Introduction to React.js

What is React.js?

“A JavaScript **library** for building user interfaces”

“An open-source JavaScript library providing a view for data rendered as HTML”

“V in MVC”

~~“React vs. Angular”~~

Why React?

- ★ Components everywhere
    - ○ Knockout, Ember
    - ○ Directives in Angular
    - ○ native Web Components
- ★ JSX, Virtual DOM, Unidirectional data flow
    - ○ Not first, but first that reaches critical mass

- ★ Encapsulation ✓
- ★ Extremely fast ✓
- ★ Can scale up to complex UIs ✓

HTML should be a projection of application state, not the source of truth.

A view layer

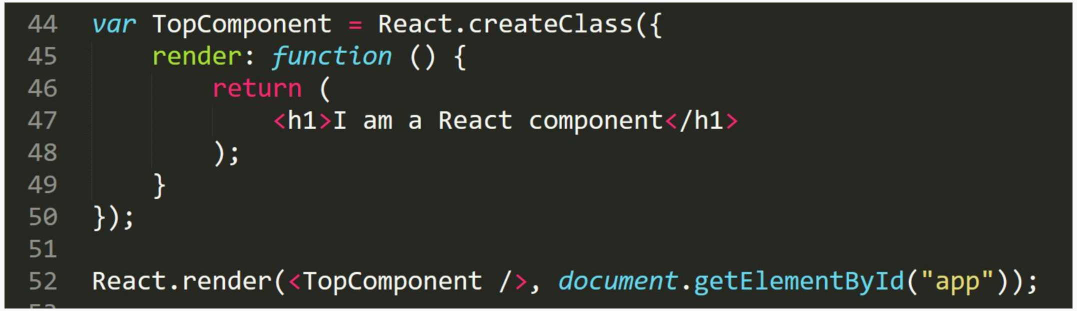
★ Possible tools combination:

  - **Node.js** to run Server-side JS
  - **Browserify** lets you require('modules') in the browser by bundling up all of your dependencies
  - React as our Components library
  - **React Router** to handle client-side Routing
  - Facebook's **Flux** to handle our application's data flows
  - **Gulp** as the Task runner that wires all of this together in an easy-to-use script

Core Concepts

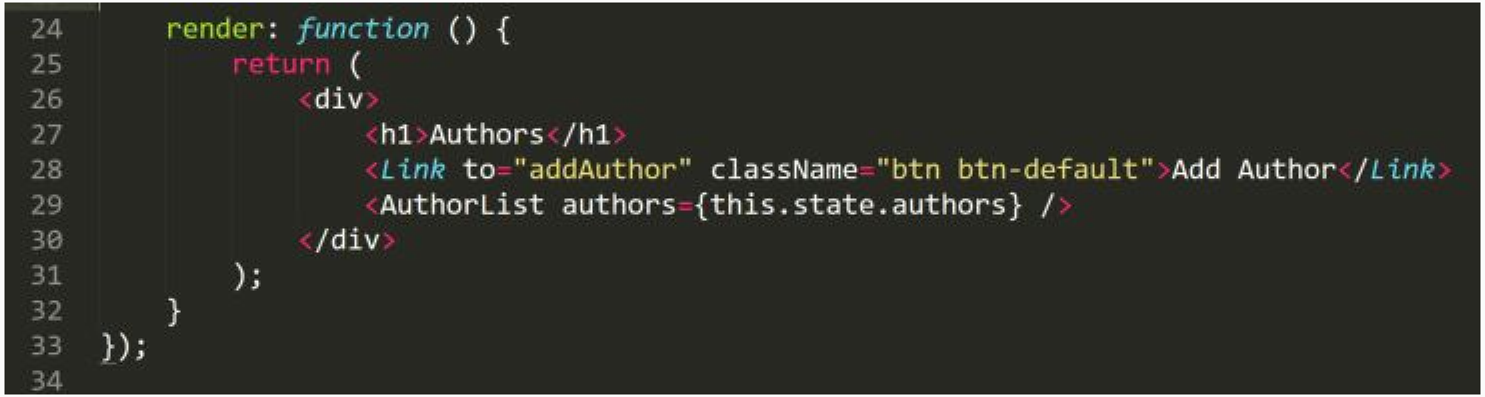
★ **Building reusable components**

(the only thing actually)


```
var TopComponent = React.createClass({
    render: function () {
        return (
            <h1>I am a React component</h1>
        );
    }
});

React.render(<TopComponent />, document.getElementById("app"));
```

