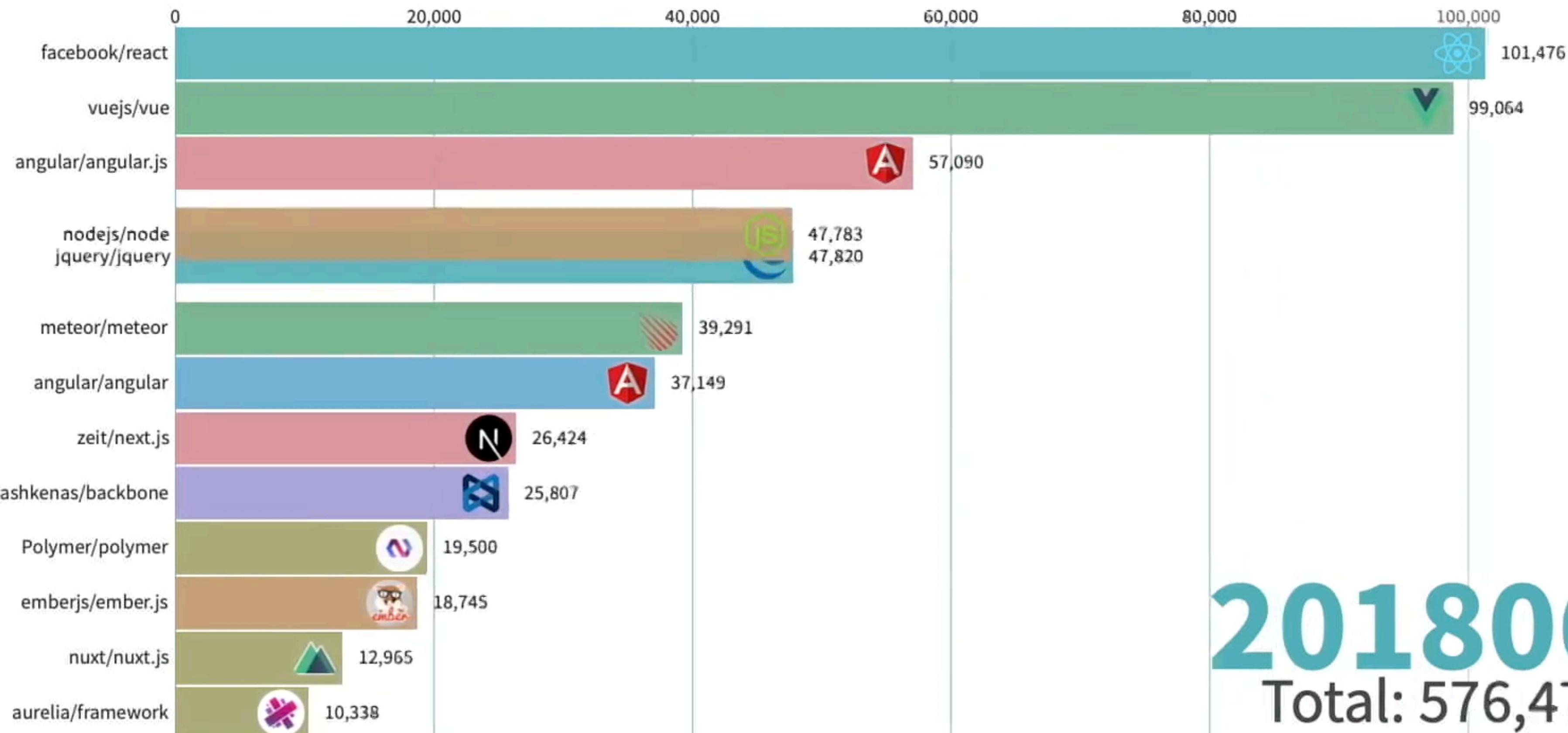


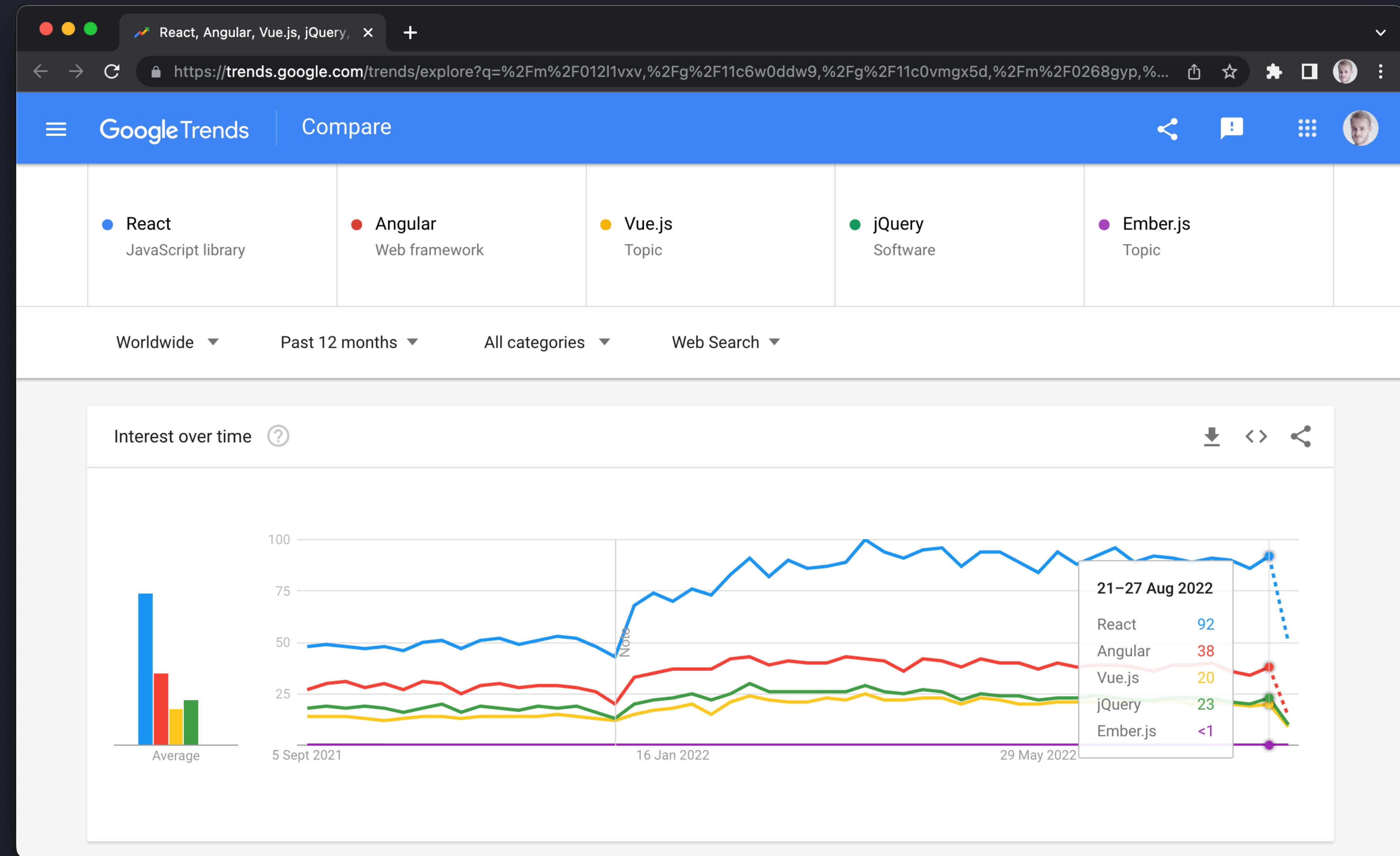
## Most popular JavaScript frameworks

2012 - 2019



**201806**  
Total: 576,471

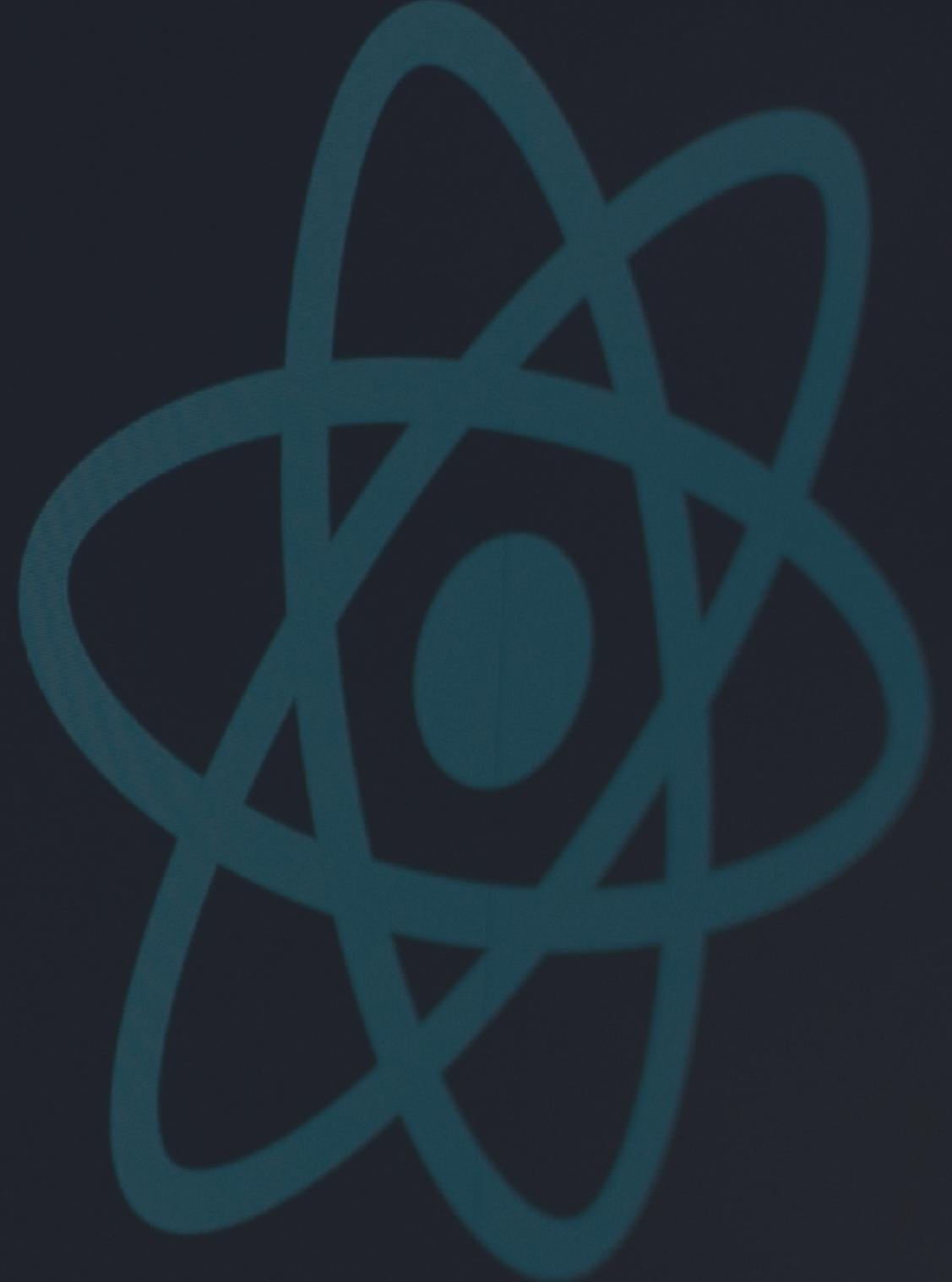
Source: GitHub Archive



# React

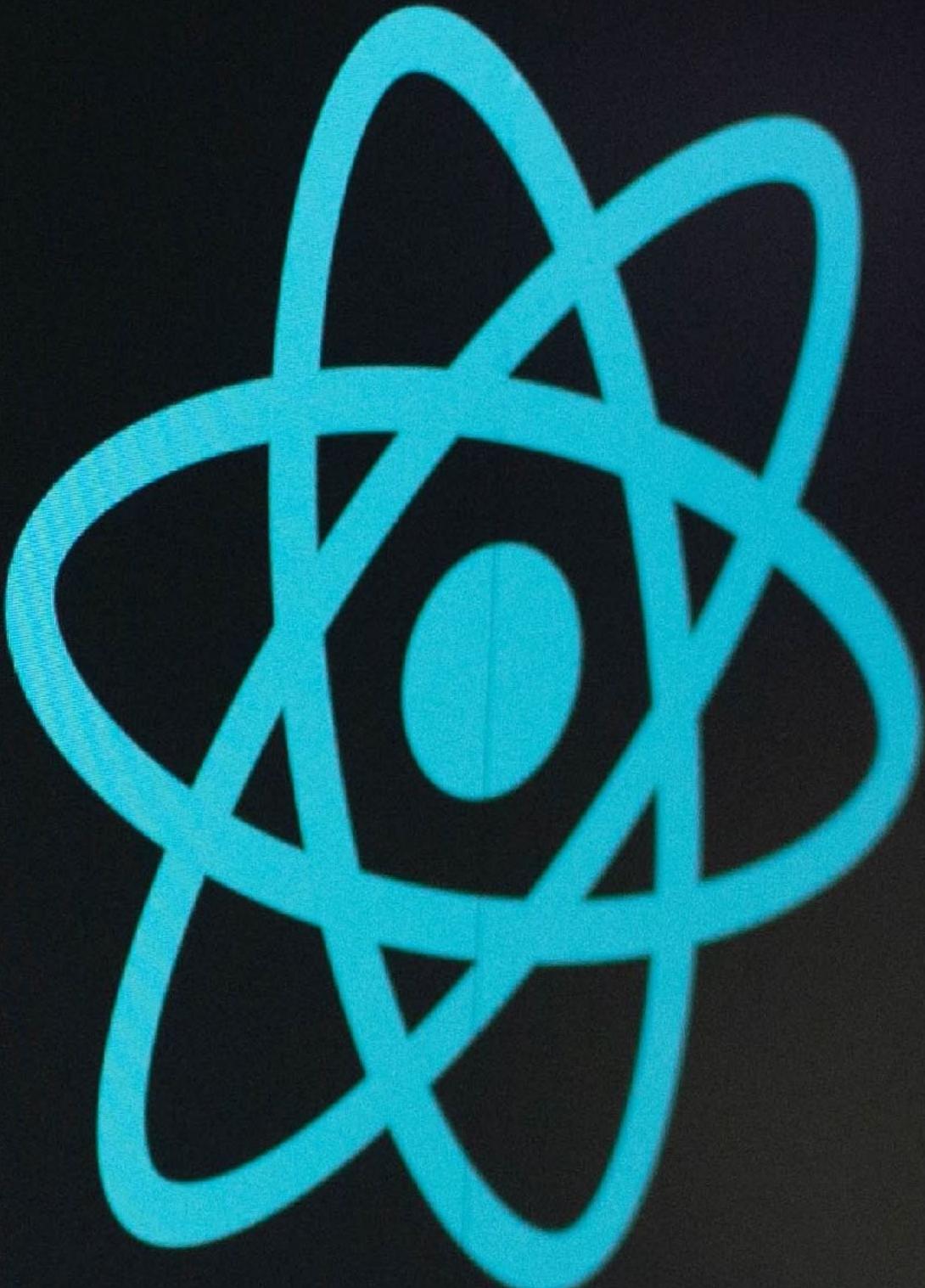
## Frontend Programming

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



# Purpose

- Introduction to React.
- Gain an understanding of React Core Concepts.
- Hands-on with React CRUD SPA.



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

# React CRUD

React Firebase REST Post App https://race-rest.web.app

POSTS CREATE

 Morten Algy Bonderup  
Senior Lecturer



**Qui est esse**  
Est rerum tempore vitae sequi sint nihil reprehenderit dolor beatae ea dolores neque fugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis qui aperiam non debitis possimus qui neque nisi nulla

 Dan Okkels Brendstrup  
Lecturer



**Consequuntur deleniti eos quia temporibus ab aliquid at**  
Voluptatem cumque tenetur consequatur expedita ipsum nemo quia explicabo aut eum minima consequatur tempore cumque quae est et et in consequuntur voluptatem voluptates aut

 Kim Elkjær Marcher-Jepsen  
Senior Lecturer



**At nam consequatur ea labore ea harum**  
Cupiditate quo est a modi nesciunt soluta ipsa voluptas error itaque dicta in autem qui minus magnam et distinctio eum accusamus ratione error aut

 Birgitte Kirk Iversen  
Senior Lecturer



 Jes Arbov  
Lecturer



 Maria Louise Bendixen  
Senior Lecturer



<https://race-rest.web.app/>

# Good to know...

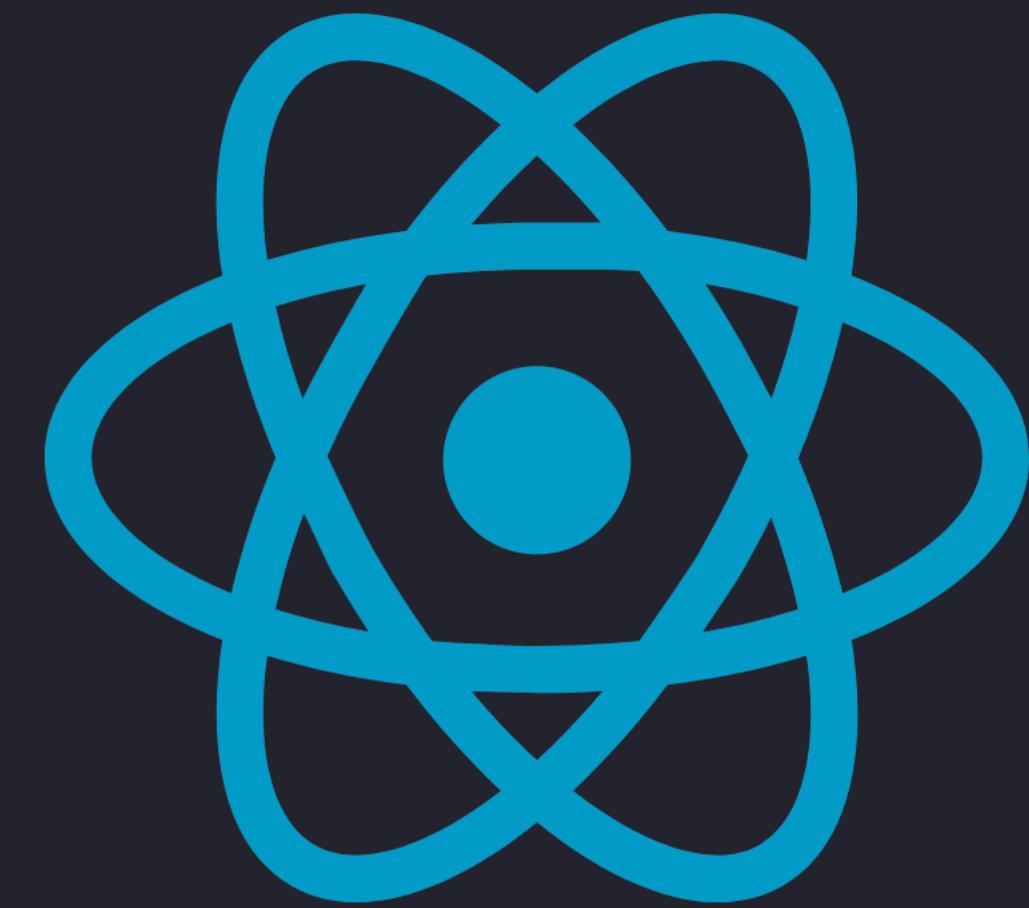
HTML & CSS Basics, CSS Layouts, Grids & Flexbox, Selectors  
Variables, Objects, Arrays, Loops, Functions, DOM-  
Manipulation, Template String, ES6+ Features, fetch, API,  
JSON, CRUD, Forms & Events

- What is React?
- The Core Concepts of React
  - Function Component
  - Components, components, components
  - Props
  - React Hooks: useState & useEffect
- create-react-app
- Single Page Apps
- React Router
- React CRUD



# Exercises

- [Your First React App](#)
- [Component Page Layout](#)
- [Add Props: Your First React App](#)
- [Add Props: Page Layout](#)
- [Tryout useState](#)
- [React Fetch Posts #1](#)
- [React Fetch Posts #2](#)
- [Download & install Node.js & npm](#)
- [Generate create-react-app](#)
- [Canvas Users Case w/ React](#)
- [Create React SPA](#)
- [Boilerplate for React Post App](#)
- [More React Practice](#)

