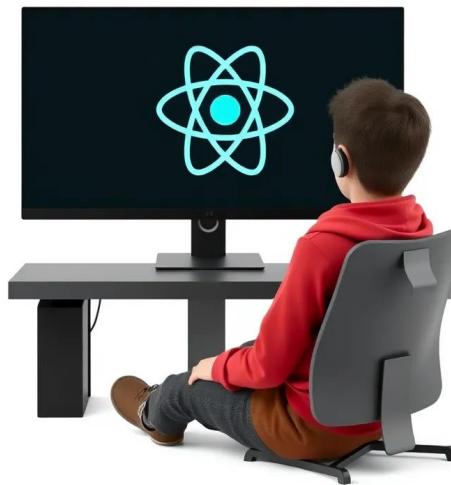
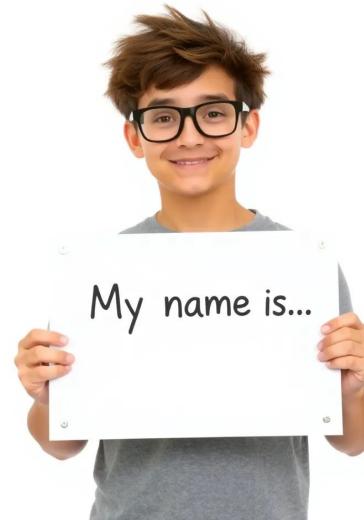


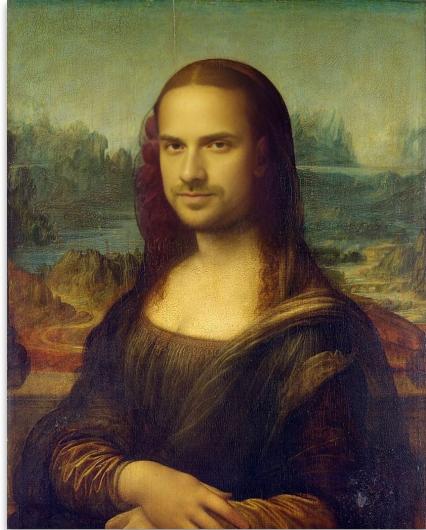
Introduction to React



Introduction Round

- Name
- Current area of responsibility in the company
- Prior knowledge in Web Development
- An interesting fact about yourself (optional)





- **Entwickler seit 1590 (geschätzt)** – Schon in den 90er Jahren Webseiten mit HTML & JavaScript gebaut.
 - **Von Entwicklung zu KI & Sicherheit** – Technik-Enthusiast, aber JavaScript war meine erste “Entwicklerliebe”.
 - **Informatik Master & MBA** – Eine Mischung aus technischer Expertise und Unternehmertum.
 - **Theorie? Klar. Praxis? Noch mehr.** – Vom Entwickler zum CTO – ich habe Einiges gesehen.
- ▶ <https://www.youtube.com/@ai-for-devs>



Vergangene React Projekte

Highenobiety – Shop & Culture ⓘ
Highend, Fashion, Sneakers, Style
Titel Media GmbH
★★★★★ 4.8 - 270 Ratings
[Folge](#)

iPhone Screenshots

The Highenobiety app puts the best of sneakers, luxury men's fashion, designer brands, and cultural content right at your fingertips. With a global shop from over 100+ brands featuring exclusives you won't find anywhere else. From established fashion houses to emerging labels, Highenobiety helps you stay ahead of the curve and find what's next in style.

What's New
We've made important improvements to your story-browsing experience with smoother navigation and improved responsiveness. Update to enjoy a more reliable and enjoyable way to explore stories!

Ratings and Reviews
4.8 out of 5
210 Ratings

Highenobiety App Review
Highenobiety is a standout platform in the realm of streetwear and contemporary culture. Without a doubt, it's a must-have app for those who are regulars and newcomers.

Stay Ahead of the Curve
Current events and latest trends.

Shop What's Next
Products that makes fashion history.

New Arrivals
Breaking news and daily updates.

In the Know
Breaking news and daily updates.

Strenger ⓘ
Haus, Wohnung, Objekt, Kauf, Mieten, Services, Über Strenger
BAU STOLZ

Baukonzept Haus / Wohnungsgut Bauleitplan Hochnahigkeit Ausbauhaus Standorte Kapitalanlage

Baukonzept von Strenger. Mehr Immobilie fürs Geld geht nicht.

Einfach clever, vom Fundament bis unter das Dach.

Unser Baukonzept ist ein intelligenter Planer, der Ihnen die optimale Nutzung Ihres Grundstücks ermöglicht. Unser hochwertiges, nachhaltiges und vorgeplante Bauwerk und Wohnungsbaurecht zu modernen Dachterrassen, nachhaltige und vorgeplante Bauweise und eine perfekte Planung wird nicht nur ein Festpreis, sondern auch eine optimale Nutzung des Grundstücks. Ein Baukonzept ist eine optimale Nutzung des Grundstücks und Wohnungen können im Baukonzept-Konfigurator online konfiguriert werden. Nach einem Mausklick sind Haus oder Wohnung fertiggestellt und ein zuverlässiger Fachgeschäftsmeister kümmert sich darum.

Unsere Baustolz Haustypen in verschiedenen Designs.

Carhartt WIP ⓘ
Work In Progress Textilhandels GmbH
Entwickelt für iPad
Gratis
[Anzeigen in Mac App Store](#) ⓘ

Screenshots ⓘ
iPad iPhone

The Carhartt WIP App
Easily discover, shop, and manage your purchases with the Carhartt WIP App, which features intuitive browsing and an efficient search function, as well as a wishlist to store and keep track of your favorite items. The app also offers shipping to 24 countries.

Neuheiten
Functionality first — we have made some improvements to the app including various bug fixes.

Bewertungen und Rezensionen
4,8 von 5
362 Bewertungen

Carhartt WIP App Review
Gut gestalteter App
Leider keine Info über den Bezahlungsaufwand in der App nach Bestellung noch keine Verbandsbestätigung. Über den Webkoffer wurde bisher spätestens 2 Tage nach Eingabe

Alle anzeigen

Expectations

- Grundsätzliches Verständnis
- Wie arbeite ich mit React
- States
- TypeScript Grundlagen
- Grundwissen auf dem man weiter aufbauen kann
- React Umfeld (Anbindung an System, Kafka)
- APIs (REST, GraphQL)
- Umgang mit Tabellen (Nutzereingabe, Relationale Datenbank simulieren)
- Debugging

- Microservices abwegen
- Node (Basics)
- AWS (wird inHouse genutzt), Hosting-Möglichkeiten (Scaling)
- Dynamo (key, value)
- ...

