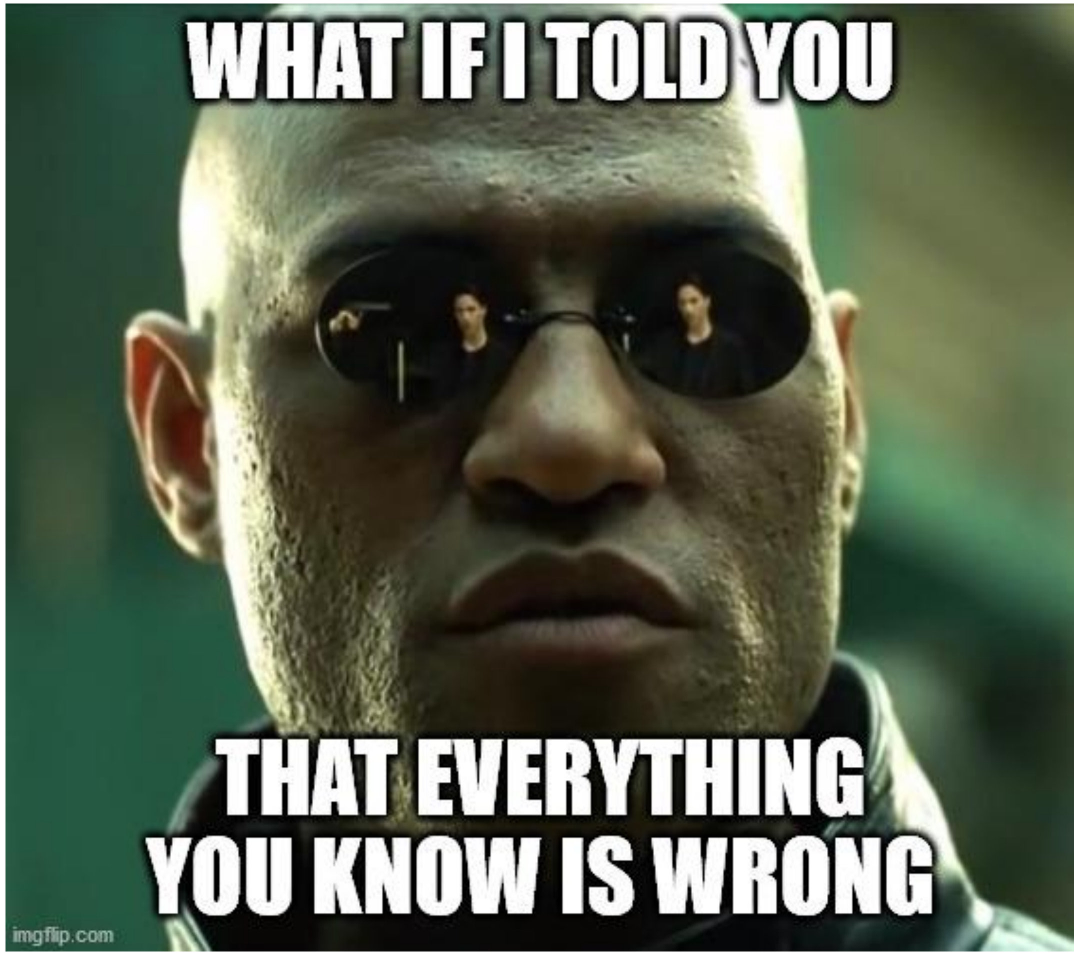
<WA1/>
<AW1/>
2021

Introduction to React

JS Frameworks to the rescue

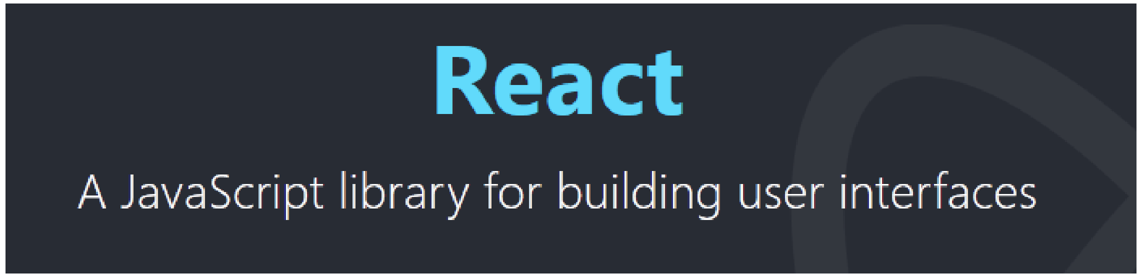
Fulvio Corno
Luigi De Russis
Enrico Masala



