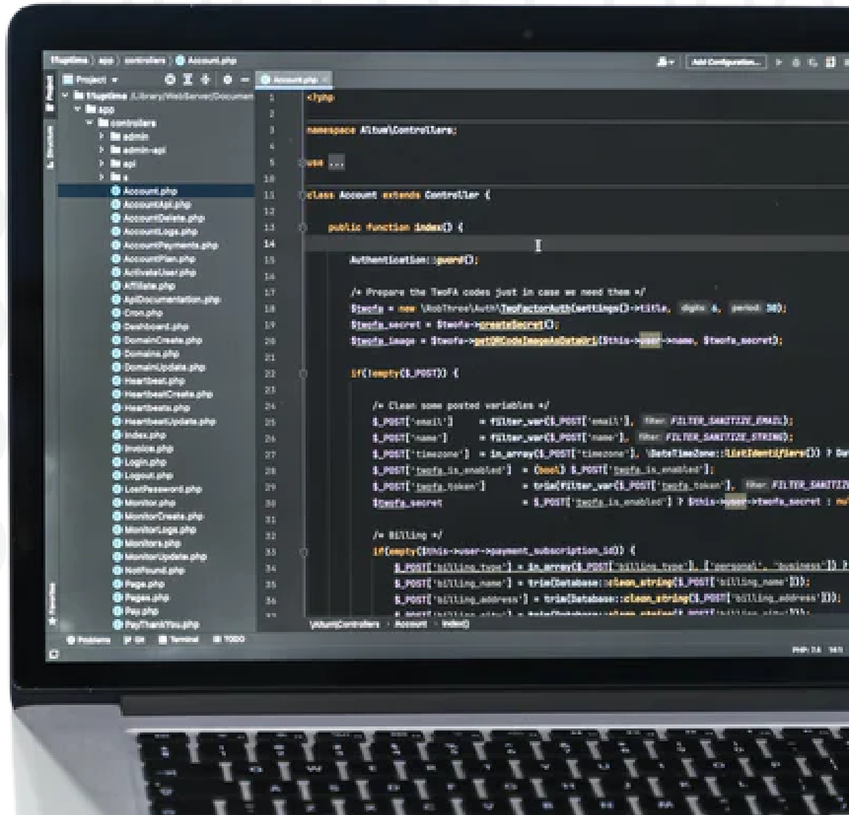
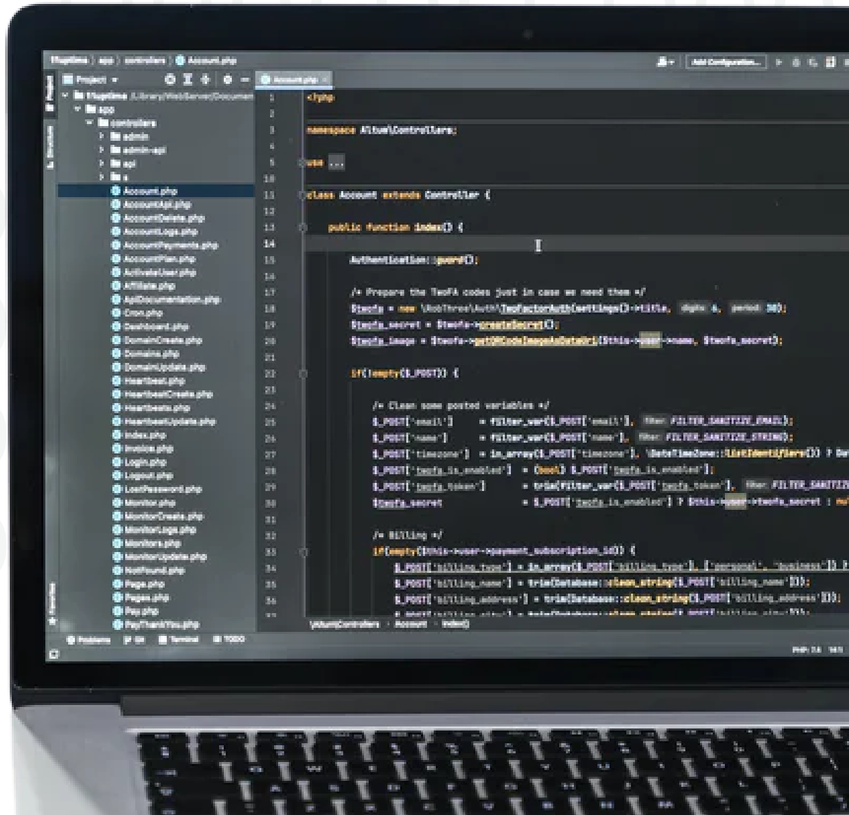
REACT JS

