Introduction to React, for Angular Developpers

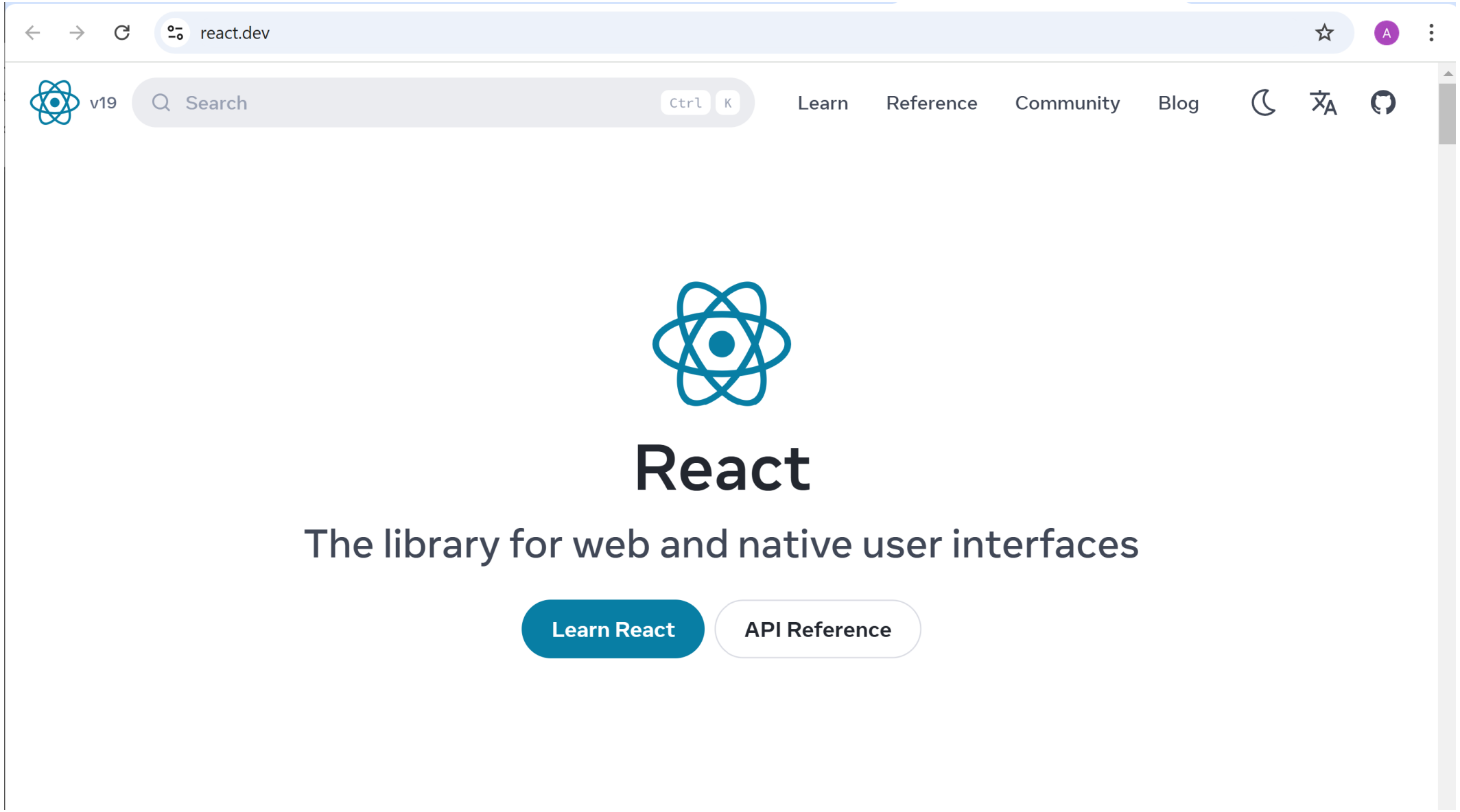
Course Esilv 2024

arnaud.nauwynck@gmail.com

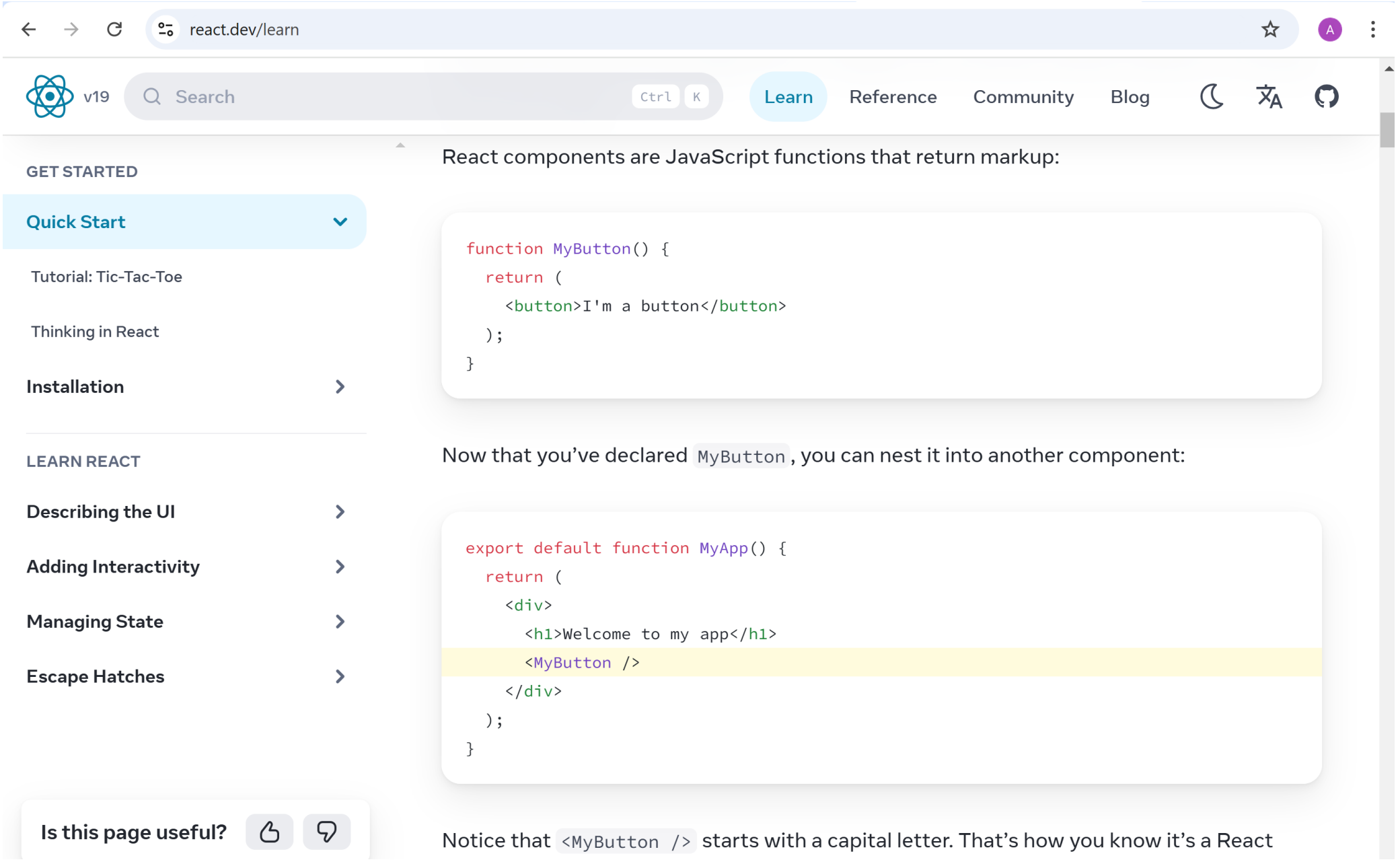
this document:

https://github.com/Arnaud-Nauwynck/Presentation/web/

https://react.dev



https://react.dev/learn



JSX, TSX ?

ChatGPT

explain tsx langage

TSX (TypeScript JSX) is a syntax extension for TypeScript that allows you to write JSX code with TypeScript's type-checking and additional features. It is commonly used in React projects where developers benefit from TypeScript's static typing and JSX's syntax for creating user interfaces.

What is JSX?

JSX (JavaScript XML) is a syntax extension for JavaScript used with React. It allows you to write HTML-like code within JavaScript, making it easier to describe what the UI should look like. For example:

jsx Copy code

```jsx
const element = <h1>Hello, world!</h1>;
```

What makes TSX different?

TSX is JSX combined with TypeScript, so you can:

1. Use TypeScript's static typing to enforce and check types.
2. Leverage TypeScript's interfaces, enums, and types for better type safety in your React components and props.
3. Write cleaner, safer, and more maintainable React code.

TSX -> TS + React runtime Virtual DOM

how to transform jsx to js ?

Transforming JSX to plain JavaScript involves converting the JSX syntax into equivalent `React.createElement` function calls, which are how JSX is interpreted by React under the hood. Here's how you can do it:

---

1. Understand the Equivalent JavaScript Code

JSX is syntactic sugar for `React.createElement`. For example:

JSX:

jsx Copy code

```jsx
const element = <div className="container">Hello, world!</div>;
```

JavaScript:

js Copy code

```js
const element = React.createElement(
  "div",
  { className: "container" },
  "Hello, world!"
);
```

https://babeljs.io/docs/babel-plugin-transform-react-jsx

JavaScript

```
const profile = (
  <div>
    <img src="avatar.png" className="profile" />
    <h3>{[user.firstName, user.lastName].join(" ")}</h3>
  </div>
);
```

Out

JavaScript

```
import { jsx as _jsx } from "react/jsx-runtime";
import { jsxs as _jsxs } from "react/jsx-runtime";

const profile = _jsxs("div", {
  children: [
    _jsx("img", {
      src: "avatar.png",
      className: "profile",
    }),
    _jsx("h3", {
      children: [user.firstName, user.lastName].join(" "),
    }),
  ],
});
```

2 new langages mix of JS + HTML

TSX = extension of TS (Typescript) containing Html
JSX = extension to JS (Javascript) containing Html

JSX is javascript that contains HTML <> ... </>
and nested HTML can contains <>{ javascript}<> !!
...which can contains sub-nested Html {<> .. </>}
... wich can contain Js: <>{ js.. <> { js } </> ...} </>

1 TSX File vs Angular 1 ts or 1 ts+1 html Files


```
function ProfilePhoto() {
  return <img src="profile-photo.jpg" alt="Your profile photo">;
}
```

Pros/Cons of 2 Files for Angular ?

