From Vibes to Validation: A Hands-On Approach to Product Discovery

Welcome to Our Hands-On Workshop!

Goal: By the end of this session, you'll be able to start from scratch and create a working software prototype that serves as a mechanism for user and customer feedback—turning your intuition into actionable insights.

Workshop Overview:

This 80-minute workshop will walk participants through a real-time, hands-on product discovery flow using vibe coding: a fast, intuitive way to turn an inkling of an idea into something testable and valuable. We'll move quickly from vague concepts to working prototypes—exploring how to generate requirements, build iteratively, test with users, and shape what works into real product momentum.

Workshop Outline:

Welcome & Framing (5 min)

- Introduction
- Why we're here: Vibe Coding for faster, truer product discovery
- Quick poll (PM/Designer/Engineer) (Tool Familiarity)
- Brief story: turning a hunch into a build
- What we'll do in the next 75 minutes
    - Get started together, up to the point of “Hello World”
    - From there, choose your own adventure:
        - Continue along with me, building the Custom Kanban prototype OR
        - Use each step of the workshop to build your own prototype. (Note: the steps in this workshop are meant for building a UI-heavy web application with a single integration to an LLM for one text generation feature.)

Choosing the Right Tools (5 min)

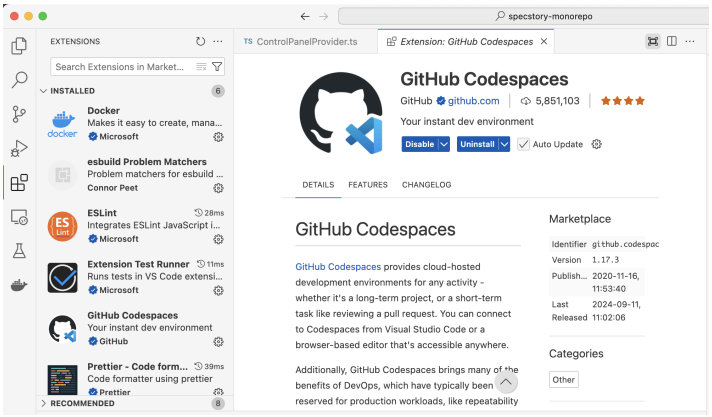
- Design Prototype: Lovable.dev, v0.dev, Bolt.new == build + deploy for fast frontend feedback

- Working Prototype: Cursor, VSCode, or Replit == pair programming with AI for prototyping more than just frontend (data analysis, AI integration, invoking other services)
- When to choose which (speed, team, fidelity, deployability)
- For this workshop we'll build more than just a UI. We'll incorporate an AI feature into our prototype, so that puts us into the second category of tools.
- Here we'll use **VSCode w/Copilot + GitHub Codespaces** because this eliminates most issues with your local laptop and offers generous free usage.

Setup Your Tools and Accounts (10 min)

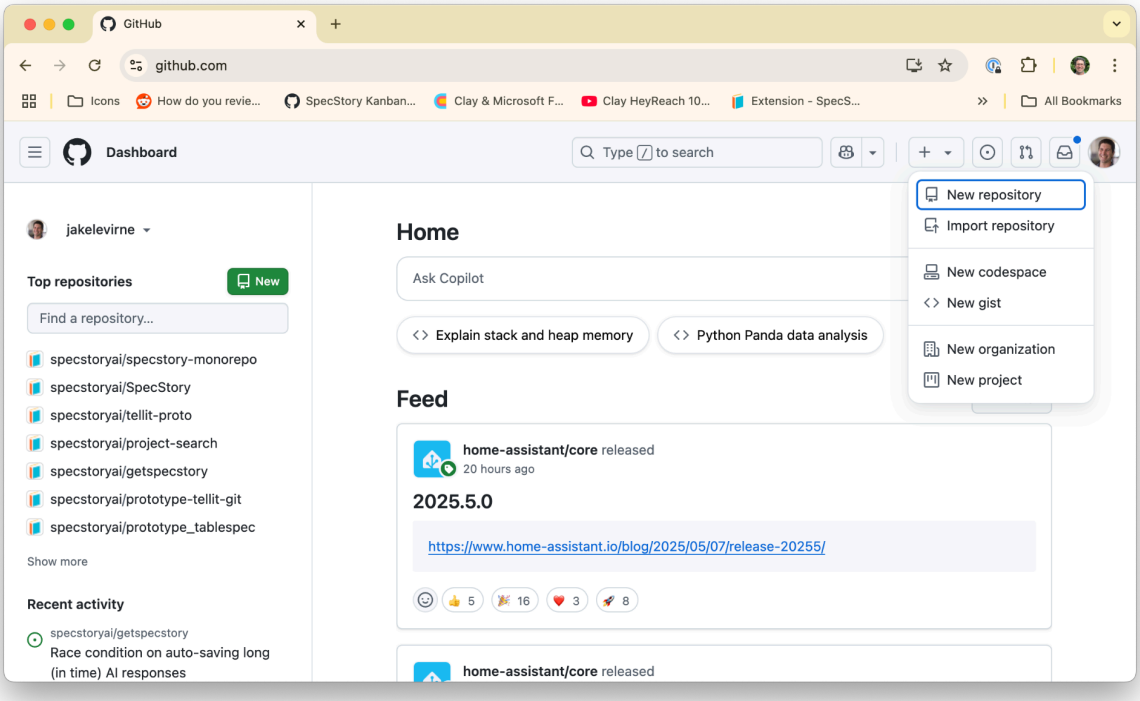
At the end of this step, you'll have all the tools and accounts you need to begin vibe coding.

- Sign up for a GitHub account if you don't already have one
    - Ensure you've got Copilot enabled and fully configured in your GitHub account
    - If you want to have access to the best AI models for coding, upgrade to Copilot Pro (free 30-day trial). If not, for this workshop, you can make due with Copilot Free though you might hit more dead ends.
- Install VSCode
    - Connect VSCode and GitHub Copilot (see the Copilot FAQ if you have any issues).
    - Install the GitHub Codespaces extension into VSCode.

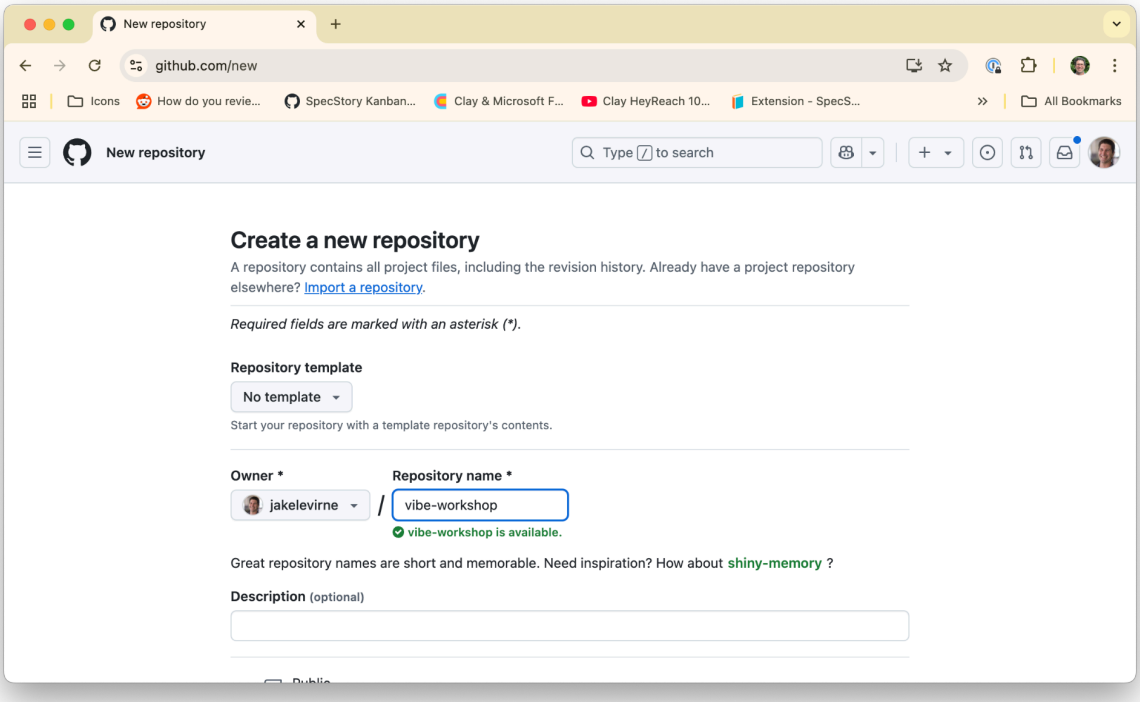


    - Install the SpecStory extension into VSCode if you want to share what you learned at the end of the workshop.
- Create a GitHub repository and enable a Codespace.
    - Note: You should be able to create a Codespace in your personal GitHub account without a problem. Be sure to choose your name (e.g. `jakelevirne`) and not an organization name when creating the repository below.

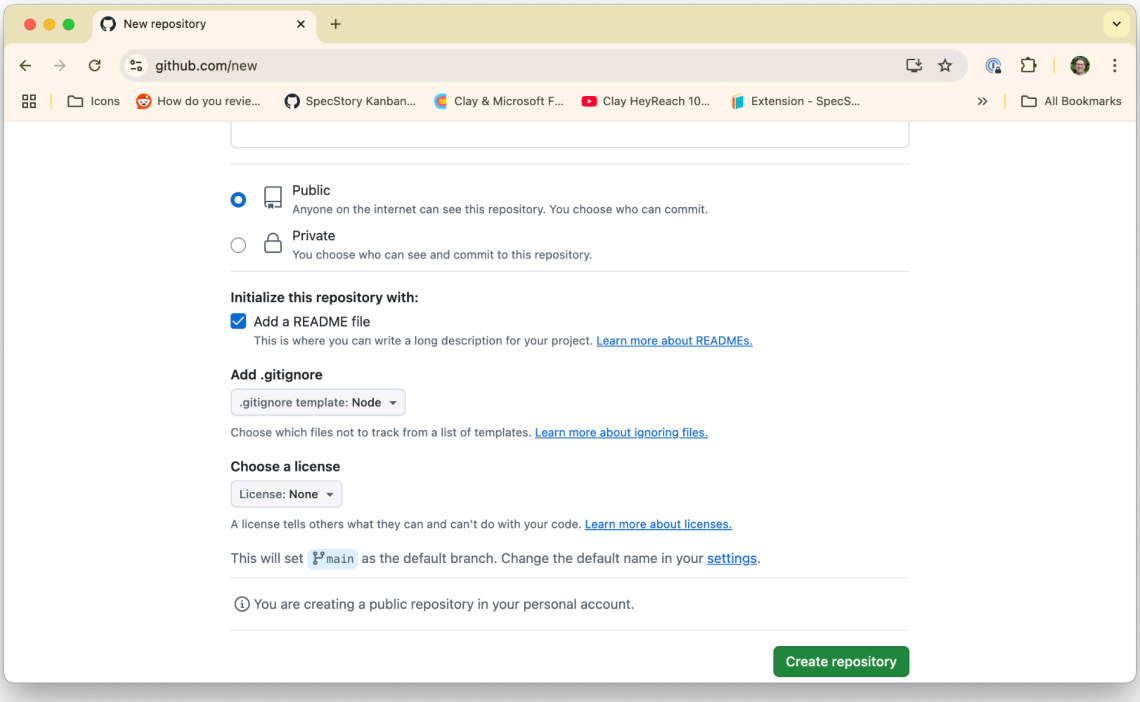
Visit https://github.com and click on the + and then `New repository`



