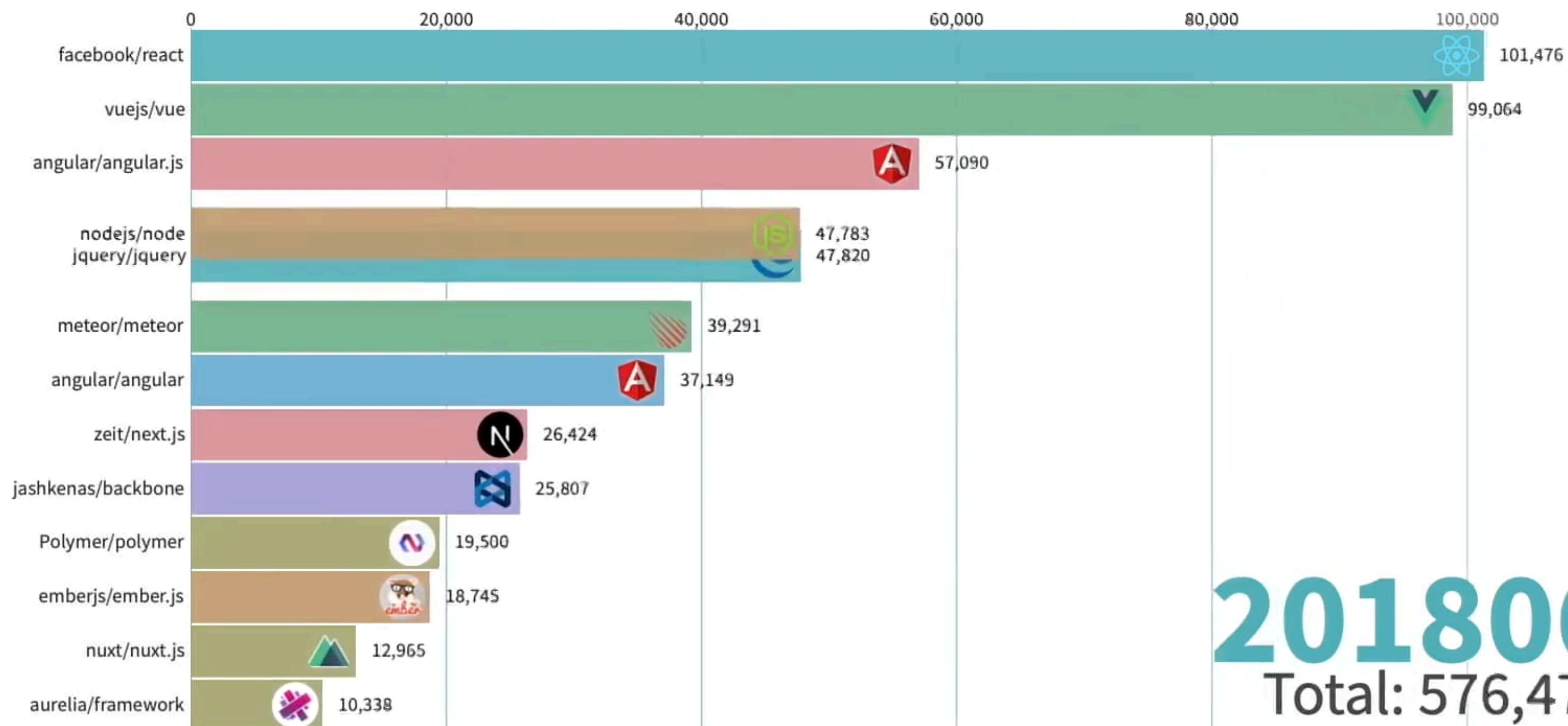


REMINDER

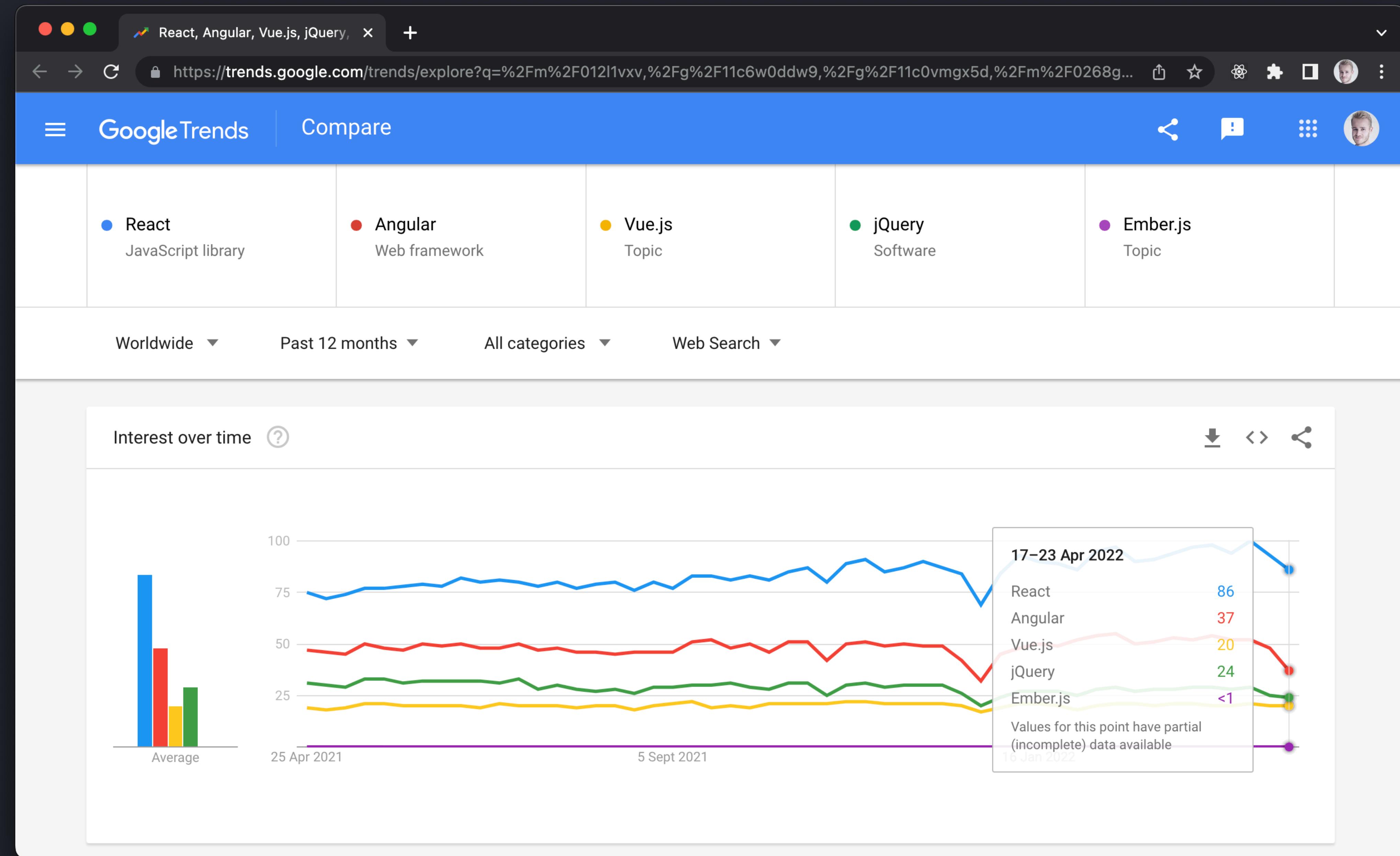


Most popular JavaScript frameworks

2012 - 2019



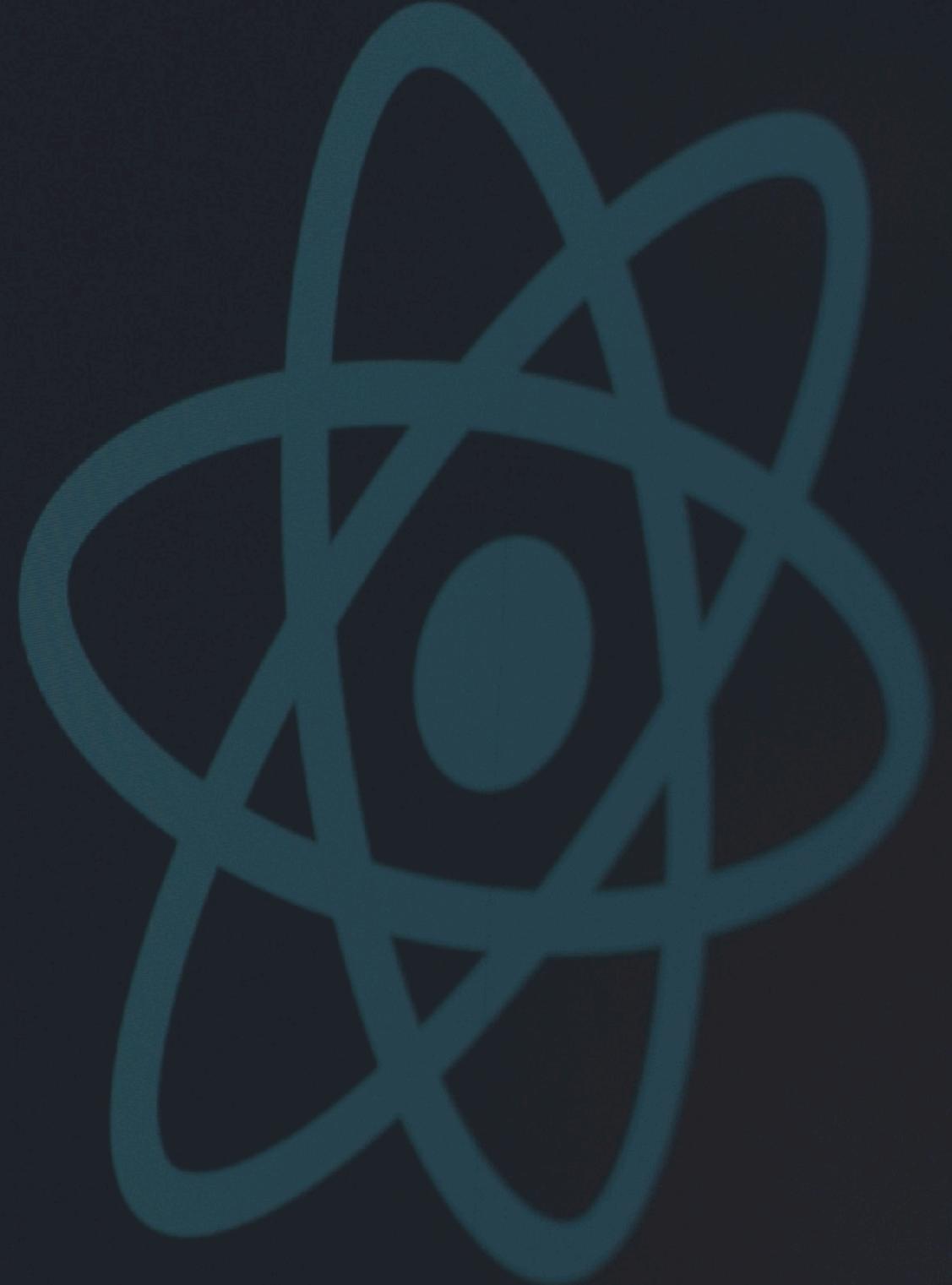
Source: GitHub Archive



React & Async JS

Frontend Programming

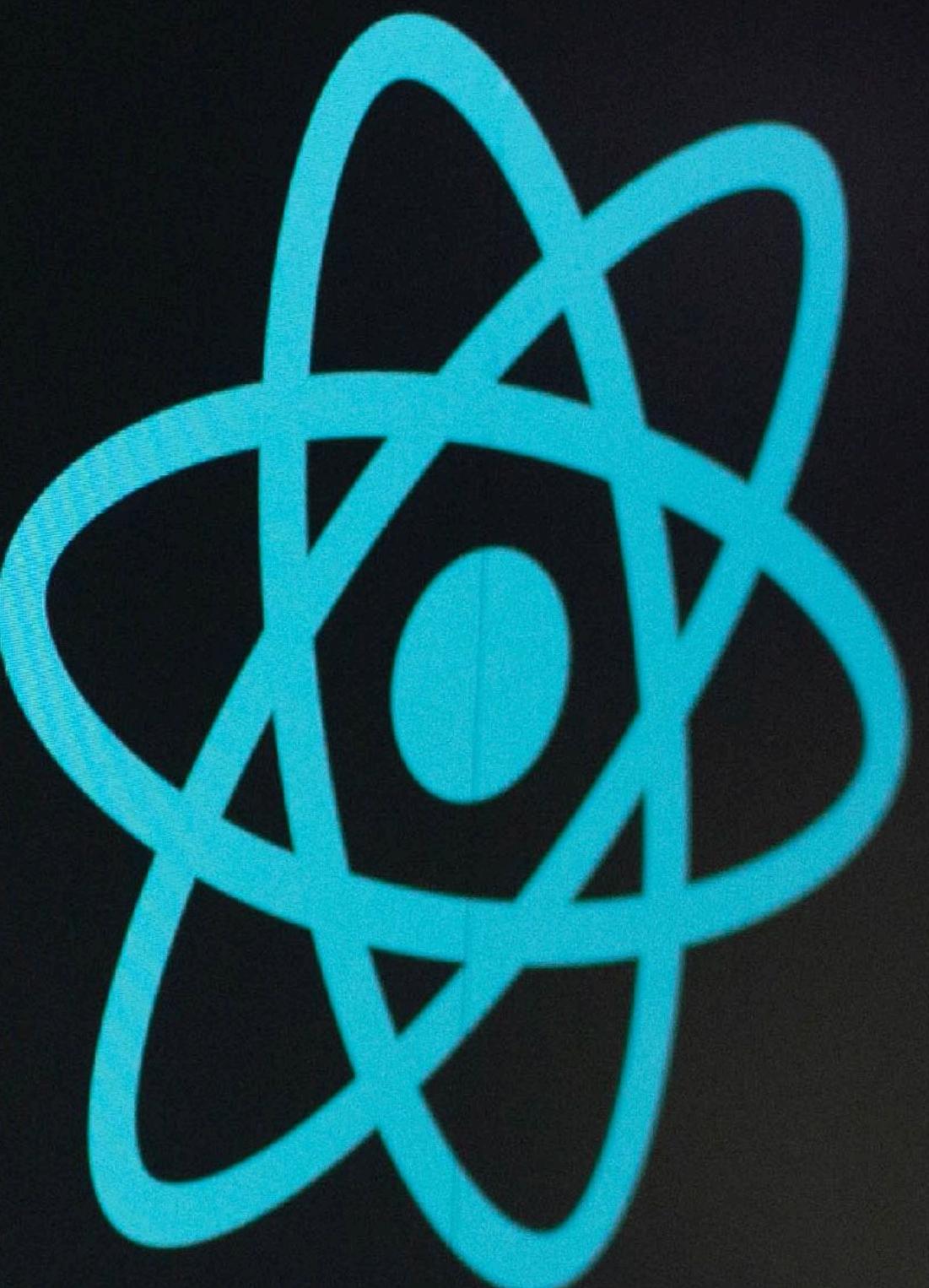
Edit src/App.js and save to reload
Learn React



Purpose

- Gain a deeper understanding of React Core Concepts.
- Improve your knowledge about fetch & Async JS.
- Single Page Apps & Routing

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



- React Core Concepts
- React Hooks
- Async JS in React
- CLI, Build Tools & create-react-app
- React Router & SPA

Agenda

- Tryout useState
- React Fetch Posts
- Download & install Node.js & npm
- Generate create-react-app
- Create React SPA



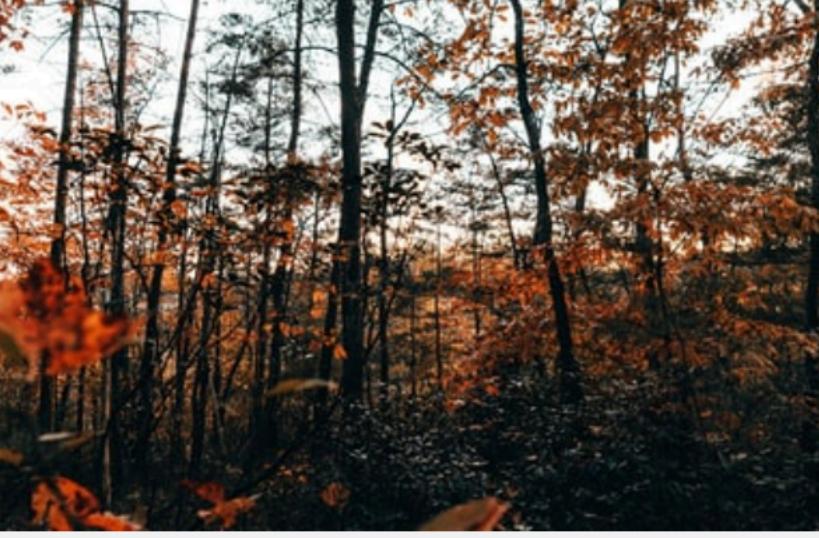
localhost:3000

React CDN Template

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

POSTS CREATE

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



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

Make sure you have a basic (JS) knowledge ...

