WiT Hackathon Web App Starter User Guide

Contents

🧩 Step 0: Choose Your Tech Stack

💡 What's a Tech Stack?

A **tech stack** is the set of tools and technologies you use to build your app — like ingredients in a recipe. It usually includes:

- A **frontend** (what users see),
- A **backend** (how data is processed),
- And a **database** (where data is stored).

Recommendations

✅ If You're New to Coding:

Use: **Next.js + MongoDB**

- You'll write everything in JavaScript, which is beginner-friendly.
- Next.js lets you build both the frontend (what users see) and backend (how data is handled) in one place — no need to set up a separate backend.
- MongoDB is used to store your user data.
- Here is a **Next.js starter project** , you can use to kickstart your project —follow the steps in the README to get set up, plug in your features and go!

✅ **If You Have Coding Experience:**

Use: React + Java or Python backend + MongoDB

- React handles the frontend.
- Java or Python can be used to build a separate backend.
- MongoDB stores your data.
- Here are starter projects for React and Java/Python which you can use to kickstart your project —follow the steps in the README to get set up, plug in your features and go!

📘 We have User Guides available to help you set up your app using either of these options.

In this guide, we'll walk you through building the **frontend** of your application using **React and JavaScript**.

What is React?

React is a popular tool that helps you build websites. It lets you create small parts of a website called components - like buttons, navbars, or forms - that you can reuse. This makes building websites easier and faster.

What is Javascript?

Javascript is the programming language that makes websites interactive. It lets buttons, forms, and content react when users click, type or scroll -basically, it makes your React app work and respond to users.

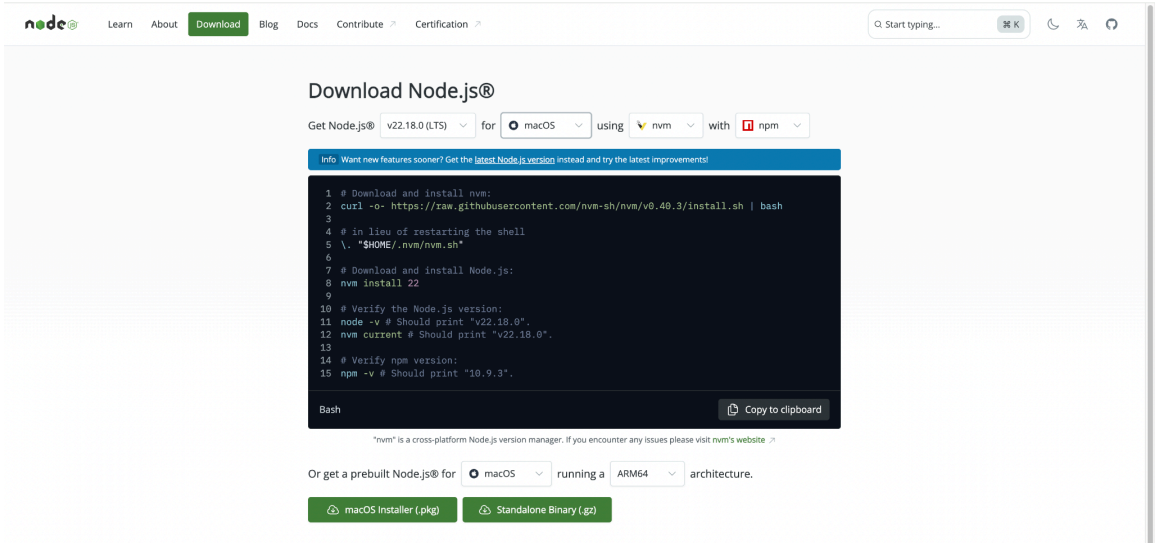
Using this guide, you'll learn how to use React and JavaScript together to build a web application for your hackathon project.

Building your First React Web App

Step 1: Install your tools

Before, you start, you need two things:

1. **Node.js**
    a. Lets your computer run JavaScript outside of a browser and install packages like React.
    b. Go to nodejs.org and download the LTS version (recommended for most users). Follow the steps provided.

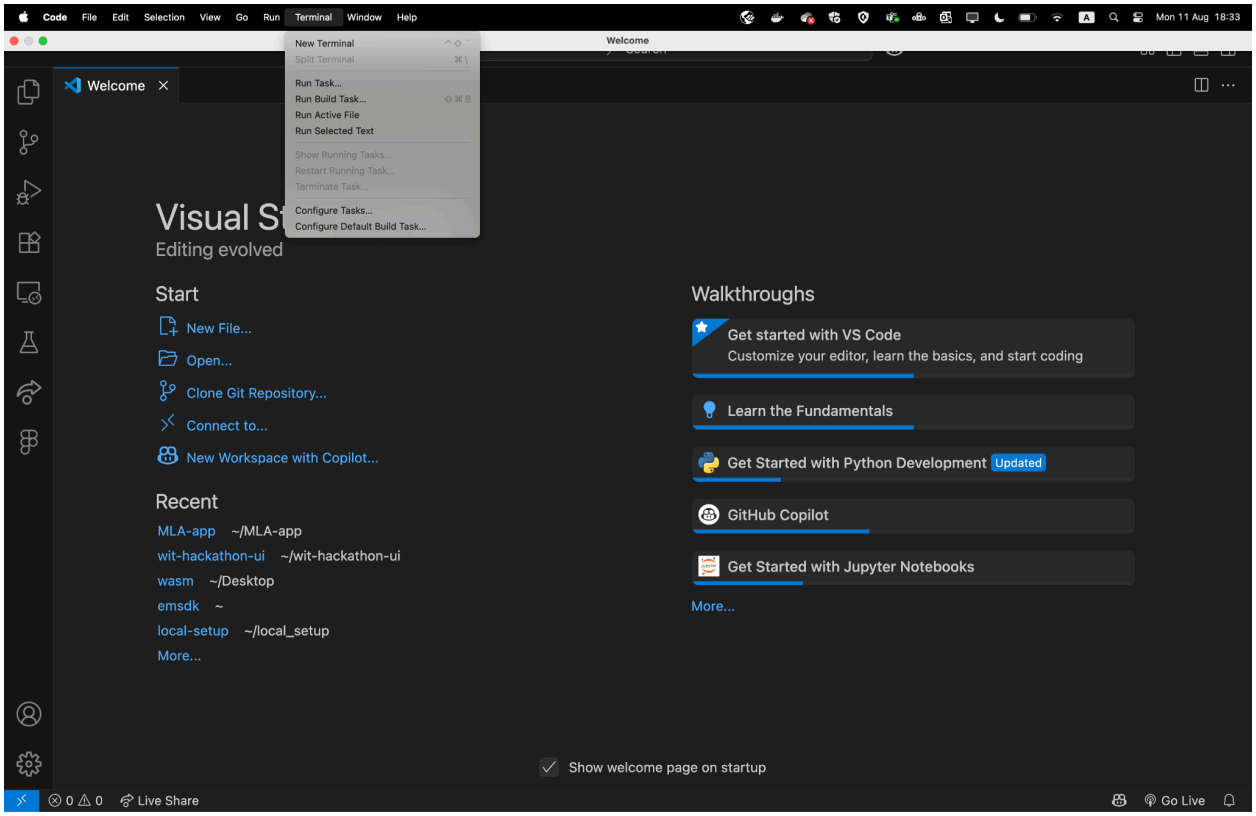


2. **Visual Studio Code (VS Code)**
    a. What it is: A free code editor made by Microsoft. It is where you will write, organise, and run your code - like Microsoft Word, but for coding.
    b. Download it from code.visualstudio.com and install it.

Step 2: Create a new React app

To do this, we will use **Create React App,** a tool that sets up a React web application for you, without any manual configuration.

