ReactJS

The future of web development

https://github.com/facebook/react/wiki/Sites-Using-React

History

- In 2010, Facebook released an extension for PHP called XHP.
- XHP help to decrease XSS attack and make front-end much both readable and understand.

```
<?php
if ($_POST['name']) {
?>
  <span>Hello, <?=$_POST['name']?>.</span>
<?php
} else {
?>
  <form method="post">
  What is your name?<br>
  <input type="text" name="name">
  <input type="submit">
  </form>
<?php
}
```

PHP

```
<?php
if ($_POST['name']) {
  echo <span>Hello,
{$_POST['name']}</span>;
} else {
  echo
    <form method="post">
      What is your name?<br />
      <input type="text" name="name" />
      <input type="submit" />
    </form>;
}
```

XHP

History (cont)

- But…
- There was a distinct problem with XHP: dynamic web applications require many roundtrips to the server.
- XHP did not solve this problem.
- A Facebook engineer negotiated with his manager to take XHP into the browser using JavaScript and was granted six months to try it.
- And…

React was born

ReactJS {purpose}

- Creating user interface(V in MVC model).
- **Building large applications with data that changes over time**.

```
var React = React.createClass({
 render: function() {
  return (
   <h1> Hello React </h1>
  );
 }
});

ReactDOM.render(<React />,document.getElementById('container'));
```

Syntax

ReactJS {contents}

ReactJS {contents}

ReactJS {JSX}

- JSX is a JavaScript syntax extension that looks similar to XML.
- Concise and familiar syntax for defining tree structures with attributes.
- Make large trees easier to read than function calls or object literals.
- Can be used in both HTML tags and Components.

ReactJS {JSX – examples}

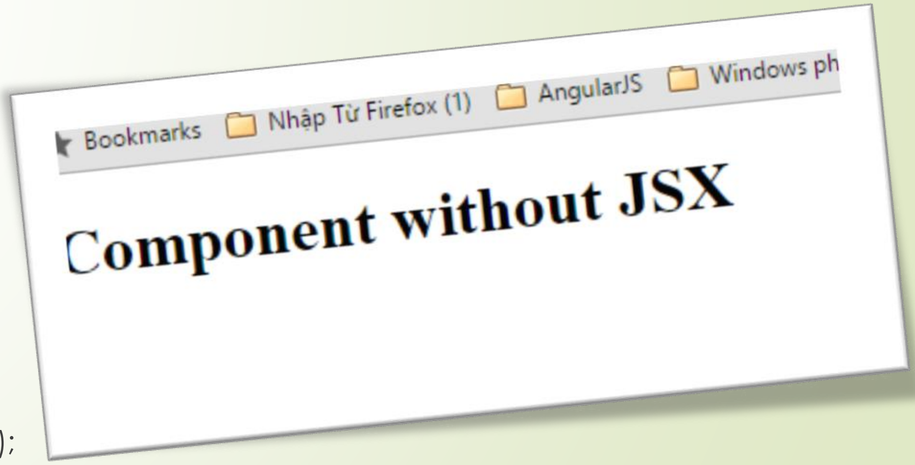
- **HTML tags**
  - var myDivElement = <div className="foo" > HTML tags </div>;
  - ReactDOM.render(myDivElement, document.getElementById('example'));
- **Component**
  - var MyComponent = React.createClass({/*...*/});
  - var myElement = <MyComponent />;
  - ReactDOM.render(myElement, document.getElementById('example'));

ReactJS {JSX – examples (cont)}

- **HTML tags (without JSX)**
  - var myDivElement = React.createElement('div', {className: 'foo'}, 'HTML tags');
  - ReactDOM.render(myDivElement, document.getElementById('example'));

- **Component (without JSX)**

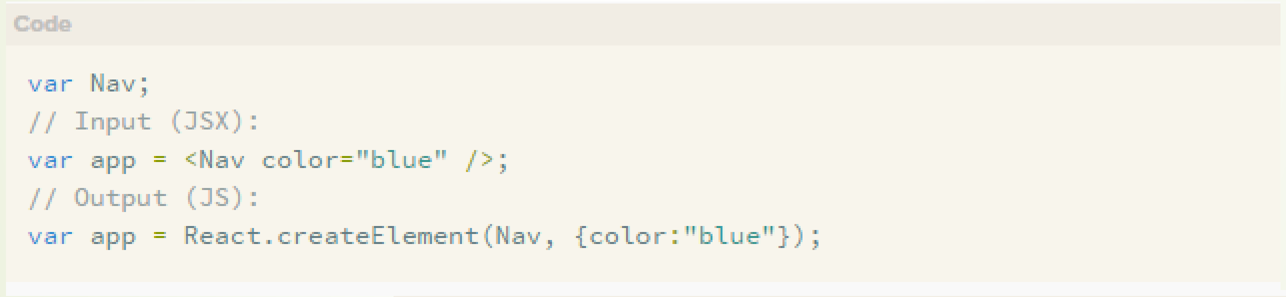
```
var MyComponent = React.createClass({
   render: function() {
      return (
         React.createElement('h1',{}, 'Component without JSX')
       );
      }
    });
 var myElement = <MyComponent />;
 ReactDOM.render(myElement, document.getElementById('content'));
```



Component without JSX

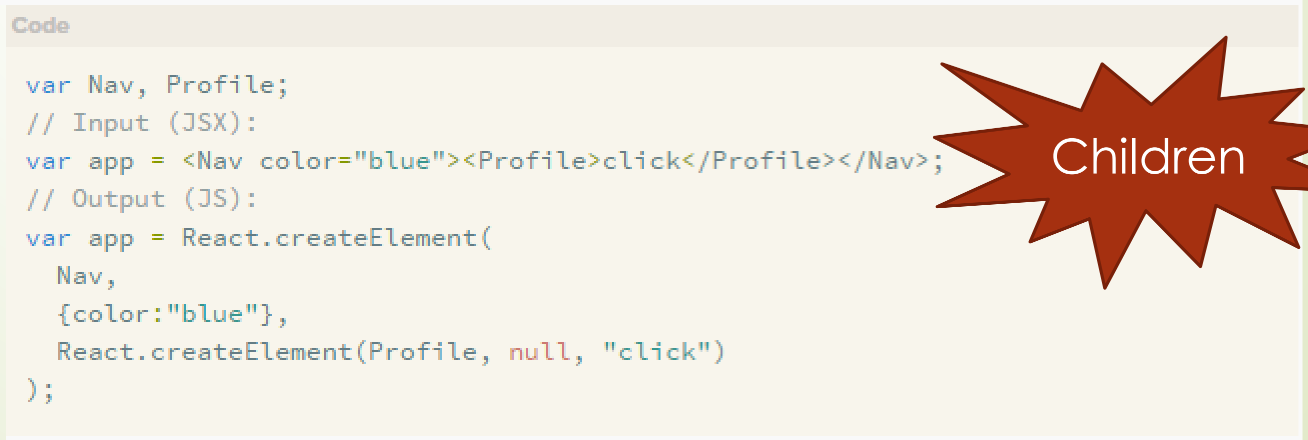
ReactJS {JSX – Transform}

- React JSX transforms from an XML-like syntax into native JavaScript.
- XML elements, attributes and children are transformed into arguments that are passed to **React.createElement.**



Code

```
var Nav;
// Input (JSX):
var app = <Nav color="blue" />;
// Output (JS):
var app = React.createElement(Nav, {color:"blue"});
```



Code

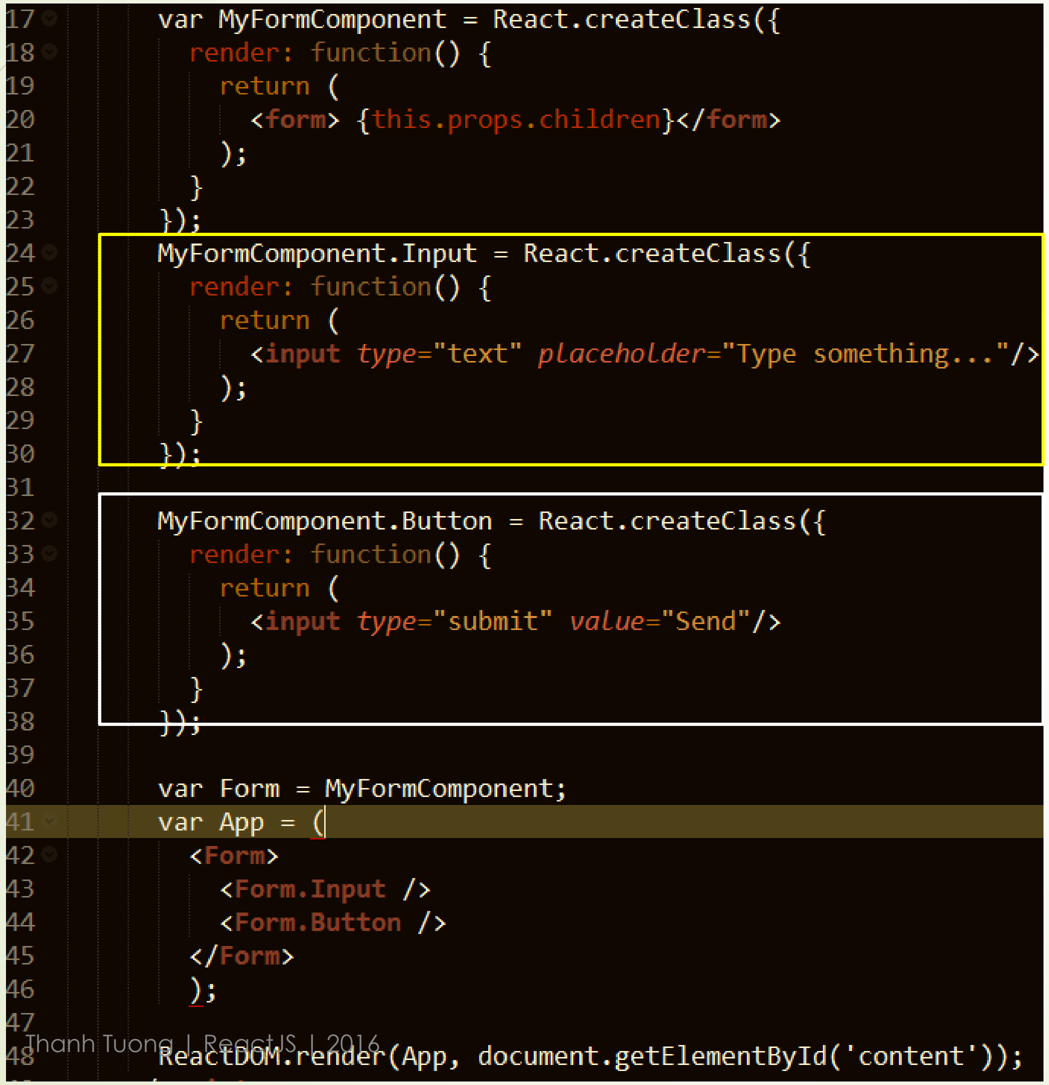
```
var Nav, Profile;
// Input (JSX):
var app = <Nav color="blue"><Profile>click</Profile></Nav>;
// Output (JS):
var app = React.createElement(
  Nav,
  {color:"blue"},
  React.createElement(Profile, null, "click")
);
```

Children

ReactJS {JSX –Namespaced}

- What if you are building a component with many children? For example Form.
- Namespaced components help to make component simpler and easier.
- You just need to create your "*sub-components*" as attributes of the main component.