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

Computer science student



Senior developer, 10+ years experience



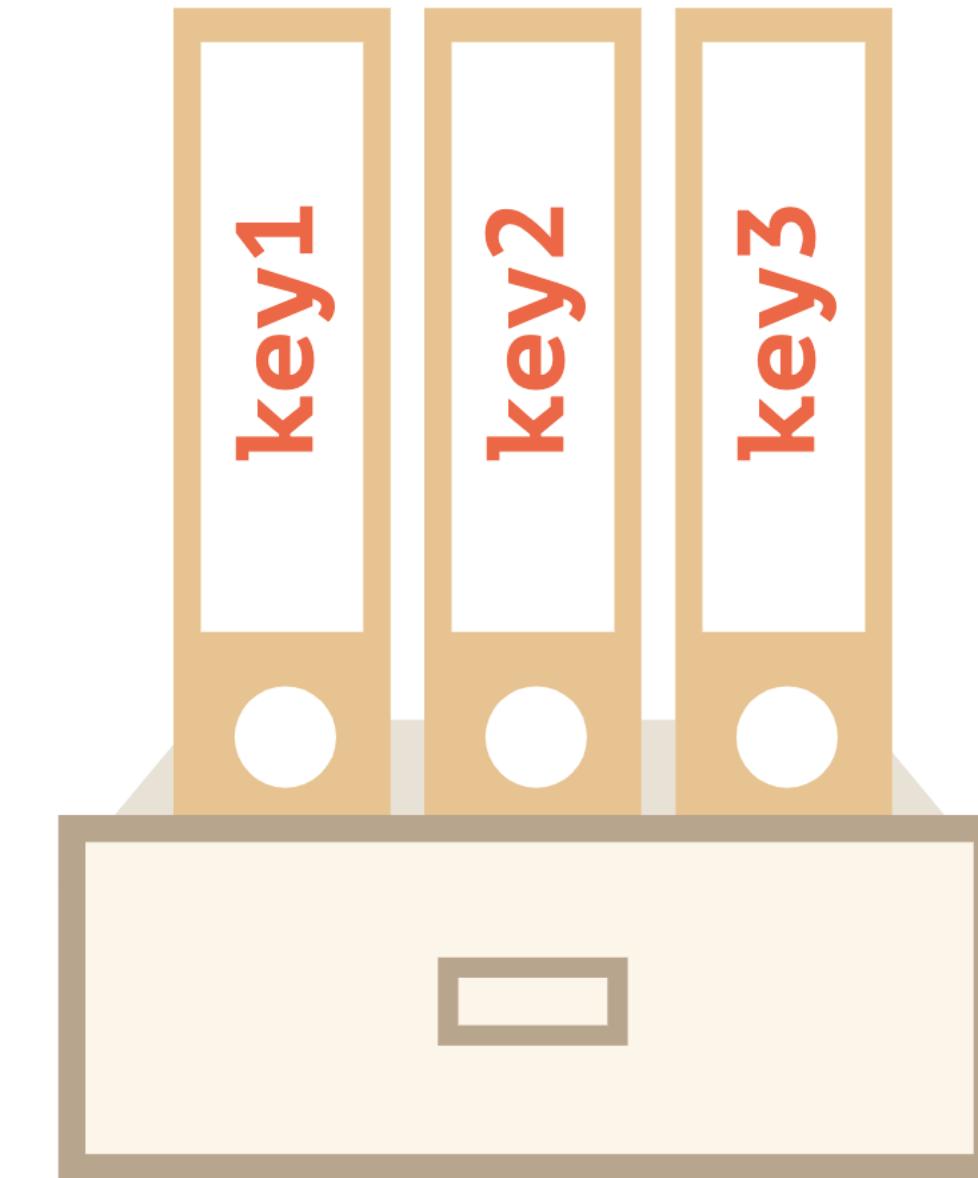
I'm not kidding!
It's an
important skill!



linkedin.com/posts/javascript-developer_activity-6922443466585088001-z1Fl/

O B J E C T S

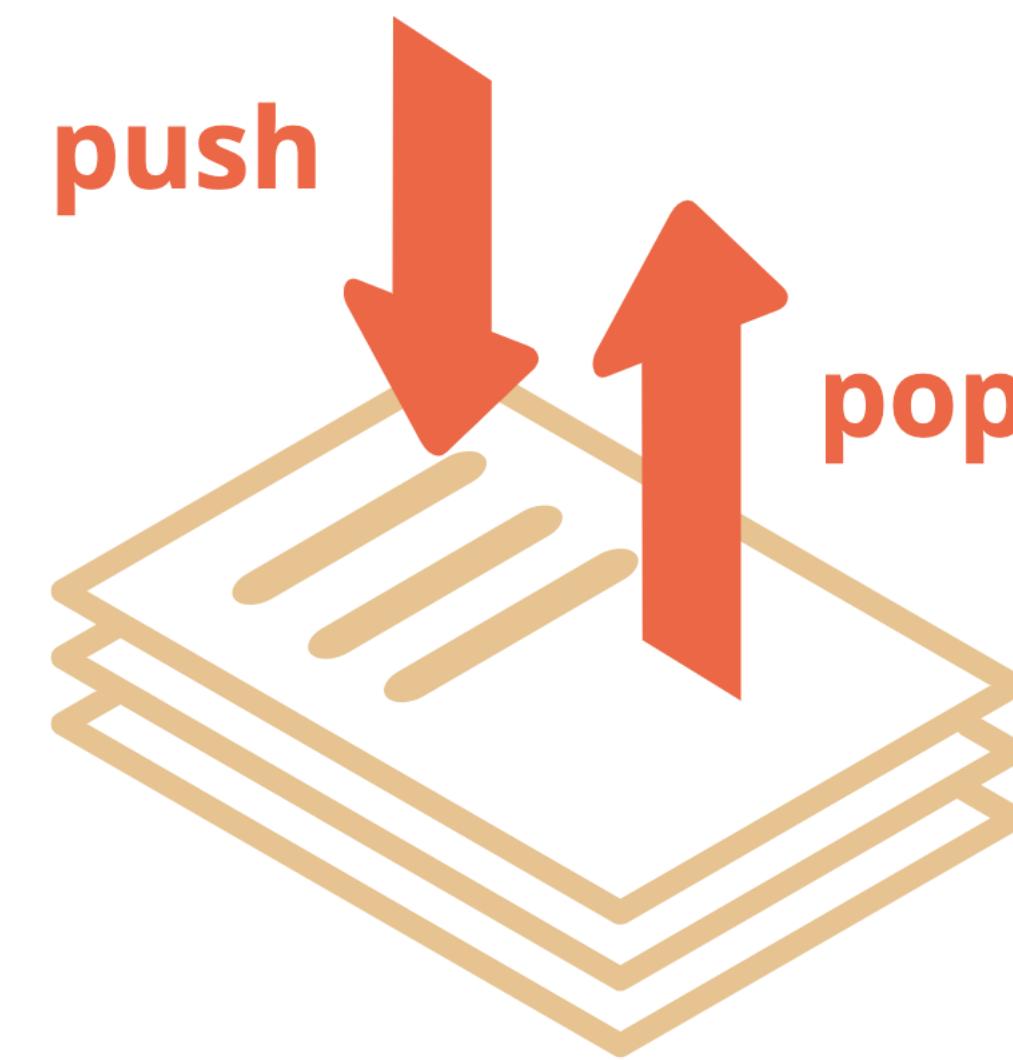
Objects are used to store keyed collections of various data



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(...)`

■ ■ ■ ■	.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(...)

...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" }
];

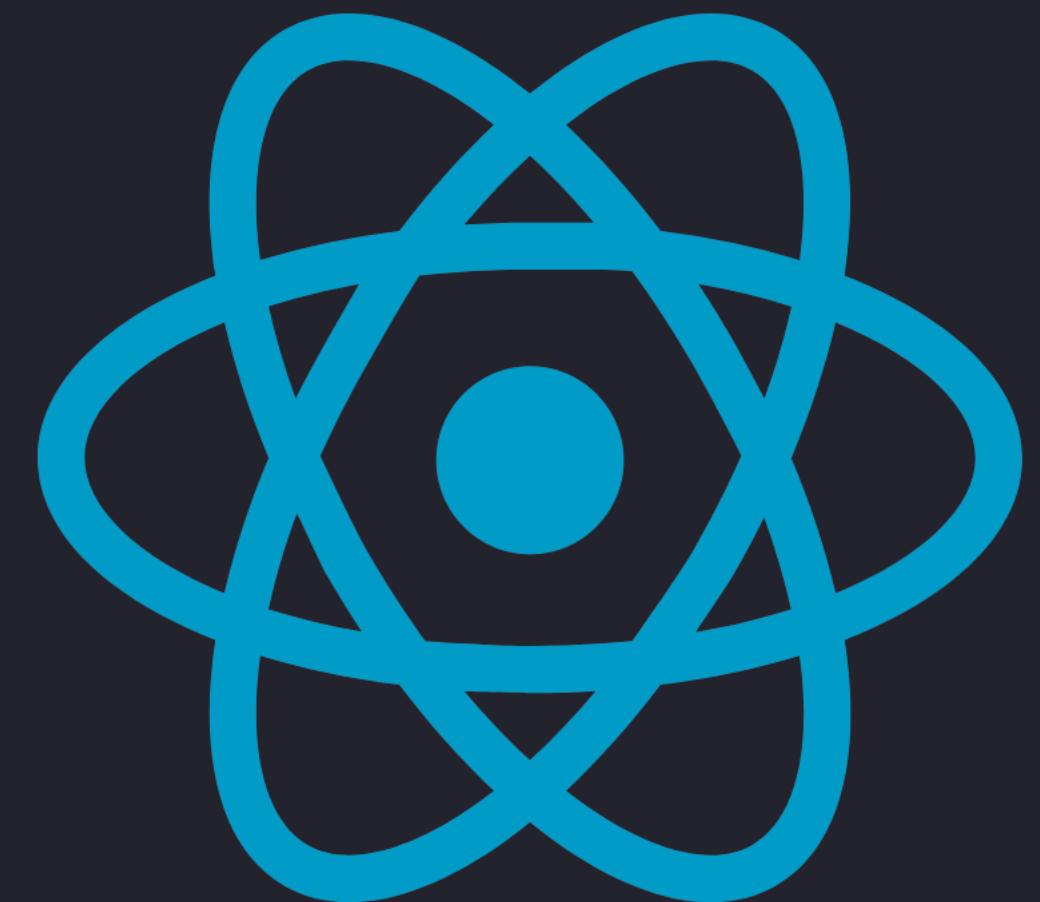
const mapped = persons.map(person => {
  return {
    name: `${person.firstname} ${person.lastname}`
  };
});