Day 1: React Fundamentals and Initial Steps

- JavaScript Basics
- Overview of React and Its Ecosystem
- Insights into Virtual DOM Functionality
- Fundamental Principles of React
- Understanding Components and Properties
- Comprehensive Analysis of JSX
- Initiating Your First Project with Next.js
- Methods of Conditional Rendering in React
- Approaches to Render Lists in React

Day 2: Advanced React Development

- Building a Portfolio Page Using React
- Handling Events and Propagation in React
- State Management in React: Techniques and Practices
- Integrating React with External Systems
- Dynamic Routing Implementation in React
- Data Retrieval from External APIs in React
- Employing React's Context API for Efficient State Management

Day 3: Leveraging TypeScript and React Advanced Topics

- Introduction to TypeScript for React
- Utilizing Type Aliases in TypeScript
- Interface Implementation in TypeScript
- TypeScript Functions: Definition and Usage
- Generic Types in TypeScript: An Overview
- Applying TypeScript in React Projects
- Comparing Server vs Client Rendering Techniques
- Exploring React in the Context of React Native
- Basics of IT Security in Web Development

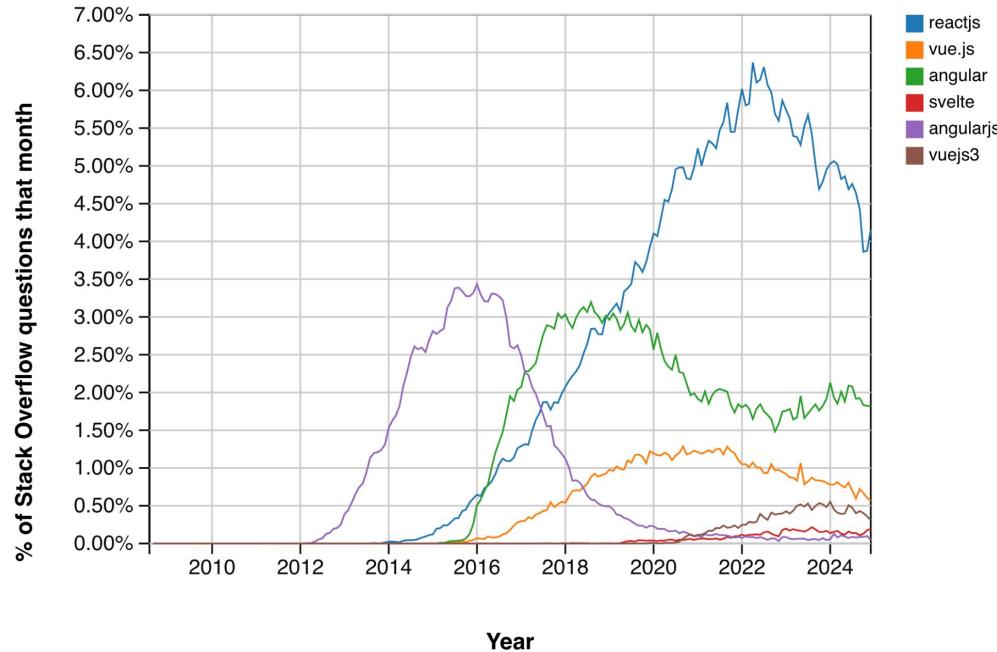
JavaScript Popularity



JavaScript Popularity

Feb 2025	Feb 2024	Change	Programming Language	Ratings	Change
1	1		 Python	23.88%	+6.72%
2	3		 C++	11.37%	+0.84%
3	4		 Java	10.66%	+1.79%
4	2		 C	9.84%	-1.14%
5	5		 C#	4.12%	-3.41%
6	6		 JavaScript	3.78%	+0.61%
7	7		 SQL	2.87%	+1.04%
8	8		 Go	2.26%	+0.53%
9	12		 Delphi/Object Pascal	2.18%	+0.78%
10	9		 Visual Basic	2.04%	+0.52%
11	11		 Fortran	1.75%	+0.35%
12	15		 Scratch	1.54%	+0.36%
13	18		 Rust	1.47%	+0.42%
14	10		 PHP	1.14%	-0.37%
15	21		 R	1.06%	+0.07%
16	13		 MATLAB	0.98%	-0.28%
17	14		 Assembly language	0.95%	-0.24%
18	19		 COBOL	0.82%	-0.18%
19	20		 Ruby	0.82%	-0.17%
20	24		Prolog	0.80%	+0.03%

React Popularity



Role of JS Developer in 2025?



JavaScript Essentials

JavaScript Essentials



**var, let,
const ?**

JavaScript Essentials

Core JavaScript knowledge is crucial for React.

- Variables & Scope: let, const, understanding function vs. block scope.
- Data Structures: Working with objects, arrays, and JSON.
- Functions: Defining and using functions, arrow functions, and scope.
- Asynchronous JS: Promises, async/await, handling asynchronous operations.
- ES6 Features: Destructuring, spread operator, template literals.
- DOM vs HTML
- Module System (Import & Export)

Variables & Scope - var

- If you declare a variable with var inside a function, it's scoped to that function.
- Outside of a function, it becomes a global variable.
- Declarations made using var are hoisted to the top of their scope.

js

[Copy](#) [Edit](#)

```
function exampleVar() {  
  console.log(x); // undefined (due to hoisting)  
  var x = 10;  
  console.log(x); // 10  
}  
exampleVar();
```

Variables & Scope - *let*

- A variable declared with let is limited to the nearest enclosing block.
- If you declare a variable with let, you cannot redeclare it again in the same scope.

js

Copy Edit

```
function exampleLet() {
  // console.log(y); // ReferenceError if uncommented (TDZ)
  let y = 10;
  console.log(y); // 10
  // let y = 20; // SyntaxError: Identifier 'y' has already been declared
}
exampleLet();
```

Variables & Scope - *global*

- Creates a global variable x on the window (in browsers) or global object (in Node.js).
- This is generally considered bad practice as it can lead to accidental global variables.
- In strict mode, this will throw a ReferenceError

```
js
```