Goal

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

React


React

A JavaScript library for building user interfaces

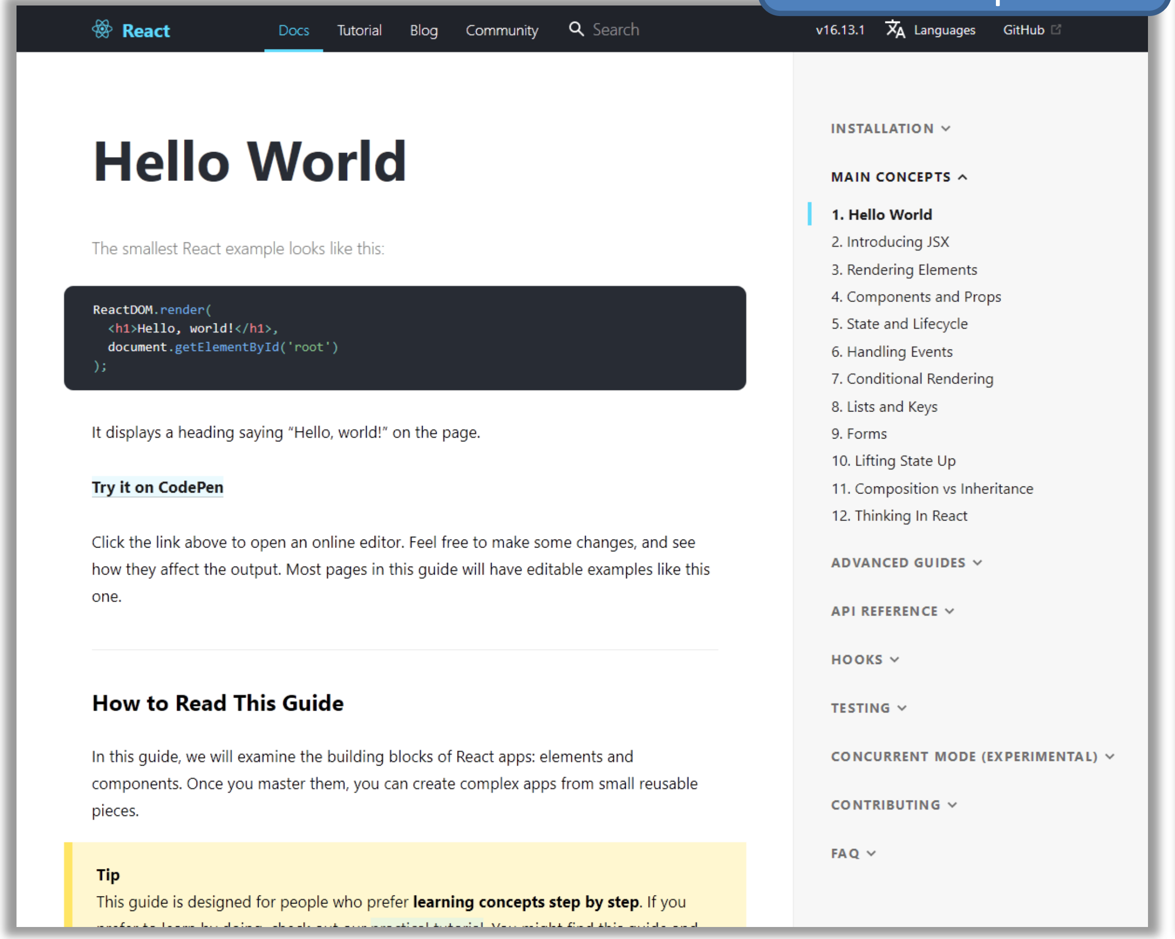
https://reactjs.org/

Why a Framework?

- 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

Learning the main concepts


React Docs Tutorial Blog Community Search v16.13.1 Languages GitHub

Hello World

The smallest React example looks like this:

```
ReactDOM.render(
  <h1>Hello, world!</h1>,
  document.getElementById('root')
);
```

It displays a heading saying "Hello, world!" on the page.

Try it on CodePen

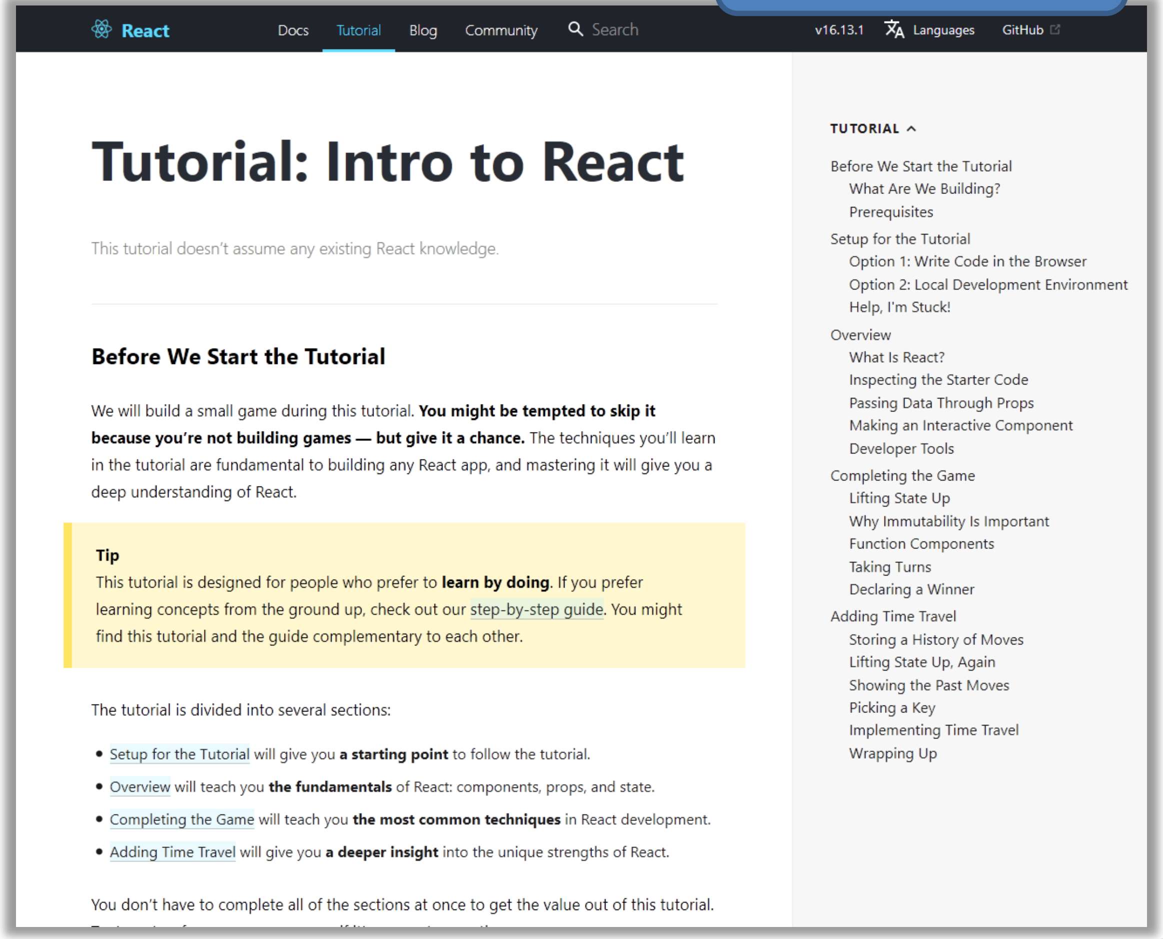
Click the link above to open an online editor. Feel free to make some changes, and see how they affect the output. Most pages in this guide will have editable examples like this one.