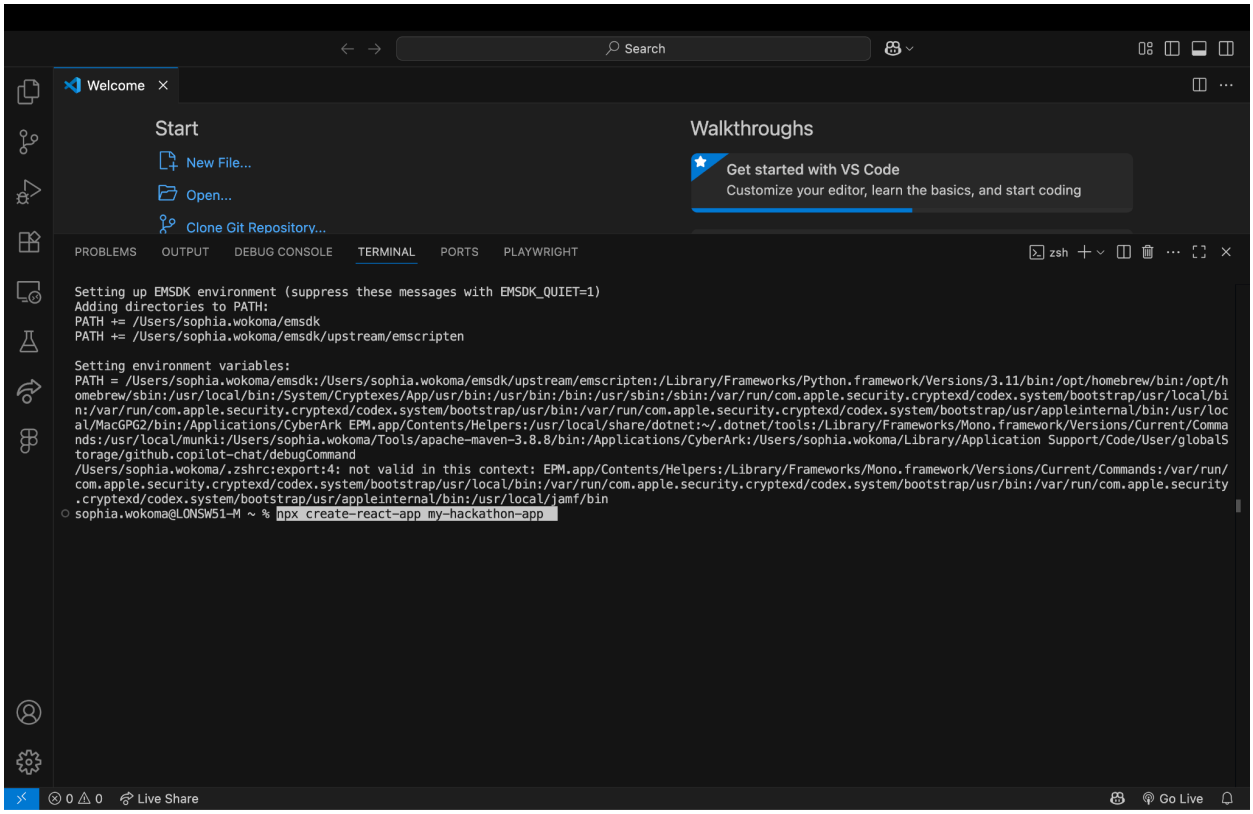
1. Open up Visual Studio Code, launch a terminal (select New Terminal).



In the terminal, run:

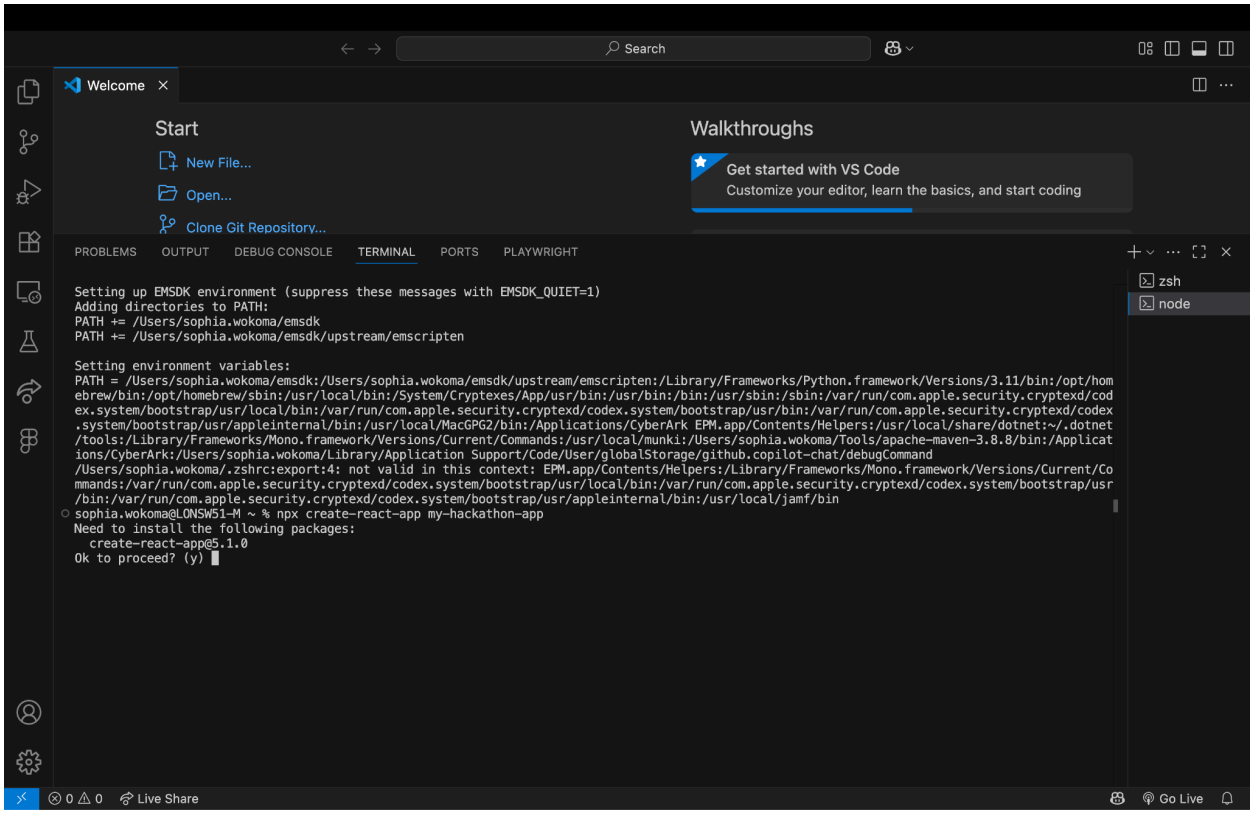
- Paste it in and hit **Enter** to run
- Feel free to change the name of your application , we have used **my-hackathon-app**

```
npx create-react-app my-hackathon-app
```