Copy Edit

```
x = 10;
```

Variables & Scope - *const*

- Like let, a const-declared variable is scoped to the nearest enclosing block.
- A variable declared with const must be assigned an initial value.
- Once a const variable has been assigned, you cannot reassign it

js

[Copy](#)

```
const x = 10;  
x = 20; // TypeError: Assignment to constant variable
```

Essentials: Data Structures

Objects, arrays, and JSON are foundational structures in JS.

```
// Objects store key-value pairs
const person = {
  name: 'Alice',
  age: 25,
};

// Arrays hold ordered collections
const fruits = ['apple', 'banana', 'cherry'];

// JSON - Stringified object
const jsonData = JSON.stringify(person);
```

Essentials: Functions

Functions are blocks of reusable code, and arrow functions offer a concise syntax.

```
// Function declaration
function greet(name) {
  return `Hello, ${name}!`;
}

// Arrow function with implicit return
const greet = (name) => `Hello, ${name}!`;

// Arrow function with explicit return
const greet = (name) => {
  return `Hello, ${name}!`;
};
```

Essentials: Asynchronous JS

Handle asynchronous operations with Promises and async/await

```
// Promise usage
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => console.log(data));

// Async/await syntax
async function fetchData() {
  const response = await fetch('https://api.example.com/data');
  const data = await response.json();
  console.log(data);
}
```

Laufzeitumgebungen

1 **Browser:** Webbrowser bieten eine Umgebung, in der JavaScript-Code direkt auf Webseiten ausgeführt wird.

2 **Node.js:** Eine serverseitige Plattform, die es ermöglicht, JavaScript außerhalb des Browsers auszuführen.

ECMAScript und JavaScript

ECMAScript	JavaScript
Ein Standard für Skriptsprachen.	Die populärste Implementierung des ECMAScript-Standards.
Sprachen wie JS basieren auf dem ECMAScript-Standard.	Die Kernfunktionen von JavaScript basieren auf ECMAScript.
Basierend auf mehreren Technologien, darunter JavaScript (Netscape) und JScript (Microsoft).	JavaScript wurde zur Standardisierung bei ECMA eingereicht. Aufgrund von Markenrechtsproblemen mit dem Namen "JavaScript" wurde der Standard als "ECMAScript" bezeichnet.

Wichtigste ECMAScript 6 (ES6) Features

- **Klassen:** Einführung von Klassen-basierten Objektorientierung.
- **Pfeilfunktionen (=>):** Kompaktere Syntax für anonyme Funktionen und kein Binden von this.
- **Template Strings (` `):** Einfache String-Interpolation und Mehrzeilige Strings.
- **Destrukturierung:** Schnelles Extrahieren von Werten aus Arrays oder Objekten
- **let & const:** Block-Scoped-Variablen, let für veränderbare und const für unveränderbare Werte.
- **Iteratoren & Generatoren:** Neue Wege, um durch Daten zu iterieren

Hinweis: Alle diese ECMAScript-Features werden zu 100% von TypeScript unterstützt.

 http://w3schools.com/js/js_versions.asp

Essentials: ES6 Features

ES6 introduced a more expressive syntax for writing JS.

```
// Destructuring an object
const { name, age } = person;

// Spread operator for array
const newFruits = [...fruits, 'dragonfruit'];

// Template literals for strings
const greeting = `My name is ${name} and I am ${age} years old.`;
```

Essentials: Module System

Reusable pieces of code can be organized into modules, which can be imported and exported to enhance modularity and reusability.

```
// export-example.js
export const add = (a, b) => a + b;
export const subtract = (a, b) => a - b;

// import-example.js
import { add, subtract } from './export-example.js';

const sum = add(2, 3);
const difference = subtract(5, 2);
```

Exercise: Superhero Creator



- Create an object representing a superhero with properties like `name`, `power`, and `city`.
- Use `const` to ensure the superhero object itself cannot be reassigned.
- Use `let` to allow modifying a hero's stats dynamically.
- Use destructuring to extract and display hero properties.
- Use template literals to print a cool introduction

```
js                                         ⌂ Copy ⌂ Edit

// 1. Erstelle ein Superhelden-Objekt mit const
const superhero = {
  ...: "...",
  ...: "...",
  ...: "...",
  ...: ...
};

// 2. Nutze Destructuring, um Werte aus dem Objekt zu extrahieren
const { ..., ..., ..., ... } = superhero;

// 3. Gib die Superhelden-Infos mit Template Literals aus
console.log(`🌟 Meet ..., the guardian of ...!`);
console.log(`⚡ Superpower: ...`);
console.log(`💪 Strength Level: ...`);

// 4. Verwende let, um eine Eigenschaft dynamisch zu aktualisieren
let newStrength = ...;
console.log(`🔥 After training, ...'s ... increased to ...`);

// 5. Versuche, das gesamte Objekt neu zuzuweisen (dies sollte fehlschlagen)
// superhero = {}; // ❌ Fehler: Assignment to constant variable

console.log("🌟 Superhero setup complete! Ready to save the world!");
```

Solution: Superhero Creator



```
js                                         ⚡ Copy  ⚡ Edit

// 1. Define a superhero using const and an object
const superhero = {
  name: "Lightning Bolt",
  power: "Super Speed",
  city: "Metropolis",
  strength: 80
};

// 2. Use destructuring to extract properties
const { name, power, city, strength } = superhero;

// 3. Print superhero details using template literals
console.log(`🌟 Meet ${name}, the guardian of ${city}!`);
console.log(`⚡ Superpower: ${power}`);
console.log(`💪 Strength Level: ${strength}`);

// 4. Use let to modify the strength of the superhero
let newStrength = strength + 20;
console.log(`🔥 After training, ${name}'s strength increased to ${newStrength}!`);

// 5. Try to reassign the superhero object (this should fail)
// superhero = {} // ✖ Error: Assignment to constant variable

console.log("🌟 Superhero setup complete! Ready to save the world!");
↓
```