JSX

- ★ XML-like syntax for markup
- ★ Compiles to JS
- ★ Just an abstraction over JS
- ★ Optional
- ★ “It’s like you see an ugly baby for the first time”



```
    render: function () {
        return (
            <div>
                <h1>Authors</h1>
                <Link to="addAuthor" className="btn btn-default">Add Author</Link>
                <AuthorList authors={this.state.authors} />
            </div>
        );
    }
});
```

HTML in JS?!

- ★ Why are we ignoring separation of concerns?
- ★ Angular / Ember / Knockout
  - ○ Effectively put JS into HTML (!!)
- ★ React
  - ○ you can enjoy all the power of JavaScript when you're composing your markup
- ★ Are JS and HTML really separated?
- ★ No explicit interface between HTML & JS
  - ○ e.g. in C# : strongly typed interfaces which enable separation of concerns **and** enforce a common interface that must be implemented
- ★ So we have to do it manually :(
- ★ HTML is not strictly parsed, like JS
  - ○ errors are hard to find
- ★ Browsers were designed from the beginning to be very liberal in what they accept.
  - ○ Is that a good place for logic?

“Give it Five Minutes”

Virtual DOM

- ★ Compares the current state of the DOM to the new desired state and determines the most efficient way to update the DOM
- ★ Updating DOM is expensive

