

React & Single Page Apps

Frontend Programming



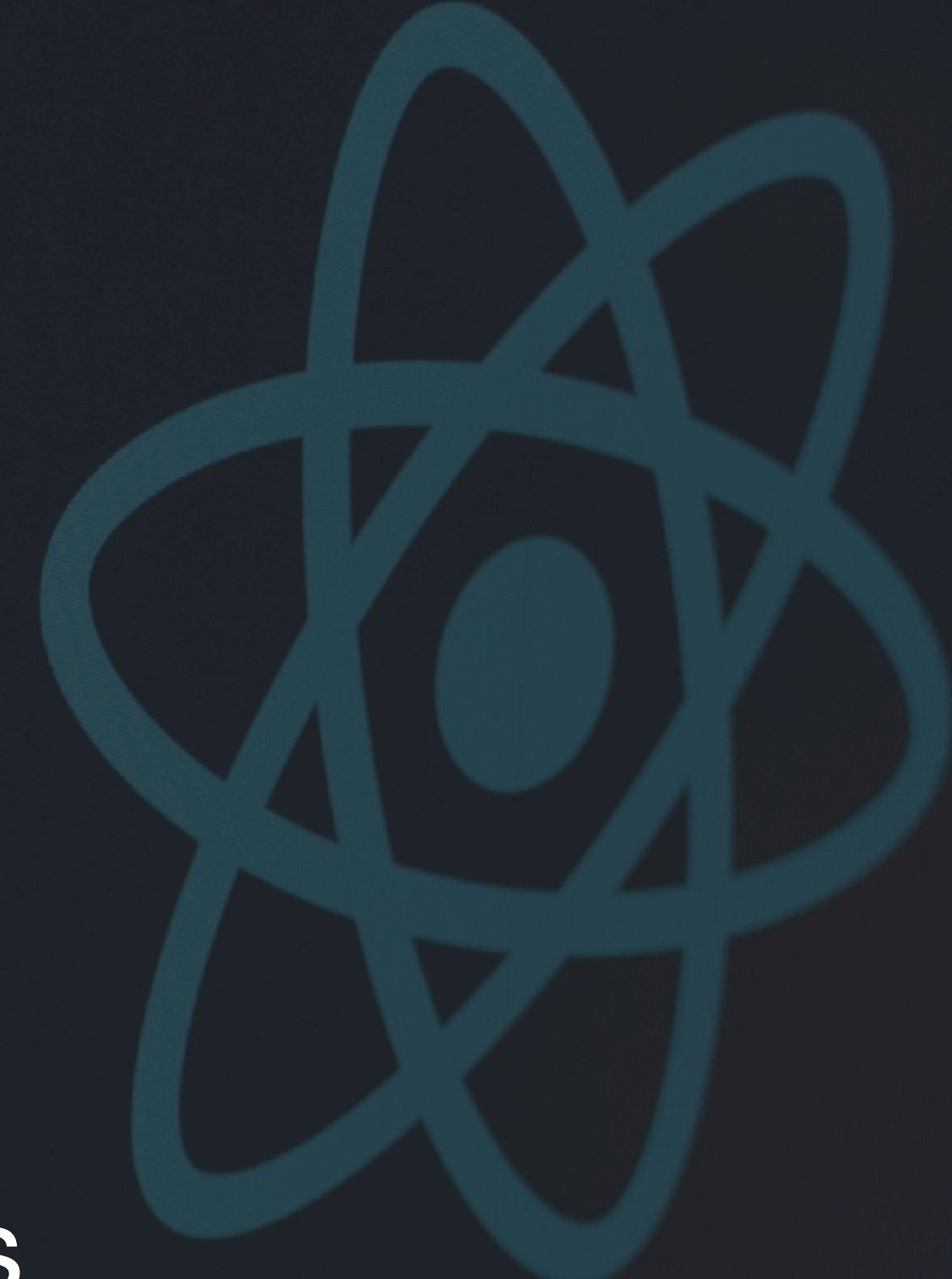
Edit src/App.js and save to reload
Learn React

Purpose

Wrap up some JS Basics

Introduce you to React & React Basics

Single Page Apps & Routing



Edit src/App.js and save to reload

Learn React

- Recap on JS Basics
- Introduction to React
- Thinking in React
- Single Page Apps & React Router



Agenda

- Array.map(...)
- JS Fetch Posts
- Your First React App
- Component Page Layout
- Add Props
- Tryout useState
- React Fetch Posts



JS & React Fun

RACE, slow down, please!

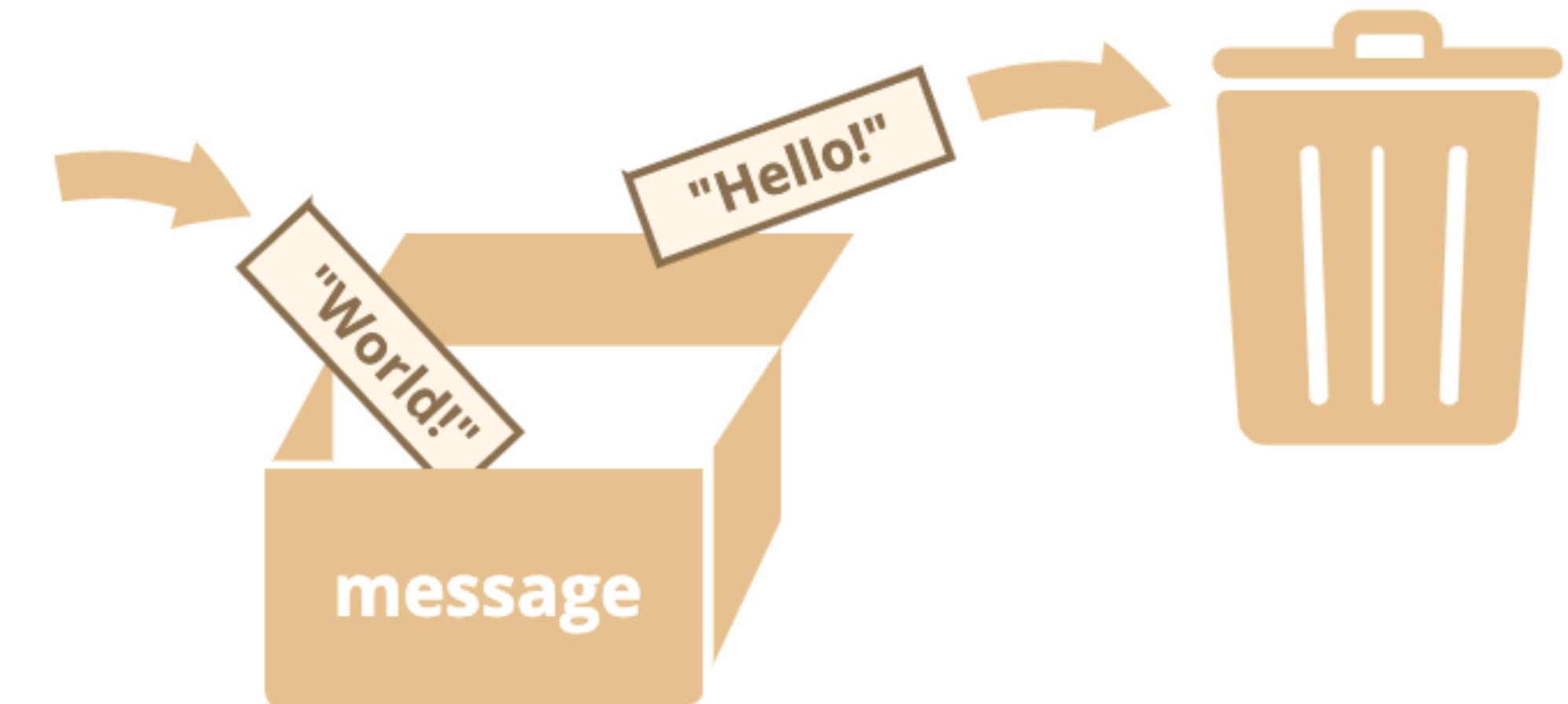
HTML & CSS Basics, CSS Layouts, Grids & Flexbox, Selectors
Variables, Objects, Arrays, Loops, Functions, DOM-
Manipulation, Template String, ES6+ Features, fetch, API, JSON,
Web Apps, Single Page Apps, Routing, CRUD, Forms & Events

Recap On

Variables, Objects, Arrays & Array Methods, Loops,
Functions, DOM-Manipulation, Template String &
Fetch

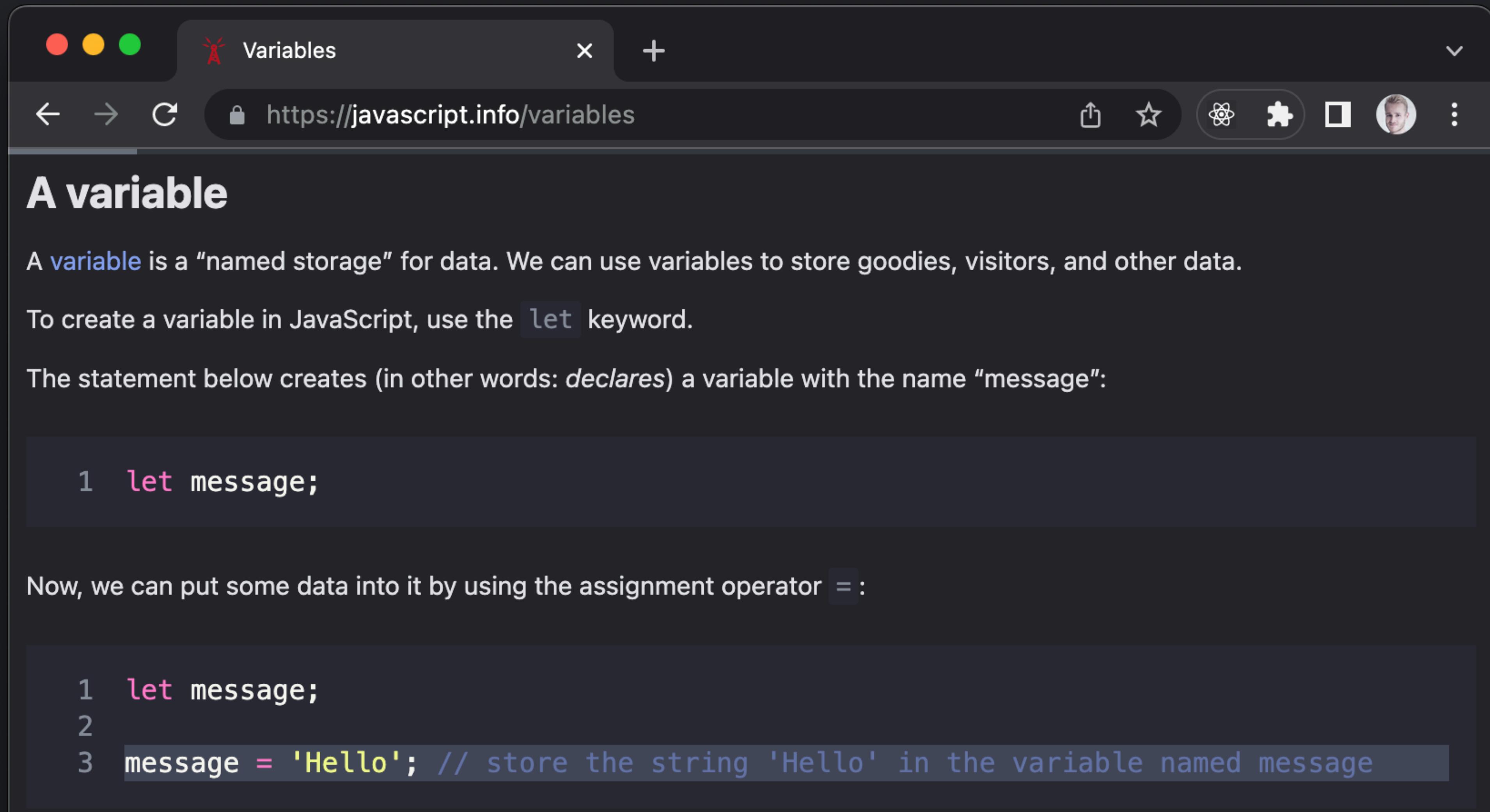
VARIABLE

A variable is a “named storage” and stored in the memory of the browser.



We can change the value of the variables as many times as we want.

JavaScript.info/Variables



The screenshot shows a dark-themed web browser window. The title bar reads "Variables". The address bar shows the URL "https://javascript.info/variables". The main content area displays the following text:

A variable

A **variable** is a “named storage” for data. We can use variables to store goodies, visitors, and other data.

To create a variable in JavaScript, use the `let` keyword.

The statement below creates (in other words: *declares*) a variable with the name “message”:

```
1 let message;
```

Now, we can put some data into it by using the assignment operator `=`:

```
1 let message;
2
3 message = 'Hello'; // store the string 'Hello' in the variable named message
```

CONST

Const is an unchanging variable.

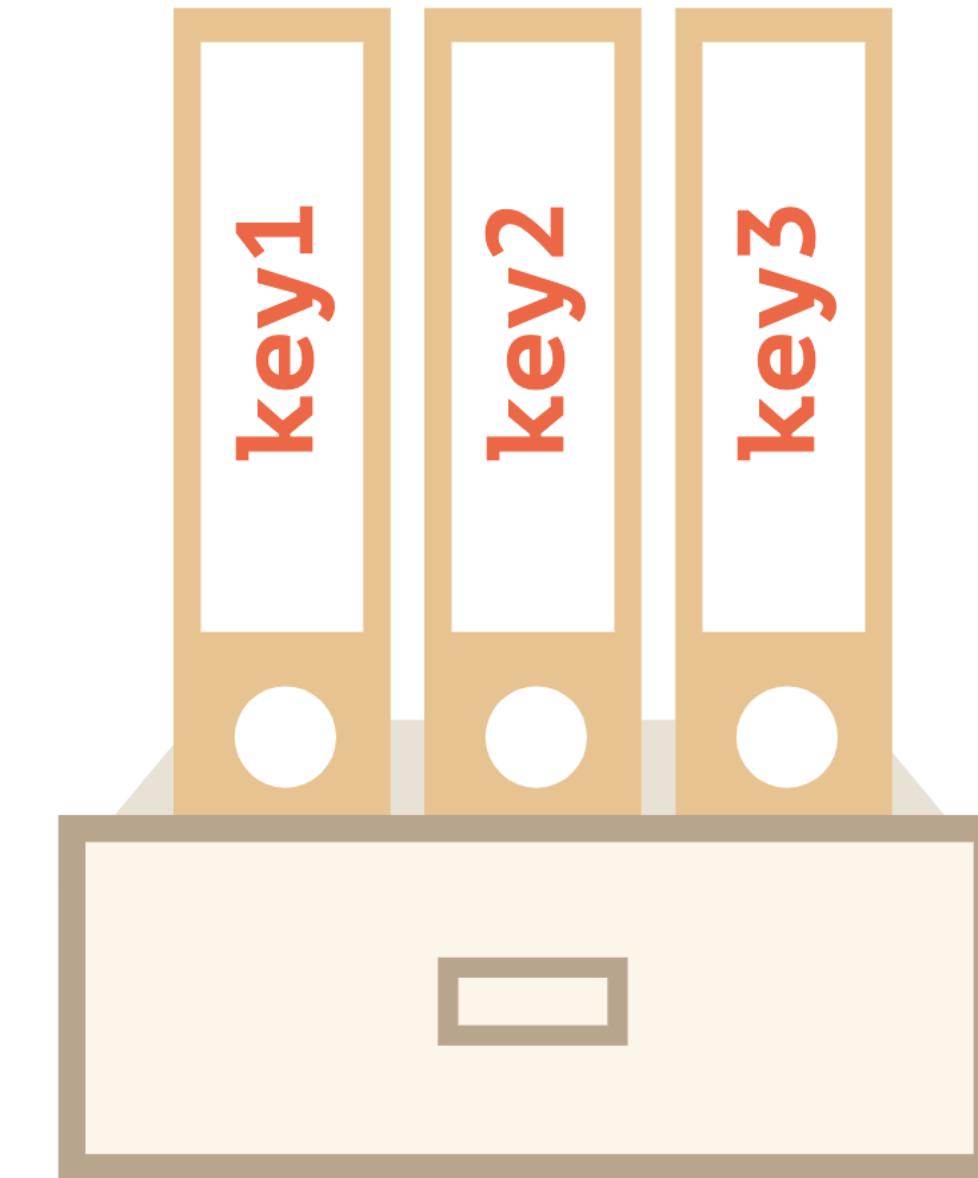
```
const myBirthday = "12-03-1990";
myBirthday = "12-03-1989";
// Uncaught TypeError: can't reassign the constant!
```

const cannot be reassigned.

If you try to, an error will be thrown.

