MERN and REACT.js

Introduction to Web Development

Web development stack is nothing but a set of tools typically used in tandem to develop web apps. It refers to the technologies that individual developer specializes in and use together to develop new pieces of software.

Web technology sets that include all the essential parts of a modern app are as follows: the **frontend framework**, the **backend solution** and the **database (relational or document-orinted)**

Introducton to stack:

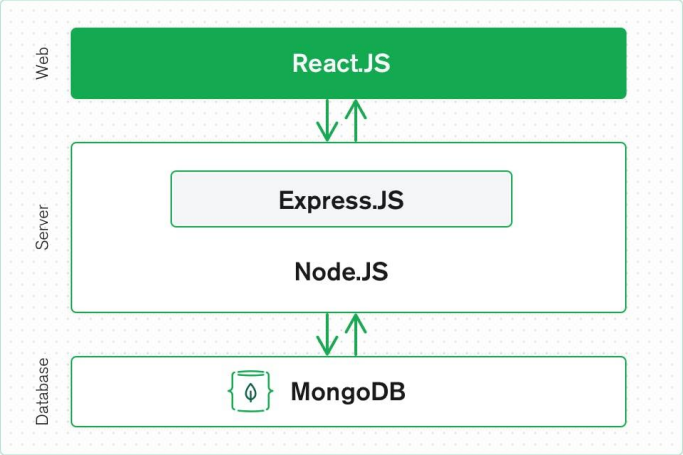
Any web application made by using multiple technologies. The combination of these technologies is called a **"stack,"** popularized by the LAMP stack, which is an acronym for Linux, Apache, MySQL, and PHP, which are all open-source components. As the web development world is continually changing, its technology stacks too changing. Top web development stacks are

- MEAN
- MERN
- Meteor.js
- Flutter
- The serverless Technology stack
- The LAMP technology stack
- Ruby on Rails Tech Stack

MERN stands for **MongoDB, Express, React, Node**

- MongoDB - Document database
- Express(.js) - Node.js web framework
- React(.js) - A client-side JavaScript library
- Node(.js) - The premier JavaScript web server

Allows you to easily construct a 3-tier architecture (frontend, backend, database) entirely using JavaScript and JSON.



Why MERN?

Ideally suited for web applications that have a large amount of interactivity built into the front-end.

- JavaScript Everywhere
- JSON Everywhere
- Isomorphic

REACT.JS

Free and open source front end JS Library for building UI and UI Components. Maintained by Facebook, a community of individual developers and companies. This can be used as base in the development of Single page applications and mobile applications and

is concerned with state management and rendering the state to DOM.

It is the declarative JavaScript Library for creating dynamic client-side applications

Builds up complex interfaces through simple Components, connect them to data on your backend server, and render them as HTML. Provides support for forms, error handling, events and render them as HTML.

Key points about React.js

- **Properties of React:**

  Declarative, Simple, Component based, Supports server side, Mobile support, Extensive, Fast , Easy to learn

- **Single way data flow**
  - A set of immutable values are passed to the components renderer as properties in its HTML tags. The component cannot directly modify any properties but can pass a call back function with the help of which we can do modifications.
  - This complete process is known as "**properties flow down; actions flow up**".

- **Virtual DOM**
  - Creates an in-memory data structure cache which computes the changes made and then updates the browser.
  - Allows a special feature that enables the programmer to code as if the whole page is rendered on each change whereas react library only renders components that actually change

Creation of React App

Can be done in two ways

- **Using node package manager(npm):** To setup a build environment for React that typically involved use of npm (node package manager), webpack, and Babel

- **Directly importing Reactjs library in HTML Code.** Defined in two .js files (**React** and **ReactDOM**)
  - <script src="https://unpkg.com/react@17/umd/react.development.js" crossorigin ></script>
  - <script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js" crossorigin> </script>
  - The files differ for development and production.
    When deploying, replace "development.js" with "production.min.js"

