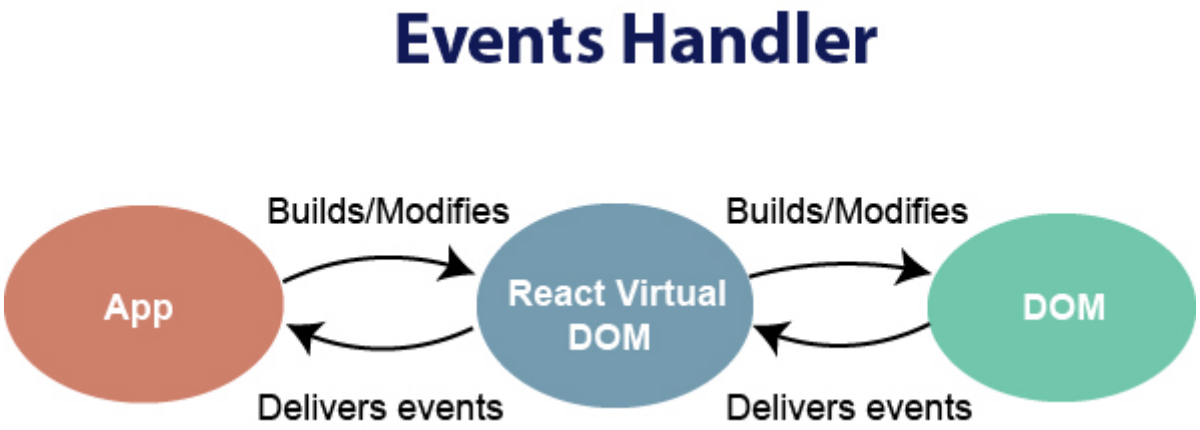
PART-B:

1.What is an event in React?

An event is an action that could be triggered as a result of the user action or system generated event. For example, a mouse click, loading of a web page, pressing a key, window resizes, and other interactions are called events.

React has its own event handling system which is very similar to handling events on DOM elements. The react event handling system is known as Synthetic Events. The synthetic event is a cross-browser wrapper of the browser's native event.



Handling events with react have some syntactic differences from handling events on DOM. These are:

1. React events are named as **camelCase** instead of **lowercase**.
2. With JSX, a function is passed as the **event handler** instead of a **string**. For example:

Event declaration in plain HTML:

```
<button onclick="showMessage()">
    Hello JavaTpoint
</button>
```

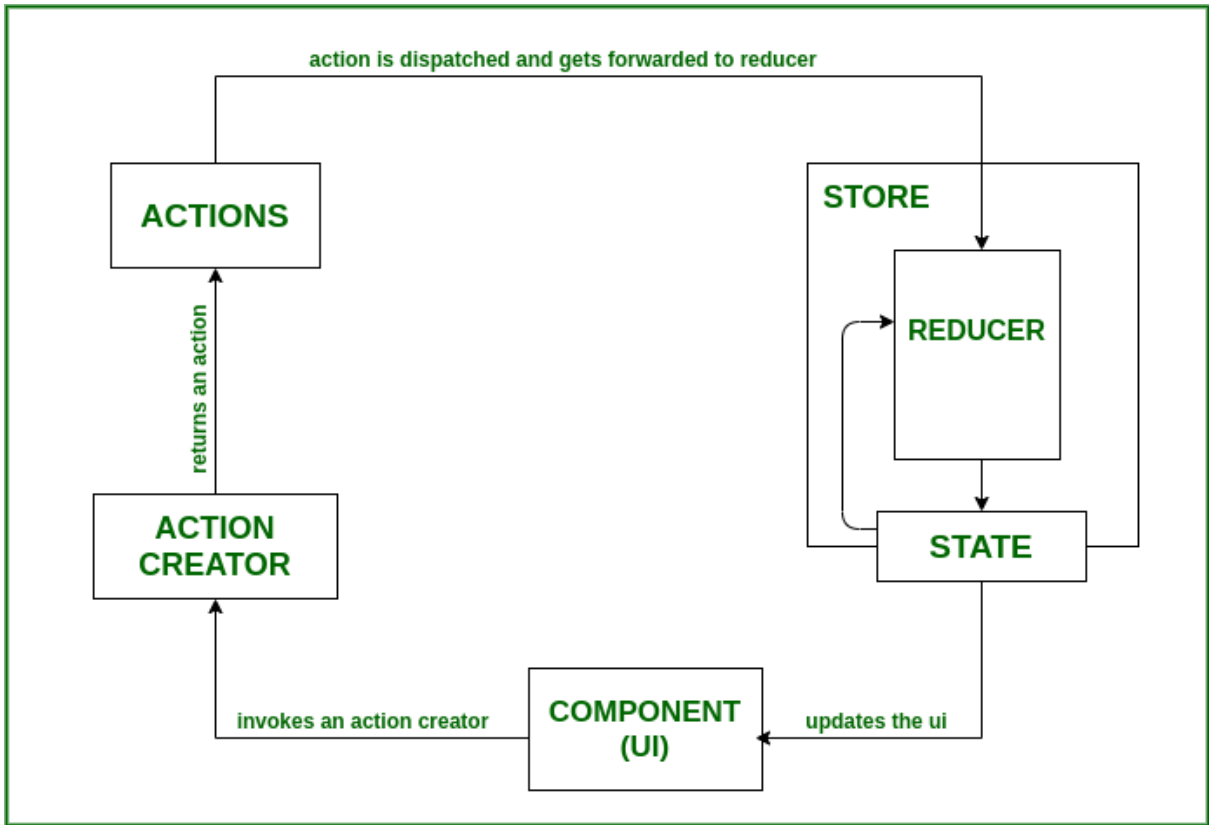
Event declaration in React:

```
<button onClick={showMessage}>
    Hello JavaTpoint
</button>
```