O B J E C T S

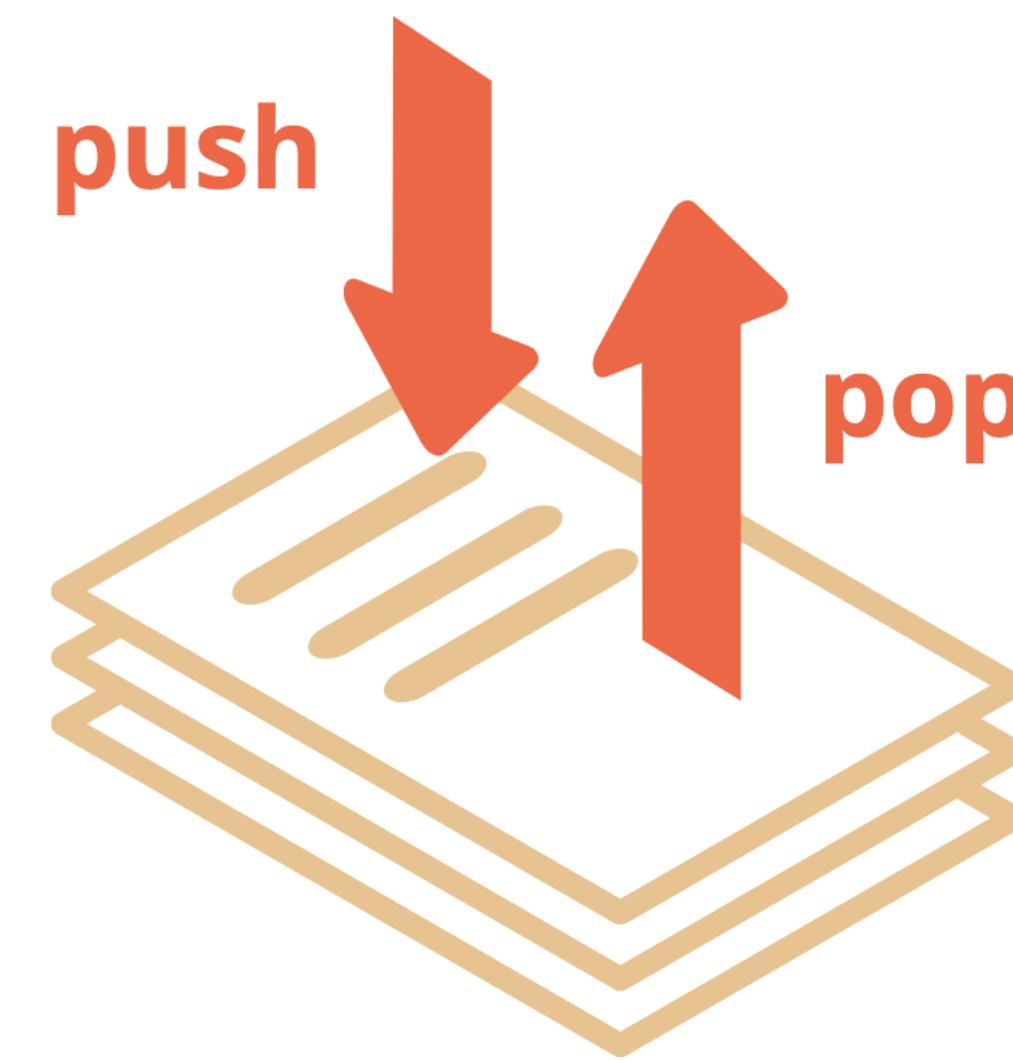
Objects are used to store keyed collections of various data



Containers for named values called properties. A property is a “key: value” pair

ARRAYS

Arrays are ordered collections



An array is a way to hold more than one value at a time we have a 1st, a 2nd, a 3rd, a 4th element and so on.

Array Methods to know

.push (...)

.map (...)

.filter (...)

.find (...)

.sort (...)

Computer science student



Senior developer, 10+ years experience



<https://www.instagram.com/p/BxWAgatgSmn/>

JavaScript.info/array#loops

One of the oldest ways to cycle array items is the `for` loop over indexes:

```
1 let arr = ["Apple", "Orange", "Pear"];
2
3 for (let i = 0; i < arr.length; i++) {
4   alert( arr[i] );
5 }
```



But for arrays there is another form of loop, `for..of`:

```
1 let fruits = ["Apple", "Orange", "Plum"];
2
3 // iterates over array elements
4 for (let fruit of fruits) {
5   alert( fruit );
6 }
```



Array.map(...)

...iterate over an array and modify each element.

Array.map(...) calls a callback function for each element in the array.

```
const persons = [
  { firstname: "Birgitte", lastname: "Iversen" },
  { firstname: "Lykke", lastname: "Dahlen" },
  { firstname: "Rasmus", lastname: "Cederdorff" }
];

const mapped = persons.map(person => {
  return {
    name: `${person.firstname} ${person.lastname}`
  };
});

console.log(mapped);
```

▼ (3) [{...}, {...}, {...}] *i*

- 0: {name: 'Birgitte Iversen'}
- 1: {name: 'Lykke Dahlen'}
- 2: {name: 'Rasmus Cederdorff'}

length: 3

■ ■ ■ ■	.map(■ → ●)	→	● ● ● ●
■ ■ ● ■	.filter(■)	→	■ ■ ■
● ● ■ ■	.find(■)	→	■
● ● ● ■	.findIndexof(■)	→	3
■ ■ ■ ■	.fill(1, ●)	→	■ ● ● ●
● ■ ■ ●	.some(■)	→	true
■ ■ ■ ●	.every(■)	→	false

<https://javascript.info/array-methods>

<https://medium.com/@mandeepkaur1/a-list-of-javascript-array-methods-145d09dd19a0>

Array.map(...)

- Project template: array-map
- Use .map to map over the persons array and concatenate firstName and lastName to a new property called name.
- console.log() the result
- Add a property called birthYear for each person object. Use map to map over the array and add the property birthYear dynamically based on birthDate (hint: use String.split(...) or .slice(...)).

```
const persons = [  
  {  
    firstName: "Jane",  
    lastName: "Doe",  
    birthDate: "1992-03-04"  
  },  
  {  
    firstName: "Jens",  
    lastName: "Jensen",  
    birthDate: "1992-07-04"  
  },  
  {  
    firstName: "Birgitte",  
    lastName: "Iversen",  
    birthDate: "1990-10-04"  
  },  
  {  
    firstName: "Lykke",  
    lastName: "Dahlen",  
    birthDate: "1987-06-04"  
  },  
  {  
    firstName: "Kasper",  
    lastName: "Topp",  
    birthDate: "1989-03-07"  
  }];  
  
const result = persons.map(person => {  
  console.log(person);  
  // manipulate and return value  
});
```

`template string`

“Template literals are literals delimited with backtick (`) characters, allowing for multi-line strings, for string interpolation with embedded expressions, and for special constructs called tagged templates.”

```
let name = "Alicia";  
let age = 6;
```

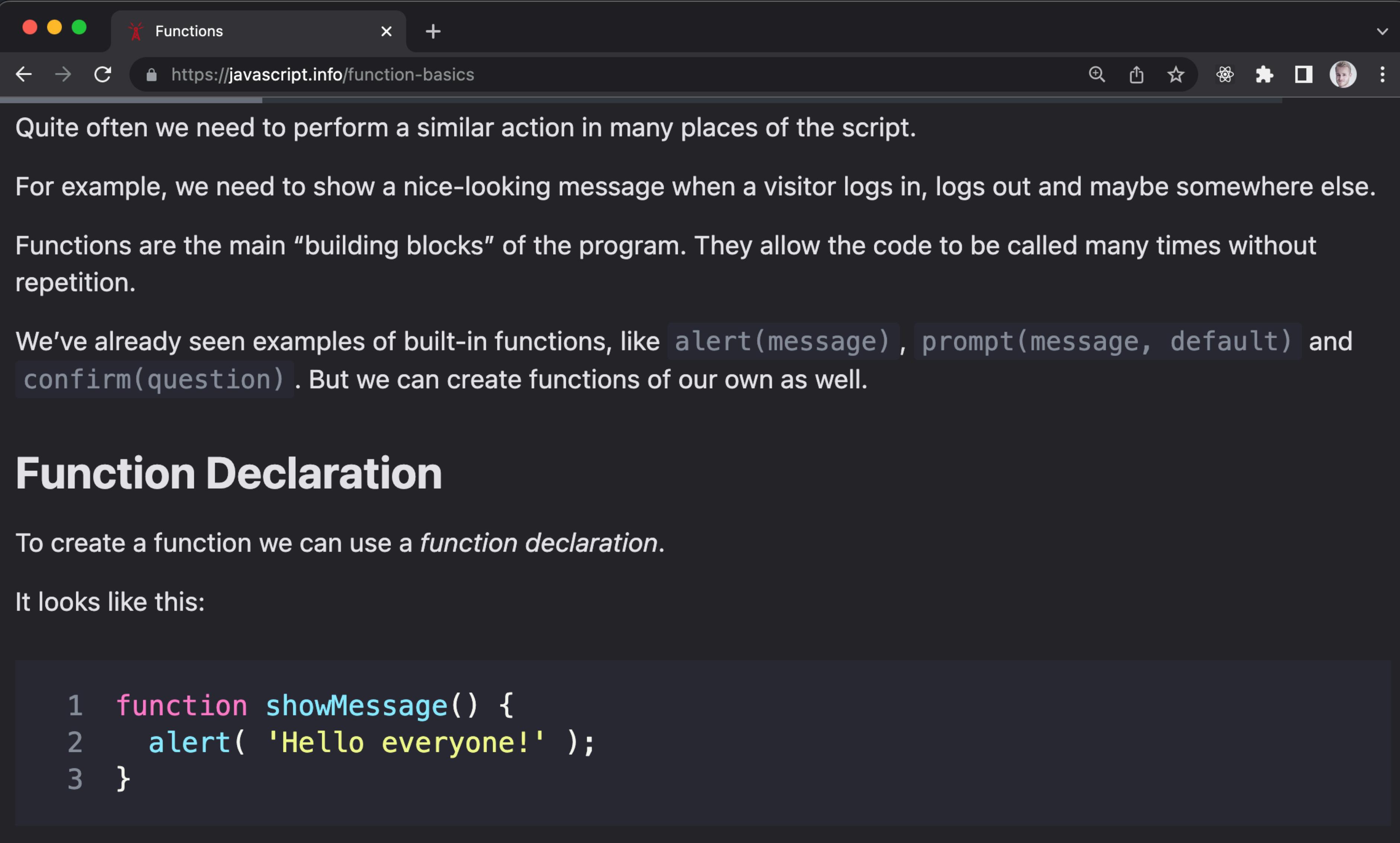
```
console.log(name + " is " + age + " years old.");
```

```
console.log(` ${name} is ${age} years old. `);
```

```
Alicia is 6 years old.  
Alicia is 6 years old.
```

[main.js:10](#)
[main.js:12](#)

JavaScript.info/Function-Basics



The screenshot shows a dark-themed web browser window with the title bar "Functions". The address bar displays the URL "https://javascript.info/function-basics". The main content area contains text explaining the purpose and benefits of functions, followed by a code example.

Quite often we need to perform a similar action in many places of the script.
For example, we need to show a nice-looking message when a visitor logs in, logs out and maybe somewhere else.
Functions are the main “building blocks” of the program. They allow the code to be called many times without repetition.
We've already seen examples of built-in functions, like `alert(message)`, `prompt(message, default)` and `confirm(question)`. But we can create functions of our own as well.

Function Declaration

To create a function we can use a *function declaration*.

It looks like this:

```
1 function showMessage() {  
2     alert( 'Hello everyone!' );  
3 }
```

JavaScript.info/function-basics#parameters

We can pass arbitrary data to functions using parameters.

In the example below, the function has two parameters: `from` and `text`.

```
1 function showMessage(from, text) { // parameters: from, text
2   alert(from + ': ' + text);
3 }
4
5 showMessage('Ann', 'Hello!'); // Ann: Hello! (*)
6 showMessage('Ann', "What's up?"); // Ann: What's up? (**)
```

JavaScript.info/function-basics#local-variables

Local variables

A variable declared inside a function is only visible inside that function.

For example:

```
1 function showMessage() {  
2   let message = "Hello, I'm JavaScript!"; // local variable  
3  
4   alert( message );  
5 }  
6  
7 showMessage(); // Hello, I'm JavaScript!  
8  
9 alert( message ); // <-- Error! The variable is local to the function
```

Scopes:

- Local Variable
- Global Variable

Outer variables

A function can access an outer variable as well, for example:

```
1 let userName = 'John';  
2  
3 function showMessage() {  
4   let message = 'Hello, ' + userName;  
5   alert(message);  
6 }  
7  
8 showMessage(); // Hello, John
```

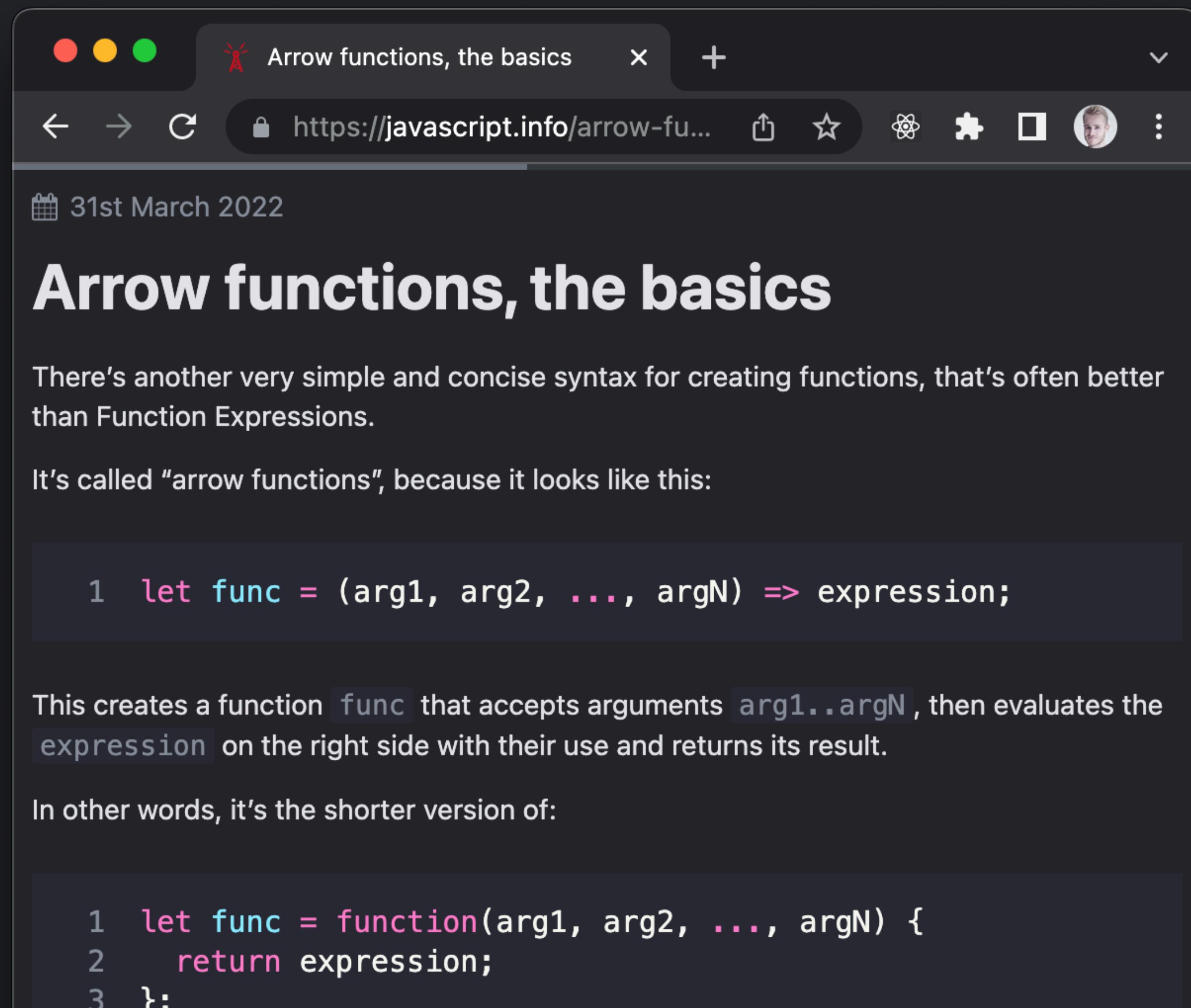
Function called by another function

Use the name of
the function to
call / execute

```
let users = [...  
];  
  
function appendUsers(users) {  
  let htmlTemplate = "";  
  for (const user of users) {  
    console.log(user);  
    htmlTemplate += /*html*/`  
      <article>  
          
        <h2>${user.name}</h2>  
        <a href="mailto:${user.email}">${user.email}</a>  
        <p>Role: ${user.enrollment_type}</p>  
      </article>  
    `;  
  }  
  document.querySelector("#users").innerHTML = htmlTemplate;  
}  
  
function initApp() {  
  appendUsers(users);  
}  
  
initApp();
```

users refers to our declared variable, users, (global variable)

Javascript.info/Arrow-Functions-Basics



The screenshot shows a dark-themed web browser window. The title bar says "Arrow functions, the basics". The address bar shows the URL "https://javascript.info/arrow-fu...". Below the address bar, there's a date "31st March 2022". The main content area has a large heading "Arrow functions, the basics". Below the heading, a text block says: "There's another very simple and concise syntax for creating functions, that's often better than Function Expressions. It's called "arrow functions", because it looks like this:". A code block shows the syntax: "1 let func = (arg1, arg2, ..., argN) => expression;". Below this, a text block explains: "This creates a function `func` that accepts arguments `arg1..argN`, then evaluates the `expression` on the right side with their use and returns its result. In other words, it's the shorter version of:". Another code block shows the equivalent function expression: "1 let func = function(arg1, arg2, ..., argN) { 2 return expression; 3 };".

