1. Introduction to React

1.1. Introduction: Why are we learning ReactJS?

- **Why we learn ReactJS**
  - The main reason this course targets React is because a beginner level programmer has a much higher percentage of success rate to penetrate the market if they know about React.
  - React is not that hard to learn.
  - There is a high demand for React developers now. It is a relatively new technology, and most people, including developers, do not know about it yet.
  - **Note**: Everything we have learned so far was laying the groundwork to become a React developer. The combination of the above reasons gives you a better chance of being a competitor in the market if you become a decent React developer
- **Why is React in demand?** There are so many open-source platforms for making the front-end web application development easier, like Angular. There are multiple reasons why everyone is shifting to the React.js library. Below are some of the main reasons:
  - **Simplicity**: React's component-based approach, well-defined lifecycle, and use of just plain JavaScript make React very simple to learn, build a professional web (and mobile applications), and support. React uses a special syntax called JSX, which allows you to mix HTML with JavaScript. This is not a requirement; the Developer can still write in plain JavaScript, but JSX is much easier to use.
  - **Performance**: React allows developers to create large web applications that can alter data on the browser in a more performance-efficient way. This better performance is due to React's

faster DOM manipulation. We know that extensive DOM manipulation slows performance because the entire DOM object is updated for a simple change in part of the DOM. React solved this using a virtual DOM and by updating only the part of the ReactDOM that has changed, not the entire DOM.

- **Reusability of code**: React allows us to create reusable UI components. Reusing components will save time for developers as they do not have to write various codes for the same features.
- **Speed**: Better speed in the development process because different developers can write individual parts on both client-side and the server-side.
- **Integration with other libraries**: React can be used with a combination of other JavaScript frameworks and libraries because it conveniently integrates with them.
- **Simple to learn**: React is less complicated to learn (specifically when compared to Angular.js).
- **Native Approach**: React can be used to create mobile applications (using React Native). React is now upgraded and can be used for developing mobile native applications for both Android & iOS platforms.
- **Testability**: ReactJS applications are super easy to test. React views can be treated as functions of the state, so we can manipulate the state we pass to the ReactJS view and look at the output and triggered actions, events, functions, etc.

1.2. What is ReactJS?

- **What is React.js?**
  - **Definition 1**: React. js is a JavaScript library that is used for building part of a website/app that the user interacts with, also known as a

user interface. This means that React is used for front-end development and runs in the browser. We have already used another JavaScript library earlier in this course, the jQuery library.

- **Definition 2**: React is a JavaScript library that aims to increase the performance of applications and simplify the development of visual interfaces. React.js is used for handling the view layer of the full MVC stack for web and mobile apps.

- **Why is React called the view in the view layer of MVC?**
  - **MVC (Model-View-Controller) framework**: MVC is an architectural/design pattern in software design that separates an application into three main logical components: Model, View, and Controller. In its simplest form, it can be as simple as three folders, each dedicated to store the Model, View, and Controller components of an application in an organized manner.
    - **Model component**: This component stores all the data and the logic of our website/app
    - **View component**: This component defines how the website/app should display the data to the user. In short, this component generates part of the website that the user views or interacts with.
    - **Controller component**: This component, upon mouse/keyboard inputs from the user, orders the Model to update its data and then notifies the View component to display the updated data. In short, this is a component that serves in the middle between the Model and Controller components by receiving and sending data from the View component to the Model component and vice versa.

- **Why React is called the View in the MVC layer**: React is usually considered as the View layer of MVC because it is used to build and render part of a website that the user interacts with. In short, React only focuses on the View component.