2.Draw a diagram showing how data flows through Redux.

There are four fundamental concepts that govern the flow of data in React-Redux applications.

1. **Redux store:** The Redux store, simply put, is an object that holds the application state. A redux store can consist of small state objects which are combined into one large object. Any component in the application can easily access this state (store) by hooking up to it through the connect method.
2. **Action creators:** Action creators, as the name suggests, are functions that return actions (objects). Action creators are invoked when the user interacts with the application through its UI (button click, form submission, etc) or at certain points in a component's lifecycle (component mounts, component un-mounts, etc).
3. **Actions:** Actions are simple objects which conventionally have two properties- type and payload. The type property is usually a string that specifies identifies the action, and the payload is an optional property that contains some data that is required to perform any particular task. The main function of action is to send data from the application to the Redux store.
4. **Reducers:** Reducers are pure functions that update the state of the application in response to actions. Reducers take a previous state and an action as the input and return a modified version of the state. Since the state is immutable, a reducer always returns a new state, which is an updated version of the previous state.



React-Redux Application Flow

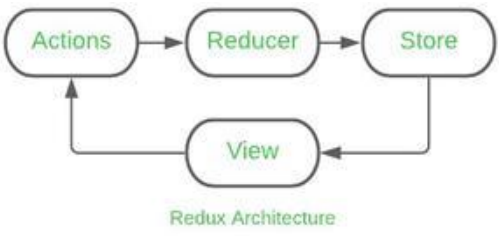
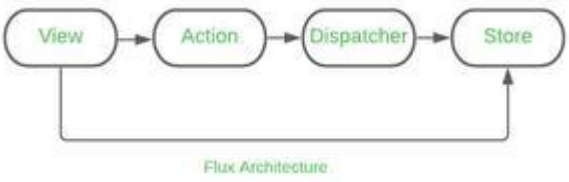
- The flow of data in a React-Redux application begins at the component level when the user interacts with the application UI. This interaction leads to the action creators dispatching an action.
- When an action is dispatched, it is received by the root reducer of the application and is passed on to all the reducers. Thus, it becomes the reducer's task to determine if it needs to update the state based on the dispatched action.
- This is checked by using a simple switch statement to filter out the required actions. Each (smaller) reducer in the application accepts the dispatched action and if the type of the dispatched action matches, it returns a newly updated state.
- It is essential to note here that the state never actually changes in redux. Instead, the reducer always generates a new state which is a copy of the old state, but with some modifications.
- The store then informs the component about the new state which in turn retrieves the updated state and re-renders the component.
- Another important observation here is that flow of data in a React-Redux application is unidirectional, i.e., it only goes in one direction.

3.How will you distinguish Redux from Flux ?

Flux: Flux is the application architecture or we can say JavaScript architecture that uses for building client-side web applications or UI for

client applications. you can start using flux without a lot of new code. flux overcome the drawbacks of MVC such as instability and complexity.

Difference between Redux and Flux:

o	Redux	Flux
1.	It was developed by Dan Abramov & Andrew Clark.	It was developed by Facebook.
2.	It is an Open-source JavaScript library used for creating the UI.	It is Application architecture designed to build client-side web apps.
3.	Redux has mainly two components • Action Creator • Store	Flux has main four components : • Action • Dispatcher • Store • View
4.	Redux does not have any dispatcher.	Flux has a single dispatcher.
5.	Redux has only a single store in the application.	Flux has multiple stores in one application.
6.	In Redux, Data logics are in the reducers.	In flux, Data logic is in store.
7.	In Redux store's state cannot be mutable	In flux store's state can be mutable.
8.	 Redux Architecture	 Flux Architecture

4.What are the advantages of using Redux?

Redux is a **state management tool** for JavaScript applications. It is more commonly used with ReactJS but is also compatible with many other frameworks such as Angular, Vue, Preact, as well as vanilla JavaScript. It is

important to note that even though React and Redux are frequently used together, they are independent of each other!

Advantages of using Redux:

1. Centralized state management system i.e. Store: React state is stored locally within a component. To share this state with other components in the application, props are passed to child components, or callbacks are used for parent components. Redux state, on the other hand, is stored globally in the store. All the components of the entire application can easily access the data directly. This centralizes all data and makes it very easy for a component to get the state it requires. So while developing large, complex applications with many components, the Redux store is highly preferred.

