuild the cart UI.

This let me create a small but functional e-commerce cart system without needing any backend experience, and it helped me understand how data persistence can still work in a frontend-only project.

4. Plans for Additional Features & Backend Integration

4.1 Planned Additional Features

4.1.1 Improved Cart Functionality

- Quantity increase/decrease buttons
- Subtotal calculation per item
- Styled toast notifications instead of alerts

4.1.2 Product Detail Pages

Each product could have its own dedicated page with:

- Larger images
- Descriptions
- Additional metadata

4.1.3 Wishlist or Favorites System

Users could mark products to review later, saved to localStorage.

4.1.4 User Accounts (requires backend)

- Sign in / Sign up
- Saved carts
- Purchase history

4.2 Backend Integration Plans

If expanded to a full application, backend features may include:

4.2.1 Database-Backed Product Management

Admins could add/edit/remove products from a database.

4.2.2 Secure Cart & Checkout System

- Server-side cart storage
- Stripe or PayPal integration

- Order confirmation system

4.2.3 Email Handler for Contact Form

- Instead of staying on the frontend, valid contact messages could:
- Send directly to company email
- Save into a support ticket system

4.2.4 Inventory & Stock Tracking

- Prevent users from ordering unavailable items.

Conclusion

The Medicare+ application shows proficiency in JavaScript, DOM manipulation, form validation, event-driven programming, and client-side state management. It successfully creates an interactive, user-friendly storefront experience while remaining fully frontend-based.

The clear architecture and modular functions provide a strong base for expanding the system into a full-stack, production-ready project with backend integration in the future.