CLI steps to scaffold React project in Vite - JUN25

*See notes below: to run up a development version on your machine you will have to regenerate the node modules using **npm install** or **npm i** for short.*

Managed app commands:

> **npm create vite@latest**
OR
> npx create-vite

Follow the prompts

> **cd vite-react-project**
> **npm install**
> **npm run dev**

IF THIS PROCESS FAILS, see separate PDF on updating Node by downloading and running the Windows installer. Similar workflow for MacOS or Linux and more generic than trying to use the command line, which often requires installing OTHER packages in order to perform the update.

To create production directory at /dist

> **npm run build**

To backup project: DELETE /node_modules directory

To reinstall project:

> **npm i**

CORRECTION FOR FILE PATHS TO BUNDLED JS, CSS: ADD TO PACKAGE.JSON:

"build": "vite build --base=./",

Install json-server to mock API

We used DummyJson.com instead, but json-server is more like Postman, and will allow genuine POST operations to be performed.

https://www.npmjs.com/package/json-server

npm install json-server

Start server and point to JSON at top level of project directory:

npx json-server my-json.json

Use **- - watch** flag to broadcast live updates when using POST commands to edit the JSON, e.g.

npx json-server - - watch my-json.json

Use the **-p** flag to change the default port 3000 to one of your choosing, e.g.

npx json-server - - watch -p 4000 my-json.json

It will open a server on localhost:3000. This will only point to the json-server homepage though. To point to your data, include the endpoint inside the JSON object after the port number, e.g.

http://localhost:3000/products for products data

The terminal window in which json-server is running will need to remain open, so use a different one for anything else such as launching the Vite dev server, or adding more packages.

Install axios

% **npm install axios**

axios wraps a Promise chain using Fetch and cuts out the step that converts the response to JSON.

We can log the result of the axios call and see it logs a Promise

You have to deal with async work in the Promise chain, it cannot be referenced synchronously!

BUT we can pass in a callback to the Promise chain and the callback can concern itself with what happens in the DOM.

React useState setter functions asynchronous themselves and may be passed in.

```
useEffect(() => {
    axios
      .get("http://localhost:3000/warehouseState")
      .then((response) => setCarParts(response.data));
  }, []);
```

Response has all the message headers etc, drill down into data for the array of car parts.

Plumbing in Tailwind CSS utility - and never needing to write a line of CSS

We didn't get to plumb in Tailwind as dependency but it is relatively simple and involves

1. **one npm install**
2. **Changes to the vite.config file**
3. **One CSS @-rule in index.css, right at the top level. It should be your ONLY thing in index.css as we have been writing custom styles in App.css instead.**

https://tailwindcss.com/docs/guides/vite

1. npm install tailwindcss @tailwindcss/vite

2. Edit /vite.config file (additions ***in bold italics***)

```
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";
import tailwindcss from "@tailwindcss/vite";

// https://vite.dev/config/
export default defineConfig({
  plugins: [react(), tailwindcss()],
});
```

3. add directive to /index.css

```
@import "tailwindcss";
```

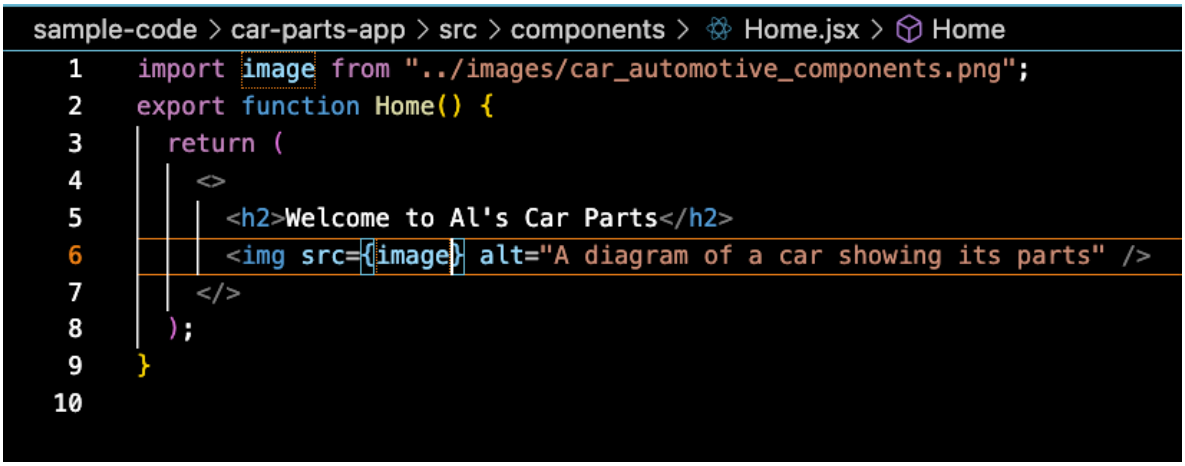
Search "Tailwind cheatsheet" for choices that are up to date, for example

https://tailwindcomponents.com/cheatsheet/

https://www.creative-tim.com/twcomponents/cheatsheet

There's always a lag with cheatsheets behind the current version which is 4

Images in Vite + React



Images need to be iimported into React components as JS variables. Vite will then pick them up and automatically bundle them for production, along with fonts, JS, and CSS.

See https://vite.dev/guide/assets

For complete control over image optimisation in your build use

vite-plugin-image-optimizer https://www.npmjs.com/package/vite-plugin-image-optimizer

Which has itself two optional dependencies:

Sharp https://www.npmjs.com/package/sharp
SVGO (which does claim to optimise SVGs) https://www.npmjs.com/package/svgo

Being from a graphics/photography background, I would always prefer to optimise my images visually, so just the right amount of compression is achieved, and then use those directly in the build in the way described above for React. But that approach doesn't scale well. If I had,

say, 500 images, I would be more inclined to spend a bit of time setting up the config file and risk letting the Vite build process apply the same amount of compression to all.

In development:

In production (note auto-generated filename suffix):