ReactJS {JSX –Namespaced (cont)}

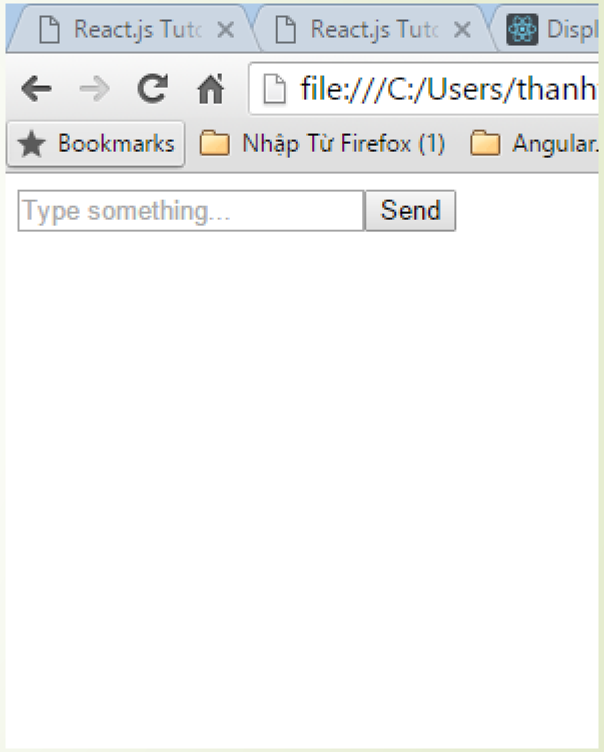


```
var MyFormComponent = React.createClass({
  render: function() {
    return (
      <form> {this.props.children}</form>
    );
  }
});
MyFormComponent.Input = React.createClass({
  render: function() {
    return (
      <input type="text" placeholder="Type something..."/>
    );
  }
});

MyFormComponent.Button = React.createClass({
  render: function() {
    return (
      <input type="submit" value="Send"/>
    );
  }
});

var Form = MyFormComponent;
var App = (
  <Form>
    <Form.Input />
    <Form.Button />
  </Form>
  );

ReactDOM.render(App, document.getElementById('content'));
```



ReactJS {contents}

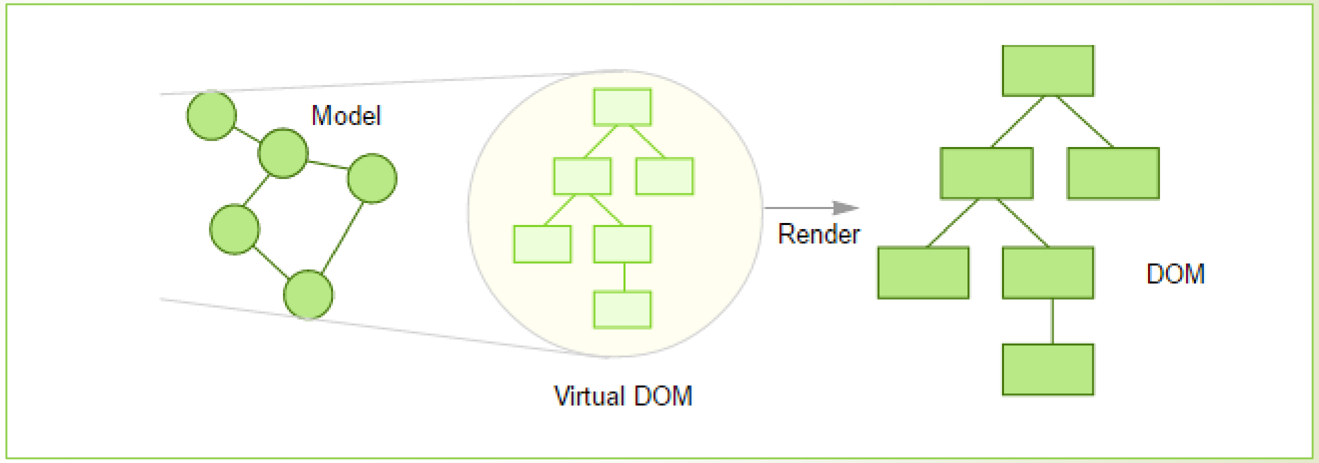
- **JSX**
- **Virtual-DOM**
- Props
- PropTypes
- State
- Refs
- LifeCycle
- Flux Architech
- Thinking in React
- Routing

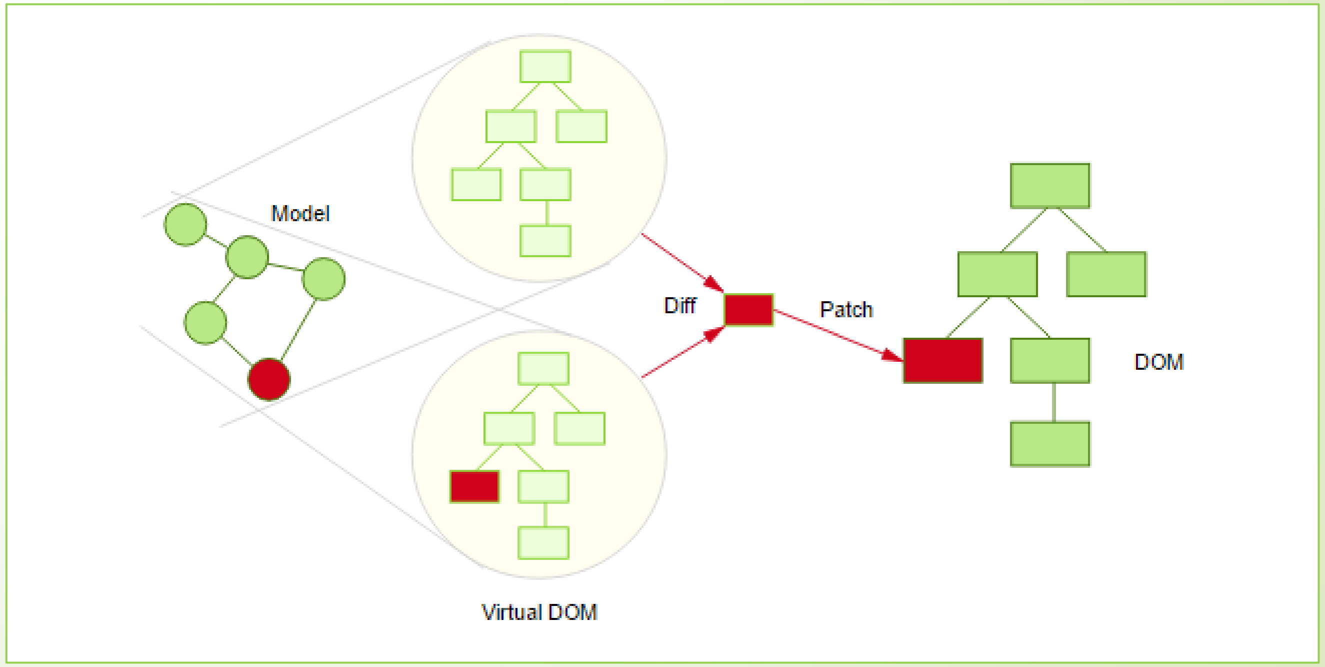
ReactJS {Virtual-DOM}

- **Problem:**
  - DOM manipulation is expensive.
  - Re-render all parts of DOM make your app slowly.
- When the component's state is changed, React will compare with DOM element to make smallest change.
- Is made by React.createElement().
- https://www.youtube.com/watch?v=BYbgopx44vo

ReactJS {Virtual-DOM}

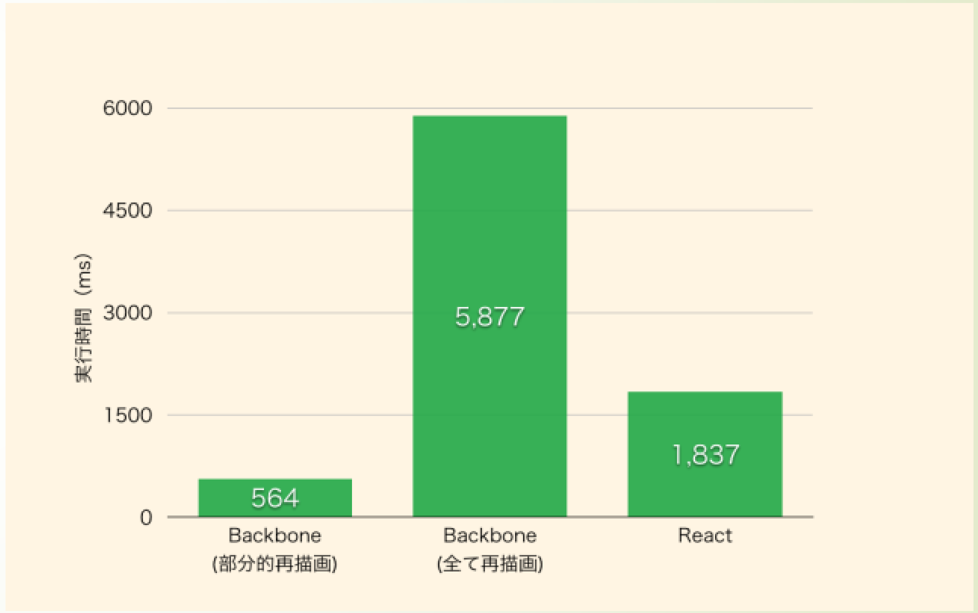
Only diff changes from the two V-DOMs are applied to real DOM





ReactJS {Virtual-DOM (cont)}

- 1. Backbone.js recontruct DOM elements marked as “change”.
- 2. Backbone.js recontruct All DOM elements.
- 3. ReactJS recontruct DOM elements base on calculate the difference.