Fetch

... get & post data from and to a data source

```
// Simple javascript 😞  
  
//Synchronous fetch using async/await.  
  
// Usual way  
✓ const jsonData = fetch('URL')  
    .then(response => response.json())  
    .then(json => console.log(json));  
  
// Using await  
✓ const jsonData = await fetch('URL').then(res => res.json())  
  
// Shorter syntax 😊  
✓ const jsonData = await (await fetch('URL')).json();
```

<https://www.instagram.com/p/B0nxQjXj9Zi/>

Fetch

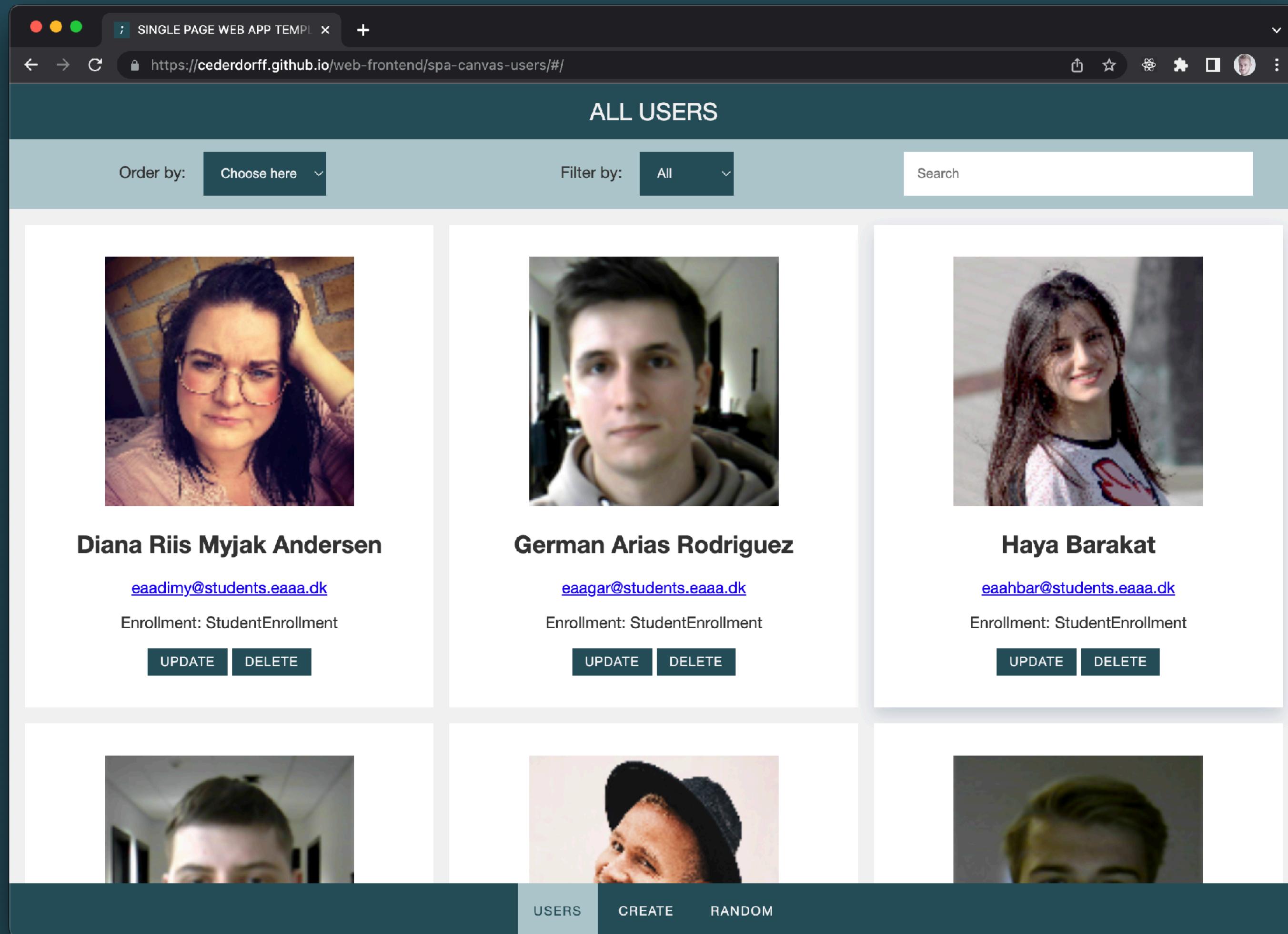
... can perform network requests to a server

```
1 let promise = fetch(url, [options])
```

- `url` – the URL to access.
- `options` – optional parameters: method, headers etc.

Without `options`, this is a simple GET request, downloading the contents of the `url`.

Canvas Users SPA Case

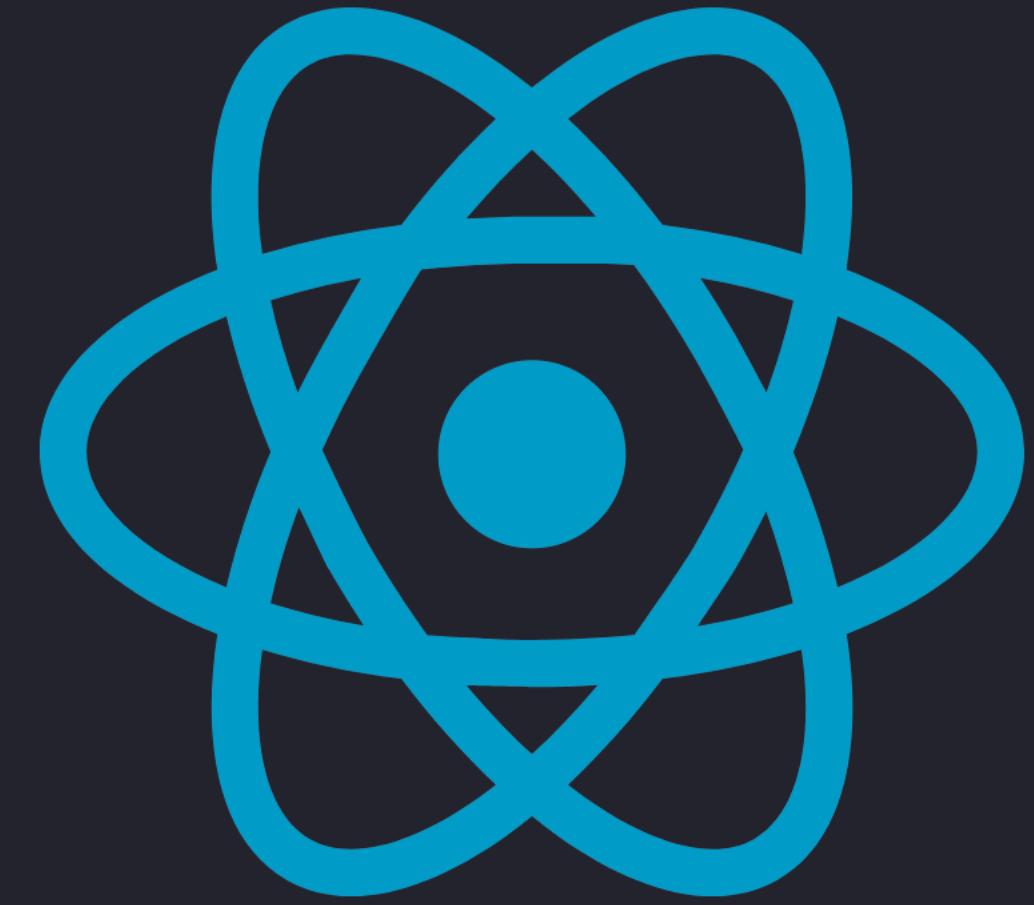


Template: spa-canvas-users-template

Solution: [spa-canvas-users](#)

1. Make a copy of project-template and use it as your codebase.
2. In JS, declare a global variable called `_posts`.
3. Create a function called `getPosts` fetching from the URL. Save the fetched post data in the global variable.
4. Create a function called `appendPosts`, using a for-of loop to loop through `_posts` and append them to the DOM. All Posts must be displayed in a grid or flexbox.
5. Create a search bar. On keyup call a function named `searchPosts`, searching through `_posts` and finding all matches on `post.title.includes(searchValue)`. Make sure the search logic works on all type of characters. Update the DOM using `appendPosts` with the filtered result array.
6. Reimplement `appendPersons` using `Array.map(...)` to generate HTML of every post and append the HTML to the DOM.
7. Extra: Implement creating a new post, using a form with input fields. Add `onsubmit` event on the form calling a function named `createPost` using `_posts.push(...)` to add a new post. Make sure the DOM is updated.

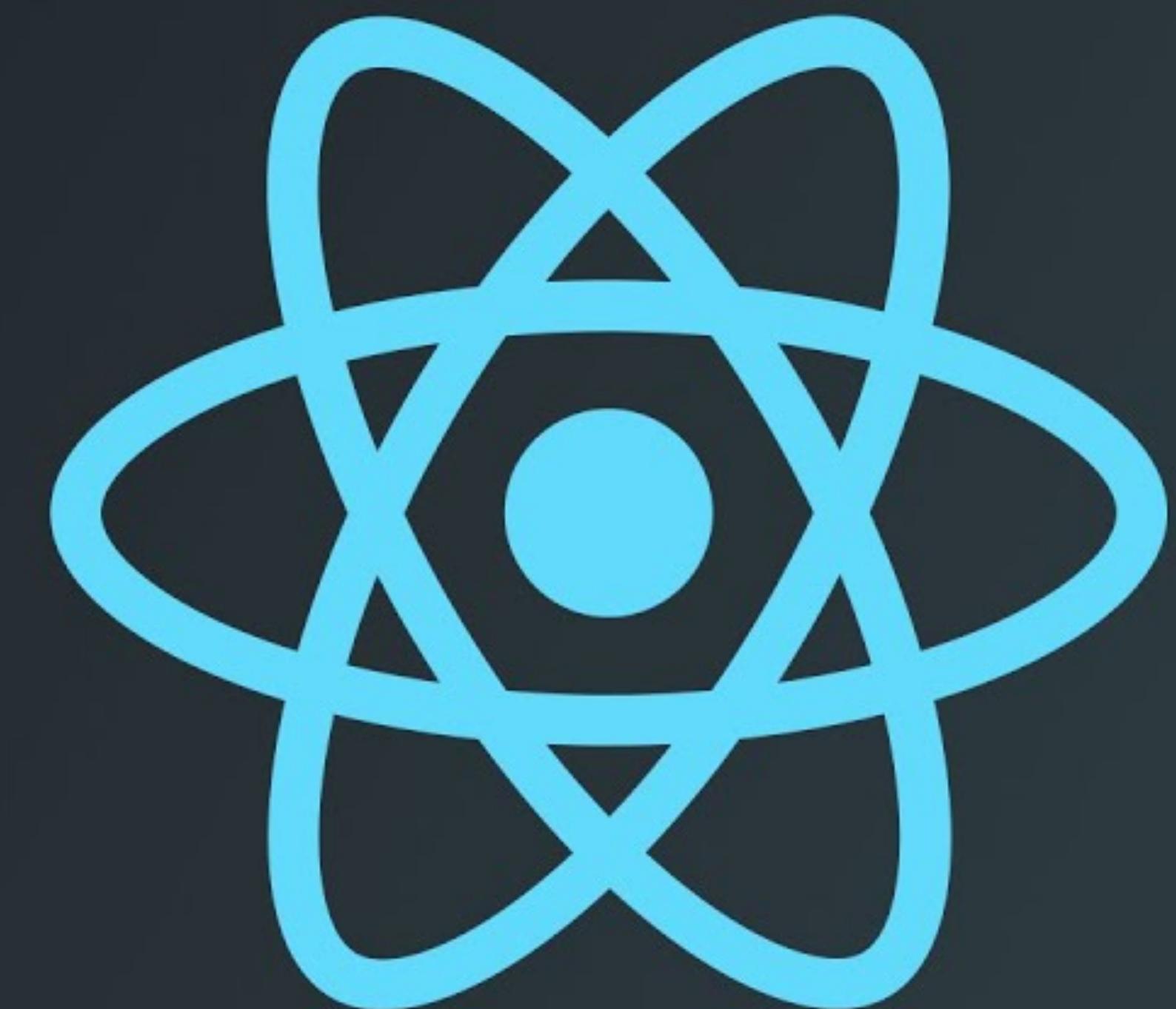


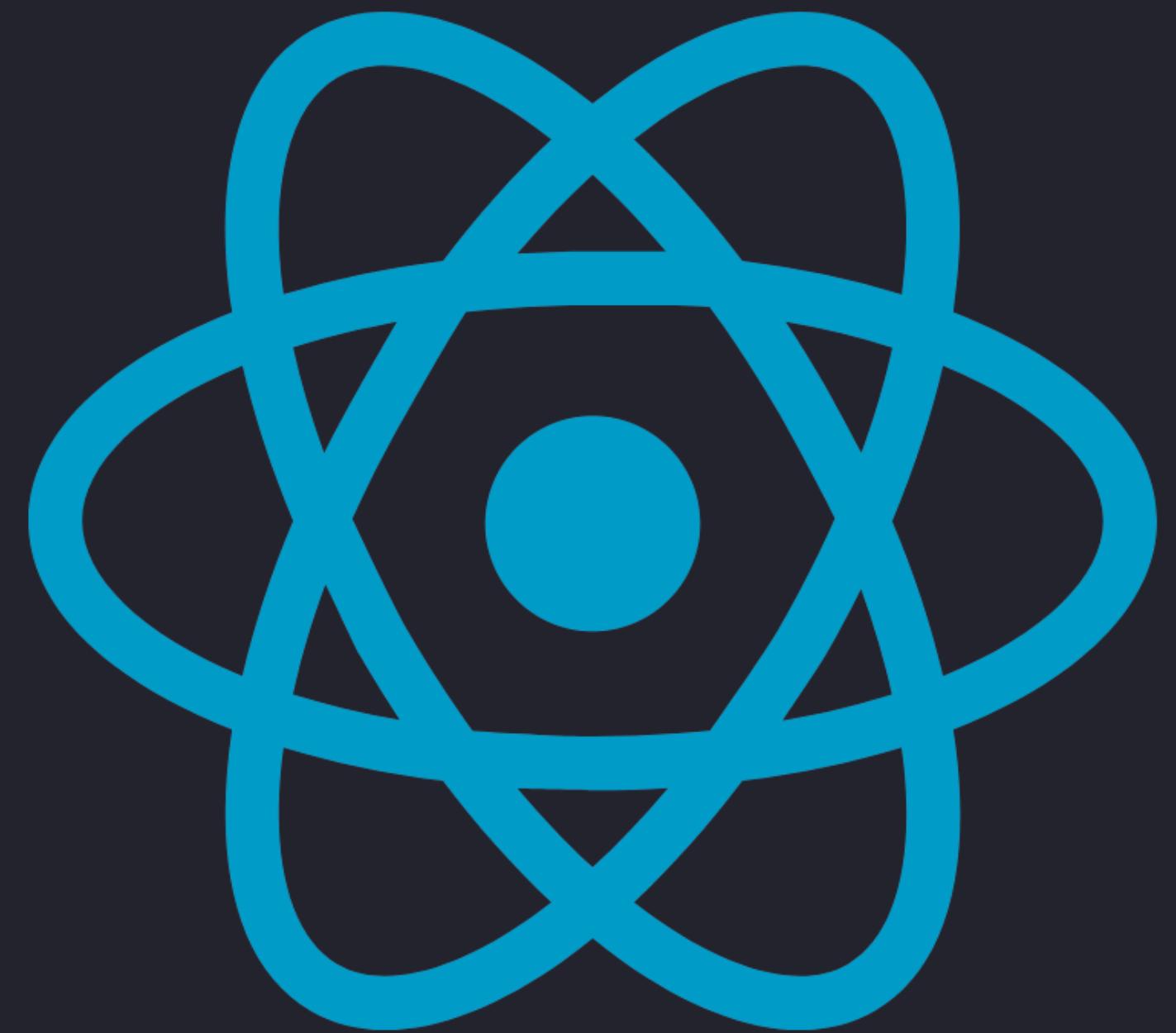


React

A JavaScript library for building User Interfaces

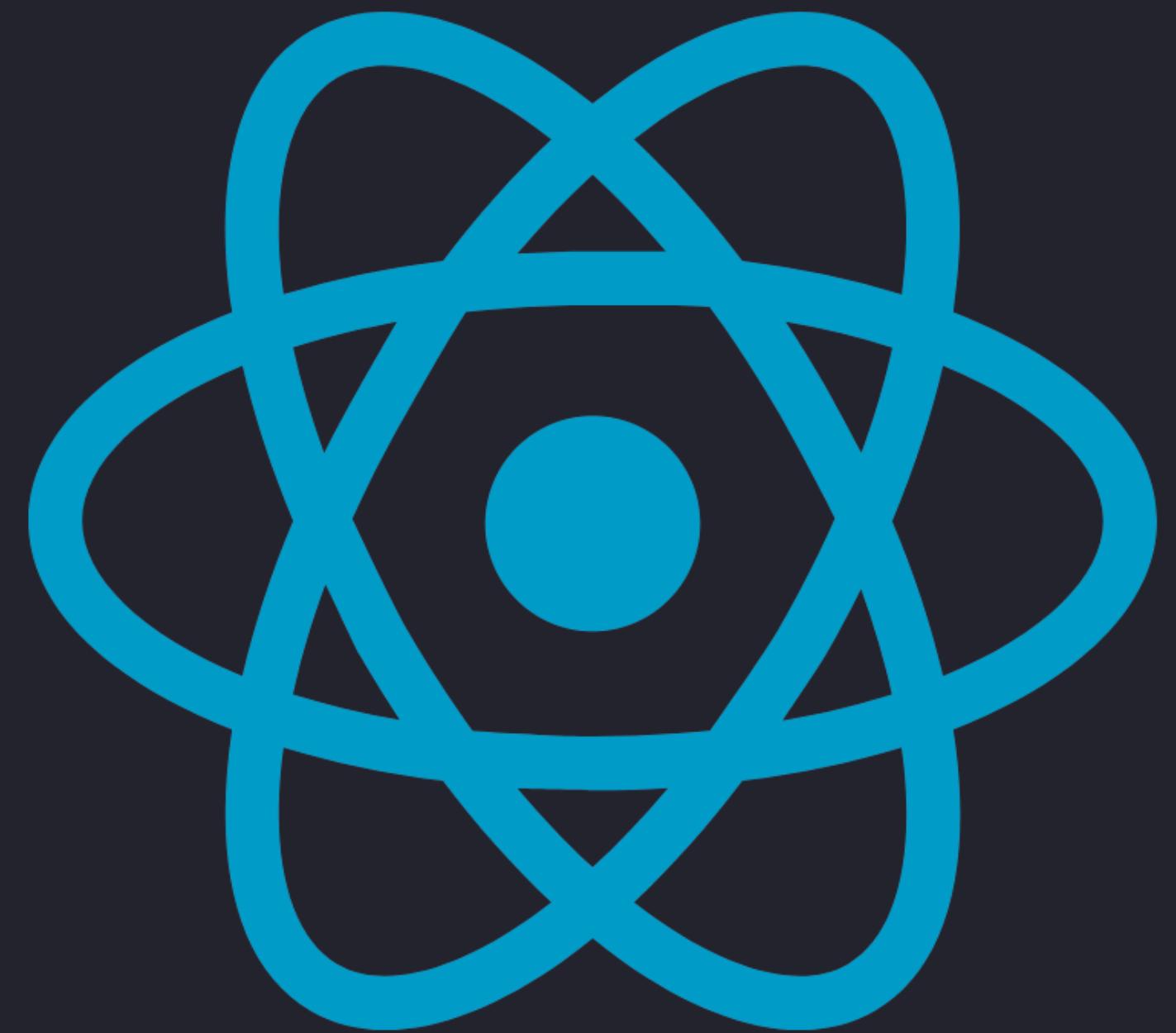
100 *SECONDS OF*





What is React?