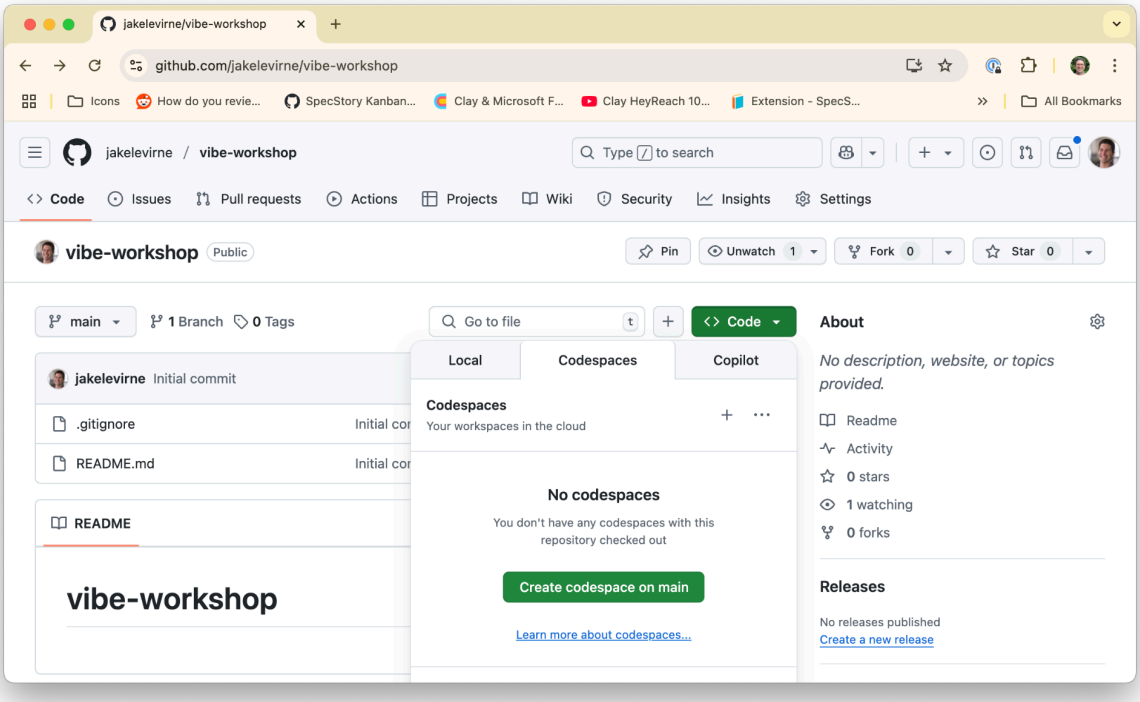
Give your repository a name (like `vibe-workshop`)



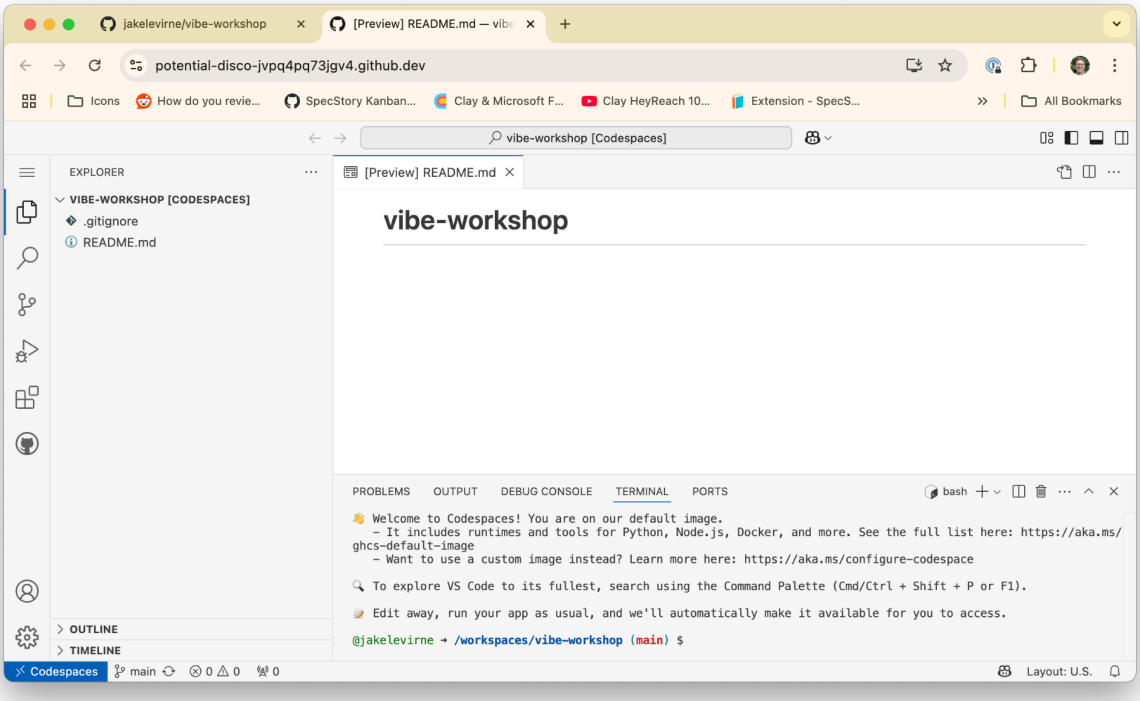
Leave the repo Public if you want to share with others, check the box to `Add a README file` and choose to `Add .gitignore` with a template of Node.



Your project repository gets created. On the resulting page, click the `<> Code` button and then click the Codespaces tab and click `Create codespace on main`

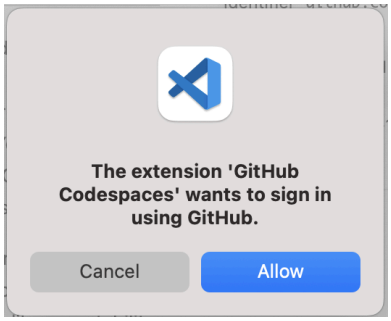


It'll take a couple of minutes, and when it finishes, you'll see this in your browser:



You can now close this tab because we're going to connect to this Codespace from VSCode on our desktop and we don't want two connections to a single Codespace at the same time.

In VSCode, go to the `View` menu and select `Command Palette…`. Then type “codespaces” and select `Codespaces: Connect to Codespace`. Choose “Allow” when prompted to let the extension sign in using GitHub.



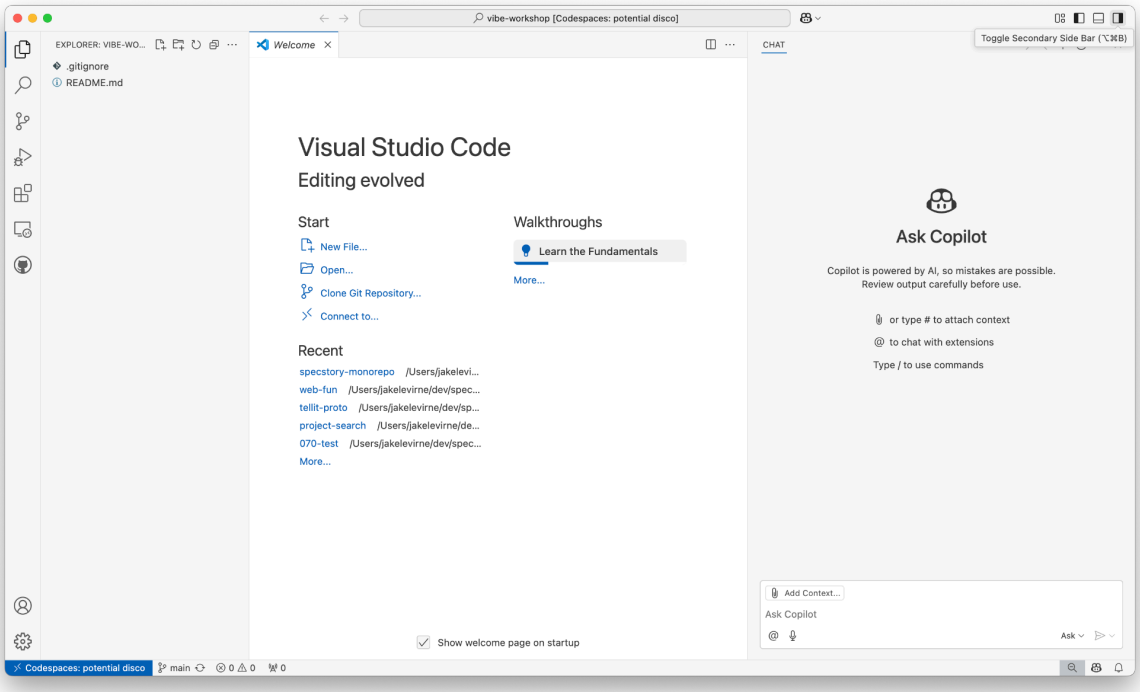
Choose the codespace you just created for your `vibe-workshop` project repository.

Don’t worry if you can’t get your tools and accounts set up for this session. You can follow along with the demo, and then refer to the video afterwards to try again on your own.

Hello World (10 min)

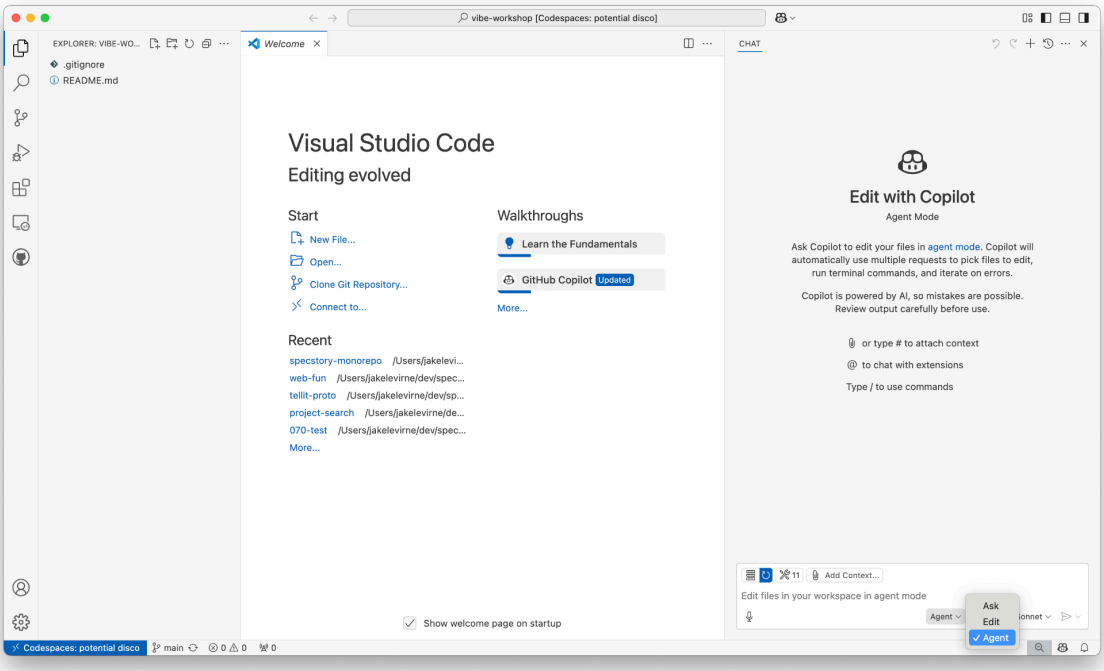
At the end of this step, you’ll have successfully run and made a change to your web application.

In VSCode, click the button in the far upper right to Toggle Secondary Side Bar and make sure that “Ask Copilot” is showing.



Then in the bottom right, click on the Copilot icon  and make sure Copilot is enabled (Code Completions all files).

Then in the bottom right, click on “Ask” and switch into “Agent” mode and make sure your model is set to “Claude 3.7 Sonnet”.

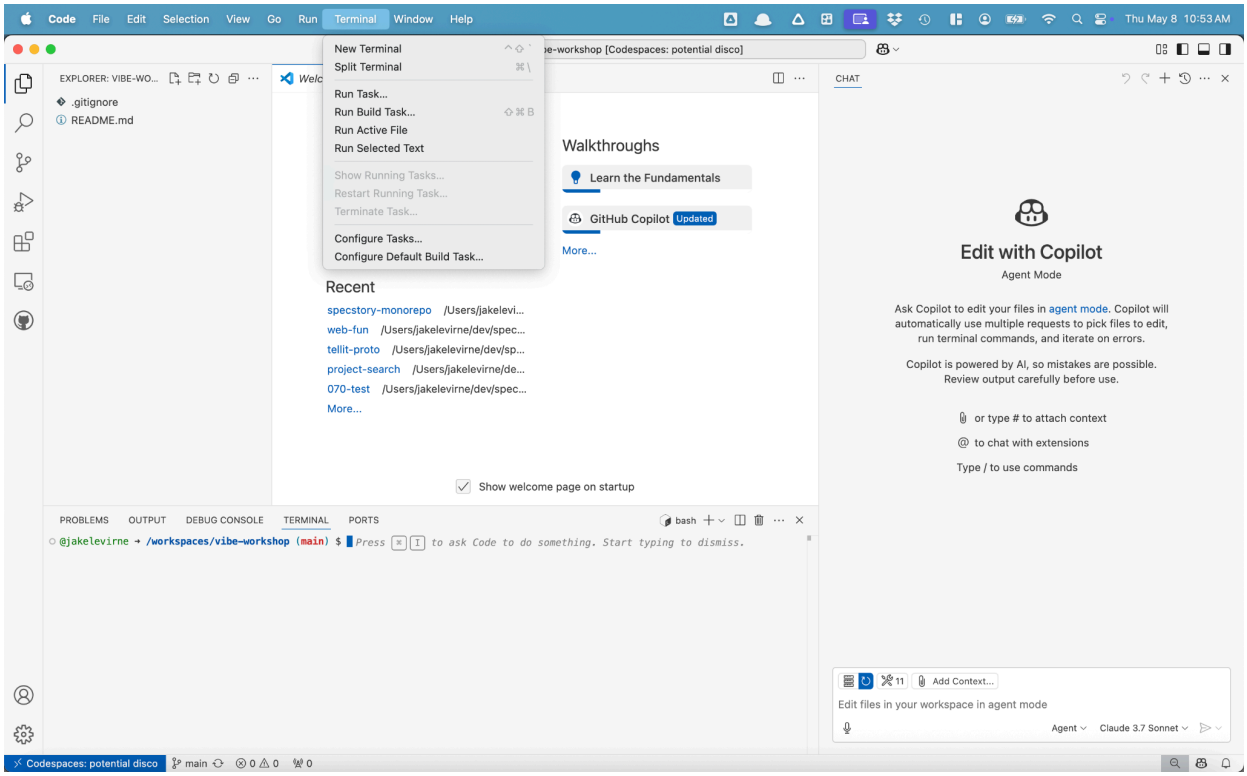


Generally speaking, this is the mode we want to stay in. If you ever notice the AI isn't behaving the way you expect, make sure you're in Agent mode on Claude 3.7 Sonnet. And in general, when the agent prompts for permissions to take action, you should allow it. Remember we're working on a remote Codespace so these actions won't affect your laptop.