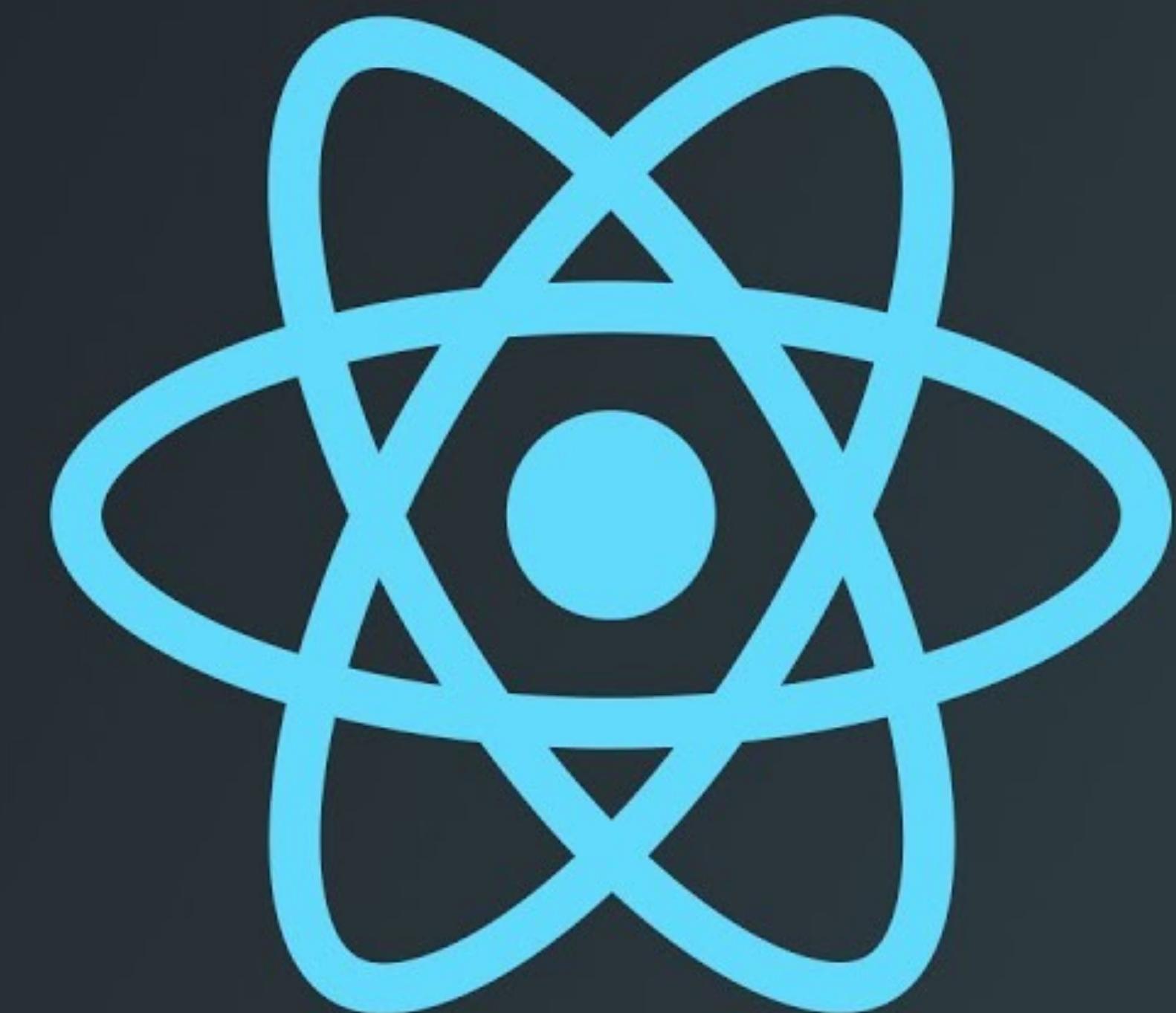


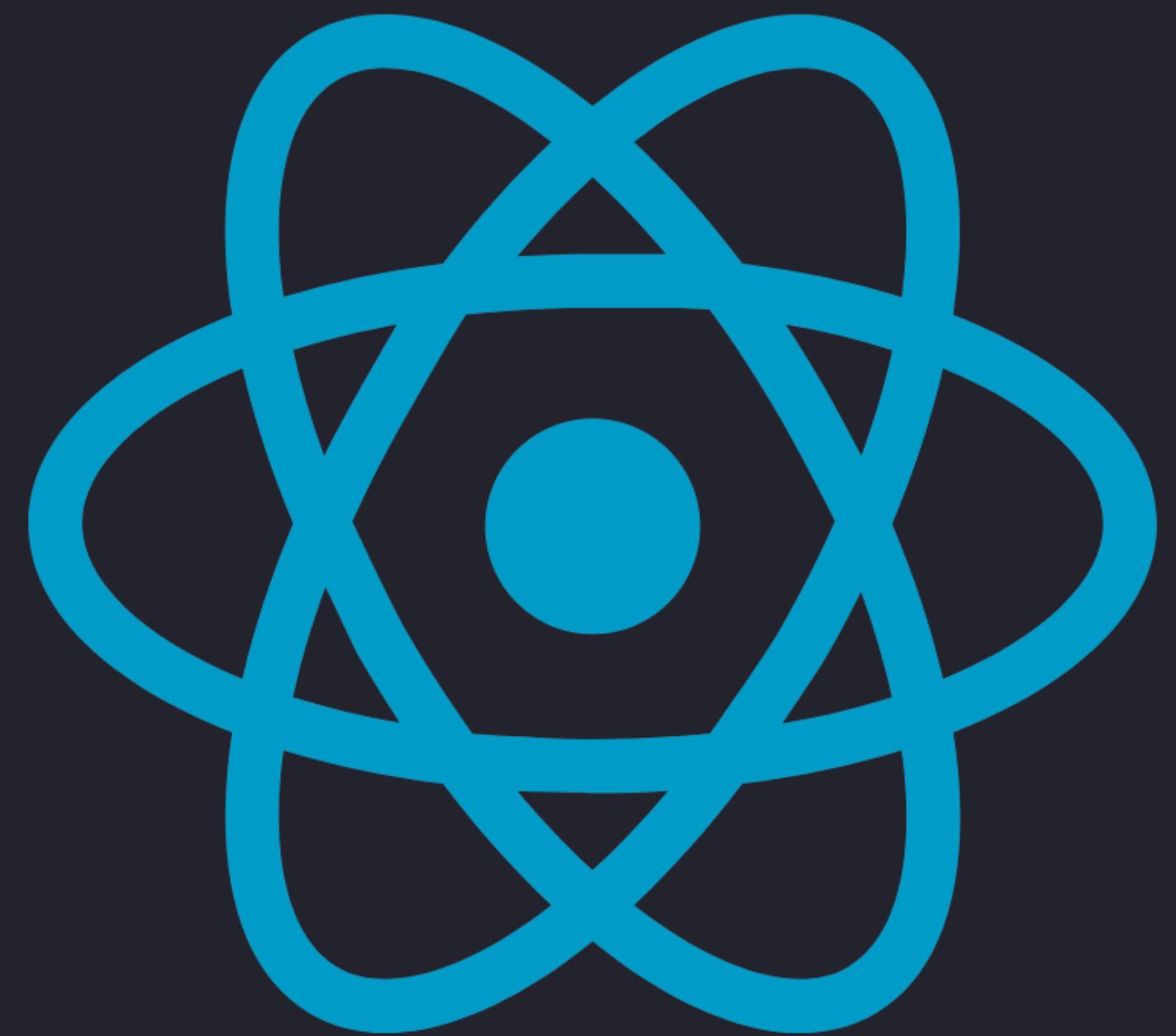
# React

A JavaScript library for building User Interfaces

**100** *SECONDS OF*

---



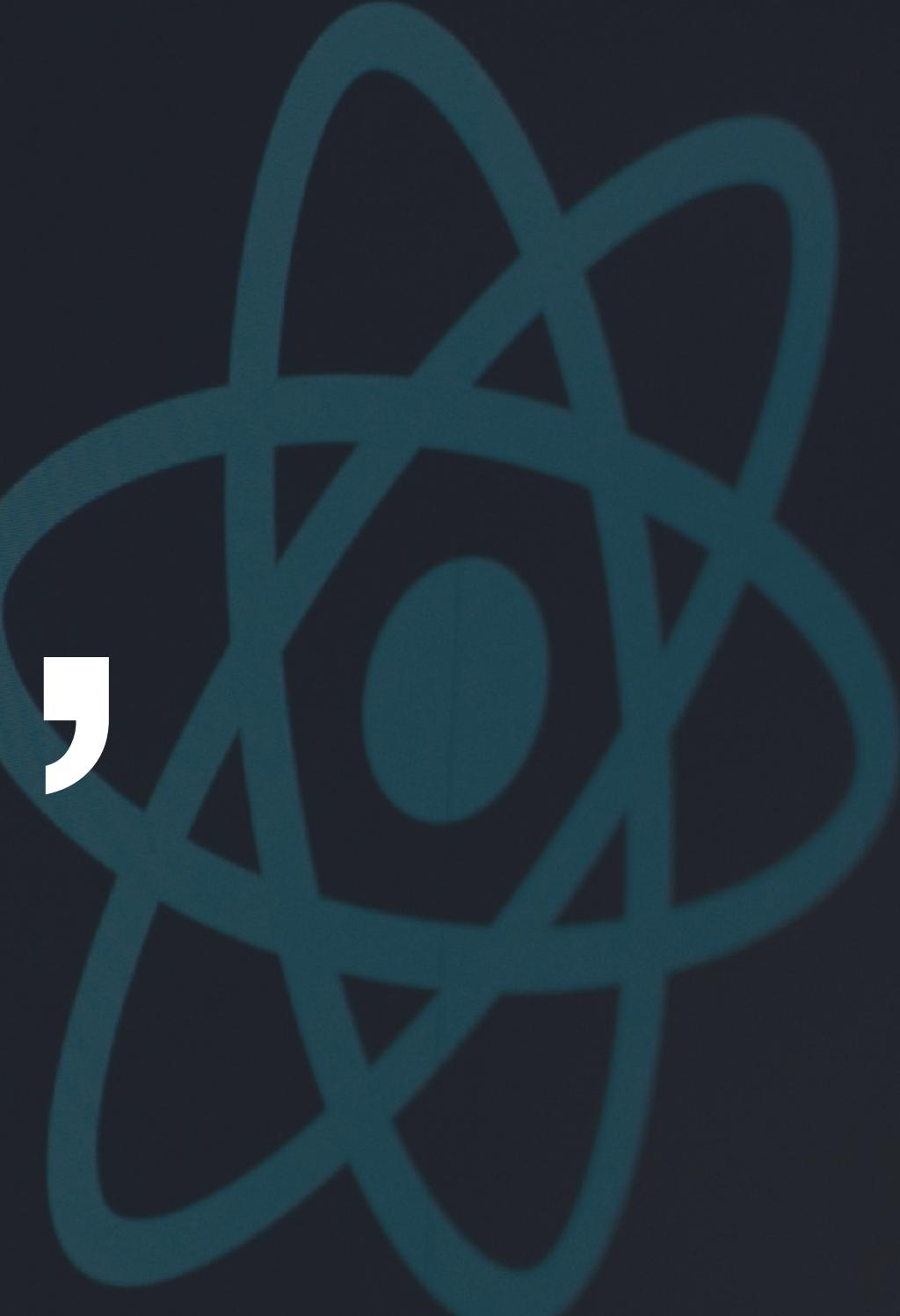


# What is React?

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

# Components, Props & States

The Core Concepts of React

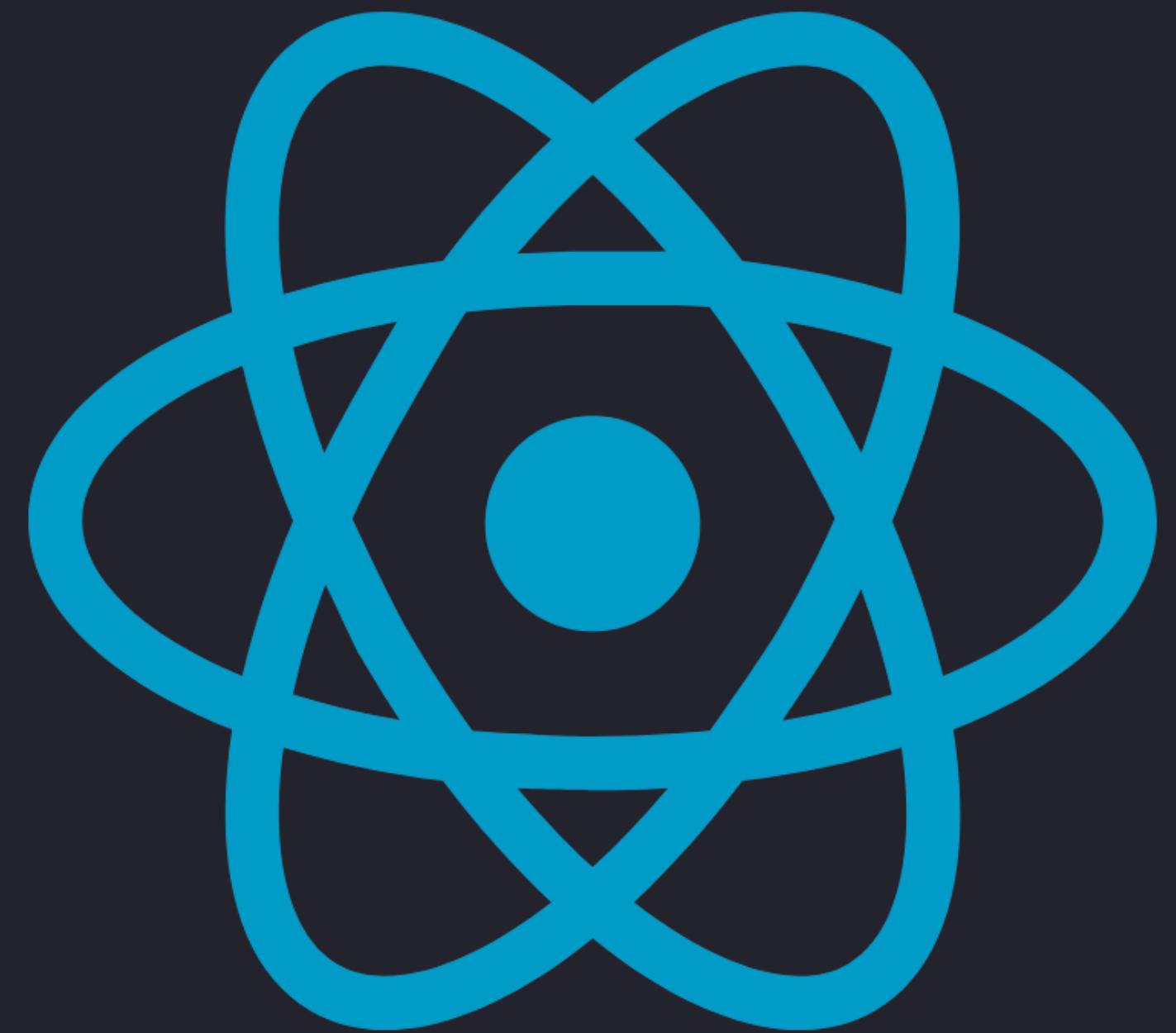


Edit src/App.js and save to reload  
Learn React

# Thinking in React

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

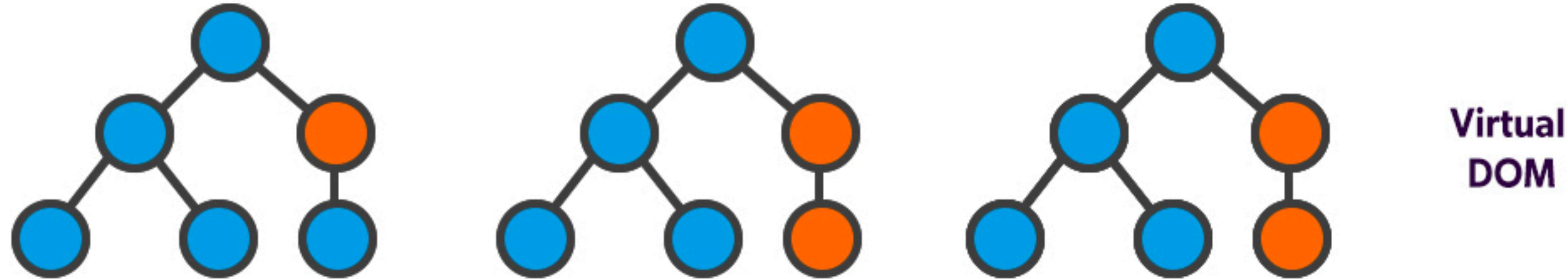
Edit src/App.js and save to reload  
Learn React



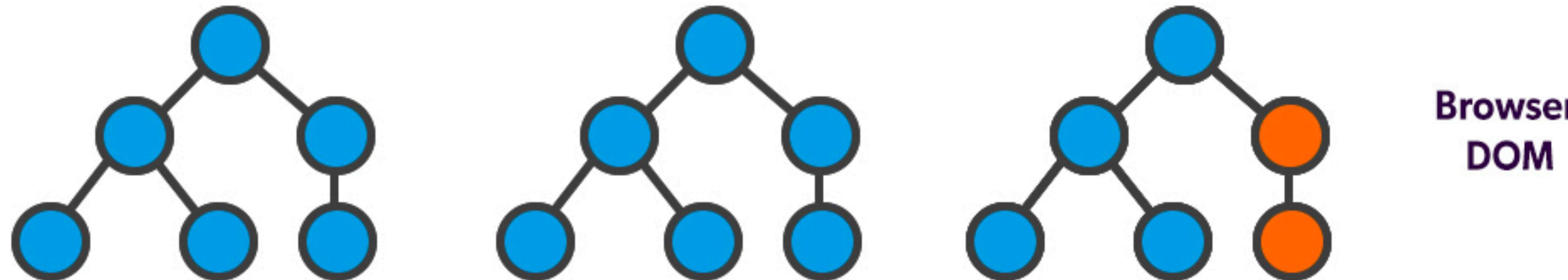
# But how?

- 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!

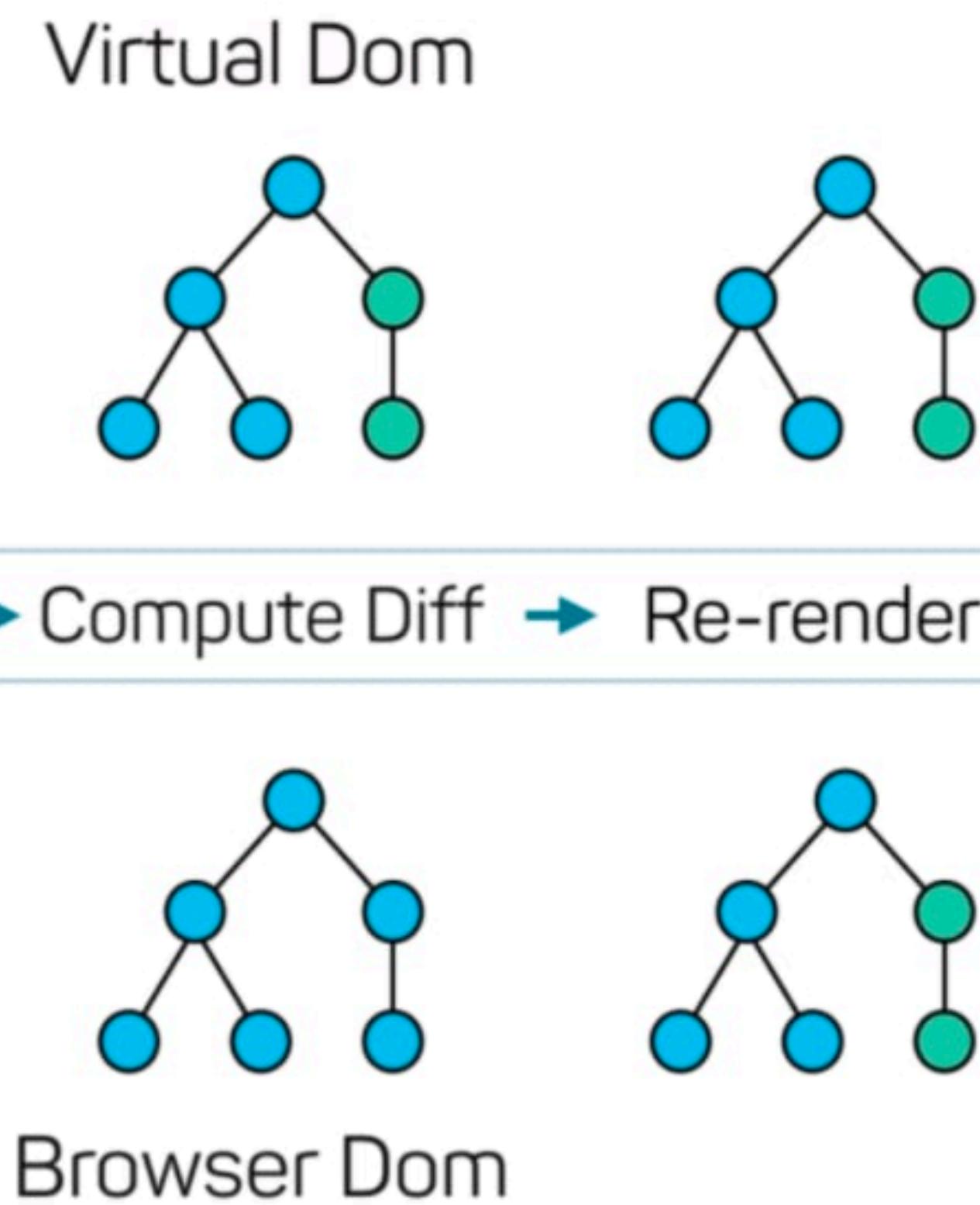
# React Virtual DOM



Statechange → Compute Diff → Re-render



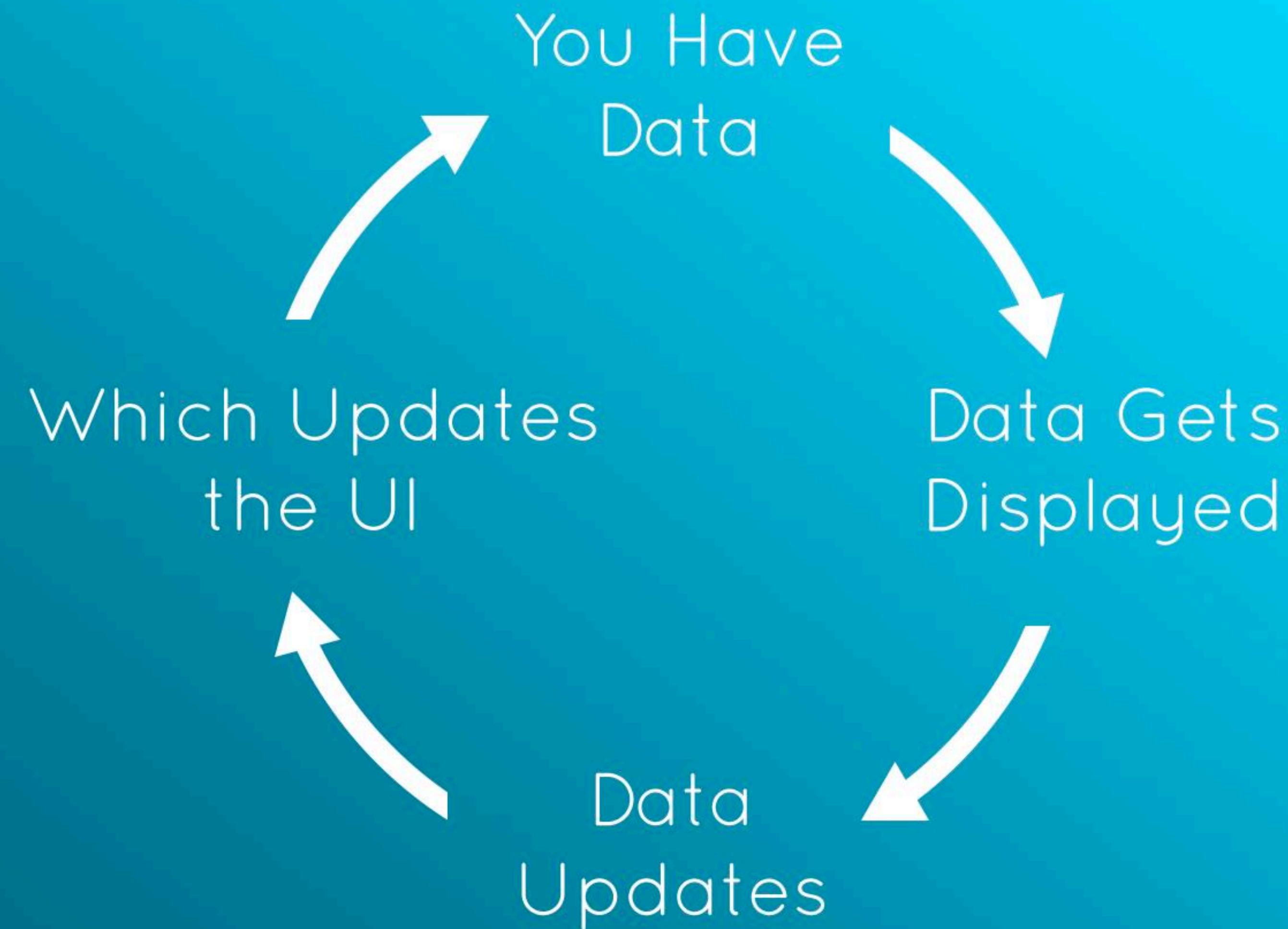
# React Virtual DOM



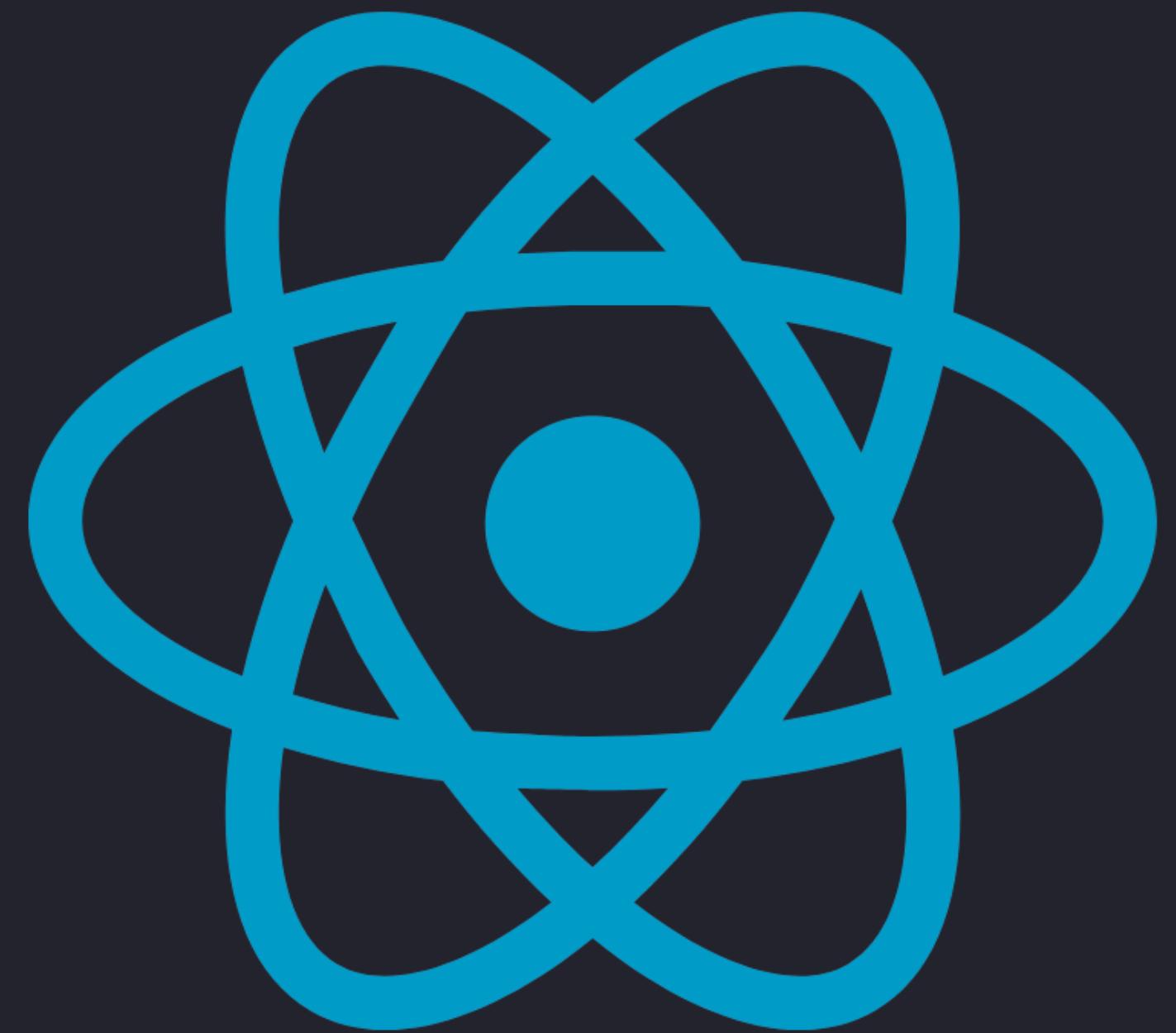
“The virtual DOM (VDOM) is a programming concept where an ideal, or “virtual”, representation of a UI is kept in memory and synced with the “real” DOM by a library such as ReactDOM.”

“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 re-rendering all elements - only the needed elements.

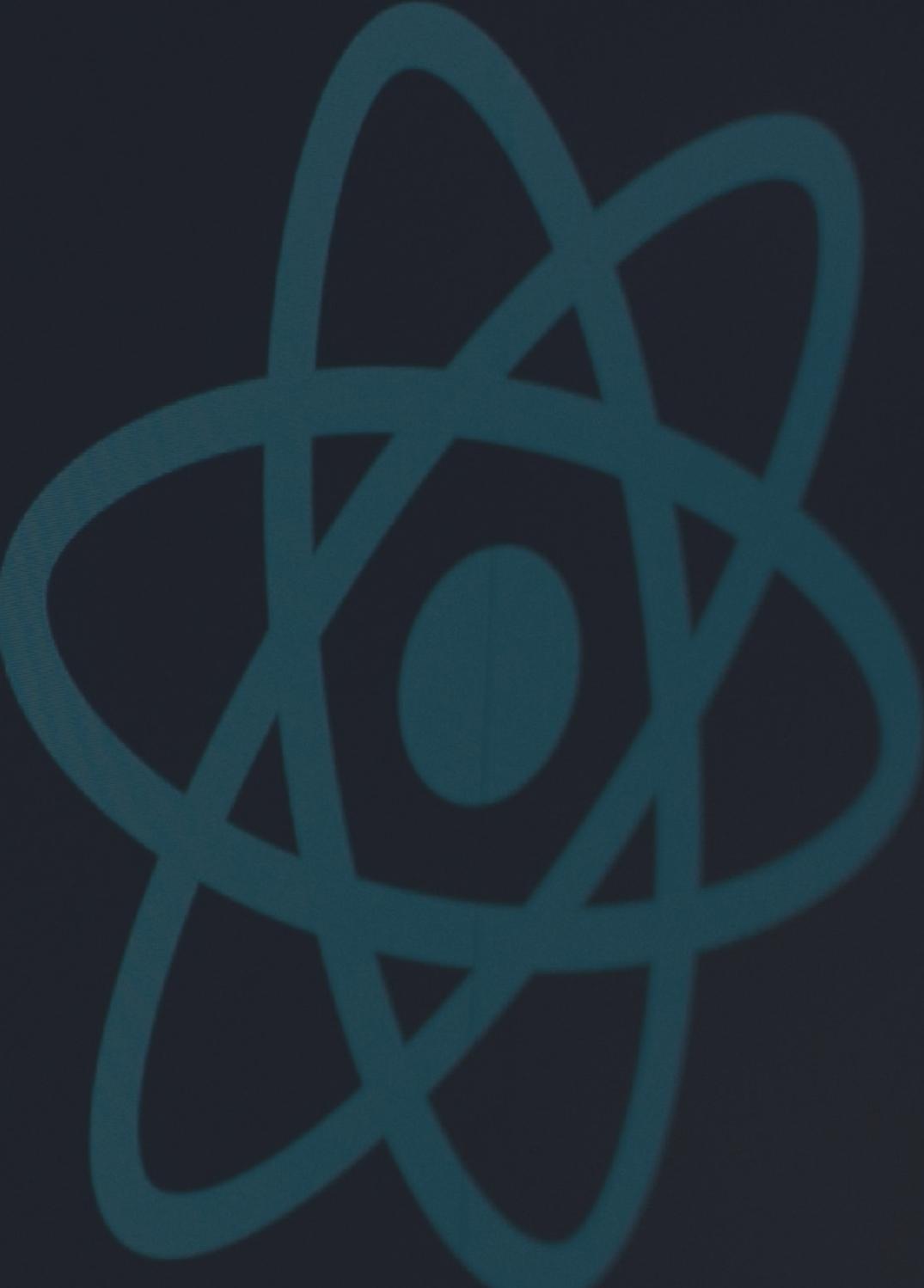
# Resources

React Resources

React Self-study

Guides & Tutorials

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



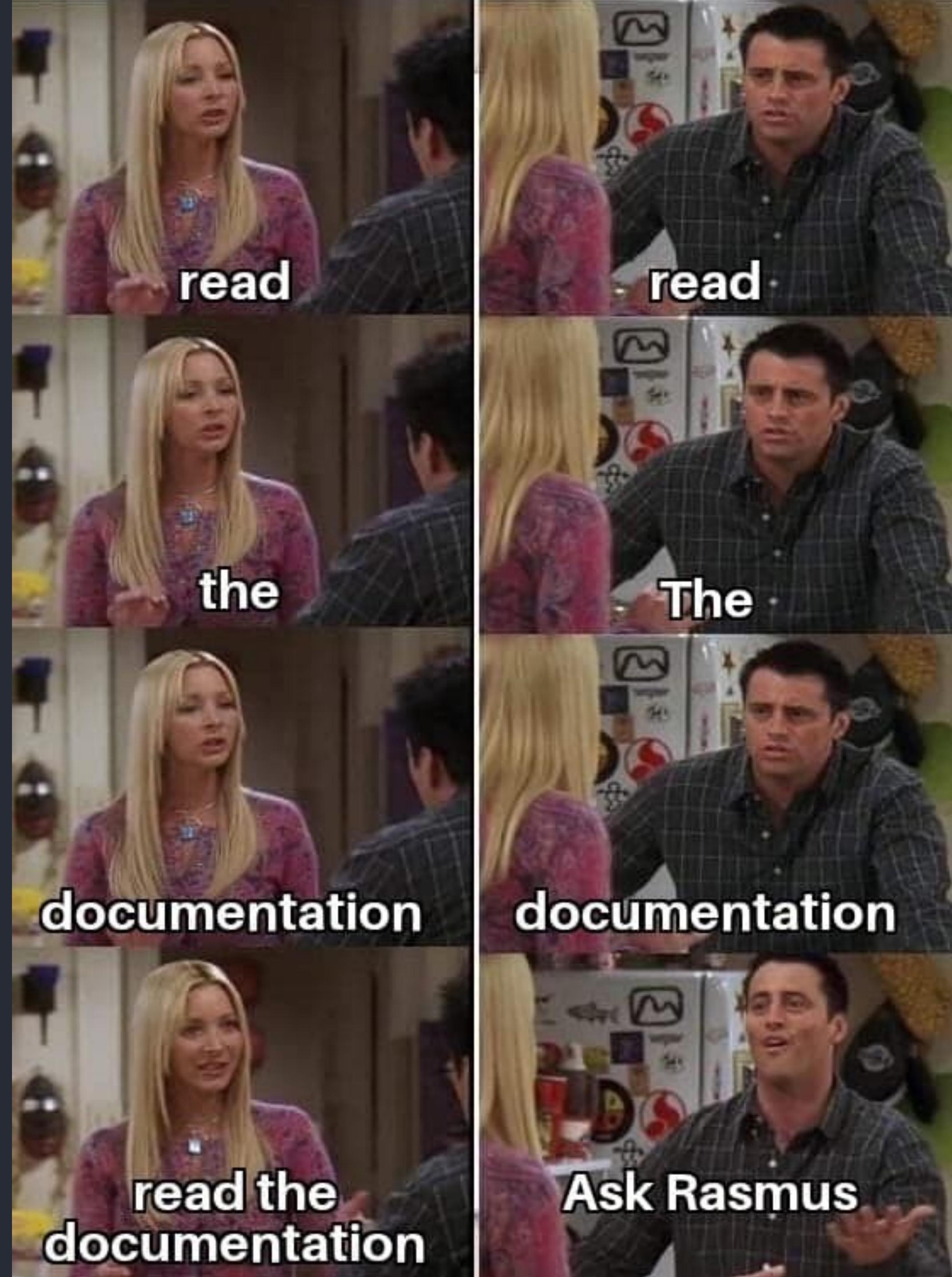
# React Beta Docs

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 text 'React Docs' with a 'BETA' badge. Below this are two main sections: 'Learn React' (describing step-by-step explanations and interactive examples) and 'API Reference' (describing 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?

