WEB102 | Intermediate Web Development

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

Week 5: Lab 4 - Cap!

Overview

Need a high quality screenshot to share an interesting site? This app uses the ApiFlash API to take screenshots of a given website with a variety of parameters. First, you will sign up for an ApiFlash account and explore the query builder that allows you to test out queries. Then, you will try making a single static API call from your site. Finally, you will build an interface that allows you to get your own screenshots and insert them into the page.

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

🎯 Goals

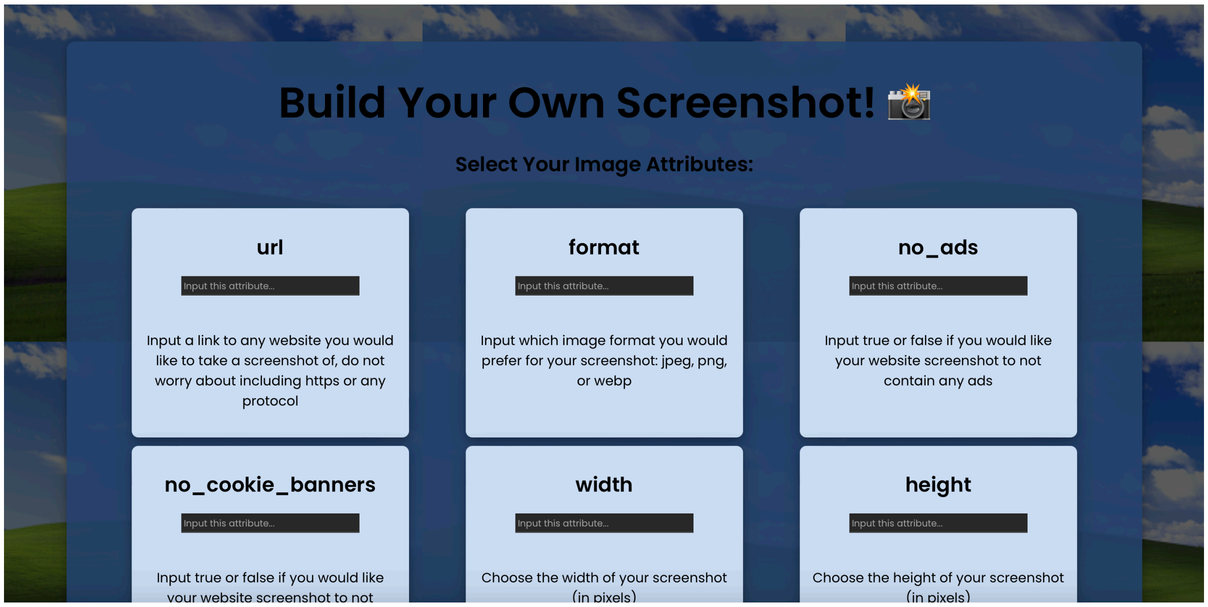
By the end of this lab you will be able to...

- [ ] Make a static API call using `async` / `await` and save the results to a state variable
- [ ] Be able to add and edit query parameters for API calls

Features

Required Features

- [ ] Allow the user to add at least three parameters to a query to the ApiFlash API
- [ ] Display the screenshot returned by the ApiFlash call on the page
- [ ] On a separate part of the page, display all the images the user has queried thus far



Build Your Own Screenshot! 📸

Select Your Image Attributes:

url

Input a link to any website you would like to take a screenshot of, do not worry about including https or any protocol

format

Input which image format you would prefer for your screenshot: jpeg, png, or webp

no_ads

Input true or false if you would like your website screenshot to not contain any ads

no_cookie_banners

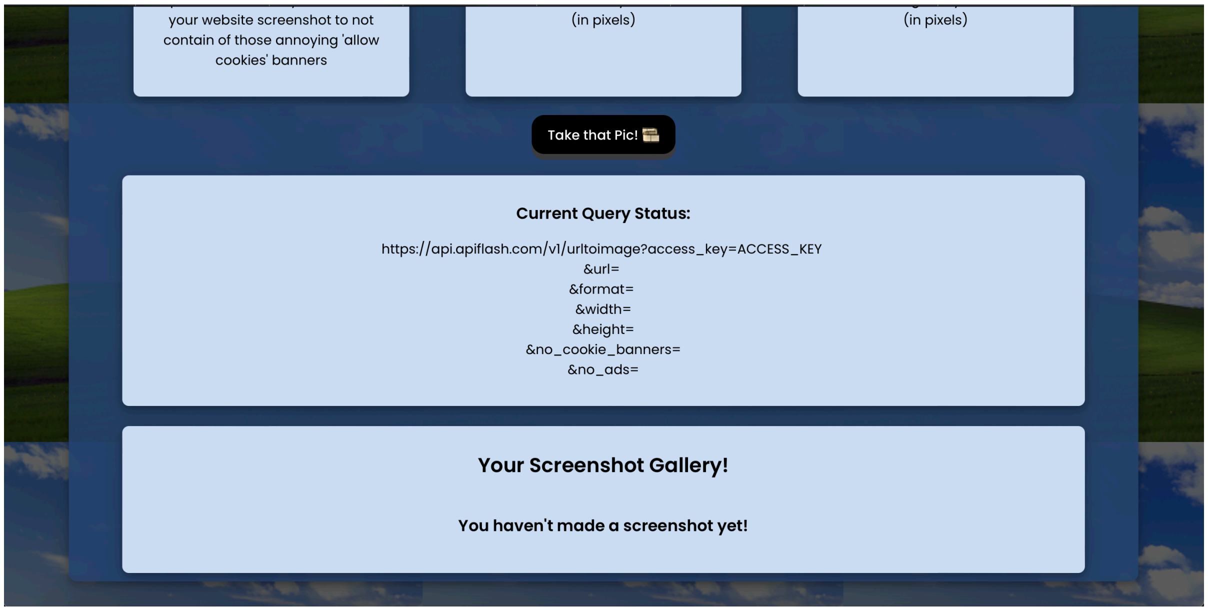
Input true or false if you would like your website screenshot to not

width

Choose the width of your screenshot (in pixels)

height

Choose the height of your screenshot (in pixels)



your website screenshot to not contain of those annoying 'allow cookies' banners

(in pixels)

(in pixels)

Current Query Status:

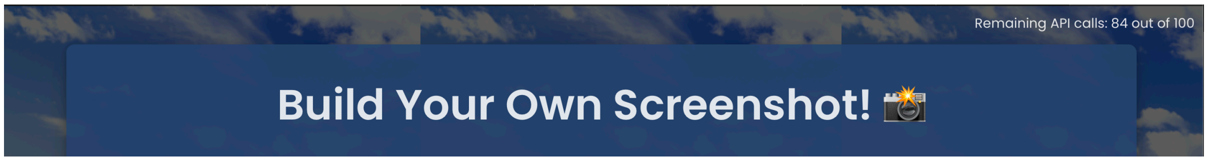
```
https://api.apiflash.com/v1/urltoimage?access_key=ACCESS_KEY
&url=
&format=
&width=
&height=
&no_cookie_banners=
&no_ads=
```

Your Screenshot Gallery!