console.log(mapped);
```

▼ (3) [{...}, {...}, {...}] *i*

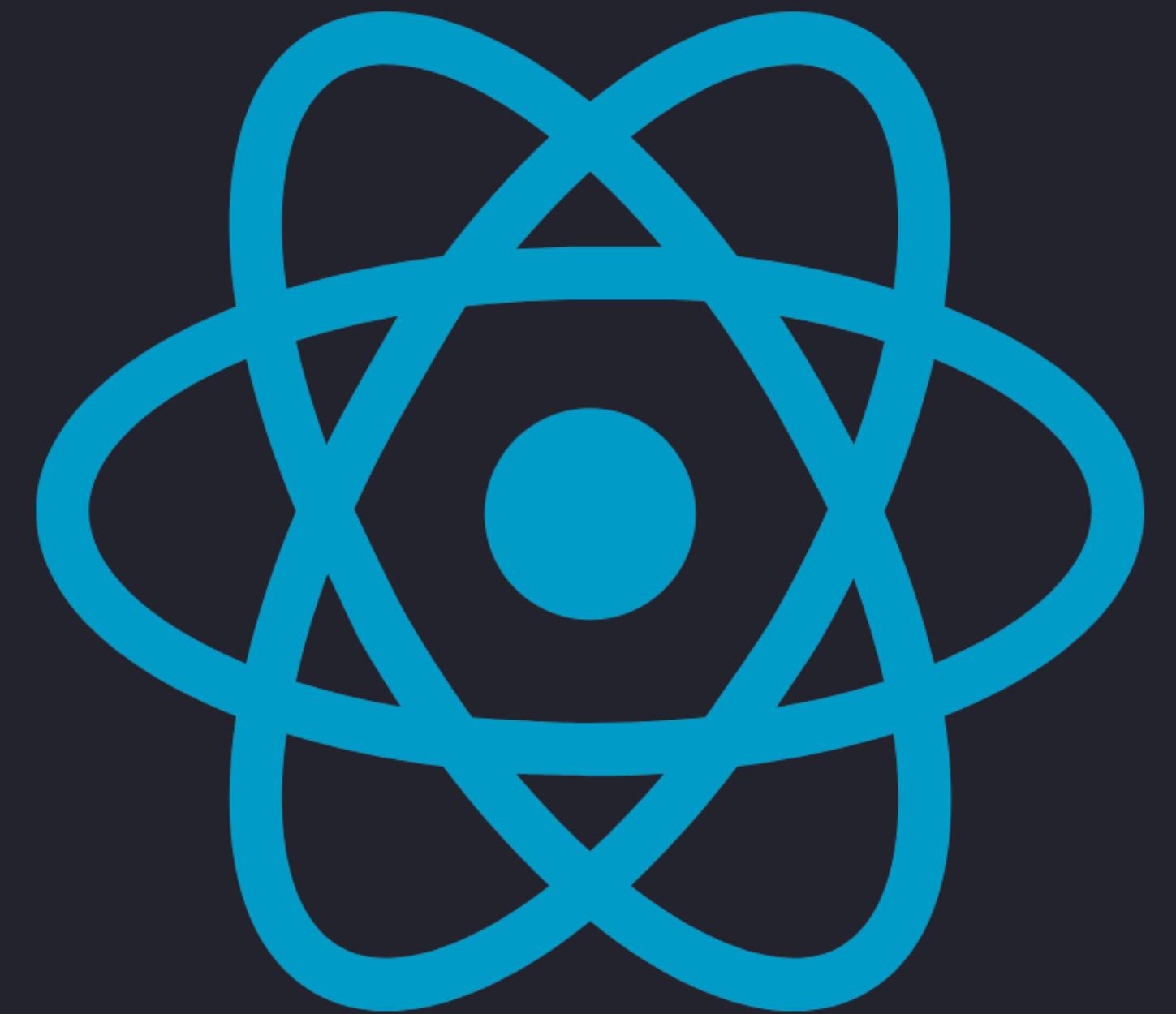
- 0: {name: 'Birgitte Iversen'}
- 1: {name: 'Lykke Dahlen'}
- 2: {name: 'Rasmus Cederdorff'}

length: 3



React

Core Concepts



What is React?

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

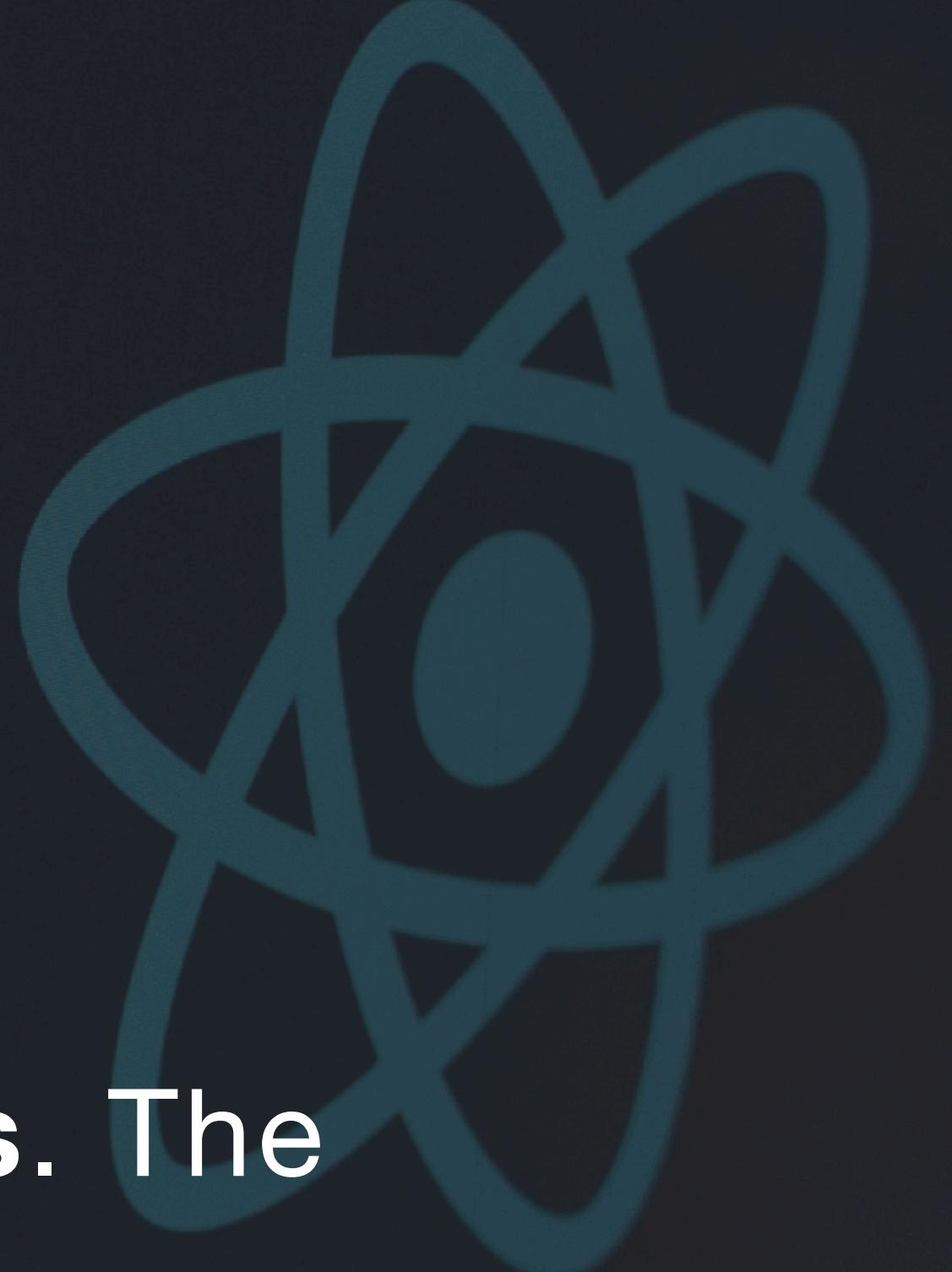
Components, Props & States

The Core Concepts of React

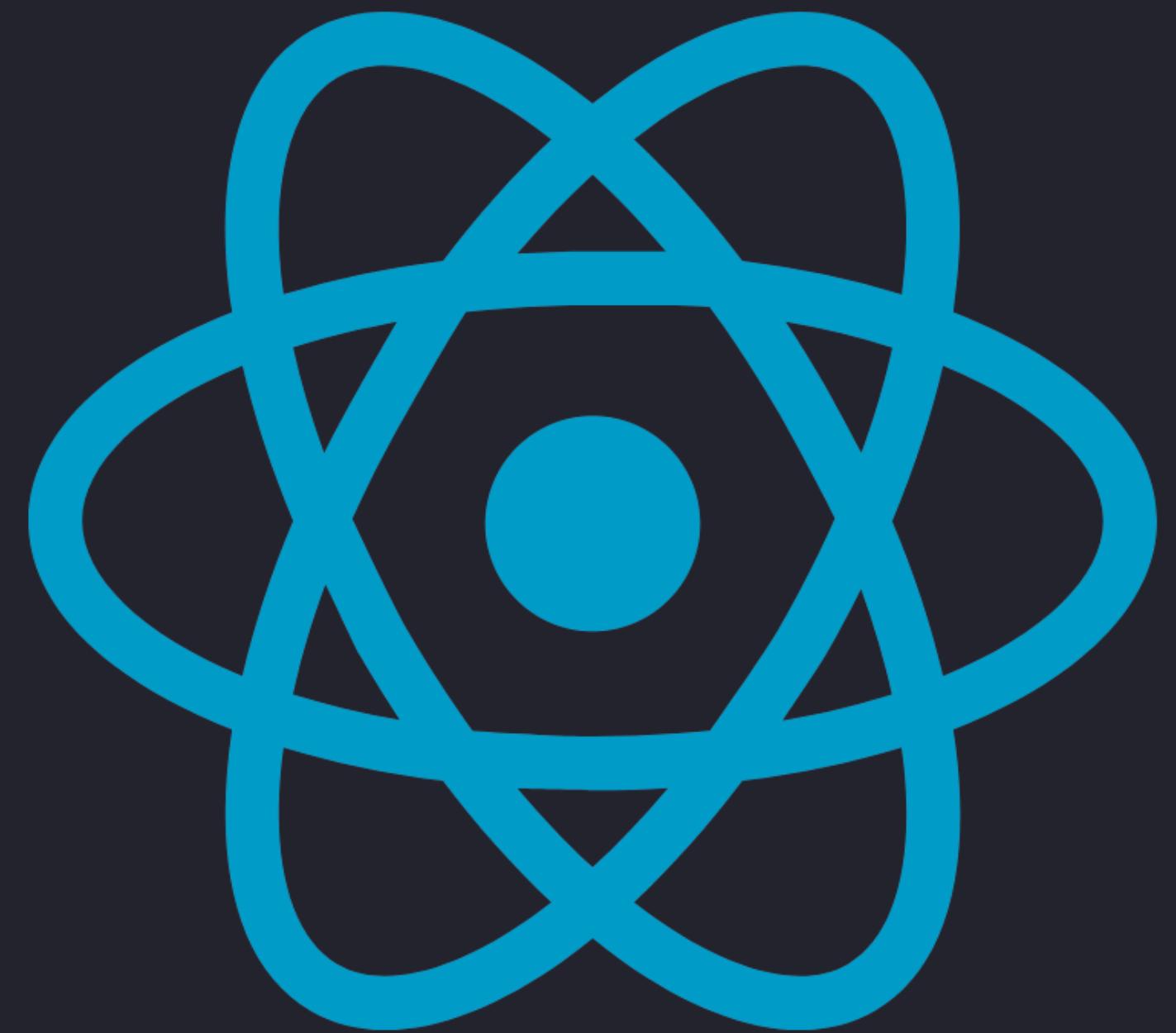


Thinking in React

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



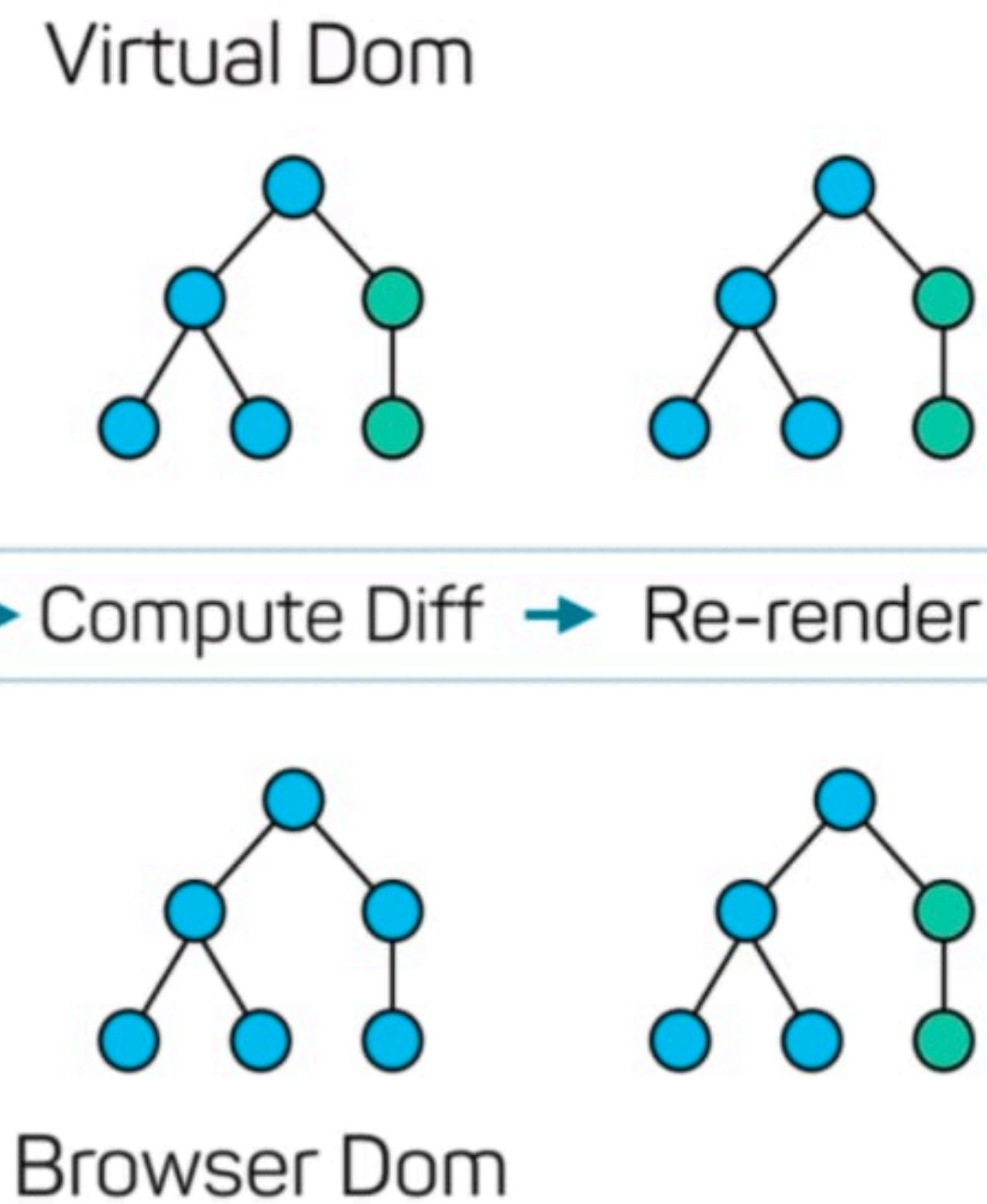
Edit src/App.js and save to reload
Learn React



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!

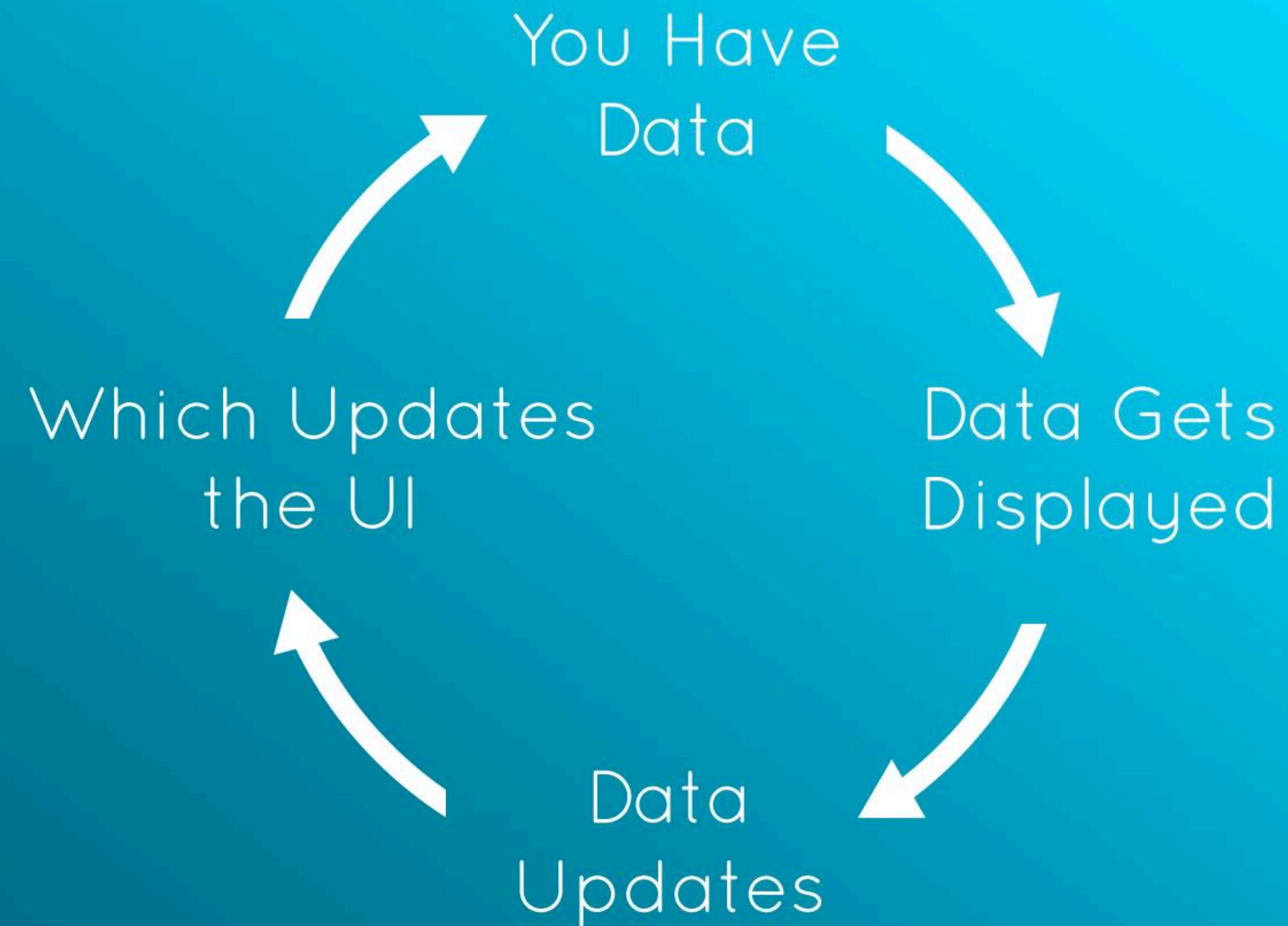
React Virtual DOM



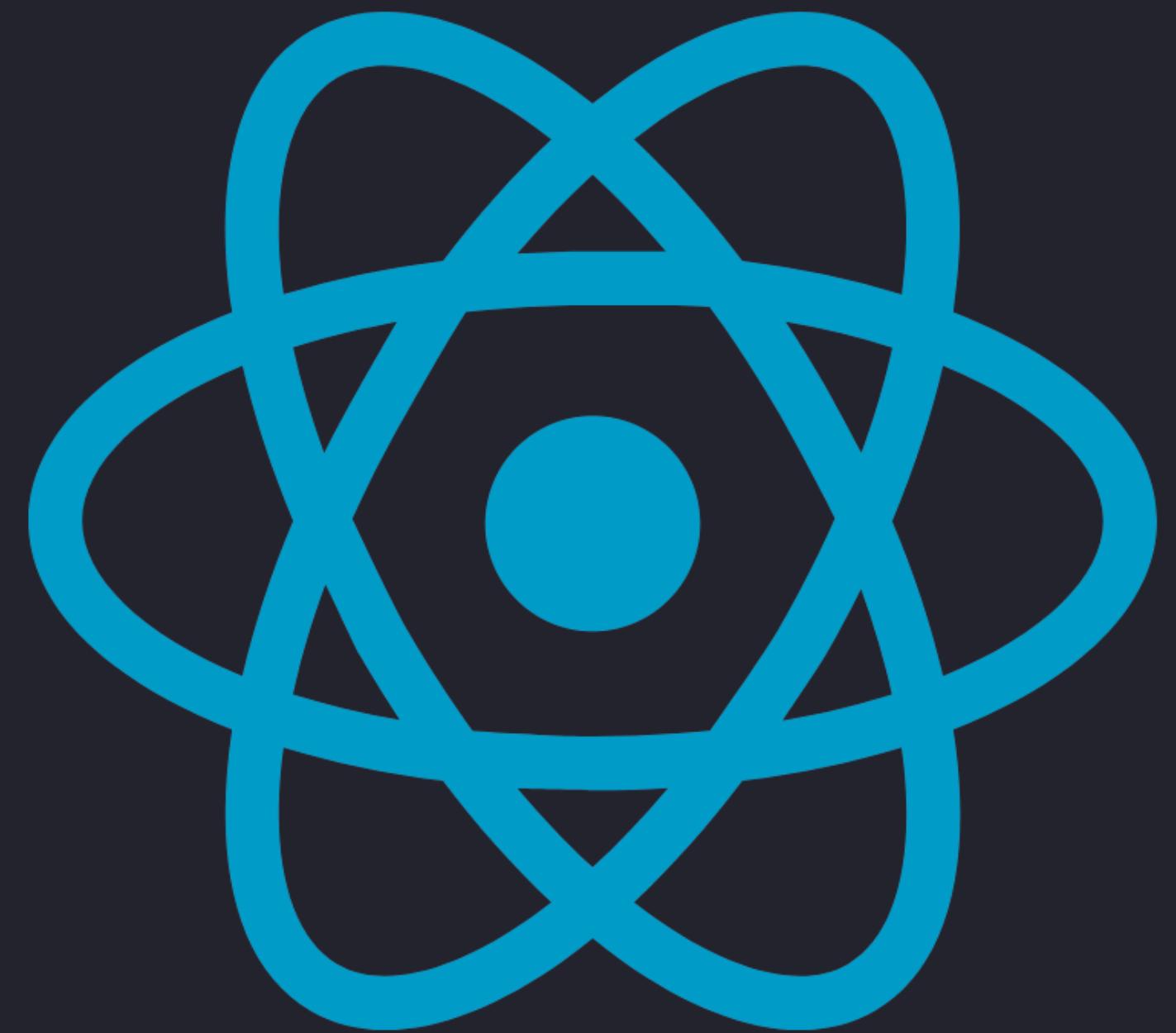
“The virtual DOM (VDOM) is a programming concept where an ideal, or “virtual”, representation of a UI is kept in memory and synced with the “real” DOM by a library such as ReactDOM.”

“You tell React what state you want the UI to be in, and it makes sure the DOM matches that state. This abstracts out the attribute manipulation, event handling, and manual DOM updating [...].”

The React UI Cycle



This happens AUTOMATICALLY
once you define components

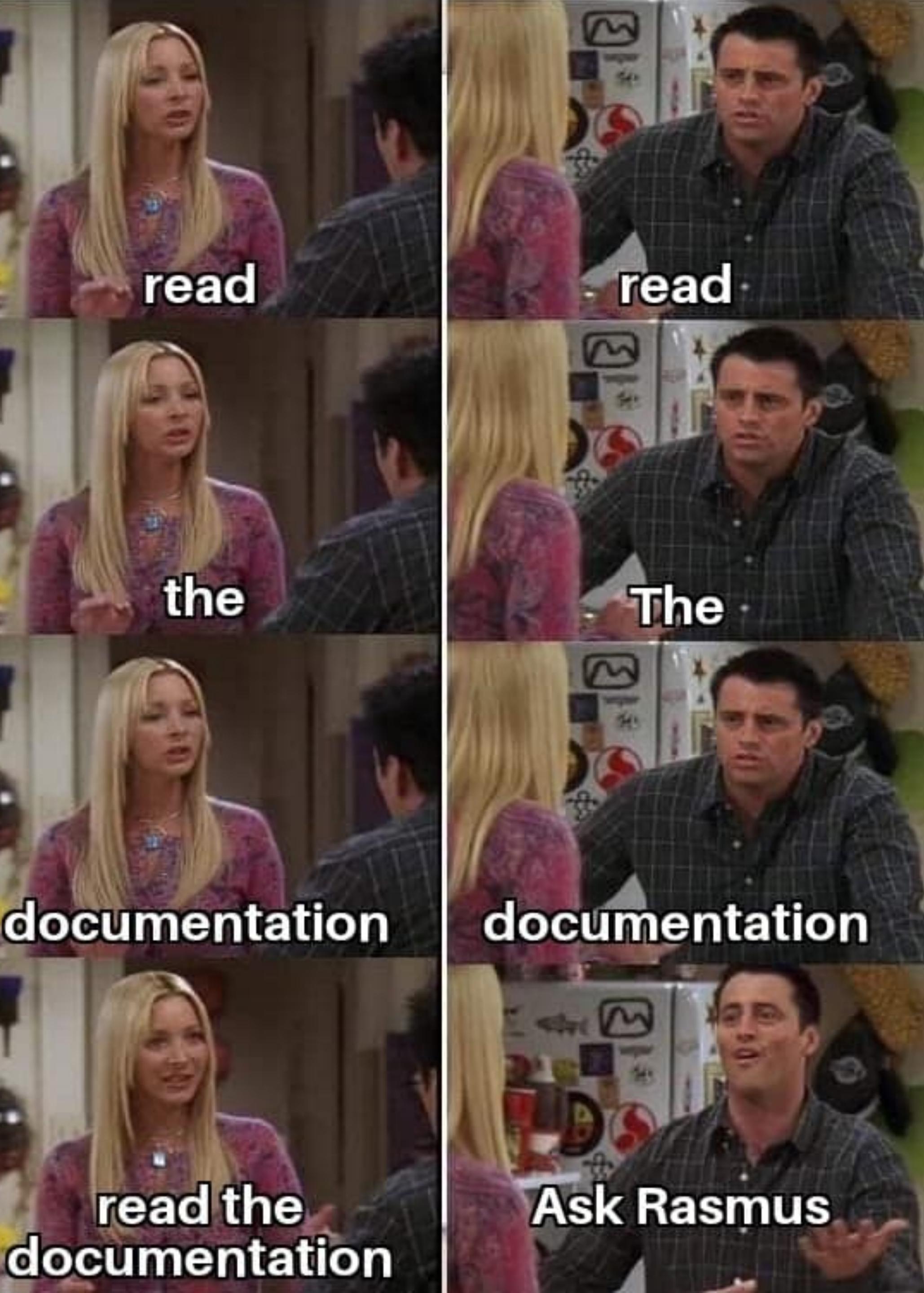


But why?

- It's so hard to work with the DOM and DOM manipulation is slow.
- 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.

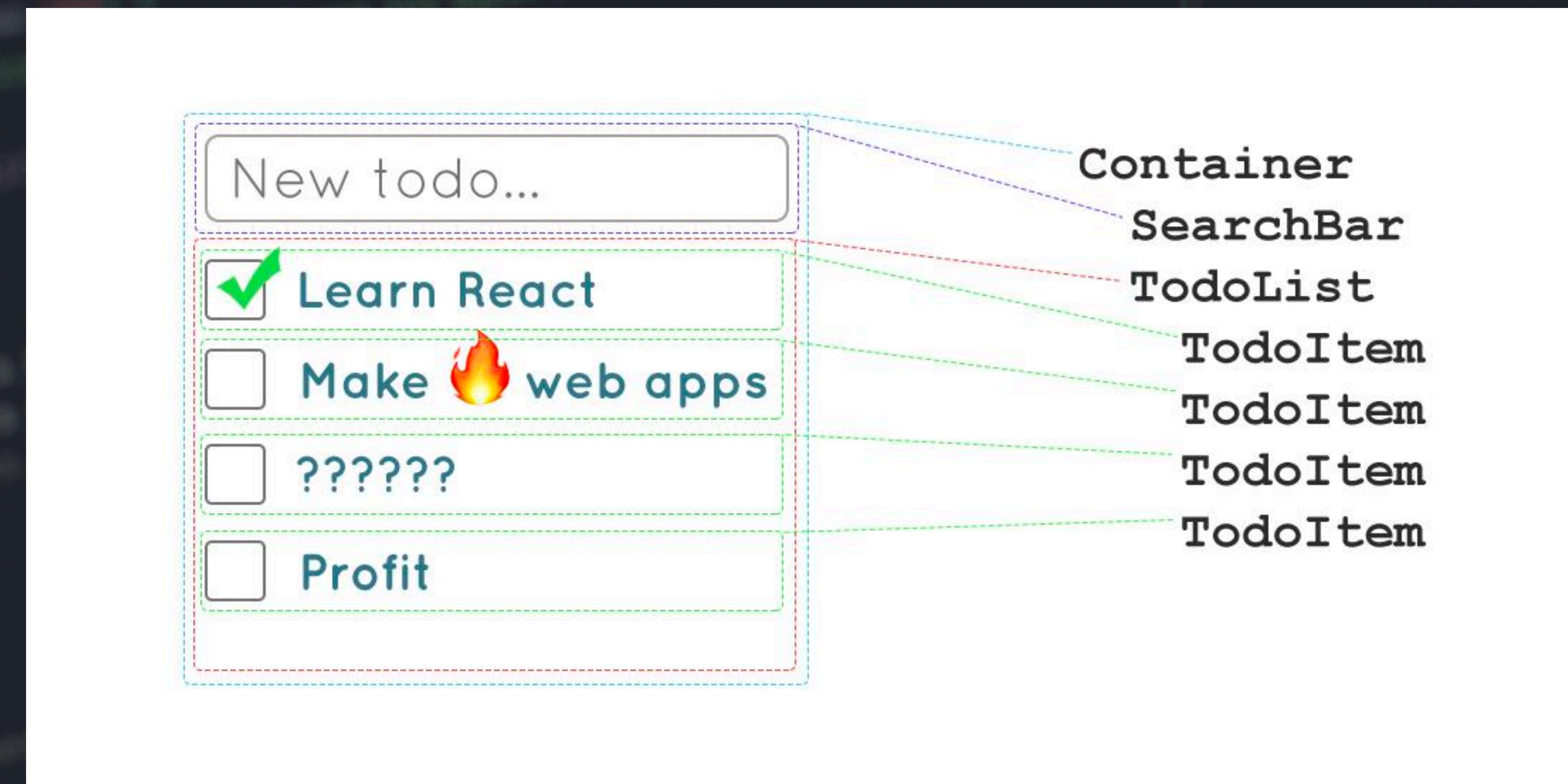
But RACE, how do you know all that stuff?

- [React Resources](#)
- [React Self-study](#)
- [Guides & Tutorials](#)
- [From Vanilla JavaScript to React Developer](#)



How to Think in React

Review slides & note questions



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>  
  );  
}
```