Elite
IT Academy



SESSION 1

INTRODUCTION TO REACT



Elite
IT Academy

TOPICS

- React Elements
- Function and Class Components
- Working with React Components

Elite
IT Academy

What is REACT-JS

- React is a client-side JavaScript library/framework to build user interfaces
- It helps build modern reactive user interfaces for the web
- Uses JavaScript to manipulate the HTML Structure (DOM) of the web page, without fetching a new HTML page
- Declarative, Component focussed programming approach
- Used to develop SPA (Single Page Application)
    - Server sends the main HTML Page
    - React then takes over and controls the UI
- Build Component-Driven User Interfaces
- Build Interactive and Scalable UI's

What is REACT-JS

- Evaluate and Render JSX
- Manage State and Props
- React to User Events & Input
- Re-evaluate Components upon change in State or Props

Setting up for REACT

- Install NodeJs
  - https://nodejs.org/en/download/
- Create a REACT project using the create-react-app program
  - npx create-react-app first-app
- The application is created, and navigated to the directory using
  - cd first-app
- Start the application using
  - npm start

The REACT project

- Project Folders
  - **node_modules** - The node_modules folder is used to save all downloaded packages from NPM on your computer for the REACT project
  - **src** - Contains the source code of the application
    - App.js
    - App.css
    - index.css
    - index.js
  - **package.json -** is used to store the metadata associated with the project as well as to store the list of dependency packages

index.js

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

import modules that are exported by the file.

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

The main entry point of the application

Specifies where the new elements in the application be placed

index.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8" />
  <title>React App</title>
 </head>
 <body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div>
  </body>
</html>
```

app.js

```
import "./App.css";

function App() {
 return (
  <div>
   <h1> Hello World </h1>
  </div>
 );
}

export default App;
```

 **Export app component**

JSX

```
<h1> Hello World </h1>
```

The above code is in JSX (JavaScript XML)

- JSX is a syntax extension to JavaScript
- JSX produces React 'Elements' that will be rendered
- JSX allows programmers to combine markup and UI Logic in a single component
- Babel (JavaScript compiler) compiles JSX down to React.createElement() calls

```
const element = (<h1 className="greeting">
    Hello, world!
  </h1>);
```

```
const element = React.createElement(
    'h1',
    {className: 'greeting'},
     'Hello, world!'
);
```

React.createElement - Examples

```
import React from 'react';
import ReactDOM from 'react-dom';
const title = React.createElement('h1', {}, 'My First React Code');
const container = React.createElement('div', {},title);
ReactDOM.render(container,document.getElementById('global'));
```

React.createElement - Examples

```
import React from 'react';
import ReactDOM from 'react-dom';
const list = React.createElement('div', {},
              React.createElement('h1', {}, 'My favorite ice cream flavors'),
              React.createElement('ul', {},
              [React.createElement('li', { className: 'brown' }, 'Chocolate'),
               React.createElement('li', { className: 'white' }, 'Vanilla'),
               React.createElement('li', { className: 'yellow' }, 'Banana')
               ])) ;

ReactDOM.render(list,document.getElementById('global'));
```

How REACT Works

```
function App() {
 return (
 <div>
 <p> Hello World <p>
 </div>
 );
}
```

```
const para = document.createElement('p');
para.textContent = 'Hello World';
document.getElementById('root').append(para);
```

Creating a new component