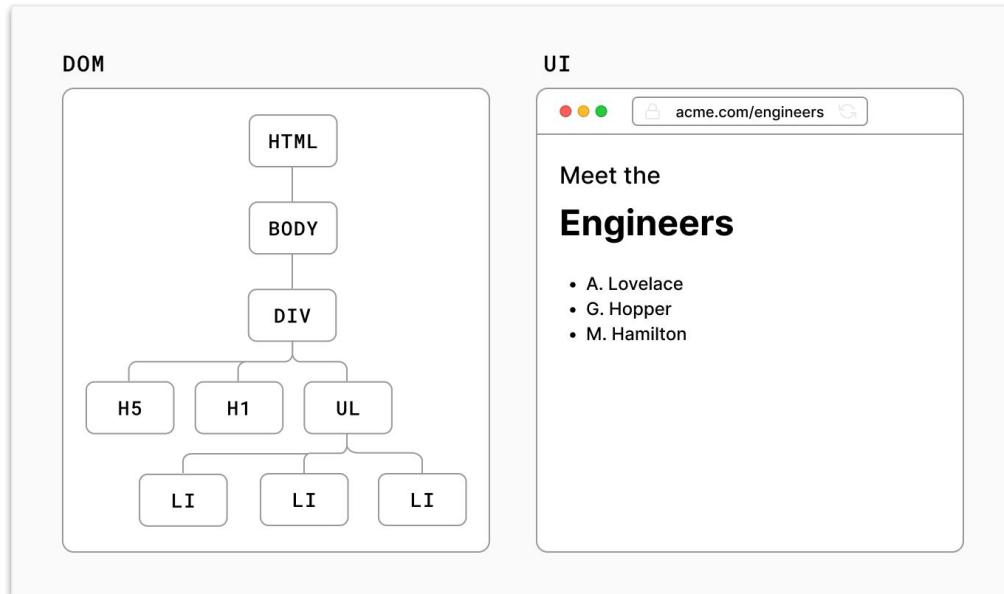
DOM

HTML vs. the DOM



What is the DOM?

The DOM is an object representation of the HTML elements. It acts as a bridge between your code and the user interface, and has a tree-like structure with parent and child relationships.



DOM Manipulation

Document Access & Selection

- Interact with the page structure and find elements using `getElementById()` or `querySelector()`.

Event Handling

- Use `addEventListener()` to respond to clicks, key presses, or other user actions.

Updating the DOM

- Change text, attributes, or styles (`innerHTML`, `setAttribute()`, etc.).

Creating & Removing Elements

- Generate new nodes with `createElement()` and attach them using `appendChild()`.

DOM Manipulation

```
1  <html>
2  |  <head> </head>
3  |  <body>
4  |    <div id="app"></div>
5  |    <script>
6  |      const app = document.getElementById('app');
7  |      const header = document.createElement('h1');
8  |      const text = 'Develop. Preview. Ship.';
9  |      const headerContent = document.createTextNode(text);
10 |      header.appendChild(headerContent);
11 |      app.appendChild(header);
12 |
13 |      const button = document.createElement('button');
14 |      const buttonText = 'Click me';
15 |      const buttonContent = document.createTextNode(buttonText);
16 |      button.appendChild(buttonContent);
17 |      app.appendChild(button);
18 |
19 |      button.addEventListener('click', () => {
20 |        alert('Hello');
21 |      });
22 |
23 |    </script>
24 |  </body>
25 </html>
```

HTML vs. the DOM

Open the developer tools to see the live DOM, which may differ from the original HTML due to newly added or updated elements.

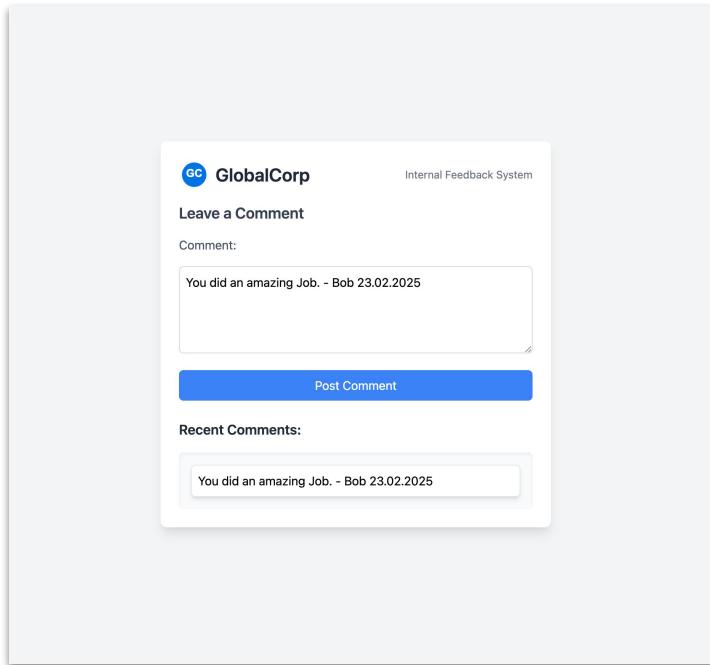
DOM

```
1 <html>
2 <head></head>
3 <body>
4   <div id="app">
5     <h1>Develop. Preview.
6 Ship. 🚀</h1>
7   </div>
8   <script type="text/
9 javascript">...</script>
10  </body>
11 </html>
```

SOURCE CODE (HTML)

```
1 <html>
2 <head></head>
3 <body>
4   <div id="app"></div>
5   <script type="text/
6 javascript">...</script>
7   </body>
8 </html>
9
10
11
```

Sometimes Dangerous ...



Exercise: DOM Manipulation

- Create an index.html file with an empty `<div>` that has `id="app"`.
- Create a card (with a title, a text, and a button).
- Append the card to the `#app` div.
- When clicking the button, change the text inside the card.

