# React Basics

### What Is React?

React is a JavaScript library for creating dynamic UI (it’s not a framework)

* The “V” in MVC - Model View Controller
* Component Based Encapsulation
* Component State & Properties
* Virtual DOM
* Render on Client/Server
* Independent From Other Parts

### Why React?

* Fast & Responsive. Renders faster than the browser’s DOM
* Composable Components
* Pluggable - can plug into frameworks, such as Redux
* Simple to learn
* The brain of Facebook

### What is JSX?

JSX is an XML live syntax introduced by React. It is almost identical to HTML Upon rendering, JSX is compiled into plain JavaScript JSX looks like HTML, but sits in the middle of JavaScript:

render: function() {

return (

<div>

<p>Android - $199</p>

<button onClick={this.buy}>Buy</button>

</div>

);

}

You don’t have to use JSX with React, you can you plain JavaScript. But, for best practices, use JSX.

To code with React, ensure NodeJS is installed.

To create a new React app:

npx create-react-app my-app

‘npx’ is a package runner tool. ‘create-react-app’ doesn’t handle backend logic or databases; it just creates a frontend build pipline, so you can use it with any backend you want. Under the hood, it uses Babel and webpack.

Change the directory:

cd my-app

From this directory, you can run several commands:

**npm start** (Starts the development server)

**npm run build** (Bundles the app into static files for production)

**npm test** (Starts the test runner)

**npm run ejec**t (removes this too and copies build dependencies, configuration files and scripts into the app directory.

If you do this, you can go back) DON’T EJECT!!!

### Recommended Toolchains

* If you’re **learning React** or **creating a new[single-page](https://reactjs.org/docs/glossary.html" \l "single-page-application) app,** use [Create React App](https://reactjs.org/docs/create-a-new-react-app.html" \l "create-react-app).
* If you’re building a **server-rendered website with Node.js,** try [Next.js](https://reactjs.org/docs/create-a-new-react-app.html" \l "nextjs).
* If you’re building a **static content-oriented website,** try [Gatsby](https://reactjs.org/docs/create-a-new-react-app.html" \l "gatsby).
* If you’re building a **component library** or **integrating with an existing codebase**, try [More Flexible Toolchains](https://reactjs.org/docs/create-a-new-react-app.html" \l "more-flexible-toolchains).

### Main Concepts

The smallest React example looks like this:

ReactDOM.render(

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

document.getElementById(‘root’)

);

### What is React?

React is a declariative, efficient, and flexible JavaScript library for building user interfaces. It lets you compose complex UIs from small and isolated pieces of code called “components”.

React has a few different kinds of components, but we’ll start with React.Component subclasses:

class ShoppingList extends React.Component {

render() {

return (

<div className=”shopping-list”>

<h1>Shopping List for {this.props.name}</h1>

<ul>

<li>Instagram</li>

<li>WhatsApp</li>

<li>Oculus</li>

</ul>

</div>

);

}

}

We use components to tell React what we want to see on the screen. When our data changes, React will efficiently update and re-render our components.

Here, ‘ShoppingList’ is a **React component class**, or **React component type**. A component takes in parameters, called ‘props’ (short for ‘properties’), and returns a hierarchy of views to display via the ‘render’ method. render returns a **React.createElement**, which is a lightweight description of what to render:

return React.createElement(‘div’, {className: ‘shopping-list’),

React.createElement(‘h1’, /\* ... h1 children ...\*/),

React.createElement(‘ul’, /\* ...ul children ... \*/)

);

JSX comes with the full power of JavaScript. You can put any JavaScript expressions within braces of JSX.

The ShoppingList component above only renders built-in DOM components like <div /> and <li / >. But you can compose and render custom React components too. For example, we can now refer to the whole shopping list by writing <ShoppingList />. Each React component is encapsulated and can operate independently; this allows you to build complex UIs from simple components.

For event handling, (i.e. ‘onClick’) use ‘{() => alert(‘clicked!’)};

To remember things, components use **state** (this.state) in their constructors.

this.state should be considered as private to a React component that it’s defined in:

constructor(props) {

super(props);

this.state = {

value: null,

}

}

Note: In JavaScript classes, you need to always call super when defining the constructor of a subclass. All React component classes that have a constructor should start it with a super(props) call.

When you call setState in a component, React automatically updates the child components inside of it too; these are called **controlled components**.

### Why Immutability Is Important

There are generally two approaches to changing data. The first approach is to mutate the data by directly changing the data’s values. The second approach is to replace the data with a new copy which has the desired changes:

**Mutation**:

var player = {score: 1, name: ‘Jeff’};

player.score = 2;

**Without Mutation**:

var player = {score: 1, name: ‘Jeff’};

var newPlayer = ({}, player, {score: 2});

Immutability makes complex features much easier to implement. Avoiding direct mutation lets us keep previous versions and reuse them later.

Detecting changes in mutable objects is difficult because they are modified directly. Detecting changes in immutable objects is considerably easier.

The main benefit of immutability is that it helps you build pure components in React. This is done using shouldComponentUpdate() or React.PureComponent.

**‘const**’ is a signal that the identifierr won’t be reassigned.

const myConst = “Some text”;

‘**let**’ is a signal that the variable may be reassigned, such as a counter in a loop, or a value swap in an algorithm. It also signals that the variable will be used **only in the block it’s defined in**, which is not always the entire containing function.

for (let i = 0; i < someLength; i++) {

}