React is used for handling the **view layer for web and mobile apps.** Allows developers to create **large web applications that can change data**, without reloading the page. Also allows to **create reusable UI components.** The main purpose of React is to **be fast, scalable, and simple. Works only on user interfaces** in the application

React Elements:

The browser DOM is made up of DOM elements. Similarly, the React DOM is made up of React elements. DOM elements and React elements may look the same, but they are actually quite different. A React element is a description of what the actual DOM element should look like. In other words, React elements are the instructions for how the browser DOM should be created. Syntax: React.createElement(type,{props},children);

The first one is the type of element we're creating, in this case an <h1> tag. This could also be another React component. Second is the properties list in the form of objects. Third argument is the content of the element used in the first argument.

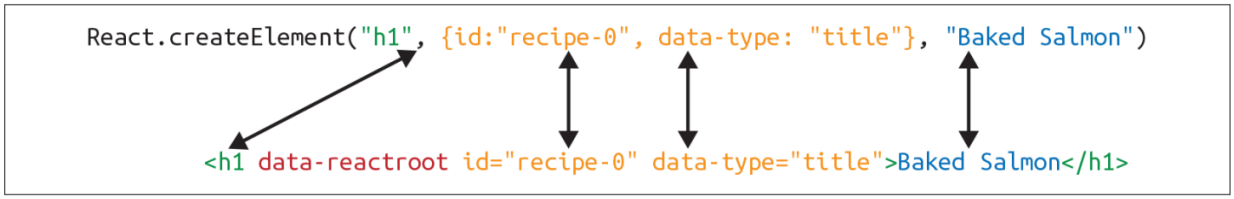
We can create a React element to represent an h1 using React.createElement

```
React.createElement("h1", null, "Baked Salmon")
```

When an element has attributes, they can be described with properties. Here is a sample of an HTML h1 tag that has id and data-type attributes.

```
React.createElement("h1", {id: "recipe-0", 'data-type': "title"}, "Baked Salmon")
```

Relationship between createElement and the DOM element



Note: data-reactroot will always appear as an attribute of the root ele- ment of your React component

React Component in brief:[In detail in L2 and L3]

A user control that has **code to represent visual interfaces and data.** An isolated piece of code which can be reused in one or the other module **.**Contains a root component in which other subcomponents are included. 2 types of components in React.js can be created.

- **Stateless Functional Component**
  - Includes simple JavaScript functions and immutable properties, i.e., the value for properties cannot be changed.

```
function Demo(props) {
  return <h1> Welcome to REACT JS, {props.Name} </h1>;
}
```

- **Stateful Class Component**
  - Classes which extend the Component class from React library. The class component must include the render method which returns HTML.

```
class Demo extends React.Component{
        render(){    return <h1>
                    Welcome to REACT JS, {props.Name} </h1>;    }
}
```

Calling/rendering the components:

ReactDOM: Contains the tools necessary to render React elements in the browser. All the tools necessary to generate HTML from the virtual DOM are found in this library.

ReactDOM.render() is responsible for rendering a React component. The first

parameter is the component class name. Second parameter is the destination where the component is to be rendered.

```
ReactDOM.render(
    React.createElement(Demo, null, null),
    document.getElementById('root')
);
```

Coding Example 1: Serverless Hello world: Using react, displaying a simple page on the browser

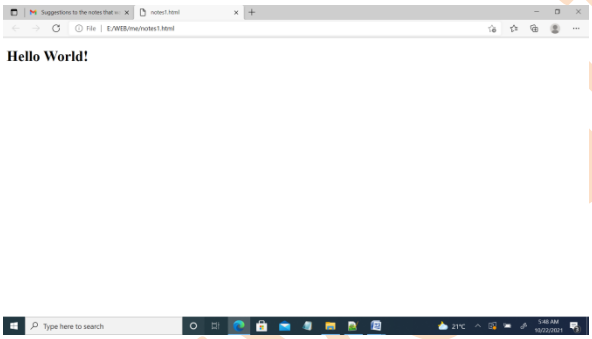
```
<!DOCTYPE HTML>
<html>
<head>
    <script crossdomain
src="https://unpkg.com/react@16/umd/react.development.js"> </script>
    <script crossdomain src="https://unpkg.com/react-dom@16/umd/react-
dom.development.js"> </script>
    <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js">
    </script>
</head>
<body>
  <div id="contents"></div><!-- this is where our component will appear -->
  <script type="text/babel">    // Important
      var contentNode = document.getElementById('contents');
      var component = <h1>Hello World!</h1>; // A simple JSX component
      ReactDOM.render(component, contentNode);
                              // Render the component inside the content Node
  </script>
</body>
</html>
```

