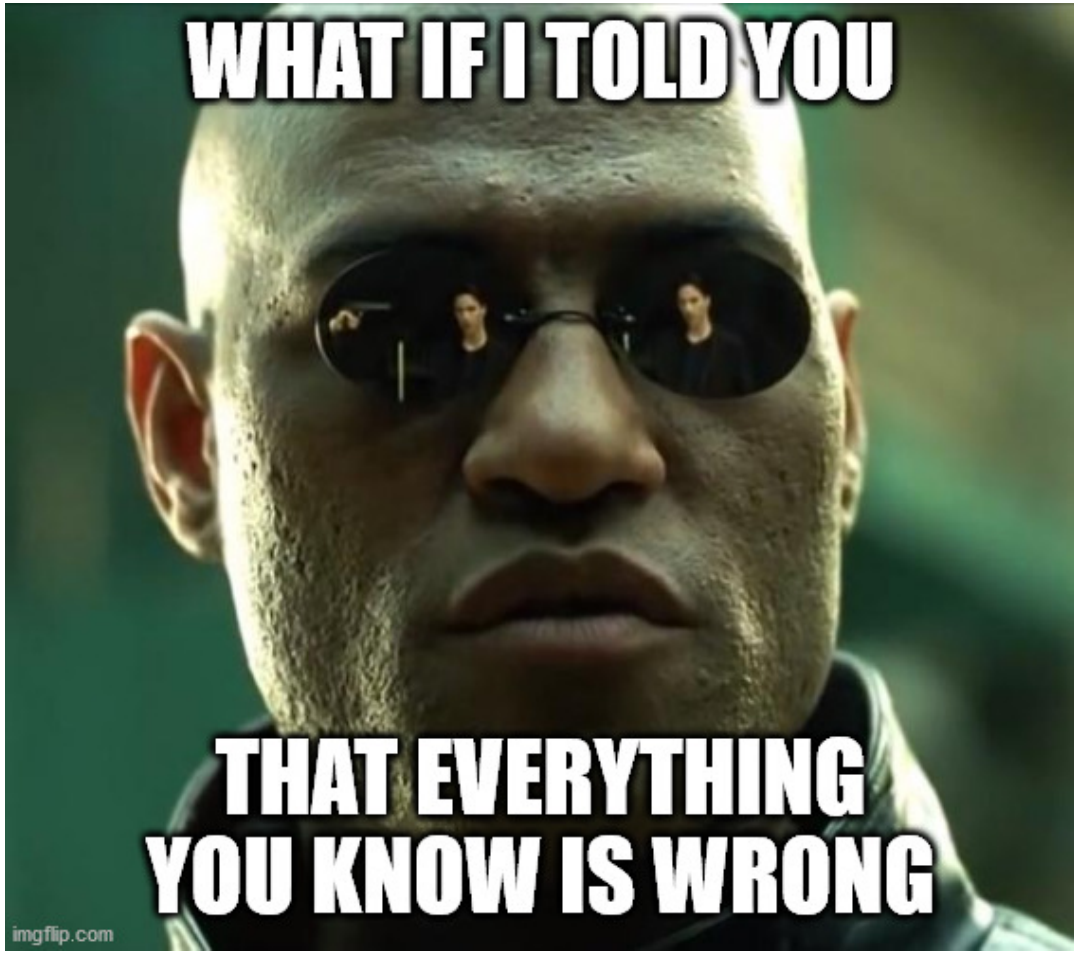
<WA1/>
<AW1/>
2023

Introduction to React

JS Frameworks to the rescue

Fulvio Corno
Luigi De Russis
Enrico Masala



Goal

Version 18.2.0
Released on June 14, 2022

- Learn one of the most popular front-end libraries
  - Basic principles
  - Application architecture
  - Programming techniques
- Leverage the knowledge of JS concepts

React

The library for web and native user interfaces

https://react.dev/
https://github.com/facebook/react

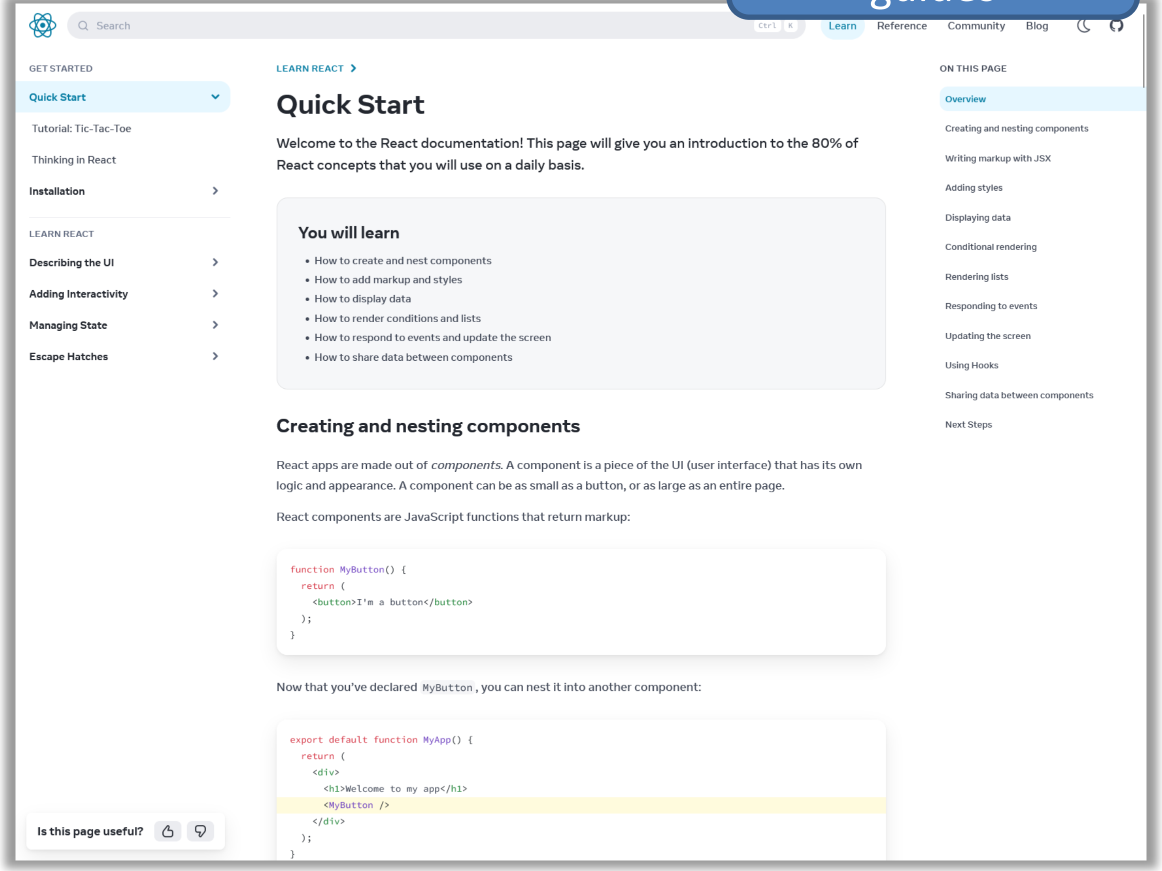
Why a Library?