Angular Html Templates are REAL Html syntaxicaly correct

(the *ngIf or   @if() { ..} are just attribute / text in plain old html)

++ having 2 files with extension is better supported by all IDE

-- It is overkill for component with few line(s)
-- templateHtml=' .. ' is not supported by all IDE as Html

-- TSX / TSX : neither valid Typescript, nor valid HTML.
    merge of both langages, require special IDE support

Create a New React Project

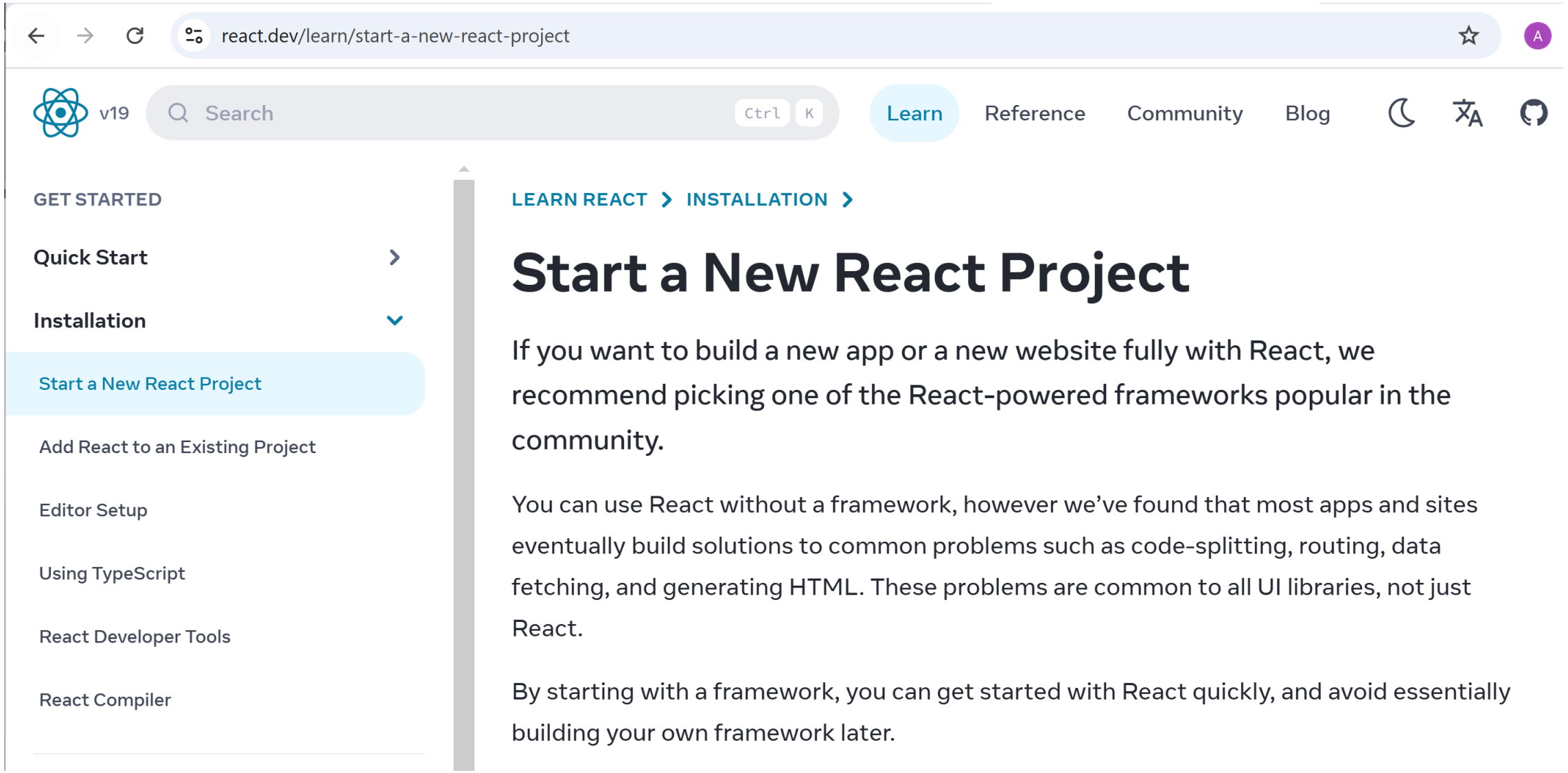
try 1 : tested official doc React + Next.JS Framework
npx create-next-app
next dev ... FAILED

try 2 : tested
npx create-react-app
npm install ... FAILED

try 3 : tested **IDEA > New Project**
Run ... FAILED

new project using React + Next.Js

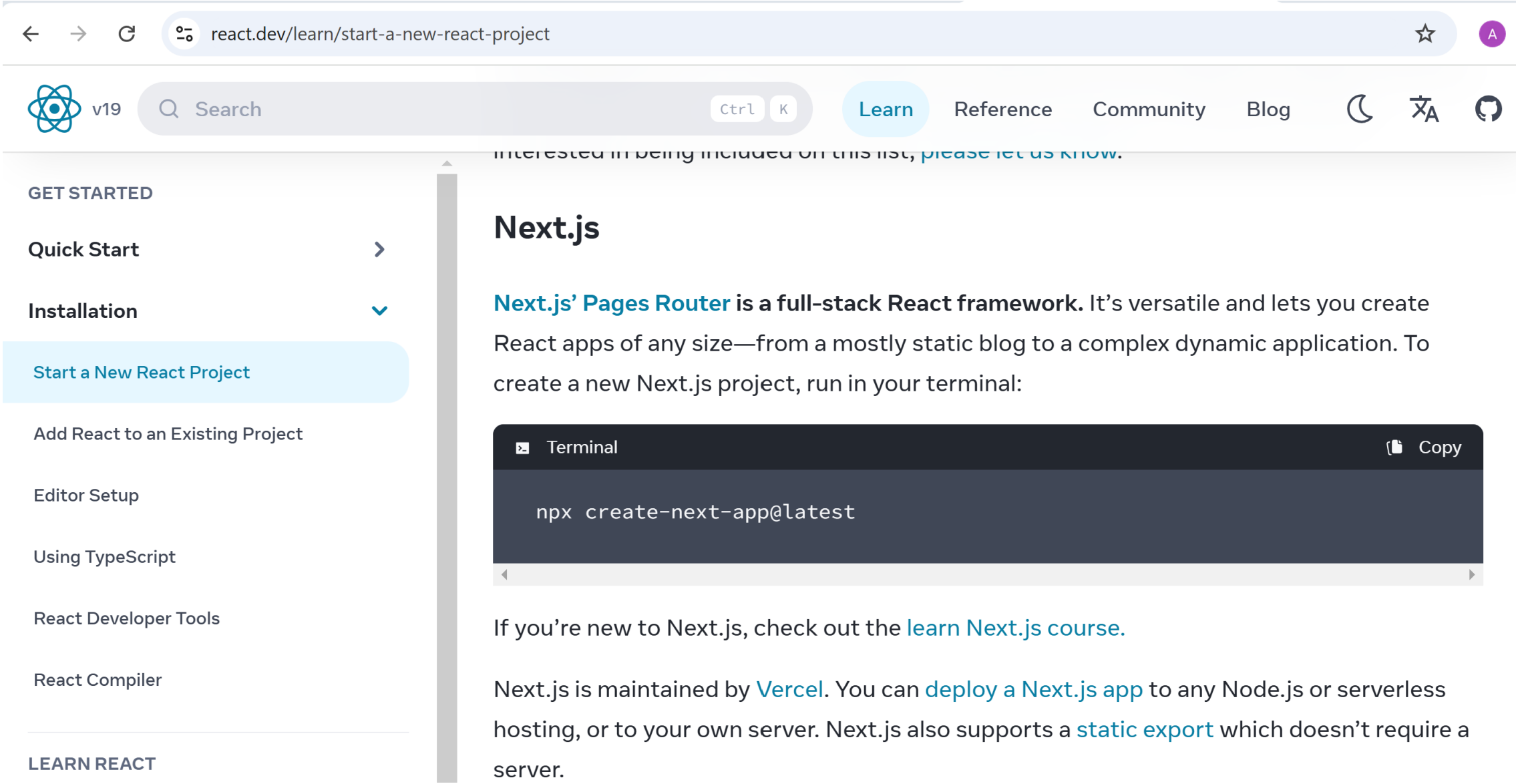
New React Project + choose a Framework



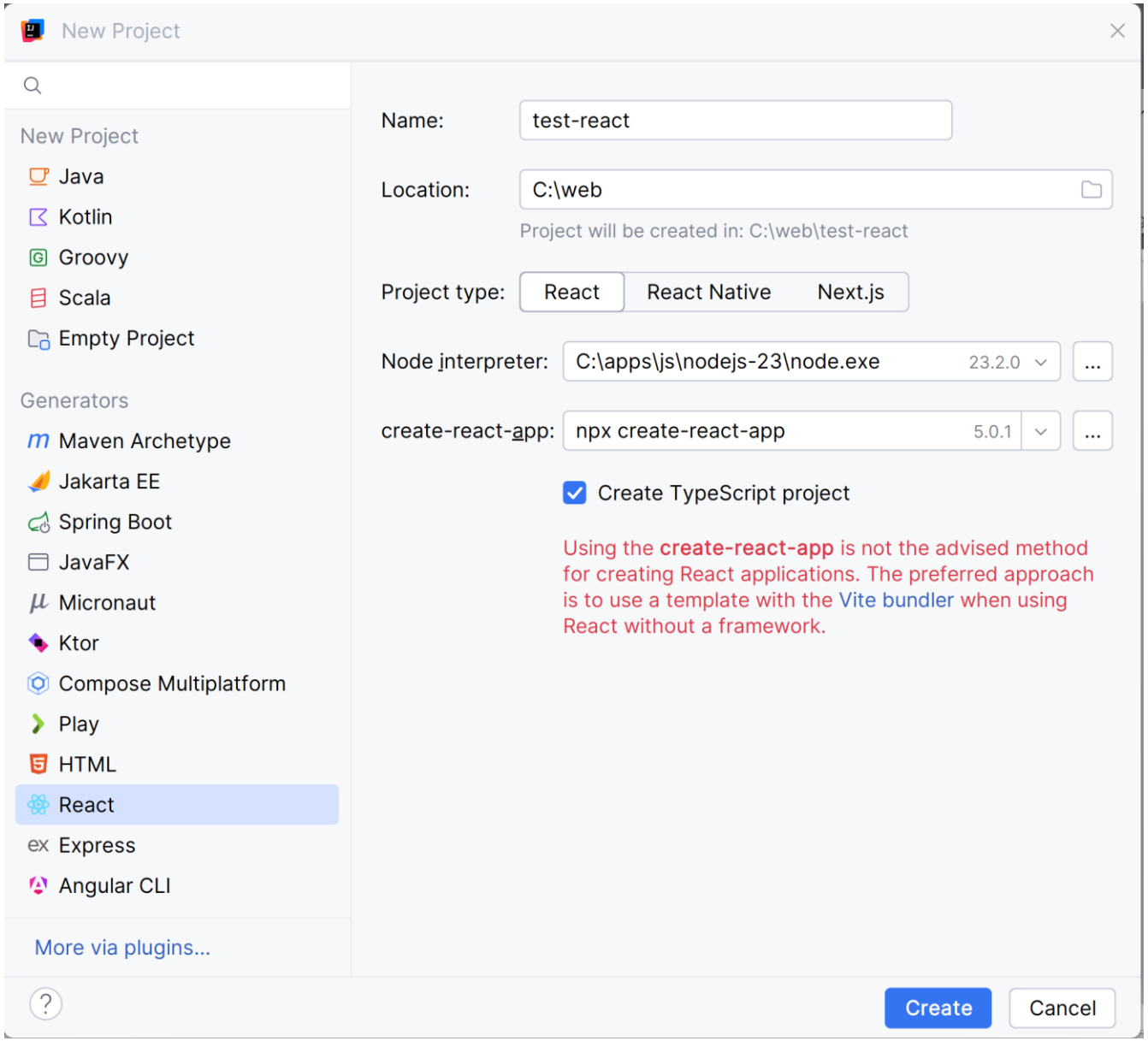
React is NOT enough,
You need additional Framework(s)

example: for Router,
for state : Redux,
better bindings : MOBX,..

Next.JS

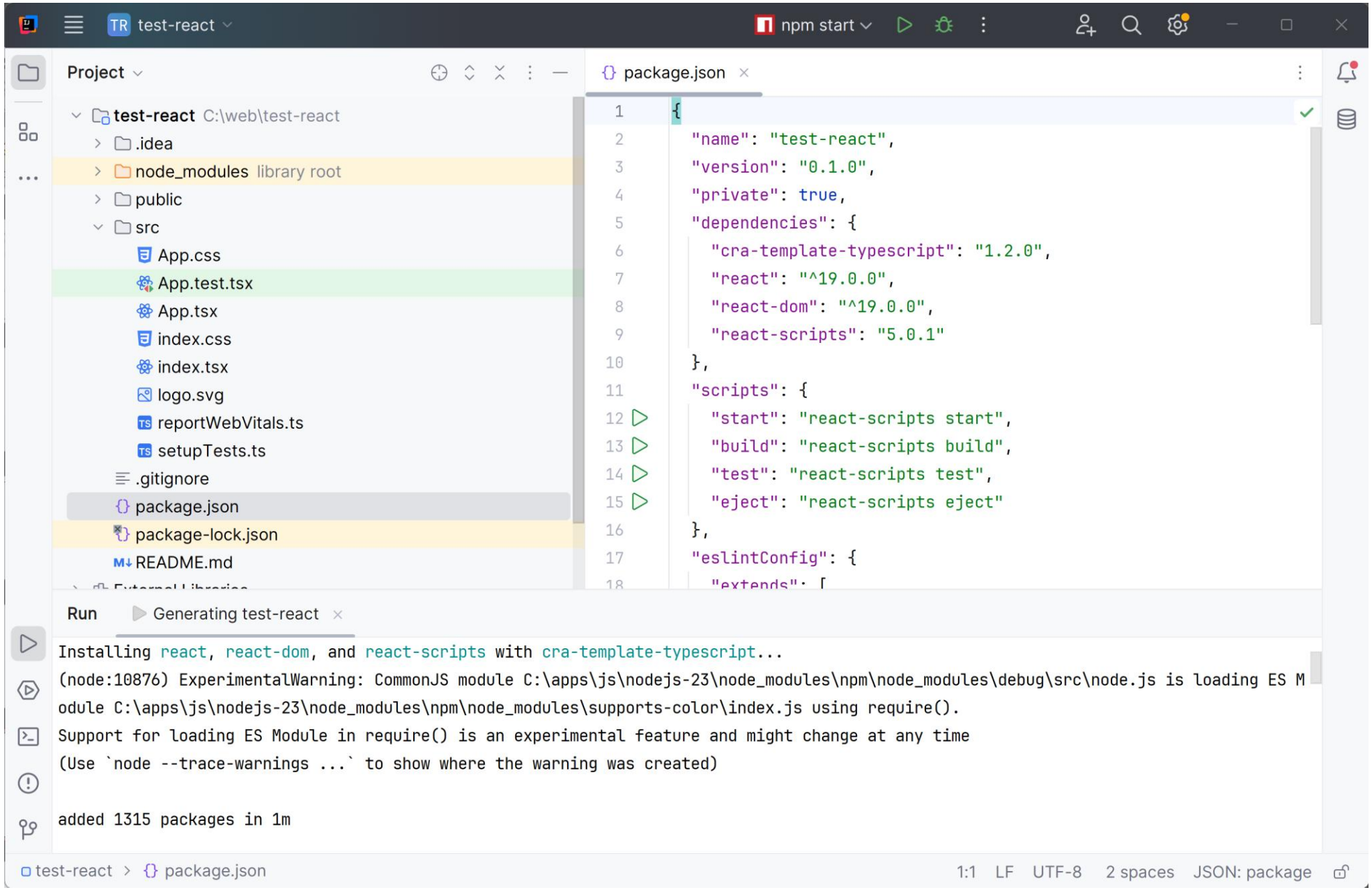

react.dev/learn/start-a-new-react-project
v19
Search
Ctrl
K
Learn
Reference
Community
Blog
GET STARTED
Quick Start
Installation
Start a New React Project
Add React to an Existing Project
Editor Setup
Using TypeScript
React Developer Tools
React Compiler
LEARN REACT
Next.js
Next.js' Pages Router is a full-stack React framework. It's versatile and lets you create React apps of any size—from a mostly static blog to a complex dynamic application. To create a new Next.js project, run in your terminal:
Terminal
Copy
npx create-next-app@latest
If you're new to Next.js, check out the learn Next.js course.
Next.js is maintained by Vercel. You can deploy a Next.js app to any Node.js or serverless hosting, or to your own server. Next.js also supports a static export which doesn't require a server.

From Idea: New Project > "React"

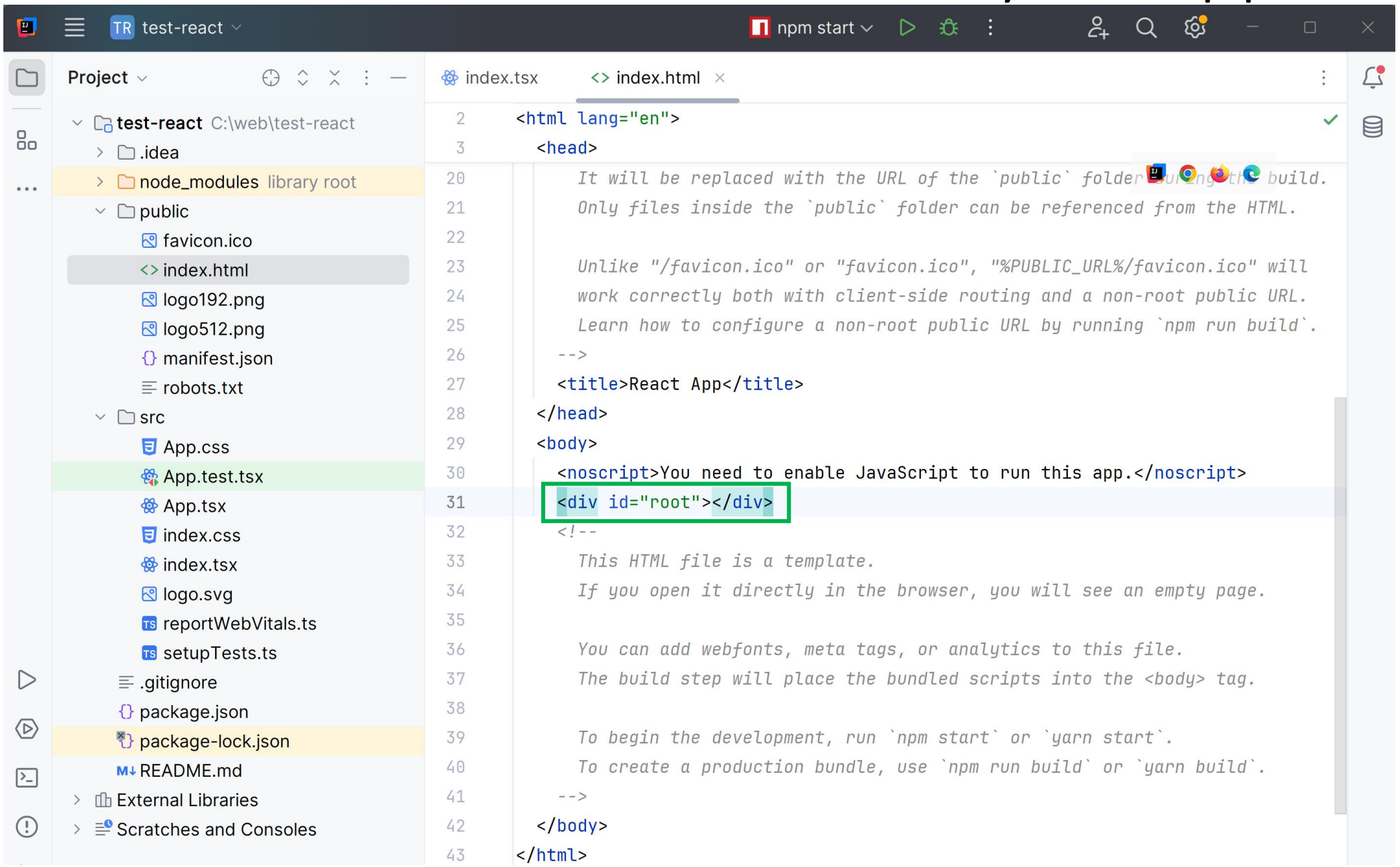


Exploring a Minimal React App

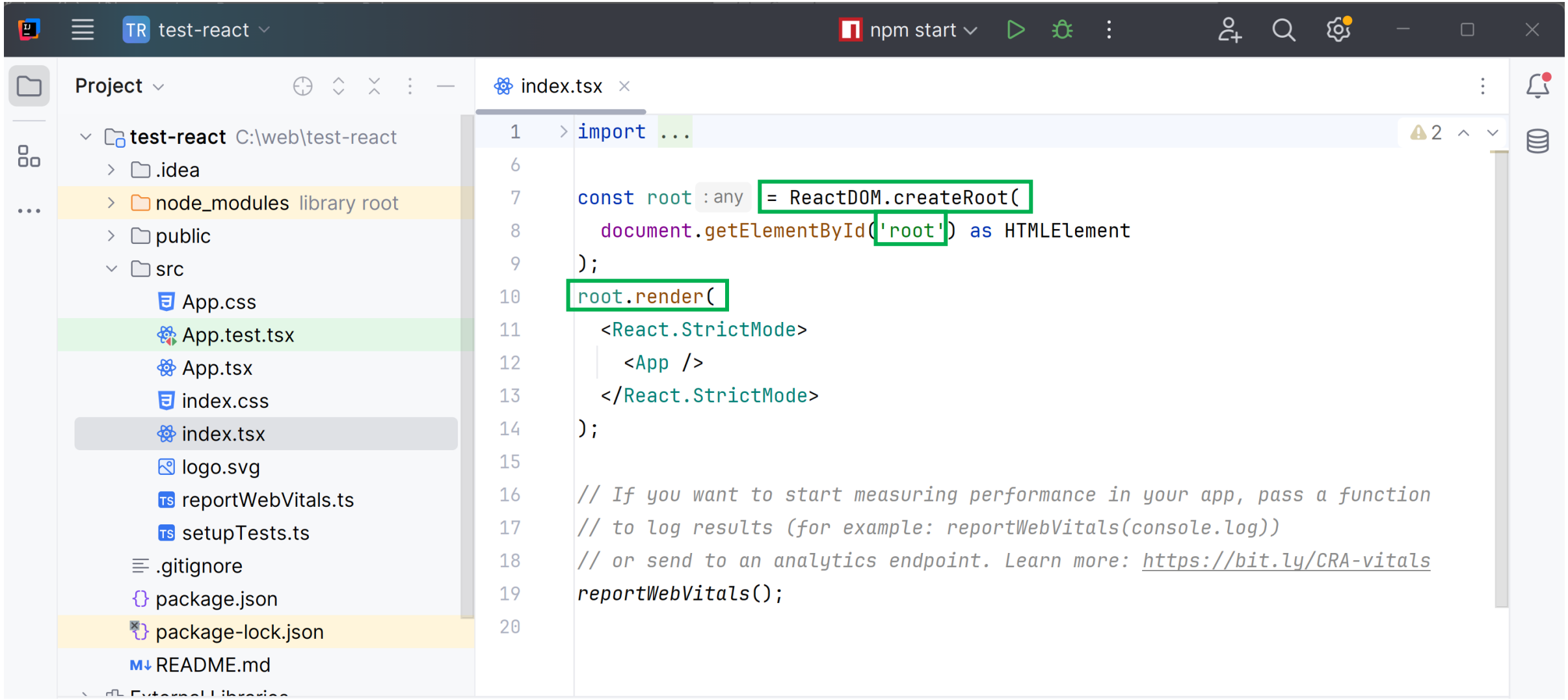
React Project



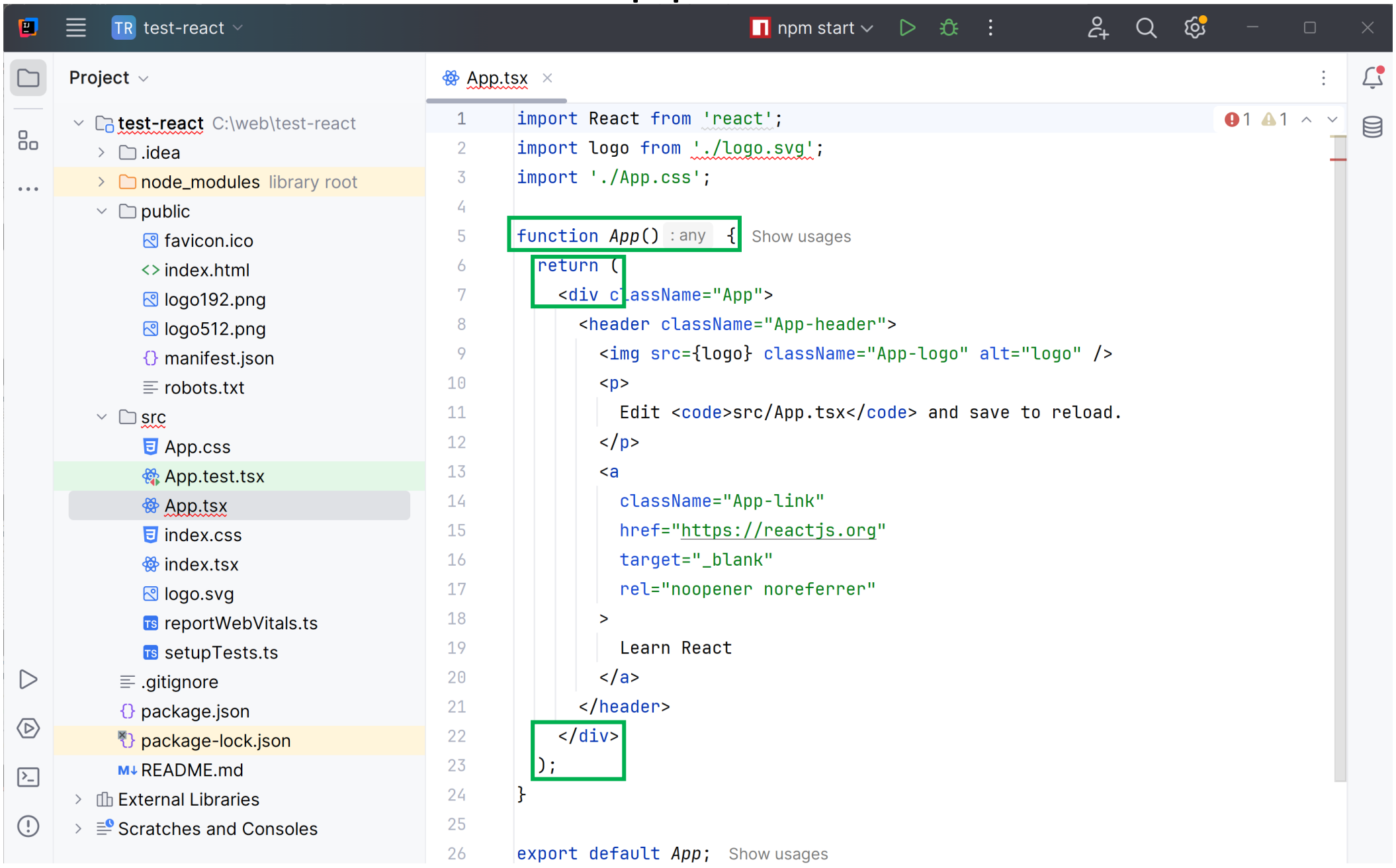
Index.html : Main Container for your App