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>DOM Exercise</title>
</head>
<body>

    <div id="app"></div>

    <script>
        const app = document.getElementById("...");

        const card = document.createElement("...");
        const title = document.createElement("...");
        const text = document.createElement("...");
        const button = document.createElement("...");

        title.textContent = "...";
        text.textContent = "...";
        button.textContent = "...";

        card.appendChild(...);
        card.appendChild(...);
        card.appendChild(...);
        app.appendChild(...);

        button.addEventListener("...", () => {
            text.textContent = "...";
        });
    </script>

</body>
</html>
```

React

Building Blocks of a Web Application

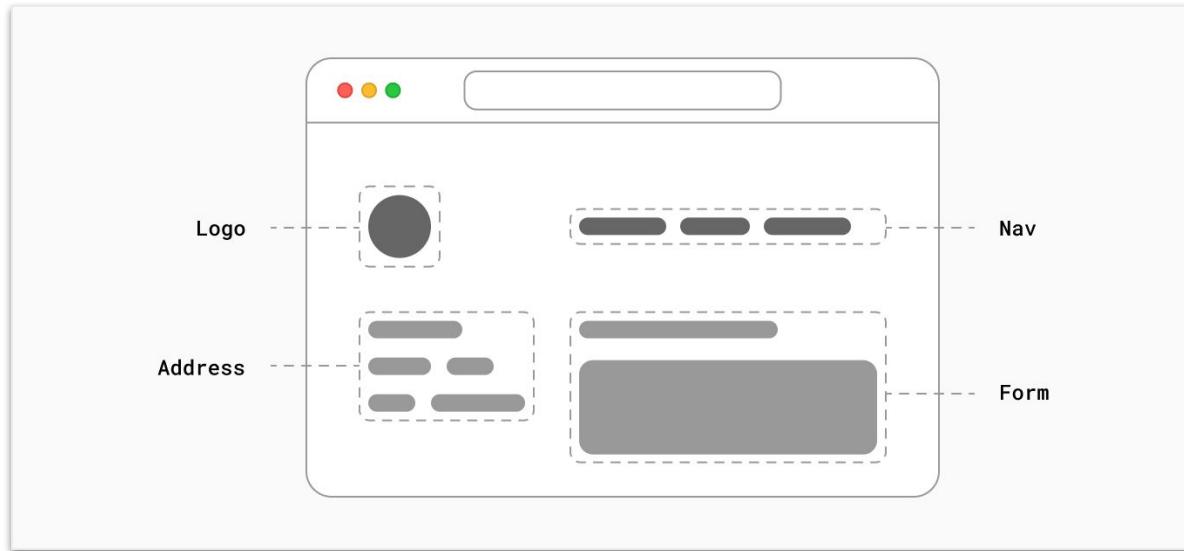
There are a few things you need to consider when building modern applications. Such as:

- **User Interfaces:** Streamlined interfaces for optimal user interaction.
- **Routing:** Efficient user flow between application segments.
- **Data Fetching:** Strategic data sourcing and retrieval methods.
- **Content Delivery:** Dynamic and static content rendering techniques.
- **Deployment Strategy:** Use of Serverless, CDN, and Edge for code execution.
- **Speed Optimization:** Enhancements for peak application performance.

For each part of your application, you will need to decide whether you will build a solution yourself or use other tools such as libraries and frameworks.

What is React?

React is a JavaScript library for building interactive user interfaces.



What is React?

React is a declarative, efficient, and flexible JavaScript library for building user interfaces.

- **Declarative UI:** Simplifies code readability and maintenance.
- **Component-Based:** Encapsulates code into reusable components.
- **Virtual DOM:** Optimizes rendering for performance gains.
- **Unidirectional Data Flow:** Ensures stability via controlled data.
- **Ecosystem Flexibility:** Integrates with various libraries and tools.

Imperative Programming (tell what to do)

Updating the DOM with plain JavaScript is very powerful but verbose. You've written all this code to add an `<h1>` element with some text.

With this approach, developers spend a lot of time writing instructions to tell the computer **how it should do things**.

```
6   const app = document.getElementById('app');
7   const header = document.createElement('h1');
8   const text = 'Develop. Preview. Ship.';
9   const headerContent = document.createTextNode(text);
10  header.appendChild(headerContent);
11  app.appendChild(header);
12
13  const button = document.createElement('button');
14  const buttonText = 'Click me';
15  const buttonContent = document.createTextNode(buttonText);
16  button.appendChild(buttonContent);
17  app.appendChild(button);
```

Declarative Programming (tell what you want)

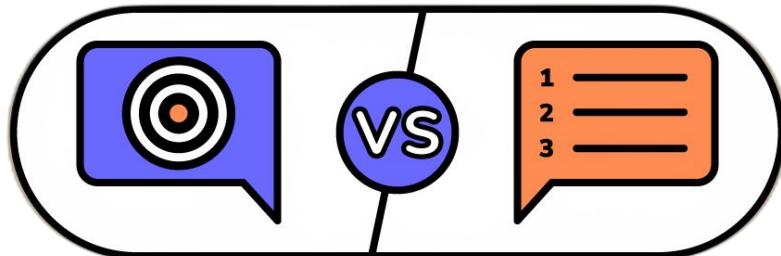
But wouldn't it be nice to describe what you want to show and let the computer figure out how to update the DOM?

```
<Header text="Develop. Preview. Ship. 🚀" />
```

Declarative vs Imperative Programming

In other words, imperative programming is like giving a chef step-by-step instructions on how to make a pizza.

Declarative programming is like ordering a pizza without being concerned about the steps it takes to make the pizza. 



Exercise: Basic Example

Copy the code and run the example on your computer.

 https://github.com/verwec/02_React_Example