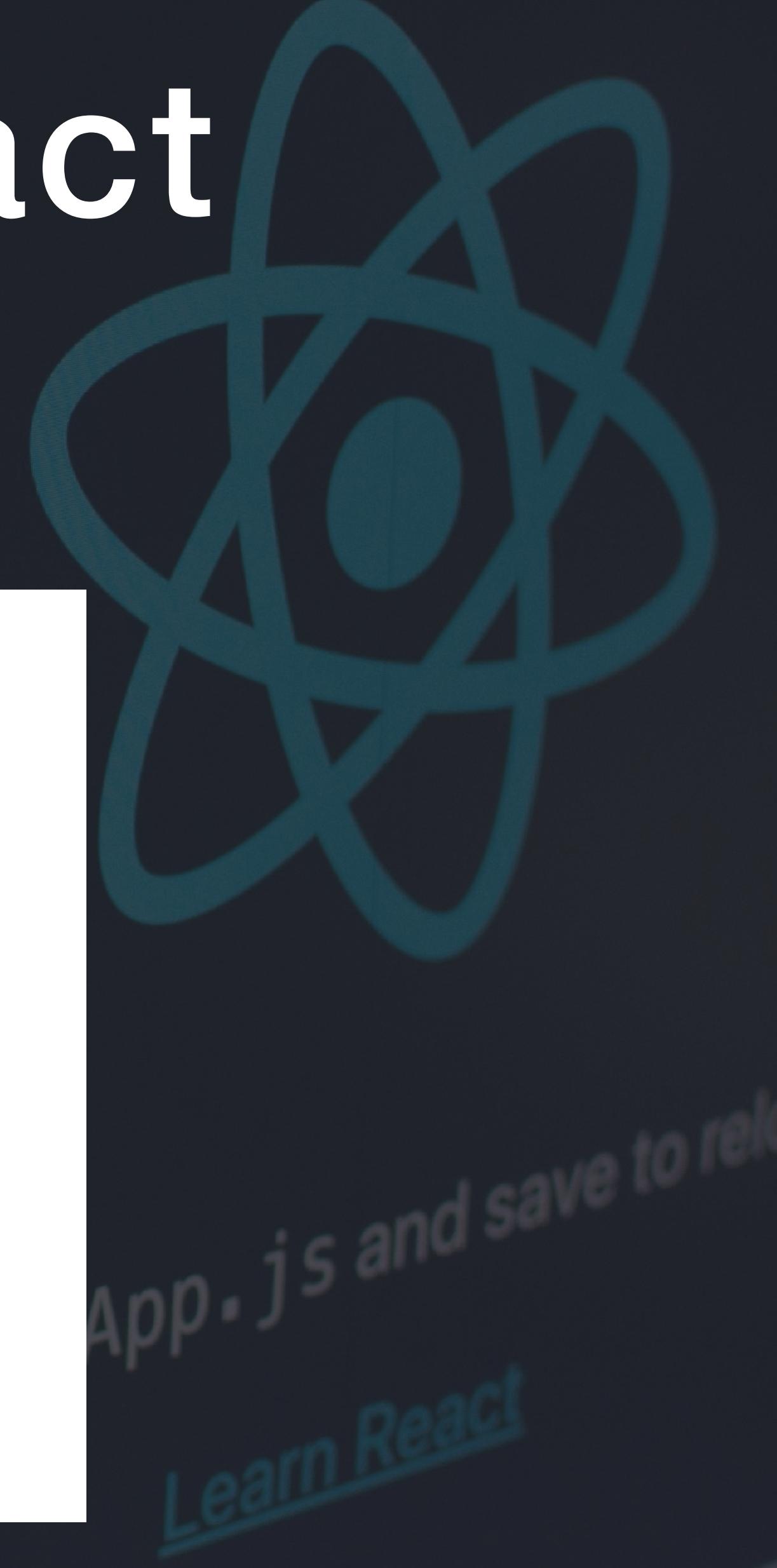
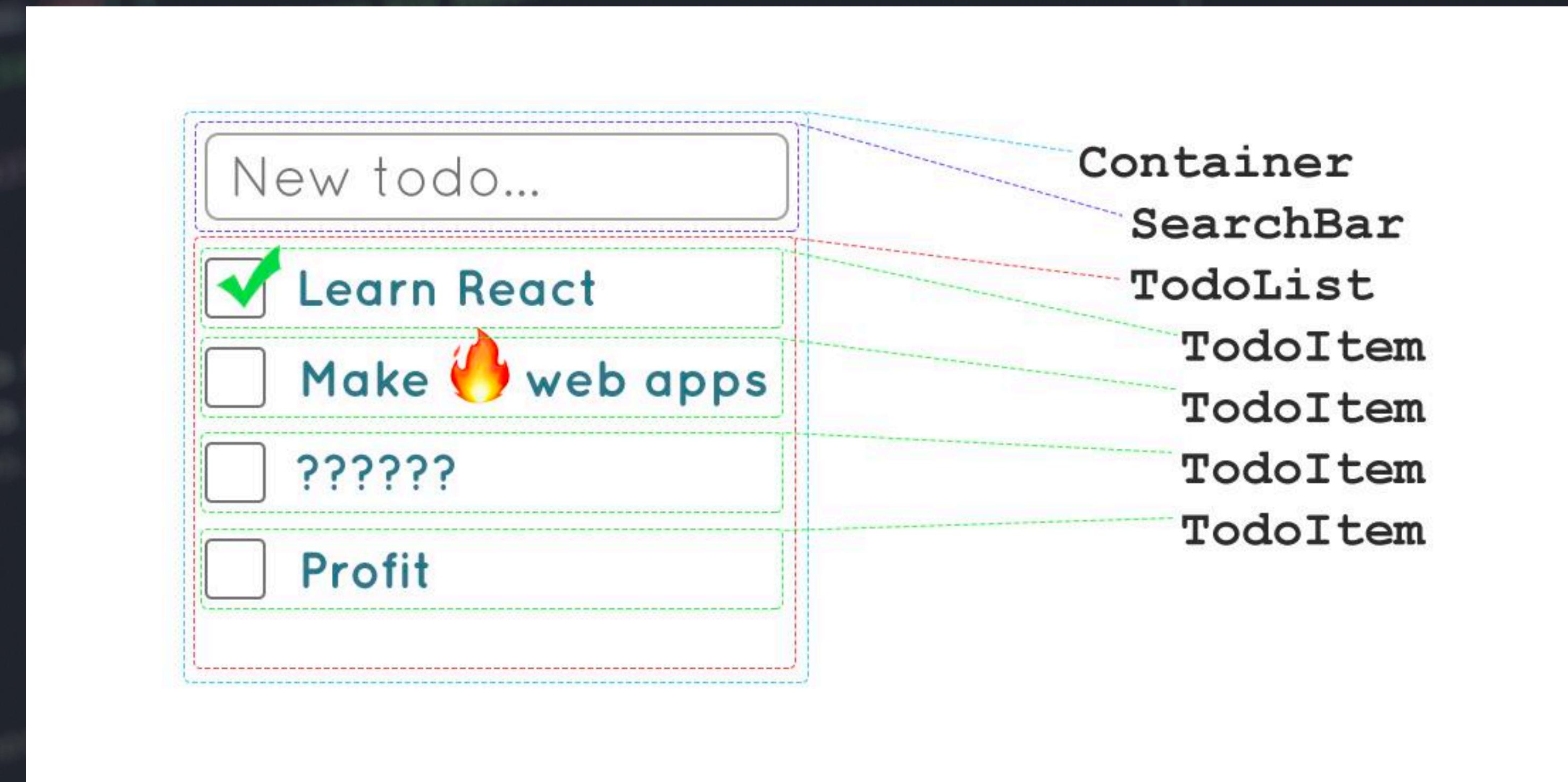
We are rewriting the React documentation with a few differences:

- 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.



# 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>  
  );  
}
```

1. Create a new react project and explore App.js and the App component.
2. 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.
3. Render Person component inside of your App component.
4. Render several Person components inside of your App component.



# 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>  
  );  
}
```

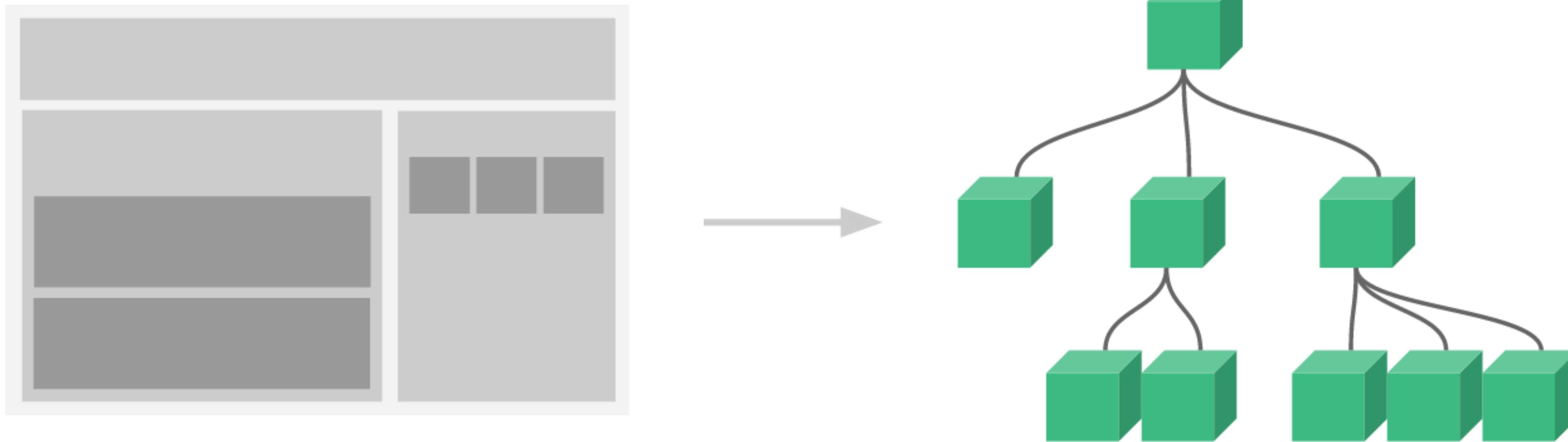
Define the component

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

Use Greeting component  
in another component

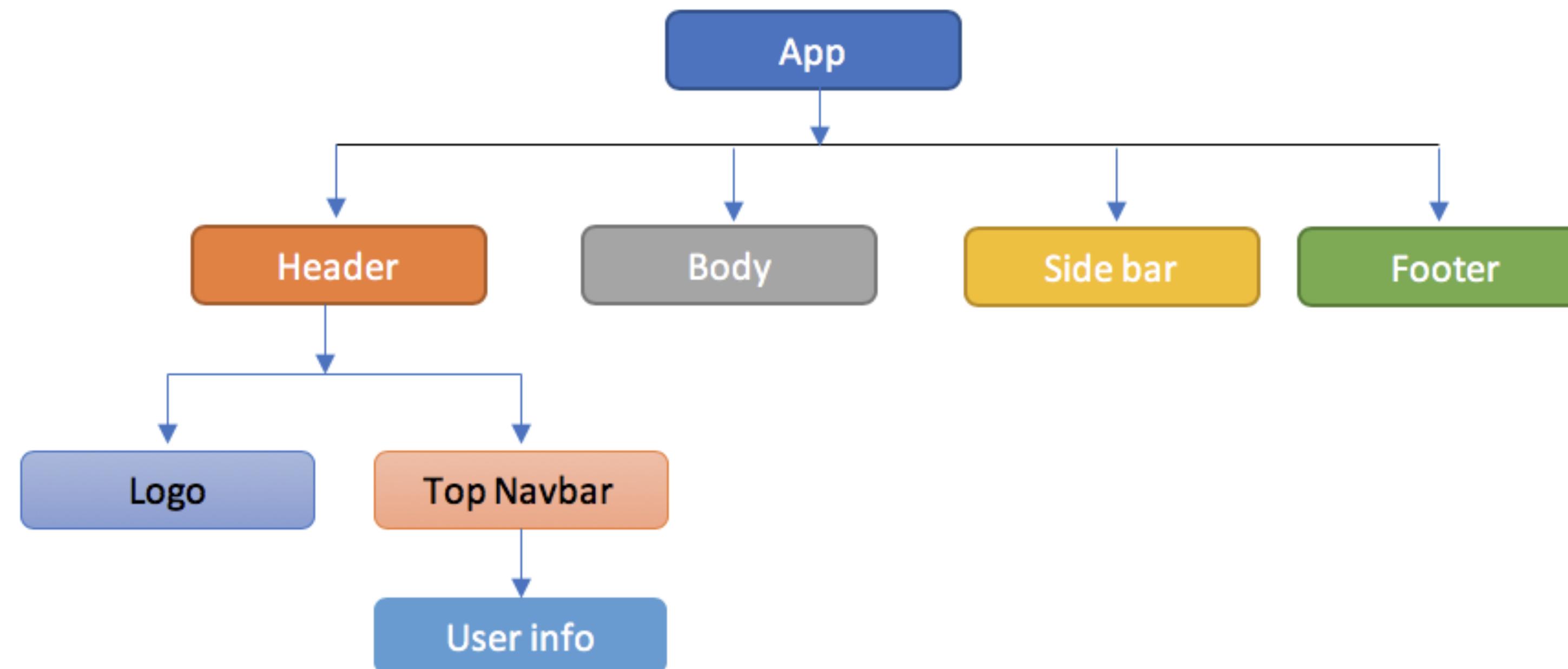
Components without  
a closing tag  
REQUIRE a closing  
slash

# Components, Components, Components



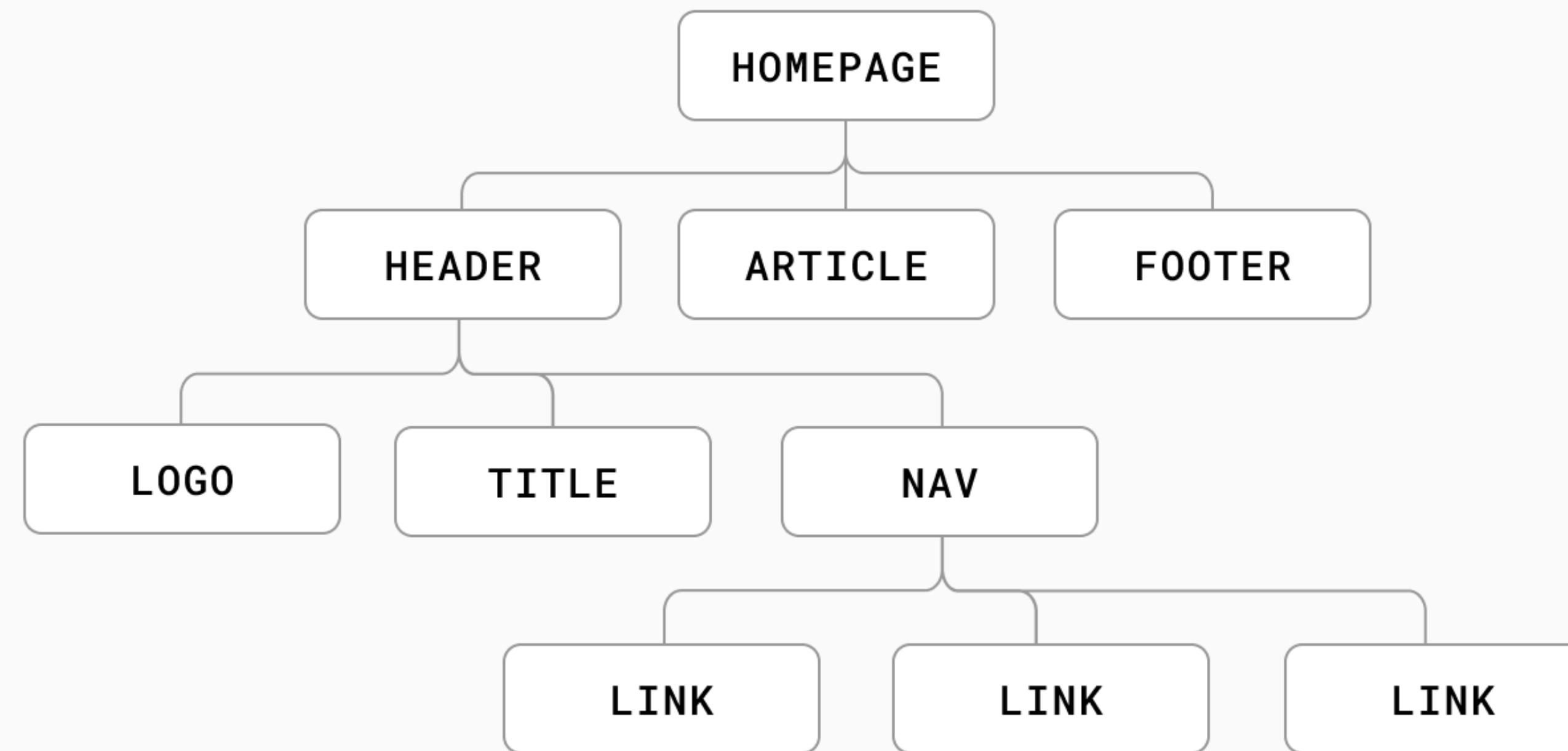
# Build Reusable Components

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



# 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

Pro NEW Get Certified Website Login

HTML CSS JAVASCRIPT SQL PYTHON PHP BOOTSTRAP HOW TO W3.CSS JAVA JQUERY C C++ C# R React

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

- What is a Hook?
- useState
- useEffect
- useContext
- useRef
- useReducer
- useCallback
- useMemo
- Custom Hooks

React Exercises

# 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.

In older React code bases, you may find Class components primarily used. It is now suggested to use Function components along with Hooks, which were added in React 16.8. There is an optional section on Class components for your reference.

## Create Your First Component

When creating a React component, the component's name *MUST* start with an upper case letter.

### Class Component

We just launched  
W3Schools videos

Explore now

COLOR PICKER

f i n d

Get certified  
by completing  
a React.js  
course today!

# Components?

This screenshot shows a web application interface with a dark header bar containing 'React CDN Template' and a URL. Below the header is a navigation bar with 'POSTS', 'CREATE', and 'PROFILE' buttons. The main content area displays a grid of six cards, each representing a post. Each card includes a user profile picture, the user's name and title, a preview image, and a short snippet of text.

Post	Content Preview
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	

This screenshot shows a web application interface titled 'ALL USERS' with a dark header bar containing 'SINGLE PAGE WEB APP TEMP' and a URL. The page features a search bar and filters for 'Order by' and 'Filter by'. It displays a grid of user profiles, each with a photo, name, email, and enrollment status. There are 'UPDATE' and 'DELETE' buttons for each user entry.

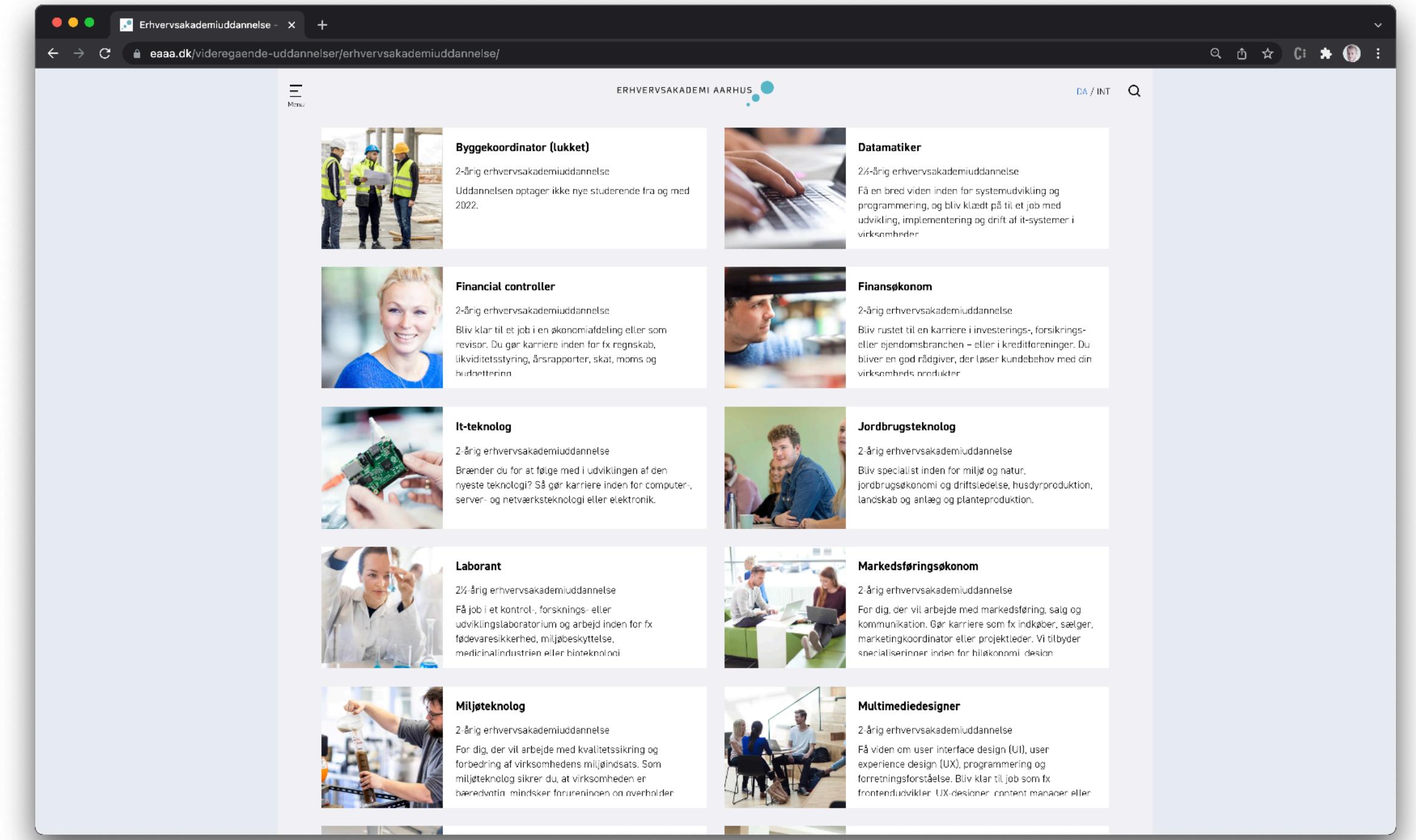
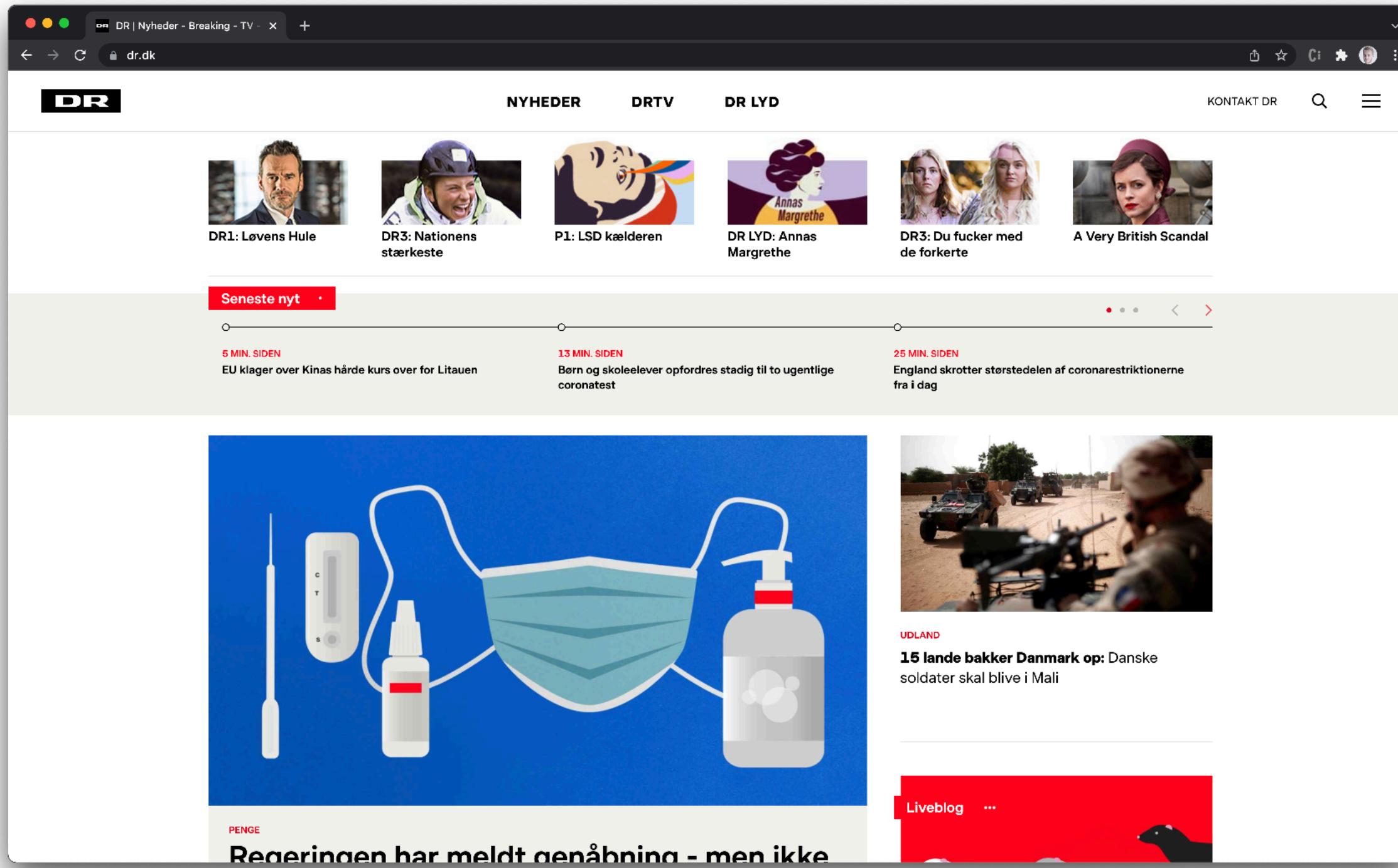
User	Email	Enrollment
Diana Riis Myjak Andersen	<a href="#">eaadimy@students.eaaa.dk</a>	StudentEnrollment
German Arias Rodriguez	<a href="#">eaagar@students.eaaa.dk</a>	StudentEnrollment
Haya Barakat	<a href="#">eaahbar@students.eaaa.dk</a>	StudentEnrollment

react-cdn-firebase-post-app-auth

Template: spa-canvas-users-template

Solution: spa-canvas-users

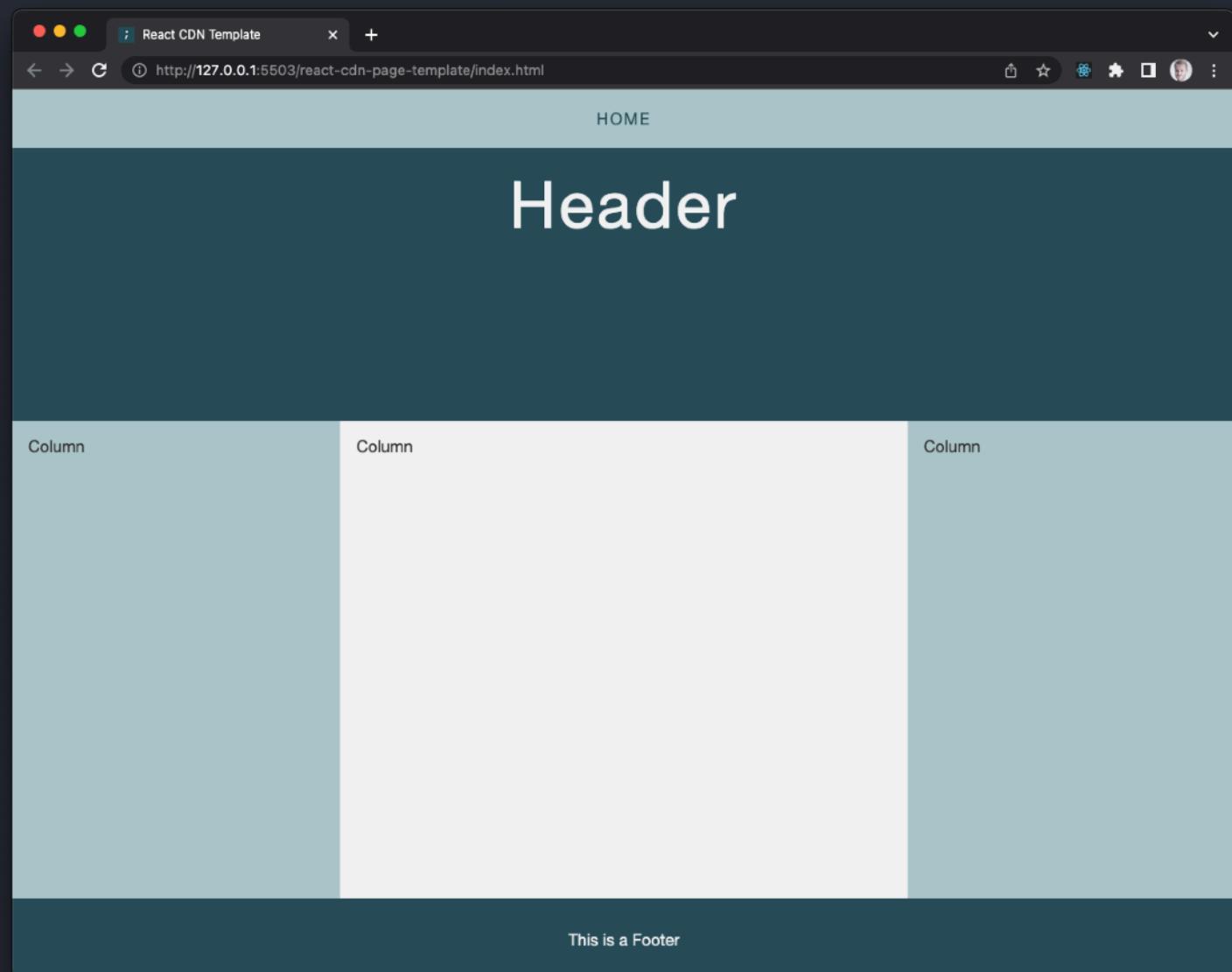
# And?



