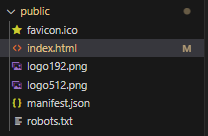
**React and Python Full Stack App Course**

**Creating a React App**[**https://create-react-app.dev/docs/getting-started/**](https://create-react-app.dev/docs/getting-started/)

Once created, you can run **npm start** and the app should start in a localhost tab.

**Changing Favicon**

There are different ways to edit the favicon icon. Favicon is not actually a single image.

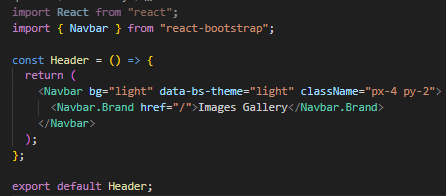
In the public folder, there are files such as **favicon.ico** or **logo.png**.

In the manifest.json, there is an array of different icons that are available for usage, and different client devices, different web browsers could access different icons.

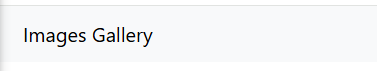
If a web browser requests an icon with size 32x32, then the application will service the relevant icon file.

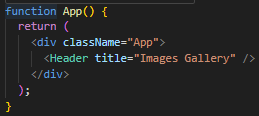
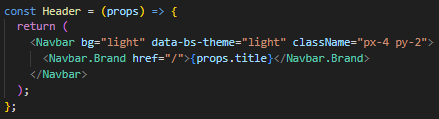
**Using React Boostrap**<https://react-bootstrap.netlify.app/docs/getting-started/introduction>

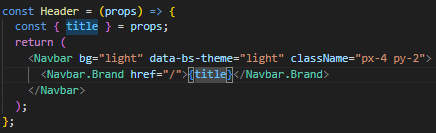
Before importing Bootstrap components, we need to import the specific stylesheet.



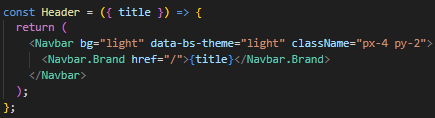
Use a basic navbar design from the link provided (make sure to add spacing if required).

You will get a very subtle whiteish grey navigation bar as a header once you import and display the JSX component in App.js.

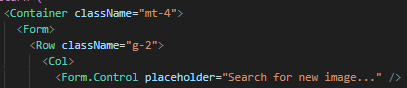
**Using Props in React Components**

Here, we pass the title prop through props into the Header arrow function component, which uses it.

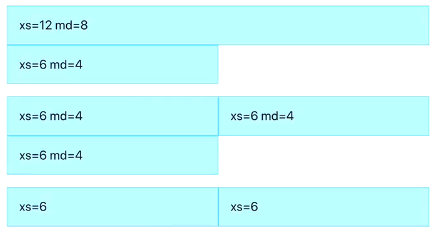
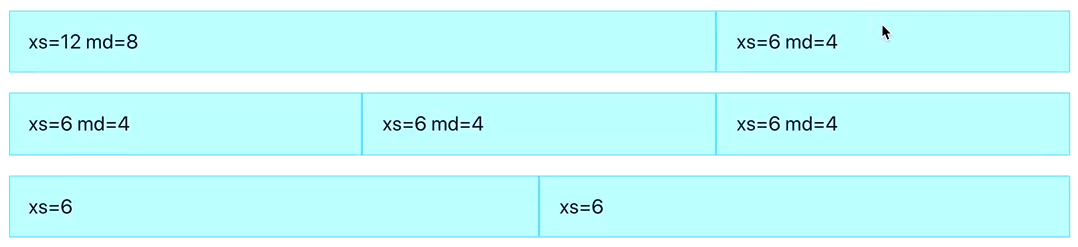
You can also extract the title prop using destructuring and then pass the prop name to the relevant place without dot notation.



Finally, the best way of doing it is to simply destructure the prop name itself in curly braces in the function, and pass this to the relevant place in the component.

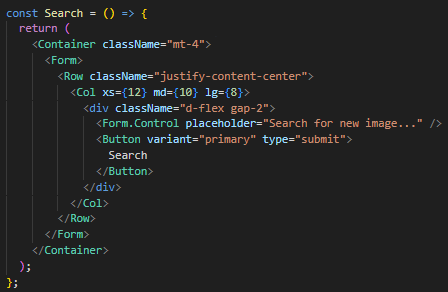
**React Bootstrap Styling**For example, **mt-4** or **g-2**.

