datefix

Build Guide

How the DateFix Demo Was Built — A Complete Breakdown

For Emily's Portfolio & Learning Reference

This document walks through every step of building the DateFix interactive product demo — from generating the initial images with AI, to extracting and masking assets in Photoshop, to assembling everything into an interactive web experience with Claude Code. It covers the full creative and technical pipeline so you can understand and reproduce the process.

What's Inside

1. Overview

The DateFix Demo is an interactive web page that showcases the DateFix product line — a series of date-based fruit gel pouches in four flavors: Original, Ginger, Turmeric, and Cinnamon. The tagline is **"Fresh Fruit Made To Go"** and each pouch contains dates blended with orange blossom water.

The goal was to create a polished, portfolio-quality brand demo that shows off all four flavors with real product photography aesthetics, interactive hover states, and informational popups — all built entirely from AI-generated imagery and assembled through a combination of Photoshop, code, and Claude AI assistance.

The Four Flavors

Original

Ginger

Turmeric

Cinnamon

2. Image Generation

The entire project started with AI image generation using **Nano Banana**. This tool was used to create the original product photography-style images of the DateFix pouches.

Step 1 — Portrait Generation (1080×1920)

The first generation was at **1080 × 1920 portrait** orientation. This produced a tall image showing all four pouches with their ingredients. Useful as a starting reference but not ideal for the web layout.

Portrait generation (1080×1920) — initial source

Step 2 — Landscape Re-generation

The image was then **regenerated in horizontal / landscape** to better suit a web UI. This wider composition became the primary base for extracting all individual assets.

Landscape re-generation — this became the base for all asset extraction

3. Asset Extraction in Photoshop

Each element — every packet and every ingredient — needed to be isolated on its own transparent layer so it could be positioned independently in the web demo. This was the bulk of the manual work.

Masking Techniques Used

1. Manual Selection

For areas with clean, well-defined edges, manual selection tools (Pen tool, Lasso) were used to cut elements out. Most precise but most time-consuming.

2. Select Color Range

Photoshop's **Select** → **Color Range** was used to quickly select the white background behind packets and ingredients. Click a color, adjust the Fuzziness slider to control range. Especially effective because the product shots had clean white backgrounds.

3. Blend If (Blending Options)

The key technique for **perfect masks**. In **Layer Style** → **Blending Options**, the "Blend If" sliders let you make white pixels disappear non-destructively. Hold **Alt/Option** and drag one half of the white slider to split it, creating a smooth feathered transition. Combined with Select Color Range, this gave near-perfect cutouts.

What Was Extracted

Asset	Description	Notes
Packets (×4)	Each flavor pouch on transparent bg	Original, Ginger, Turmeric, Cinnamon
Dates	Two split dates (for Original)	Bottom layer + top overlap layer
Ginger	Ginger root (for Ginger)	Single layer, no overlap
Turmeric	Turmeric root with powder	Single layer, no overlap
Cinnamon	Cinnamon sticks with powder	Bottom layer + top overlap layer

4. Asset Preparation

Consistent Canvas Sizing

This step is **critical** and easy to overlook. Every single asset — all four packets and all the ingredient layers — needed to be exported at the **exact same canvas size and aspect ratio**.

Why? Because in the web demo, each image is positioned and scaled relative to a common coordinate system. If one packet image is 500×800 and another is 600×900, they won't align properly even if you scale them to the same display size — the content will be offset within the canvas. By keeping all canvases identical, when you overlay them in CSS at the same position, everything lines up pixel-perfectly.

The Overlap Layer Trick

For the **Dates** and **Cinnamon** flavors, the ingredient partially overlaps the packet in the original photo. To recreate this depth in the web demo, each was split into **two separate layers**:

Bottom layer: The full ingredient, positioned *behind* the packet in CSS (lower z-index). This is the main body that sits below the pouch.

Top layer: Just the overlapping portion, cut out onto its own layer, positioned *in front of* the packet (higher z-index). This creates the illusion that the ingredient wraps around the packet.

> *Both layers share the same canvas size and alignment, so when stacked in the browser with the packet sandwiched between them, the depth illusion is seamless.*

Top Overlap Layers

Top date — overlaps in front of packet

Top cinnamon — overlaps in front of packet

5. Project Setup

Folder Structure & GitHub

A new folder called **datefix-demo** was created and connected to a GitHub repository. All the extracted PNG assets were placed inside an assets folder within the project. This makes them accessible to the web code and version-controlled.

Briefing Claude Code

The project was built with **Claude Code** (Anthropic's AI coding assistant). The process started by giving Claude a detailed brief — uploading all the assets and describing the desired layout, interactions, and visual style. A planning conversation defined the project scope.

Brand Assets Collected

Logo: A high-resolution screenshot of the DateFix logo was taken directly from the official site, then cropped and cleaned up for use in the demo header.

Brand Colors: Each packet's signature color was sampled using the eyedropper tool, giving exact hex values for each flavor. These colors were used throughout the UI for hover states, shadows, and flavor popups.

Flavor	Color	Used For
Original	Light Blue (#89B4D4)	Packet accent, shadow, popup
Ginger	Green (#8BC34A)	Packet accent, shadow, popup
Turmeric	Orange (#F5A623)	Packet accent, shadow, popup
Cinnamon	Pink (#D86DA8)	Packet accent, shadow, popup

6. Building the UI

The longest phase — taking all the individual assets and assembling them into a cohesive, interactive web page.

Layer Structure in CSS

The key architectural decision was recreating Photoshop's layer system in the browser. Each flavor "slot" consists of multiple stacked elements:

Z-Order	Layer	Purpose
4 (front)	Top ingredient	Overlap portion (dates & cinnamon only)
3	Packet	The actual product pouch image
2	Bottom ingredient	Main ingredient behind the packet
1 (back)	Color overlay	Tinted duplicate for glow/shadow on hover

Alignment Challenges

Getting everything to line up correctly was one of the most time-consuming parts. Because each asset is a separate image being positioned with CSS, even small misalignments between the packet and its ingredient layers become immediately visible. This led to building custom debug tools (Section 7) for real-time visual adjustment.

Reference Image Overlay

The original horizontal AI-generated image was placed as a **semi-transparent overlay** on the web page. At ~30% opacity, it served as a visual guide — like tracing paper. Individual layers could be adjusted until they matched the reference positions exactly. The debug menu controlled its opacity, size, and position.

7. The Custom Debug Menu

One of the most interesting parts of this build — a custom debug interface accessible by pressing the **equals key (=)** on the DateFix demo site.

The custom debug menu — layout controls, shadow settings, and per-flavor adjustments

Layout Tab — Controls for overall packet spacing and global positioning.

Reference Image Controls — Toggle the overlay on/off with sliders for opacity, scale, and vertical offset. Essential during alignment: turn on the reference, lower opacity, tweak each element to match.

Per-Flavor Controls — Each flavor has color-coded sliders for Scale, X position, and Y position of its ingredient layer relative to the packet.

Shadows Tab — Modeled after Photoshop's Drop Shadow dialog. Adjust offset, blur, spread, and color in real-time. Much faster than hand-writing CSS box-shadow values.

Copy Config Button — The key workflow tool. Once positioning and shadows looked right visually, this button copies all configuration values to the clipboard. That config was then fed back into Claude Code to hard-code the final values. This loop — *visual tweaking → copy config → paste into code* — was the core iteration process.

8. Photoshop-Style Transform Controls

At a certain point, the slider-based debug menu wasn't precise enough. So **Photoshop-style transform controls** were built directly into the web page — the familiar bounding-box handles you see when you select a layer in Photoshop.

Each packet and ingredient layer got its own set of transform handles. Click any element to select it (showing its bounding box), then drag corners to scale or drag the center to reposition. This made it dramatically faster to get precise placement, especially for ingredients that needed to sit at very specific positions relative to their packets.

Like the shadow controls, these transforms were exportable — final positions and scales could be copied and hard-coded into production.

> *This is a great example of building custom tooling to solve a specific problem. Rather than fighting with CSS values in code, you build a visual tool that lets you work the way you naturally think (dragging and resizing), then export the result back into code.*

9. Interactivity

Hover Effect — Packet Rise

When you hover over any packet, it **rises up** with a smooth CSS transition. The transform applies to the entire packet group (image + color overlay + ingredient layers) so everything moves together, giving a tactile, app-like feel.

Color Overlay Glow

Behind each packet sits a **color overlay duplicate** — a copy tinted entirely in the flavor's brand color. This acts as a colored glow/shadow. On hover it becomes more visible, enhancing the lifted, glowing effect.

Flavor Popup / Info Card

When you **click** on a packet, a small informational popup appears displaying the flavor name, the product description ("dates blended with orange blossom water"), and a brief flavor-specific description. Styled to match the brand aesthetic, it adds interactivity that makes the demo feel like a real product page.

Reactive Drop Shadows

The drop shadows weren't just static CSS — they were carefully tuned using the Photoshop-inspired shadow debug panel. Each packet has custom shadow values (offset, blur, spread, color) that were visually dialed in and then hard-coded. Shadows respond to hover state, enhancing the lift effect.

10. Summary — The Full Pipeline

1. Generate — Use Nano Banana to generate product photography images at 1080×1920 (portrait), then regenerate in landscape for the web layout base.

2. Extract — Open in Photoshop. Use manual selection, Select Color Range, and Blend If to mask each element onto its own transparent layer.

3. Split Overlaps — For ingredients that overlap packets (dates, cinnamon), cut the overlapping portion onto a separate "top" layer aligned to the same canvas.

4. Standardize — Make every asset the exact same canvas size and aspect ratio. Export all as PNGs with transparency.

5. Collect Brand — Screenshot the logo from the official site. Color-pick each packet's brand color with the eyedropper.

6. Set Up Project — Create folder, connect to GitHub, drop in all assets. Brief Claude Code with the full visual and interaction spec.

7. Build Layers — Assemble in HTML/CSS: color overlay → bottom ingredient → packet → top ingredient, using z-index for depth.

8. Build Debug Tools — Create the custom debug menu (= key) with layout sliders, shadow controls, reference overlay, and per-flavor adjustments.

9. Align Visually — Use the reference overlay at low opacity as a guide. Adjust each element with debug sliders and transform handles.

10. Copy Config — Use Copy Config to export all visual settings, feed back into Claude Code to hard-code into source.

11. Add Interactivity — Implement hover rise, color glow, click-to-open flavor popups, and reactive drop shadows.

12. Polish & Ship — Remove debug tools from production, final QA, push to GitHub.

Key Takeaways

AI as starting point, not final product. The AI-generated images were a foundation — they still required significant Photoshop work to extract, mask, and prepare individual assets.

Build your own tools. The debug menu and transform controls were custom-built for this specific project. When standard tools aren't enough, building your own workflow tools saves enormous time.

Visual iteration → code export. The Copy Config workflow (visually adjust → export values → hard-code) is a powerful pattern. Work visually where your eyes judge quality, then programmatically lock in results.

Layer thinking translates. Photoshop's mental model of layers and z-index maps directly to CSS stacking. Understanding one helps you master the other.

Consistent canvas sizes matter. Making every asset the same dimensions saved hours of alignment headaches. Always think about how assets will be composited together.