React

Components-Based UI

Outline

- Introduction
- React Components
- Components Communication
- Routing

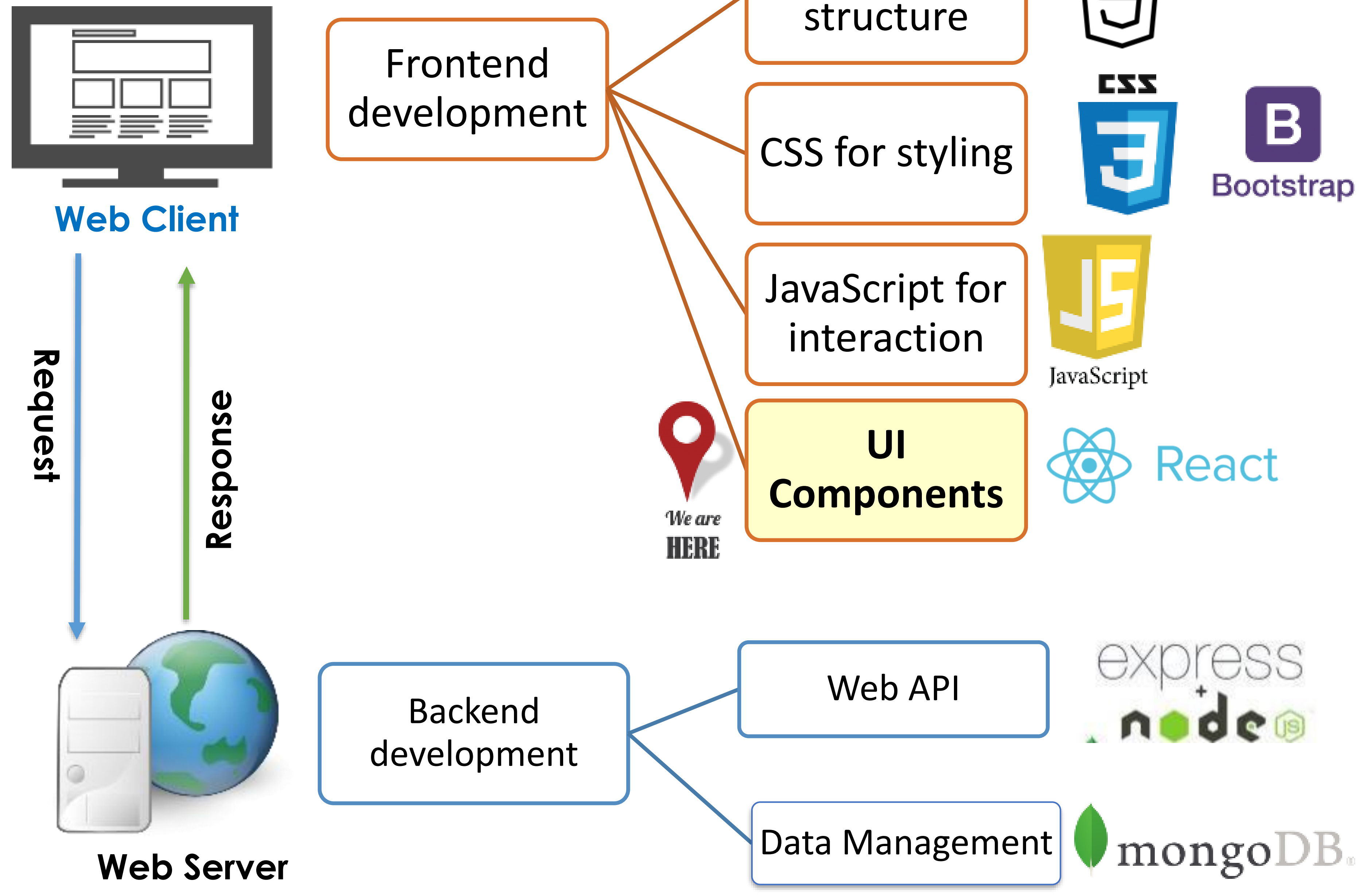
React Introduction

Used by Facebook, Instagram, Netflix, Dropbox, Yahoo, Khan Academy, ….

https://github.com/facebook/react/wiki/Sites-Using-React

< Back

Course Roadmap

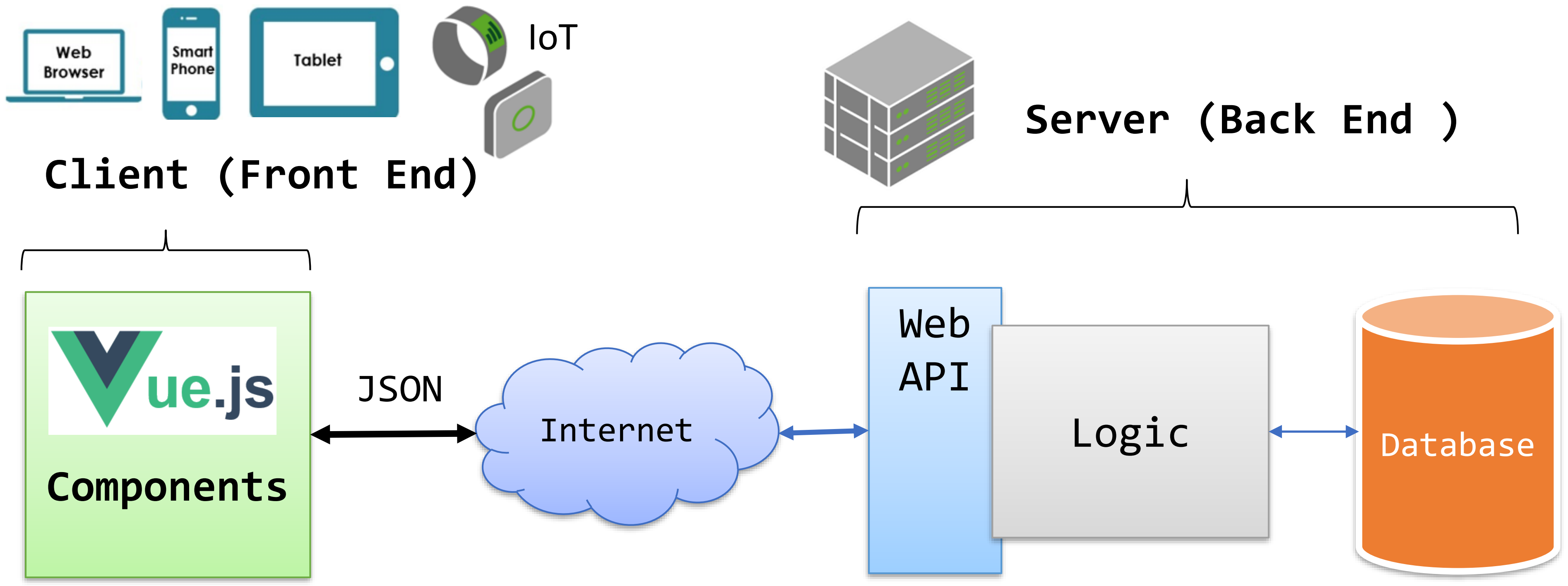


What is React?