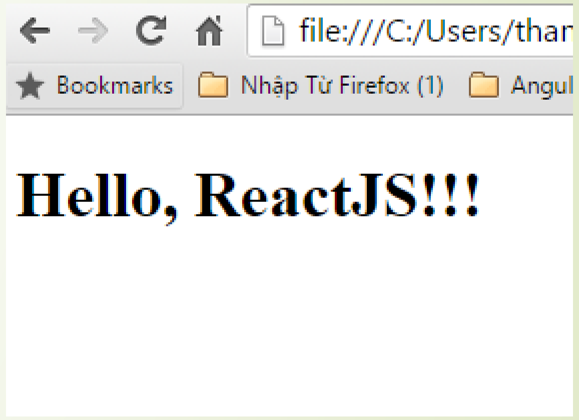
ReactJS {contents}

- **JSX**
- **Virtual-DOM**
- **Props**
- PropTypes
- State
- Refs
- LifeCycle
- Flux Architech
- Thinking in React
- Routing

ReactJS {props}

- Used to pass parameter from parent to children.
- var HelloReact = React.createClass({

```
  render: function() {
    return (
      <h1> Hello, {this.props.name} </h1>
    );
  }
});
ReactDOM.render(<HelloReact name="ReactJS!!!" />, node);
```



Hello, ReactJS!!!

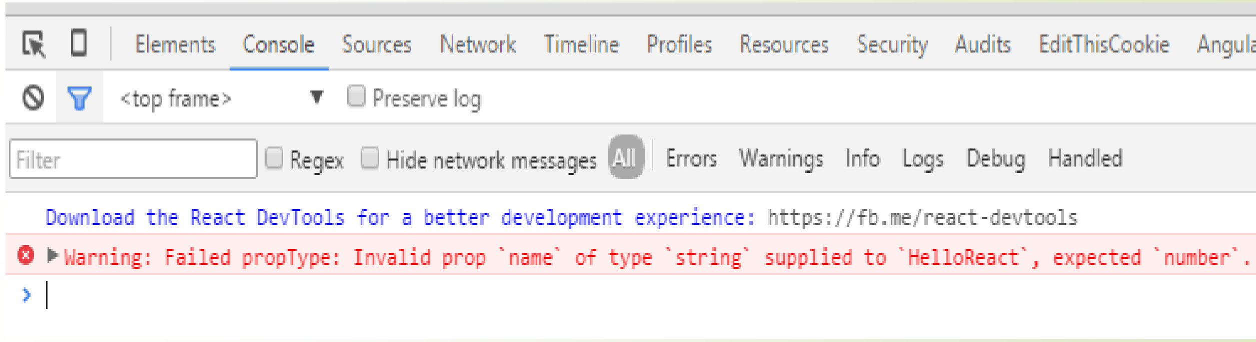
ReactJS {contents}

- **JSX**
- **Virtual-DOM**
- **Props**
- **PropTypes**
- State
- Refs
- LifeCycle
- Flux Architech
- Thinking in React
- Routing

ReactJS {PropTypes}

- **For validate the prop’s value input.**
- var HelloReact = React.createClass({

```
 propTypes: {
   name: React.PropTypes.number
 },
 render: function() {
   return (
     <h1> Hello, {this.props.name} </h1>
   );
 }
});
ReactDOM.render(<HelloReact name="thanh" />, document.getElementById('content'));
```



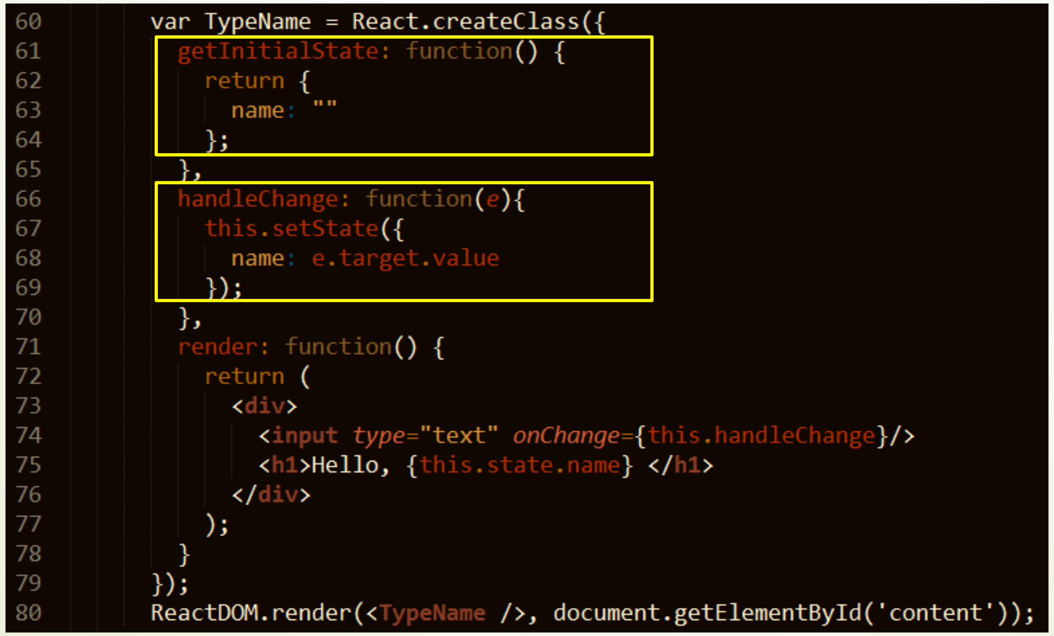
```
Download the React DevTools for a better development experience: https://fb.me/react-devtools
▶Warning: Failed propType: Invalid prop `name` of type `string` supplied to `HelloReact`, expected `number`.
> |
```

ReactJS {contents}

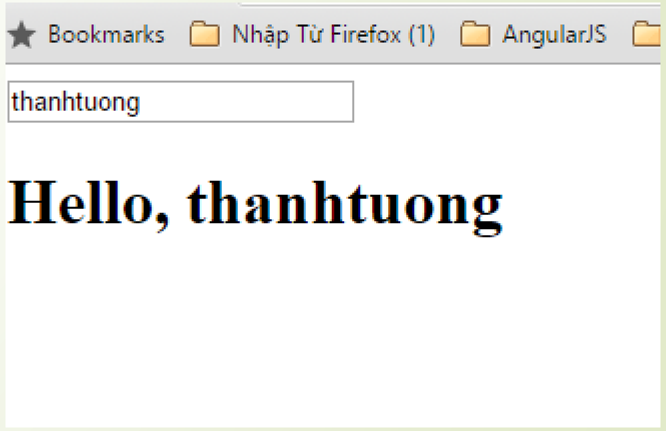
- **JSX**
- **Virtual-DOM**
- **Props**
- **PropTypes**
- **State**
- Refs
- LifeCycle
- Flux Architech
- Thinking in React
- Routing

ReactJS {state}

- To manage state inside component.
- **getInitialState()** function: init value for variable.
- **setState()** function: update new value for variable.



```
var TypeName = React.createClass({
  getInitialState: function() {
    return {
      name: ""
    };
  },
  handleChange: function(e){
    this.setState({
      name: e.target.value
    });
  },
  render: function() {
    return (
      <div>
        <input type="text" onChange={this.handleChange}/>
        <h1>Hello, {this.state.name} </h1>
      </div>
    );
  }
});
ReactDOM.render(<TypeName />, document.getElementById('content'));
```



thanhtuong

Hello, thanhtuong

ReactJS {state-(cont)}

- **When you should use state?**
  - Respond to user input.
  - Server request.
  - or the passage of time.

ReactJS { props vs state }

Features	props	state
Can get initial value from parent Component?	Yes	Yes
Can be changed by parent Component?	Yes	No
Can set default values inside Component?	Yes	Yes
Can change inside Component?	No	Yes
Can set initial value for child Components?	Yes	Yes
Can change in child Components?	Yes	No

