UNIT 2 : FUNDAMENTALS OF REACT.JS

2.1 Overview of React

2.1.1 Concept of React

2.1.2 Using React with HTML

2.1.3 React interactive components : Components within Components and files

2.1.4 Passing Data through props

2.2 Class Components

2.2.1 React class and Class Components

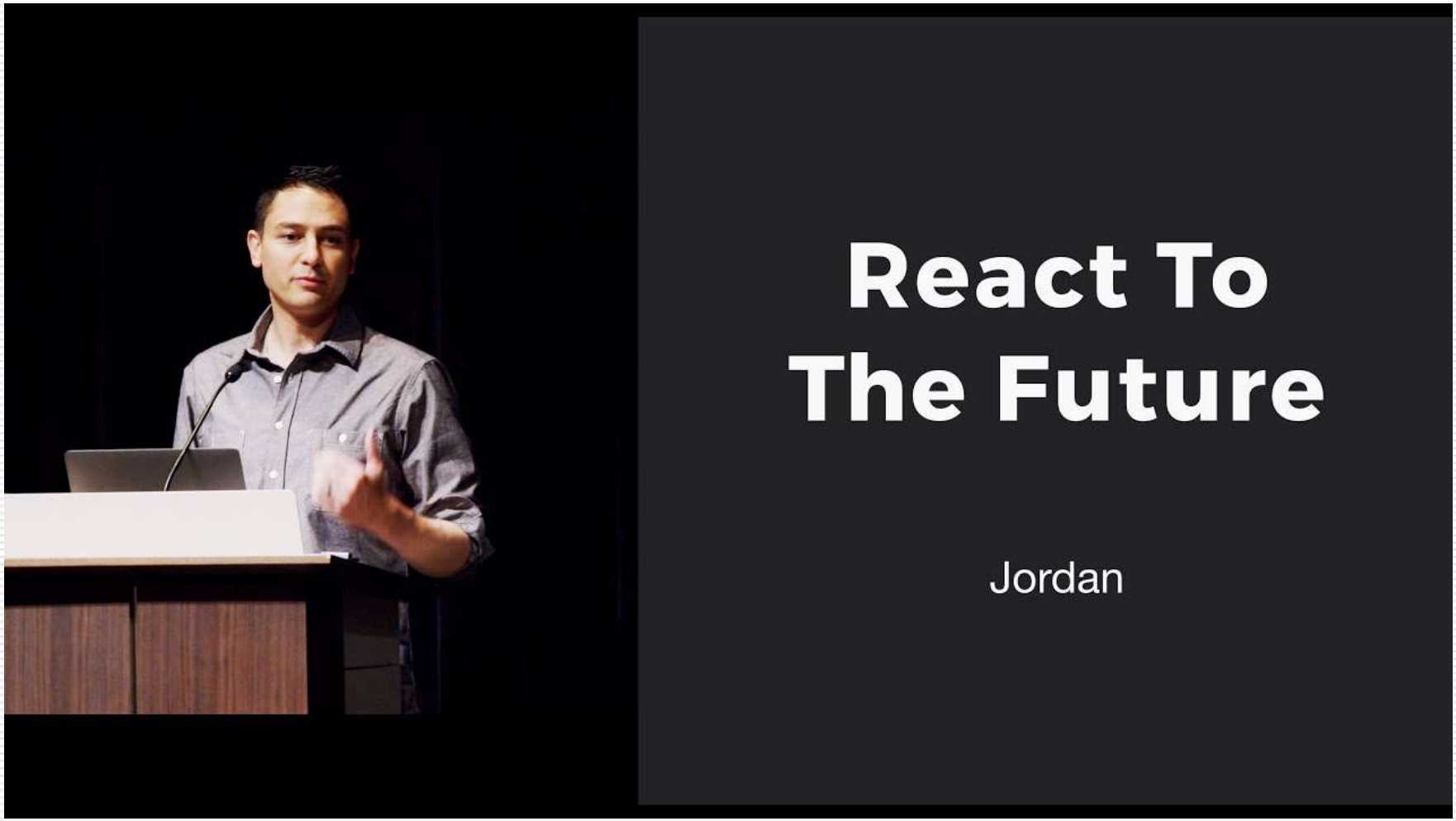
2.2.2 Conditional Statements, Operators, Lists

