Real-World-React Workshop

Event Details: https://www.eventbrite.com/e/workshop-building-real-world-apps-with-react-tickets-21593560922?aff=eemailordconf
other resources:
  Slides and starter App: https://github.com/reactcg/react-workshop
  Base Starter for App:

INTRO

WHY REACT?

- Declarative (to add an new item to a list for ex you don't have to go and append it like you would in)
- Painless Iteration path (easy to bring React into stack/ integrates easily —> you can just start writing components like a button or something simple and it will integrate easily with other tech)
- Fun (easy to reason about things + hot coding <no need to refresh page to see changes in code>)

JSX?

*rich templating engine (build rich DOM or React native —> UI)

- Tag syntax (looks like HTML)
- First Class JavaScript (an extension of JavaScript)
- Has some gotchas but not too bad
- SYNTAX :

```
const myDivElement = <div className="foo" />; // create a div: <div class="foo">
</div>       --> self closing tag
ReactDOM.render(myDivElement, document.getElementByID('example')); // Render it to the DOM
by calling render ( helps you worry less about memory leaks )

const form = (
  <form action='/login' method="post"/> // note -
- same as html except you put everything inside this form = ( ... )
)

// You can make tags you want --> <DropdownMenu> for example
// <DropdownMenu>
//   ...
// </DropdownMenu>
```

You can have conditionals in your "html"
ex:
 { isAdmin && <link to='/admin' /> } —> curly braackets ===> JavaScript

Cannot write : class or for, no if-else (expressions only), must wrap consecutive tags (can't but an h1 and then a form —> youd need to wrap it all in a div)

Components in React?

*Groups of JSX and JAVASCRIPT

- React.createClass is how you create a component

```
import React from 'react';

const App = React.createClass({
  render()
  return (
    <div/> etc
    ...
  )
});
```

Three ways to create components :

1. Original React.createClass   * we will be using this method
2. ES6 class syntax, extend React.component
3. Functional

Benefits :

- Encapsulation —> markup and view related functionality tohether at last :
    - HTML(JSX), Derived values (no need to store data like a full name instead of first + last), event handlers (JSX lets you put them right on the tag), modularized, testable (because they are small you can test easily)
- Composability —> use anywhere
    - DRYer code, Utilities for UI (like you would write a function), Modularize/testable, large and vibrant ecosystem (ex. this presentation is actually built with React)

Rendering to the DOM

```
import { render } from 'react-dom';  // install React and react-dom and pull it into file
import App from './App.js ';
const el = document.getElementById('root'); // render at any DOM node anywhere

render(<App />, el);
// react will update the DOM instead of rewriting it
```

DATA

```
(data) => markup // your data is an input and the output is the markup rendered to DOM
```

PROPS

- component doesn't care where they came from
- important when integrating with frameworks (pass from existing app as props to React and then react can work its magic

```
import React from 'react';
import { render } from 'react-dom';

const Header = React.createClass({
  render()
  <div className="Header">
    <h1> {this.props.title}</h1>
  </div>
})

render(<Header title='My Super Header'>, document.getElementById('header'));
```

NOTE: ES6 classes don't support mixins this is why we write the classes in ES5 —> Depends on your use case wether you use ES5 and ES6

STATE

- Interactivity (if the user does something like type in a form this should be in states instead of props)
- Changes that happen over time (like async data fetching —> we want to update state to reflect data returned once it is done returning)
- exs :
    - value of an <input />
    - wether the UI is loading or not
    - Unread message count
- Think about your state as something not kept in the DOM

```

const Toggle = React.createClass({
  getInitialState() {
    return {
      isOn: false,
    };
  },

  toggle(e) {
    this.setState({
      isOn: !this.state.isOn, // toggle by setting to opposite of whatever value it is
    });
  },


  render() {
    const { isOn } = this.state;   // this is destructuring from ES6
    return (
      <div
        style=
{{ color: isOn ? 'green' : 'black' }}  // this comes from this.state ( just like this.props
 )
        onClick={this.toggle}> // on click we run this.toggle function
        {isOn ? 'On' : 'Off'}
    )
  }
});
```

