Diving into React.js/Flux

Hasitha Liyanage
(hliyan@github.io)

```
// the old way of updating UI based on AJAX
$.ajax({
  url: "foo.com/users",
}).done(function(data) {
  data.forEach(function(item) {
    $(item.id).html(item.title);
  });
});
```

```
<!-- pretty much every client side templating language -->
<script id="some-template" type="text/x-handlebars-template">
  <table>
    <tbody>
      {{#users}}
        <tr>
          <td>{{username}}</td>
          <td>{{firstName}} {{lastName}}</td>
          <td>{{email}}</td>
        </tr>
      {{/users}}
    </tbody>
  </table>
</script>
```

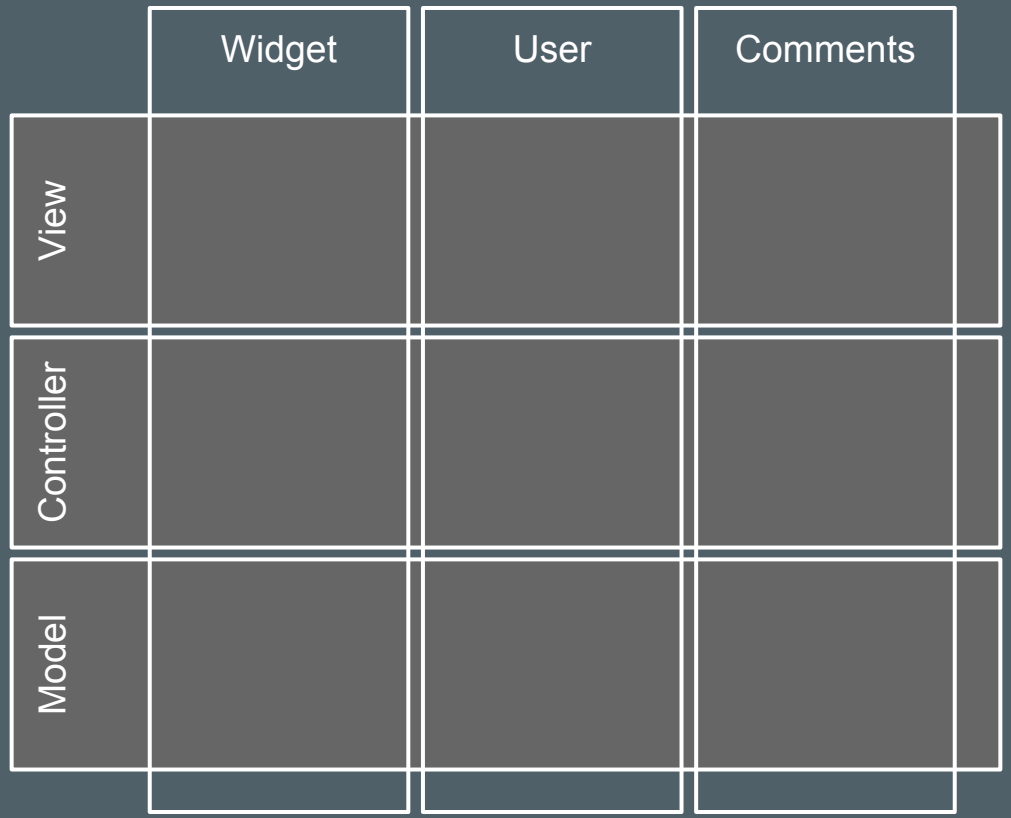
Then and now

~~Page refreshes~~
~~Server side rendering~~
~~Manual UI updates~~
~~Client side MVC~~

AJAX/SPA
Client side templating
Reactive UIs
Web Components
Flux

```
<!-- imagine: custom tags! -->

<App title="Hello">
  <Menu>
    <Link title="Home" route="/" />
    <Link title="Widgets" route="/widgets" />
  </Menu>
  <Main>
    {currentPage}
  </Main>
</App>
```

```
<!-- imagine: self-contained components -->

<WidgetIndex>
  <Search url="/widgets" data={widgets} />
  <List>
    <Widget data={widgets[0]} />
    <Widget data={widgets[1]} />
    <Widget data={widgets[2]} />
    <Widget data={widgets[3]} />
  </List>
</WidgetIndex>
```


Widget
User
Comments
View
Controller
Model

```
// imagine: the inside of 'custom tags'

// widget.js
var Widget = createComponent({
  render: function() {
    return (<div id={this.data.id}>{this.data.name}</div>);
  }
});

// index.html
<script>
  var widget = { id: 1, name: 'Something' };
  renderComponent(<Widget data={widget} />,
     document.getElementById('widget'));
</script>
```

What is React.js?