Get a Basic Web App Working

We'll create a NextJS application using a set of UI components called ShadCN. AI is good at coding these types of applications because there are a lot of examples. But to initialize a new one, it's much easier to just run the single create command ourselves than to rely on AI.

Open a terminal in VSCode (`Terminal`->`New Terminal`)



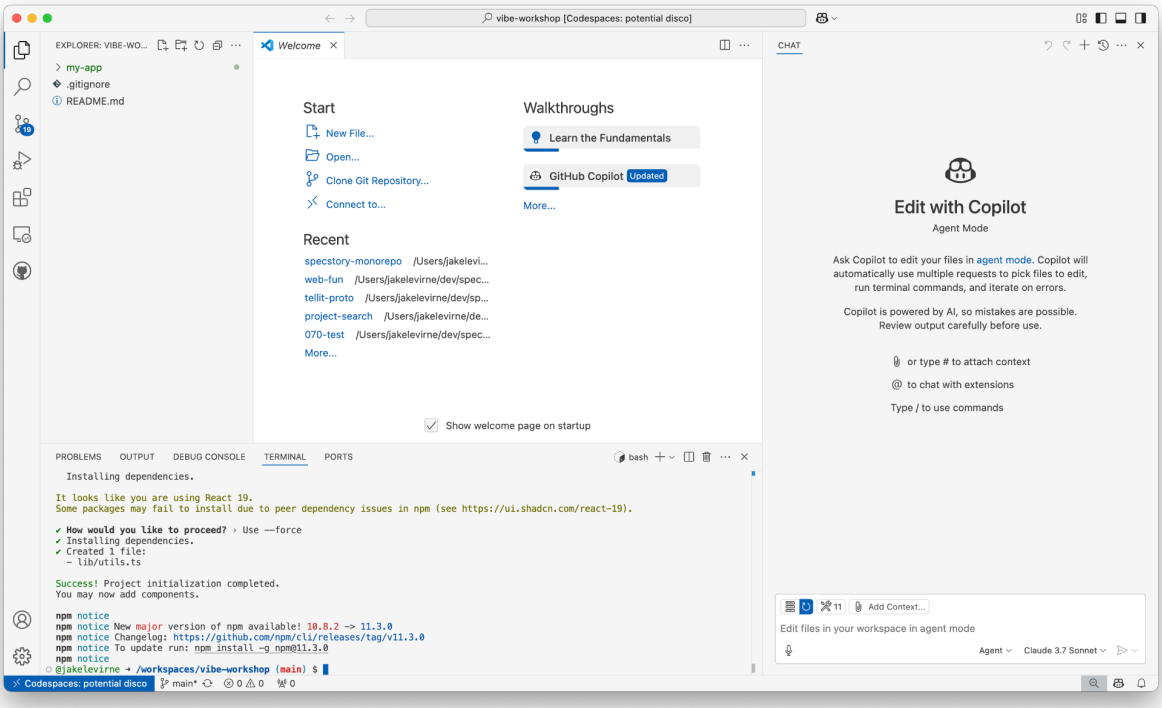
Then, type this exact command into the terminal inside VSCode:

```
npx shadcn@latest init
```

And select the default (enter) for every question the command asks.

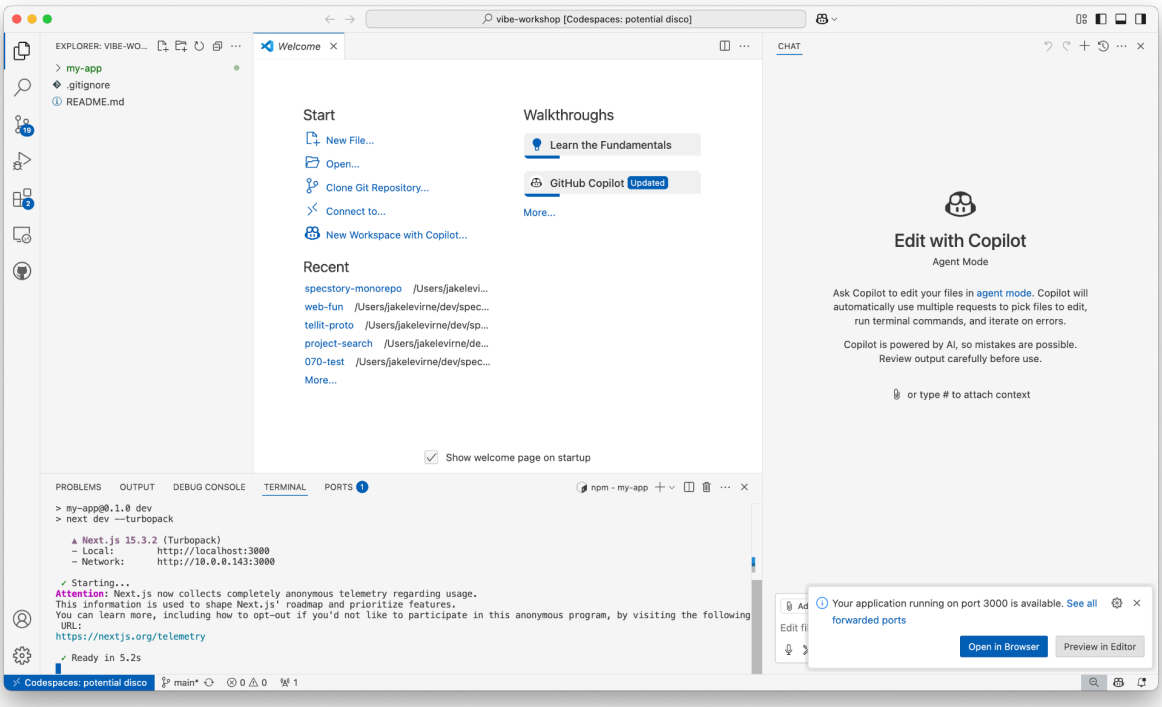
```
Need to install the following packages:
shadcn@2.5.0
Ok to proceed? (y)

npm warn deprecated node-domexception@1.0.0: Use your platform's native DOME
xception instead
✔ The path . does not contain a package.json file.
  Would you like to start a new project? › Next.js
✔ What is your project named? … my-app
✔ Creating a new Next.js project.
✔ Which color would you like to use as the base color? › Neutral
✔ Writing components.json.
✔ Checking registry.
✔ Updating CSS variables in app/globals.css
  Installing dependencies.

It looks like you are using React 19.
Some packages may fail to install due to peer dependency issues in npm (see
https://ui.shadcn.com/react-19).

✔ How would you like to proceed? › Use --force
✔ Installing dependencies.
✔ Created 1 file:
  - lib/utils.ts

Success! Project initialization completed.
You may now add components.
```

After a few minutes, you'll see the command complete in your terminal and you'll see a new directory named `my-app` in the left hand file explorer.

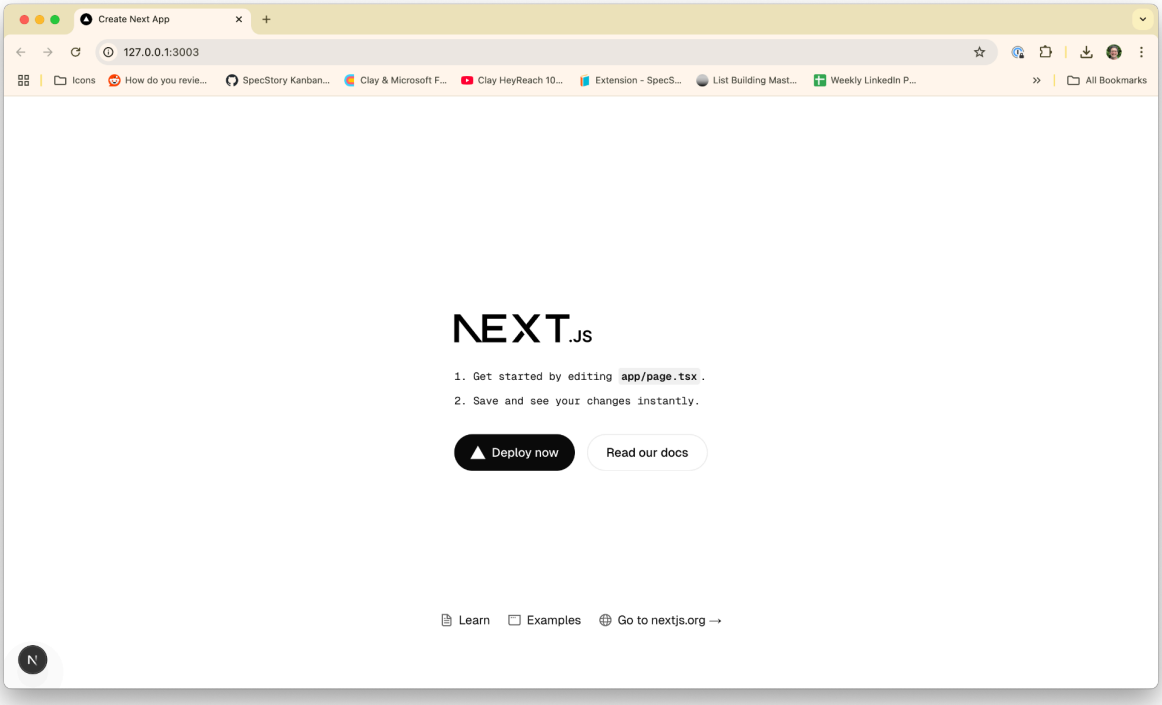


Then, run these commands:

```
cd my-app
npm run dev
```



And then click “Open in Browser” to see the NextJS starter app.