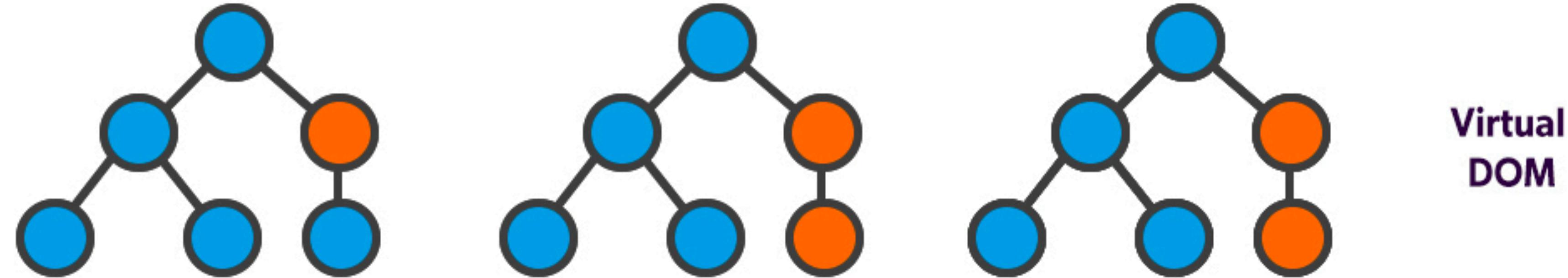
- A JavaScript library for building User Interfaces.
- Allow us to create reusable UI Components.
- Single Page Apps



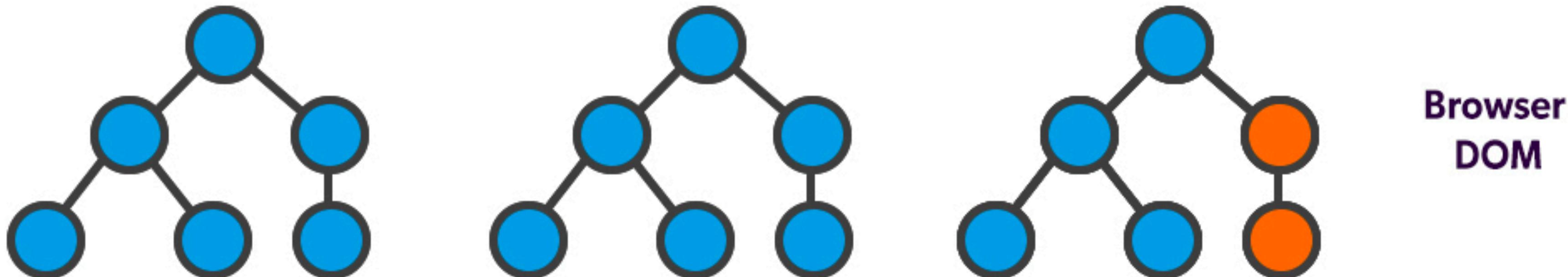
But how?

- React creates a Virtual DOM instead of manipulating directly with the browser's DOM.
- React makes the changes in Virtual DOM and compares it to the browser DOM.
- Then React detects what's needs to be changed in the browser DOM and changes **only** what needs to be changed!

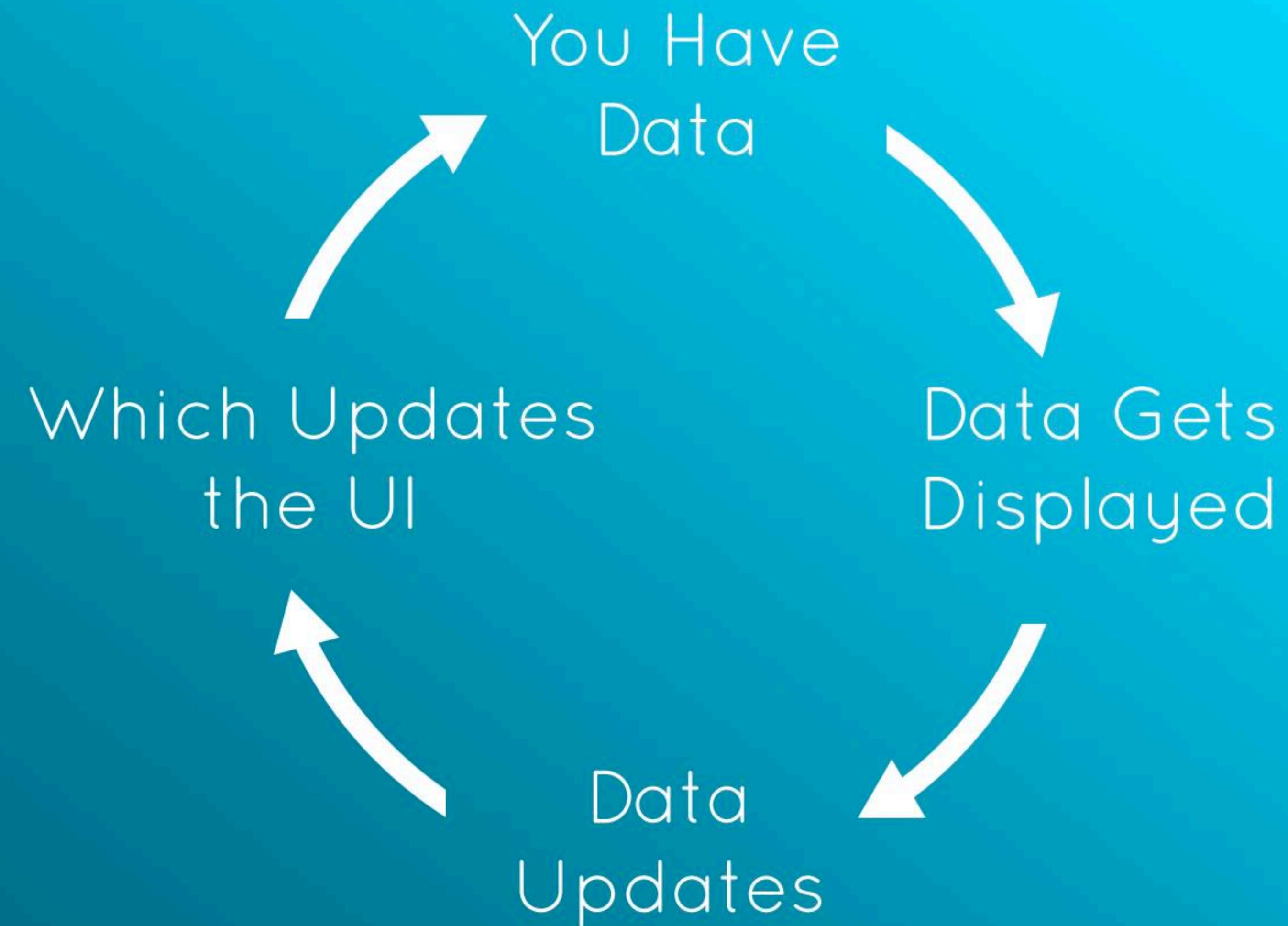
React Virtual DOM



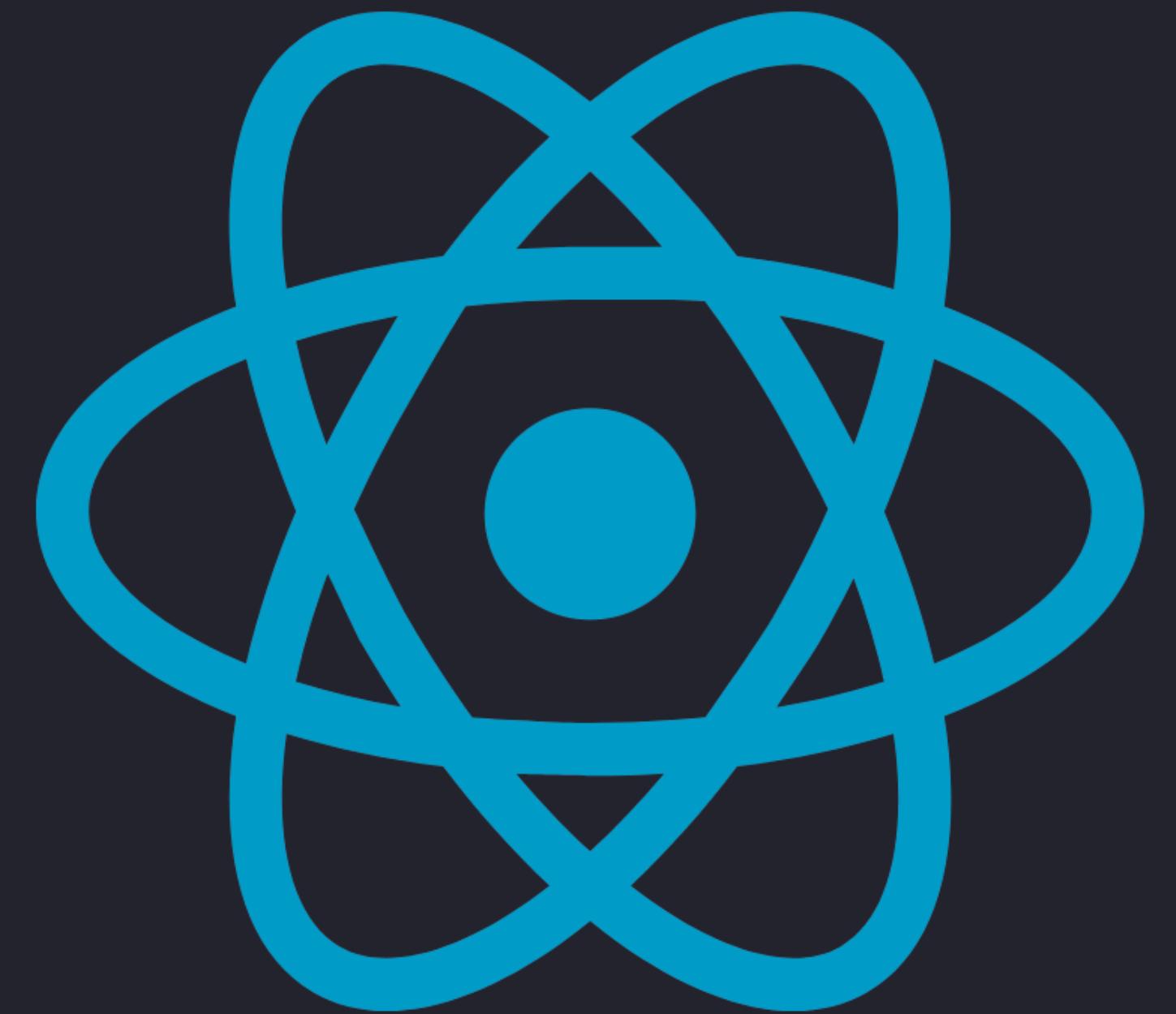
Statechange → Compute Diff → Re-render



The React UI Cycle



This happens AUTOMATICALLY
once you define components



But why?

- It's so hard to work with the DOM
- You must make sure the DOM is constantly updated and synchronised with the state of your data.
- React *reacts* to **changes** and then updates the DOM automatically and performant without rerendering all elements - only the needed elements.

1. Copy and explore the following template: [react-cdn-template](#)
2. In `app.jsx` and `App` component, replace the value of `name` attribute in the `Greeting` component.
3. Repeat the `Greeting` component and type another name value.
4. Create a new component called `Person`. Inside of the `Person` component, define a `const name` with a value. Return the `name` variable in an `h2` wrapped in an `article` tag.

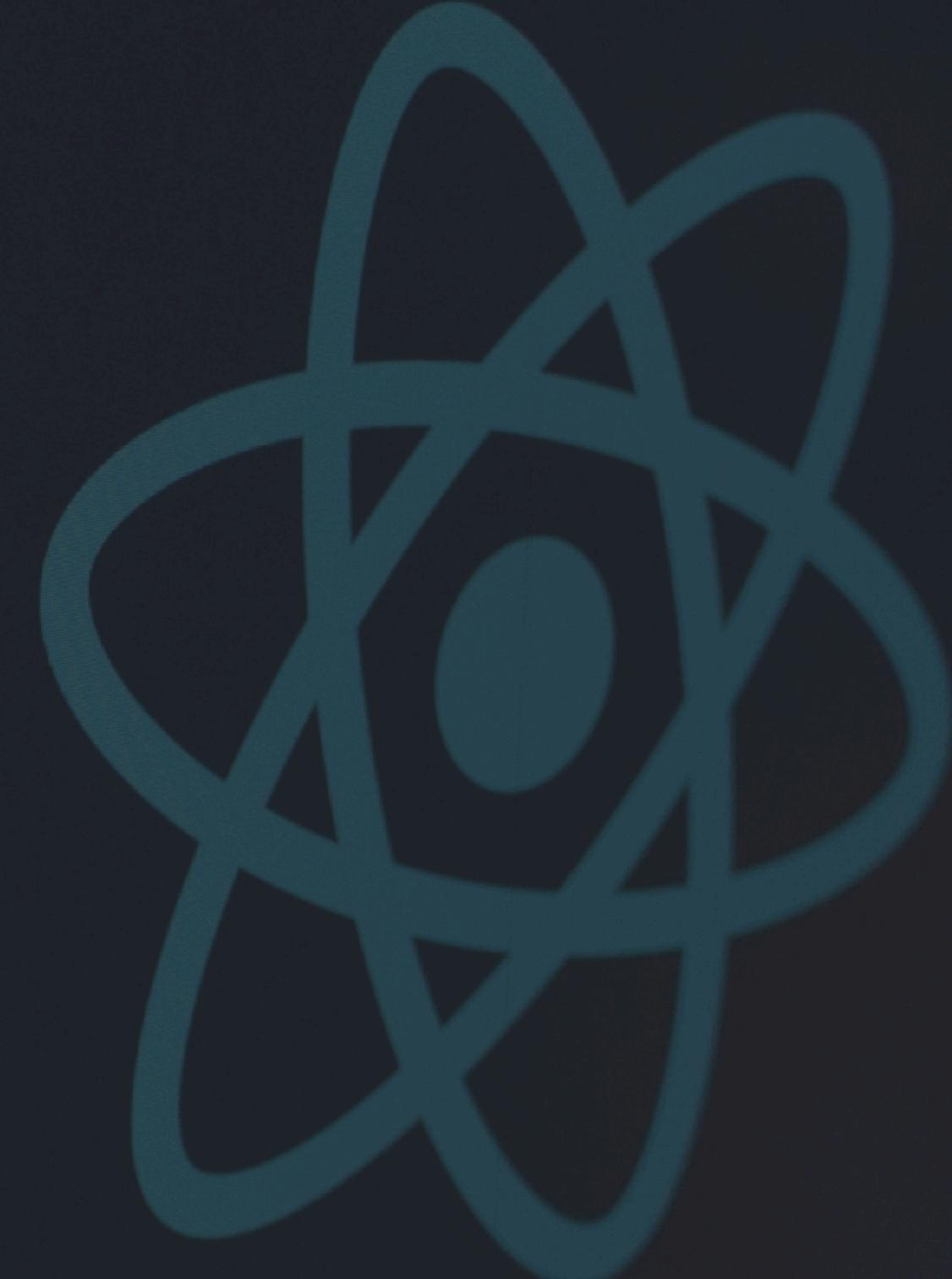


Resources

React Resources

React Self-study

Guides & Tutorials



Edit src/App.js and save to reload
Learn React

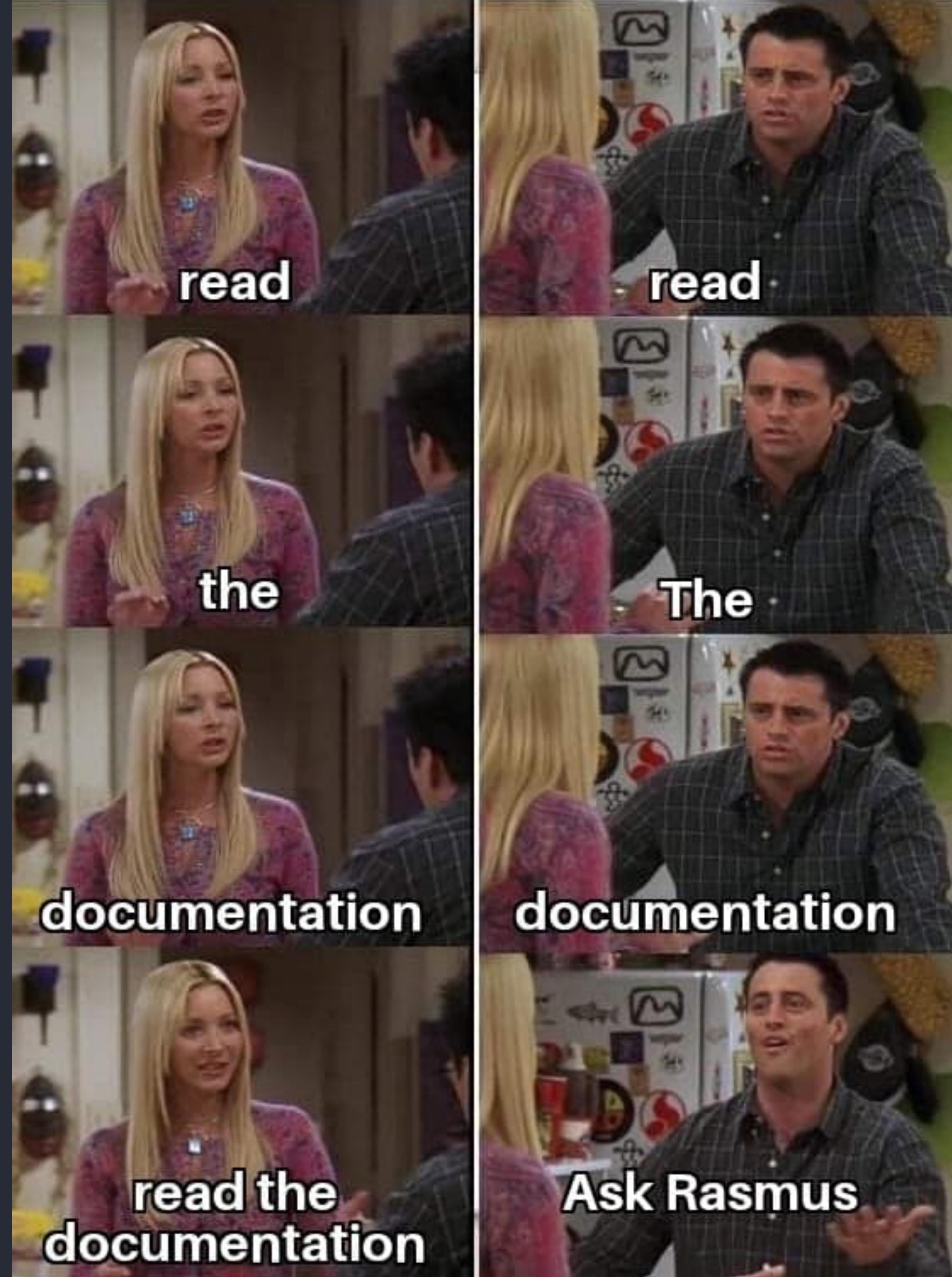
React Beta Docs

The screenshot shows a dark-themed web browser window for 'React Docs Beta'. The address bar displays 'https://beta.reactjs.org'. A prominent banner at the top reads 'Support Ukraine' with a link to 'Help Provide Humanitarian Aid to Ukraine'. The main navigation bar includes links for 'Home' (underlined), 'Learn', and 'API', along with a search bar and keyboard shortcut keys ('⌘ K'). On the left, there's a 'Community' section with a right-pointing arrow. The central content area features the React logo and the title 'React Docs' with a 'BETA' badge. Below this are two main sections: 'Learn React' (described as learning how to think in React with step-by-step explanations and interactive examples) and 'API Reference' (described as looking up API signatures and visual code diagrams). Each section has a 'Read More >' button. The overall design is clean and modern, with a focus on dark mode.

