



TASK

React - Overview

Visit our website

Introduction

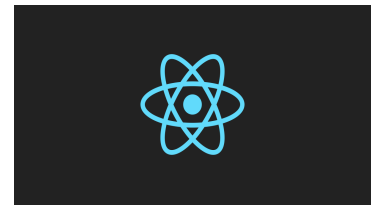
WELCOME TO THE REACT - OVERVIEW TASK!

Web developers often make use of web development frameworks and libraries to create web applications more quickly and efficiently. React.js or ReactJS (more commonly known simply as React) is a JavaScript library for building front-end web applications.

In this task, you will learn more about what React is, and how to use a React starter kit to start developing a React application with minimal configuration. You will also learn to create and render elements using React.

WHAT IS REACT?

React is used to generate user interfaces (UIs). as web browsers render HTML. All web UIs ultimately need to be converted to HTML. So why can't we just use HTML and CSS to create user interfaces for the web? We can, but HTML and CSS alone can only display static pages.



As we know, we are unable to implement logic (validating user input, looping, performing calculations etc.) with HTML. This is why JavaScript was developed in the first place. JavaScript can be used to make web pages dynamic, and this is where React comes in. With React, we *use JavaScript to write HTML!*

React is a JavaScript library for building UIs (note that React is a library, not a framework). It was originally created by [Jordan Walke](#), a software engineer at Meta (formerly known as Facebook) and community. It was first deployed on Facebook's Feed (formerly known as Facebook's News Feed) in 2011, and later on Instagram in 2012. The Initial Public Release was on 29 May 2013. React is maintained by Meta and a community of individual developers and companies.

With React, we use JavaScript to describe what we want our UI to look like and React makes it happen. In other words, React is declarative. This means we tell React **what** we want to do, **not how** to do it.

Before we get started with React, there are a few concepts that we need to understand. Each of these core concepts is discussed under the subsequent headings in this task. Some of the code examples that will be used in this task are from React's official documentation, which can be found [here](#).



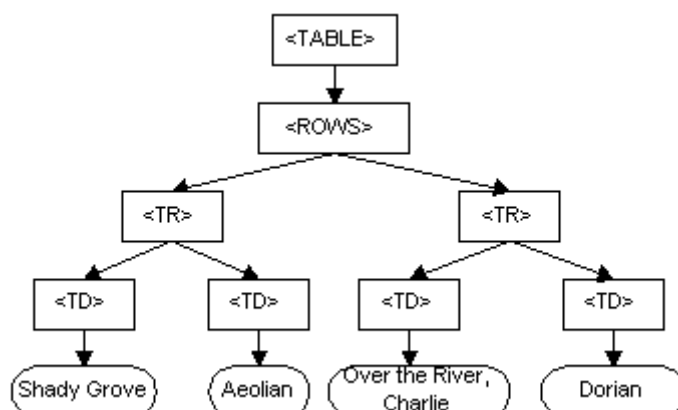
A note from the HyperionDev Team

Why use React over other libraries or frameworks?

Currently, React is one of the most popular libraries for front-end web development. Vue.js and Angular are, however, also popular. This [blog post by HyperionDev](#) compares React, Vue, and Angular.

THE VIRTUAL DOM

The image below, from the [World Wide Web Consortium](#) (W3C), visualises a Document Object Model (DOM) for an HTML table. As you already know, the DOM is a programming API for HTML and XML documents that defines the logical structure of documents and how they are accessed and manipulated.



DOM representation of the example table. Source: <https://www.w3.org/TR/WD-DOM/table.gif>

As web developers, we use the DOM to manipulate HTML documents. We can add, delete, or even change HTML elements using the DOM; in fact, you have already been doing so by making use of JavaScript! For example:

```
let div = document.getElementById("container");
let imgProfile = document.createElement("img");
imgProfile.src = "./pictures/profilePic.png";
div.appendChild(imgProfile);
let paragraph = document.createElement("p");
paragraph.innerHTML = "Dynamically adding a paragraph to HTML";
div.appendChild(paragraph);
```

However, rewriting the DOM every time the HTML document changes can significantly slow down your web application/site. To address this problem, React uses a virtual DOM.

