





React

Slavomír Moroz

TODO List

-   Wash dishes 
-   Kill spider 
-   Test item 

Create new

Styles & Resources

- Need to add styles?
 - [css-loader](#) | [sass-loader](#) | [less-loader](#)
- How do you want them to be linked?
 - Inline: [style-loader](#)
 - Separate file: [extract-loader](#) + [file-loader](#)
- Need some pictures?
 - Graphics, fonts & other binary resources
 - [file-loader](#) | [url-loader](#)

Getting started...

Learning materials that covers commit [7-ToDoList-component](#):

- [Named vs default exports](#)
- [State & lifecycle](#)
 - Do Not Modify State Directly
 - State Updates May Be Asynchronous
 - State Updates are Merged
- [React components](#)
 - [Constructor\(\)](#)
 - [SetState\(\)](#)
- [Elements List & Keys](#)
- [Handling events](#)
 - If calling `bind` annoys you, and you are using the experimental **property initializer** syntax, you can use property initializers to correctly bind callbacks:
 - [Stage-X \(Experimental Babel Presets\)](#) + babel-eslint parser

```
Code
class LoggingButton extends React.Component {
  // This syntax ensures `this` is bound within handleClick.
  // Warning: this is *experimental* syntax.
  handleClick = () => {
    console.log('this is:', this);
  }

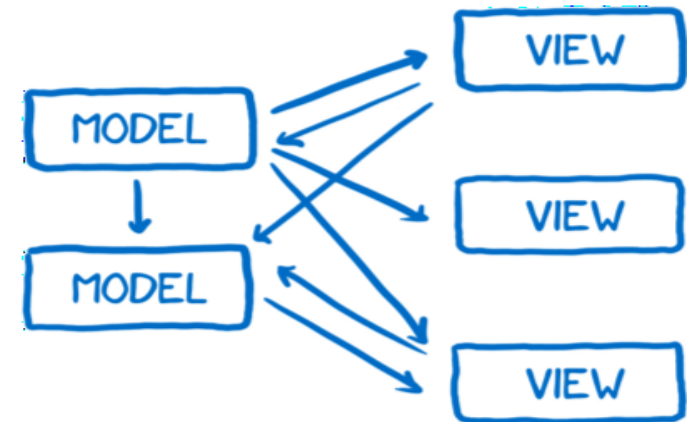
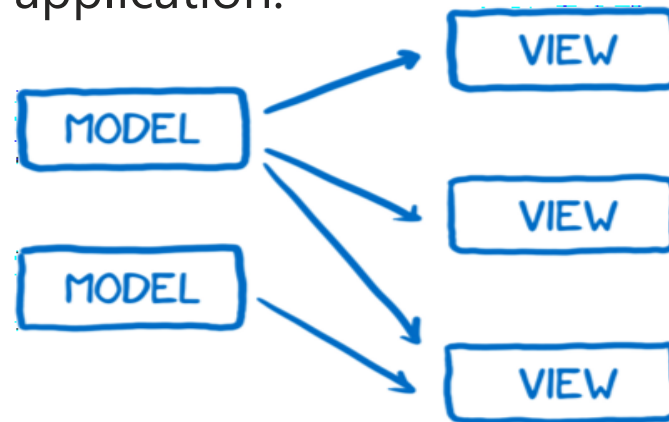
  render() {
```

Immutable

Changes made to objects and arrays by direct mutation will not be detected, and components will not re-render.

Side effects like mutation or asynchronous behavior will cause time travel to alter behavior between steps, breaking the application.

```
> a = {x: 1}
b = {A: a}
a.y = "I have mutated!";
console.log(b)
▼ {A: {...}} ⓘ
  ► A: {x: 1, y: "I have mutated!"}
```



Object immutability

Cloning an object

- `Object.assign(target, ...sources)`
 - `var obj = { a: 1 }; var copy = Object.assign({}, obj); console.log(copy); // { a: 1 }`
- Spread operator
 - `var obj = { a: 1 }; var copy = {...obj}; console.log(copy); // { a: 1 }`

Immutable object implementations

- Flow or Typescript objects / interfaces without mutation capabilities
 - `interface IObj { readonly a: number }`
- [Immutable-js](#) - immutable collections for JavaScript
- [Immutable-js Record](#) – immutable objects

Composing components

- Components accept inputs (called "props") and return React elements describing what should appear on the screen.
- Components can refer to other components in their output.
- Components must return a single root element.