```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  ||  <title>React App</title>
5  |</head>
6  <body>
7  |  <div id="app"></div>
8  |  <script src="https://unpkg.com/react@18/umd/react.development.js"></script>
9  |  <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>
10 |  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
11 <script type="text/babel">
12   function App() {
13     return (
14       <div>
15         <h1>Develop. Preview. Ship.</h1>
16       </div>
17     );
18   }
19
20   ReactDOM.render(<App />, document.getElementById('app'));
21 </script>
22 </body>
23 </html>
```

React Library Split

React and ReactDOM are separate to modularize client and renderer.

- **React Core:** Defines components and manages application logic.
- **ReactDOM Bindings:** Connects React to the DOM for web rendering.

✓ **Platform Agnostic:** Core concepts are universal, not tied to web or native.

✓ **Cleaner Updates:** Independent versioning and updates for each library.

Babel Transformation

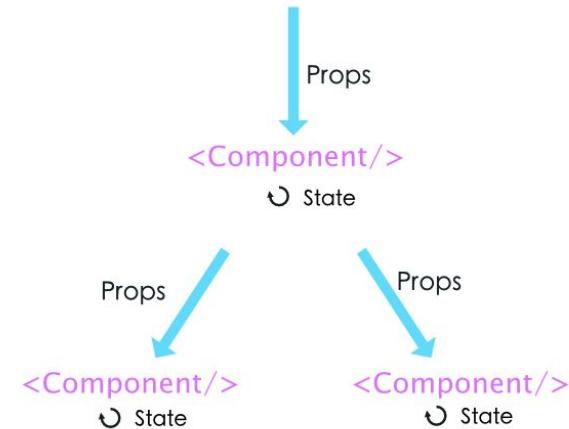
Babel transpiles modern JavaScript and JSX to browser-compatible code.

- **JSX to JavaScript:** Converts JSX syntax into plain JavaScript objects.
- **ES6+ Support:** Translates modern ES6+ code to ES5 for broader compatibility.
- **Browser Compatibility:** Ensures code runs on older web browsers.
- **Development Efficiency:** Allows developers to use the latest JS features.
- **Plugin Ecosystem:** Offers plugins for advanced code transformations and features.

React Core Concepts

There are three core concepts of React that you'll need to be familiar with to start building React applications. These are:

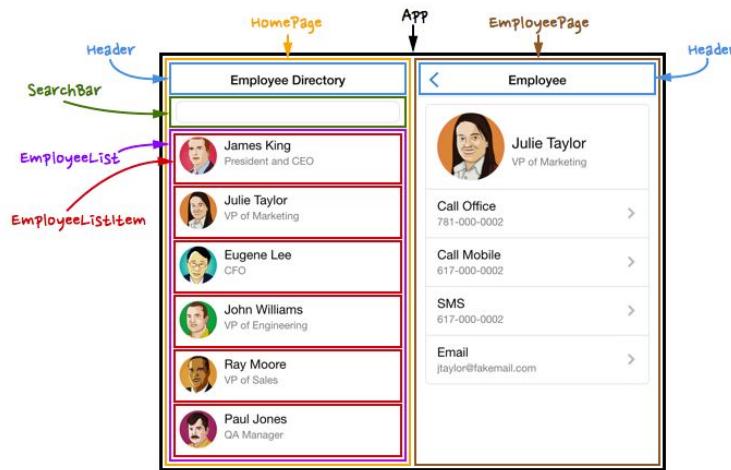
- Components
- Props
- State



Components

Components Hierarchy

Break up your user interface (design) into separate elements.



Nesting Components

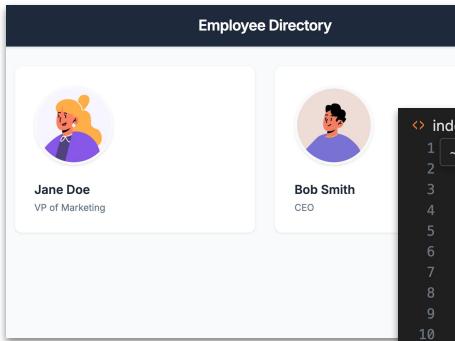
Applications usually include more content than a single component. You can nest React components inside each other like you would regular HTML elements.

```
function App() {
  return (
    <div>
      <header>
        <h1>Employee Directory</h1>
      </header>
      <main>
        <EmployeeCard />
        <OtherEmployeeCard />
      </main>
    </div>
  );
}
```

```
function EmployeeCard() {
  return (
    <div className="employee-card">
      
      <strong>Jane Doe</strong>
      <p>VP of Marketing</p>
    </div>
  );
}
```

Exercise: Simple React “Employee Directory”

Create the component App and implement all nested Components.



```
<> index.html > ...
1 ~/courses/react/03_COMPONENTS/index.html • 1 problem in this file
2 <!DOCTYPE html>
3 <html lang="en">
4   <head>
5     ...
6   </head>
7   <body>
8     <div id="root"></div>
9     <script type="text/babel">
10    ...
11
12    function App() {
13      return (
14        <div>
15          <header>
16            <h1>Employee Directory</h1>
17          </header>
18          <main>
19            <EmployeeCard />
20            <OtherEmployeeCard />
21          </main>
22        </div>
23      );
24
25      ReactDOM.render(<App />, document.getElementById('root'));
26    </script>
27  </body>
```

```
function EmployeeCard() {
  return (
    <div className="employee-card">
      
      <strong>Jane Doe</strong>
      <p>VP of Marketing</p>
    </div>
  );
}
```

Solution: Simple React “Employee Directory”

```
8   <script type="text/babel">
9     function EmployeeCard() {
10       return (
11         <div className="employee-card">
12           
16           <strong>Jane Doe</strong>
17           <p>VP of Marketing</p>
18         </div>
19       );
20     }
21
22     function OtherEmployeeCard() {
23       return (
24         <div className="employee-card">
25           
29           <strong>Bob Smith</strong>
30           <p>CEO</p>
31         </div>
32       );
33     }
34
35     function App() {
36       return (
37         <div>
38           <header>
39             <h1>Employee Directory</h1>
40           </header>
41           <main>
42             <EmployeeCard />
43             <OtherEmployeeCard />
44           </main>
45         </div>
46       );
47     }
48
49     ReactDOM.render(<App />, document.getElementById('root'));
50   </script>
```

Components & Props

Displaying Data without Props

Lot of code duplication :-(

```
function EmployeeList() {
  return (
    <>
      | <h1>Employee List</h1>
      | <div>
      |   | <Employee />
      |   | <hr />
      |   | <Employee2 />
      | </div>
    </>
  );
}

function Employee() {
  return (
    <div>
      | <div>Name: Bob</div>
      | <div>Age: 25</div>
      | <div>Salary: $100,000</div>
    </div>
  );
}

function Employee2() {
  return (
    <div>
      | <div>Name: Lisa</div>
      | <div>Age: 31</div>
      | <div>Salary: $80,000</div>
    </div>
  );
}
```

Exercise: Remove duplication

Remove the duplication with props.

```
function EmployeeList() {
  return (
    <>
      <h1>Employee List</h1>
      <div>
        <Employee name="Bob" age="25" salary="100,000" />
        <hr />
        <Employee name="Lisa" age="31" salary="80,000" />
      </div>
    </>
  );
}

function Employee({name, age, salary}) {
  return (
    <div>
      <div>Name: {name}</div>
      <div>Age: {age}</div>
      <div>Salary: {salary}</div>
    </div>
  );
}
```

JSX Rules

The Rules of JSX

- Return a single root element.
- To return multiple elements from a component, wrap them with a single parent tag.