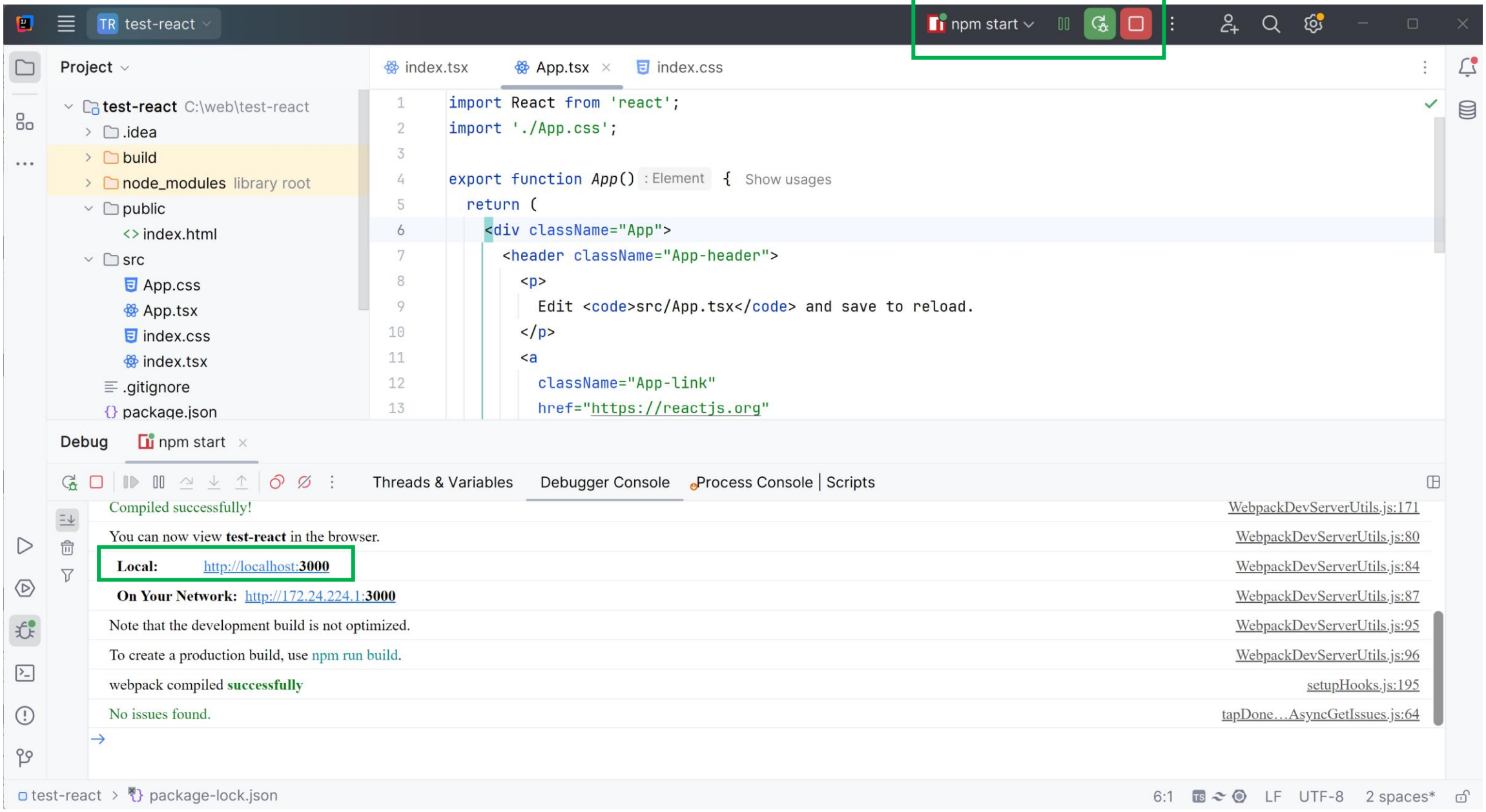
index.tsx : Bootstrapping ReactDOM.render()



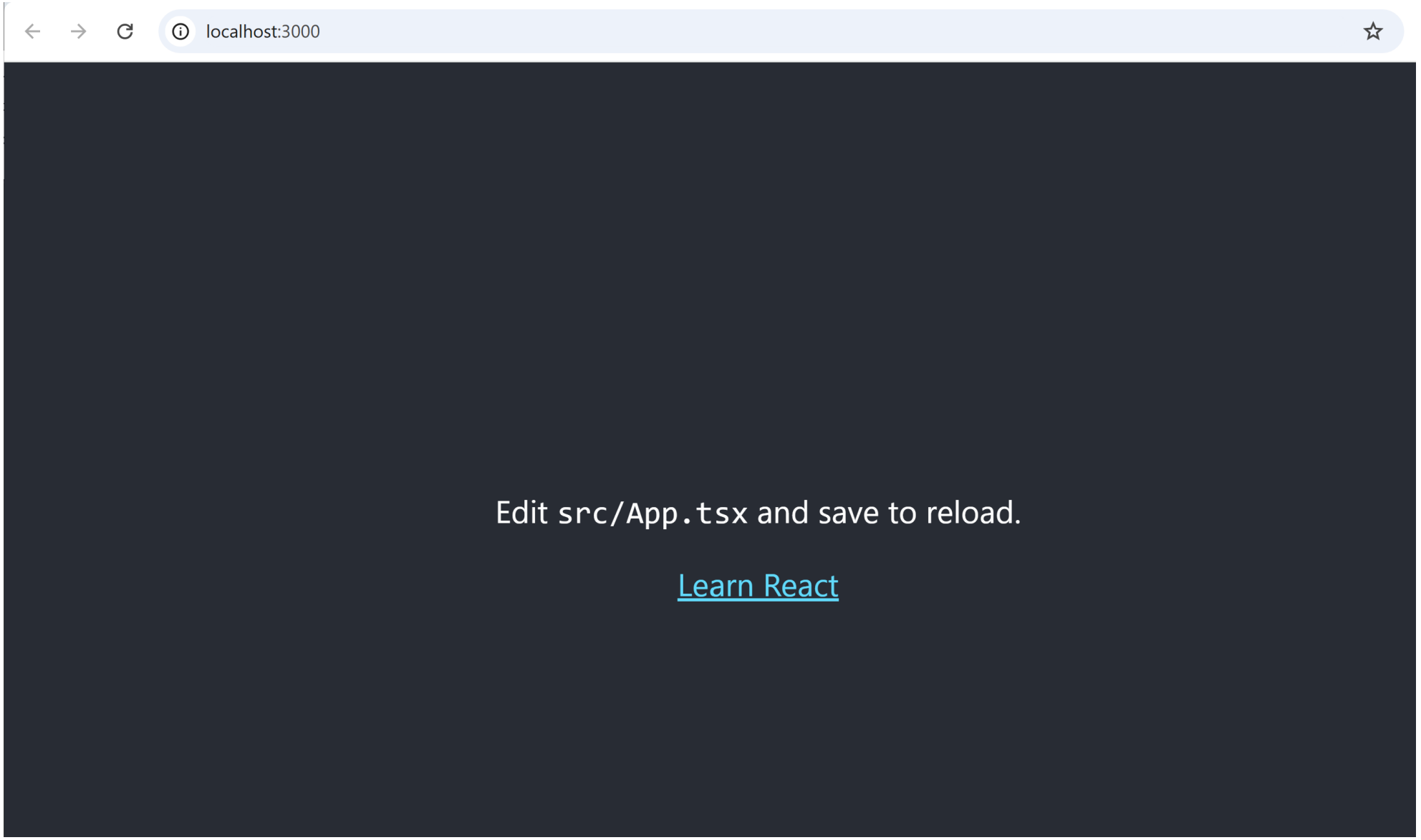
App.tsx


test-react
npm start
Project
test-react C:\web\test-react
.idea
node_modules library root
public
favicon.ico
index.html
logo192.png
logo512.png
manifest.json
robots.txt
src
App.css
App.test.tsx
App.tsx
index.css
index.tsx
logo.svg
reportWebVitals.ts
setupTests.ts
.gitignore
package.json
package-lock.json
README.md
External Libraries
Scratches and Consoles
App.tsx
1 import React from 'react';
2 import logo from './logo.svg';
3 import './App.css';
4
5 function App() : any { Show usages
6 return (
7
8 <header className="App-header">
9
10
11 Edit <code>src/App.tsx</code> and save to reload.
12
13 <a
14 className="App-link"
15 href="https://reactjs.org"
16 target="_blank"
17 rel="noopener noreferrer"
18 >
19 Learn React
20
21 </header>
22
23);
24 }
25
26 export default App; Show usages