- ★ Backbone against React:
    - ○ http://joelburget.com/media/img/monkeys.gif

Flux

A pattern for unidirectional data flows

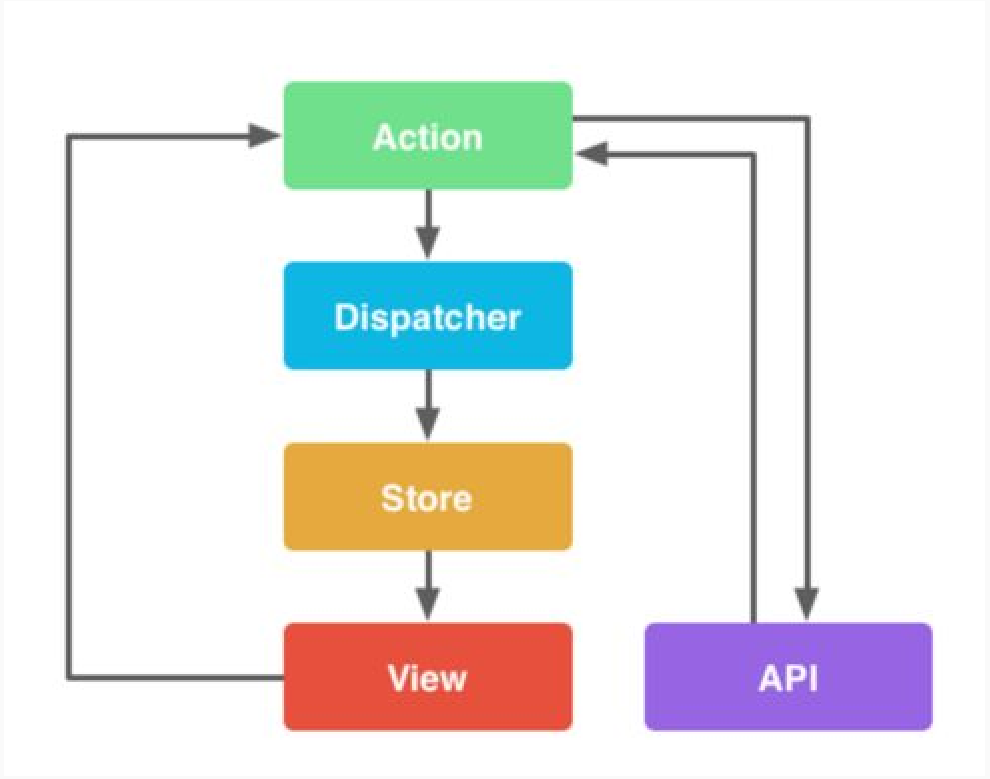
Core Flux Concepts

★ One Dispatcher

★ Actions

★ Stores

★ React Components

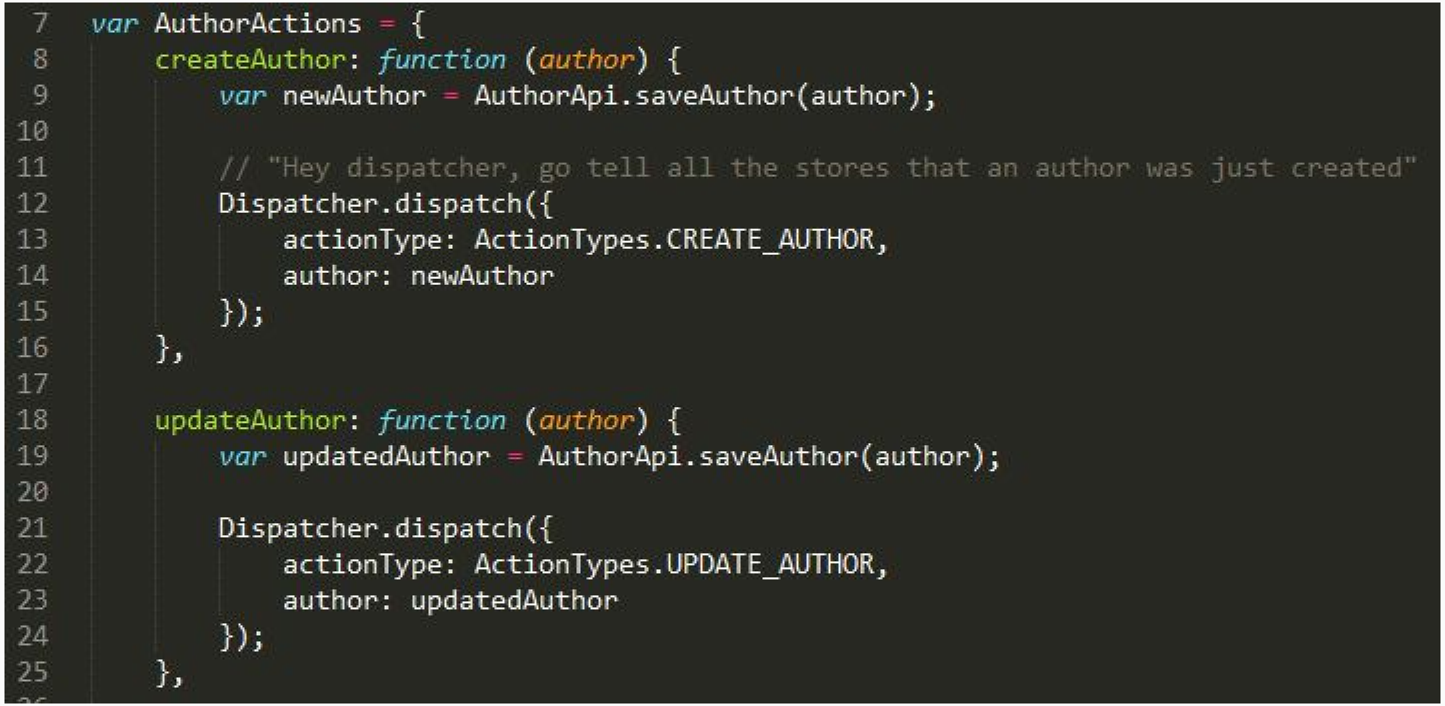


Actions

- ★ Dispatcher exposes a method that allows us to trigger a dispatch to the stores and to include a payload of data, which is called an action
- ★ Payload has type and data
- ★ Action creators are dispatcher helper methods and they describe all the actions that are possible in the application