How to Read This Guide

In this guide, we will examine the building blocks of React apps: elements and components. Once you master them, you can create complex apps from small reusable pieces.

Tip
This guide is designed for people who prefer **learning concepts step by step**. If you

Learn by doing tutorial


React Docs Tutorial Blog Community Search v16.13.1 Languages GitHub

Tutorial: Intro to React

This tutorial doesn't assume any existing React knowledge.

Before We Start the Tutorial

We will build a small game during this tutorial. **You might be tempted to skip it because you're not building games — but give it a chance.** The techniques you'll learn in the tutorial are fundamental to building any React app, and mastering it will give you a deep understanding of React.

Tip
This tutorial is designed for people who prefer to **learn by doing**. If you prefer learning concepts from the ground up, check out our step-by-step guide. You might find this tutorial and the guide complementary to each other.

The tutorial is divided into several sections:

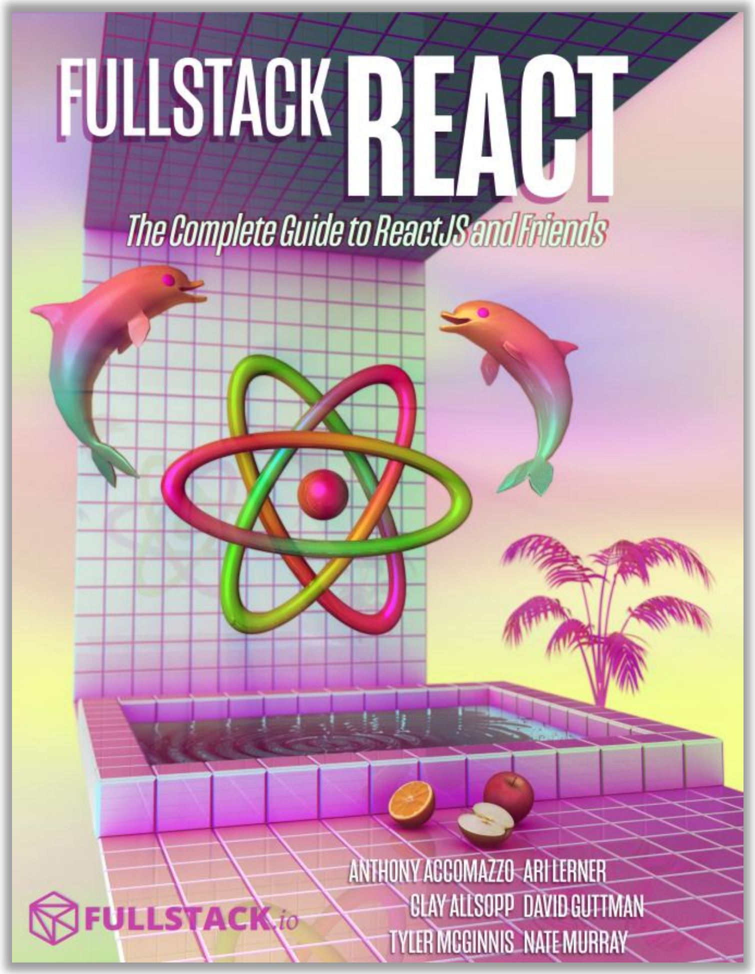
- Setup for the Tutorial will give you **a starting point** to follow the tutorial.
- Overview will teach you **the fundamentals** of React: components, props, and state.
- Completing the Game will teach you **the most common techniques** in React development.
- Adding Time Travel will give you **a deeper insight** into the unique strengths of React.

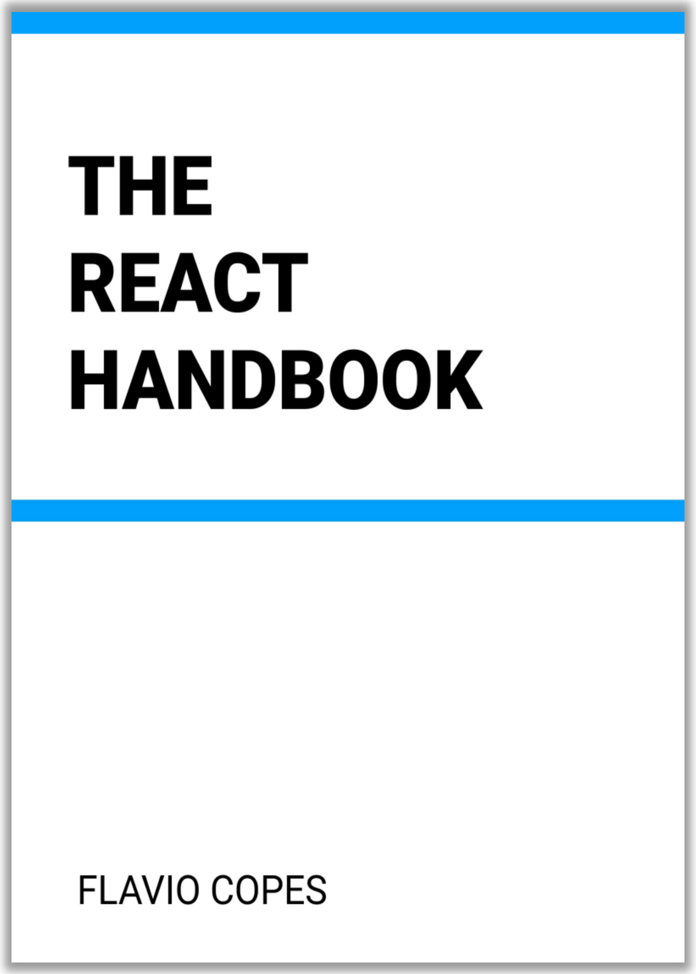
You don't have to complete all of the sections at once to get the value out of this tutorial.