What is this site?

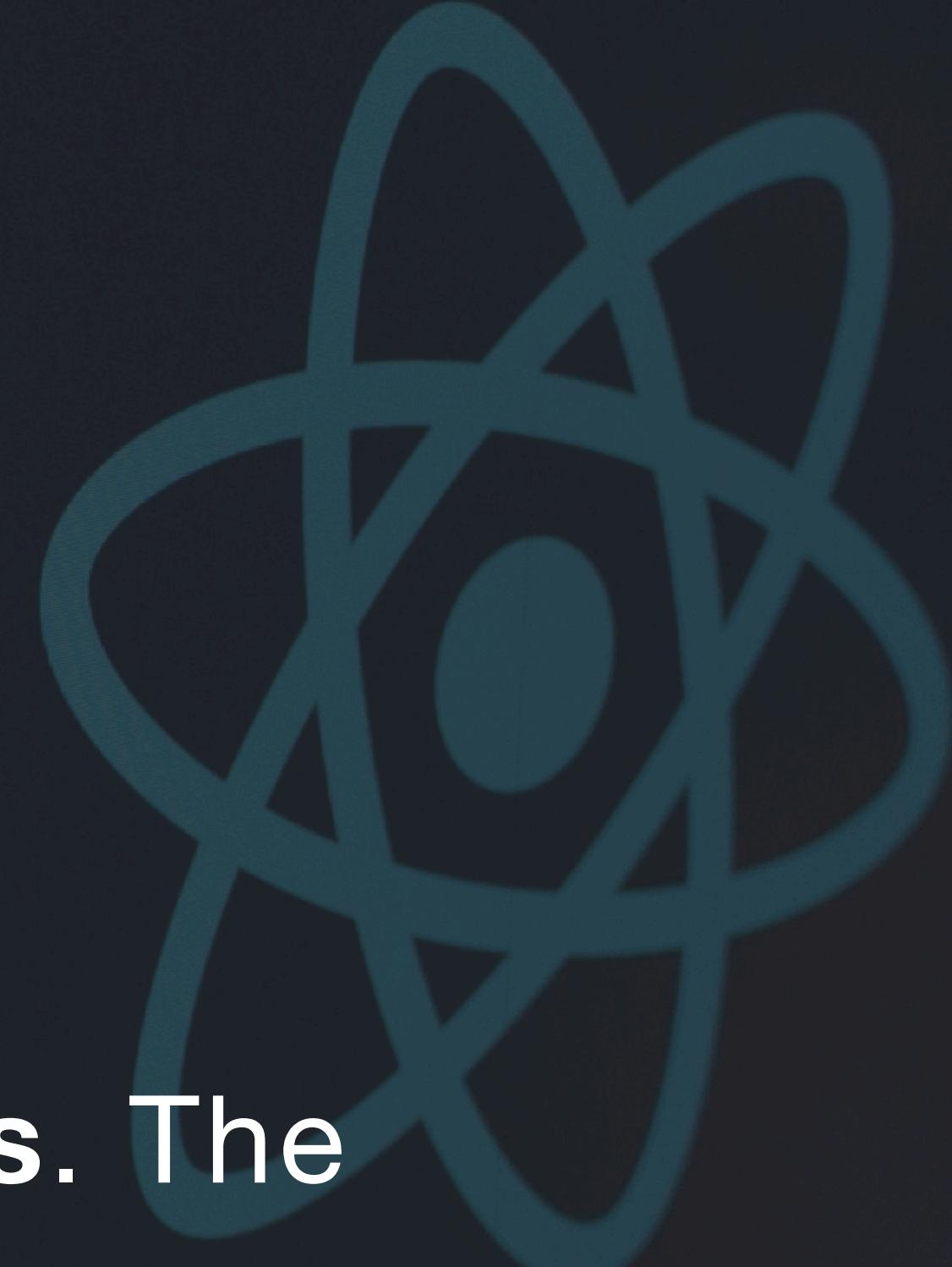
We are rewriting the React documentation with a few differences:

- All explanations are **written using Hooks** rather than classes.
- We've added **interactive examples** and visual diagrams.
- Guides include **challenges (with solutions!)** to check your understanding.



Thinking in React

In React, UI is a **function** of **props & states**. The UI is built with (function) **Components**.



Edit src/App.js and save to reload
Learn React

JS Nice to know for React

RACE: From Vanilla JavaScript to React Developer

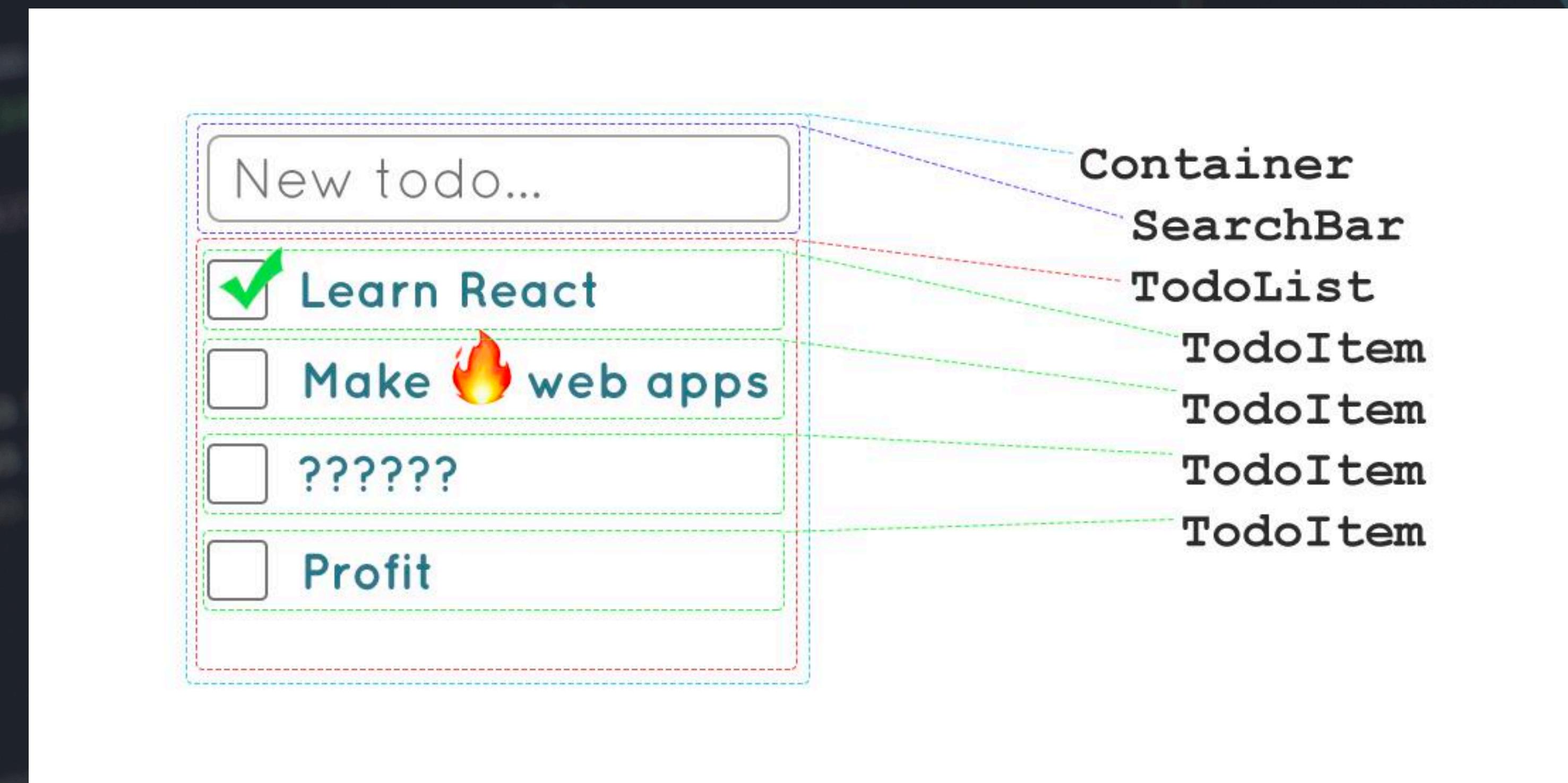
Components, Props & States

The core concepts of React

Edit src/App.js and save to reload
Learn React



How to Think in React



App.js and save to reload
Learn React

Function Component

A Component is simply just a function that returns HTML (JSX).

Components are independent blocks of code.

Components MUST return a single element. We can wrap in a parent element or use <></>.

```
function MyComponent() {  
  const name = "RACE";  
  
  return (  
    <section>  
      <h1>Hello, {name}</h1>  
    </section>  
  );  
}
```

JSX

JavaScript XML

- Allow us to write HTML in JavaScript.
- Makes it easier to write and add HTML in React.
- We can create our own tags.
- We MUST close all tags.

// With JSX

```
const myelement = <h1>I Love JSX!</h1>;
```

```
ReactDOM.render(myelement, document.getElementById("root"));
```

// Without JSX

```
const myelement = React.createElement("h1", {}, "I do not use JSX!");
```

```
ReactDOM.render(myelement, document.getElementById("root"));
```

Function Component

```
Regular function →  
function MyComponent() {  
  const name = "React"  
  return ( ← Return JSX  
    <div>  
      <h1>Hello, {name}</h1>  
    </div>  
  )  
}  
Starts with a capital letter  
Anything inside  
curly braces  
will execute as  
JavaScript  
Use any HTML tag  
to build component
```

Function Component

```
function Greeting() {  
  return (  
    <div>  
      | <h1>Hello, React!</h1>  
      | "Greeting"  
    </div>  
  );  
}
```

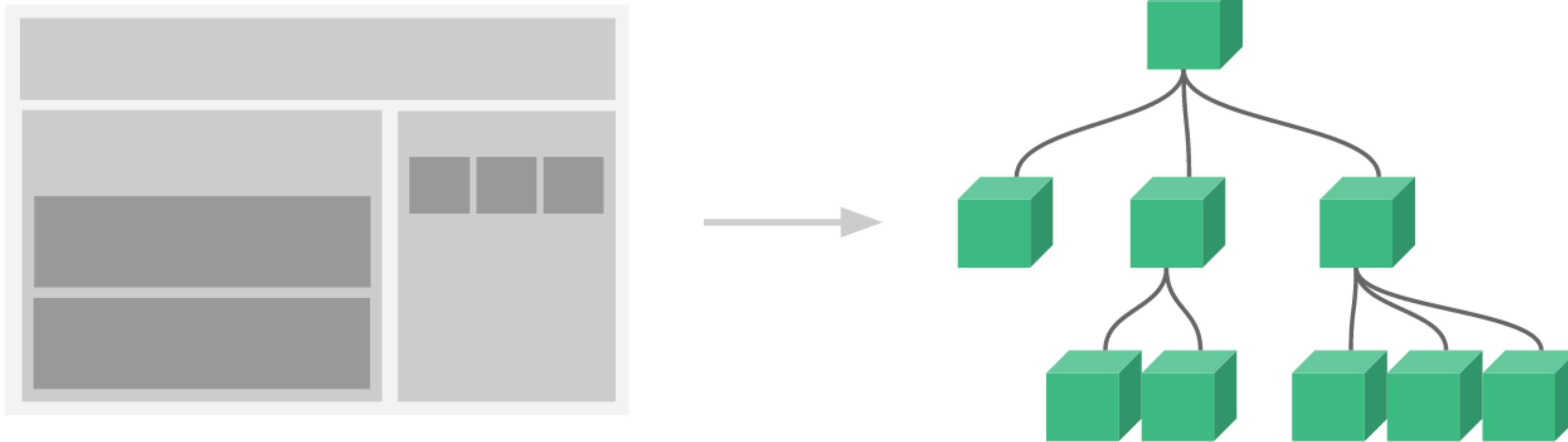
Define the component

```
function App() {  
  return (  
    <div>  
      | <Greeting />  
      | <div>  
      |   | <Greeting />  
      |   | </div>  
      | </div>  
  );  
}
```

Use Greeting component
in another component

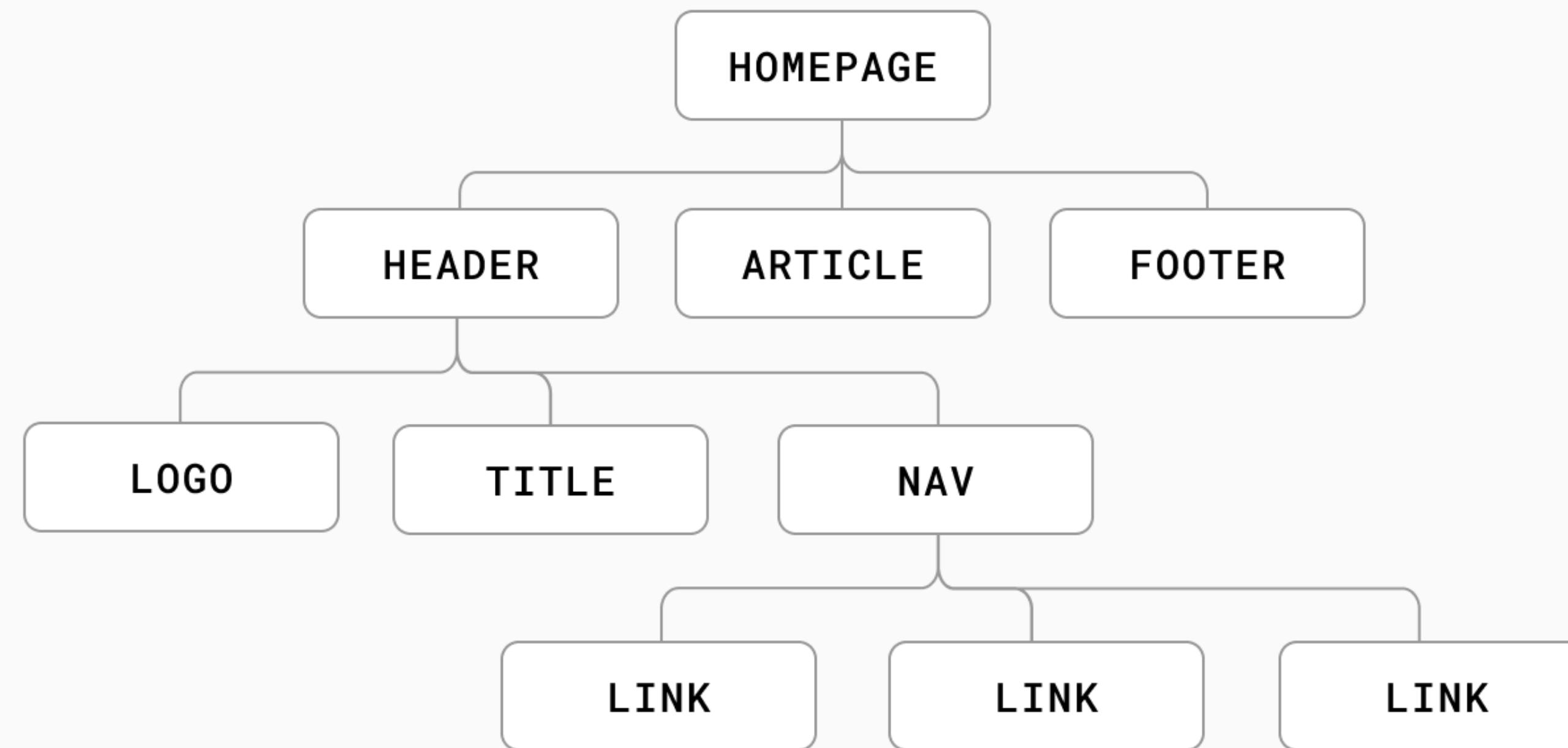
Components without
a closing tag
REQUIRE a closing
slash

Components, Components, Components



Build Reusable Components

UI can be broken down to small block or modules called components



It's all Components!

```
function Navigation() {
  return (
    <nav>
      <a href="#">Home</a>
    </nav>
  );
}

function Header() {
  return (
    <header>
      <h1>React Page Template</h1>
    </header>
  );
}

function PageContent() {
  return (
    <section>
      <h2>Here goes some page content</h2>
    </section>
  );
}

function Footer() {
  return (
    <footer>
      <p>This is my Footer</p>
    </footer>
  );
}

function App() {
  return (
    <>
      <Navigation />
      <Header />
      <PageContent />
      <Footer />
    </>
  );
}

ReactDOM.render(<App />, document.getElementById("root"));


```

react-cdn-page-template

Components?