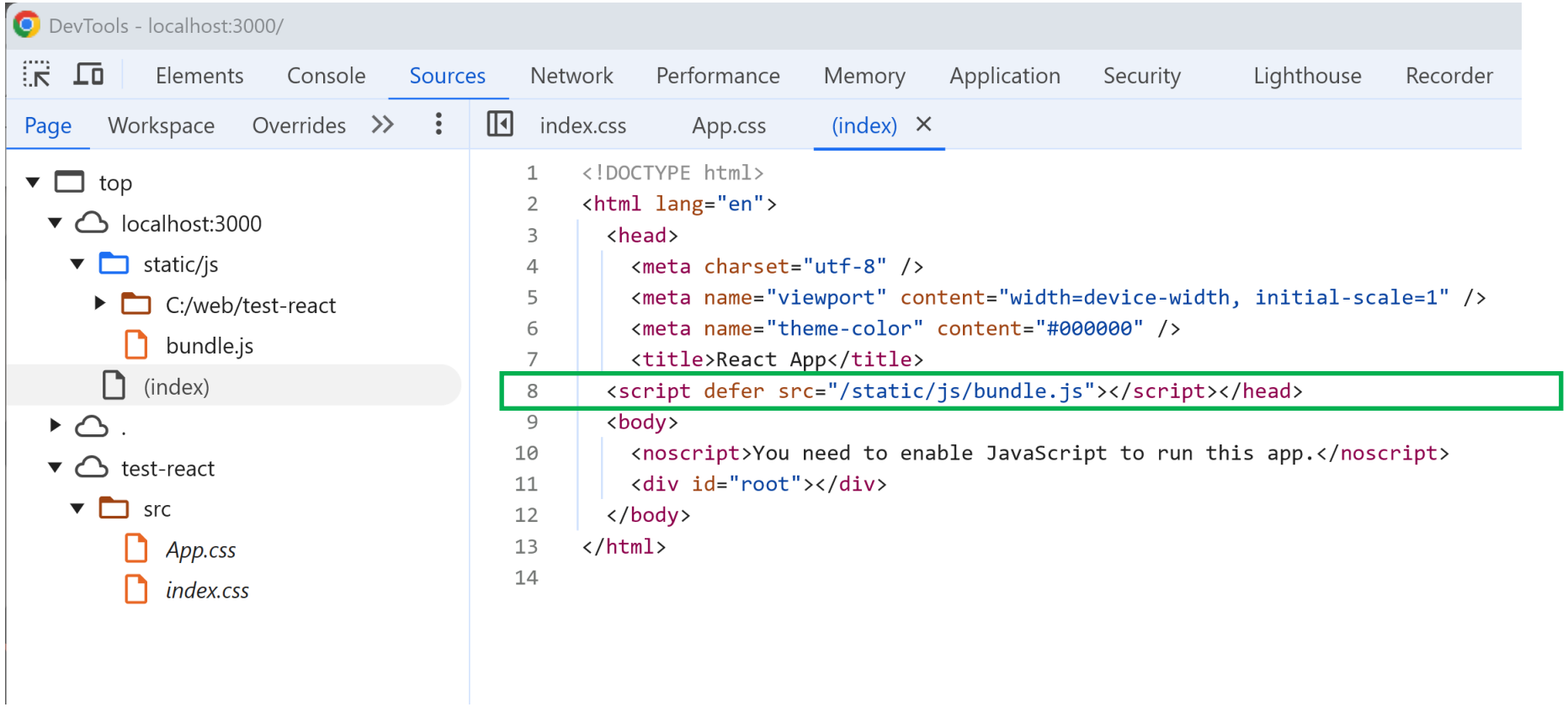
Idea Run / npm run start



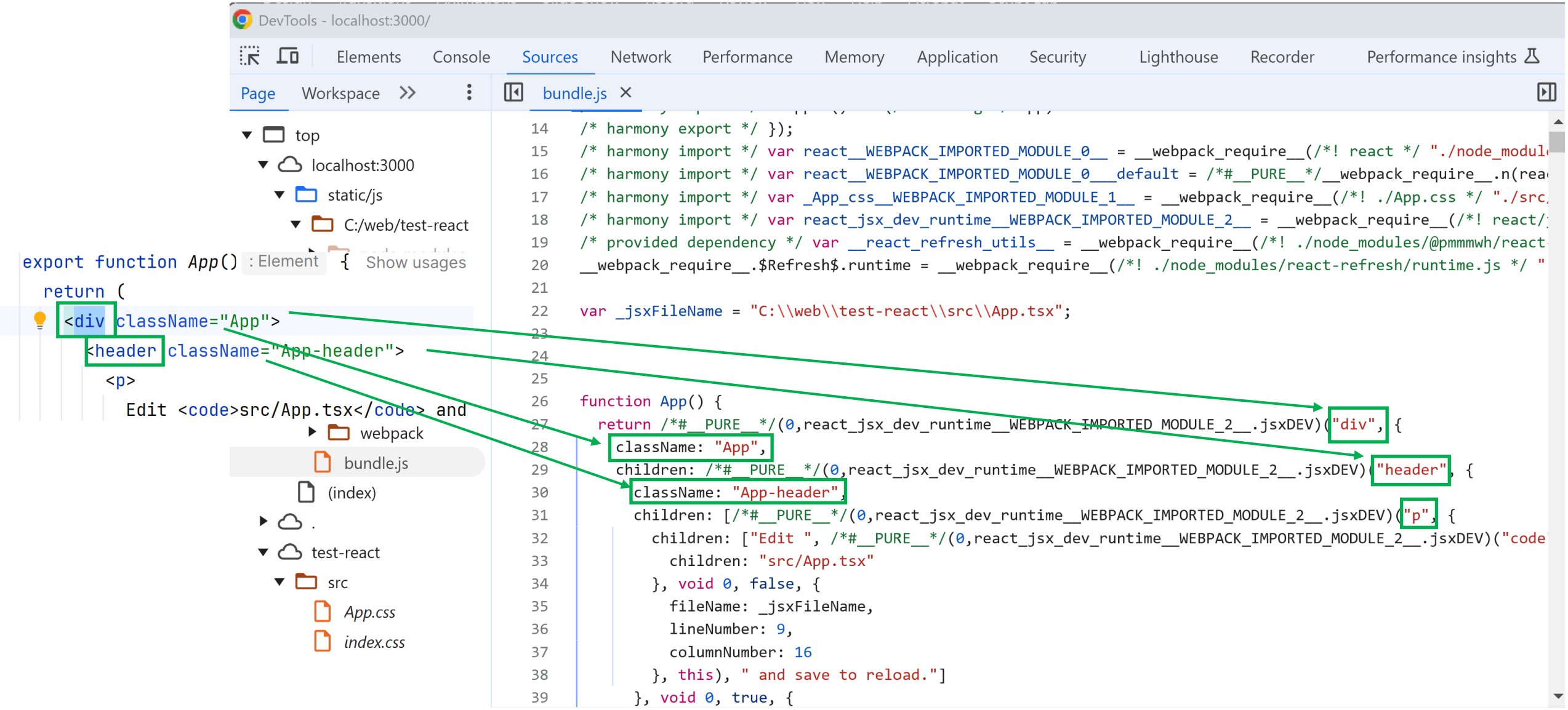
http://localhost:3000



DevTools ... viewing generated "/js/bundle"



bundle … Transpiled TSX -> to JS + react runtime on Virtual DOM



Exploring React Features comparison with Angular

Basic Features

Html Rendering template in Component

Calling a child component from a Component

templating with "If" and "For"

binding properties to a Component

binding Callback

Bi-Directional Binding ?

Routing ?

More ?

Basic Features
Comparison with Angular

Html Rendering template in Component	→	{{ expression }}	expression : field in class component
Calling a child component from a Component	→	<app-child />	selector="app-child" on child class
templating with "If" and "For"	→	@if(..) { } @else { .. } @for() { .. }	cf also legacy syntaxes *ngIf="cond" *ngFor="let i of items"
binding properties to a Component	→	field = input(); <app-child [field]="expr">	cf also legacy syntax @Input()
binding Callback	→	onEvent = signal(); <app-child (onEvent)="cb()">	cf also legacy syntax @Output()
Bi-Directional Binding ?	→	<app-child [(field)] ="parentField">	...magic & simple
Routing ?	→	<a routerLink="/compA"> or router.navigate(['compA'])	
More ?	→	form, i18n, animation, etc etc	

Html Rendering template in Component

return (<App attribute={JS}> </App>);

But what if you want to dynamically specify the `src` or `alt` text? You could **use a value from JavaScript by replacing `"` and `"` with `{` and `}`**:

App.js

Download Reset Fork

```
1  export default function Avatar() {
2    const avatar = 'https://i.imgur.com/7vQD0fPs.jpg';
3    const description = 'Gregorio Y. Zara';
4    return (
5      <img
6        className="avatar"
7        src={avatar}
8        alt={description}
9      />
10   );
11 }
12
```

return (<App> text {JS} </App>);

Using curly braces: A window into the JavaScript world

JSX is a special way of writing JavaScript. That means it's possible to use JavaScript inside it—with curly braces `{ }`. The example below first declares a name for the scientist, `name`, then embeds it with curly braces inside the `<h1>`:

App.js

Download Reset Fork

```
1  export default function TodoList() {
2    const name = 'Gregorio Y. Zara';
3    return (
4      <h1>{name}'s To Do List</h1>
5    );
6  }
7
```

Exploring React Features
Comparison with Angular

☑ Html Rendering template in Component	return (<> text { expr } </>)	{{ expr }}
Calling a child component from a Component		<app-child />
templating with "If" and "For"		@if(..) { } @else { .. } @for() { .. }
binding properties to a Component		field = input(); <app-child [field]="expr">
binding Callback		onEvent = signal(); <app-child (onEvent)="cb()">
Bi-Directional Binding ?		<app-child [(field)] ="parentField">
Routing ?		<a routerLink="/compA"> or router.navigate(['compA'])
More ?		form, i18n, animation, etc etc

Class are Statefull
!= pure Function are stateless


```
@Component( ..)
export class MyComp  {

   stateField:  string = 'Hey?';

}
```

```
<app-MyComp>
   {{ stateField }}
</app-MyComp>
```


```
function MyComp() {
return (
    <>
       { stateField }
    </>
  );
}
```

???????????
not a parameter
not a global variable
can not be in pure fonction

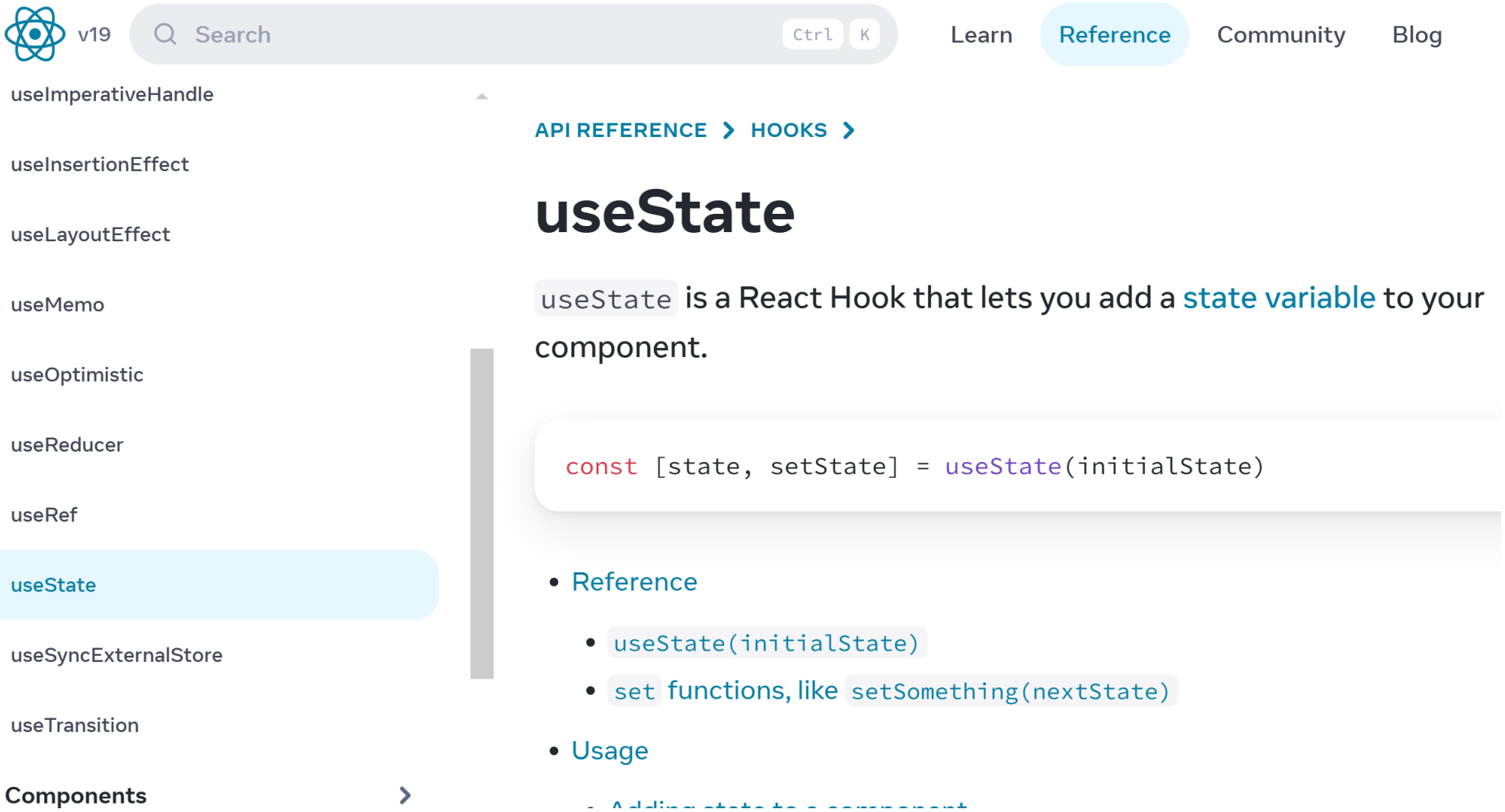
Class are Statefull
!= pure Function are stateless
BUT React use "Hook" useState() !!


```
@Component( ..)
export class MyComp {

    stateField:  string = 'Hey?';

}
```

```
<app-MyComp>
   {{ stateField }}
</app-MyComp>
```


```
import { useState } from 'react';

function MyComp () {
  const [ stateField, setStateField ] = useState<string>('Hey?)
  return (
    <>
      { stateField }
    </>
  );
}
```

useState is a React Hook for state variable



Notice ... state "Variable" is "const" !! but there is a setter

```
const [state, setState] = useState(initialState)
```