2. Performance Optimizations: By default, whenever a component is updated, React re-renders all the components inside that part of the component tree. In such a case when the data for a given component hasn't changed, these re-renders are wasted (cause the UI output displayed on the screen would remain the same). Redux store helps in improving the performance by skipping such unnecessary re-renders and ensuring that a given component re-renders only when its data has actually changed.

3. Pure reducer functions: A pure function is defined as any function that doesn't alter input data, doesn't depend on the external state, and can consistently provide the same output for the same input. As opposed to React, Redux depends on such pure functions. It takes a given state (object) and passes it to each reducer in a loop. In case of any data changes, a new object is returned from the reducer (re-rendering takes place). However, the old object is returned if there are no changes (no re-rendering).

4. Storing long-term data: Since data stored in redux persists until page refresh, it is widely used to store long-term data that is required while the user navigates the application, such as, data loaded from an API, data submitted through a form, etc. On the other hand, React is suitable for storing short-term data that is likely to change quickly (form inputs, toggles, etc.)

5. Time-travel Debugging: In React, it becomes a tedious task to track the state of the application during the debugging process. Redux makes debugging the application an easy process. Since it represents the entire state of an application at any given point in time, it is widely used for time-travel debugging. It can even send complete error reports to the server!

6. Great supportive community Since redux has a large community of users, it becomes easier to learn about best practices, get help when stuck, reuse your knowledge across different applications. Also, there are a number of extensions for redux that help in simplifying the code logic and improving the performance.

5 Where would you put AJAX calls in your React code?

In most modern web applications, the backend is separated from the frontend. Therefore, it needs to fetch data from a remote endpoint(server), by making AJAX requests.

These AJAX requests should be made *after the component has been rendered to the browser*, and usually, they need to be made only once. Thus, the best place to make the requests is **inside an useEffect hook**. Let us take a look at how we can implement this.

Creating React Application:

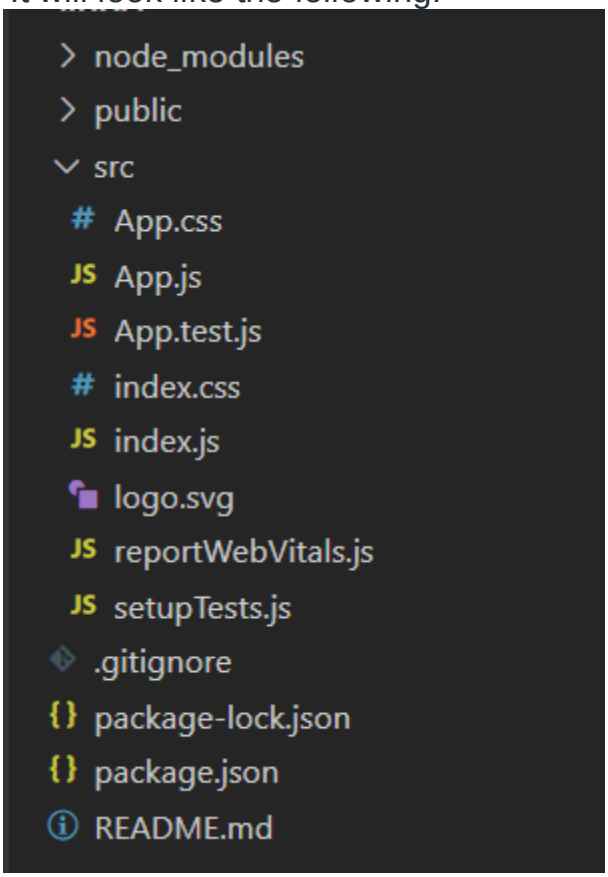
Step 1: Create a React application using the following command:

```
npx create-react-app myapp
```

Step 2: After creating your project folder, i.e. myapp, move to it using the following command:

```
cd myapp
```

Project Structure: It will look like the following.



Initial folder structure

Implementation: Now that we have the project set up, let us replace the content of App.js, located inside the **src** folder, with the following code.