* **Container  
  *container****:* A responsive fixed-width container  
  ***container****-fluid:* Always takes 100% of the width  
  ***container****-{breakpoint}:* Fixed-width container at the specified breakpoint, and fluid below it.
* **Row  
  *gx-0*** to ***gx-5:***Horizontal gutter (column spacing)  
  ***gy-0*** to ***gy-5:*** Vertical gutter (row spacing)
* **Col  
  *xs, sm, md, lg, xl, xxl:*** Control column width for each screen size. Accepts numbers 1-12, ***auto*** *or* ***true*** *(equal width).*
* **Bootstrap Breakpoints (used in Container, Row, Col)  
  *xs*** *–* extra small (<576px)  
  ***sm*** *–* small (>= 576px)  
  ***md***– medium (>=768px)  
  ***lg*** – large (>=992px)  
  ***xl***– extra large (>= 1200px)  
  ***xxl***– extra extra large (>=1400px)
* **Spacing Utilities (Margin and Padding)  
  *m, p:***margin, padding  
  ***t, b, s, e, x, y:***top, bottom, start, end, horizontal, vertical  
  ***0-5:***size scale (0 = 0, 1 = 0.25rem, …, 5 = 3rem)  
  ***auto:*** use auto value
* **Text Utilities  
  *text-start:*** Left-aligned text  
  ***text-end:*** Right-aligned text  
  ***text-center:*** Centre-aligned text  
  ***text-uppercase:*** Transform text to uppercase  
  ***text-muted:*** Muted (grey) text colour  
  ***fw-bold***: Font-weight bold  
  ***fs-4:*** Font size heading scale (1-6)
* **Display and Flex Utilities  
  *d-none:*** display: none  
  ***d-flex:*** display: flex  
  ***d-inline:*** display: inline  
  ***flex-column:*** flex direction column  
  ***justify-content-center:*** centre children horizontally  
  ***align-items-center:*** centre children vertically
* **Width/Height  
  *w-100:*** width: 100%  
  ***w-auto:*** width: auto  
  ***h-50***: height: 50%  
  ***min-vh-100:*** min-height: 100vh
* **Responsive Variants**If you were to give the **className=“mt-3 mt-md-5”**, this would mean that the margin-top would be 3 on all screen sizes, but overrides this to 5 when the screen size is md or up.

****You can also mix and match breakpoints to create different grids depending on the screen size. The screen is split into 12 imaginary columns (1-12).  
When the screen is of medium size, the columns will take up (e.g.) 2/3rds and 1/3rd of the screen respectively (md=8, md=4)

When the screen is super small (xs), then column 1 will take up the whole of the screen and column 2 will take up half of the screen (xs=12, xs=6).

**Xs** and **md** are not both applied at the same time, it is only when the screen size changes that they kick into action with their specified proportions.

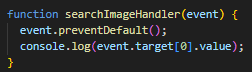
**Explaining my Code**

**Mt-4 –** adds a top spacing between the container and whatever is above it.

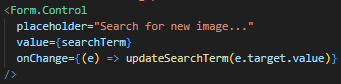
**Justify-content-center –** Centres the content of the row (the col below) in the horizontal space.

**Col xs md lg –** Responsive width that gets narrower on larger screens, helping centre and contain the form nicely.

**D-flex gap-2 –** input and button are laid out in a row (side by side), with 0.5rem space between.

**Why you need to use State in forms****This is incorrect.** We are accessing the form value, which is fine, but this approach couples the logic to the structure of the DOM, meaning the value itself lives in the DOM and not in React’s memory.

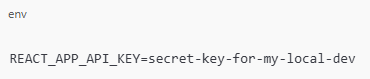
Instead, we need to use state, which causes React to track every keystroke, and update its value in React’s memory. This not only allows you to track the input more closely, but you can add features, such as enabling/disabling buttons based on input, resetting the input after submission and validate/transform the input as the user types.

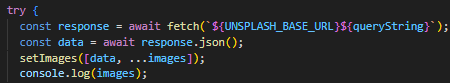
In the search component’s form, we see its value is equal to the current state. This is updated whenever a change is detected.

However, if **onChange** was commented out, we would be unable to type anything in the search bar as its value is equal to state, and if we can’t update state, its value cannot be updated.

**Using UnsplashAPI**<https://unsplash.com/developers>

**Different ENV files**

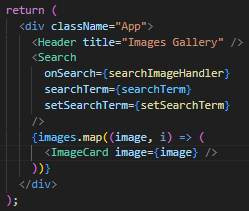
* **.env.local –** this is used only in your local development – by default it is in the .gitignore file so you can store any secrets here, such as API keys.  
    
  
* When deploying an application, you need to add the hidden API keys manually to your hosting provider’s environment variables settings.
* For default values or non-sensitive config that can be safely committed to version control, you can use the **.env** file, for instance:  
  ****

**How set state works when adding to arrays**

Each time an image is searched, it is added to the array. However, it is only when you search for the next image, you see the array update with the previous search.

This is because it is **asynchronous**, so logging the images object in the same statement will not display the updated state yet. Move the console log underneath the state statements.