A virtual DOM is a virtual representation of the HTML document in memory. React works by taking a *snapshot* of the DOM before changes are made. As changes are made, the virtual DOM is updated. After the changes, the virtual DOM and the snapshot of the DOM before any changes were made (which is also saved in memory) are compared to see where these changes are. Instead of rewriting the entire DOM, the real DOM is then updated with only the changes that were made.

REACT ELEMENTS

React elements are similar to DOM elements (like **div**, **body**, **head**, etc.). You should already be comfortable creating static HTML pages using various DOM elements. Like HTML, with React, you can use elements to build UIs.

The [official definition](#) of React elements is as follows:

“React elements are the building blocks of React applications. One might confuse elements with a more widely known concept of ‘components’. An element describes what you want to see on the screen. React elements are immutable¹.”

```
const element = <h1>Hello, world</h1>;
```

Typically, elements are not used directly, but get returned from components.”

We will learn more about how to create React components and the relationship between React elements and React components in the next task.

CREATING REACT ELEMENTS

There are 3 basic steps that you will use to create any React elements:

1. Create React and ReactDOM objects by importing React and React-DOM
2. Create the React Element using either JavaScript or JSX
3. Render the element that you have created to the DOM using the **render()** method

¹ Immutable: unable to be changed

1. Create React and ReactDOM objects by importing React and React-DOM

Before creating any React elements, you need to import the React library and the React-Dom package. The React library is required to create React elements (and eventually React components), whereas the React-Dom package provides DOM-specific methods including the `render()` and `createRoot()` methods. You import these as shown below.

```
import React from 'react';  
import ReactDOM from 'react-dom/client';
```



A note from the
HyperionDev Team

The [import statement](#), which may be new to you, is used to import functions, objects, or variables which are exported by another module. You will learn more about this later but for now, know that importing from React returns an object that we can use for creating React apps.

2. Create the React Element using either JavaScript or JSX

There are various ways in which you can create a React element. You could simply declare a variable that stores HTML or JSX, or a combination of HTML and JSX, or your variable could consist of more than one HTML element.

So what is JSX?

JSX is a Javascript Syntax Extension, sometimes referred to as JavaScript XML, that can make creating React applications a lot quicker and easier. JSX can be thought of as a mix of JavaScript and XML. Like XML, JSX tags have a tag name, attributes and children. With JSX, if an attribute value is enclosed in quotes, it is a string and if the value is wrapped in curly braces `{ }`, it is an enclosed JavaScript Expression.

In the example below we declared a variable called `name` with a String value of “Hyper Dave” assigned to it. JSX was then used to assign the value of the `name` variable to a variable called `element` by wrapping the `name` variable within curly braces `{ }`.

```
const name = ("Hyper Dave");
const element = (<h1>Welcome back, {name}</h1>);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    {element}
  </React.StrictMode>
);
```

When interpreted (rendered), the above example would result in the *element* variable containing the value **<h1>Welcome back, Hyper Dave</h1>**. Therefore, a JSX Expression can almost be seen as “*Embedding JavaScript within HTML elements*”. Note, if the curly braces `{ }` were removed in the second line, then the *element* variable would contain the value of **<h1>Welcome back, name</h1>**.

In the previous example, we used JSX to refer to a variable called **name**. However, we can take this even further; it is possible to execute any [JavaScript expression](#) by placing it within curly braces `{ }` in JSX.

In the following example, we will be creating a JavaScript object named **user** which contains some generic user data, and a JavaScript function named **formatName()**. The **formatName()** function receives one parameter named **person** and returns a String containing the person's full name (the result of calling said function). With JSX we can embed the result of calling the **formatName()** function directly into an **<h1>** element.

```
import profilePicUser from "./profiles/HD.png"

function formatName(person) {
  return person.firstName + " " + person.lastName;
}

const user = {
  firstName: "Hyper",
  lastName: "Dave",
  profilePic: {profilePicUser}
};
```

```
const element = (
  <h1>
    Welcome back, {formatName(user)}!
  </h1>
);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    {element}
  </React.StrictMode>
);
```

When interpreted (rendered), the above-mentioned example would result in the *element* variable containing a value of **<h1>Welcome back, Hyper Dave!</h1>**. The **formatName()** function is passed an argument named **user**, which is then used within this function to produce a result. Note, if the curly braces **{ }** were removed within the **<h1>** (in the **element** variable), then the **element** variable would contain a value of **<h1>Welcome back, formatName(user)!</h1>**.