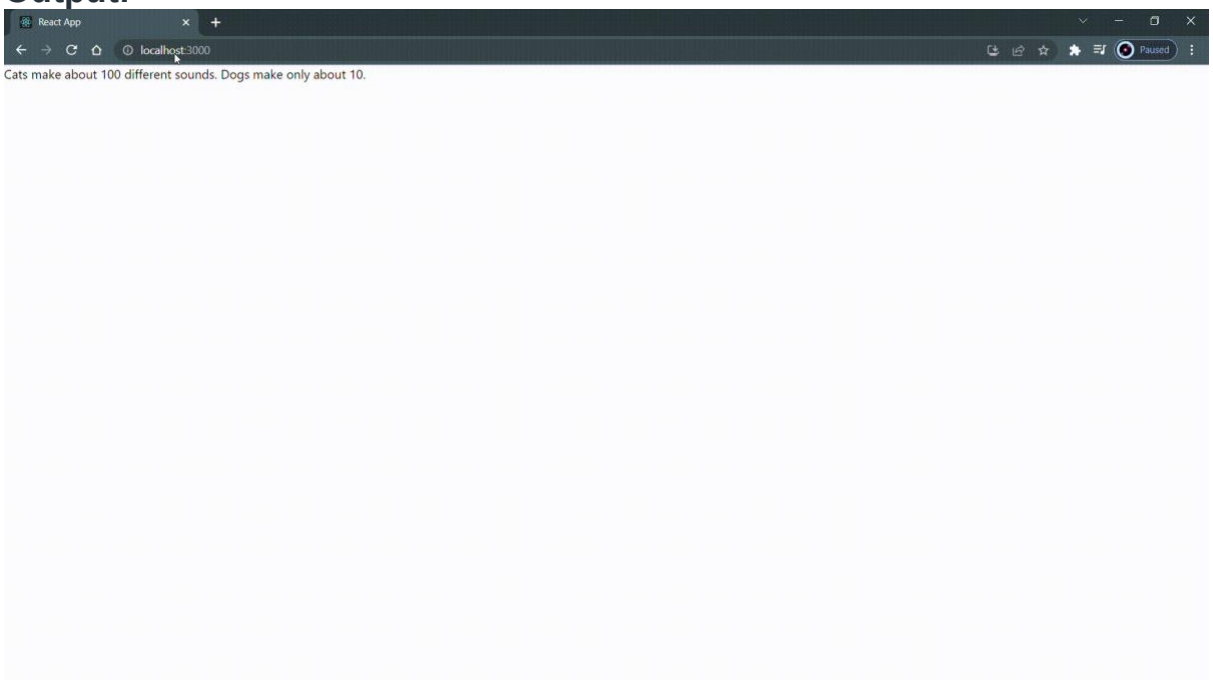
- App.js

```
import React, { useState, useEffect } from "react";

const App = () => {
  const [text, setText] = useState("Data has not been fetched yet!");

  useEffect(() => {
    fetch("https://catfact.ninja/fact")
      .then((response) => response.json())
      .then(({ fact }) => setText(fact))
      .catch((error) => {
        setText("An error occurred!");
      });
  }, []);

  return <div>{text}</div>;
};

export default App;
```

Step to run the application: Use the following command to run the app in development mode, and open http://localhost:3000 to view it in the browser:

```
npm start
```

Output:



Every time we reload, the displayed text changes from "Data has not been fetched yet!" to the response data

Explanation: Here, we make an AJAX request inside the useEffect hook, taking the help of a free-to-use API. The useEffect hook has a second parameter: an empty array. This array is known as the dependency array, and if any item present in the dependency array changes between successive

renders of the component, the code inside that useEffect is executed (there can be more than one useEffect in a component). Thus, an empty dependency array means that the code inside our useEffect will run only once, at first render.
Once we get the response, we change the value of the text variable from *"Data has not been fetched yet!"* to the response data. Thus we can see that the AJAX request has been made successfully.

6 You must've heard that "In React, everything is a component." What do you understand from the statement?

Components are the building blocks of a React application's UI. These components split up the entire UI into small independent and reusable pieces. Then it renders each of these components independent of each other without affecting the rest of the UI.

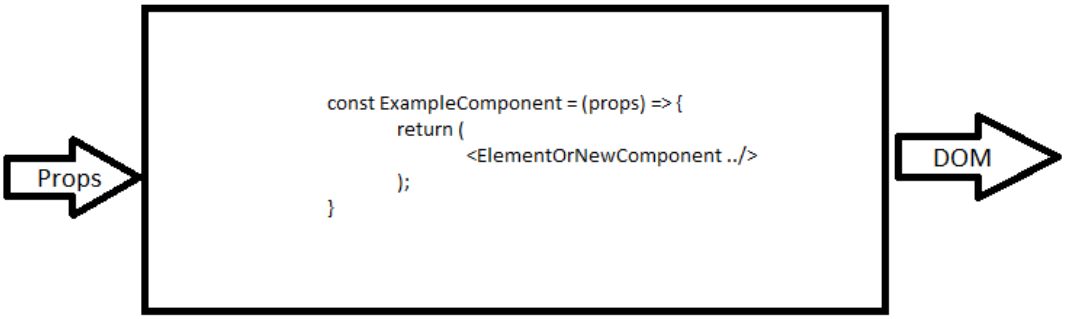
A Component in React.js is one of the most important concepts to understand. As the official website, reactjs.org explains it – Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called "props") and return React elements describing what should appear on the screen. Components let you split the UI into independent, reusable pieces, and think about each piece in isolation. It is an independent, reusable code that contains HTML + Javascript.

React component can be of two types – It can be either **class** or **function** component. You might have heard different terms to describe these types, i.e **stateless** and **stateful**. This state can be modified according to user action or other factors. Without further ado, let's get to the heart of the matter which is 2 ways to create simple React components:

1) Stateless Functional Component

A function component is the simplest form of a React component. It is a simple function that returns a simple contract. The function component receives an object of properties

which is usually named **props.** It returns what looks like HTML, but is really a special JavaScript syntax called JSX. We call such components “function components” because they are literally JavaScript functions.