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

# Component Page Layout

1. Create a new react project
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 and/or flexbox. Get inspired by [react-cdn-page](#) ([code](#))



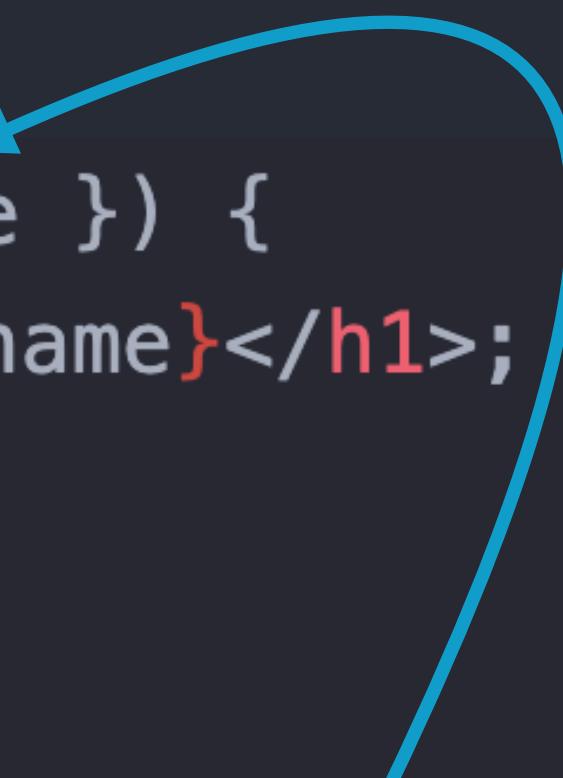
```
function App() {
  return (
    <main className="page-layout">
      <Navigation />
      <Header />
      <MainContent />
      <Footer />
    </main>
  );
}

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>  
  );  
}
```

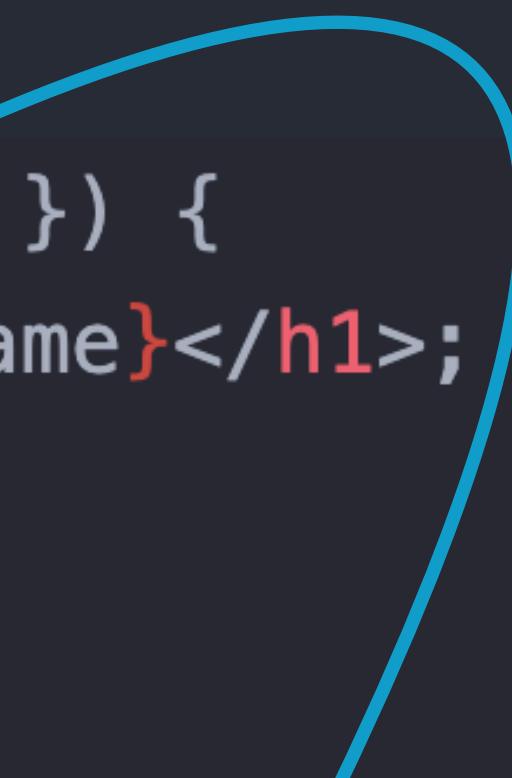
Set a prop on a child component by passing an attribute

Different instances of component can have different prop values

# Add Props: Your First React App

1. Use your implementation of [Your First React App](#).
2. Delete the `const name` variable Person component.
3. Add a prop to your Person component called `name`. The `name` prop is an argument pass to the function (component)t.
4. Make sure the `name` prop is rendered in the JSX of your Person component.
5. Locate your Person components in your App component and add a `name` attribute. Pass in a name to every Person component.
6. Test your solution and try to change the `name` attribute value.
7. Consider why props are useful in addition to creating dynamic web apps and reusable code.
8. Implement other props like `mail`, `phone`, etc., and make sure the props are rendered inside of your Person component.

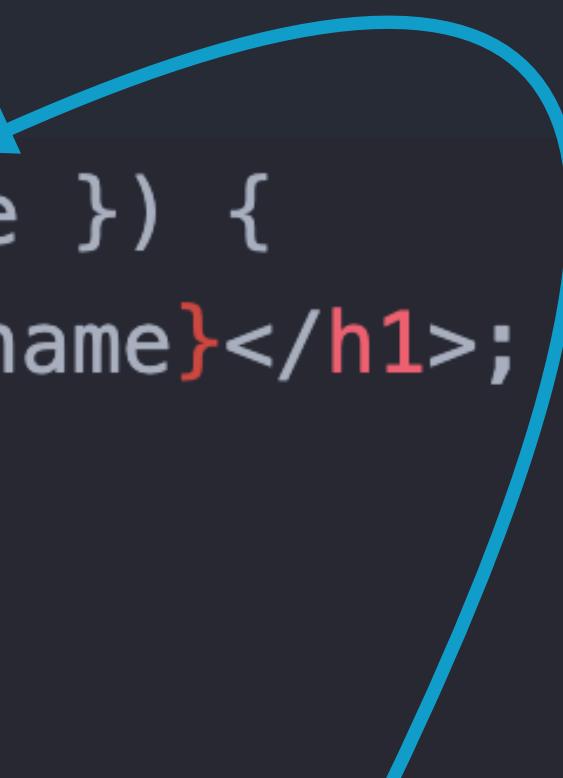
```
function Greeting({ name }) {  
  return <h1>Hello, {name}</h1>;  
}  
  
function App() {  
  return <Greeting name="world" />;  
}
```



# Add Props: Page Layout

1. Use your implemented Page Layout.
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.

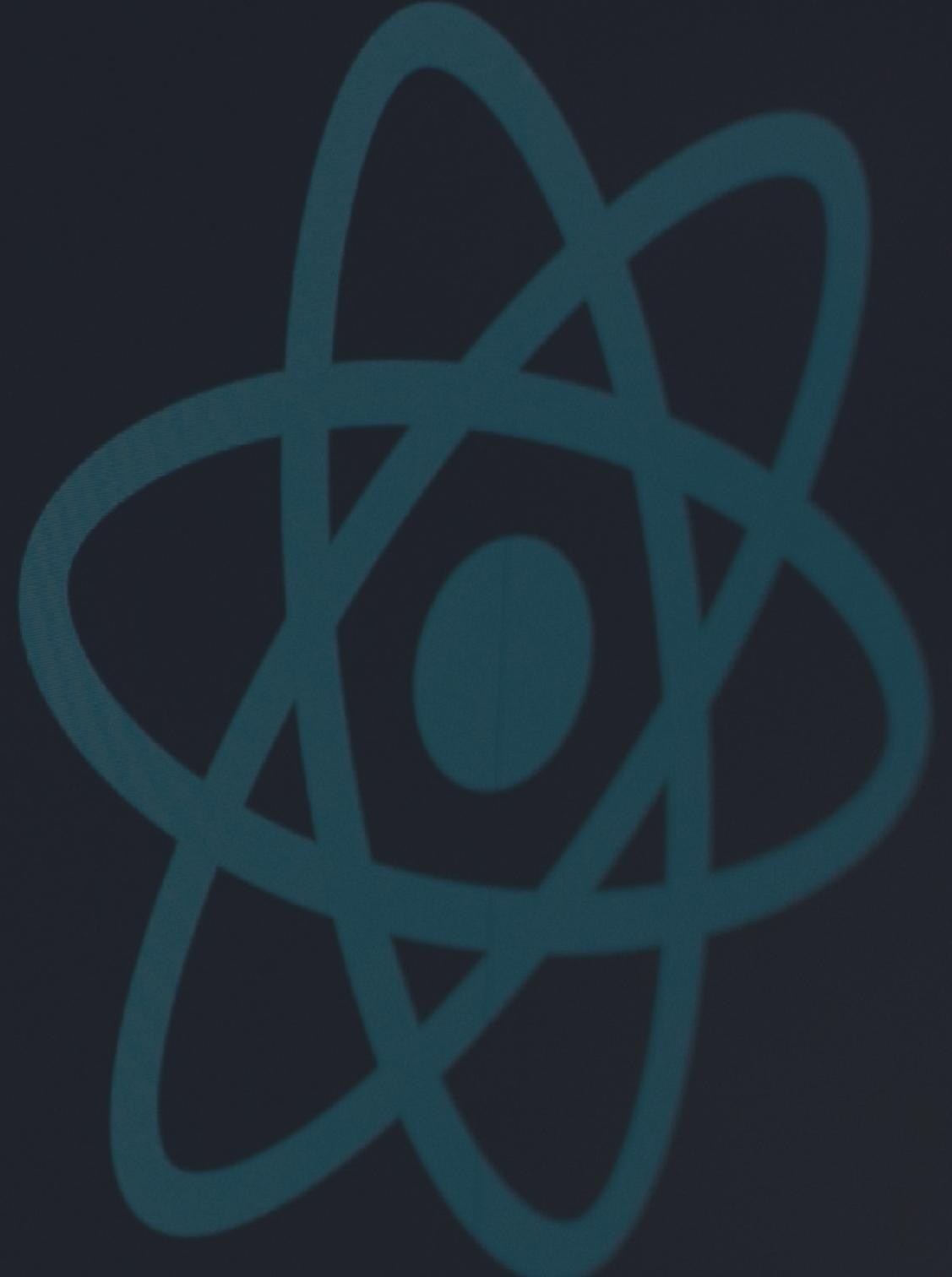
```
function Greeting({ name }) {  
  return <h1>Hello, {name}</h1>;  
}  
  
function App() {  
  return <Greeting name="world" />;  
}
```

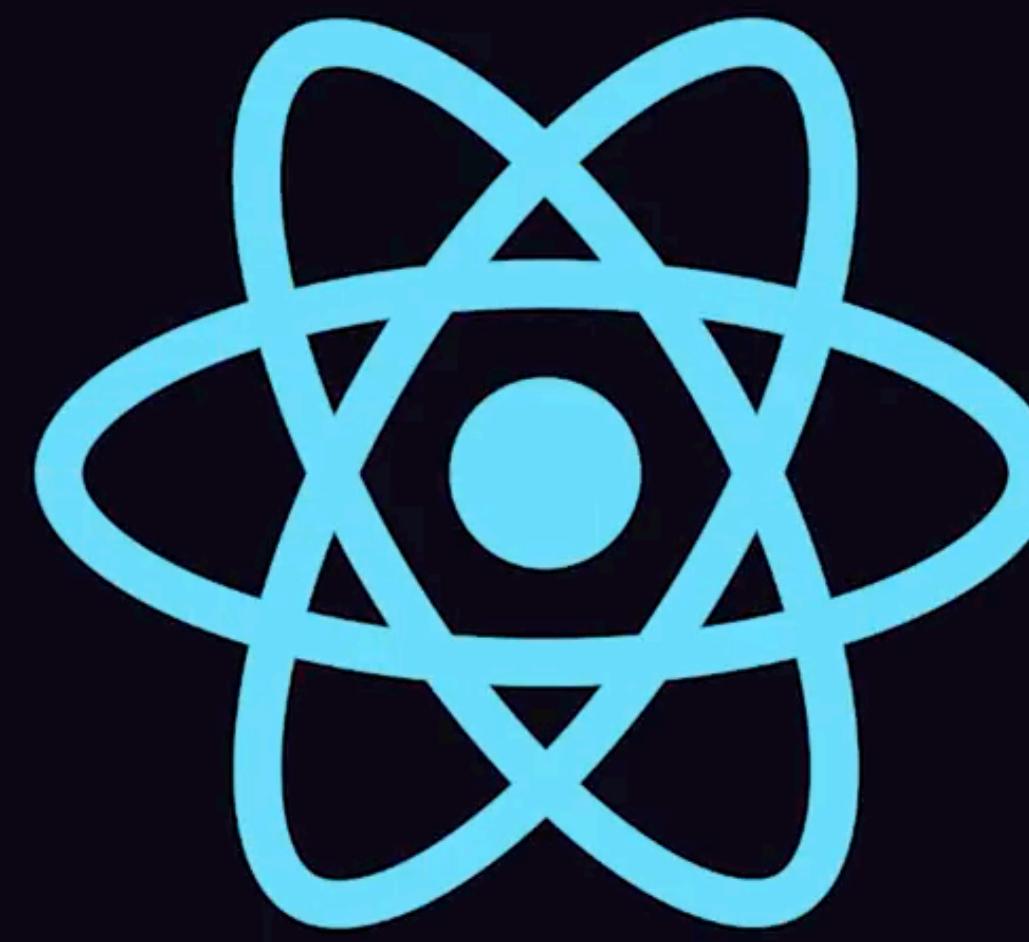


# React Hooks

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](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 change, 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. The title bar reads "From Vanilla JavaScript to React X". The URL in the address bar is "https://race.notion.site/From-Vanilla-JavaScript-to-React-Developer-4e35ed29a5c9460badbd7f9c64f565f1#7...". The main content area contains the following 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 create a new.
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 MyList component 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

# List, loop and map

```
const contacts = [
  {
    id: 1,
    name: "Birgitte Hansen",
    mail: "b@hansen.com"
  },
  {
    id: 2,
    name: "Jens Petersen",
    mail: "jens@petersen.com"
  },
  {
    id: 3,
    name: "Hanne Hansen",
    mail: "hanne@hansen.com"
  }
];

export default function ContactList() {
  return (
    <section className="contact-list">
      {contacts.map(contact => (
        <article className="card" key={contact.id}>
          <h2>{contact.name}</h2>
          <a href={`mailto:${contact.mail}`}>{contact.mail}</a>
        </article>
      )));
    </section>
  );
}
```

Data list

Map over the list and return jsx

# 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

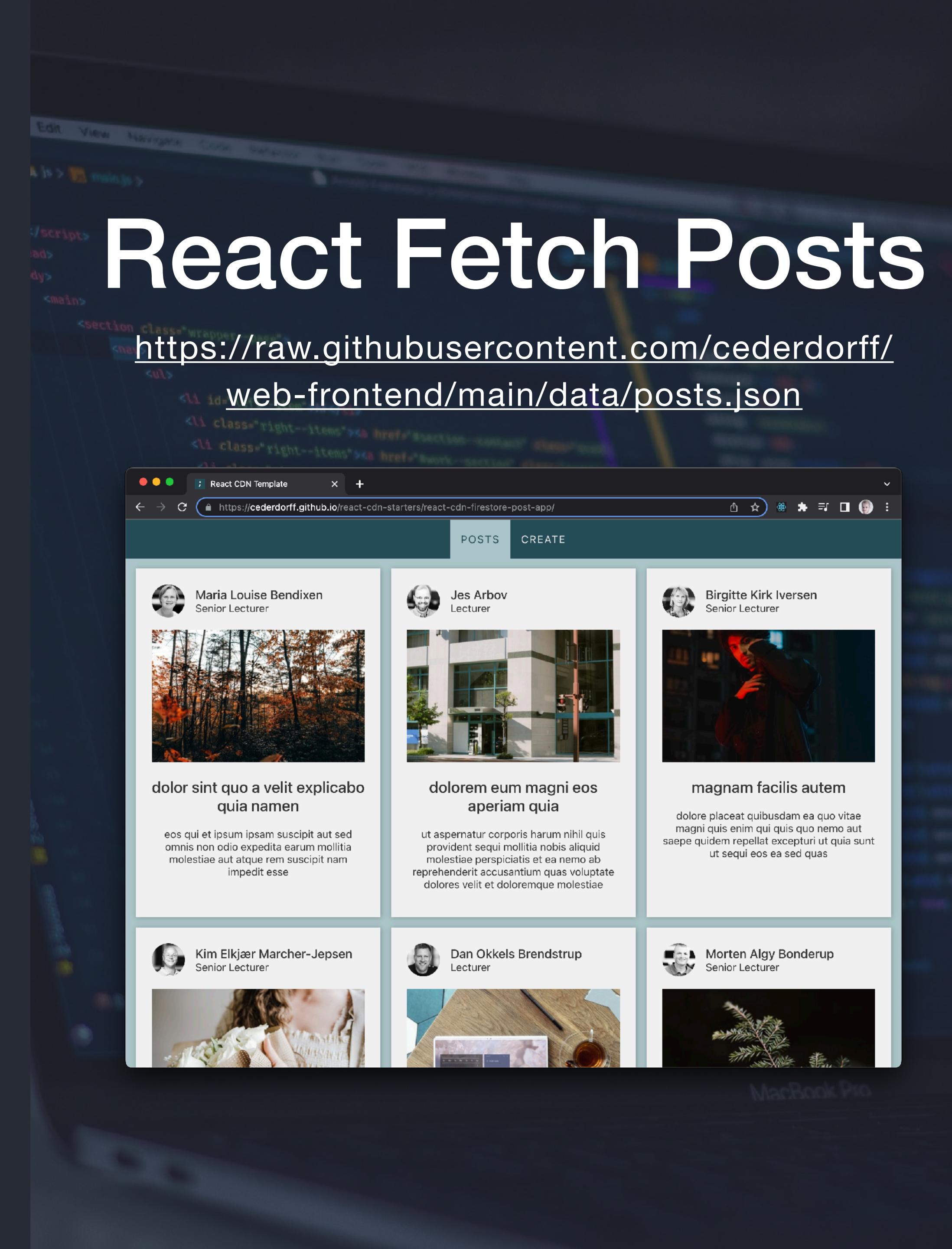
In useEffect, perform the sideeffect (fetch) to get data from a data source.  
And save the response data in a state using useState.

```
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();
  }, []);

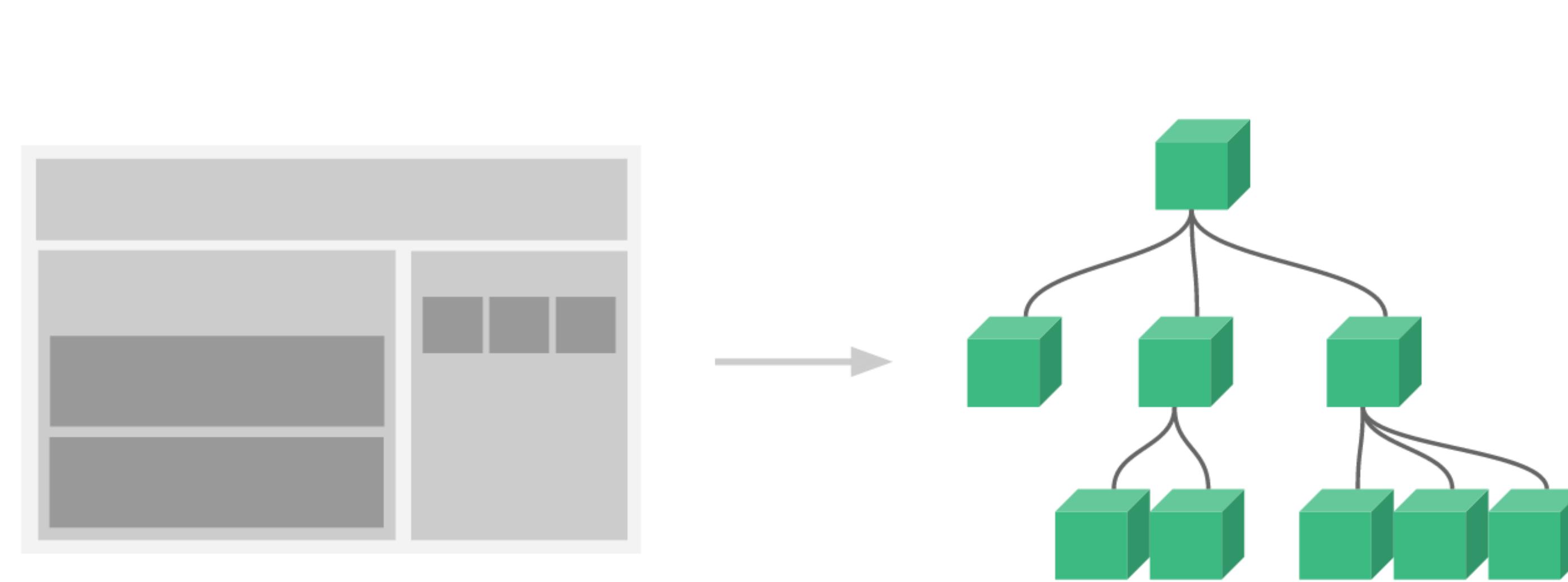
  return (
    <section className="page">
```

1. Use a new create project. The goal is to fetch and display a grid of posts using `useState` and `useEffect`. Similar to previous exercises (w/ Vanilla JS).
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.

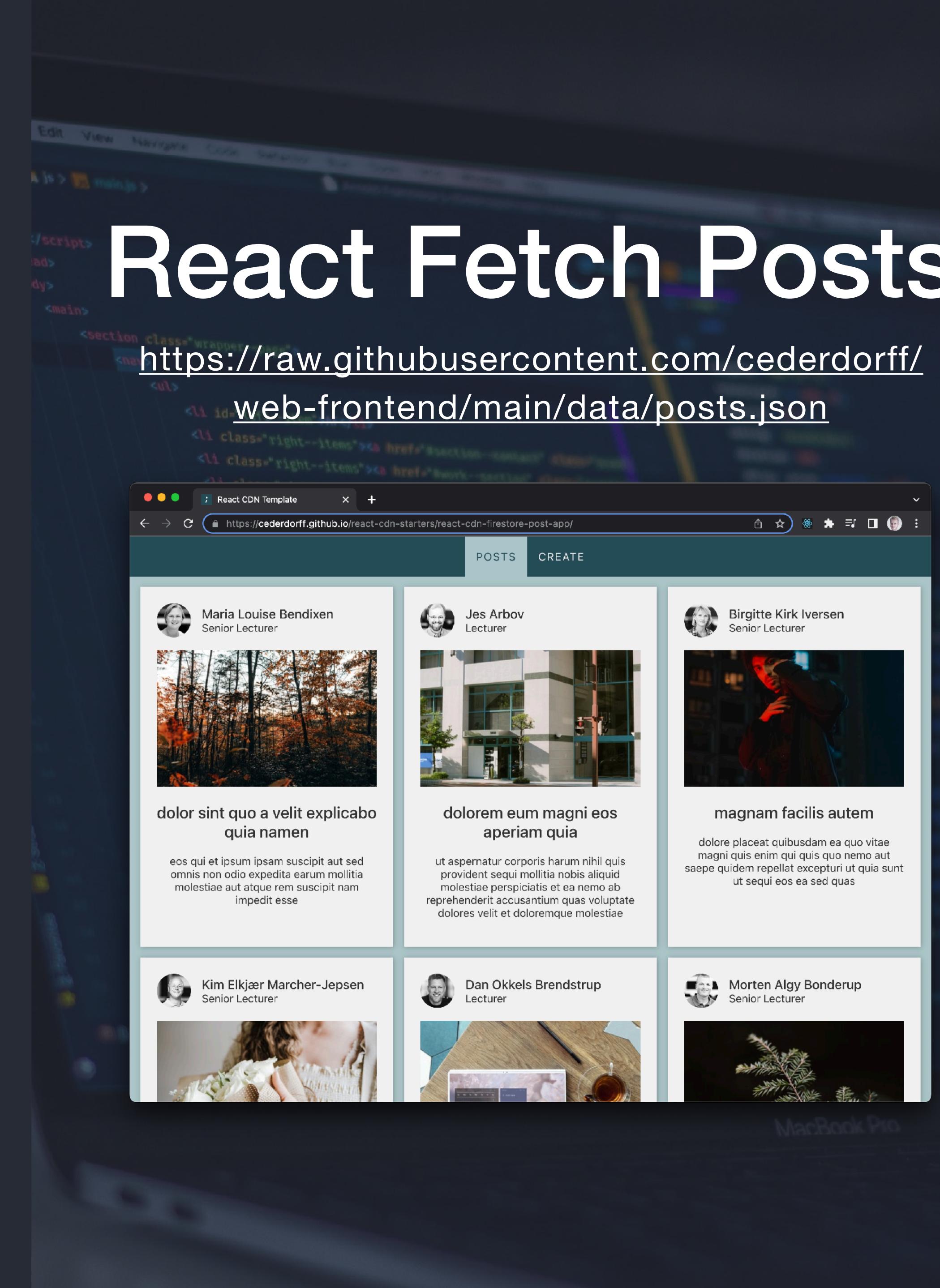


# Build Reusable Components

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 "Chapter" (JavaScript Fundamentals), "Lesson navigation" (The "if" statement, Boolean conversion, The "else" clause, Several conditions: "else if", Conditional operator "?", Multiple "?", Non-traditional use of "?", Tasks (5), Comments, Share (Twitter, Facebook), and Edit on GitHub. There is also an advertisement for Kendo UI Kits for Figma.

**Conditional operator ?'**

Sometimes, we need to assign a variable depending on a condition.

For instance:

```
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);
```

The so-called "conditional" or "question mark" operator lets us do that in a shorter and simpler way.

The operator is represented by a question mark `?`. Sometimes it's called "ternary", because the operator has three operands. It is actually the one and only operator in JavaScript which has that many.

The syntax is:

```
1 let result = condition ? value1 : value2;
```

The `condition` is evaluated: if it's truthy then `value1` is returned, otherwise – `value2`.

For example:

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

<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 a code example showing an error due to attempting to access a non-existent property.

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.

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:

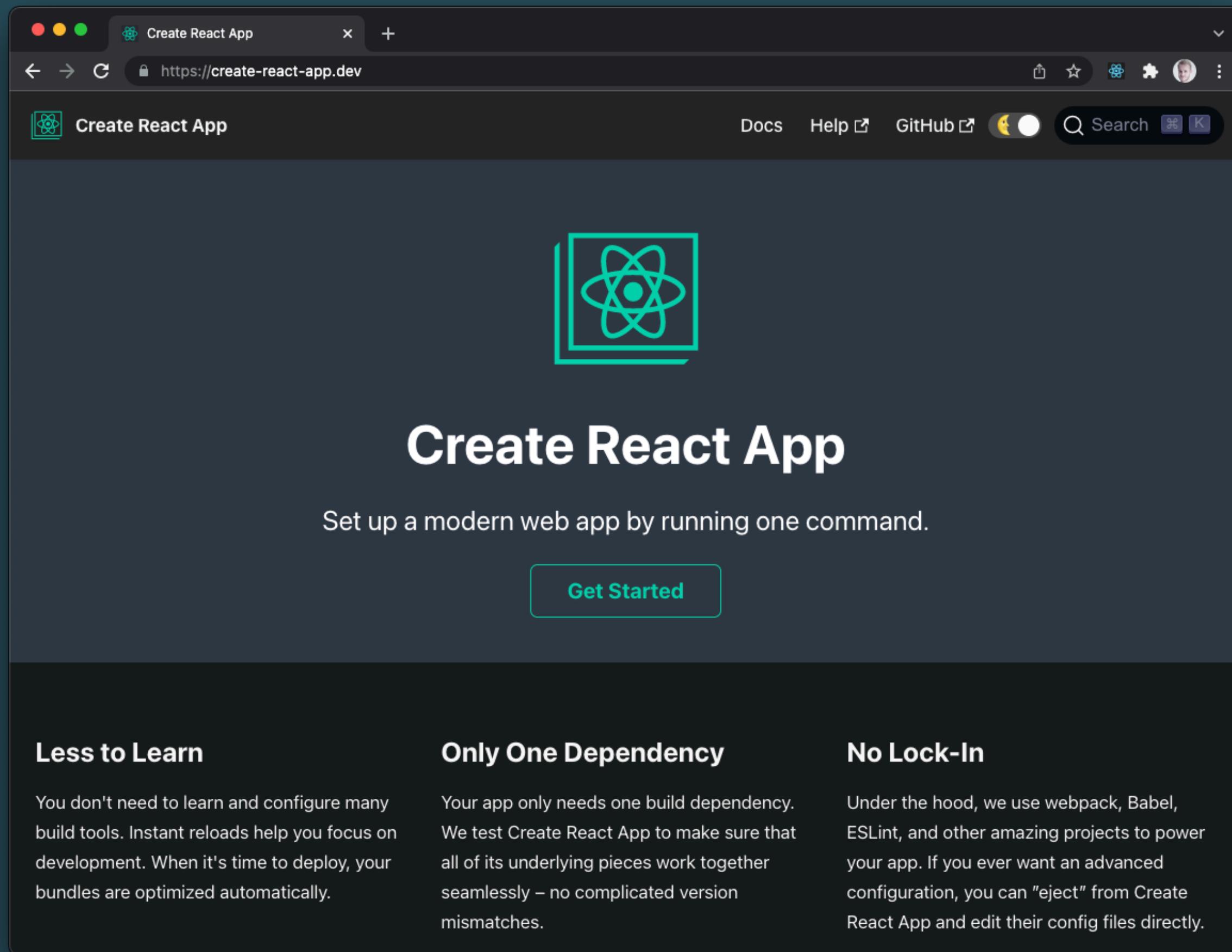
```
1 result = (a !== null && a !== undefined) ? a : b;
```

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".