Both previous examples have a very similar result, therefore you might wonder why anyone would take the longer route of the second example? Well, the answer lies in the fact that the JavaScript Expression - in this case, a function - can perform a lot of logic and calculations before returning a result.

There are some rules that you should be aware of when using JSX, some of which have been implemented in the examples above.

- As with HTML, with JSX you can specify various elements that can have different attributes.
 - With JSX you can use any HTML elements like **div**, **img**, or **h1**. You can also create user-specified elements.
- With JSX, if an attribute's value is enclosed within quotes, it is a String, also known as a String literal (follow this [link](#) if you require a recap on how Strings are used in JavaScript). If the value assigned to the attribute is wrapped in curly braces, it is an enclosed JavaScript Expression. An attribute's value can be assigned using *either*:
 - String literals can be used for online images, e.g. ``
String literals should be enclosed in quotation marks.
OR
 - A JavaScript Expression, e.g. ``
JavaScript Expressions should always be enclosed in curly braces instead of quotation marks, e.g. `{user.profilePic}`.

- When using images from your current local computer it must always be imported eg. `import profilePicUser from './profiles/HD.png'`
- In both previous examples, our React element (the `element` JavaScript variable) only contained one HTML element, namely an `h1`. It is however possible to assign multiple HTML elements to a single React element. To accomplish this, we have to take note of the following:
 - Each React element can only contain one parent element, for example, the `div` HTML element in the next example.
 - The parent element, however, can contain many child elements. You'll notice that the `h1` and `img` HTML elements are nested within the `div` HTML.
- The entire value has to be contained within braces `()`.

```
const element = (
  <div>
    <h1>I'm learning React with HyperionDev</h1>
    
  </div>
);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    {element}
  </React.StrictMode>
);
```

Note: The image in the code sample above is a PNG file. As PNG (Portable Network Graphics) supports transparency, it might look like the image is not loaded (as the background of the image is transparent). Adding styling (CSS) to React Elements will be discussed later in this task.

With the previous examples, we took the approach of simply assigning HTML and JSX to a variable that would ultimately be rendered. This approach is just **syntactic sugar** to help us speed up development. The React library in itself also contains a **createElement()** method, which can be used to create React elements. This method is *implicitly* executed in the previous examples. We can however, create React elements by *explicitly* executing this method, for example:

```
const element = React.createElement("h1", {}, "Hello World!");

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    {element}
  </React.StrictMode>
);
```

The **createElement()** method expects three arguments, namely:

1. **Type:** this could be an HTML tag name (such as **div** or **h1**) or a React Component (which you will learn more about later). It describes what type of element you are creating.
2. **Props:** props are properties that can be passed from parent to child elements. Props that can be passed include descriptions of HTML attributes (such as **src** or **class**). Props will be discussed in more detail in a later task.
3. **Children:** used to pass the child elements that this element should have. This could be a string, another React Element, or even an array of elements.

For example;

```
const reactElement = React.createElement("h1", {className: "header"}, "welcome back");
```

↑ ↑ ↑
Type **Props** **Children**



A note from the HyperionDev Team

What is the purpose of React Strict Mode?

Consider the example above. The `render()` method contains a `<React.StrictMode>` component. In the development environment, this component is merely used to indicate any potential errors/problems within your application. You can read more about React Strict Mode [here](#).

Babel

Learning any new concept, like JSX, can be a daunting experience at first, however, Babel is a tool that can be used to convert JSX to JavaScript. Go to the [Babel REPL](#) website and copy and paste the code examples above into the 'Write code here' space provided. Babel will then show you the differences between the examples of React code (using JSX) and code written using just JavaScript. Note that Babel conversion is automatically incorporated into the React compilation process, so you never need to worry about it directly.