This function is a valid React component because it accepts a single “props” (which stands for properties) object argument with data and returns a React element.

When React sees an element representing a user-defined component, it passes JSX attributes to this component as a single object. We call this object “props”. Props are similar to arguments for pure functions argument. Props of the component are passed from parent component which invokes component. Props in a component cannot be modified (Immutable).

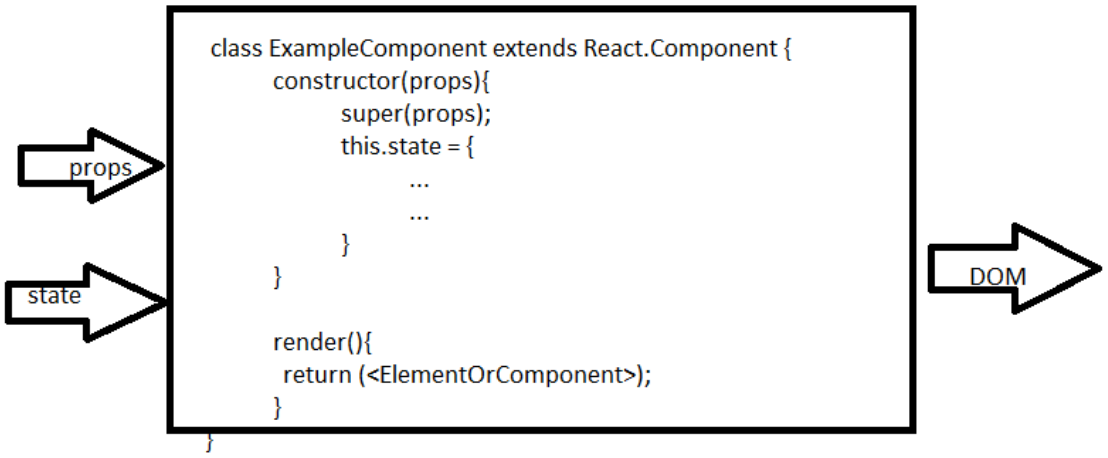
Example:

```
const ExampleComponent = () => {
    return <div>
            <h1>Hello People!</h1>
            <p>This is Functional React Component.</p>
        </div>
}
```

Here we don't need write **render()** function. We just return the HTML div. In this way, we can reduce the amount of code.

2) Class Component

A **class component** is a more featured way to define a React component. It also acts like a function that receives props, but that function also considers a private internal state as additional input that controls the returned JSX.



This internal state is what gives React its **reactive** nature. When the state of a class component changes, React will re-render that component in browser.

The State and Props objects have one important difference. Inside a class component, the State object can be changed while the Props object represents fixed values. Class components can only change their internal state, not their properties. This is a core idea to understand in React.

Example:

```
class ExampleComponent extends React.Component{
    render() {
      return <div>
            <h1>Hello People!</h1>
            <p>This is my first React class Component.</p>
          </div>
      }
  }
```

You would then use ReactDOM.render() to target the HTML element by id.

And that's it! These are the 2 ways you can create a React component.

7. Can browsers read JSX?

JSX is not a valid JavaScript as they are embedded in HTML elements. As JSX is combination of HTML and JavaScript it is not supported by Browsers. So, if any file contains JSX file, Babel transpiler converts the JSX into JavaScript objects which becomes a valid JavaScript. Thus, browsers understands the code and executes. Browsers can't read JSX because there is no inherent implementation for the browser engines to read and understand them. JSX is not intended to be implemented by the engines or browsers, it is intended to be used by various transpilers to transform these JSX into valid JavaScript code.

8 Define HOC in React?What are the benefits of HOC?

A higher-order component (HOC) is an advanced technique in React for reusing component logic. HOCs are not part of the React API, per se. They are a pattern that emerges from React's compositional nature.

Concretely, **a higher-order component is a function that takes a component and returns a new component.**

```
const EnhancedComponent = higherOrderComponent(WrappedComponent);
```

Whereas a component transforms props into UI, a higher-order component transforms a component into another component.
HOCs are common in third-party React libraries, such as Redux's `connect` and Relay's `createFragmentContainer`.

Reason to use Higher-Order component:

- Easy to handle
- Get rid of copying the same logic in every component
- Makes code more readable

9 What is a store in Redux?

A store is an immutable object tree in Redux. A store is a state container which holds the application's state. Redux can have only a single store in your application. Whenever a store is created in Redux, you need to specify the reducer.

Let us see how we can create a store using the **createStore** method from Redux. One need to import the createStore package from the Redux library that supports the store creation process as shown below −

```
import { createStore } from 'redux';
import reducer from './reducers/reducer'
const store = createStore(reducer);
```

A createStore function can have three arguments. The following is the syntax −

```
createStore(reducer, [preloadedState], [enhancer])
```

A reducer is a function that returns the next state of app. A preloadedState is an optional argument and is the initial state of your app. An enhancer is also an optional argument. It will help you enhance store with third-party capabilities.

A store has three important methods as given below −

getState

It helps you retrieve the current state of your Redux store.