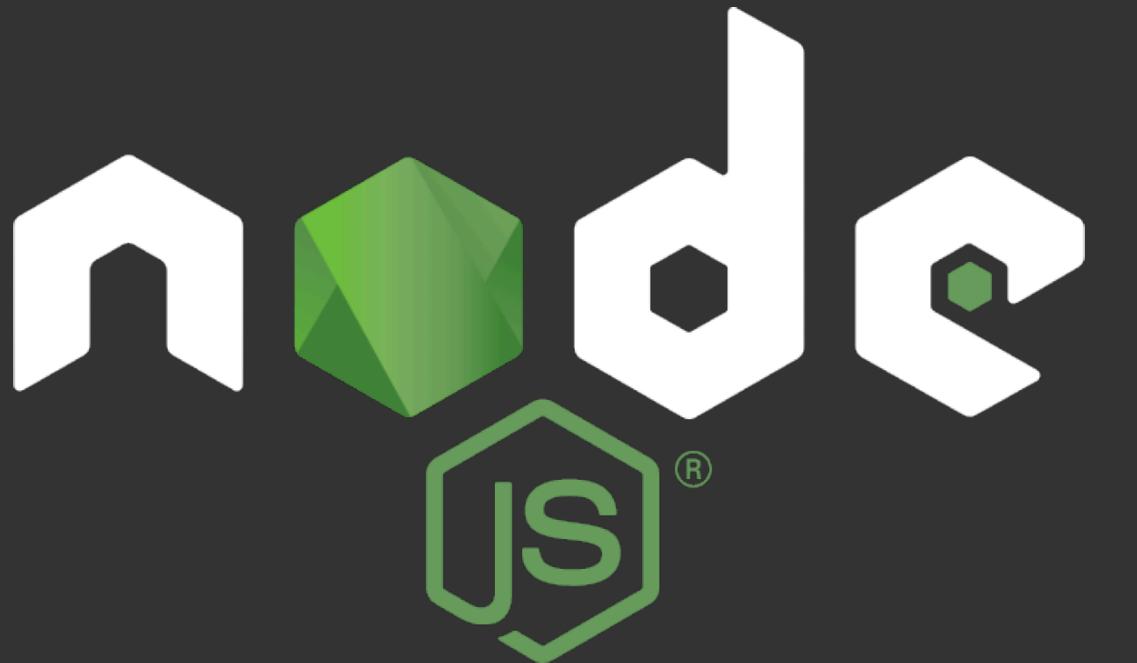
<https://javascript.info/nullish-coalescing-operator>

# create-react-app

Starter template for your react app



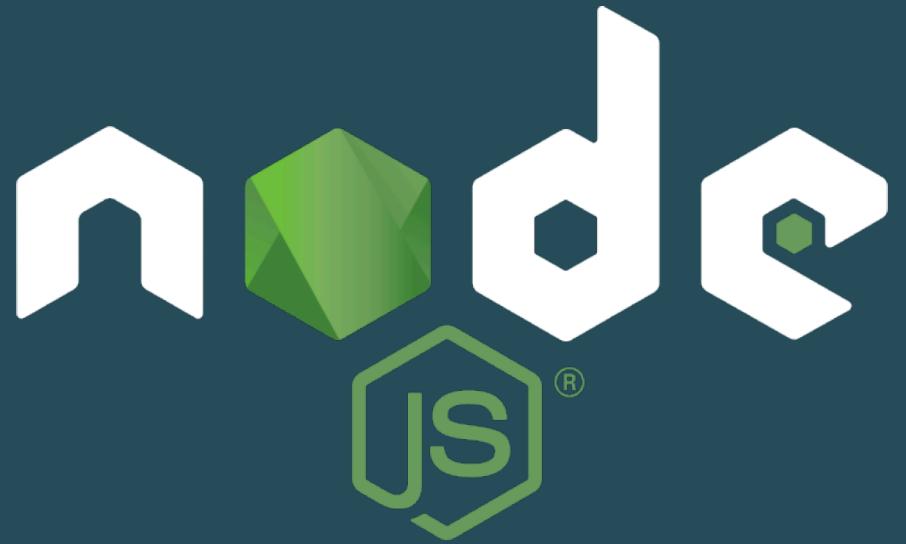
Generate create-react-app



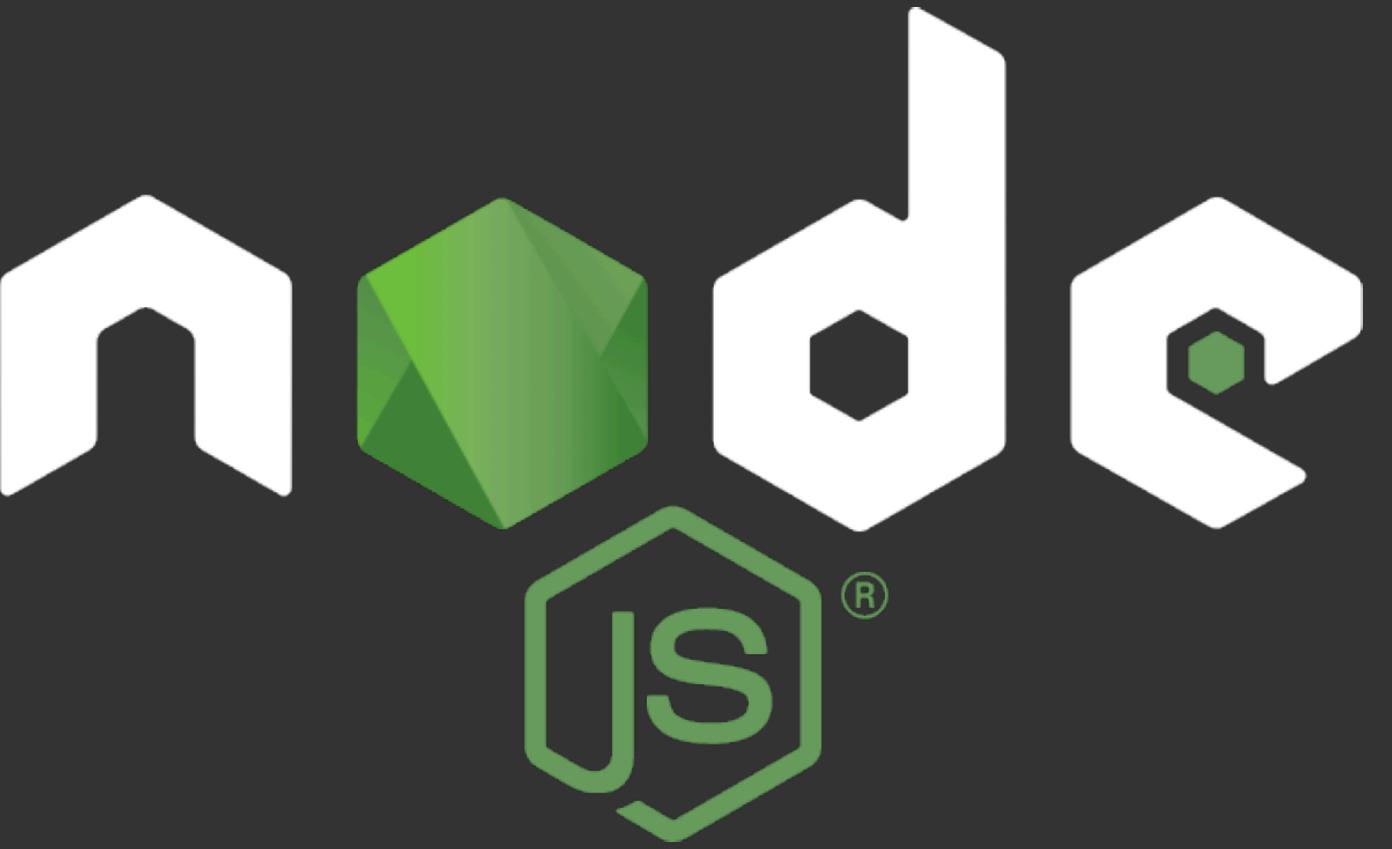
# Getting started

Download & install Node.js & npm: [nodejs.org/en/](https://nodejs.org/en/)

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

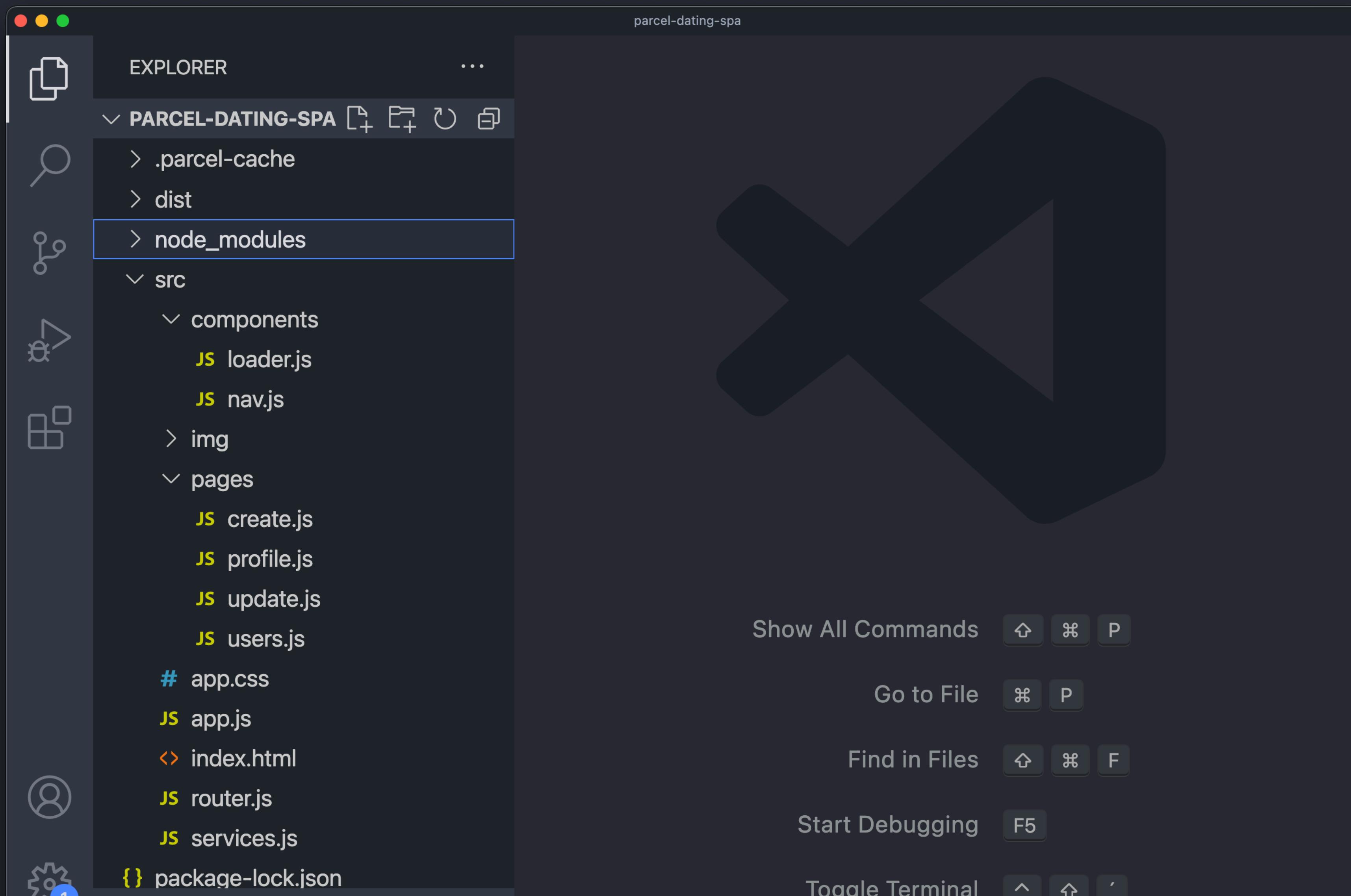
Npm is a package manager for node.js  
packages - or modules



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

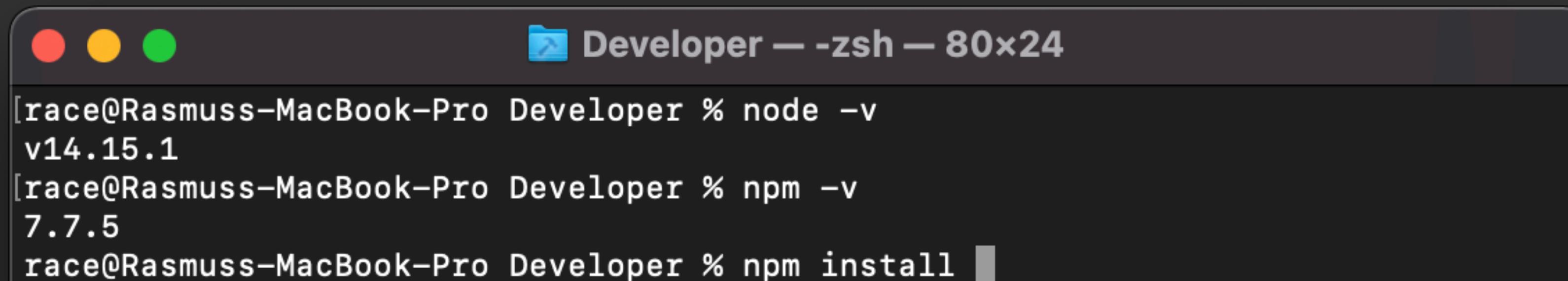
1. Website to discover packages
2. Command line interface (cli)
3. A registry - database with JS software / node modules

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

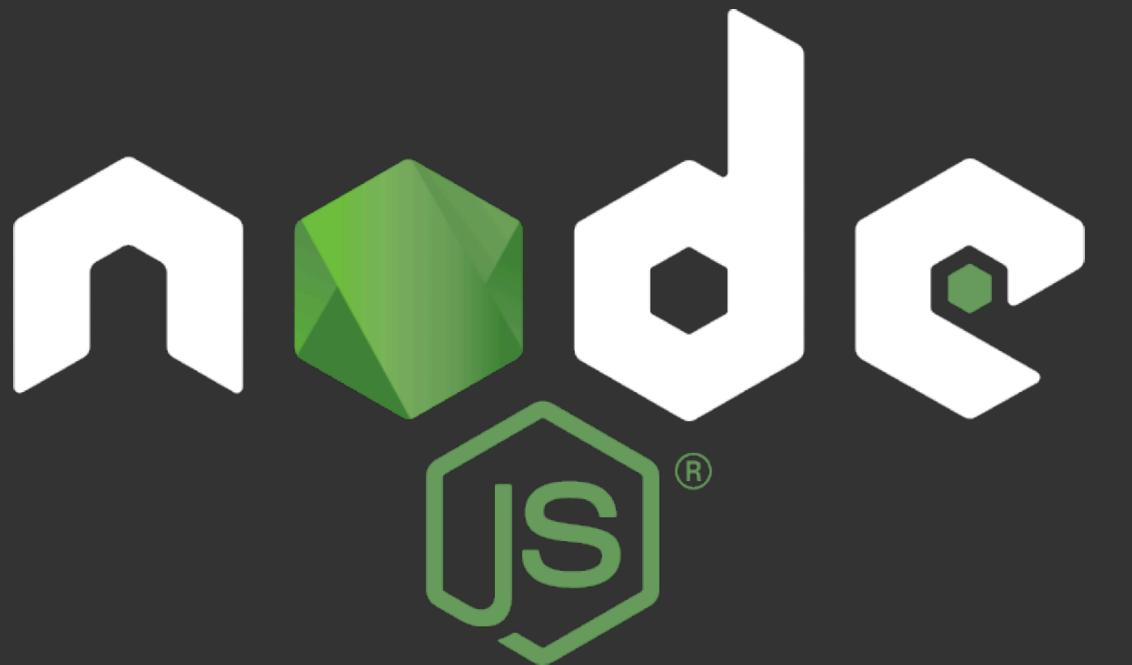
# Command Line Interface

... 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/](https://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...

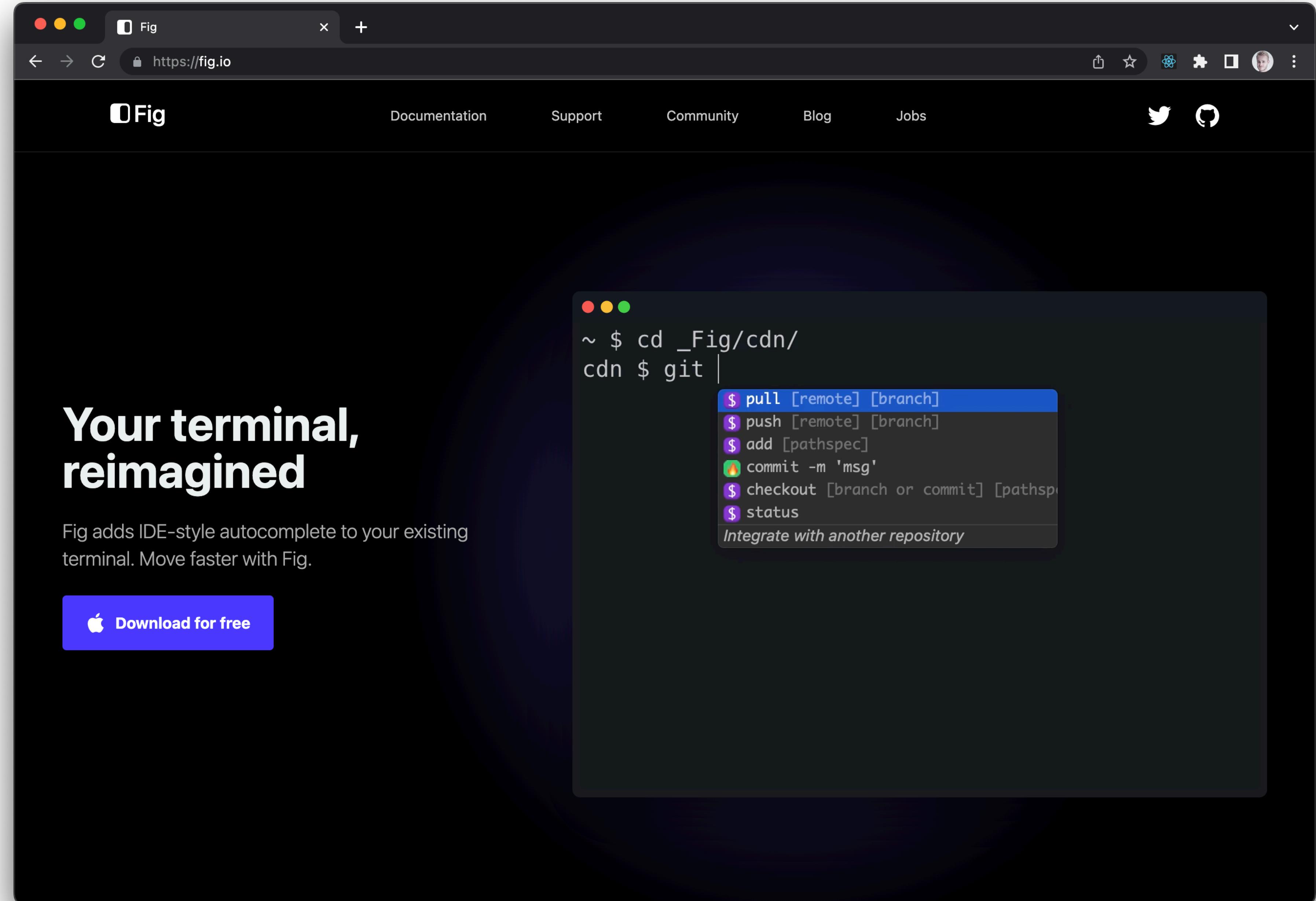
Configure Default Build Task...

