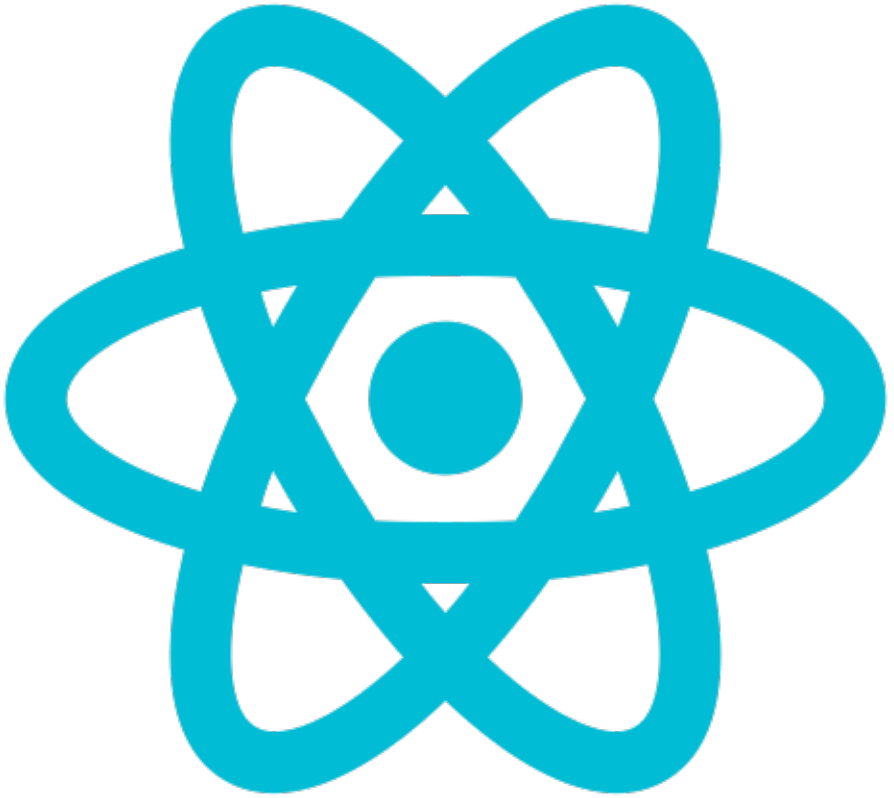
Introduction to React



React libraries

- This link after Javascript revision https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript

- React: https://reactjs.org/docs/react-api.html
- ReactDom: https://reactjs.org/docs/react-dom.html

Create React App

- Create React App is a comfortable environment for **learning React**, and is the best way to start building **a new single-page application** in React.
- It sets up your development environment so that you can use the latest JavaScript features, provides a nice developer experience, and optimizes your app for production. You'll need to have Node >= 10.16 and npm >= 5.6 on your machine. To create a project, run:

```
npx create-react-app my-app
cd my-app
npm start
```

JavaScript build toolchain

- A **package manager**, such as Yarn or npm. It lets you take advantage of a vast ecosystem of third-party packages, and easily install or update them.
- A **bundler**, such as webpack or Parcel. It lets you write modular code and bundle it together into small packages to optimize load time.
- A **compiler** (transpiler) such as Babel. It lets you write modern JavaScript code that still works in older browsers.

