

IP TT2

MODULE 4: 5 marks

---(Aman, Ravi)

Q.1. Explain hooks in React Js with example.

Ans. 1. Hooks are the new feature introduced in the React 16.8. It allows you to use state and other React features without writing a class.

2. Before hooks, you mostly had to use class components to use state or lifecycle methods, but hooks allow you to do this within functional components.
3. Hooks promote a more functional and modular approach to component development.
4. Hooks were introduced in the 16.8 version of React.

Rules of Hooks

Hooks are similar to JavaScript functions, but you need to follow these two rules when using them. Hooks rule ensures that all the stateful logic in a component is visible in its source code. These rules are:

1. Only call Hooks at the top level

Do not call Hooks inside loops, conditions, or nested functions. Hooks should always be used at the top level of the React functions. This rule ensures that Hooks are called in the same order each time a components renders.

2. Only call Hooks from React functions

You cannot call Hooks from regular JavaScript functions. Instead, you can call Hooks from React function components. Hooks can also be called from custom Hooks.

Common Hooks:

1. **useState:** Manages state within functional components.
2. **useEffect:** Performs side effects like data fetching or DOM manipulation.
3. **useContext:** Provides access to context values without drilling props.
4. **useReducer:** Manages complex state with a reducer function.
5. **useRef:** Creates a mutable ref object.
6. **useCallback:** Memoizes a callback function.
7. **useMemo:** Memoizes a value.

This example demonstrate the use of [react useState hook](#) in the application.

```
// Filename - index.js

import React, { useState } from "react";
import ReactDOM from "react-dom/client";
function App() {
  const [click, setClick] = useState(0);

  // Using array destructuring here
  // to assign initial value 0
  // to click and a reference to the function
  // that updates click to setClick
  return (
    <div>
      <p>You clicked {click} times</p>
      <button onClick={() => setClick(click + 1)}>
        Click me
      </button>
    </div>
  );
}

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

Output:

You clicked 0 times



Q.2. Write short note on React Ref.

Ans. 1. Refs are a function provided by React to access the DOM element and the React elements created in components.

2. They are used in cases where we want to change the value of a child component, without making use of props and state.

3. They allow us to interact with these elements outside the typical rendering workflow of React.

Creating refs in React:

ReactJS Refs can be created using [React.createRef\(\)](#) function and attached to a React element via the ref attribute. Use the [useRef](#) hook to create a ref object.

Assign the ref object to the [ref](#) attribute of a DOM element or React component.

Accessing Refs:

Access the current value of the ref object using the [current](#) property.

Example : In this example, we use refs to add a callback function indirectly with the help of the **update function** and [onChange event handler](#).