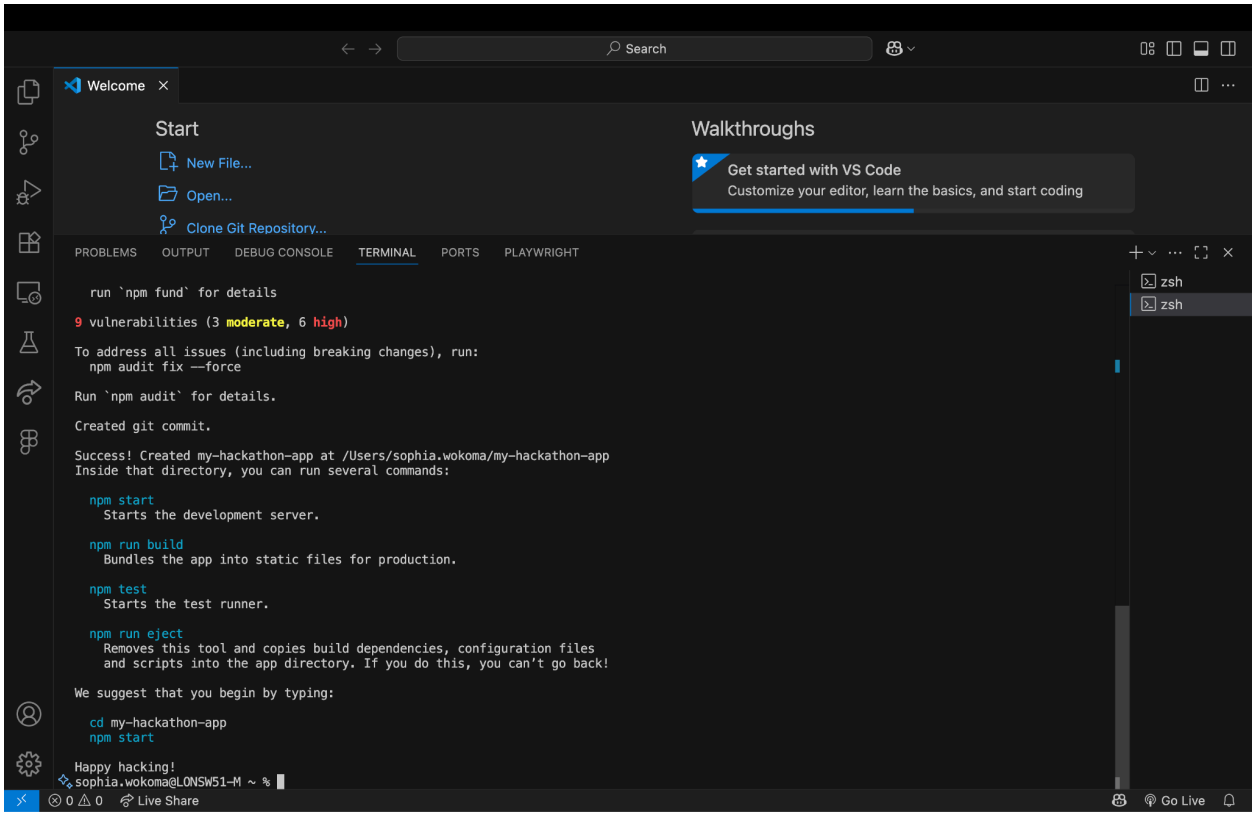


You will probably be asked to install the **create-react-app** package, if you have not before:

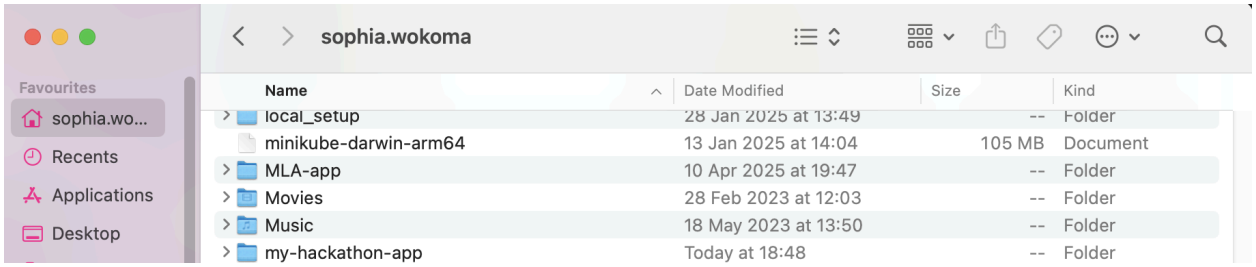
Type in **Y** and then hit **Enter**



Once installed, this will create a folder named **my-hackathon-app** (you can choose a different name) with all the files you need. You should see a success message similar to the one below .



To confirm, you can also check in your Finder / File Explorer and you should see the my-hackathon-app or the name of your folder has been created for you.



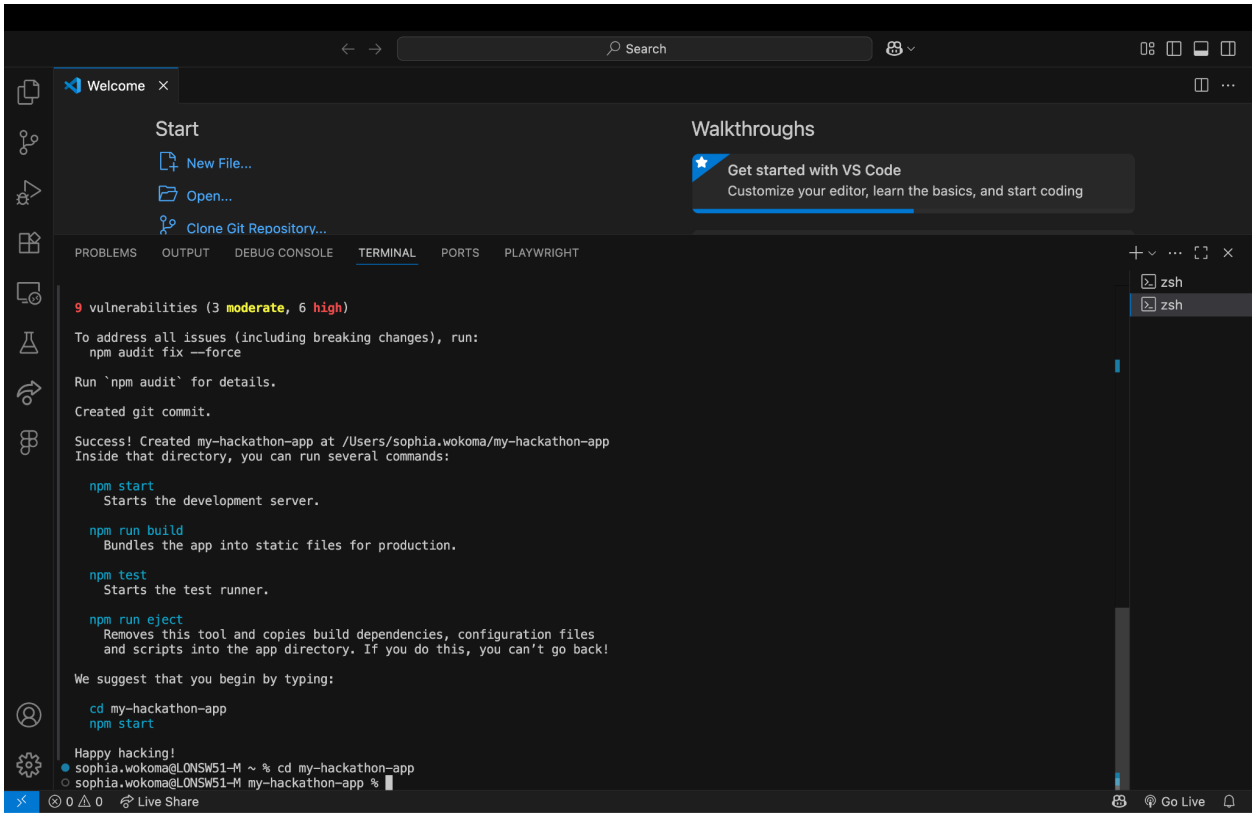
Step 3: Start your React app

Now that we have set up the app, let's run it, so we can see it in the browser !

In the same VS Code terminal, navigate into your app folder:

```
cd my-hackathon-app
```