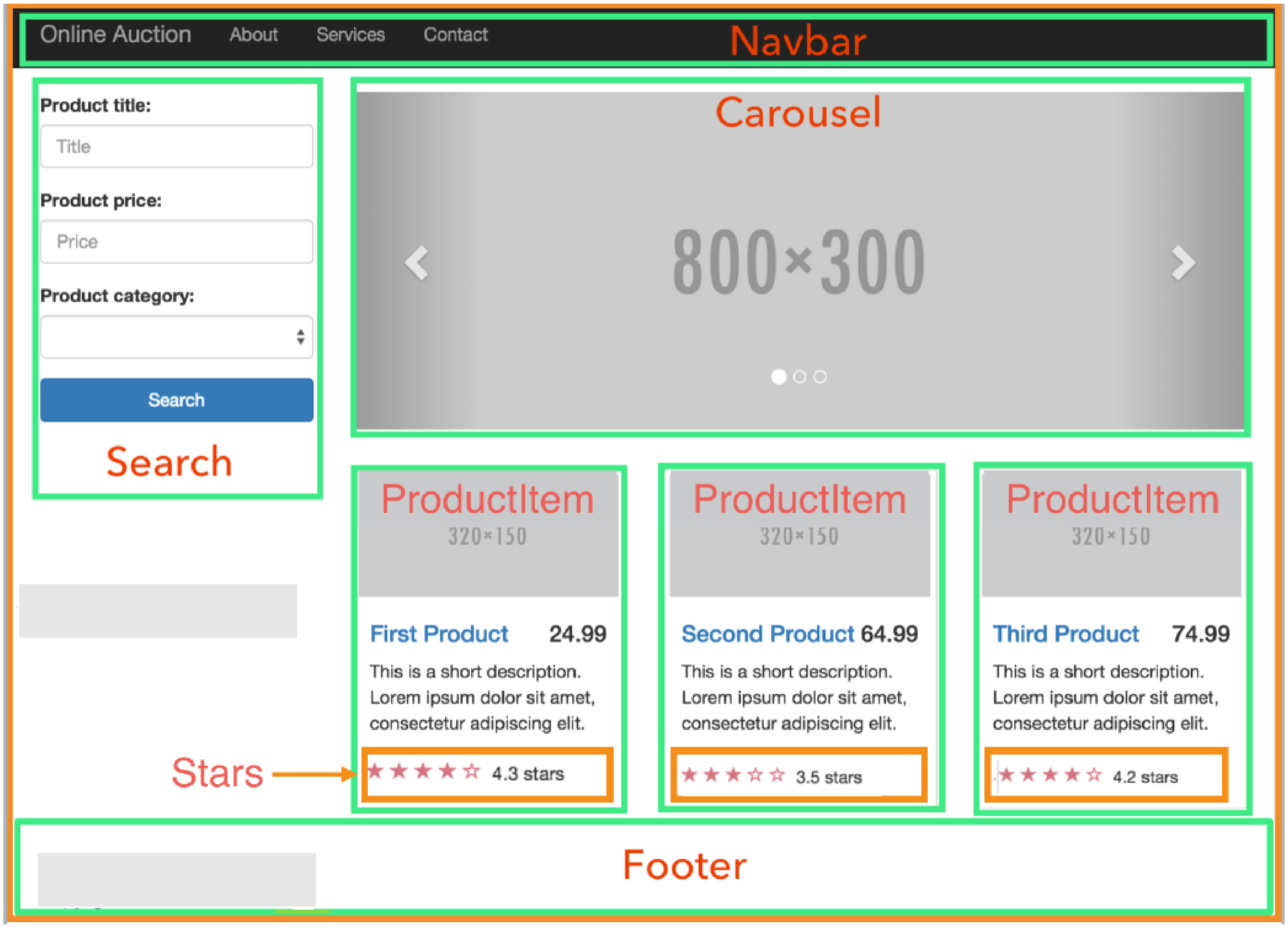
- React is an open source library for building **components-based user interfaces (UI)**
  - UI is **composed** of small reusable **components**
  - A component encapsulates **UI elements** and the **behavior** associated with them
- Ease creating a Single Page Application (SPA)
  - SPA is a Web app that load a single HTML page and **dynamically loads components** as the user interacts with the app
- Open-sourced by Facebook mid-2013
  - https://reactjs.org/
- Competing with Angular http://angular.io and Vue.js https://vuejs.org/

Components of Single Page Application (SPA)

- A **Single-Page Application (SPA)** has **1 main shell page** and **multiple UI components loaded** in response to user actions