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 % █
```

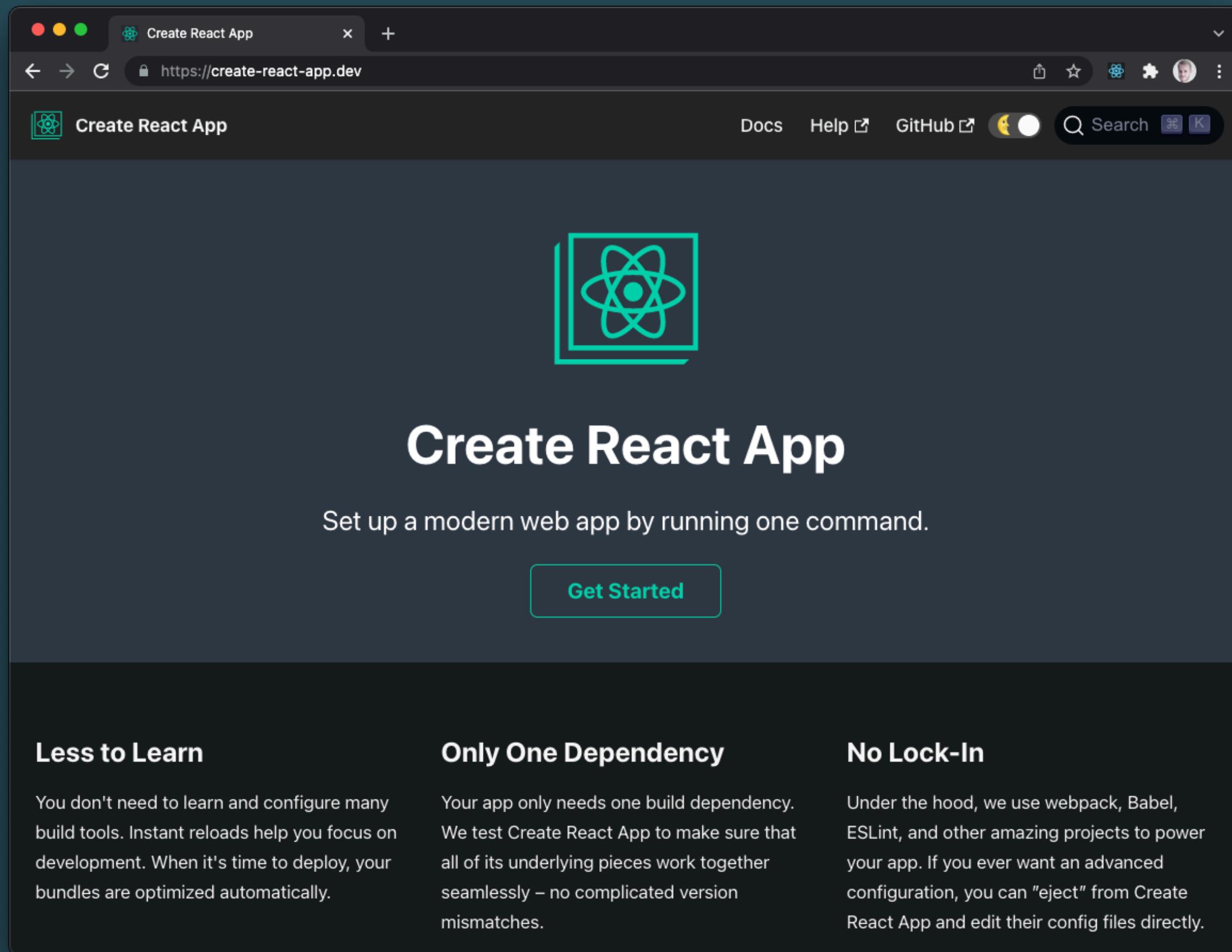
zsh + × □ └ ^ ×



<https://fig.io/>

# create-react-app

Starter template for your react app



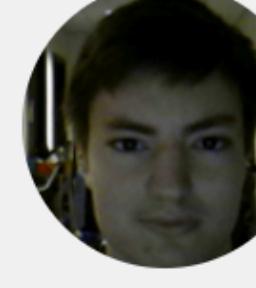
Generate create-react-app

# Canvas Users Case w/ React

React App <http://localhost:3000> Incognito (2)

## Canvas Users

Show Teachers  Search  Sort by

 <b>Anders Husted</b> <a href="mailto:eaaahu@students.eaaa.dk">eaaahu@students.eaaa.dk</a> Role: Student	 <b>Anders Lassen Skrydstrup</b> <a href="mailto:eaaanls@students.eaaa.dk">eaaanls@students.eaaa.dk</a> Role: Student	 <b>Aron Ingi Svansson</b> <a href="mailto:eaaarsv@students.eaaa.dk">eaaarsv@students.eaaa.dk</a> Role: Student
 <b>Benjamin Tranberg Nikolajsen</b> <a href="mailto:eaabtn@students.eaaa.dk">eaabtn@students.eaaa.dk</a> Role: Student	 <b>Camilla Østrup Nørgaard</b> <a href="mailto:camilla.nrgaard@yahoo.dk">camilla.nrgaard@yahoo.dk</a> Role: Student	 <b>Casper Hedegaard Hansen</b> <a href="mailto:eaacahh@students.eaaa.dk">eaacahh@students.eaaa.dk</a> Role: Student
		

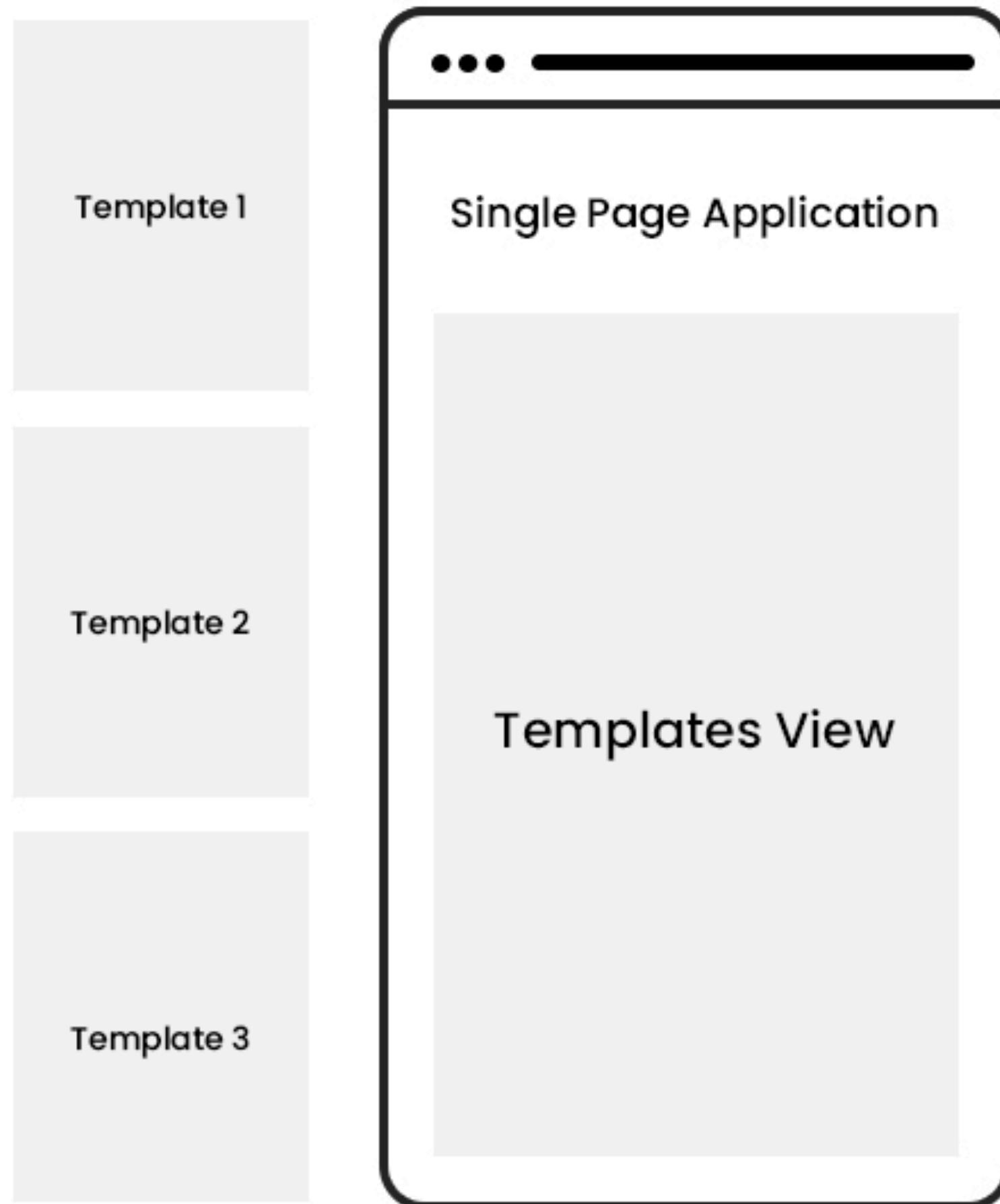
<https://race.notion.site/Canvas-Users-Case-w-React-191f2533ac8248f4874c077796497d86>

# 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.”*

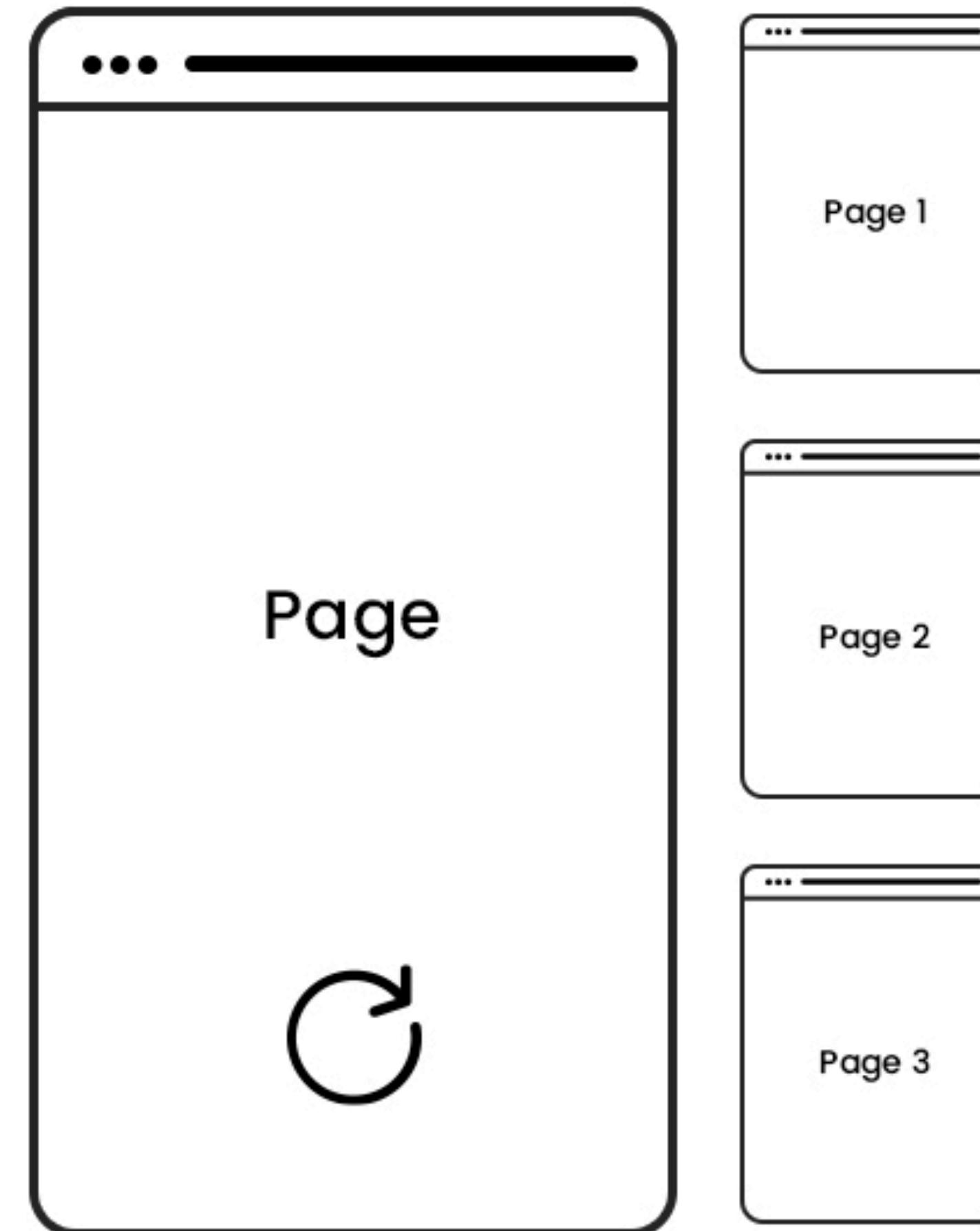
<https://reactjs.org/docs/glossary.html#single-page-application>

# SPA



No page refresh on request

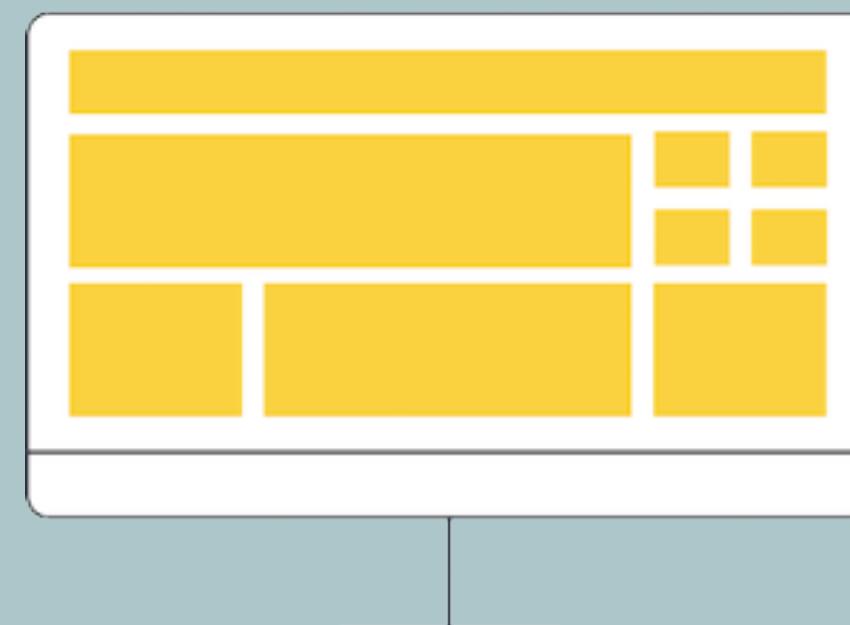
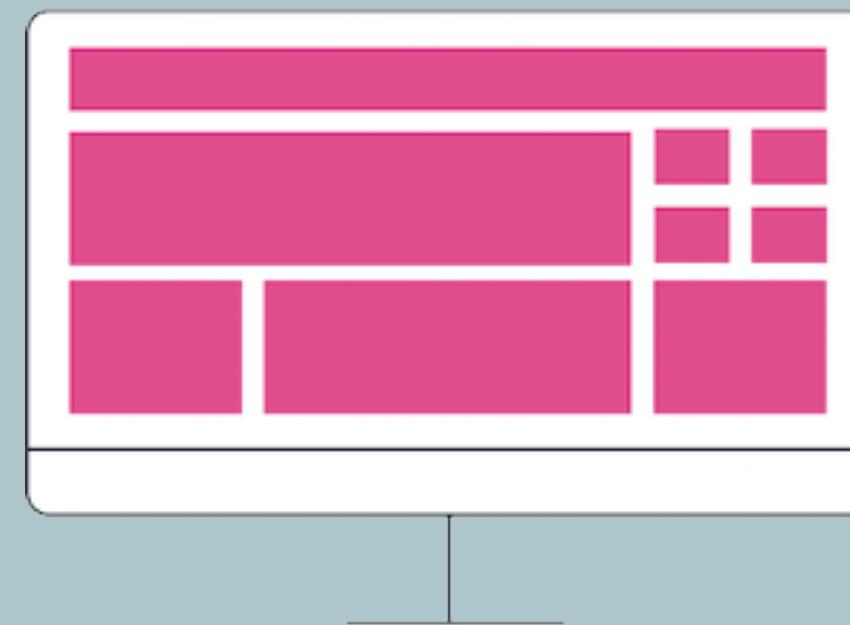
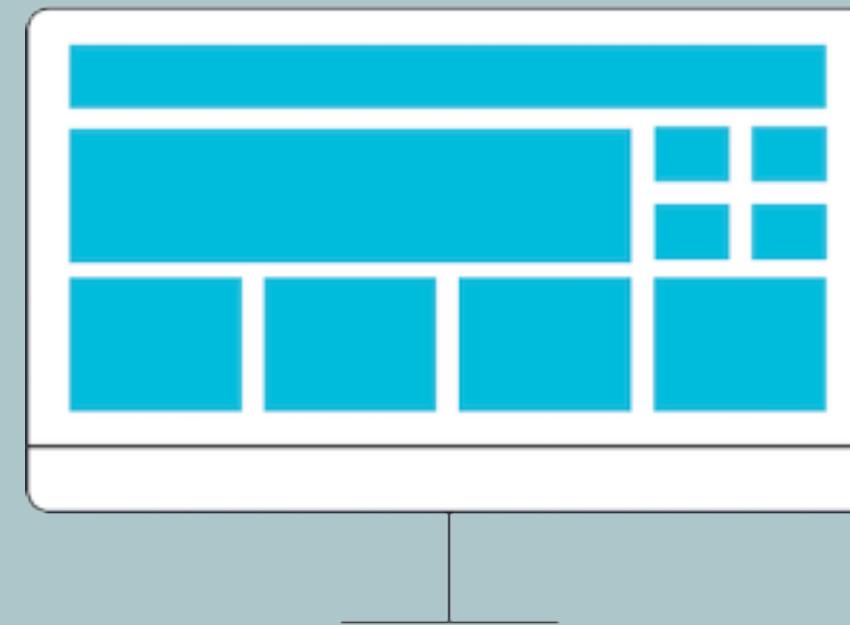
# MPA



Whole page refresh on request

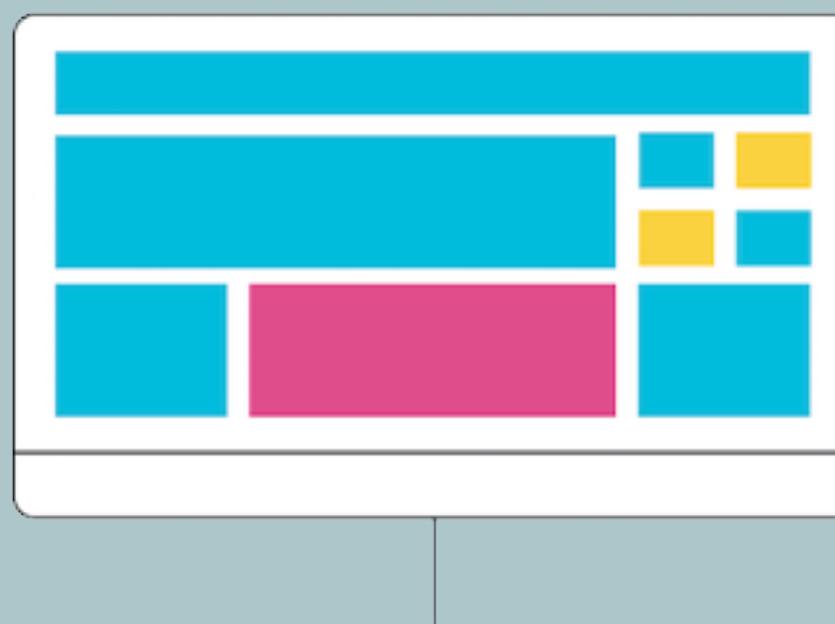
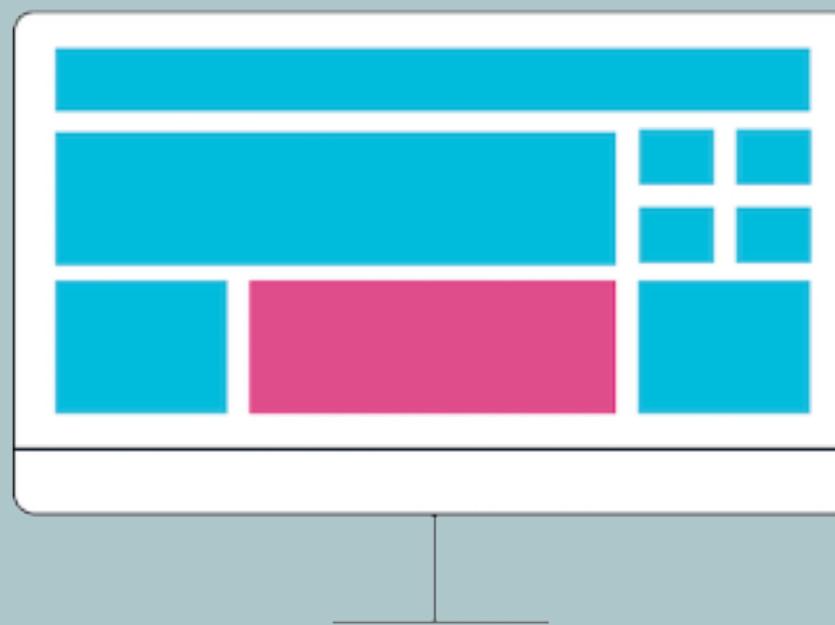
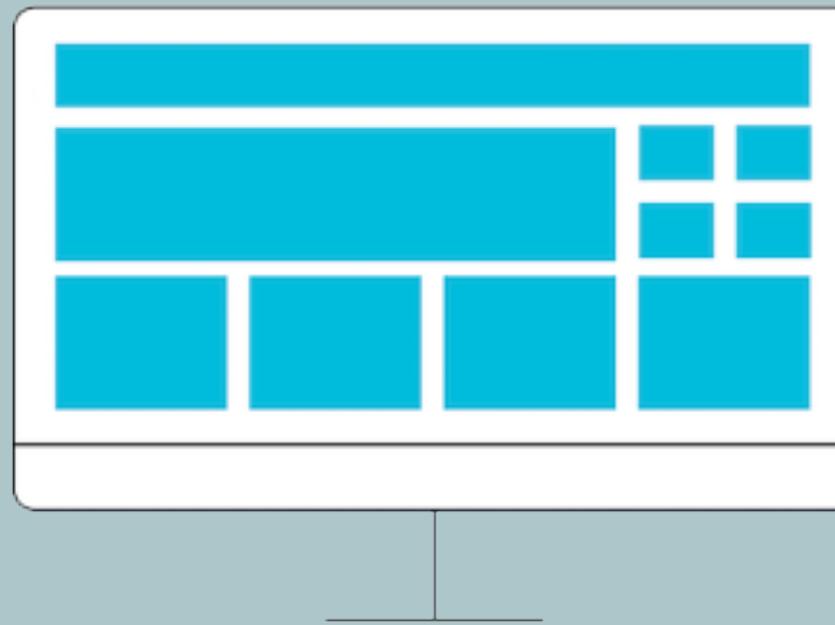
# MPA

- Every request for new information gives you a new version of the whole page.
- We have to load it all over again every time we want to change data and get new data.
- The browser has to render the page again.

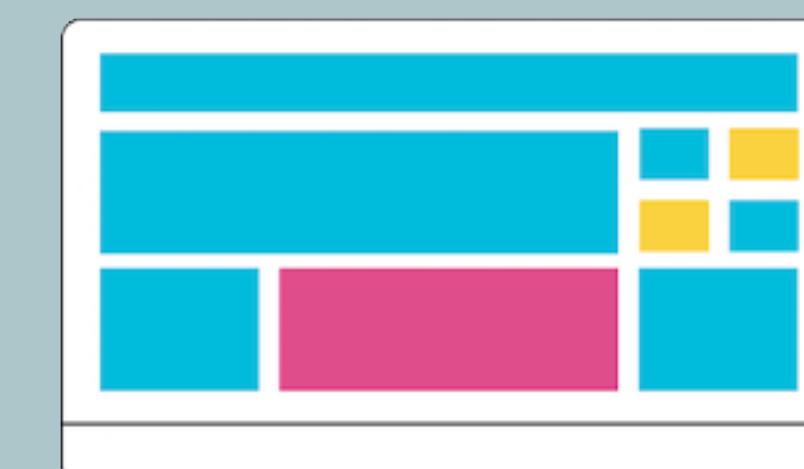
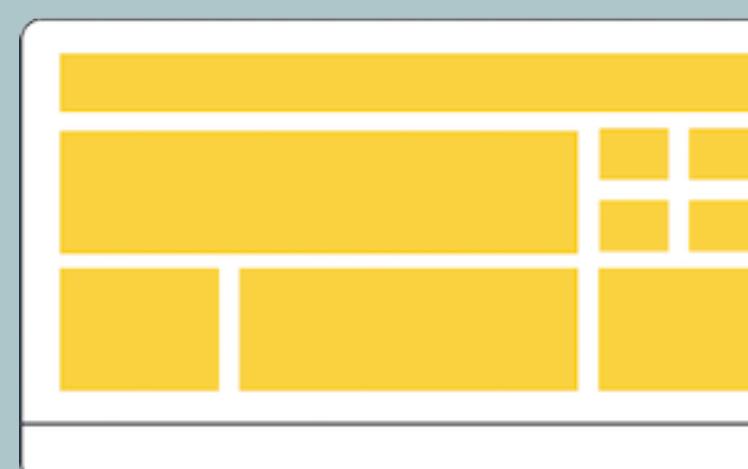
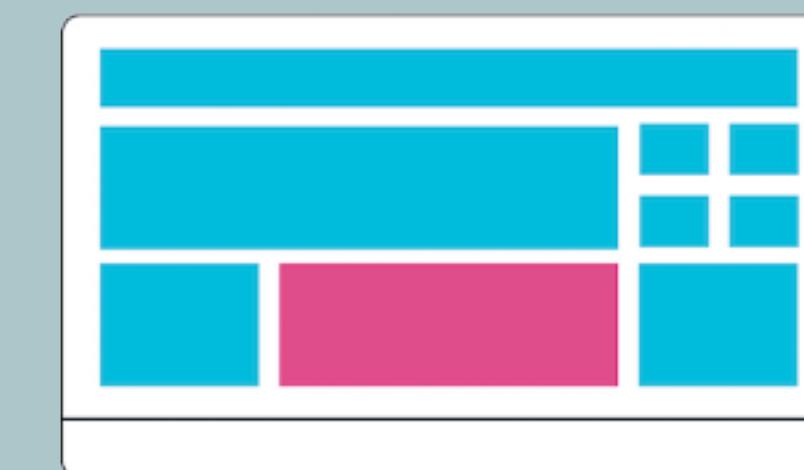
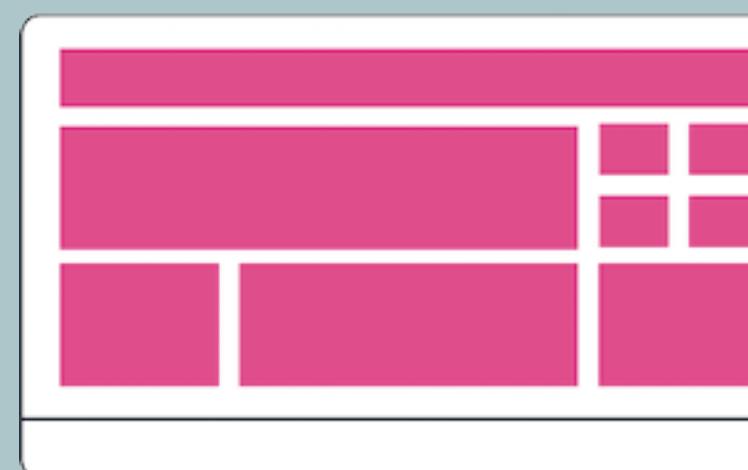
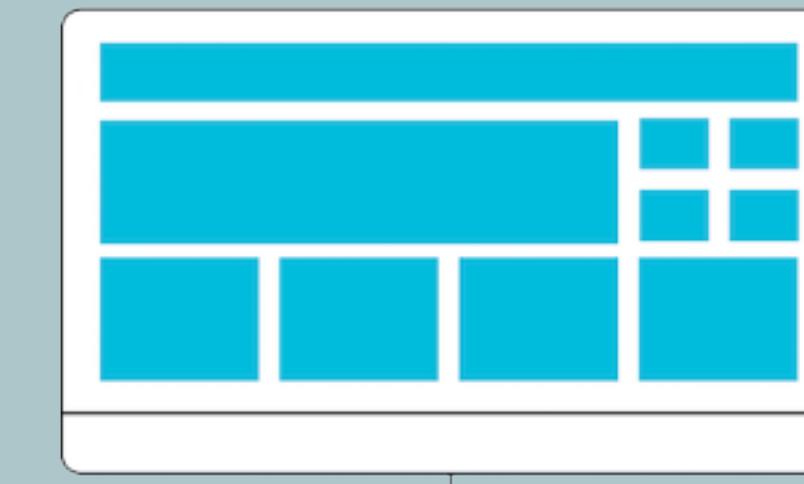
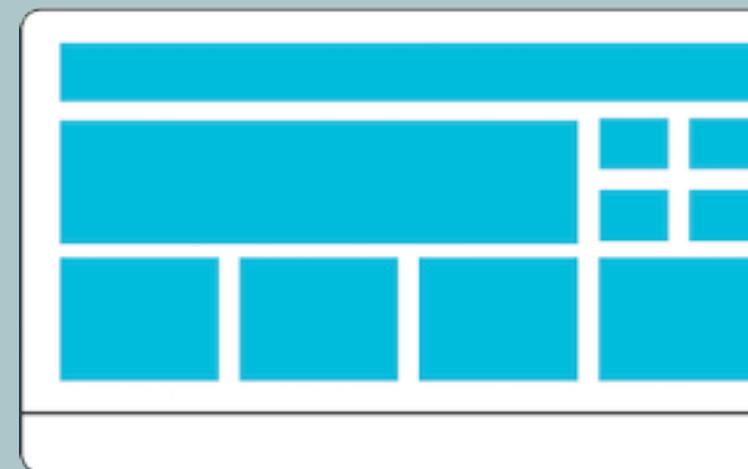


# SPA

- We only request and load the data we want.
- No page reload/ refresh.
- Client side & JavaScript.
- Less server work required.
- Advanced UI & Animations on page change.
- Better performance.

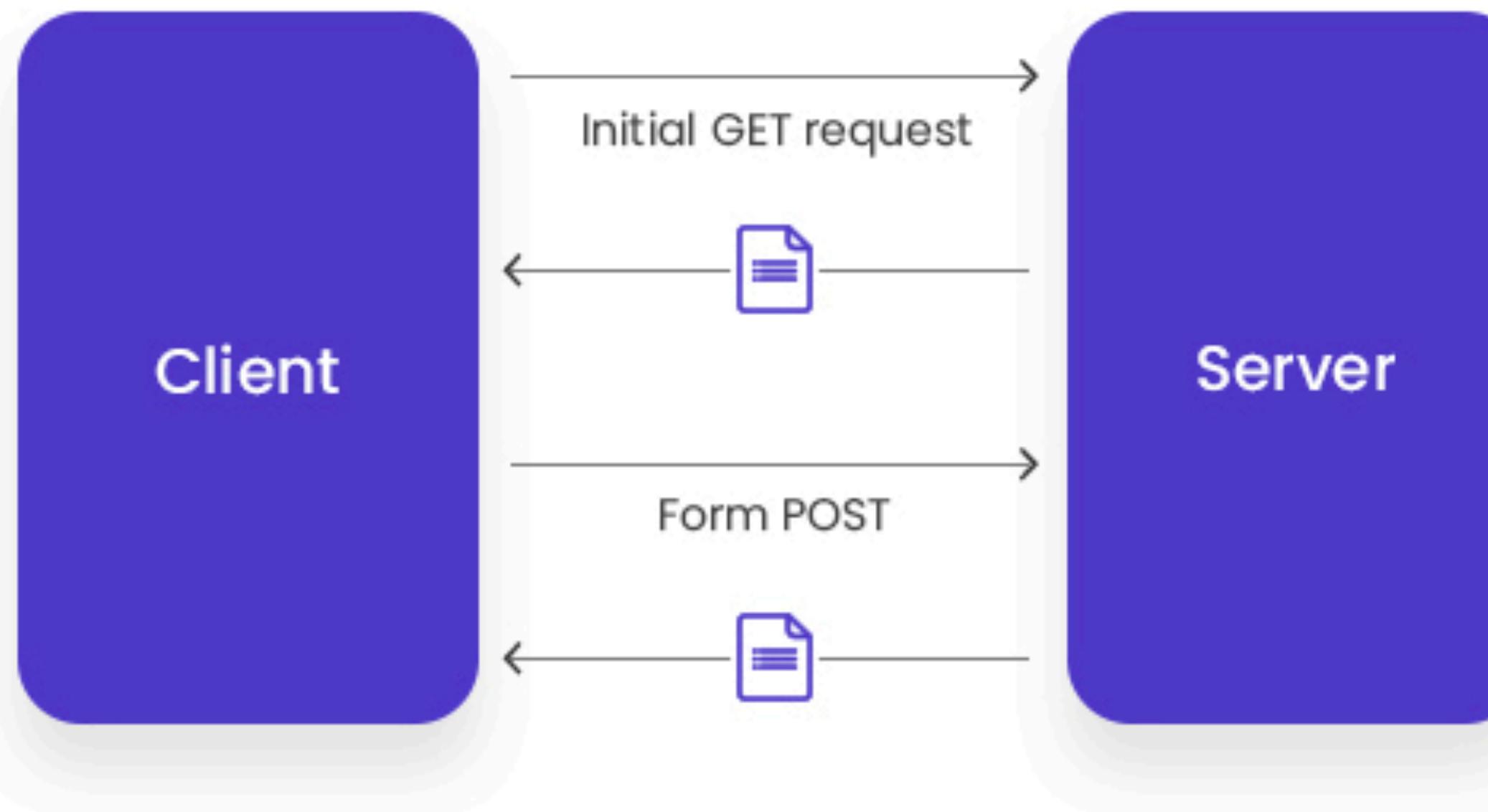


# MPA VS SPA

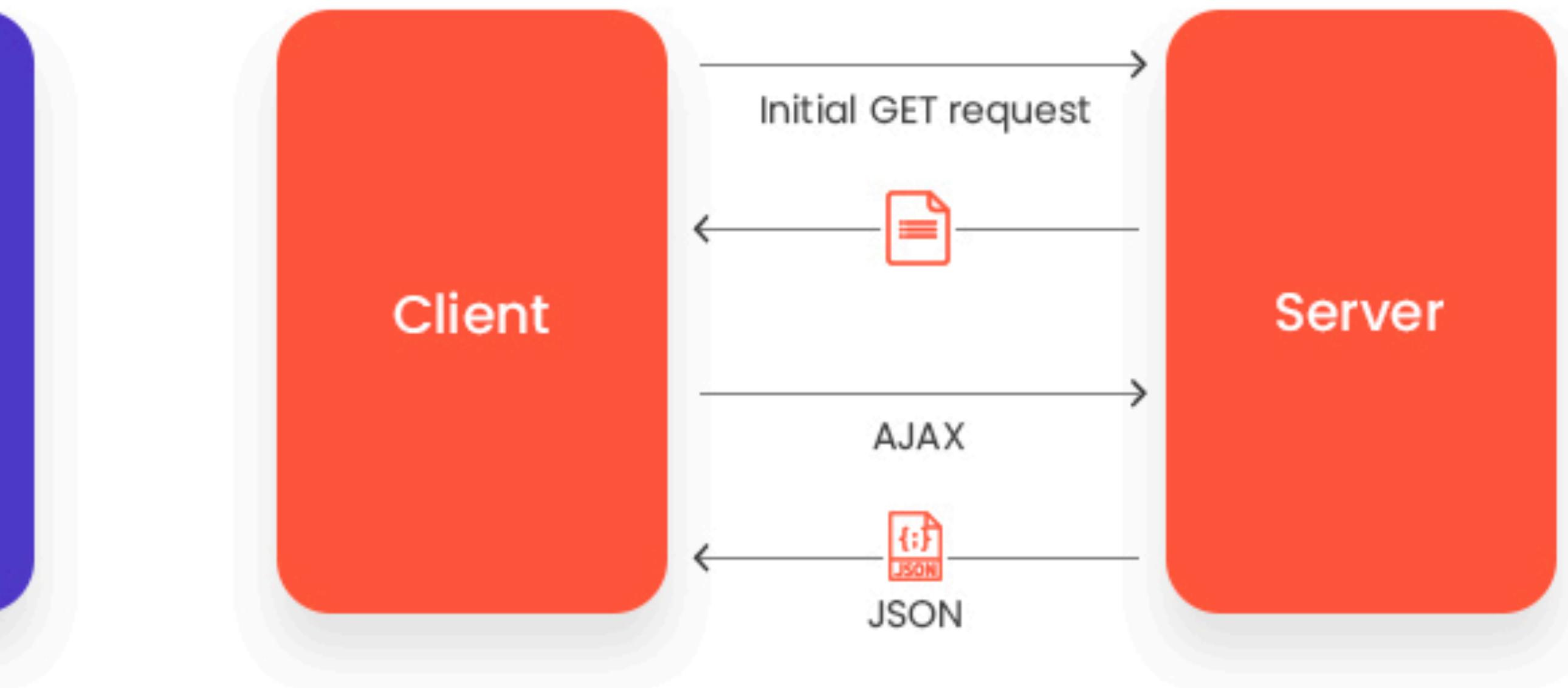


# MPA VS SPA

Traditional Page Lifecycle



SPA Lifecycle



# SPA's

Gmail

Google Maps

Facebook

Twitter

Google Drive

Instagram

Pinterest

# 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.

# SPA & Router

In an SPA the router is in charge of displaying the right view by watching changes on the URL.

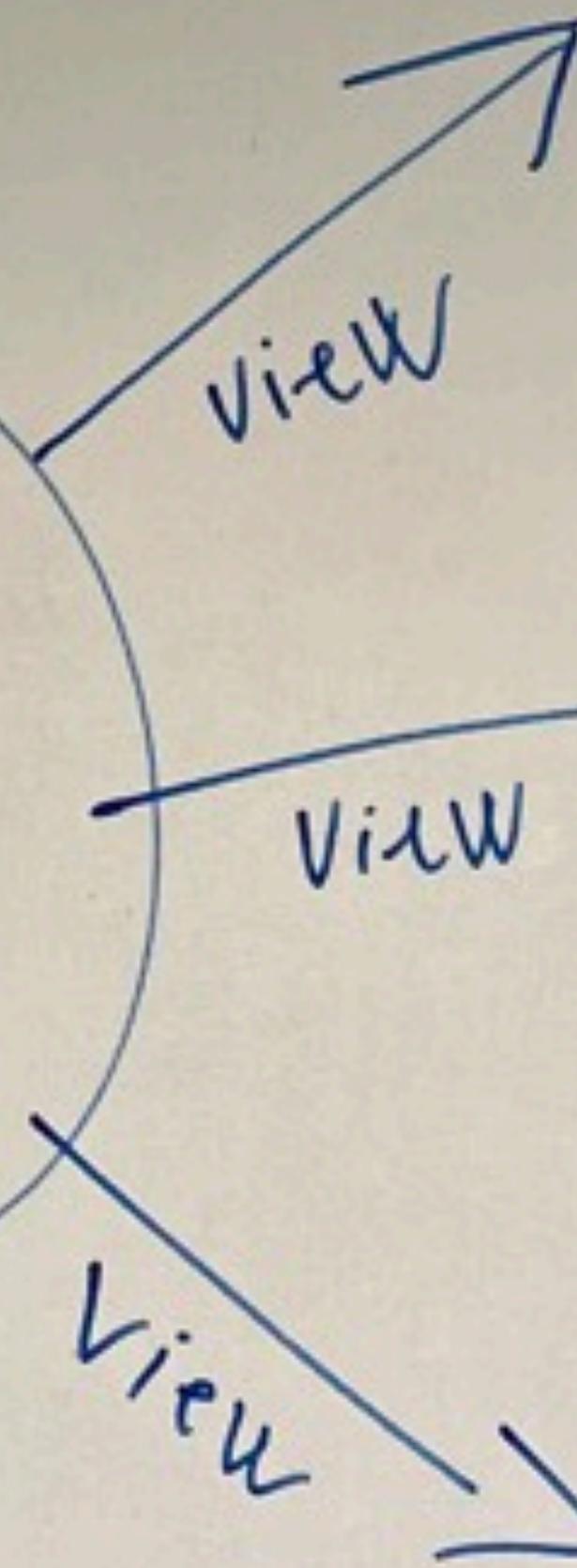
The router makes it looks like the web app has multiple pages and HTML files. But it's just changing the view, what is displayed, in side of the single `index.html` file. The page transition and change of content, is done dynamically by JavaScript (`router.js`).

[https://medium.com/@fro\\_g/routing-in-javascript-d552ff4d2921](https://medium.com/@fro_g/routing-in-javascript-d552ff4d2921)



/ create  
route

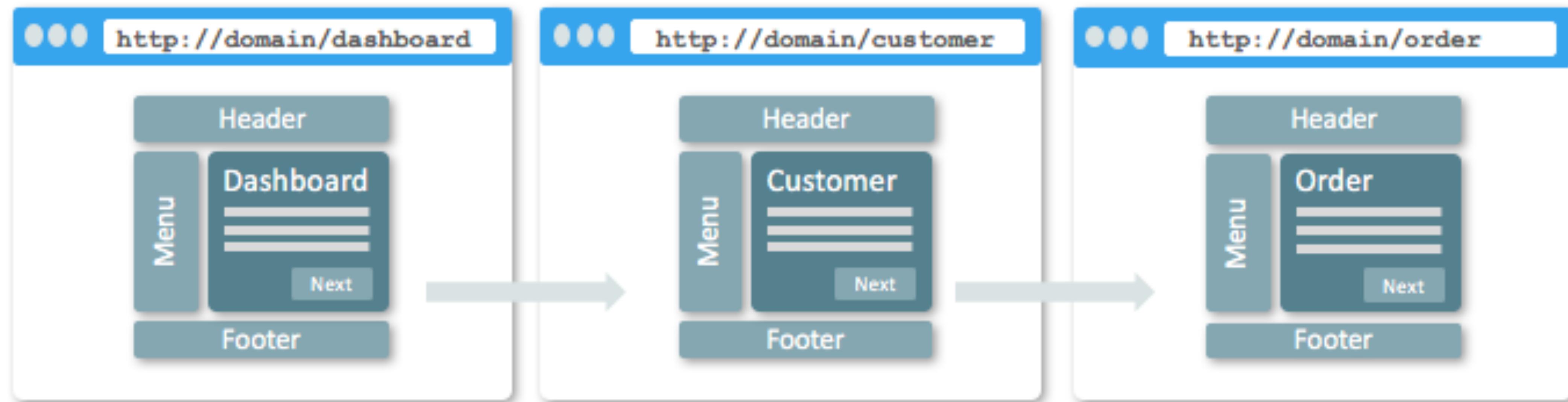
Router  
(router.js)



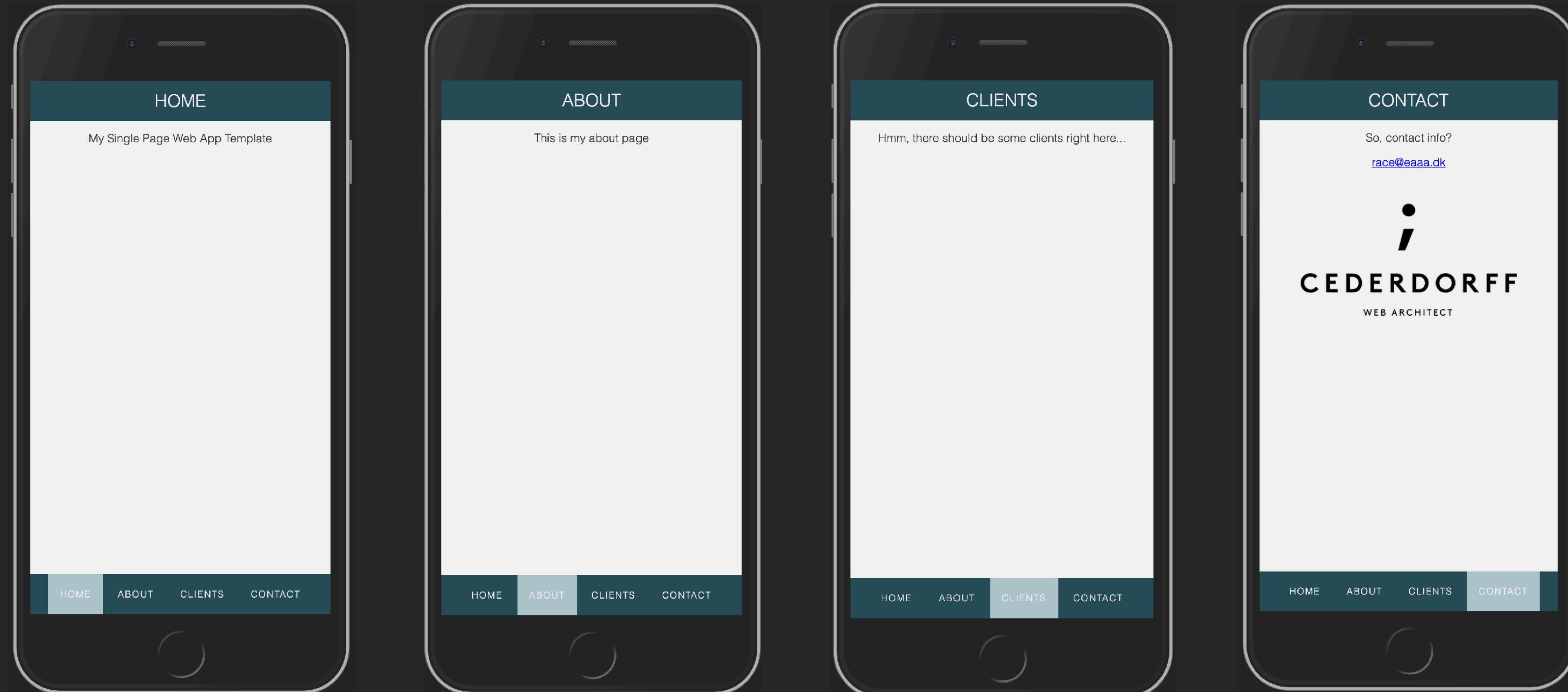
users page

create page

update page

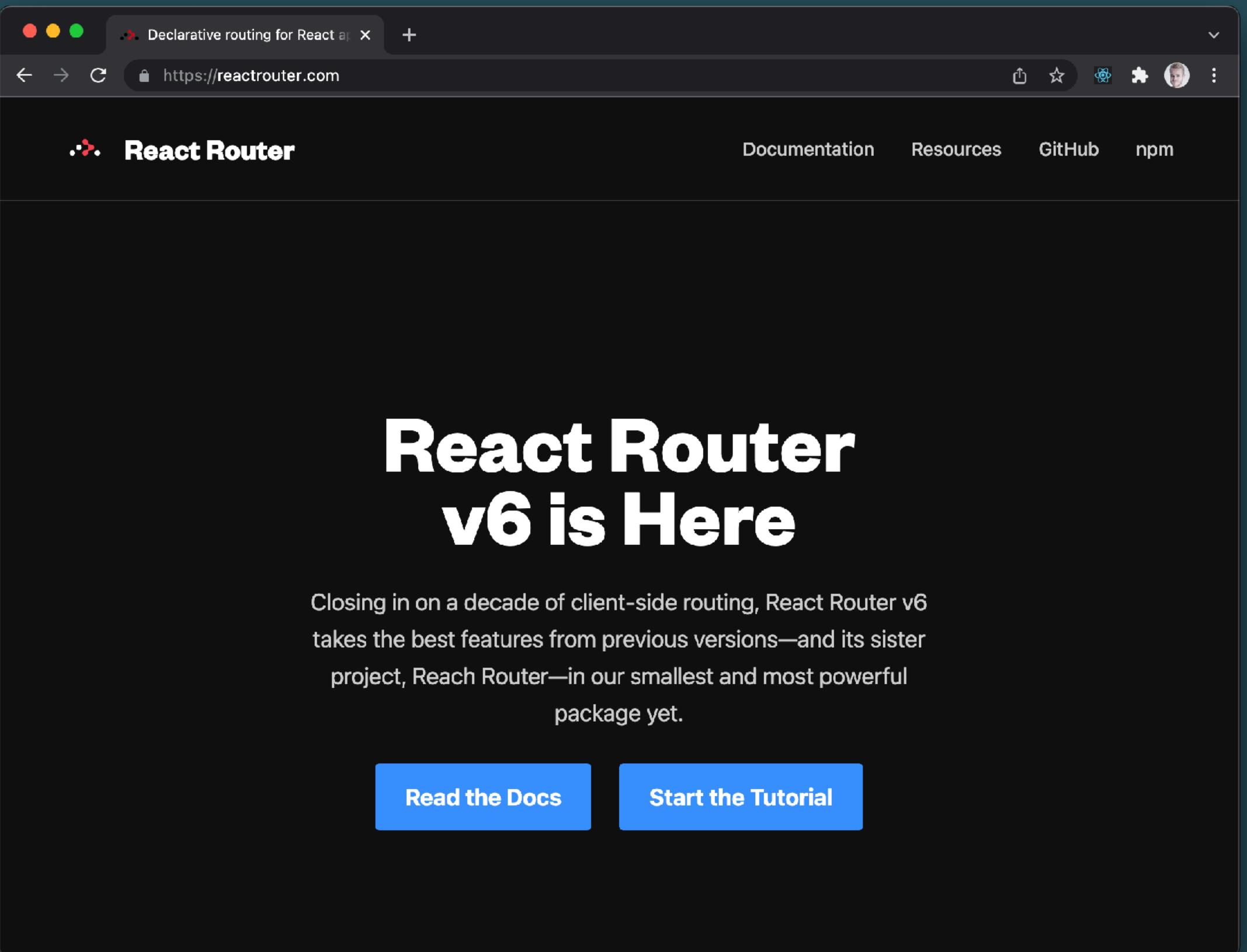


<https://dzone.com/articles/single-page-applications-in-oracle-jet>



# Create React SPA

With React Router



Create React SPA

# React CRUD

React Firebase REST Post App https://race-rest.web.app

POSTS CREATE

 Morten Algy Bonderup  
Senior Lecturer



**Qui est esse**  
Est rerum tempore vitae sequi sint nihil reprehenderit dolor beatae ea dolores neque fugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis qui aperiam non debitis possimus qui neque nisi nulla

 Dan Okkels Brendstrup  
Lecturer



**Consequuntur deleniti eos quia temporibus ab aliquid at**  
Voluptatem cumque tenetur consequatur expedita ipsum nemo quia explicabo aut eum minima consequatur tempore cumque quae est et et in consequuntur voluptatem voluptates aut

 Kim Elkjær Marcher-Jepsen  
Senior Lecturer



**At nam consequatur ea labore ea harum**  
Cupiditate quo est a modi nesciunt soluta ipsa voluptas error itaque dicta in autem qui minus magnam et distinctio eum accusamus ratione error aut

 Birgitte Kirk Iversen  
Senior Lecturer



 Jes Arbov  
Lecturer

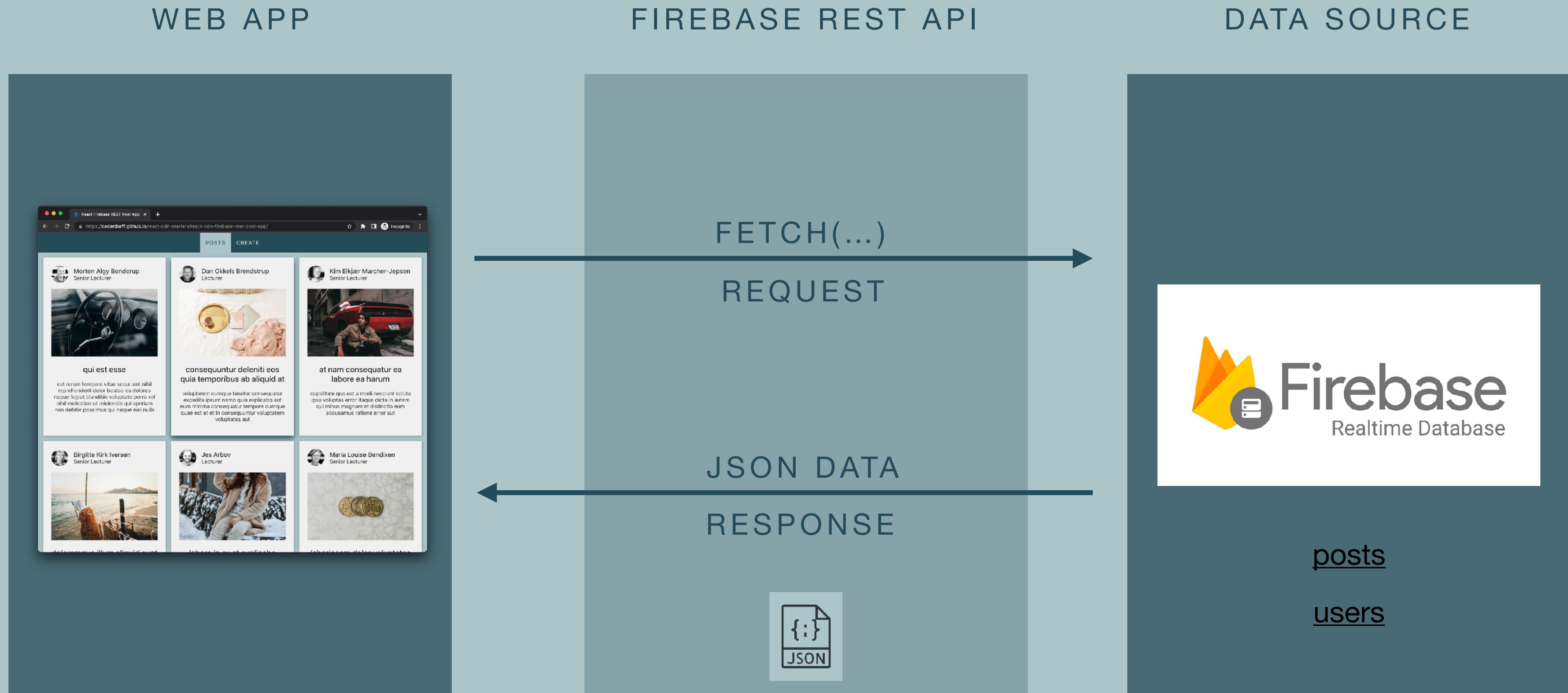


 Maria Louise Bendixen  
Senior Lecturer

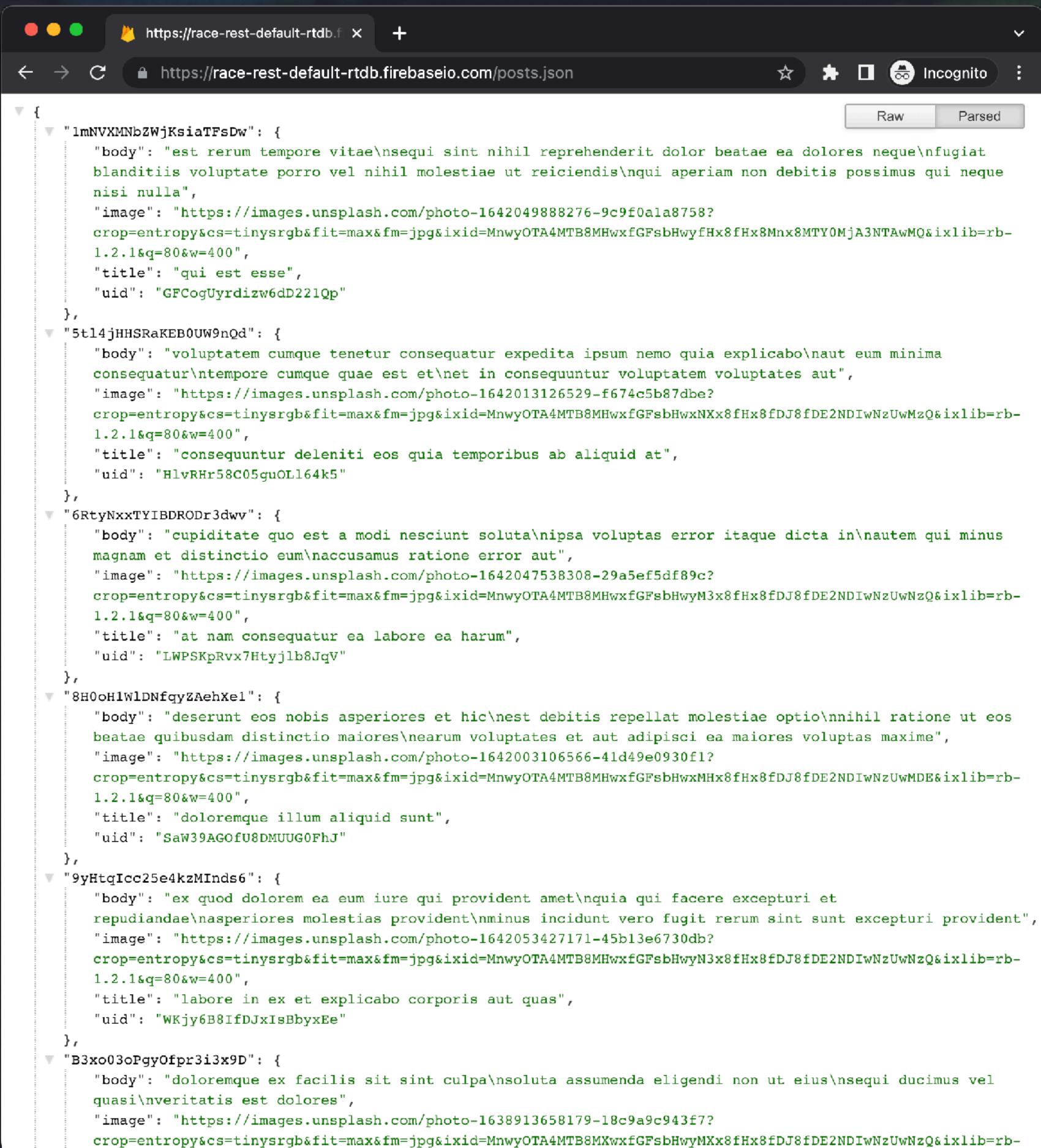


<https://race-rest.web.app/>

# Web Development



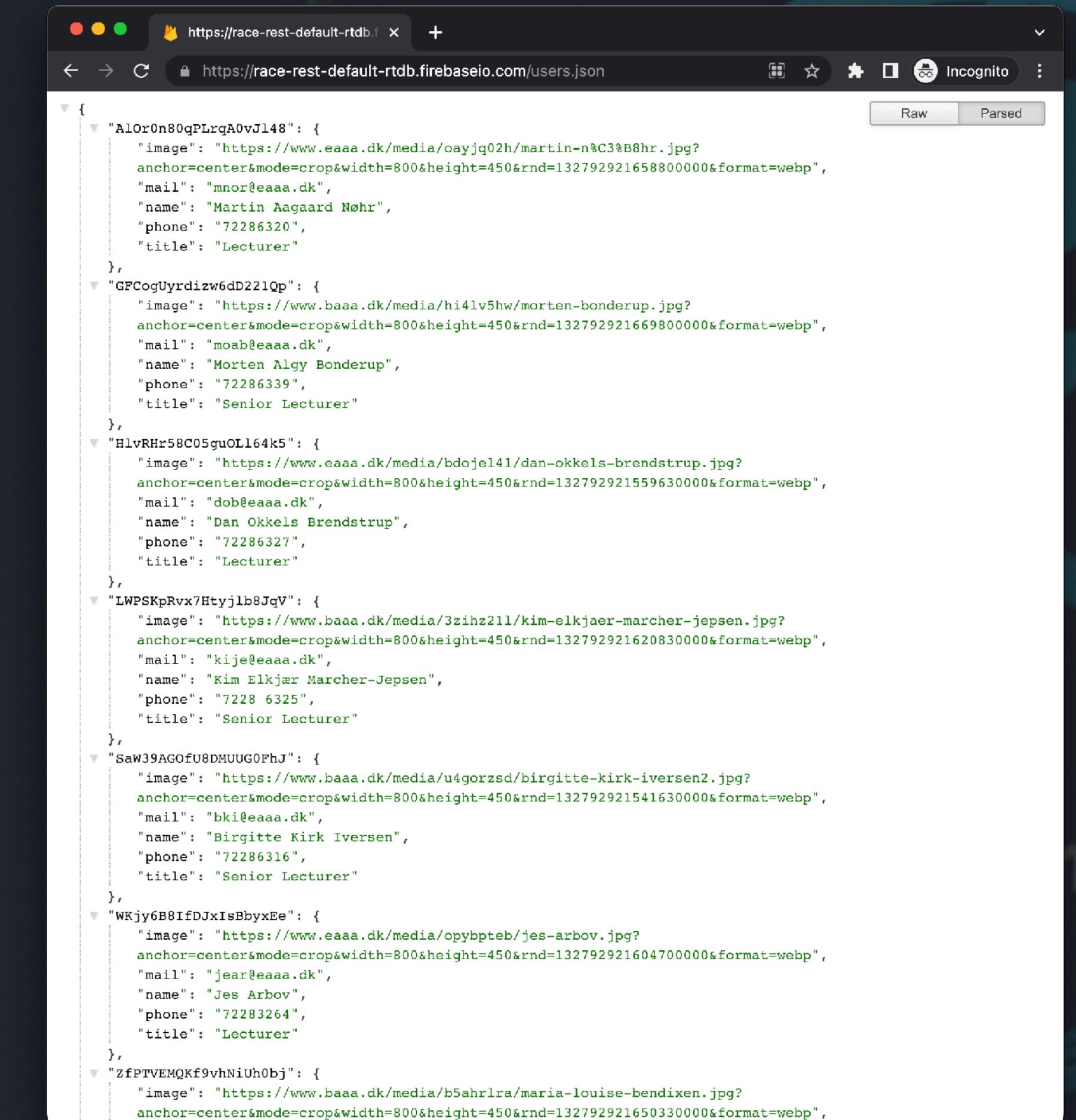
# Two data sources