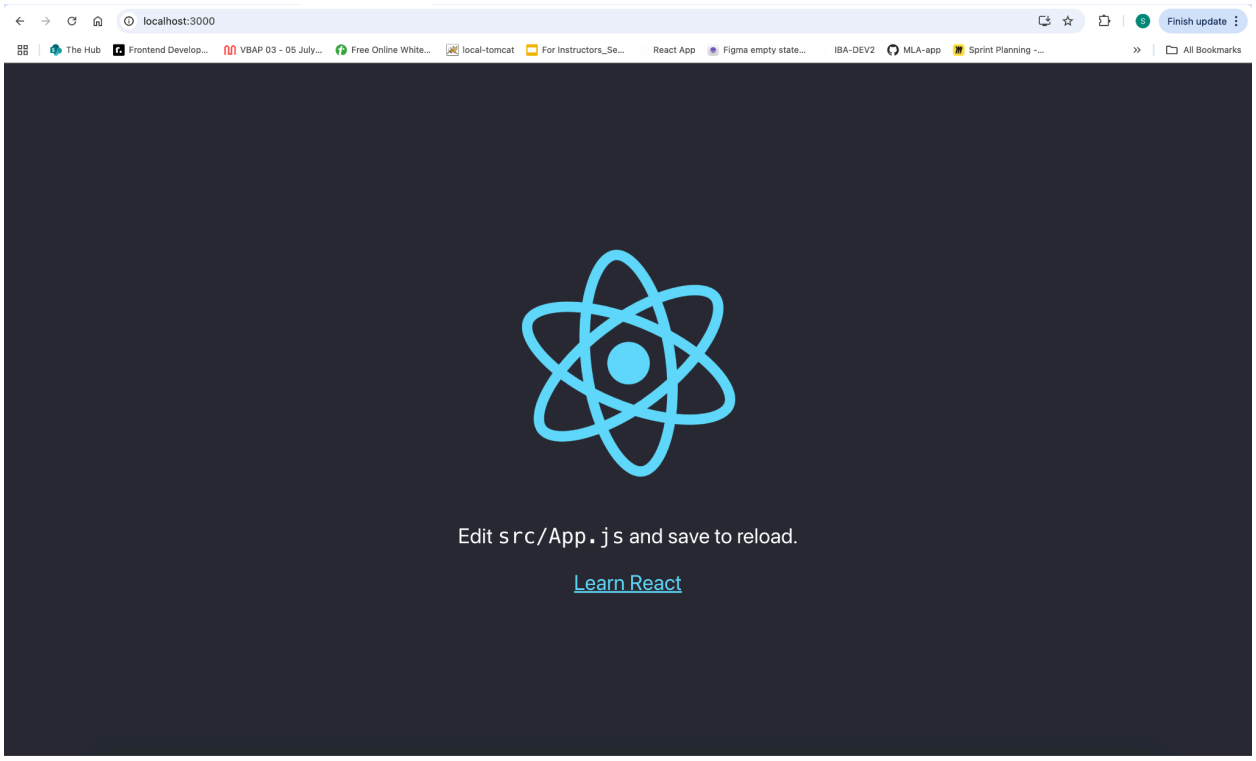
*Replace **my-hackathon-app** with your folder name if different.*



Start the app by running:

```
npm start
```

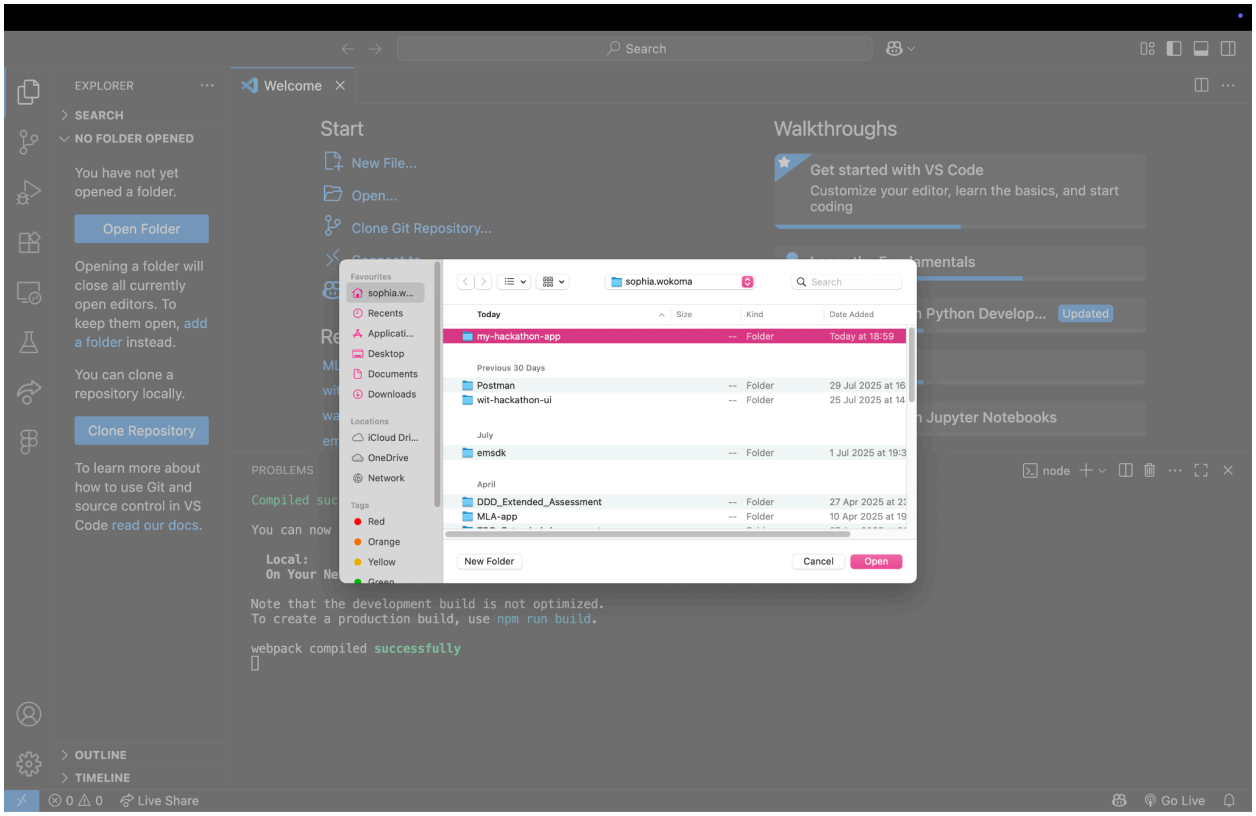
Your browser will open at http://localhost:3000 , showing your running React app !



Step 4: Understand the basic folder structure

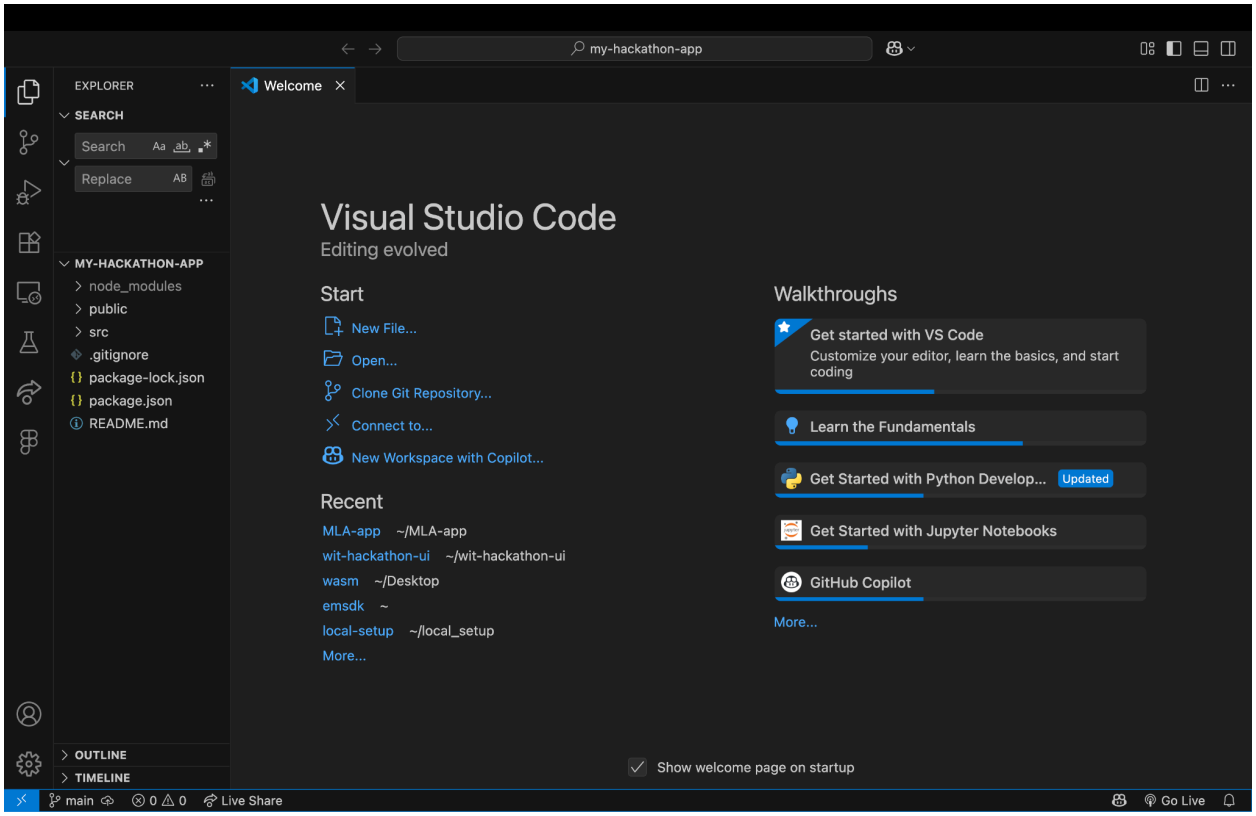
Open the my-hackathon-app in VS Code. To do this you can:

1. Click Open Folder on the ExplorerPanel on the left or the Open Link in the Start menu items in the workspace
2. Select the my-hackathon-app or whatever folder you created for the project and open.