An app = a tree of components

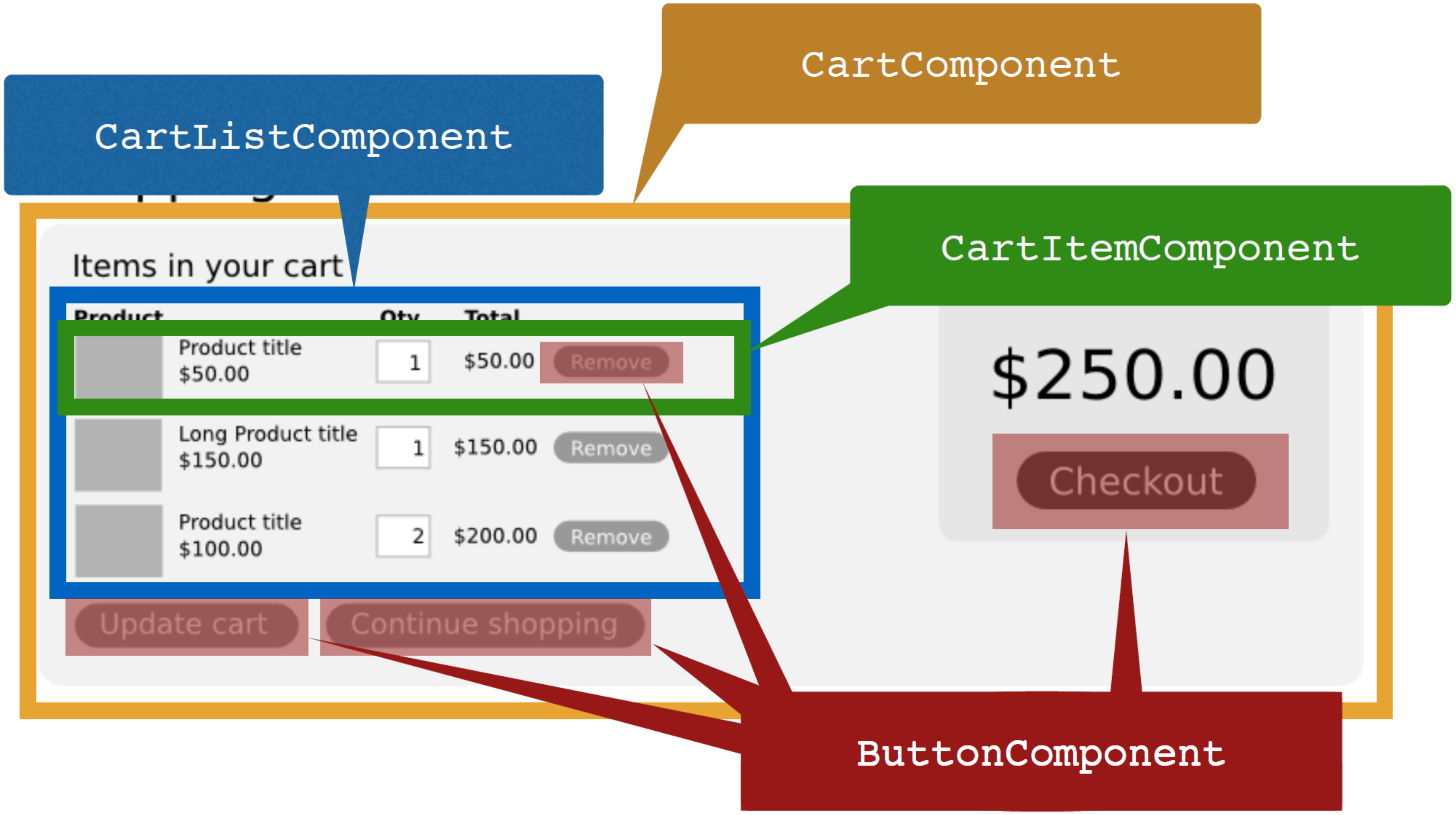


This is a short description. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

This is a short description. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

This is a short description. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

An app = a tree of components



React App Creator

- Run the React app creator:

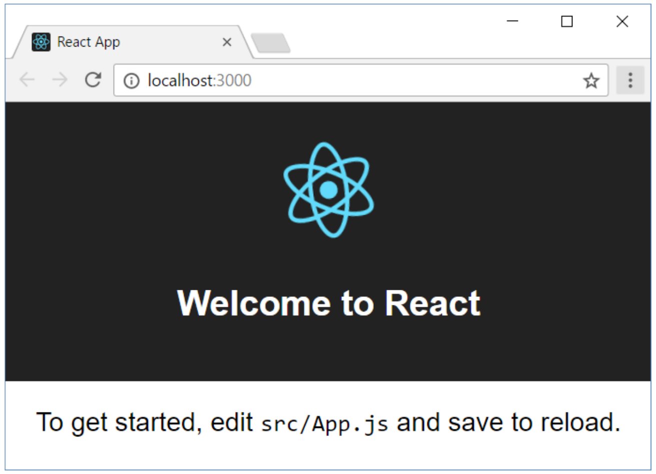
```
npx create-react-app my-app
```

- Starts your React app from the command line

```
cd my-app
npm start
```

- Browse you app from

http://localhost:3000

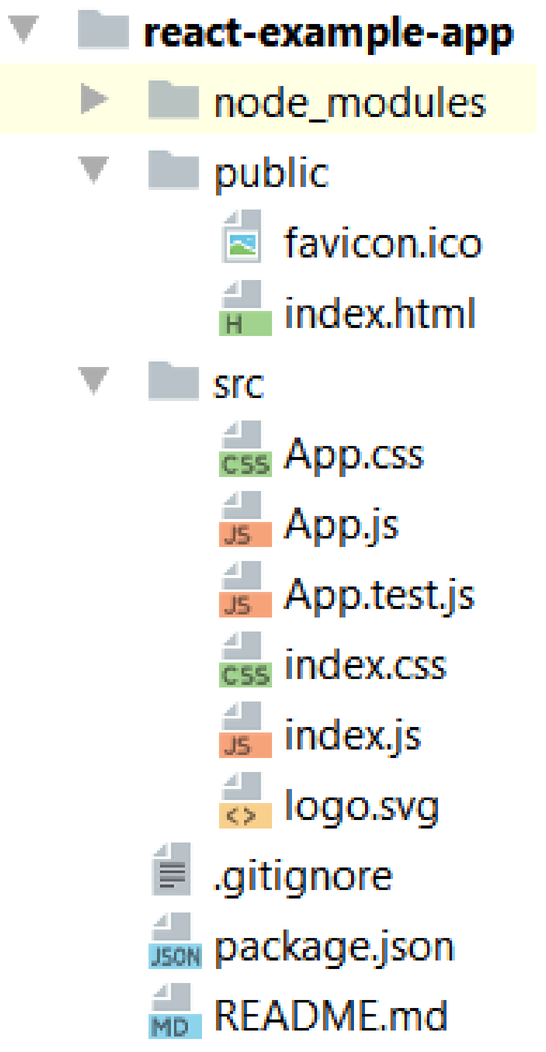


To get started, edit `src/App.js` and save to reload.

More info

https://github.com/facebook/create-react-app

React App Structure



```
react-example-app
  node_modules
  public
    favicon.ico
    index.html
  src
    App.css
    App.js
    App.test.js
    index.css
    index.js
    logo.svg
  .gitignore
  package.json
  README.md
```

