React Fundamentals

What is React?

- Rendering and event handling
- Maintained by Facebook
- Novel and revolutionary ideas
- Declarative
- Composable components

What is React?

Rendering

conversion of

- **data** that describes the state of the user interface

into

- **document object model objects** that the browser can use to produce a user interface that the user can see and interact with

Event handling

lets the programmer **detect user interactions** with their program and to specify **program response**.

What is React?

Maintained by Facebook

- created by Facebook
- maintained by Facebook.
- significant piece of Facebook's technology repertoire
- used in many of their projects.

What is React?

Novel and revolutionary ideas :

- Influence from functional programming
  - modeling components as functions,
  - programming by transforming values,
  - separating the calculation of UI changes from the application of those changes
- One-way data flow
  - enforce a symmetry between the UI model and the rendered user interface
- Virtual DOM
  - JavaScript object model that React uses to calculate user interface changes
- Vanilla JS for templating
  - no special UI template syntax

What is React?

- Declarative
  - A React application is a **set of components**, each of which **declaratively defines a mapping** between
    - some states and
    - the desired user interface
  - The interface is only changed by changing the state.
- Composable components
  - self-contained units of functionality
  - **interface** that defines
    - their inputs as properties and
    - their outputs as callbacks
  - components can be freely nested within each other (**composition**)

Advantages and Disadvantages

Advantages

Conceptual simplicity

Speed

Simple model for server-side rendering

Disadvantages

Limited in scope

Productivity

Complex tooling

React vs. Angular

React	Angular
Renders UI and handles events	A complete UI framework
Uses JavaScript for view logic	Custom “template expression” syntax
JavaScript	TypeScript

Demo

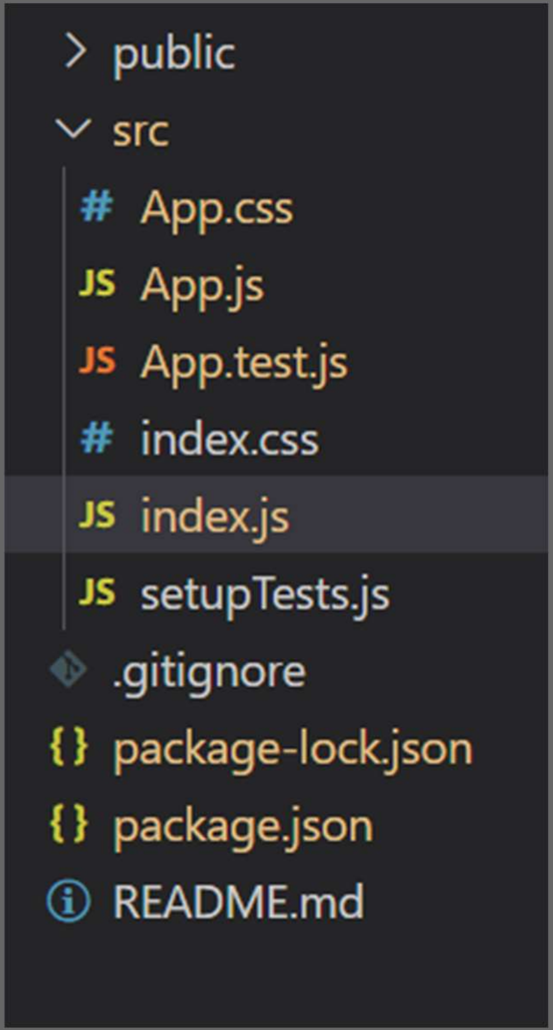
Setting up a React development environment

React Development Environment

- Initially React was a simple JavaScript file.
  - it is no longer practical
- Today it is necessary to use a **build system**
  - process many JavaScript files written with
    - modern JavaScript (ES6+ / ES2016...) and
    - React's JSX syntax
  - convert them to a format that could be loaded into browsers
- **create-react-app** (https://github.com/facebook/create-react-app)
  - React application bootstrapper released by Facebook
    - bootstraps extremely simple, lacking functionalities such as state management and routing
    - It will be needed to add and configure many other libraries on top of it
  - npx create-react-app <project name>

Initial Application: index.js

```
import ReactDOM from ‘react-dom’;
import React from ‘react’;
import './index.css';
import App from './App';

ReactDOM.render(
    <React.StrictMode>
      <App />
    </React.StrictMode>,
    document.getElementById(‘root’)
);
```



```
> public
v src
  # App.css
  JS App.js
  JS App.test.js
  # index.css
  JS index.js
  JS setupTests.js
  .gitignore
  {} package-lock.json
  {} package.json
  ⓘ README.md
```

Initial Component: app.js

```
import './App.css';

function App() {
  return (
    <div className="App">
      <h1>Initial React</h1>
    </div>
  );
}

export default App;
```

Initial Component test: app.test.js

```
import { render, screen } from '@testing-library/react';
import App from './App';

test('renders learn react link', () => {
  render(<App />);
  const linkElement = screen.getByText(/React/i);
  expect(linkElement).toBeInTheDocument();
});
```

Comandos npm

- Lifecycle scripts included:
  - **start** -> react-scripts start
  - **test** -> react-scripts test
- available via `npm run`:
  - **build** -> react-scripts build
  - **eject** -> react-scripts eject

Demo

Setting up a React development environment

Building a simple React application

What is the DOM?

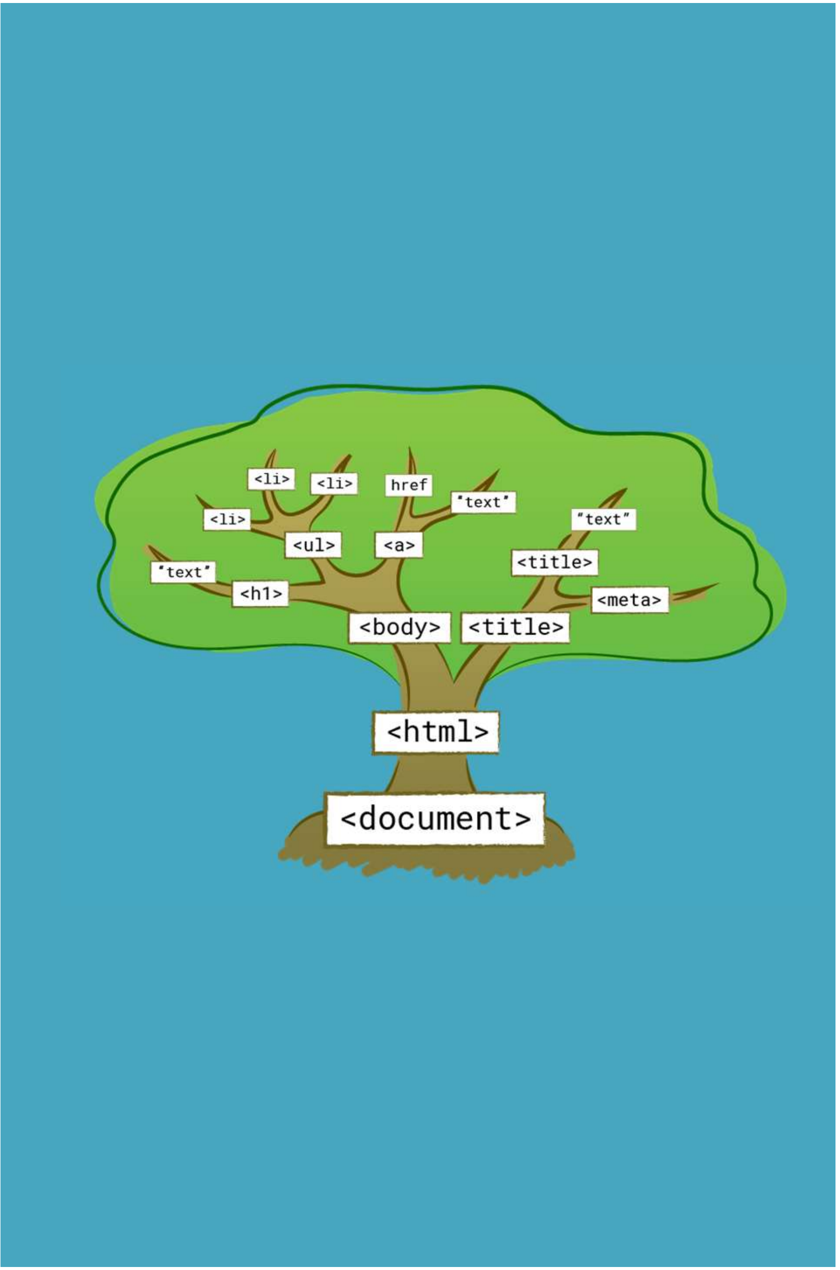
An API for HTML and XML documents

Defines the logical structure of documents

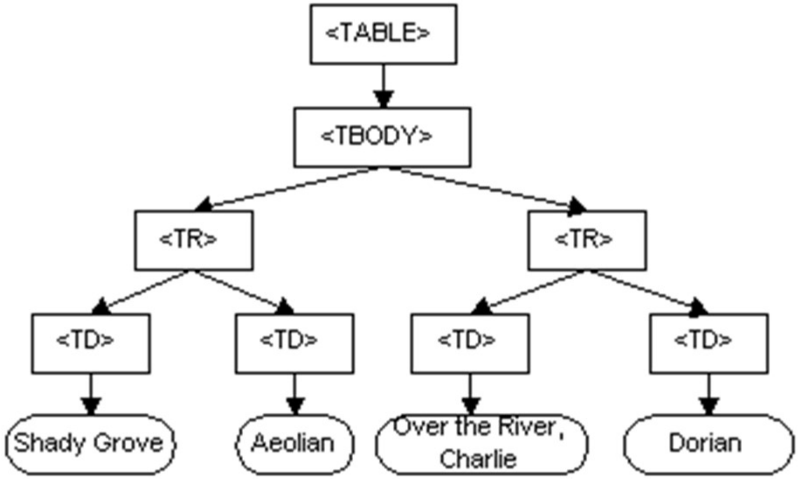
Defines the way a document is accessed and manipulated

With the Document Object Model, programmers can *build* documents, *navigate* their structure, and *add*, *modify*, or *delete* elements and content.

https://www.w3.org/TR/REC-DOM-Level-1/



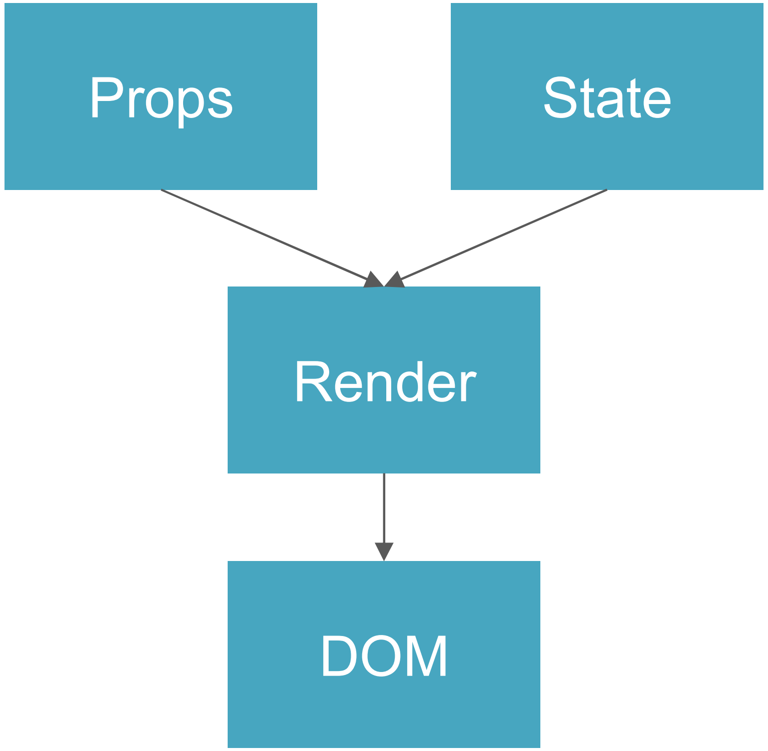
Anything found in an HTML or XML document can be *accessed*, *changed*, *deleted*, or *added* using the Document Object Model.



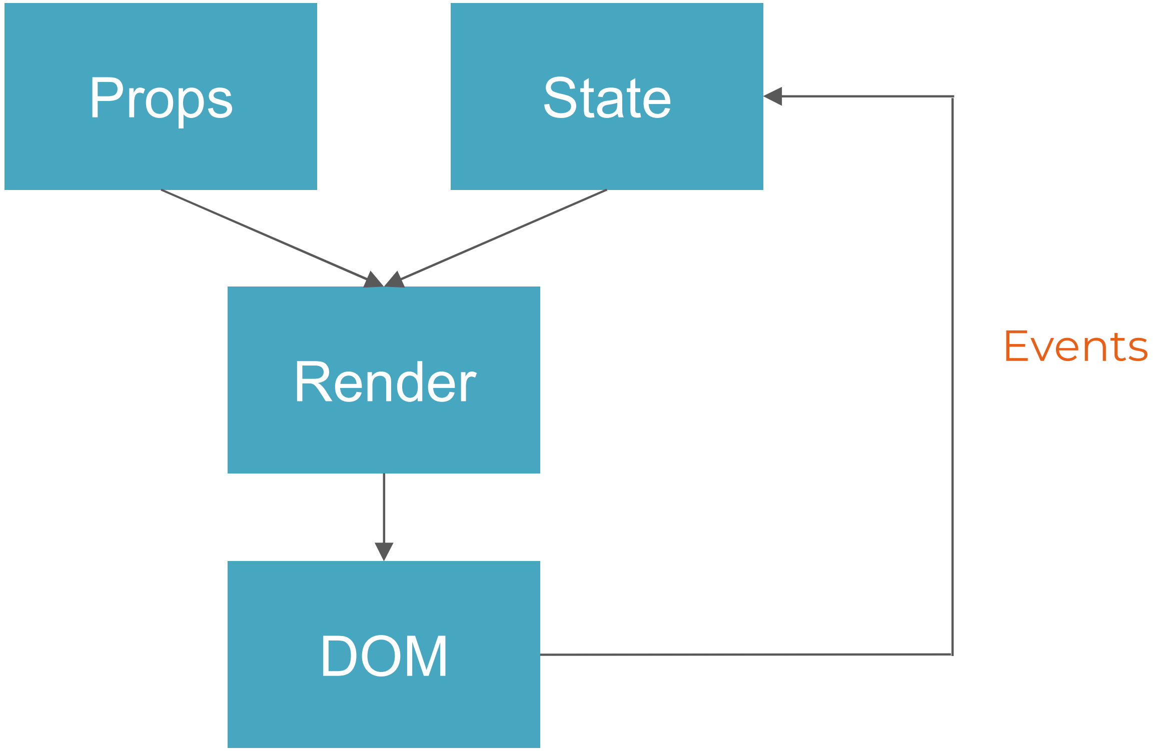
Consider an *Structured Model* rather than a tree or grove.

```
<TABLE>
  <TBODY>
    <TR>
      <TD>Shady Grove</TD>
      <TD>Aeolian</TD>
    </TR>
    <TR>
      <TD>Over the River, Charlie</TD>
      <TD>Dorian</TD>
    </TR>
  </TBODY>
</TABLE>
```

Architecture



Architecture

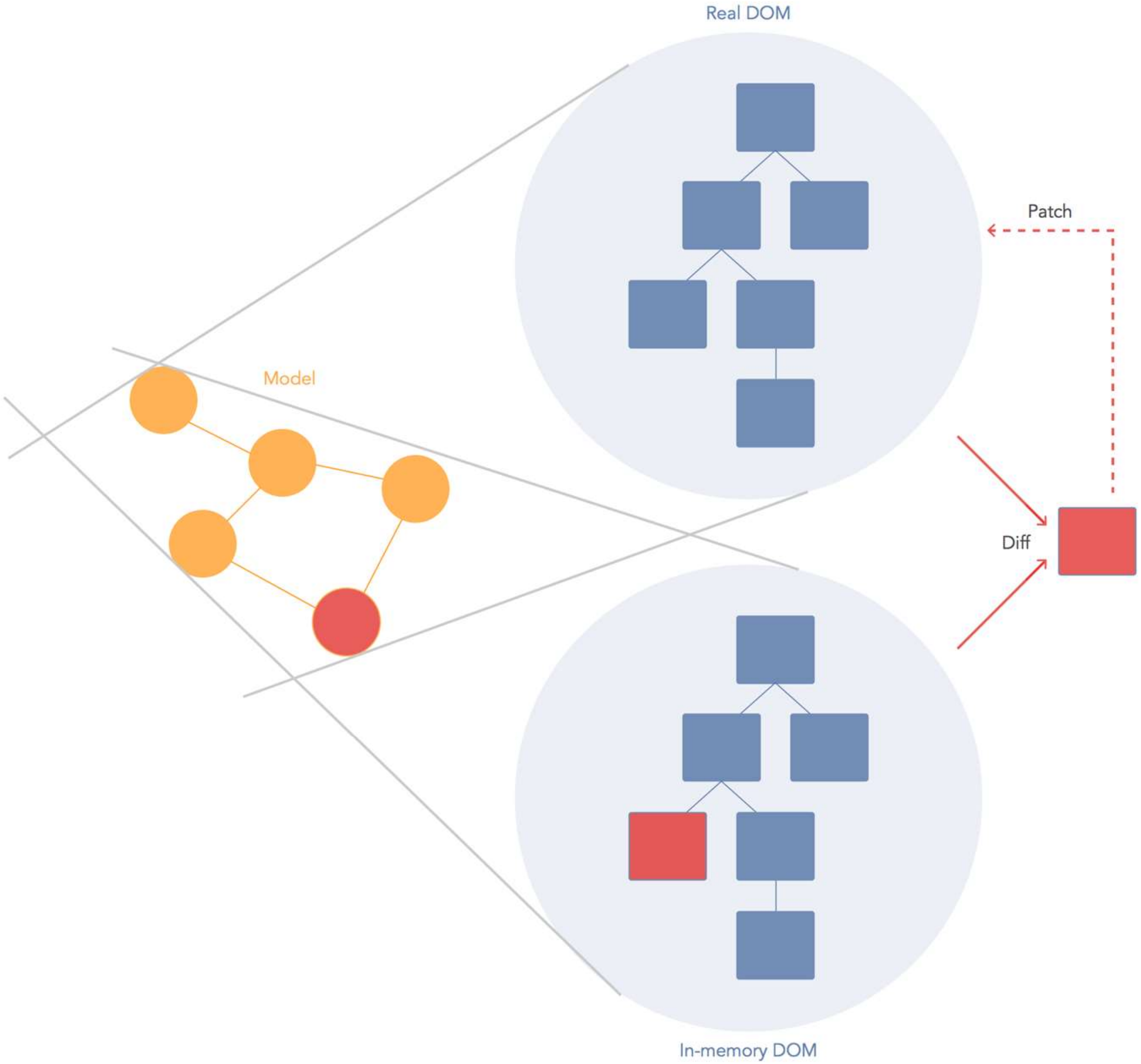


Model + Component = DOM

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.