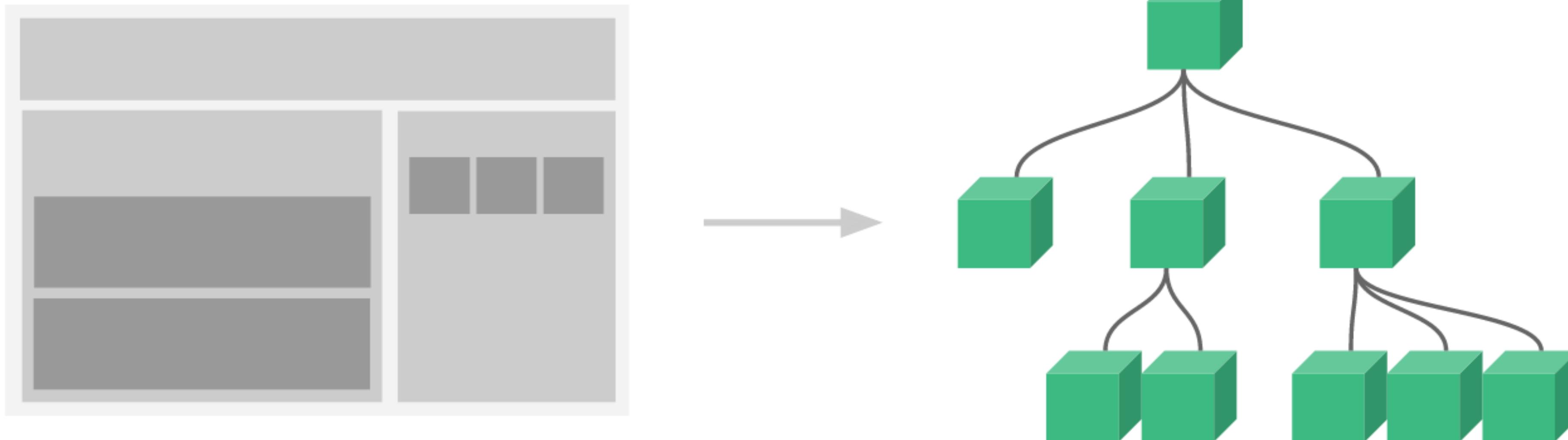
Define the component

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

Use Greeting component
in another component

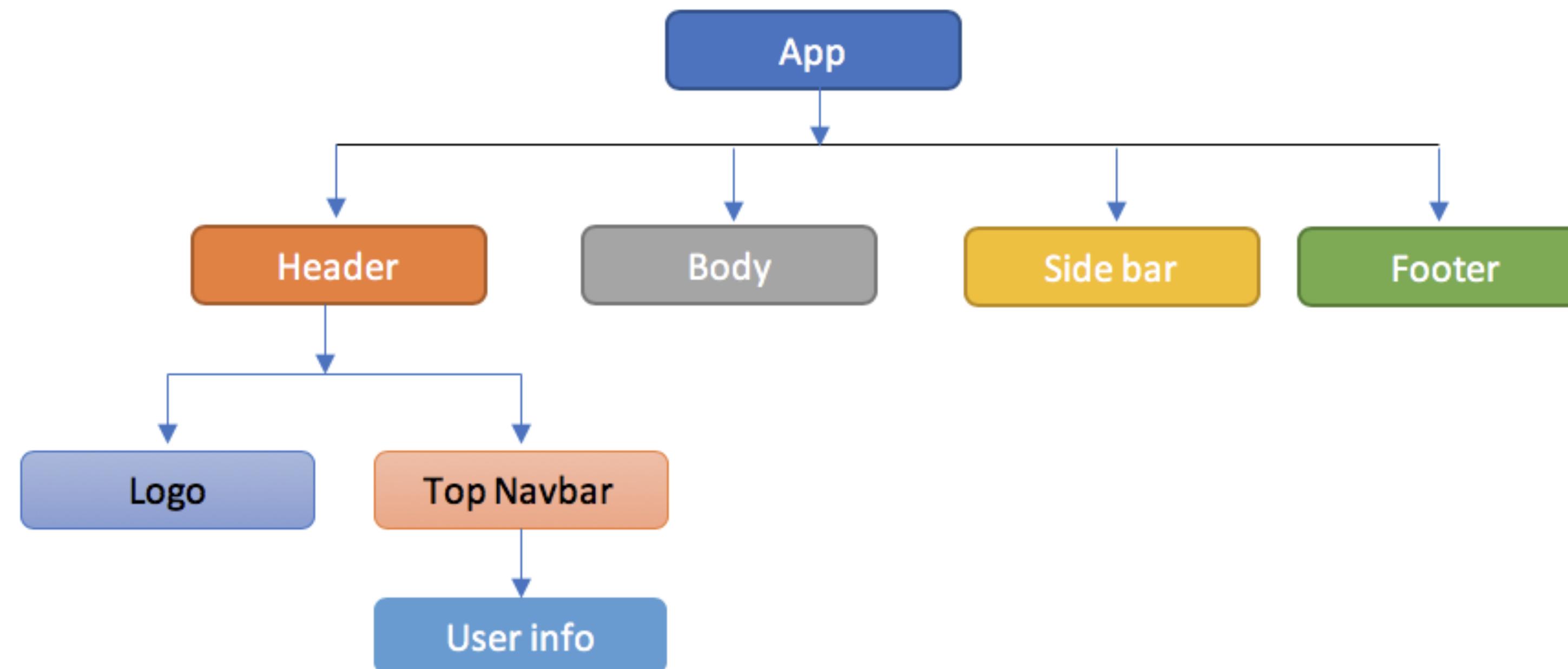
Components without
a closing tag
REQUIRE a closing
slash

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

React Components

https://www.w3schools.com/react/react_components.asp

Tutorials ▾ References ▾ Exercises ▾ Videos NEW

Website NEW Paid Courses Log in

HTML CSS JAVASCRIPT SQL PYTHON PHP BOOTSTRAP HOW TO W3.CSS JAVA

React Tutorial

- React Home
- React Intro
- React Get Started
- React ES6
- React Render HTML
- React JSX
- React Components**
- React Class
- React Props
- React Events
- React Conditionals
- React Lists
- React Forms
- React Router
- React Memo
- React CSS Styling
- React Sass Styling
- React Hooks

affaldsindsamlingen.dk

Forårssrengøring i naturen

ÅBN

React Components

◀ Previous Next ▶

Components are like functions that return HTML elements.

React Components

Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML.

Components come in two types, Class components and Function components, in this tutorial we will concentrate on Function components.

ADVERTISEMENT

Forårssrengøring i naturen

Danmarks Naturfredningsforening inviterer til Danmarks største indsamling af affald. affaldsindsamlingen.dk

Components?

A screenshot of a web browser displaying a React application. The page has a dark header with tabs for 'POSTS', 'CREATE', and 'PROFILE'. Below the header is a grid of six cards, each representing a post. Each card contains a user profile picture, the user's name, their title, a preview image, and a short snippet of text. The cards are arranged in two rows of three.

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

A screenshot of a web browser displaying a Single Page Web App (SPA). The page has a dark header with a 'Users' tab, a 'Create' button, and a 'Random' button. Below the header is a table titled 'ALL USERS' with columns for 'Order by' (set to 'Choose here'), 'Filter by' (set to 'All'), and a search bar. The table lists four users, each with a profile picture, name, email, and enrollment status. There are 'UPDATE' and 'DELETE' buttons for each user entry.

User	Email	Enrollment	Actions
Diana Riis Myjak Andersen	eaadimy@students.eaaa.dk	StudentEnrollment	UPDATE DELETE
German Arias Rodriguez	eaagar@students.eaaa.dk	StudentEnrollment	UPDATE DELETE
Haya Barakat	eaahbar@students.eaaa.dk	StudentEnrollment	UPDATE DELETE

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".

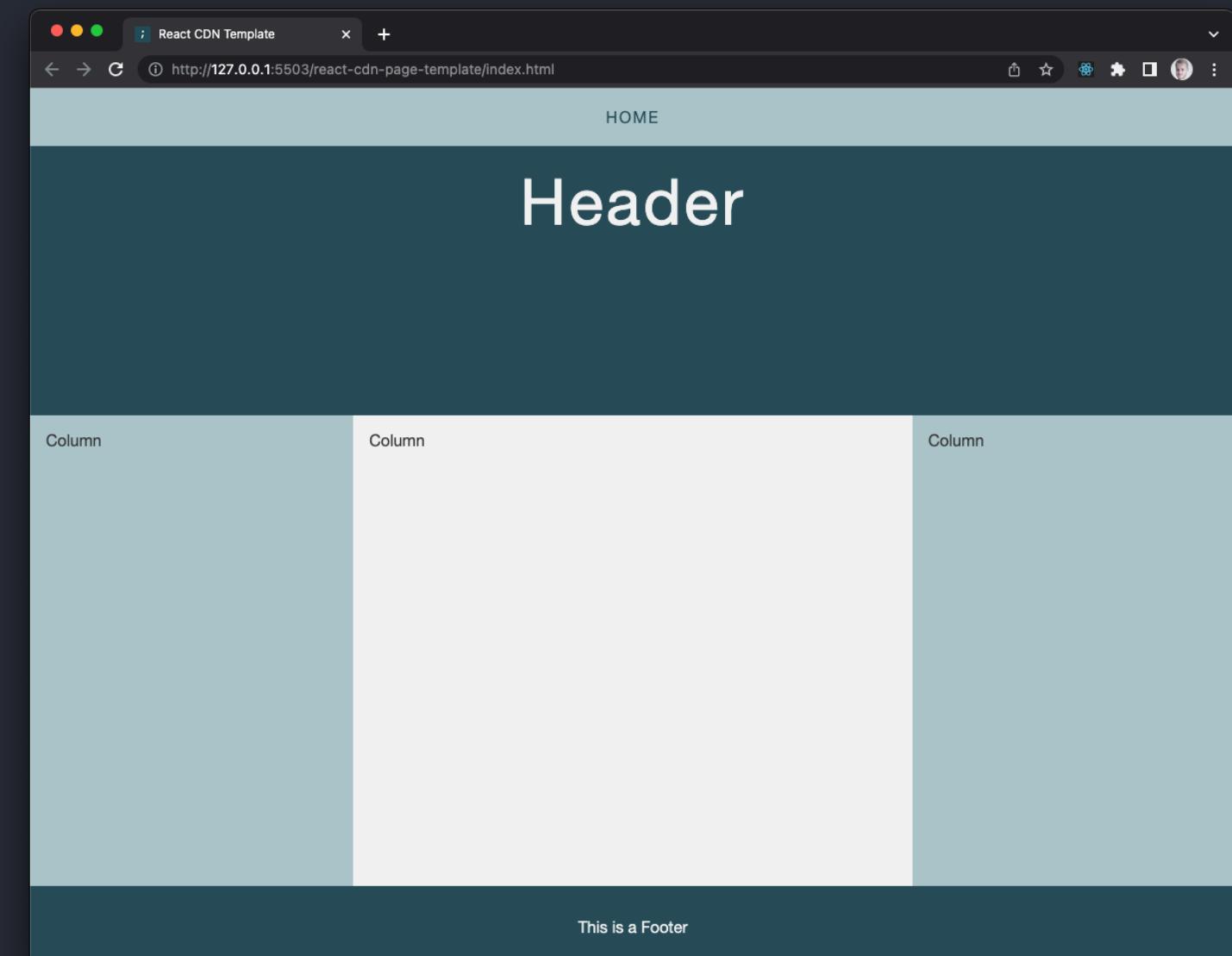
The screenshot shows the website for Erhvervsakademiet Aarhus. The page displays a grid of vocational training programs. Each program includes a thumbnail image, a title, and a brief description. The programs listed are:

- 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

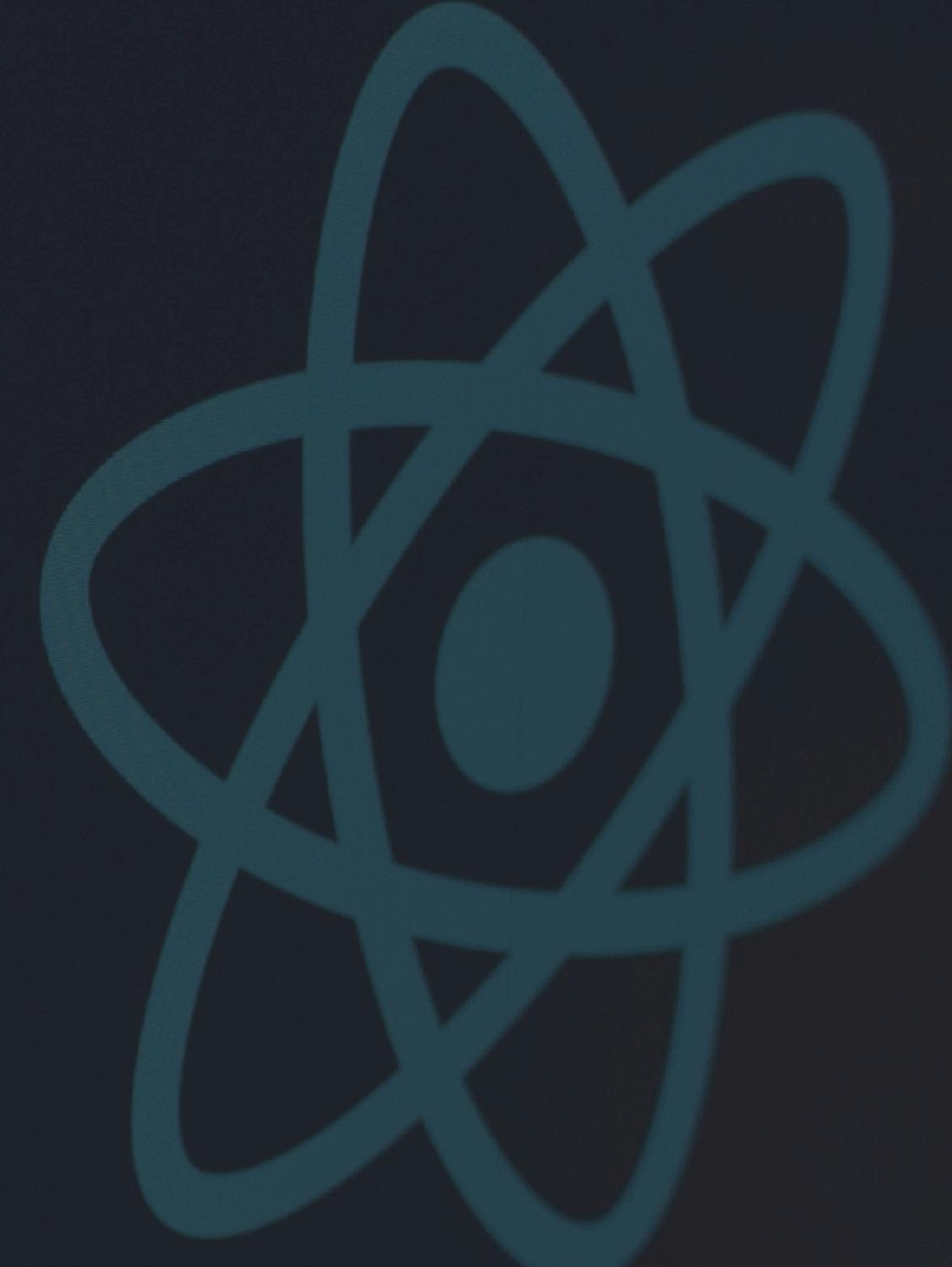
1. Use your implemented Page Layout or `react-cdn-page-template`.
2. Add a prop to your `Header` component called `title`. Make sure the component (function) takes in an argument. The `title` prop must be rendered in the JSX inside the `h1`.
3. Locate the `Header` component in `App` and add the attribute `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.