https://reactjs.org/docs/hello-world.html

https://reactjs.org/tutorial/tutorial.html

Main Resources

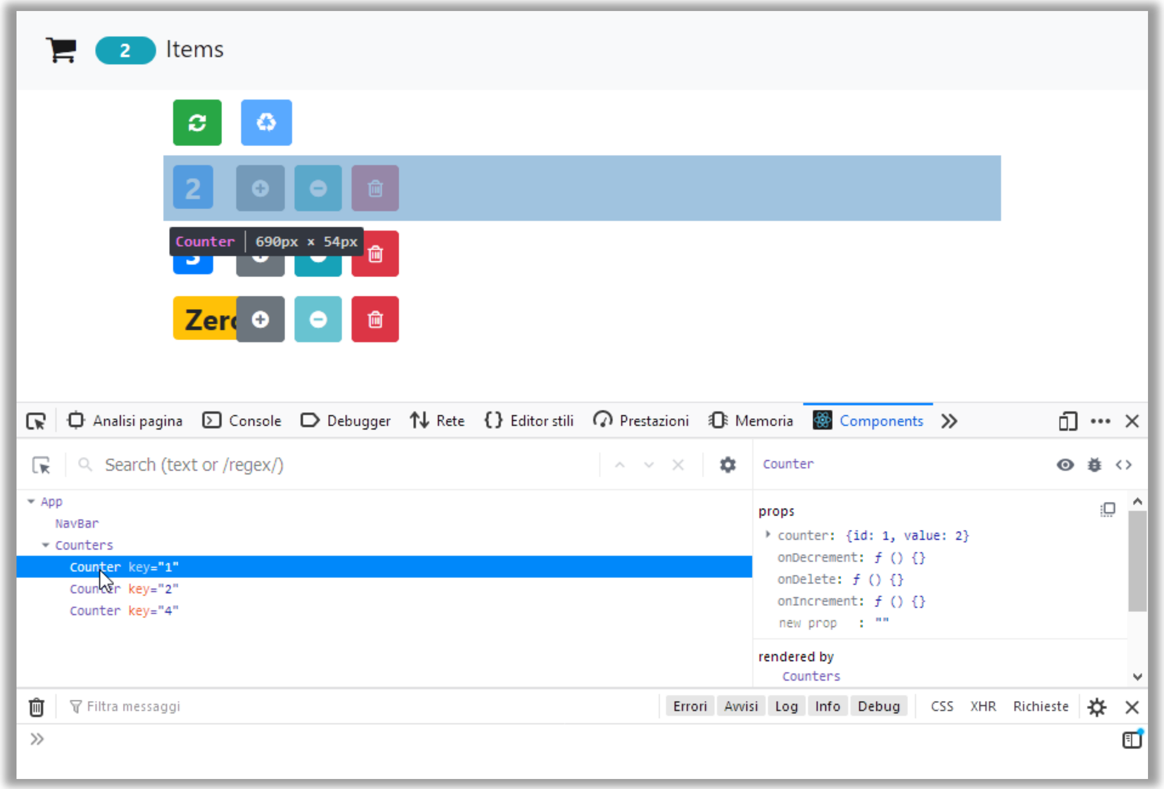

FULLSTACK REACT
The Complete Guide to ReactJS and Friends
ANTHONY ACCOMAZZO ARI LERNER
CLAY ALLSOPP DAVID GUTTMAN
TYLER MCGINNIS NATE MURRAY
FULLSTACK.io



https://www.newline.co/fullstack-react/

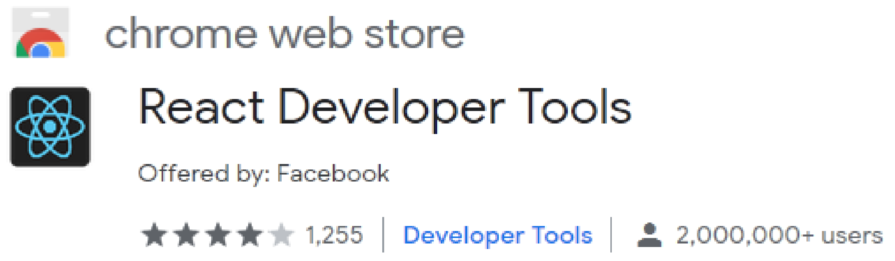
https://flaviocopes.com/page/react-handbook/