`package.json` – project configuration

- Module name, dependencies, build actions

`index.html`

- App main HTML file

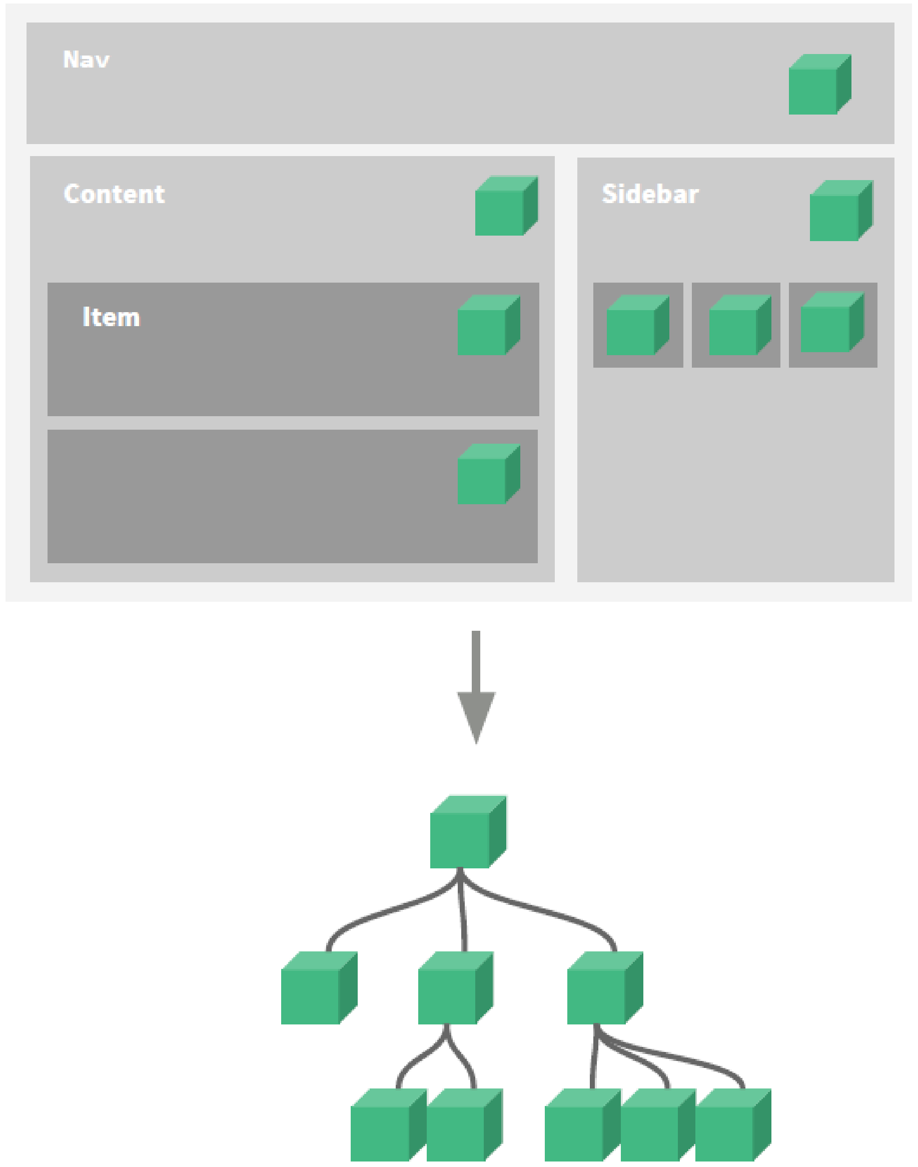
`index.js`

- App main JS file (startup script)

`App.js`, **`App.css`**

- React component "**App**"

React Components



Back

React Component

- A React App is composed of self-contained and often reusable **components**
- React automatically manages all UI **updates** when underlying data changes
- A ***component*** has:
  - **HTML *elements*** to provide the UI
  - Associated **properties** and **functions** to ***provide data*** and ***handle events*** raised from the UI elements
- Encapsulates **UI** and **logic** in a component
- Components allows creating new ‘HTML tags’

Component Example

- Create a **Welcome** component
  - Can accept a parameter called ***props***
  - Returns **JSX** : an HTML-like syntax to define the component UI
  - Component name must start with a capital letter

```
import React from "react";
function Welcome(props) {
    return <h1>Welcome to {props.appName}</h1>;
}
export default Welcome;
```

- Use the **Welcome** component

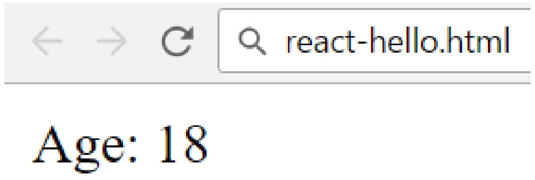
```
<Welcome appName='React Demo App' />
```

What is JSX?

JSX is a React's syntax for mixing HTML with JavaScript

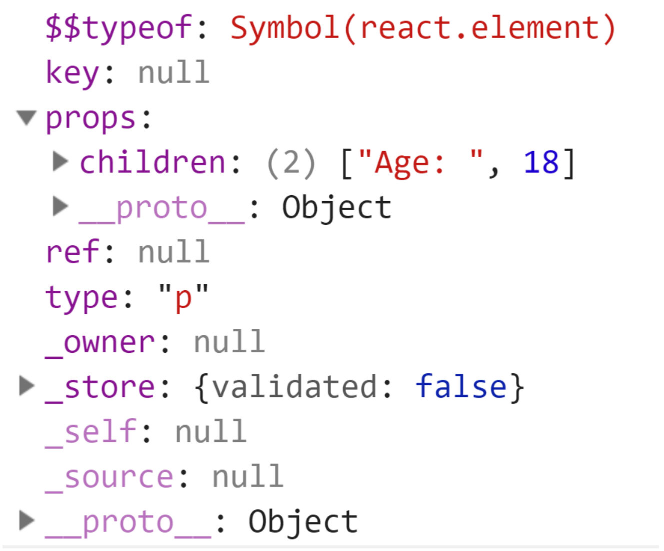
You can embed JavaScript expressions in JSX

```
const age = 18;
const ageElement = <p>Age: {age}</p>;
ReactDOM.render(ageElement,
    document.querySelector('#contentDiv'));
```



react-hello.html

Age: 18



```
  $$typeof: Symbol(react.element)
  key: null
▼ props:
  ▶ children: (2) ["Age: ", 18]
  ▶ __proto__: Object
  ref: null
  type: "p"
  _owner: null
▶ _store: {validated: false}
  _self: null
  _source: null
▶ __proto__: Object
```

JSX

- Syntactic extension to JavaScript
- Shorthand notation to represent JavaScript functions calls that evaluate to JavaScript objects

JSX

```
const element = (
      <h1 className="greeting">
            Hello, world!
      </h1>
);
```

$\Rightarrow$

JavaScript

```
const element = React.createElement(
      'h1',
      {className: 'greeting'},
      'Hello, world!'
);
```

It's just JavaScript!!

A Component can be Styled using CSS

Error: Invalid login!

```
function ErrorBox(props) {
    const css = {
        color: 'red', fontWeight: 'bold',
        border: '1px solid red', padding: '8px'
    };
    return <div style={css}>Error: {props.msg}</div>;
}
```

- Use the **ErrorBox** component

```
    <ErrorBox msg='Invalid login!'/>
```

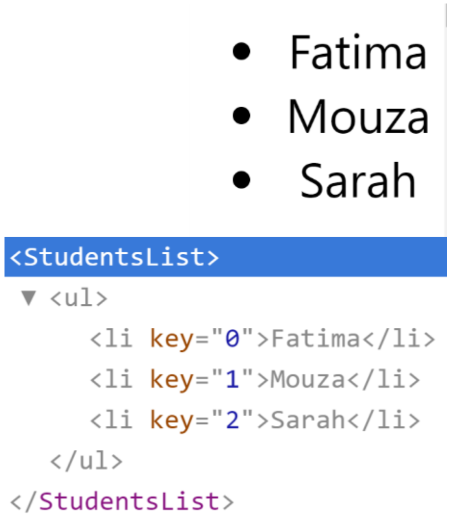
Attributes can be passed as parameters to the component. They are available in the component as **props** object

Rendering a List of items (with .map())

Lists are handled using **.map** array function

```
function StudentsList({students}) {
    return <ul>
                {students.map( (student, i) =>
                    <li key={i}>{student}</li>
                )}
           </ul>
}
```