This process is called reconciliation.

https://reactjs.org/docs/faq-internals.html

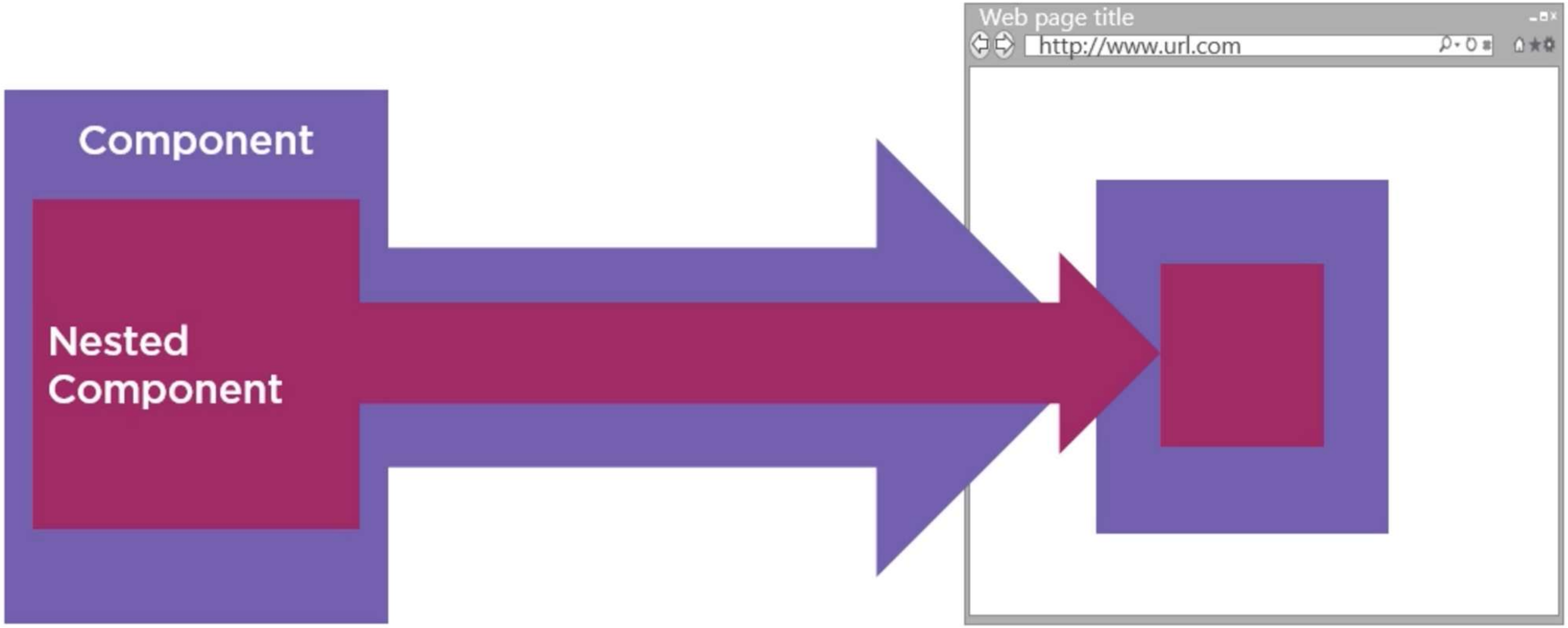


Virtual DOM

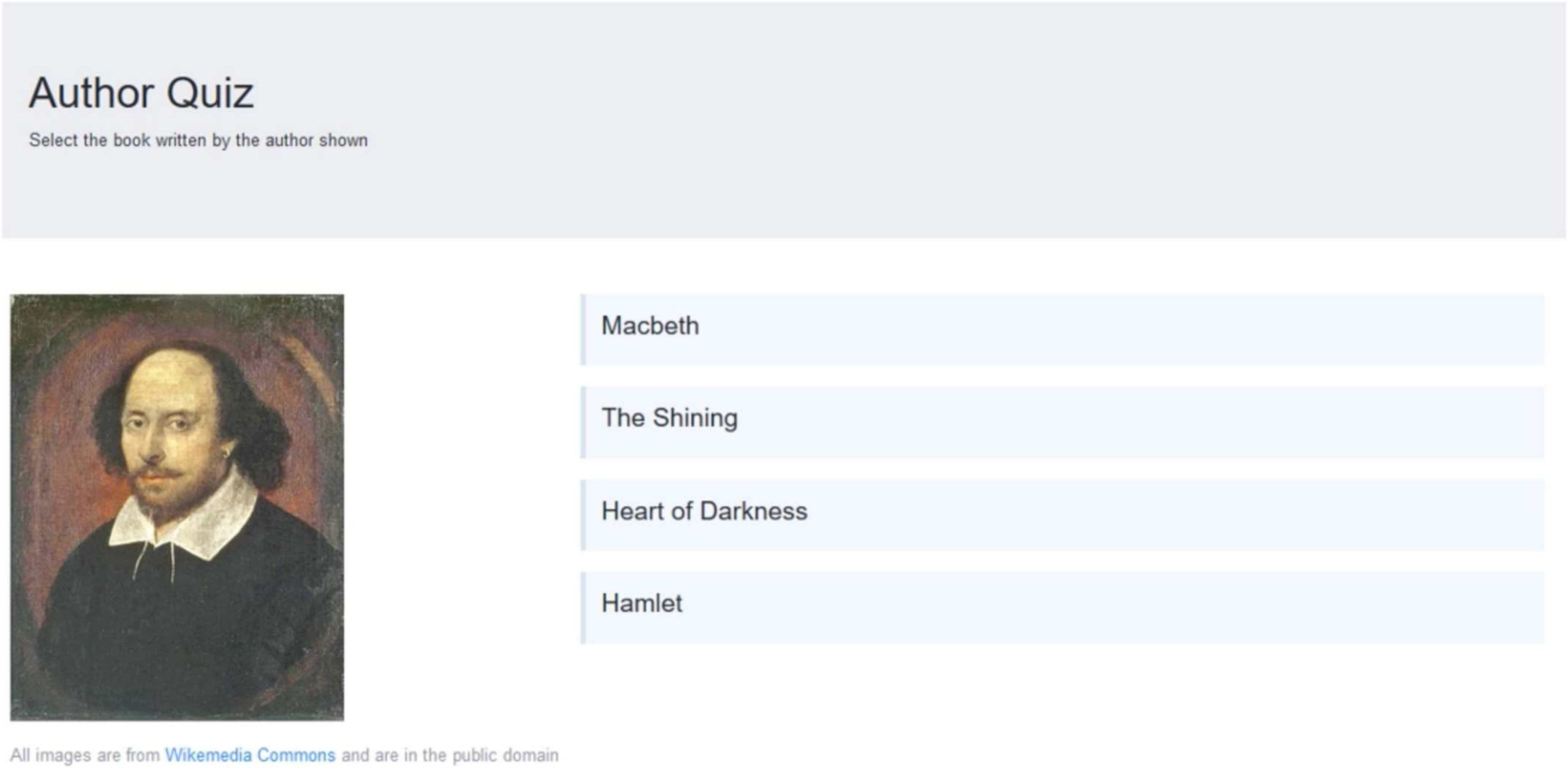
What Is a Component?