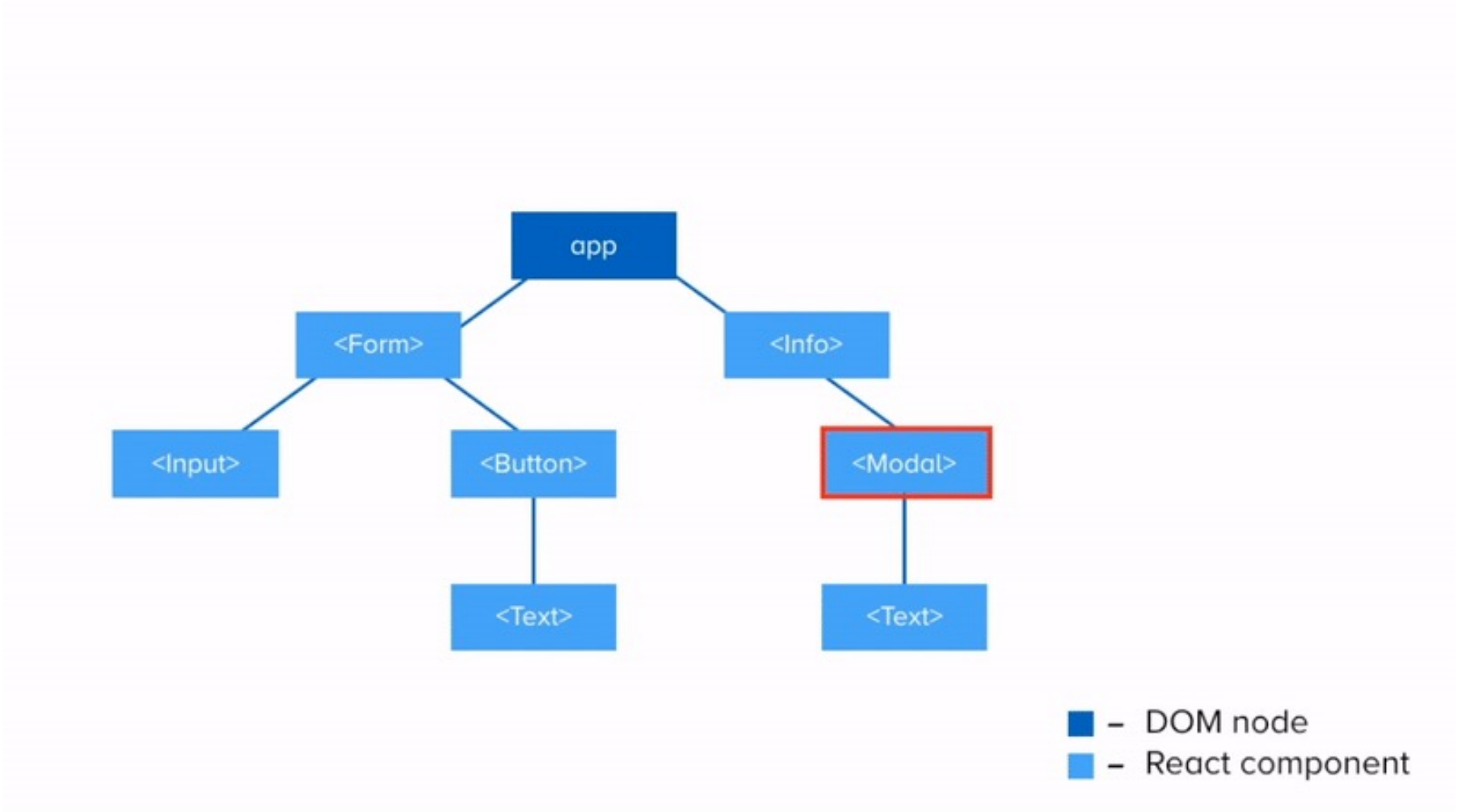
How does It work

What is the Virtual DOM?

- The virtual DOM (VDOM) is a programming concept where an ideal, or “virtual”, representation of a UI is kept in memory and synced with the “real” DOM by a library such as ReactDOM. This process is called reconciliation.

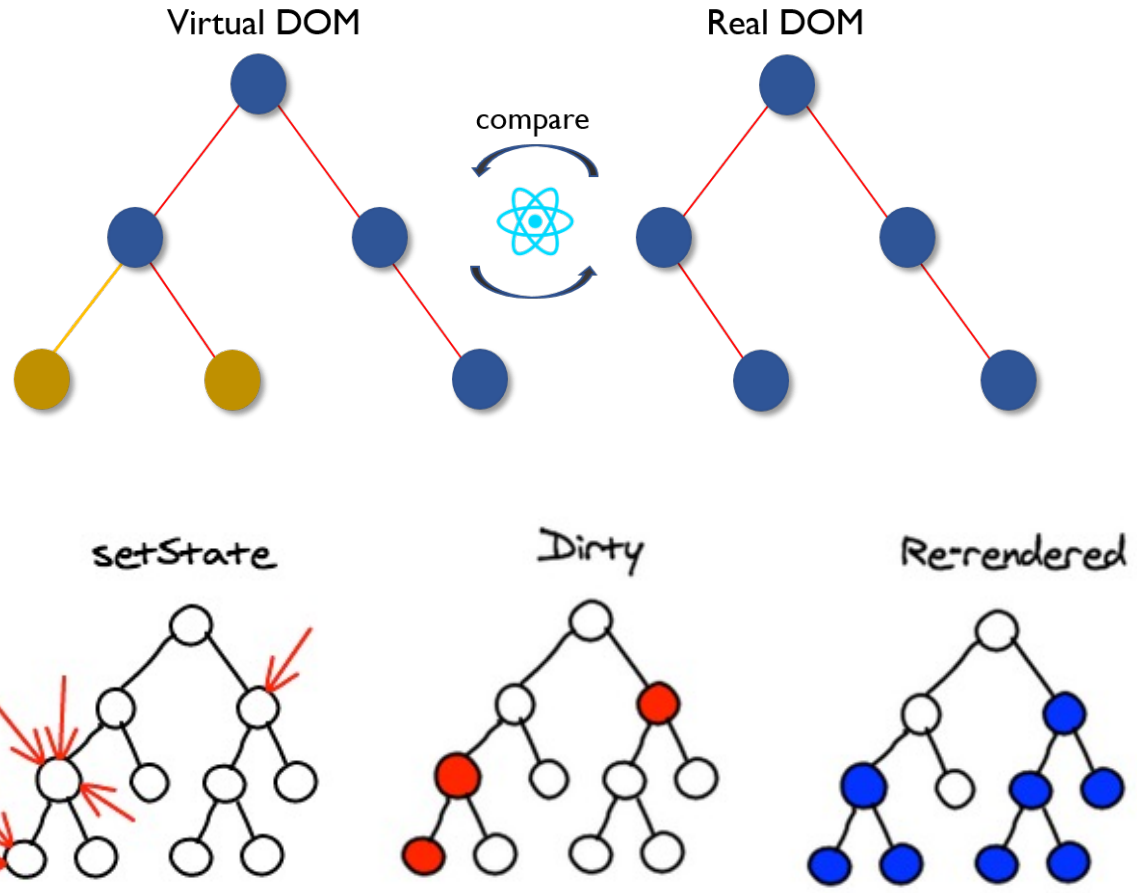
- When you use React, at a single point in time you can think of the render() function as creating a tree of React elements. On the next state or props update, that render() function will return a different tree of React elements. React then needs to figure out how to efficiently update the UI to match the most recent tree.