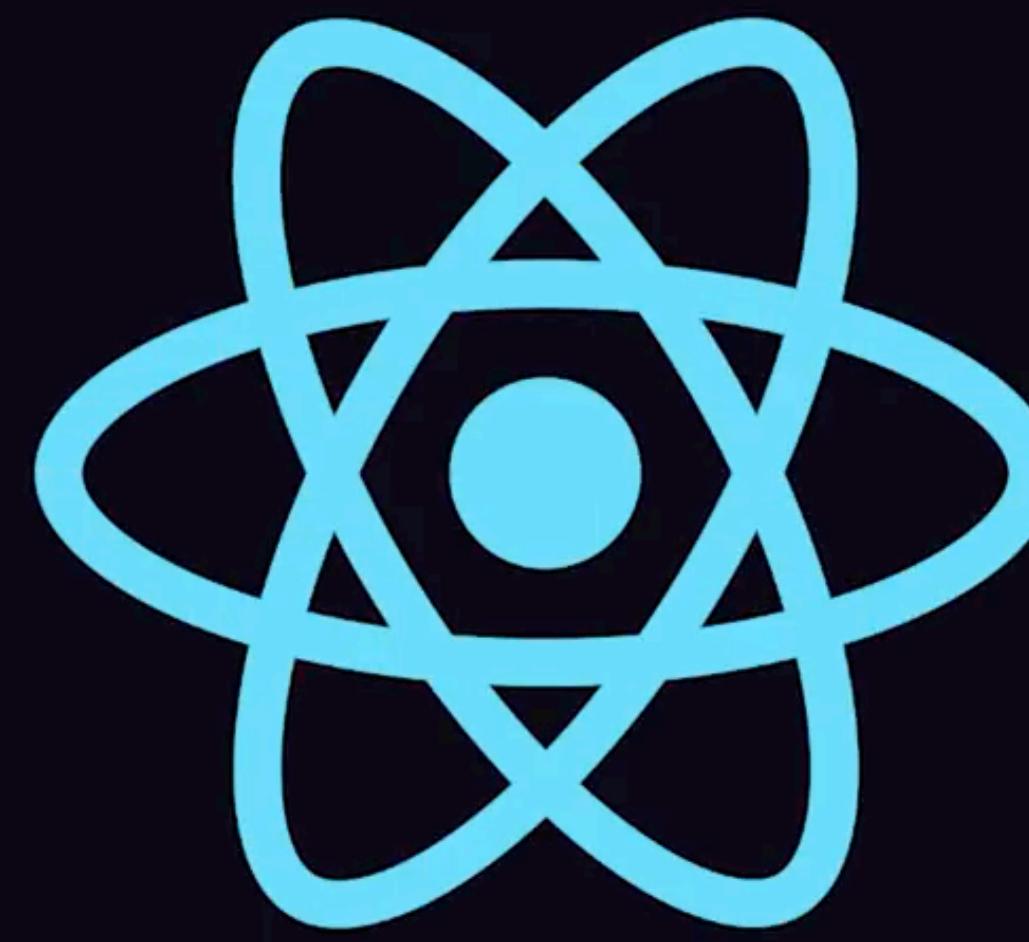
```
function Greeting({ name }) {  
  return <h1>Hello, {name}</h1>;  
}  
  
function App() {  
  return <Greeting name="world" />;  
}
```

React Hooks

useState & useEffect

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





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

React Hooks

The screenshot shows a web browser window with the title "React Hooks" at the top. The URL in the address bar is https://www.w3schools.com/react/react_hooks.asp. The page content is titled "React Hooks". It includes a sidebar with "React Tutorial" links such as React Home, React Intro, React Get Started, React ES6, React Render HTML, React JSX, React Components, React Class, React Props, React Events, React Conditionals, React Lists, React Forms, React Router, React Memo, React CSS Styling, and React Sass Styling. A specific link "What is a Hook?" is highlighted with a green background. The main content area starts with a note about Hooks being added in version 16.8, followed by a paragraph explaining that Hooks allow function components to access state and other features, making class components unnecessary. A yellow callout box states: "Although Hooks generally replace class components, there are no plans to remove classes from React." Below this, a section titled "What is a Hook?" defines Hooks as allowing us to "hook" into React features like state and lifecycle methods. An "Example:" section provides a brief code snippet. On the right side of the page, there are promotional banners for W3Schools videos, a color picker, and social media sharing options (Facebook, Instagram, LinkedIn, YouTube). A footer banner encourages users to get certified by completing a course.

[w3schools.com/react/react_hooks.asp](https://www.w3schools.com/react/react_hooks.asp)

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

Destructuring

The screenshot shows a web browser window with a dark theme, displaying a Notion page titled "From Vanilla JavaScript to React". The page content is organized into sections:

- 1.7. Destructuring**
 - What:** Extract what we need from an existing array or object.
 - Why:** Makes it easy to extract only what we need from an existing array or object.
- Objects**

```
const teacher = {
  name: "Morten",
  email: "moab@eaaa.dk"
};

//choose name and email
const { name, email } = teacher;
//name: "Morten"
//email: "moab@eaaa.dk"
```
- Arrays**

```
const teachers = ["Rasmus", "Morten", "Dan"];

//choose two names
const [mrFrontend, mrWebComponents] = teachers;
//mrFrontend: "Rasmus"
//mrWebComponents: "Morten"

//choose 1 & 3 in the array
const [mrFrontend, , mrReact] = teachers;
```

Destructuring

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. Use a previous React project or [react-cdn-template](#),
2. Create a component called Counter.
3. 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.
4. Render the Counter component in your App component or another component.
5. Test and think of the pros of using React and states compared to Vanilla JS.
6. Try to render the Counter component 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>
  )
}
```

Annotations for the code:

- Wrap entire JS in curly braces
- Map over data that was passed in as props
- Pass unique key attribute to each element
- Use curly braces like normal to display data
- Return JSX from each iteration

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]);
```

Fetch

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

Fetch & useEffect in React

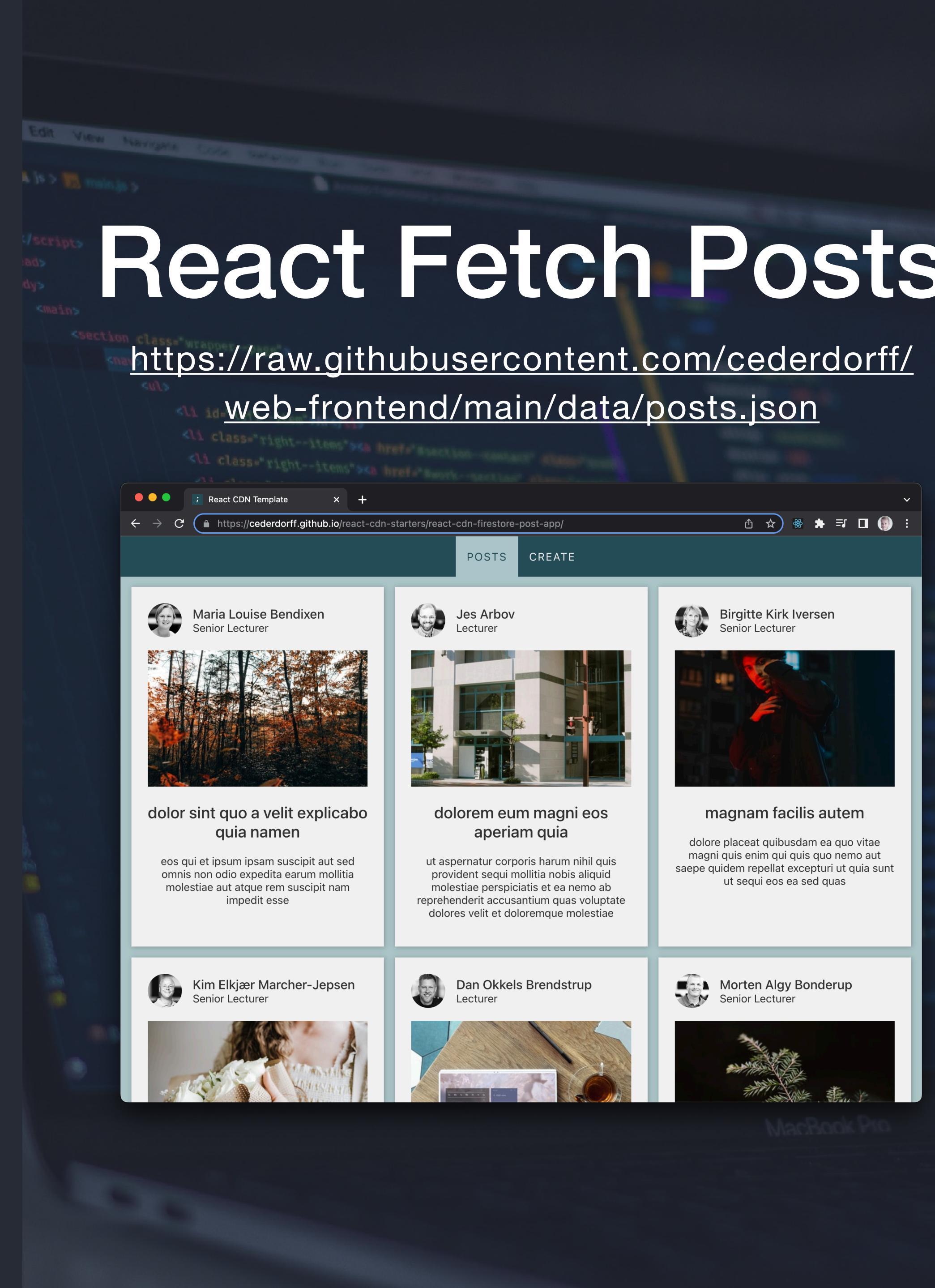
In useEffect, perform the sideeffect (fetch) to get data from a data source.
And save the response data in a state using useState.

```
function PostsPage() {
  const [posts, setPosts] = useState([]);

  useEffect(() => {
    async function getPosts() {
      const url = "https://raw.githubusercontent.com/cederdorff/web-frontend/main/data/posts.json";
      const response = await fetch(url);
      const data = await response.json();
      setPosts(data);
    }
    getPosts();
  }, []);

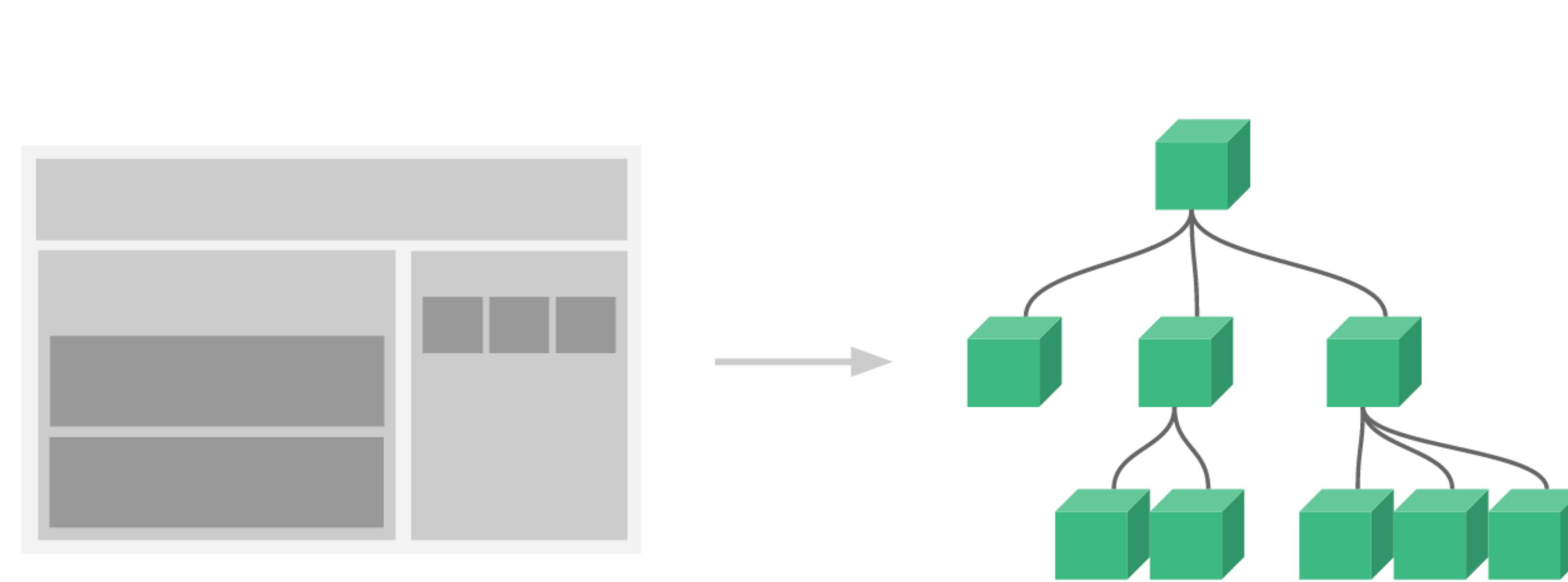
  return (
    <section className="page">
      {posts.map(post => (
        <div>
          <h2>{post.title}</h2>
          <p>{post.content}</p>
        </div>
      ))}
    </section>
  );
}
```

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 (previous lecture). 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.

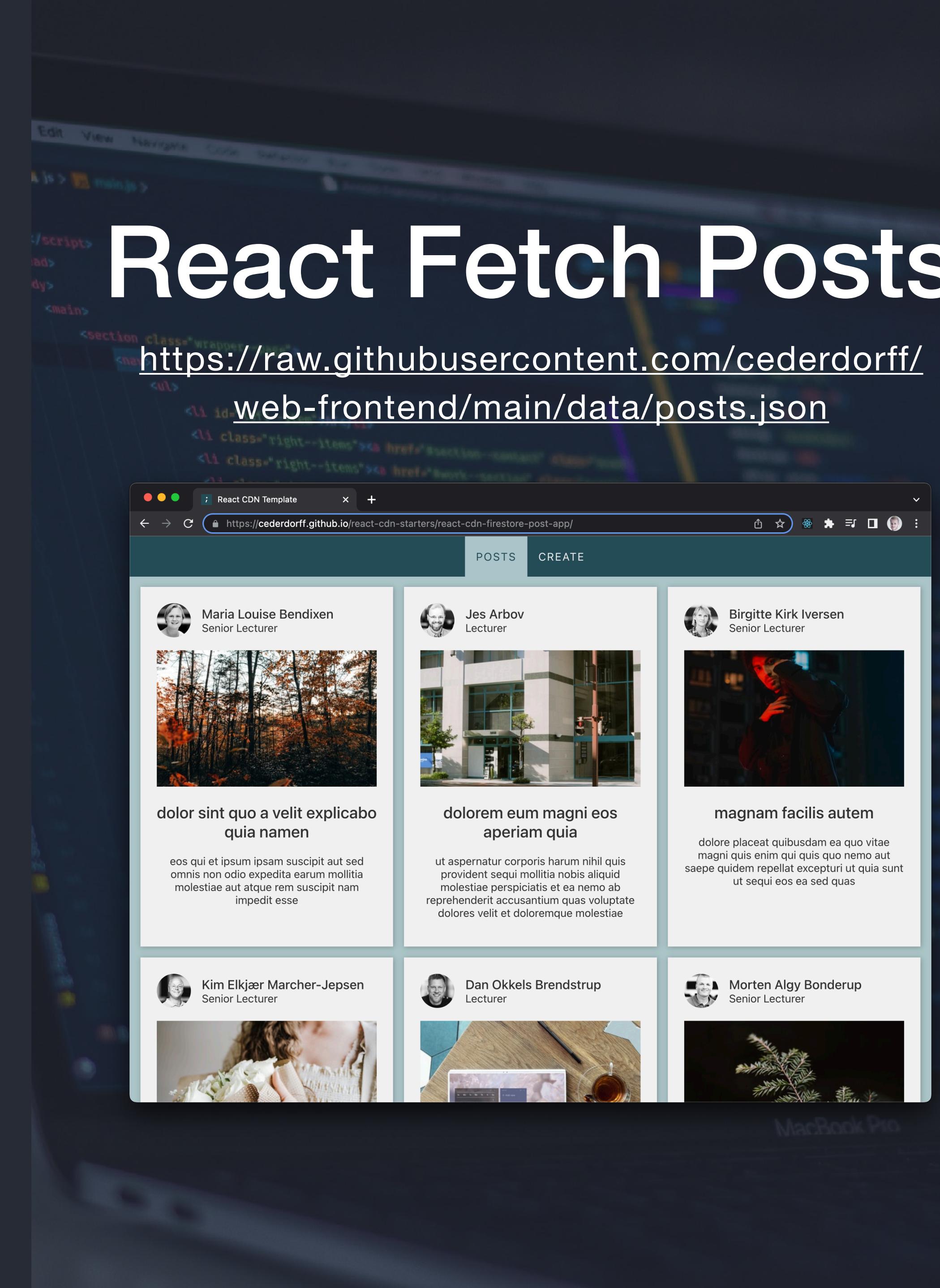


Build Reusable Components

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

Shorter and simpler way to do if/else

The screenshot shows a web browser window with the title "Conditional branching: if, ?" and the URL <https://javascript.info/ifelse#conditional-operator>. The page content is about the conditional operator (ternary operator). It includes a sidebar with navigation links for chapters like "JavaScript Fundamentals" and "Lesson navigation". The main content area has a heading "Conditional operator '?'". It explains that sometimes we need to assign a variable depending on a condition, and provides an example code snippet:

```
1 let accessAllowed;
2 let age = prompt('How old are you?', '');
3
4 if (age > 18) {
5   accessAllowed = true;
6 } else {
7   accessAllowed = false;
8 }
9
10 alert(accessAllowed);
```

It then states that the so-called "conditional" or "question mark" operator lets us do that in a shorter and simpler way. It describes the syntax: `let result = condition ? value1 : value2;`. It explains that the `condition` is evaluated: if it's truthy then `value1` is returned, otherwise – `value2`. An example is shown:

```
1 let accessAllowed = (age > 18) ? true : false;
```

At the bottom, there is an advertisement for "Kendo UI Kits for Figma: Material, Bootstrap, Default" with a "GET UI KITS" button.

<https://javascript.info/ifelse#conditional-operator>
Ternary Operator

? . is a safe way to access nested object properties

The screenshot shows a dark-themed web browser window displaying the [JavaScript.info](https://javascript.info/optional-chaining) website. The page title is "Optional chaining '?.'". The URL in the address bar is <https://javascript.info/optional-chaining>. The page content is about optional chaining, a recent addition to the JavaScript language. It discusses the "non-existing property" problem and provides code examples. A sidebar on the left contains navigation links for chapters like "Objects: the basics" and "Lesson navigation". A sidebar at the bottom is sponsored by LogicMonitor.

Chapter
Objects: the basics

Lesson navigation

The "non-existing property" problem

Optional chaining

Short-circuiting

Other variants: `?()`, `?[]`

Summary

Comments

Share

[Edit on GitHub](#)

LogicMonitor

Intelligent insights you can actually use. Smarter troubleshooting, more possibilities for your applications.

Optional chaining '?.' is a safe way to access nested object properties, even if an intermediate property doesn't exist.

Optional chaining '?.'

A recent addition

This is a recent addition to the language. Old browsers may need polyfills.

The optional chaining `?.` is a safe way to access nested object properties, even if an intermediate property doesn't exist.

The "non-existing property" problem

If you've just started to read the tutorial and learn JavaScript, maybe the problem hasn't touched you yet, but it's quite common.

As an example, let's say we have `user` objects that hold the information about our users.

Most of our users have addresses in `user.address` property, with the street `user.address.street`, but some did not provide them.

In such case, when we attempt to get `user.address.street`, and the user happens to be without an address, we get an error:

```
1 let user = {}; // a user without "address" property
2
3 alert(user.address.street); // Error!
```

<https://javascript.info/optional-chaining>

?? returns the first argument if it's not null/undefined

The screenshot shows a dark-themed web browser window displaying the [Nullish coalescing operator ??](https://javascript.info/nullish-coalescing-operator) article from the [JavaScript.info](https://javascript.info) website.

The page title is "Nullish coalescing operator '??'" and the subtitle indicates it is a "Recent addition". A note states: "This is a recent addition to the language. Old browsers may need polyfills."

The main content explains that the nullish coalescing operator is written as two question marks `??`. It treats `null` and `undefined` similarly. The result of `a ?? b` is:

- if `a` is defined, then `a`,
- if `a` isn't defined, then `b`.

In other words, `??` returns the first argument if it's not `null/undefined`. Otherwise, the second one.

The nullish coalescing operator isn't anything completely new. It's just a nice syntax to get the first "defined" value of the two.