Set State VS Replace State

- setState does a merge, assign, or extend and won't erase other info and React will recognize when this happens so if you don't use setState a lot of bugs will happen
    - for ex if we had multiple properties in the getInitialState and then used replace state we would override any other properties but with setState we are just changing what we want and leaving the rest alone
- Never assign or mutate explicity this.state > use the above syntax using the state API

When to use Props Vs State

Use Props as much as possible, state can change over time and your app can get into a weird state and you will get bugs
ie. if something isn't going to change, use props

Events

- attached directly to components
- aggregated by React
- Standardized across all browsers

note: you don't need to unbind or detach event handlers unless you manually attached handler on with a jQuery plugin or something

```
<button onClick={() => console.log('Clicked')}>
  Click Me!
</button>
```

Build a simple counter

Exercise 1
have a + / - that ups and down

Form Values and what you do with them

```
const FormValues = React.createClass({
  render() {
    return
    ... find code on github from slides

  }
})
```

More Events

- React standardizes events with SyntheticEvent
- Function just like you would expect
- Can ‘preventDefault', ‘stopPropagation' and access event ‘target'

```
import React from 'react';

export const FormValues = React.createClass({
  getInitialState() {
    return {
      val: '',
    };
  },

  handleSubmit(e) {
    e.preventDefault(); // this prevents browser from redirecting or reloading page
    const val = e.target.elements.input.value.trim() // e.target is the selector so you don't have to go searching it, give us the input
   // e.target.elements.input.value = '' ; would reset the input
    this.setState({ val }); // ES6 syntax
  },

  render() {
    return (
      <div>
      <form onSubmit={this.handleSubmit}> // check github slides for full code

    )
  }

});
```

Render a List

take input and put input into list of items

```
handleSubmit(e) {
  e.preventDefault();
  const input = e.target.elements.input;
  const val = input.value.trim();
  const { stuff } = this.state;
  const thing = {
    id: stuff.length,
    val,
  };

  //Add the thing
  this.setState({ stuff: [ ...stuff, thing ] });  // put the item in our data and the UI at
o-updates
  // this.setState({ stuff.concat( [thing] ) }); this is how we do it in ES5
},

reset() {
  // this.setState({ stuff: [] }); this is to reset stuff specifically but we might want to
 reset all of state
  this.setState(this.getInitialState()); // just call the initital state method and it will
 resets
},

render()
  return (
    <div>
    ...
    <ul>
      {this.state.stuff.map(thing => (
        <li key={thing.id}>{thing.val}</li>
        ))
  )
```

Array Methods

- Array.prototype.concat
- Array.prototype.map
- Array.prototype.filter
- Array.prototype.reduce

So you need to add a key property when you are adding elements to the DOM, use something like the ID on your data
ex: <li key={

```
//jQuery
//imperative coding not declarative like React
$("li").each(function(i, element) {
  $(element).html("This is element #" + i); // side effect
});
```

Ex in React

```
var shoppingList = [
  {id:1, val: 'food1'},
  {id:2, val: 'food2'},
  {id:2, val: 'food3'},
];

render() {
  var components = shoppingList.map(item => (
    <li key={item.id}>{item.val}</li>
  ));

  return (
    <ul>
      {components}
    </ul>
  )
}

//Better :
const shoppingList = [
  {id:1, val: 'food1'},
  {id:2, val: 'food2'},
  {id:2, val: 'food3'},
];

render() {
  return (
    <ul>
      {shoppingList.map(item => (
      <li key={item.id}>{item.val}</li>
      ))}
    </ul>
  )
},
```

Loading Async Data

```
import React from 'react';

const Item = React.createClass({
  render() {
    return (
      <li>Label: {this.props.item.label}
</li>   /// you can send things into Item in the render
    );
  },
});

export const AsyncLoader = React.createClass({
  getInitialState() {
    return {
      items: [],   // empty list to start
    };
  },

  // Check out the lifecycle docs
  // https://facebook.github.io/react/docs/component-specs.html   ******************
  componentDidMount() {     // Docs recommend that we use this componentDidMount for AJAX c
alls
    setTimeout(() => {      // this uses setTimeout because it is not a real world example
but it shows the async
      this.setState({
        items: [
          { id: 1, label: 'Item One' },
          { id: 2, label: 'Item Two' },
          { id: 3, label: 'Item Three' },
        ],
      });
    }, 2000);
  },

  start() {
    this.setState({ load: true });
  },

  render() {
    const { items } = this.state;
    return (
      <div>
        {items.length ? null : <h1>Loading...</h1>} /// if no items - 'its loading'
        <ul style={{ color: 'white', margin: '0 auto', width: 300 }}>
          {items.map(item => (                    // turn every item into an Item component f
rom above
            <Item key={item.id} item={item} />
          ))}
        </ul>
      </div>
    );
  },
});
```

ComponentDidMount

```
componentDidMount() {   // once react looks at JSX subscription and renders into the DOM th
en this fires --> setState will re-render so only if async
  $.getJson('/api/items', items => {    // this uses jQuery to call to the server
    this.setState({ items });
  });
}
```

Exercise: Fetch from the Reddit API

add .json to the end of any subreddit url
Take input from the user, make an Async request that displays the link to Reddit

Last Lecture - LEVEL UP

inputs:

- uncontrolled forms (classic HTML style)
- same as HTML5 forms
- value of input is immediately rendered

Uncontrolled Form : https://github.com/reactcg/react-workshop/blob/master/slides/level-

[up/assets/uncontrolled.example]

```
// http://jsbin.com/redigaxugu/edit?js,console,output
const FancyInput = React.createClass({
  displayName: 'FancyInput',

  handleSubmit: function(e) {
    e.preventDefault();
    console.log('submitted', e.target.myInput.value); // use name given in name prop on form -> input
  },

  handleChange: function(e) {
    console.log('value changed', e.target.value);
  },

  render() {
    return (
      <form onSubmit={ this.handleSubmit }>
        <input
          name='myInput'    // give this a name property that you can reference on your event handlers
          onChange={ this.handleChange }
          type='text' />
      </form>
    );
  }
});
```

Controlled Inputs

define the value of the field in your component

- using value and stat
- React will display componenet
- if value does not chnage, input filed does not change
- Manipulate the input feild using JavaScript

```
  // https://jsbin.com/gacomo/edit?js,console,output
const FancyInput = React.createClass({
  displayName: 'FancyInput',

  getInitialState: () => ({ inputValue: 'foo' }),

  handleSubmit: function(e) {
    e.preventDefault();
    console.log('submitted', e.target.myInput.value);
  },

  handleChange: function(e) {
    console.log('value changed', e.target.value);
    this.setState({ inputValue: e.target.value });  // without this React won't show you what you are typing even though its regestering
                                                    // to see this clearly, go to link at t
op of this code box and comment out line 14 where we set state
    //this.setState({ inputValue:               // this will change the way what you type looks, in this case 'f's will become 'x's
      // e.target.value.split('f').join('x') }); // production value of this is to add anchor tag when you come across a link to make it clickable in DOM
  },

  render() {
    return (
      <form onSubmit={ this.handleSubmit }>
        <input
          name='myInput'
          onChange={ this.handleChange }
          type='text'
          value={ this.state.inputValue } />
      </form>
    );
  }
});
```

Container Pattern

The idea is you do not put a bunch of fetching and rendering in one component
First: Container or Smart component does the fetching and data manipulation and then it passes down to the Dump (AKA Presentational) component

- dump (presentational) components render state
    - Only renders props
    - can render other dump components
    - Renders the same view for the same props
    - ex:

```
// dumb component
// https://jsbin.com/runuwi/edit?js,output
var Post = React.createClass({
  render: function() {
    var href = this.props.href;
    var title = this.props.title;
    return (
      <div className='Post'>
        <a
          href={ href }
          target='_blank'>
          { title }
        </a>
      </div>
    );
  }
});
```

- smart components
    - few or no view elements of its own

- may be responsible for data fetching
- may have internal state
- passes state down to child componenets through props
- ex of Smart:

```
// https://jsbin.com/tebegi/edit?js,output
var App = React.createClass({
  getInitialState: function() {
    return {
      posts: []
    };
  },

  componentDidMount: function() {
    fetch(ENDPOINT)
      .then(res =>  res.json())
      .then(json => json.data.children)
      .then(posts => this.setState({ posts: posts }) )
      .catch(err => console.error('There was an error fetching.', err));

  },

  render: function() {
    const { posts } = this.state;
    return (
      <div className='App'>
        { posts.length ? (
          <div className='posts'>
            { posts.map((post, i) => (
              <Post key={i} post={post} />
            )) }
          </div>
        ) : <h2>No reddit posts</h2>}
      </div>
    );
  }
});
```

BOTH TOGETHER:

```
// https://jsbin.com/tebegi/edit?js,output

var Post = React.createClass({
  render: function() {
    var href = this.props.href;
    var title = this.props.title;
    return (
      <div className='Post'>
        <a
          href={ href }
          target='_blank'>
          { title }
        </a>
      </div>
    );
  }
});

var App = React.createClass({
  getInitialState: function() {
    return {
      posts: []
    };
  },

  componentDidMount: function() {
    fetch(ENDPOINT)
      .then(res =>  res.json())
      .then(json => json.data.children)
      .then(posts => this.setState({ posts: posts }) )
      .catch(err => console.error('There was an error fetching.', err));

  },

  render: function() {
    const { posts } = this.state;
    return (
      <div className='App'>
        { posts.length ? (
          <div className='posts'>
            { posts.map((post, i) => (
              <Post key={i} post={post} />
            )) }
          </div>
        ) : <h2>No reddit posts</h2>}
      </div>
    );
  }
});
```

Talking to Parents

Use callbacks!

- children should not know about parent structure
- children can send information up throuh callbacks
- callbacks created by parents and passed down

EX with Filter

```
const FormContainer = React.createClass({
  getInitialState: function() {
    return {
      filter: ''
    };
  },

  updateFilter: function(e) {
    var newFilter = e.target.value;
    this.setState({ filter: newFilter });
  },

  render: function() {
    return (
      <MyForm
        filterValue={ this.state.filter }
        updateFilter={ this.updateFilter } />
    );
  }
});
```

Simple filter function :

```

```

Great Resources for React

- Immutable.js (won't allow for changing data structures)
- Redux (if state gets very complicated, its like a big container : when passing props gets to complex in scaling, look to Redux, implementation of Flux and its a way to manage state —> clean, simple API, mostly plain (f)unctions and Objects)
- Relay (UI happiness, lets you do async data requests easily, give you optimistic updates like when your user sends a new thig to the server and it hasn't responded yet, you can update the UI before the server does — DECLARATIVE)
- GraphQL ( )
- Rx.js (Library that builds on Observer Pattern, simple once you begin using it, makes your whole app reactive instead of saying, I'll get this stuff from here or there, it lets you open a pipeline of data flow —> ex: when you want to do something everytime user clicks somewhere on the screen , RXjs takes care of you)
- Webpack (it is in the buildpack from today — you can localize your css (eg. SaaS, Less) - powered the hot update)
- ES6 + Babel (browsers will eventually support it all — webpack does the ES6 compilation to ES5 using Babel)
- Egghead.io (Paid streamcast service — but they have a good free redux and rxJS and cycleJS(framework buillt using RxJS) screencasts)
- FreeCodeCamp —> good base sources where you can also get involved in building projects for charities
- http://tiny.cc/react-workshop