The syntax for getState is as follows −

```
store.getState()
```

dispatch

It allows you to dispatch an action to change a state in your application.

The syntax for dispatch is as follows −

```
store.dispatch({type:'ITEMS_REQUEST'})
```

subscribe

It helps you register a callback that Redux store will call when an action has been dispatched. As soon as the Redux state has been updated, the view will re-render automatically.

The syntax for dispatch is as follows −

```
store.subscribe(()=>{ console.log(store.getState());})
```

Note that subscribe function returns a function for unsubscribing the listener. To unsubscribe the listener, we can use the below code −

```
const unsubscribe = store.subscribe(()=>{console.log(store.getState());});
unsubscribe();
```

10 What is an arrow function and how is it used in React?

Arrow functions are **a new way to write anonymous function expressions**, and are similar to lambda functions in some other programming languages, such as Python. Arrow functions differ from traditional functions in a number of ways, including the way their scope is determined and how their syntax is expressed.

Arrow functions were introduced in ES6.

Arrow functions allow us to write shorter function syntax:

```
let myFunction = (a, b) => a * b;
```

Before:

```
hello = function() {
  return "Hello World!";
}
```

With Arrow Function:

```
hello = () => {
  return "Hello World!";
}
```

It gets shorter! If the function has only one statement, and the statement returns a value, you can remove the brackets *and* the `return` keyword:

Arrow Functions Return Value by Default:

```
hello = () => "Hello World!";
```

11 How is React different from React Native?

ReactJS

ReactJS is an open-source JavaScript library used to build the user interface for Web Applications. It is responsible only for the view layer of the application. It provides developers to compose complex UIs from a small and isolated piece of code called "components." ReactJS made of two parts first is components, that are the pieces that contain HTML code and what you want to see in the user interface, and the second one is HTML document where all your components will be rendered.

Jordan Walke, who was a software engineer at Facebook, develops it. Initially, it was developed and maintained by Facebook and was later used in its products like WhatsApp & Instagram. Facebook developed ReactJS in 2011 for the newsfeed section, but it was released to the public in May 2013.

React Native

React Native is an open-source JavaScript framework used for developing a mobile application for iOS Android, and Windows. It uses only JavaScript to build a cross-platform mobile app. React Native is same as React, but it uses native components instead of using web components as building blocks. It targets mobile platforms rather than the browser.

Facebook develops the React Native in 2013 for its internal project Hackathon. In March 2015, Facebook announced that React Native is open and available on GitHub.

SN	ReactJS	React Native
1.	The ReactJS initial release was in 2013.	The React Native initial release was in 2015.
2.	It is used for developing web applications.	It is used for developing mobile applications.
3.	It can be executed on all platforms.	It is not platform independent. It takes more effort to be executed on all platforms.
4.	It uses a JavaScript library and CSS for animations.	It comes with built-in animation libraries.
5.	It uses React-router for navigating web pages.	It has built-in Navigator library for navigating mobile applications.
6.	It uses HTML tags.	It does not use HTML tags.

7.	It can use code components, which saves a lot of valuable time.	It can reuse React Native UI components & modules which allow hybrid apps to render natively.
8.	It provides high security.	It provides low security in comparison to ReactJS.
9.	In this, the Virtual DOM renders the browser code.	In this, Native uses its API to render code for mobile applications.

12 How is React different from Angular?

React.js: React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It's 'V' in MVC. ReactJS is an open-source, component-based front-end library responsible only for the view layer of the application. It is maintained by Facebook. React uses a declarative paradigm that makes it easier to reason about your application and aims to be both efficient and flexible. It designs simple views for each state in your application, and React will efficiently update and render just the right component when your data changes. The declarative view makes your code more predictable and easier to debug.

Features of React js:

- **Scalability:** It is reasonable for enormous scale applications because of its adaptable structure and scalability.
- **Rich JavaScript Library:** Developers from everywhere throughout the world are placing in an exertion to include significantly more features.
- **Code Reusability:** It enables the developers to reuse the code components of different levels while working on the project.
-

Angular: It is a popular open-source JavaScript framework created by Google for developing web applications. Front-end developers use frameworks like Angular or React for presenting and manipulating data efficiently. Updated Angular is much more efficient compared to the older version of Angular, especially, the core functionality was moved to different modules. That's why it becomes so much fast and smooth compare to the older one. Newly added angular CLI. With that package, you can create a scaffolding for your Angular project.

Features of Angular:

- **Accessibility:** It provides easier access as it holds ARIA-enabled components.

- **Templates:** Angular holds several templates to create UI views in a much faster manner by providing and suggesting some syntax.
- **Testing:** The program here is broken into several chunks which makes it easier to test. Testing is performed by the protractor in this case.

- **Difference between React.js & Angular:**
- Though both react js and angular seems similar, but still there is significant contrast that differentiates both React.js and Angular, which is given below.

