React Code Splitting :-

The react app budled their files using tools like Webpack or Browsefy. Bundling is a process which takes multiple files and merge them into a single file, which is called bundle. The bundle is responsible for loading an entire app at ones on the webpage.

Code splitting improves:-

① The performance of the app.
② The impact on memory
③ The downloadable killobytes size.

React.lazy:-

The best way for code splitting into the app is through the dynamic import () syntax. The React.lazy function allows us to render a dynamic import as a regular component.

Before:-

```
import ExampleComponent from '/ExampleComponent';
function MyComponent (){
  return (
    <div>
    <ExampleComponent />
    </div>
  );
}
```

After:-

```
const ExampleComponent = React.lazy (() =>
          import ('/ExampleComponent'));
```

```
function MyComponent () {
  return (
    <div>
    <ExampleComponent/>
    </div>
  );
}
```

Error Boundaries:-
If any module fails to load, for example, due to network failure, we will get an error. We can handle these errors with Error Boundaries. Ones we have created the Error Boundary, we can use it anywhere

```
import MyErrorBoundary from '/MyErrorBoundary
const ExampleComponent = React.lazy(() =>
                    import ('/Example Component'
Const Example Component = React.lazy ())
  Const MyComponent = () => (
    <div>
    <MyErrorBoundary >
   Suspence fallback = {<div > Loading </div>}
    <section>
    <ExampleComponent/>
    <ExampleComponent />
    </section>
    </suspense>
   </MyErrorBoundary>
  </div>
  );
```

React Context :-

Context allows passing data through the component tree without passing props down manually at every level.

React Context API :-

1. React.createContext
2. Context.provider.
3. Context.Consumer.
4. class.contextType.

① React.createContext :-

It creates a context object

```
const MyContext = React.createContext(default.value
```

② Context.Provider :-

Every Context object has a provider React component which allows consuming components to subscribe to context changes.

```
<MyContext.Provider value = {/* Some value */}>
```

③ Context.Consumer.

It is an React Component which subscribes to the context changes. It allows us to the context within an function.

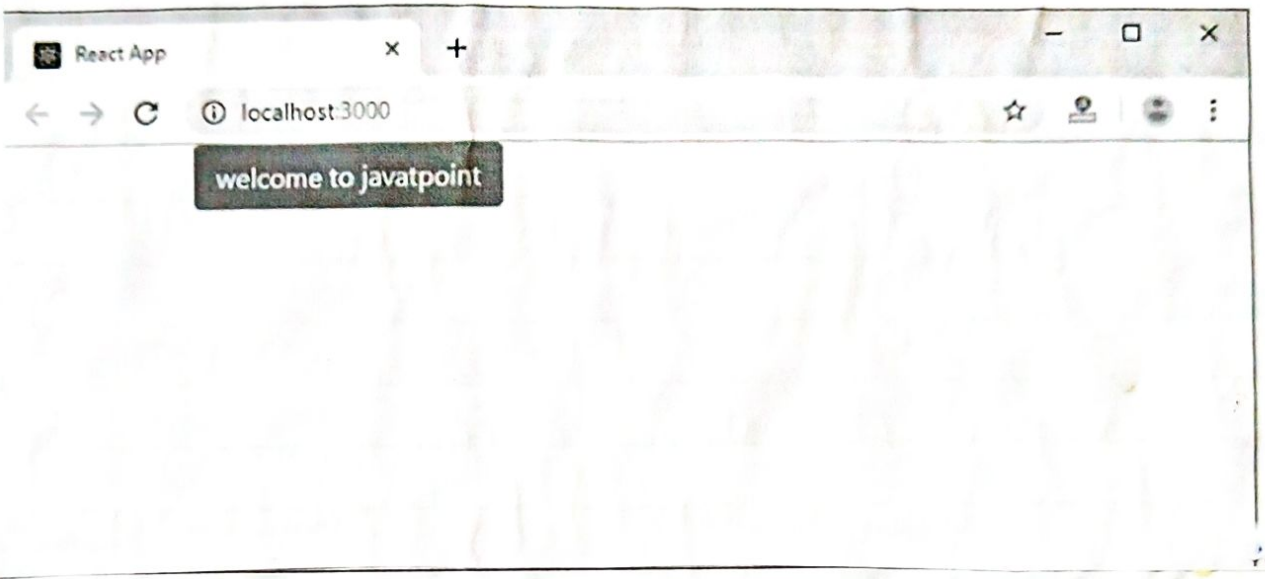
```
<MyContext.Consumer>
    {value => /* render something which is
        based on the context value */}
```