```
// using refs
class App extends React.Component {
  constructor() {
    super();
    this.state = { sayings: "" };
  }
  update(e) {
    this.setState({ sayings: this.refs.anything.value });
  }
  render() {
    return (
      <div>
        Mukul Says <input type="text" ref="anything"
          onChange={this.update.bind(this)} />
        <br />
        <em>{this.state.sayings}</em>
      </div>
    );
  }
}
ReactDOM.render(< App />, document.getElementById('root'));
```

Output:

Mukul Says
using ref :)

Q.3. Explain Flux architecture in detail.

Ans. 1. Flux is a unidirectional data flow architecture pattern for building user interfaces.

2. It was introduced by Facebook to manage complex state in large-scale React applications.

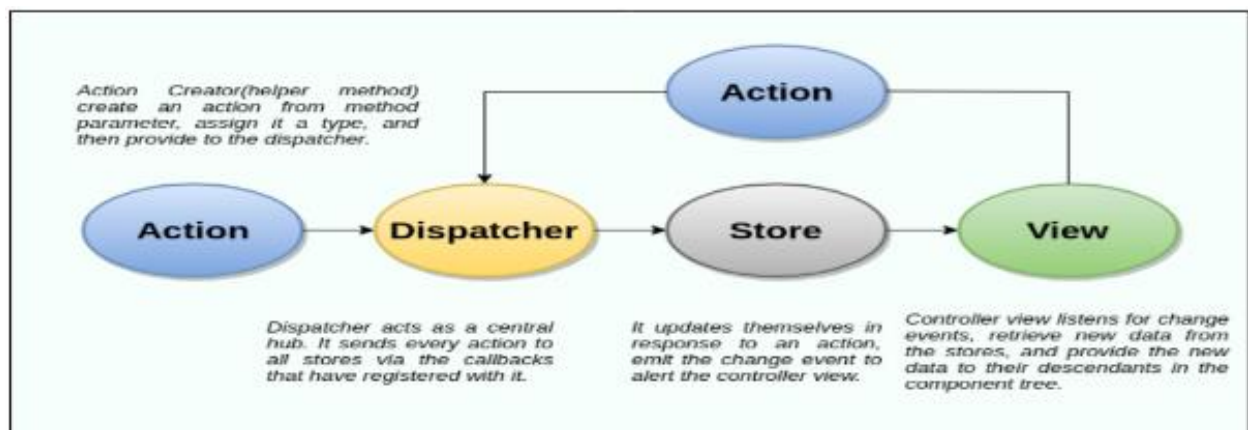
3. It is not a framework or a library.

4. It is useful when the project has dynamic data, and we need to keep the data updated in an effective manner.

5. It reduces the runtime errors.

Working of flux architecture:

- 1. User Interaction:** A user interacts with a view (e.g., clicking a button).
- 2. Action Creation:** The view dispatches an action object to the dispatcher.
- 3. Action Propagation:** The dispatcher broadcasts the action to all registered stores.
- 4. Store Updates:** Stores that are interested in the action update their state based on the action's payload.
- 5. View Rendering:** Views re-render themselves to reflect the updated state.



Benefits of Flux

- **Predictable Data Flow:** The unidirectional flow makes it easier to understand how data changes in the application.
- **Scalability:** Flux can handle complex applications with many components and state changes.
- **Reusability:** Stores can be reused across different parts of the application.
- **Testability:** The separation of concerns between actions, stores, and views makes it easier to write unit tests.

MODULE 6: 5 marks

Q.1. Explain REST API, Explain the principle on REST API.

Ans. 1. REST stands for REpresentational State Transfer. It is an architectural style that defines a set of rules in order to create Web Services.

2. In a client-server communication, REST suggests to create an object of the data requested by the client and send the values of the object in response to the user.

3. For example, if the user is requesting for a movie in Bangalore at a certain place and time, then you can create an object on the server-side.



Principles of REST:

1. Client-server – The client-server architecture enables a uniform interface and separates clients from the servers. This enhances the portability across multiple platforms as well as the scalability of the server components.

2. Stateless – The requests sent from a client to a server will contain all the required information to make the server understand the requests sent from the client. This can be either a part of URL, query-string parameters, body, or even headers.

3. Cacheable – Cache constraints require that the data within a response to a request be implicitly or explicitly labelled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.

4. Uniform interface – To obtain the uniformity throughout the application, REST has the following four interface constraints:

- Resource identification
- Resource Manipulation using representations
- Self-descriptive messages
- Hypermedia as the engine of application state

5. Layered system – The layered system style allows an architecture to be composed of hierarchical layers by constraining component behaviour such that each component cannot “see” beyond the immediate layer with which they are interacting.

6. Code on demand (optional) – REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts.

Q.2. Explain Routing in Express Js with example.

Ans. 1. Routing in Express.js is the process of mapping HTTP requests to specific functions that handle those requests. This allows you to create different routes for different actions within your web application.

2. Routing in Express is like showing your web app where to go. It's about deciding what your app should do when users go to various URLs. You get to set the actions for things like going to the homepage, submitting forms, or clicking links.

3. Express makes it simple by letting you create rules that connect to specific parts of your code.

Steps to Implement Routing in Express:

Step 1: Initialising the Node App using the below command:

npm init -y

Step 2: Installing express in the app:

npm install express

Here's a simple example of how to create a basic route in Express.js:

JavaScript

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Hello, world!');
});

app.listen(3000, () => {
  console.log('Server listening on port 3000');
});
```

In this example:

- The app.get('/') line defines a route for GET requests to the root path (/).
- When a user visits <http://localhost:3000>, the callback function is executed, and the response "Hello, world!" is sent to the client.

MODULE 5: 2 marks

Q.1. Write short note on Asynchronous.

Ans. Asynchronous programming is a core feature of Node.js, enabling it to handle multiple operations concurrently without blocking the execution of other tasks. This is crucial for building efficient and scalable applications.

