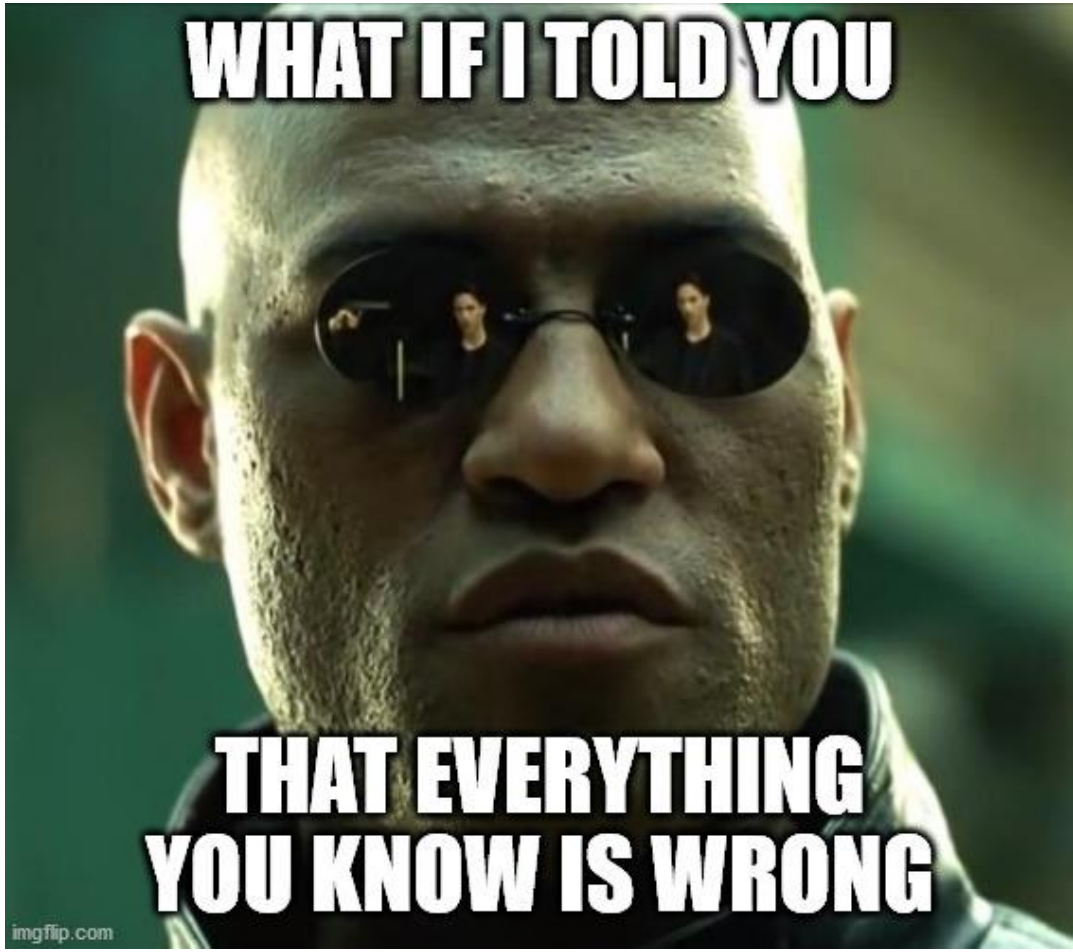
<WA1/>
<AW1/>
2025

Introduction to React

JS Frameworks to the rescue

Fulvio Corno
Luigi De Russis



Goal

- Learn one of the most popular front-end libraries
  - Basic principles
  - Application architecture
  - Programming techniques
- Leverage the knowledge of JS concepts
- Get to know the browser's object models (BOM and DOM)

Version 19.0
Released on December, 2024

React

The library for web and native user interfaces

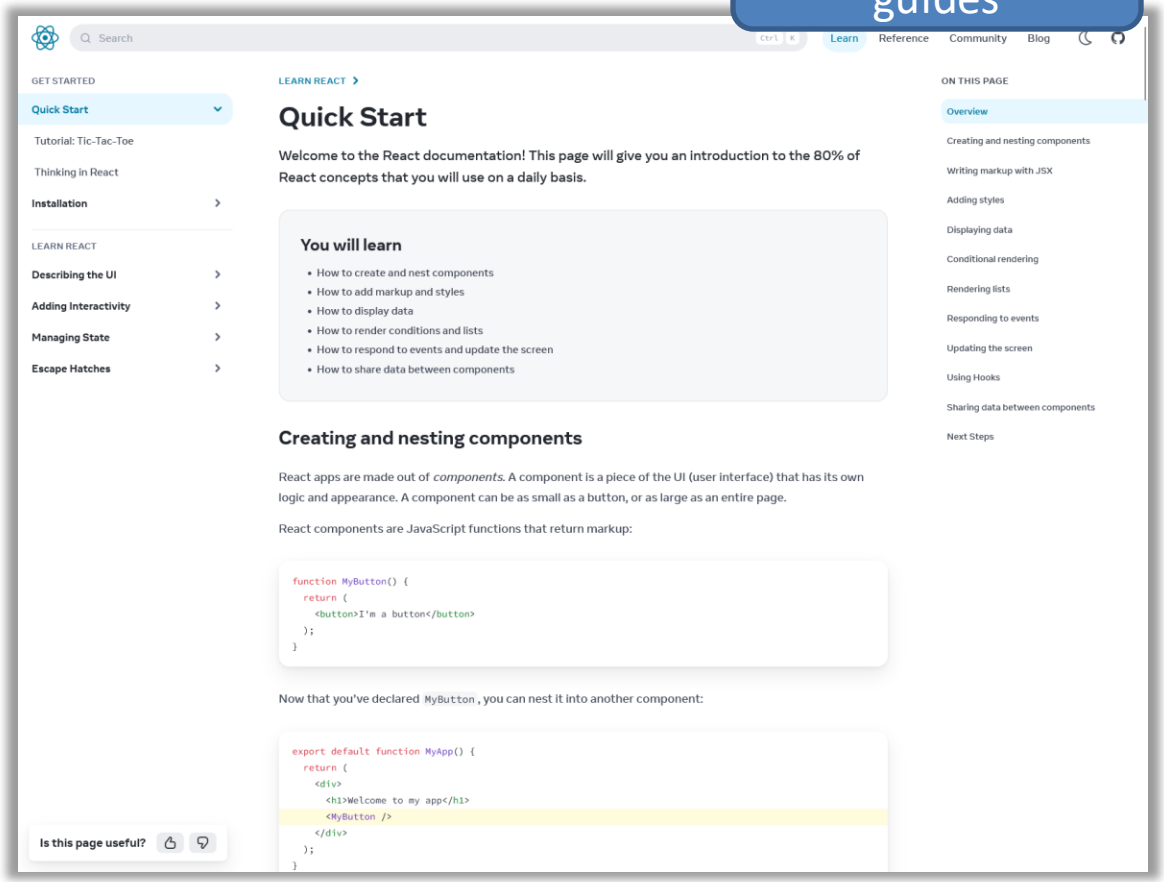
https://react.dev/
https://github.com/facebook/react