In TypeScript, this is equivalent of declaring 2 constants

const state: Type = initialState;

const setState: (value: Type) => void = function(value) { magic update that trigger re-rendering ... }

This does not compile "state = newValue", but ok with "setState(newValue)"

Setter then Automatically re-rendering

In angular, you just do

this.state = newValue;

=> it automatically re-renders

May use explicit "PushMode" for performance

May also use angular 17 "signal()" and "computed()"

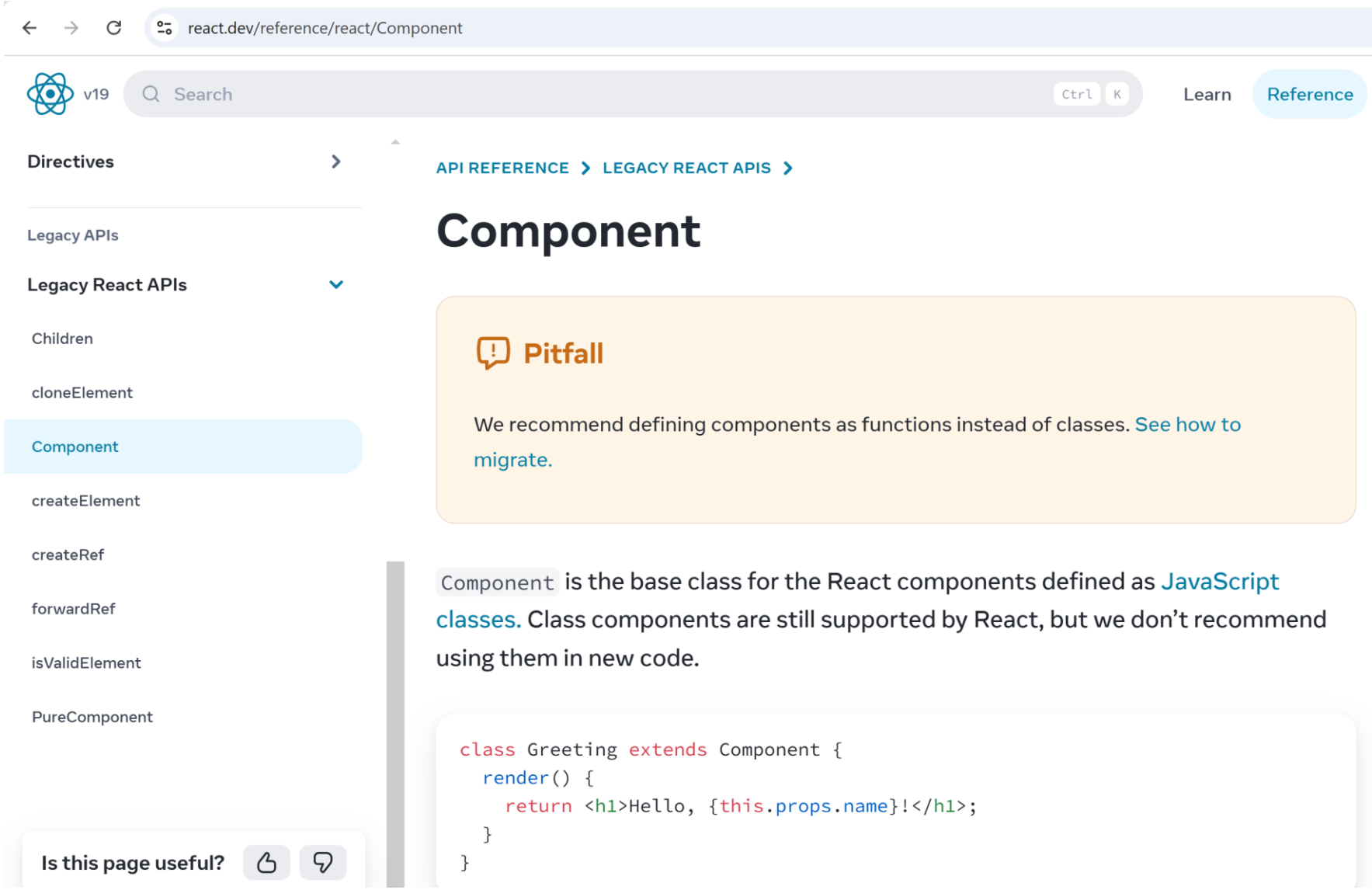
In React, you just do

setState(newValue);

=> it automatically re-renders

May also use "effect(() => { ... })" to do outside of method call

Why React Functions ?
it used to be "Class" (now legacy)



React moving away from W3C Standard Spec " CustomElement are classes"

CustomElement = class + register it

html.spec.whatwg.org/multipage/custom-elements.html#custom-elements-autonomous-example

§ **4.13.1.1 Creating an autonomous custom element**

This section is non-normative.

For the purposes of illustrating how to create an autonomous custom element, let's define a custom element tl
to be able to use it like so:

```
<flag-icon country="nl"></flag-icon>
```

To do this, we first declare a class for the custom element, extending `HTMLElement`:

```
class FlagIcon extends HTMLElement {
  constructor() {
    super();
    this._countryCode = null;
  }

  static observedAttributes = ["country"];

  attributeChangedCallback(name, oldValue, newValue) {
    // name will always be "country" due to observedAttributes
    this._countryCode = newValue;
    this._updateRendering();
  }
  connectedCallback() {
    this._updateRendering();
  }

  get country() {
    return this._countryCode;
  }
  set country(v) {
    this.setAttribute("country", v);
  }

  _updateRendering() {
    // Left as an exercise for the reader. But, you'll probably want to
```

class lifecycle methods

attr, getter, setter

We then need to use this class to define the element:

```
customElements.define("flag-icon", FlagIcon);
```

At this point, our above code will work! The parser, whenever it sees the `flag-icon` tag, will construct a
`country` attribute, which we then use to set the element's internal state and update its rendering (when

You can also create `flag-icon` elements using the DOM API:

```
const flagIcon = document.createElement("flag-icon")
flagIcon.country = "jp"
document.body.appendChild(flagIcon)
```

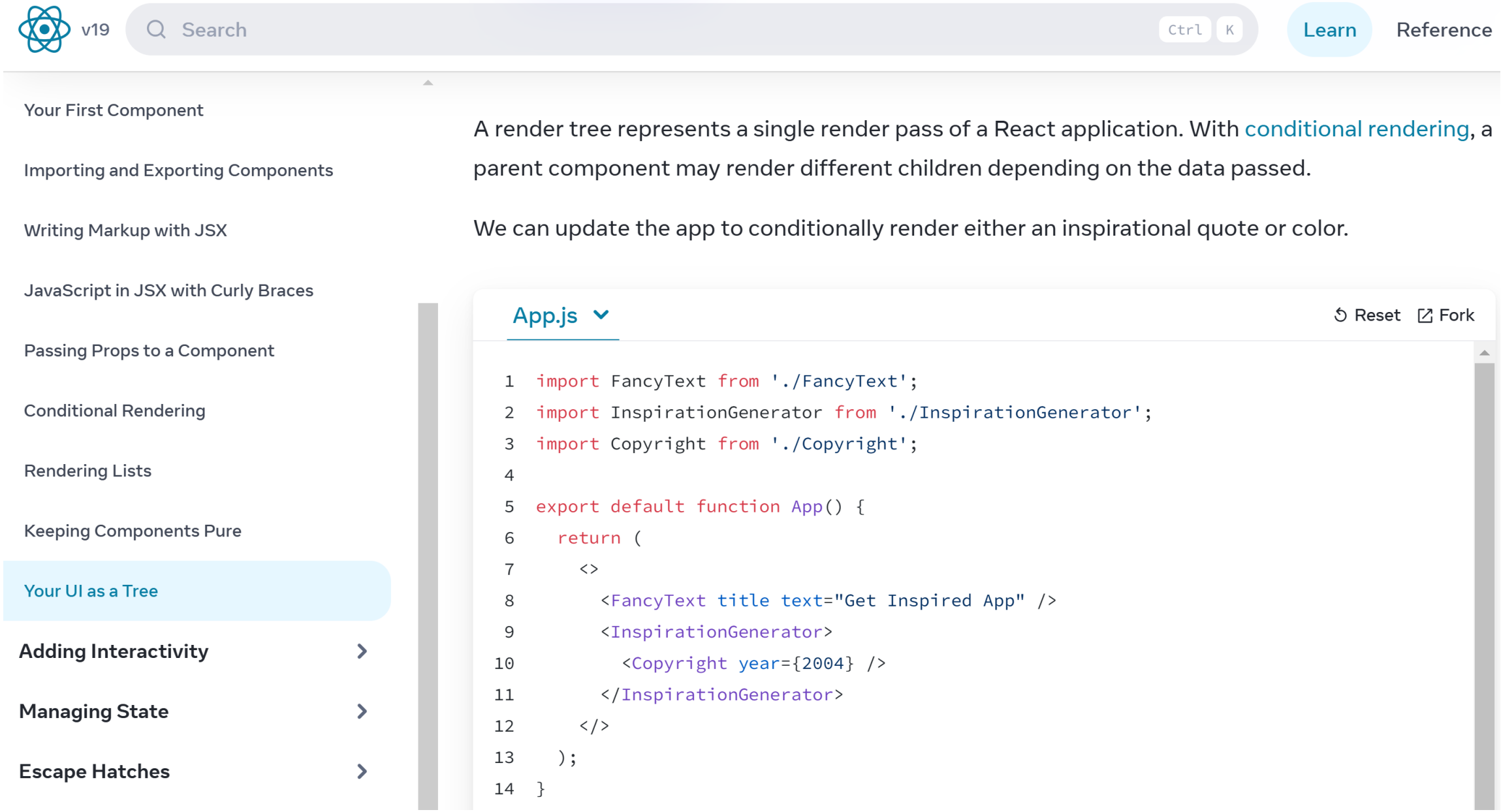
Finally, we can also use the custom element constructor itself. That is, the above code is equivalent to:

```
const flagIcon = new FlagIcon()
flagIcon.country = "jp"
document.body.appendChild(flagIcon)
```

Exploring React Features
Comparison with Angular

☑ Html Rendering template in Component	const [x, setX] = useState() return (<> text { expr } </>)	{{ expr }}
Calling a child component from a Component		<app-child />
templating with "If" and "For"		@if(..) { } @else { .. } @for() { .. }
binding properties to a Component		field = input(); <app-child [field]="expr">
binding Callback		onEvent = signal(); <app-child (onEvent)="cb()">
Bi-Directional Binding ?		<app-child [(field)] ="parentField">
Routing ?		<a routerLink="/compA"> or router.navigate(['compA'])
More ?		form, i18n, animation, etc etc

Calling a child component from a Component



Exploring React Features
Comparison with Angular

Html Rendering template in Component	const [x, setX] = useState() return (<> text { expr } </>)	{{ expr }}
✅ Calling a child component from a Component	return (<> <AppChild /> </>)	<app-child />
templating with "If" and "For"		@if(..) { } @else { .. } @for() { .. }
binding properties to a Component		field = input(); <app-child [field]="expr">
binding Callback		onEvent = signal(); <app-child (onEvent)="cb()">
Bi-Directional Binding ?		<app-child [(field)] ="parentField">
Routing ?		<a routerLink="/compA"> or router.navigate(['compA'])
More ?		form, i18n, animation, etc etc

Equivalent of NG @if()
{ cond && <Elt/> }

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.** With `&&`, you could conditionally render the checkmark only if `isPacked` is `true`:

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

plain old "If()", and local variable with <HTML>

App.js

Download Reset Fork

```
1  function Item({ name, isPacked }) {
2    let itemContent = name;
3    if (isPacked) {
4      itemContent = (
5        <del>
6          {name + " ✅"}
7        </del>
8      );
9    }
10   return (
11     <li className="item">
12       {itemContent}
13     </li>
14   );
15 }
16
```

equivalent of NG @if() { } @else { }

If you're not familiar with JavaScript, this variety of styles might seem overwhelming at first. However, learning them will help you read and write any JavaScript code — and not just React components! Pick the one you prefer for a start, and then consult this reference again if you forget how the other ones work.

