COLLEGIO TECHNOLOGIES INC.

REACT: INTRODUCTION AND SETTING UP YOUR ENVIRONMENT

WHAT IS REACT?

WHAT IS REACT?

- Written in JavaScript.
- Developed and maintained by Facebook.
- Primarily used as a front-end library for building user interfaces.
- Uses a component-based architecture.
- Used to drive applications of many major companies such as Facebook, Instagram, Netflix and Salesforce.

WHY USE REACT?

- Due to it's widespread adoption, it is projected to drive apps for a long time coming.
  - Better to be ahead of the curve than trying to catch up!
- There is a very large community out there, providing a lot of commonly used libraries and packages that save you the trouble of having to write tedious code.
- Even though it is just JavaScript, it is not just used to build web apps. In addition to web, React is used to build apps for:
  - Mobile (React Native)
  - VR (React VR)
- As mentioned above, *it is just JavaScript*. You don't need to worry about jumping around to different languages, as views, components and models are all built using JavaScript.

WHY USE REACT?

- React adopts a component-based architecture.
  - Component-based architectures ***hate*** repetition!

- Due to the nature of a component-based system, which relies on building reusable components to build out the app, there are a multitude of benefits:
  - Better and more efficient code scaling
  - Partial reloading of the page on change, saving valuable resources and allowing for speedy rendering
  - Cleaner and more focused code!

ENOUGH INFO ABOUT

REACT, LET'S GET STARTED!

WHAT DO I NEED TO WORK WITH REACT?

- A text editor
  - Your choice, a text editor is just a text editor. I will be using Visual Studio Code for any coding portions of the course.
- A command line tool
  - Terminal or iTerm2 for Mac, cmd for Windows
- Node.js
  - https://nodejs.org/en/download/
  - In addition to Node.js, you will also get NPM (Node Package Manager), which we will use for installing some additional command line packages
  - You can verify that these are installed by running the commands **node -v** and **npm -v**
- Yarn
  - https://yarnpkg.com/en/docs/install
  - **Windows Users:** You need to restart your machine after installing Yarn in order to use it!

THE REACT ENVIRONMENT

- Web Server
  - Once again, this can be your choice, but for any examples we will be using the live-server package.
  - To install: **npm install -g live-server**
- React (Obviously!) and ReactDOM
  - For our projects, we'll be using the React and ReactDOM CDN links
  - React is used to load the core React framework, while ReactDOM is used to render components to the web browser.
- Babel
  - https://babeljs.io/
  - Babel is the JavaScript compiler that we will use to convert our React code into browser-readable JavaScript.
- Webpack
  - https://webpack.js.org/
  - Webpack is a module bundler that we'll use to automatically run a bunch of tasks. However, for this lesson, it is not needed, and we will be covering it in future lessons.

EXERCISE: SET UP YOUR REACT ENVIRONMENT

- Take a few moments to set up React so that we can begin coding. Ensure that all of the following are installed:
  - Node.js
  - Yarn
  - live-server (omit if you're using something different)
- Create a new project folder. Inside of the folder, create another folder called ***public***.
- Inside of public folder, create an HTML file called ***index.html***, and a folder called ***scripts***. Create a file called ***app.js*** inside of the scripts folder, but leave it blank for now.
- Inside of your index.html file, create a simple webpage with a title, and a single paragraph to display.
- Add react to your page by adding the following two scripts to your index.html:
  - **<script src="https://unpkg.com/react@16.0.0/umd/react.development.js"></script>**
  - **<script src="https://unpkg.com/react-dom@16.0.0/umd/react-dom.development.js"></script>**
- Run your server, pointing it at your public directory, and ensure your text from index.html is showing.
  - In live-server, we run this with the command **live-server public** from the project directory

JSX (JAVASCRIPT XML)

- JSX, or JavaScript XML, is what is used to render HTML inside of your react applications.
  - Although it looks like HTML, it's actually JavaScript!
  - We'll be using JSX throughout this course to develop our applications, and really unlock why we use this over standard HTML.
- JSX is what's known as a *language extension*.
  - A language extension is used, as the name implies, to extend the features of a given language. Two other examples of language extensions are the CSS extensions *SASS* and *LESS*, which allow for different functions not available in CSS.
- Let's try it out! Inside of your project's index.html, add a div with id **app**, load up your app.js script just below React and ReactDOM, and add the following JavaScript to app.js:

```
var template = <p>Let's learn some React!</p>;

ReactDOM.render(template, document.getElementById('app'));
```

JUST IN CASE: PROJECT SAMPLE HTML SO FAR

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Free Agent Tracker</title>
    </head>
    <body>
        <div id="app"></div>

        <script src="https://unpkg.com/react@16.0.0/umd/react.development.js"></script>
        <script src="https://unpkg.com/react-dom@16.0.0/umd/react-dom.development.js"></script>
        <script src="/scripts/app.js"></script>
    </body>
</html>
```

WAIT A SECOND...WHAT JUST HAPPENED?

- It appears that there was an issue turning our JSX into JavaScript and HTML. This is because we haven't specified anything that we can use to actually perform this conversion.

- For our purposes, we use **Babel.js** to perform any Javascript transpiling for us.
  - Transpiling is the process of translating code written in one language to another language.
  - This is a feature that is required for ensuring backwards compatibility with older browsers that don't support newer languages, such as JavaScript ES6 and ES7.
  - We can develop using new techniques, yet still support old code!

- Although it can be used for more than this, we will be using Babel.js to transpile any ES6, ES7 and JSX that we create to ES2015 JavaScript, readable by every browser.

- Having trouble grasping this concept? No problem! The Babel.js website provides a live demo to show you exactly what your React code will compile into, check it out at https://babeljs.io/repl/.

BABEL.JS

- In order to use Babel.js inside of our project, we will first need to install the Babel command line tools (babel-cli), and the presets locally within the project itself.
  - The presets are used to define exactly what we are looking at and transpiling to.
  - You can review the presets available for Babel.js at babeljs.io/docs/plugins.
  - We will be using the *react* preset to transpile our JSX to HTML, as well as the *env* preset to transpile our ES6 and ES7 code into ES2015.

- You can install the babel-cli via npm: **npm install -g babel-cli**
  - Once it is installed, you can verify by typing **babel --help**

- Now that we have the babel command line tools installed, we'll install our presets via yarn:
  - inside of your project folder, type **yarn init**. This will initialize yarn inside of your project
  - install your presets by typing **yarn add babel-preset-react babel-preset-env**

EXERCISE: BABEL.JS

- Let's now use Babel to convert our React into plain JavaScript! Inside of your project folder, add a folder called ***src***.
- Inside of src, add a file called ***app.js*** (just like in public/scripts)
- Now, inside of the app.js file *inside of public/scripts*, copy the contents and paste them in side of *src/app.js*. The app.js located inside of public/scripts will be overwritten by Babel.
- Finally, in addition to your command line terminal running your server, open a new command line window, navigate to your project, and type the following:
  - **babel src/app.js --out-file public/scripts/app.js --presets=env,react --watch**
- If you refresh your browser, you should see your JSX show. It would be tedious to have to run babel all the time in a command window, but don't worry, we won't be for long. We'll eventually use Webpack to take care of this.

A FEW QUICK THINGS BEFORE WE BEGIN...

GIT AND GITHUB

- Any sample projects that will be created during this project will be stored on Github using git. Although it is not required to have knowledge of git, it is strongly recommended that you look into how to use Github if you don't know already.

- Github can be accessed via **git** CLI, or though several different downloadable applications, including the Github app itself.

- For more info on git and Github, please see the following links:
  - https://github.com/
  - https://guides.github.com/activities/hello-world/
  - https://www.crunchbase.com/organization/github

ES6 AND ES7

- We'll be using code from newer versions of JavaScript for this course, so knowledge of some of the newer features is required to fully interact with the course materials.
- However, fear not! We will go over any features that are needed in order to use React to it's maximum potential.

WORKING WITH JSX

JSX INTRODUCTION

- So far, we've only used JSX in it's most basic form with a single tag. Let's look more into how we can really use JSX to make some elegant and dynamic HTML:

- If you are adding more than one element to your JSX variable, it is good practice to separate out the elements, as you would regular HTML.

- In addition, although not required, it is good practice to encapsulate your JSX values in round brackets for readability.

```
var template = <div><h1>Let's learn some React!</h1><p>This is our paragraph!</p></div>;
    // Not very readable...

var template = (
    <div>
      <h1>Let's learn some React!</h1>
      <p>This is our paragraph!</p>
    </div>
);  // Much nicer!
```

JSX INTRODUCTION

- You can add more than one tag to a variable containing JSX, but keep in mind that there must always be a single root element.
  - This is commonly known as a *wrapper element*.
  - If you do not include a wrapper element, your code will throw an error.

```
var template = (
    <h1>Let's learn some React!</h1>
    <p>This is our paragraph!</p>
);  // WRONG!!!

var template = (
    <div>
        <h1>Let's learn some React!</h1>
        <p>This is our paragraph!</p>
    </div>
);  // RIGHT!!!
```

EXPRESSIONS IN JSX

- In order for our JSX to display dynamic HTML, we need to have a way to add values and expressions into our code. Let's begin with values:
  - You can inject values into your JSX by using curly brackets ({}). This method of dynamic value injection may be familiar for those who have worked with Angular and Laravel in the past.
  - You can also perform operations such as concatenation on any injected values. Basically, think of it as you can do anything inside of it that you could if you were printing a variable.

```
var myName = 'Rob';
var template = (
    <div>
        <h1>Hey {myName + '!'}</h1>
        <p>Let's learn some React!</p>
    </div>
);  // will show 'Hey Rob!' in the header
```

EXPRESSIONS IN JSX

- A few tips regarding JSX values:
  - You cannot directly reference an object inside of your JSX to print, as it will throw an error. You must instead reference specific properties of elements. In addition, you should traverse your array in order to see it's values (more on this soon!)

```
var user = {
    name: 'Rob',
    occupation: 'App Developer',
    skill_level: 'basic'
};

var template = (
    <div>
        <h1>Hey {user}</h1>
        <p>Let's learn some React!</p>
    </div>
);  // WRONG!!!

var template = (
    <div>
        <h1>Hey {user.name}</h1>
        <p>Let's learn some React, you can even learn with {user.skill_level} knowledge!</p>
    </div>
);  // RIGHT!!!
```

CONDITIONS IN JSX

- In addition to injecting dynamic values, we can also conditionally show specific values inside of our JSX.
- In order to conditionally render our JSX, we will first declare a condition, and inject it inside of our original JSX template.

```
var user = {
    name: 'Rob',
    occupation: 'App Developer',
    skill_level: 'basic'
};

function printWelcomeComment(level) {
    if (level == 'basic') {
        return <p>Remember to ask questions if you have any!</p>;
    }
    else {
        return <p>Let's learn some React!</p>;
    }
}

var template = (
    <div>
       <h1>Hey {user.name}</h1>
       {printWelcomeComment(user.skill_level)}
    </div>
);  // Will print the result of the above function
```

CONDITIONS IN JSX

- You can also use the *ternary* operator to conditionally display values as well.
  - Remember, the ternary operator is in the form: ***condition ? true statement : false statement***

```
var user = {
    name: 'Rob',
    occupation: 'App Developer',
    skill_level: 'basic'
};

var template = (
    <div>
        <h1>Hey {user.name}</h1>
        <p>{(user.skill_level == 'basic') ? 'Remember to ask questions!' : "Let's get started!"}</p>
    </div>
);  // Will print the result of the above function


ReactDOM.render(template, document.getElementById('app'));
```

CONDITIONS IN JSX

- Finally, we can use the **&&** operator to conditionally display something in cases where we want to either display some data or nothing at all.

```
var user = {
    name: 'Rob',
    occupation: 'App Developer',
    skill_level: 'intermediate'
};

var template = (
    <div>
        <h1>Hey {user.name}</h1>
        <p>Let's get started!</p>
        {user.skill_level == 'basic' && <p>Remember to ask questions!</p>}
    </div>
);  // Will only print the header and first paragraph
```

ATTRIBUTES IN JSX

- You can attach attributes to tags like you can in HTML. However, because JSX is still JavaScript, there are some differences between tag attributes in JSX and in HTML:
  - A lot of attributes have adopted a camel-case style of naming in JSX. For instance, what is normally ***autofocus*** in HTML would be ***autoFocus*** in JSX.
  - In addition, there are some attributes whose names are just not compatible with JSX. For these names, they have been modified to something that can be used to identify the attribute. The most prominent example of this is the ***class*** attribute: instead of **class**, in JSX you use ***className***.
  - For a complete list of supported attributes, please see https://reactjs.org/docs/dom-elements.html

```
let counter = 10;
const template = (
    <div>
        <h1>Countdown!</h1>
        <p>T-Minus: {counter} seconds.</p>
        <button id="countdown-button" className="btn btn-primary">Next</button>
    </div>
);
```

ARRAYS IN JSX

- Although you can directly inject an array inside of JSX, it isn't typically the way that you want to do it, as JSX will just render each item in succession without any spacing or formatting.
  - However, there is one exception to this: when we have an array of JSX. And, speaking of…

- In addition to the standard values, arrays can also be comprised of a list of JSX values.

```
const arrayOfJSX = [
    <p>My first paragraph!</p>,
    <p>My second paragraph!</p>,
    <p>My third paragraph!</p>
]
```

SIDE NOTE: THE KEY ATTRIBUTE

- You may notice that if you plug in the previous array, you'll see React complaining about a unique "key" prop.
  - This is because, in order for React to be as efficient as possible when rendering and re-rendering the user display, it needs to be able to uniquely identify each element inside of a JSX variable.
  - The key attribute is what helps React identify unique elements to re-render.
  - Typically, in a system where data is pulled from a database, the key is the row ID.

```
const arrayOfJSX = [
    <p key="first">My first paragraph!</p>,
    <p key="second">My second paragraph!</p>,
    <p key="third">My third paragraph!</p>
];
```

ARRAYS IN JSX

- The typical use-case with arrays in JSX is to perform some functionality on each array item, and then send back some data if needed to re-render. We will see this as we go through our list of players in the free agent tracker.

- We will be doing this by using the ES6 **map** function.

```
const roster = ['Sidney Crosby', 'Evgeni Malkin', 'Phil Kessel'];
const template = (
    <ul>
        { roster.map((player) => <li key={player}>Name: {player}</li>) }
    </ul>
);
```

JSX AND EVENTS

- One of the core requirements to any user-interface framework is the ability to handle user and system triggered events.
- All standard user-triggered events are available for use inside of JSX tags; they adopt the camel-case property that was mentioned earlier.
  - For instance, in order to declare a click event, you would use the ***onClick*** JSX attribute.
  - You will pass the event attributes as declared functions. **Please Note:** Moving forward, we will be using arrow functions for this, so if you are having trouble grasping this concept, please see the additional ES6 video located in the resources.

JSX AND EVENTS EXAMPLE (HMMM....)

```
// create an initial counter variable
let counter = 10;

// create the event function
const countdown = () => {
    counter--;
    console.log(counter);
};

// use the event arrowfunction inside of the button
const template = (
    <div>
        <h1>Countdown!</h1>
        <p>T-Minus: {counter} seconds.</p>
        <button onClick={countdown}>Next</button>
    </div>
);

ReactDOM.render(template, document.getElementById('app'));
```

JSX AND EVENTS

- If you run the code in the previous example, you will notice that the counter displayed on the screen actually isn't changing.
- This is because data-binding is not present inside of JSX, like it is with Angular.
  - We will be going over how we can do this when we go over components, and how React's virtual DOM re-renders the user display very efficiently.
  - For now, we will encapsulate our main JSX into a function and re-run it whenever we need to refresh.
- You can see the complete list of events available to use in React at https://reactjs.org/docs/events.html

JSX AND INPUTS

- Now that we've seen how events will work, let's see how we can get inputted data from the user.

- Inside of your JSX, you will encapsulate all of your input data inside of a ***form*** element, similar to HTML.
    - Like with the click event, you can listen for event submission by adding the ***onSubmit*** attribute to your form tag inside of your JSX.

- See https://reactjs.org/docs/events.html#form-events for the list of form-specific events.

```
const formDidSubmit = (e) => {
    e.preventDefault();

    const theName = e.target.elements.yourName.value;
    if (theName) {
        console.log(theName);
    }
};

const template = (
    <div>
        <h1>My Form!</h1>
        <form onSubmit={formDidSubmit}>
            <input type="text" name="yourName" />
            <button>Submit Your Name</button>
        </form>
    </div>
);
```

JSX: CONCLUSION

- That's it! The basics that we'll need to work with JSX in React! As mentioned a few times earlier, if you are not familiar or having trouble with ES6 arrow and map functions, take some time to review the supplemental video.

- Let's complete a quick exercise to drive some of these concepts home, and when we're done, we'll look at component-based architecture and how we can use it inside of our applications.

EXERCISE: JSX (30 MINUTES)

- Let's take a moment to create the Free Agent Tracker in it's most basic form! We'll create a very basic app (no styles needed), and inside of the app, we'll include the following:
  - A list of free agents
  - A form to allow the addition of new free agents. New free agents will have a name and gender.
  - The ability to reset the list of free agents.

RESOURCES

- **Node.js:** https://nodejs.org/en/download/
- **Yarn:** https://yarnpkg.com/en/docs/install
- **Babel.js:**
  - **Official Site:** https://babeljs.io/
  - **Babel.js Live Demo:** https://babeljs.io/repl/
  - **Babel.js Plugins:** babeljs.io/docs/plugins
- **Webpack:** https://webpack.js.org/
- **Github Links:**
  - https://github.com/
  - https://guides.github.com/activities/hello-world/
  - https://www.crunchbase.com/organization/github
- **React:**
  - **DOM Attributes:** https://reactjs.org/docs/dom-elements.html
  - **Events:** https://reactjs.org/docs/events.html
- **Lesson Code:** https://github.com/collegio/intro_react_jsx
- **Lesson Video:** https://youtu.be/AmxwuxTlmcw
- **ES6 Examples Video:** https://youtu.be/654C3l3Vawg

DISCLAIMER

- Copyright 2018 Collegio Technologies Inc.
- All content within this presentation has been compiled by Collegio Technologies Inc. All material has been sourced through open source resources and source code and as such conforms to Canadian copyright laws and regulations. All rights reserved.