3. Render the element that you have created to the DOM

Once you have created a React element, you still have to render it so that it is visible to the user. You will notice that each of the code examples above contains a `render()`, and `ReactDOM.createRoot()` method, for example;

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    {element}
  </React.StrictMode>
);
```

The `ReactDOM.createRoot()` method is used to obtain a container node to which you will render the React element (or React component) that you have created. The container node in all previous examples was `document.getElementById('root')`, a reference to, in this case, an *HTML* element which contains an `id` attribute with a value of `root` (follow this [link](#) if you

require a recap on how the `document.getElementById()` method works). Later in this task, you will be shown where this HTML element is.

Once we have created a container node, we still need to call the `render()` method on this node. The argument you pass to the `render()` method is the content that you want to display to the user, in the examples, it was the React element named `element`.



A note from the HyperionDev Team

As a developer, you will never stop learning new techniques

As technology improves, the approaches and/or programming languages used to create applications and/or Web sites/apps are also updated. At the point that this task was last updated, the current version of React was React 18, therefore, all examples are based on this version. When doing your own research you might come across some previous approaches, which might still be widely used. One of these approaches was how we rendered React elements (and components) as per the example below;

```
ReactDOM.render(element, document.getElementById('root'));
```

In previous versions of React, the first argument passed to the `render()` method was the React element/component which should be rendered. The second argument passed was the DOM element to which you want to append the React element/component.

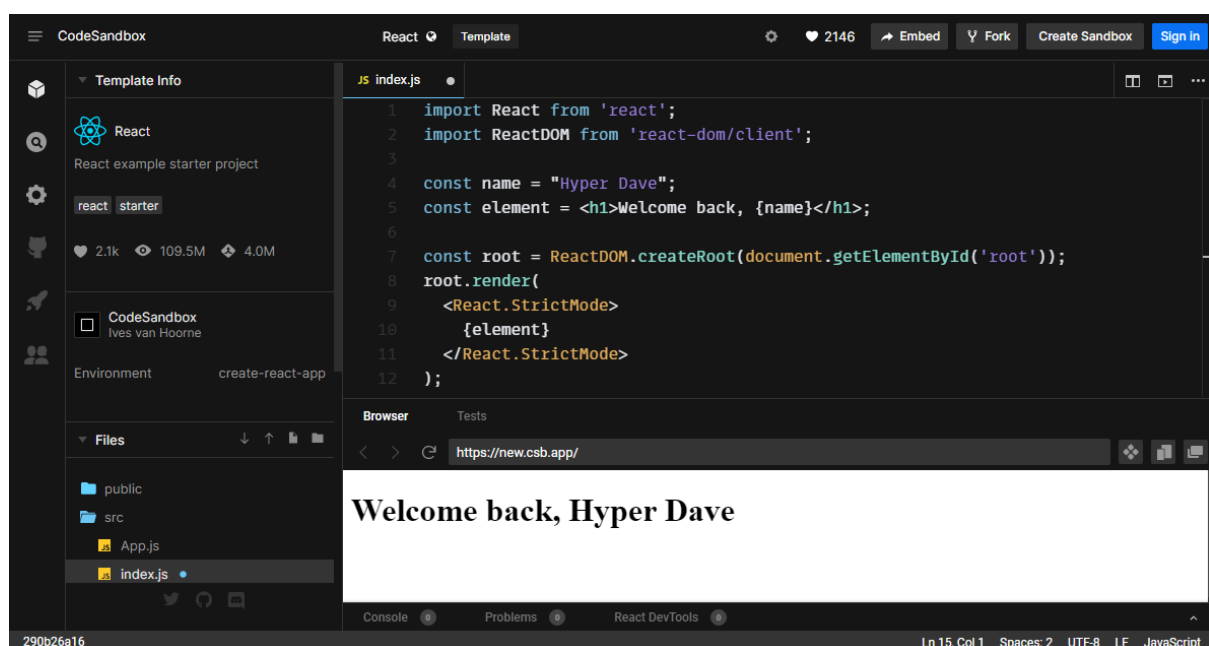
CODESANDBOX

Now that we have some introduction theory out of the way, let's implement the examples used so far. A bit later in this task, we will make use of Create React App (CRA), which is used to create a React app, for development, on your local machine. However, a lot of online code editors exist, one of which is [CodeSandbox](#). CodeSandbox is an online code editor that allows us to easily create, test, and share basic React apps.

Let's execute some of our previous examples:

1. Follow the following [link](#) to open CodeSandbox's online editor within your browser.

2. Select the **index.js** file (it should be located under Files, within the src folder).
3. Delete everything in **index.js** except for the first two import statements. Remember, you'll need these import statements every time you create React elements.
4. Copy and paste each of the code examples that we have considered above (code examples in the JSX section of this document) into CodeSandBox one at a time. For each element, study the code and the output in CodeSandbox. Be sure you understand the code in each code example.
5. Amend some of the React elements (code examples) and not the changes in output, to familiarise yourself with how they are rendered.



The first example is rendered in CodeSandbox

STYLING REACT ELEMENTS

In previous tasks you learned how to style HTML elements. Using CSS effectively to create attractive UIs is an indispensable skill. Styling React elements is very similar to styling HTML elements.

When styling HTML elements/pages, you can create your own external custom stylesheets. To apply the style rules in the external CSS to the HTML elements, you would link the external CSS file with the **link** HTML element to your HTML page, for example:

```
<link href="./index.css" rel="stylesheet" type="text/css" />
```

You have most likely linked external CSS files to HTML pages before, however, with React we ultimately use JavaScript to produce/render our HTML pages. Therefore the approach of linking external CSS files changes slightly, as we need to *link* the external CSS file to our JavaScript file *instead* of our HTML file. This can be accomplished by making use of the **import** JavaScript statement. Simply import the external CSS file to the JavaScript file where you are creating the element(s), for example:

```
import './index.css';
```

Follow this [link](#) for more information about linking external stylesheets with React.

We don't have to start styling from scratch when creating React components. There are several CSS libraries that can make creating components quicker and easier by reusing and modifying existing code. Let's consider just two of these:

- **[React-Bootstrap](#):** React-Bootstrap is a complete re-implementation of the Bootstrap components using React. Using React-Bootstrap is similar to using normal Bootstrap. To use React-Bootstrap:
 1. Install React-Bootstrap using NPM (Node Package Manager). Using your terminal/command line, navigate to the project directory for your React app and type: **npm install --save react-bootstrap bootstrap**
 - NPM will be explained a bit later in this task
 2. Import the React-Bootstrap component you want to use. For example, if you want to use the React-Bootstrap Button component you need to add the following line of code to the JavaScript file where you want to use this component: **import Button from 'react-bootstrap/Button';**
 3. Include the latest styles from the React-Bootstrap CDN. You can obtain the link from the official [React-Bootstrap website](#). It would look similar to the following example:

```
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
integrity="sha384-1BmE4kWBq78iYhF1dvKuhfTAU6auU8tT94WrHftjDbrCEX
SU1oBoqyl2QvZ6jIW3" crossorigin="anonymous" />
```

- **Reactstrap:** Reactstrap has fewer features and is a good choice for projects that need smaller builds.

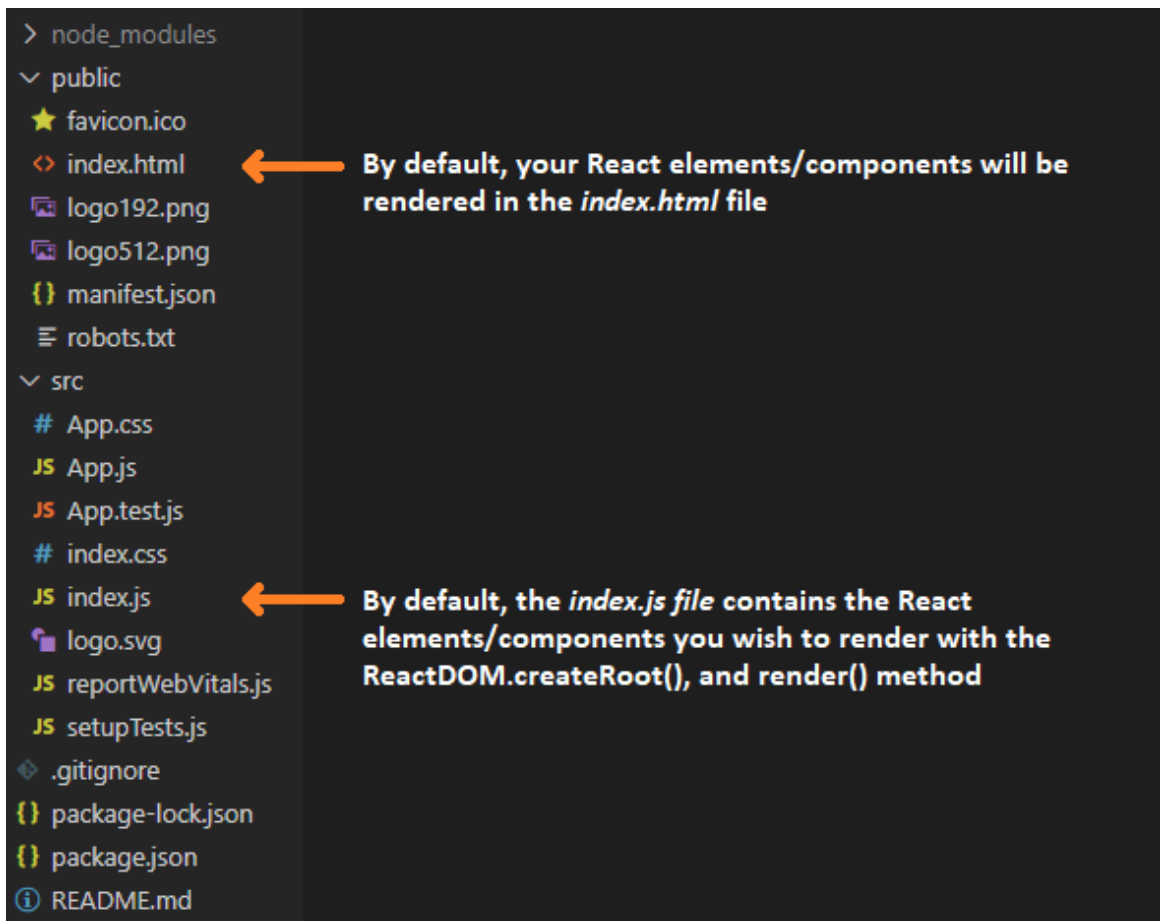
REACT STARTER KITS

React relies heavily on other libraries to function properly. As we have already seen, **Babel** is a JavaScript compiler used for writing next-generation JavaScript. It is particularly relevant to developers who work with React because it can convert JSX syntax.

React works best when used with **Webpack**. When you create apps using React, many modules and other resources, like images and CSS files, are created. Webpack is a module bundler for modern JavaScript applications. It creates a dependency graph that is used to handle all the resources you need for your app. The intricate details and inner workings surrounding Webpack are a bit beyond the scope of this Bootcamp, however, should you wish to learn more about Webpack, follow this [link](#).

Although you could manually install and configure React and all its dependent libraries separately, Meta (the creators of React), recommend that you make use of a starter kit instead. A starter kit automatically installs all the libraries and modules needed to create a basic React app. It also does some basic configuration so that you are able to start creating a React app immediately. Although there are several starter kits available for you to use, in this Bootcamp we will be using one of the official starter kits recommended by Meta, namely **Create React App** (CRA) which is used to create single-page applications.

When you create a React app using CRA, a project folder with a specific directory structure will be created, for example:



By default, the React elements you create will be rendered in the **index.js** file that has been created for you by CRA. To find this file, open the project directory that was created with CRA. In this directory, you will see a directory called **src** which contains the file **index.js** file. In this file, you will see an instruction to render any React elements or components.