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
```



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

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 is Declarative

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

React Key Concepts

- Functional design approach
- Components
- Re-render everything on every change
- Virtual DOM
- Synthetic Events
- Controls 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

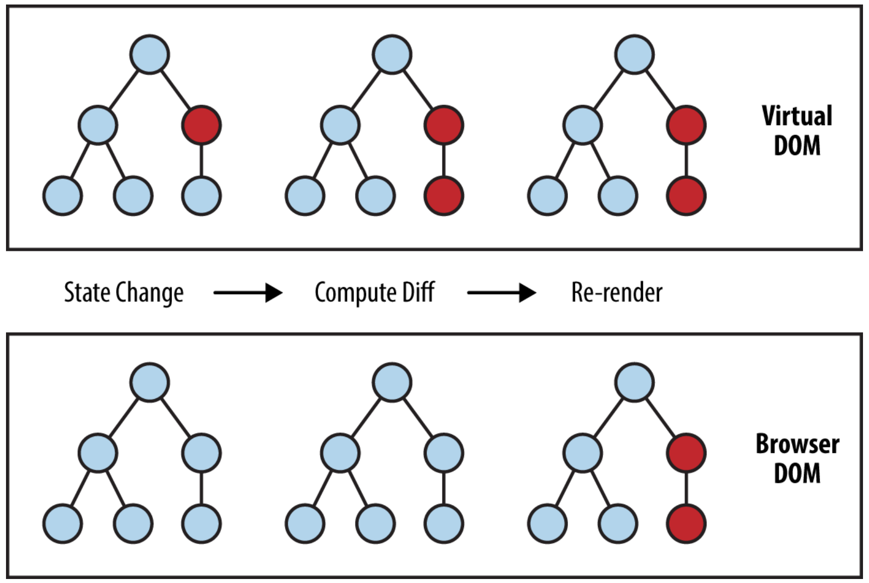
- The application is made of Components
- The entire application is re-rendered
  - Every time a state is changed
  - Every time a property is changed
- Each Component will re-build itself from scratch
  - With minor variations, or
  - Radically different
- Performance?

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 Looks Like

```
ReactDOM.render(

  <h1>Hello, world!</h1> ,

  document.getElementById('root')

);
```

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

React **element**

DOM **container** node

JSX Syntax

JSX Syntax

```
ReactDOM.render(

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

    document.getElementById('myapp')

);
```

Equivalent

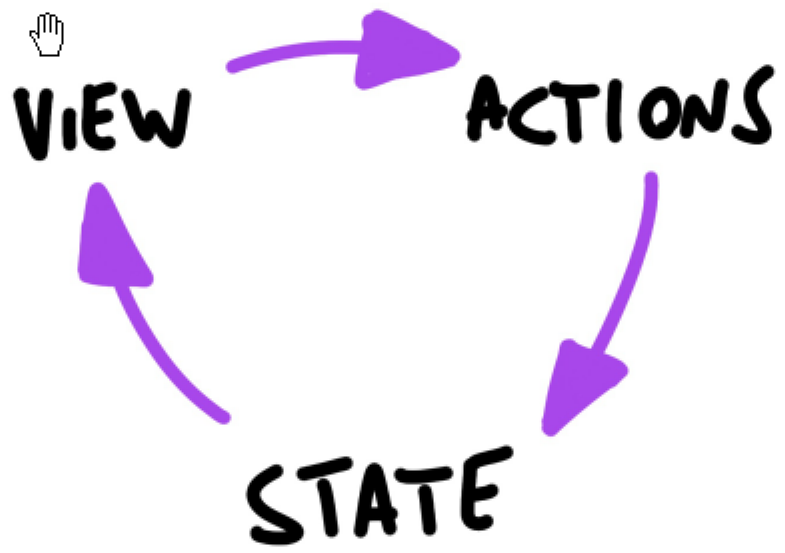
Transpiling (Babel)

JS calls to React.createElement

```
ReactDOM.render(

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

    document.getElementById('myapp')

);
```

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

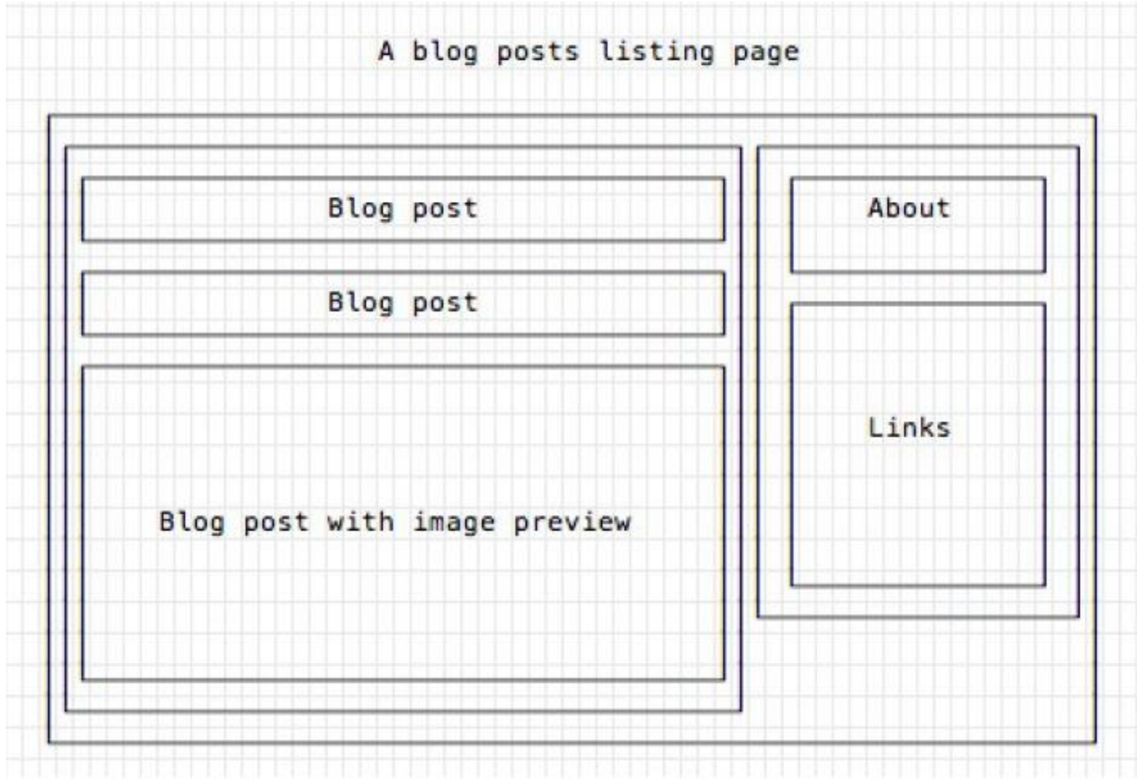


Corollary

- 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.

Components

- Everything on a page is a Component
  - Even simple HTML tags (React.DOM.element)
- Components may be **nested**
- ReactDOM.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>
  )
}
```