```
<>
  <h1>Employee List</h1>
  <div>
    <Employee name="Bob" age="25" salary="100,000" />
    <hr />
    <Employee name="Lisa" age="31" salary="80,000" />
  </div>
</>
```

The Rules of JSX

JSX requires tags to be explicitly closed: self-closing tags like `` must become ``, and wrapping tags like ``oranges must be written as `oranges`.

```
<>
  
  <ul>
    <li>Name: {name}</li>
    <li>Age: {age}</li>
    <li>Salary: {salary}</li>
  </ul>
</>
```

The Rules of JSX

- React uses camelCase for HTML/SVG attributes (e.g. `stroke-width` → `strokeWidth`).
- `class` is reserved, so React uses `className` instead.

```
return (
  <div className="employee">
    
    <ul>
      <li>Name: {name}</li>
      <li>Age: {age}</li>
      <li>Salary: {salary}</li>
    </ul>
  </div>
);
```

Using “double curly braces”

- In addition to strings, numbers, and other JavaScript expressions, you can even pass objects in JSX.
- Objects are also denoted with curly braces. Therefore, to pass a JS object in JSX, you must wrap the object in another pair of curly braces



```
<div style={{border: '1px solid #ccc', padding: '10px'}}>
  
  <ul>
    <li>Name: {name}</li>
    <li>Age: {age}</li>
    <li>Salary: {salary}</li>
  </ul>
</div>
```

create-react-app

Demo: Create React App



Exercise: Create React App

Install react globally

- `npm install -g create-react-app`

Create a new React app

- `npx create-react-app my-app`
- `cd my-app`
- `npm start`

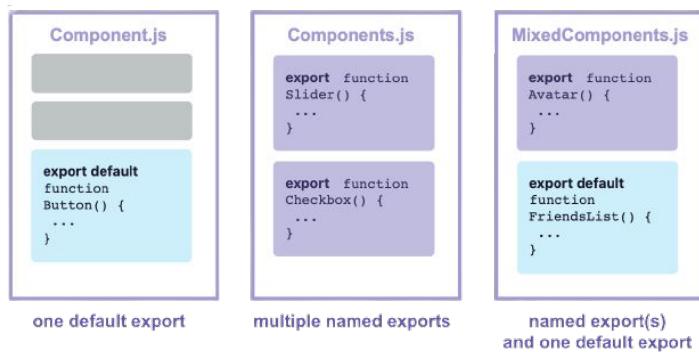
Show A List of 2-3 Employees in `App.js`

```
1 import logo from './logo.svg';
2 import './App.css';
3
4 function App() {
5   return (
6     <>
7       <h1>Employee List</h1>
8       <div>
9         <div>
10           <p>Name: Bob</p>
11           ...
12         </div>
13         <hr />
14         ...
15       </div>
16     </>
17   );
18 }
19
20 export default App;
21 //L to chat, #K to generate
```

In- & Exporting

Makes the code more modular and reusable in other files. Done in three steps:

1. Make a new JS file to put the components in.
2. Export your function component from that file (using either default or named exports).
3. Import it in the file where you'll use the component (using the corresponding technique for importing default or named exports).



In- & Exporting

```
1 import React from 'react';
2
3 function Employee() {
4   return <div>Employee</div>;
5 }
6
7 export default Employee;
```

```
3
4 import Employee from './Employee';
5
```



In- & Exporting

```
my-app > src > JS Employee.js > [?] default
1  import React from 'react';
2
3  function Employee({ name, age, salary }) {
4    return (
5      <div className="employee-card">
6        <h2>{name}</h2>
7        <p>Age: {age}</p>
8        <p>Salary: ${salary}</p>
9      </div>
10   );
11 }
12
13 function OfficeDog({ breed, age }) {
14   return (
15     <div className="employee-card">
16       <h2>{breed}</h2>
17       <p>Age: {age}</p>
18     </div>
19   );
20 }
21
22 export { OfficeDog };
23
24 export default Employee;
```



Props

Destructuring vs. Direct Prop Access

```
function OfficeDog({ breed, age }) {  
  return (  
    <div className="employee-card">  
      <h2>{breed}</h2>  
      <p>Age: {age}</p>  
    </div>  
  );  
}  
  
function OfficeCat(props) {  
  let breed = props.breed;  
  let age = props.age;  
  
  return (  
    <div className="employee-card">  
      <h2>{breed}</h2>  
      <p>Age: {age}</p>  
    </div>  
  );  
}
```

Highly recommend!

Props Default Values

```
function OfficeDog({ breed, age, isVaccinated=true }) {  
  return (  
    <div className="employee-card">  
      <h2>{breed}</h2>  
      <p>Age: {age}</p>  
      <p>Vaccinated: {isVaccinated ? "Yes" : "No"}</p>  
    </div>  
  );  
}
```



Props Forwarding

```
function AnimalOfTheWeek(props) {
  return (
    <>
    <h1> Animal of the Week</h1>
    <OfficeDog {...props} />
  </>
);
}

function OfficeDog({ breed, age, isVaccinated=true }) {
  return (
    <div className="animal-card">
      <h2>{breed}</h2>
      <p>Age: {age}</p>
      <p>Vaccinated: {isVaccinated ? "Yes" : "No"}</p>
    </div>
);
}
```

Passing JSX as children

```
function Card({children}) {
  return (
    <div className="p-6 max-w-sm mx-auto rounded-xl shadow-lg">
      {children}
    </div>
  );
}

function AnimalOfTheWeek(props) {
  return (
    <Card>
      <h1 className="text-2xl font-bold mb-2"> Animal of the Week</h1>
      <OfficeDog {...props} />
    </Card>
  );
}
```

Animal of the Week

Labrador

Age: 5

Vaccinated: Yes

Exercise: Refactoring & Styling

Remove the code duplication from your App.js.

- Use Employee components
- Integrate `<script src="https://cdn.tailwindcss.com"></script>` in your index.html
- Checkout <https://tailwindcss.com/docs/styling-with-utility-classes>
- Improve Styling by using the `{children}` syntax
(e.g. a wrapping Card Component)



John Doe

Age: 30

Salary: \$100000



Jane Doe

Age: 25

Salary: \$80000



Jim Doe

Age: 35

Salary: \$120000

Conditional Rendering

Conditional Rendering

Simple Solution:

App.js

```
1 function Item({ name, isPacked }) {
2   if (isPacked) {
3     return <li className="item">{name} ✓</li>;
4   }
5   return <li className="item">{name}</li>;
6 }
7
8 export default function PackingList() {
9   return (
10     <section>
11       <h1>Sally Ride's Packing List</h1>
12       <ul>
13         <Item
14           isPacked={true}
15           name="Space suit"
16         />
17     </ul>
18   )
19 }
```

Download Reset Fork

Sally Ride's Packing List

- Space suit ✓
- Helmet with a golden leaf ✓
- Photo of Tam

Show more

Conditional Rendering

JavaScript has a compact syntax for writing a conditional expression – the conditional operator or “ternary operator”.