This will open the folder in Visual Studio code with all the initial files that the create-react-app has provided. In the Explorer panel on the left you should see the my-hackathon-app / your folder name as a dropdown. If you open the dropdown you will see all the files and folders which have been created to help you organise and get started on your React web app.

Here is a simple overview of the main files:



- **node_modules folder**
  - Contains ready-made code (called packages) that your app uses.
  - A package is like a toolbox of pre-written code - you can install and use without building it yourself.
  - You don't need to open or change anything here - just know it's important for your app to run!
  - If you decide to install extra packages they'll be saved in this folder automatically.
- **public folder**
  - Acts as the backdrop of your website.
  - Contains the main page (index.html) that your browser shows.
  - You can also put images, logos or icons here that don't change often.
- **src folder**
  - This is where you write your app's code.
  - You create small pieces called components here that together build your website.
  - The main file/ component is usually named **App.js**, which is like the heart of your site.

- **package.json**
  - Lists all the tools and packages your app needs to work.
  - Contains built-in commands (instructions given to the computer to perform a specific task), like **npm start** which you have already used to run your app.

- **index.html (inside public/)**
  - You will find in it , this HTML code `<div id="root"></div>`
  - This is the container where React renders your app.
  - Think of it like an empty stage: the structure exists but nothing is shown until React puts components there**.**
- **index.js (inside src/)**
  - This is the entry point for your React app.
  - Tells React **where to start** and which component to render first (usually App.js)
- **App.js**
  - The top level component in your project
  - Acts as the parent component for any components which you later create to build applications , e.g like Home, About , Navbar, Footer, Button etc
  - You can then import these components in the App.js as well as each other and other resources such as images, styling files (CSS files), packages in them to use in your UI.
  - React builds a component tree starting from App.js and renders it in the browser.

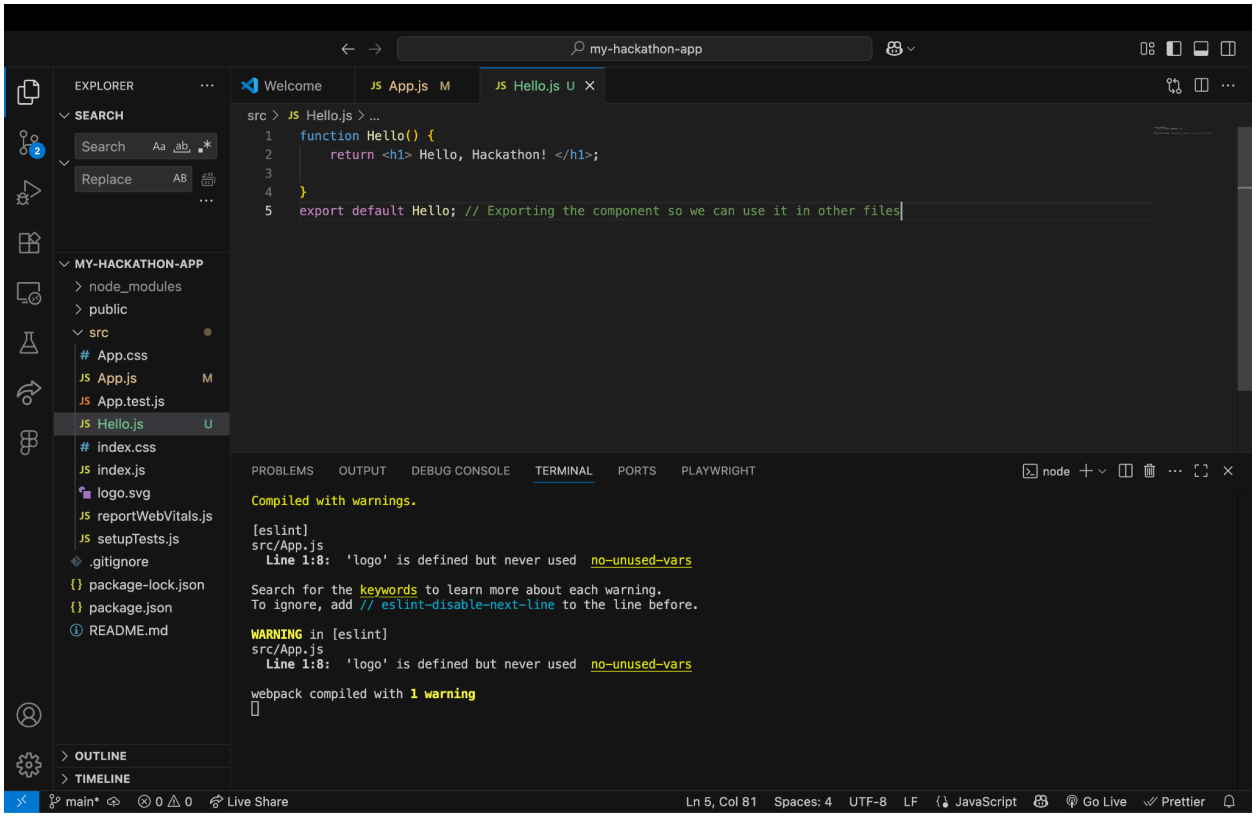
Step 5: Create your first React component

A component is like a building block of your website. Let's make a simple one!

1. Open the **src/** folder in your project.
2. Create a new file (Right Click on **src** + Select **New File**) called **Hello.js.**

3. Copy and paste this code into Hello.js
4. Save Changes (Ctrl/ Cmd + S)

```
function Hello() {
      return <h1> Hello, Hackathon! </h1>;
}
export default Hello;
```

