

My client-side only AI web application workflow

Tom Rochette <tom.rochette@coreteks.org>

October 14, 2025 — 0d6e91b0

Over the past month I've started building client-side only AI web applications (e.g., [ai-text-editor](#), a private ai-language-assistant to teach myself Chinese Mandarin).

I've gone with this approach because it lends itself very well to vibe coding with [vibe-kanban](#).

I've configured the `Dev Server Script` option to open the `index.html` file in my browser and it works immediately, no need to build assets or start a backend server.

This speeds up iteration cycle considerably.

This approach is also great because it produces a tool that can then be used directly in the browser from any device, without needing to install anything.

I can run what I built on my phone, on my work computer, on someone else's computer easily by pointing them out to the project's GitHub Pages URL which serves the application and is "production" ready.

I haven't really picked any frontend libraries or frameworks, just vanilla HTML/CSS/JS.

That is something I need to explore (e.g., react, svelte, solid, tailwind, vue.js, etc.) but for now I want to keep things simple.

For the past few projects I've used Claude Code.

I ask Claude to create the common `CLAUDE.md`.

Additionally, I ask Claude to maintain a `SPEC.md` file that describes the features of the application.

In many cases the database relies on the browser's `localStorage` API and IndexedDB to store data, which is sufficient for my needs.

I ask Claude to maintain a `DATABASE_SPEC.md` file to describe the database schema.

Claude typically goes for a `index.html`, `app.js`, and `styles.css` file structure.

While it is ok for the first few iterations of the project, I usually ask Claude to refactor the code to split it into multiple files and modules as the codebase grows.

Doing so speeds up some of the iteration process since it doesn't end up reading large irrelevant chunks of the file when making edits.

It also makes it easier to review changes since I use the files modified as an indication of whether it worked on the right part of the codebase.

The main downside of this approach is that generally the `app.js` file ends up being quite large (e.g., 1000+ lines of code) since it contains all kind of global state and logic.

With this approach I've been able to fairly effectively work on small projects (~40 hours of work) and get something functional out the door that would have taken me weeks and where I'd probably have given up mid-project due to all kind of minor problems.

1 The template

- Use an `importmap` and javascript modules
- Imports
 - `llm.js` to do LLM calls directly from the browser

```
<script type="importmap">
{
```

```
    "imports": {  
      "llm.js": "https://esm.sh/@themaximalist/llm.js@1.0.1?target=node",  
    }  
  }  
</script>  
<script type="module" src="app.js"></script>
```