```
https://race-rest-default.firebaseio.com/posts.json
```

```
{ "1mNVXMMNbZWjKsiaTFsDw": { "body": "est rerum tempore vitae\\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\\nqui aperiam non debitis possimus qui neque nisi nulla", "image": "https://images.unsplash.com/photo-1642049888276-9c9f0ala8758?crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwyOTA4MTB8MHwxfgFsbHwyfHx8fHx8Mnx8MTY0MjA3NTAwMQ&ixlib=rb-1.2.1&q=80&w=400", "title": "qui est esse", "uid": "GFCogUyrdizw6dD221Qp" }, "5t14jHHSRAKEB0UW9npd": { "body": "voluptatem cumque tenetur consequatur expedita ipsum nemo quia explicabo\\naut eum minima consequatur\\ntempore cumque quae est et\\net in consequuntur voluptatem voluptates aut", "image": "https://images.unsplash.com/photo-1642013126529-f674c5b87dbe?crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwyOTA4MTB8MHwxfgFsbHwxNx8fHx8fDJ8fDE2NDIwNzUwMzQ&ixlib=rb-1.2.1&q=80&w=400", "title": "consequuntur deleniti eos quia temporibus ab aliquid at", "uid": "HlvRHR58C05guOLl64k5" }, "6RtyNxxTYIBDRDr3dwv": { "body": "cupiditate quo est a modi nesciunt soluta\\nipsa voluptas error itaque dicta in\\nautem qui minus magnam et distinctio eum\\naccusamus ratione error aut", "image": "https://images.unsplash.com/photo-1642047538308-29a5ef5df89c?crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwyOTA4MTB8MHwxfgFsbHwyM3x8fHx8fDJ8fDE2NDIwNzUwNzQ&ixlib=rb-1.2.1&q=80&w=400", "title": "at nam consequatur ea labore ea harum", "uid": "LWPSKpRvx7Htyjl8JqV" }, "8H0oH1wLDNfgyZAhXel": { "body": "deserunt eos nobis asperiores et hic\\nest debitis repellat molestiae optio\\nnihil ratione ut eos beatae quibusdam distinctio maiores\\nearum voluptates et aut adipisci ea maiores voluptas maxime", "image": "https://images.unsplash.com/photo-1642003106566-41d49e0930f1?crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwyOTA4MTB8MHwxfgFsbHwxMHx8fHx8fDJ8fDE2NDIwNzUwMDE&ixlib=rb-1.2.1&q=80&w=400", "title": "doloremque illum aliquid sunt", "uid": "SaW39AGOfU8DMUUG0PhJ" }, "9yHtqIcc25e4kzMInds6": { "body": "ex quod dolorum ea eum iure qui provident amet\\nquia qui facere excepturi et repudiandae\\nasperiores molestias provident\\nminus incidente vero fugit rerum sint sunt excepturi provident", "image": "https://images.unsplash.com/photo-1642053427171-45b1e6730db?crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwyOTA4MTB8MHwxfgFsbHwyN3x8fHx8fDJ8fDE2NDIwNzUwNzQ&ixlib=rb-1.2.1&q=80&w=400", "title": "labore in ex et explicabo corporis aut quas", "uid": "WKjy6B8IfDJxIsBbyxEe" }, "B3x003oPgyOfpr3i3x9D": { "body": "doloremque ex facilis sit sint culpa\\nsoluta assumenda eligendi non ut eius\\nsequi ducimus vel quasi\\nveritatis est dolores", "image": "https://images.unsplash.com/photo-1638913658179-18c9a9c943f7?crop=entropy&cs=tinysrgb&fit=max&fm=jpg&ixid=MnwyOTA4MTB8MXxfGFsbHwyMxx8fHx8fDJ8fDE2NDIwNzUwNzQ&ixlib=rb-1.2.1&q=80&w=400" } }
```

[race-rest-default.firebaseio.com/posts.json](https://race-rest-default.firebaseio.com/posts.json)

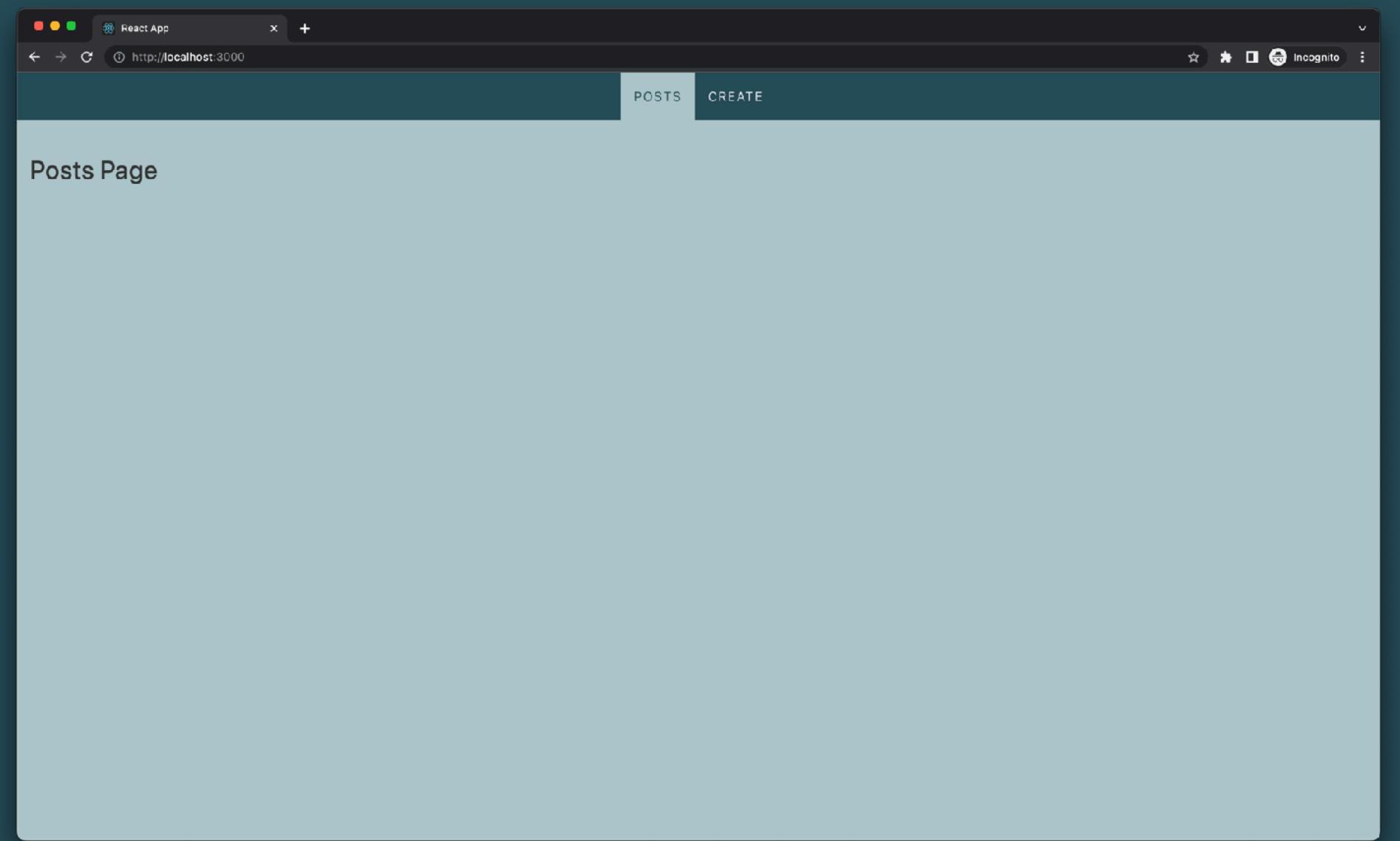