Key helps identify which items have changed, added or removed



- Fatima
- Mouza
- Sarah

```
<StudentsList>
 ▼ <ul>
     <li key="0">Fatima</li>
     <li key="1">Mouza</li>
     <li key="2">Sarah</li>
   </ul>
</StudentsList>
```

- Use the **StudentsList** component

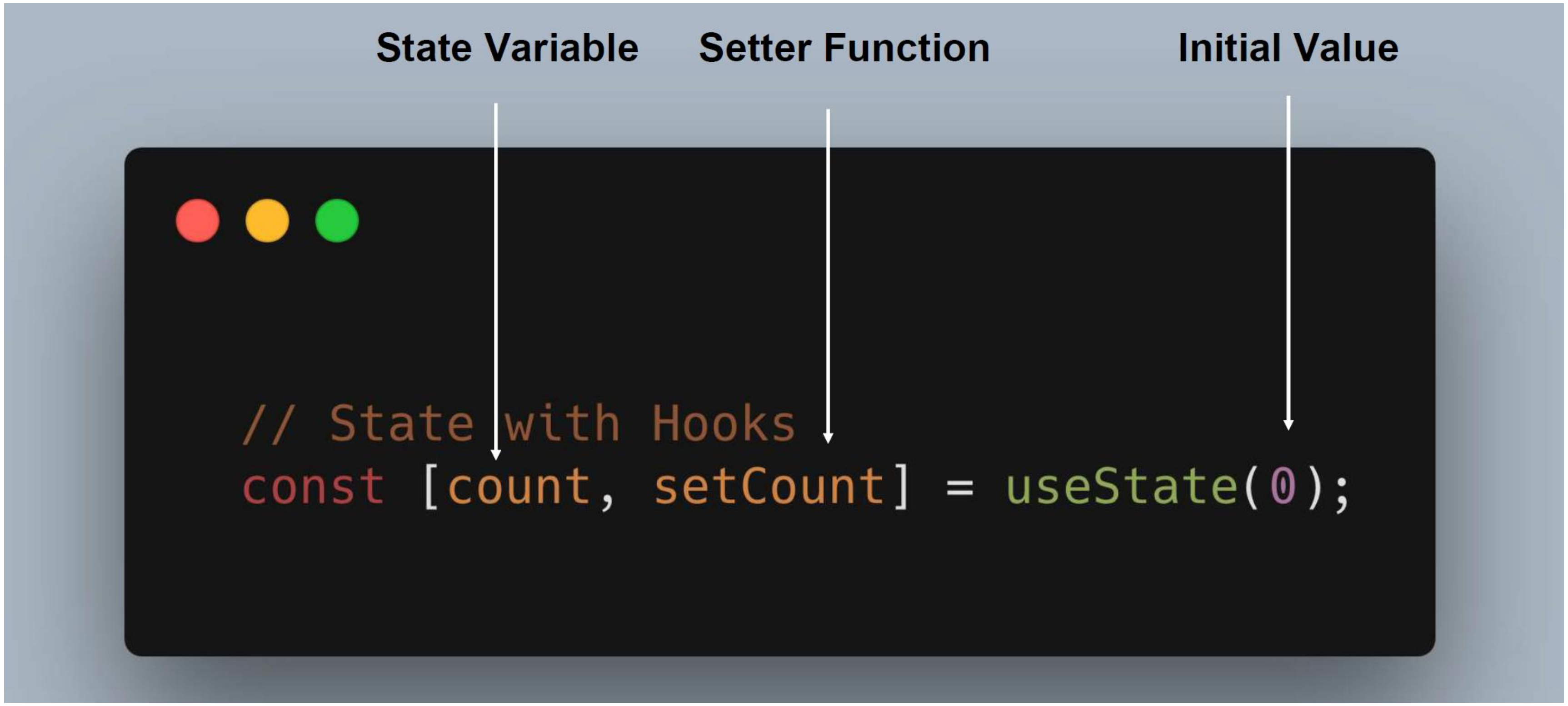
```
<StudentsList students={['Fatima', 'Mouza', 'Sarah']}/>
```

Component State

- Each component can store its own local data aka **state**
  - Private and fully controlled by the component
  - Can be passed as **props** to children
- Use **`useState`** Hook to create a *state variable* and an associated *function* to update the state
  - `const [count, setCount] = useState(0);`

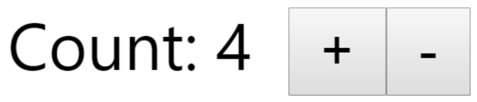
  Returns a state variable ***`count`*** initialized with 0 and a function ***`setCount`*** that updates it
  - Calling ***`setCount`*** causes React to **re-render the app components** and **update the DOM** to reflect the state changes
  - Never change the state directly butting assigning a value to the state variable

useState Hook



```
// State with Hooks
const [count, setCount] = useState(0);
```

Component With a State + Events Handling



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

function Counter(props) {
    const [count, setCount] = useState(props.startValue);
    const increment = () => { setCount(count + 1); };

    const decrement = () => { setCount(count - 1); };

    return <div>
            Count: {count}
            <button type="button" onClick={increment}>+</button>
            <button type="button" onClick={decrement}>-</button>
        </div>
}
export default Counter;
```

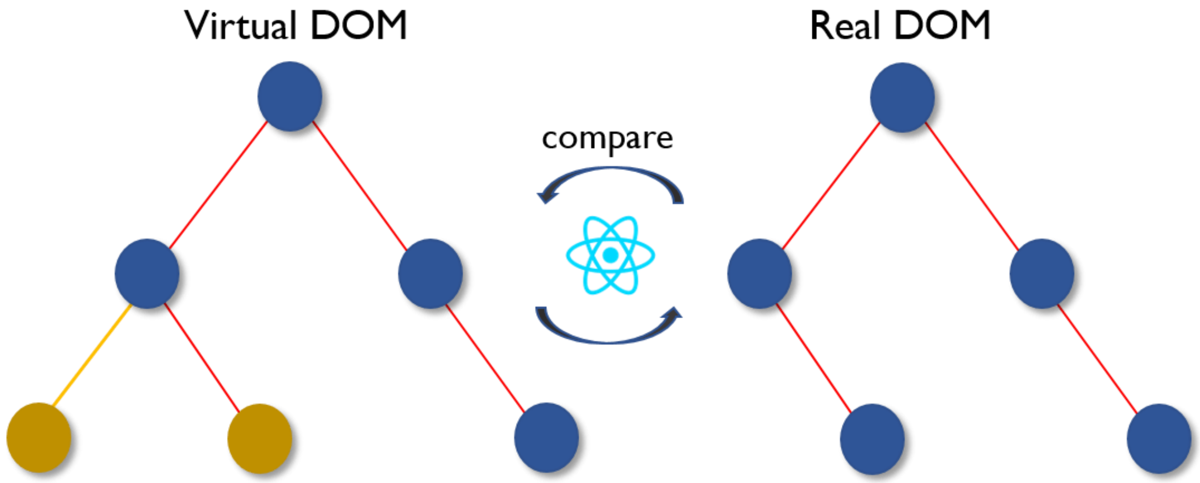
Handling events is similar to the way events are handled on DOM elements

- Use the **Counter** component