```
function SubTitle() {
 const name = "My Name";
 return (
 <h2> Hello {name} </h2>
 );
}

export default SubTitle;
```

```
import SubTitle from './components/SubTitle'

function App() {
 return (
 <div>
 <p> Hello World <p>
 <SubTitle />
 </div>
 );
}

export default App
```

Adding Styles

```
.subtitle {
 display: flex;
 justify-content: space-between;
 align-items: center;
 box-shadow: 0 2px 8px rgba(0, 0, 0, 0.25);
 padding: 0.5rem;
 margin: 1rem 0;
 border-radius: 12px;
 background-color: #4b4b4b;
}
```

```
import "./SubTitle.css";

function SubTitle() {
 const name = "My Name";
 return (
  <div className="subtitle">
   <h2> Hello {name} </h2>
   <p className="subtitle-para">
      This is a paragraph of data in containing a long line
   </p>
  </div>
 );
}

export default SubTitle;
```

Passing Parameters to components

```
function App() {
 return (
  <div>
   <h1> Hello World </h1>
   <SubTitle title="Hello Mayank" para="This is para for Mayank" />
   <SubTitle title="Hello Nitin" para="This is para for Nitin" />
  </div>
 );
}
```

```
function SubTitle(props) {
 return (
  <div className="subtitle">
   <h2> {props.title} </h2>
   <p className="subtitle-para">{props.para}</p>
  </div>
 );
}
```

Passing Parameters to components

```
function App() {
 return (
  <div>
   <h1> Hello World </h1>
   <SubTitle title="Hello Mayank" para="This is para for Mayank" />
   <SubTitle title="Hello Nitin" para="This is para for Nitin" />
  </div>
 );
}
```

```
function SubTitle(props) {
 return (
  <div className="subtitle">
   <h2> {props.title} </h2>
   <p className="subtitle-para">{props.para}</p>
  </div>
 );
}
```

Assignment

Create a Label Component and Display 3 labels in a row on the page.

Use params to pass data into the Label

<table>
<tr><td colspan="2">Name:
Age:</td></tr>
<tr><td>Address:
Line 1
Line 2
Zip Code:</td><td>Contact Details
Email ID:
Tel No:</td></tr>
</table>

Splitting Components

SubTile.js

```
import SubTitleName from "./SubTitleName";
import SubTitlePara from "./SubTitlePara";

function SubTitle(props) {
 return (
  <div className="subtitle">
   <SubTitleName title={props.title} />
   <SubTitlePara para={props.para} />
  </div>
 );
}
```

SubTitleName.js

```
function SubTitleName(props) {
 return <h2> {props.title} </h2>;
}

export default SubTitleName;
```

SubTitlePara.js

```
import "./SubTitle.css";

function SubTitlePara(props) {
 return <p className="subtitle-para">{props.para}</p>;
}

export default SubTitlePara;
```

Assignment

Split the below component into 3 different components

1. TopLabel
2. Address
3. Contact Details

<table>
<tr><td colspan="2">Name:
Age:</td></tr>
<tr><td>Address:
Line 1
Line 2
Zip Code:</td><td>Contact Details
Email ID:
Tel No:</td></tr>
</table>

Creating an Event Handler

SubTitlePara.js

```
import React from "react";
import "./SubTitle.css";

function SubTitlePara(props) {
 return (
  <p  className="subtitle-para"
   onClick={() => {
    console.log("Clicked");
   }}
  >
   {props.para}
  </p>
 );
}
export default SubTitlePara;
```

SubTitleName.js

```
function SubTitleName(props) {
 const clickHandler = () => {
  console.log("THis is Clicked");
 };

 return <h2 onClick={clickHandler}>
{props.title} </h2>;
}

export default SubTitleName;
```

Adding State

SubTitleName.js

```
import React, {useState} from 'react';

function SubTitleName(props) {
  const [title, setTitle] = useState(props.title);
  const clickHandler = () => {
     setTitle("Title Changed");
     console.log(title);
 };

 return <h2 onClick={clickHandler}> {title} </h2>;
}

export default SubTitleName;
```

Inputting Data to Forms

InputForm.js

```
import React from "react";
import "./InputForm.css";

const InputForm = () => {
 return (
  <form>
   <div className="form-controls">
    <div className="form-control">
     <label>Title:</label>
     <input type="text" />
    </div>
    <div className="form-control">
     <label>Para: </label>
     <input type="text" />
    </div>
    <div className="form-control">
     <button type="submit">Add Title</button>
    </div>
   </div>
  </form>
 );
}; export default InputForm;
```

InputForm.css