In the above code, <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js">

</script> , this is the **babel library which is a JSX transformer**. Then, there is the type of the script that you specify in the wrapping <script> tags around the JSX ode. The browser-based JSX compiler looks for all inline scripts of type “text/babel” and compiles the contents into the corresponding JavaScript. The other two scripts, react.js and react-dom.js, are the core React libraries that handle react component creation and rendering

```
<script type="text/babel">
```

Screenshot of example code 1 output:

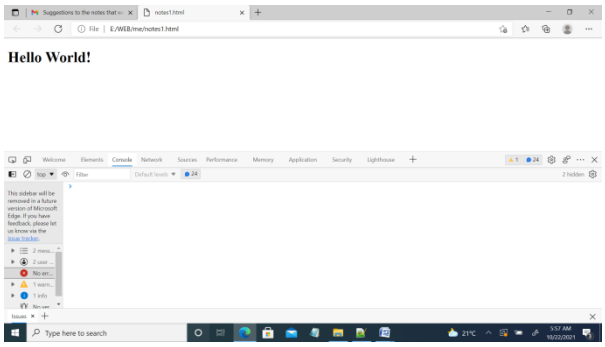


Coding Example 2: Multiple component calls/renders will render only the last one

In the above code, add these below lines and oberve the output

```
ReactDOM.render(component, contentNode);
ReactDOM.render(component, contentNode);
ReactDOM.render(component, contentNode);
ReactDOM.render(component, contentNode);
```

Screenshot of example code 1 output:



Including the JSX Code:

JSX uses a special syntax which allows you to mix HTML with JavaScript. JSX is a JavaScript XML used in React applications. An extension to JavaScript. Uses HTML syntax to create elements and components. Has tag name, attributes, and children. JSX compiles the code into pure JavaScript which can be understood by the browser.

Include the library:

```
<script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"> </script>
<script type="text/babel">
                ReactDOM.render(<h1>Welcome     to REACTJS</h1>,
                document.getElementById('root')
  );
```

Babel:

A JavaScript compiler that can translate markup or programming languages into JavaScript. Available for different conversions. React uses Babel to convert JSX into JavaScript.

Using React, print welcome to REACTJS on the web page. Use JSX and NonJSX both

Case 1: Using JSX Syntax

```
<html>
<head>
<script crossdomain    src="https://unpkg.com/react@16/umd/react.development.js">
</script>
<script crossdomain src="https://unpkg.com/react-dom@16/umd/react-
dom.development.js"> </script>
<script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"> </script>
</head>
<body>
        <div id = "root"></div>
        <script type = "text/babel">
```

```
        ReactDOM.render(<h1>welcome</h1>,document.getElementById("root"))
        </script>
</body>
</html>
```

Case 2: Using Non-JSX Syntax: small change in the above code

```
<html>
<head>
<script crossdomain     src="https://unpkg.com/react@16/umd/react.development.js">
</script>
<script crossdomain src="https://unpkg.com/react-dom@16/umd/react-
dom.development.js"> </script>
<script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"> </script>
</head>
<body>
        <div id = "root"></div>
        <script type = "text/babel">
        ReactDOM.render(React.createElement("h1","null","welcome to
reactJS"),document.getElementById("root"))
        </script>
</body>
</html>
```

References:

i) What is the use of React.createElement ? - GeeksforGeeks

ii) Textbooks from the syllabus copy