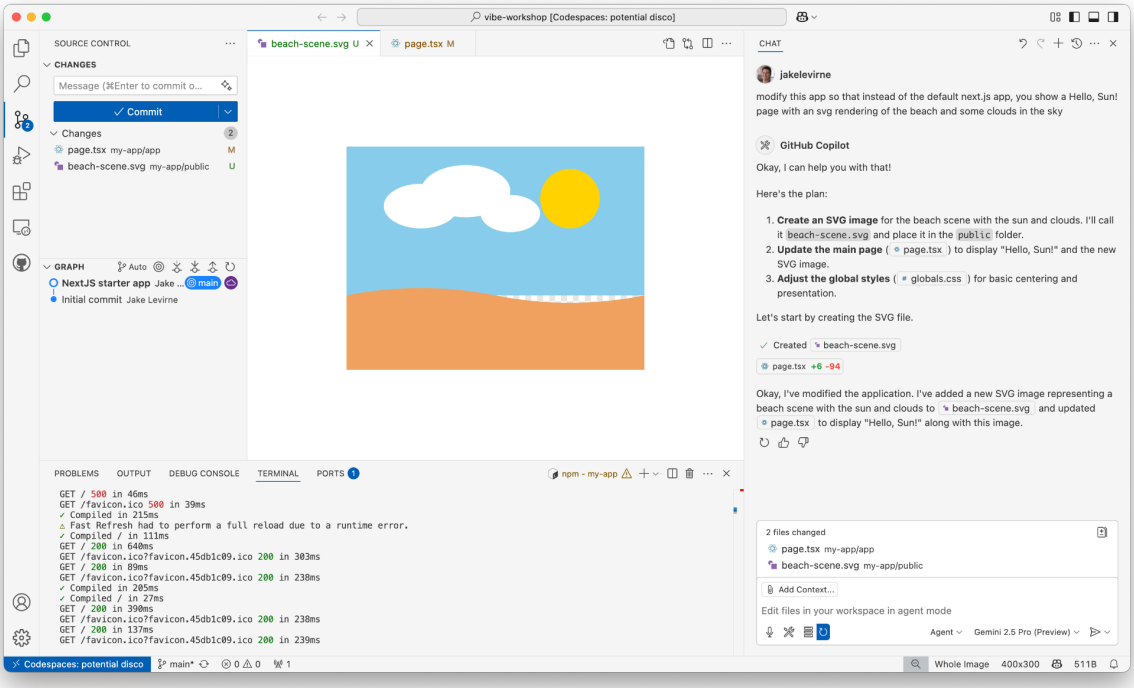


And now, let’s get the Agent making a basic change for us. In VSCode, at the bottom of the Edit with Copilot window, make sure you’re in Agent mode and have Gemini 2.5 Pro (or Claude 3.5

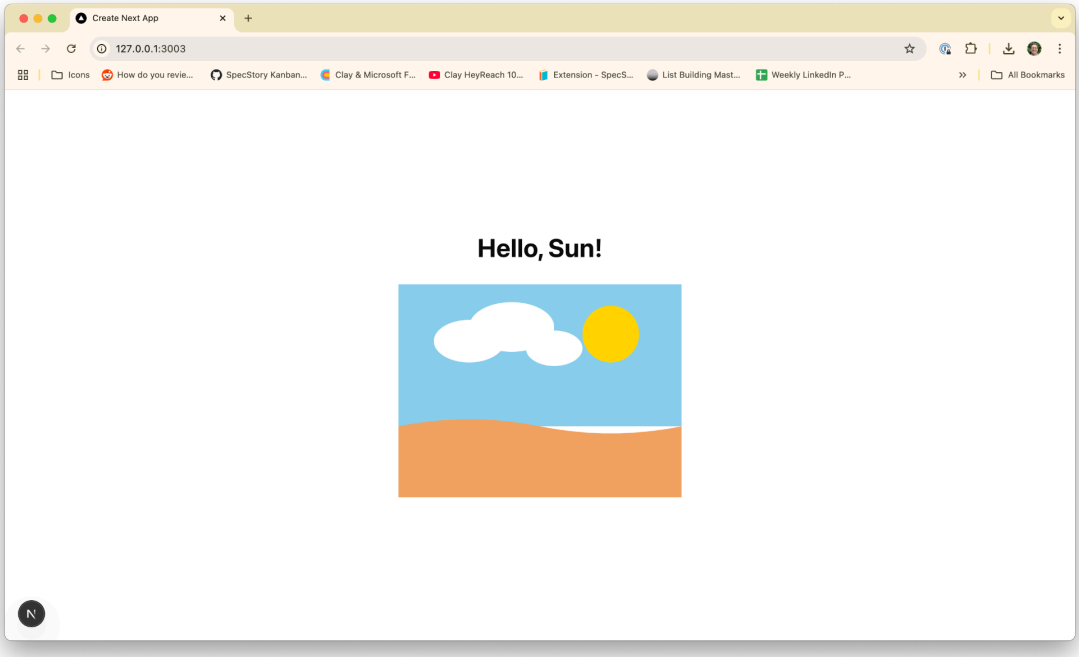
Sonnet if you're in Free mode) selected. If this model is erroring out or extremely slow, try another model like GPT-4o or GPT-4.1


```
modify this app so that instead of the default next.js app, you show a Hello, Sun! page with an svg rendering of the beach and some clouds in the sky
```

You'll get something similar to (but not exactly like) this:



And if you switch back to your browser and reload the page, you'll see it updated automatically. It will continue to do so as you make changes, as long as your Next server (`npm run dev`) is still running in the VSCode terminal. If it stops, just restart it with the same command.

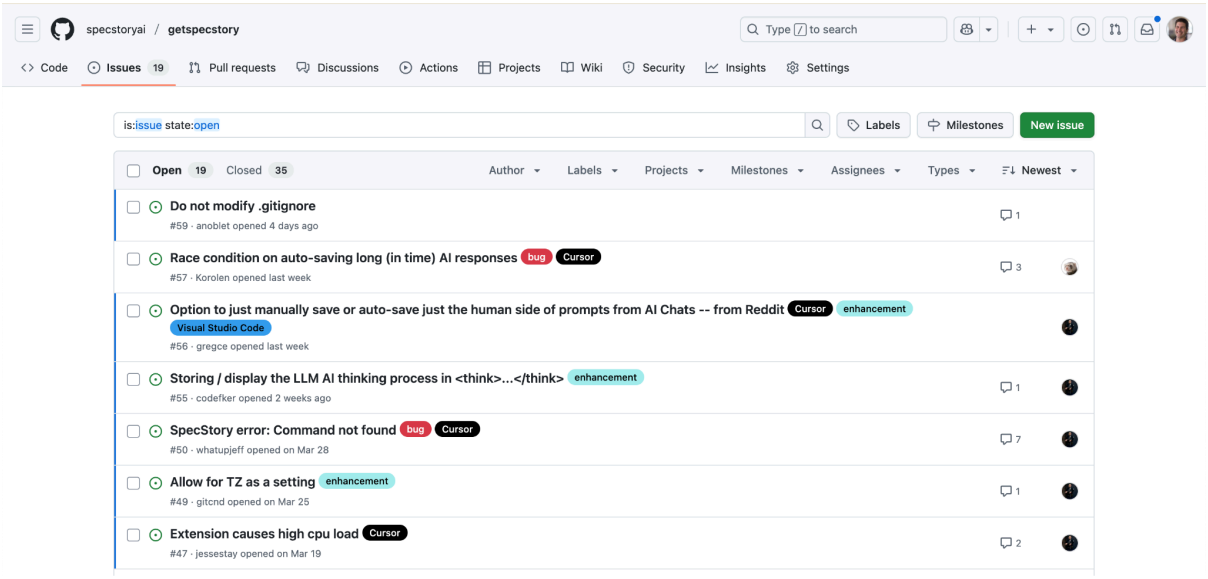


One Shot a Basic Prototype (10 min)

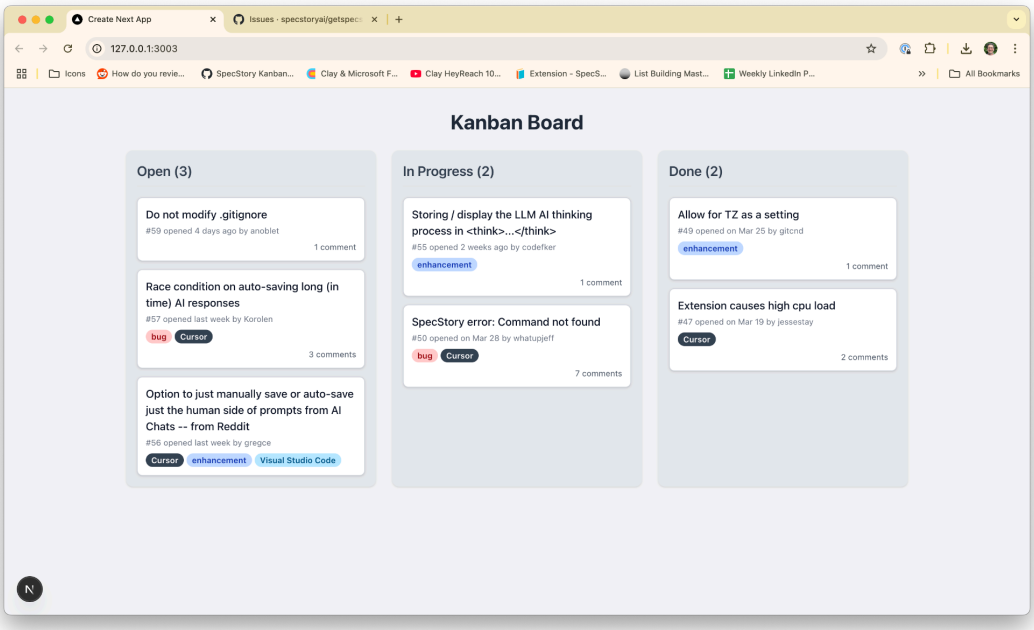
To give yourself (and the AI) something to react to, it can be helpful to get some sort of UI prototype working as quickly as possible. Crafting a good initial prompt here is often about constraining what the AI is trying to do in this first pass so it doesn't go off the rails.

```
Ok, what we want now is the basic outline of a kanban board
application with some dummy data that looks to be in the style of the
attached screenshot of Github. It does not need to function. It's just
a mockup.
```

(The screenshot isn't necessary, but it's always interesting to see what it does with it. Just use your favorite screenshot tool and then paste or drag the screenshot right into the Agent chat box.) Here's the screenshot I used, for reference.



And if all goes well, you'll see something that looks similar to (but not exactly like) this.



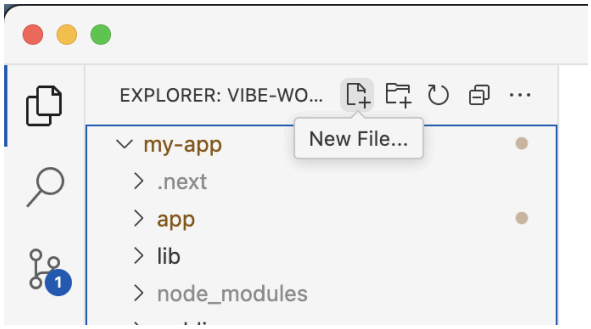
At this point you might feel like, "nailed it.. I'm done!" And if all you need is a visual prototype, this could be a great stopping point. But if that's all you need there are easier ways to get here (v0.dev, lovable.dev).

Requirements & Workplan Generation (10 min)

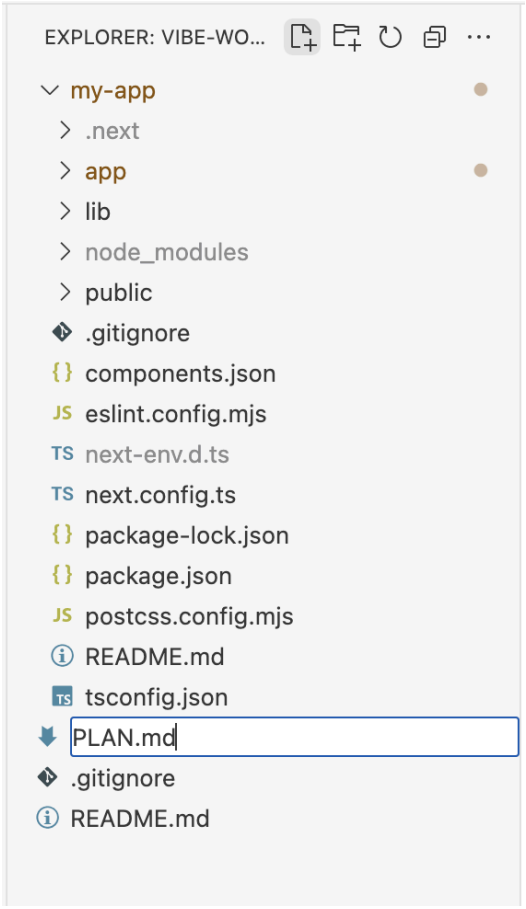
Ok, at this point you could just keep prompting the Agent to make changes bit by bit. But there's a more methodical approach that's helpful especially as your project grows. And for the Product

people out there, this will feel very familiar… we're going to write requirements and work plans. But we're going to have the Agent help us write them.

In the VSCode File Explorer, choose to create a new file



And name it PLAN.md (it'll be a markdown file).