Key Concepts:

- 1. Non-Blocking I/O:** Node.js handles multiple requests without waiting for previous operations to finish, enhancing efficiency.
- 2. Event-Driven Architecture:** Actions like file reads or network requests, trigger events, allowing the server to manage other tasks simultaneously.
- 3. Callbacks:** Functions passed to other functions, executed after the asynchronous task completes.
- 4. Promises:** Objects that manage asynchronous operations, enabling easier error handling and task chaining.
- 5. Async/Await:** Syntax to write asynchronous code more clearly, improving readability and maintainability.

Q.2. State the features of node js.

Ans. Features of node js are as follows:-

- **Asynchronous and Event Driven** – All Node.js APIs are non-blocking, meaning the server doesn't wait for an API to return data. It moves to the next task and gets notified via events when the previous task completes.
- **Very Fast** – Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
- **Single Threaded but Highly Scalable** – Node.js uses a single-threaded model with event looping, allowing it to handle many requests efficiently compared to traditional servers like Apache that create limited threads.
- **No Buffering** – Node.js applications never buffer any data. These applications simply output the data in chunks.

Q.3. Write syntax to write the data from the node buffer with example.

Ans. // Syntax

```
buffer.write(string, offset, length, encoding);
```

// Example

```
const buffer = Buffer.alloc(256);
const length = buffer.write('Hello, world!', 0, 'utf8');
console.log(`Written ${length} bytes: ${buffer.toString('utf8', 0, length)}`);
```

Q.4. Write syntax to read the data from the node buffer with example.

Ans. // Syntax

```
buffer.toString(encoding, start, end);
```

// Example

```
const buffer = Buffer.from('Hello, world!', 'utf8');  
const data = buffer.toString('utf8', 0, buffer.length);  
console.log(`Read data: ${data}`);
```

Q.5. Explain blocking and non-blocking in detail.

Ans. 1. Blocking operations stop the execution of further code until the current operation completes. The program waits for the operation to finish before moving on.

Example: In a blocking I/O operation, if a program reads the data from a file, it will pause and wait until the entire file is read before proceeding to the next line of code.

2. Non-blocking operations allow the execution of the further code without waiting for the current operation to finish. The program can handle other tasks concurrently while the operation completes in the background.

Example: In a non-blocking I/O operation, the program can request data from a file and immediately move on to execute other tasks. Once the data is ready, the program gets notified through a callback or an event.

Q.6. Explain NPM.

Ans. NPM is a short form of “Node Package Manager”, which is the world's largest software registry. NPM is a crucial tool in the Node.js ecosystem, designed to manage packages and dependencies for JavaScript applications.

NPM mainly consists of three different components, these are:

Website: The NPM official website is used to find packages for your project, create and set up profiles to manage and access private and public packages.

Command Line Interface (CLI): The CLI runs from your computer's terminal to interact with NPM packages and repositories.

Registry: The registry is a large and public database of JavaScript projects and meta-information. You can use any supported NPM registry that you want or even your own. You can even use someone else's registry as per their terms of use.

Q.7. Explain REPL in detail.

Ans. REPL stands for Read-Eval-Print-Loop. It is a quick and easy way to test simple Node.js/JavaScript code.

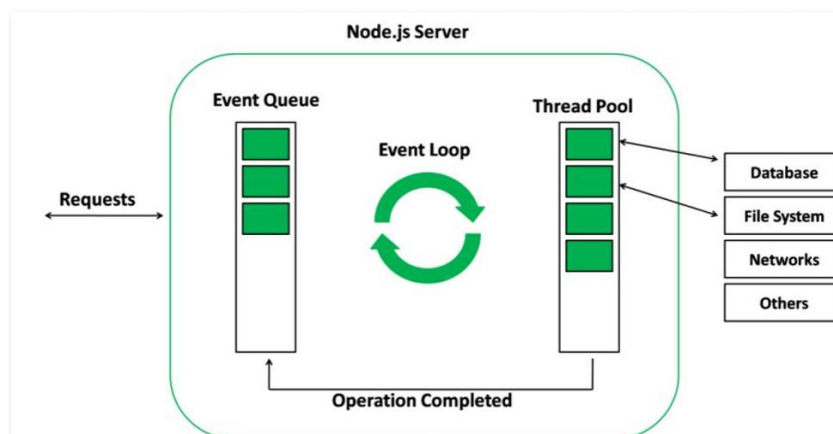
A REPL has the following:

- A read function, which accepts an expression from the user and parses it into a data structure in memory.
- An eval function, which takes the data structure and evaluates the expression.
- A print function, which prints the result.
- A loop function, which runs the three commands above until termination

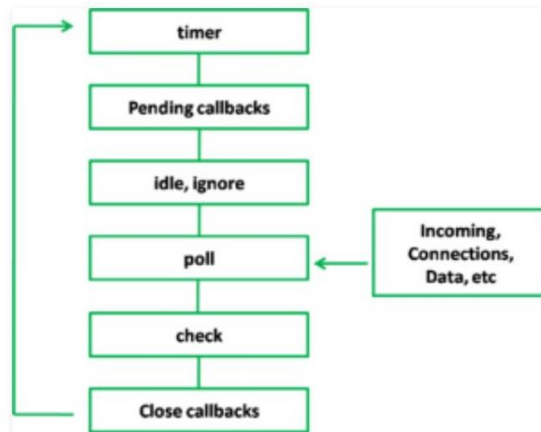
Q.8. Draw and explain event loop operation in node js.

Ans. The Node.js event loop allows Node.js to perform non-blocking I/O operations by offloading operations to the system kernel whenever possible.

The following diagram is a proper representation of the event loop in a Node.js server:



Phases of the Event loop: The following diagram shows a simplified overview of the event loop order of operations:



Here's how it operates:

- **Timers:** Callbacks scheduled by `setTimeout()` or `setInterval()` are executed in this phase.
- **Pending Callbacks:** I/O callbacks deferred to the next loop iteration are executed here.
- **Idle, Prepare:** Used internally only.
- **Poll:** Retrieves new I/O events.
- **Check:** It invokes `setImmediate()` callbacks.
- **Close Callbacks:** It handles some close callbacks. Eg: `socket.on('close', ...)`

Q.9. Write a node js program to read a file with the help of syntax.

Ans. // Import the 'fs' module

```
const fs = require('fs');
```

// Read the file asynchronously

```
fs.readFile('example.txt', 'utf8', (err, data) => {  
  if (err) {  
    console.error('Error reading the file:', err);  
    return;  
  }  
  console.log('File content:', data);  
});
```

Q.10. Write a node js program to delete a file with the help of syntax.

Ans. // Import the 'fs' module

```
const fs = require('fs');
```

// Delete the file asynchronously

```
fs.unlink('example.txt', (err) => {
```

```
if (err) {  
  console.error('Error deleting the file:', err);  
  return;  
}  
console.log('File deleted successfully');  
});
```

Q.11. List and explain most common and used file in node js.

Ans. 1) package.json: Contains metadata about the project, including dependencies, scripts, and version control. Key for project configuration.

2) server.js or app.js: Main entry point where the server is configured and started. Handles routing and middleware setup.

3) .env: Stores environment variables, such as API keys and database URIs, keeping sensitive information separate from the codebase.

4) README.md: Provides documentation, setup instructions, and usage guidelines, essential for understanding and using the project.


5) routes/ Directory: Contains route files that define the endpoints of the application, keeping routing logic organized and modular.

Q.12. Explain different types of strings event in node js.

Ans.

1. `charAt(index)` : Returns the character at the specified index.


Javascript

 Copy

```
const str = 'Hello';  
console.log(str.charAt(0)); // Output: 'H'
```

2. `concat(string2, ... , stringN)` : Joins multiple strings into one.


Javascript

 Copy

```
const str1 = 'Hello';  
const str2 = 'World';  
console.log(str1.concat(' ', str2)); // Output: 'Hello World'
```

3. `includes(searchValue, start)` : Checks if the string contains a specified value.


Javascript

 Copy

```
const str = 'Hello World';  
console.log(str.includes('World')); // Output: true
```

4. `slice(beginIndex, endIndex)` : Extracts a section of a string.


Javascript

 Copy

```
const str = 'Hello World';  
console.log(str.slice(0, 5)); // Output: 'Hello'
```

5. `toUpperCase()` : Converts the string to upper case.

Javascript

 Copy

```
const str = 'Hello World';  
console.log(str.toUpperCase()); // Output: 'HELLO WORLD'
```