ReactJS {contents}

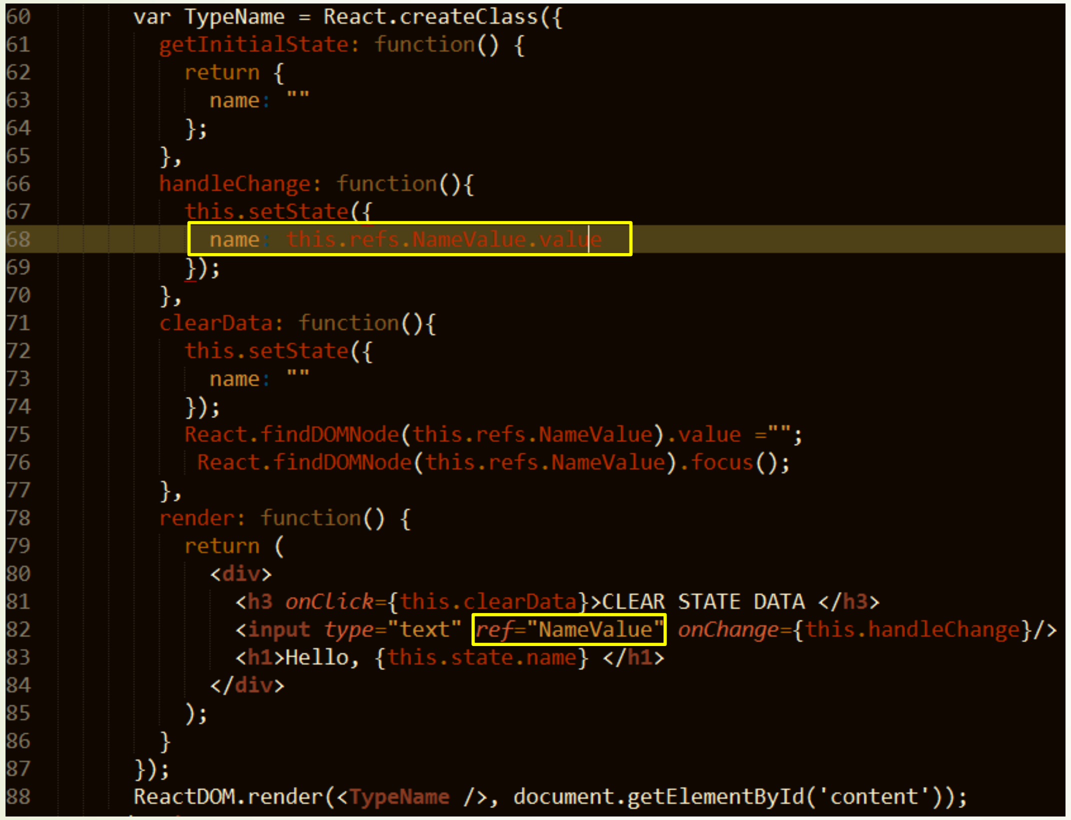
- **JSX**
- **Virtual-DOM**
- **Props**
- **PropTypes**
- **State**
- **Refs**
- LifeCycle
- Flux Architech
- Thinking in React
- Routing

ReactJS {refs}

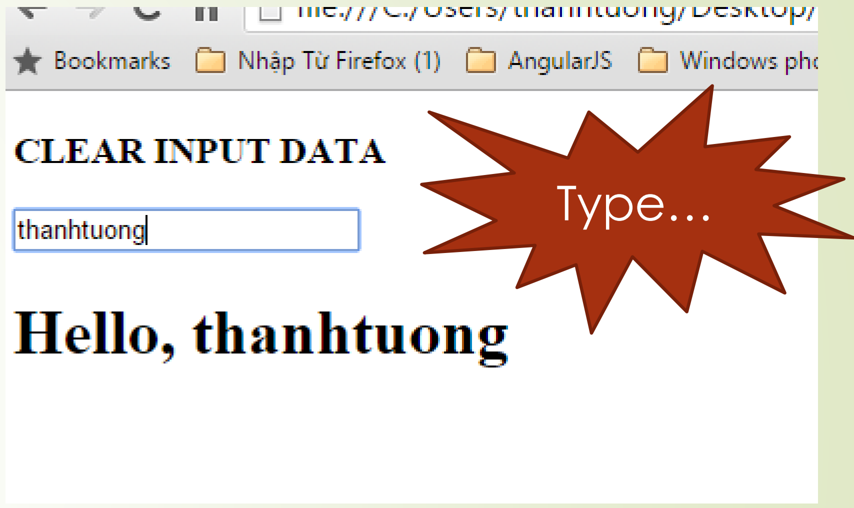
- How we make focus to input element after clear data from input element?
- How we can make a search with many criteria ?
- …

Refs

ReactJS {refs-(cont)}



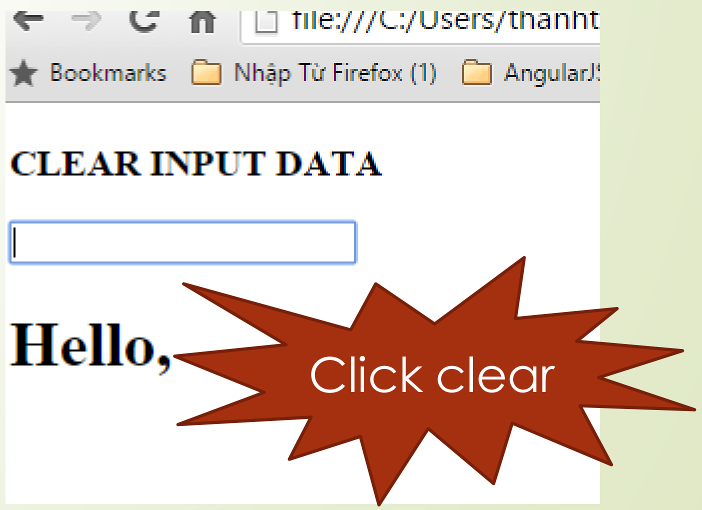
```
    var TypeName = React.createClass({
      getInitialState: function() {
        return {
          name: ""
        };
      },
      handleChange: function(){
        this.setState({
          name: this.refs.NameValue.value
        });
      },
      clearData: function(){
        this.setState({
          name: ""
        });
        React.findDOMNode(this.refs.NameValue).value ="";
         React.findDOMNode(this.refs.NameValue).focus();
      },
      render: function() {
        return (
          <div>
            <h3 onClick={this.clearData}>CLEAR STATE DATA </h3>
            <input type="text" ref="NameValue" onChange={this.handleChange}/>
            <h1>Hello, {this.state.name} </h1>
          </div>
        );
      }
    });
    ReactDOM.render(<TypeName />, document.getElementById('content'));
```



CLEAR INPUT DATA

thanhtuong

Hello, thanhtuong



CLEAR INPUT DATA

Hello,

ReactJS {contents}

- **JSX**
- **Virtual-DOM**
- **Props**
- **PropTypes**
- **State**
- **Refs**
- **LifeCycle**
- Flux Architech
- Thinking in React
- Routing

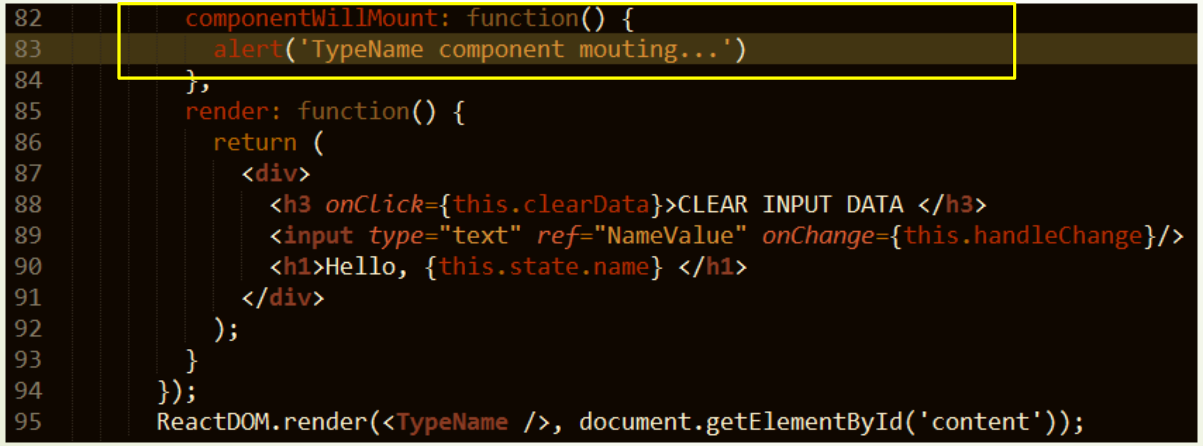
ReactJS {LifeCycle}

- Each component has its own lifecycle events.
- **Ex:**
  - If we wanted to make an ajax request on the initial render and fetch some data, where would we do that?
  - If we wanted to run some logic whenever our props changed, how would we do that?
  - …

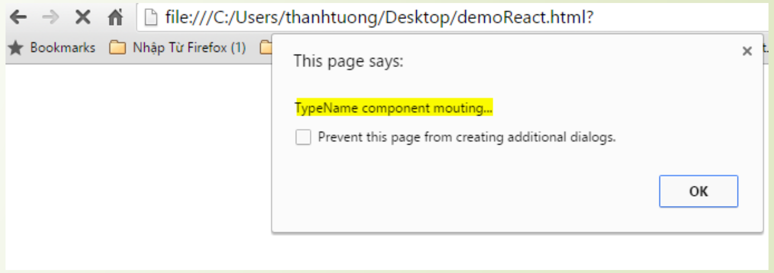
LifeCycle events

ReactJS {LifeCycle (cont)}

- **componentWillMount**
  - Invoked once (both on the client and server) before the *initial* render.
  - Good place to make connection to your db service (ex: firebase,...)
  - Do not call set state method here.



```
  componentWillMount: function() {
    alert('TypeName component mouting...')
  },
  render: function() {
    return (
      <div>
        <h3 onClick={this.clearData}>CLEAR INPUT DATA </h3>
        <input type="text" ref="NameValue" onChange={this.handleChange}/>
        <h1>Hello, {this.state.name} </h1>
      </div>
    );
  }
});
ReactDOM.render(<TypeName />, document.getElementById('content'));
```



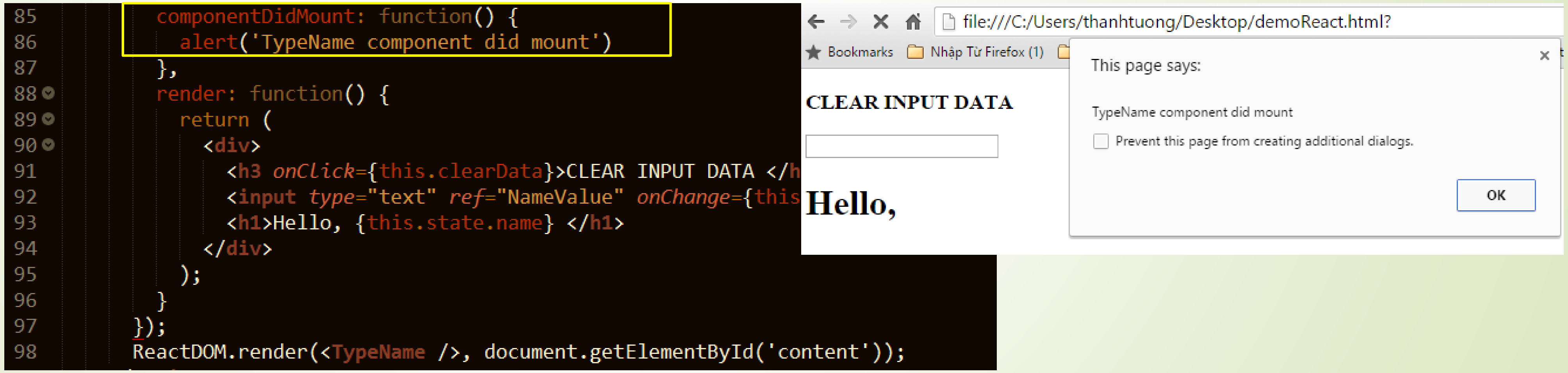
This page says:

TypeName component mouting...

