On The Ground.

ON THE GROUND · CLAUDE CODE WORKSHOPS

How to Claude Code a web app or webpage (with GitHub and Vercel!)

The complete start-to-finish path from a blank folder to a live website: install Claude Code, connect GitHub and Vercel, write your first prompt, fix errors, and ship changes.

Authored by Haojun See · AI-assisted drafting: Claude

Updated 6 August 2026

ontheground.agency/resources/how-to-claude-code-a-web-app

Contents

Overview 3

What you'll need before you start 3

Step 1 — Install Claude Code 3

Step 2 — Create your GitHub account 4

Step 3 — Create your Vercel account and link it to GitHub 4

Step 4 — Your first prompt: build something real 4

Step 5 — Push it to GitHub and go live on Vercel 5

Step 6 — Making changes 5

Step 7 — Opening the developer console when something looks wrong 5

Step 8 — Setting environment variables 6

Step 9 — Reading Vercel's logs when a deploy breaks 6

Step 10 — Leveling up the design 6

The one-page cheat sheet 7

How this guide was made 8

Overview

This is the guide version of the training session — every step from "I have nothing" to "I have a live website with my own domain-ready URL", written down so you don't have to remember it.

It covers five tools working together: **Claude Code** (writes and edits your website), **the terminal** (where you talk to Claude Code), **GitHub** (stores your code and its full history), **Vercel** (hosts your site and makes it live on the internet), and **your browser's developer console** (shows you what's actually broken when something looks wrong).

You don't need to know how to code to follow this. You do need to be comfortable copying and pasting, and okay with things breaking a little before they work — that's normal, and Step 6 is entirely about what to do when they do.

What you'll need before you start

- A computer (Mac or Windows) with an internet connection.
- A free GitHub account.
- A free Vercel account.
- A Claude.ai account with access to Claude Code (Pro or Max plans include it; there's also a pay-as-you-go API option).
- Node.js installed — download the **LTS** version, and click through the installer with the defaults.

That's it. No design software, no separate code editor required (though installing VS Code later makes reading files easier — optional, not required for this guide).

Step 1 — Install Claude Code

Claude Code is a command-line assistant: you type what you want in plain English, inside a **terminal**, and it writes, edits, and runs the actual code for you in a folder on your computer.

- **On Mac**: open **Terminal** (press Cmd+Space, type "Terminal", hit Enter).
- **On Windows**: open **PowerShell** or **Windows Terminal**.

With Node.js already installed (see above), type this command and press Enter:

```
npm install -g @anthropic-ai/claude-code
```

Next, create a folder for your project and move into it:

`mkdir my-first-site` then `cd my-first-site`

Now start Claude Code:

```
claude
```

The first time you run this, it opens your browser and asks you to log in with your Claude.ai account. Approve it, come back to the terminal — you're in. Everything you type from this point is a message

to **Claude**, not a regular terminal command.

Step 2 — Create your GitHub account

GitHub is where your code lives permanently, with a full history of every change you ever make — think Google Drive for code, with an undo button that goes back years.

Go to github.com and sign up for a free account if you don't have one. You don't need to create a repository yet — Claude Code will do that for you once there's actually something to save. Just have the account ready.

Step 3 — Create your Vercel account and link it to GitHub

Vercel is what makes your site visible on the internet, at a real URL anyone can open. This is also the "linking together" step: Vercel and GitHub connect at the account level, once, and after that every project you push to GitHub can be deployed with a couple of clicks.

- Go to vercel.com and click **Sign Up**.
- Choose **Continue with GitHub**. This is deliberate — it authorizes Vercel to see your GitHub repositories, which is what makes deploying later a one-click action instead of a manual upload.
- Approve the GitHub permission screen that pops up.

You now have three accounts that all know about each other: Claude.ai (via Claude Code, on your machine), GitHub (storage), and Vercel (hosting, linked to GitHub). Nothing is live yet — that happens once there's a project to deploy.

Step 4 — Your first prompt: build something real

Back in your terminal, inside Claude Code, describe what you want in plain language. Be specific about what the page is for — Claude Code fills in the rest.