React Application

DOM



The Author Quiz



Author Quiz

Select the book written by the author shown

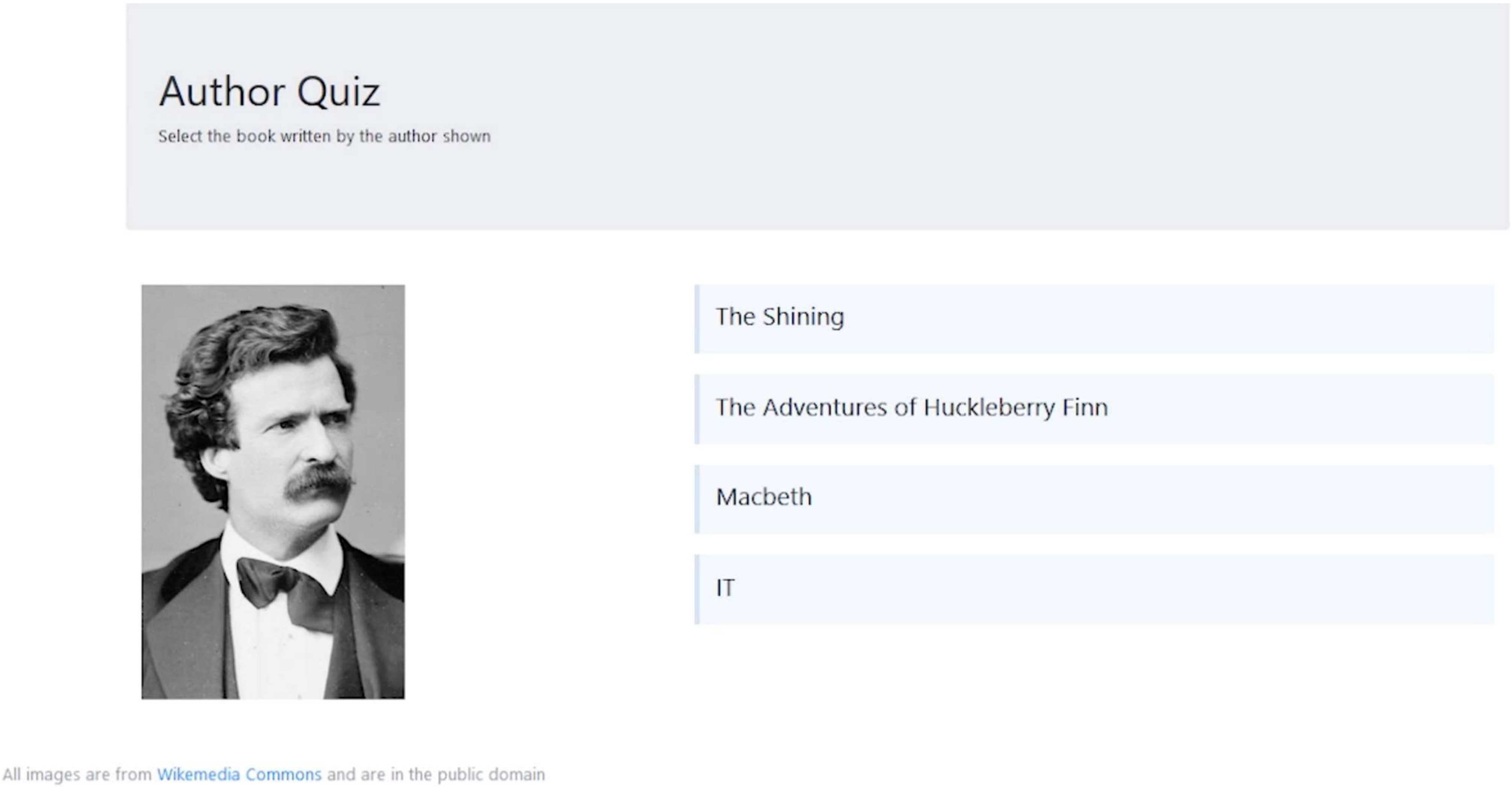
Macbeth

The Shining

Heart of Darkness

Hamlet

All images are from Wikemedia Commons and are in the public domain



Author Quiz

Select the book written by the author shown

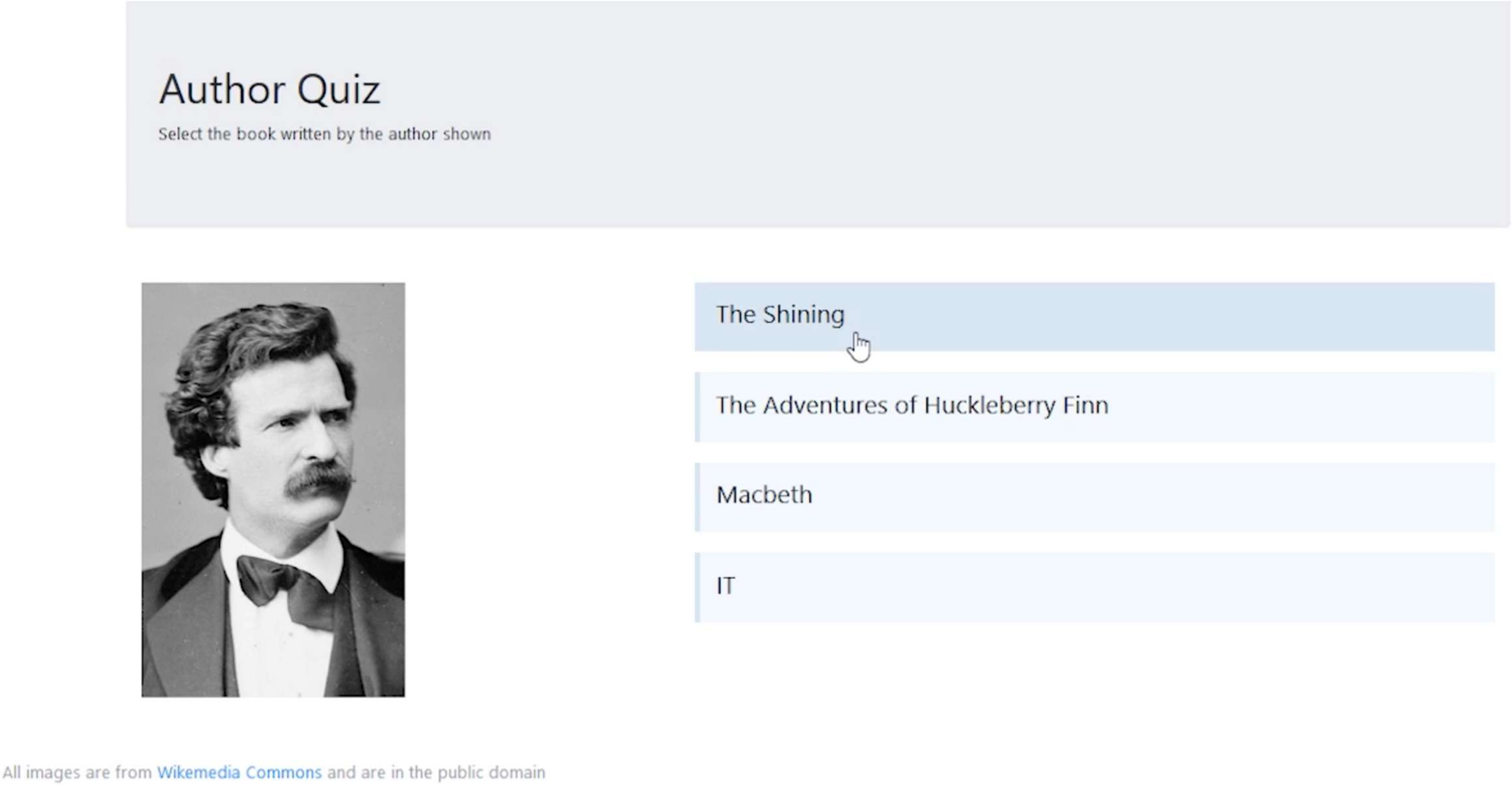
The Shining

The Adventures of Huckleberry Finn

Macbeth

IT

All images are from Wikemedia Commons and are in the public domain



Author Quiz

Select the book written by the author shown

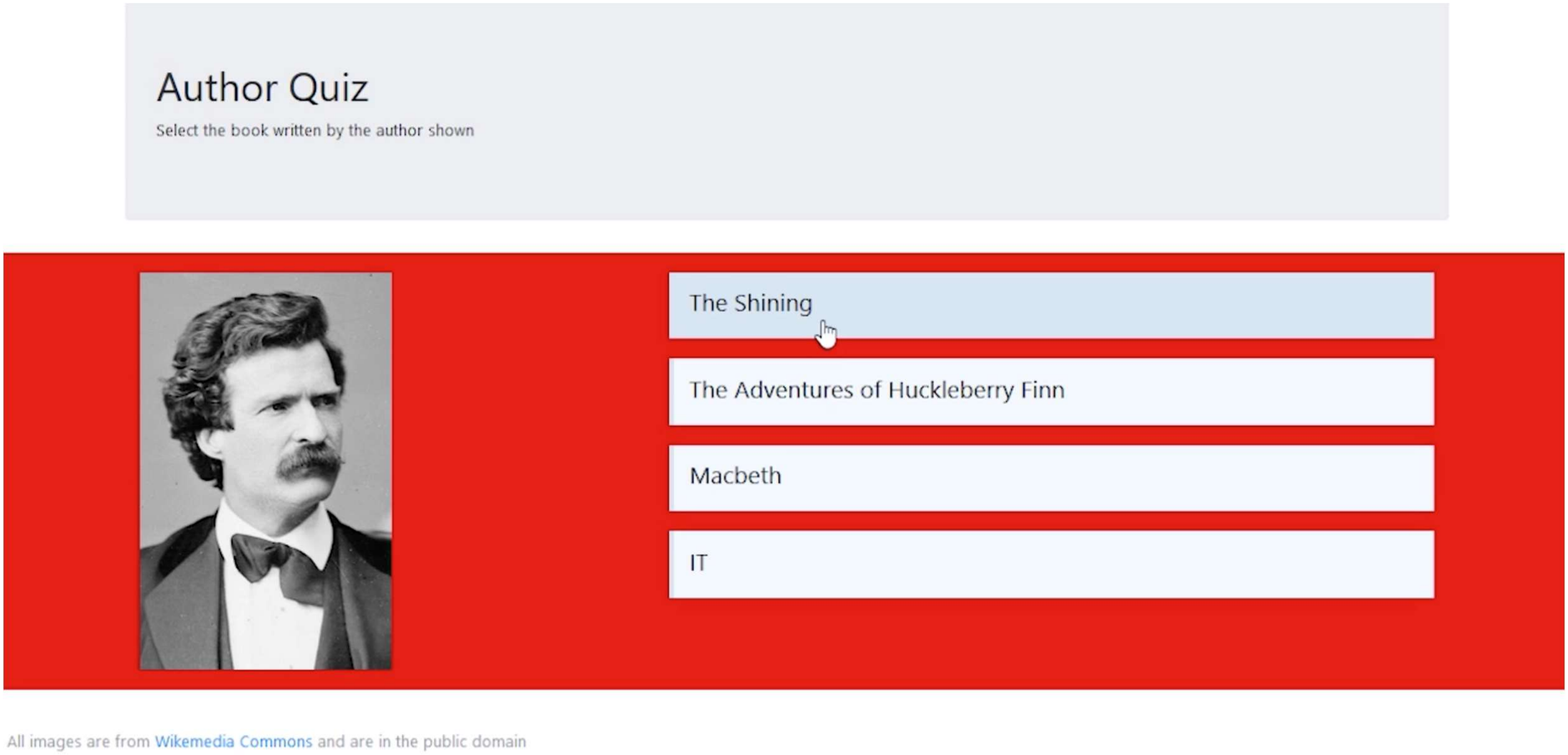
The Shining

The Adventures of Huckleberry Finn

Macbeth

IT

All images are from Wikemedia Commons and are in the public domain



Author Quiz

Select the book written by the author shown

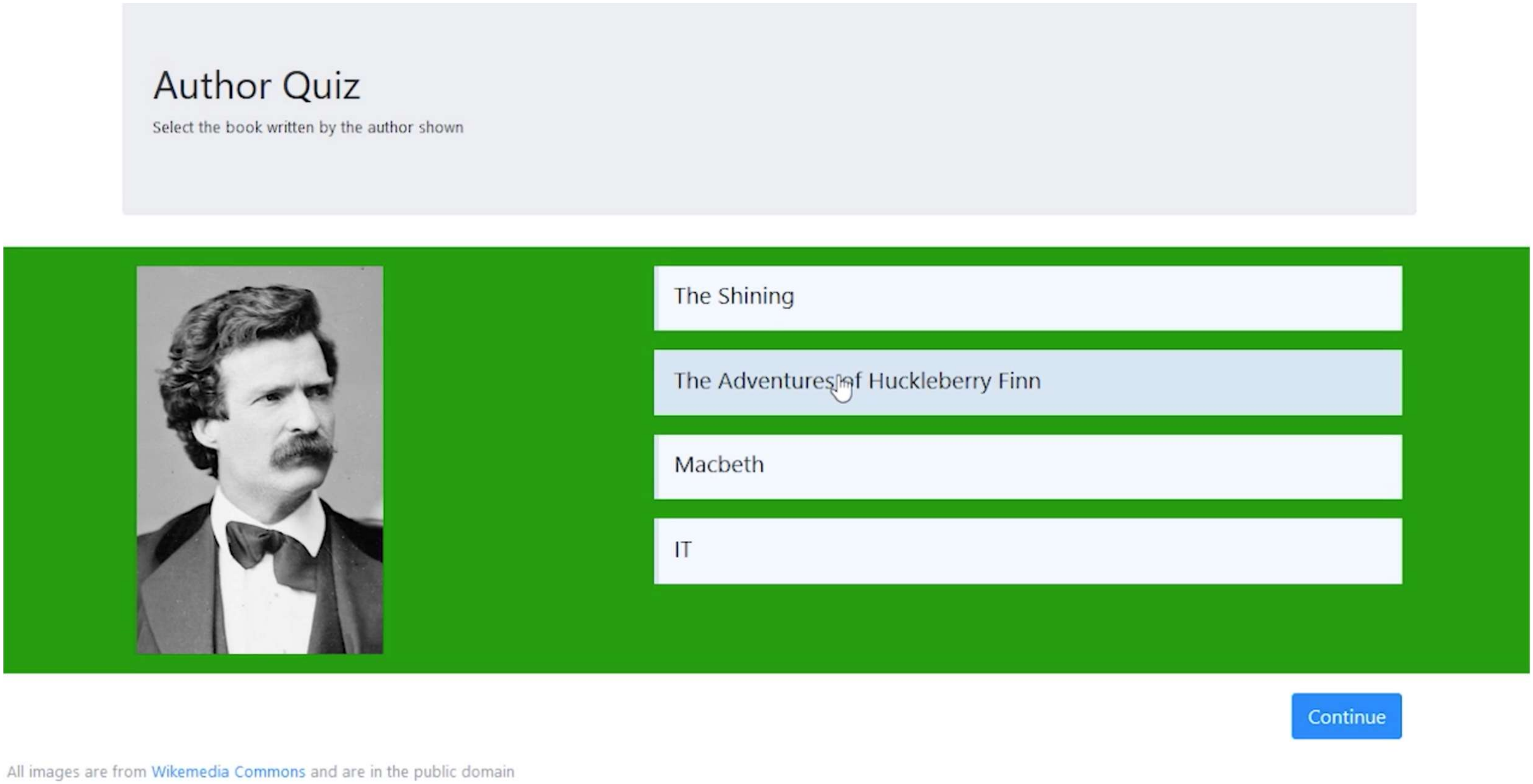
The Shining

The Adventures of Huckleberry Finn

Macbeth

IT

All images are from Wikemedia Commons and are in the public domain



Author Quiz

Select the book written by the author shown

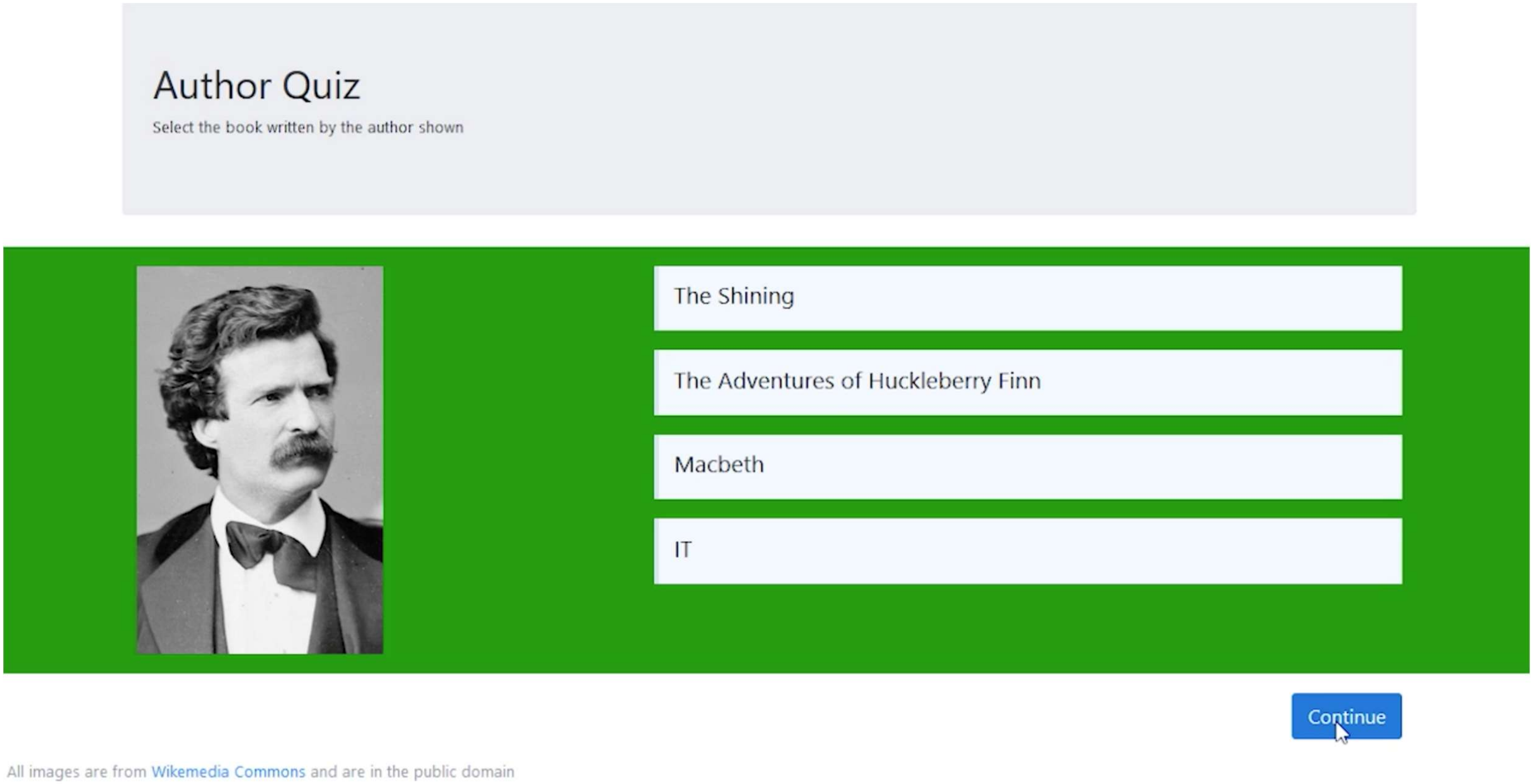
The Shining

The Adventures of Huckleberry Finn

Macbeth

IT

All images are from Wikemedia Commons and are in the public domain



Author Quiz

Select the book written by the author shown

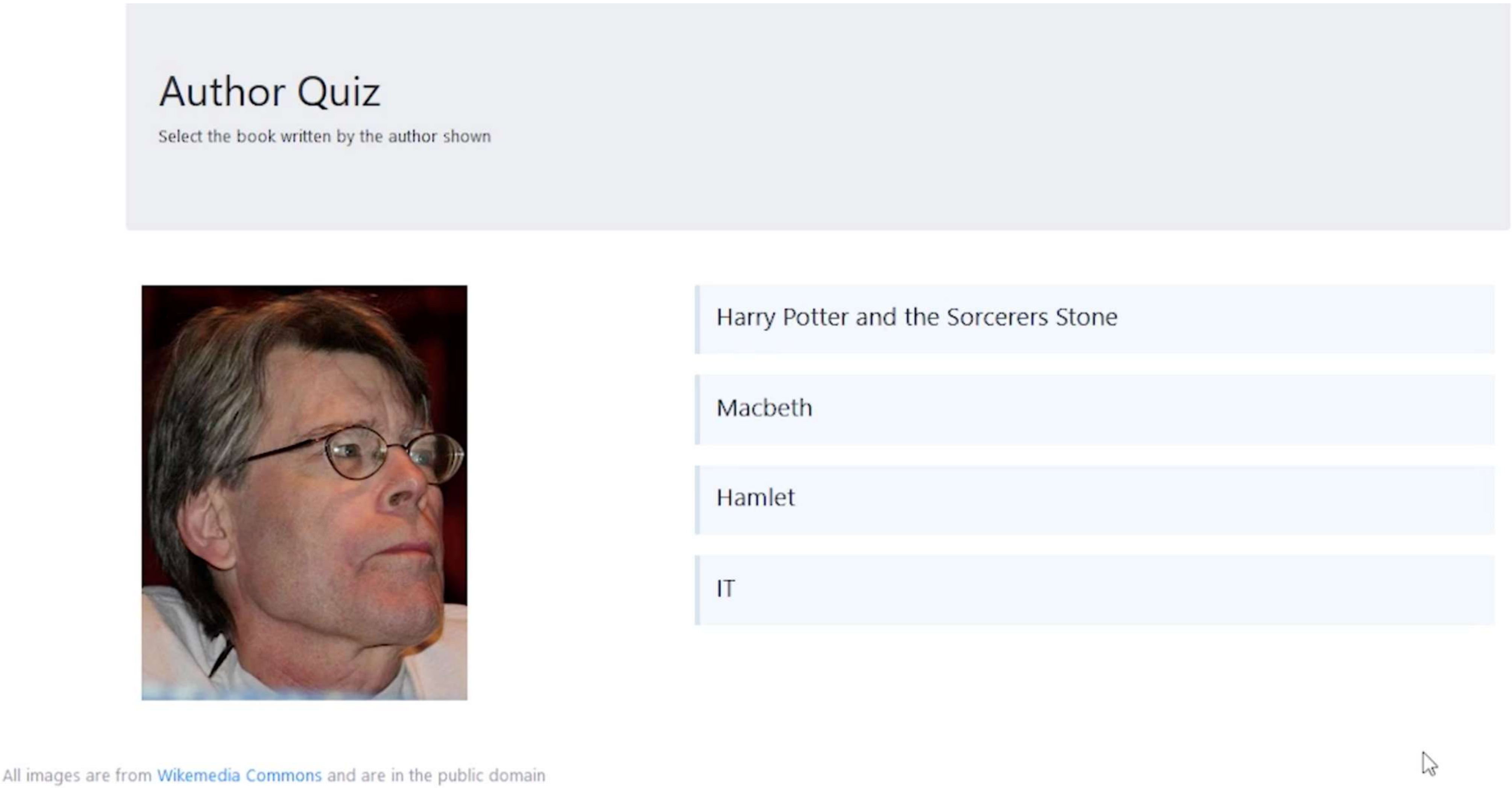
The Shining

The Adventures of Huckleberry Finn

Macbeth

IT

All images are from Wikemedia Commons and are in the public domain



Author Quiz

Select the book written by the author shown

Harry Potter and the Sorcerers Stone

Macbeth

Hamlet

IT

All images are from Wikemedia Commons and are in the public domain

```
function Hello(props) {
    return <h1>Hello at {props.now}</h1>
}
```

Defining a Component

Value return from the function -> JSX (markup language that React compiles to JavaScript)

- **Model data** is passed into the component as the argument to the function
- Curly braces are used to indicate a **JavaScript expression**
- It should be **evaluated** and interpolated into the output.
- This **output** is a piece of UI, that incorporate the model data

Rendering a Component

```
import ReactDOM from ‘react-dom’;
import React from ‘react’;

function Hello(props) {
    return <h1>Hello at {props.now}</h1>
}

ReactDOM.render(<Hello now={new Date().toISOString()} />,
    document.getElementById(‘root’)
);
```

Bootstrapping the Author Quiz

Initialize
Generate a new
application skeleton

Style
Create the basic
layout and styles

Component
Define the top-level
AuthorQuiz
component

Components: props + state

- Components can render based on data from two sources, props and state.
- Props contain immutable data passed from parent components.
- State contains local mutable data.
- Avoid using state where possible

Elements represented in JSX

```
ReactDOM.render(<div id=”mydiv”></div>,
    document.getElementById(‘root’)
);
```

```
function Sum(props) {
    return (
            <h1>{props.a} + {props.b} = {props.a + props.b}</h1>
    );
}
ReactDOM.render(<Sum a={4} b={2} />,
    document.getElementById(‘root’)
);
```

Props

```
props = {
        a: 4,
        b: 2
}
```

```
function Sum(props) {
    return (
            <h1>{props.a} + {props.b} = {props.a +
props.b}</h1>
    );
}
ReactDOM.render(<Sum a={4} b={2} />,
    document.getElementById(‘root’)
);
```

“All React components must act like pure functions with respect to their props.”