Prevent this page from creating additional dialogs.

ReactJS {LifeCycle (cont)}

- **componentDidMount**
  - Invoked once, only on the client (not on the server).
  - Immediately after the initial rendering occurs.
  - It is good place for you to make AJAX request to fetch data for first used.



```
    componentDidMount: function() {
      alert('TypeName component did mount')
    },
    render: function() {
      return (
        <div>
          <h3 onClick={this.clearData}>CLEAR INPUT DATA </h
          <input type="text" ref="NameValue" onChange={this
          <h1>Hello, {this.state.name} </h1>
        </div>
      );
    }
  });
  ReactDOM.render(<TypeName />, document.getElementById('content'));
```

file:///C:/Users/thanhtuong/Desktop/demoReact.html?

Bookmarks | Nhập Từ Firefox (1)

CLEAR INPUT DATA

Hello,

This page says:

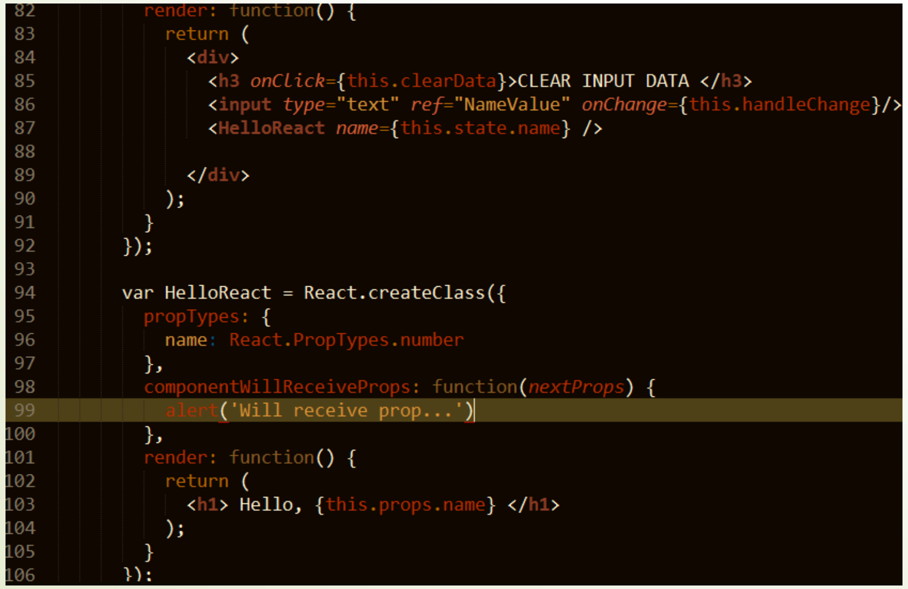
TypeName component did mount

☐ Prevent this page from creating additional dialogs.

OK

ReactJS {LifeCycle (cont)}

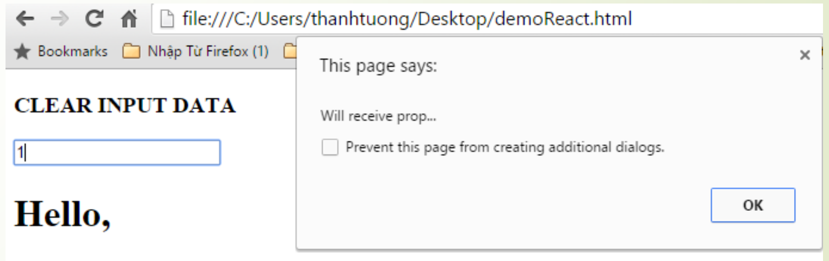
- **componentWillReceiveProps**
  - Invoked when a component is receiving new props.
  - This method is not called for the initial render.
  - Use this method as a way to react to a prop change before render() is called by updating the state with setState.



```
    render: function() {
      return (
        <div>
          <h3 onClick={this.clearData}>CLEAR INPUT DATA </h3>
          <input type="text" ref="NameValue" onChange={this.handleChange}/>
          <HelloReact name={this.state.name} />

        </div>
      );
    }
  });

  var HelloReact = React.createClass({
    propTypes: {
      name: React.PropTypes.number
    },
    componentWillReceiveProps: function(nextProps) {
      alert('Will receive prop...')
    },
    render: function() {
      return (
        <h1> Hello, {this.props.name} </h1>
      );
    }
  });
```



file:///C:/Users/thanhtuong/Desktop/demoReact.html

Bookmarks Nhập Từ Firefox (1)

CLEAR INPUT DATA

1

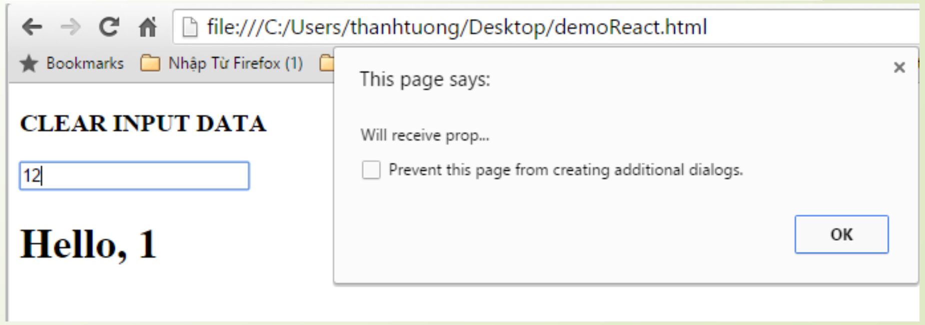
Hello,

This page says:

Will receive prop...

☐ Prevent this page from creating additional dialogs.

OK



file:///C:/Users/thanhtuong/Desktop/demoReact.html

Bookmarks Nhập Từ Firefox (1)

CLEAR INPUT DATA

12

Hello, 1

This page says:

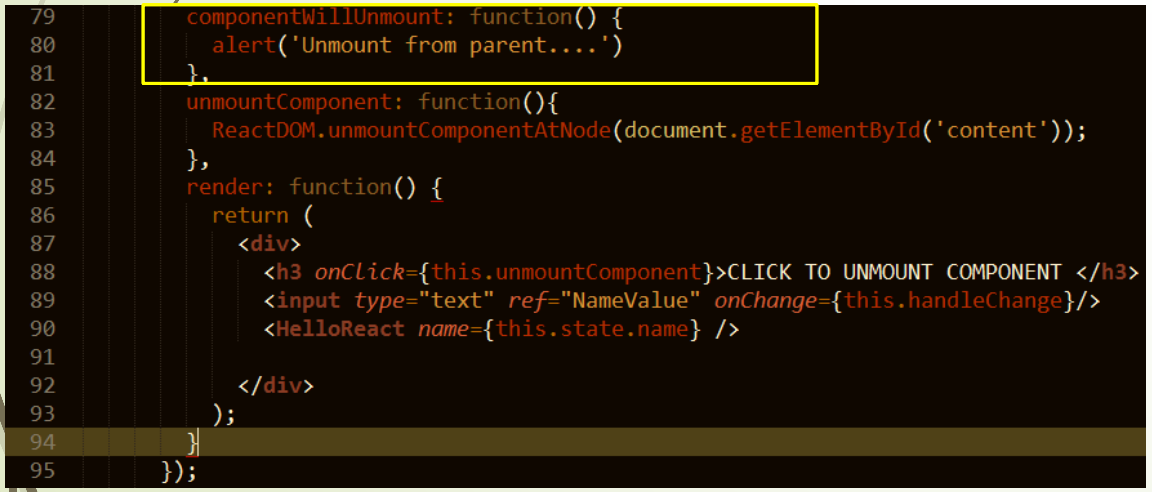
Will receive prop...

☐ Prevent this page from creating additional dialogs.

OK

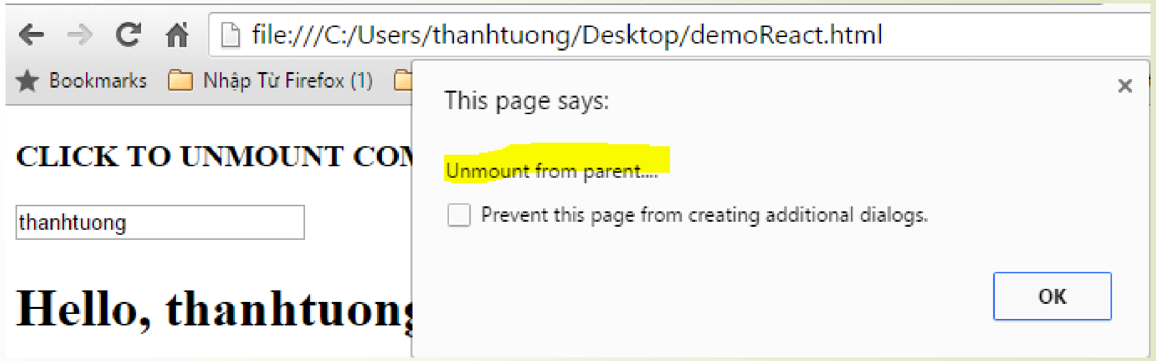
ReactJS {LifeCycle (cont)}

- **componentWillUnmount**
  - Invoked immediately before a component is unmounted from the DOM.
  - Perform any necessary cleanup in this method(Ex: invalidating timers, clear up DOM elements were created at componentDidMount)



```
    componentWillUnmount: function() {
      alert('Unmount from parent....')
    },
    unmountComponent: function(){
      ReactDOM.unmountComponentAtNode(document.getElementById('content'));
    },
    render: function() {
      return (
        <div>
          <h3 onClick={this.unmountComponent}>CLICK TO UNMOUNT COMPONENT </h3>
          <input type="text" ref="NameValue" onChange={this.handleChange}/>
          <HelloReact name={this.state.name} />

        </div>
      );
    }
  });
```