```
.form-controls {
 display: flex;
 flex-wrap: wrap;
 gap: 1rem;
 margin-top: 1rem;
 margin-bottom: 1rem;
 text-align: left;
}

.form-control {
 display: flex;
 flex-wrap: wrap;
 gap: 1rem;
 margin-bottom: 1rem;
 text-align: left;
}
```

OnChange Event Handler

InputForm.js

```
 const titleChangedHandler = (event) => {
  console.log(event.target.value);
 };

<input type="text" onChange={titleChangedHandler} />
```

InputForm.js

```
 const [formData, setFormData] = useState({
  title: "",
  para: "",
 });

 const titleChangedHandler = (event) => {
  setFormData({
   ...formData,
   title: event.target.value,
  });
  console.log(event.target.value);
 };

 const paraChangedHandler = (event) => {
```

Using a single state & previous state

InputForm.js

```
const [formData, setFormData] = useState({
 title: "",
 para: "",
});

const titleChangedHandler = (event) => {
 setFormData({
  ...formData,
  title: event.target.value,
 });
 console.log(event.target.value);
};

 const paraChangedHandler = (event) => {
 setFormData((prevState) => {
  return {
   ...prevState,
   para: event.target.value,
  };
 });
 console.log(event.target.value);
};


/>
```

2-Way binding

InputForm.js

```
 <div className="form-control">
      <label>Title:</label>
      <input type="text" value={title} onChange={titleChangedHandler} />
</div>
<div className="form-control">
      <label>Para: </label>
      <input type="text" value={para} onChange={paraChangedHandler} />
</div>
<div className="form-control">
      <button type="submit">Add Title</button>
</div>
```

Child to Parent Communication

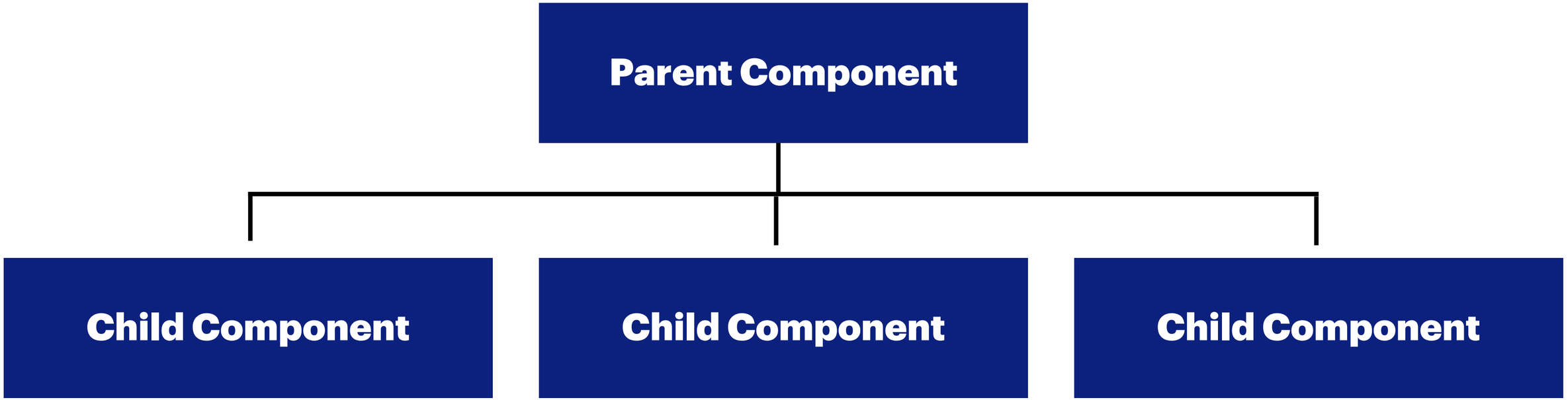
App.js

```
// add handler to call in child
 const onChangedForm = (newData) => {
  console.log(newData);
 };

// Pass the handler to the child as a prop
 <InputForm onChangedData={onChangedForm} />
```

InputForm.js