**Using map to map over elements in the array**

****The images array contains multiple image objects.

We map over each object, and create a JSX component that takes the image as a parameter and creates an image card for it.

**Requests and Responses**

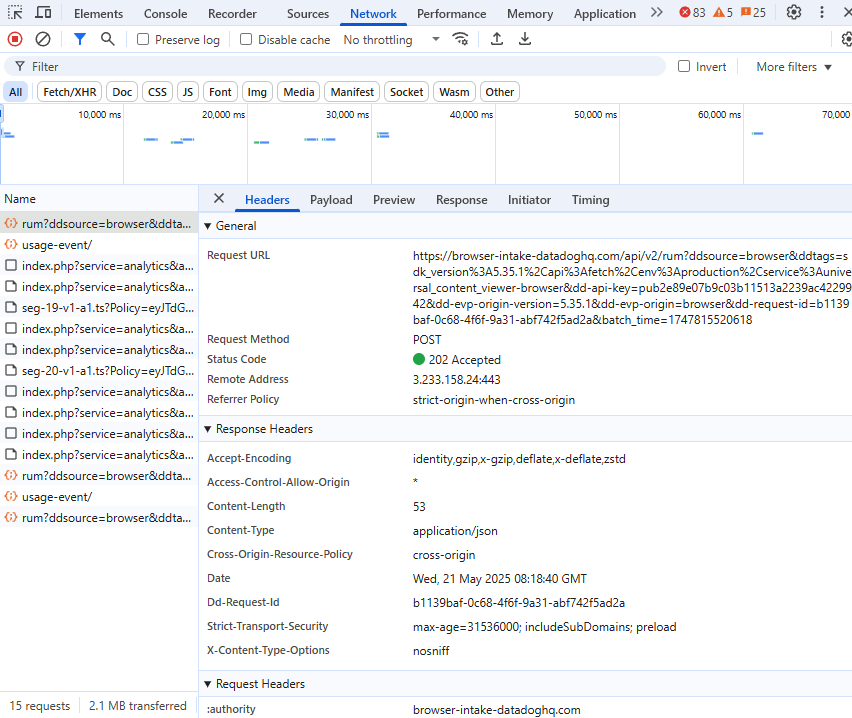
**Request –** sent from the client to the server

* **URL –** Identifier of the specific resource
* **Method –** GET, POST, DELETE and others
* **Headers –** Cookie, User-Agent, Authorisation and others
* **Data (body) –** (Optional) - JSON, Form Data or others

**Response –** sent from the server to the client

* **Status line –** Protocol version, status code, status text
* **Headers –** Content-Type, Date, Server, Set-Cookie and others
* **Data (body) –** (Optional) – JSON, Document, Stylesheet, PNG and others

**Understanding Requests from Browser**

****In the browser, under Network, you can click on any event name, and it will break down the type of request it was under the ‘Headers’ tab.

**Request Summary**

* **Request URL –** The full endpoint being called
* **Request Method –** HTTP method used (e.g. GET, POST)
* **Status Code –** (e.g. 200 OK) – the request succeeded
* **Remote Address –** The IP address and port of the server

**Request Headers**These are sent from your browser to the server (Client 🡪 Server)

* **Accept –** Tells the server what content types the client can handle
* **Content-type –** The format of the request body (e.g. application/json)
* **Cookie –** All relevant cookies tied to your session and identity
* **User-agent –** Your browser type, OS etc…
* **:authority, :method, :path, :scheme –** Pseudo-headers used in HTTP/2 for routing
* **Content-length –** Size of the request body in bytes

**Response Headers**These are the server’s metadata in response (Server 🡪 Client)

* **Access-control-allow-origin –** Controls CORS (Cross-Origin Resource Sharing) – allows web pages to access resources from a different domain that the one serving the page.
* **Content-type –** the format of the response body (e.g. application/json)
* **Cf-cache-status –** From Cloudflare – DYNAMIC means content is generated dynamically.
* **Strict-transport-security –** Enforces HTTPS (prevents downgrade to HTTP)
* **X-content-type-options –** Prevents MIME type sniffing (nosniff)
* **X-robots-tag –** Tells search engines not to index this content (nofollow)
* **Cf-ray, server, date –** Cloudflare-specific debug/trace info