You haven't made a screenshot yet!

Stretch Features

- [ ] Display whether or not your web app has hit the maximum number of queries allowed using another API call



Remaining API calls: 84 out of 100

Build Your Own Screenshot! 📸

Resources

- APIFlash Documentation
- Fetch API
- Async/Await

Lab Instructions

Required Features

Step 0: Set up

In this step, we'll create our new Vite project and get used to the APIFlash API we need for this lab!

- [ ] Go to the APIFlash website and make a free account with your email. After making a free account, the website will make a first free access key for you. Go to the **Access Keys** tab in order to see it.
- [ ] Go to the APIFlash query builder and play around making a few queries so that you understand how exactly this API works. Read through each parameter definition and take notes of which parameters you would like to include for users to control in later steps!

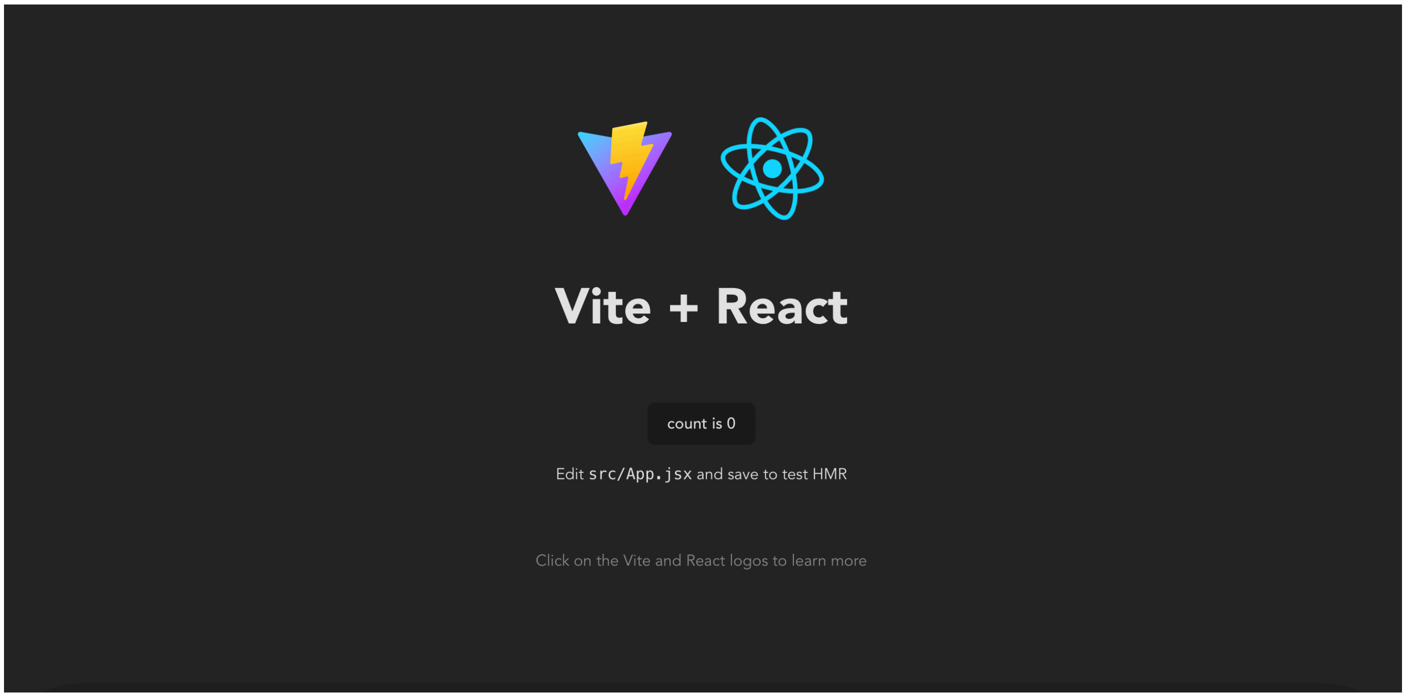
> 🥶 *Potential Blocker*: Each ApiFlash account is limited to 100 queries per month. If you run out of queries, you can create another account using the unlimited Gmail account trick of adding + and then incrementing numbers after the, e.g. xyz@gmail.com becomes xyz+1@gmail.com, xyz+2@gmail.com etc.

- [ ] Create a new React project. To do this, run the command `npm create vite@latest cap`.
- [ ] Follow the prompts and instructions the Terminal gives you. This project will be with the React framework in Javascript.
- [ ] In your Terminal, run the following commands:

```
cd cap
npm install
npm run dev
```

> 💡 **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`.

You should see this screen:


Vite + React
count is 0
Edit src/App.jsx and save to test HMR
Click on the Vite and React logos to learn more

- [ ] Open up `App.jsx` in your newly created project and remove everything inside the `return` statement so that when you check your local running version of the app, there is nothing there.
- [ ] Delete the line below from the top of the file:

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

We will also need to bring our APIFlash access key into our project so that the app can use our APIFlash account.

- [ ] Create a new file called `.env` in your `cap` folder.
- [ ] Create a environment variable called `VITE_APP_ACCESS_KEY` that is set to your access key between the quotation marks.

```
VITE_APP_ACCESS_KEY = "your_key_here"
```

- [ ] Then, back in `App.jsx`, create a variable called `ACCESS_KEY` at the top of the file under the `import` statements that is set to the environment variable `VITE_APP_ACCESS_KEY`.

```
const ACCESS_KEY = import.meta.env.VITE_APP_ACCESS_KEY;
```

✨ AI Opportunity

▼ *Use AI to explain code concepts* → Environment Variables

Are you wondering what environment variables are or why we are accessing the `VITE_APP_ACCESS_KEY` with `import.meta.env` in front of it? Ask Copilot what environment variables are or for examples of types of environment variables a developer might use in a web app. Try also highlighting and right-clicking the line of code with the `ACCESS_KEY` variable, and choose **Copilot > Explain this** for an explanation.