React documentation

Class Components [Deprecated?]

```
class Sum extends React.Component {
    render() {
           return (<h1>
                  {this.props.a} + {this.props.b}
                  = {this.       props.a + this.props.b}
               </h1>);
    }
}
ReactDOM.render(<Sum a={4} b={2} />,
    document.getElementById(‘root’)
);
```

Convert a function component to a class

1. Create an ES6 class, with the same name, that extends React.Component.

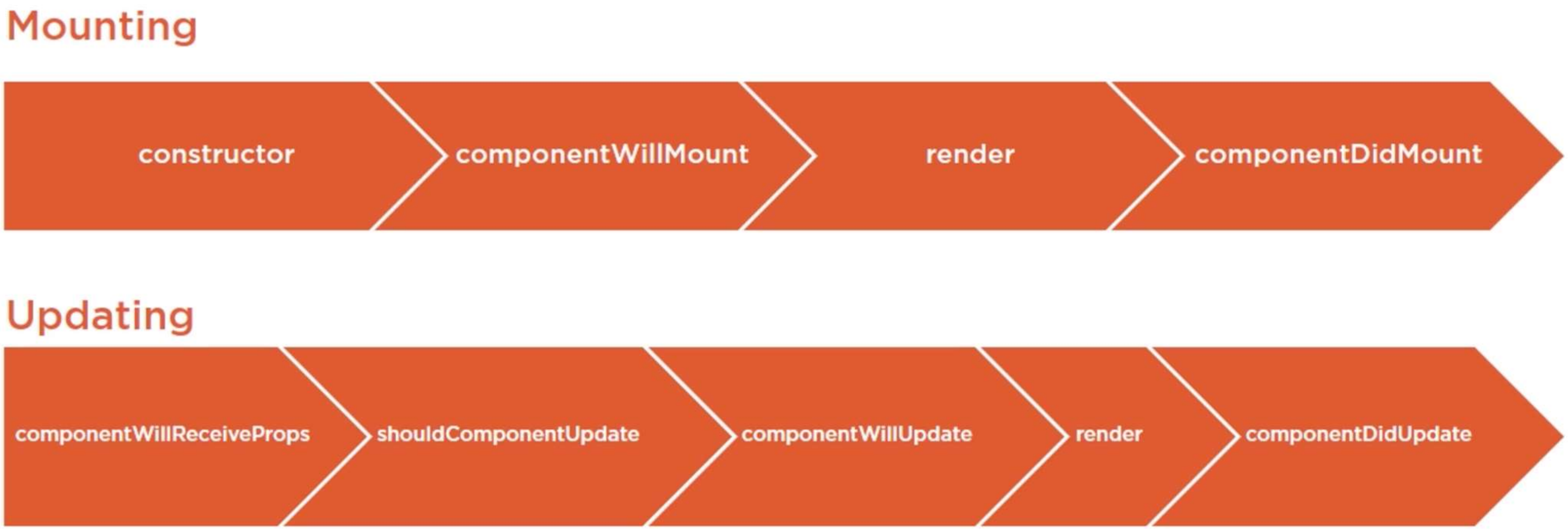
2. Add a single empty method to it called render().

3. Move the body of the function into the render() method.

4. Replace props with this.props in the render() body.

5. Delete the remaining empty function declaration.

Component Lifecycle



Mounting

constructor → componentWillMount → render → componentDidMount

Updating

componentWillReceiveProps → shouldComponentUpdate → componentWillUpdate → render → componentDidUpdate

State

Alternative component data container

State is local, mutable data

More complex

Class statefull Components

```
class ClickCounter extends React.Component {
         constructor(props) {
                   super(props);
                   this.state = {clicks: 0};
         }

         render() {
         return    <div onClick={ () =>
                   {this.setState({clicks:
              this.state.clicks + 1}); }}>
                   This div has been clicked
                   {this.state.clicks} times.
                   </div>;
     }
}
```

Functions for statefull componets

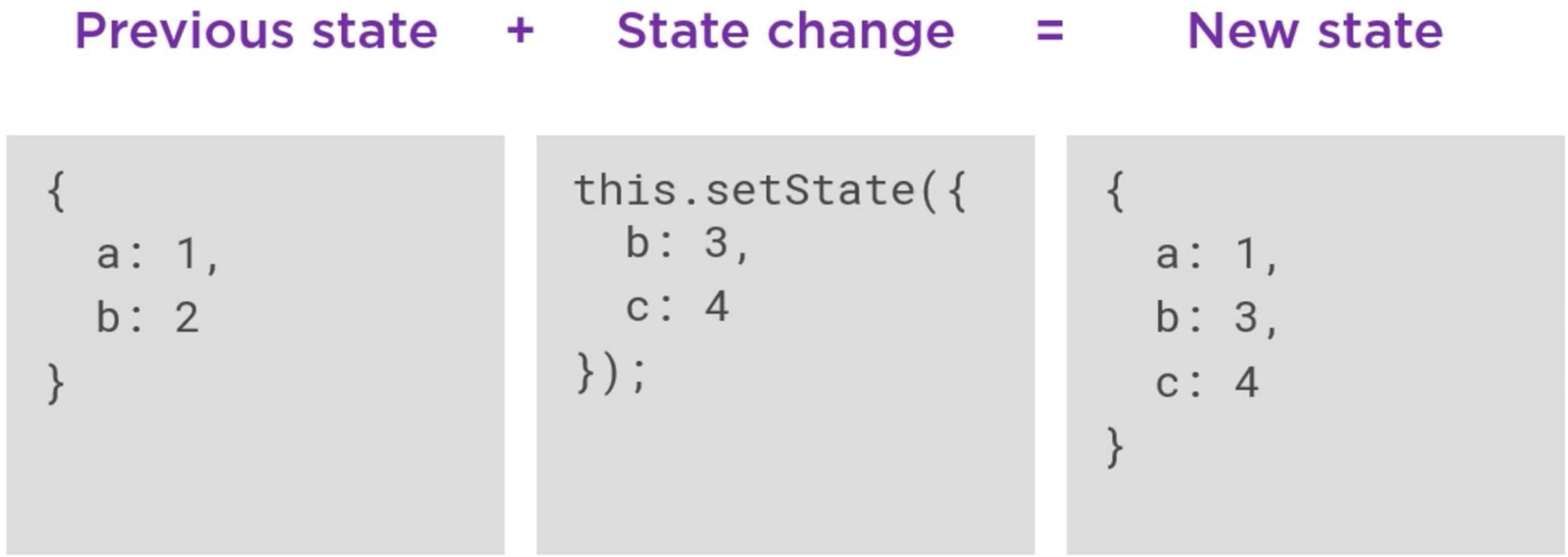
```
import React, { useState } from 'react’;

function Sample () {
  const [state, setState] = useState([])
  return (
    <div>… {state} </div>
  )
}

ReactDOM.render(<Sample />,
    document.getElementById(‘root’)
);
```

Function statefull Components

```
function App() {

  // el state representa el modelo de datos
  const [model, setModel] = useState({clicks: 0})

  // una función que permite modificar el modelo
  const clickHandler = ()=>{
    setModel({ clicks: model.clicks + 1})
  }
  return (…
     <ClickCounter clicks={model.clicks}
          onClick={clickHandler}></ClickCounter>)
  …
```

```
export function ClickCounter(props) {

    return (
      <div>
        <h2>Click Counter</h2>
        <p className="button" onClick={props.onClick}>
            This click has been clicked {props.clicks} times
        </p>
      </div>
    )
}
```

setState



Previous state + **State change** = **New state**

```
{
  a: 1,
  b: 2
}
```

```
this.setState({
  b: 3,
  c: 4
});
```

```
{
  a: 1,
  b: 3,
  c: 4
}
```

Hooks

- Special functions added in versión 16.8 (early 2019)
  - useState
  - useEffect
  - useContext
- Rules
  - Only Call Hooks at the Top Level
    - Don’t call Hooks inside loops, conditions, or nested functions.
  - Only Call Hooks from React Functions
    - Call Hooks from React function components
    - Call Hooks from custom Hooks
- https://reactjs.org/docs/hooks-intro.html

Prop Types

```
function Sum(props) {
    return (
            <h1>{props.a} + {props.b} = {props.a +
props.b}</h1>
    );
}
ReactDOM.render(<Sum a={4} b={2} />,
    document.getElementById(‘root’)
);
```

Prop Types

```
function Sum(props) {
    return (
            <h1>{props.a} + {props.b} = {props.a +
props.b}</h1>
    );
}
ReactDOM.render(<Sum a={“key”} b={“board”} />,
    document.getElementById(‘root’)
);
```

Prop Types

```
function Sum(props) {
    return (
           <h1>{props.a} + {props.b} = {props.a +
props.b}</h1>
    );
}
ReactDOM.render(<Sum a={“a”} b={2} />,
    document.getElementById(‘root’)
);
```

Prop Validation: PropTypes

```
import PropTypes from ‘prop-types’;
function Sum(props) {
    return (
           <h1>{props.a} + {props.b} = {props.a +
props.b}</h1>
    );
}
Sum.propTypes = {
    a: PropTypes.number.isRequired,
    b: PropTypes.number.isRequired
}
ReactDOM.render(<Sum a={“a”} b={2} />,
    document.getElementById(‘root’)
);
```

TypeScript and Flow

```
interface SumProps{
    a: number;
    b: number;
}
function Sum(props: SumProps) {
    return (
            <h1>{props.a} + {props.b} = {props.a +
props.b}</h1>
    );
}
ReactDOM.render(<Sum a={4} b={2} />,
    document.getElementById(‘root’)
);
```

Demo

Setting up a React development environment with Typescript

```
npx create-react-app <project>
--template typescript
```

What is JSX?

Supports xml-like syntax in JavaScript

Each element is transformed into a JavaScript function call

JSX

```
<Sum a={4} b={3} />
```

JavaScript

```
React.createElement(
        Sum,
        {a: 4, b: 3},
        null
)
```

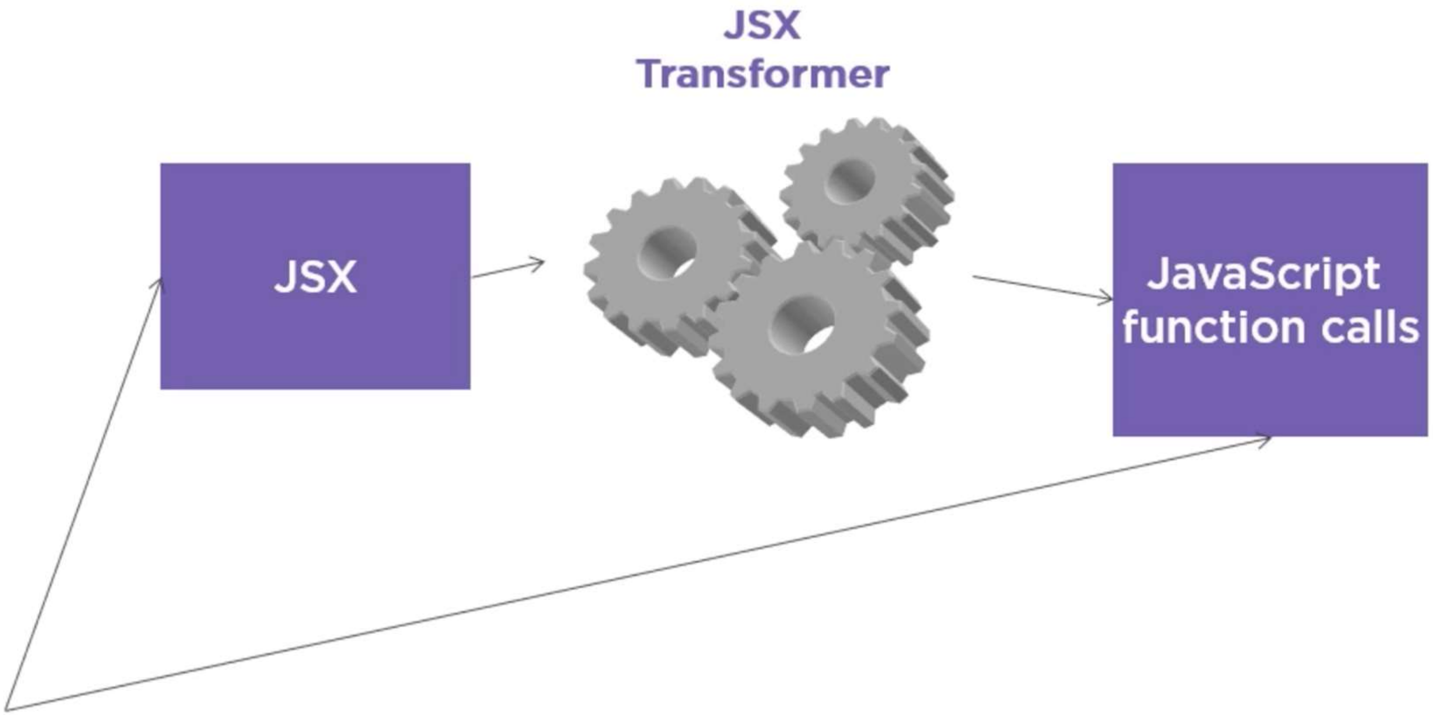
JSX

```
<h1>
        <Sum a={4} b={3} />
</h1>
```

JavaScript

```
React.createElement(
        h1,
        null,
        React.createElement(
                Sum,
                {a: 4, b: 3},
                null
        )
)
```

Not Using JSX



JSX

JSX Transformer

JavaScript function calls

```
<Hello now={new Date().toISOString()} />
```

Props in JSX

JSX attributes become component props

```
<Hello now={new Date().toISOString()} />

<Hello now=”Literal string value” />
```

Props in JSX

JSX attributes become component props

```
const props = {a: 4, b: 2};
const element = <Sum {...props} />
```

Props in JSX

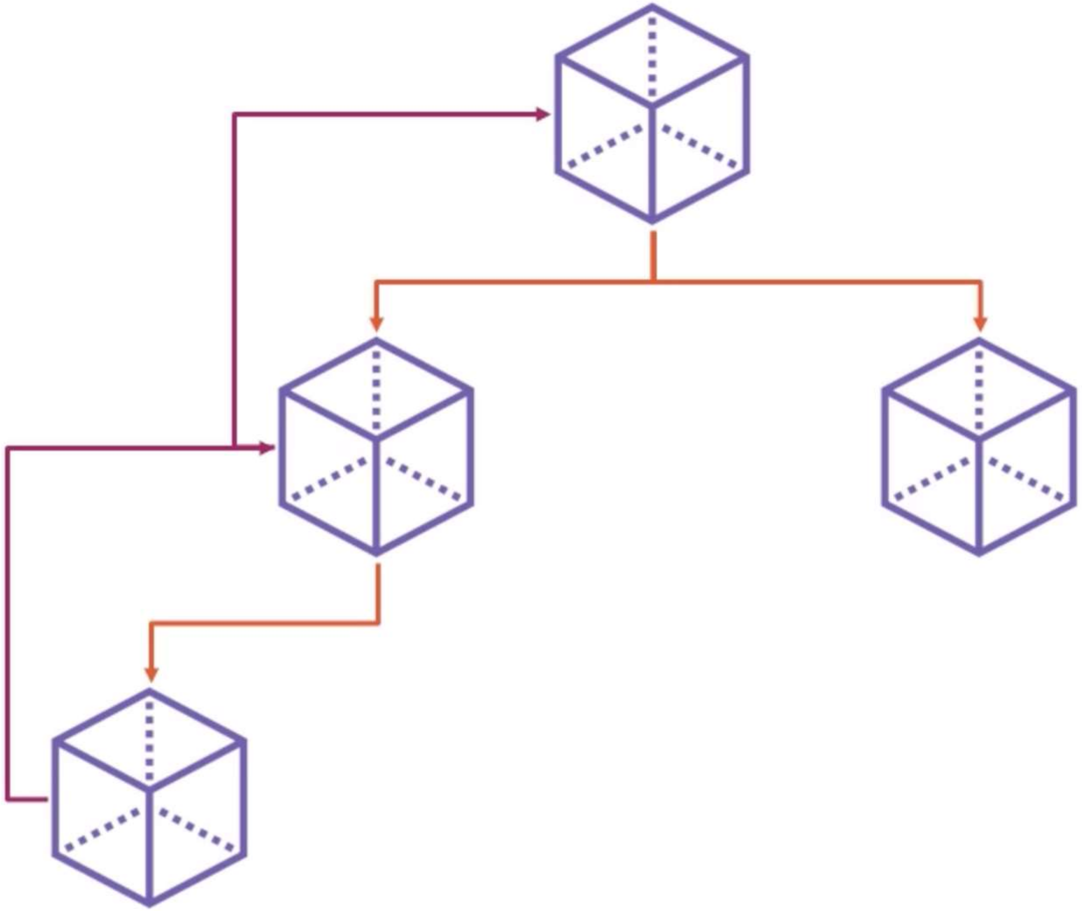
Spread Attributes

```
function Clicker({handleClick}) {
    return <button onClick={(e) => {handleClick(‘A’);}}>A</button>
}
const el = <Clicker handleClick={(letter) => {log(letter);}} />
```

Props in JSX

Events

React Data Flow



JSX

```
<label
        htmlFor=”name”
        className=”highlight”
        style={{
                backgroundColor:
                “yellow”
        }}
>
        Foo Bar
</label>
```

HTML

```
<label
        for=”name”
        class=”highlight”
        style=”background-
color:
                                yellow”
>
    Foo Bar
</label>
```

```
<div dangerouslySetInnerHTML={{__html=”<p>foo</p>}} />
```

Unescaping Content

React escapes content by default

```
<Hello>
    <First />
    <Second />
</Hello>
```

Child Expressions and Elements

JSX elements can be nested

`props.children`

Child Expressions and Elements

JSX elements can be nested

Child Expressions and Elements

```
function ConditionalDisplay(props) {
    return <div>
           {props.isVisible ? props.children : null}
    </div>;
}

ConditionalDisplay.propTypes = {
    isVisible: PropTypes.bool.isRequired
}
```

Child Expressions and Elements

```
<ConditionalDisplay isVisible={state.showSum}>
    <h1>A <span>Sum</span></h1>
    <Sum a={4} b={2} />
</ConditionalDisplay>
```

Form Elements

Just like HTML

Preserve React’s rendering semantics

```
<input type=”text” value=”react” />
```