```
https://race-rest-default.firebaseio.com/users.json
```

```
{ "AlOr0n80qPLrqA0vJ148": { "image": "https://www.eaaa.dk/media/oayjq02h/martin-n%C3%B8hr.jpg?", "anchor": "center", "mode": "crop", "width": 800, "height": 450, "rnd": 132792921658800000, "format": "webp", "mail": "mnor@eaaa.dk", "name": "Martin Aagaard Nøhr", "phone": "72286320", "title": "Lecturer" }, "GFCogUyrdizw6dD221Qp": { "image": "https://www.baaa.dk/media/hi4lv5hw/morten-bonderup.jpg?", "anchor": "center", "mode": "crop", "width": 800, "height": 450, "rnd": 132792921669800000, "format": "webp", "mail": "moab@eaaa.dk", "name": "Morten Algy Bonderup", "phone": "72286339", "title": "Senior Lecturer" }, "HlvRHR58C05guOLl64k5": { "image": "https://www.eaaa.dk/media/bdoje141/dan-okkels-brendstrup.jpg?", "anchor": "center", "mode": "crop", "width": 800, "height": 450, "rnd": 132792921559630000, "format": "webp", "mail": "dob@eaaa.dk", "name": "Dan Okkels Brendstrup", "phone": "72286327", "title": "Lecturer" }, "LWPSKpRvx7Htyjl8JqV": { "image": "https://www.baaa.dk/media/3zihz211/kim-elkjær-marcher-jepsen.jpg?", "anchor": "center", "mode": "crop", "width": 800, "height": 450, "rnd": 132792921620830000, "format": "webp", "mail": "kije@eaaa.dk", "name": "Kim Elkjær Marcher-Jepsen", "phone": "7228 6325", "title": "Senior Lecturer" }, "SaW39AGOfU8DMUUG0PhJ": { "image": "https://www.baaa.dk/media/u4gorzsdbirgitte-kirk-iversen2.jpg?", "anchor": "center", "mode": "crop", "width": 800, "height": 450, "rnd": 132792921541630000, "format": "webp", "mail": "bki@eaaa.dk", "name": "Birgitte Kirk Iversen", "phone": "72286316", "title": "Senior Lecturer" }, "WKjy6B8IfDJxIsBbyxEe": { "image": "https://www.eaaa.dk/media/opybpptebsjes-arbov.jpg?", "anchor": "center", "mode": "crop", "width": 800, "height": 450, "rnd": 132792921604700000, "format": "webp", "mail": "jeair@eaaa.dk", "name": "Jes Arbov", "phone": "72283264", "title": "Lecturer" }, "zfPTVEMQKF9vhNiUh0bj": { "image": "https://www.baaa.dk/media/b5ahrlra/maria-louise-bendixen.jpg?", "anchor": "center", "mode": "crop", "width": 800, "height": 450, "rnd": 132792921650330000, "format": "webp" } }
```

[race-rest-default.firebaseio.com/users.json](https://race-rest-default.firebaseio.com/users.json)

# Boilerplate



react-post-app-boilerplate

- Download: [react-post-app-boilerplate](#)
- Paste project folder in your local developer folder.
- Open folder in VS Code.
- Make sure you are in the root of the project.
- Open the terminal in VS Code and run `npm install`

# Read

GET is the default request method for fetch.

```
import { useState, useEffect } from "react";
import PostCard from "../components/PostCard";

export default function HomePage() {
  const [posts, setPosts] = useState([]);

  useEffect(() => {
    async function getPosts() {
      const url = "https://race-rest-default.firebaseio.com/posts.json";
      const response = await fetch(url);
      const data = await response.json();
      const postsArray = Object.keys(data).map(key => ({ id: key, ...data[key] }));
      setPosts(postsArray);
    }
    getPosts();
  }, []);

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

# PostCard

Use the prop post to implement the structure of the PostCard component. Skip handleClick for now.

```
import { useNavigate } from "react-router-dom";

export default function PostCard({ post }) {
  const navigate = useNavigate();

  /**
   * handleClick is called when user clicks on the Article (PostCard)
   */
  function handleClick() {
    navigate(`posts/${post.id}`);
  }

  return (
    <article onClick={handleClick}>
      <img src={post.image} alt={post.title} />
      <h2>{post.title}</h2>
      <p>{post.body}</p>
    </article>
  );
}
```

# PostCard

Add user details on every PostCard. Add it in PostCard or create a separate component to handle the logic. I created a separate UserAvatar component.

```
import { useNavigate } from "react-router-dom";
import UserAvatar from "./UserAvatar";

export default function PostCard({ post }) {
  const navigate = useNavigate();

  /**
   * handleClick is called when user clicks on the Article (PostCard)
   */
  function handleClick() {
    navigate(`posts/${post.id}`);
  }

  return (
    <article onClick={handleClick}>
      <UserAvatar uid={post.uid} />
      <img src={post.image} alt={post.title} />
      <h2>{post.title}</h2>
      <p>{post.body}</p>
    </article>
  );
}
```

# UserAvatar

Based on the prop uid, user detail is fetched from Firebase and saved in a state. Note that the user state is initialised with a default object with placeholders.

```
import { useState, useEffect } from "react";
import placeholder from "../assets/img/user-placeholder.jpg";

export default function UserAvatar({ uid }) {
  const [user, setUser] = useState({
    image: placeholder,
    name: "User's Name",
    title: "User's Title"
  });
  const url = `https://race-rest-default-firebaseio.com/users/${uid}.json`;

  useEffect(() => {
    async function getUser() {
      const response = await fetch(url);
      const data = await response.json();
      console.log(data);
      setUser(data);
    }
    getUser();
  }, [url]);

  return (
    <div className="avatar">
      <img src={user.image} alt={user.id} />
      <span>
        <h3>{user.name}</h3>
        <p>{user.title}</p>
      </span>
    </div>
  );
}
```

# Create

Implement a form with a field for title, body (description) and image. Also add an image preview. Add states for every field including and error message state.

```
export default function CreatePage() {
  const [title, setTitle] = useState("");
  const [body, setBody] = useState("");
  const [image, setImage] = useState("");
  const [errorMessage, setErrorMessage] = useState("");

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Title
        <input type="text" value={title} placeholder="Type a title" onChange={e => setTitle(e.target.value)} />
      </label>
      <label>
        Body
        <input type="text" value={body} placeholder="Type a body text" onChange={e => setBody(e.target.value)} />
      </label>
      <label>
        Image
        <input type="file" className="file-input" accept="image/*" onChange={handleImageChange} />
        <img className="image-preview" src={image} alt="Choose" onError={event => (event.target.src = imgPlaceholder)} />
      </label>
      <p className="text-error">{errorMessage}</p>
      <button type="submit">Save</button>
    </form>
  );
}
```

# Create

## Implement handleImageChange

```
/**  
 * handleImageChange is called every time the user chooses an image in the fire system.  
 * The event is fired by the input file field in the form  
 */  
function handleImageChange(event) {  
    const file = event.target.files[0];  
    if (file.size < 500000) {  
        // image file size must be below 0,5MB  
        const reader = new FileReader();  
        reader.onload = event => {  
            setImage(event.target.result);  
        };  
        reader.readAsDataURL(file);  
        setErrorMessage(""); // reset errorMessage state  
    } else {  
        // if not below 0.5MB display an error message using the errorMessage state  
        setErrorMessage("The image file is too big!");  
    }  
}
```

# Create

onSubmit call handleSubmit. The function creates a new post object based on the states. Also add a hardcoded uid = "fTs84KR0Yw5pRZEWCq2Z"  
Use fetch and post to post the new post object to the data source.

```
async function handleSubmit() {
  const newPost = {
    title: title,
    body: body,
    image: image,
    uid: "fTs84KR0Yw5pRZEWCq2Z" // default user id added
};

const url = "https://race-rest-default-rtdb.firebaseio.com/posts.json";
const response = await fetch(url, {
  method: "POST",
  body: JSON.stringify(newPost)
});
const data = await response.json();
console.log(data);
navigate("/");
}
```

# Update

Using PUT to an  
existing post object  
by given id.

You have two choices:

- Implement an update form similar to the create form.
- Or implement a [PostForm](#) component you can reuse for CreatePage and UpdatePage.

# Update

## PostForm

### Component #1

```
import { useEffect, useState } from "react";
import imgPlaceholder from "../assets/img/img-placeholder.jpg";

export default function PostForm({ savePost, post }) {
  const [title, setTitle] = useState("");
  const [body, setBody] = useState("");
  const [image, setImage] = useState("");
  const [errorMessage, setErrorMessage] = useState("");

  useEffect(() => {
    if (post) {
      // if post, set the states with values from the post object.
      // The post object is a prop, passed from UpdatePage
      setTitle(post.title);
      setBody(post.body);
      setImage(post.image);
    }
  }, [post]); // useEffect is called every time post changes.

  /**
   * handleImageChange is called every time the user chooses an image in the fire system.
   * The event is fired by the input file field in the form
   */
  function handleImageChange(event) {
    const file = event.target.files[0];
    if (file.size < 500000) {
      // image file size must be below 0,5MB
      const reader = new FileReader();
      reader.onload = event => {
        setImage(event.target.result);
      };
      reader.readAsDataURL(file);
      setErrorMessage(""); // reset errorMessage state
    } else {
      // if not below 0.5MB display an error message using the errorMessage state
      setErrorMessage("The image file is too big!");
    }
  }
}
```

# Update

## PostForm

### Component #2

```
function handleSubmit(event) {
  event.preventDefault();
  const formData = {
    // create a new object to hold the value from states / input fields
    title: title,
    image: image,
    body: body
  };

  const validForm = formData.title && formData.body && formData.image; // will return false if one of the properties doesn't have a value
  if (validForm) {
    // if all fields/ properties are filled, then call savePost
    savePost(formData);
  } else {
    // if not, set errorMessage state.
    setErrorMessage("Please, fill in all fields.");
  }
}

return (
  <form onSubmit={handleSubmit}>
    <label>
      Title
      <input type="text" value={title} placeholder="Type a title" onChange={e => setTitle(e.target.value)} />
    </label>
    <label>
      Body
      <input type="text" value={body} placeholder="Type a body text" onChange={e => setBody(e.target.value)} />
    </label>
    <label>
      Image
      <input type="file" className="file-input" accept="image/*" onChange={handleImageChange} />
        <img className="image-preview" src={image} alt="Choose" onError={event => (event.target.src = imgPlaceholder)} />
    </label>
    <p className="text-error">{errorMessage}</p>
    <button type="submit">Save</button>
  </form>
);
}
```

# Update

## CreatePage Component

```
import { useNavigate } from "react-router-dom";
import PostForm from "../components/PostForm";

export default function CreatePage({ showLoader }) {
  const navigate = useNavigate();

  async function createPost(newPost) {
    newPost.uid = "fTs84KR0Yw5pRZEWCq2Z"; // default user id added

    const url = "https://race-rest-default-firebase.firebaseio.com/posts.json";
    const response = await fetch(url, {
      method: "POST",
      body: JSON.stringify(newPost)
    });
    const data = await response.json();
    console.log(data);
    navigate("/");
  }

  return (
    <section className="page">
      <h1>Create New Post</h1>
      <PostForm savePost={createPost} />
    </section>
  );
}
```

# Update

## UpdatePage Component

```
import { useEffect, useState } from "react";
import { useNavigate, useParams } from "react-router-dom";
import PostForm from "../components/PostForm";

export default function UpdatePage() {
  const [post, setPost] = useState({});
  const params = useParams();
  const navigate = useNavigate();
  const url = `https://race-rest-default-firebase.com/posts/${params.postId}.json`;

  useEffect(() => {
    async function getPost() {
      const response = await fetch(url);
      const data = await response.json();
      setPost(data);
    }

    getPost();
  }, [url]);

  async function savePost(postToUpdate) {
    const response = await fetch(url, {
      method: "PUT",
      body: JSON.stringify(postToUpdate)
    });
    const data = await response.json();
    console.log(data);
    navigate("/");
  }

  return (
    <section className="page">
      <h1>Update Post</h1>
      <PostForm post={post} savePost={savePost} />
    </section>
  );
}
```

# Delete

Using DELETE to an object  
by a given id.  
Add deletePost logic in  
UpdatePage component.

```
import { useEffect, useState } from "react";
import { useNavigate, useParams } from "react-router-dom";
import PostForm from "../components/PostForm";

export default function UpdatePage() {
  const [post, setPost] = useState({});
  const params = useParams();
  const navigate = useNavigate();
  const url = `https://race-rest-default-firebase.firebaseio.com/posts/${params.postId}.json`;

  useEffect(() => {
    // ...
  }, [url]);

  async function savePost(postToUpdate) {
    // ...
  }

  async function deletePost() {
    const confirmDelete = window.confirm(`Do you want to delete post, ${post.title}?`);
    if (confirmDelete) {
      const response = await fetch(url, {
        method: "DELETE"
      });
      const data = await response.json();
      console.log(data);
      navigate("/");
    }
  }

  return (
    <section className="page">
      <h1>Update Post</h1>
      <PostForm post={post} savePost={savePost} />
      <button className="btn-delete" onClick={deletePost}>
        Delete Post
      </button>
    </section>
  );
}
```

# 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 of the page, 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.

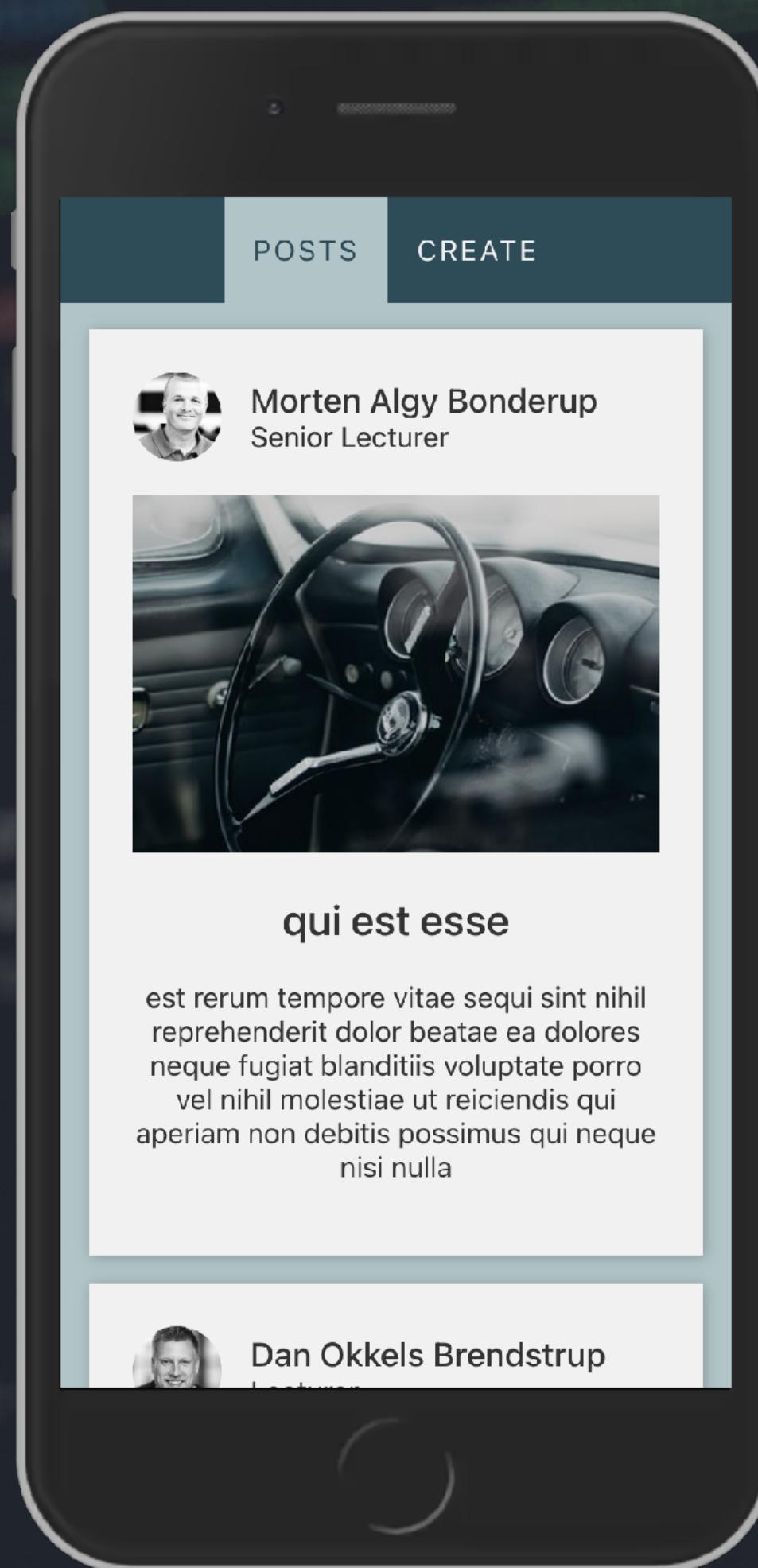
We can rewrite `result = a ?? b` using the operators that we already know, like this:

```
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>

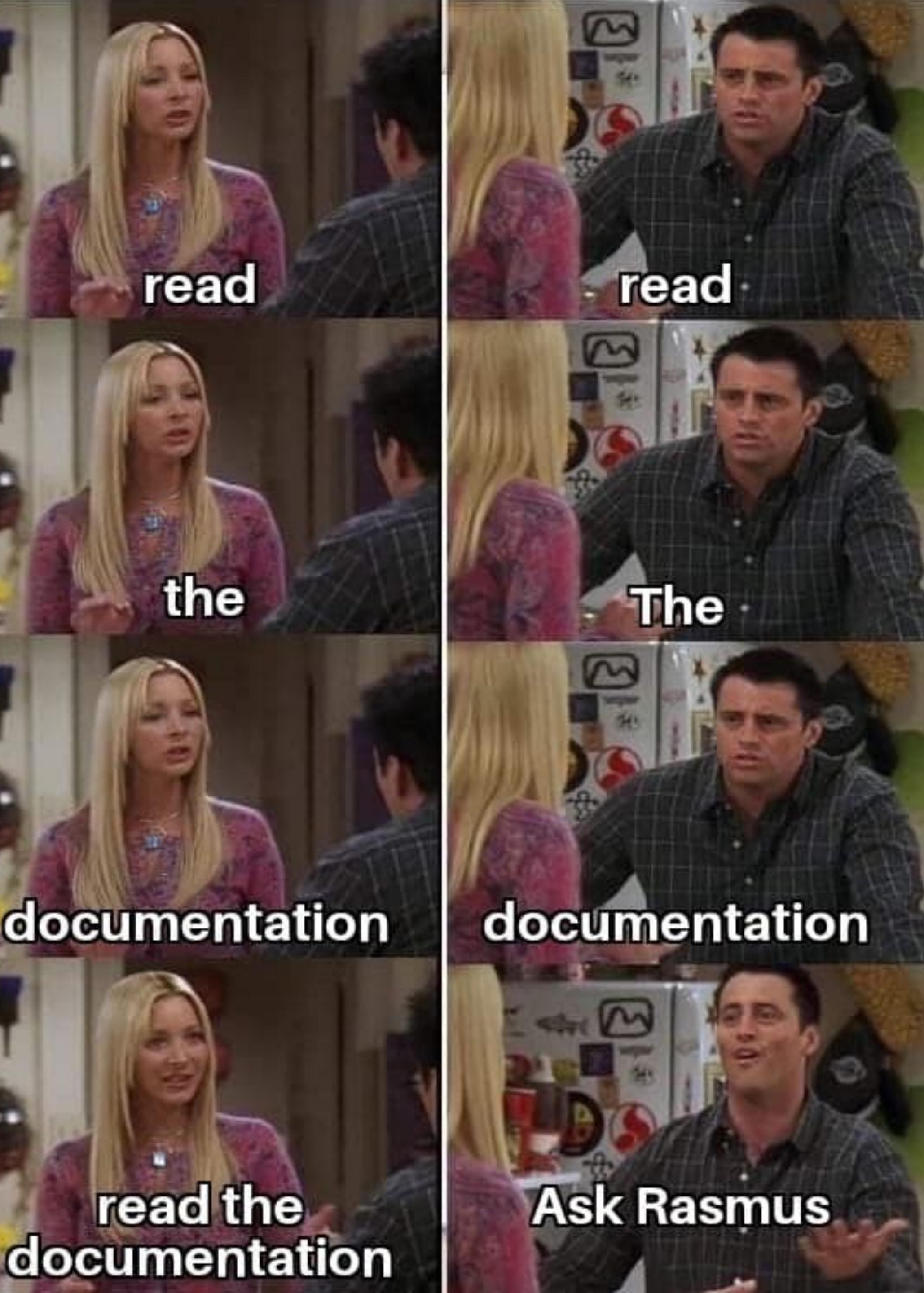
# Solution Proposal



- GitHub Repo (code)
- Running solution
- Data Sources:
  - posts
  - users

# But RACE, how do you know all that stuff?

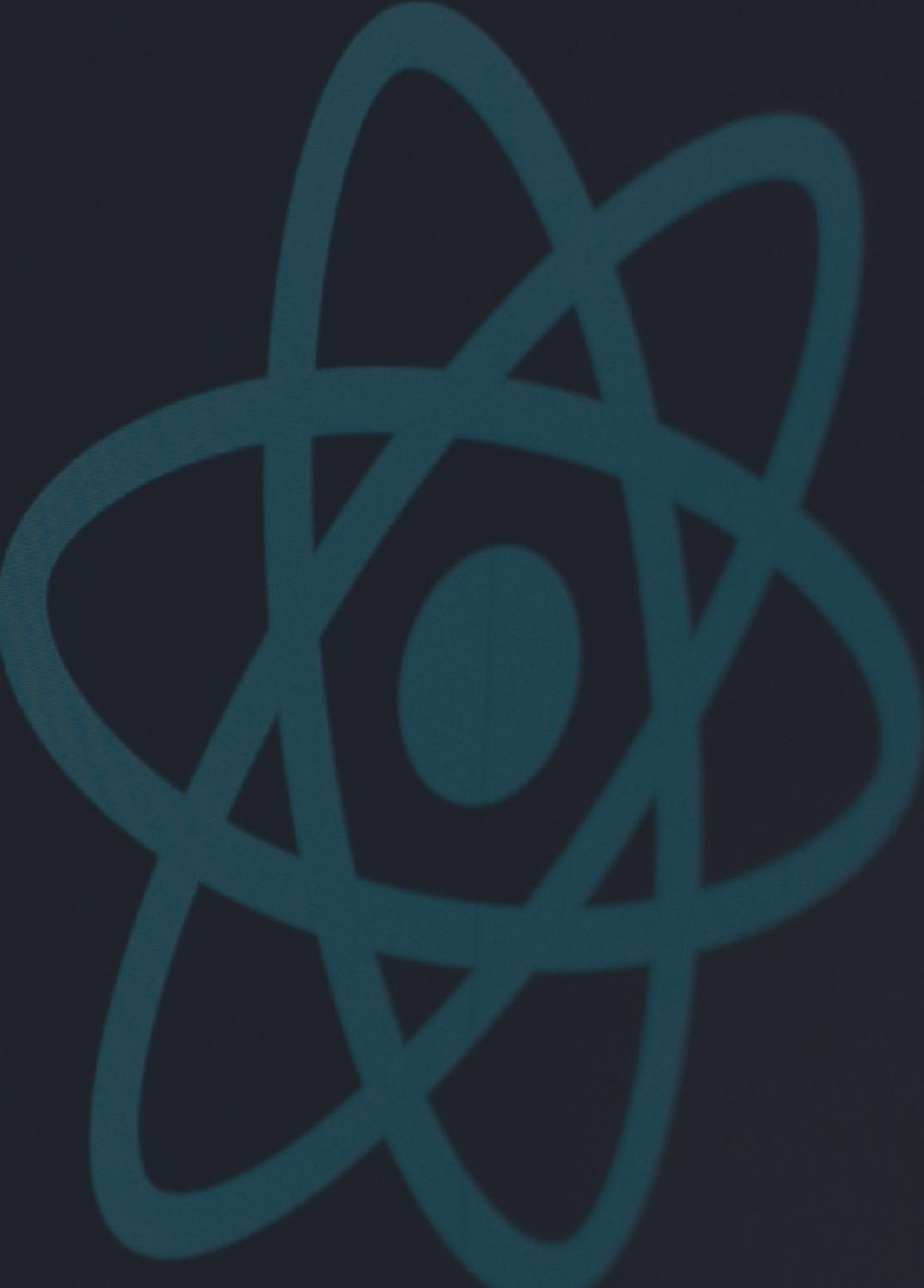
- [React Resources](#)
- [React Self-study](#)
- [Guides & Tutorials](#)
- [From Vanilla JavaScript to React Developer](#)



# 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.  
Inspiration on GitHub: [react-starters](#) and [react-cdn-starters](#)
3. Checkout [React Self-study](#) and [Guides & Tutorials](#)



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

# Code Every Day

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