Step 1: Create your form for user input on the screenshot attributes

In this step, we want to make the form that users will use in order to establish some base qualities about their screenshots.

We first want to establish that there is a set of inputs that we will be asking each user to specify for their screenshot.

- [ ] Add these in a state variable dictionary in `App.jsx`. Feel free to add more inputs or change out some of the ones here, however you **need** to have the `url`, `width`, and `height` input values.

```
const [inputs, setInputs] = useState({
  url: "",
  format: "",
  no_ads: "",
  no_cookie_banners: "",
  width: "",
  height: "",
});
```

Next, let's make a new component for handling the form.

- [ ] Create a `components` folder.
- [ ] Inside your `components` folder, create a file called `APIForm.jsx`.
- [ ] In the `return` statement in `APIForm.jsx`, create a `div` element. Inside the `div`,
  - [ ] Add an `h2` element with `"Select Your Image Attributes:"` as the text.
  - [ ] Add opening and closing `form` tags with the class name `"form-container"`.

✨ AI Opportunity

▼ *Use AI as a pair programming partner* → Prompting tips

If you want to use Copilot to generate this code for you, make sure to clearly specify the requirements it needs to fulfill. In this case, you might try telling it that you need a `div` that contains an `h2` with the text `"Select Your Image Attributes:"` and opening and closing `form` tags with the class name `"form-container"`. Compare the code it generates with the requirements to check if it makes sense and make any changes needed as you add it to your code.

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

```
const APIForm = () => {

  return (
    <div>
      <h2> Select Your Image Attributes: </h2>
      <form className="form-container">

      </form>
    </div>
  );
};

export default APIForm;
```

We will be using regular text boxes, so all we'll need to do is loop through our `inputs` and display those titles with an `input` tag below them.

Before we do that, we'll need to pass in some props to `APIForm` since we are working in a form so that we can edit our `inputs` state variable properly.

- [ ] Add `inputs`, `handleChange`, and `onSubmit` as props to `APIForm`.

```
const APIForm = ({inputs, handleChange, onSubmit}) => {
```

☐ Now we are able to make our form with our text boxes. Before using `.map()`, we first need to turn our dictionary into an array of key and values to loop through using `Object.entries()`

```
<form className="form-container">
  {inputs &&
    Object.entries(inputs).map(([category, value], index) => (
      <li className="form" key={index}>
        <h2>{category} </h2>
        <input
          type="text"
          name={category}
          value={value}
          placeholder="Input this attribute..."
          onChange={handleChange}
          className="textbox"
        />
        <br></br>
        <br></br>
        <p> {inputsInfo[index]}</p>
      </li>
    ))}
</form>
```

✨ AI Opportunity

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

What's going on here? Highlight and right-click the code, then choose **Copilot > Explain this** to get an explanation so you understand what each line is doing. If you have questions about its explanation, ask! It'll happily answer any questions you have. You could also ask for additional examples using any of the concepts in this code so you can see these used in different ways.

Next, we need to add our button for submitting!

☐ Create a button with the appropriate type, class name of `"button"`, and that calls the `onSubmit()` function when the button is clicked. Set the text of the button to `"Take that Pic! 🎞"`

✨ AI Opportunity

▼ *Use AI to explain code concepts* → Button reminders

Forgot how to create a submit button? Ask Copilot for help! Remember, clearly specify your requirements and where you're stuck so you can get the code you need.

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

```
</form>
<button type="submit" className="button" onClick={onSubmit}>
  Take that Pic! 📎
</button>
```

Also, for a great user experience, we should make it clear to users what each input means for our API calls.

- [ ] Create an array to hold small descriptions of each parameter in the same order they show up in your `inputs` variable:

```
const inputsInfo = [
  "Input a link to any website you would like to take a screenshot of. Do not include http
  "Input which image format you would prefer for your screenshot: jpeg, png, or webp",
  "Input true or false if you would like your website screenshot to not contain any ads",
  "Input true or false if you would like your website screenshot to not contain of those a
  "Choose the width of your screenshot (in pixels)",
  "Choose the height of your screenshot (in pixels)",
];
```

Note: The reason we cannot make `inputsInfo` a dictionary with keys representing each of the inputs is that in our `.map()` function, we can only access one dictionary object at a time. In this case, it is our `inputs` state variable. Therefore, we have to make this an array and utilize the `index` looping variable in our `.map()` (which is why it is important that your `inputsInfo` has the descriptions in the same order the inputs show up in).

- [ ] And next, let's add our `inputsInfo` for each input right below its text box.

```
...
  onChange={handleChange}
  className="textbox"
  />

  <br></br>
  <p> {inputsInfo[index]}</p>

</li>
```

- [ ] And lastly, let's use our `APIForm` component in `App.jsx` so that we can see it! In `App.jsx` let's import this file at the top:

```
import APIForm from './Components/APIform';
```

- [ ] Inside of our `return` statement, let's add a page header and our form component with the proper props.

```
return (
    <div className="whole-page">
      <h1>Build Your Own Screenshot! 📷</h1>

      <APIForm
        inputs={inputs}
        handleChange={(e) =>
          setInputs((prevState) => ({
            ...prevState,
            [e.target.name]: e.target.value.trim(),
          }))
        }
        onSubmit={submitForm}
      />
      <br></br>

    </div>
  );
}

export default App
```