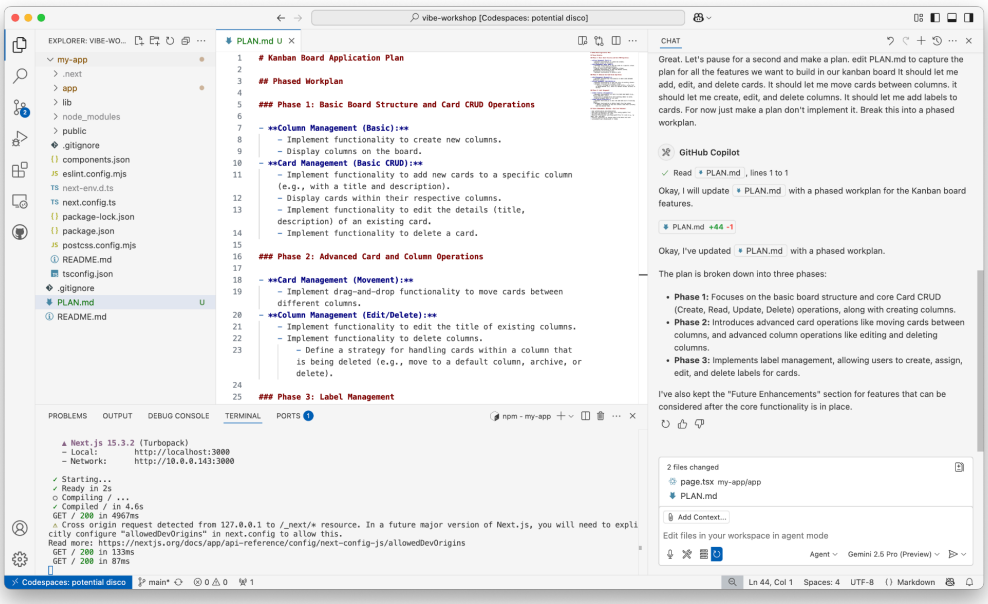


Make sure it ends up at the top level of your project. If not, drag it to the right place. There doesn't need to be any text in it; we'll use the Agent to write and edit it for us. Use a prompt like this:

```
Great. Let's pause for a second and make a plan. edit PLAN.md to
capture the plan for all the features we want to build in our kanban
```

```
board It should let me add, edit, and delete cards. It should let me move cards between columns. it should let me create, edit, and delete columns. It should let me add labels to cards. For now just make a plan don't implement it. Break this into a phased workplan.
```

You should get a phased plan written for you, like this.



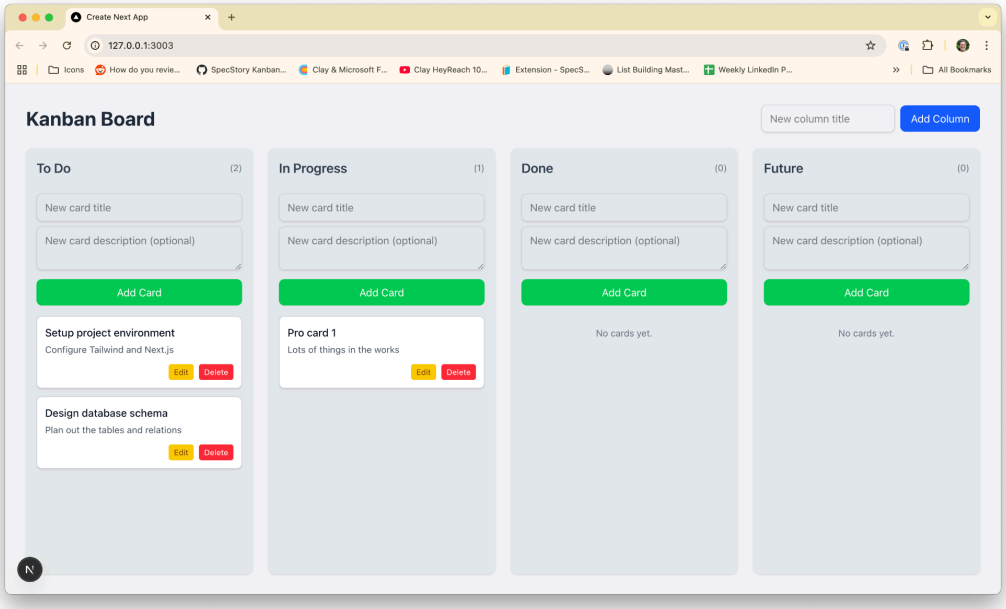
Review the plan at a high level, make sure it makes sense. Change ordering and make edits either directly in the doc or via the Agent.

Iteratively Implement and Refine the Plan (10 min)

It gets easy and fun at this point. Just tell the Agent

```
begin implementing phase 1, checking off items in the PLAN as you complete them
```

Keep checking in on your running web app to see the progress.



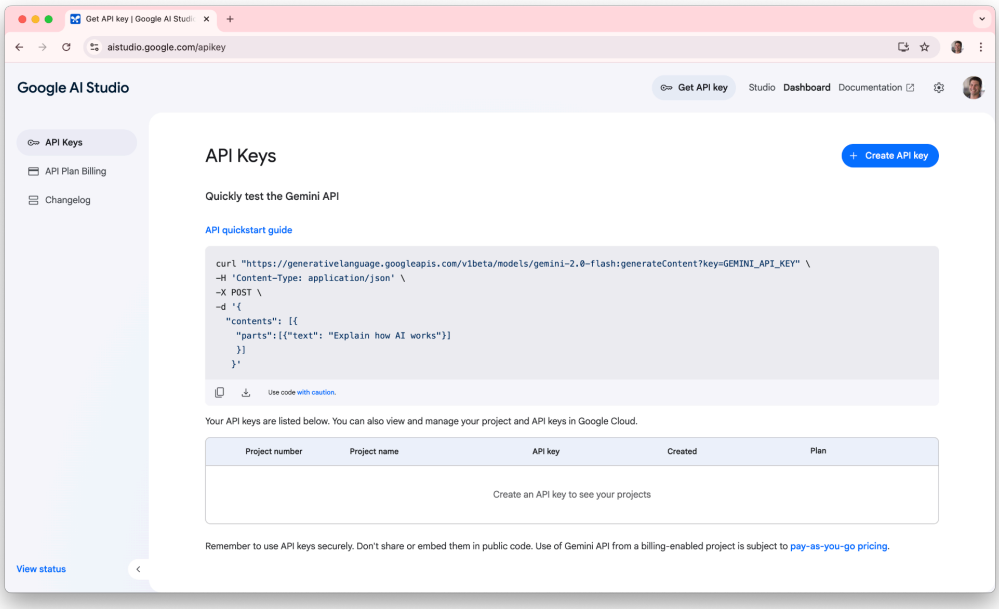
Ask AI to refine your PLAN.md as you come up with new ideas.

Third-Party APIs & AI Integration (10 min)

With this step, we'll learn two things: how to set up an API integration to an existing service (in this case, Gemini) and how to integrate AI text generation into our app.

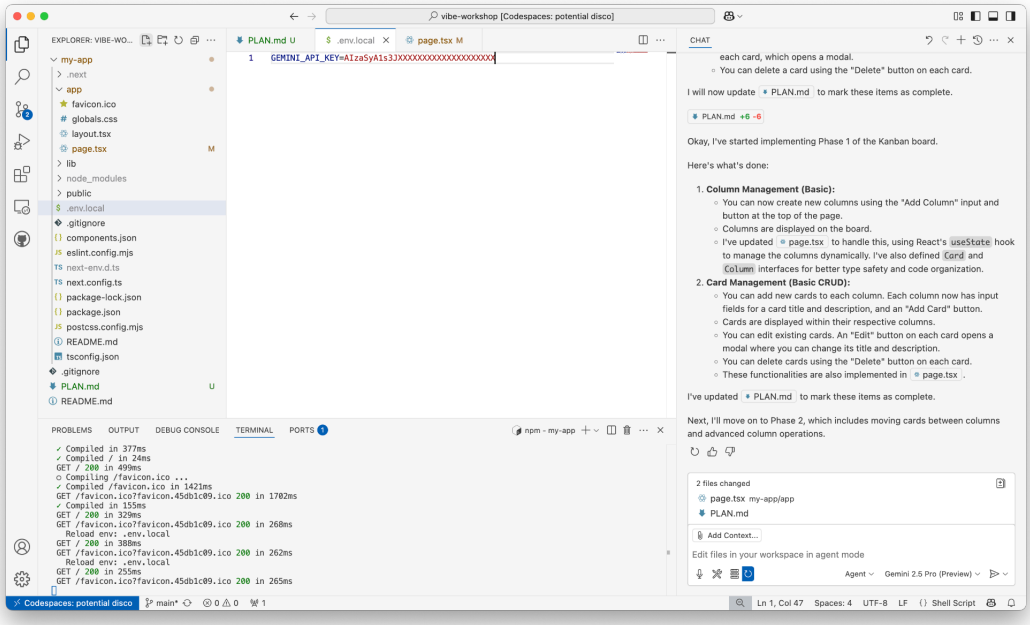
This requires the use of environment variables in our NextJS project. If you're not familiar with this concept, some AI chats and Googling can go a long way. Here's the [reference guide on using environment variables in NextJS apps](#).

First, use a Google login and access https://aistudio.google.com and click "Get API key" and then "Create API key" and finally "Create API key in new project"



Copy the resulting API key and get ready to paste it into a new file in your project. Right click on the `my-app` folder in your project, create a new file and name it `.env.local` (including the dot at the beginning of the filename).

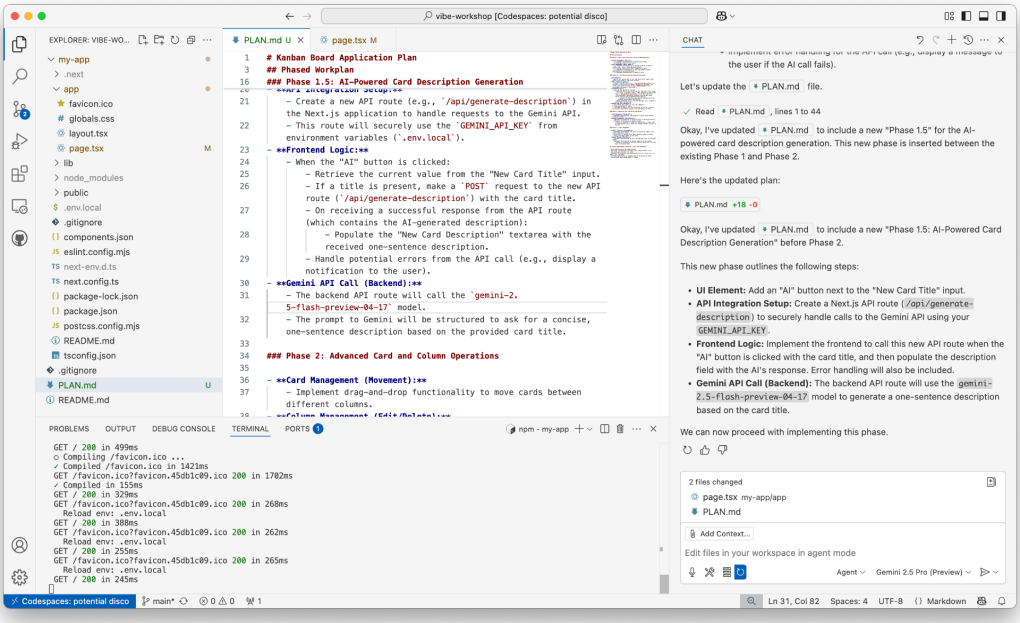
In this new file, type `GEMINI_API_KEY=` and then paste your copied key, like below and then save the file.



Try this prompt to get it to plan our AI integration.

```
Before moving on, insert a new phase in PLAN.md that we'll work on now. We want to add an AI button next to the New Card Title. When the user types in a title and clicks the AI button, the app makes a call to gemini-2.5-flash-preview-04-17 with the card title, asking for a one sentence description back. I've already got my Gemini API key defined in .env.local as GEMINI_API_KEY. Make the plan before we start implementing.
```

We'll end up with something like this.