Field	*React.js*	*Angular*
Used as	React.js is a JavaScript library. As it indicates react js updates only the virtual DOM is present and the data flow is always in a single direction.	Angular is a framework. Angular updates the Real DOM and the data flow is ensured in the architecture in both directions.
Architecture	React.js is more simplified as it follows MVC ie., Model View Control. This like angular includes features such as navigation but this can be achieved only with certain libraries like Redux and Flux. Needs more configuration and integration.	The architecture of angular on the other hand is a bit complex as it follows MVVM models ie., Model View-ViewModel. This includes lots of tools and other features required for navigation, routing, and various other functionalities.
Performance	React.js holds JSX hence the usage of HTML codes and syntax is enabled. But this doesn't make react js a subset of HTML. This is purely JavaScript-based.	Angular, on the other, is a mere subset of HTML.
Preference	React.js is preferred when the dynamic content needed is intensive. As react js holds more straightforward programming and since it is reliable many apps such as Instagram, Facebook, and Twitter still prefer to react js over angular.	Angular is platform-independent and hence is compatible to work in any platform. Hence, the HTML app which is compatible with all the browsers can prefer angular. One major app which uses angular is YouTube.

Field	*React.js*	*Angular*
Written	React.js written in JavaScript.	Written in Microsoft's Typescript language, which is a superset of ECMAScript 6 (ES6).
Dependency Injection	React.js Does not use the Dependency Injection concept.	Angular Hierarchical Dependency Injection system used.

- **Note:** Angular is a great framework it has many improvements in terms of ReactJS, it is good at bigger applications. If you are a beginner or have less coding practice also if you want stability for your project you can go with ReactJS.

13 What are the components of Redux?

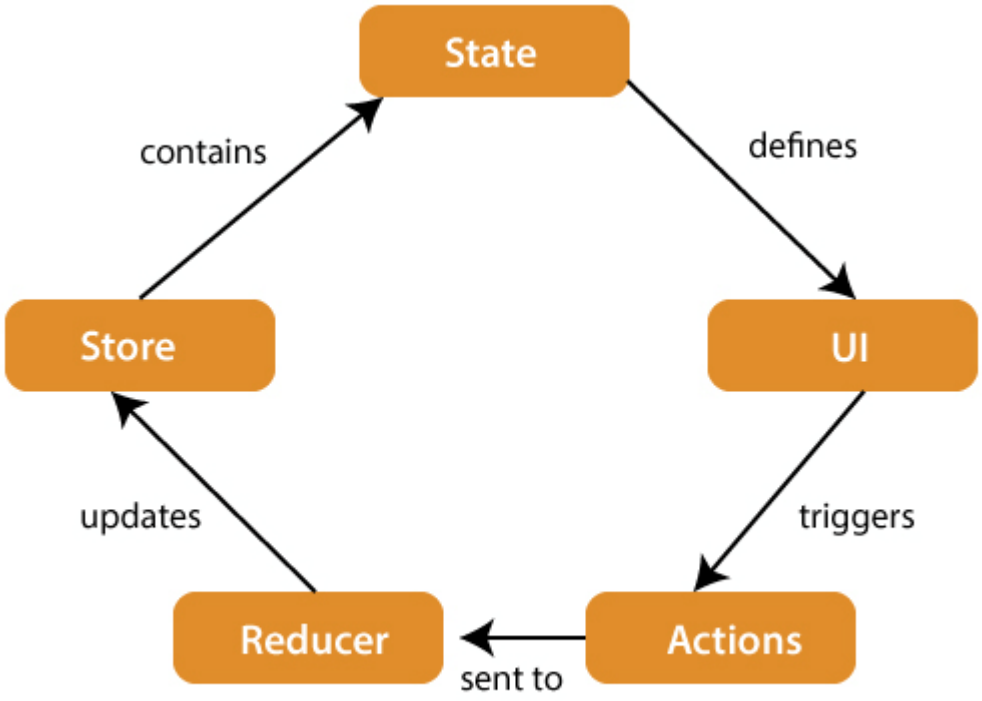
Redux is an open-source JavaScript library used to manage application state. React uses Redux for building the user interface. It was first introduced by **Dan Abramov** and **Andrew Clark** in **2015**.

The components of Redux architecture are explained below.

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(action) function. It is like a brain responsible for all moving parts in Redux.

ACTION: Action is sent or dispatched from the view which are payloads that can be read by Reducers. It is a pure object created to store the information of the user's event. It includes information such as type of action, time of occurrence, location of occurrence, its coordinates, and which state it aims to change.

REDUCER: Reducer read the payloads from the actions and then updates the store via the state accordingly. It is a pure function to return a new state from the initial state.



Redux architecture:

14 What are pure components?

Generally, In **ReactJS**, we use **shouldComponentUpdate()** Lifecycle method to customize the default behavior and implement it when the React component should re-render or update itself.

Now, **ReactJS** has provided us a **Pure Component**. If we extend a class with **Pure Component**, there is no need for **shouldComponentUpdate()** Lifecycle Method. **ReactJS Pure Component** Class compares current state and props with new props and states to decide whether the React component should re-render itself or Not.
In simple words, If the previous value of state or props and the new value of state or props is the same, the component will not re-render itself.
Since **Pure Components** restricts the re-rendering when there is no use of re-rendering of the component. Pure Components are Class Components which extends **React.PureComponent**.