✨ AI Opportunity

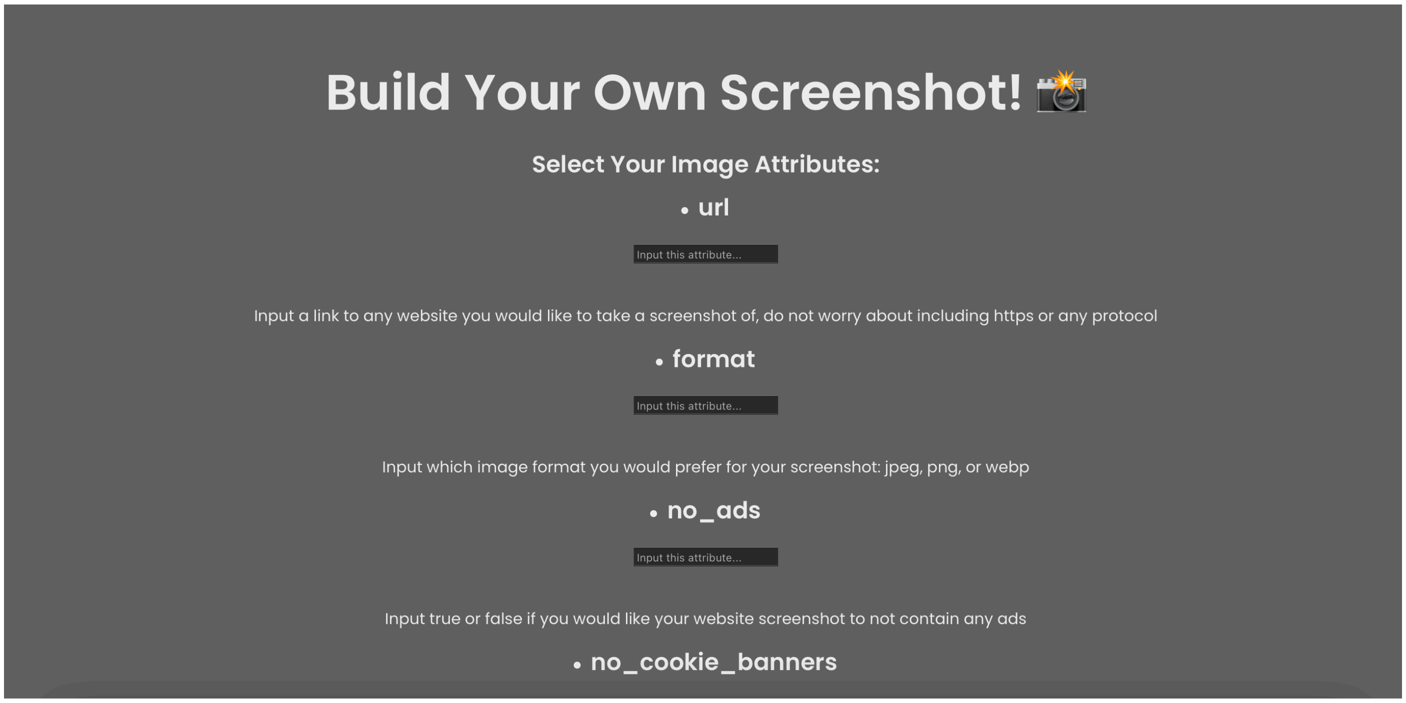
▼ *Use AI to understand provided code* → "Explain this" with Copilot

Need some help understanding what's going on in this code? Highlight and right-click the code, then select **Copilot > Explain this** to get an explanation of what it's doing. If you have questions about specific lines, highlight and right-click only the lines you want to know more about.

- [ ] Add an empty `submitForm()` function above the `return` statement in `App.jsx` that we will fill in in the next step.

```
const submitForm = () => {


}
```

📍 **Checkpoint 1**: At the end of this step, you should be able to see your form on the home page with a text box for each input you would like users to adjust and a little info about each input below it's respective text box.


Build Your Own Screenshot! 📸

Select Your Image Attributes:

- url

Input this attribute...

Input a link to any website you would like to take a screenshot of, do not worry about including https or any protocol

- format

Input this attribute...

Input which image format you would prefer for your screenshot: jpeg, png, or webp

- no_ads

Input this attribute...

Input true or false if you would like your website screenshot to not contain any ads

- no_cookie_banners

Step 2: Make the query with the user input, create an API call, and display your screenshot

In this step, we want to take our user inputs, do some input/error handling with them, and then make our API call to display our created image!

To start, let's make our `submitForm()` function that we created in `App.jsx`. To do this, we want to include some error handling that will handle cases in which users do not input any value in the form.

- [ ] To handle this error, let's include some default values that are automatically assigned when our input values for those parameters are checked and empty.

```
const submitForm = () => {
  let defaultValues = {
    format: "jpeg",
    no_ads: "true",
    no_cookie_banners: "true",
    width: "1920",
    height: "1080",
  };

}
```

- [ ] We also want to make sure that there is a `url` provided by the user. If there is no url, the API query should not be made and we should let the user know using `alert()`.

✨ AI Opportunity

▼ *Use AI as a pair programming partner* → Error handling

Need some help implementing this step? Try telling Copilot what you're trying to do to get some help. For example:

You:

> I need to make sure that the user provides a url. If there is no url, I need to use alert() to let the user know that they forgot to submit a url. Can you help me write this?

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

```
if (inputs.url == "" || inputs.url == " ") {
  alert("You forgot to submit an url!");
}
```

- [ ] If there is a proper `url`, we will check the rest of our inputs to see if we need to include any default values. Then, we are ready to make our API query string.

```
else {
  for (const [key, value] of Object.entries(inputs)) {
    if (value == "") {
      inputs[key] = defaultValues[key]
    }
  }
}
```

- [ ] Next, create a helper function called `makeQuery()`. We'll use this function to take our input values and assemble them into the right query string format that our API call needs.

- [ ] Inside our `makeQuery()` function, we also want to include some important parameters that don't need to be chosen by users but still need to be included.

```
const makeQuery = () => {
  let wait_until = "network_idle";
  let response_type = "json";
  let fail_on_status = "400%2C404%2C500-511";
  let url_starter = "https://";
  let fullURL = url_starter + inputs.url;
```

✨ AI Opportunity

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

Want to know what these parameters mean? Highlight and right-click on the code, and choose **Copilot > Explain this** to get an explanation.

If you're still not sure or want more information, check out the documentation.

- [ ] Now we can assemble our query:

```
let query = `https://api.apiflash.com/v1/urltoimage?access_key=${ACCESS_KEY}&url=${fullURL
```

Note: Pay attention to what quotes we use to surround the query string so that we can input our variables, it is not the traditional single quote but instead the one on the same key with the tilde on the keyboard.

- [ ] After this we want to go back and make sure to call our `makeQuery()` function after we have properly checked our `inputs` variable in our `submitForm()` function.

```
else {
  for (const [key, value] of Object.entries(inputs)) {
    if (value == "") {
      inputs[key] = defaultValues[key]
    }
  }
  makeQuery();
}
```

The next function we will need is our async function to make our API call with our newly created query, `allAPI()` . We will also need a state variable to hold and display the screenshot after we make our API call.

- [ ] Create a state variable to hold and display the current screenshot.

✨ AI Opportunity

▼ *Use AI to explain code concepts* → Choosing a strategy

What should I call this state variable? What should I set the value to? These and other questions are great questions that AI tools like Copilot can help with. Tell it what you need to do and ask how you should write this code.

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

```
const [currentImage, setCurrentImage] = useState(null);
```

- [ ] In our `callAPI()` function, we will make a `fetch` call with `await`, and save the response as a simple json since we added that nice `response_type=json` parameter to our query string. (Feel free to include `console.log(json);` if you would like to see what our API gives us back.)

```
const callAPI = async (query) => {
  const response = await fetch(query);
  const json = await response.json();
}
```

It's possible to get error messages instead of a screenshot when using `callAPI()`. We want to keep track of these error messages. Unfortunately, the response errors from this API are not super informational.