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

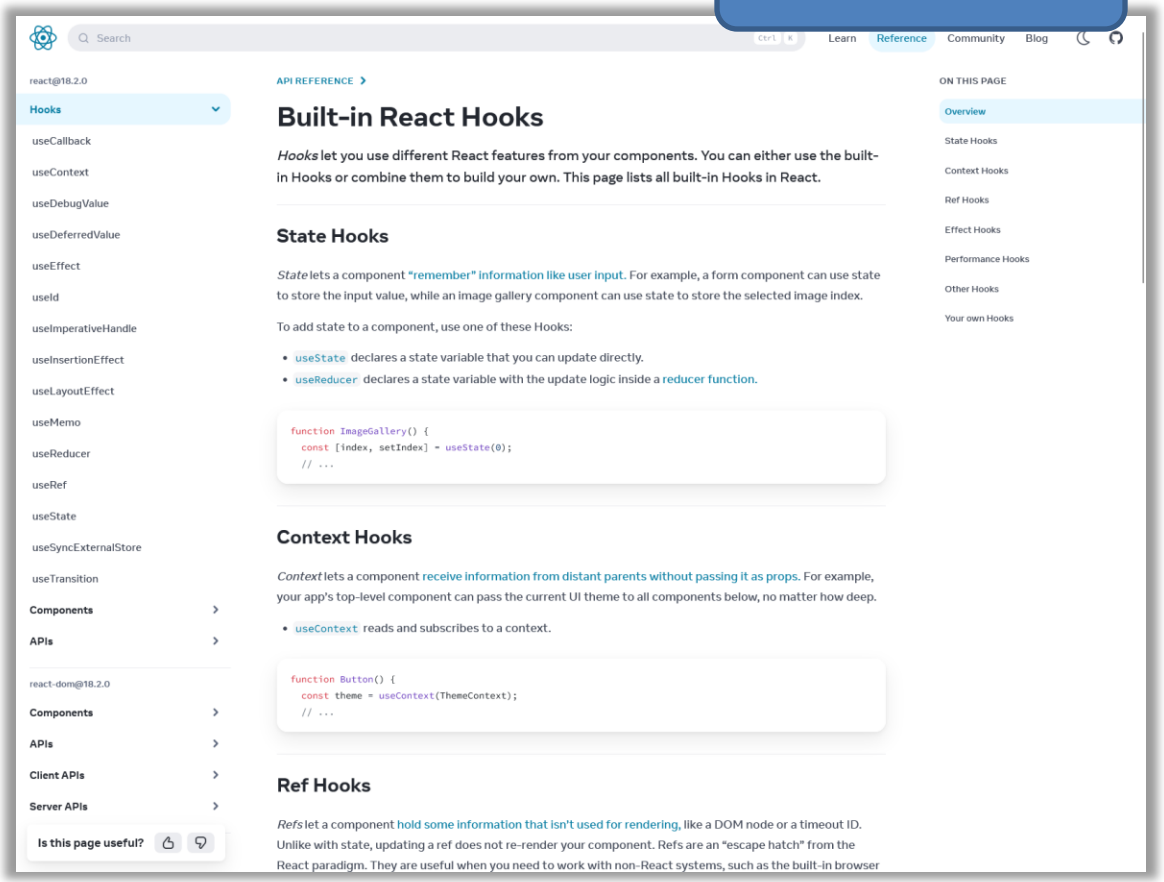
Main Resources

Tutorials and guides



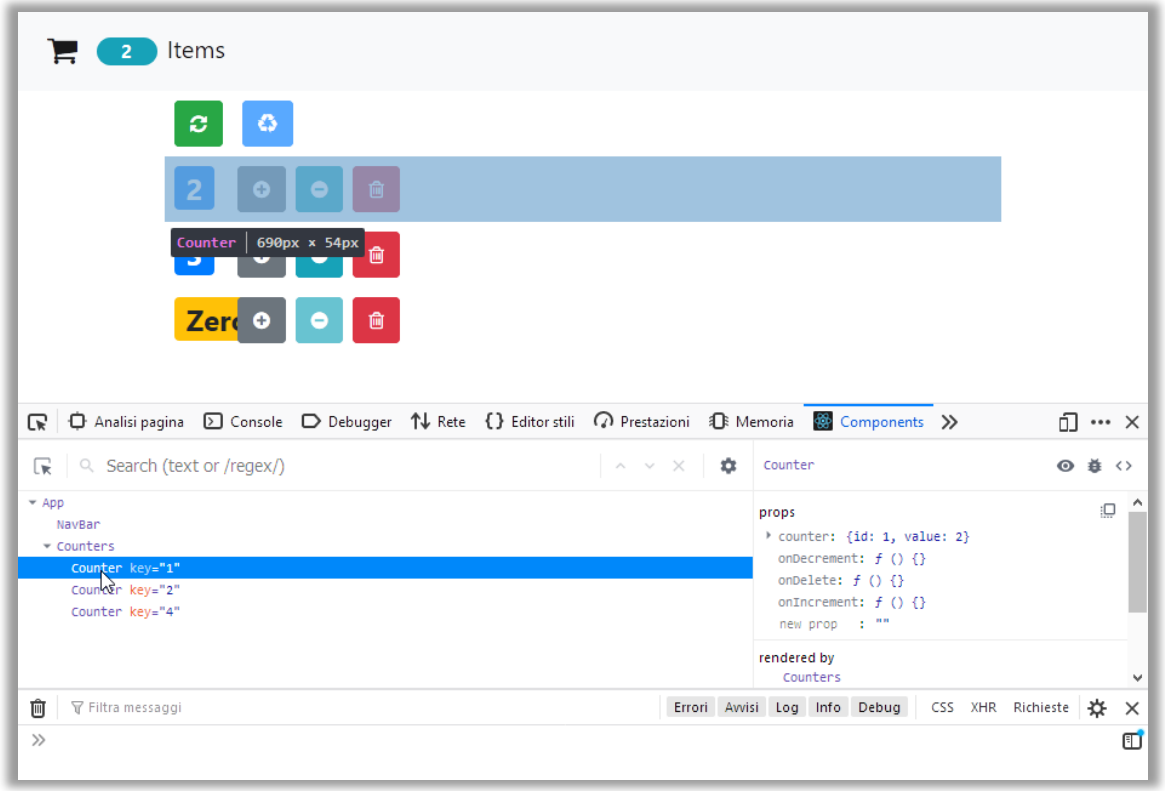
https://react.dev/learn

API Reference



https://react.dev/reference/react

Browser Development Tools



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

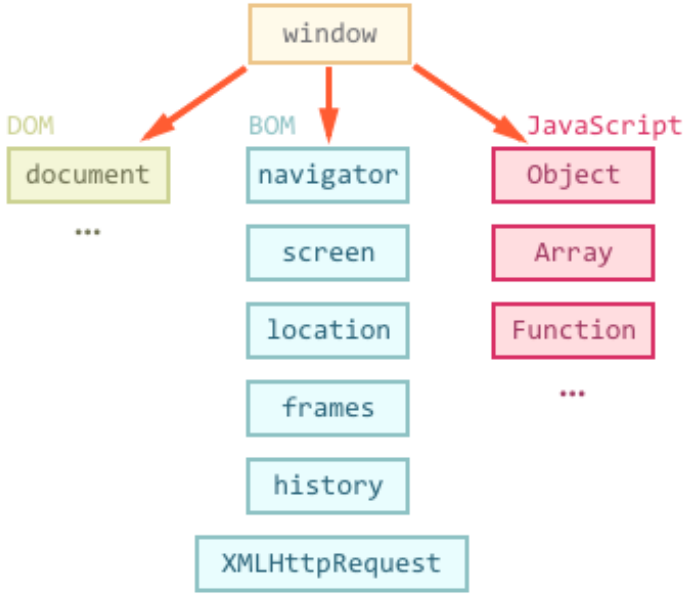
https://addons.mozilla.org/en-US/firefox/addon/react-devtools/

Before diving in…

BROWSER'S OBJECT MODELS

Browser Main Objects

- `window` represents the window that contains the Document Object Model (DOM) document
  - allows to interact with the browser via the BOM: Browser Object Model (not standardized)
  - global object, contains all JS global variables
    - can be omitted when writing JS code in the page
- `document`
  - represents the DOM tree loaded in a window
  - accessible via a `window` property: `window.document`



https://medium.com/@fknussel/dom-bom-revisited-cf6124e2a816

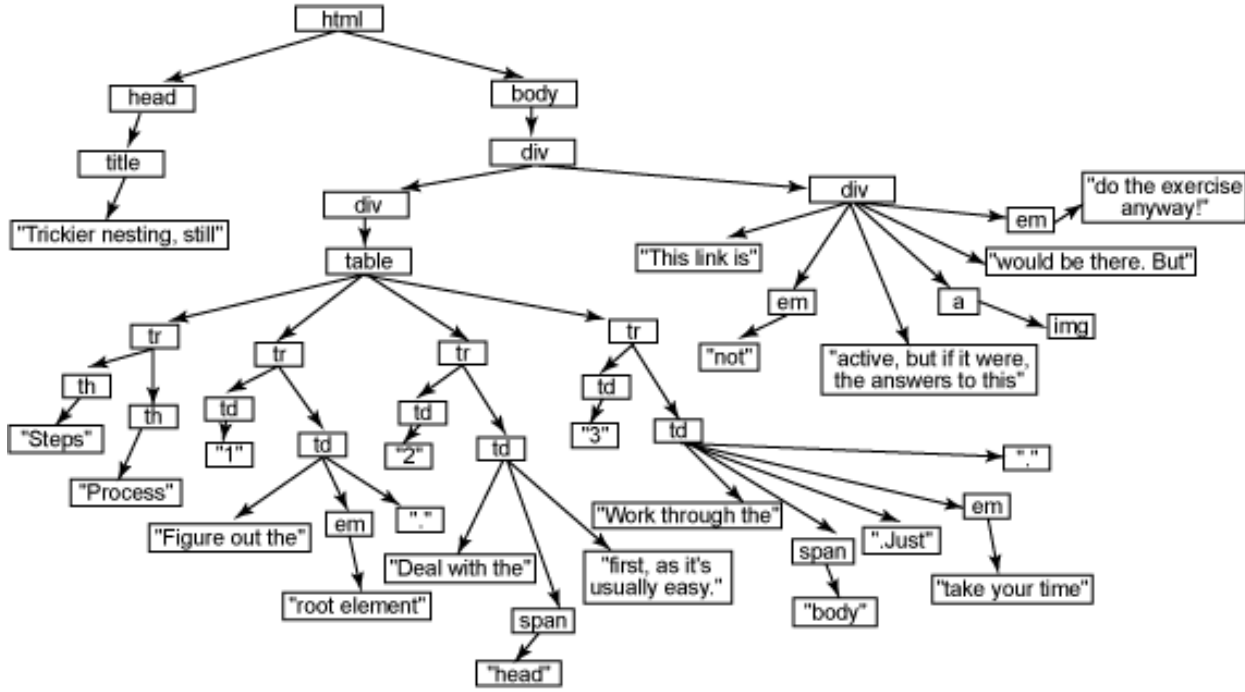
Browser Object Model

- `window` properties
  - `console`: browser debug console (visible via developer tools)
  - `document`: the document object
  - `history`: allows access to History API (history of URLs)
  - `location`: allows access to Location API (current URL, protocol, etc.). Read/write property, i.e., can be set to load a new page
  - `localStorage` and `sessionStorage`: allows access to the two objects via the Web Storage API, to store (small) info locally in the browser