15 How would you create a form in React?

We can use the `useState` Hook to keep track of each inputs value and provide a "single source of truth" for the entire application.

Example:

Use the `useState` Hook to manage the input:

```
import { useState } from 'react';
import ReactDOM from 'react-dom/client';

function MyForm() {
  const [name, setName] = useState("");

  return (
    <form>
      <label>Enter your name:
        <input
          type="text"
          value={name}
          onChange={(e) => setName(e.target.value)}
        />
      </label>
    </form>
  )
}

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

Submitting Forms

You can control the submit action by adding an event handler in the `onSubmit` attribute for the `<form>`:

Example:

Add a submit button and an event handler in the `onSubmit` attribute:

```
import { useState } from 'react';
import ReactDOM from 'react-dom/client';

function MyForm() {
  const [name, setName] = useState("");

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`The name you entered was: ${name}`)
  }

  return (
    <form onSubmit={handleSubmit}>
      <label>Enter your name:
        <input
          type="text"
          value={name}
          onChange={(e) => setName(e.target.value)}
        />
      </label>
      <input type="submit" />
    </form>
  )
}
```

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

root.render(<MyForm />);
```

Multiple Input Fields

You can control the values of more than one input field by adding a `name` attribute to each element.

We will initialize our state with an empty object.

To access the fields in the event handler use the `event.target.name` and `event.target.value` syntax.

To update the state, use square brackets [bracket notation] around the property name.

Example:

Write a form with two input fields:

```
import { useState } from 'react';
import ReactDOM from 'react-dom/client';


function MyForm() {
  const [inputs, setInputs] = useState({});


  const handleChange = (event) => {
    const name = event.target.name;
    const value = event.target.value;
    setInputs(values => ({...values, [name]: value}))
  }


  const handleSubmit = (event) => {
    event.preventDefault();
    alert(inputs);
```

```
  }

  return (
    <form onSubmit={handleSubmit}>
      <label>Enter your name:
      <input
        type="text"
        name="username"
        value={inputs.username || ""}
        onChange={handleChange}
      />
      </label>
      <label>Enter your age:
        <input
          type="number"
          name="age"
          value={inputs.age || ""}
          onChange={handleChange}
        />
        </label>
        <input type="submit" />
    </form>
  )
}

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

16 What are the advantages of using Redux?.

**********repeated**********

17 What is Redux?.

Redux is an open-source JavaScript library used to manage application state. React uses Redux for building the user interface. It was first introduced by **Dan Abramov** and **Andrew Clark** in **2015**.

React Redux is the official React binding for Redux. It allows React components to read data from a Redux Store, and dispatch **Actions** to the **Store** to update data. Redux helps apps to scale by providing a sensible way to manage state through a unidirectional data flow model. React Redux is conceptually simple. It subscribes to the Redux store, checks to see if the data which your component wants have changed, and re-renders your component.

Redux was inspired by Flux. Redux studied the Flux architecture and omitted unnecessary complexity.

- Redux does not have Dispatcher concept.
- Redux has an only Store whereas Flux has many Stores.
- The Action objects will be received and handled directly by Store.

18.How are forms created in React?

**********repeated**********

19 What are the three principles that Redux follows?

In order to understand the core of redux, you only need to know three principles.

1. **Redux is a Single Source of Truth:** The global state of an app is stored within an object tree in a single store. As a result, it becomes easier to build universal apps due to the server's state being serialized into the client without any extra codes. A single state tree, in turn, makes it easy to inspect an app. Besides this, it also enables a faster and shorter development cycle. Furthermore, some of the functionalities that have been traditionally tough to implement, such as Undo/Redo, can become trivial for implementation when the state is in a single tree.
2. **The State is Read-only State:** There is only one way for changing the state–emit an action or an object that describes what happened. As per the second principle, neither the network nor the views callbacks would ever write to the state. Instead of it, these express

intent for the transformation of the form. Since all of these changes are centralized and these can happen only in a strict order, there are no conditions to look for. Since actions are plain objects, these can be serialized, logged, stored, and then replayed to debug or test.

3. **The Modifications are Done with Pure Functions:** In order to specify how can the state tree be transformed by actions, you can write pure reducers. The reducers are merely pure functions, which take the previous state as well as action and move it to the next state. You should remember that you should return to new state objects other than mutating to the last state. For first, you should start with one reducer. Now, while your application grows, you can split it off into small reducers, which can handle specific parts of the state tree. Since reducers are merely functions, you should control the order, wherein the order can turn into reusable reducers for the common tasks.

20 What do you understand by “Single source of truth”?

Redux is a Single Source of Truth: The global state of an app is stored within an object tree in a single store. As a result, it becomes easier to build universal apps due to the server’s state being serialized into the client without any extra codes. A single state tree, in turn, makes it easy to inspect an app. Besides this, it also enables a faster and shorter development cycle. Furthermore, some of the functionalities that have been traditionally tough to implement, such as Undo/Redo, can become trivial for implementation when the state is in a single tree.