Class contextType:-

The contextType property on a class used to asign a context object which is created by React.createContext(). It allows you to consume the closet current value of that context using this.context.

```
import React, {Component} from 'react';
import 'bootstrap/dist/css/bootstrap.min.css';
Const BtnColorContext = React.createContext('
  btn btn-darkyellow");
class App extends Component {
  render() {
    return(
  <BtnColorContext.Provider value='btn btn-info
  </Button>
  </BtnColorContext.Provider>
  );
  }
}
function Button(props) {
  return (
  <div className="container">
  </ThemeButton />
</div>
  };
}
class ThemedButton extends Component {
  static contextType = BtnColorContext;
  render() {
  return <button className={this.context}>
  Welcome to JavaTpoint
```

```
</button>
}
}
export default App;
```



React Hooks:-

Hooks are the new feature introduced in React 16.8 version. It allows you to use state and other React feature without writing a class.

Rules of Hooks:-

1. Only call Hooks at the top level
2. Only call Hooks from React functions.

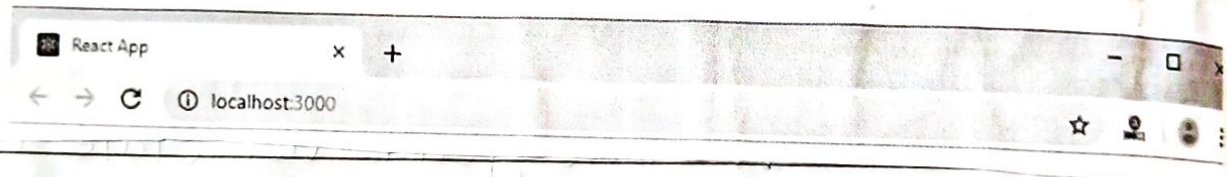
Hooks State:-

Hook state is the new way of declaring a state in React app. Hook uses useState() functional component for setting and retrivi g state.

App.Js

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

```
function CountApp() {
  const [count, setCount] = useState(0);
  return (
   <div>
   <p> You clicked {count} times </p>
   <button onClick={() => setCount(count+1
    Click me
   </button>
  </div>
  );
}
export default CountApp
```



React Flux Vs MVC

MVC

MVC stands for Model View Controller. It is an architectural pattern used for developing the user interface