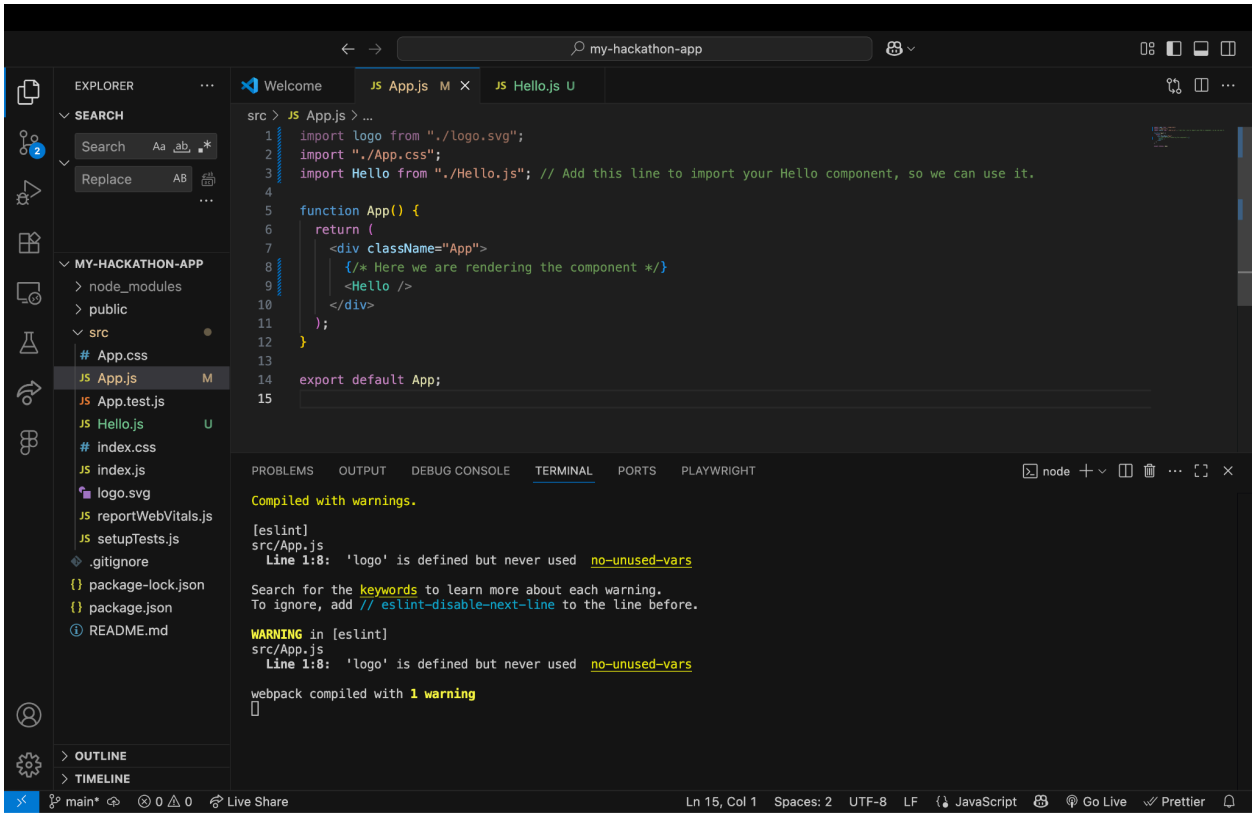


5. Now, open the **src/App.js** file
6. Import your **<Hello> component** at the top of the file. Importing in React, is where you bring external resources e.g other components, CSS files, images into your current file so you can use them in your UI. Here we are importing the Hello component we just created into our App.js file which is the parent file / component,

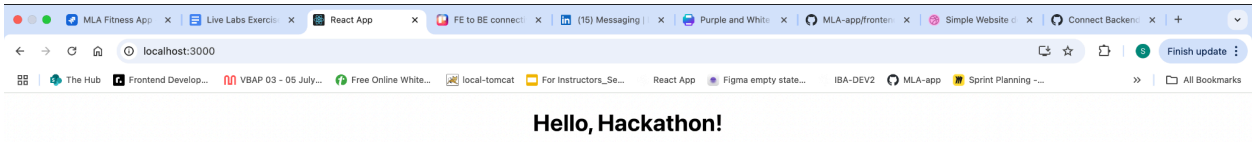
which will allow us to actually see the changes Hello component provides in the browser.

```
import Hello from './Hello';
```

Replace everything inside the ** ** with your new component:



Save your files (Cmd + S for mac / Ctrl + S for Windows), and your browser should automatically refresh and show, Hello Hackathon!

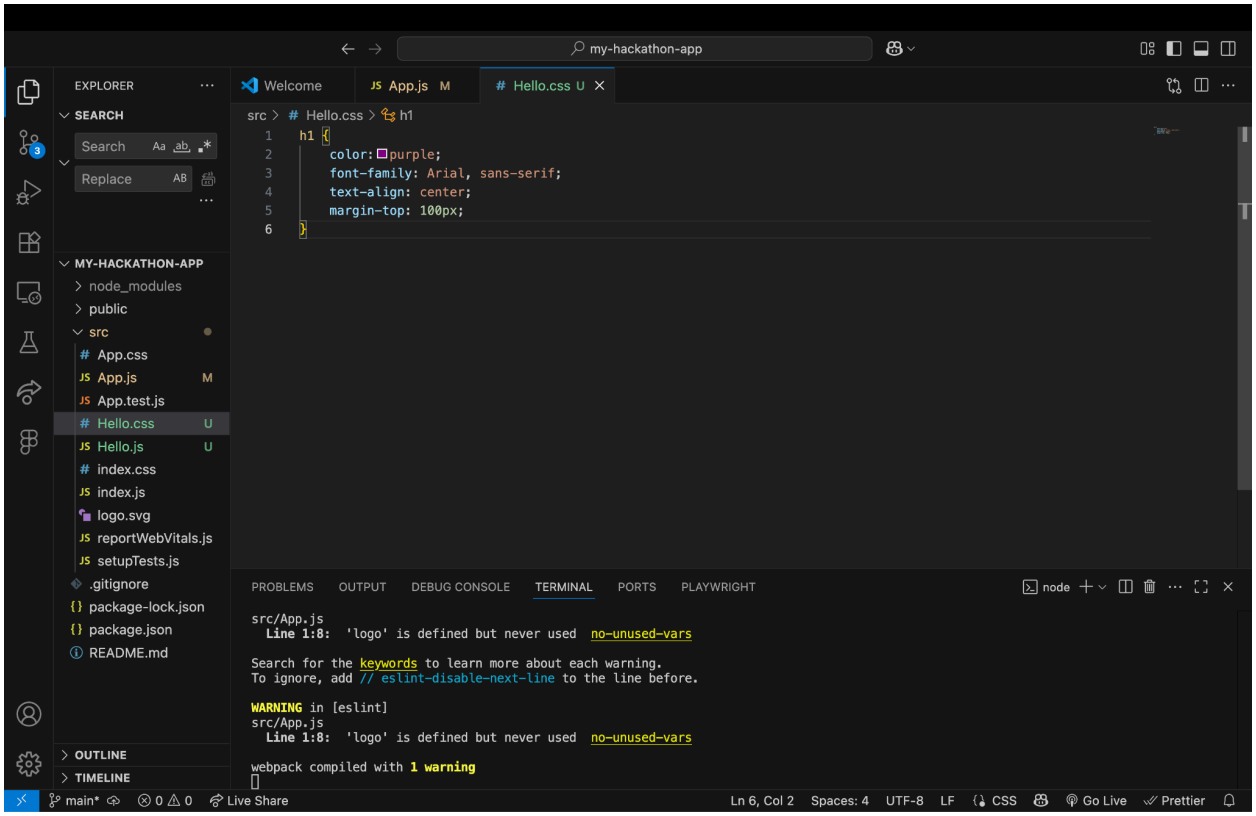


Step 6: Add styles to your React component

Let's make your greeting look nicer by adding some styles.

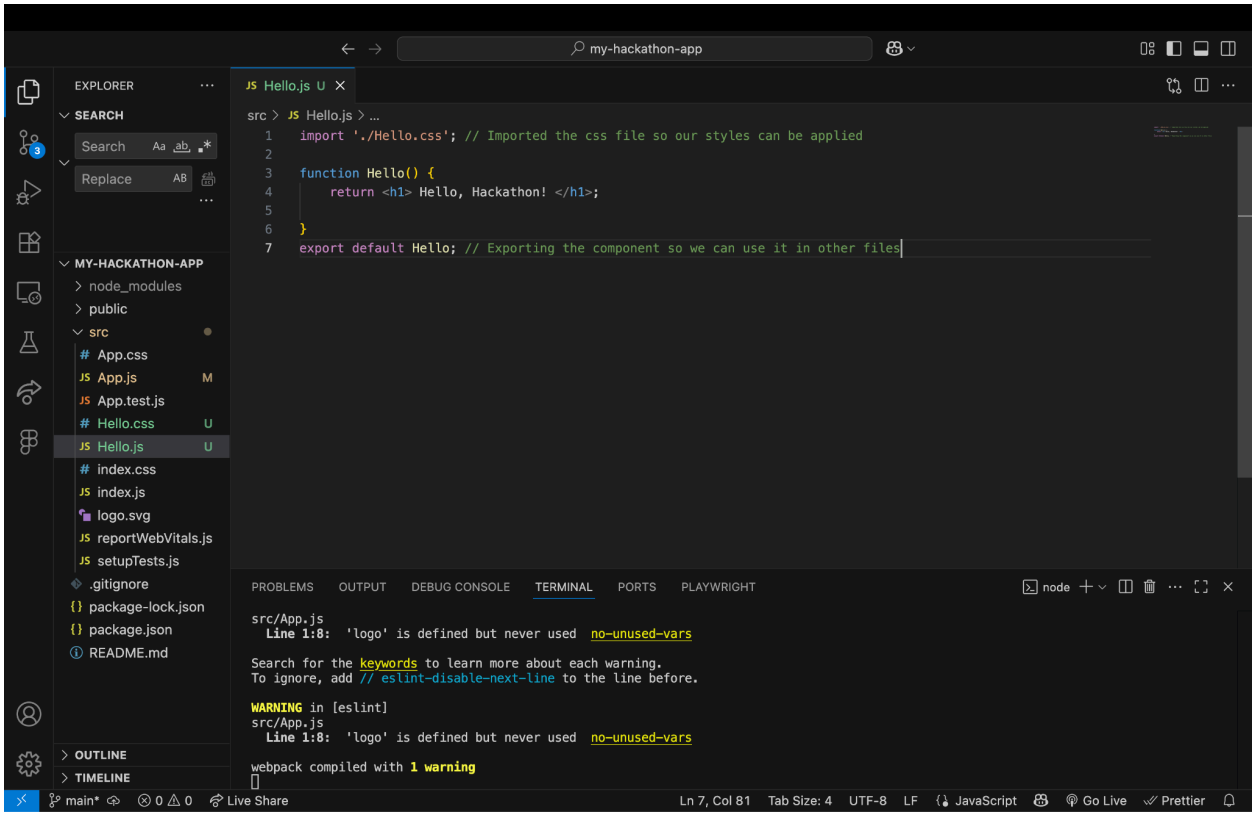
In the src/ folder , create a new file called **Hello.css**.
Add some styles inside Hello.css and save your file.

```
h1 {
      color:purple;
      font-family: Arial, sans-serif;
      text-align: center;
      margin-top: 100px;
}
```



