WEB102 | Intermediate Web Development

Intermediate Web Development Summer 2025 (@ Section 1 | Wednesdays 4PM - 6PM PDT)
Personal Member ID#: **107789**

Week 6: Lab 5 - Crypto Hustle Lite

Overview

In this lab, you will develop an app that displays information about cryptocurrencies, such as their value, market cap, and percent change in value, which you will access using API calls. You will also implement a search bar that allows users to search for a specific currency.

View an exemplar of what you'll be creating in this lab here!

Required Features

- User can view a list of at least 30 cryptocurrencies, including the image, name, and price of the coin in US dollars
- User can search for a specific coin in the list of cryptocurrencies by symbol

Stretch Features

- User can view a list of cryptocurrency scams on a separate pane in the page

🎯 Goals

By the end of this lab you will be able to...

- [ ] Create an API key
- [ ] Read API documentation to understand how to make requests, either in GitHub docs or using the examples on the organization's website
- [ ] Use the `useEffect()` hook as a replacement for previous mount/unmount functions
- [ ] Fetch API data with async/await syntax
- [ ] Use props to send variable data to a component for use in a variety of use cases
- [ ] Use conditional rendering, either with if statements or ternary operators, to display information on the page only when necessary and return to a default state if not
- [ ] Additionally practice using CSS to format pages

Resources

- CryptoCompare Docs
- CryptoScamDB Docs

Lab Instructions

Required Features

Step 0: Set up

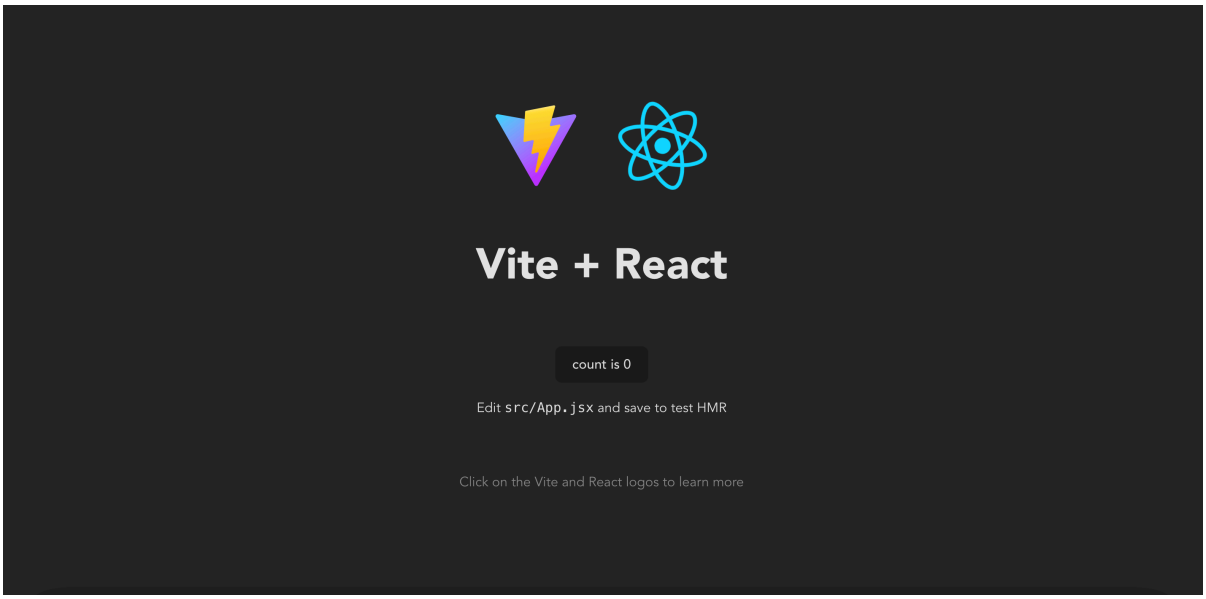
In this step, we'll set up an account and get an API key from CryptoCompare.

- [ ] Go to the CryptoCompare website, and make a free account with your email.
- [ ] After making a free account, go to **Account > API Keys** and make a new API Key.

- [ ] There will be a pop up asking what level of permissions your key needs. Check the **"Read All Price Streaming and Polling Endpoints"** option. Copy the API Key for use later.
- [ ] Create a new React project by running the command
  `npm create vite@latest crypto-hustle --template react`
- [ ] Follow the prompts and instructions the Terminal gives you. This project will be with the React framework in JavaScript.
- [ ] Then, in your Terminal, run:

```
cd crypto-hustle
npm install
npm run dev
```

You should see this screen:



💡 **Tip:** running `npm run dev` will start up a server that stays active and constantly updates your project as you type. If you want to quit this behavior, press Ctrl + C.

Now let's set up your files to start development.

- [ ] Open `App.jsx` in your newly created project.
- [ ] Remove everything in the `return()` statement so that when you check your local running version of the app, there is nothing there.
  - *Note*: You may receive an error if your `return()` statement is empty. You can add an empty `<div>` element to prevent this error.
- [ ] Delete the following line from the top of the file:

```
import reactLogo from './assets/react.svg'
```

'e will need to bring our CryptoCompare API Key into our project so that we can use it.

- [ ] Create a new file called `.env` in your `crypto-hustle` folder.

- [ ] Add an envirnoment variable for your API key and paste your API key in between the quotation marks.

```
VITE_APP_API_KEY="your_key_here"
```

- [ ] Then, back in `App.jsx`, import the environment variable at the top of the file under the `import` statements:

```
const API_KEY = import.meta.env.VITE_APP_API_KEY
```

Step 1: Query the CryptoCompare API to get a list of all the coins

In this step, we will make our first API call and use that to display a list of crypto coins in our react app.

In your `App.jsx` file, we will use the React hook `useEffect()` to call our API. This will guarantee that our API call is only used when the page renders, and allow the browser to display the screen even if all the information isn't quite ready yet making for a better user experience.

We will also use the `await` / `async` keywords to make our API calls so that our page will not try to load the data before we have retrieved the content from the API call.

In `App.jsx`, find the following line in the starter code:

```
const [count, setCount] = useState(0)
```

- [ ] In that line, change `count` and `setCount` to `list` and `setList`.
- [ ] Change the `0` in the `useState()` hook to `null`.
- [ ] Underneath that, add the `useEffect()` hook.

```
useEffect(() => {

}, [])
```

- [ ] Inside this hook, define an async function called `fetchAllCoinData()`.
- [ ] Inside the async function, use `fetch` to make a call to the API.
  - Check out the API documentation to see what link you will need to make a request to get the **entire list of coins**.

▶ 💡 **Need a hint on where to find it?**

- [ ] Save the JSON response returned to the `list` state variable.

✨ AI Opportunity

▶ *Use AI as a pair programming partner* → Getting unstuck

▶ **Want to double-check your code? Compare what you have to this** 🔽

- [ ] Lastly in our hook before the closing `}, [])` of our hook, we want to make sure to call our `fetchAllCoinData()` method and handle any errors that could pop up with it

```
fetchAllCoinData().catch(console.error)
```

After we have gone through the process of making our hook and calling our API, now we have to return and render it on our page.

Inside the `return()`:

- [ ] Create a `div` with the class name `"whole-page"`.
- [ ] Inside the `div`, add an `h1` element with the text `"My Crypto List"` and an unordered list element where we will display each coin.

```
<div className="whole-page">
  <h1>My Crypto List</h1>
      <ul>

      </ul>
</div>
```

When it comes time to render our actual list of coins, we first want to check that our API is not still waiting on results to fill up our `list`. Once we know `list` is not empty, we can display it.

Inside the `ul` tags:

- [ ] Check if `list` exists and use `Object.entries()` to get an array of key-value pairs from `list.Data`.
- [ ] Use `filter()` to filter coins by the `Algorithm` and `ProofType` properties
  - [ ] Remove coins with an `Algorithm` value of `"N/A"` or `ProofType` value of `"N/A"`
- [ ] Use `map()` to iterate over each key-value pair.
  - [ ] For each coin, create an `li` element and set the key and the text to the coin's `FullName`.
  - [ ] Otherwise, return `null`.

✨ AI Opportunity

▶ *Use AI to understand why your code isn't working* → Checking your code with AI

▶ **Want to double-check your code? Compare what you have to this** 🔽

- [ ] Next, let's check out these lines:

```
.filter(([_, coinData]) =>
  coinData.IsTrading &&
  coinData.Algorithm !== "N/A" &&
  coinData.ProofType !== "N/A"
  )
```

Here we are using the `filter()` function to filter down some of our coins so that we are only seeing coins that:

- are currently trading
- have an algorithm type that is not "N/A"
- have a proof type that is not "N/A"

The algorithm is the set of rules that determines how data is securely encrypted and linked in the blockchain, while the proof type is the method used to validate transactions. `isTrading` is `true` only for coins that are actively being traded. Filtering our page this way helps us filter the results to only real blockchain coins that are currently in use. This makes our page a little more efficient. Displaying all the coins would make the page too cluttered and unwieldy. If you would like, you can remove this `filter()` statement for now to see how many coins are available from this API.

> 💡 **Tip:** Feel free to add `console.log(list.Data)` above your `return()` statement so that you can open up the Page Inspector and see how the returned API data looks or to check if the data is being returned at all. Keep in mind that, in the `return()` statement, we only show the coins on the blockchain, and this query will show all types of coins, but you can still see for each coin what type of information we have. You can also check this in the API documentation when you execute a sample query.

📍 **Checkpoint 1**:

At the end of this checkpoint, you should now just have a page with a list of crypto coins on it. Make sure that your parenthesis `( )` and curly brackets `{ }` all line up and open and close around the information that you want it to.

Step 2: Create `CoinInfo` component so that coin prices and images can be queried and displayed

Now at this step, we don't just want to show our list of coins, but also the prices of them and their logos. We will create a component so that we can access this information dynamically for each coin that we parse.

- [ ] Inside the `src` folder, create a new folder called `Components`.
- [ ] In that folder, make a file called `CoinInfo.jsx`. This is where we will make a new component to keep track of what coin we are looking at, its price, and logo icon.
- [ ] At the top of `CoinInfo.jsx`, the `useEffect` and `useState` hooks and your API key.

```
import { useEffect, useState } from "react"
const API_KEY = import.meta.env.VITE_APP_API_KEY
```

- [ ] Create a `CoinInfo` component that takes `image`, `name`, and `symbol` as props.
- [ ] Create a state variable for `price` and set its value to `null`.

▶ **Want to double-check your code? Compare what you have to this ⤵️**

- [ ] Still inside the `CoinInfo` component, set up a `useEffect()` hook similarly to how we did in Step 1. This time, we'll pass in `symbol` inside the closing `[]`.

```
useEffect(() => {

}, [symbol])
```

This means that now, instead of `useEffect()` running on every render, it will now run whenever the symbol we pass in changes. So every time we give a new coin symbol to get the info for, the `useEffect()` hook will run.

✨ AI Opportunity

▶ *Use AI to explain code concepts* → Using `useEffect()` with `[]`

Similarly to Step 1, we need to get the price data for one coin at a time, and will define an async function to call the API for this data.

- [ ] Create an async `getCoinPrice()` function in the `useEffect()` hook.
  - Check out the API documentation again to see what link you will need to make a request to get **the price data for a single coin at a time**.
  - Since we are passing in a symbol to our API call, you will need to add the reference to the symbol, `${symbol}` somewhere in the link for the API call.

▶ 💡 **Need a hint on where to find the API query?**

✨ AI Opportunity

▶ *Use AI as a pair programming partner* → Asking for help with async functions

▶ **Want to double-check your code? Compare what you have to this ⤵️**

- [ ] Lastly, before the closing `}, [])` of our `useEffect()` hook, we want to make sure to call our `getCoinPrice()` method and handle any errors that could pop up with it.

```
getCoinPrice().catch(console.error)
```

- [ ] Next, we need to display our `CoinInfo` properly so that it can be sent back to wherever our component is being called. Underneath the `useEffect()` hook in `CoinInfo.jsx`, add:

```
return (
  <div>
    {price ? ( // rendering only if API call actually returned us data

    ) :
    null
    }
  </div>
)
```

Inside of this `return()` statement is where we will add our list items for each coin and all of their info. From all of the fields that the API call to `all coins` gives us, we can extract the image URL, full name of the coin, and symbol for the coin (which we used to get the price data). We want to display them all as a list item.

- [ ] In the `return()` statement, create an `li` element with the class name `"main-list"` and set the `key` to `{symbol}`.
- [ ] Inside the opening and closing `li` tags, create an `img` element with the class name `"icons"`. Set the `src` to `https://www.cryptocompare.com${image}` and the `alt` text to `Small icon for ${name} crypto coin`
- [ ] After the `img` element (but still inside the `li` tags), insert `{name}` followed by a `span` element with the class name `"tab"` followed by `${price.USD} USD`

▶ **Want to double-check your code? Compare what you have to this** 🔽

- [ ] Lastly, add the component to `App.jsx`. At the top of the file, include:

```
import CoinInfo from "./Components/CoinInfo"
```

- [ ] Where we previous had,

```
<li key={coinData.FullName}>{coinData.FullName}</li>
```

replace that with:

```
<CoinInfo
  image={list.Data[coin].ImageUrl}
  name={list.Data[coin].FullName}
  symbol={list.Data[coin].Symbol}
/>
```

Checkpoint 2:

Now you should see not only your list of coins, but also their price and image icon. Since we haven't added any style or anything, the images will be super huge, and will make the page look a little cluttered. Feel free to comment out the line adding the images if this bothers you until after Step 4.

Step 3: Create a search bar and use conditional rendering to search and filter for specific coins

In this step, we will add our search bar functionality so that we can search and filter our results based on a user input. We will just be working in `App.jsx`.

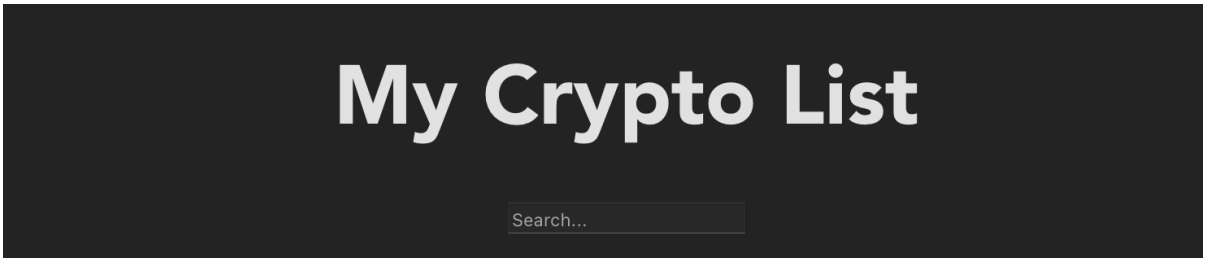
- [ ] Add an `input` element for the search bar in the `return()` statement underneath your page title.

```
<h1>My Crypto List</h1>
<input
  type="text"
  placeholder="Search..."
  onChange={(inputString) => searchItems(inputString.target.value)}
/>
```

✨ AI Opportunity

▶ *Use AI to understand provided code* → "Explain this" with Copilot

If you check your page now, you should see a small search bar underneath your title.



Now what we need to do is define the `onChange()` event handler that will search our list of coins when we type things in the search bar.

- [ ] First, create two state variables to keep track of our `searchValue` and `filteredData` so that we can display and edit them within the component. Place them underneath where the `list` state variable is made.

```
const [filteredResults, setFilteredResults] = useState([])
const [searchInput, setSearchInput] = useState("")
```

- [ ] Now, we need a function to take our `inputString` from the search bar and use it to filter through the results of our API call.

```
const searchItems = searchValue => {
  setSearchInput(searchValue)
  if (searchValue !== "") {
    const filteredData = Object.keys(list.Data).filter((item) =>
      Object.values(item)
        .join("")
        .toLowerCase()
        .includes(searchValue.toLowerCase())
    )
    setFilteredResults(filteredData)
  } else {
    setFilteredResults(Object.keys(list.Data))
  }
}
```

✨ AI Opportunity

▶ *Use AI to understand provided code* → "Explain this" with Copilot

Next, we need to use some conditional rendering so that if we have any search input, our `return()` statement will render our list of `CoinInfo` components from the `filteredData` list. If not, we pull from our original `list` from our API query results.

- [ ] To get you started, our condition for what we render is based on if we have any input in our search bar:

```
{searchInput.length > 0
  ? // what happens if we have search input? what list do we use to display coins?


  : // what happens if we don't have search input? what list do we use to display coins?
}
```

Try to fill in the rest for yourself! Both cases in our conditional rendering will look similar to how we have been rendering our coin list in Step 2. The only difference is what list of data we are pulling from.

> 💡 **Tip:** Keep in mind that in both cases we still need to do some additional conditional rendering for the algorithm and proof type so we don't have too many coin results.

✨ AI Opportunity

▶ *Use AI as a pair programming partner* → Getting unstuck

▶ **Want to double-check your code? Compare what you have to this** 🔽

📍 **Checkpoint 3**:

After this step, you should now have your list of coins with their full name, symbol, price, and icon image. Then you should have a working search bar where when you can type in some crypto symbols and it will start to filter the list and display those filter results on the page.

Step 4: Add CSS styling to make your page look unique!

In this step, we will make our page more visually appealing so that users will want to use an engage with it.

- [ ] In `App.css`, let's add some styling to your project. This is your chance to get creative! You should add some padding to space out each list item, make the .icon images smaller, and increase the font size of our text.

Other than that, feel free to change up the background color, text color, if there are separating lines tween each coin list item, or whatever else you want! (Looking up different CSS patterns and rules to ry can be super fun!)

✨ AI Opportunity

▶ *Use AI to brainstorm ideas for your code* → Style ideas

▶ 💡 If you want some ideas of what to do, the example project uses these rules...

Optional Additional Styling Task!

If you would like to add more spacing between the name of the crypto coin and its price, we can simulate a tab using some HTML and CSS. To make this change:

In `CoinInfo.jsx`, change this section in the `li` in the `return()` statement:

```
{name}
{price && price.USD ? ` $${price.USD} USD` : null}
```

to look like:

```
{name} <span className="tab"></span>
{price && price.USD ? ` $${price.USD} USD` : null}
```

Then, in `App.css`, create a new `.tab` class selector and add these rules:

```
.tab {
  display: inline-block
  margin-left: 75px
}
```

📍 **Checkpoint 4**:

After this step, your image icons should be much smaller and easier to see next to the coins they represent and there are some other stylistic changes.

And what the example app looks like after searching:

🎉 Congratulations, you've completed this lab! 🎉

If you have time left over, continue on to the stretch features to customize and improve your app!

Stretch Features

Step 5: Add a side nav bar to your page for viewing additional crypto information

In this step, we will add a cool sidebar with more crypto-related information that users can read through.

- [ ] Create a new file in the `Components` folder called `SideNav.jsx`.
- [ ] In `SideNav.jsx`, create a `div` element with the class name `"sidenav"`.
- [ ] Inside the `div` element, create an `h1` with the text `"here is my side nav bar"`.

▶ **Want to double-check your code? Compare what you have to this ⤵️**

Now to add it on the screen, we need to import the `SideNav` component into our page and render it on our page.

- [ ] In `App.jsx`, import the `SideNav` component.

```
import SideNav from "./Components/SideNav"
```

- [ ] Add the `SideNav` component to the `return()` statement, underneath `<div className="whole-page">`

```
<SideNav />
```

Finally, to make it look like a real side navigation bar, we need to add some styling.

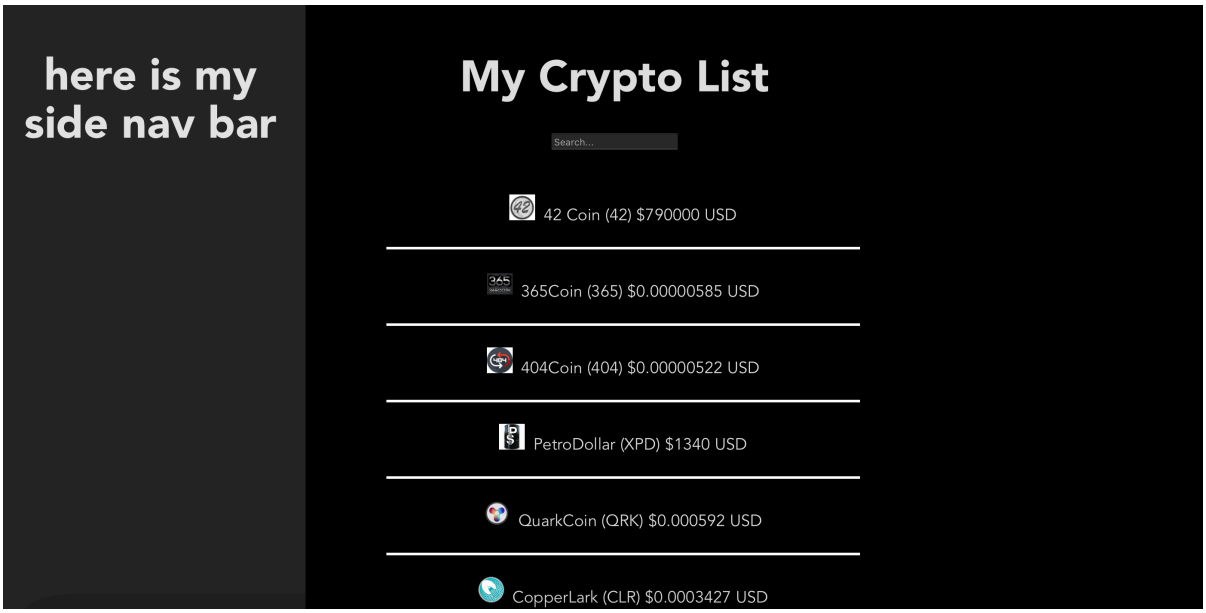
- [ ] In `App.css`, add a selector for the `.sideNav` class selector and add these rules:
  - [ ] Set `height` to `100%`
  - [ ] Set `width` to `25%`
  - [ ] Set `position` to `fixed`
  - [ ] Set `top` to `0`
  - [ ] Set `left` to `0`
  - [ ] Set `bottom` to `0`
  - [ ] Set `background-color` to `#232323`
  - [ ] Set `padding-top` to `15px`
  - [ ] Set `padding-bottom` to `15px`
  - [ ] Set `padding-left` to `10px`
  - [ ] Set `padding-right` to `10px`
  - [ ] Set `overflow-y` to `scroll`
  - [ ] Set `display` to `flex`
  - [ ] Set `flex-direction` to `column`
  - [ ] Set `align-items` to `center`

✨ AI Opportunity

▶ *Use AI to check your code for errors*

📍 **Checkpoint 5**:

After this step, you should have a cool side navigation bar on your page that we can use to add more information to for users and it just has some default text in it.



Step 6: Create a new component to query for recent crypto news, and add that to your page in a side navigation bar

Since we have not yet covered how to go between pages in React apps, we cannot turn our `SideNav` into a traditional one that links to multiple pages. However, we can add information to our `SideNav` so that it can serve a different purpose on our page. In this case, we will query our API for a list of recent news articles about cryptocurrency and display links to those in our `SideNav`.

- [ ] Create a new file in the `Components` folder called `CryptoNews.jsx`.
- [ ] In this file, we'll make a new `CryptoNews` component, which will call an API and then display the results.

```
import { useEffect, useState } from "react"
const API_KEY = import.meta.env.VITE_APP_API_KEY

function CryptoNews() {

}

export default CryptoNews
```

- [ ] Create a state variable for a `newsList` to keep track of recent scams that we can query from our API. Use the `useEffect()` hook and async/await notation to do so.
    - Check out the API documentation to see what link you will need to make a request to get the **latest news articles**.

- [ ] Once you have your `newsList` results from the API, display them using `.map()`, making sure that the `newsList` is populated first.
    - [ ] For each news article, create an `<li>` element and set the key and the text to the article's `title`.
    - [ ] Within each `<li>` element, create an `<a>` hyperlink element and set the `href` attribute to the article's `url` and the hyperlink's text content to the article's `title`.

✨ AI Opportunity

▶ *Use AI as a pair programming partner* → Getting unstuck

▶ **Want to double-check your code? Compare what you have to this** 🔽

Now, let's add this `CryptoNews` component to our `SideNav` component to display it on the page.

- [ ] In `SideNav.jsx`, import the `CryptoNews` component:

```
import CryptoNews from "./CryptoNews"
```

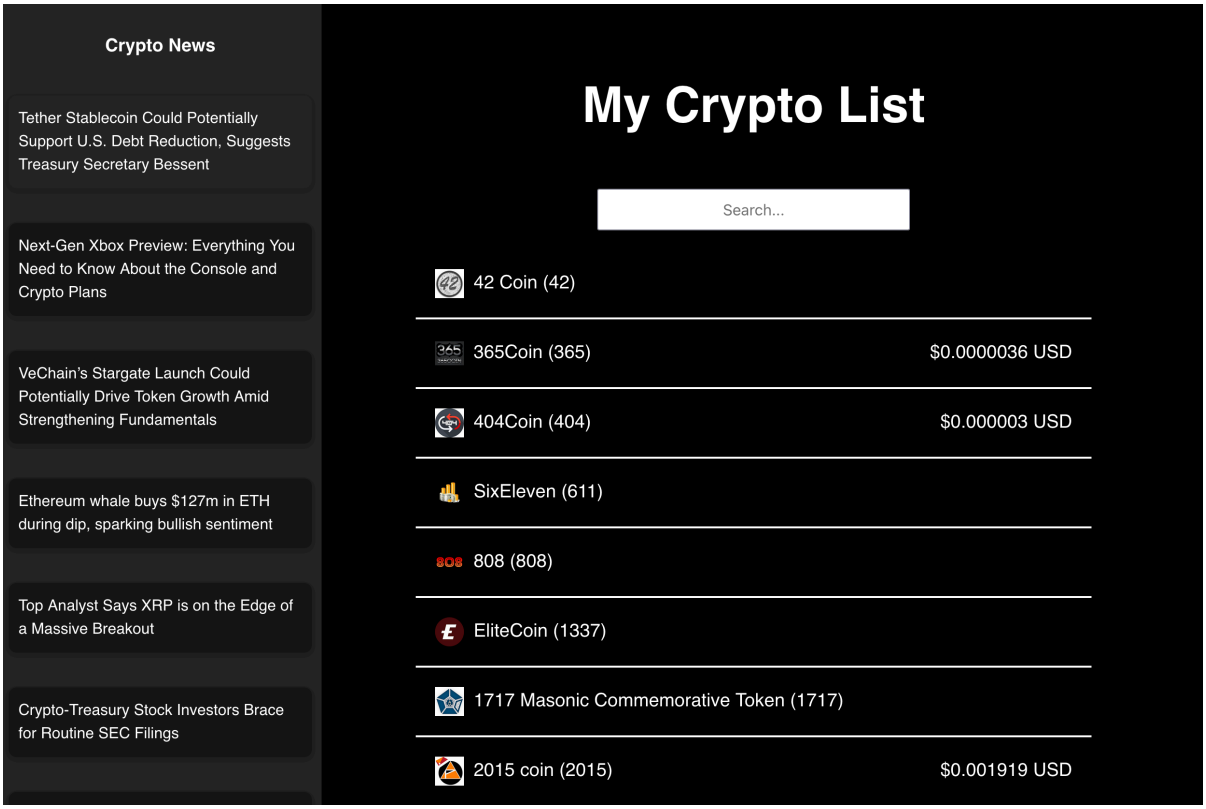
- [ ] In between the `div` tags for the `SideNav`, add the `CryptoNews` component.

```
<CryptoScam />
```

💡 **Tip:** Feel free to go back and play around with some of CSS rules for the navbar and/or add new ones to your `CryptoNews` component's elements if you would like!

📍 **Checkpoint 6**:

You are now done! You should now see your list of blockchain crypto coins, with icons and prices, that can be searchable and filter your list as you type. There should also be sideNav board with information about the most recent cryptocurrency news.



🎉 Congratulations 🎉

You've completed this lab AND stretch goals! 🚀

💡 **Tip:** Remember to come back and reference this lab when you need to do similar things in your project!