Text Input

HTML

```
<textarea>
       Foo Bar
</textarea>
```

React

```
<textarea value=”Foo Bar” />
```

HTML

```
<select>
        <option
value=”saturday”>
                Saturday
        </option>
        <option value=”sunday”>
                Sunday
        </option>
</select>
```

React

```
<select value=”sunday”>
        <option
value=”saturday”>
                Saturday
        </option>
        <option value=”sunday”>
                Sunday
        </option>
</select>
```

Allowing User Input

Form elements are read-only

Component state supports editing

A Read-Only Form

```
Class Identity extends React.Component {
        render() {
        return (
        <form>
                <input type=”text” value=”” placeholder=”First
name” />
                <input type=”text” value=”” placeholder=”Last name”
/>
        </form>
        );
    }
}
```

Allowing User Input

Add state to the component

Bind inputs to the component state

Use onChange handler to update state

A Read-Only Form

```
Class Identity extends React.Component {
    constructor() {
        super();
        this.state = {
            firstName: “”,
            lastName: “”
        };
        this.onFieldChange = this.onFieldChange.bind(this);
    }
    onFieldChange(event) {
        this.setState({
            [event.target.name]: event.target.value
        });
    }
// ...
```

A Read-Only Form

```
// Now, update the old form controls

<input type=”text” placeholder=”First name”
       name=”firstName”
       value={this.state.firstName}
       onChange={this.onFieldChange}
/>
```

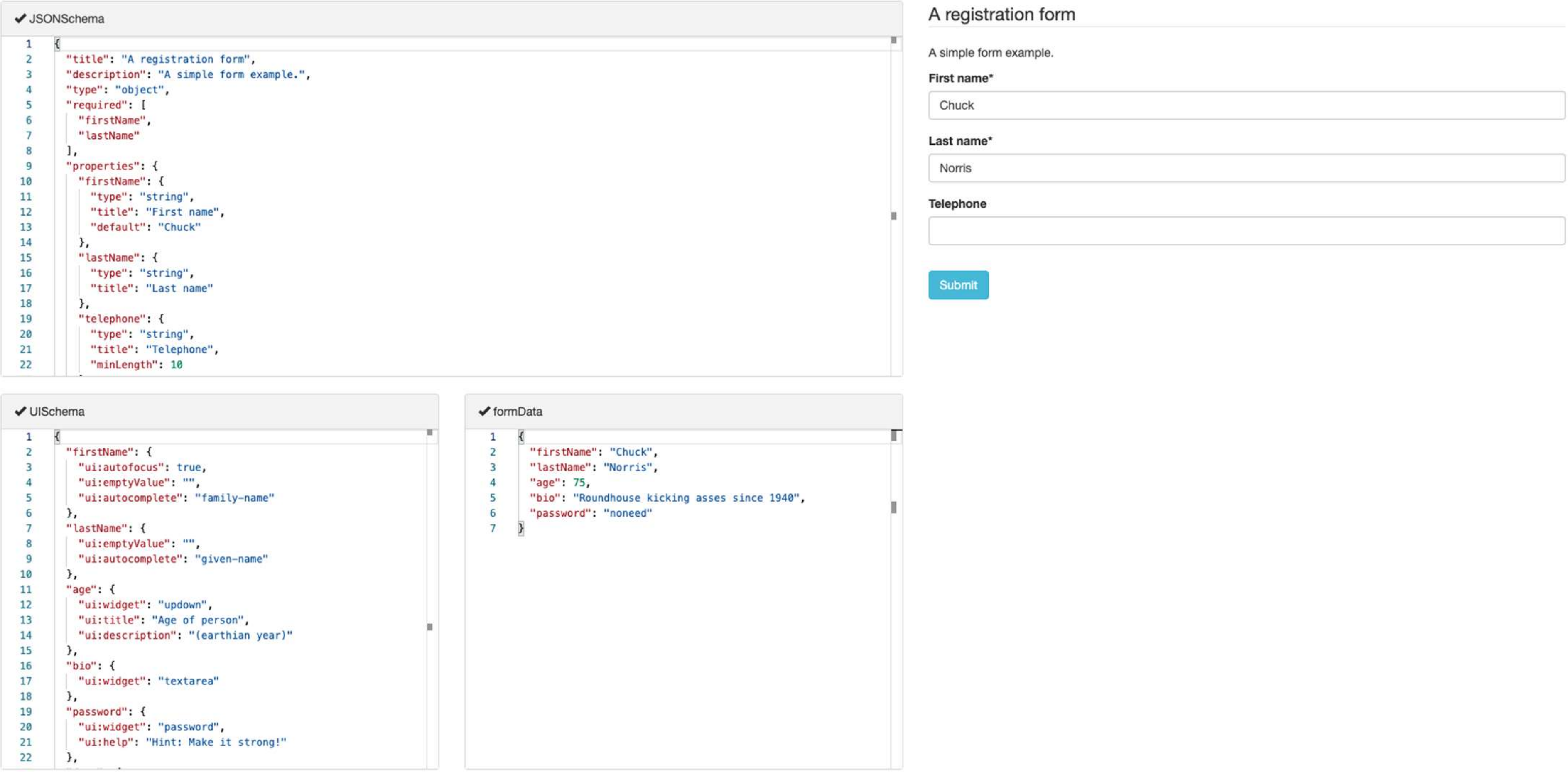
React Forms

Forms can be time consuming

Many form libraries are available

Increase productivity, decrease control

JSON Schema
https://rjsf-team.github.io/react-jsonschema-form/



```
{
  "title": "A registration form",
  "description": "A simple form example.",
  "type": "object",
  "required": [
    "firstName",
    "lastName"
  ],
  "properties": {
    "firstName": {
      "type": "string",
      "title": "First name",
      "default": "Chuck"
    },
    "lastName": {
      "type": "string",
      "title": "Last name"
    },
    "telephone": {
      "type": "string",
      "title": "Telephone",
      "minLength": 10
```

```
{
  "firstName": {
    "ui:autofocus": true,
    "ui:emptyValue": "",
    "ui:autocomplete": "family-name"
  },
  "lastName": {
    "ui:emptyValue": "",
    "ui:autocomplete": "given-name"
  },
  "age": {
    "ui:widget": "updown",
    "ui:title": "Age of person",
    "ui:description": "(earthian year)"
  },
  "bio": {
    "ui:widget": "textarea"
  },
  "password": {
    "ui:widget": "password",
    "ui:help": "Hint: Make it strong!"
  },
```

```
{
  "firstName": "Chuck",
  "lastName": "Norris",
  "age": 75,
  "bio": "Roundhouse kicking asses since 1940",
  "password": "noneed"
}
```

A registration form

A simple form example.

First name*	Chuck
Last name*	Norris
Telephone	

Form Validation

Use a form library or implement

Validate on change or on submission

Display errors inline or aggregated elsewhere

Refs

Access DOM elements

Use React.createRef()

Refs

```
class Identity extends React.Component {
    constructor() {
           super();
           this.myDiv = React.createRef();
    }
}
```

Refs

```
render() {
    return <div ref=”{this.myDiv}>
           {“Set in render <strong>Safe</strong>”}
    </div>;
}
```

Refs

```
componentDidMount() {
    this.myDiv.current.innerHTML += “<br /> Set on the wrapped
DOM element. <strong>Unsafe</strong>”;
}
```

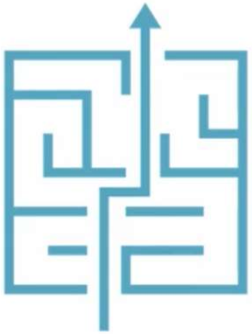
Client-Side Routing with HTML5 pushState

pushState allows JavaScript to update the browser URL

Uses proper URLs

Requires server-side support

React Router

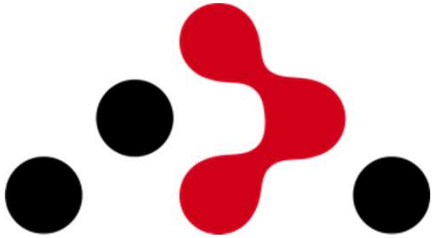


Client side router for React

Conditional rendering based on routes

Supports path updates

React Router Dom



Popular routing library (react-router-dom)

Declare routes via React components

- “Load this component for this URL”

Key Components

Router	Wrap app entry point
Route	“Load this component for this URL
Link	Anchors

Pick a Router

#about

HashRouter

Places hashes in the URL

/about

BrowserRouter

Uses HTML5 History API for clean URLS

No URL

MemoryRouter

Useful for testing and React Native

Demo

React Router route configuration

Links (Anchors)

Target URL /hero/14

Route <Route path=”/hero/:heroId” component={Hero} />

JSX <Link to=”/hero/14”>Bombasto</Link>

Anchor <a href=”/hero/14”Bombasto</a>

404 Page

```
<Switch>
    <Route path=”/” exact component=”{Home}” />
    <Route path=”/about” component=”{About}” />
    <Route component=”{PageNotFound}” />
</Switch>
```

Redirects

Need to change the URL? Use a Redirect.

```
<Redirect to=”/heroes” />
```

Redirects

Need to change the URL? Use a Redirect.

```
{ this.state.redirectToUsers && <Redirect to=”/heroes” /> }
```

Redirects

Need to change the URL? Use a Redirect.

```
<Redirect from=”old-path” to=”new-path” />
```

Programmatic Redirect

```
props.history.push(‘new/path’);
```

URL Parameters

```
// Given a route like this
<Route path=”/hero/:heroName” component={HeroDetail} />
```

URL Parameters

```
// Given a route like this
<Route path=”/hero/:heroName” component={HeroDetail} />

// And a URL like this
ToH.com/hero/bombasto?level=60
```

URL Parameters

```
// Given a route like this
<Route path=”/hero/:heroName” component={HeroDetail} />

// And a URL like this
ToH.com/hero/bombasto?level=60

// Props will be
Function HeroDetail(props) {
    props.match.params.heroName; // bombasto
    props.location.query; // {level: 60}
    Props.location.pathname; // /hero/bombasto?level=60
```

URL Parameters

```
// Given a route like this
<Route path=”/hero/:heroName” component={HeroDetail} />

// And a URL like this
ToH.com/hero/bombasto?level=60

// Props will be
Function HeroDetail(props) {
   props.match.params.heroName; // bombasto
   props.location.query; // {level: 60}
   Props.location.pathname; // /hero/bombasto?level=60
```

Page Transitions

```
<Prompt
    when=”{isBlocking}”
    message=”Are you sure you want to navigate away?”
/>
```