- Simplify the browser environment
  - Uniform DOM methods
  - More explicit hierarchy
  - **Higher-level** components than HTML elements
  - **Automatic** processing of events and updates

- Simplify the development methods
  - Predefined programming **patterns** and application architecture
  - Lots of compatible plugins and extensions
  - Explicit and rigid **state** management

Main Resources

Tutorials and guides



LEARN REACT >

Quick Start

Welcome to the React documentation! This page will give you an introduction to the 80% of React concepts that you will use on a daily basis.

You will learn

- How to create and nest components
- How to add markup and styles
- How to display data
- How to render conditions and lists
- How to respond to events and update the screen
- How to share data between components

Creating and nesting components

React apps are made out of *components*. A component is a piece of the UI (user interface) that has its own logic and appearance. A component can be as small as a button, or as large as an entire page.

React components are JavaScript functions that return markup:

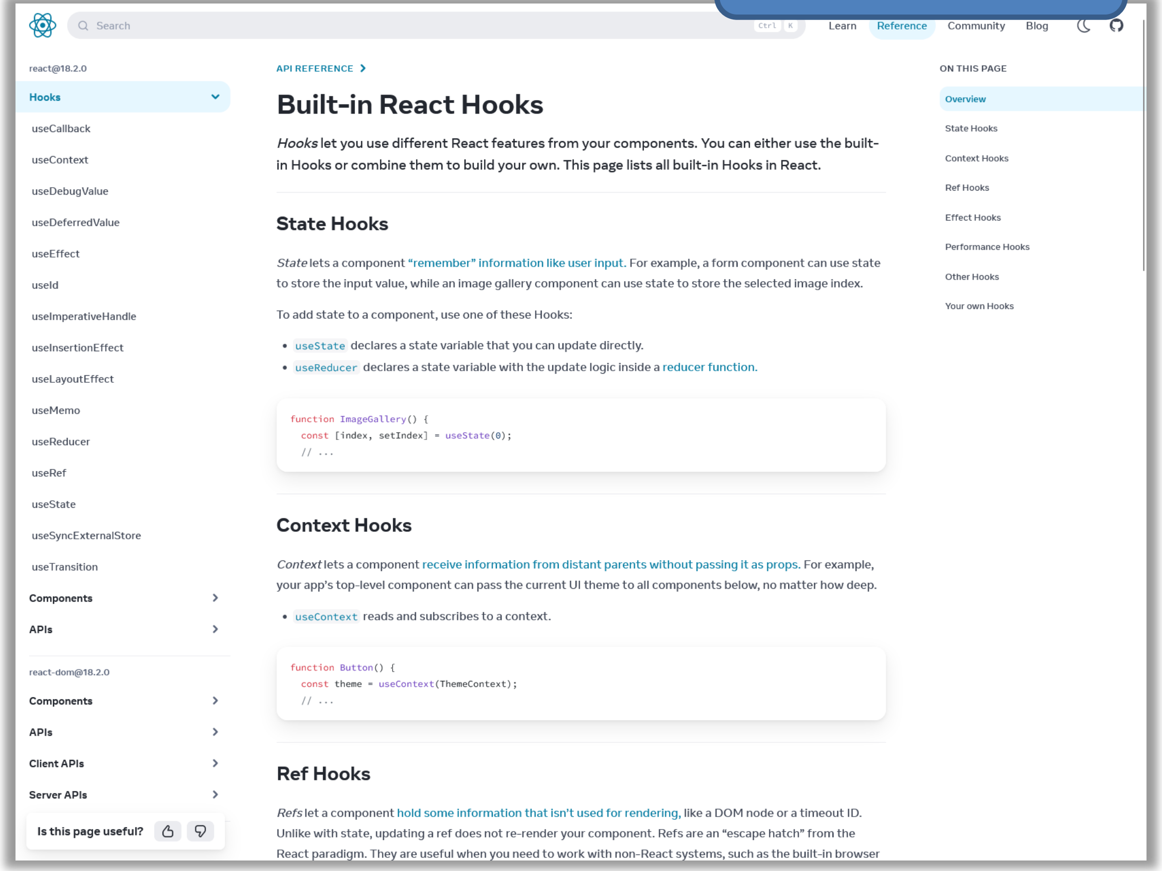
```
function MyButton() {
  return (
    <button>I'm a button</button>
  );
}
```

Now that you've declared `MyButton`, you can nest it into another component:

```
export default function MyApp() {
  return (
    <div>
      <h1>Welcome to my app</h1>
      <MyButton />
    </div>
  );
}
```

https://react.dev/learn

API Reference



API REFERENCE >

Built-in React Hooks

Hooks let you use different React features from your components. You can either use the built-in Hooks or combine them to build your own. This page lists all built-in Hooks in React.

State Hooks

State lets a component "remember" information like user input. For example, a form component can use state to store the input value, while an image gallery component can use state to store the selected image index.