2.2.3 React Events: Adding Events, Passing arguments, Event Objects

Introduction

- ReactJS is a **declarative**, **efficient**, and **flexible** JavaScript library for building reusable UI components.
- It is an **open-source**, **component-based front end library** responsible only for the **view layer** of the application.
- It was created by **Jordan Walke**, who was a software engineer at Facebook.
- It was initially **developed and maintained** by **Facebook** and was later used in its products like **WhatsApp & Instagram**.
- Facebook developed **ReactJS** in **2011** in its **newsfeed** section, but it was released to the public in the month of **May 2013**.

- Today, most of the websites are built using **MVC (Model View Controller)** architecture. In **MVC** architecture, React is the '**V**' which stands for **View**, whereas the architecture is provided by the **Redux or Flux**.
- Latest React Version is **18.0** which was introduced on **29th March, 2022**


React To
The Future
Jordan

- To learn REACTJS, basic requirements are HTML, CSS, JS and ECMAScript (**European Computer Manufacturers Association Script**)

2.1 Overview of React

2.1.1 Concept of React

2.1.2 Using React with HTML

2.1.3 React interactive components : Components withing Components and files

2.1.4 Passing Data through props

1. Install node.js
   - nodejs.org
2. Check installed node.js version
   - Node -v
3. To install create-react-app
   - Npx create-react-app app_name
4. To check installed create-react-app version
   - create-react-app --version
5. Create new folder where you want to create your project
   - Mkdir folder_name
6. Change directory to new folder
   - Cd folder_name
7. Inside folder run this command to create react app
   - Npx create-react-app app_name
8. To run react app
   - Npm start

Installation of REACT

1. Install NODEJS and NPM
2. Install Visual Code / Sublime / Atom / Brackets
3. Install React from terminal
   - Npm install –g create-react-app
   - Create-react-app --version
   - Create-react-app <projectname>

Installation of REACT in OFFLINE MODE

1. Install node.js
   - nodejs.org
2. Check installed node.js version
   - Node -v
3. Change directory from current to project folder.
4. Run the following command.
   - npm install -g create-react-app-offline
   - Crao –n myapp

2.1.1 Concept of React

- React is a **JavaScript library** for building **user interfaces**.
- React is used to build **Single-Page Applications**.
- React allows us to create **reusable UI components**.

2.1.2 Using React with HTML

ReactDOM.render()

- render has been replaced with createRoot in React 18.
- React's goal is in many ways to render HTML in a web page.
- React renders HTML to the web page by using a function called ReactDOM.render().
- The ReactDOM.render() function takes two arguments, HTML code and an HTML element.
- The purpose of the function is to display the specified HTML code inside the specified HTML element.
- But render where?
- There is another folder in the root directory of your React project, named "public". In this folder, there is an index.html file.

Example:

- index.js

```
var React=require('react');
var ReactDOM=require('react-dom');

ReactDOM.render(<h1>Hello
World</h1>,
document.getElementById('root')
```

- index.html

```
<body>
  <div id="root"></div>
</body>
```

Exercise

- Make a website header and display on webpage.
- Make a table with 5 students detail and print it on browser.
- Use 2-3 images and display on browser.
- Make one webpage that contains h1, p and 5 ordered list elements (Netflix, Web Series, name of 5 web series)

React JSX

- JSX stands for **JavaScript XML**.
- JSX allows us to write **HTML** in **React**.
- JSX makes it easier to **write and add HTML in React**.
- JSX converts **HTML tags into react elements**.