<https://cederdorff.github.io/react-cdn-starters/react-cdn-firebase-post-app-auth/>

React CDN Template

POSTS CREATE PROFILE

Maria Louise Bendixen
Senior Lecturer
dolor sint quo a velit explicabo
quia namen
eos qui et ipsum ipsam suscipit aut sed omnis
non odio expedita earum mollitia molestiae aut
atque rem suscipit nam impedit esse

Jes Arbov
Lecturer
dolorem eum magni eos aperiam
quia
ut aspernatur corporis harum nihil quis provident
sequi mollitia nobis aliquid molestiae perspiciatis
et ea nemo ab reprehenderit accusantium quas
voluptate dolores velit et doloremque molestiae

Birgitte Kirk Iversen
Senior Lecturer
magnam facilis autem
dolore placeat quibusdam ea quo vitae magni
quis enim qui quis quo nemo aut saepe quidem
repellat excepturi ut quia sunt ut sequi eos ea sed
quas

Kim Elkjær Marcher-Jepsen
Senior Lecturer
Dan Okkels Brendstrup
Lecturer
Morten Algy Bonderup
Senior Lecturer

SINGLE PAGE WEB APP TEMP

ALL USERS

Order by: Choose here Filter by: All Search

Diana Riis Myjak Andersen
eaadimy@students.eaaa.dk
Enrollment: StudentEnrollment
[UPDATE](#) [DELETE](#)

German Arias Rodriguez
eaagar@students.eaaa.dk
Enrollment: StudentEnrollment
[UPDATE](#) [DELETE](#)

Haya Barakat
eaahbar@students.eaaa.dk
Enrollment: StudentEnrollment
[UPDATE](#) [DELETE](#)

USERS CREATE RANDOM

react-cdn-firebase-post-app-auth

Template: spa-canvas-users-template

Solution: spa-canvas-users

And?

The screenshot shows the homepage of DR Nyheder. At the top, there are links for NYHEDER, DRTV, and DR LYD. Below this, there are several thumbnail images of TV shows: DR1: Løvens Hule, DR3: Nationens stærkste, P1: LSD kælderen, DR LYD: Annas Margrethe, DR3: Du fucker med de forkerte, and A Very British Scandal. A red button labeled "Seneste nyt" is visible. Below the thumbnails, there are three news snippets: "EU klager over Kinas hårde kurs over for Litauen" (5 MIN. SIDEN), "Børn og skoleelever opfordres stadig til ugentlige coronatest" (13 MIN. SIDEN), and "England skrætter størstedelen af coronarestriktionerne fra i dag" (25 MIN. SIDEN). The main content area features a large graphic of medical supplies, including a mask, a thermometer, and a hand sanitizer bottle, set against a blue background. A headline below the graphic reads "15 lande bakker Danmark op: Danske soldater skal blive i Mali". At the bottom, a red banner says "PENGE" and "Regeringen har meldt genåbning - men ikke".

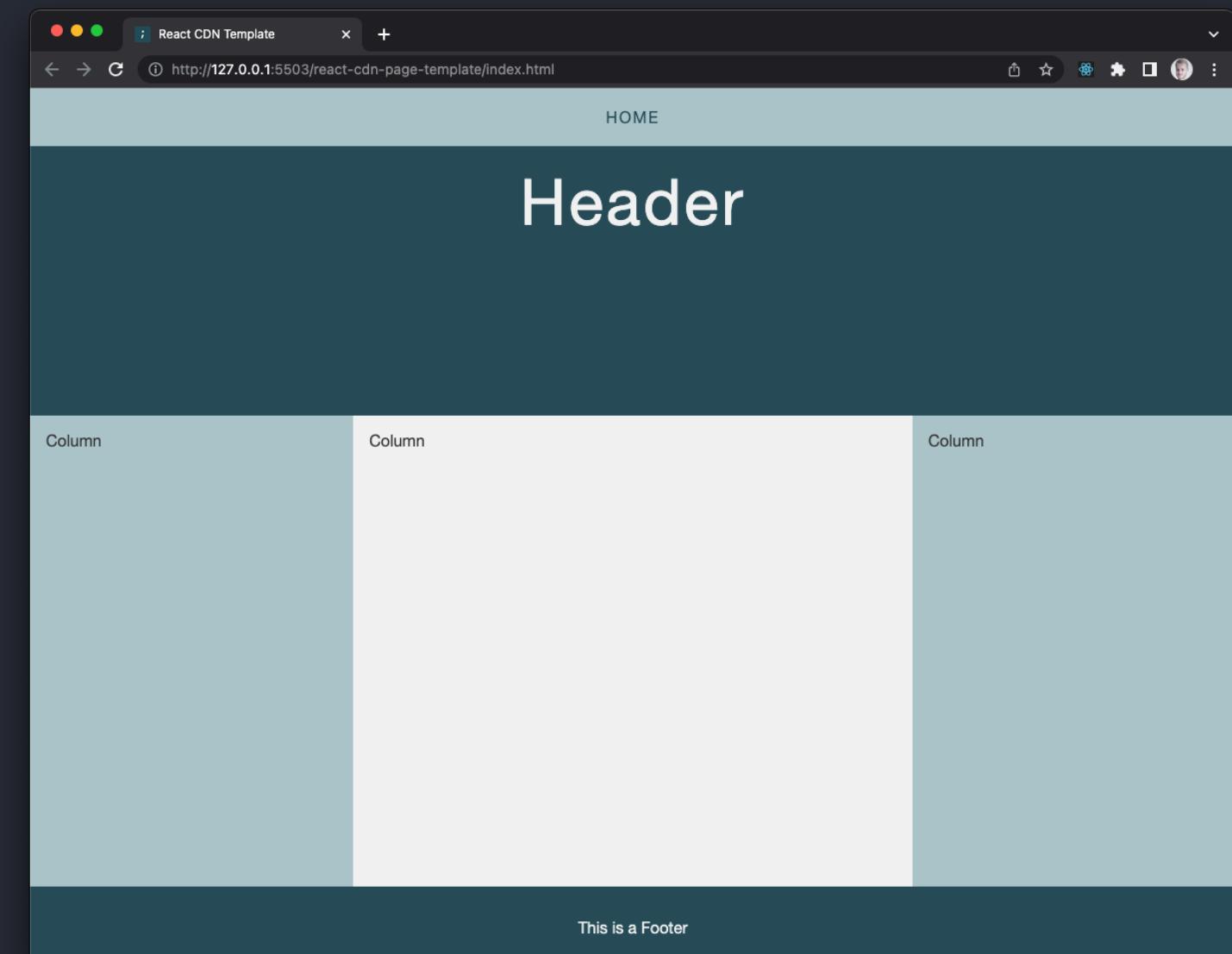
The screenshot shows the website for Erhvervsakademiet Aarhus. The page displays a grid of vocational training programs. Each program includes a thumbnail image, a title, and a brief description. The programs listed are:

- Byggekoordinator (lukket)**: 2-årig erhvervsakademiuddannelse. Uddannelsen optager ikke nye studerende fra og med 2022.
- Datamatiker**: 2-årig erhvervsakademiuddannelse. Få en bred viden inden for systemudvikling og programmering, og bliv kvalificeret på et job med udvikling, implementering og drift af IT-systemer i virksomheden.
- Financial controller**: 2-årig erhvervsakademiuddannelse. Bliv klar til et job i en økonomialafdeling eller som revisor. Gør karriere inden for fx regnskab, likviditetsstyring, årsrapporter, skat, moms og husholdninger.
- Finansøkonom**: 2-årig erhvervsakademiuddannelse. Bliv rustet til en karriere i investerings-, forsikrings- eller ejendomsbranchen – eller i kreditinstitutter. Du bliver en god rådgiver, der løser kundebehov med din virksomhedens konkrete problemer.
- It-teknolog**: 2-årig erhvervsakademiuddannelse. Brænder du for at følge med i udviklingen af den nyeste teknologi? Så gør karriere inden for computer-, server- og netværksteknologi eller elektronik.
- Jordbrugsteknolog**: 2-årig erhvervsakademiuddannelse. Bliv specialist inden for miljø og natur, jordbruksøkonomi og driftsledelse, husdyrproduktion, landskab og anlæg og planteproduktion.
- Laborant**: 2½-årig erhvervsakademiuddannelse. Få job i et kontrol-, forsknings- eller udviklingslaboratorium og arbejd inden for fx fødevare sikkerhed, miljøbeskyttelse, medicinalindustrien eller bioteknologi.
- Markedsføringsøkonom**: 2-årig erhvervsakademiuddannelse. For dig, der vil arbejde med markedsføring, salg og kommunikation. Gør karriere som fx indkøber, sælger, marketingkoordinator eller projektleder. Vi tilbyder specialiseringer inden for helseøkonomi, design
- Miljøteknolog**: 2-årig erhvervsakademiuddannelse. For dig, der vil arbejde med kvalitetssikring og forbedring af virksomhedens miljøindsats. Som miljøteknolog sikrer du, at virksomheden er bæredygtigt mindsker forurenningen og overholder
- Multimediedesigner**: 2-årig erhvervsakademiuddannelse. Få viden om user interface design (UI), user experience design (UX), programmering og formændsforståelse. Bliv klar til job som fx frontendenudvikler, UX-designer, content manager eller

... but we thought it was all objects and array?

Component Page Layout

1. Use [react-cdn-template](#)
2. Implement a page layout and create the following components: Navigation, Header, MainContent & Footer
3. Add some placeholder content in every component like headlines and/or text.
MainContent should consist of 3 columns
(use the CSS classes left, middle & right).
4. Make sure to render all components in the App component.
5. Style your page layout using a grid, flexbox or use the existing CSS styling from app.css
(.page-layout, nav, header, footer, left, middle & right)



```
function App() {  
  return (  
    <main className="page-layout">  
      <Navigation />  
      <Header />  
      <MainContent />  
      <Footer />  
    </main>  
  );  
}  
  
ReactDOM.render(<App />, document.getElementById("root"));
```

Props

... arguments passed into a React Component. Props are passed to components via HTML attributes.

```
function Greeting({ name }) {  
  return <h1>Hello, {name}</h1>;  
}  
  
function App() {  
  return <Greeting name="world" />;  
}
```

Function Component with Props

```
function Greeting(props) {  
  return (  
    <div>  
      | <h1>Hello, {props.name}!</h1>  
    </div>  
  );  
}
```

props is first argument to function components

Access props inside of curly braces to show value

```
function App() {  
  return (  
    <div>  
      | <Greeting name="React" />  
      <div>  
        | <Greeting name="Chris" />  
      </div>  
    </div>  
  );  
}
```

Set a prop on a child component by passing an attribute

Different instances of component can have different prop values

Add Props

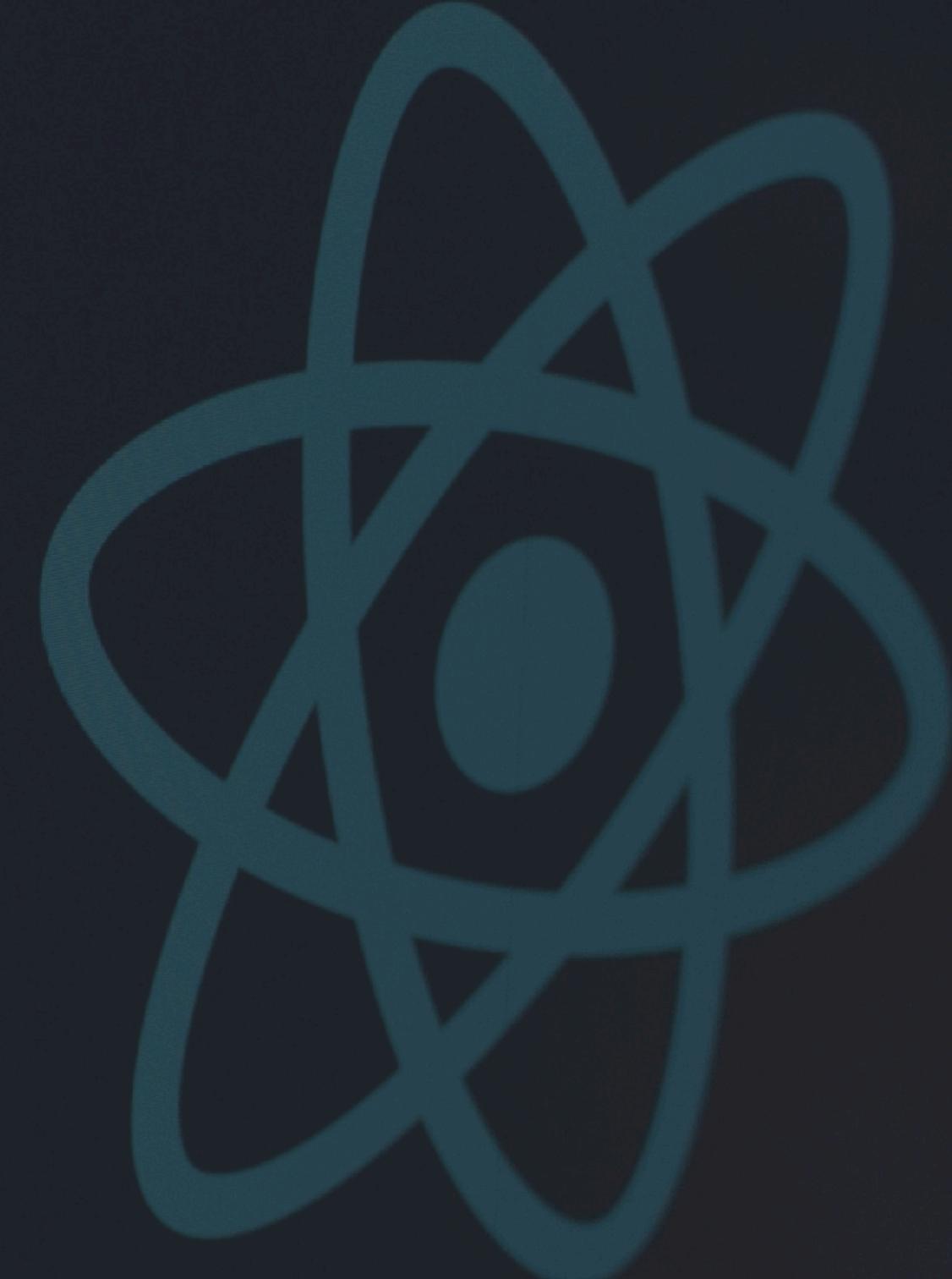
1. Use your implemented Page Layout or `react-cdn-page-template`.
2. Add a prop to your `Header` component called `title`. Make sure the component (function) takes in an argument. The `title` prop must be rendered in the JSX inside the `h1`.
3. Locate the `Header` component in `App` and add the attribute `title` and pass in a title ("My Title", "Your Name" etc.).
4. Test your solution and try to change the `title` attribute value.
5. Implement and try out props for some of the other components in your Page Layout.

```
function Greeting({ name }) {  
  return <h1>Hello, {name}</h1>;  
}  
  
function App() {  
  return <Greeting name="world" />;  
}
```

Hooks

useState & useEffect

Edit src/App.js and save to reload
Learn React



useState()

<https://www.youtube.com/watch?v=TNhalSOUy6Q>

useState

... a hook that allows us to track states in a function component.

The purpose is to handle reactive data. When data in a state changes, React reacts and re-renders the UI.

useState(...)

... a hook that allows us to track states in a function component.

The purpose is to handle reactive data. When data in a state change, React reacts and re-renders the UI.

```
import React, { useState } from "react";  
  
function Greeting(props) {  
  const [count, setCount] = useState(0)  
  const updateCount = () => {  
    setCount(count + 1)  
  }  
  
  return (  
    <div>  
      <h1>Hello, {props.name}!</h1>  
      <p>You've clicked {count} times</p>  
      <button onClick={updateCount}>  
        Click me  
      </button>  
    </div>  
  );  
}
```

Import useState from React

Call useState and pass in a default value

useState returns the current value and an update function

Display state in curly braces

When States & Props
change, your components
automatically is updated!

React *reacts* to the
changes.

```
import React, { useState } from "react";

function Greeting(props) {
  const [count, setCount] = useState(0)

  const updateCount = () => {
    setCount(count + 1)
  }

  return (
    <div>
      <h1>Hello, {props.name}!</h1>
      <p>You've clicked {count} times</p>
      <button onClick={updateCount}>
        Click me
      </button>
    </div>
  );
}


```

Call the update function
with a new value to set the state.
NEVER set the value directly

count will update
AUTOMATICALLY!

Use curly braces
to set attribute
to JS value

Set onClick attribute of
a button to a custom function

Tryout useState

1. In your previous React project, create a component called Counter.
2. The Counter component should have a similar structure as the component to the right, with a count state, updateCount function, displaying the count state and a button calling updateCount.
3. Render the Counter component in your PageComponent.
4. Test and think of the pros of using React and states compared to Vanilla JS.
5. Try to render the Counter in another component.

```
import React, { useState } from "react";  
  
function Greeting(props) {  
  const [count, setCount] = useState(0)  
  const updateCount = () => {  
    setCount(count + 1)  
  }  
  
  return (  
    <div>  
      <h1>Hello, {props.name}!</h1>  
      <p>You've clicked {count} times</p>  
      <button onClick={updateCount}>  
        Click me  
      </button>  
    </div>  
  );  
}
```

Import useState from React

Call useState and pass in a default value

useState returns the current value and an update function

Display state in curly braces

List, loop and map

```
const DATA = [
  { id: 4, title: 'A New Hope' },
  { id: 5, title: 'The Empire Strikes Back' }
  { id: 6, title: 'Return of the Jedi' }
]
```

```
const App = () => <MyList items={DATA} />
```

```
function MyList(props) {  
  return (  
    <div>  
      {  
        props.items.map(item => {  
          return <p key={item.id}>{item.title}</p>  
        })  
      }  
    </div>  
)  
}  
)  
  ) Return JSX from  
  each iteration  
}  
}  
  ) Map over data that was  
  passed in as props  
  ) Use curly braces like  
  normal to display data  
  ) Pass unique  
  key attribute  
  to each element  
  ) Wrap entire JS in  
  curly braces
```

useEffect()

<https://www.youtube.com/watch?v=TNhalSOUy6Q>

useEffect(...)