To add state to a component, use one of these Hooks:

- `useState` declares a state variable that you can update directly.
- `useReducer` declares a state variable with the update logic inside a reducer function.

```
function ImageGallery() {
  const [index, setIndex] = useState(0);
  // ...
```

Context Hooks

Context lets a component receive information from distant parents without passing it as props. For example, your app's top-level component can pass the current UI theme to all components below, no matter how deep.

- `useContext` reads and subscribes to a context.

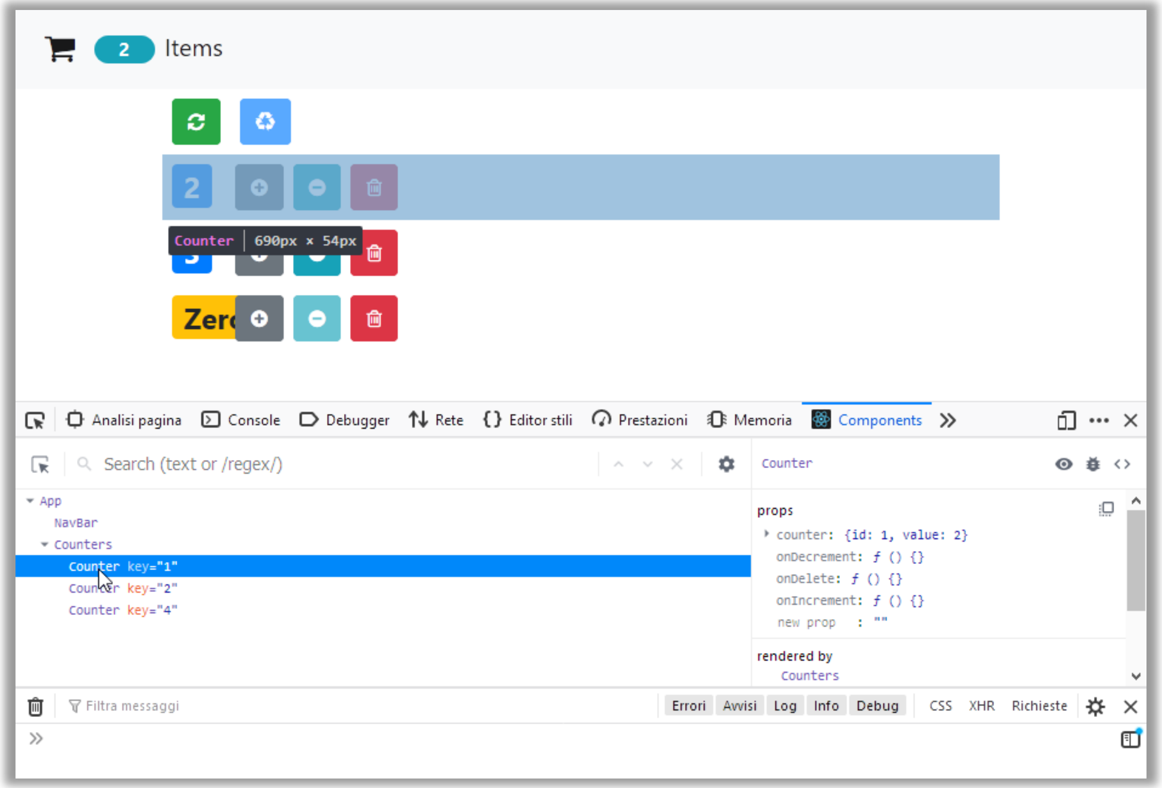
```
function Button() {
  const theme = useContext(ThemeContext);
  // ...
```

Ref Hooks

Refs let a component hold some information that isn't used for rendering, like a DOM node or a timeout ID. Unlike with state, updating a ref does not re-render your component. Refs are an "escape hatch" from the React paradigm. They are useful when you need to work with non-React systems, such as the built-in browser

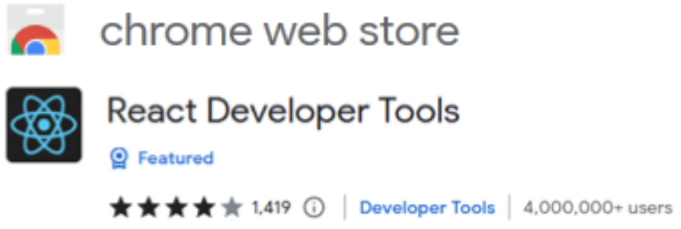
https://react.dev/reference/react

Browser Development Tools



```
▾ App
    NavBar
  ▾ Counters
      Counter key="1"
      Counter key="2"
      Counter key="4"
```

```
Counter

props
 ▸ counter: {id: 1, value: 2}
   onDecrement: ƒ () {}
   onDelete: ƒ () {}
   onIncrement: ƒ () {}
   new prop   : ""

rendered by
   Counters
```



chrome web store

React Developer Tools

Featured

★★★★★ 1,419 | Developer Tools | 4,000,000+ users

https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?hl=en

Firefox Browser ADD-ONS

React Developer Tools

by React

https://addons.mozilla.org/en-US/firefox/addon/react-devtools/

The React Handbook, Flavio Copes
https://flaviocopes.com/page/react-handbook/

A first high-level run about the main design concepts in React

DESIGN PRINCIPLES

React Key Concepts

- Declarative approach
  - Never explicitly manipulate the DOM
  - Never explicitly define the order of operations
  - Just define how each component is going to render itself

- Functional design approach
  - Components as functions
  - Re-render everything on every change (Virtual DOM)
  - Explicit management of the *state* of the application

React is Functional

- UI Fragment = f(`state`, `props`)

- Many components don’t need to manage state
- UI Fragment = f(`props`)
  - Idempotent
  - Immutable

- Jargon note: `props` = *properties*

http://slides.com/johnlynch/reactjs

Immutability

- Reacts exploits **Immutability** of objects, for ease of programming and efficiency of processing
- Component ‘`props`’ are immutable (read-only by the component)
- Component ‘`state`’ is not directly mutable (can be changed only through special calls)
- Functions are ‘pure’ (have no side-effects besides computing the return value)
  - Idempotency (re-rendering the same component always yields the same result)
  - Predictability

Re-Rendering

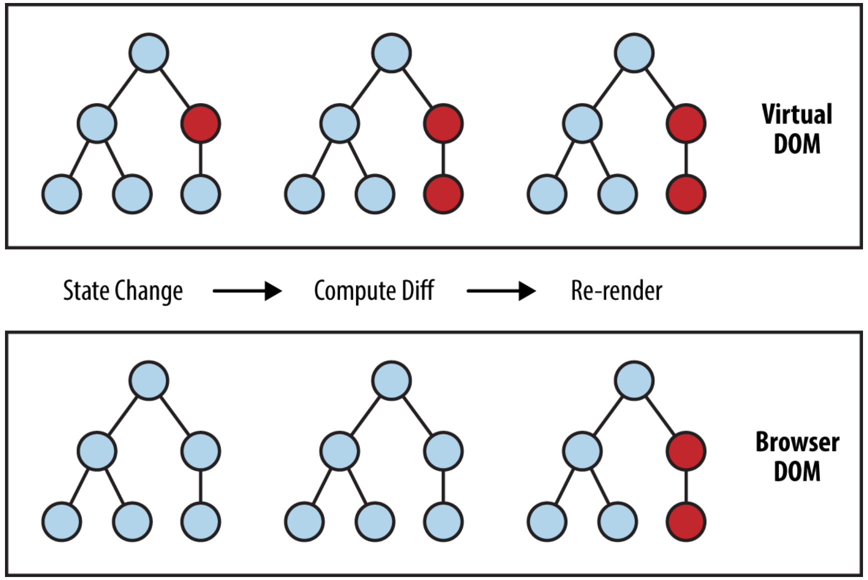
- The application is made of Components
- The entire application is re-rendered:
  - Every time a state is changed
  - Every time a property is changed
- Each Component will re-build itself from scratch
  - With minor variations, or
  - Radically different
- Performance?

Re-Rendering Performance

- Modifications to the DOM are expensive (re-computing layout and updating GUI)
- React implements a **Virtual DOM** layer
  - Internal in-memory data structure, optimized and very fast to update
  - Corrects some DOM anomalies and asymmetries
  - Manages its own set of “synthetic” events
  - After components re-render, React computes the difference between the “old” DOM and the new modified Virtual DOM
  - Only modifications and differences are selectively applied to the browser’s DOM, in batch

Update Cycle

- Build new Virtual DOM tree
- Diff with old one
- Compute minimal set of changes
- Put them in a queue
- Batch render all changes to browser



https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch02.html

http://slides.com/johnlynch/reactjs

Synthetic Events

- React implements its own event system
- A single native event handler at root of each component
- Normalizes events across browsers
- Decouples events from DOM

http://slides.com/johnlynch/reactjs

How React Code is integrated in the DOM

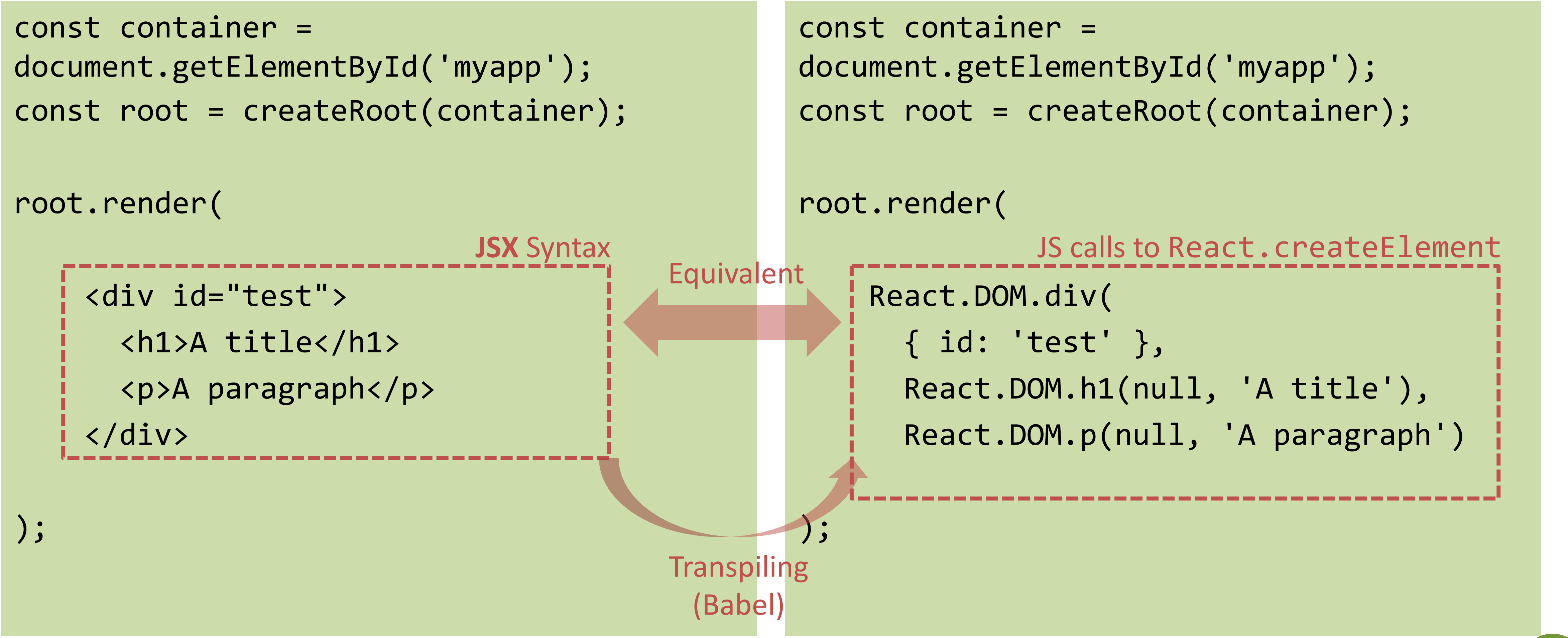
```
const container =
  document.getElementById('root');

const root = createRoot(container);

root.render(<h1>Hello, world!</h1>);
```