As a class, with a render() method

```
import React, { Component } from 'react'

class BlogPostExcerpt extends Component {
  render() {
    return (
      <div>
        <h1>Title</h1>
        <p>Description</p>
      </div>
    )
  }
}
```

Types of Components

Presentational Components

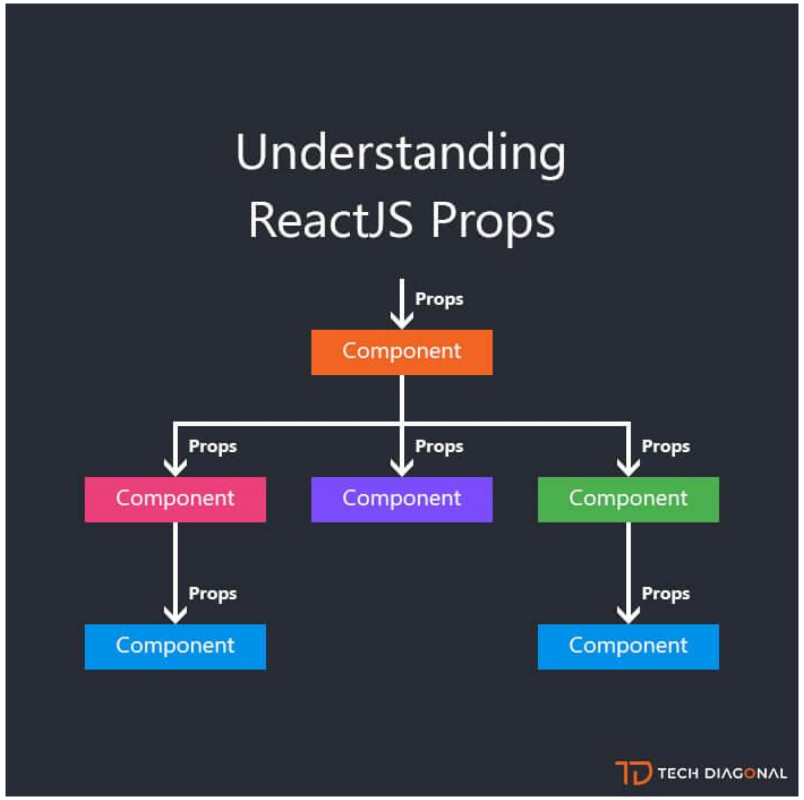
- Generate DOM nodes to be displayed
- Do not manage application state
- Might have some internal state, uniquely for presentation purposes

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

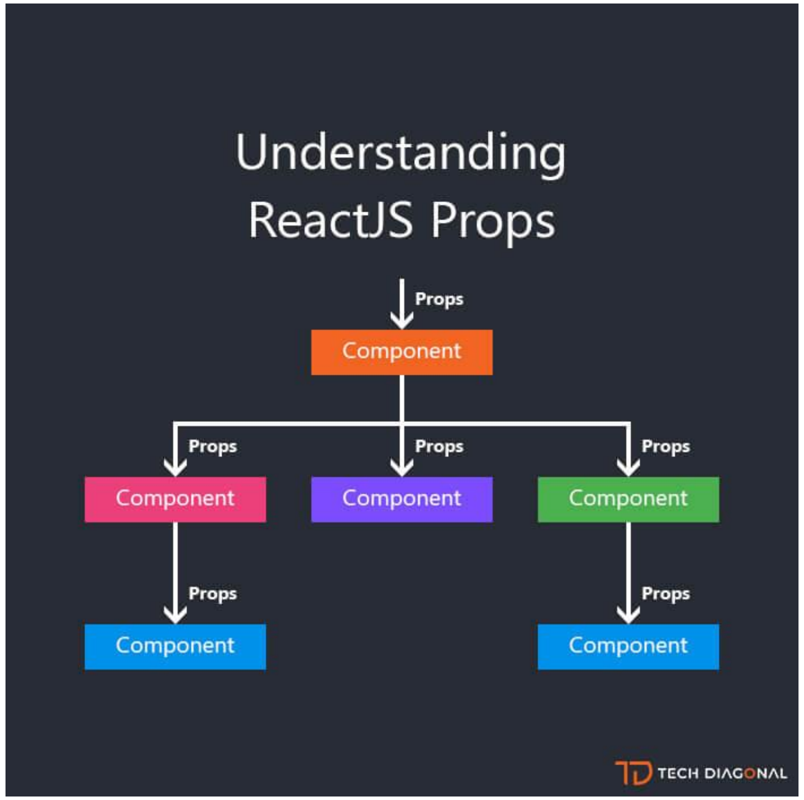
- **Props** (properties) are passed to a component by its parent
  - Values (strings, objects, …) to configure how the component displays or behaves
    - Top-to-bottom data flow
  - Functions (callbacks) to access the parent's methods
    - Bottom-to-top action requests



