**TERRA Plant E-commerce Website**

**Project Overview**

TERRA is a beautifully designed, fully-functional online store built to provide plant enthusiasts with a seamless shopping experience. The platform features a clean and intuitive user interface, a complete e-commerce workflow from product discovery to checkout, and innovative features like AI-powered plant recommendations and personalized consultation bookings. The entire user journey is designed to be interactive and engaging, utilizing modals and dynamic UI updates to feel like a modern, single-page application.

**Design Choices & Technology Stack**

The project was built using foundational web technologies to demonstrate strong core development skills while delivering a high-performance user experience.

* **Structure: HTML5**
  + **Reasoning:** The application is built on a foundation of semantic HTML5, using tags like <header>, <main>, and <section> to create a well-organized, accessible, and SEO-friendly structure for all pages.
* **Styling: Tailwind CSS**
  + **Reasoning:** Tailwind CSS was chosen for its **utility-first methodology**. This allowed for the rapid development of a custom, highly responsive, and visually consistent design system without writing a single line of conventional CSS. The clean product cards, smooth modal pop-ups, and consistent green-themed branding were all achieved efficiently with Tailwind's utility classes.
* **Interactivity & Logic: JavaScript (Vanilla JS)**
  + **Reasoning:** All dynamic functionality, from the interactive shopping cart to the AI recommendations, was built using plain **vanilla JavaScript**. This choice demonstrates a deep understanding of the DOM and core programming principles, resulting in a fast, lightweight application with no external framework dependencies.

**Key Features & Implementation**

1. **Dynamic Product Catalog with Filtering & Sorting**
   * **Feature:** Users can browse product categories and instantly filter the results using a search bar or sort them by price and name.
   * **Implementation:** This was achieved entirely on the client-side with JavaScript. An event listener on the search input and sort dropdown triggers a function that iterates through the product data. It then manipulates the DOM to hide non-matching items and re-orders the product elements in the grid according to the user's selection, providing instant feedback.
2. **AI-Powered Plant Recommendations**
   * **Feature:** A unique "Need Help Choosing a Plant?" section allows users to describe their needs in natural language and receive AI-generated plant suggestions.
   * **Implementation:** This feature uses the JavaScript **fetch API**. When a user submits their query, an asynchronous request is sent to a backend endpoint or a third-party AI service (like OpenAI). The JavaScript code then parses the JSON response and dynamically injects the AI's recommendations into a modal pop-up for the user to view.
3. **Complete E-commerce Functionality (Wishlist, Cart, Checkout)**
   * **Feature:** The site includes a full e-commerce experience: adding/removing items from a wishlist and cart, viewing totals, and a multi-step checkout process, all handled smoothly within modals.
   * **Implementation:** This complex stateful system was managed in vanilla JavaScript by leveraging the browser's **localStorage**. The contents of the cart and wishlist are stored as JSON arrays in localStorage, which ensures the data persists even if the user refreshes the page. A suite of JavaScript functions (addToCart, removeFromWishlist, etc.) was created to manage this data, and a central updateUI() function is called after any change to re-render the cart/wishlist modals and update the counters in the header.
4. **Order Tracking System**
   * **Feature:** Users can view their order history and track the status of any given order through a clean, visual timeline.
   * **Implementation:** The order status tracker is a great example of dynamic styling with JavaScript. The timeline component is built with HTML and Tailwind CSS. Based on the order's status data, JavaScript dynamically adds specific Tailwind classes (e.g., bg-green-500, border-green-500) to the appropriate steps in the timeline to visually represent the order's progress from "Placed" to "Delivered".

**Challenges & Solutions**

1. **Challenge: Managing E-commerce State without a Framework**
   * **Problem:** The most significant challenge was managing the complex state of the shopping cart and wishlist (adding items, updating quantities, calculating totals) across the application without the help of a framework like React or Vue.
   * **Solution:** I designed a mini "state management" system using **localStorage as the single source of truth**. I created a global JavaScript object to hold the current cart and wishlist arrays. Any function that modified this data (e.g., addToCart) would first update the object, then immediately save the updated object back to localStorage. A single, powerful render function was responsible for reading this data and updating every relevant part of the DOM, ensuring the UI was always in sync with the stored data.
2. **Challenge: Building a Seamless Modal-Based User Flow**
   * **Problem:** The entire user journey—from adding an item to the cart to placing an order and tracking it—happens within modals. Managing the opening, closing, and content-switching of these modals without creating messy code was difficult.
   * **Solution:** I implemented a centralized modal manager in JavaScript. A single, full-screen overlay was used. When a modal was triggered, its specific HTML content was made visible within the overlay. I used **event delegation** to handle all close events efficiently, listening for clicks on a .close-button class or on the overlay background itself. This approach kept the code clean and provided a smooth, app-like experience for the user.