A good first prompt for a single page:

"Build me a single-page website for a small bakery called Warm Loaf. It needs a hero section with the bakery name and a short tagline, a section showing three signature items with names and short descriptions, an About section, and a contact section with an address and opening hours. Use a warm, simple design. Make it a static HTML page with all CSS included, no build tools."

Claude Code will explain what it's about to do, then create the files. When it's done, ask it: "Open this in my browser so I can see it" — or simply open the `index.html` file it created by double-clicking it in Finder/Explorer.

If you'd rather build a full multi-page site (not just one HTML file), say so up front — e.g. "Scaffold this as a Next.js project" — and Claude Code will set up a proper project structure instead of a single file. Either is fine for what follows.

Step 5 — Push it to GitHub and go live on Vercel

Once you're happy with the first version, tell Claude Code:

"Push this to a new GitHub repository called warm-loaf-site."

Claude Code runs the git commands for you (`git init`, `git add`, `git commit`, and creating + pushing to the GitHub repo) and gives you the repository URL when it's done. If it asks you to authenticate with GitHub in your browser, approve it — that's normal for the first push.

Now make it live:

- Go to vercel.com/new.
- Find your new repository in the list and click **Import**.
- Leave the settings on their defaults (Vercel usually detects the right framework automatically).
- Click **Deploy**.

In under a minute, Vercel gives you a live URL — something like `warm-loaf-site.vercel.app`. Open it. That's your site, on the internet, for real.

Step 6 — Making changes

This is the part you'll do over and over: you look at the live site, decide what to change, and tell Claude Code in plain English.

"Make the hero section text bigger and center it." "Change the accent color from blue to a warm orange." "Add a fourth item to the signature items section called Sourdough Batard."

Claude Code edits the files and shows you a **diff** — a before/after of exactly what changed, with removed lines in red and added lines in green. Read it before accepting; it's the fastest way to learn what's actually happening in your own project.

When you're happy, ask Claude Code to "commit and push this" — that one instruction saves the change to GitHub *and* triggers Vercel to automatically rebuild and redeploy your live site. No extra steps. This GitHub-to-Vercel connection from Step 3 is doing its job quietly in the background from here on.

Step 7 — Opening the developer console when something looks wro

Sometimes a change doesn't look right — a button doesn't work, a section is blank, an image won't load. Before you guess, look at what your browser actually says is wrong.

- **Open the console**: right-click anywhere on the page and choose **Inspect**, then click the **Console** tab. (Shortcut: Cmd+Option+J on Mac, Ctrl+Shift+J on Windows/Chrome.)
- Red text in the console is an error. It usually names the exact file and line.
- Also check the **Network** tab if something isn't loading — a request shown in red with a status like 404 (not found) or 500 (server error) tells you what failed.

You don't need to understand the error. **Copy the exact red text and paste it straight into Claude**

Code , with something like: "I'm seeing this error in the browser console, can you fix it:" followed by the pasted text. This is the single most useful habit in this whole guide — it turns a vague "it's broken" into something Claude Code can act on directly.

Step 8 — Setting environment variables

Environment variables are how you store secrets — API keys, passwords, database connection strings — without putting them directly in your code (where they'd be visible to anyone who sees the GitHub repository).

Locally: ask Claude Code to create a `.env.local` file for a new key you need, and it will also make sure `.env.local` is listed in `.gitignore` so it never gets pushed to GitHub by accident.

On Vercel (so the live site also has access to the same keys):

- Go to your project on vercel.com -> **Settings** -> **Environment Variables**.
- Add the name and value, and tick which environments it applies to (Production, Preview, Development).
- Click **Save**.

One detail worth remembering: **you must redeploy after adding or changing an environment variable** — Vercel doesn't retroactively apply it to a deployment that already happened. Ask Claude Code to "push an empty commit to trigger a redeploy" if you don't have any code changes ready, or just click **Redeploy** on the latest deployment in the Vercel dashboard.

Step 9 — Reading Vercel's logs when a deploy breaks

Two different things can go wrong on Vercel, and they show up in two different places.

The build fails (the site never goes live): Go to your project on Vercel -> **Deployments** -> click the failed deployment (marked with a red ') Scroll the build log for the first line in red — that's almost always the actual cause, even if there's a lot of text after it. Copy that red section.

The site is live but something breaks while running (a form doesn't submit, a page errors after loading): Go to your project -> **Logs** (sometimes labelled **Runtime Logs**) to see what happened on the server side, in real time or just after it happened.