```
<Counter startValue={3}/>
```

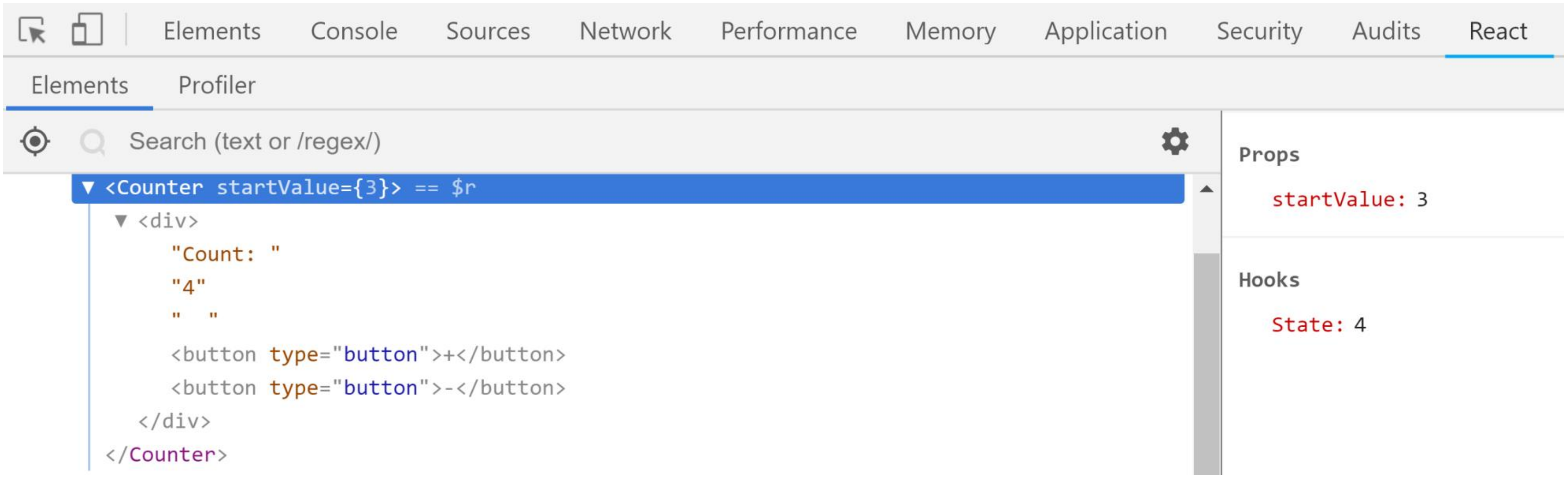
Virtual DOM

- Virtual DOM = Pure JS lightweight DOM, totally separate from the browser's slow JavaScript/C++ DOM API
- Every time you call setState...
  - A new virtual DOM tree is generated
  - New tree is **diffed** against old...
  - ...producing a minimum set of changes to be performed on real DOM to bring it up to date



Using the React Dev Tools

React Dev Tools
https://github.com/facebook/react-devtools

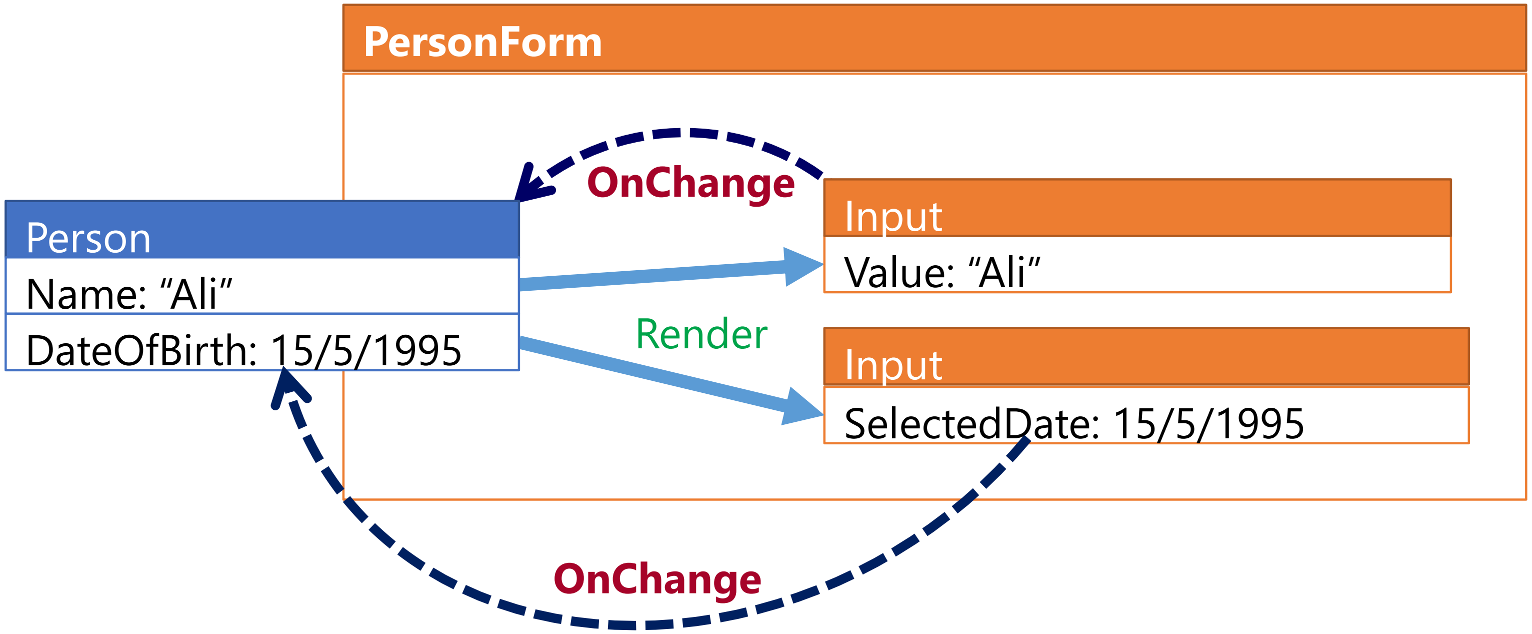


```
▼ <Counter startValue={3}> == $r
  ▼ <div>
      "Count: "
      "4"
      "  "
      <button type="button">+</button>
      <button type="button">-</button>
    </div>
</Counter>
```

Props
startValue: 3

Hooks
State: 4

Unidirectional Data Flow



Forms with React

```
<form onSubmit={handleSubmit}>
    <input
        name="email"
        type="email" required
        value={values.user}
        onChange={handleChange} />
    <input
        name="password"
        type="password" required
        value={values.password}
        onChange={handleChange} />
    <input type="submit" />
</form>
```

Form UI

Form State and Event Handlers

```
const [values, setValues] = useState({ email: "", password: "" });

const handleChange = e => {
    const name = e.target.name;
    const value = e.target.value;
    //Merge the object before change with the updated property
    setValues({ ...values, [name]: value });
};

const handleSubmit = e => {
    e.preventDefault();
    alert(JSON.stringify(values));
};
```

Uni-directional Data Flow: Props vs. State

Components can be passed data (**props**)

Props **(a.k.a. Parameters)**

- Public
- Read-only

Recommended

Components can manage their own **state**

State

- Private
- Modifiable

Only if needed

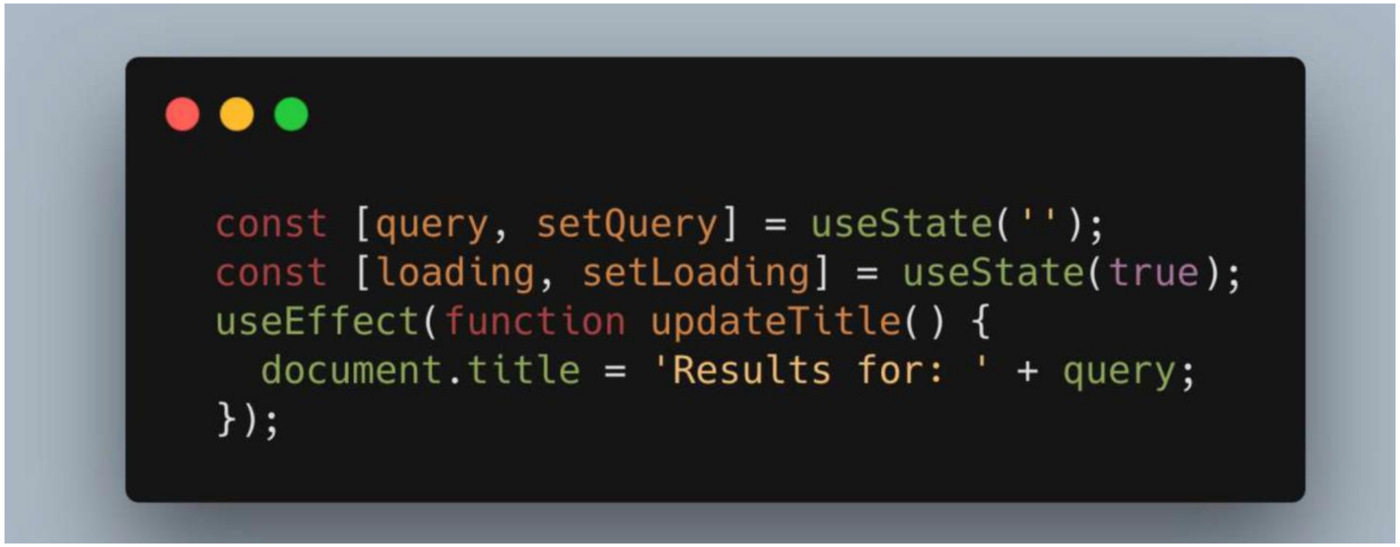
Props = Data handed from parent to child

useEffect

- Initialize state data when the component loads

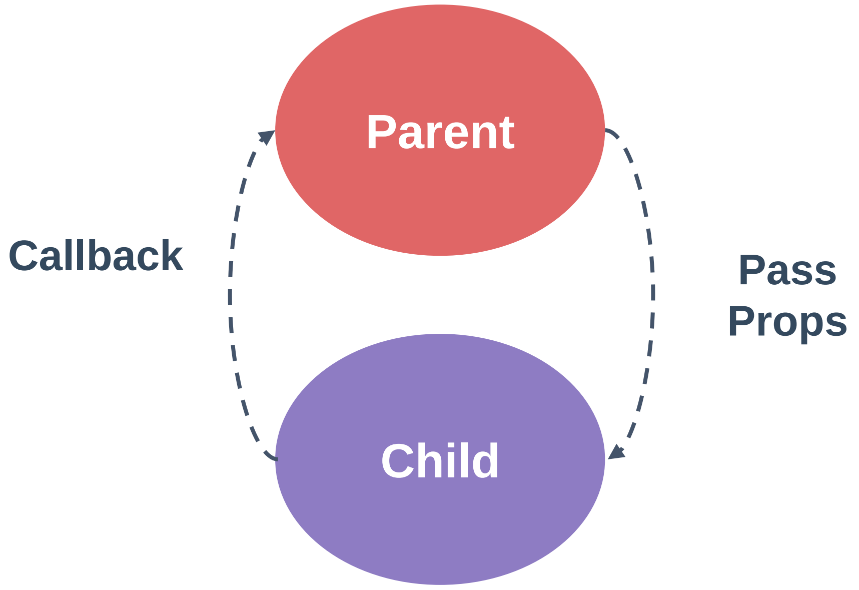
```
const [users, setUsers] = useState([]);

useEffect(async () => {
   const response = await fetch("https://api.github.com/users");
   const data = await response.json();
   setUsers(data); // set users in state
}, []); // pass empty array to run this effect once
```

- Executing something on every render



```
const [query, setQuery] = useState('');
const [loading, setLoading] = useState(true);
useEffect(function updateTitle() {
  document.title = 'Results for: ' + query;
});
```

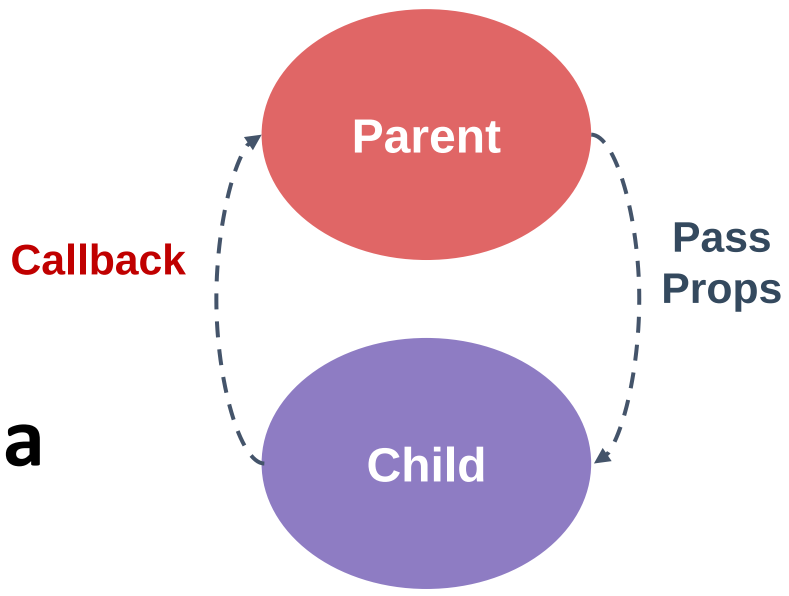
Components Communication



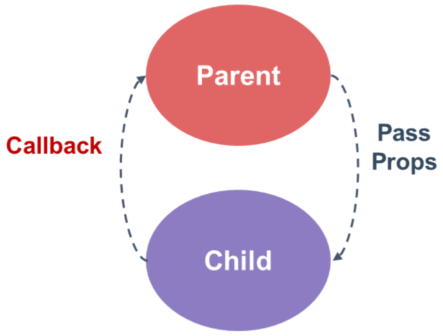
Back

Composing Components

- Components are meant to be used together, most commonly in parent-child relationships.
- Parent passes data down to the child via **props**,
- The child notify its parent of **a state change via** callbacks (a parent must pass the child a callback as a parameter)



Parent-Child Communication



Parent

```
<Counter startValue={3}
        onChange={count => console.log(`Count from the child component: ${count}`)}/>
```

Child

```
function Counter(props) {
    const [count, setCount] = useState(props.startValue);

    const increment = () => {
        const updatedCount = count + 1;
        setCount(updatedCount);
        props.onChange(updatedCount);
    };

    return <div>
        Count: {count}
        <button type="button" onClick={increment}>+</button>
    </div>
}
```

Routing

https://reacttraining.com/react-router/

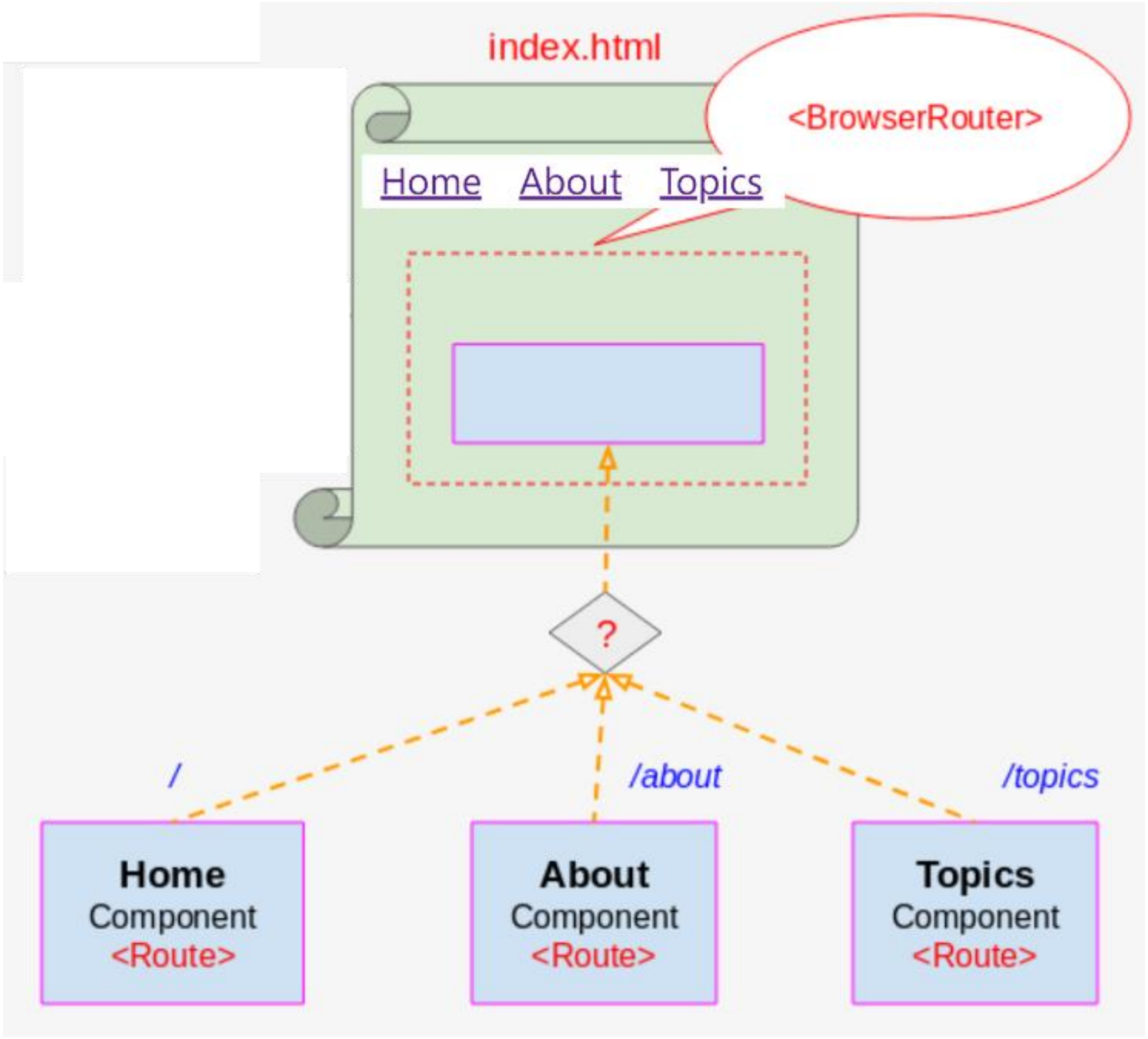
Back

Routing

- Routing implements client-side navigation for SPA:

  - Configure routes: map relative Url to the corresponding components in a declarative way

  - On URL change the router loads the associated component


index.html
<BrowserRouter>
Home About Topics
?
/
/about
/topics
Home Component <Route>
About Component <Route>
Topics Component <Route>

Routing with React Router

```
import React from "react";
import { BrowserRouter as Router, Route, Link } from "react-router-dom";

function RouterBasicExample() {
  return (
    <Router>
        <div>
            <ul>
                <li> <Link to="/">Home</Link> </li>
                <li> <Link to="/about">About</Link> </li>
                <li> <Link to="/topics">Topics</Link> </li>
            </ul>
            <hr />
            <Route exact path="/" component={Home} />
            <Route path="/about" component={About} />
            <Route path="/topics" component={Topics} />
        </div>
    </Router>
  );
}
```

Router programmatic access

- Request the router to navigate to a Url programmatically

```
props.history.push('/calculator');
```

- Get route parameter

```
props.match.params.heroId
```

Route configured in the router:

```
<Route path="/hero/:heroId" component={HeroForm} />
```

React Styled Components

- **Material-UI**: React components with Material Design

https://material-ui.com/

- ReactStrap Bootstrap Components

https://reactstrap.github.io

Summary

- React is awesome
- Decompose UI into self-contained and often reusable components

  - UI = Composition of Components
- React DOM uses JSX (JavaScript Extension) syntax to define component's UI
- Component Router ease loading components as the user interacts with the page

Resources

- Thinking in React

https://reactjs.org/docs/thinking-in-react.html

- Useful list of resources

https://github.com/enaqx/awesome-react

- Books

https://www.manning.com/books/react-in-action

- React Styled Components

https://reacttraining.com/react-router/