... a hook that allows us to do side effects in a component, like fetching data, directly updating the DOM, and timers.

```
import React, { useState, useEffect } from "react";
```

```
function GetData(props) {
  const [data, setData] = useState({});

  useEffect(() => {
    fetch("https://swapi.co/api/people/" + props.id)
      .then(response => response.json())
      .then(result => setData(result));
  }, [props.id]);

  return (
    <div>
      <p>Name: {data && data.name}</p>
    </div>
  );
}
```

Import useEffect from React

Call useEffect with a function as the first argument

The second argument is an array of dependencies. Don't forget this!

Do async actions like data fetching inside the callback

useEffect(...)

useEffect accepts two arguments.

We should always include the second parameter which accepts an array of dependencies.

useEffect runs on every render and when the dependencies change.

```
useEffect(() => {  
  |   //Runs on every render  
});
```

```
useEffect(() => {  
  |   //Runs only on the first render  
}, []);
```

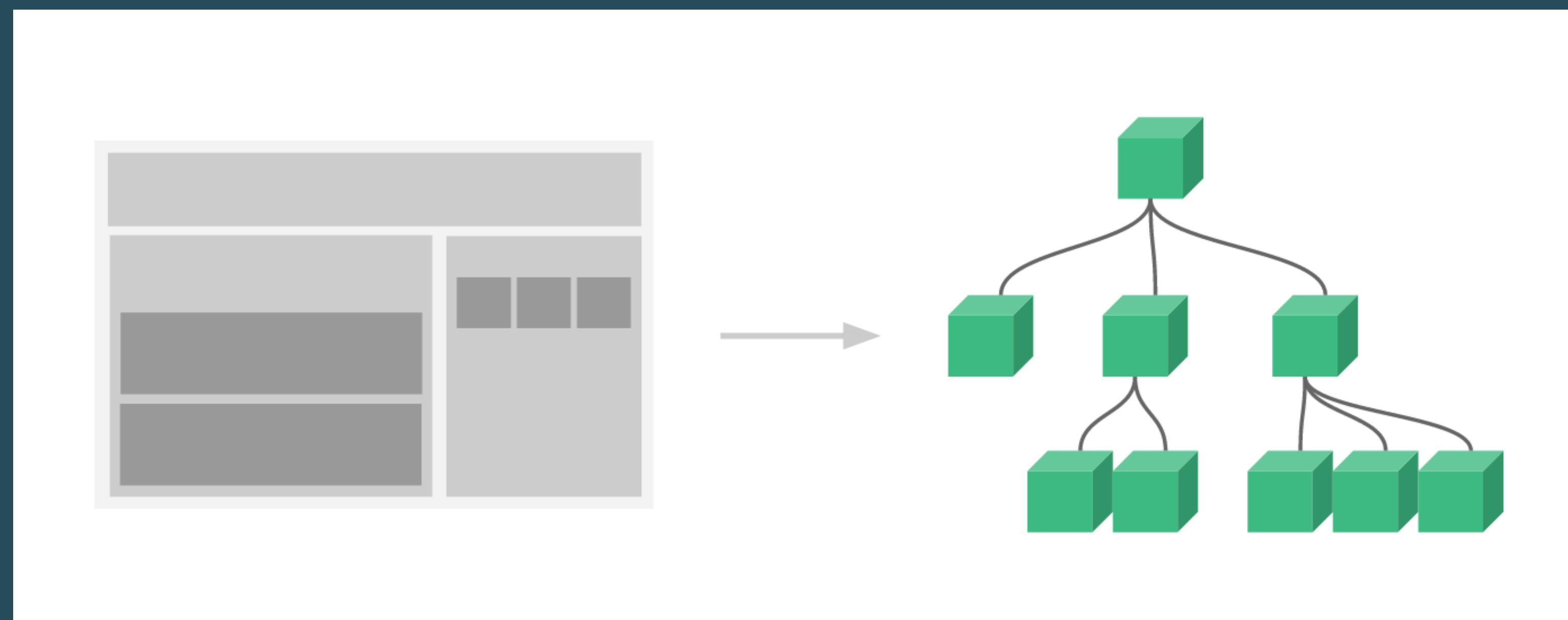
```
useEffect(() => {  
  |   //Runs on the first render  
  |   //And any time any dependency value changes  
}, [prop, state]);
```

1. Use `react-cdn-template` or `react-cdn-page-template`. The goal is to fetch and display a grid of posts using `useState` and `useEffect`. Similar to the [JS Fetch Posts](#). You can use the predefined grid styling in `app.css`
2. Create a component called `Posts` returning a section with an `h1` (with a title) and another section as your grid container. Render the `Post` component in the `App` and test it in the browser.
3. In `Posts` component, implement a state called `posts`. The state must be initialised with an empty array in `useState`.
4. Use `useEffect` to fetch post data from the URL to the right. Save the fetched posts in the state. Use `console.log(...)` to test your fetch call.
5. In the grid container, map through the `posts` state and create an `article` for every post (similar to the [JS Fetch Posts](#) exercise). Display the properties `image`, `title` and `body` in matching HTML tags inside of the `article` tag. Also add a `key` attribute on the `article` tag using the `post.id` value.



Build Reusable Components

UI can be broken down to small block or modules called components. We strive for independent (and small) components.



6. Split your Posts component into more components. Start by creating a new component called PostCard.
7. Extract the post article logic from the Posts component to the PostCard component (only the article not the .map part).
8. Make sure PostCard takes in a prop called post. Use the prop to render the post in PostCard.
9. Inside of your .map in your Posts component render a PostCard and pass in post as a prop. A PostCard component will be rendered for every post. Remember to add a key attribute on the PostCard. (and remove it from the article).



Style your component

Regular Stylesheet

```
import "./styles.css";

function NormalCSS() {
  return <p className="big-text">
    BIG!
  </p>;
}
```

Use `className` instead
of `class`, because
`class` is a restricted
word in javascript

Inline Styles

```
function InlineStyle() {
  return (
    <p
      style={{

        fontSize: 20,
        color: "#0000ff"
      }}
    >
      Blue Text
    </p>
  );
}
```

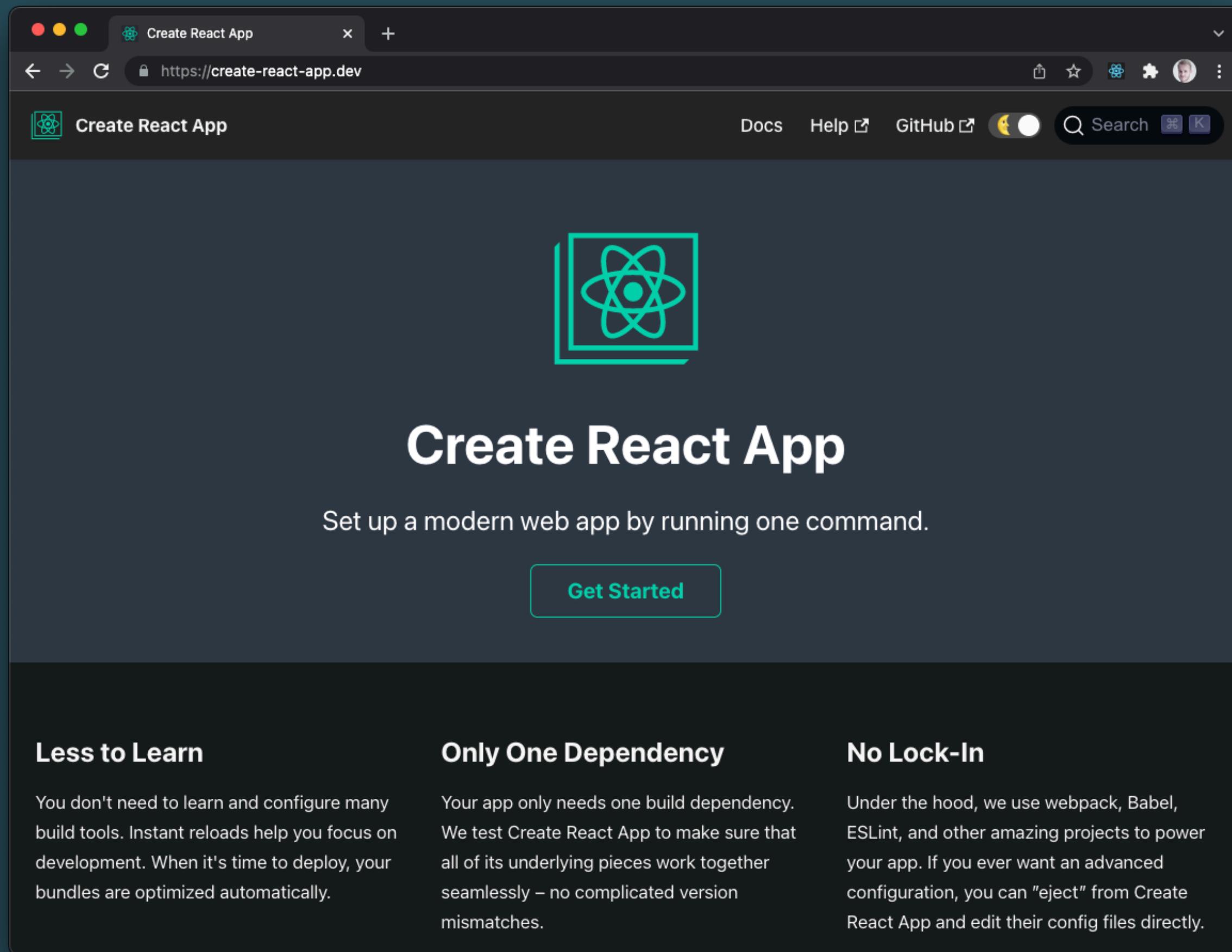
Double curly braces
because you're defining
a JS object inside of JSX

Camel case
attributes

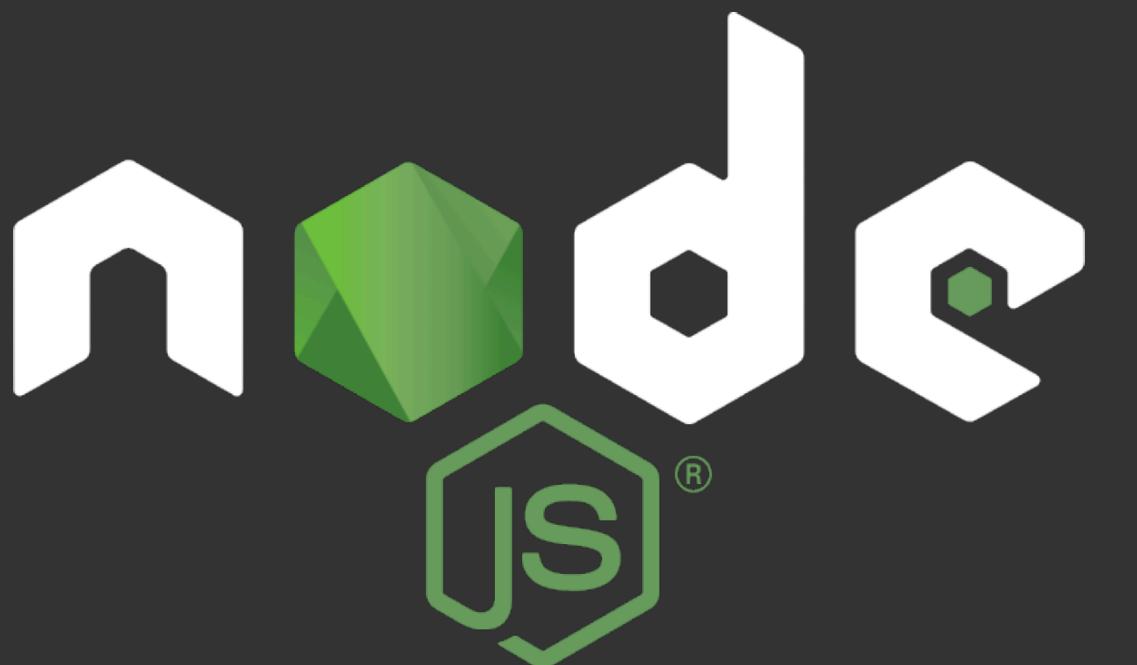
Values like colors
must be in strings

create-react-app

Starter template for your react app



Generate create-react-app



1. Download & install Node.js & npm: nodejs.org/en/
2. Check your version of npm & Node.js

```
race@macbookpro-9720 ~ % node -v
v16.13.1
race@macbookpro-9720 ~ % npm -v
8.6.0
race@macbookpro-9720 ~ %
```

EXPLORER

REACT-PROJECTS

- > react-carousel
- > react-firebase-hooks
- > react-firebase-favorite-
- > react-firebase-post-ap|
- > react-firebase-read-create
- > react-spa-template ●
- > react-user-crud ●

New Terminal ⌘T

Split Terminal ⌘⇧T

Run Task...

Run Build Task... ⌘B

Run Active File

Run Selected Text

Show Running Tasks...

Restart Running Task...

Terminate Task...

Configure Tasks...

Configure Default Build Task...

react-projects

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
race@macbookpro-9720 react-projects % node -v
v16.13.1
race@macbookpro-9720 react-projects % npm -v
8.6.0
race@macbookpro-9720 react-projects % █
```

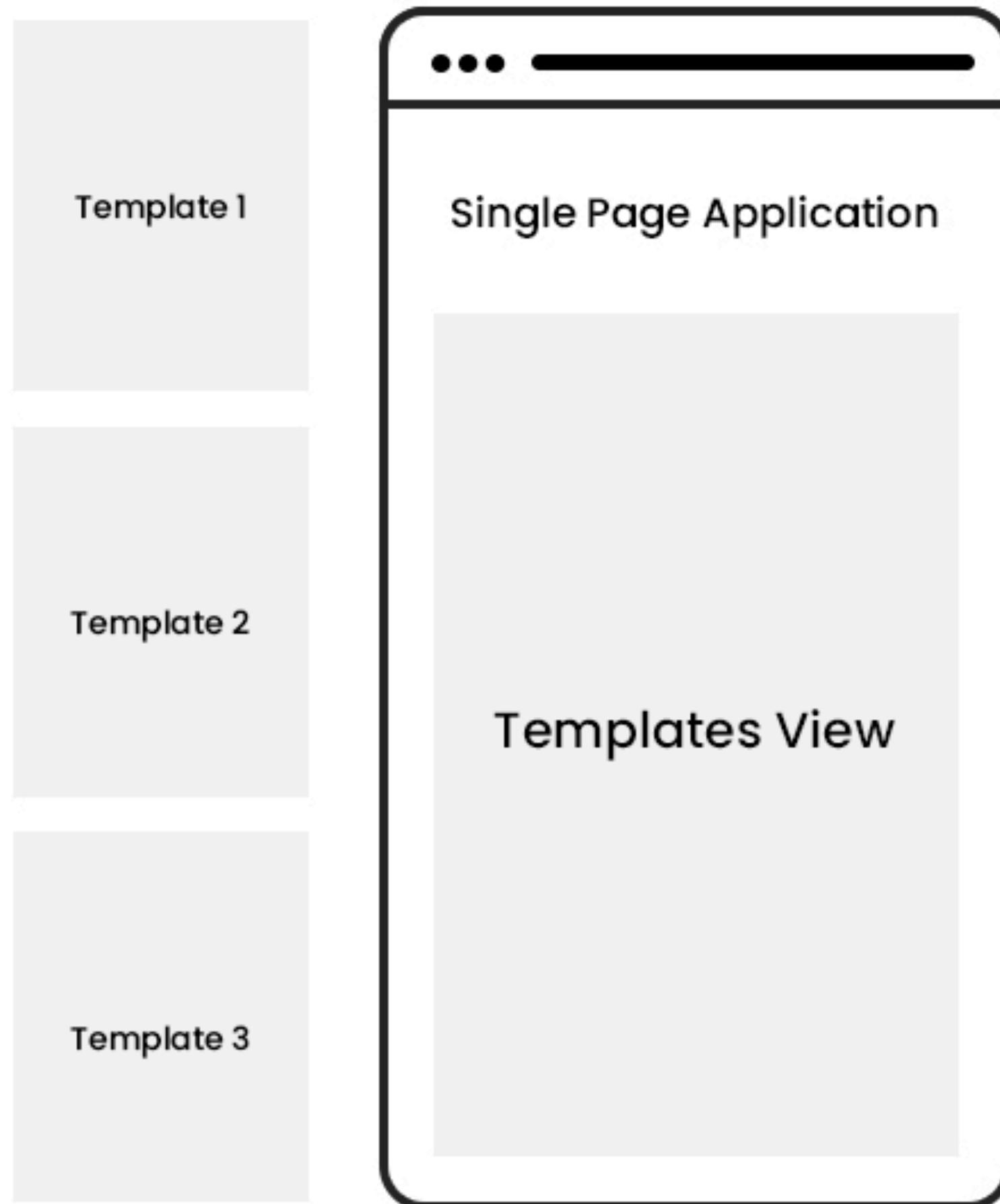
zsh + × □ └ ^ ×

Single Page Apps

“A single-page application is an application that loads a single HTML page and all the necessary assets (such as JavaScript and CSS) required for the application to run. Any interactions with the page or subsequent pages do not require a round trip to the server which means the page is not reloaded.”

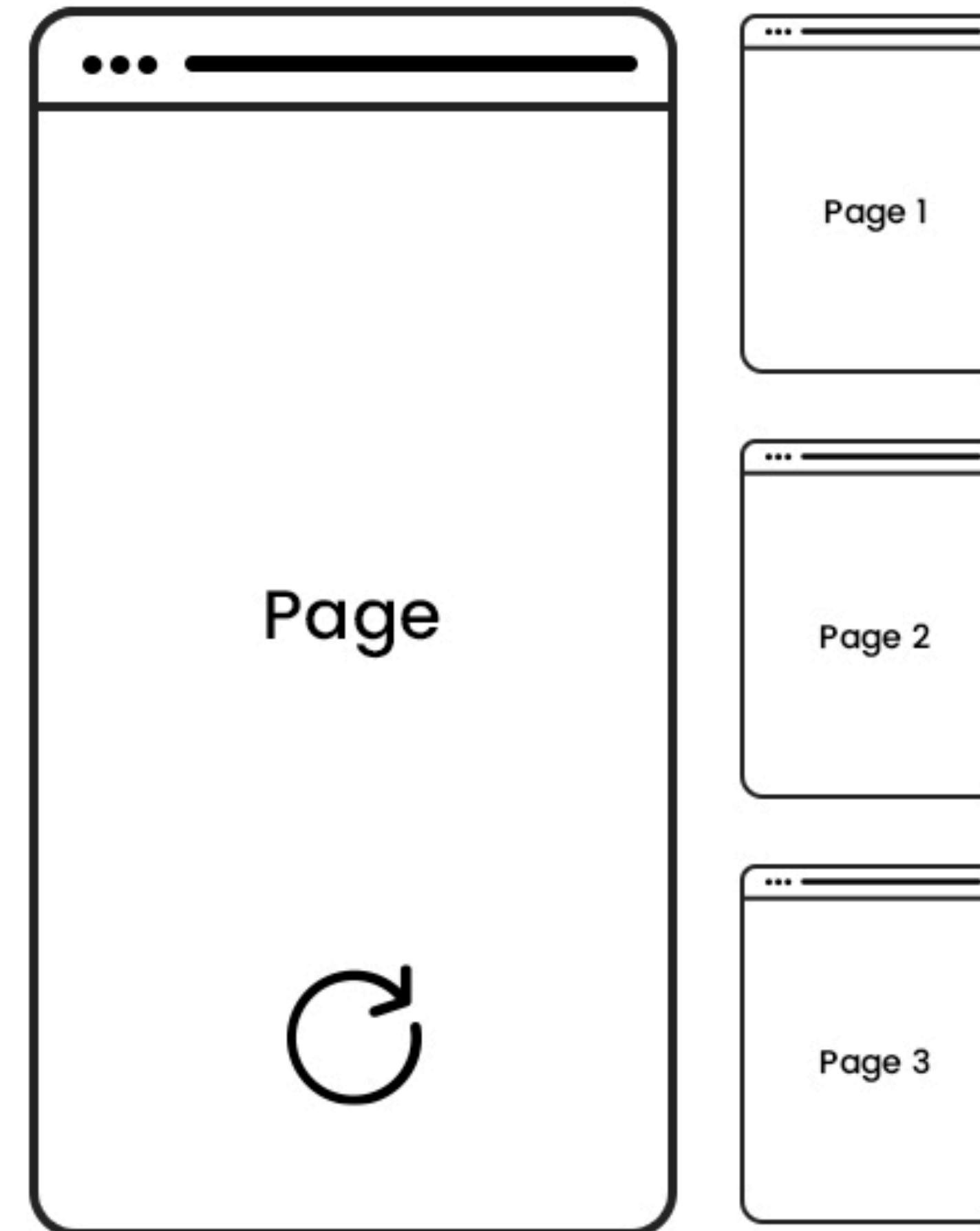
<https://reactjs.org/docs/glossary.html#single-page-application>

SPA



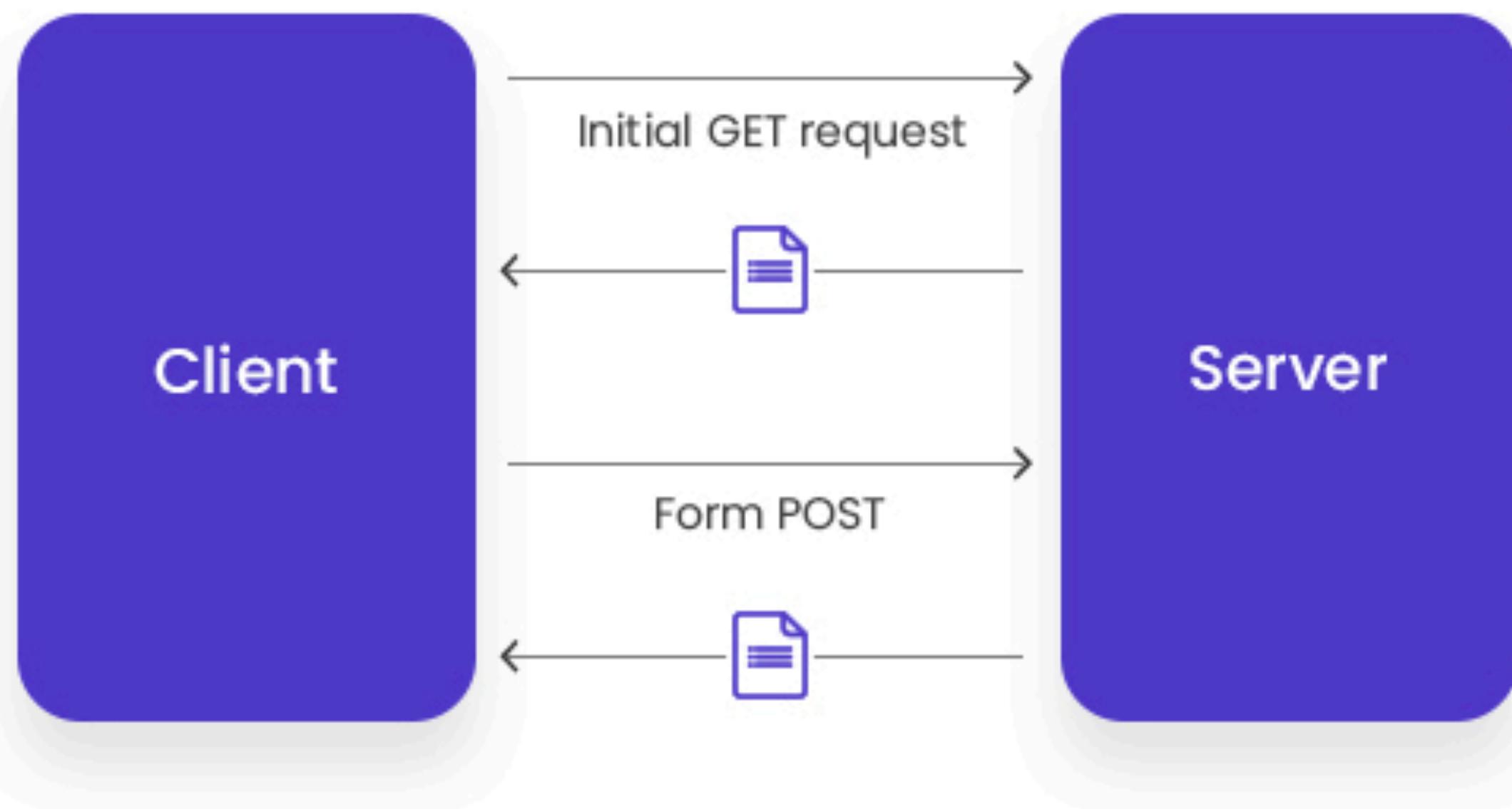
No page refresh on request

MPA

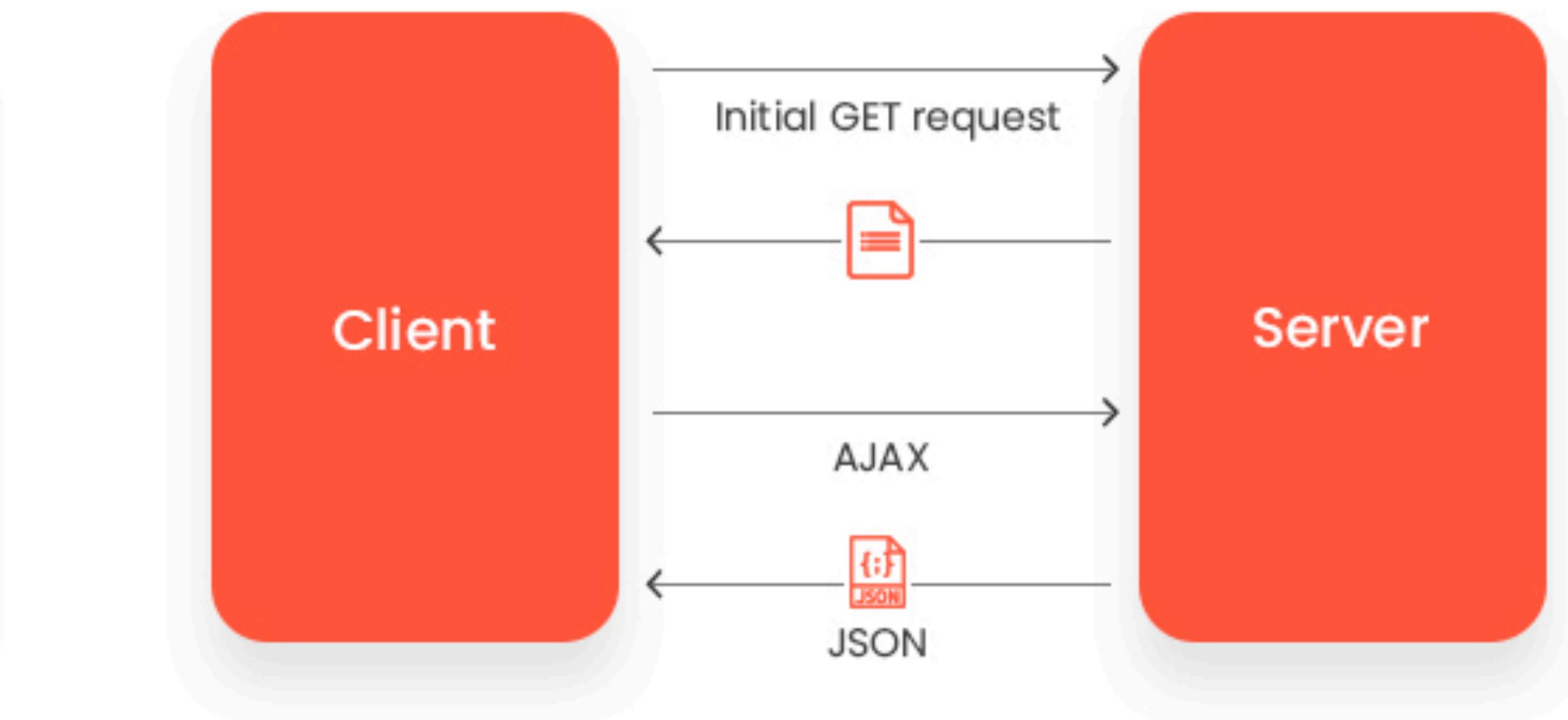


Whole page refresh on request

Traditional Page Lifecycle



SPA Lifecycle



React Router

... a client & server-side routing library for React

```
<Routes>
  <Route path="/" element={<HomePage />} />
  <Route path="about" element={<AboutPage />} />
  <Route path="*" element={<Navigate to="/" />} />
</Routes>
```

What's a Router?

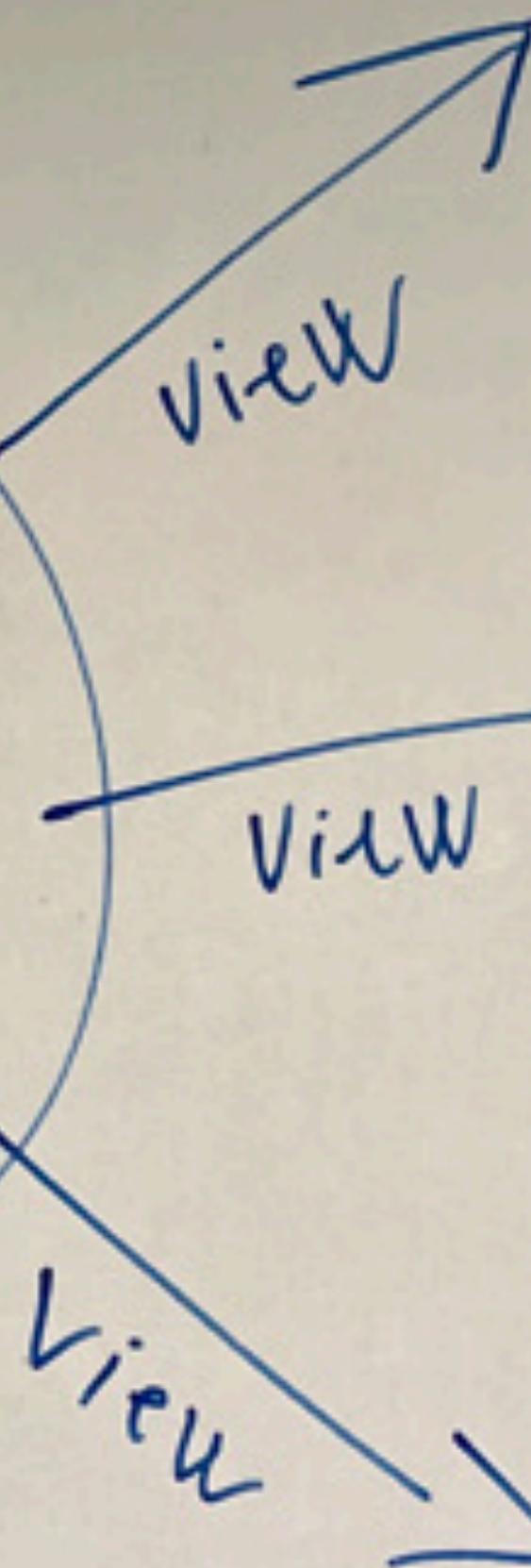
“It is the piece of software in charge to organize the states of the application, switching between different views.”

It's a key component in Single Page Applications.



/create
route

Router
(router.js)



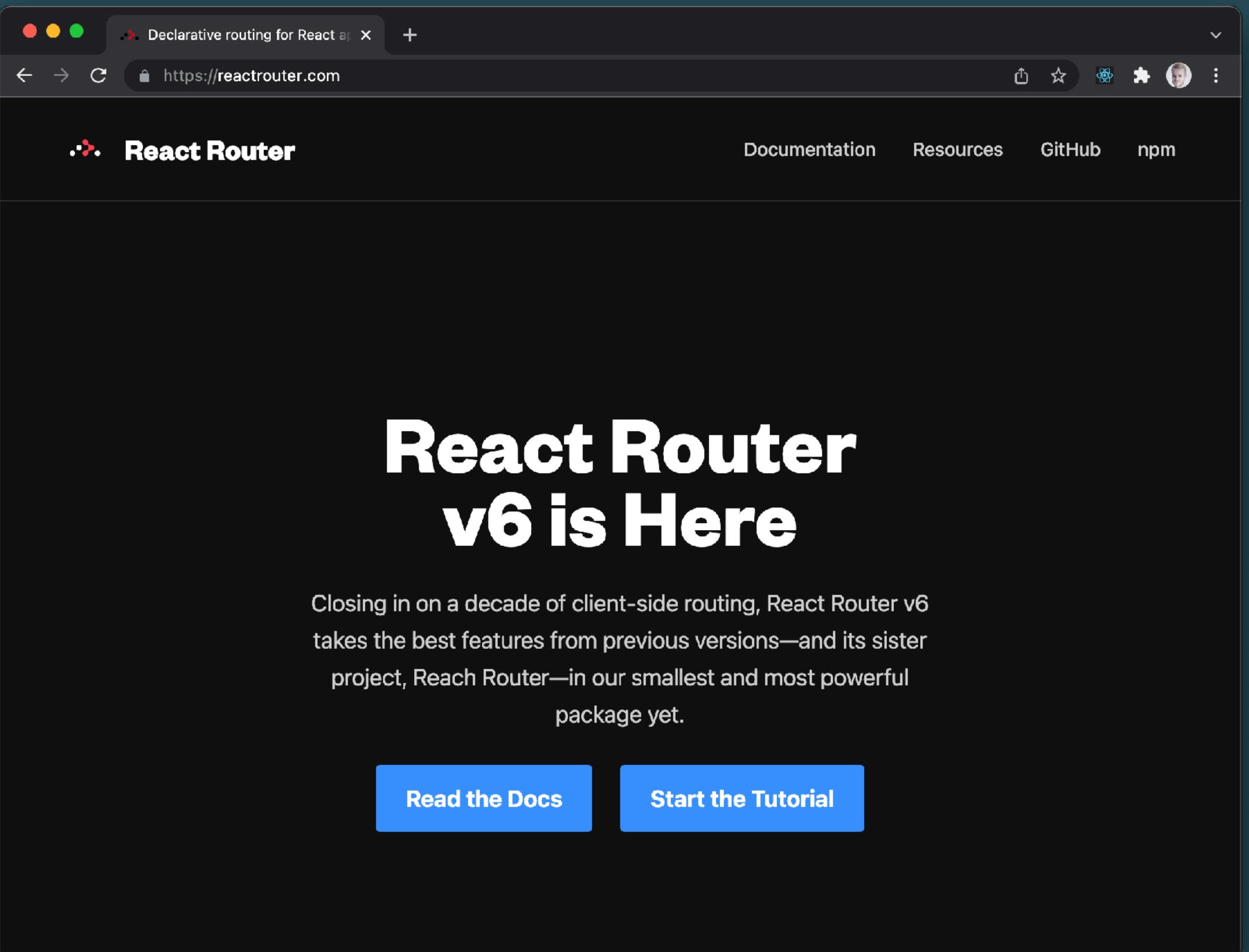
users
page

create
page

update
page

Create React SPA

With React Router



Create React SPA

React Practice

Practice your React knowledge with [create-react-app](#) and [react-router](#):

1. Build a Portfolio site template with pages (components) like Home, Projects, About and Contact.
2. Do a remake of Canvas Users SPA with React. You can find inspiration on GitHub: [react-starters](#) and [react-cdn-starters](#)
3. Checkout [React Self-study](#) and [Guides & Tutorials](#)

Code Every Day

Edit src/App.js and save to reload
[Learn React](#)