https://developer.mozilla.org/en-US/docs/Web/API/Window

Document Object Model

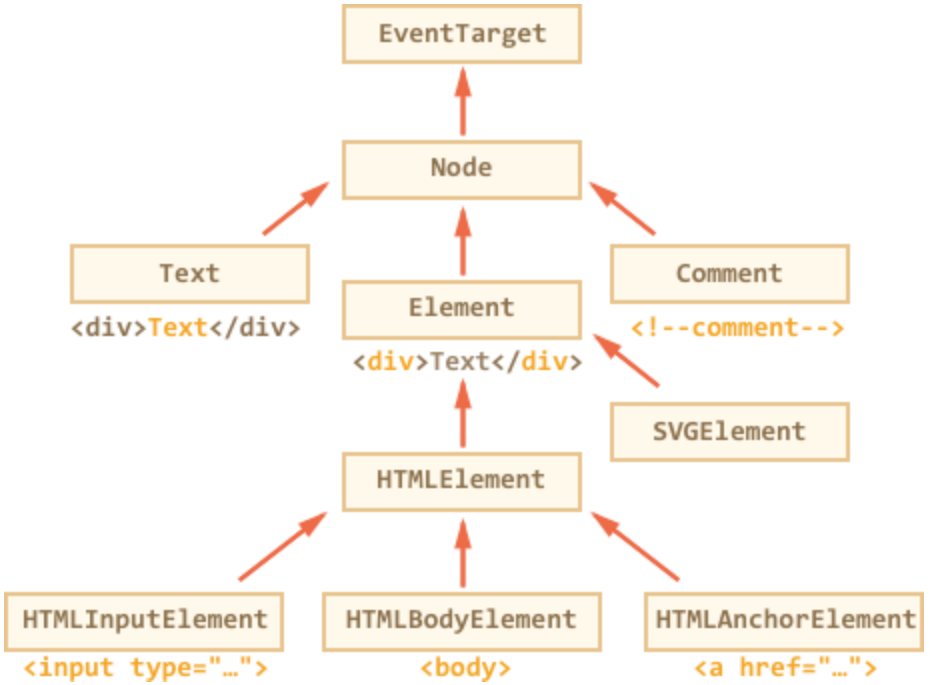
- Browser's internal representation of a web page
  - Obtained through parsing HTML
- Browsers expose an API (in JavaScript) that you can use to interact with the DOM
  - Access the page metadata and headers
  - Inspect the page structure
  - Edit any node in the page
  - Change any node attribute
  - Create/delete nodes in the page
  - Edit the CSS styling and classes
  - Attach or remove *event listeners*



https://flaviocopes.com/dom/

Types of Nodes (Classes)

- **Document**: the document Node, the root of the tree
- **Element**: an HTML tag
- **Attr**: an attribute of a tag
- **Text**: the text content of an Element or Attr Node
- **Comment**: an HTML comment
- **DocumentType**: the Doctype declaration



Event Listeners

- JavaScript in the browser uses an *event-driven* programming model
  - Everything is triggered by the firing of an event
- Events are determined by
  - The Element generating the event (event ~~source~~ target)
  - The type of generated event

https://flaviocopes.com/javascript-events/

Event Categories

- User Interface events (load, resize, scroll, etc.)
- Focus/blur events
- Mouse events (click, dblclick, mouseover, drag,
- Keyboard events (keyup, etc.)
- Form events (submit, change, input)
- Mutation events (DOMContentLoaded, etc.)
- HTML5 events (invalid, loadeddata, etc.)
- CSS events (animations etc.)

Category	Type	Attribute	Description	Bubbles	Cancelable
Mouse	click	onclick	Fires when the pointing device button is clicked over an element. A click is defined as a mousedown and mouseup over the same screen location. The sequence of these events is: ▪ mousedown ▪ mouseup ▪ click	Yes	Yes
	dblclick	ondblclick	Fires when the pointing device button is double-clicked over an element	Yes	Yes
	mousedown	onmousedown	Fires when the pointing device button is pressed over an element	Yes	Yes
	mouseup	onmouseup	Fires when the pointing device button is released over an element	Yes	Yes
	mouseover	onmouseover	Fires when the pointing device is moved onto an element	Yes	Yes
	mousemove[6]	onmousemove	Fires when the pointing device is moved while it is over an element	Yes	Yes
	mouseout	onmouseout	Fires when the pointing device is moved away from an element	Yes	Yes
	dragstart	ondragstart	Fired on an element when a drag is started.	Yes	Yes
	drag	ondrag	This event is fired at the source of the drag, that is, the element where dragstart was fired, during the drag operation.	Yes	Yes
	dragenter	ondragenter	Fired when the mouse is first moved over an element while a drag is occurring.	Yes	Yes
	dragleave	ondragleave	This event is fired when the mouse leaves an element while a drag is occurring.	Yes	No
	dragover	ondragover	This event is fired as the mouse is moved over an element when a drag is occurring.	Yes	Yes
	drop	ondrop	The drop event is fired on the element where the drop occurs at the end of the drag operation.	Yes	Yes
	dragend	ondragend	The source of the drag will receive a dragend event when the drag operation is complete, whether it was successful or not.	Yes	No
Keyboard	keydown	onkeydown	Fires before keypress, when a key on the keyboard is pressed.	Yes	Yes
	keypress	onkeypress	Fires after keydown, when a key on the keyboard is pressed.	Yes	Yes
	keyup	onkeyup	Fires when a key on the keyboard is released	Yes	Yes
HTML frame/object	load	onload	Fires when the user agent finishes loading all content within a document, including window, frames, objects and images For elements, it fires when the target element and all of its content has finished loading	No	No
	unload	onunload	Fires when the user agent removes all content from a window or frame For elements, it fires when the target element or any of its content has been removed	No	No
	abort	onabort	Fires when an object/image is stopped from loading before completely loaded	Yes	No
	error	onerror	Fires when an object/image/frame cannot be loaded properly	Yes	No
	resize	onresize	Fires when a document view is resized	Yes	No
	scroll	onscroll	Fires when an element or document view is scrolled	No, except that a scroll event on document must bubble to the window[7]	No
HTML form	select	onselect	Fires when a user selects some text in a text field, including input and textarea	Yes	No
	change	onchange	Fires when a control loses the input focus and its value has been modified since gaining focus	Yes	No
	submit	onsubmit	Fires when a form is submitted	Yes	Yes
	reset	onreset	Fires when a form is reset	Yes	No
	focus	onfocus	Fires when an element receives focus either via the pointing device or by tab navigation	No	No
	blur	onblur	Fires when an element loses focus either via the pointing device or by tabbing navigation	No	No
User interface	focusin	(none)	Similar to HTML focus event, but can be applied to any focusable element	Yes	No
	focusout	(none)	Similar to HTML blur event, but can be applied to any focusable element	Yes	No
	DOMActivate	(none)	Similar to XUL command event. Fires when an element is activated, for instance, through a mouse click or a keypress.	Yes	Yes
Mutation	DOMSubtreeModified	(none)	Fires when the subtree is modified	Yes	No
	DOMNodeInserted	(none)	Fires when a node has been added as a child of another node	Yes	No
	DOMNodeRemoved	(none)	Fires when a node has been removed from a DOM-tree	Yes	No
	DOMNodeRemovedFromDocument	(none)	Fires when a node is being removed from a document	No	No
	DOMNodeInsertedIntoDocument	(none)	Fires when a node is being inserted into a document	No	No
	DOMAttrModified	(none)	Fires when an attribute has been modified	Yes	No
	DOMCharacterDataModified	(none)	Fires when the character data has been modified	Yes	No
Progress	loadstart	(none)	Progress has begun.	No	No
	progress	(none)	In progress. After loadstart has been dispatched.	No	No
	error	(none)	Progression failed. After the last progress has been dispatched, or after loadstart has been dispatched if progress has not been dispatched.	No	No
	abort	(none)	Progression is terminated. After the last progress has been dispatched, or after loadstart has been dispatched if progress has not been dispatched.	No	No
	load	(none)	Progression is successful. After the last progress has been dispatched, or after loadstart has been dispatched if progress has not been dispatched.	No	No
	loadend	(none)	Progress has stopped. After one of error, abort, or load has been dispatched.	No	No

https://en.wikipedia.org/wiki/DOM_events

Preventing Default Behavior

- Many events cause a default behavior
  - Click on link: go to URL
  - Click on submit button: form is sent
- Can be prevented by `event.preventDefault()`