1.3. How does ReactJS work? What makes React faster?

- **The “react" library and the “react-dom” package**: There are two main React libraries we use to build the UI components of a web application:
  - **The “react” library**: React is a JavaScript library that lets you develop a top user interface/UI for both web and mobile applications. It also manages/tracks changes
    - **Why import React from the “react” library?** We import React because the JSX code we write in our React component must be converted into vanilla JavaScript. Even if it is Babel that does the conversion of your JSX into vanilla JavaScript, Babel needs to use the createElement method available under the React (React.createElement method).
    - **No need to import React from “react” for simple components**: React has introduced a new JSX transform with the release of React 17, which automatically transforms JSX without using React.createElement. This allows us to not import React, however, you will need to import React if, in your component, you use Hooks and other exports that React provides
  - **The "react-dom" package**: The "react-dom" is a package provided by the “react” library and is responsible to render UI in the browser. "react-dom" provides DOM specific methods that can be used at the top level of a web app to enable an efficient way of accessing and

modifying the DOM. Note: Most components do not need to use the "react-dom" package. "react-dom" provides an API containing the various methods like render(), findDOMNode(), and unmountComponentAtNode(). The most important of all the ReactDOM's methods is the render() method.

- **render() method from "react-dom" package**: This function/method is used to render a single React Component or several Components wrapped together in a Component or a div element. render() function is the point of entry where the tree of React elements is created and updated.

- **What is ReactDOM (Virtual DOM)**? What makes React efficient is the fact that it uses the React Virtual DOM (ReactDOM) to handle changes on DOM. Let us explain what we mean by that step by step
  - **Real DOM's manipulation is slower for performance**: To refresh your phase 2 classes on DOM, Document Object Model (DOM) is a tree-like representation of your website that gets loaded into the browser. DOM represents the web page (the document containing HTML, CSS, JS) as an object model with properties and methods. We use these properties/methods to select and change/manipulate the elements on the document.
    - **Why real DOM manipulation is slow?** Every time there is a change in the document (change can be CSS whereby an element's background color changes to "yellow"), the updated element and its children must be re-rendered. This is a very resource-intensive process because the entire HTML and CSS need to be recreated and re-rendered, again. It is this re-rendering of the entire document object (even for a slight change in the document) that makes real DOM's manipulation slow. React has a different way of handling

changes on DOM, and this is where Virtual DOM comes in handy.

- **React's virtual DOM manipulation for better performance**: React starts by creating a replica of the real DOM, called Virtual DOM (VDOM or ReactDOM).
    - **VDOM (ReactDOM)**: It is the virtual representation of the real DOM. Every time there is a change in the DOM, it is the virtual DOM that gets updated, not the real DOM. This replica is just a JavaScript object that contains the necessary description of what we want to be added on the DOM later.
- **How does React keep track of changes in DOM?**
    - 1/ When new elements are added to/removed from the UI, React starts by creating a replica of the Real DOM, the virtual DOM. We said that the virtual DOM is represented as a tree, and each element is a node on this tree. If the state of any of these elements changes, a new virtual DOM tree is created. **Note**: React always keeps the previous version of the virtual DOM before change.
    - 2/ Then the current virtual DOM is compared/diffed to the previous virtual DOM compares. **Diffing**: Comparing the difference between a previous version of the virtual DOM with the current version of the virtual DOM. **Note**: React uses an efficient diff algorithm to compare the versions of the virtual DOM.
    - 3/ Then the virtual DOM identifies the difference between the 2 virtual DOM versions.
    - 4/ Once it identifies the difference, the virtual DOM calculates the best possible method to make these changes to

the real DOM. It then goes back to the real DOM and only updates the part that is changed.

- **Why DOM manipulation fast in Virtual DOM?**
  - **Only the object that changed will be updated in the real DOM**: When there is a change between the previous version of the virtual DOM and the current Version of the virtual DOM, React first updates the current virtual DOM. It then goes to the real DOM and updates ONLY the specific object that has changed. In short, there will be no re-rendering of the ENTIRE document when there is a change in the DOM, rather, only the change (which is the difference between the 2 versions of the VDOM) will be inserted into the real DOM.
  - **React follows batch updates to update the real DOM**: In React, updates to the real DOM are sent in batches instead of sending updates for every single change in state. Re-rendering the entire DOM object is the most expensive part, and React efficiently ensures that the real DOM receives only batched updates to re-render the UI.