DOM **container** node

React **element**

Render **element** into **container**

JSX Syntax



```
const container =
document.getElementById('myapp');
const root = createRoot(container);

root.render(
```

JSX Syntax

```
    <div id="test">
      <h1>A title</h1>
      <p>A paragraph</p>
    </div>

);
```

Equivalent

Transpiling
(Babel)

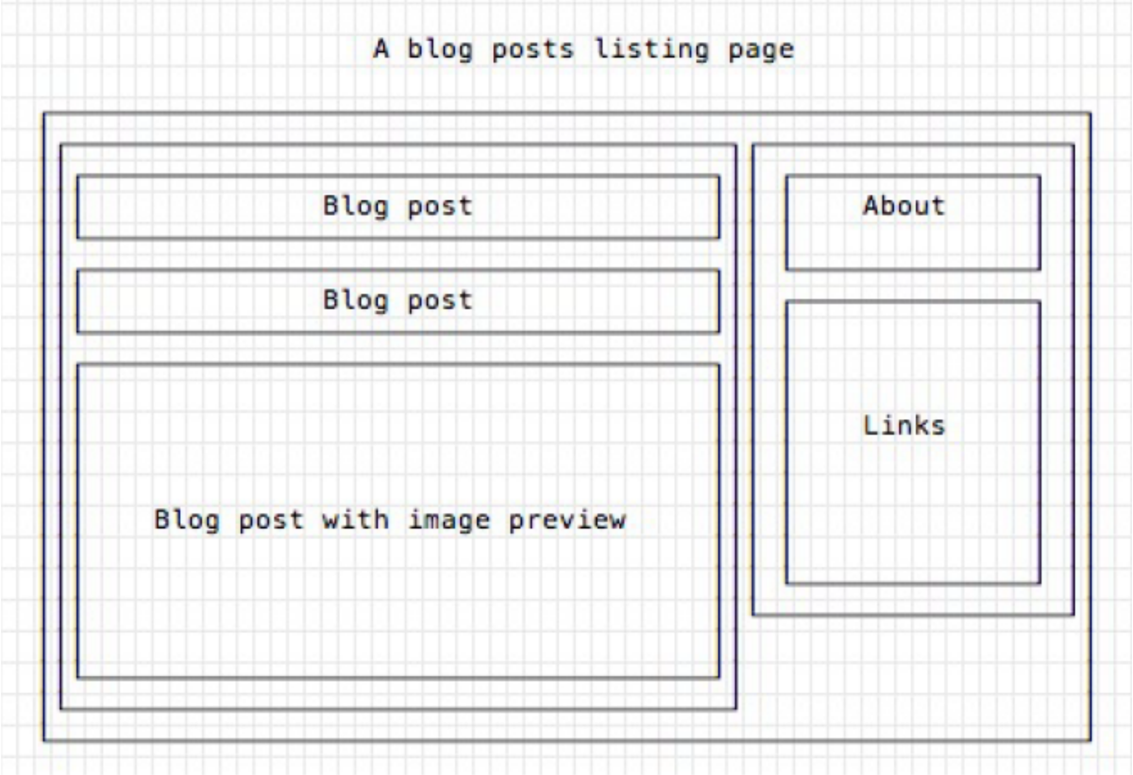
```
const container =
document.getElementById('myapp');
const root = createRoot(container);

root.render(
```

JS calls to React.createElement

```
    React.DOM.div(
      { id: 'test' },
      React.DOM.h1(null, 'A title'),
      React.DOM.p(null, 'A paragraph')

);
```

Components

- Everything on a page is a Component
  - Even simple HTML tags (React.DOM.element)
- Components may be **nested**
- ReactDOM.createRoot().render() builds a component and attaches it to a DOM container



Defining Custom Components

As a function, returning DOM elements

```
const BlogPostExcerpt = () => {
  return (
    <div>
      <h1>Title</h1>
      <p>Description</p>
    </div>
  )
}
```

The function may receive some props

```
const BlogPostContent = (props) => {
  return (
    <div>
        <p>{props.content}</p>
    </div>
  )
}
```

Types of Components

Presentational Components

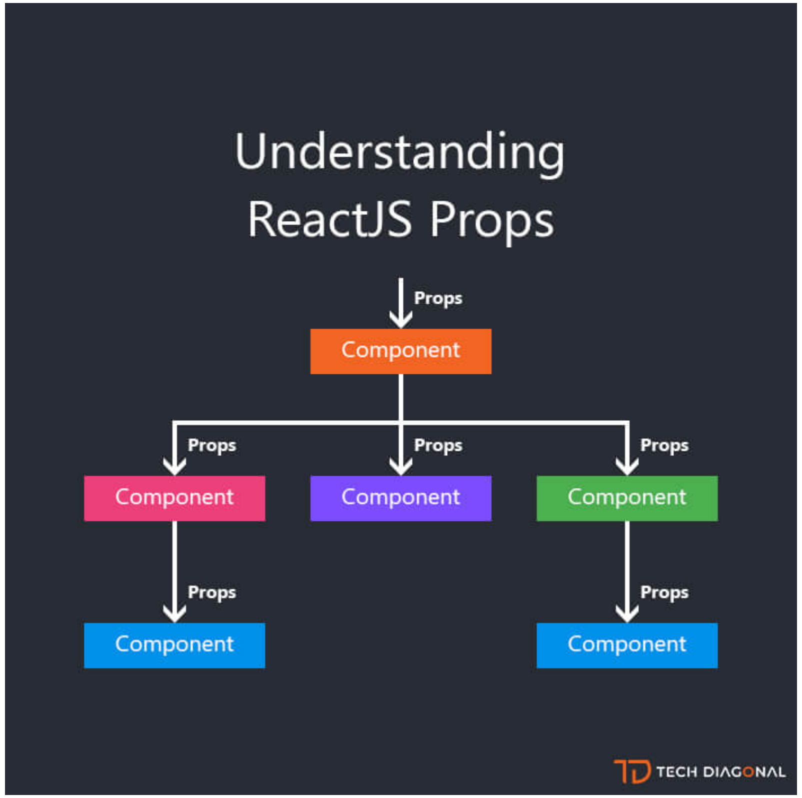
- Generate DOM nodes to be displayed
- Do not manage application state
- Might have some internal state, uniquely for **presentation** purposes