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 do not 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

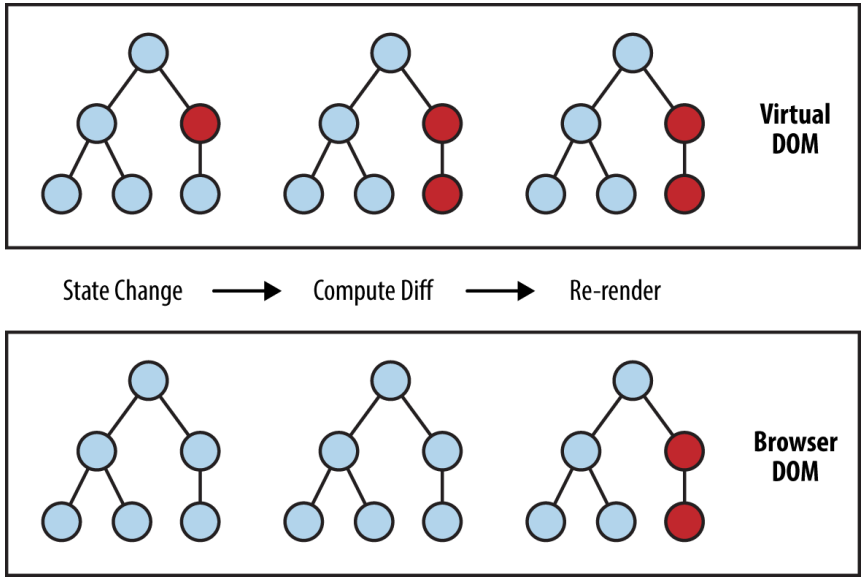
- The application is made of Components
- The entire application is re-rendered:
  - Every time a state is changed
  - Every time a property is changed
- Each Component will re-build itself from scratch
  - With minor variations, or
  - Radically different
- Performance?