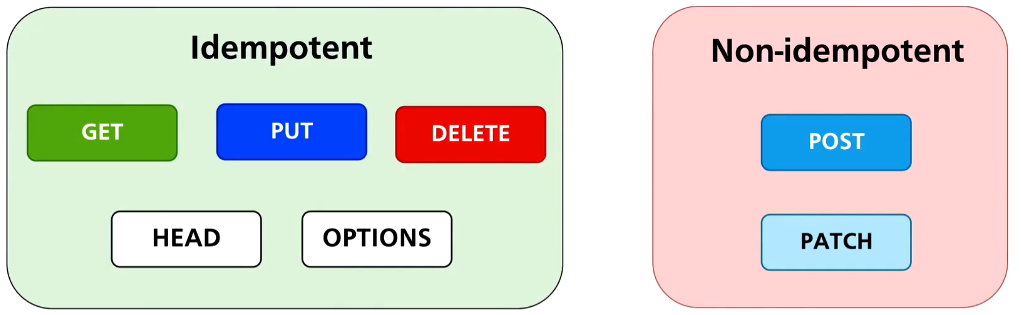
**Cookies**These carry authentication tokens, session state, preferences, and are sent automatically with the request due to the browser’s handling of sessions.

* **\_Secure-next-auth.session-token –** your ChatGPT session
* **Cf\_clearance, \_cf\_bm –** Cloudflare cookies used for bot protection and rate-limiting

**Security Context**

* **CORS Headers –** access-control-\* - ensure requests come from allowed origins.
* **Referrer Policy, Strict-Transport-Security,** and **Cross-Origin-Opener-Policy** are all mechanisms to harden security against XSS, data leaks, or clickjacking.

**Idempotent HTTP Methods**Idempotent operations produce the same result when executed multiple times.

**GET –** calling GET once or 100 times will return the same data and won’t change anything on the server.

**PUT –** sending the same PUT request multiple times has the same effect as sending it once – it keeps overwriting the resource with the same data.

**DELETE –** deleting something once or multiple times has the same end result: the resource is gone. Even if it’s already deleted, repeating the request doesn’t change anything more.

**HEAD –** is read-only and doesn’t change anything.

**OPTIONS –** just asks for information and doesn’t change anything.

**POST –** calling POST multiple times usually creates multiple resources. Each call can have a different effect.

**PATCH –** It **can** be idempotent, if the same PATCH request results in the same change every time. But it’s not **guaranteed** to be idempotent like PUT.

**What are HEAD and OPTIONS?**

* **HEAD –** similar to GET, but only asks for headers – not the actual content. For example, checking if a page exists or how big a file is, without downloading it.
* **OPTIONS –** Asks the server what methods are allowed for a resource. For example – “what can I do with this URL?”

**Using Pipenv and its Virtual Environment**

**Advantages of Pipenv**

1. **Automatic Virtual Environment Management**

* Pipenv creates a separate, isolated Python environment for each project.
* Prevents version conflicts between dependencies used in different projects.
* You don’t need to manually create/activate virtual environments, Pipenv does it for you.

1. **Simplified Dependency Management**

* Pipfile replaces *requirements.txt*and separates:  
  [packages] 🡪 for production dependencies  
  [dev-packages] 🡪 for development tools (e.g. linters, test runners)

1. **Lock Files for Reproducibility**

* Pipfile.lock ensurs consistent installs across machines and deployments.
* Guarantees exact versions, preventing “it works on my machine” bugs

1. **Cleaner and Easier to Use**

* Commands like *pipenv install*, *pipenv uninstall*, and *pipenv shell* are more user-friendly than juggling pip + virtualenv manually.
* Easy to activate a shell in the environment: *pipenv shell*

1. **Built-in Safety Checks**

* Automatically checks for known security vulnerabilities in the installed packages (*pipenv check*)

**Advantages of Flask**Flask is a micro web framework for Python. It’s minimal, but highly extensible.

1. **Lightweight and Minimal**

* Comes with only the essentials: routing, request/response handling, and templating.
* You can add only what you need (ORM, form validation, etc…)

1. **Modular and Flexible**

* No rigid structure – great for small projects, APIs, and even larger apps if architected carefully.
* Easy to integrate with other libraries like SQLAlchemy, Jinja2, WTForms, etc…

1. **Quick to Develop**

* Ideal for rapid prototyping and MVPs.
* Simple to get a “Hello, World” server running in minutes.

1. **Extensive Ecosystem**

* Huge number of plugins and extensions (*Flask-Login, Flask-Migrate* etc…)
* Large and active community with lots of tutorials and third-party tools.

1. **WSGI-Compatible and Deployment-Ready**

* Runs on any WSGI-compliant server (e.g. Gunicorn, uWSGI)
* Plays well with Docker, Heroku, AWS, and other deployment platforms.

**Pipenv + Flask = Ideal for Many Python Web Projects**

* Clean project isolation
* Easier collaboration and deployment
* Fast development cycle
* Security and reproducibility

**Setting Python Interpreter to .venv**This allows you to run your code using the interpreter inside **.venv** and not the global or Anaconda Python.

You can install and manage packages inside **.venv**, so dependencies are isolated for this project only.