We can rewrite `result = a ?? b` using the operators that we already know, like this:

```
1 result = (a !== null && a !== undefined) ? a : b;
```

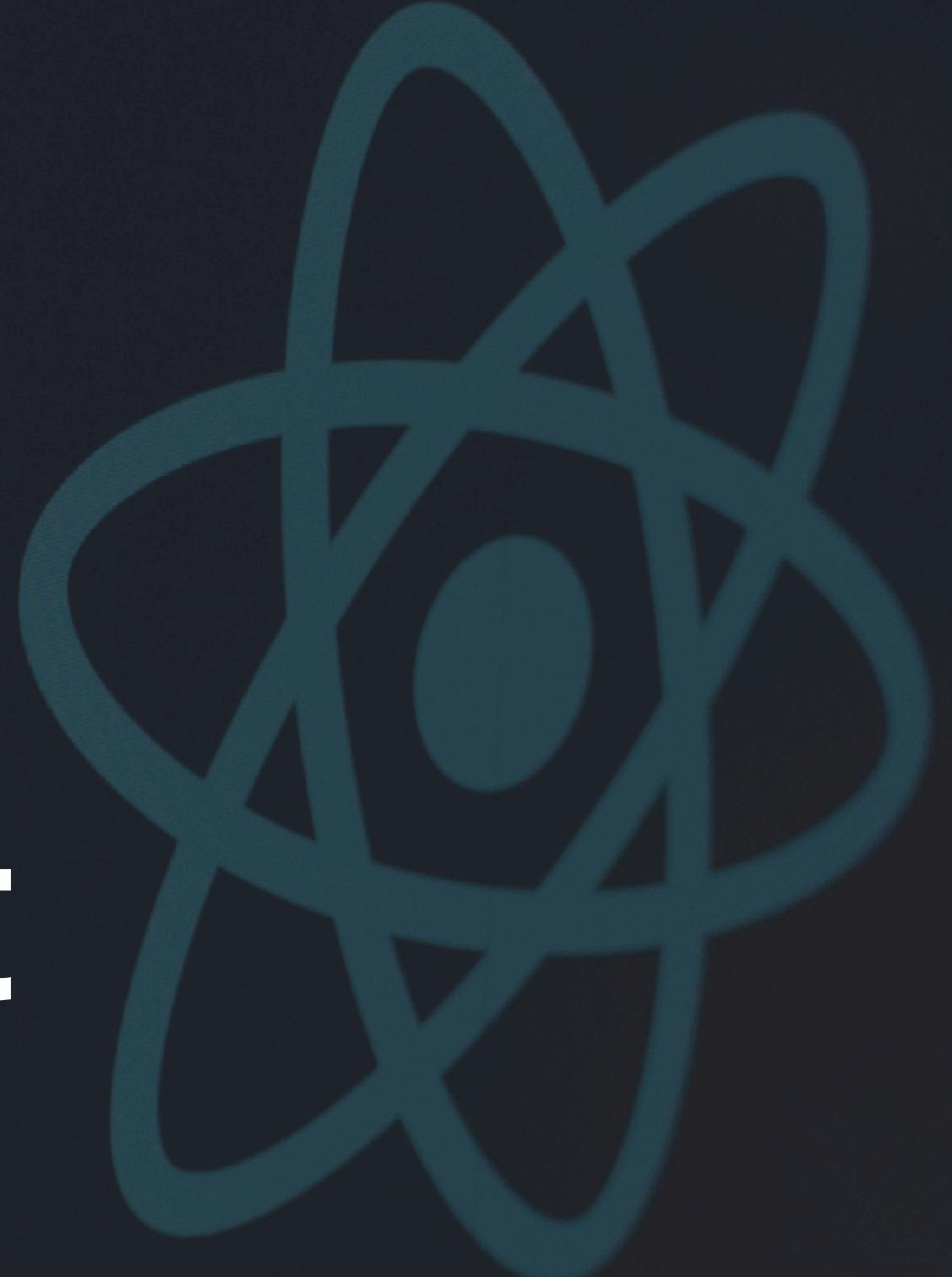
The left sidebar contains a navigation menu with links to "Chapter", "JavaScript Fundamentals", "Lesson navigation", "Comparison with ||", "Precedence", "Summary", "Comments", "Share" (with Twitter and Facebook icons), and "Edit on GitHub". There is also an advertisement for "Ad by Carbon" featuring "PagerDuty" and an "eBook" titled "Best practices for full-service ownership".

<https://javascript.info/nullish-coalescing-operator>

Async JS In React

fetch(...), await & async

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



Async JS

JavaScript reads and runs the script from top to bottom.

JavaScript functions are executed in the sequence they are called. Not in the sequence they are defined.

... by default JavaScript is synchronous.

JS is Synchronous & Single-Threaded

```
function myFirst() {  
  console.log("Hello");  
}
```

```
function mySecond() {  
  console.log("Goodbye");  
}
```

```
mySecond();  
myFirst();
```

```
/* ----- Global Variables ----- */
let _users = [];
let _selectedUserId;

/* ----- */

async function fetchUsers() { ... }
function appendUsers(usersArray) { ... }

// ===== INIT APP =====

async function initApp() {
    await fetchUsers();
    appendUsers(_users);
}

initApp();
```

With callbacks, we can make JS Asynchronous

```
setTimeout(() => {
  console.log("Hey, I'm async!");
}, 3000);

btn.addEventListener('click', () => {
  alert("Hey, you clicked me!");
});
```

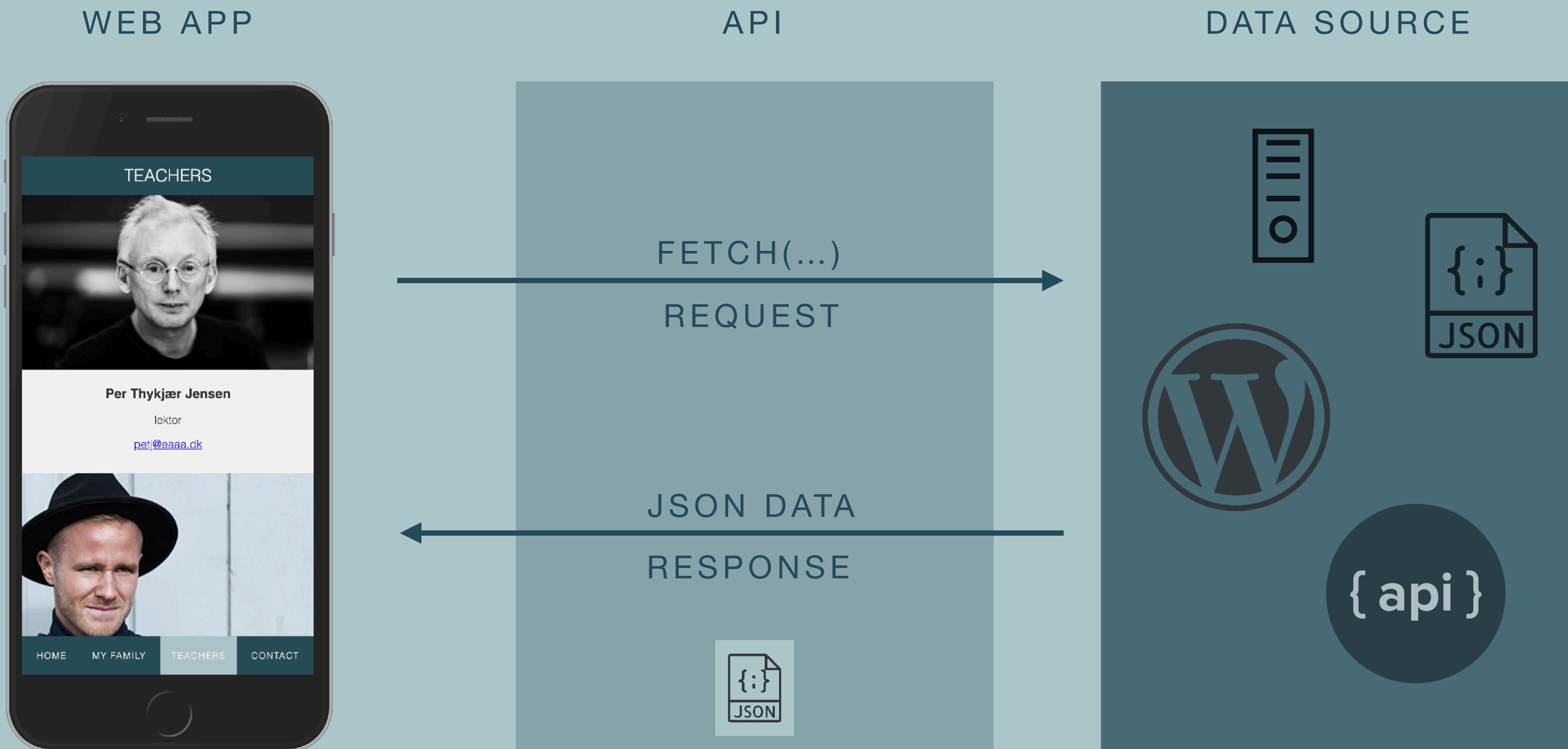
```
// fetch with callbacks
fetch("https://cederdorff.github.io/web-frontend/canvas")
  .then(function (response) {
    return response.json();
  })
  .then(function (data) {
    console.log(data);
  });
});
```

Fetch is Asynchronous

And it's about making HTTP requests in JavaScript.
... and a way to get & post date from and to a data source.

```
// fetch with callbacks
fetch("https://cederdorff.github.io/web-frontend/canvas-users/data.json")
  .then(function (response) {
    return response.json();
  })
  .then(function (data) {
    console.log(data);
  });
// or with async/await
const response = await fetch("https://cederdorff.github.io/web-frontend/canvas-users/data.json");
const data = await response.json();
console.log(data);
```

Web Development



**"async and await make promises
easier to write"**

async makes a function return a Promise

await makes a function wait for a Promise

https://www.w3schools.com/js/js_async.asp

Fetch returns a promise

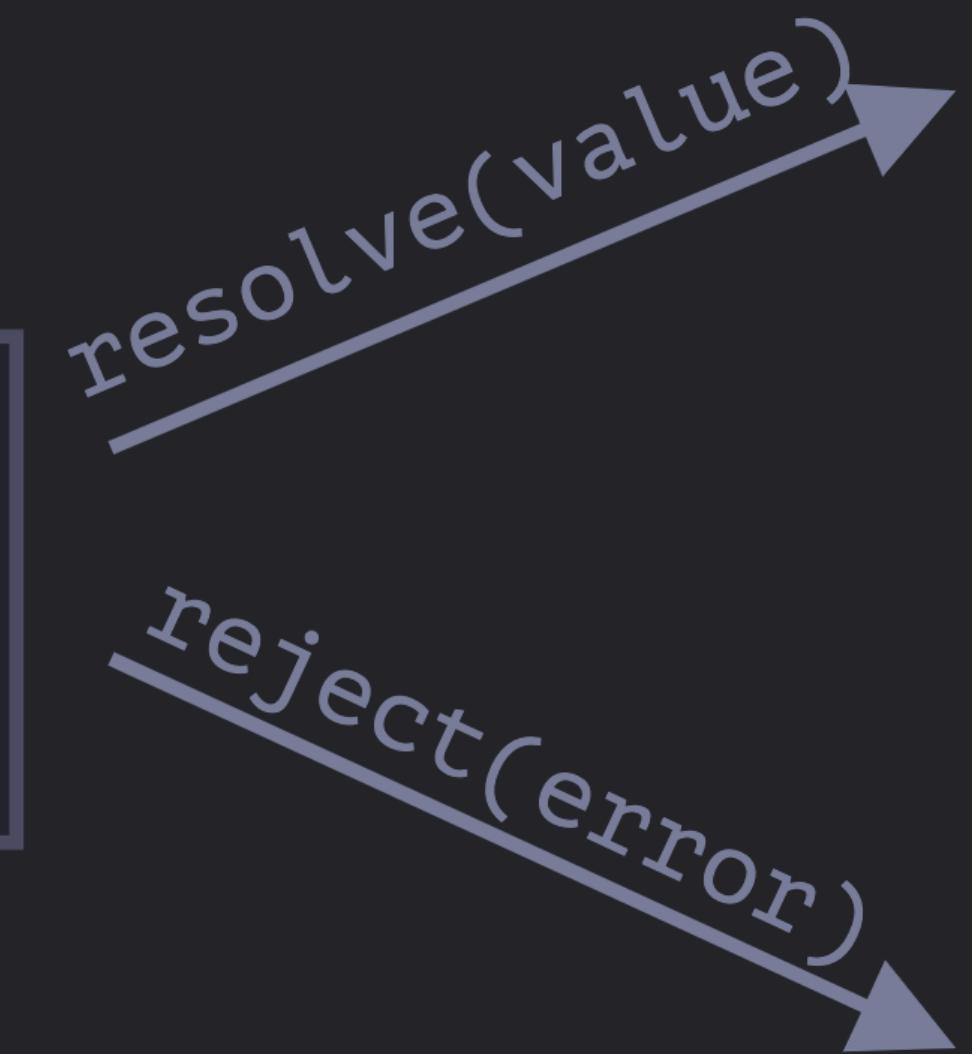
We can use `await` to wait for `fetch` to finish



https://www.w3schools.com/js/js_async.asp

```
new Promise(executor)
```

state: "pending"
result: undefined



state: "fulfilled"
result: value

state: "rejected"
result: error

Fetch: callback (then) vs async/await

```
// fetch with callbacks
fetch("https://cederdorff.github.io/web-frontend/canvas-users/data.json")
  .then(function (response) {
    return response.json();
  })
  .then(function (data) {
    console.log(data);
  });
// 
// 
// or with async/await
const response = await fetch("https://cederdorff.github.io/web-frontend/canvas-users/data.json");
const data = await response.json();
console.log(data);
```

All you need to know

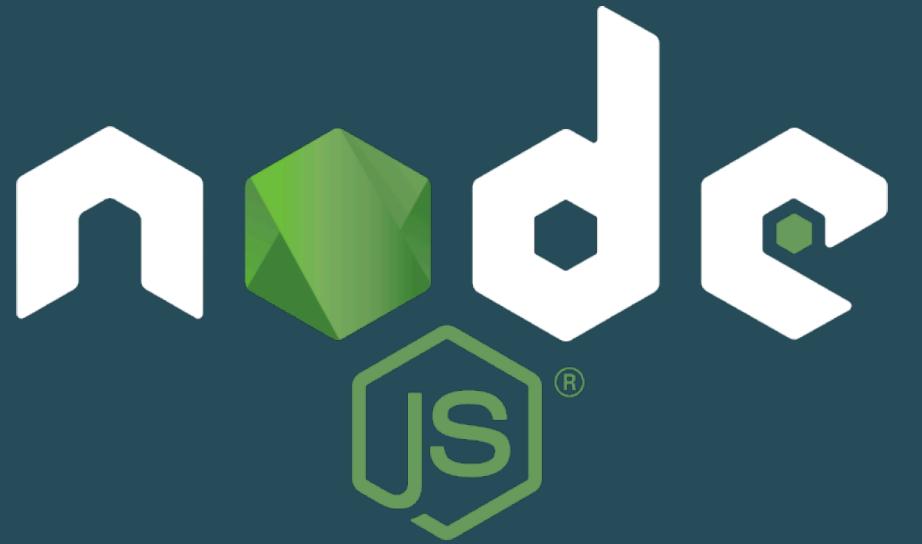
- Use await to tell JS to wait for a fetch call to finish.
- When using await you must tell JS that here goes some asynchronous code by wrapping it in an async function.

```
function PostsPage() {
  const [posts, setPosts] = useState([]);

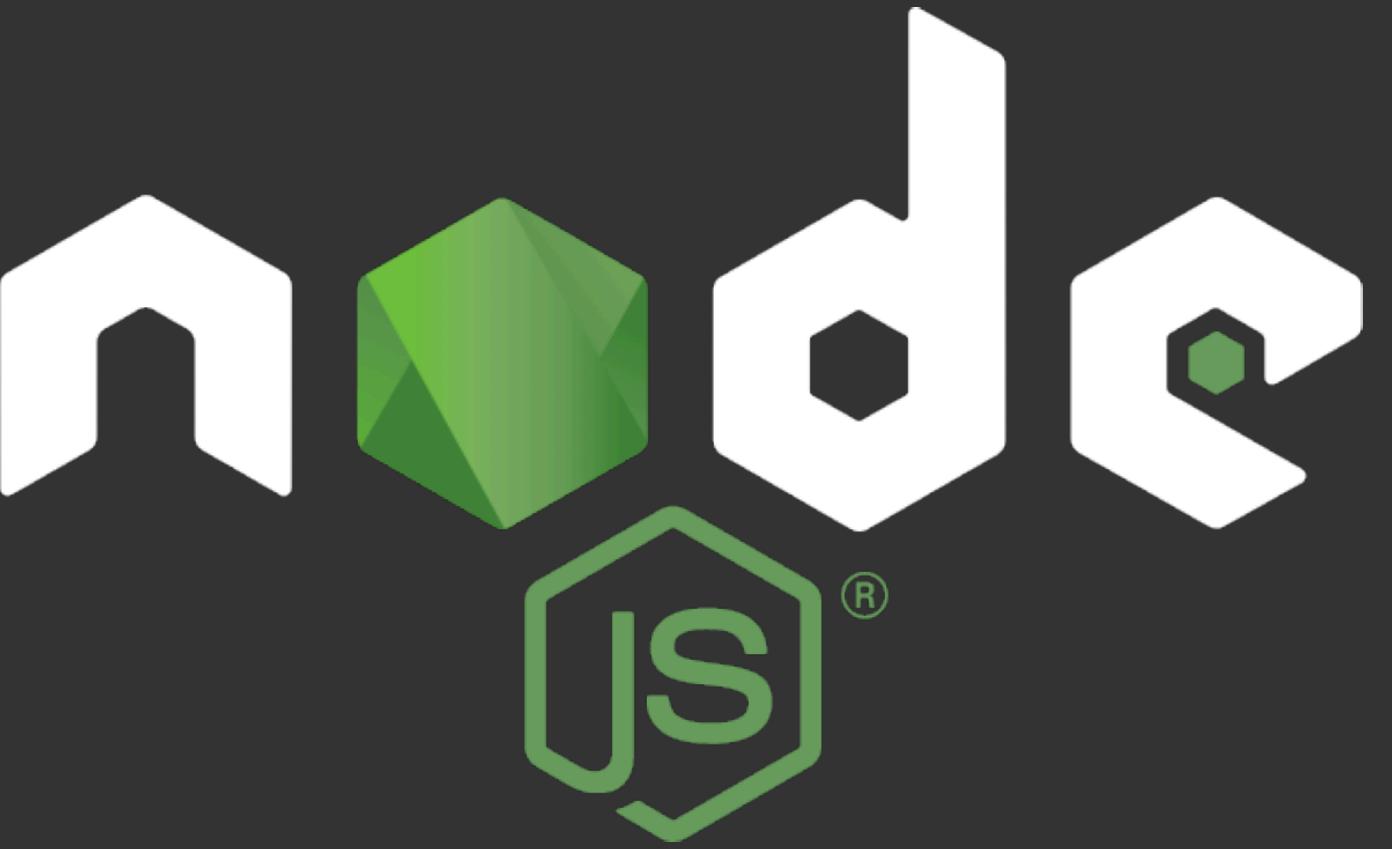
  useEffect(() => {
    async function getPosts() {
      const url = "https://raw.githubusercontent.com";
      const response = await fetch(url);
      const data = await response.json();
      setPosts(data);
    }
    getPosts();
  }, []);

  return (
    <section className="page">
      <h1>Posts</h1>
      <section className="grid-container">
        {posts.map(post => (
          <PostItem post={post} key={post.id} />
        ))}
      </section>
    </section>
  );
}
```

Builds tools,
Package Managers,
Compilers, Transpilers,
Module Bundlers &
Command-line Interface (CLI)



It is simply just tools!



JavaScript runtime environment
Executes JavaScript outside of the browser

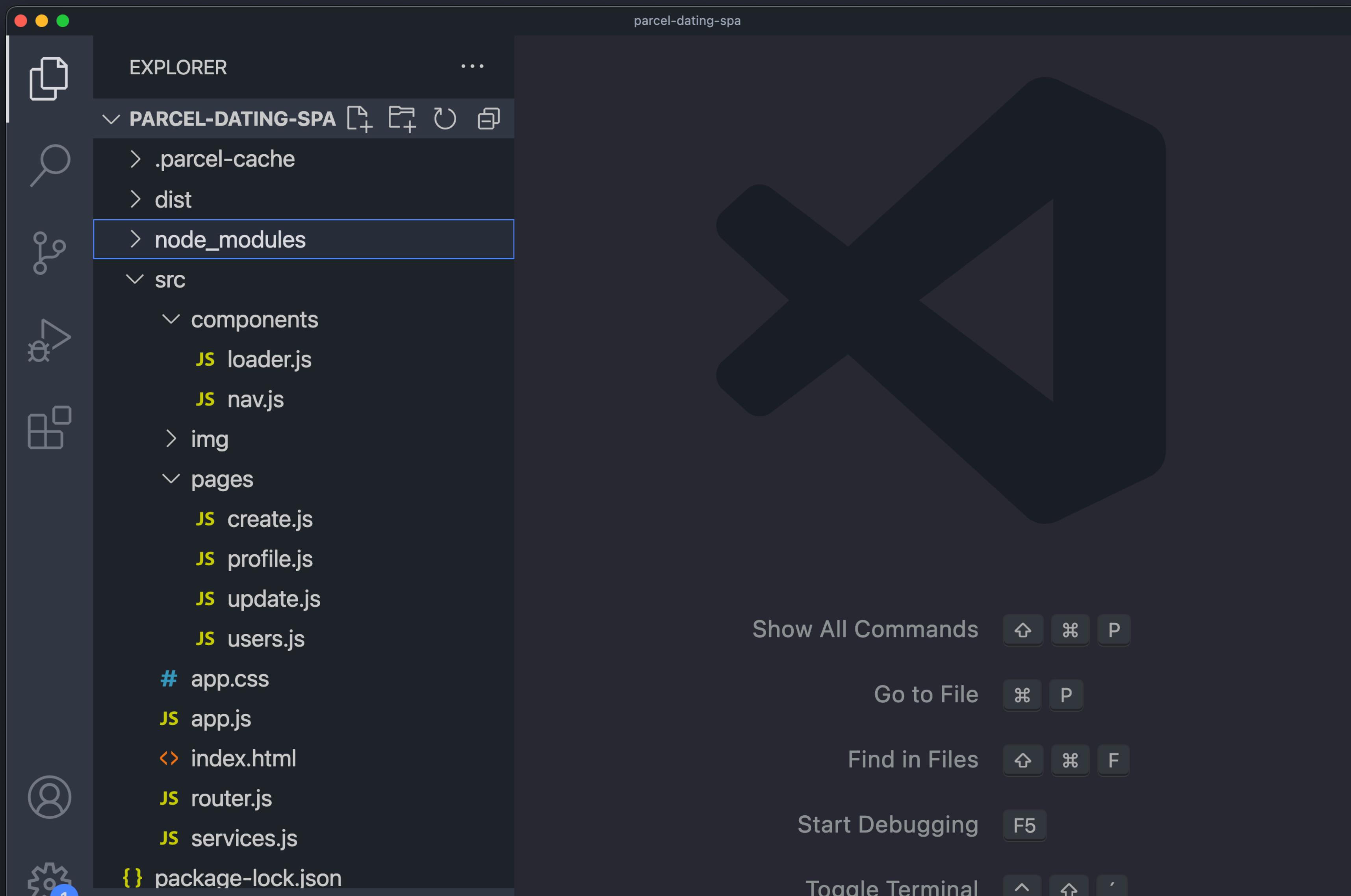
Npm is a package manager for node.js
packages - or modules



Node.js packages contains all the files you need for
a module.

Modules are JavaScript libraries you can include in
your project.

node_modules



npm consists of

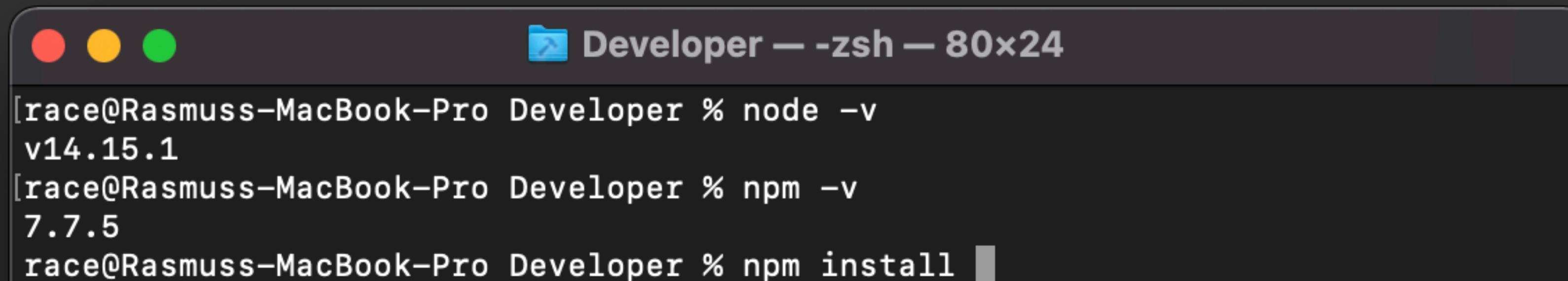
1. Website to discover packages
2. Command line interface (cli)
3. A registry - database with JS software / node modules

<https://docs.npmjs.com/about-npm/>

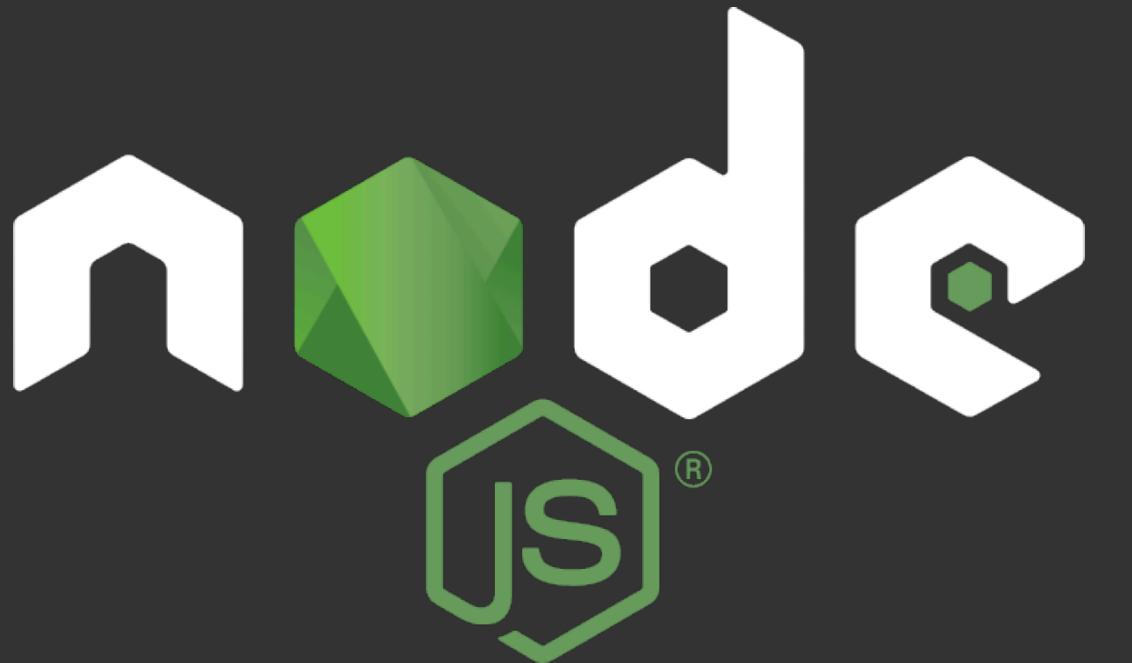
Command Line Interface

... a program that allows users to type text commands to execute functions.

npm uses CLI to install software & packages.



```
[race@Rasmuss-MacBook-Pro Developer % node -v
v14.15.1
[race@Rasmuss-MacBook-Pro Developer % npm -v
7.7.5
race@Rasmuss-MacBook-Pro Developer % npm install ]
```



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...

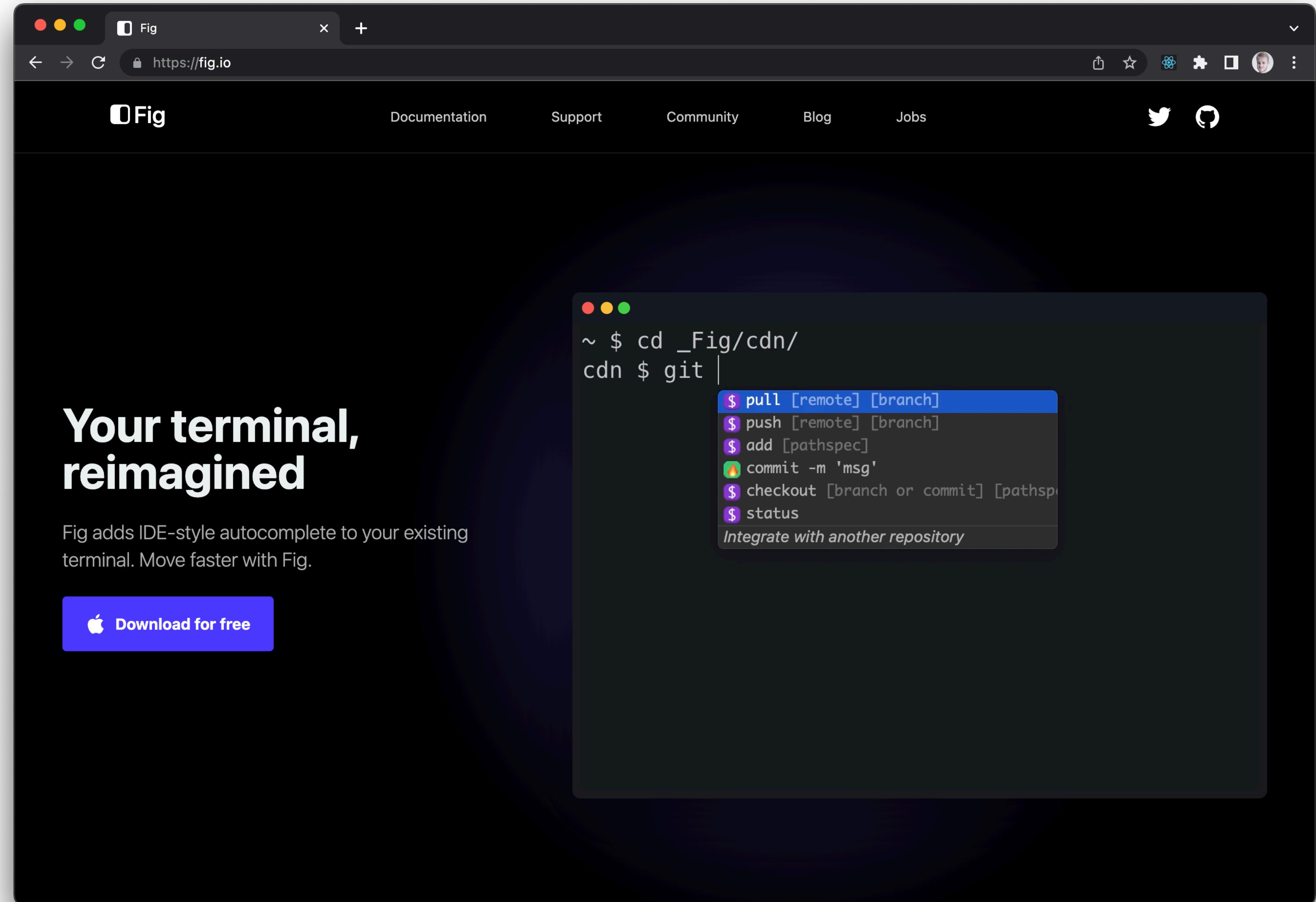
Configure Default Build Task...