```
return (
  <li className="item">
    {isPacked ? name + ' ✓' : name}
  </li>
);
```

Logical AND operator (&&)

Another common shortcut you'll encounter is the JavaScript logical AND (&&) operator. Inside React components, it often comes up when you want to render some JSX when the condition is true, or render nothing otherwise.

```
return (
  <li className="item">
    {name} {isPacked && '✓'}
  </li>
);
```

Exercise: Conditional Rendering

Add a 'Grandpa' emoji for all Employees that are older 50 years 😊



John Doe

Age: 30

Salary: \$100000



Jane Doe

Age: 25

Salary: \$80000



Jim Doe

Age: 35

Salary: \$120000



Jill Doe 😊

Age: 60

Salary: \$150000

Rendering Lists

Rendering Lists

You will often want to display multiple similar components from a collection of data.

1. **Map Functionality:** Render array items as components with map().
2. **Filter Usage:** Display only certain items using filter().
3. **React Keys:** Assign unique keys for stable identities.
4. **Component Arrays:** Transform data to component arrays for UI.
5. **Selective Rendering:** Control which elements to render and when.

Rendering data from arrays

App.js

Download Reset Fork

```
1 const people = [
2   'Creola Katherine Johnson: mathematician',
3   'Mario José Molina-Pasquel Henríquez: chemist',
4   'Mohammad Abdus Salam: physicist',
5   'Percy Lavon Julian: chemist',
6   'Subrahmanyam Chandrasekhar: astrophysicist'
7 ];
8
9
10 export default function List() {
11   const listItems = people.map(person =>
12     <li>{person}</li>
13   );
14   return <ul>{listItems}</ul>;
15 }
```

- Creola Katherine Johnson: mathematician
- Mario José Molina-Pasquel Henríquez: chemist
- Mohammad Abdus Salam: physicist
- Percy Lavon Julian: chemist
- Subrahmanyam Chandrasekhar: astrophysicist

▼ Console (1) (x)

Warning: Each child in a list should have a unique "key" prop.

Check the render method of `List`. See <https://reactjs.org/link/warning-keys> for more information.

at li
at List

Filtering arrays of items

App.js data.js utils.js

5 Reset 2 Fork

```
1 import { people } from './data.js';
2 import { getImageUrl } from './utils.js';
3
4 export default function List() {
5   const chemists = people.filter(person =>
6     person.profession === 'chemist'
7   );
8   const listItems = chemists.map(person =>
9     <li>
10       <img
11         src={getImageUrl(person)}
12         alt={person.name}
13       />
14       <p>
15         <b>{person.name}</b>
16         {' ' + person.profession + ' '}
17         known for {person.accomplishment}
18       </p>
19     </li>
20   );
21   return <ul>{listItems}</ul>;
22 }
```



Mario José Molina-Pasquel Henríquez: chemist known for discovery of Arctic ozone hole



Percy Lavon Julian: chemist known for pioneering cortisone drugs, steroids and birth control pills

▼ Console (1)

Warning: Each child in a list should have a unique "key" prop.

Check the render method of 'List'. See <https://reactjs.org/link/warning-keys> for more information.

at li
at List

^ Show less

Keeping list items in order with key

1. You need to give each array item a key – a string or a number that uniquely identifies it among other items in that array. A well-chosen key helps React infer what exactly has happened, and make the correct updates to the DOM tree.

```
<li key={person.id}>...</li>
```

Exercise 1: Conditional Rendering

- Replace the hardcoded employee components with a map function based on the list below.
- Create an Employee component for each item in the array.

```
const employees = [
  { name: "John Doe", age: 30, salary: 100000 },
  { name: "Jane Doe", age: 25, salary: 80000 },
  { name: "Jim Doe", age: 35, salary: 120000 },
  { name: "Jill Doe", age: 60, salary: 150000 }
];
```

Exercise 2: Group the list (filter)

- Add a new property `active`
- Group by `active` and `Inactive` (using filter)

Active Employees



John Doe

Age: 30

Salary: \$100000



Jane Doe

Age: 25

Salary: \$80000

Inactive Employees



Jim Doe

Age: 35

Salary: \$120000



Jill Doe 😊

Age: 60

Salary: \$150000

Use Effect

Demo API Access

SimpleReactRestAPIExample / src / App.js

verwec Simple API Example

Code Blame 33 lines (27 loc) · 716 Bytes

```
1 import logo from './logo.svg';
2 import './App.css';
3
4 import Employee from './Employee';
5 import { useEffect, useState } from 'react';
6
7
8 function App() {
9   const [isLoading, setIsLoading] = useState(true);
10  const [employees, setEmployees] = useState([]);
11
12  useEffect(() => {
13    fetch('https://jsonplaceholder.typicode.com/users')
14      .then(response => response.json())
15      .then(data => {
16        setEmployees(data);
17        setIsLoading(false);
18      });
19  }, []);
20  if (isLoading) {
21    return <div>Loading...</div>;
22  }
23
24  return (
25    <>
26      {employees.map(employee => (
27        <Employee key={employee.id} name={employee.name} email={employee.email} />
28      ))}
29    </>
30  );
31}
32
33 export default App;
```

<https://github.com/verwec/SimpleReactRestAPIExample/blob/main/src/App.js>

Exercise 3: API Access

- Show Pikachu :-)

```
import { useState, useEffect } from 'react';
import './App.css';

function App() {
  const [pokemon, setPokemon] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    // Fetch Pikachu's data as an example
    fetch('https://pokeapi.co/api/v2/pokemon/pikachu')
      .then(response => response.json())
      .then(data => {
        setPokemon(data);
        setLoading(false);
      })
      .catch(error => {
        console.error('Error fetching Pokemon:', error);
        setLoading(false);
      });
}, []);

export default App;
```

pikachu



Height: 4

Weight: 60