1.4. Generating React elements using vanilla JavaScript vs generating React elements using React(JSX)

- **Vanilla/plain JavaScript**: It is simply JavaScript written without libraries like React or jQuery.
- **React element(s)**: React element is the smallest entity in react. React element is an object representation of a DOM element in memory. It's important to note that a React element isn't actually the thing you'll see on your screen. Instead, it's just an object representation of what you see on your screen. React elements can be created using vanilla JavaScript or React (JSX).

- **Generating traditional DOM vs generating React DOM**:
  - **Traditional DOM is generated from HTML code**: Traditionally, the DOM is generated from an HTML code that is passed to the browser directly.
  - **React DOM is generated from JSX**: To improve the efficiency of recreating DOM, React decided to generate the HTML using JavaScript (JSX) and manage it as JavaScript object instead. In other words, when you build your application using React, you create your HTML using JSX, a markup that allows us to write HTML in JavaScript by converting HTML tags into React elements. JSX is not a requirement for using React.
  - **JSX is not a requirement for using React**: Each JSX element is just syntactic sugar for calling React.createElement(component, props, ...children). So, anything you can do with JSX can also be done with just plain JavaScript. **Note**: Using React without JSX is especially convenient when you do not want to set up compilation in your build environment. To help us understand this concept, let us start by building a simple page where we create the HTML using palin JavaScript.
- **Steps to generating React elements using vanilla JavaScript**: Let us use React & vanilla JavaScript to build a simple JavaScript application
  - 1. Create an HTML document
  - 2. Include “react” library and “react-dom” package into your HTML document
    - **Note**: Both React and ReactDOM are available over a CDN under the following link
    - https://reactjs.org/docs/cdn-links.html
    - **Why include “react” library**? We included this library because it has a method called “createElement()” that is

needed to create elements. **Syntax**: createElement(type, {props}, children)

- **The "type" parameter**: This represents the type of the HTML element. Example: <h1>, <p>, etc.
- **The "{props}" parameter**: This represents a JavaScript object containing the properties required to construct the element. **Example**: {style: {size:10px}} or {className: "firstPar"} or {Eventhandlers}, etc.
- **The "children" parameter**: This represents any child/nested element we want to create

- **Why include the react-dom" package**: We included this package in our HTML because "react-dom" has a method/function called "render()" that will stick/render the created React elements into the DOM to be displayed by the browser.

  Syntax: ReactDOM.render(element, containerElement)

  - **The "element" parameter:** The element that needs to be rendered in the DOM
  - **The "containerElement" parameter**: Where to render in the dom

- 3. Create a JavaScript file where you will write your React logic.
- 4. Include your JavaScript file in your HTML. Make sure to always include custom JavaScript files below any JavaScript library you include in your HTML.
- 5. Go to your JavaScript file and create a to hold the title of your document, "ul" and 3 "li" elements as children of your <ul> and display them on the browser when you open your HTML in your browser.

// index.html

```
<body>
 <div id="myDiv"></div>
<!-- Include React library/CDN  here
Include reactDom package/CDN here
 Include your cutom.js here            -->
</body>
```

// custom.js

```
var parElement = React.createElement("div",
{ style: { backgroundColor: "pink" } },
"I am a div for title",
 React.createElement("ul", null,
 React.createElement("li", null, "First child of ul"),
 React.createElement("li", null, "Second child of ul")
 ),
    React.createElement("div",    null,    "Div    to    hold    2nd
paragraph")
 );
   ReactDOM.render(parElement,    document.querySelector("#
myDiv "));
```

- **For further explanation**, please watch the class demo on how to generate React elements using vanilla JavaScript.

- **Generating React elements using JSX**: As we have seen above, we can create React elements using plain JavaScript. We can also create React elements using React (JSX).
  - **Why JSX preferred to create React elements?**
    - Creating React elements using vanilla JavaScript takes time.
    - Creating React elements using vanilla JavaScript will not allow us to use the benefit of React's virtual DOM. This means that every time there is a change to the DOM elements, the entire DOM object needs to re-render, which slows down the performance of our website. Therefore, using the React library and JSX to create React elements provide us a way to manage changes in the DOM without the need to re-render the entire DOM as the React library has its own efficient mechanism of handling it.
    - JSX also allows React to show more useful error and warning messages.
- In the coming sections, we will discuss how to set our React development environment and use JSX to create React elements.