Recap

- In React, you control branching logic with JavaScript.
- You can return a JSX expression conditionally with an `if` statement.
- You can conditionally save some JSX to a variable and then include it inside other JSX by using the curly braces.
- In JSX, `{cond ? <A /> : <B />}` means *"if `cond`, render `<A />`, otherwise `<B />`"*.
- In JSX, `{cond && <A />}` means *"if `cond`, render `<A />`, otherwise nothing"*.
- The shortcuts are common, but you don't have to use them if you prefer plain `if`.

Equivalent of NG @for()
using JS .map(x => (<> {x} </>)) function!

1. **Move** the data into an array:

```
const people = [
  'Creola Katherine Johnson: mathematician',
  'Mario José Molina-Pasquel Henríquez: chemist',
  'Mohammad Abdus Salam: physicist',
  'Percy Lavon Julian: chemist',
  'Subrahmanyan Chandrasekhar: astrophysicist'
];
```

2. **Map** the `people` members into a new array of JSX nodes, `listItems`:

```
const listItems = people.map(person => <li>{person}</li>);
```

3. **Return** `listItems` from your component wrapped in a `<ul>`:

```
return <ul>{listItems}</ul>;
```

Exploring React Features
Comparison with Angular

Html Rendering template in Component	const [x, setX] = useState() return (<> text { expr } </>)	{{ expr }}
Calling a child component from a Component	return (<> <AppChild /> </>)	<app-child />
☑ templating with "If" and "For"	{cond && (<> </>) } {items.map(i => (<> {i} </>)) }	@if(..) { } @else { .. } @for() { .. }
binding properties to a Component		field = input(); <app-child [field]="expr">
binding Callback		onEvent = signal(); <app-child (onEvent)="cb()">
Bi-Directional Binding ?		<app-child [(field)] ="parentField">
Routing ?		<a routerLink="/compA"> or router.navigate(['compA'])
More ?		form, i18n, animation, etc etc

Passing Properties
equivalent to NG input() or "@Input"

Now you can read these props inside the `Avatar` component.

Step 2: Read props inside the child component

You can read these props by listing their names `person, size` separated by the commas inside `({` and `})` directly after `function Avatar`. This lets you use them inside the `Avatar` code, like you would with a variable.

```
function Avatar({ person, size }) {
  // person and size are available here
}
```

Reminder... TypeScript
Object Parameter De-Structuring

In Typescript (nothing specific to React)

```
myFunc ( { a: number, b: string, ...others } ) {  a ... }
```

is equivalent to ..

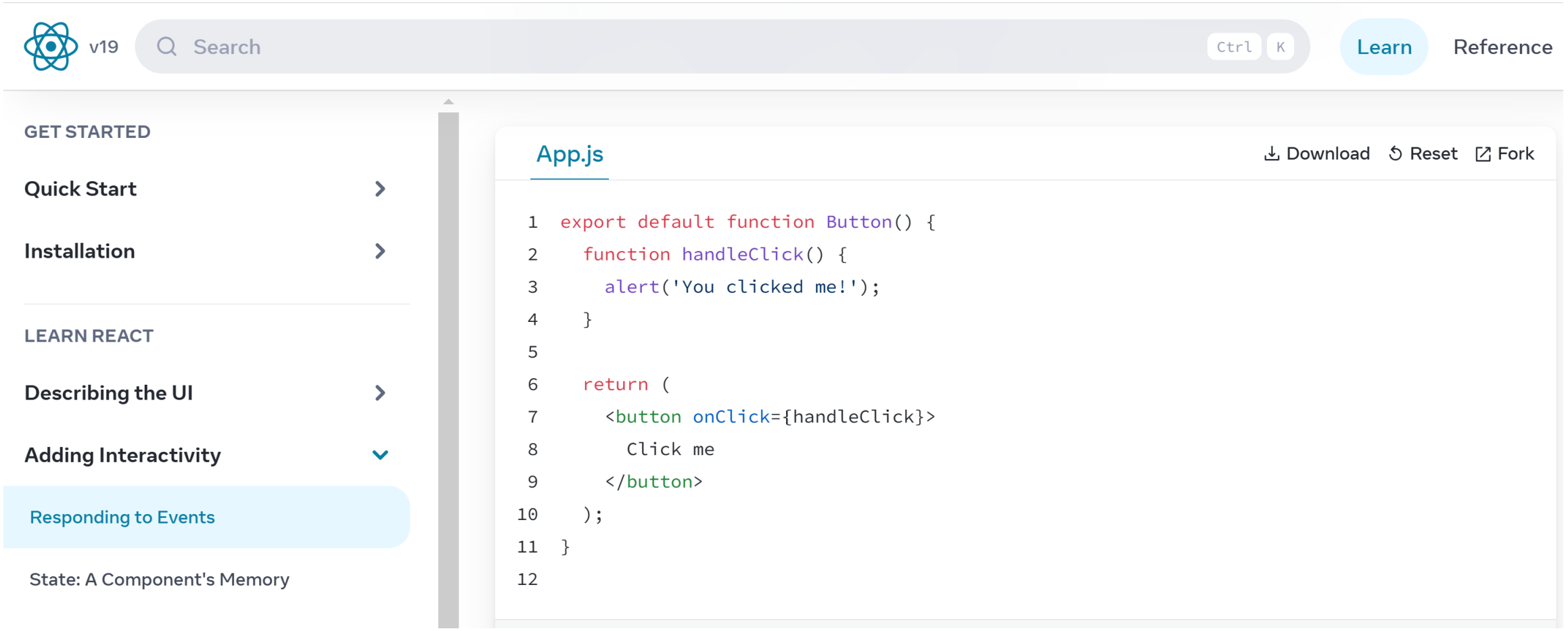
```
interface TProps {
  a: number;
  b: string;
}
myFunc ( props :  TProps ) {    props.a ...   }
```

... NOTICE: props can contains others members

Exploring React Features
Comparison with Angular

Html Rendering template in Component	const [x, setX] = useState() return (<> text { expr } </>)	{{ expr }}
Calling a child component from a Component	return (<> <AppChild /> </>)	<app-child />
templating with "If" and "For"	{cond && (<> </>) } {items.map(i => (<> {i} </>)) }	@if(..) { } @else { .. } @for() { .. }
☑ binding properties to a Component	function AppChild({ field }) {..} <AppChild field={expr} />	field = input(); <app-child [field]="expr">
binding Callback		onEvent = signal(); <app-child (onEvent)="cb()">
Bi-Directional Binding ?		<app-child [(field)] ="parentField" />
Routing ?		<a routerLink="/compA"> or router.navigate(['compA'])
More ?		form, i18n, animation, etc etc

Responding to Event



binding callback Outside of Component
.. same as properties: pass function callback

```
import { memo } from 'react';

const ShippingForm = memo(function ShippingForm({ onSubmit }) {
  // ...
});
```

!! Re-Rendering Performance !!

With this change, `ShippingForm` will skip re-rendering if all of its props are the *same* as on the last render. This is when caching a function becomes important! Let's say you defined `handleSubmit` without `useCallback`:

```
function ProductPage({ productId, referrer, theme }) {
  // Every time the theme changes, this will be a different function...
  function handleSubmit(orderDetails) {
    post('/product/' + productId + '/buy', {
      referrer,
      orderDetails,
    });
  }

  return (
    <div className={theme}>
      {/* ... so ShippingForm's props will never be the same, and it will re-render every time
      <ShippingForm onSubmit={handleSubmit} />
    </div>
  );
}
```

pass Function pointer,
or recreate: function() { ..} or () => { ..}

Pitfall

Functions passed to event handlers must be passed, not called. For example:

passing a function (correct)	calling a function (incorrect)
`<button onClick={handleClick}>`	`<button onClick={handleClick()}>`

The difference is subtle. In the first example, the `handleClick` function is passed as an `onClick` event handler. This tells React to remember it and only call your function when the user clicks the button.

In the second example, the `()` at the end of `handleClick()` fires the function *immediately* during rendering, without any clicks. This is because JavaScript inside the JSX `{` and `}` executes right away.

When you write code inline, the same pitfall presents itself in a different way:

passing a function (correct)	calling a function (incorrect)
`<button onClick={() => alert('...')}>`	`<button onClick={alert('...')}>`

binding callback Outside of Component wrap with Hook "useCallback(() => ...)"

Skipping re-rendering of components

When you optimize rendering performance, you will sometimes need to cache the functions that you pass to child components. Let's first look at the syntax for how to do this, and then see in which cases it's useful.

To cache a function between re-renders of your component, wrap its definition into the `useCallback` Hook:

```
import { useCallback } from 'react';

function ProductPage({ productId, referrer, theme }) {
  const handleSubmit = useCallback((orderDetails) => {
    post('/product/' + productId + '/buy', {
      referrer,
      orderDetails,
    });
  }, [productId, referrer]);
  // ...
```

<App onXX={useCallback((x) => f(x,y), [y])} >

```
import { useCallback } from 'react';

function ProductPage({ productId, referrer, theme }) {
  const handleSubmit = useCallback((orderDetails) => {
    post('/product/' + productId + '/buy', {
      referrer,
      orderDetails,
    });
  }, [productId, referrer]);

  // ...
  return (
    <div className={theme}>
      <ShippingForm onSubmit={handleSubmit} />
    </div>
  );
```

Exploring React Features
Comparison with Angular

Html Rendering template in Component	const [x, setX] = useState() return (<> text { expr } </>)	→	{{ expr }}
Calling a child component from a Component	return (<> <AppChild /> </>)	→	<app-child />
templating with "If" and "For"	{cond && (<> </>) } {items.map(i => (<> {i} </>)) }	→	@if(..) { } @else { .. } @for() { .. }
binding properties to a Component	function AppChild({ field }) {..} <AppChild field={expr} />	→	field = input(); <app-child [field]="expr">
☑ binding Callback	<AppChild onX="{ () => f() }" on={useCallback((x)=>f(x,y), [y])}>	→	onEvent = signal(); <app-child (onEvent)="cb()">
Bi-Directional Binding ?		→	<app-child [(field)] ="parentField">
Routing ?		→	<a routerLink="/compA"> or router.navigate(['compA'])
More ?		→	form, i18n, animation, etc etc

NO Bi-Directional Binding in React !!??

in react, is there bi-directional binding ?

In React, there is no native support for **bi-directional binding** like in frameworks such as Angular. Instead, React follows a **unidirectional data flow**, where:

1. **State** flows from parent to child components via props.
2. **Events** flow from child to parent via callback functions.

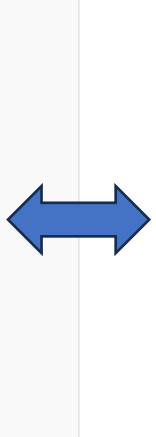
That said, you can implement functionality similar to bi-directional binding by combining **state management** and **event handlers**. This is commonly used in form elements.

useState + explicit {render} + callback setter

tsx

Copy code

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

const BiDirectionalBinding: React.FC = () => {
    const [value, setValue] = useState<string>("");

    const handleChange = (event: React.ChangeEvent<HTMLInputElement>) => {
        setValue(event.target.value); // Updates state when input changes
    };

    return (
        <div>
            <input type="text" value={value} onChange={handleChange} />
            <p>Current Value: {value}</p>
        </div>
    );
};

export default BiDirectionalBinding;
```



value = '';

<input type="text" **[(ngModel)]="value"**>

How it Works:

1. The `value` prop of the input element is controlled by the `value` state.
2. The `onChange` event updates the state when the user types in the input field.
3. The state (`value`) flows back into the input, creating a loop resembling bi-directional binding.

Bi-directional Bindings Frameworks, or Redux ?

frameworks for react bi-directional bindings ?

Here are some frameworks and libraries that facilitate **bi-directional bindings** in React applications. These tools simplify synchronizing state and user input, ensuring updates propagate seamlessly between the UI and the underlying data model.

1. React Hook Form

- **Description**: Lightweight library for managing form state in React.
- **Features**:
  - Minimizes re-renders for better performance.
  - Simple integration with validation libraries (e.g., Yup, Zod).
  - Supports complex forms with nested fields.

2. Formik

- **Description**: Form state management library, ideal for handling forms and validation.
- **Features**:
  - Declarative form management.
  - Built-in support for validation with Yup.
  - Suitable for larger, complex forms.

3. MobX

- **Description**: State management library emphasizing reactivity.
- **Features**:
  - Automatic UI updates for observable state changes.
  - Simple, scalable, and great for bi-directional binding.

4. Redux Form (Deprecated but still used)

- **Description**: Extends Redux for form state management.
- **Features**:
  - Form state is stored in the Redux store.
  - Supports validation and field-level state.

5. React Final Form

- **Description**: Lightweight library for managing form state, similar to Redux Form.
- **Features**:
  - Declarative form state management.
  - Built-in validation support.

6. Recoil

- **Description**: State management library for React, designed for fine-grained state updates.
- **Features**:
  - Atom-based state management for easy form state handling.
  - Reactivity with minimal boilerplate.

7. Zustand

- **Description**: Lightweight state management library.
- **Features**:
  - Easy-to-use and small footprint.
  - Great for local and global state in forms.

8. Valtio

- **Description**: A proxy-based state management library.
- **Features**:
  - Seamless bi-directional bindings with React.
  - Reactive state updates without actions or boilerplate.

One Framework a Day, The Doctor Away

Summary of Frameworks:

Framework/Library	Key Features	When to Use
React Hook Form	Lightweight, minimal re-renders	Simple to medium forms with validation.
Formik	Declarative, Yup integration	Large, complex forms with validation needs.
MobX	Reactive state management	Apps needing global state with form support.
Redux Form	Form state in Redux store	Legacy projects already using Redux.
React Final Form	Declarative and simple	Alternative to Redux Form.
Recoil	Atom-based state	Fine-grained state management.
Zustand	Lightweight, flexible	Simple apps or lightweight state needs.
Valtio	Proxy-based reactive state	Modern apps with minimal boilerplate.

Choose based on your app's complexity, performance requirements, and familiarity with the library.

Using MobX

sample react bi-directional binding with mobx

Using MobX for bi-directional data binding in a React application involves setting up observable state and binding it to input elements. Here's a step-by-step example:

---

1. **Install MobX and MobX-React**

bash Copy code

```
npm install mobx mobx-react-lite
```

MobX Store (=Model, ~Model-View pattern)

2. **Create an Observable Store**

Define a MobX store with an observable property and an action to update it.

typescript Copy code

```typescript
// store.ts
import { makeAutoObservable } from "mobx";

class FormStore {
  name: string = "";

  constructor() {
    makeAutoObservable(this);
  }

  setName(value: string) {
    this.name = value;
  }
}

export const formStore = new FormStore();
```

3. **React Component with Bi-Directional Binding**

Use `observer` from `mobx-react-lite` to make the React component reactive.

tsx Copy code

```tsx
// App.tsx
import React from "react";
import { observer } from "mobx-react-lite";
import { formStore } from "./store";

const App = observer(() => {
  return (
    <div>
      <label>
        Name:
        <input
          type="text"
          value={formStore.name}
          onChange={(e) => formStore.setName(e.target.value)}
        />
      </label>
      <p>Typed Name: {formStore.name}</p>
    </div>
  );
});
```

Exploring React Features
Comparison with Angular

Html Rendering template in Component	const [x, setX] = useState() return (<> text { expr } </>)	→	{{ expr }}
Calling a child component from a Component	return (<> <AppChild /> </>)	→	<app-child />
templating with "If" and "For"	{cond && (<> </>) } {items.map(i => (<> {i} </>)) }	→	@if(..) { } @else { .. } @for() { .. }
binding properties to a Component	function AppChild({ field }) {..} <AppChild field={expr} />	→	field = input(); <app-child [field]="expr">
binding Callback	<AppChild onX="{ () => f() }" on={useCallback((x)=>f(x,y), [y])}>	→	onEvent = signal(); <app-child (onEvent)="cb()">
☑ Bi-Directional Binding ?	NOT NATIVE.. use Mobx?	→	<app-child [(field)] ="parentField">
Routing ?		→	<a routerLink="/compA"> or router.navigate(['compA'])
More ?		→	form, i18n, animation, etc etc

NO native Routing in React !!

is there routing native in react ?

No, **routing is not natively built into React**. React is a JavaScript library focused solely on building user interfaces, and it does not include routing or other advanced features like state management or data fetching. These functionalities can be added through third-party libraries, with **React Router** being the most commonly used library for routing in React applications.

Why Isn’t Routing Native to React?

React adheres to a "library" philosophy rather than being a full-fledged framework like Angular. This means:

1. It focuses on **UI rendering and component composition**.
2. Other functionalities, such as routing, state management, and side effects, are left to external libraries.

npm install react-router-dom

```
npm install react-router-dom
```

Basic Example

Setting up Routes:

1. **Wrap your app with** `<BrowserRouter>`**:** This enables routing in your application.
2. **Define routes using** `<Routes>` **and** `<Route>`**:** Specify the paths and their associated components.

tsx Copy code

```tsx
import React from "react";
import ReactDOM from "react-dom";
import { BrowserRouter, Routes, Route } from "react-router-dom";

const Home = () => <h1>Home Page</h1>;
const About = () => <h1>About Page</h1>;

const App = () => {
    return (
        <BrowserRouter>
            <Routes>
                <Route path="/" element={<Home />} />
                <Route path="/about" element={<About />} />
            </Routes>
        </BrowserRouter>
    );
};

ReactDOM.render(<App />, document.ge mentById("root"));
```

Exploring React Features
Comparison with Angular

Html Rendering template in Component	const [x, setX] = useState() return (<> text { expr } </>)	→	{{ expr }}
Calling a child component from a Component	return (<> <AppChild /> </>)	→	<app-child />
templating with "If" and "For"	{cond && (<> </>) } {items.map(i => (<> {i} </>)) }	→	@if(..) { } @else { .. } @for() { .. }
binding properties to a Component	function AppChild({ field }) {..} <AppChild field={expr} />	→	field = input(); <app-child [field]="expr">
binding Callback	<AppChild onX="{ () => f() }" on={useCallback((x)=>f(x,y), [y])}>	→	onEvent = signal(); <app-child (onEvent)="cb()">
Bi-Directional Binding ?	NOT NATIVE.. use Mobx?	→	<app-child [(field)] ="parentField">
✅ Routing ?	NOT NATIVE.. use react-router?	→	<a routerLink="/compA"> or router.navigate(['compA'])
More ?		→	form, i18n, animation, etc etc

No Form in native React ?!!

how to have form with validation in native react?

Creating a form with validation in **native React** involves managing the form state and implementing validation logic yourself. React doesn't have built-in form handling or validation utilities, but you can achieve this using **state** and **event handlers**.

Here's a step-by-step guide to creating a form with validation in native React:

Exploring React Features
Comparison with Angular

Html Rendering template in Component	const [x, setX] = useState() return (<> text { expr } </>)	→	{{ expr }}
Calling a child component from a Component	return (<> <AppChild /> </>)	→	<app-child />
templating with "If" and "For"	{cond && (<> </>) } {items.map(i => (<> {i} </>)) }	→	@if(..) { } @else { .. } @for() { .. }
binding properties to a Component	function AppChild({ field }) {..} <AppChild field={expr} />	→	field = input(); <app-child [field]="expr">
binding Callback	<AppChild onX="{ () => f() }" on={useCallback((x)=>f(x,y), [y])}>	→	onEvent = signal(); <app-child (onEvent)="cb()">
Bi-Directional Binding ?	NOT NATIVE.. use Mobx?	→	<app-child [(field)] ="parentField">
Routing ?	NOT NATIVE.. use react-router?	→	<a routerLink="/compA"> or router.navigate(['compA'])
☑ More ?	NOT NATIVE.. do it yourself !	→	form, i18n, animation, etc etc

React PROs

React is very fashion since ~4 years

Used in lot of companies and Open-Source Projects

becoming a de-facto standard, among the top 3 frameworks: React, Angular, Vue.Js

React PROS

Simplistic things (no binding, no state) are possible in a simple way in React

Very easy to call stateless sub components, as Pure functions.
React forces you to use "pure" function, with exceptions to "purity"

React favors a functional approach, and encourage creating many smalll function components

TSX is compacting both the HTML and the TS/JS in the same location (is it a PROS or a CONS?)

React PROS (or CONS ?)
React is Minimalist : only Rendering DOM

React "looks" SIMPLER than Angular, because there is almost NOTHING in it !

React is just a DOM rendering template engine

React features is a SMALL subset of Angular: "{{ exr }}" , [field]="", (event)=""

major features relies on TSX/JSX for html <-> {JavaScript} interleaving
and passing parameters in JS/TS functions

React CONS : no class
going away from W3c CustomElement Standard

The Hook "functionalities" in React are very tricky / difficult to understand.
Example: useState(), useEffect(), useCallback() ...

They are conter intuitives, and need HUGE documentation/training to master strange effects compared to the very SMALL features they provide

React used to have classes, but now legacy !?
... going away from Standard W3C Specification of "Custom Element" as classes

React CONS : Rendering with Virtual DOM

React = Only a "rendering" engine, using a Virtual DOM
  very bad for memory performances : everything x2 for memory

Once upon a time, It used to be "good" to have Virtual DOM
  because Microsoft Internet Explorer DOM was so bad. now useless.

Example of lighter/modern alternatives: Solid.JS, Svelte, Vue.js
Solid.JS = very similar to React TSX, but NO Virtual DOM... only real fast DOM

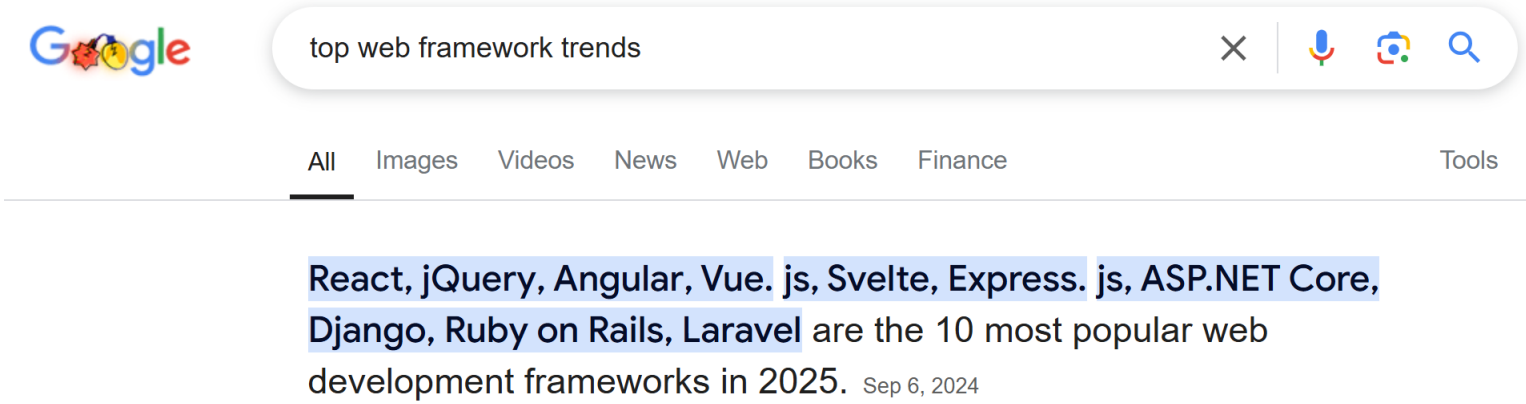
React CONS

React looks SIMPLER than Angular, because there is almost NOTHING in it !

For all features, you need additional frameworks : MOBX, Router, etc..

There is
- No Router
- No Bi-Directional Binding
- tons of frameworks, difficult to choose
- Redux is a headache, overkill
- No Form

Final Words



React is #1 in 2024

The Web is Absurd ... why jQuery still in 2024? rails? ... confusing SPA technologies and server-side

Questions ?

arnaud.nauwynck@gmail.com