Container Components

- Manage the **state** for a group of children
- May interact with the back-end
- Create (presentational) children to display the information

Props and State

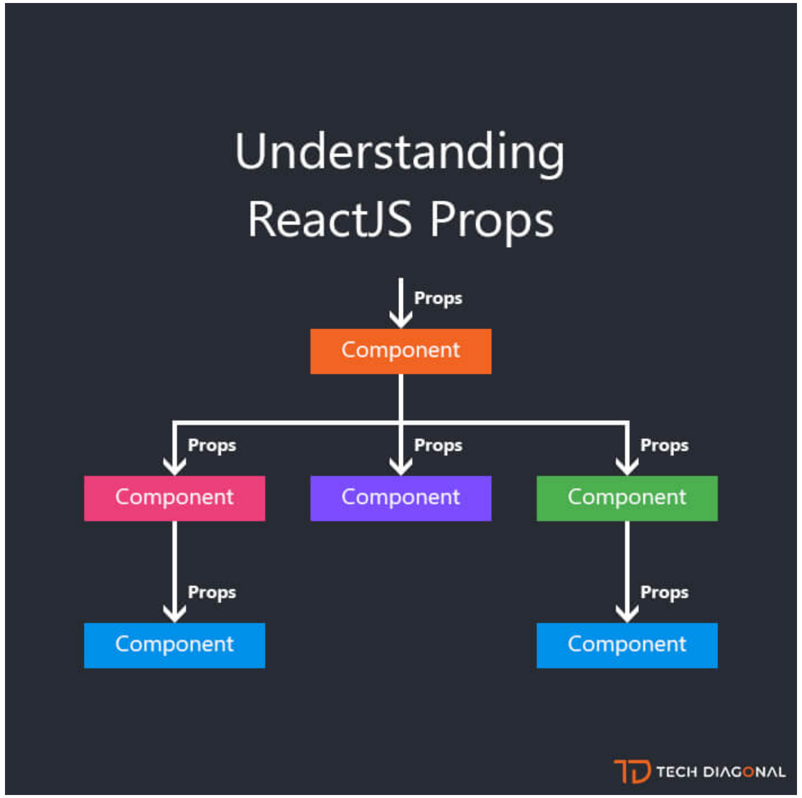
- **Props** (properties) are passed to a component by its parent
  - – Values (strings, objects, …) to configure how the component displays or behaves
    - Top-to-bottom data flow
  - – Functions (callbacks) to access the parent’s methods
    - Bottom-to-top action requests



https://www.techdiagonal.com/reactjs_courses/beginner/understanding-reactjs-props/

Props and State

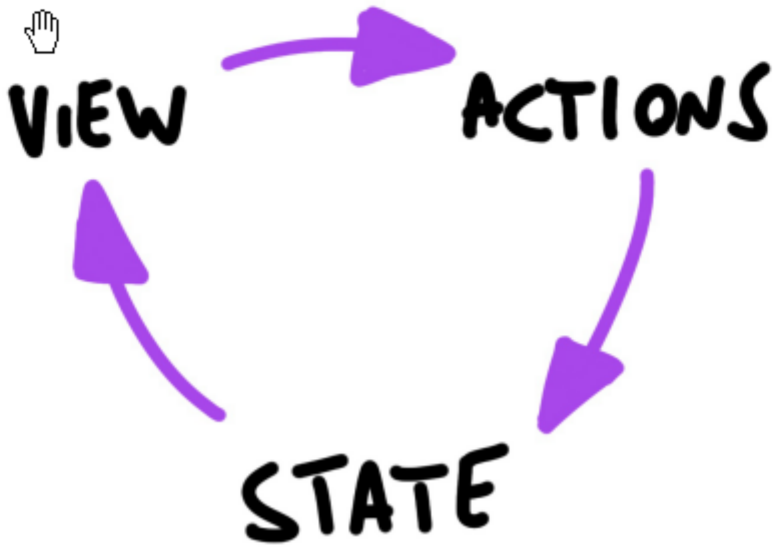
- **State** is a set of variables local to the component
  - Initialized with default value or by props' values
  - Can be mutated only by calling specific methods
    - Asynchronous
    - Will initiate re-rendering of the Virtual DOM
  - Current state value can be passed to children (as props)



https://www.techdiagonal.com/reactjs_courses/beginner/understanding-reactjs-props/

Unidirectional Data Flow

- State is passed to the view and to child components
- Actions are triggered by the view
- Actions can update the state
- The state change is passed to the view and to child component



Corollary

- A **state** is always **owned by one Component**
  - Any data that's affected by this state can only affect Components below it: its children.
- Changing state on a Component will never affect its parent, or its siblings, or any other Component in the application
  - Just its children
- For this reason, state is often **moved up** in the Component tree, so that it can be **shared** between components that need to access it.