```
// In child call the handler defined in the props.

const submitHandler = (event) => {
  event.preventDefault();
  let enteredData = {
   title: title,
   para: para,
  };
  props.onChangedData(enteredData);
  setTitle("");
  setPara("");
 };
```

Lifting state up



Controlled, Stateful components

Elite
IT Academy

useRef

- Ref's help you get access to DOM elements in your React program
- Set's up a connection between the HTML Element and Javascript code
- import {useRef} from 'react'
- within the component
  - const nameInputRef = useRef();
  - const addrInputRef = useRef(); etc.
- In the Input tag for the HTML element add a ref prop
  - ref={nameInputRef}
- In SubmitHandler use the nameInputRef to access the current object value
  - nameInputRef.current.value

Class based components

SubTitle.js

```
import { Component } from "react";
import "./SubTitle.css";
import SubTitleName from "./SubTitleName";
import SubTitlePara from "./SubTitlePara";

class SubTitle extends Component {
 render() {
  return (
   <div className="subtitle">
    <SubTitleName title={this.props.title} />
    <SubTitlePara para={this.props.para} />
   </div>
  );
 }
}
export default SubTitle;
```

Converting InputForm to Class-Based

InputForm.js

```
import React, { Component } from "react";
import "./InputForm.css";

class InputForm extends Component {
 constructor() {
  super();
  this.state = {
   title: "",
   para: "",
  };
 }
```

```
titleChangedHandler(event) {
 this.setState({
  title: event.target.value,
 });
 console.log(event.target.value);
}

paraChangedHandler(event) {
 this.setState({
  para: event.target.value,
 });
 console.log(event.target.value);
}
```

Converting InputForm to Class-Based

```
submitHandler(event) {
    event.preventDefault();
    let enteredData = {
        title: this.state.title,
        para: this.state.para,
     };
    this.props.onChangedData(enteredData);
    this.setState({
    title: "",
    para: "",
    });
}
```

Converting InputForm to Class-Based

```
render() {
 return (
  <form onSubmit={this.submitHandler.bind(this)}>
   <div className="form-controls">
    <div className="form-control">
     <label>Title:</label>
     <input
      type="text"
      value={this.state.title}
      onChange={this.titleChangedHandler.bind(this)}
     />
    </div>
    <div className="form-control">
    <label>Para: </label>
    <input
    type="text"
    value={this.state.para}
    onChange={this.paraChangedHandler.bind(this)}
    />
    </div>
    <div className="form-control">
    <button type="submit">Add Title</button>
    </div>
    </div>
    </form>
    );
    }
} export default InputForm;
```

lifecycle methods

componentDidMount() - Called once component is mounted i.e. evaluated and rendered

componentDidUpdate() - Called when component is updated (state changed)

componentWillUnmount() - Called before the component is about to be removed from the DOM

Rendering lists of data

```
function App() {
 const lineStr = "This is para for";
 let subtitles = [
  { Name: "Mayank", Line: lineStr },
  { Name: "Nitin", Line: lineStr },
 ];

 const onChangedForm = (newData) => {
  console.log(newData);
 };
```

```
return (
 <div>
  <InputForm onChangedData={onChangedForm} />
  <h1> Hello World </h1>
  <SubTitle
   title={"Hello " + subtitles[0].Name}
   para={subtitles[0].Line + " " + subtitles[0].Name}
  />
  <SubTitle
   title={"Hello " + subtitles[1].Name}
   para={subtitles[1].Line + " " + subtitles[1].Name}
  />
 </div>
);}
```

Rendering lists of data

```
function App() {
 const lineStr = "This is para for";
 let subtitles = [
  { Name: "Mayank", Line: lineStr },
  { Name: "Nitin", Line: lineStr },
 ];

 const onChangedForm = (newData) => {
  console.log(newData);
 };
 return (
 <div>
     <InputForm onChangedData={onChangedForm} />
     <h1> Hello World </h1>
     {subtitles.map((el) => (
         <SubTitle title={"hello " + el.Name}
                   para= {el.Line + " " + el.Name} />
       ))}
 </div>
 );
}
export default App;
```

useEffect

- Used to handle side effects
- useEffect(() => {...}, [dependencies]
  - A function body that is executed after every component evaluation, if the specified dependencies have changed
  - Dependencies of this effect, if no dependencies are specified then the useEffect is called only once when the component is created.
  - If dependencies then the function is called whenever the dependencies are changed.
  - return () => {...}; cleanup function called before the next call to useEffect and when the component is downloaded.

React Contexts

- **Used to share state across multiple components.**

Step1: Create the context object