Now prompt the Agent:

```
Implement this phase of the plan
```

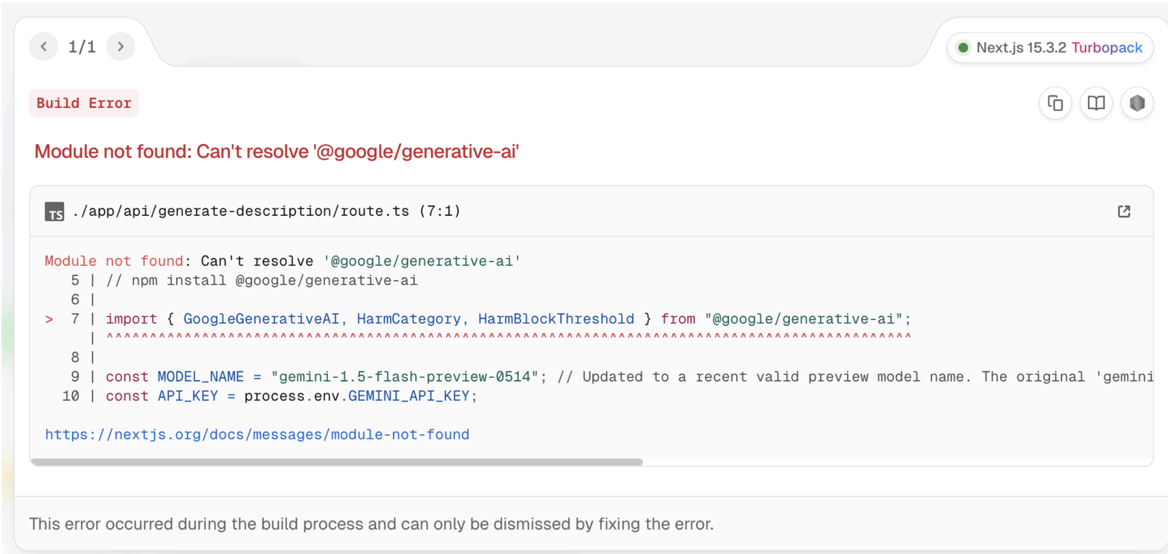
At this point, it created a [NextJS back end API route] for me. I couldn't easily test this without a UI, so I prompted

```
continue the implementation
```

Sidebar: Troubleshooting

Ironically, I almost always end up with some kind of error when I use Gemini to generate the code for calling the Gemini AI API 🤦‍♂️. This is good practice for troubleshooting. See the troubleshooting tips in the Reference section of this document for help.

As I told it to continue implementing, I ran into this error while testing the button:

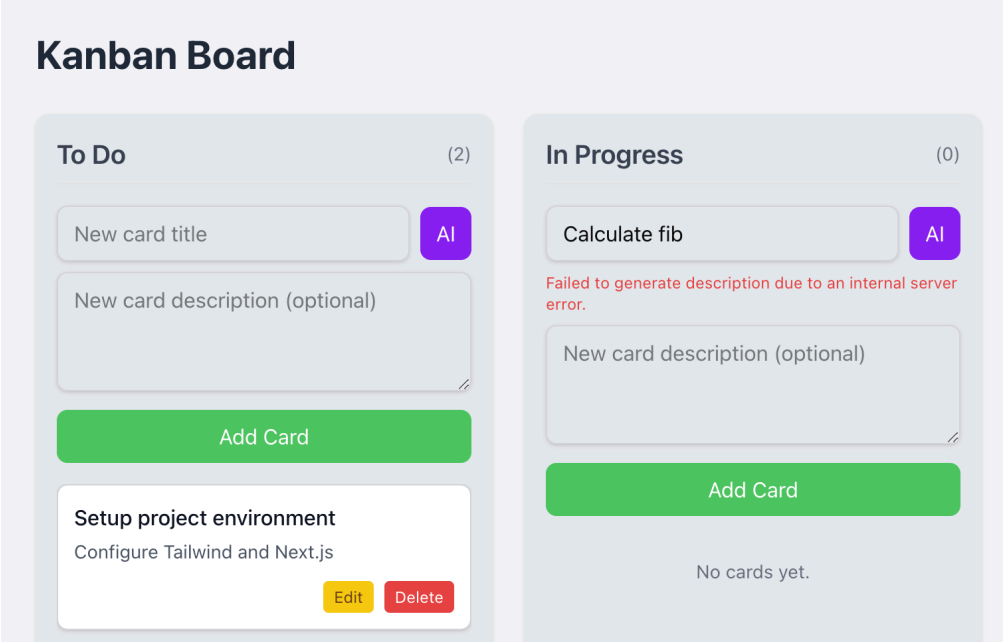


This one is tricky. Through a bunch of AI questions, and thanks to some past experience, I realized that the `npm install @google/generative-ai` command had been run in the wrong directory. I needed to change into the `my-app` directory first before running it. Like this:

```
cd my-app
npm install @google/generative-ai
```

(All this in the existing terminal by hitting Ctrl-C to stop `npm run dev`, then running the commands above, and then starting the dev server up again. Sometimes this is needed, especially when installing new NPM packages.)

We got past that error, but then hit this new problem in the UI.



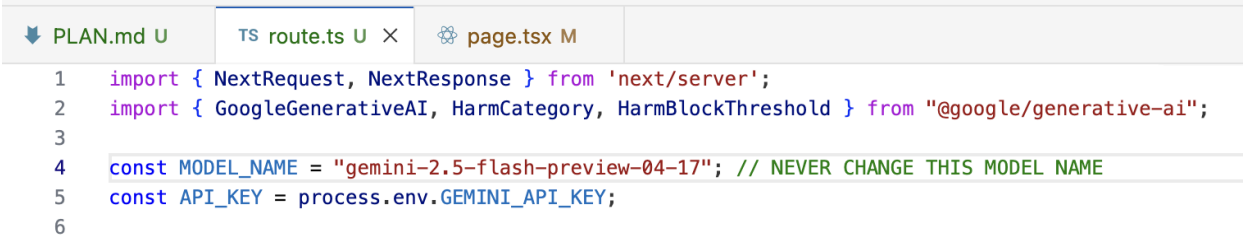
And when we go to the Terminal in VSCode we see this.

In another twist of irony, the Gemini coding agent doesn't know the name of the latest Gemini model. So in this case, I had to look at this code, correctly type the model name (`gemini-1.5-flash-preview-04-17`).

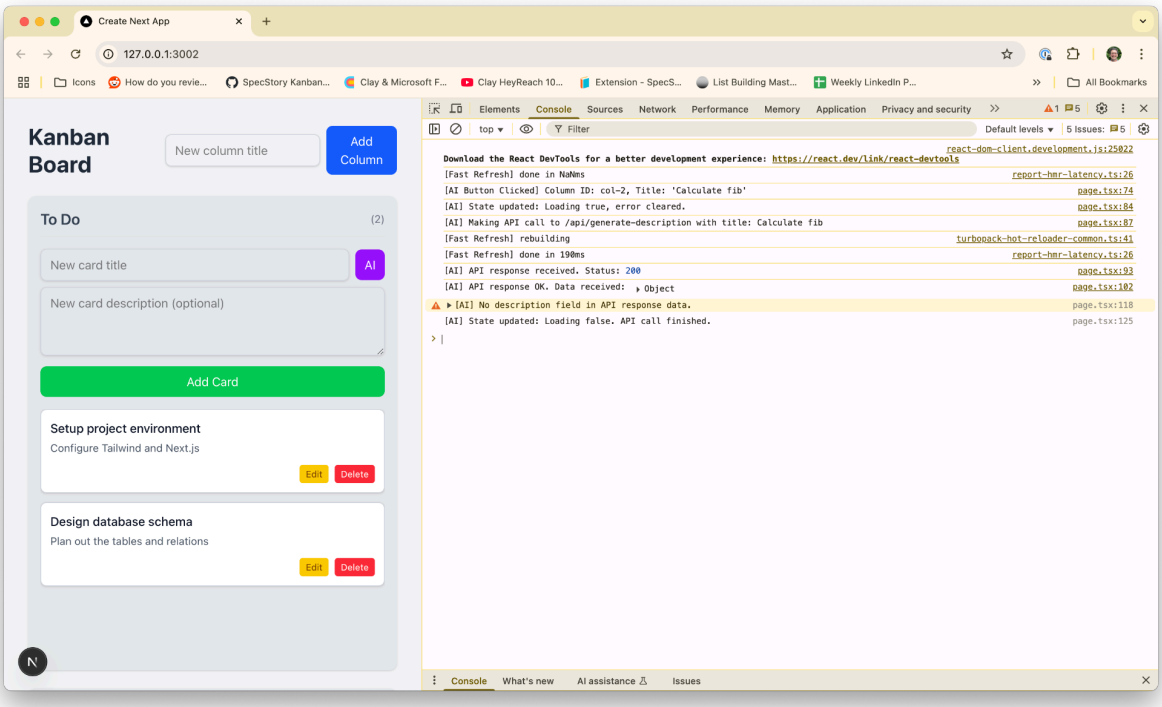
At this point, I no longer got an error. But clicking the AI button produced no result. I prompted the Agent:

```
Pressing the AI button does something, but it doesn't update the
description. add lots of logging. Output as much information as you
can both on the client and server side.
```

When it made that logging update, it changed my model name again. Sometimes the Agent is aggressively wrong! So I changed it back by hand, with this comment in hopes it would help:



And (waiting with baited breath), I got … nothing. I had typed a card title and hit the AI button, but got no description. Returning to the logs, I saw this:

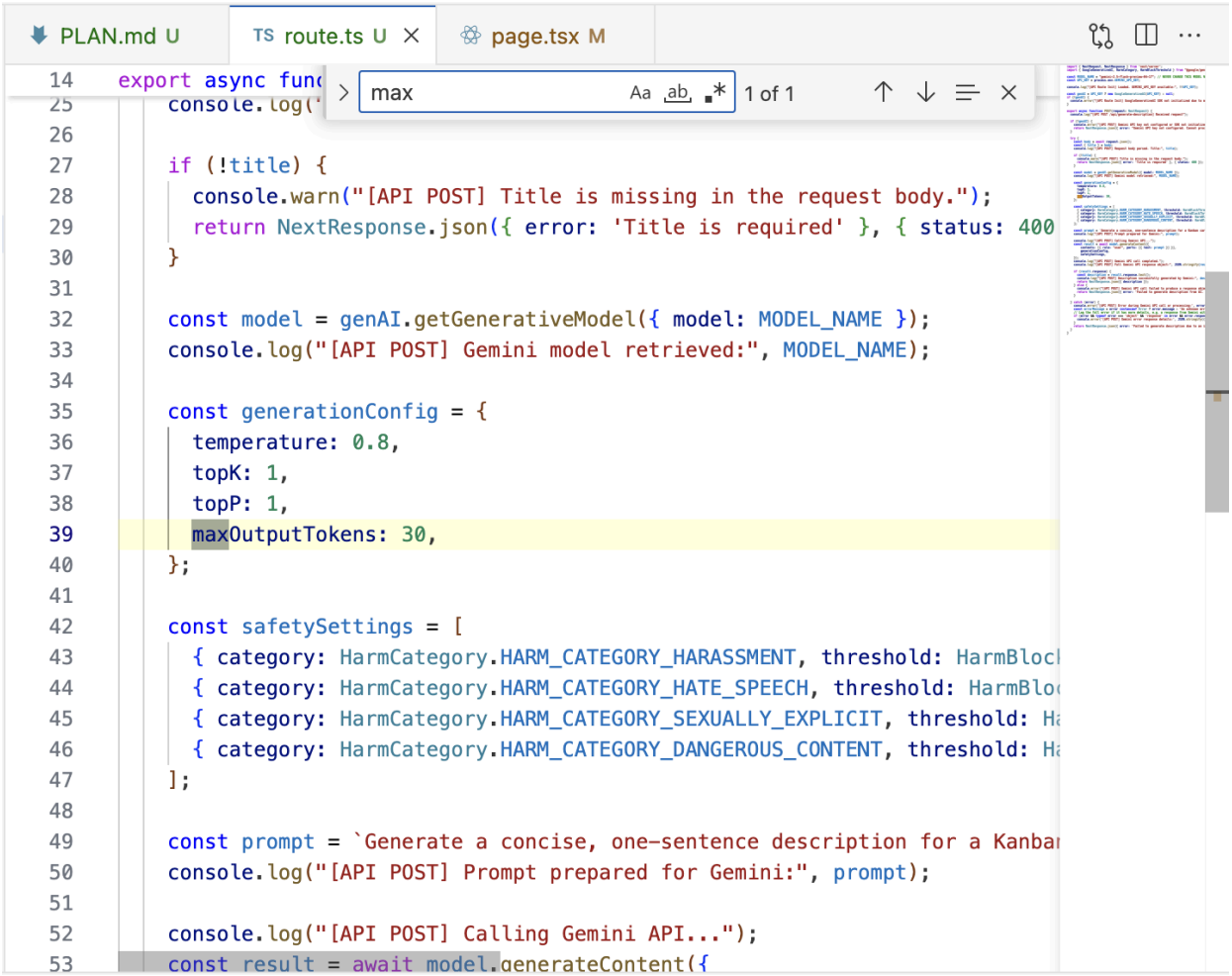


“No description field in API response data.” And looking at the server logs I saw this.