Installing, configuring and running the Hello World

FIRST REACT APPLICATION

Basic requirements

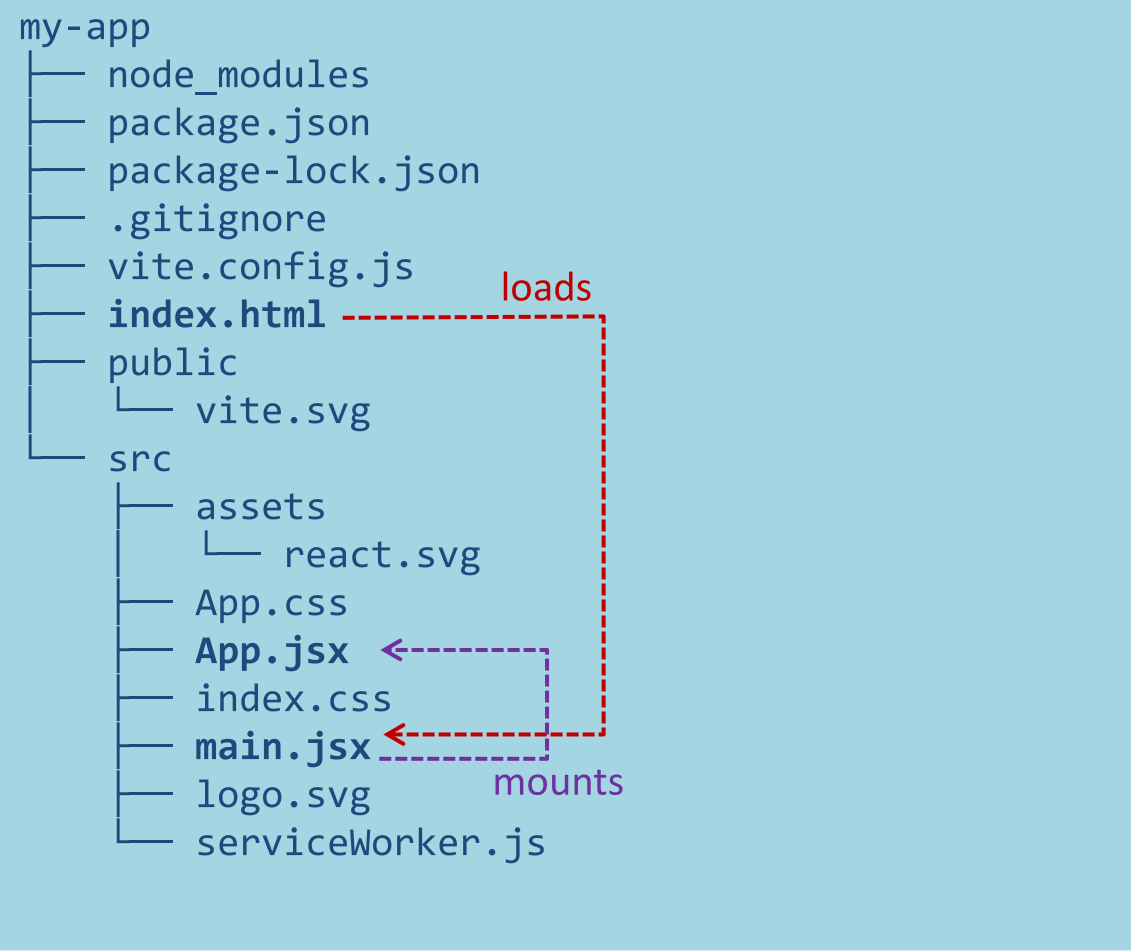
- Import the React library
  - Import several needed libraries
- We want to use **JSX**
  - Babel required
- We need to run on a web server
  - To be able to use modules
    - `import` in JS code
    - `<script type='module'>` in HTML code
  - Avoid problems with CORS
- Implement polyfills for browser compatibility
- Ease app development (edit-save-reload cycle)
- …

Starting With All The Needed Infrastructure

https://vitejs.dev/

1. `npm create vite@latest my-app`
2. From the menu, choose React, then JavaScript
3. `cd` *`my-app`*
4. `npm install`
5. *… 65 Megabytes later …*
6. `npm run dev`
7. Visit http://localhost:5173

Folder Structure



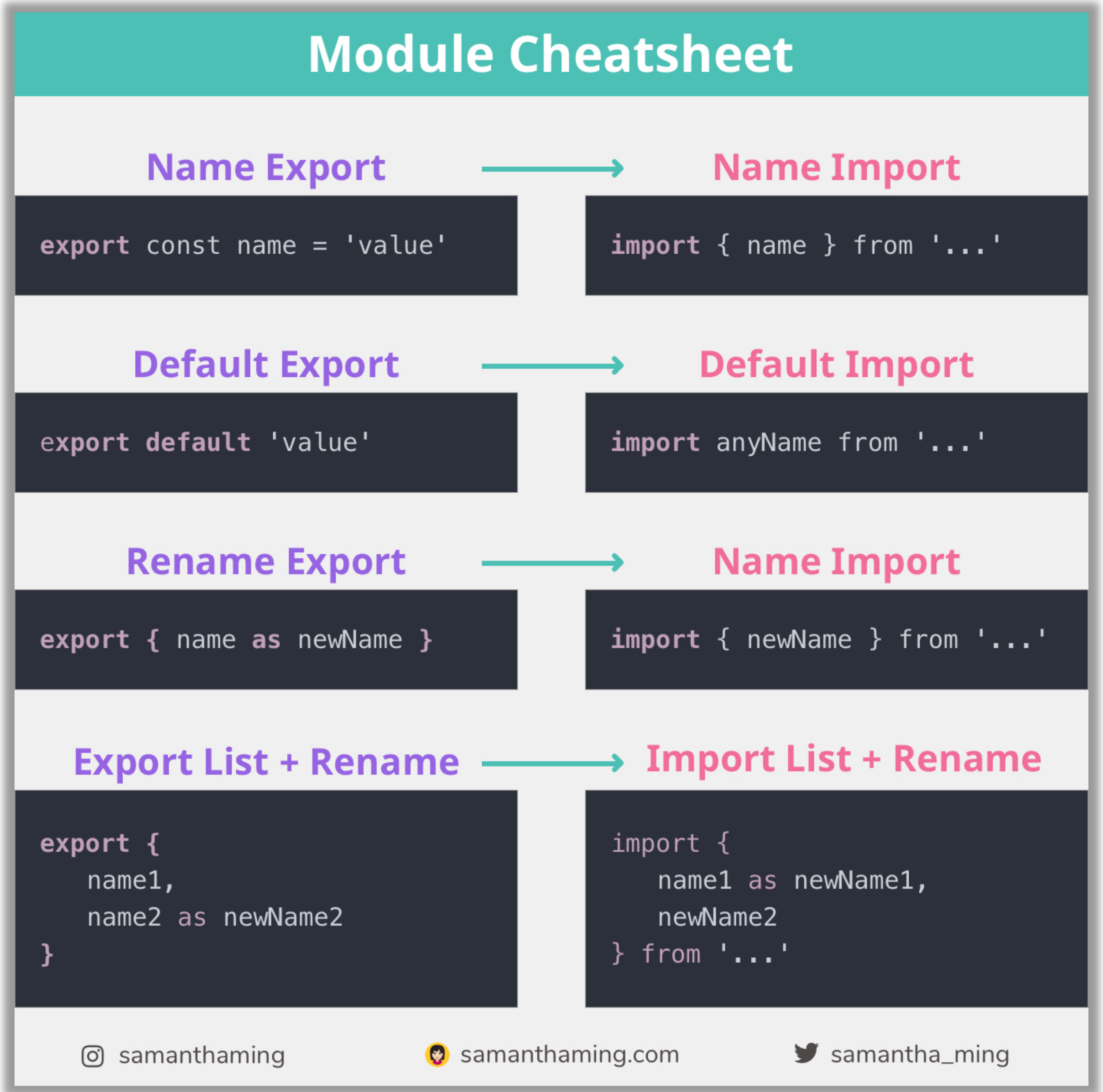
```
my-app
├── node_modules
├── package.json
├── package-lock.json
├── .gitignore
├── vite.config.js
├── index.html
├── public
│   └── vite.svg
└── src
    ├── assets
    │   └── react.svg
    ├── App.css
    ├── App.jsx
    ├── index.css
    ├── main.jsx
    ├── logo.svg
    └── serviceWorker.js
```