- [ ] To give more information to our users, check if we didn't receive a URL back from our API call (meaning a proper screenshot couldn't be taken), and display a message to the user with `alert()`. If we do get a URL back, make it our `currentImage`.

✨ AI Opportunity

▼ *Use AI to explain code concepts* → Requesting examples

If you get stuck writing this code, tell Copilot what you're trying to do to get help. For example:

You:

> I need to check if a url was received from the API call. If we didn't receive a url, I need to display a message to the suer with alert(). If we do get a url back, I need to set it to the currentImage. Can you give me an example of how to do this?

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

```
if (json.url == null) {
  alert("Oops! Something went wrong with that query, let's try again!")
    }
else {
  setCurrentImage(json.url);
}
```

- [ ] Let's put this call to use `callAPI()` at the end of our `makeQuery()` function:

```
callAPI(query).catch(console.error);
```

✨ AI Opportunity

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

What does this do? Highlight and right-click the code, and choose **Copilot > Explain this** to get an explanation.

- [ ] Finally, create another helper function called `reset()`.
- [ ] In this function, set the current `inputs` values to `""` so that our form is cleared after our API call.

✨ AI Opportunity

▼ *Use AI as a pair programming partner* → Writing a reset function

If you get stuck writing this code, tell Copilot what you're trying to do to get help. For example:

You:

> I am writing a reset() function to set the current inputs values to "" so that the form is cleared after the API call. Can you help me write this function?

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

```
const reset = () => {
  setInputs({
    url: "",
    format: "",
    no_ads: "",
    no_cookie_banners: "",
    width: "",
    height: "",
  });
}
```

- [ ] Call this function in the `else` block of our `callAPI()` function.

```
else {
  setCurrentImage(json.url);
  reset();
}
```

Lastly, we want to allow users to see a few things from the web page to show them their resulting screenshot and how their query is progressing based on their inputs in case they are interested.

- [ ] Below the form, include this conditional rendering to show the `currentImage` if it exists, and nothing otherwise:

```
{currentImage ? (
  <img
    className="screenshot"
    src={currentImage}
    alt="Screenshot returned"
  />
) : (
  <div> </div>
)}
```

✨ AI Opportunity

▼ *Use AI to explain code concepts* → Conditional rendering

Ask Copilot for examples of React apps using conditional rendering. You could even copy and paste the examples into your IDE so you can see the results.

- [ ] Below that, let's add a container with the mock query string that users are assembling with their `inputs` in the form.

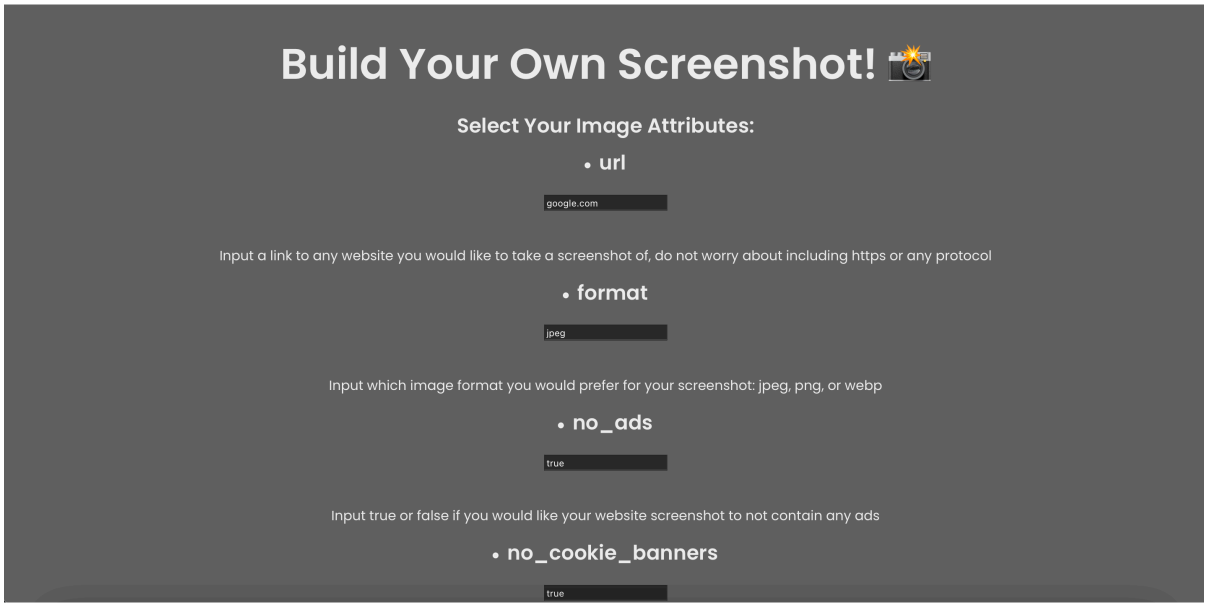
```
<div className="container">
  <h3> Current Query Status: </h3>
  <p>
    https://api.apiflash.com/v1/urltoimage?access_key=ACCESS_KEY
    <br></br>
    &url={inputs.url} <br></br>
    &format={inputs.format} <br></br>
    &width={inputs.width}
    <br></br>
    &height={inputs.height}
    <br></br>
    &no_cookie_banners={inputs.no_cookie_banners}
    <br></br>
    &no_ads={inputs.no_ads}
    <br></br>
  </p>
</div>

<br></br>
```

✨ AI Opportunity

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

What does this do?? Highlight and right-click the code to find out!

📍 **Checkpoint 2**: After this step, you should be able to fill in the form with inputs, see your sample query string change in real time, and when you submit your query, see the resulting image below!