Flux Implementations

Flux

Facebook's Flux

Alt

Delorean

Reflux

NuclearJS

Flummox

Marty

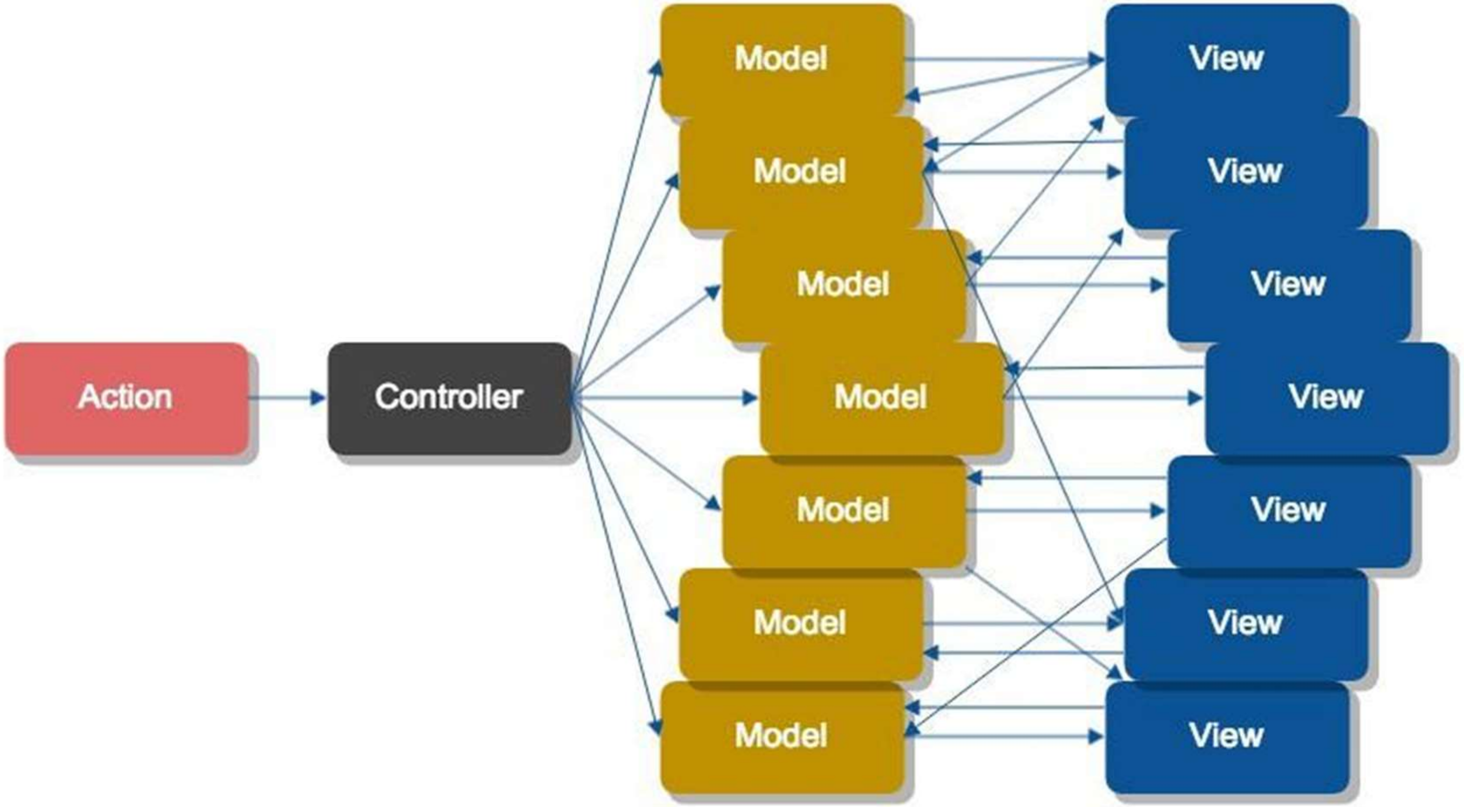
Fluxxor

Fluxible

Redux

They call it Flux for a reason

Good Luck Debugging This



Flux Implementations

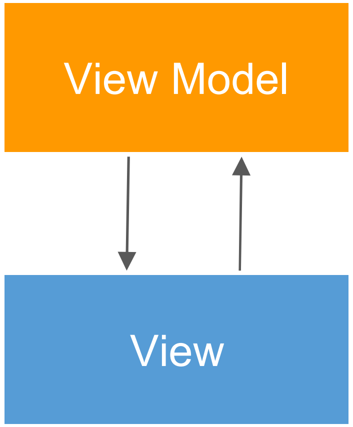
A pattern

Centralized dispatcher

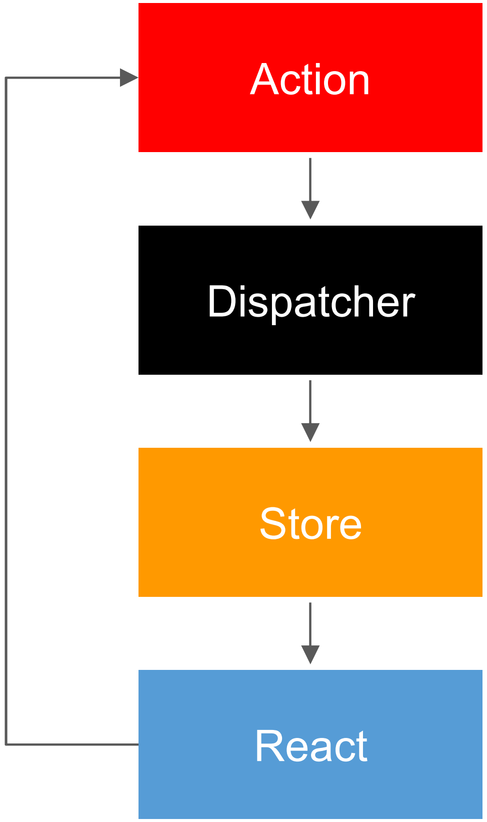
Unidirectional data flows

Two-way Bindings vs Unidirectional

Two-way Binding

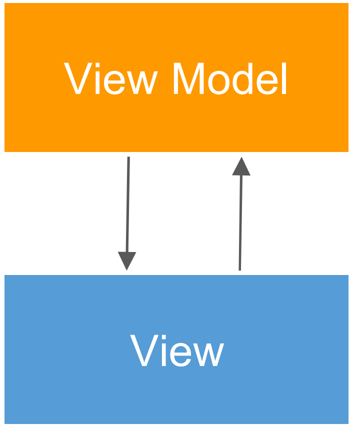


Unidirectional



Two-way Bindings vs Unidirectional

Two-way Binding

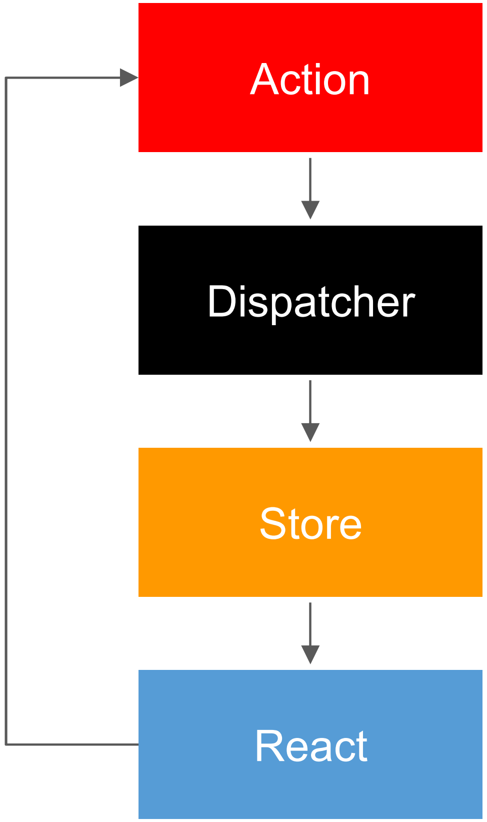


View Model

View

Unidirectional

Benefits:
Clear
Testeable
Scalable
Predictable



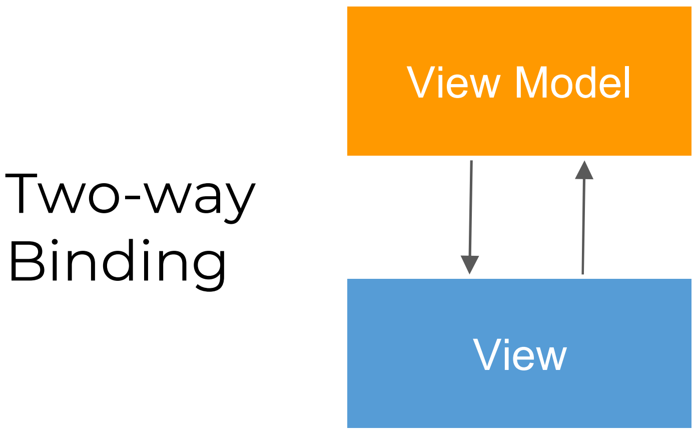
Action

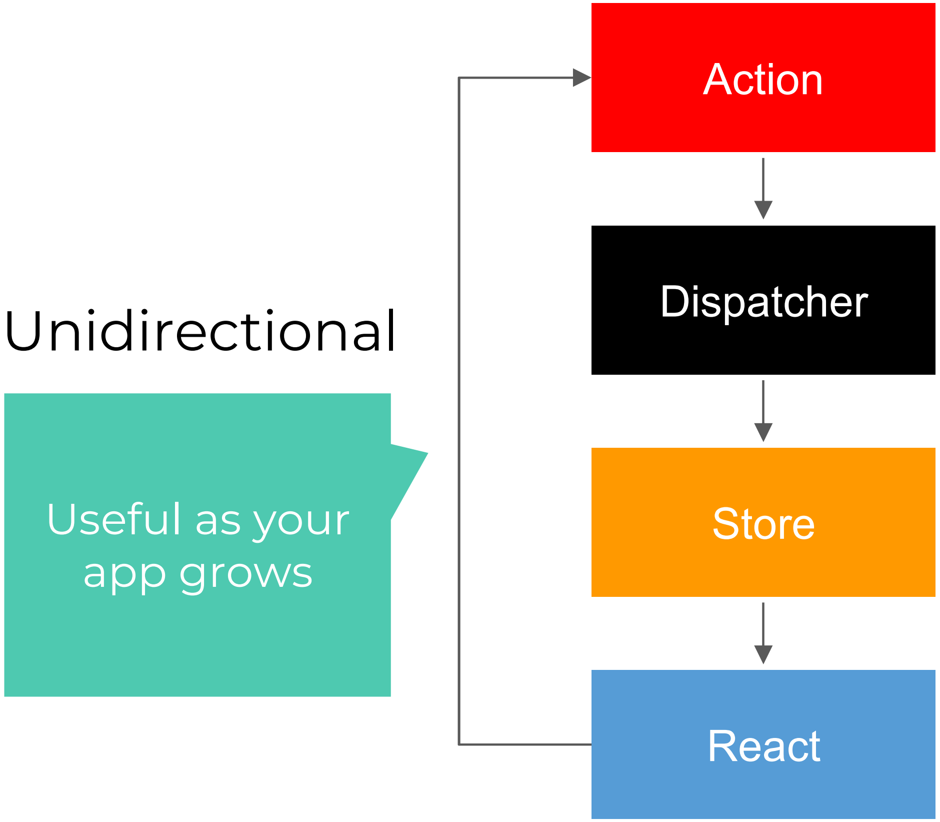
Dispatcher

Store

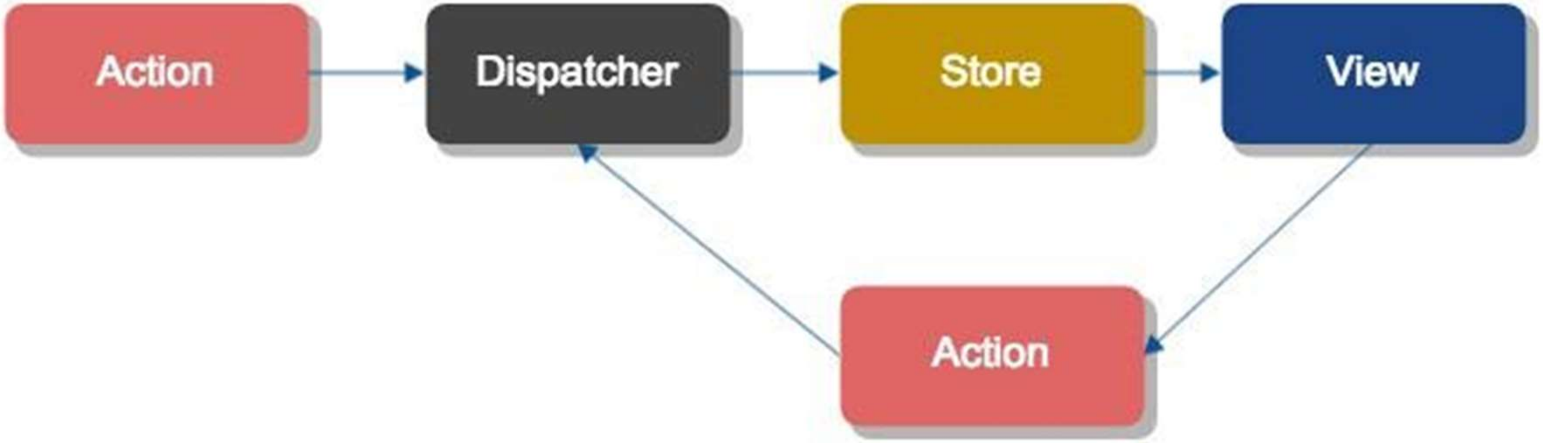
React

Two-way Bindings vs Unidirectional

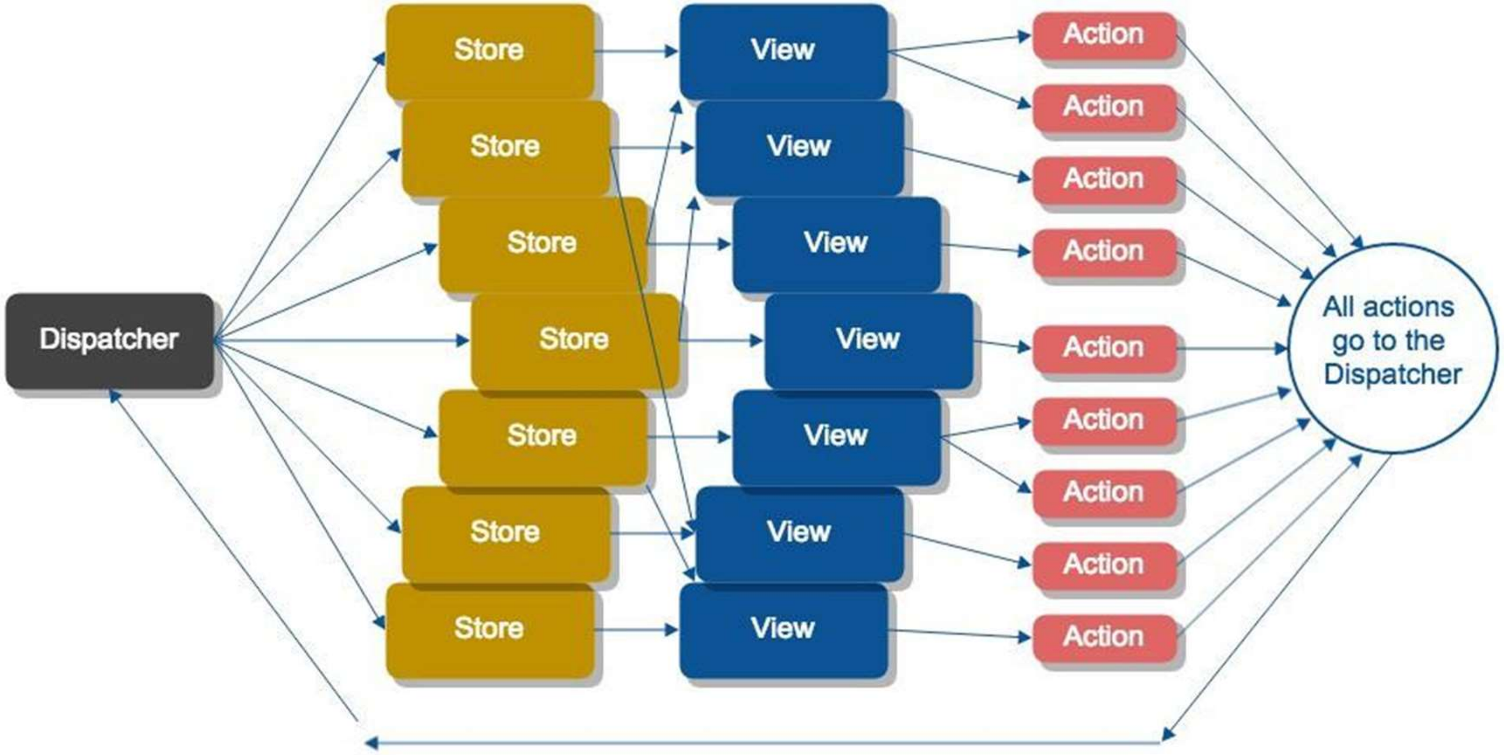




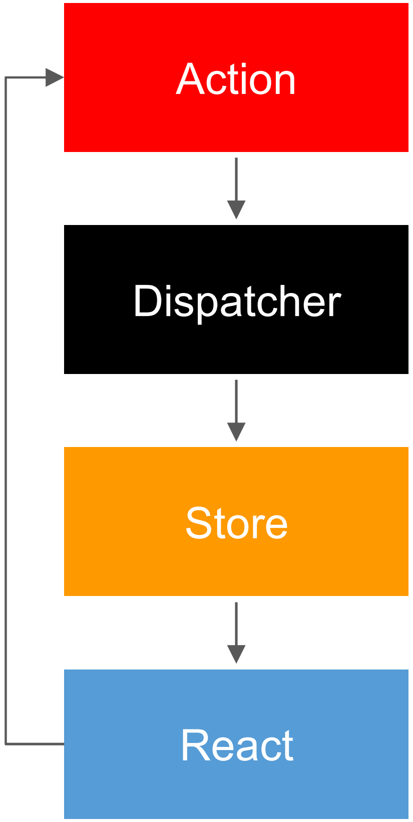
Unidirectional data flow as a solution



Unidirectional data flow as a solution



Flux: 3 Parts



Action

Dispatcher

Store

React

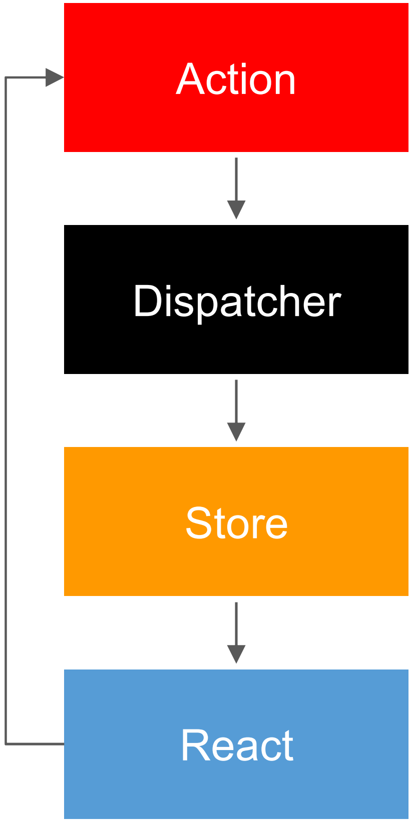
Delete hero

Notify everyone who cares

Holds app state and logic

Holds data in state

Actions

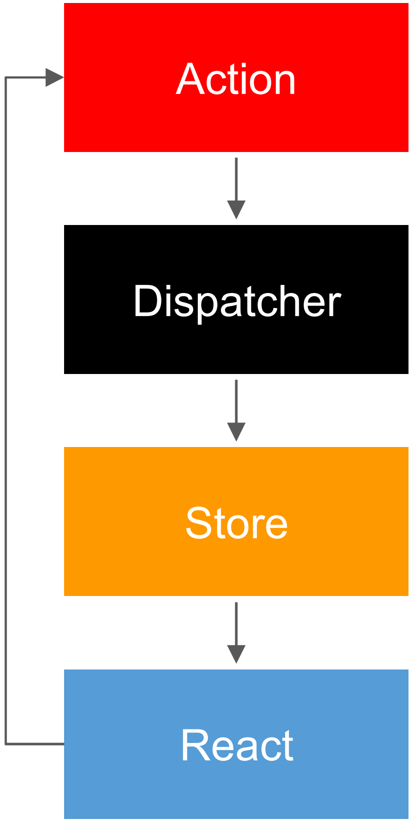


Encapsulate events

Triggered by user interactions and server

Passed to dispatcher

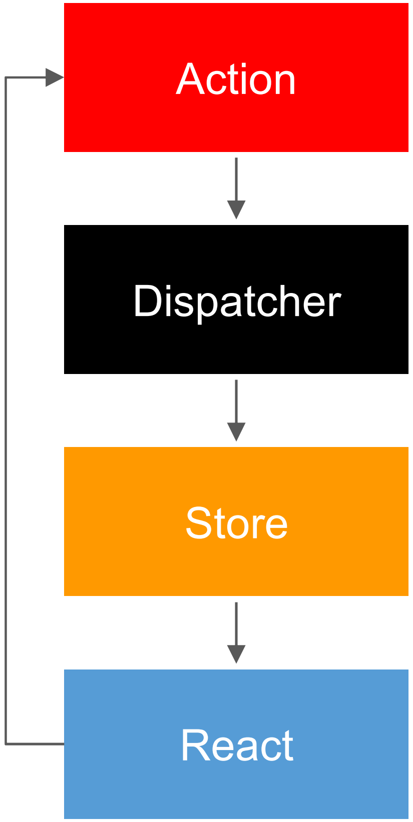
Actions



Payload has a type and data

```
{
        type: “HERO_SAVED”
        data: {
                heroId: 14,
                heroName: Bombasto
        }
}
```

Actions

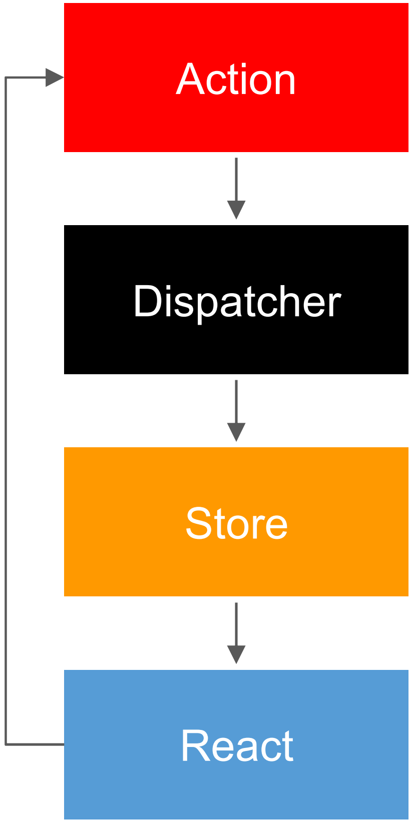


Payload has a type and data

Only the type property is required

```
{
          type: “HERO_SAVED”
          data: {
                    heroId: 14,
                    heroName: Bombasto
          }
}
```

Dispatcher



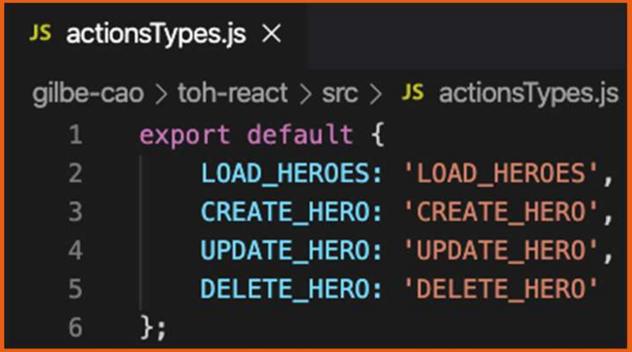
Central Hub - There’s only one

Holds a list of callbacks

Broadcasts payload to registered callbacks

Sends actions to stores

Constants



JS actionsTypes.js ×

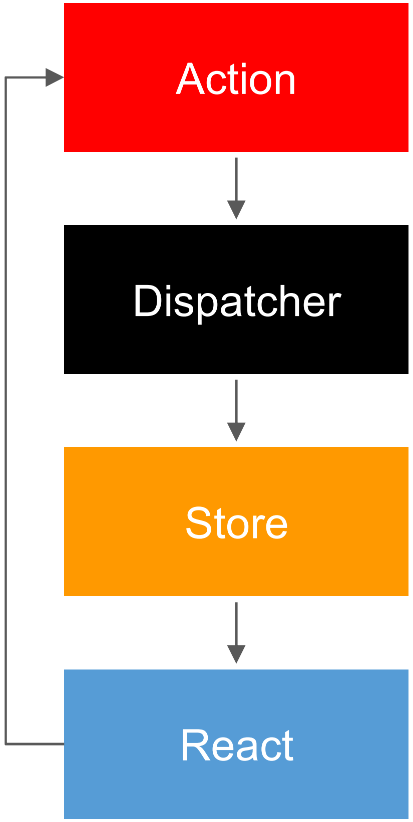
gilbe-cao > toh-react > src > JS actionsTypes.js

```
export default {
    LOAD_HEROES: 'LOAD_HEROES',
    CREATE_HERO: 'CREATE_HERO',
    UPDATE_HERO: 'UPDATE_HERO',
    DELETE_HERO: 'DELETE_HERO'
};
```