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/

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

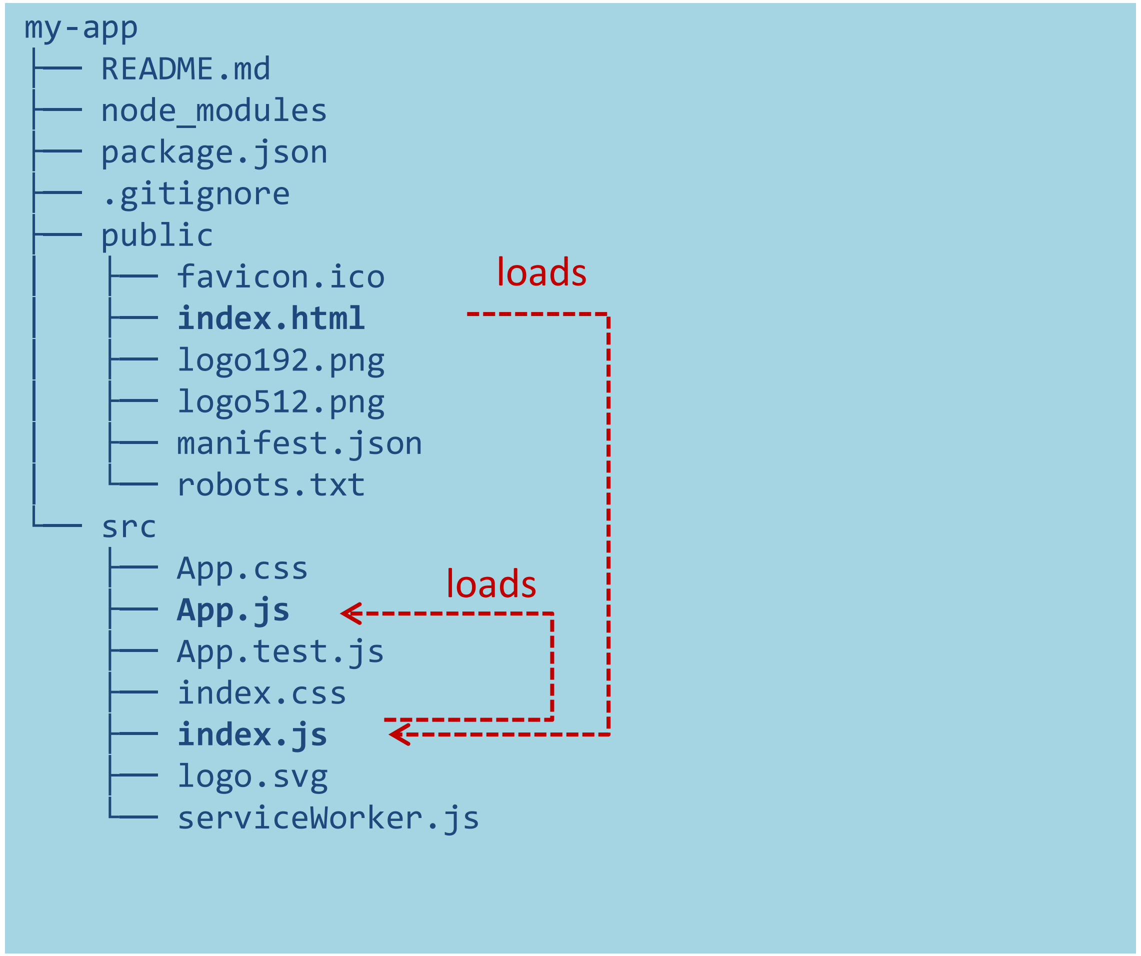
1. `npx create-react-app` *`my-app`*
2. ⌛ *… 270 Megabytes later …* ⌛
3. `cd` *`my-app`*
4. `npm start`
5. Visit http://localhost:3000

npx downloads the npm module and runs it immediately.

It will save the downloaded package in a local cache, but outside the current project.

https://create-react-app.dev/

Folder Structure



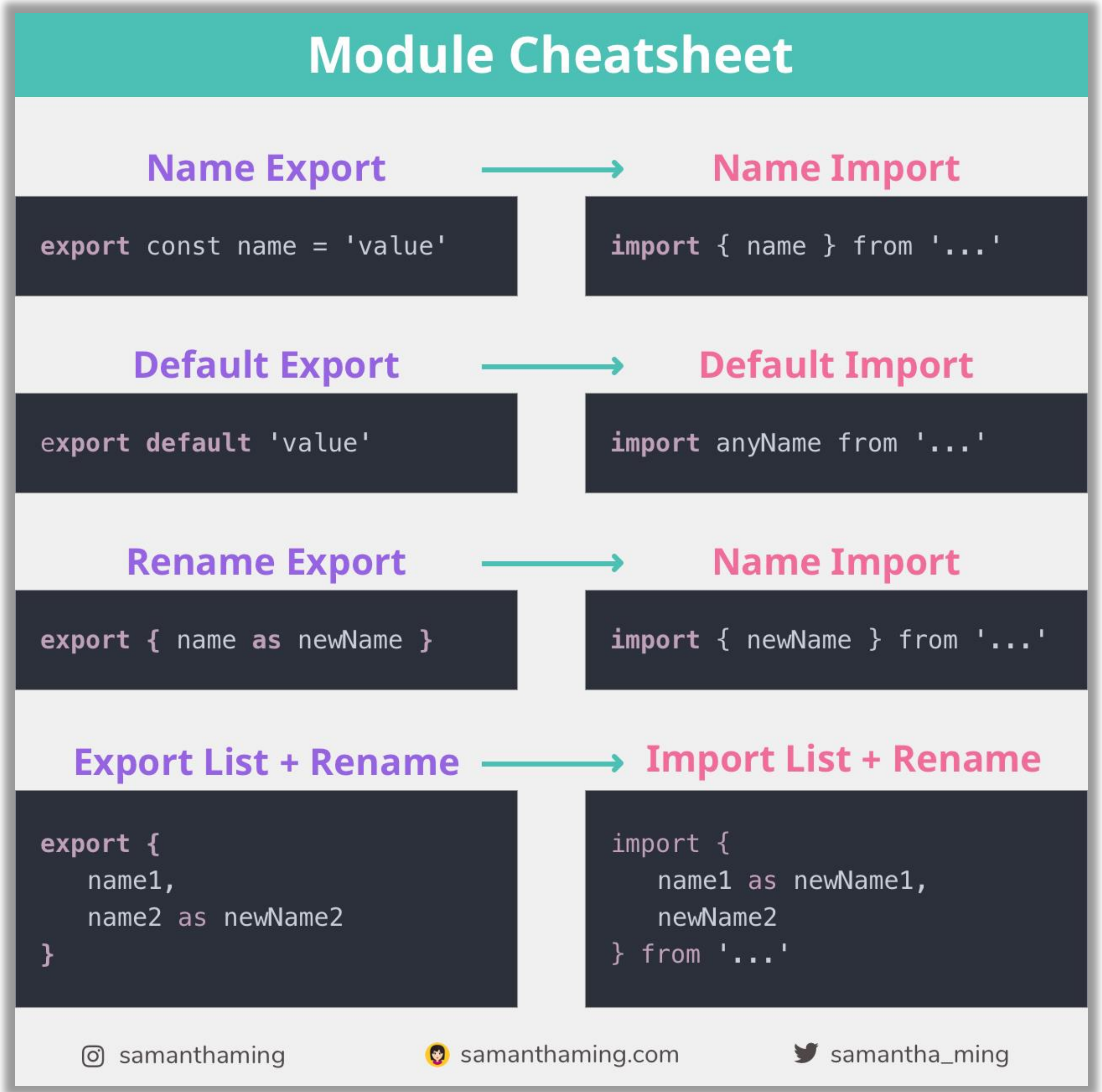
```
my-app
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
│   ├── favicon.ico
│   ├── index.html
│   ├── logo192.png
│   ├── logo512.png
│   ├── manifest.json
│   └── robots.txt
└── src
    ├── App.css
    ├── App.js
    ├── App.test.js
    ├── index.css
    ├── index.js
    ├── logo.svg
    └── serviceWorker.js
```