file:///C:/Users/thanhtuong/Desktop/demoReact.html

CLICK TO UNMOUNT COM

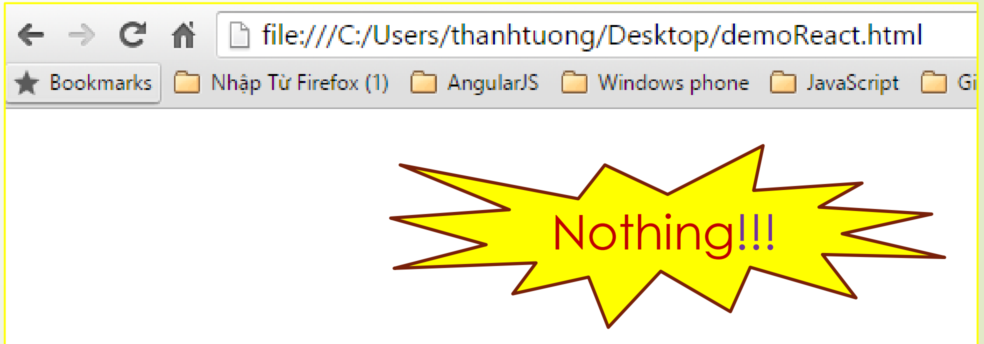
thanhtuong

Hello, thanhtuong

This page says:

Unmount from parent....

Prevent this page from creating additional dialogs.



file:///C:/Users/thanhtuong/Desktop/demoReact.html

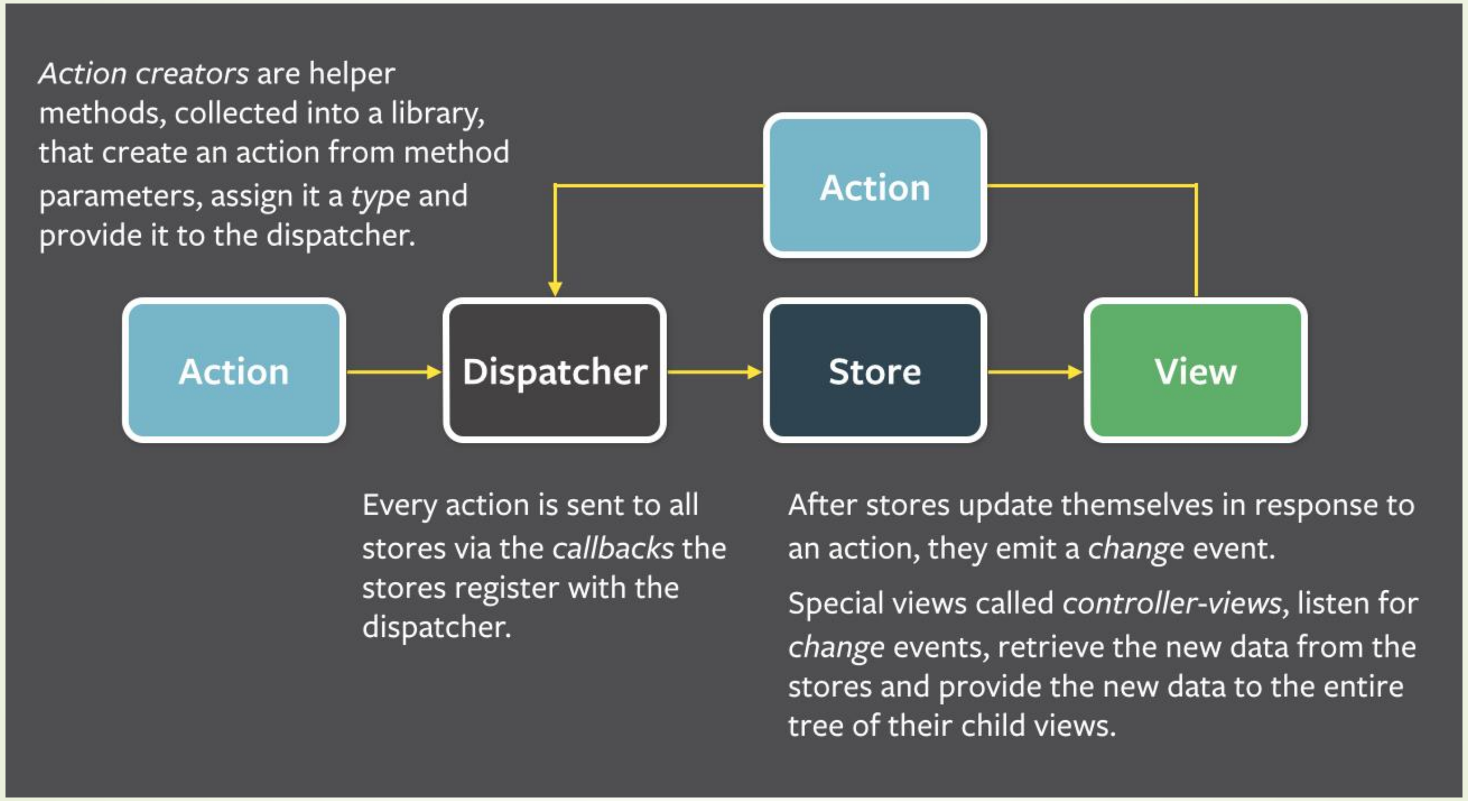
http://facebook.github.io/react/docs/component-specs.html

ReactJS {contents}

ReactJS {Flux}

- Flux is the application architecture.
- **Making data changes easy.**
- Remove the burden of having a component manage its own state.
- The data is moved to the central called Store.
- If your app doesn't have and or care about dynamic data, Flux might not be the best choice.
- Unidirectional data flow.

ReactJS {Flux - flow}


Action creators are helper methods, collected into a library, that create an action from method parameters, assign it a *type* and provide it to the dispatcher.

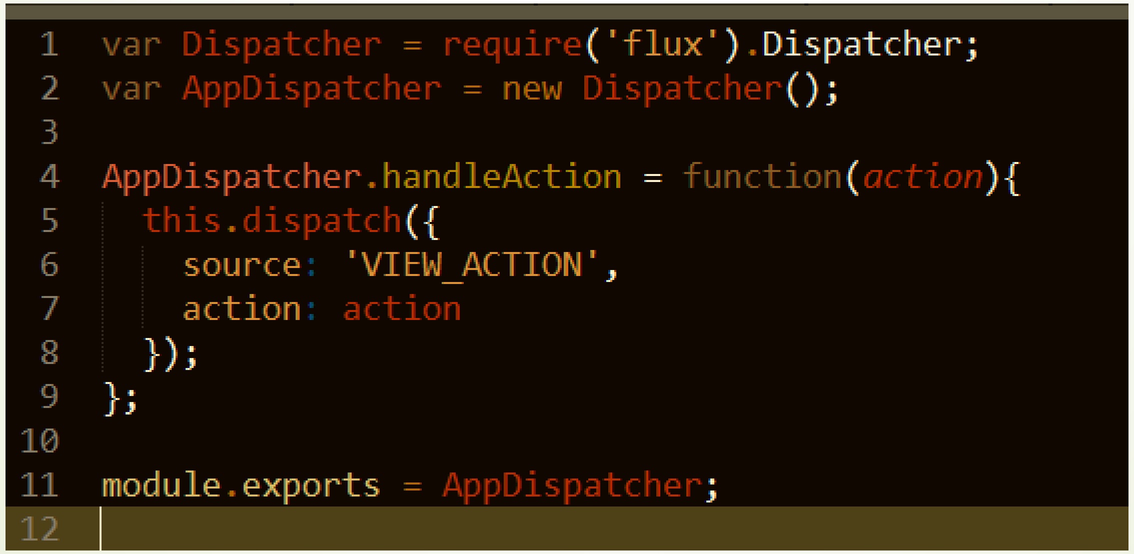
Every action is sent to all stores via the *callbacks* the stores register with the dispatcher.

After stores update themselves in response to an action, they emit a *change* event.

Special views called *controller-views*, listen for *change* events, retrieve the new data from the stores and provide the new data to the entire tree of their child views.

ReactJS {Flux - flow}

- **Dispatcher**
  - Is the central hub that manages all data flow in a Flux application.
  - Essentially a registry of callbacks into the stores.