Props

- A component must **never modify** its own **props** - all React components must act like pure functions with respect to their props.
- [PropTypes](#) - React built-in typechecking abilities.

<https://facebook.github.io/react/docs/components-and-props.html>

Functional components

- A.K.A Stateless components

Code

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

- This function is a valid React component because it accepts a single "props" object argument with data and returns a React element.

Browser events & DOM attributes

Some attributes work differently in React and HTML

<https://facebook.github.io/react/docs/dom-elements.html#differences-in-attributes>

SyntheticEvent

- a cross-browser wrapper around the browser's native event.
- works identically across all browsers.
- similar interface to native event
 - includes *stopPropagation()* and *preventDefault()*
- *SyntheticEvent* object will be reused and all properties will be nullified after the event callback has been invoked.
 - You **cannot access** the **event** in an **asynchronous way**.

Lifecycle

- Each component has several "lifecycle methods" that you can override to run code at particular times in the process.
 - Methods prefixed with `will` are called right before something happens
 - methods prefixed with `did` are called right after something happens

<https://facebook.github.io/react/docs/react-component.html#the-component-lifecycle>

Pure components

shouldComponentUpdate()

- The default behavior is to re-render on every state & props change
- If *shouldComponentUpdate()* returns false, then *componentWillUpdate()*, *render()*, and *componentDidUpdate()* will not be invoked.

To improve performance you may inherit from `React.PureComponent` which implements *shouldComponentUpdate()* with a shallow prop and state comparison.

- use immutable objects to facilitate fast comparisons of nested data
- shallow equality check of immutable objects produces same results as deep equality check of mutable object, but it's faster

<https://facebook.github.io/react/docs/react-api.html#react.purecomponent>

<https://facebook.github.io/react/docs/optimizing-performance.html#shouldcomponentupdate-in-action>

Refs to DOM elements

When the `ref` attribute is used on an HTML element, the `ref` callback receives the underlying DOM element as its argument.

When to Use Refs

- Managing focus, text selection, or media playback.
- Calculating viewport position.
- Triggering imperative animations.
- Integrating with third-party DOM libraries.

```
focus() {  
  // Explicitly focus the text input using the raw DOM API  
  this.textInput.focus();  
}  
  
render() {  
  // Use the `ref` callback to store a reference to the text input DOM  
  // element in an instance field (for example, this.textInput).  
  return (  
    <div>  
      <input  
        type="text"  
        ref={{input) => { this.textInput = input; }} />  
      <input  
        type="button"  
        value="Focus the text input"  
        onClick={this.focus}  
      />  
    </div>  
  );  
}
```

Children

In JSX expressions that contain both an opening tag and a closing tag, the content between those tags is passed as a special prop: `props.children`.

```
<MyContainer>  
  <MyFirstComponent />  
  <MySecondComponent />  
</MyContainer>
```

<https://facebook.github.io/react/docs/jsx-in-depth.html#children-in-jsx>

Demo example – [React Transition Group](#)

Higher order components

A [higher-order component](#) (HOC) is an advanced technique in React for reusing component logic.

Concretely, a **higher-order component is a function that takes a component and returns a new component.**

HOC *composes* the original component by *wrapping* it in a container component.

```
const EnhancedComponent = higherOrderComponent(WrappedComponent);
```

Demo example – [React-dnd](#)

Container components

- idea of separating presentation markup from logic in react components

	Presentational Components	Container Components
Purpose	How things look (markup, styles)	How things work (data fetching, state updates)
To read data	Read data from props	Subscribe to state
To change data	Invoke callbacks from props	Set state

[Presentational and Container components](#)

[Container components](#)

Debug

[React dev tools](#)

- lets you inspect the React component hierarchy, including component props and state.

[Component display name](#)

- The displayName string is used in debugging messages.
- [babel-plugin-add-react-displayname](#)
 - Automatically detects and sets displayName for React components.
- [babel-plugin-styled-components](#)
 - adds the components' name and displayName to the class name attached to the DOM node
 - `<button class="sc-Button-asdf123 asdf123" />` instead of just `<button class="asdf123" />`.