Virtual DOM Update



Virtual DOM Update

- **React** follows the observable pattern and listens for state changes. When the state of a component changes, **React updates** the **virtual DOM** tree. Once the **virtual DOM** has been **updated**, **React** then compares the current version of the **virtual DOM** with the previous version of the **virtual DOM**. This process is called "diffing".



Main concepts

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.

Introducing JSX

JSX

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

- This funny tag syntax is neither a string nor HTML.
- It is called JSX, and it is a syntax extension to JavaScript. It is recommend using it with React to describe what the UI should look like. JSX may remind you of a template language, but it comes with the full power of JavaScript.

Embedding Expressions in JSX

- In the example below, we declare a variable called name and then use it inside JSX by wrapping it in curly braces:

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

JSX is an Expression Too

- After compilation, JSX expressions become regular JavaScript function calls and evaluate to JavaScript objects.
- This means that you can use JSX inside of if statements and for loops, assign it to variables, accept it as arguments, and return it from functions:

```
function getGreeting(user) {
  if (user) {
    return <h1>Hello, {formatName(user)}!</h1>;
  }
  return <h1>Hello, Stranger.</h1>;
}
```

Specifying Attributes with JSX

- You may use quotes to specify string literals as attributes:

```
const element = <div tabIndex="0"></div>;
```

- You may also use curly braces to embed a JavaScript expression in an attribute:

```
const element = <img src={user.avatarUrl}></img>;
```

- Don’t put quotes around curly braces when embedding a JavaScript expression in an attribute. You should either use quotes (for string values) or curly braces (for expressions), but not both in the same attribute.

Warning

- Since JSX is closer to JavaScript than to HTML, React DOM uses camelCase property naming convention instead of HTML attribute names.

- For example, class becomes `className` in JSX, and `tabindex` becomes tabIndex.

Specifying Children with JSX

- If a tag is empty, you may close it immediately with />, like XML:

```
const element = <img src={user.avatarUrl} />;
```

JSX tags may contain children:

```
const element = (
  <div>
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
);
```

JSX Represents Objects

- Babel compiles JSX down to `React.createElement()` calls.

  These two examples are identical:

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

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

Rendering Elements

Element to render

- An element describes what you want to see on the screen:

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

- Unlike browser DOM elements, React elements are plain objects, and are cheap to create. React DOM takes care of updating the DOM to match the React elements.

- *Elements are not components!*

Rendering an element

- Let’s say there is a somewhere in your HTML file:

```
<div id="root"></div>
```

- We call this a “root” DOM node because everything inside it will be managed by React DOM.
- Applications built with just React usually have a single root DOM node. If you are integrating React into an existing app, you may have as many isolated root DOM nodes as you like.
- To render a React element into a root DOM node, pass both to ReactDOM.render():

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

Try it on CodePen

Updating the Rendered Element

- React elements are immutable. Once you create an element, you can't change its children or attributes. An element is like a single frame in a movie: it represents the UI at a certain point in time.
- With our knowledge so far, the only way to update the UI is to create a new element, and pass it to ReactDOM.render().
- Consider this ticking clock example:

```
function tick() {
  const element = (
    <div>
      <h1>Hello, world!</h1>
      <h2>It is {new Date().toLocaleTimeString()}.</h2>
    </div>
  );
  ReactDOM.render(element, document.getElementById('root'));
}

setInterval(tick, 1000);
```

Try it on CodePen