A Reactive,
Component Based
JavaScript UI Library

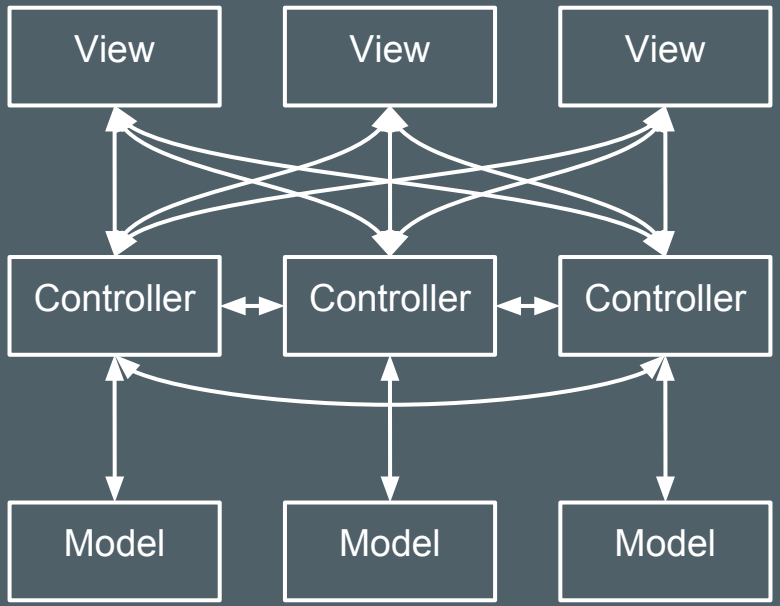
```
var Hello = React.createClass({
    render: function() {
        return <div>Hello {this.props.name}</div>; // JSX!
    }
});

ReactDOM.render(
     <Hello name="World" />,
    document.getElementById('container')
);
```

http://facebook.github.io/react/

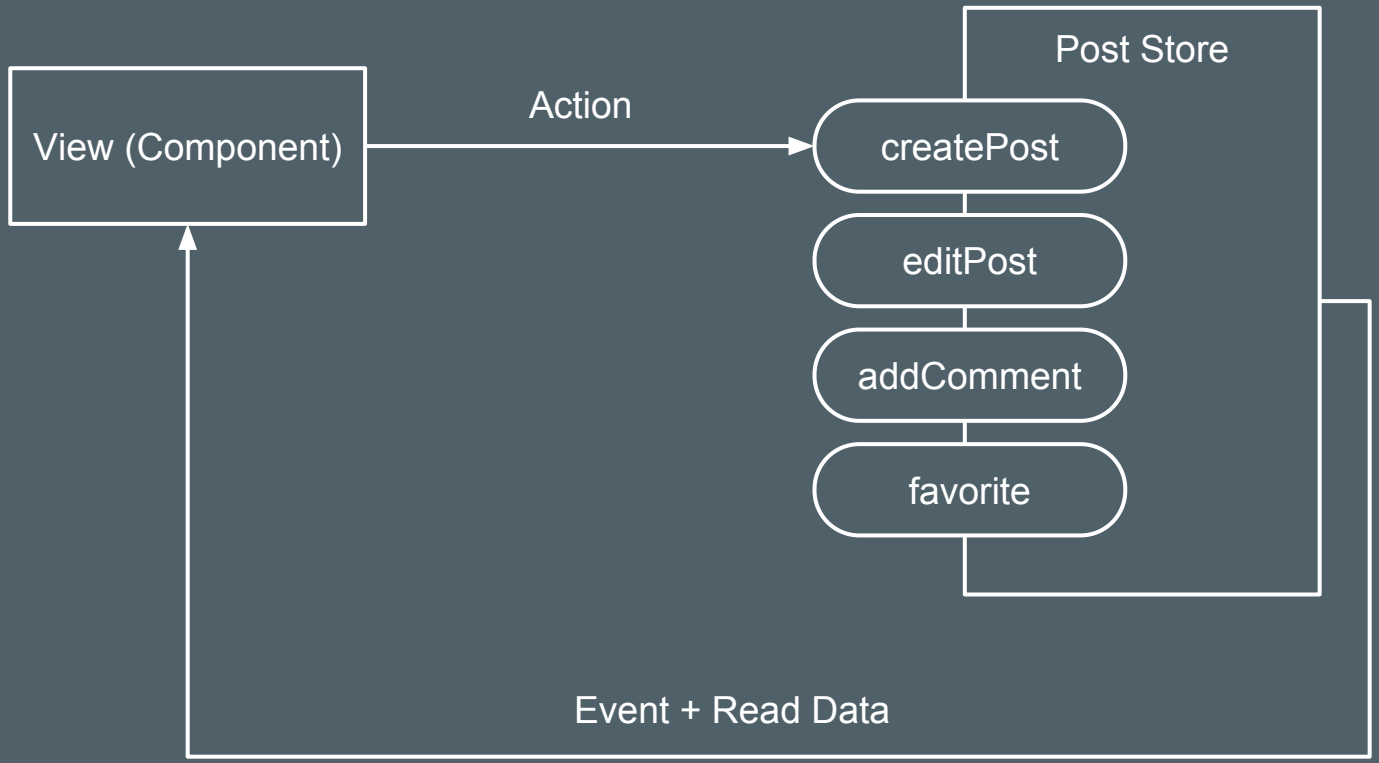
```
var Timer = React.createClass({
  render: function() {
    return (
      <div>Seconds Elapsed: {this.state.secondsElapsed}</div>
    );
  },
  getInitialState: function() {
    return {secondsElapsed: 0};
  },
  componentDidMount: function() {
    this.interval = setInterval(this.tick, 1000);
  },
  componentWillUnmount: function() {
    clearInterval(this.interval);
  },
  tick: function() {
    this.setState({secondsElapsed: this.state.secondsElapsed + 1});
  }

});

// http://facebook.github.io/react/
```