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

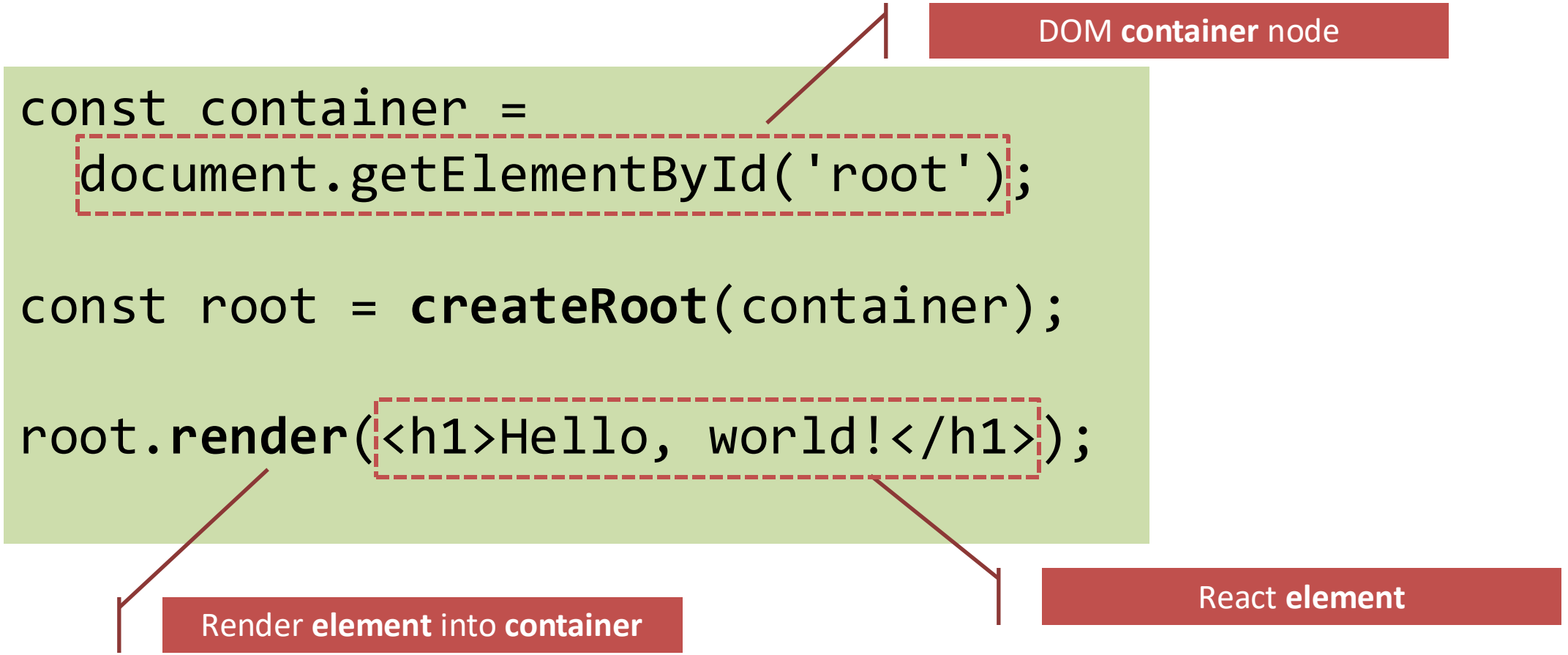
http://slides.com/johnlynch/reactjs

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



JSX Syntax

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

root.render(
    <div id="test">
      <h1>A title</h1>
      <p>A paragraph</p>
    </div>
);
```

JSX Syntax

Equivalent

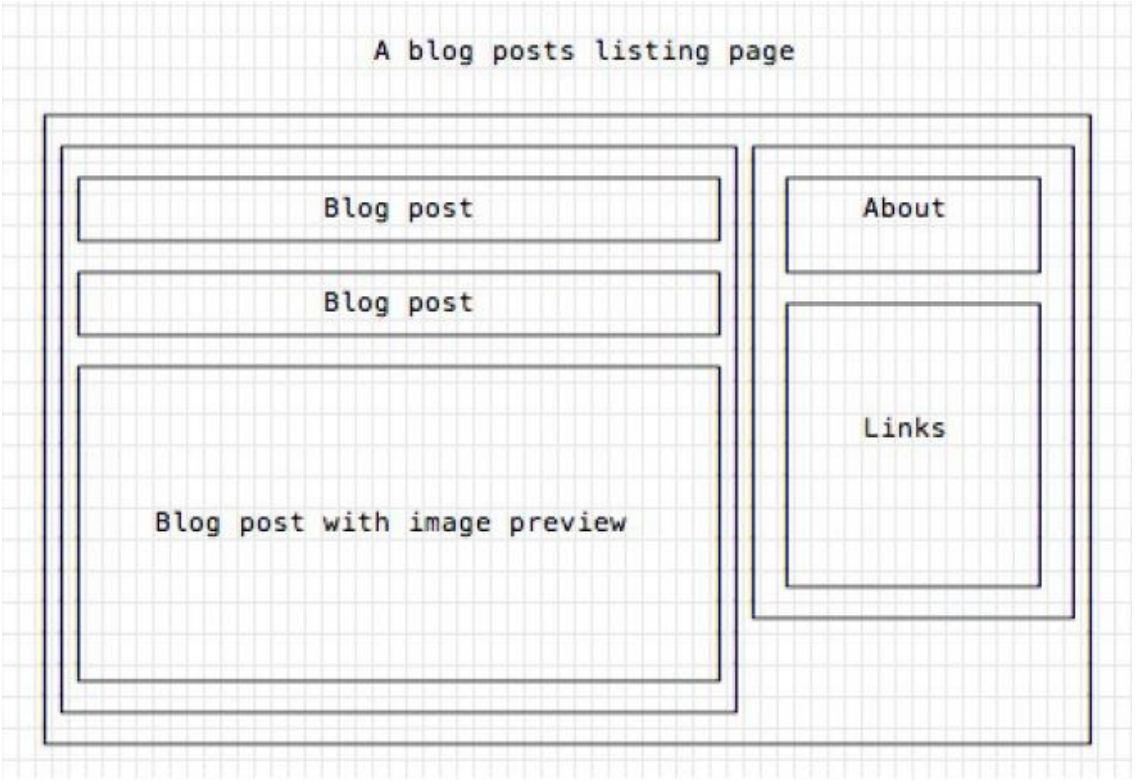
Transpiling (Babel)

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

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

JS calls to React.createElement

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

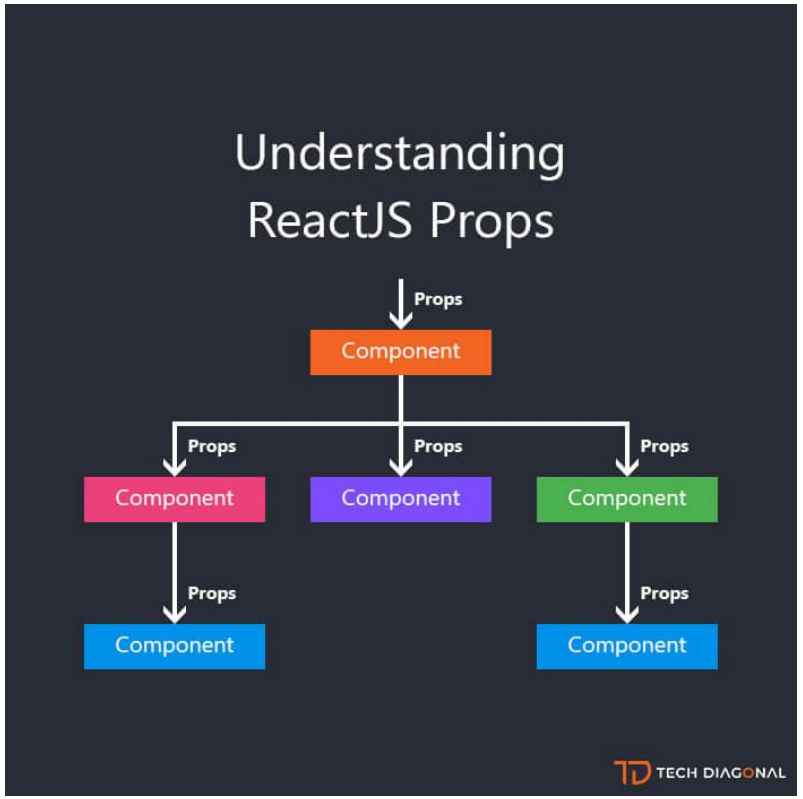
- Generate DOM nodes to be displayed
- Do not manage application state