Build Your Own Screenshot! 📸

Select Your Image Attributes:

- url

google.com

Input a link to any website you would like to take a screenshot of, do not worry about including https or any protocol

- format

jpeg

Input which image format you would prefer for your screenshot: jpeg, png, or webp

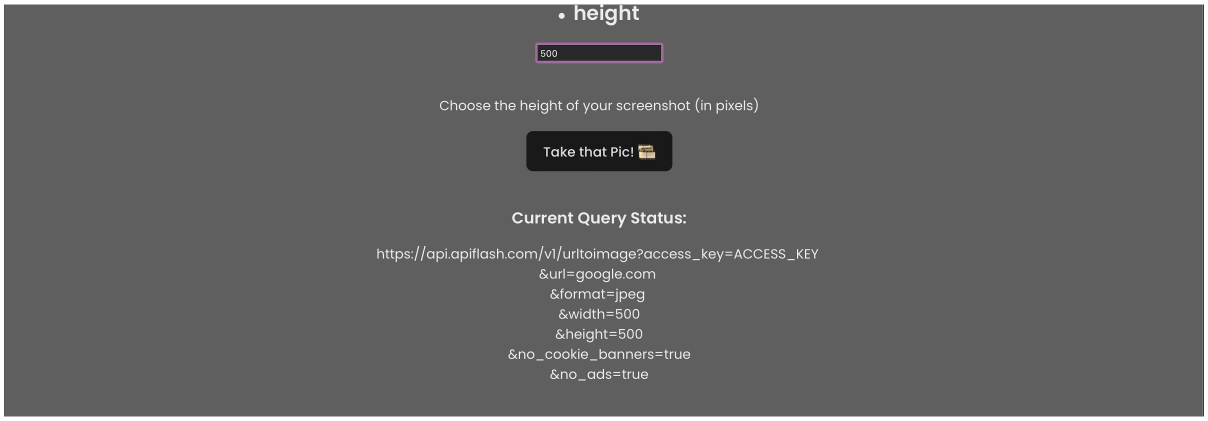
- no_ads

true

Input true or false if you would like your website screenshot to not contain any ads

- no_cookie_banners

true



- height

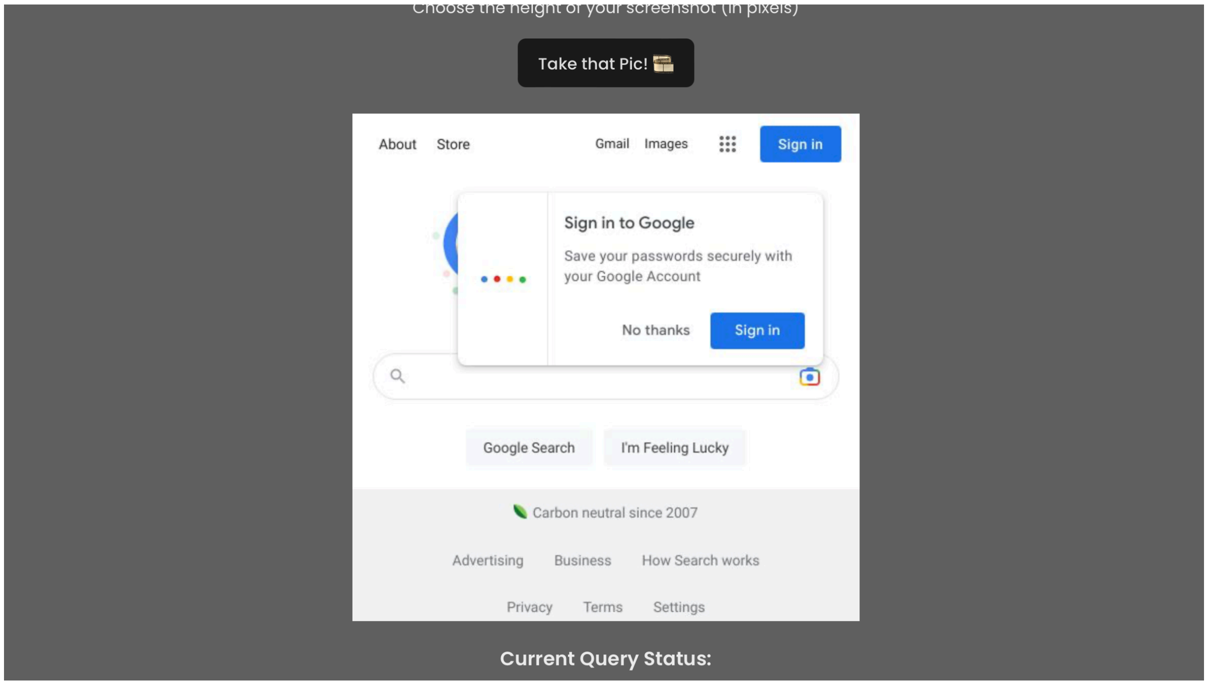
500

Choose the height of your screenshot (in pixels)

Take that Pic! 🎞️

Current Query Status:

```
https://api.apiflash.com/v1/urltoimage?access_key=ACCESS_KEY
&url=google.com
&format=jpeg
&width=500
&height=500
&no_cookie_banners=true
&no_ads=true
```



Choose the height of your screenshot (in pixels)

Take that Pic! 🎞️

Current Query Status:

Step 3: Create gallery for displaying previous screenshots

At this step, we want to create a gallery feature for our previous screenshots so that we can keep track of ' of the different sites that we have seen. We will save our images in a state array and display them at .e bottom of our page.

- [ ] We know that we will need to keep track of all of the images we have taken. Create a state variable for this in `App.jsx`.

✨ AI Opportunity

▼ *Use AI as a pair programming partner* → State variable

If you get stuck writing this code, tell Copilot what you're trying to do to get help. For example:

You:

> I need a state variable to keep track of all the images that have been taken. Help me create a state variable for this.

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

```
const [prevImages, setPrevImages] = useState([]);
```

- [ ] We need to update this state variable whenever we get a new image, so let's update this variable in our `callAPI()` function.

```
else {
  setCurrentImage(json.url);
  setPrevImages((images) => [...images, json.url]);
  reset();
}
```

- [ ] Create a new file called `Gallery.jsx` in your `components` folder for a new `Gallery` component. Now that we know that we have an array of images, we can add that in as a prop we expect.

```
const Gallery = ({ images }) => {

  return (
    <div>

    </div>
  );
};

export default Gallery;
```

- [ ] We can use some conditional rendering for our `images` array inside of our `return` statement so that we only show our photo gallery when we actually have some images there and display a message if there are not any photos to show.

✨ AI Opportunity

▼ *Use AI to explain code concepts* → Conditional rendering

We used conditional rendering earlier, which we can use as an example to ask Copilot how we should implement this step. For example:

You:

> Here is my code where I used conditional rendering:
>
> (your code here)
>
> I need to use the same structure to conditionally render the photo gallery when there are images or display a message if there are not any photos to show. Can you show me how I would do this?

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

```
<h2> Your Screenshot Gallery!</h2>
<div className="image-container">
  {images && images.length > 0 ? (


    )
  ) : (
    <div>
      <h3>You haven't made a screenshot yet!</h3>
    </div>
  )}
</div>
```

- [ ] When we do have images, we can loop through them and display them in a list.