Keeps things organized

Provides high level view of what the app actually does

Store



Holds app state, logic, data retrieval

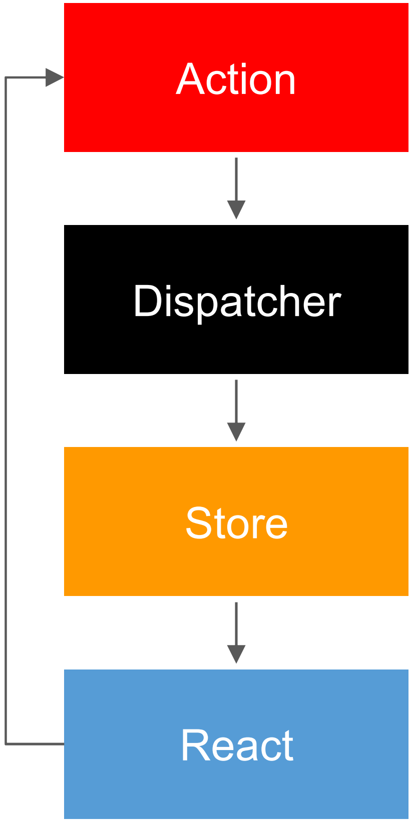
Not a model - *Contains* models

One, or many

Registers callbacks with dispatcher

Uses Node’s EventEmitter

Store



Holds app state, logic, data retrieval

Not a model - *Contains* models

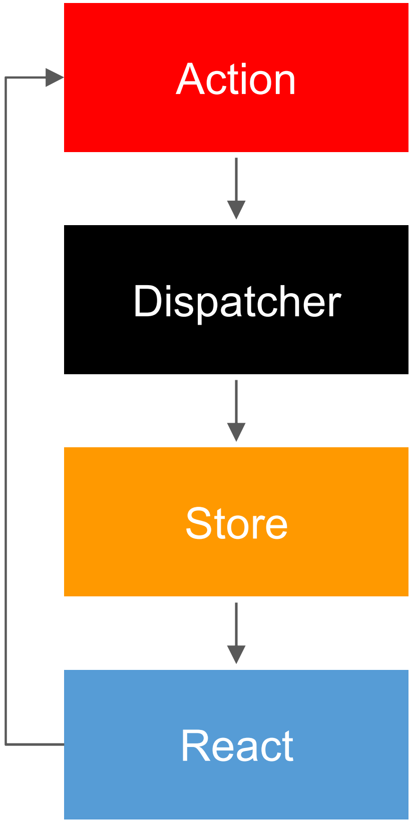
One, or many

Registers callbacks with dispatcher

Uses Node’

Hey dispatcher, when an action occurs, let me know.

Store



Holds app state, logic, data retrieval

Not a model - *Contains* models

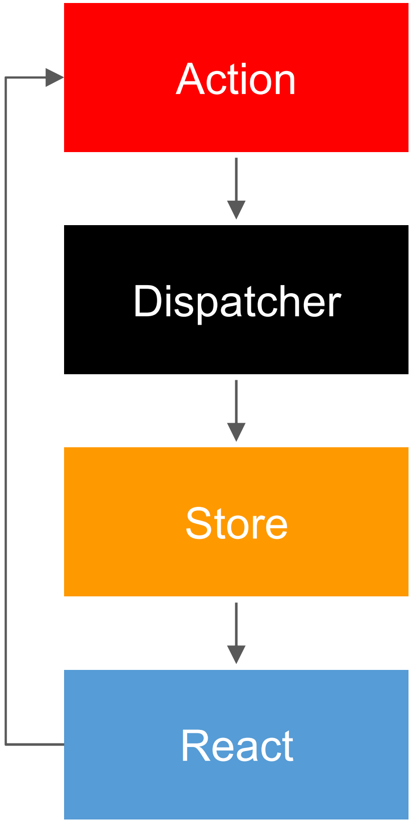
One, or many

Registers callbacks with dispatcher

Uses Node's EventEmitter

Only the store
can update data

Store



Holds app state, logic, data retrieval

Not a model - *Contains* models

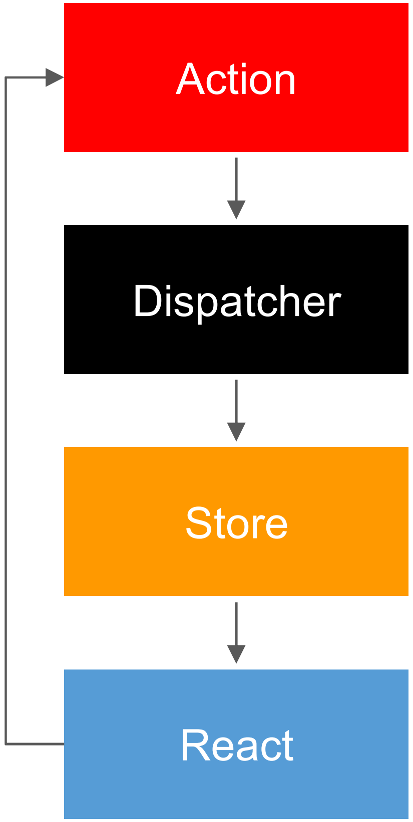
One, or many

Registers callbacks with dispatcher

Uses Node’s EventEmitter

When stores update, they emit a change event so React gets the new data

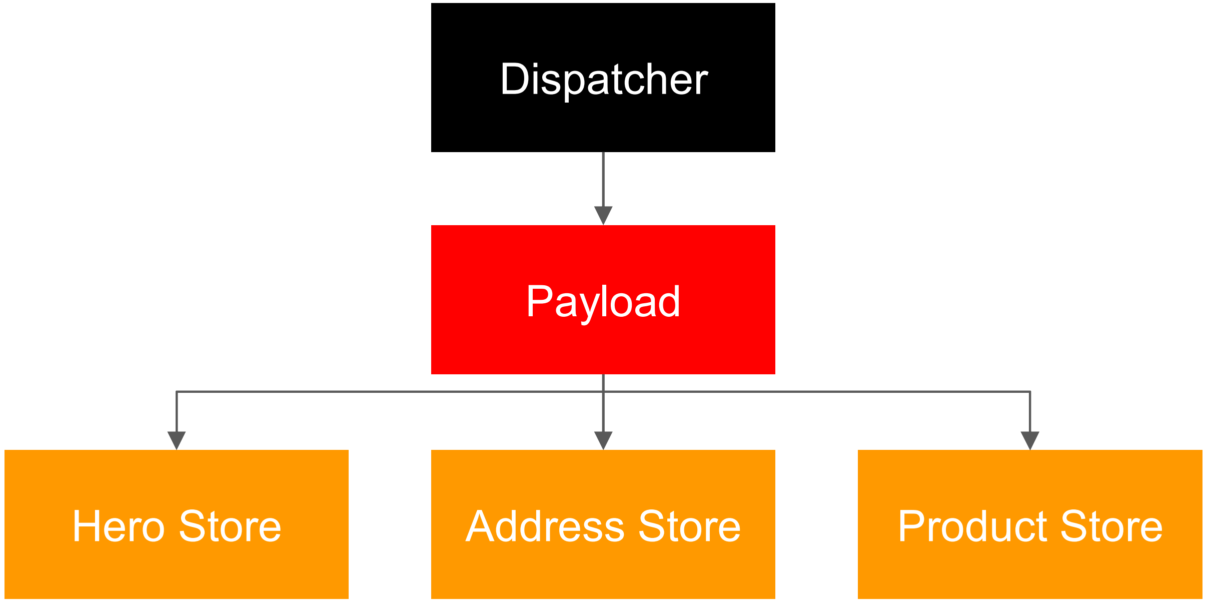
The Structure of a Store



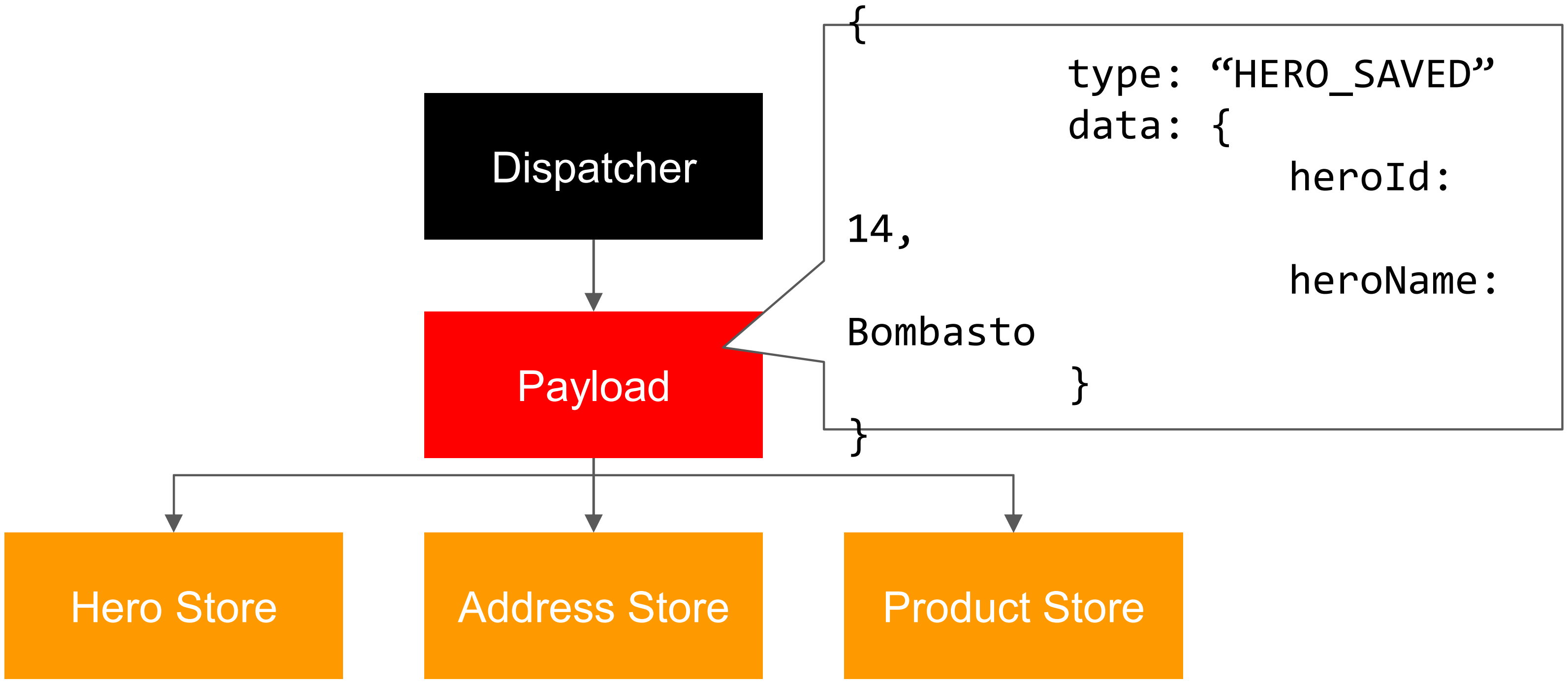
Every store has these common traits (aka interface)

1. Extend EventEmitter
2. addChangeListener and removeChangeListener
3. emitChange

The Structure of a Store



The Structure of a Store



Dispatcher

Payload

```
{
              type: “HERO_SAVED”
              data: {
                        heroId:
14,
                        heroName:
Bombasto
              }
}
```

Hero Store

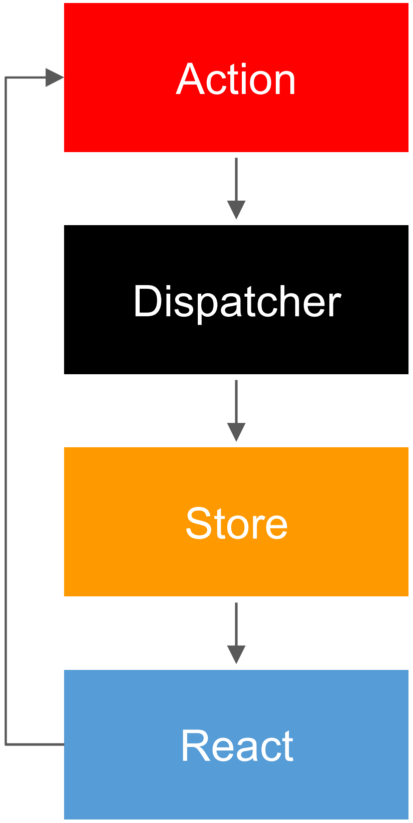
Address Store

Product Store

> As an app grows, the dispatcher becomes more vital, as it can be used to manage dependencies between the stores by invoking the registered callbacks in a specific order. Stores can declaratively wait for other stores to finish updating, and then update themselves accordingly.

Flux documentation

Controller Views



Top level component

Interacts with Stores

Holds data in state

Sends data to children as props

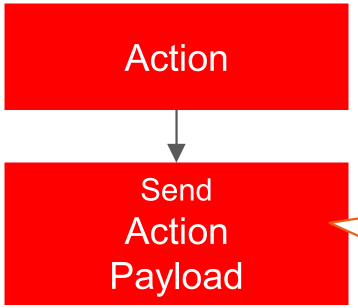
Flux Flow

Action

User clicked “Save Hero” button…

Flux Flow

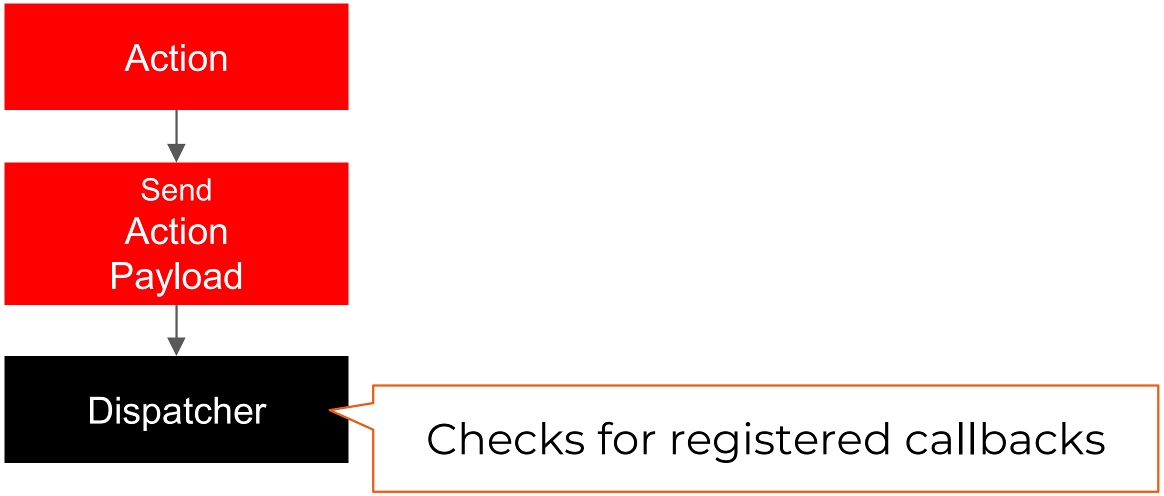
```
{
          type: “HERO_SAVED”
          data: {
                    heroId:
14,
                    heroName:
Bombasto
          }
}
```



Payload sent to dispatcher

Flux Flow

```
{
          type: “HERO_SAVED”
          data: {
                    heroId:
14,
                    heroName:
Bombasto
          }
}
```