Go back to your **Hello.js** file.
At the top, import the CSS file:

```
import './Hello.css'; // import our new css file
```



What This Will Do

This will style the <h1> heading in our Hello.js component so that:

- The **text is purple**
- The **font is Arial**
- The text is **centered on the page**
- There is **space above the heading** (using margin)

Step 7: Using AI to build the Rest of Your React App

7.1 Understand What AI Can Do For You

Think of AI as a helpful teammate. It can:

- Write code for you.
- Explain code in plain English
- Help fix errors.
- Suggest improvements and new features.

You don't have to know the right words - **just clearly describe what you want.**

7.2 Defining Your Application

Before coding, describe your app in plain English.

- **Pages:** What pages will it have?
- **Components:** What are the building blocks of each page? Think of these as the pieces of the page that do one job - like a recipe card, a button, or a navigation bar.
- **Functionality:** What should each component do ?

Example:

- A "RecipeCard" shows the title, ingredients, and instructions.
- A SearchBar lets users type and filters recipes.

Tips:

- Start simple with one core feature.
- Be detailed - describe how users interact with your app and the value you want each feature to provide, describe as if you're writing for a blind person
- Aim for 2-3 paragraphs for your initial overview .

Action: Open up a Word Doc / Note Editor and write a detailed overview of what you

want to build. Feed this into your AI tool and ask for feedback or improvements.

When you have refined the idea to a state that you are happy with, ask your AI tool something with a prompt, here are some examples of how to structure it:

Example if completely starting from scratch.

```
Ok, I'm ready to begin building. I would like to use React and
JavaScript for the frontend. Please suggest a beginner-friendly
backend framework and database to go with it, and explain briefly why
you recommend them.

Then, guide me on how to set up the folder structure so that the
frontend and backend code are kept separate but can still work
together.

I will begin building on my local machine using [coding editor].
Please give me clear, step-by-step instructions because I am a
complete beginner.
```

Example if using the Starter Project provided.

```
Ok, I'm ready to begin building. I am using React and JavaScript for
the frontend. The backend is a Spring Boot (Java) application
connected to MongoDB (NoSQL database), and it includes a health check
using Spring Boot Actuator.

The current backend endpoints (ways the frontend can talk to the
backend) look like this:

// Customers API
export const fetchCustomers = () => api.get('/customers');
export const fetchCustomerById = (id) => api.get(`/customers/${id}`);
export const addCustomer = (payload) => api.post('/customers',
payload);
export const deleteCustomer = (id) => api.delete(`/customers/${id}`);
```

```
// Products API
export const fetchProducts = () => api.get('/products');
export const fetchProductById = (id) => api.get(`/products/${id}`);
export const addProduct = (payload) => api.post('/products',
payload);
export const deleteProduct = (id) => api.delete(`/products/${id}`);

Please guide me on:

How to use these APIs in my React frontend (e.g. fetching data,
displaying lists, handling forms).

How to extend or modify the backend if I need extra endpoints (new
ways for the frontend to get or send data) to support my app idea:
[INSERT APP IDEA HERE].

How to enhance the project with new features such as: [INSERT YOUR
FEATURE IDEAS HERE].

Be highly descriptive in your explanation, as I am  a complete
beginner.
```

Depending on your approach, whether starting from scratch, or using the starter app provided, the AI tools should give you a detailed overview, if you have any preference on language or tech used, say so, but if not, you can let your AI tool choose it for you.

7.3 Start Building

Focus on building out the core functionality first.

For example if you envision a web app with a login page that, once logged in, allows users to visit a page that allows them to upload and process images, then focus first on the processing of those images or documents, and come back to the login flow and home page later.

The core of your application (what it actually does) is what is most important and will be the most fun but time consuming to build !

Once that is done, then go back and build the other user flows around that core.

Prompt Template

```
I am building a [type of app].
I want a [feature] that does [what you want it to do].
I am using [ React version or tools, if you know them].
Please write code for me and explain it like I am a beginner.
```

Example using Template:

```
I am building a React recipe app.
I want a "RecipeCard" component  that shows a recipe name, image and
ingredients list.
I am using create-react-app.
Please write the code and explain each part in beginner friendly
terms.
```

7.4 How to Copy AI Code into Your Project

- Step 1: Copy the code from AI .
- Step 2: Create a new **.js** file in your src folder (like RecipeCard.js)
- Step 3: Paste the code there
- Step 4: Import and use it in your main App.js file or parent component

Example in App.js

```
import RecipeCard from './RecipeCard';
function App() {
     return (
       <div>
```

```
          <RecipeCard/>
        </div>
    )
}

export default App;
```

7.5 Ask AI to Fix Errors

1. **Locate the error**, there are a few places where you can see errors displayed when programming.

- **Problems tab** (VS Code) → Shows syntax errors in your code and points to the line where the issue occurs. Open it by clicking the ⚠️ icon at the bottom of VS Code or pressing Ctrl+Shift+M (Windows) / Cmd+Shift+M (Mac).
- **Terminal** (VS Code) → Shows runtime errors when you run your code. Open it with Ctrl+` (backtick) or Cmd+` (Mac).
- **Browser console** (e.g, Google Chrome) → Shows errors when your website runs in the browser. Open it by right-clicking the page → Inspect → Console tab (or press F12 / Cmd+Option+J on Mac).

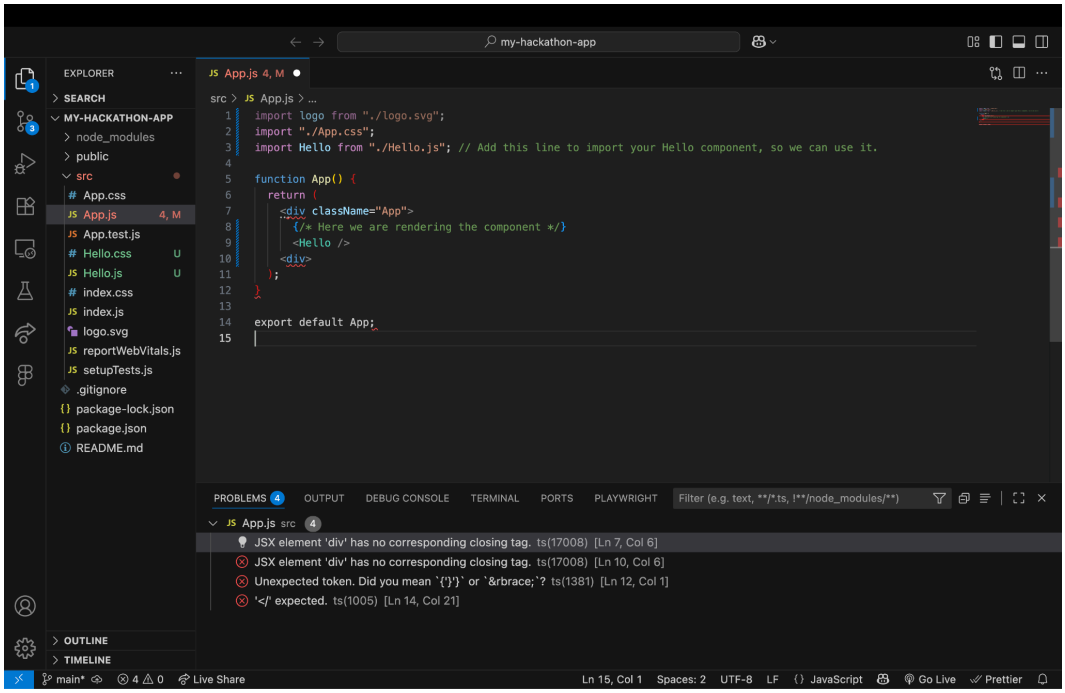
2. **Ask AI for help**: Copy the error message and the relevant code, then ask:

   "Here's the error, can you help fix it?"

   - If AI can fix it, update code to reflect changes
   - If AI can't fix it: Search the error online, including the programming language which is used (e.g. Javascript) in the query. Forums (e.g. Stack Overflow) and knowledge bases often have solutions.
     - Refine with AI: Take insights from your search and provide them back to AI: **"With this additional info and the error I shared earlier, please help me resolve it."**

Pro tip: The line highlighted in your Problems tab is usually the first place to check—it often shows the simplest fix.

Example of Error



Example Error Prompt

```
I got this error in my React app: "JSX element ‘div’ has no
corresponding closing tag ts (17008) [Ln7, Col 6]"
Here's the code from App.js : [paste your code]
Can you tell me what's wrong and how to fix it ?
```

7.6 Iterate and Improve

Don’t aim for perfection first. You can make small changes as you go along like:

```
"Make the button bigger and blue."
"Add a search bar at the top."
"Change the font to something stylish."
"Make it work on mobile screens."
```

7.7 Tips for Working With AI

AI tools can be incredibly helpful when building your app—especially if you're new to coding. Here's a step-by-step guide to working effectively with AI throughout your project.

🧠 Step 1: Plan What You Want to Build

Start with a clear idea of what your app should do.

Tips:

- Use wireframes or sketches to visualize each page.
- Write down the main features (e.g., log workouts, view history, track progress).
- Don't worry about technical details yet—just describe the user experience.
- Follow our UX/UI guide which goes through all these steps here

---

💬 Step 2: Talk to AI Clearly

You don't need to know code—just explain what you want in plain English.

Tips:

- Be specific about what you want the app or page to do.
- Ask for step-by-step guidance.
- If something seems off, ask AI to double-check or only give highly confident answers.

📄 Step 3: Build Each Page

Use your wireframes as a guide. Build one page at a time.

Prompts to try:

- "Help me build a workout logging form with a Save button."
- "Create a table that shows past workouts with date and duration."

Tips:

- Ask AI to explain the code it gives you.
- Review the code before using it—don't copy blindly.

⚙️ Step 4: Add Interactivity

Make your app respond to user actions (e.g., saving a workout).

Prompts to try:

- "When someone clicks Save, I want the workout to be stored in MongoDB."
- "Show a message that says 'Workout saved!' after clicking Save."

🧩 Step 5: Build in Small Chunks

Avoid overwhelm by adding features gradually.

Tips:

- Start with one component (e.g., a form or a table)
- Add functionality step by step
- Once it works, improve the design and layout

🧪 Step 6: Test and Debug

Use your app like a real user. If something breaks, ask AI for help.

Best way to ask:

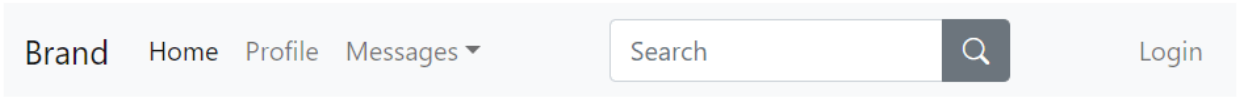
- Describe the problem
- Share the code
- Say what you expected
- Include any error messages

Example prompt:

> "I built a form in Next.js. When I click Save, nothing happens. Here's my code. I want it to save the workout to MongoDB. Can you help me fix it?"

Quick Prompts for Common Features

Navbars



A navbar (short for navigation bar) is like the menu at the top of a website or app. It helps people find their way around by showing links to the main pages - for example:

- Home (the main page)
- About (information about the site)
- Contact (how to get in touch)

Example Prompt:

```
Create a Navbar component in React that shows links for Home, About, and Contact at the top of the page.The navbar should have a dark background, white text, and a hover effect that changes the link colour to light blue.Make it responsive so that on small screens it turns into a hamburger menu that can be clicked to show the links.Please explain the code in beginner-friendly terms.
```

You can use and adapt this prompt to generate a Footer.

Buttons

A button is something on a website or app that you can click which typically prompts some sort of response

When you press a button, it might:

- Send a message or information you typed in
- Take you to another page or part of the app

Example Prompt:

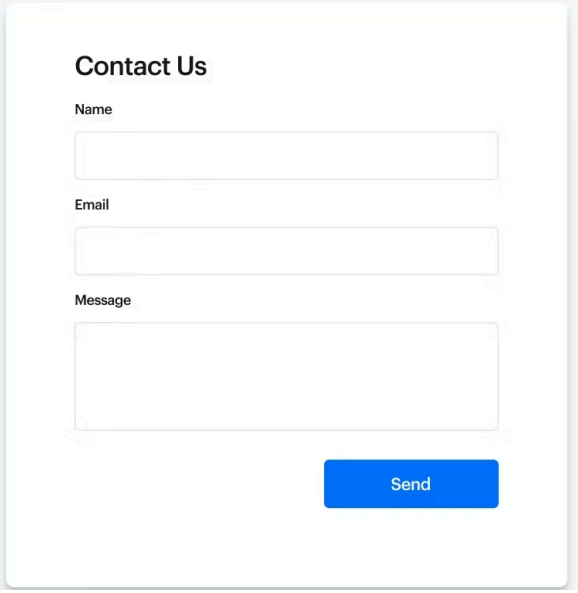
```
Create a Button component in React that is large, rounded, and has a background colour of bright blue with white text.When hovered, the button should become a darker blue and slightly grow in size. Please explain the code in beginner-friendly terms.
```

Forms

A forms are great tools to use when you want to collect user information.

It usually has:

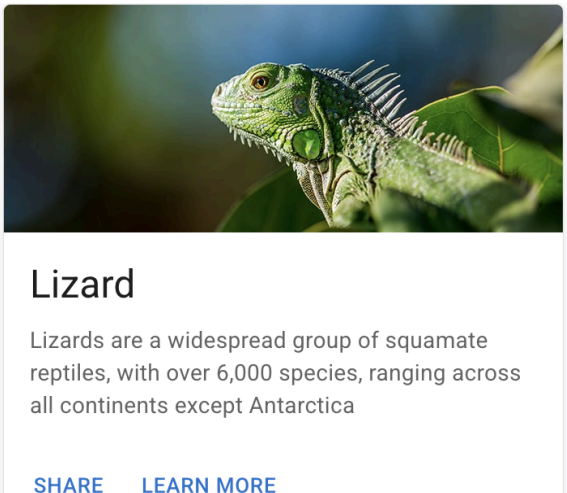
- Boxes to type things in like name or email
- Buttons to send the information.
- Validation - checking that what someone types in a form is correct before sending it.



Example Prompt:

```
Create a ContactForm component in React that has a box where people can type their name, another box for their email, and a bigger box for their message. There should also be a send button. When someone clicks send, it should check that they typed something in the name, email, and message boxes, and that the email looks like a real email. Please explain the code in beginner-friendly terms.
```

Cards



These are like information boxes. It usually has a neat rectangle shape, and inside you might find:

- A picture (like a product photo)
- A title (like the name of the item)
- A short description

- Maybe a button or a link (like “Buy Now” or “Read More”)

They make it easy to display small chunks of information, in a clean, organised way.

Example Prompt

```
Make a Card component in React for my Product Page that shows a
title, product image, and description with a "Buy Now" button at the
bottom.
```