- Might have some internal state, uniquely for **presentation** purposes

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

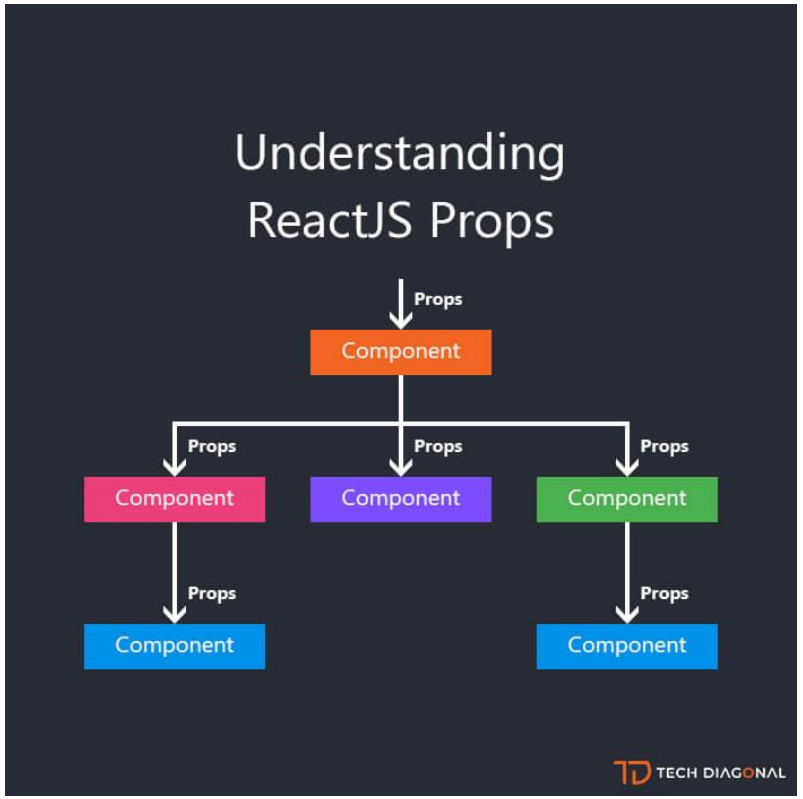
- **Props** (properties) are passed to a component by its parent
  - Values (strings, objects, …) to configure how the component displays or behaves
    - Top-to-bottom data flow
  - Functions (callbacks) to access the parent's methods
    - Bottom-to-top action requests



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

Props and State

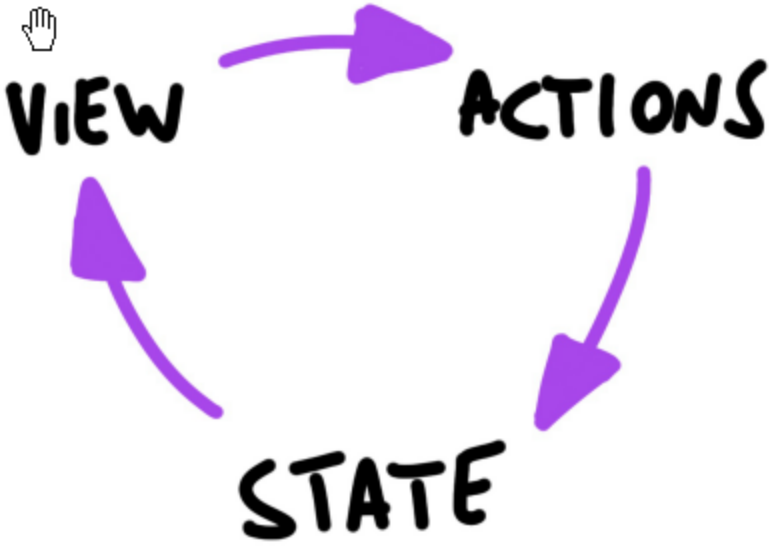
- **State** is a set of variables local to the component
  - Initialized with default value or by props' values
  - Can be mutated only by calling specific methods
    - Asynchronous
    - Will initiate re-rendering of the Virtual DOM