1.5. Getting started with React development: installing React, configuring npm, using create-react-app

- **Install Node.js**: By this time, we can safely assume that you all have Node installed on your computer. Therefore, no need to download and install Node on your computer again. By now, we also know that "NPM" comes with Node
  - **Why Node**: When we created React elements using vanilla JavaScript, we ran our code on the browser. However, when we create React elements using the React library, we will not use HTML

code, but JSX to write our HTML. Browsers cannot run our JSX, therefore, we will use Node as a runtime environment to run our JSX code

- **Using "create-react-app"**: "create-react-app" is a tool (built by developers at Facebook) that gives you a massive head start when building React apps. It saves you from time-consuming setup and configuration. You simply run one command, and Create React App sets up the tools you need to start your React project. If not for this tool, you would need to install and set up Babel, Webpack, and other modules just to get started
- **Create a React app**: The first thing you need to do every time you create a React project is to create a react app to help you set up the initial structure of your project.
  - **Syntax to create a React project**: npx create-react-app nameOfProject
  - Running this command will create a project folder with create-react-app, install the latest versions of React and React-DOM libraries, install the latest version of react-scripts, install all the dependencies (libraries/packages that your React app will depend on to properly function), and create files that will help you track these dependencies
  - **Syntax to create an app**: npx create-react-app yourProjectName
    - **Example**: npx create-react-app myfirstreactapp
- **Change your directory to go to the React app you just created**: cd myfirstreactapp
- **Run/ start your React app by typing the following command**: npm start

- A new browser window will open with your newly created React App. If not, open your browser and type localhost:3000 in the address bar.
- **Note**: When running "npm start", if at any time you get a message stating "*... Something is already running on port 3000. Would you like to run the app on another port instead? › (Y/n)...*", please just type "y" without the quotations to basically say "yes" and allow your React App to listen to another port.

1.6. Getting started with React Development: understanding the file structure of a React app

- **Your project folder and project folder name**: Your project name is created when you run "npx create-react-app my-first-react-project" in your terminal. **Note**: Your project name has to always be in lowercase.
- **The node_modules folder**: This folder contains folders of all the required dependencies & packages needed to build your React app
  - **Dependencies**: External/third-party code/software that your project depends on. **Example**: You will see a folder for Bootstrap in your node_modules folder if your React project implements Bootstrap
- **The public folder:** It contains static files such as index. html, JavaScript library files, images, and other assets
  - **Index.html:** index.html file in the public folder is the root file that displays when we run "npm start" and when the React app opens in the web browser. You must not delete this file.
  - This will be the only html file in the entire react application since React is generally written using JSX. Also, this file has a line of code . This line is very significant since all the application components are loaded into this div.

- **The src folder**: It is your working folder where you put all the JavaScript files/components, CSS files, images, and other assets for your project.
- **App.js**: This is the file for App.js component. App.js component is the main component in React, which acts as a container for all other components.
- **Index.js**: This is a JavaScript file corresponding to our index.html. It is an entry point of react app. This means all your components are imported to this file and rendered through this file to the index.html.
  - All of our modules are rendered here by reactDOM's render method that takes 2 arguments, our components (contained under app.js) and the place where those components are going to be rendered at, the root html.
  - ReactDOM's render method in index.js takes our components and inserts them to the root index.html file.
- **App.css**: This is a CSS file used to provide styling for our App.js component
- **Package.json and package-lock.json files:** These files are created by npm to manage the installed dependencies you use for your React app.

1.7. What is JSX (JavaScript XML)?

- **What is JSX? (JavaScript XML)**:
  - JSX is an extension to the JavaScript language created as syntactic sugar for React.createElement(). It is a syntax created by React developers to create HTML using JavaScript. In short, JSX makes it easier to write HTML in React and finally place them in the DOM without using React.createElement().
  - JSX follows HTML's syntax and rules of XML.

- **What is XML?** It is a Markup language, just like HTML. It is just a way of tagging data in an organized format. Unlike HTML, it does not have any pre-defined tags. Meaning that the tags are created by the writer. The only thing it has is a set of formatting rules. These are XML's syntax rules:
  - **XML documents must contain one root element**: There has to be a parent element containing all other elements inside of it.
  - **All tags must be closed**: It is illegal to omit the closing tag. All elements must have a closing tag.
  - **All tags must be properly nested**:
  - **Tag names are case sensitive**: Opening and closing tags must be written with the same case. **Example:** The tag <Letter> is different from the tag <letter>.
  - **Tag names cannot contain spaces**. However, element names can contain letters, digits, hyphens, underscores, and periods.

- **Compare a JSX code and an XML code**:
  - **Example of JSX Code**:

```
<div>
  <div className="abebe">
    <ul>
      <li>Almaz</li>
      <li>Kebede</li>
    </ul>
  </div>
</div>
```

- **Example XML Code**:

```
<class-room>
  <student country="USA">
  <SEX>M</SEX>
  <age>31</age></student>
</class-room>
```

- **Difference between HTML and JSX**: Even though JSX is very similar to HTML, notice these differences.
  - **JavaScript's reserved keywords cannot be used in JSX**: For instance, class property in HTML cannot be used in JSX and is thus replaced by “className” in JSX. Similarly, the “for” attribute in HTML is “HTMLFor” in JSX, since "for" is one of the most common JavaScript reserved keywords.
  - **In JSX, you must return a single parent element, or it won't compile**: Nested JSX must return one element, which we’ll call a parent element that wraps all other levels of nested elements. However, in HTML, multiple elements can be returned.
  - **In JSX, it is possible to write JavaScript directly**: The minute you wrap your code in curly braces {} in JSX, React will understand it to be JavaScript code. Whereas in HTML, you need a script tag or an external JavaScript file to implement JavaScript.
  - **All HTML attributes and event references in JSX must be in camelCase**: In HTML, you do not have to use camelCase for attributes, IDs, and event references. In JSX, HTML attributes and event references must be in camel case, and that is why the “onclick” event becomes “onClick” in JSX.

    - **All Tags Self-close in JSX**: In HTML almost all tags have an opening and a closing tag. In JSX, however, any element can be written as a self-closing tag, for example, , but it is not advised to do so. Your React app will not compile if you have a
 because there's no forward slash before the right-angle bracket of the line break tag
- **Expression/code inside curly braces { } in JSX:** When you wrap an expression/code inside curly braces in JSX, you are saying that the expression is a JavaScript variable, or property, or any other valid JavaScript code. In short, we use curly braces in JSX to embed a JavaScript expression
    - In React, curly braces are a special syntax to let the JSX parser/interpreter know that it needs to interpret the contents in between the braces as JavaScript instead of a string
- **Browser doesn't understand JSX. What happens behind the scenes when we write JSX code?** When we write code in JSX, it is converted "behind the scenes" into JavaScript during runtime. The "translation" from JSX into JavaScript and React is done through the use of Babel, which compiles your code into an older version of JavaScript (ES5) that all browsers support.
- **Babel JavaScript compiler**: Your browser does not understand JSX by default. Your browser only understands HTML, CSS, and JavaScript. To make your browser understand what you write using modern additions like JSX, your JSX code must be translated into the vanilla JS, the language the browser understands. For JSX, we usually use Babel. Babel ultimately converts it to vanilla JS for the browser to understand it. **Note**: Babel is already included in the boilerplate generated by create-react-app, which you most likely used to create your React app, so you don't need to configure anything to use JSX. **Example**: Let us

see how our JSX in our React component is converted to vanilla JS by Babel

- **Our functional React component is written in JSX**: Below, we have our HTML written in JavaScript or simply in JSX. See below how JSX is allowing us to write the paragraph tag <p> in JavaScript.

```
function Person( ) {
  return (
   <p>Hello React!</p>
  );
}
```

- **Babel transforms our code, which is written with JSX elements, into React.createElement calls**: Our JSX code will be transformed into the following vanilla JS code.

```
function Person() {
  return React.createElement(
        "p",
        null,
        "Hello React"
);
}
```