Vite + React + GitHub Project Setup (Option 2 - GitHub First)

This guide outlines the complete and safe workflow for setting up a new Vite + React project, connected to a GitHub repository, without unnecessary folder nesting.

Step 1: Create the GitHub Repo

Go to https://github.com/new

Name your repo: portfolio-refactored

Leave all checkboxes unchecked:

- No README
- No .gitignore
- No license

Click Create Repository

Copy the HTTPS clone URL (e.g., https://github.com/DeToxFox/portfolio-refactored.git)

Step 2: Clone the Repo Locally

Open your terminal:

```
cd ~/Projects  # or your preferred dev directory
git clone https://github.com/DeToxFox/portfolio-refactored.git
cd portfolio-refactored
```

Step 3: Scaffold Vite + React Into the Cloned Repo Folder

```
npm create vite@latest . -- --template react
```

You will be prompted:

Select a framework:

- React

Select a variant:

- JavaScript

Step 4: Install Node Modules

```
npm install
```

Step 5: Push to GitHub

```
git add .

git commit -m "Initial Vite + React scaffold"

git push -u origin main
```

Step 6: Add Tailwind CSS

Install Tailwind and its dependencies:

```
npm install -D tailwindcss postcss autoprefixer @tailwindcss/postcss
```

If you encounter an error using npx tailwindcss init -p, skip it and manually create the config files below.

Manually create tailwind.config.js:

```
/** @type {import('tailwindcss').Config} */

export default {

  content: [

    "./index.html",

    "./src/**/*.{js,ts,jsx,tsx}"

  ],

  theme: {

    extend: {},
  },
  plugins: [],
}
```

Manually create postcss.config.js (modern plugin setup):

```
import tailwindcss from '@tailwindcss/postcss';
import autoprefixer from 'autoprefixer';

export default {
  plugins: [tailwindcss, autoprefixer],
}
```

Update src/index.css:

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Ensure index.css is imported in main.jsx:

```
import './index.css';
```

(Optional) Remove default Vite boilerplate styles in index.css or App.css.

(Optional) Install Tailwind IntelliSense in VS Code:

- Open Extensions (Ctrl+Shift+X)
- Search for Tailwind CSS IntelliSense
- Install and restart VS Code if needed

Step 7: Push Tailwind Changes to GitHub

```
git add .
git commit -m "Add Tailwind CSS setup"
git push
```

Step 8: Install react-icons

```
npm install react-icons
```

Outcome

- Your project is scaffolded with Vite + React (JavaScript)
- Clean folder structure (no nesting)
- Fully connected to your GitHub repo
- Tailwind CSS is installed and working
- VS Code recognizes Tailwind classes with IntelliSense

Next Steps: Add components, layout sections, and deploy to Netlify or Vercel.

Step 9: Install react-router-dom (Required for Routing)

To enable routing in your app (navigation between sections like Home, Contact, etc.),
install the `react-router-dom` package:

```
    npm install react-router-dom
```

This allows usage of components like <BrowserRouter>, <Routes>, and <Route> inside your App.jsx.

Then push changes to GitHub:

```
    git add .
    git commit -m "Install react-router-dom for routing"
    git push
```

■ Additional Setup & Deployment Notes (Updated)

■ Installed NPM Packages:
• npm install -D netlify-cli – for local testing and Netlify function support.
• npm install concurrently – to run both the dev server and backend together.
• npm install dotenv – to support environment variables.
• npm install axios nodemailer express cors – for backend messaging and frontend handling.
■ Key Files Added:
• netlify/functions/send.js – Netlify Function for contact form email submission.
• netlify.toml – specifies Netlify's functions directory and build output.
• backend/server.js – local Express server used during development.
■ Deployment Behavior:
• For local testing: npm run dev runs both frontend and backend on ports 5173 and 5000.
• For Netlify testing: npm run netlify:dev runs full site on port 8888 using serverless functions.
• Netlify will automatically deploy when changes are pushed to the linked GitHub repo.
■ Endpoint Handling in Contact.jsx:
• Dynamically switches between local server and Netlify functions:

```
const endpoint = import.meta.env.MODE === 'development' ? 'http://localhost:5000/send' :
'/.netlify/functions/send';
```

■ Notes:
• You can continue using your local Express backend for development even after deploying Netlify Functions.
• No need to redeploy for every change — test locally first, then push to GitHub when ready.
• Netlify 'Post-processing: In progress' is normal briefly, but investigate if it lasts over 10 minutes.