  - Current state value can be passed to children (as props)



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

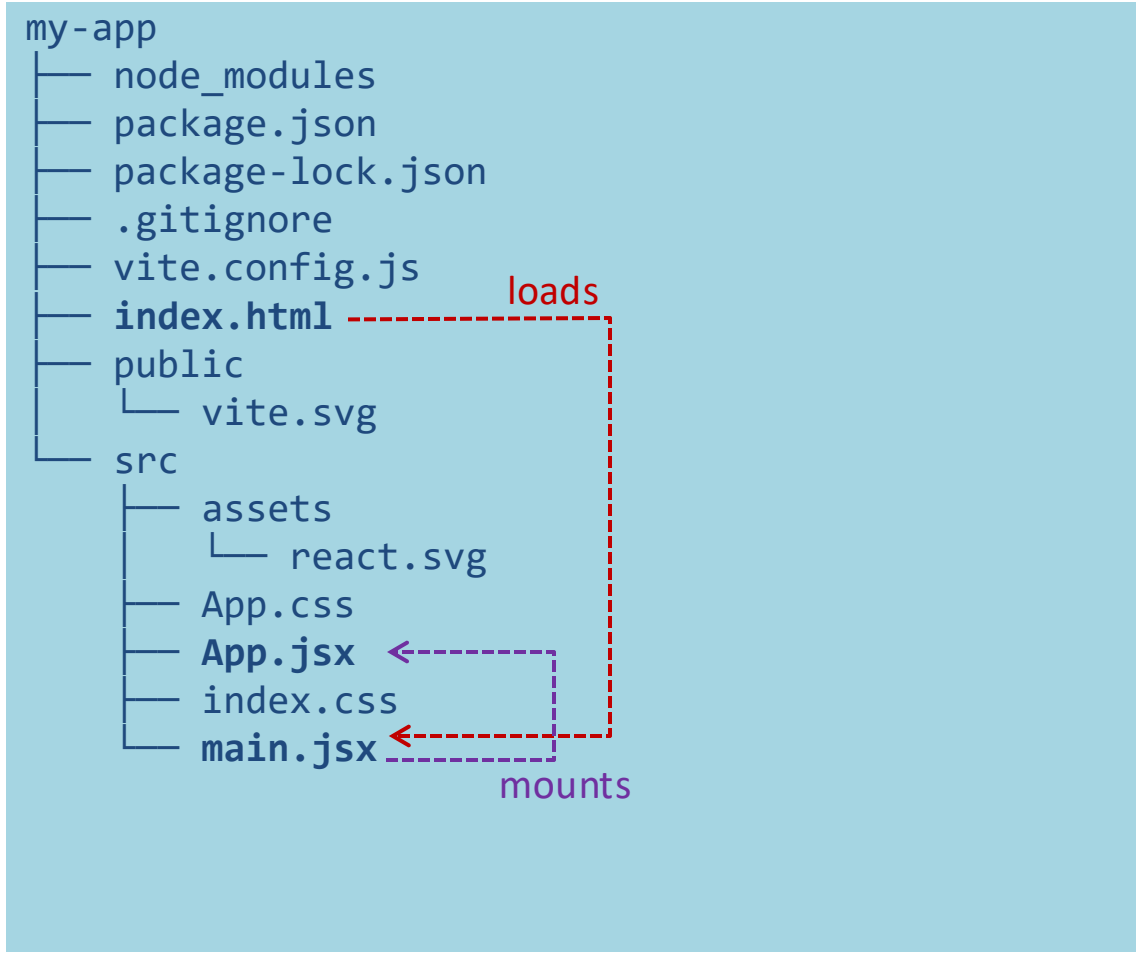
- Import the React library
  - Import several needed libraries
- We want to use **JSX**
  - Babel required
- We need to run on a web server
  - To be able to use modules
    - `import` in JS code
    - `<script type='module'>` in HTML code
  - Avoid problems with CORS
- Implement polyfills for browser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

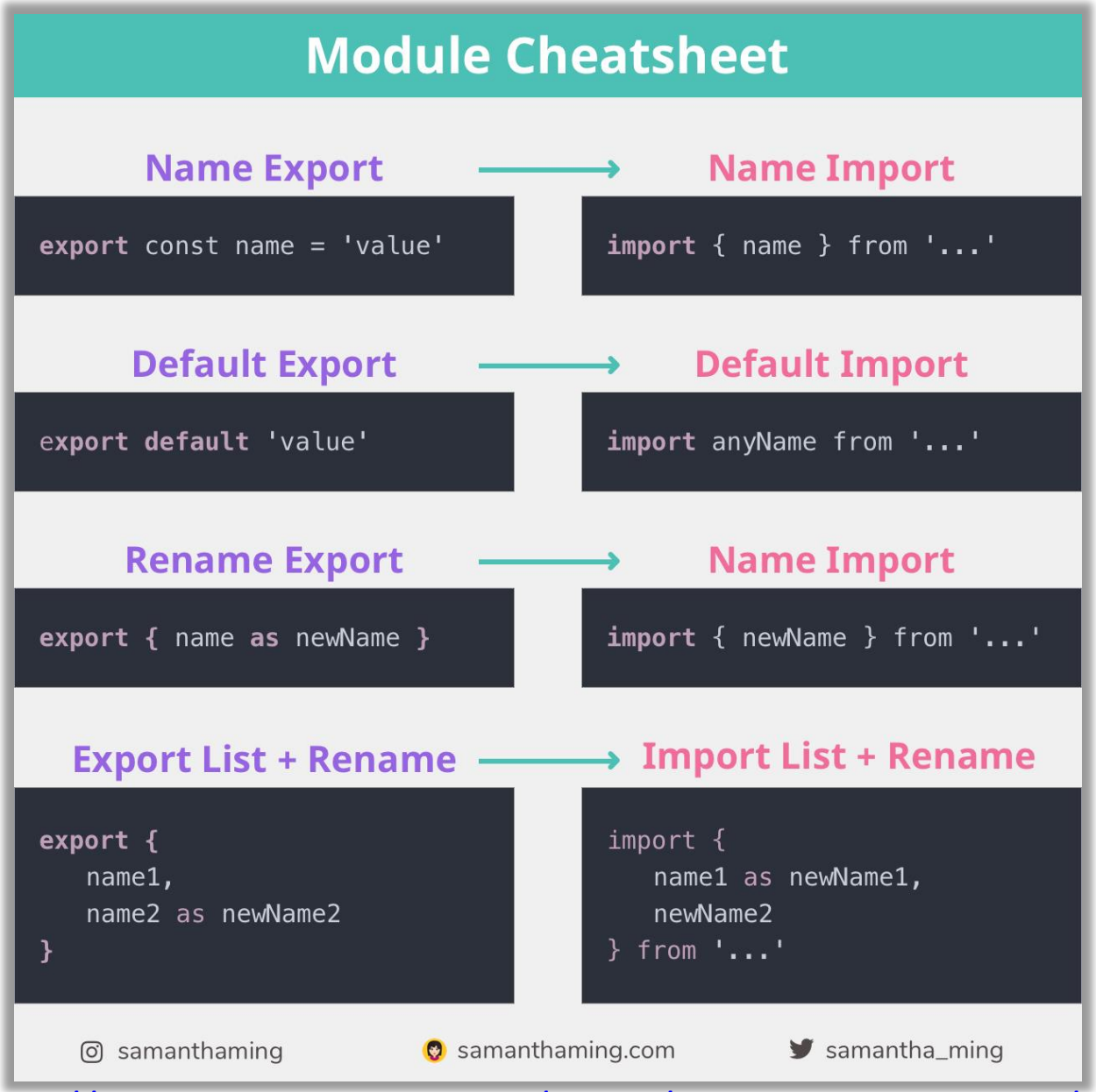
Folder Structure



- `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
  - The `require` function used in Node.js doesn’t work here



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

Example: Hello world

App.jsx

```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

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

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

export default App;
```

Button.jsx

```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

```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;
```

Example: adding Bootstrap

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

- 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

```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

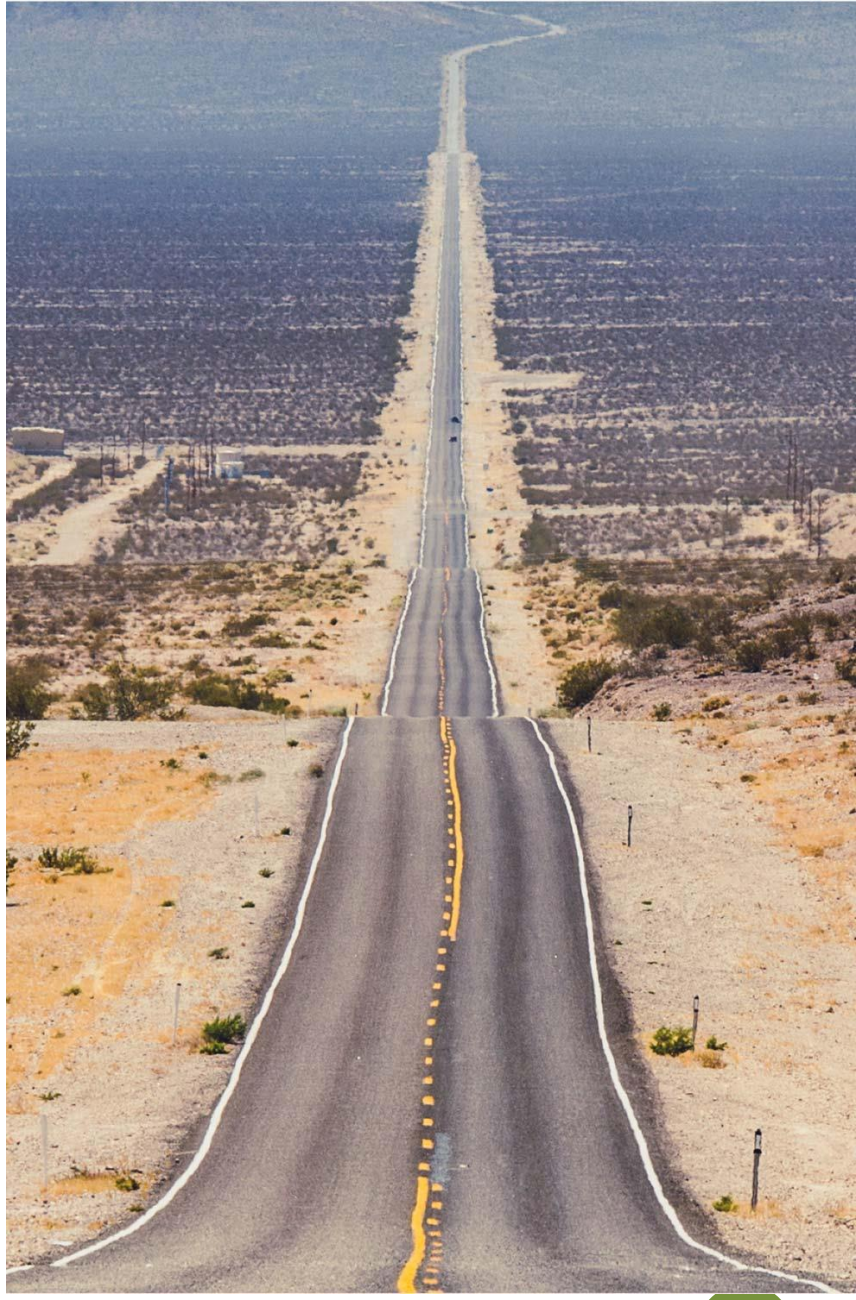
https://react-bootstrap.github.io/

Button.jsx

```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/