- `public` is the web server root
  - Static files go here
- `index.html` is the page template
  - Published at http://localhost:xxxx
  - Automatically reloads when app changes
  - No need to modify, normally
  - Contains an element with id="root"
- `src` contains all scripts
- `src/main.jsx` is the JavaScript entry point
  - Contains the `createRoot` call to mount the App in the #root element
  - Do not touch, normally
- `src/App.jsx` is the file containing your application
  - **Develop here!**
  - Feel free to `import` other components

Importing/Exporting

- The browser uses “ES6 Modules”
  - ECMA Standard
- Uses `import/export` keywords
  - Different than the `require` function used in Node.js

- *More details in a future lesson*



Module Cheatsheet

Name Export ⟶ **Name Import**

```
export const name = 'value'
```

```
import { name } from '...'
```

Default Export ⟶ **Default Import**

```
export default 'value'
```

```
import anyName from '...'
```

Rename Export ⟶ **Name Import**

```
export { name as newName }
```

```
import { newName } from '...'
```

Export List + Rename ⟶ **Import List + Rename**

```
export {
   name1,
   name2 as newName2
}
```

```
import {
   name1 as newName1,
   newName2
} from '...'
```

samanthaming samanthaming.com samantha_ming

https://www.samanthaming.com/tidbits/79-module-cheatsheet/

Example: Hello world

App.jsx

```
function Button(props) {
  if (props.lang === 'it')
    return <button>Ciao!</button>;
  else
    return <button>Hello!</button>;
}

function App() {
  return (
    <p>
      Press here: <Button lang='it' />
    </p>
  );
}

export default App;
```

- App must return the JSX of the whole application
- We may use "custom components"
  - Simply defined as JS functions
  - Receive 'props'
    - The `lang` JSX attribute becomes a property `props.lang`

Example: Components in a Separate File

App.jsx

```
import Button from './Button.jsx';

function App() {
  return (
    <p>
      Premi qui: <Button lang='it' />
    </p>
  );
}

export default App;
```

Button.jsx

```
function Button(props) {
    if (props.lang === 'it')
        return <button>Ciao!</button>;
    else
        return <button>Hello!</button>;
  }

export default Button;
```

Example: Dynamic State

Button.jsx

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

function Button(props) {
    let [buttonLang, setButtonLang] = useState(props.lang) ;

    if (buttonLang === 'it')
        return <button onClick={()=>setButtonLang('en')}>Ciao!</button>;
    else
        return <button onClick={()=>setButtonLang('it')}>Hello!</button>;
}

export default Button;
```

https://react-bootstrap.github.io/

Example: adding Bootstrap

- Bootstrap CSS may be loaded "manually" from index.html

or, better…

- The `react-bootstrap` library delivers many React Components that mimic the various Bootstrap classes
  - `npm install react-bootstrap`
  - `npm install bootstrap`

App.jsx

```
import 'bootstrap/dist/css/bootstrap.min.css';
import { Col, Container, Row } from 'react-
bootstrap';

import MyButton from './Button.jsx';

function App() {
  return (
    <Container>
      <Row>
        <Col>
          Premi qui: <MyButton lang='it' />
        </Col>
      </Row>
    </Container>
  );
}

export default App;
```

Example: adding Bootstrap

https://react-bootstrap.github.io/

Button.jsx

```
import { useState } from "react";
import { Button } from "react-bootstrap";

function MyButton(props) {
    let [buttonLang, setButtonLang] = useState(props.lang) ;

    if (buttonLang === 'it')
        return <Button variant='primary' onClick={()=>setButtonLang('en')}>Ciao!</Button>
    else
        return <Button variant='primary' onClick={()=>setButtonLang('it')}>Hello!</Button>
}

export default MyButton;
```

What’s next?

- Components and props
- JSX
- State and Hooks
- Events
- Forms
- Lifecycle
- Router
- …

License

- These slides are distributed under a Creative Commons license “**Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**”
- **You are free to:**
  - **Share** — copy and redistribute the material in any medium or format
  - **Adapt** — remix, transform, and build upon the material
  - The licensor cannot revoke these freedoms as long as you follow the license terms.
- **Under the following terms:**
  - **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
  - **NonCommercial** — You may not use the material for commercial purposes.
  - **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
  - **No additional restrictions** — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.
- https://creativecommons.org/licenses/by-nc-sa/4.0/