- ★ 2 ways to trigger actions:
  - ○ user interactions
  - ○ from the server (such as page load)

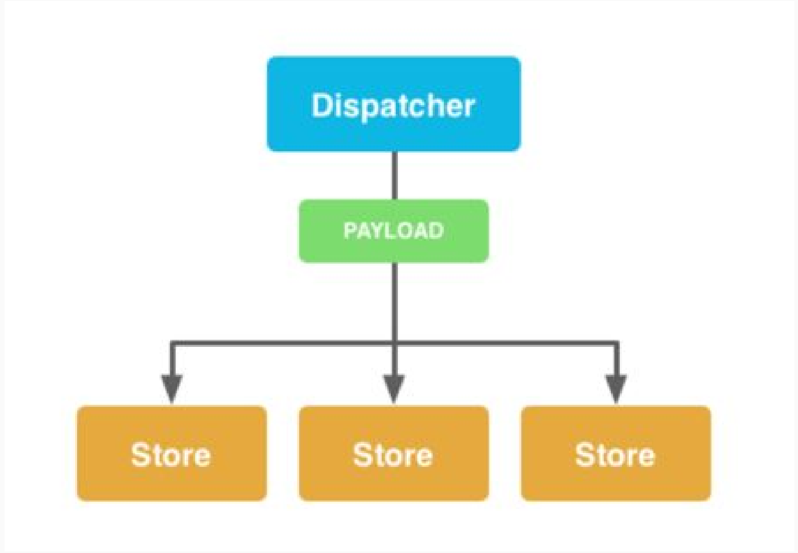
Actions



```
var AuthorActions = {
    createAuthor: function (author) {
        var newAuthor = AuthorApi.saveAuthor(author);

        // "Hey dispatcher, go tell all the stores that an author was just created"
        Dispatcher.dispatch({
            actionType: ActionTypes.CREATE_AUTHOR,
            author: newAuthor
        });
    },

    updateAuthor: function (author) {
        var updatedAuthor = AuthorApi.saveAuthor(author);

        Dispatcher.dispatch({
            actionType: ActionTypes.UPDATE_AUTHOR,
            author: updatedAuthor
        });
    },
```

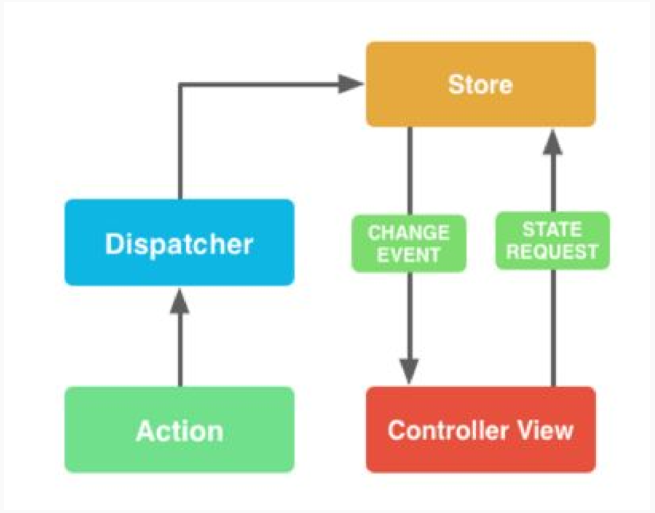
Dispatcher

- ★ All data flows through the dispatcher as a central hub
- ★ Singleton
- ★ A single point where stores can request updates when some action happens
- ★ **It distributes actions to the stores**
- ★ Invokes the callbacks that have been registered & it broadcasts the payload that it receives from the actions