```
import React from 'react'

const MyContext = React.createContext({
... State objects
}
export default MyContext
```

Step 2: Wrap components that should have access to the context

```
import MyContext from "./store/MyContext"

<MyContext.Provider> value = {{
                    ... initialize context
                    }}
... enclosed Components that will have
access to the state.
</MyContext.Provider>
```

React Contexts

Step 3: Listen to the context

Method 1: Using Consumer

```
return(
  <MyContext.Consumer>
  {(ctx) => {
      return (
      ... JSX - ctx.state can be accessed
       )
   }})
```

method 2: useContext hook

```
import React, {useContext} from 'react'
import MyContext from '../../store/MyContext'

// in functional componenent
const ctx = useContext(MyContext);
```

Using Redux for app-wide state

- Kinds of State:
  - Local State
  - Cross-Component State
  - App-Wide State
- Redux is used to create a Central store of data (State)
- Components can subscribe to the store, and are notified when the data changed
- Components cannot directly manipulate data in the store
- Reducer functions are used to update the store data
- Components trigger a change in the data store by dispatching Actions
- Redux forwards Actions to the reducer which performs the Actions which changes the state, and then subscribing components are notified about the change
- install redux and react-redux.

Using Redux

Elite IT Academy

```
Step 1: Create a Store
import {createStore} from 'redux';

const storeReducer = (state = {myData: 0}, action) => {
   if (action.type === 'savemydata') {
       return {
          myData: action.value,
        }
   return state;
};
const store = createStore();
export default store;
```

Connect application to store

Step 2: Provide store to application in index.js

```
import {Provider} from 'react-redux';
import store from './store/store';


root.render(<Provider store={store}>App </Provider>);
```

Subscribing to the Store

Step 3: In the component where the store needs to be accessed

```
import {useSelector, } from 'react-redux';
```

In functional component, subscribe to the store by using the useSelector hook

```
   const myData = useSelector(state => state.myData);
```

Now the store is subscribed to any changes in myData will reflect in the component via the variable myData.

Dispatching actions to the Store

Step 4: In the same component where the store needs to be accessed, add the dispatch hook

```
import {useSelector, useDispatch} from 'react-redux';

In the functional component,
   const dispatch = useDispatch();

In handler to access the store
   const saveHandler = () => {
       dispatch({type: 'savemydata', value: data});
    }
```

Using Redux Toolkit

npm install @reduxjs/toolkit

Step 1: Creating a Slice

```
import {createSlice} from '@reduxjs/toolkit'

const mySlice = createSlice({
    name: "mySlice",
    initialState: {myState: ""},
    reducers: {
       saveMyState(state, action) {
          state.myState = action.payload;
          }
      }
});
```

Using Redux Toolkit

Step 2: Configure the slice

```
import {createSlice, configureStore} from '@reduxjs/toolkit'

const store = configureStore({
      reducer: {
         myState: mySlice.reducer
         }
      });
```

Using Redux Toolkit

Step 3: Export action for each slice in the store

```
export const mySliceActions = mySlice.actions;
```

Step 4: In the component using the slice

```
import {mySliceActions} from "./store/store'
const myData = useSelector((state) => state.myState.myData);
```

Step 5: In the dispatch use the action object to dispatch a call to the method in the action

```
dispatch(mySliceActions.saveMyState(data));
```

useRef

- Ref's help you get access to DOM elements in your React program
- Set's up a connection between the HTML Element and Javascript code
- import {useRef} from 'react'
- within the component
  - const nameInputRef = useRef();
  - const addrInputRef = useRef(); etc.
- In the Input tag for the HTML element add a ref prop
  - ref={nameInputRef}
- In SubmitHandler use the nameInputRef to access the current object value
  - nameInputRef.current.value