Either way, the move is the same as Step 7: **copy the error text and paste it into Claude Code**, and say what you were trying to do when it happened. Claude Code can read a stack trace far faster than explaining the symptom in your own words.

Step 10 — Leveling up the design

Once the basics work, the fastest way to make a Claude Code site look less "AI-generated" is to stop asking for "modern and animated" and start naming the actual technique you want — vague prompts get you a generic fade-in; named techniques get you something closer to a studio site.

This section's framing is credited to **[@ellydoesdesign](https://www.instagram.com/ellydoesdesign/) on Instagram**, whose "Motion

Vocabulary" naming system is exactly this idea — a shortlist of animation techniques worth asking for by name. Her reference designs demonstrating these were built using **Variant**, a separate AI design tool; the technique names below are paraphrased in our own words for this guide, grouped the same way she groups them:

Scroll-driven • Sections that pin and stack on top of each other as you scroll, like cards being dealt. • Content that skews or stretches slightly based on scroll speed, then settles — gives the page a sense of weight. • Foreground/midground/background moving at genuinely different speeds (true depth-parallax, not just "background moves slower"). • Numbers that roll up digit-by-digit like an odometer when a stats section comes into view. • A horizontal gallery that takes over the scroll direction entirely for one section, then hands it back.

Cursor & hover • "Magnetic" buttons that lean toward the cursor as it approaches, then spring back. • Images that ripple or distort on hover (best used sparingly — one hero image, not a whole grid). • A custom cursor that trails behind the real one and changes shape over different elements. • Cards that tilt in 3D following the cursor, with a highlight sliding across the surface.

Typography • Headlines that reveal letter-by-letter or word-by-word, rising from behind a mask rather than fading in. • Text that briefly scrambles through random characters before locking into the real word — use once per page, not everywhere. • Type that changes weight or width as you scroll past it (needs a variable font).

Texture & transitions • A subtle film-grain overlay across the whole page — one of the fastest fixes for a site that feels too flat and generic. • Page or section transitions that happen behind an expanding shape (a circle from the click point, a diagonal wipe) instead of a plain cut. • An image that grows smoothly from a thumbnail into a full hero image across a page change, instead of the page just reloading.

Four rules that matter more than the list above:

- Pick 2–3 techniques max per site — naming all fifteen in one prompt gets you a cluttered mess, not a polished one.
- Name your easing curve. Asking for "animated" gets you a linear, robotic motion. Ask for **ease-in-out**, **spring**, or give Claude Code a specific cubic-bezier curve.
- Give real durations. Roughly 0.6–0.8 seconds for something entering the page, 0.2–0.3 seconds for hover states. Without numbers, animations tend to feel too slow or broken.
- Always ask for it to **respect reduced-motion settings** — it's an accessibility requirement, and asking for it also tends to make Claude Code build the effect more carefully in the first place.

The one-page cheat sheet

- **Install**: `npm install -g @anthropic-ai/claude-code`, then `claude` inside your project folder.
- **Start a project**: describe what you want, specifically, in one clear prompt.
- **Go live**: ask Claude Code to push to a new GitHub repo, then import that repo at vercel.com/new.
- **Change anything**: describe the change in plain English, read the diff, then ask Claude Code

to commit and push — Vercel redeploys automatically.

- **Something looks broken**: open the browser console (Cmd/Ctrl+Shift+J), copy any red text, paste it to Claude Code.
- **Secrets**: `.env.local` for local development, Vercel -> Settings -> Environment Variables for the live site — redeploy after adding one.
- **Deploy failed or a live page errors**: Vercel -> Deployments (build errors) or Logs (runtime errors), copy the red text, paste it to Claude Code.
- **Design feels generic**: name 2–3 specific motion techniques, an easing curve, and a duration — don't just ask for "modern and animated".

How this guide was made

Authored by **Haojun See**, with AI-assisted drafting by **Claude** — collating steps taught across On The Ground's Claude Code workshops into one written reference. The design section's framing follows the "Motion Vocabulary" naming system by @ellydoesdesign on Instagram; her reference designs were built with **Variant**. Full credit for that framing belongs to her — the descriptions here are our own paraphrasing for this guide.