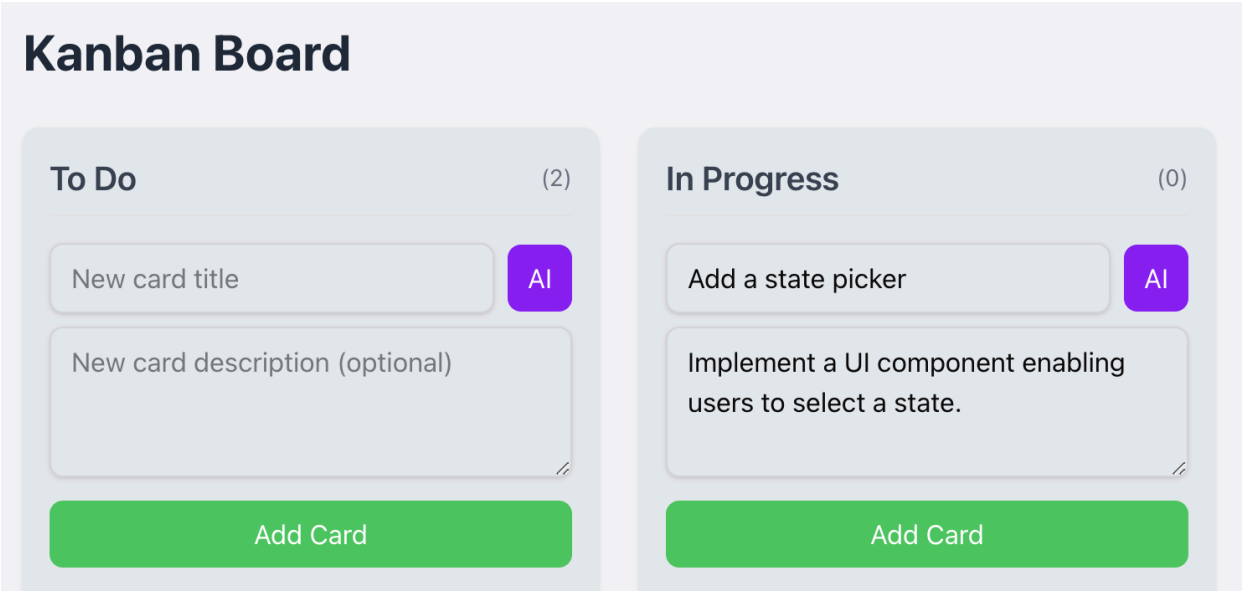
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 1 npm - my-app

```
[API POST] Request body parsed. Title: Calculate fib
[API POST] Gemini model retrieved: gemini-2.5-flash-preview-04-17
[API POST] Prompt prepared for Gemini: Generate a concise, one-sentence description for a Kanban card with the following title: "Calculate fib". The description should be suitable for a task management system.
[API POST] Calling Gemini API...
[API POST] Gemini API call completed.
[API POST] Full Gemini API response object: {
  "response": {
    "candidates": [
      {
        "content": {
          "parts": [
            {
              "text": ""
            }
          ],
          "role": "model"
        },
        "finishReason": "MAX_TOKENS",
        "index": 0
      }
    ],
    "usageMetadata": {
      "promptTokenCount": 33,
      "totalTokenCount": 63,
      "promptTokensDetails": [
        {
          "modality": "TEXT",
          "tokenCount": 33
        }
      ],
      "thoughtsTokenCount": 30
    },
    "modelVersion": "models/gemini-2.5-flash-preview-04-17"
  }
}
[API POST] Description successfully generated by Gemini:
 POST /api/generate-description 200 in 5515ms
```

This one's trickier to spot, because there's no explicit error. But the culprit is `"finishReason": "MAX_TOKENS"`. Something is causing us to run out of tokens. Switching to the code for this API route (`route.ts`), and doing a Ctrl/Cmd-F find for "max" revealed this.



It looks like the Agent wrote code that sets a very low limit for maximum output tokens from Gemini. So I just deleted the line `maxOutputTokens: 30,` and gave it another try.



Success! 💥

(And now's not a bad time to use Git to checkpoint your work. See below in the Reference section for details.)

Sidebar: Outdated Packages

Even the best Agent models are out of date in their knowledge. If you search on https://www.npmjs.com/ for the `@google/generative-ai` package we're using you'll notice that it's deprecated and will no longer be supported after August 31, 2025. After a bunch of poking around and searching I found the newer package– `@google/genai`. But since the Agent doesn't know about it, it'll take a lot of strong-arming to get the Agent to use it. You can do things like copy/paste the quick start example from the documentation into your prompt. But even this can be flaky. You can roll up your sleeves and edit the code in `route.ts` by hand– the example is simple enough. Welcome to the jagged edge of AI code generation. 😀

From Prototype to Production: Best Practices and Wrap (5 min)

- When to throw it out and rebuild
- When to evolve into a real product
- Pairing with engineers or hiring to scale
- Documenting intent to guide future dev
- It's good for rapid shape up
    - Vibes → Prototype → Signal → Bets
    - Shape smaller, ship smarter

Reference

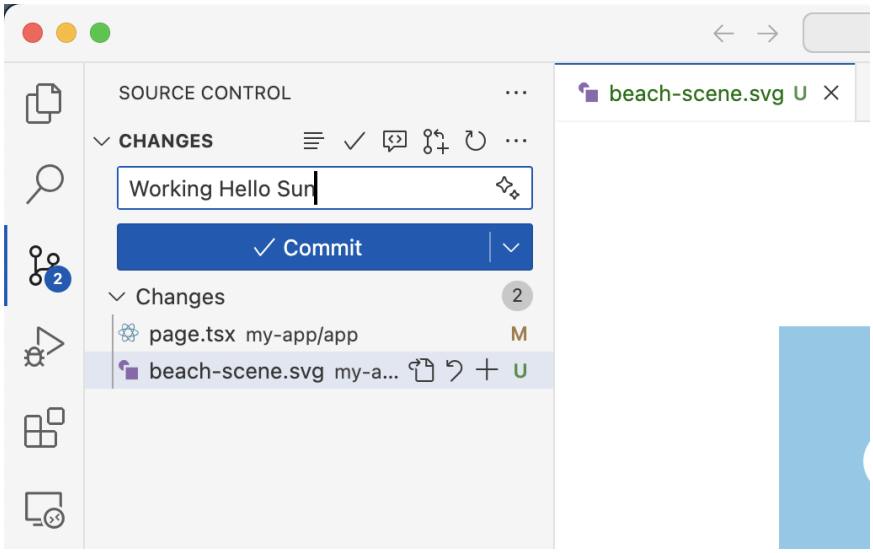
Links

- https://ui.shadcn.com/docs/components/accordion
- https://www.npmjs.com/
- https://lovable.dev
- https://cursor.com
- https://v0.dev
- https://specstory.com
- https://nextjs.org/docs/pages/guides/environment-variables

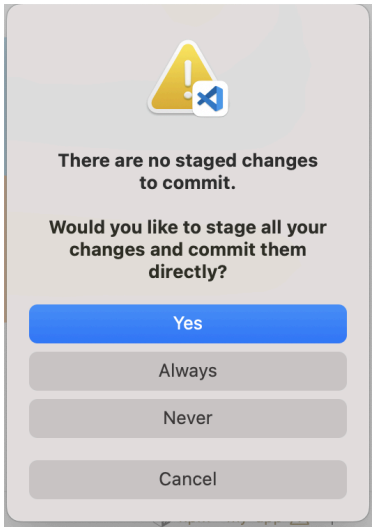
Using Git to checkpoint work

While VSCode offers the ability to Undo an Agent, it's more reliable to "checkpoint" your work using Git, which is built right into VSCode.

Click the Source Control icon on the left side (3rd from the top) and type a message, which is just a description of this checkpoint.

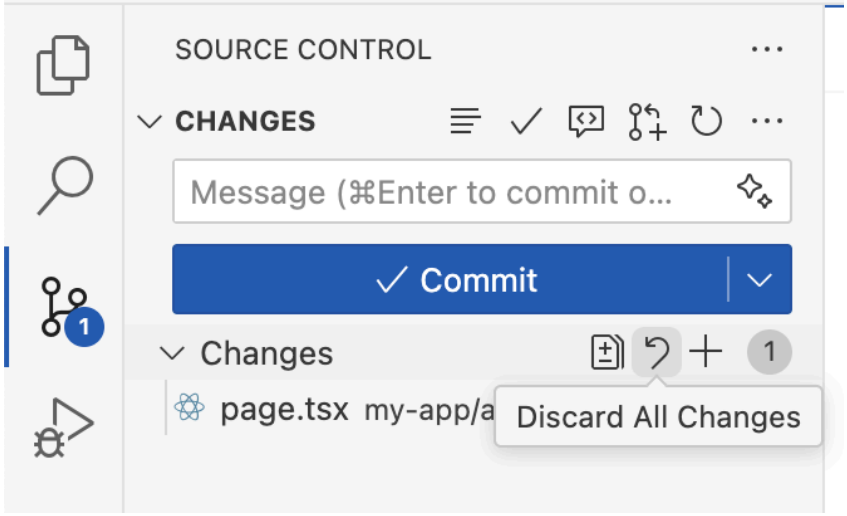


While you can individually choose which files to checkpoint, it's easier to let VSCode stage all your changes for you. Answer "Yes" to this prompt



And then, hit Sync Changes (which pushes this checkpoint to GitHub so it's saved in the cloud for you).

Now, whatever changes the Agent makes from this point forward can easily be discarded. For example, here are some more changes and the button that'll let you discard them.



There are even advanced ways to use Git from the command line to roll back further or select specific files to roll back. Ask an AI questions about `git` to get details of how to do these advanced operations.

Tips for Troubleshooting

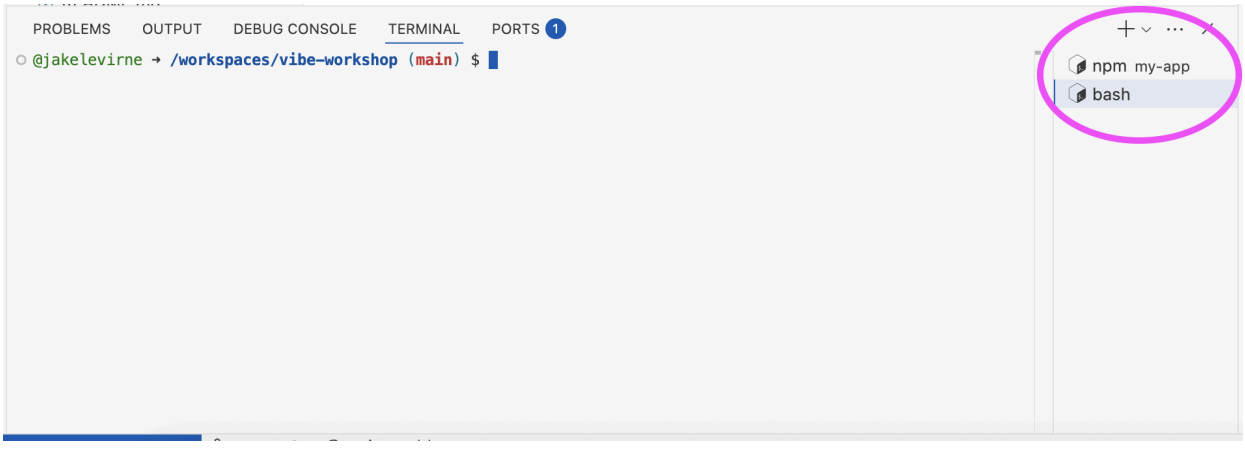
Here are some general tips for troubleshooting. Try these methodically, not all at once.

- Copy the full error message you're receiving (take a screenshot if you can't copy the text) and paste the error into the Agent prompt. Then use a prompt like `I'm getting this error. Please explain it to me and help me troubleshoot.`
- Be the Agent's eyes and hands. Describe to the Agent exactly what you're doing, what's happening, and what you instead expect to be happening.
- Start a new chat to wipe the Agent's memory. Then feed it with initial context– you can manually drop your PLAN.md into the prompt, drag specific code files into the prompt, or ask the Agent to explain the details of the feature you're working on (or a feature near the one you're working on).
- Switch models. Different models have different approaches in different situations.
- Use git to checkpoint work (see the Reference section of this doc). When you roll back with git, it often makes sense to start a new chat as well.
- Ask the agent to add logging to the area of the app where you're having problems. Then see the Reference section of this doc for tips on how to read and use the logs. A prompt like this usually works: `add lots of logging to <x>. Output as much information as you can both on the client and server side.`

Managing Terminals in VSCode

One of the nice things about VSCode is that it has a built-in command line terminal for you. For this workshop, your VSCode terminal points to the Codespace running in the cloud, not your local laptop. This is exactly what you want but it also means you won't find your local files from it. In general, if you ever need help running a command or figuring out what command to run you can just ask Copilot.