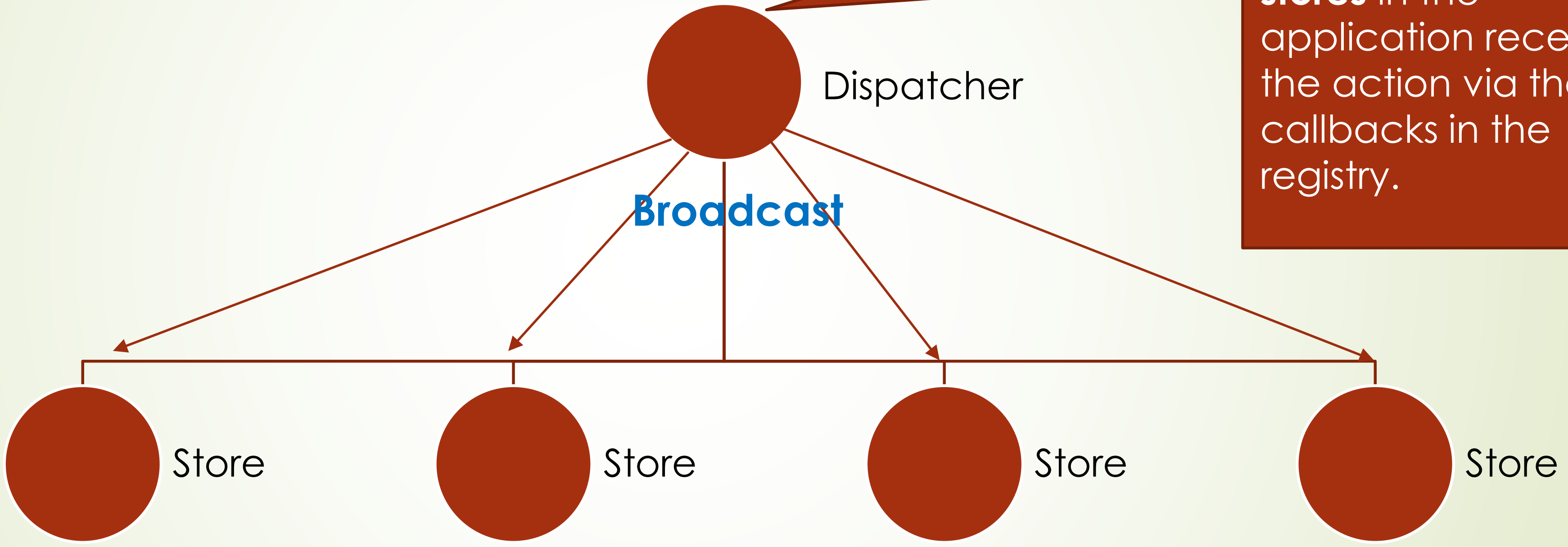
```
var Dispatcher = require('flux').Dispatcher;
var AppDispatcher = new Dispatcher();

AppDispatcher.handleAction = function(action){
  this.dispatch({
    source: 'VIEW_ACTION',
    action: action
  });
};

module.exports = AppDispatcher;

```

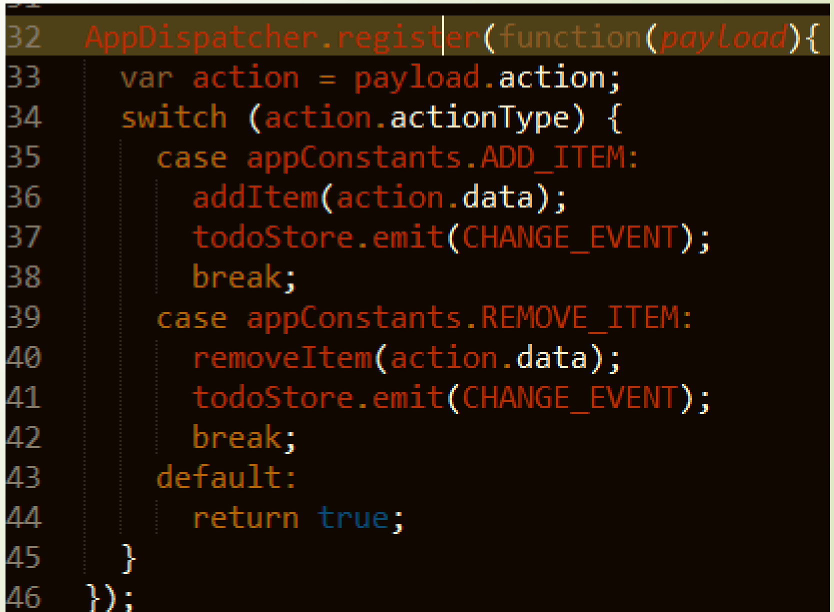
ReactJS {Flux - flow}



When an action creator provides the dispatcher with a new action, **all stores** in the application receive the action via the callbacks in the registry.

ReactJS {Flux - flow}

- **Stores**
  - Stores contain the application state and logic.
  - Manage the state of many objects.
  - Do not represent a single record of data like ORM models do.
  - Store registers itself with the dispatcher and provides it with a callback.

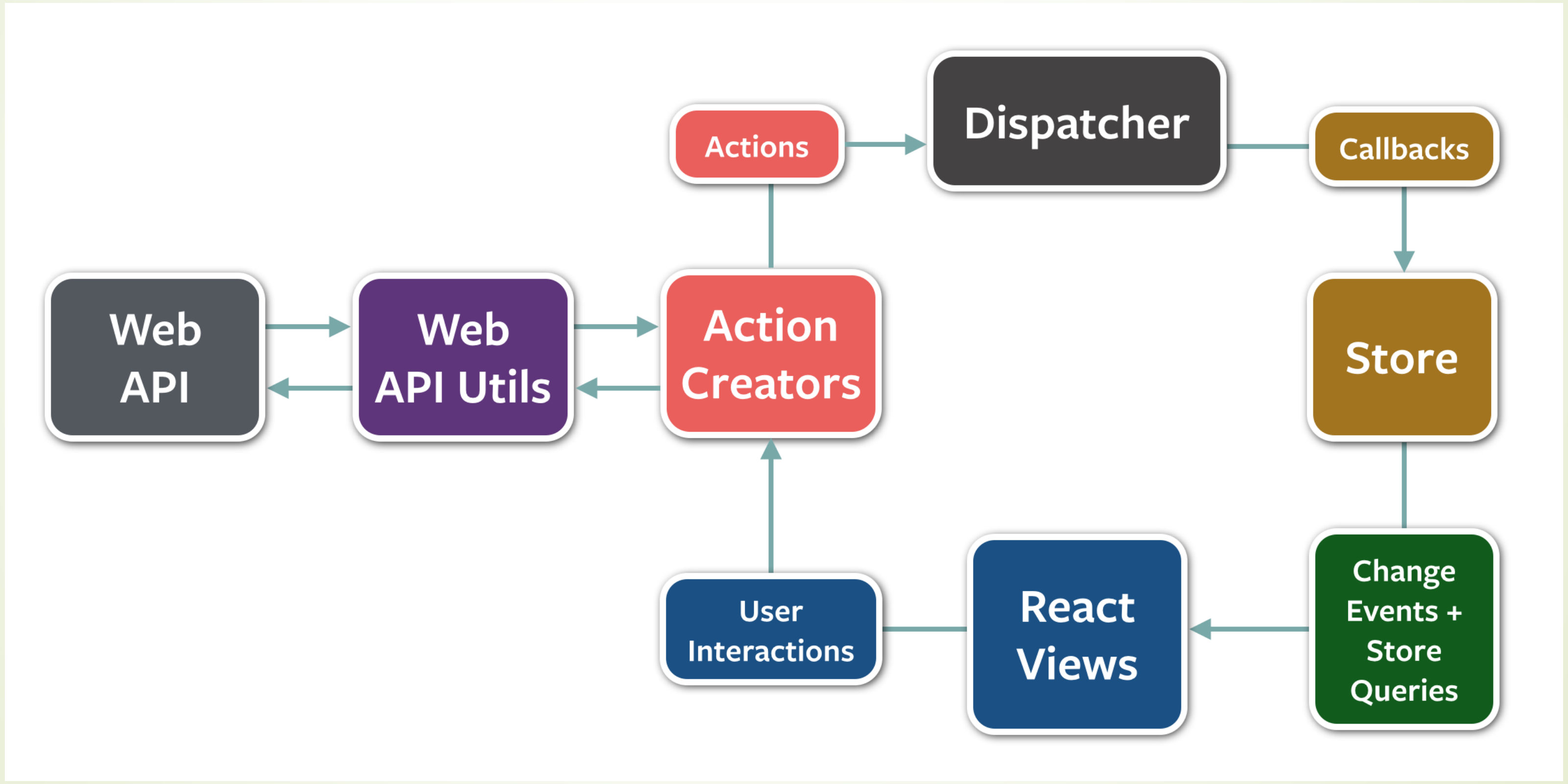


```
AppDispatcher.register(function(payload){
  var action = payload.action;
  switch (action.actionType) {
    case appConstants.ADD_ITEM:
      addItem(action.data);
      todoStore.emit(CHANGE_EVENT);
      break;
    case appConstants.REMOVE_ITEM:
      removeItem(action.data);
      todoStore.emit(CHANGE_EVENT);
      break;
    default:
      return true;
  }
});
```

ReactJS {Flux - flow}

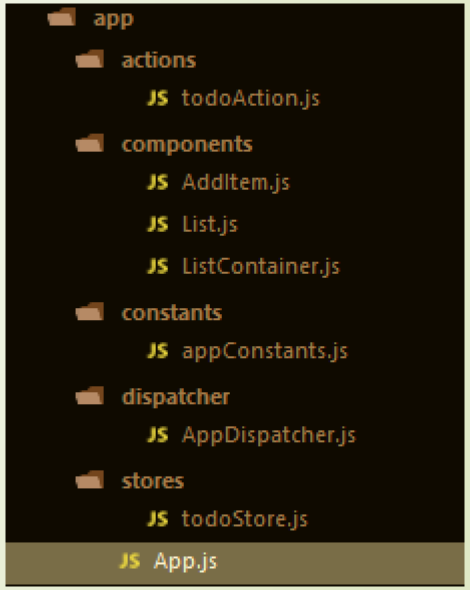
- **Views**
  - Typical React component.
  - After is mounted, it goes and get its initial state from Store and setup listener.
  - When it receives the event from the store, it first requests the new data it needs via the stores' public getter methods.
  - Then, it calls its own setState() method, causing its render() method and the render() method of all its descendants to run.