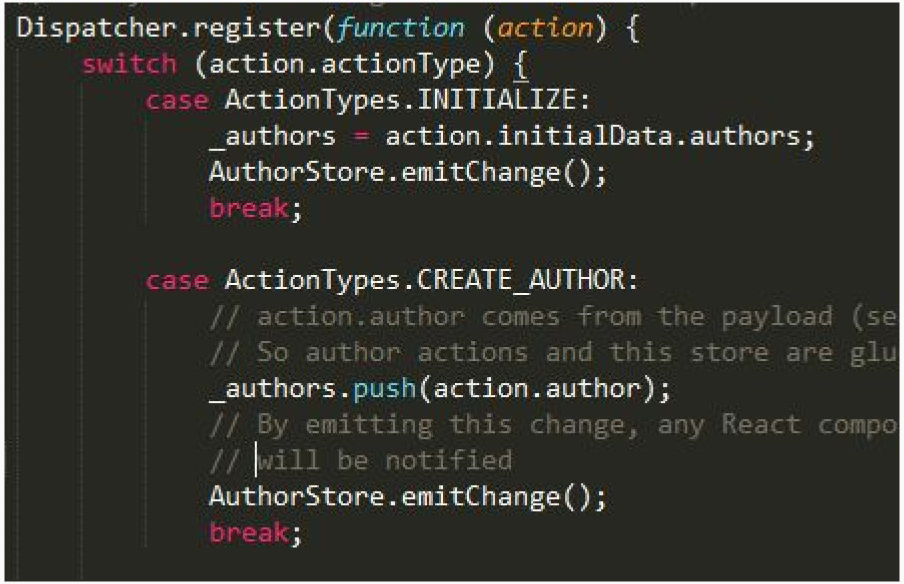


Stores

- ★ A place where our app data is saved
- ★ Hold app state, logic, data retrieval logic and dispatcher callbacks
- ★ Stores get updated because they have callbacks **registered** with dispatcher
- ★ The only thing store should do is to **know how to update data**



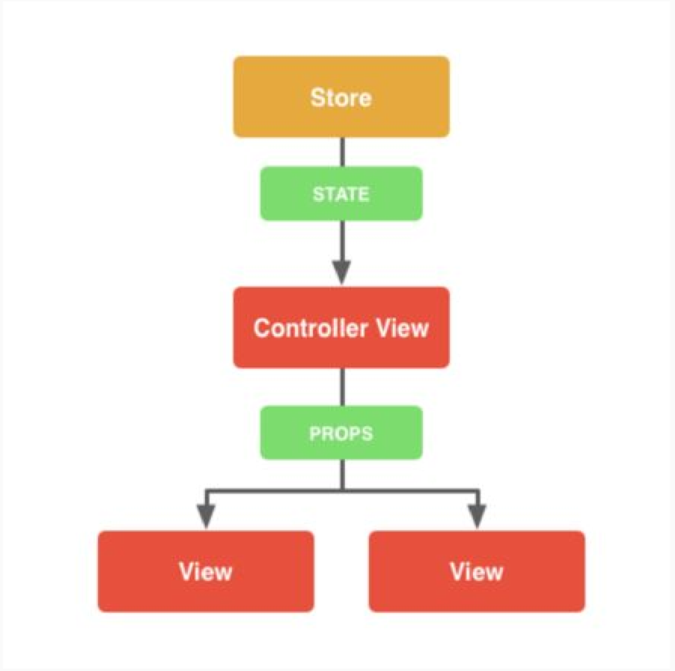
Stores



```
Dispatcher.register(function (action) {
    switch (action.actionType) {
        case ActionTypes.INITIALIZE:
            _authors = action.initialData.authors;
            AuthorStore.emitChange();
            break;

        case ActionTypes.CREATE_AUTHOR:
            // action.author comes from the payload (se
            // So author actions and this store are glu
            _authors.push(action.author);
            // By emitting this change, any React compo
            // will be notified
            AuthorStore.emitChange();
            break;
```

Controller Views

- ★ Just React components that listen to change events and retrieve Application state from Stores
- ★ Views have no responsibility other than to render the current state of the application: they are not guardians of state, nor should they be.



Flux === Publisher/Subscriber ?

