React and Friends

Agenda

- The setting - WebPack and ES2016+
- Components
- State
- Props
- JSX
- Dealing with multiple components
- Handling State management across components
- Introducing Redux
- Other APIs of note (Thunk, Saga, Router)
- Where to get my samples

Setting the table

- We run React code in ES5+ browsers
- But we want to build good software
- So we transpile!
  - ES2015 (nee ES6) and ES2016 (nee ES7)
  - The Babel transpiler
  - WebPack, a development / production tool and server
- React 0.15.x is geared toward development in modern ES/JavaScript

What is React?

- A HTML Component Library
- Components contain
  - Read only, incoming properties (props)
  - Read/write component state (created for the component)
  - A template
- Components can be nested
- React provides
  - A built-in state change/detection API
  - JSX syntax to make writing render methods more efficient

The rest, as they say…

IS PURE MADNESS

History

(courtesy Wikipedia)

- Originally authored by Jordan Walke - software engineer at Facebook
- Began by running FB news feed in 2011, Instagram in 2012
- Released in March 2013
- Open Sourced using a 3-clause BSD license and a Facebook "patent troll protection addendum" - at JSConf US 2013

React and JavaScript versions

- React works fine in ES5
- But the React team embraces ES2015/16 syntax, transpiling to ES5
- New projects typically use ES2015/16 syntax (create-react-app project does this)

What is a minimal React component?

```
class Shell extends Component {
  render() {
    return React.createElement('span',
         null, // no attributes
        'Shel-lo');
  }
}
```

- This class has the only required function, render()
- It also calls createElement which is noisy and less efficient than JSX

The same component with JSX

```
class Shell extends Component {
  render() {
    return (<span>Shel-lo</span>);
  }
}
```

- Each component *must* contain a single outer element
- JSX has powerful variable matching and other features

The Case for JSX

- Most React developers use it
- Facilitates working with designers
- Easier to picture a layout by quickly scanning it

```
render() {
    let ids = [...Array(9).keys()];
    return (
      <div style={{width: "75px"}}>
        <hr/>
        {ids.map((index) =>
          <button key={index + 1}>
            {index+1}
          </button>)}
        <hr style={{borderColor:'red'}}/>
      </div>
    );
}
```

JSX Basics

- HTML-like
- philosophy: keep markup and display logic together
- Expressions use a single curly brace { and }
- attributes can be JavaScript expressions

Some JSX Fragments

Conditionally setting button text

```
<button>{dataIsNew ? 'Add' : 'Edit'}</button>
```

Commenting

```
render() {
  // We use JavaScript comments, not HTML comments!
  return(<div>{description}</div>);
};
```

Some JSX Examples

Inline styles (can be programmatically set)

```
<div style={{'color: red'}}/>
```

Mapping over a collection

```
<div className="list-group">
  { this.props.items.map((item) => {
      return <span key={item.id}>{item.name}</span>;
    });
  }
</div>
```

Using one component from another

Expose the child component

```
class Pebble extends Component {
   render() {
     <span>I am a pebble</span>
   }
}
```

Embed child in parent

```
class Container extends Component {
  render() {
     return (<div><Pebble /></div>);
  }
}
```

Passing data into a component via props

- `props` are read-only properties passed into a component
- The props are re-calculated if they are changed in the outer component hierarchy

```
class Pebble extends Component {
  constructor(props) {
    super(props);
  }
  render() {
    return(<span>{ this.props.color } Pebble</span>);
  }
}
```

Defining Prop DataTypes

- You can add this definition to the `Pebble.js` file (after the class) to validate any properties by type, and whether they are required:

```
Pebble.propTypes = {
    color: React.PropTypes.string.required
};
```

Component State

- Mutable data attached to a component is stored in `state`
- State should be initialized in the constructor synchronously

```
this.state = {
  favoriteColors: ['Blue', 'Red', 'Green']
};
```

Roles of State

- Hold unique application data
- Track application conditions
- Provide data backing a form
- Hold security tokens

Only define state if needed

- State management = overhead
- React must reconcile state changes with the UI
- If data can be derived as a read-only property, place it in props

Change tracking in React

- React watches state changes made by `setState`:

```
setState({
  rockColors: [ ...this.state.rockColors, 'Orange']
}, () => {
  console.log('state changed!', this.state);
});
```

- State changes can cause downstream props to change
- This is resolved automatically
- Note - state changes are *asynchronous*!!!

State changes can be additive

```
setState({
  geology: 'Igneous Rock'
});
```

- It adds the properties or overwrites their value
- Existing state is left alone

React components have lifecycle methods

- `componentWillMount()`
- `componentDidMount()`
- `componentWillReceiveProps(nextProps)`
- `shouldComponentUpdate(nextProps, nextState)`
- `componentWillUpdate(nextProps, nextState)`
- `componentWillUnmount()`

How are changes made?

- React uses a Virtual DOM API
  - Initial component DOM is virtualized
  - All changes during a cycle are made to the Virtual DOM first
  - The Virtual DOM adjusts the real DOM to fit all of the changes
- The Virtual DOM speeds up DOM updates
  - By only changing what actually needs to be changed
  - By not changing an element more times than necessary

React in context…

- React is a small core API, surrounded by a number of others

Constellation of APIs

- `Redux` - a state management API, following (loosely) the Flux pattern
- `React-Router` - a standard SPA router package
- `Flow` - a JavaScript static typing system
- `GraphQL` and `Relay` - Declarative object data stores and data fetch engine
- Developer tools and Chrome Plugins
- Much more...

Quick Demos

- Redux
- Router
- Synthesizer

Resources

- This session information (samples and slides by Sunday PM) - **https://chariotsolutions.com/libertyjs-2016-react/**
- Training - **http://chariotsolutions.com/course/react-friends-introduction-react-redux-react-router-supporting-apis/**
- Me - Twitter - @krimple
- Chariot Solutions podcasts - chariotsolutions.com/podcasts
- Chariot Solutions blogs - chariotsolutions.com/blogs

THANK YOU!