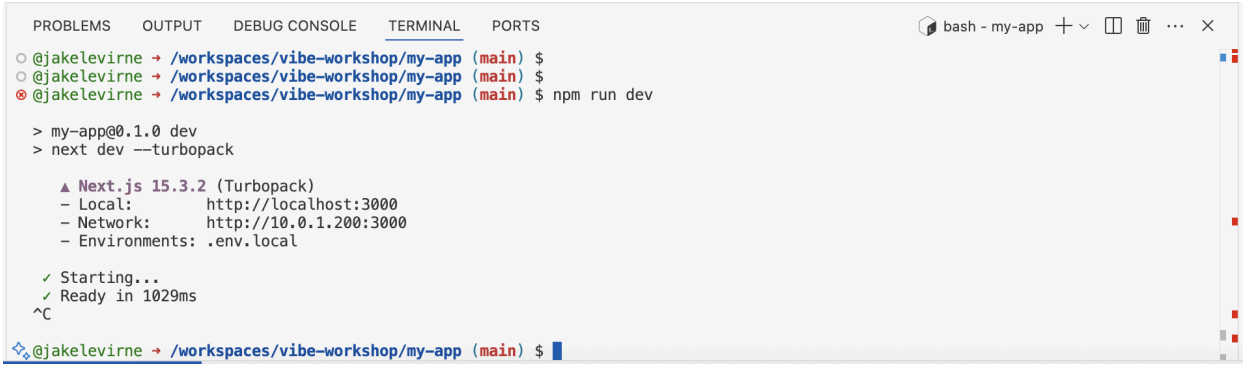
One thing about VSCode that's not obvious– **you can have multiple Terminals open at once.**



If you want to do this intentionally, hit the plus button in the Terminal. But sometimes this happens because the **Agent can open a new terminal for you**. If you want to see your running server (e.g. to view logs), you need to switch to the right Terminal (e.g. "npm my-app"). For any

terminal that's not actively running something, you can safely delete it if you want to keep things organized.

To stop a running command in a terminal, click the terminal to make sure it has focus and then hit Ctrl-C. For example, to stop and restart the server, hit Ctrl-C (which shows as `^C` in the terminal).



And then re-run `npm run dev`.

Adding Logging / Viewing Logs

When you hit bugs and errors, if a quick copy/paste of the error message into the Agent doesn't fix things, then logging can help a lot. Logs just give you a way to see more details about exactly what's happening in your app which you can then share with the Agent to help it troubleshoot.

The first step in getting useful logs is to prompt the Agent to add logging to the part of your app where you're experiencing problems. A prompt like this usually works:

```
add lots of logging to <x>. Output as much information as you can both on the client and server side.
```

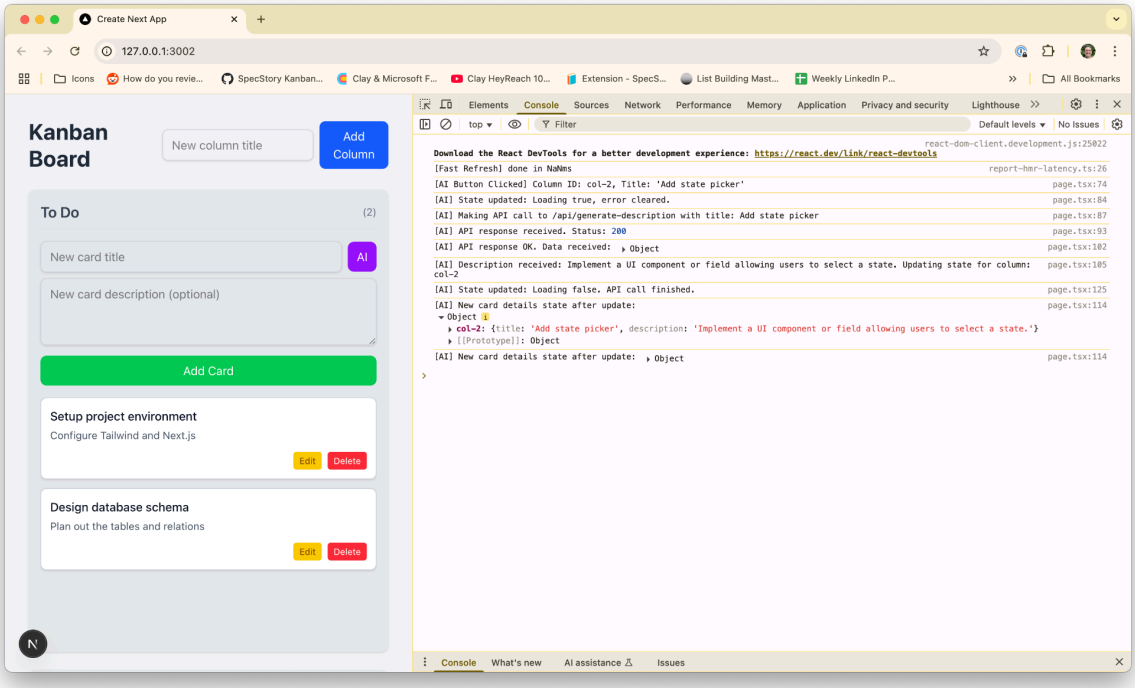
In this prompt, <x> is where you describe the part of your app where you want more logging. (E.g. `add lots of logging to the AI description generator`).

After the Agent adds logs, you need to take explicit steps to see them. NextJS apps have code that runs both on the client (in the browser) and on the server (in your terminal). To see server logs, go to VSCode and look at your Terminal. You may need to scroll up or do a find (Cmd/Ctrl-F) in your terminal to see the relevant output. Also, make sure you're in the right Terminal (see the section above about Managing Terminals in VSCode).

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS 1                    npm - my-app

[API POST /api/generate-description] Received request
[API POST] Request body parsed. Title: Calculate fib
[API POST] Gemini model retrieved: gemini-2.5-flash-preview-04-17
[API POST] Prompt prepared for Gemini: Generate a concise, one-sentence description for a Kanban card with the following title: "C
alculate fib". The description should be suitable for a task management system.
[API POST] Calling Gemini API...
[API POST] Error during Gemini API call or processing: Error: [GoogleGenerativeAI Error]: Error fetching from https://generativela
nguage.googleapis.com/v1beta/models/gemini-2.5-flash-preview-04-17:generateContent: [503 Service Unavailable] The service is curre
ntly unavailable.
    at async POST (app/api/generate-description/route.ts:52:19)
  50 |
  51 |     console.log("[API POST] Calling Gemini API...");
> 52 |     const result = await model.generateContent({
     |                    ^
  53 |         contents: [{ role: "user", parts: [{ text: prompt }] }],
  54 |         generationConfig,
  55 |         safetySettings, {
  status: 503,
  statusText: 'Service Unavailable',
  errorDetails: undefined
}
```

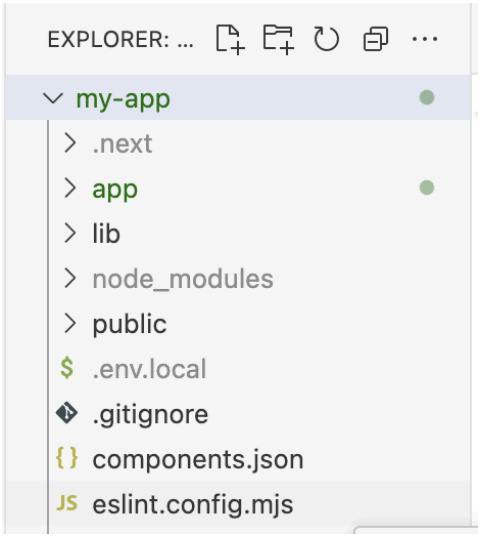
You can copy and paste text, like this log output, directly from the Terminal into the Agent prompt.

For client-side logs, you need to look in the browser. When using Chrome, get the application to the point where you've hit trouble or encountered the error you're working on. Then open Chrome Dev Tools (`View`->`Developer`->`Developer Tools` or three dots menu `More Tools`->`Developer Tools`) and then click on the Console tab. You'll see log output there, with the ability to expand/collapse different elements. Then right-click and Copy Console, or take a screenshot and paste into the Agent prompt.

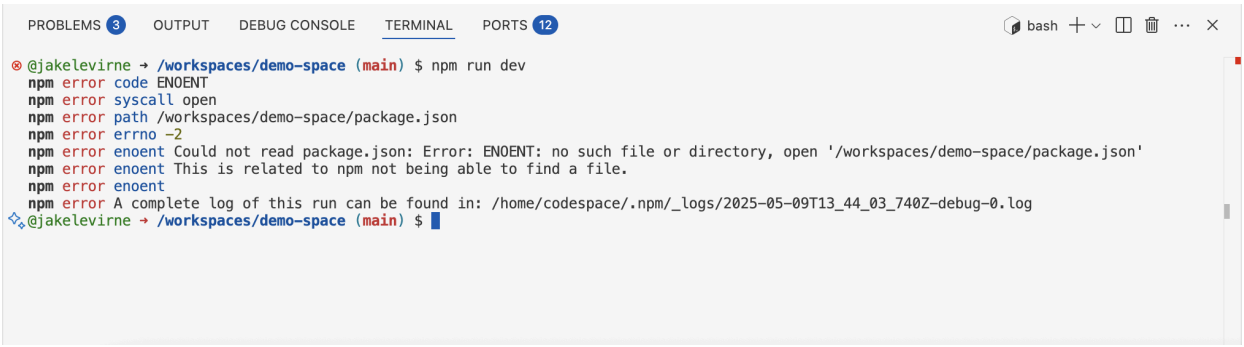


Project Directory Structure

When you first run `npx shadcn@latest init` it reaches a step where it asks you for a project name. If you select the default, it'll create a new directory in your project name `my-app`. This results in a directory structure that looks like this.



When using the terminal (for example to type `npm run dev`, you might get an error that looks like this.



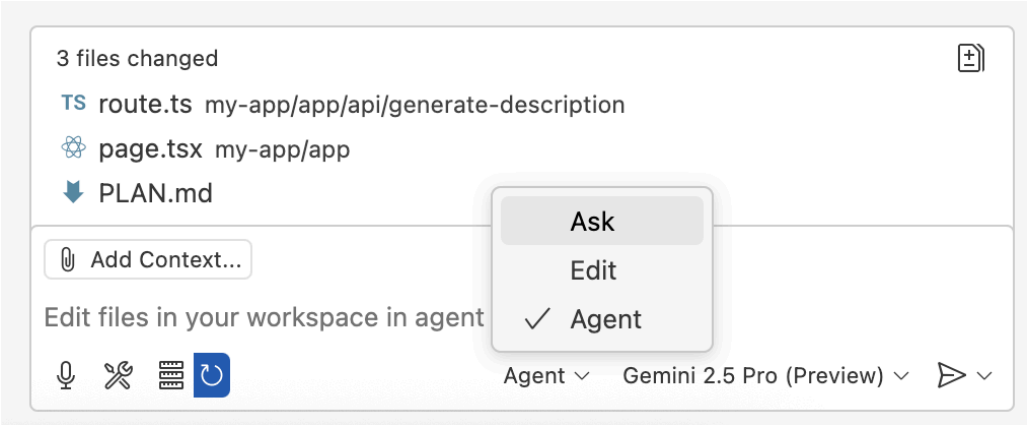
This just means you're in the wrong directory. By default, a new Terminal starts in the project folder. But our application is one level down from the project folder in a directory called `my-app`. So you first need to change directory (`cd`) into the `my-app` folder.

```
cd my-app
npm run dev
```

And things should now work.

Ask/Edit/Agent Modes

VSCode Copilot has three modes: Ask, Edit, and Agent.



You can [read more about them] on the GitHub blog. But the short answer is that Ask mode won't make any changes to your files. Edit mode will make changes to your files but will expect you to take a more active role in reviewing the edits and directing it to specific files. Agent mode will try to do everything for you, including making tool calls, applying code changes, and creating/deleting files. Agent mode is most often what we want to use when prototyping.

Agent didn't make any changes

Sometimes Agent mode says a lot of things but doesn't actually modify any of your code. The Gemini models are especially susceptible to this. If it ever happens, just nudge the agent with a prompt like: `apply these changes to the code`

I don't like the color scheme in VSCode

By default, VSCode uses a dark color scheme. If you want to change it, bring up the Palette (`View`->`Palette` or Cmd/Ctrl-Shift-P) and type/select `Preferences: Color Scheme`

Future

- Advanced: Deploy for easier sharing / feedback - Vercel
    - Vercel makes it easy to deploy a NextJS application. Follow the instructions here and you'll have your prototype running in the cloud with a URL you can share with others.
- Advanced: Posthog analytics on your prototype

- Probably overkill, but you can connect usage analytics to your prototype and see session replays using PostHog. Follow this guide and have the Agent help you add analytics into your code.