★ NO, because every store that registers with the dispatcher is notified of every single action

★ The Flux dispatcher is different from dispatchers in many other architectures:

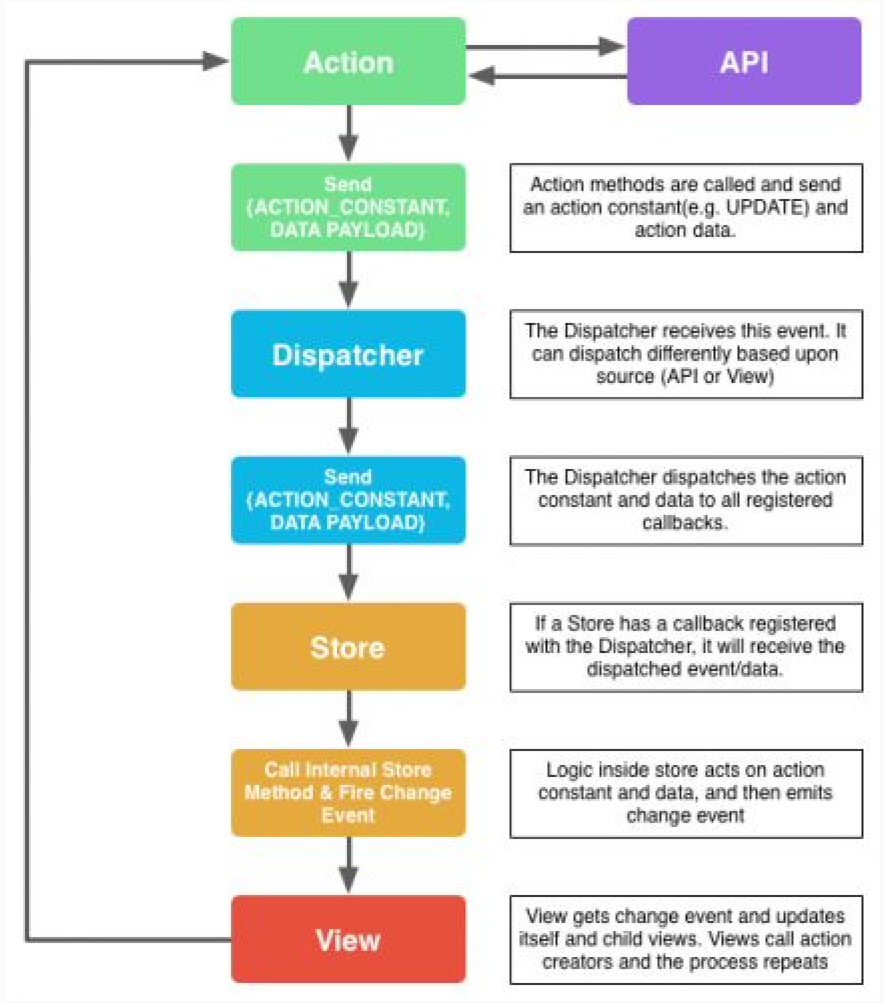
★ The action is sent to all of the registered stores regardless of what the action type is

★ The store doesn’t just subscribe to some actions. It hears about all actions and filters out what it cares about and doesn’t

Wrap Up

Check out my React app at github:

/andreasvaglic/reactflux-playground


Action
API
Send (ACTION_CONSTANT, DATA PAYLOAD)
Action methods are called and send an action constant(e.g. UPDATE) and action data.
Dispatcher
The Dispatcher receives this event. It can dispatch differently based upon source (API or View)
Send (ACTION_CONSTANT, DATA PAYLOAD)
The Dispatcher dispatches the action constant and data to all registered callbacks.
Store
If a Store has a callback registered with the Dispatcher, it will receive the dispatched event/data.
Call Internal Store Method & Fire Change Event
Logic inside store acts on action constant and data, and then emits change event
View
View gets change event and updates itself and child views. Views call action creators and the process repeats

Useful links

★ React Cheat Sheet: http://reactcheatsheet.com/

★ Cartoon guide to Flux: https://code-cartoons.com/a-cartoon-guide-to-flux-6157355ab207#.5z70agdpq

★ Getting to know Flux: https://scotch.io/tutorials/getting-to-know-flux-the-react-js-architecture

Thanks!