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 % █
```

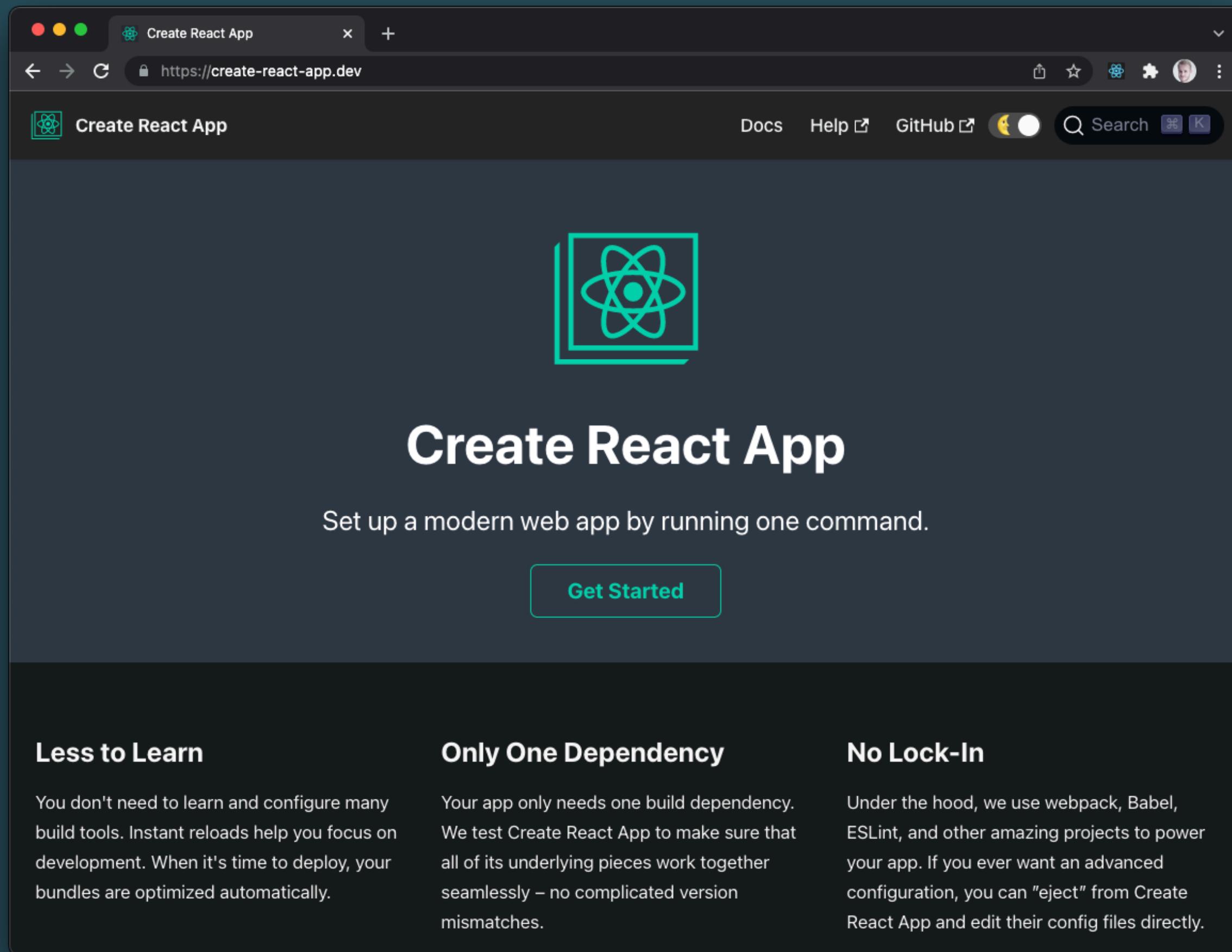
zsh + × □ └ ^ ×



<https://fig.io/>

create-react-app

Starter template for your react app



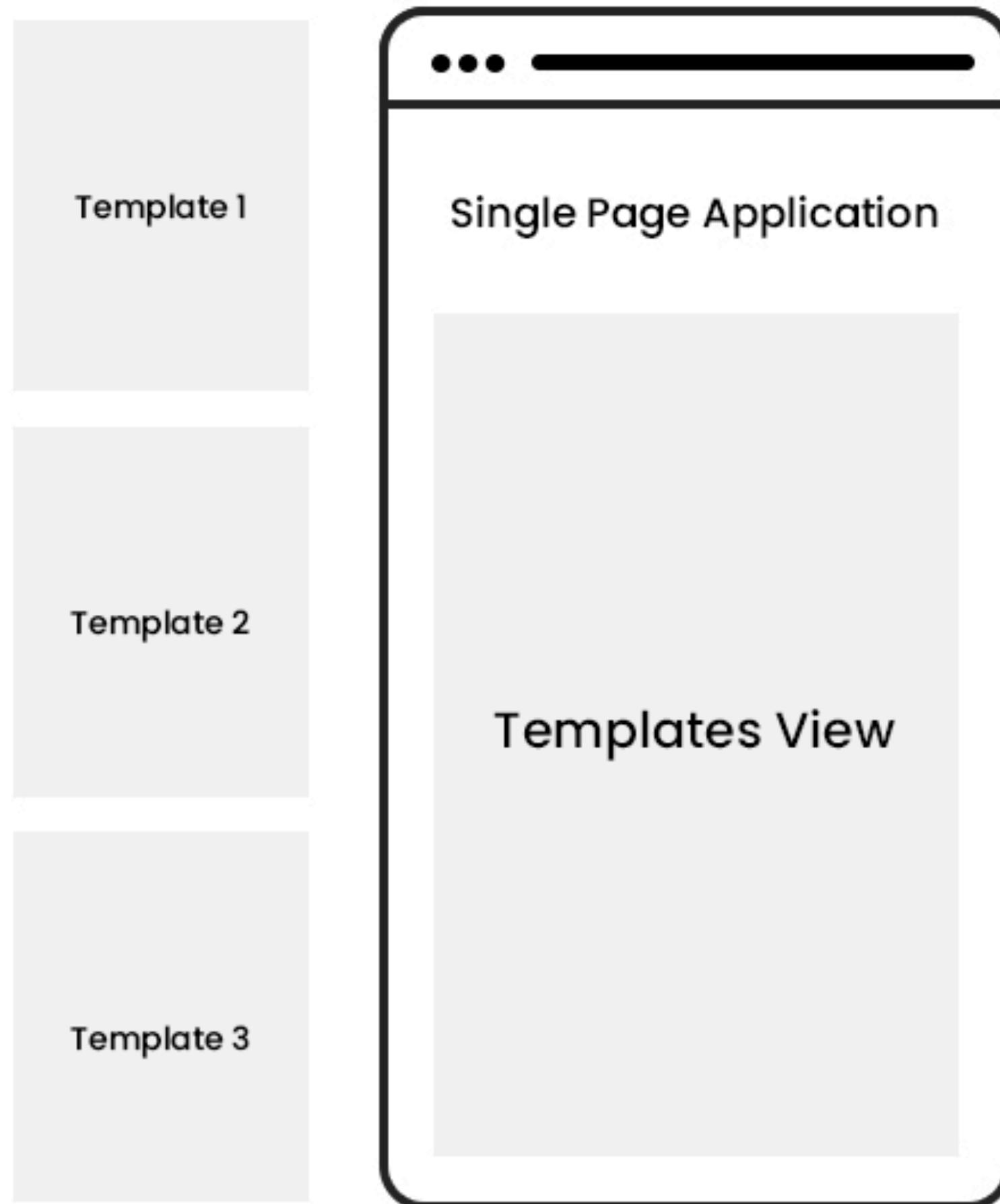
Generate create-react-app

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.”

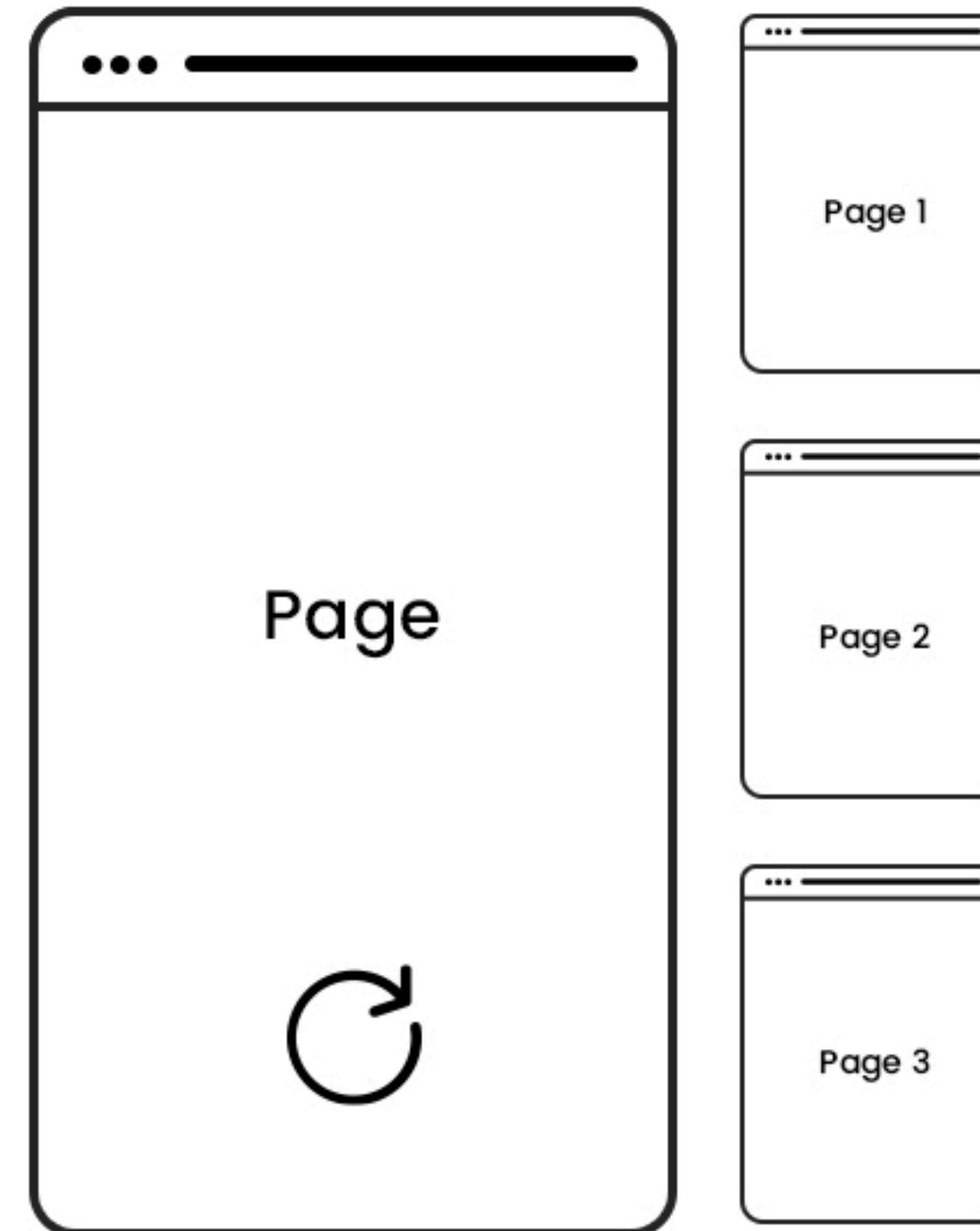
<https://reactjs.org/docs/glossary.html#single-page-application>

SPA



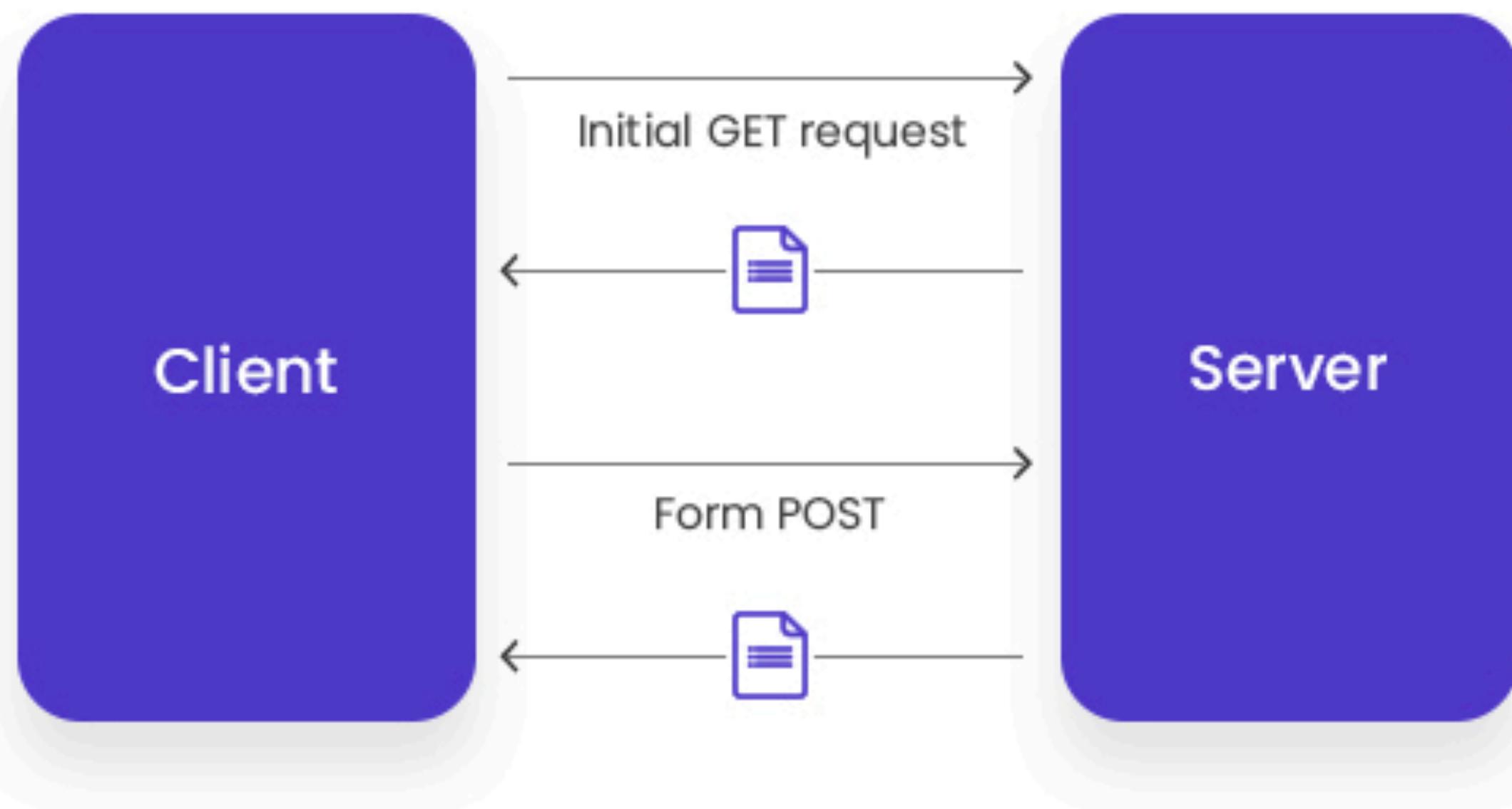
No page refresh on request

MPA

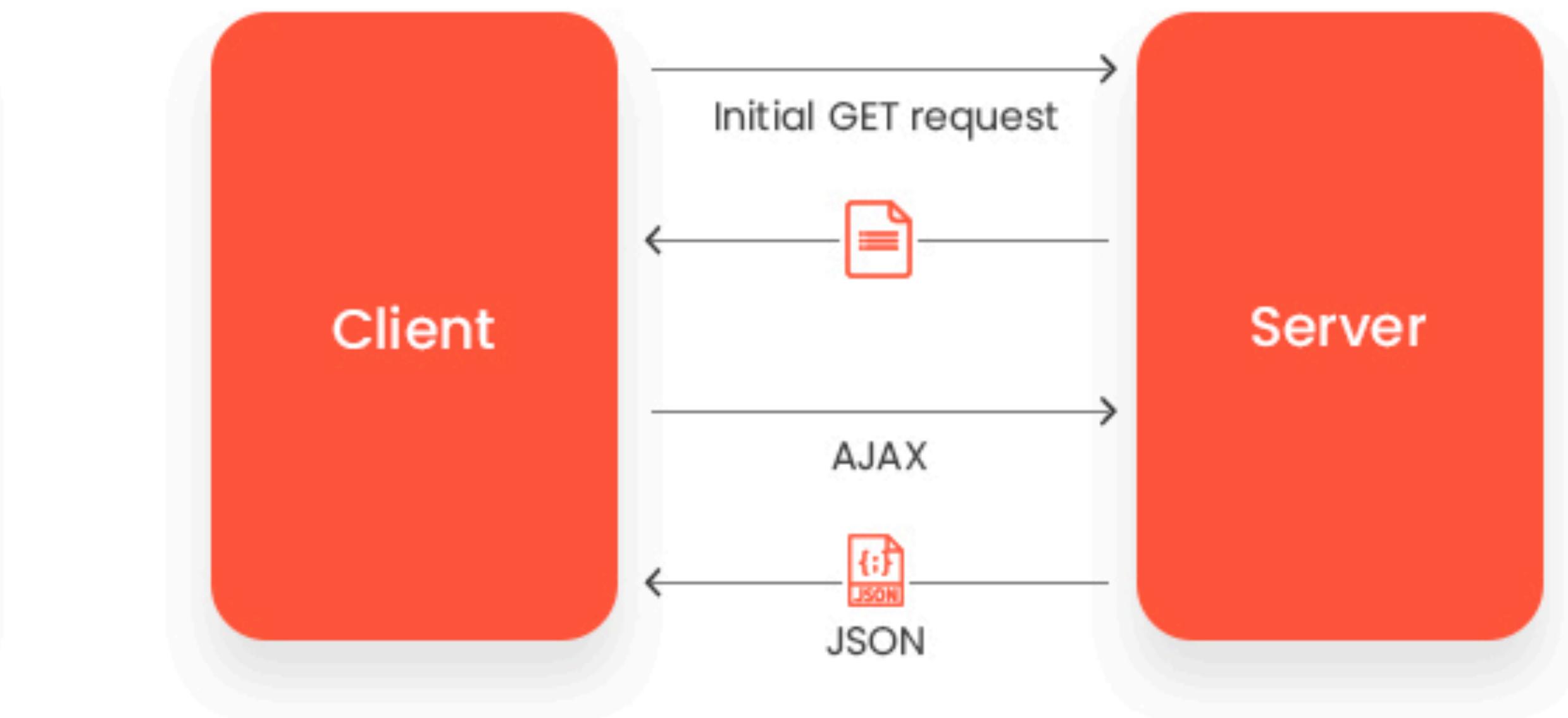


Whole page refresh on request

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?

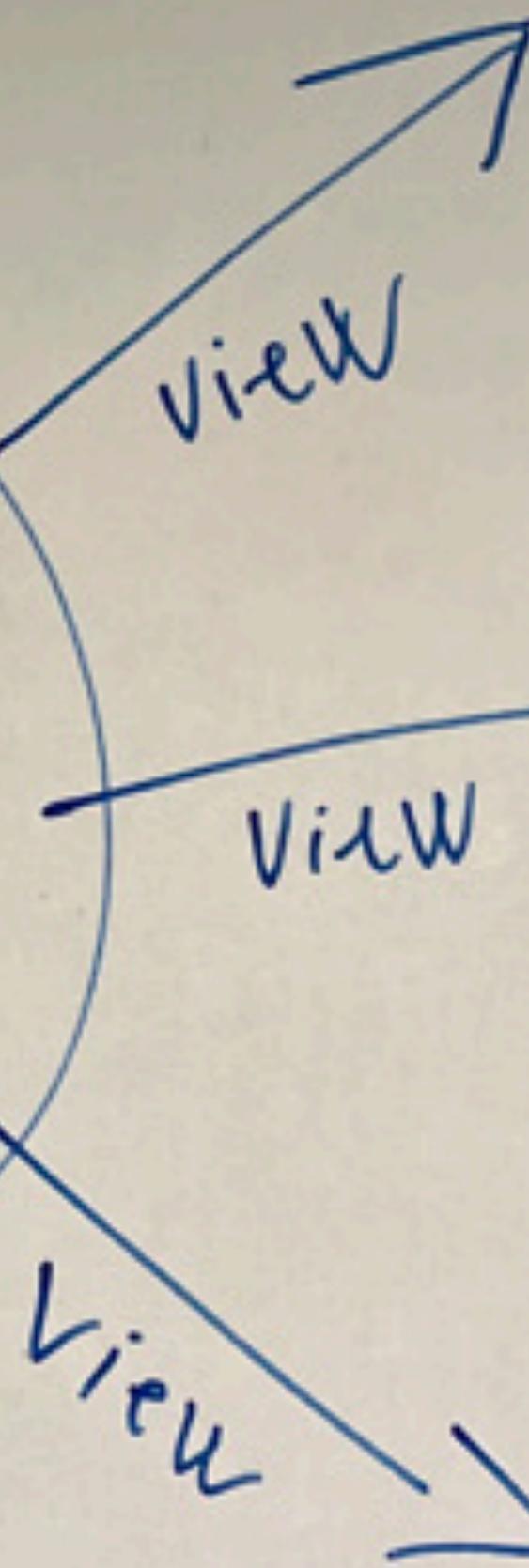
“It is the piece of software in charge to organize the states of the application, switching between different views.”

It's a key component in Single Page Applications.



/create
route

Router
(router.js)



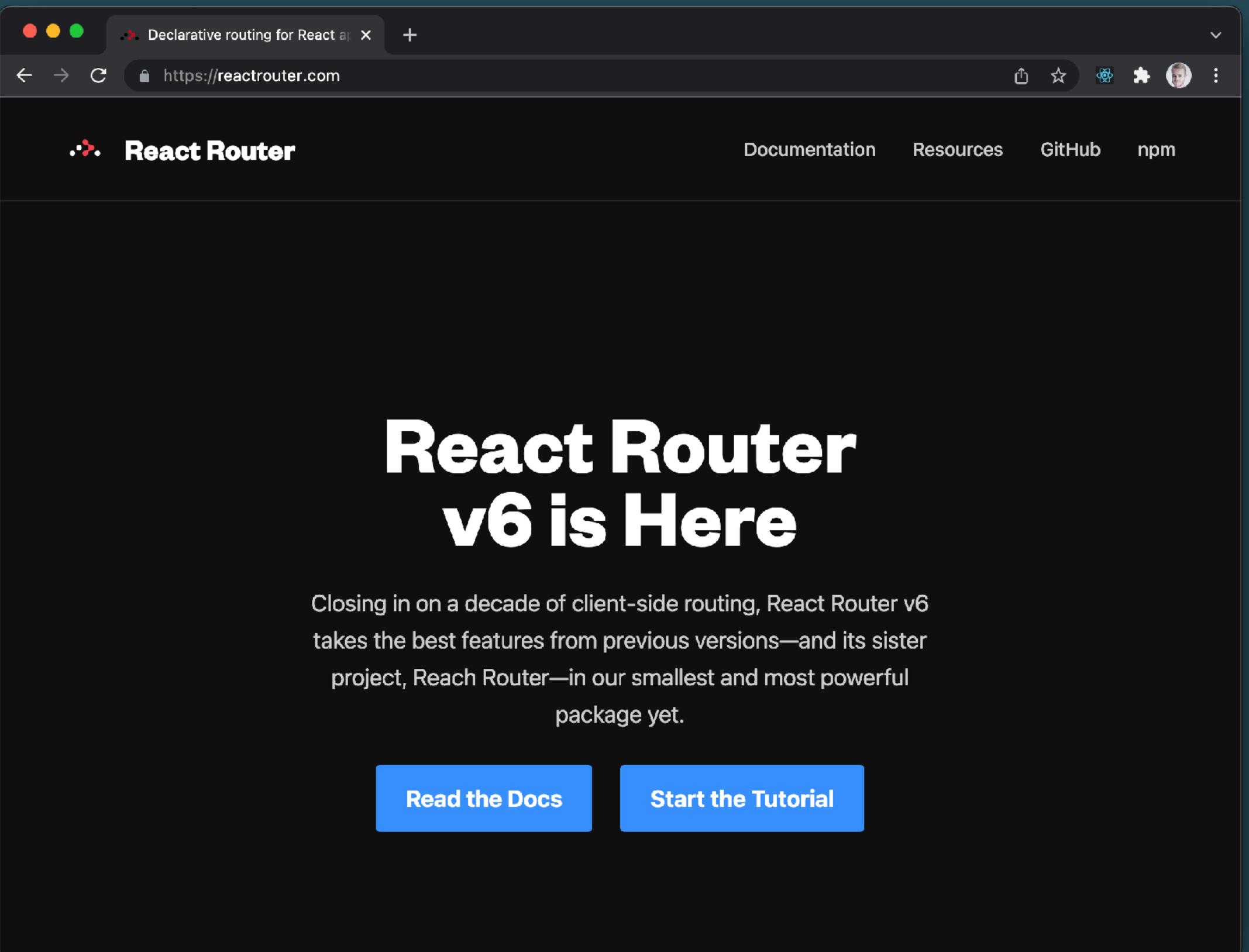
users
page

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](#)