✨ AI Opportunity

▼ *Use AI to explain code concepts* → Using map()

We've used `map()` before, which we can use as an example to ask Copilot how we should implement this step. For example:

You:

> Here is my code where I used map() to loop through a list:
>
> (your code here)
>
> I need to use the same structure to loop through the images and display them in a list. Can you show me how to do this?

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

```
images.map((pic, index) => (
  <li className="gallery" key={index}>
    <img
      className="gallery-screenshot"
      src={pic}
      alt="Undefined screenshot from query"
      width="500"
    />
  </li>
)
```

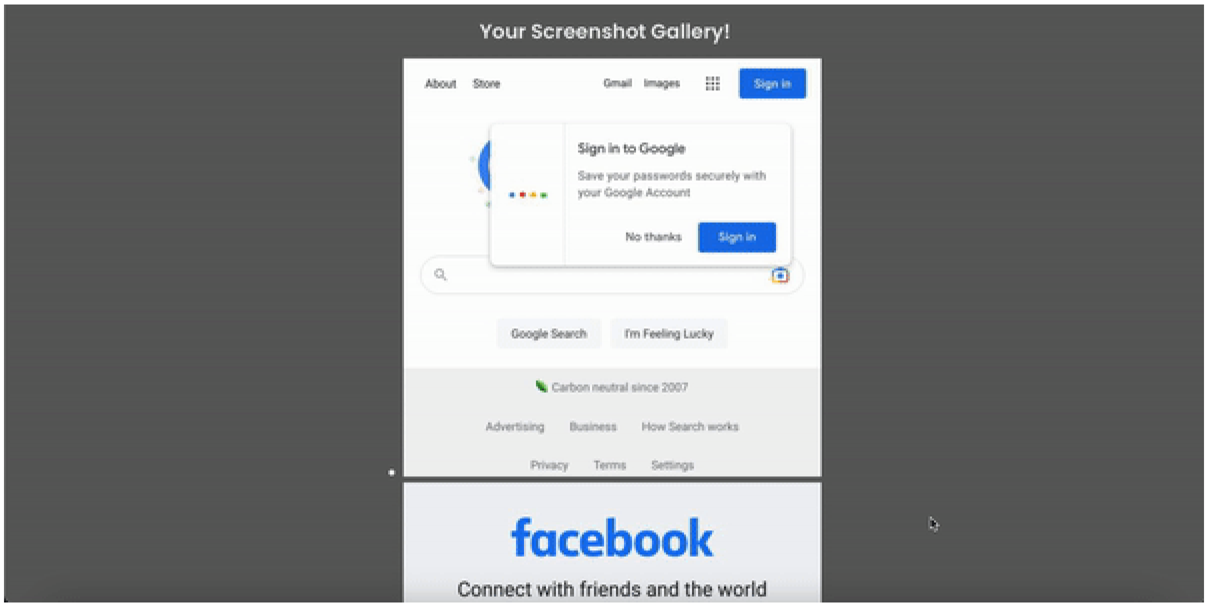
- [ ] Lastly, we need to import the `Gallery` component in our `App.jsx` file.

```
import Gallery from './Components/gallery';
```

- [ ] Add the component to the bottom of our `return` statement below your `Current Query Status`:

```
<div className="container">
  <Gallery images={prevImages} />
</div>
```

📍 **Checkpoint 3**: At the end of this step, you should now have a cool photo gallery so that any time you successfully query for a new screenshot and it is displayed, it also gets added to the photo gallery.



Step 4: Add styling with CSS

This step is where you can let your creativity go free! Make your Cap project your own in this step.

The only requirements for this step is that you use Flexbox for the form elements and for the gallery feature. Other than that, feel free to add new backgrounds, drop shadows, pseudoselectors, and all other creative elements!

✨ AI Opportunity

▼ *Use AI to brainstorm ideas for your code* → Styling ideas

Keep your files open containing your components and `App` code, then ask Copilot for ideas to style your app. Copilot will attempt to use the open files for context when making suggestions.

If you have ideas of what you want to do but aren't sure how to implement it, tell Copilot what you want to do and ask it to suggest code to achieve your goals.

▼ 💡 If you want some ideas of what to do, the example project that you have been seeing screenshots of throughout these instructions incorporates these rules...

In `App.css`:

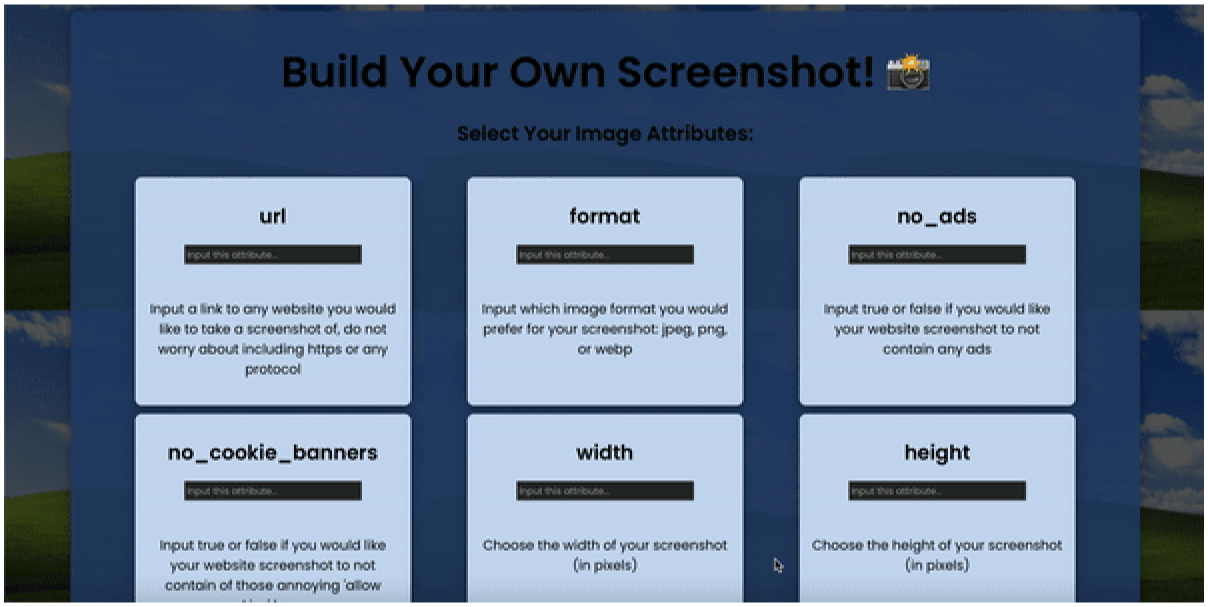
```css
@import url('https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;500;600&displ

  #root {
  max-width: 1280px;
  margin: 0 auto;
  padding: 2rem;
  text-align: center;
  font-family: 'Poppins', sans-serif;

}

li {
  list-style: none;
}

li.form {
  padding: 15px;
  box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2), 0 6px 20px 0 rgba(0, 0, 0, 0.19);
  margin: 5px;
  border-radius: 8px;
  background-color: #cadcf2;
  width: 300px;
}

input {
  font-family: 'Poppins';
}

.form-container {
  display: flex;
  justify-content: space-evenly;
  align-items: stretch;
  position: relative;
  padding: 15px;
  flex-wrap: wrap;
}

.image-container {
  display: flex;
  justify-content: space-evenly;
  padding: 10px;
  flex-wrap: wrap;
}

.whole-page {
  background-color: rgba(37, 67, 110, 0.9);
  border-radius: 10px;
  padding-top: 10px;
  padding-bottom: 10px;
  box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2), 0 6px 20px 0 rgba(0, 0, 0, 0.19);
  margin-top: 15px;
}
```

```
.button {
  color: #fff;
  background-color:black;
  border: none;
  border-radius: 15px;
  box-shadow: 0 7px rgb(62, 62, 62);
}

.button:hover {
  background-color: #1a1a1a;
}

.button:active {
  background-color: #0b0b0b;
  box-shadow: 0 3px #666;
  transform: translateY(4px);
}

.container {
  padding: 15px;
  box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2), 0 6px 20px 0 rgba(0, 0, 0, 0.19);
  margin-left: 5%;
  margin-right: 5%;
  border-radius: 8px;
  background-color: #cadcf2;
}

.screenshot {
  max-width: 90%;
  margin-bottom: 20px;
  border-radius: 8px;
}

.gallery-screenshot {
  border-radius: 8px;
`
```