Action

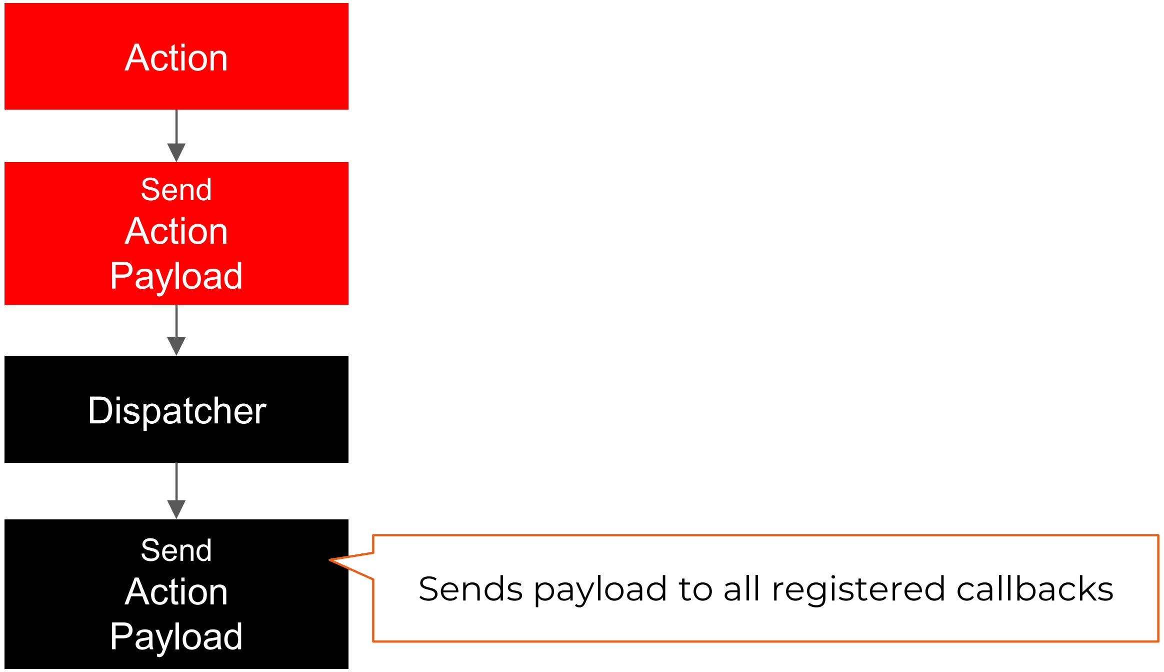
Send
Action
Payload

Dispatcher

Checks for registered callbacks

Flux Flow

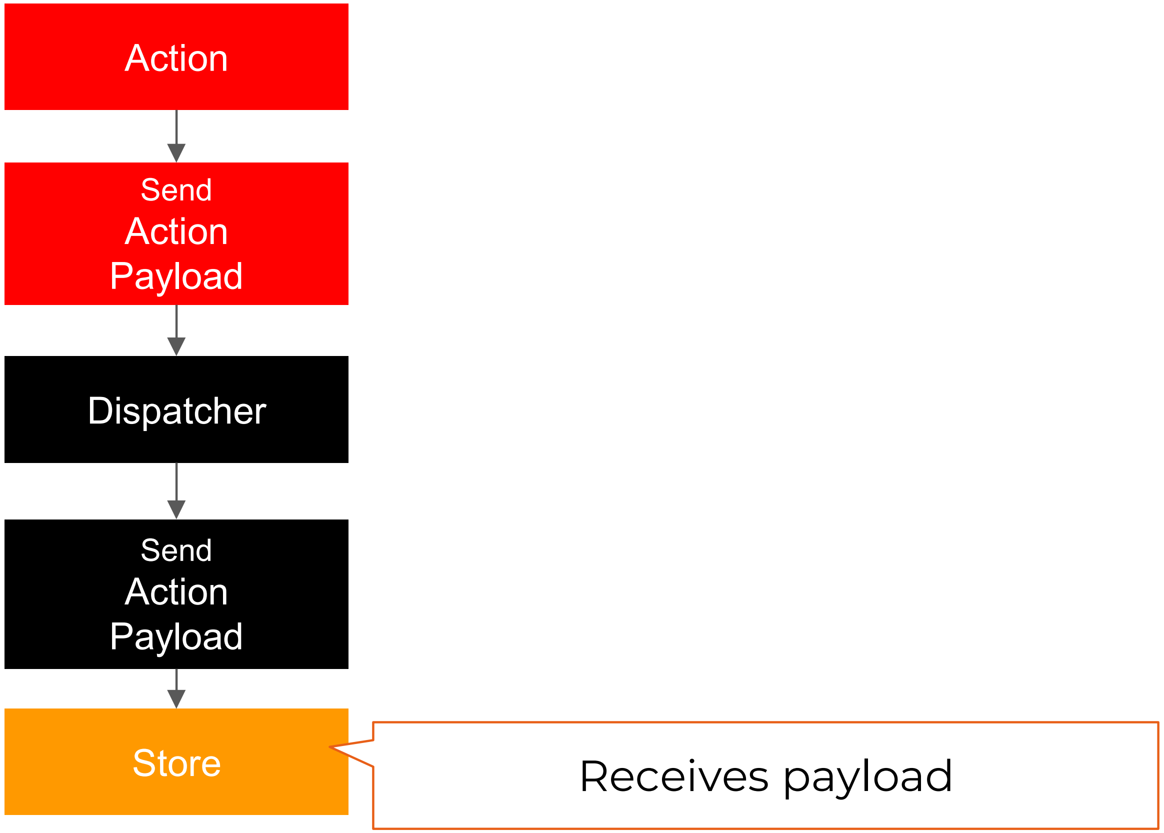
```
{
          type: “HERO_SAVED”
          data: {
                    heroId:
14,
                    heroName:
Bombasto
          }
}
```



Sends payload to all registered callbacks

Flux Flow

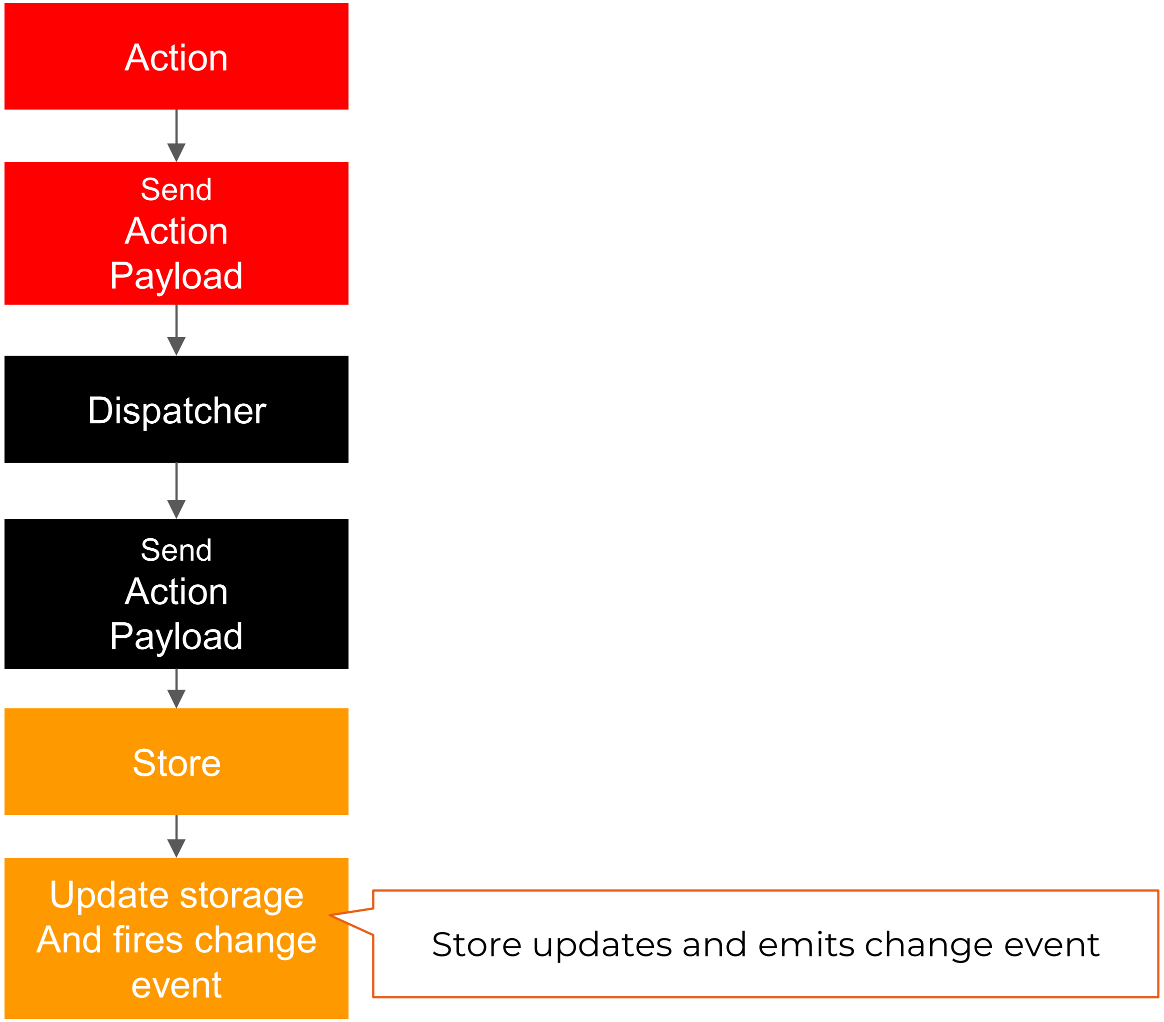
```
{
         type: “HERO_SAVED”
         data: {
                  heroId:
14,
                  heroName:
Bombasto
         }
}
```



Receives payload

Flux Flow

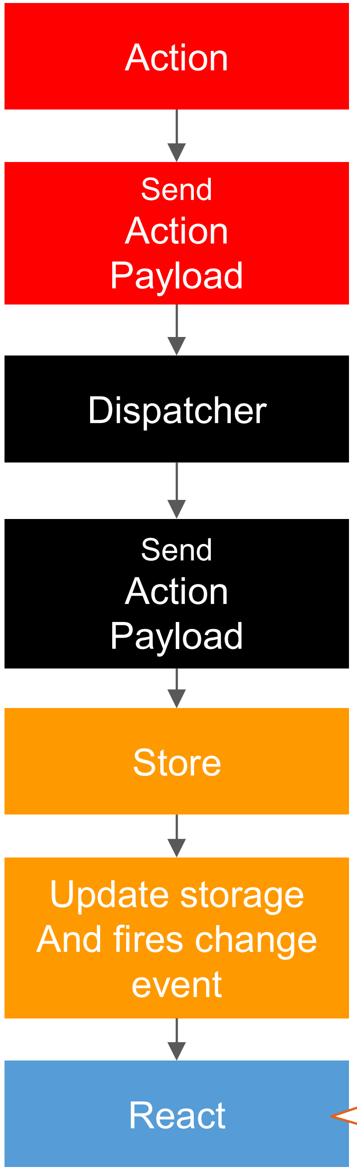
```
{
         type: “HERO_SAVED”
         data: {
                  heroId:
14,
                  heroName:
Bombasto
         }
}
```



Store updates and emits change event

Flux Flow

```
{
         type: “HERO_SAVED”
         data: {
                  heroId:
14,
                  heroName:
Bombasto
         }
}
```



Action

Send Action Payload

Dispatcher

Send Action Payload

Store

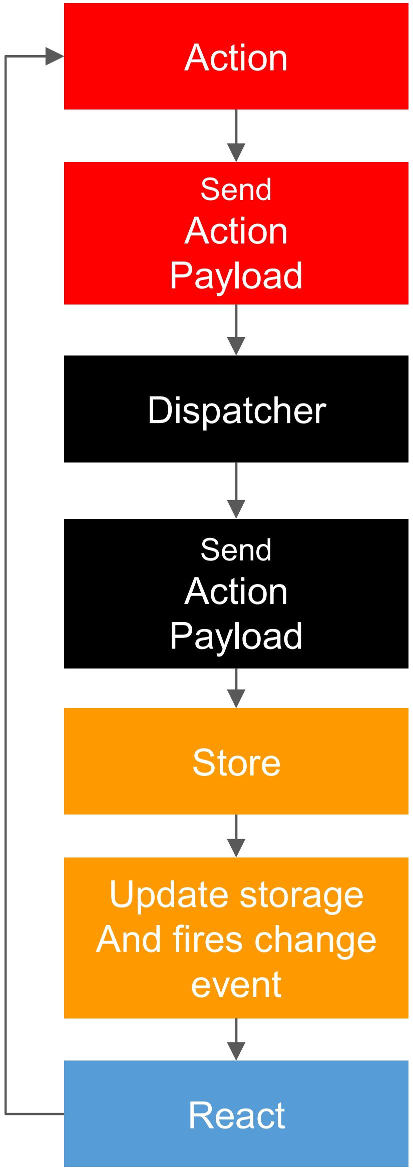
Update storage And fires change event

React

Receives change event and re-renders

Flux Flow

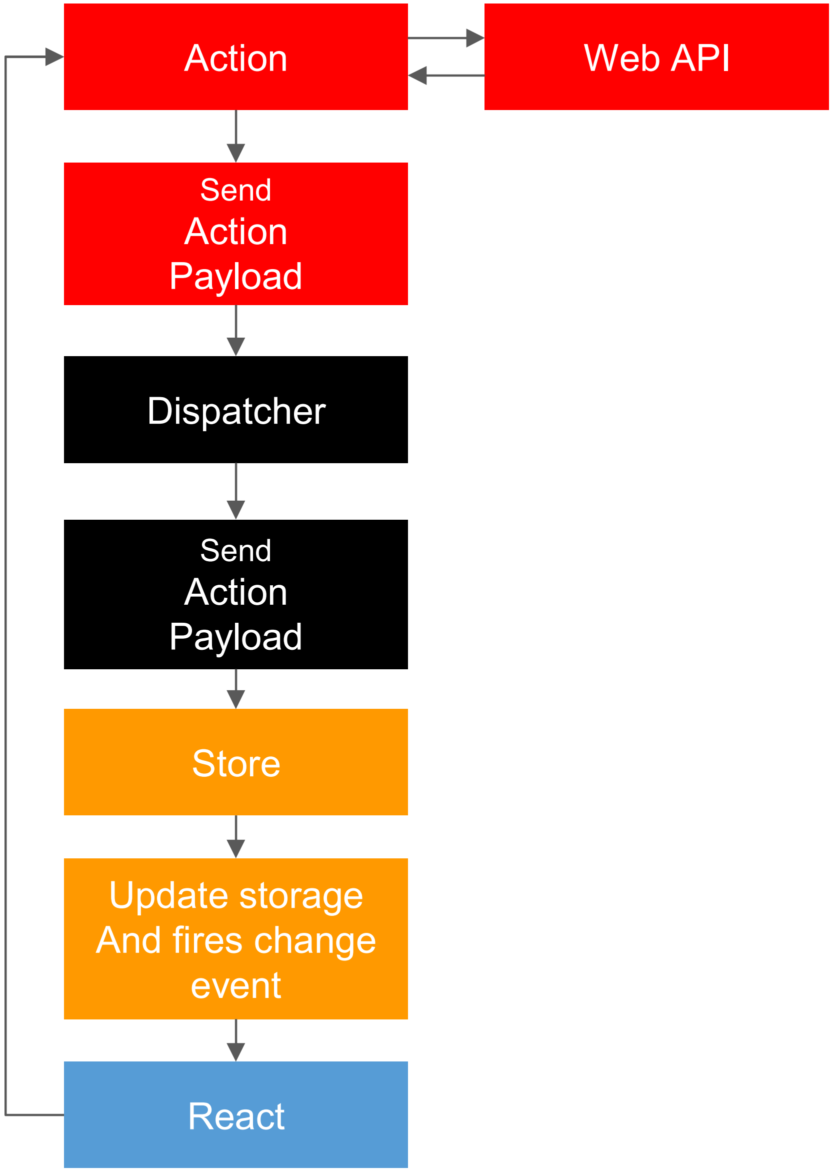
```
{
            type: “HERO_SAVED”
            data: {
                        heroId:
14,
                        heroName:
Bombasto
            }
}
```



New actions in the UI will start this flow over again

Flux Flow

```
{
          type: “HERO_SAVED”
          data: {
                    heroId:
14,
                    heroName:
Bombasto
          }
}
```



Action

Web API

Send
Action
Payload

Dispatcher

Send
Action
Payload

Store

Update storage
And fires change
event

React

A Chat With Flux

React Hey HeroAction, someone clicked this “Save Hero” button

Action Thanks React! I registered an action creator with the dispatcher, so the dispatcher should take care of notifying all the stores that care.

Dispatcher Let me see who cares about a hero being saved. Ah! Looks like the HeroStore has registered a callback with me, so I’ll let it know.

Store Hi Dispatcher! Thanks for the update! I’ll update my data with the payload you sent. Then I’ll emit an event for the React components that care.

React Ooo! Shiny new data from the store! I’ll update the UI to reflect this!

Flux API

The Flux API is 5 functions

Flux API

register(function callback) - “Hey dispatcher, run me when actions happen. - Store”

Flux API

register(function callback) - “Hey dispatcher, run me when actions happen. - Store”

unregister(string id) - “Hey dispatcher, stop worrying about this action. - Store”

Flux API

register(function callback) - “Hey dispatcher, run me when actions happen. - Store”

unregister(string id) - “Hey dispatcher, stop worrying about this action. - Store”

waitFor(array<string> ids) - “Update this store first. - Store”

Flux API

register(function callback) - “Hey dispatcher, run me when actions happen. - Store”

unregister(string id) - “Hey dispatcher, stop worrying about this action. - Store”

waitFor(array<string> ids) - “Update this store first. - Store”

dispatch(object payload) - “Hey dispatcher, tell the stores about this actions. - Action”

Flux API

register(function callback) - “Hey dispatcher, run me when actions happen. - Store”

unregister(string id) - “Hey dispatcher, stop worrying about this action. - Store”

waitFor(array<string> ids) - “Update this store first. - Store”

dispatch(object payload) - “Hey dispatcher, tell the stores about this actions. - Action”

isDispatching() - “I’m busy dispatching callbacks right now.”

Flux is a Publish-Subscribe Model?

Not quite.

Differs in two ways:

1. Every payload is dispatched to all registered callbacks
2. Callbacks can wait for other callbacks