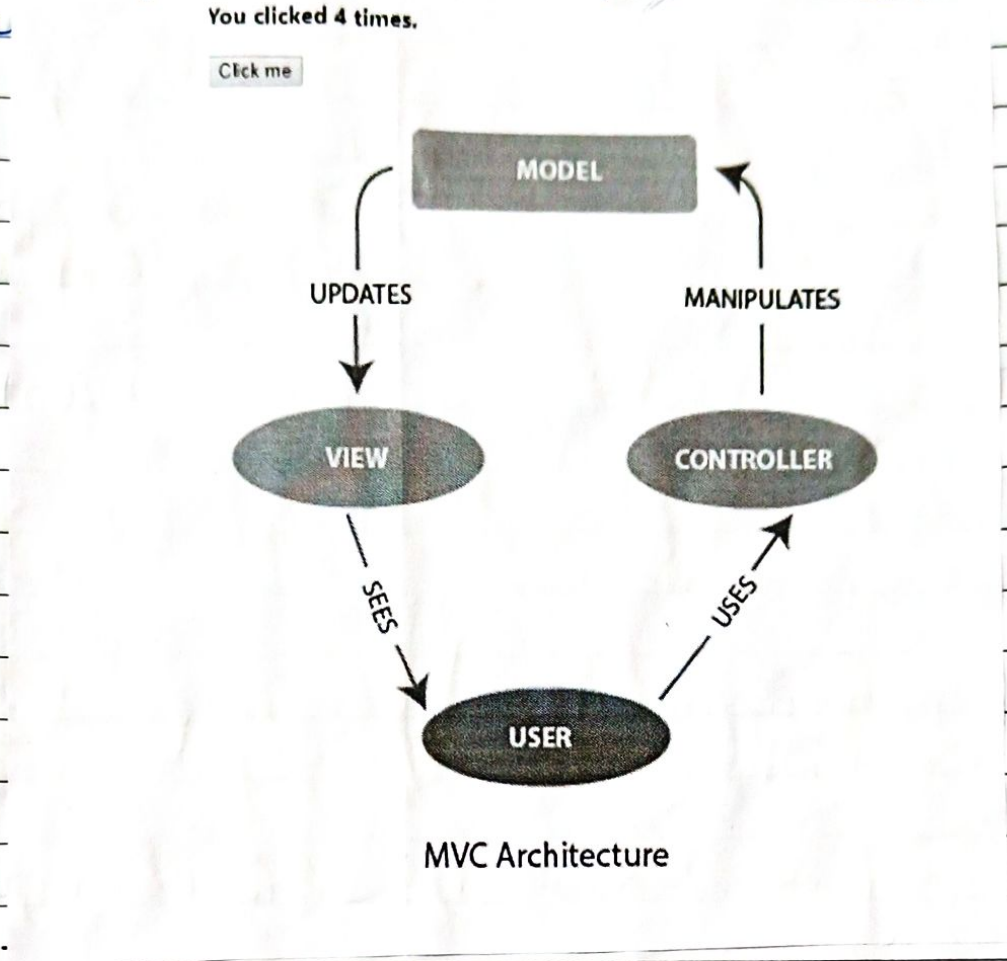
MVC Architecture :-

Model :- It is responsible for maintaining th behaviour and data of an application

View :- It is used to display the model in the user interface.

Controller :- It acts as an interface between

the Model and the view Components. It takes user input, manipulates the data and causes view

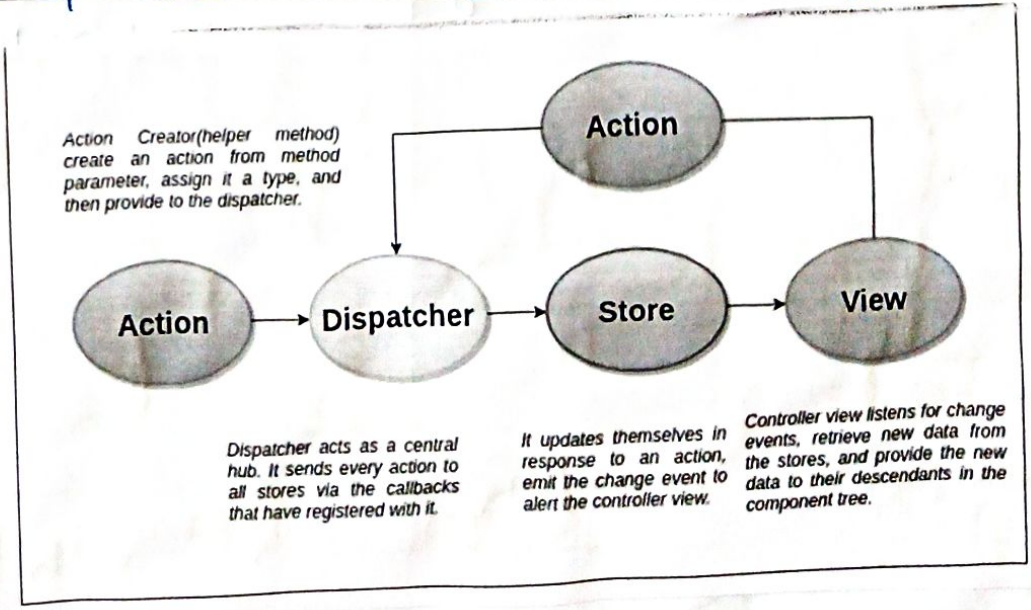


MVC Architecture

Flux :-

Flux is an application architecture that Facebook uses for building client-side web application Flux architecture has 3 Major rules in dealing with data :-

① Dispatcher ② Store ③ views.



	MVC	Flux
①	It is introduced in 1976	It was introduced just a few years ago
②	It supports Bi-directional data Flow model	It support Uni-directional data flow model
③	In this, data binding is the key	In this events or actio are key
④	It is synchronous	It is asynchronous.
⑤	Controllers handle everything	Store handle all logi
⑥	It is hard to debug	It is easy to debug.
⑦	It is difficult to understand as project size increases	It is easy to the understand.
⑧	Its maintainability is difficult as the project scope goes huge.	It maintainability is easy and reduces run-time errors.
⑨	Testing of application is difficult.	Testing of application is easy.
⑩	Scalability is complex	It can be easily scalable.