- `public` is the web server root
  - Static files go here
- `public/index.html` is the page template
  - Published at http://localhost:3000
  - Automatically reloads when app changes
  - No need to modify, normally
- `src` contains all scripts
- `src/index.js` is the JavaScript entry point
  - Contains the `ReactDOM.render` call to mount the App in the `#root` element
  - Do not touch, normally
- `src/App.js` 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.js

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

function App() {
  return (
    <p>
      Premi qui: <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.js

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

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

export default App;
```

Button.js

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

export default Button;
```

Example: Dynamic State

Button.js

```
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.js

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

import MyButton from './Button.js';

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

export default App;
```

Example: adding Bootstrap

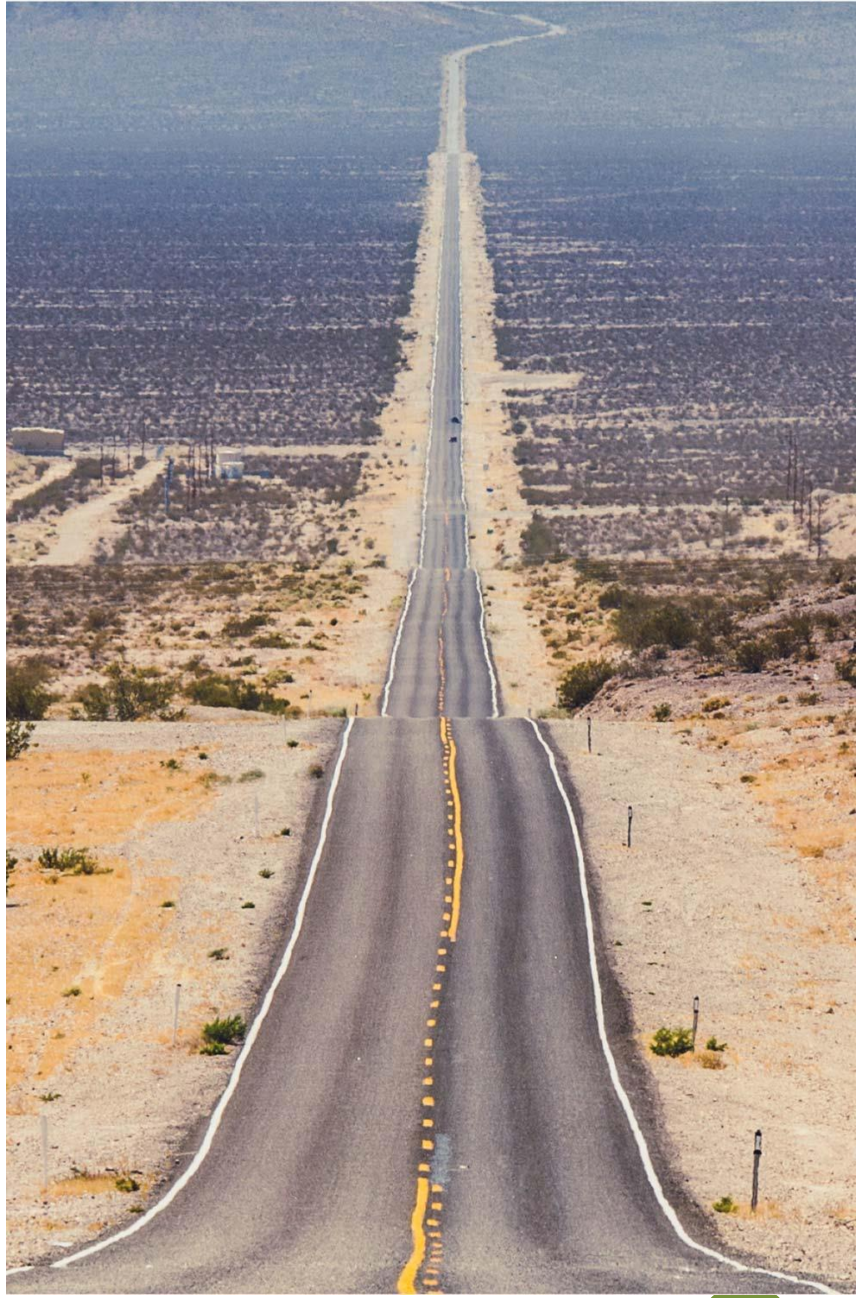
https://react-bootstrap.github.io/

Button.js

```
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