Flux

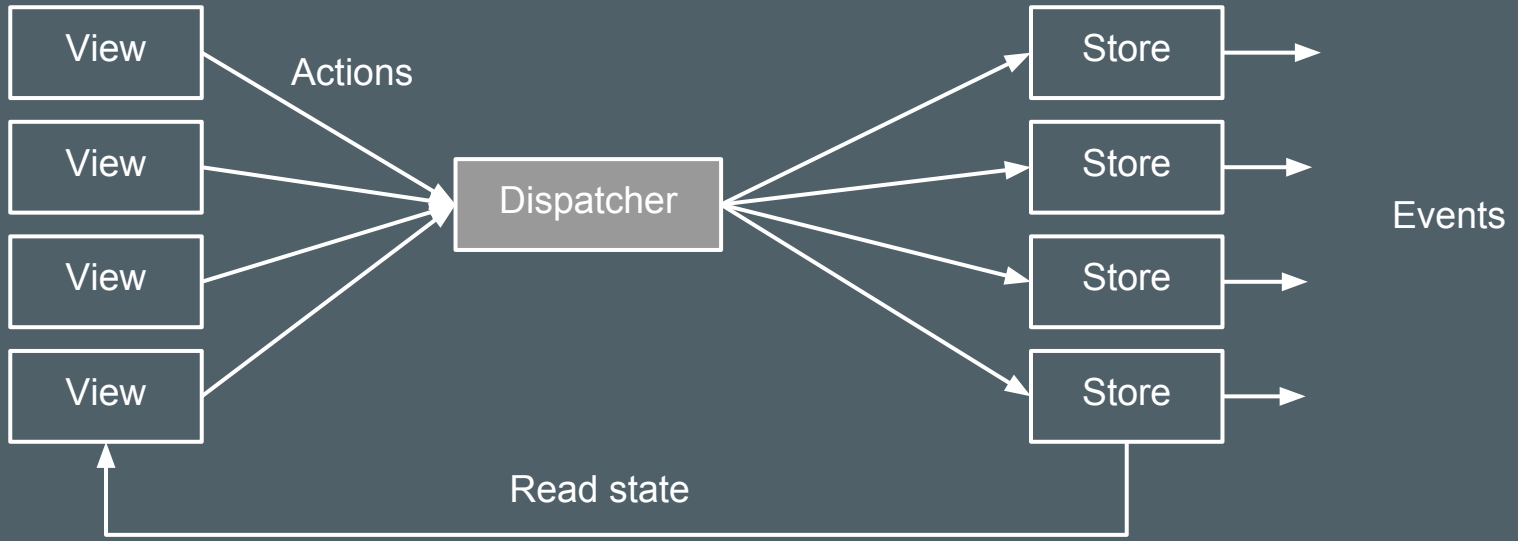
Unlike Angular and Ember, React only provides the V in MVC


View
View
View
Controller
Controller
Controller
Model
Model
Model

"Store"
A slight but very important abstraction on top of a "model"


View (Component)
Action
Post Store
createPost
editPost
addComment
favorite
Event + Read Data

This cleanly splits view logic and storage logic, but doesn't help MVC spaghettification


View
View
View
View
Actions
Dispatcher
Store
Store
Store
Store
Events
Read state

```
// from the create view
Dispatcher.dispatch({
  actionType: CREATE_WIDGET,
  name: 'My Widget',
  size: 10
});

// in the widget store
function onAction(action) {
  switch(action.actionType) {
    case CREATE_WIDGET:
      // write to storage, send to server etc.
      this.emit('change');
    break;
  }
}
```

Points to remember:

A component knows nothing about its environment except what it receives via props

this.state = your data (private)
this.props = others' data (read-only)

A component should know nothing about storage logic (only view logic)

A store should know nothing about display logic (only storage logic)

The only way to mutate the store is by sending it actions

Components should listen to change events of relevant stores and re-read data on change

Questions?

Or,

@h_liyan
hasitha@thinkcube.com
hasitha@info-share.org
hasitha@antyrasolutions.com
hasitha@sltc.lk