React Redux :-

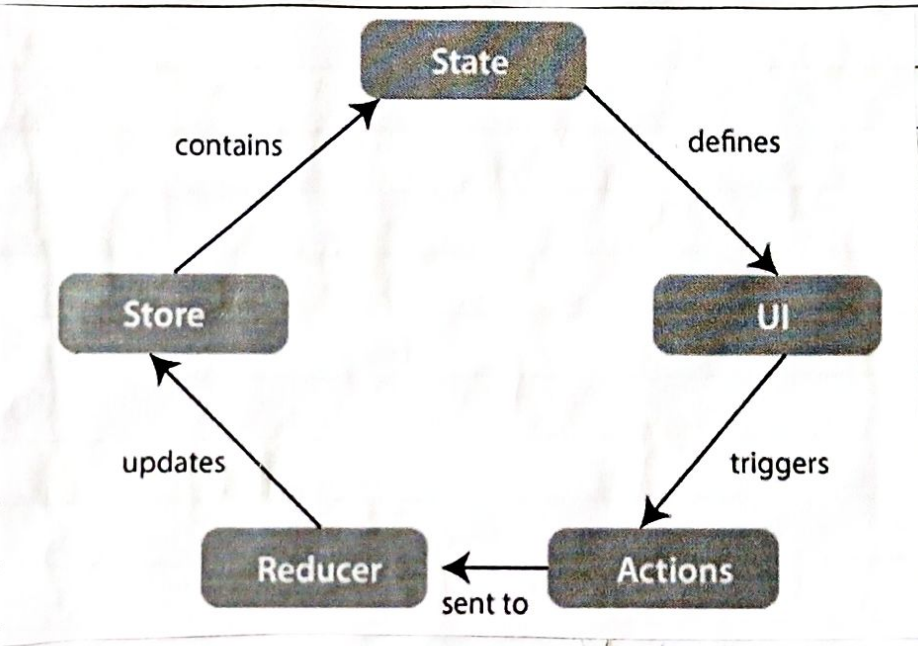
Redux was inspired by flux. Redux studied the Flux architecture.

① Redux does not have dispatcher concept.

② Redux has an only store whereas Flux has many stores.

③ The Action objects will be received and handled directly by Store.

React Architecture :-



Store :- A store is a place where the entire state of your application lists. It manages the status of the application and has a dispatch function. It is like a brain responsible for all moving parts in Redux.

Action:- Action is sent or dispatched from the view has which are payloads that can be read by Reducers. It is pure object created

to the store the information of users event. It includes information such as type of action time of occurances, location of occurances, its coordinates and which state it aims to change.

Reducer :- Reducer reads the payload from the action and then updates the store via the state accordingly. It is pure function to return a new state from the initial state.

React Portals :-

It is very tricky to render the child component outside of its parent component hierarchy. It breaks the convention where a component needs to render as a new element and follow a parent-child hierarchy.

```
ReactDOM.createPortal(child, container)
```

Example using Portal :-
We want to insert a child component in different location in the DOM

```
render() {
  return ReactDOM.createPortal(
    this.props.children,
    myNode,
  );
}
```

Features :-

- It uses React version 16 and its official API for creating portal
- It has a fallback for React version 15.
- It transports its children component into a new React portal which is appended by default to document.body.
- It can also target user specified DOM element.
- It supports server-side rendering
- It supports returning arrays.
- It doesn't produce any DOM mess.
- It does no dependancies, minimalistics.

When to use ?

1. Modals.
2. Tooltips
3. Floating menus.
4. Widgets

Explanation of React Portal.

App.Js.

```
import React, {Component} from 'react';
import './App.css'
import PortalDemo from './PortalDemo.Js';
Class App extends Component {
```

```
render() {
  return (
  <div className = "App">
    <PortalDemo />
  </div>
    );
  }
}
export default App;
```

PortalDemo.Js

```
import React from 'react'
import ReactDOM from 'react-dom'
function PortalDemo(){
   return ReactDOM.createPortal(
    <h1> Portals Demo </h1>
   document.getElementById('Portal-root')
  )
}
export default PortalDemo
```

index.html.

```
<! DOCTYPE html >
 <html lang = "en">
  <head>
  <meta charset = "utf-8" />
  <link rel = "shortcut icon" href =
                "%.PUBLIC-URL %.favicon.ico"
<meta name = "viewport" content =
        "width = device-width, initial-scale=1
<meta name = "theme-color" content =
                        "#000000" />
```

```
<title> React App </title>
 </head>
<body>
<noscript> It required to enable Javascript
    to run this app </noscript>
<div id = "root "> </div>
 </body>
</html>
```