In `index.css`, for the background:

```
:root {
.
.
.

  background-image:url('assets/windows_background.png');
  background-size: 35%;
  background-blend-mode: multiply;
}
```

📍 **Checkpoint 4**: At the end of this step, you should have all the required functionality of the lab and it has a cool interface with your personal touch! Flexbox should be utilized in the form elements and in the photo gallery.


Build Your Own Screenshot! 📸

Select Your Image Attributes:

url

input this attribute...

Input a link to any website you would like to take a screenshot of, do not worry about including https or any protocol

format

input this attribute...

Input which image format you would prefer for your screenshot: jpeg, png, or webp

no_ads

input this attribute...

Input true or false if you would like your website screenshot to not contain any ads

no_cookie_banners

input this attribute...

Input true or false if you would like your website screenshot to not contain of those annoying 'allow

width

input this attribute...

Choose the width of your screenshot (in pixels)

height

input this attribute...

Choose the height of your screenshot (in pixels)

🎉 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 query quota reminder to the top of your webpage

In this step, we'll add a reminder for users of how many queries they can make, considering that the limit for this API is low.

To do so, we will be making another API call to APIFlash, except this time it will be to their quota endpoint (Check out the specific docs with more info about this endpoint here.)

- [ ] To start, we need a state variable to keep track of our quota values (our limit of calls we started with and how many we have left).

✨ **AI Opportunity**

▼ *Use AI to explain code concepts* → Giving examples

If you've forgotten how to create a state variable, ask Copilot for an example and use that to help you create one for the quota.

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

```
const [quota, setQuota] = useState(null);
```

☐ Next, we want to create another `async` function to make our API call to this quota endpoint.

```
const getQuota = async () => {


}
```

☐ Make the API call using `fetch` to the APIFlash quota endpoint and save it in our `quota` state variable. Try it on your own!

✨ AI Opportunity

▼ *Use AI as a pair programming partner* → Navigating API calls

If you need help implementing this call, try pair programming with Copilot! You can give it the API call you wrote earlier as an example, and ask it to help navigate you through this step.

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

```
const getQuota = async () => {
  const response = await fetch("https://api.apiflash.com/v1/urltoimage/quota?access_key=
  const result = await response.json();

  setQuota(result);
}
```

- [ ] Let's call `getQuota()` at the end of `callAPI()` so every time that we make an API call, we also update our quota numbers

```
.
.
.
else {
  setCurrentImage(json.url);
  setPrevImages((images) => [...images, json.url]);
  reset();
  getQuota();
}
```

The `quota` response that will be returned by this API call will be a small dictionary with keys for the original limit, remaining calls, and when your limit resets (in UTC epoch seconds aka the number of seconds that have passed since January 1, 1970 until the day your quota actually resets 😂).

- [ ] Use conditional rendering again to display our quota.

✨ AI Opportunity

▼ *Use AI as a pair programming partner* → Conditional rendering

We used conditional rendering earlier, which we can use as an example to ask Copilot how we should implement this step. For example:

You:

> Here is my code where I used conditional rendering:
>
> (your code here)
>
> I need to use the same structure to conditionally render the photo gallery when there are images or display a message if there are not any photos to show. Can you show me how I would do this?

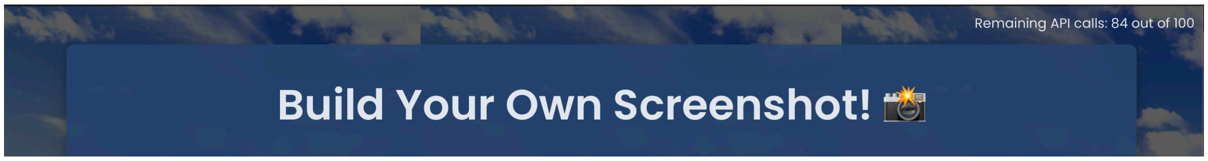
▼ **Want to double-check your code? Compare what you have to this ⤵️**

```
{quota ? (
  <p className="quota">
    {" "}
    Remaining API calls: {quota.remaining} out of {quota.limit}
  </p>
) : (
  <p></p>
)}
```

- [ ] And lastly, we can include some CSS so that it is fixed in the top right of our screen in white text (feel free to change the color for however you need it)

```
.quota {
  position: absolute;
  right: 0;
  top: 0;
  margin: 10px;
  color: white;
}
```

📍 **Checkpoint 5**: At the end of this step, you should now have a small message at the top of your webpage reminding you how many precious APIFlash queries you have left.


Remaining API calls: 84 out of 100
Build Your Own Screenshot! 📸

🎉 Congratulations 🎉

You've completed this lab AND the stretch goals! 🚀

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