ReactJS {Flux - flow}



ReactJS {Flux - Implement}

- Flux is just an **architect**. So, you can design new framework by yourself base on this architect.
- Many JavaScript libraries help you implement flux like:
  - **Flux** (by Facebook: https://github.com/facebook/flux)
  - **Reflux**(by Mikael Brassman**:** https://github.com/reflux/refluxjs)
  - **Redux**(by Dan Abramov: https://github.com/reactjs/redux)
  - …



```
app
  actions
    todoAction.js
  components
    AddItem.js
    List.js
    ListContainer.js
  constants
    appConstants.js
  dispatcher
    AppDispatcher.js
  stores
    todoStore.js
  App.js
```

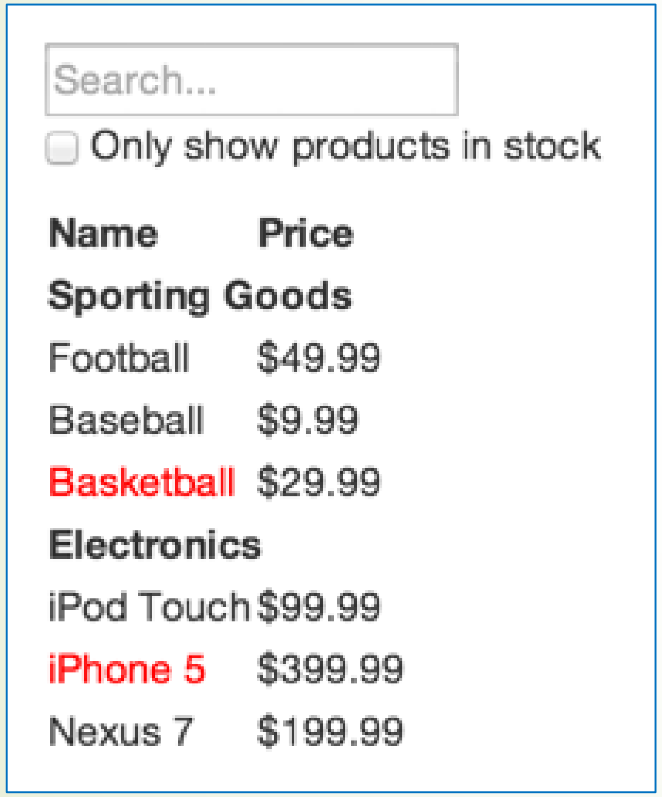
ReactJS {Flux – source code}

- https://github.com/tylermcginnis/Flux-Todolist

ReactJS {contents}

- JSX
- Virtual-DOM
- Props
- PropTypes
- State
- Refs
- LifeCycle
- Flux Architech
- Routing

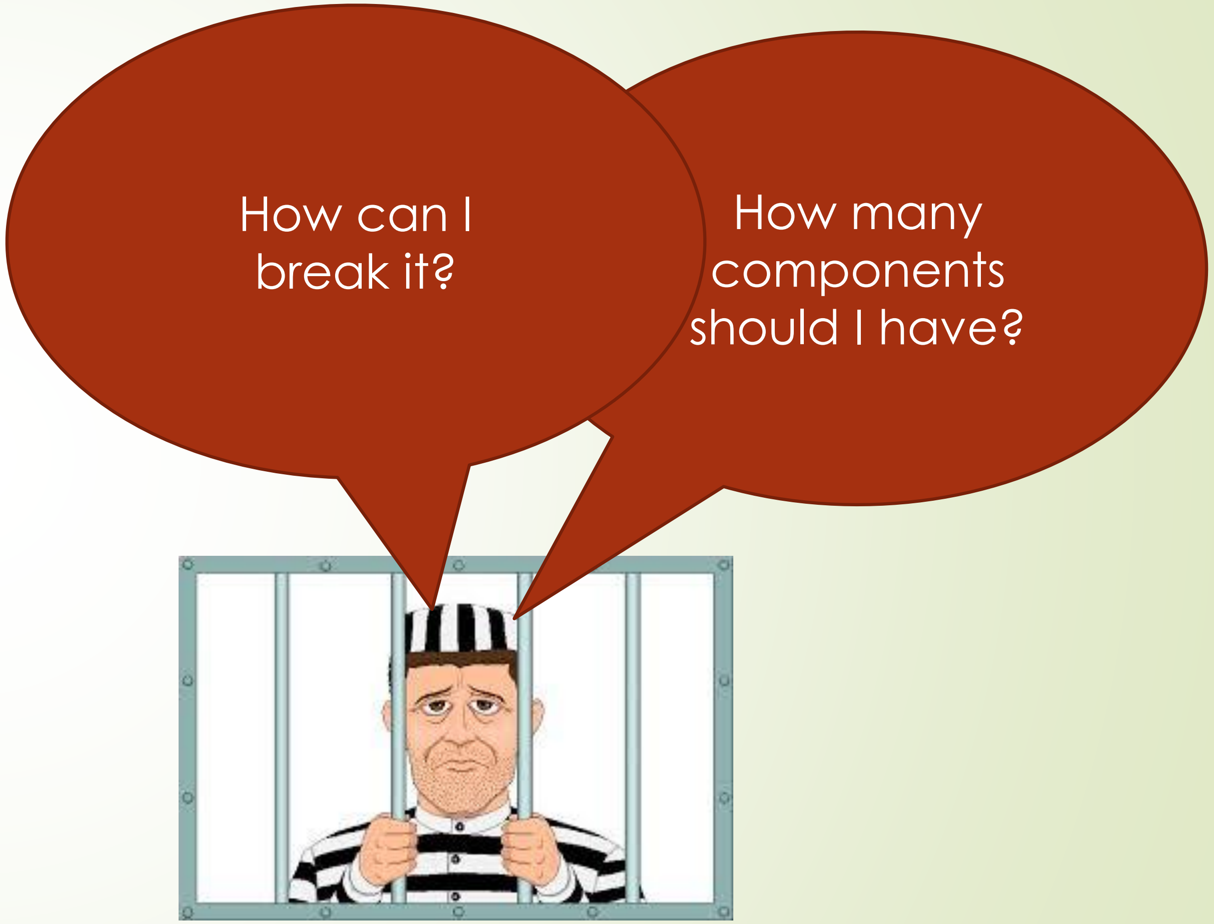
ReactJS { thinking in... }



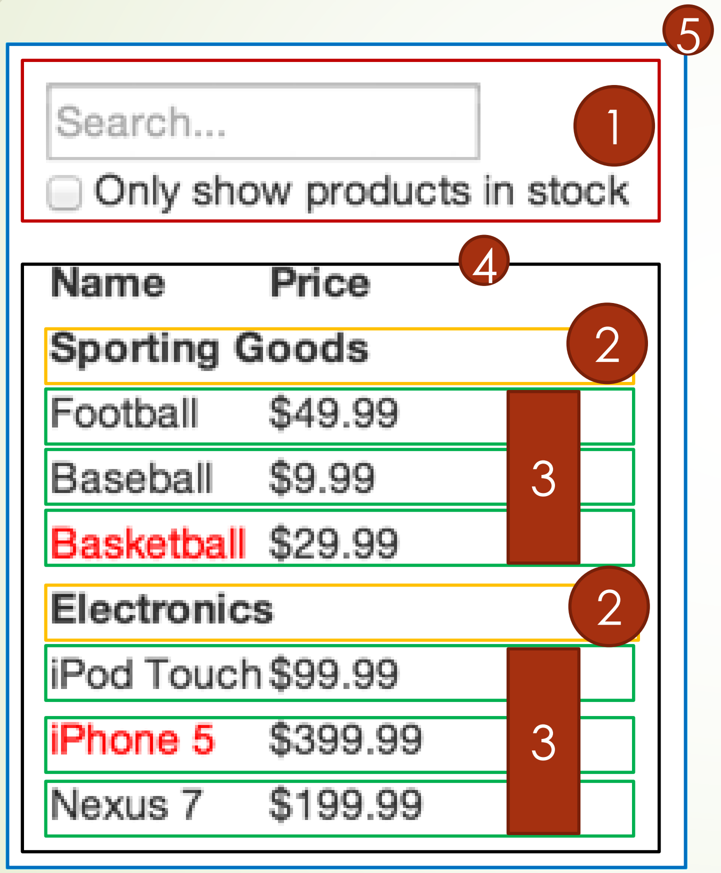
Search...

Only show products in stock

Name	Price
Sporting Goods	
Football	$49.99
Baseball	$9.99
Basketball	$29.99
Electronics	
iPod Touch	$99.99
iPhone 5	$399.99
Nexus 7	$199.99



ReactJS { thinking in… }



5 **FilterableProductTable**: contains the entirety of the example

1 **SearchBar:** receives all *user input*

4 **ProductTable:** displays and filters the *data collection* based on *user input*

2 **ProductCategoryRow**: displays a heading for each *category*

3 **ProductRow**: displays a row for each *product*

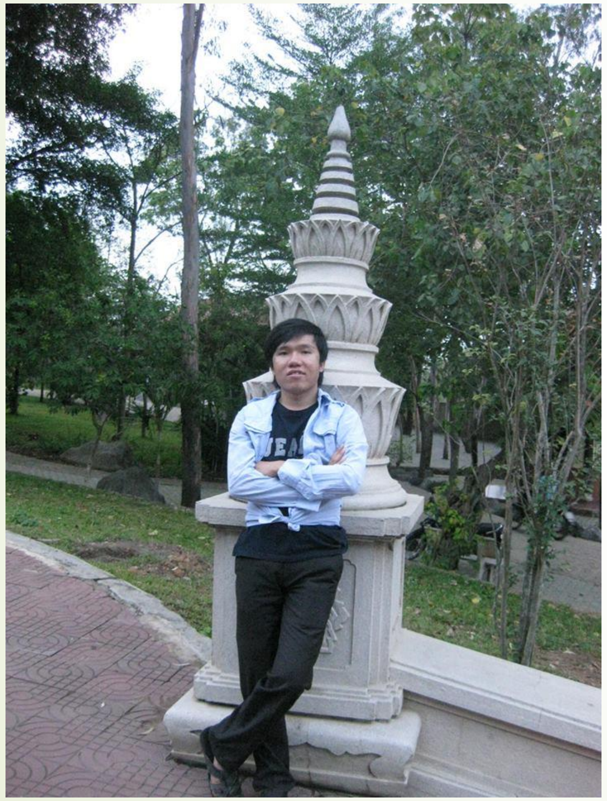
ReactJS { thinking in… }

ReactJS {Routing}

- Make UI consistent with URL.
- https://github.com/reactjs/react-router/blob/latest/docs.

ReactJS {references}

- http://tylermcginnis.com/reactjs-tutorial-a-comprehensive-guide-to-building-apps-with-react/
- https://facebook.github.io/react/docs/getting-started.html
- https://github.com/reactjs/react-router/tree/latest/docs
- http://teropa.info/blog/2015/03/02/change-and-its-detection-in-javascript-frameworks.html
- https://www.airpair.com/angularjs/posts/angular-vs-react-the-tie-breaker



@ThanhTuong
SE at KMS technology