Example:

```
const myElement = <h1>TYBCA The Great Class</h1>;

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

2.1.3 React interactive components : Components within Components and files

- Components are **independent and reusable bits of code**. They serve the same **purpose as JavaScript functions**, but work in **isolation and return HTML**.
- Components come in two types, **Class components and Function components.**

Create Your First Component

- When creating a React component, the component's name *MUST* start with an upper case letter.

Class Component

- A class component must include the extends **React**.
- Component statement: This statement creates an inheritance to React. Component, and gives your component access to React.
- Component's functions: The component also requires a render() method, this method returns HTML.

Example:

```
class Car extends React.Component {
  render() {
    return <h2>Hi, I am a Car!</h2>;
  }
}
```

Function Component

- A Function component **also returns HTML**, and behaves much the same way as a Class component, but **Function components can be written using much less code**, are easier to understand.

Example:

```
function Car() {
  return <h2>Hi, I am a Car!</h2>;
}
```

Rendering a Component

- Now your React application has a component called Car, which returns an <h2> element.

- To use this component in your application, use similar syntax as normal HTML: <Car />

Example:

```
const root =
ReactDOM.createRoot(document.getElemen
tById('root'));
root.render(<Car />);
```

2.1.4 Passing Data through props

- Components can be passed as props, which stands for properties.

- Props are like function arguments, and you send them into the component as attributes.

Example:

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

function Car(props) {
  return <h2>I am a {props.color} Car!</h2>;
}

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<Car color="red"/>);
```

2.2 Class Components

2.2.1 React class and Class Components

2.2.2 Conditional Statements, Operators, Lists

2.2.3 React Events: Adding Events, Passing arguments, Event Objects

2.2.1 React class and Class Components

2.2.2 Conditional Statements, Operators, Lists

- In React, you can conditionally render components.
- There are several ways to do this.

Without Conditional

```
function MissedGoal() {
  return <h1>MISSED!</h1>;
}

function MadeGoal() {
  return <h1>Goal!</h1>;
}
```

With Component

```
function Goal(props) {
  const isGoal = props.isGoal;
  if (isGoal) {
    return <MadeGoal/>;
  }
  return <MissedGoal/>;
}

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(<Goal isGoal={false} />);
```

Operators

- Logical && Operator
- Ternary Operator

Logical && Operator

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

function Logicaland()
{
    const a=10,b=20;
    return (
        <>
        {
            a<b &&
            <h2>VTCBCSR</h2>
        }
        </>
    );
}

export default Logicaland;

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Logicaland />);
```

Ternary Operator

- condition ? true : false

Example:

```
export default function Ternary(props)
{
    const marks=props.mark;
    return(
        <>
        { marks>36 ? <Pass/>:<Fail /> }
        </>
    );
}
function Pass()
{
    return <h1>Congrates You did it</h1>
}
function Fail()
{
    return <h1>Oops !!! Better Luck Next time</h1>
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Ternary mark=80 />);
```

Lists

- In React, you will render lists with some type of loop.

- The JavaScript map() array method is generally the preferred method.
- creates a new array from calling a function for every array element.

Example:

```
export function Fun(props)
{
    const chocolate=['Oreo Silk','Schmitten','Toblerone']
    return(
    <>
    <ul>
    {
        chocolate.map((chk)=><Chocolate brand={chk}/>)

    }
    </ul>
     </>);
}
function Chocolate(props)
{
    return <li>I like {props.brand}</li>
}
```

2.2.3 React Events: Adding Events, Passing arguments, Event Objects

- Just like HTML DOM events, React can perform actions based on user events.
- React has the same events as HTML: click, change, mouseover etc.

Adding Events

- React events are written in camelCase syntax:
- onClick instead of onclick.
- React event handlers are written inside curly braces:
- onClick={shoot} instead of onClick="shoot()".

HTML Events

- <button onclick="shoot()">Take the Shot!</button>

REACT Events

- <button onClick={shoot}>Take the Shot!</button>

Example:

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

function Event()
{
    const call=()=>{
        alert("Event Done");
    }
    return (
        <button onClick={call}>Click Me</button>
    )
}
export default Event;
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Event />);
```

Passing Arguments

- To pass an argument to an event handler, use an arrow function.

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

function Event()
{
    const call=(a)=>{
        alert(a);
    }
    return (
        <button onClick={()=>call("Demo")}>Click Me</button>
    )
}
export default Event;
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Event />);
```

Event Objects

- Event handlers have access to the React event that triggered the function.

Example:

```
function Event()
{
    const call=(a,b)=>{
        alert(b.type);
    }
    return (
        <button onClick={(event)=>call("Demo",event)}>Click
Me</button>
    )
}
export default Event;
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Football />);
```