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
```

```
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

To understand where the React element/component you created will be rendered, open the **public** directory. You should find a file called **index.html**. This HTML file contains a **div** HTML element which has an **id** attribute with a value of **'root'** assigned to it. When the `render()` method is executed within the **index.js** file, we are specifying that the React element/component you create will become a child of this **div** (which is within the **index.html** file).



A note from the HyperionDev Team

Are you unable to create a React app with CRA?

As you are aware, technology constantly improves, therefore existing software also needs to improve. The instructions as per [CRA's official documentation](#) are valid and correct, however, if you experience errors while trying to use CRA, one of the main causes could be that the version of NPM on your local machine is trying to install an earlier (older) version of CRA. You can force NPM to install the latest version of CRA, by making use of the **@latest** flag, for example; **npm create-react-app@latest my-app**.

DEBUGGING A REACT APP

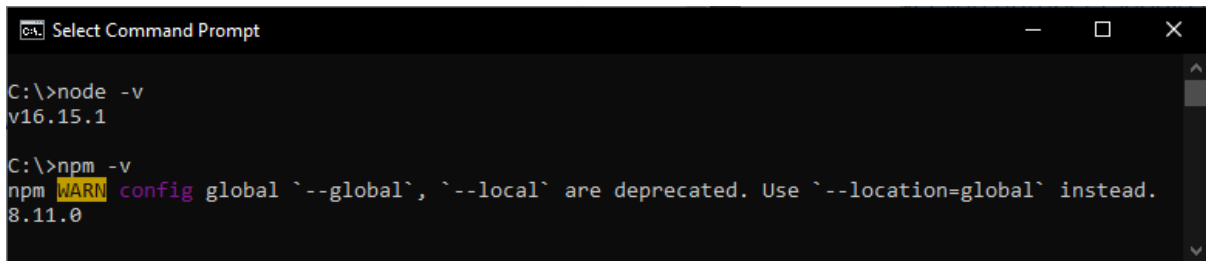
The developers of React have created various tools to assist with debugging React apps. To access these debugging tools, follow this [link](#). The official React Developer Tools can be found at this [link](#).

Alternatively, you can install additional browser extensions to help with debugging a React App. Click on the appropriate [Firefox](#) or [Chrome](#) links to access browser extensions to help with debugging. As you have already seen, you can also use [CodeSandbox](#) for simple debugging.

RUN REACT ON YOUR LOCAL MACHINE

To use React on your local machine, **Node** and **NPM** need to be installed. To see if you already have Node and NPM installed, open the command line

interface/terminal and try to see which versions of Node and NPM are installed. If correctly installed, the version numbers should be displayed, for example:



```
C:\>node -v
v16.15.1

C:\>npm -v
npm WARN config global '--global', '--local' are deprecated. Use '--location=global' instead.
8.11.0
```

If Node is not yet installed on your local machine, follow the instructions [here](#) to install Node. It is recommended to download and install the LTS (Long Term Support) version of Node. NPM is distributed with Node which means that when you download Node, NPM will be automatically installed on your local computer.



A note from the HyperionDev Team

How to set up the example that accompanies this task

As mentioned before; React relies heavily on other libraries to function properly, most of which are contained within the **node_modules** folder (which contains *a lot* of files). To run the React example on your local machine we need the **node_modules** folder. Please read through, and execute, the instructions contained within the README.md file which can be found in the root folder of the example.

Compulsory Task 1

Follow these steps:

- Use [CodeSandbox](#) to create the following elements. Copy and paste the code for each element you create from CodeSandbox into a file called **my-React-elements.txt**.
 - A button that links to your GitHub profile when clicked.
 - A heading that contains the current date and time, formatted for easy readability. Follow this [link](#) if you require help.
 - A bullet-point summary of the advantages of JavaScript frameworks/libraries (like React) above vanilla JavaScript.
- Make sure that your **my-React-elements.txt** file is saved to your Dropbox folder for this task.

Compulsory Task 2

Create a webpage with React and JSX with the following content:

- Create a folder called **ReactElements** on your local machine.
- Open the command line interface/terminal and **cd** to the folder you have created above.
- Follow the instructions found [here](#) to install React using the Create React App Starter Kit. Name your React App **react-hello**.
- Once you have started your front-end server (with **npm start**), test the default React App that was created by CRA by navigating to <http://localhost:3000/> in your browser.
- Modify **index.js** file (within the **src** directory):
 - Create a JavaScript object called **user** that stores all the details for a particular user of your app. This object should have at least the following properties: **name**, **surname**, **date_of_birth**, **address**, **country**, **email**, **telephone**, **company**, **profile_picture** (source of where the image can be found), and **shopping_cart**.

The **shopping cart** property should be used to store an array of items in the user's shopping cart.

- Create and render a React element that displays all the information about the user in an attractive way. This element should:
 - Include at least 1 [React-Bootstrap](#) component.
 - Apply a custom stylesheet that you have created.
- Once you are ready to have your code reviewed, **delete** the **node_modules** folder (please note that this folder typically contains hundreds of files which, if you're working directly from Dropbox, has the potential to **slow down Dropbox sync and possibly your computer**), compress your project folder and add it to the relevant task folder in Dropbox.



Rate us
Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

[Click here](#) to share your thoughts anonymously.



REFERENCES

React.js. (2022). Getting Started – React. Retrieved on 26 June 2022, from <https://reactjs.org/docs/getting-started.html>

React.js (2022). Hello World - React. Retrieved on 26 June 2022, from <https://reactjs.org/docs/hello-world.html>

React.js (2022). Team - React. Retrieved on 26 June 2022, from <https://reactjs.org/community/team.html>

React.js (2022). Glossary of React Terms - React. Retrieved on 26 June 2022, from <https://reactjs.org/docs/glossary.html>

React.js (2022). Strict Mode - React. Retrieved on 26 June 2022, from <https://reactjs.org/docs/strict-mode.html>

React.js (2022). ReactDOMClient - React. Retrieved on 26 June 2022, from <https://reactjs.org/docs/react-dom-client.html>

React.js (2022). Create a New React App – React. Retrieved on 26 June 2022, from <https://reactjs.org/docs/create-a-new-react-app.html#create-react-app>

React.js (2022). Higher-Order Components - React. Retrieved on 26 June 2022, from <https://reactjs.org/docs/higher-order-components.html>

React.js (2022). Higher-Order Components - React. Retrieved on 26 June 2022, from <https://reactjs.org/docs/higher-order-components.html>

React.js (2022). react/packages/react-devtools at main · facebook/react · GitHub. Retrieved on 26 June 2022, from <https://github.com/facebook/react/tree/main/packages/react-devtools>

React.js logo (2022). Retrieved on 26 June 2022, from <https://reactjs.org>

CodeSandbox (2022). Documentation - CodeSandbox Documentation. Retrieved on 26 June 2022, from <https://codesandbox.io/docs>

CodeSandbox (2022). React - CodeSandbox. Retrieved on 26 June 2022, from <https://codesandbox.io/s/new>

Create React App (2022). Adding a Stylesheet | Create React App. Retrieved on 26 June 2022 from <https://create-react-app.dev/docs/adding-a-stylesheet/>

React-Bootstrap (2022). React-Bootstrap · React-Bootstrap Documentation. Retrieved on 26 June 2022 from <https://react-bootstrap.netlify.app/>

Reactstrap (2022). Home/Installation - Page · Reactstrap. Retrieved on 26 June 2022 from <https://reactstrap.github.io/>

Webpack (2022). Concepts | Webpack. Retrieved on 26 June 2022 from <https://webpack.js.org/concepts/>

MDN (2022). Expressions and operators - JavaScript | MDN. Retrieved on 26 June 2022, from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_Operators#expressions

MDN (2022). Strings - JavaScript | MDN. Retrieved on 26 June 2022, from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

MDN (2022). Document.getElementById() - Web APIs | MDN. Retrieved on 26 June 2022, from <https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementById>