**1. What is React.js? Explain the Features of React.js**

**React** is a powerful JavaScript library for building dynamic and interactive user interfaces (UIs). It is developed by Facebook. React is known for its **component-based architecture**which allows you to create reusable UI elements, making complex web applications easier to manage and maintain. React is used to build single-page applications.

## Why Learn React JS?

React, the popular JavaScript library, offers several exciting reasons for developers to learn it.

First, React is flexible – once you learn its concepts, you can use it across various platforms to build quality user interfaces. Unlike a framework, React’s library approach allows it to evolve into a remarkable tool.

Second, React has a great developer experience, making it easier to understand and write code. Third, it benefits from Facebook’s support and resources, ensuring regular bug fixes, enhancements, and documentation updates. Additionally, React’s broader community support, excellent performance, and ease of testing make it an ideal choice for web development.

**Advantage of JS**

* Easy to Learn and USe.
* Creating Dynamic Web Applications Becomes Easier.
* Reusable Components.
* Performance Enhancement.
* The Support of Handy Tools.
* Known to be SEO Friendly. (**Search Engine Optimization** )
* The Benefit of Having JavaScript Library.
* Scope for Testing the Codes.

**Features of React.js:**

* **JSX (JavaScript XML):** A syntax extension for JavaScript that looks similar to XML or HTML and is used with React to describe what the UI should look like.
* **Components:** Building blocks of a React application. They can be functional or class-based.
* **Virtual DOM:** A lightweight copy of the actual DOM. It helps in efficient updating and rendering of the UI.
* **One-Way Data Binding:** Data flows in one direction, making the application more predictable and easier to debug.
* **Lifecycle Methods:** Special methods that allow developers to run code at specific points in a component’s lifecycle.
* **Hooks:** Functions that let you use state and other React features in functional components.

**2. What is the difference between Actual DOM and Virtual DOM?**

* **Real DOM:** The real DOM is an interface for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. Updating the real DOM is slow because it involves re-rendering the entire page.
* **Virtual DOM:** A representation of the real DOM kept in memory and synced with the real DOM by libraries such as ReactDOM. When the state of an object changes, the Virtual DOM updates only that object in the real DOM, making the process faster.

| **Real DOM** | **Virtual DOM** |
| --- | --- |
| Real DOM represent actual structure of the webpage. | Virtual DOM represent the virtual/memory representation of the Webpage. |
| DOM manipulation is very expensive | DOM manipulation is very easy |
| There is too much memory wastage | No memory wastage |
| It updates Slow | It updates fast |
| It can directly update HTML | It can’t update HTML directly |
| Creates a new DOM if the element updates. | Update the JSX if the element update |
| It allows us to directly target any specific node (HTML element) | It can produce about 200,000 Virtual DOM Nodes / Second. |
| It represents the Ul of your application | It is only a virtual representation of the DOM |

**3. What is a component in React?**

React divides the web page into multiple [components](https://www.geeksforgeeks.org/reactjs-components/) as it is component-based. Each component is a part of the UI design which has its own logic and design as shown in the below image. So the component logic which is written in JavaScript makes it easy and run faster and can be reusable.

A **component** in React is a reusable piece of the user interface. Components can be functional (stateless) or class-based (stateful).

Stateful component :- Stateful component is manage their own state and in React “state” refers to an object that determine how a component should behave and render.

Stateless component :- stateless component are known as functional component . It does not hold or maintain their local state . They simply get data form props and render the UI . By with the introduction of react hooks , functional components have become more powerful , allowing them to use state and other react features .

**4. How many types of components in React? Explain class component and functional component.**

* **Class Component:** Defined using ES6 classes and can maintain and manage their own state. They have lifecycle methods.

javascript

Copy code

class Welcome extends React.Component {

render() {

return <h1>Hello, {this.props.name}</h1>;

}

}

* **Functional Component:** Defined using functions and are stateless. With React Hooks, they can now manage state and side effects.

javascript

Copy code

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

**Functional Components vs Class Components:**

|  |  |
| --- | --- |
| [**Functional Components**](https://www.geeksforgeeks.org/reactjs-functional-components/) | [**Class Components**](https://www.geeksforgeeks.org/reactjs-class-based-components/) |
| A functional component is just a plain JavaScript pure function that accepts props as an argument and returns a React element(JSX). | A class component requires you to extend from React. Component and create a render function that returns a React element. |
| There is no render method used in functional components. | It must have the render() method returning JSX (which is syntactically similar to HTML) |
| Functional components run from top to bottom and once the function is returned it can’t be kept alive. | The class component is instantiated and different life cycle method is kept alive and is run and invoked depending on the phase of the class component. |
| Also known as Stateless components as they simply accept data and display them in some form, they are mainly responsible for rendering UI. | Also known as Stateful components because they implement logic and state. |
| React lifecycle methods (for example, componentDidMount) cannot be used in functional components. | React lifecycle methods can be used inside class components (for example, componentDidMount). |
| Hooks can be easily used in functional components to make them Stateful.  Example:  const [name,SetName]= React.useState(' ') | It requires different syntax inside a class component to implement hooks.  Example:  constructor(props) {    super(props);    this.state = {name: ' '} } |
| Constructors are not used. | Constructor is used as it needs to store state. |

**5. Explain the life cycle of a class component (Mounting, Unmounting, Updating).**

* **Mounting:** When a component is being created and inserted into the DOM.
  + constructor()
  + static getDerivedStateFromProps()
  + render()
  + componentDidMount()
* **Updating:** When a component is being re-rendered as a result of changes to props or state.
  + static getDerivedStateFromProps()
  + shouldComponentUpdate()
  + render()
  + getSnapshotBeforeUpdate()
  + componentDidUpdate()
* **Unmounting:** When a component is being removed from the DOM.
  + componentWillUnmount()

**6. How to implement life cycle phase (Mounting, Updating, Unmounting) using functional component.**

In functional components, lifecycle methods are replaced by **React Hooks**.

* **Mounting:** Use useEffect() with an empty dependency array.

javascript

Copy code

useEffect(() => {

// Code to run on mount

}, []);

* **Updating:** Use useEffect() with dependencies.

javascript

Copy code

useEffect(() => {

// Code to run on update

}, [dependencies]);

* **Unmounting:** Return a cleanup function from useEffect().

useEffect(() => {

return () => {

console.log(‘unMounting page’)

};

}, []);

**7. What do you mean by stateful and stateless component?**

* **Stateful Component:** Manages and maintains its own state.
* **Stateless Component:** Does not manage its own state, often referred to as presentational components.

**8. What is the difference between state and props?**

* **State:** Managed within the component and can change over time.
* **Props:** Read-only properties passed from parent components to child components.

**9. What is the difference between an element and components?**

* **Element:** The smallest building block of React apps, representing what you see on the screen.
* **Component:** A larger building block that can contain elements and other components.

**10. What happens when we call setState method?**

When setState is called, React schedules an update to the component's state and re-renders the component along with its children.

**11. Can we re-render component without using setState?**

Yes, by passing new props to a component or using forceUpdate() (though the latter is not recommended).

**12. What is eager loading and lazy loading in React?**

* **Eager Loading:** Loading all the components and resources upfront.
* **Lazy Loading:** Deferring the loading of components and resources until they are needed.

**13. How we load the components eagerly and lazily?**

* **Eager Loading:** Directly import the components.

javascript

Copy code

import MyComponent from './MyComponent';

* **Lazy Loading:** Use React.lazy and Suspense.

javascript

Copy code

Const MyComponent = React.lazy(() => import('./MyComponent'));

function App() {

return (

<React.Suspense fallback={<div>Loading...</div>}>

<MyComponent />

</React.Suspense>

);

}

**14. What is error boundaries in React?**

**Error Boundaries** are React components that catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI instead of the component tree that crashed.

Error boundaries are a feature in React designed to handle errors gracefully in a component tree. They are special components that catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI instead of crashing the entire application. Error boundaries catch errors during rendering, in lifecycle methods, and in constructors of the whole tree below them.

**Key Points**

* Error boundaries are implemented using class components.
* They do not catch errors in event handlers, asynchronous code (e.g., setTimeout), or errors thrown in the error boundary itself.

**Implementing an Error Boundary**

To create an error boundary, you need to define a class component that implements either or both of the lifecycle methods static getDerivedStateFromError(error) and componentDidCatch(error, info).

* **getDerivedStateFromError**: Used to update the state so the next render will show a fallback UI.
* **componentDidCatch**: Used to log the error information.

**Example**

Here is an example of how to implement and use an error boundary:

**ErrorBoundary Component:**

jsx

Copy code

import React, { Component } from 'react';

class ErrorBoundary extends Component {

constructor(props) {

super(props);

this.state = { hasError: false };

}

static getDerivedStateFromError(error) {

// Update state so the next render shows the fallback UI

return { hasError: true };

}

componentDidCatch(error, errorInfo) {

// You can also log the error to an error reporting service

console.log("Error caught by ErrorBoundary: ", error, errorInfo);

}

render() {

if (this.state.hasError) {

// You can render any custom fallback UI

return <h1>Something went wrong.</h1>;

}

return this.props.children;

}

}

export default ErrorBoundary;

**Component that may throw an error:**

jsx

Copy code

import React from 'react';

class BuggyComponent extends React.Component {

constructor(props) {

super(props);

this.state = {

throwError: false,

};

}

handleClick = () => {

this.setState({ throwError: true });

};

render() {

if (this.state.throwError) {

throw new Error('I crashed!');

}

return <button onClick={this.handleClick}>Trigger Error</button>;

}

}

export default BuggyComponent;

**Using the ErrorBoundary Component:**

jsx

Copy code

import React from 'react';

import ReactDOM from 'react-dom';

import ErrorBoundary from './ErrorBoundary';

import BuggyComponent from './BuggyComponent';

function App() {

return (

<div>

<h1>My React App</h1>

<ErrorBoundary>

<BuggyComponent />

</ErrorBoundary>

</div>

);

}

ReactDOM.render(<App />, document.getElementById('root'));

**15. How to implement routing in React? What is the use of BrowserRouter and Routes?**

* **BrowserRouter:** Wraps the entire application to enable routing.
* **Routes:** Defines the routes within the application.

javascript

Copy code

import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';

function App() {

return (

<Router>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/about" element={<About />} />

</Routes>

</Router>

);

}

Implementing routing in a React application is commonly done using the react-router-dom library. This library provides several components that help manage routing, the two most important being BrowserRouter and Routes.

### Key Concepts

* **BrowserRouter**: This component uses the HTML5 history API (pushState, replaceState, and the popstate event) to keep your UI in sync with the URL. It should wrap your entire application to provide routing functionality.
* **Routes**: This component replaces the older Switch component and is used to define a list of routes, each mapping a URL path to a component.

### Installation

First, install react-router-dom using npm or yarn:

bash

Copy code

npm install react-router-dom

**16. What is props drilling in React?**

**Props Drilling** refers to the process of passing down props through multiple layers of components, often leading to deeply nested hierarchies.

**17. What is pure component in React? How to create pure class component?**

A **PureComponent** in React is a component that only re-renders if it detects changes in state or props, performing a shallow comparison.

javascript

Copy code

import React, { PureComponent } from 'react';

class MyPureComponent extends PureComponent {

render() {

return <div>{this.props.value}</div>;

}

}

**18. How to use function component as pure component?**

Use React.memo to create a pure functional component.

javascript

Copy code

Const MyPureComponent = React.memo(function MyPureComponent(props) {

return <div>{props.value}</div>;

});

**19. What is a Higher Order Component (HOC)?**

An **HOC** is a function that takes a component and returns a new component with additional props or functionality.

javascript

Copy code

function withLogging(WrappedComponent) {

return function (props) {

console.log('Component rendered with props:', props);

return <WrappedComponent {...props} />;

};

}

**20. What is a protected(Authorization)/private(Authentication) route in React.js?**

A **Protected Route** is a route that requires a user to be authenticated before accessing it.

javascript

Copy code

import { Navigate } from 'react-router-dom';

function ProtectedRoute({ children, isAuthenticated }) {

return isAuthenticated ? children : <Navigate to="/login" />;

}

**21. What are React Fragments?**

React Fragments allow you to group multiple elements without adding an extra node to the DOM. This is useful when you want to return multiple elements from a component render method without adding an unnecessary wrapper element.

### Why Use React Fragments?

1. **Avoid Unnecessary DOM Nodes**: Adding unnecessary wrapper elements can complicate your CSS and affect the structure of your DOM, which may impact layout and styling.
2. **Cleaner JSX**: Fragments provide a cleaner way to write JSX when you need to return multiple elements.

### Using React Fragments

React Fragments can be used in two ways:

1. **Using the Fragment Component**: Explicitly importing and using React.Fragment.
2. **Using the Short Syntax**: Using the shorthand syntax (<>...</>).

### Example

#### Using React.Fragment

Here's an example of how to use React.Fragment:

jsx

Copy code

importReactfrom'react';

constFragmentExample = () => {

return (

<React.Fragment>

<h1>Title</h1>

<p>This is a paragraph.</p>

<ul>

<li>List Item 1</li>

<li>List Item 2</li>

<li>List Item 3</li>

</ul>

</React.Fragment>

);

};

exportdefaultFragmentExample;

#### Using Short Syntax

The short syntax is more concise and achieves the same result:

jsx

Copy code

importReactfrom'react';

constFragmentExample = () => {

return (

<>

<h1>Title</h1>

<p>This is a paragraph.</p>

<ul>

<li>List Item 1</li>

<li>List Item 2</li>

<li>List Item 3</li>

</ul>

</>

);

};

exportdefaultFragmentExample;

### Explanation

1. **React.Fragment Component**:
   * In the first example, React.Fragment is explicitly imported and used to wrap multiple elements. This ensures that no additional DOM node is created, and the elements are grouped together logically.
2. **Short Syntax (<>...</>)**:
   * In the second example, the shorthand syntax (<>...</>) is used to achieve the same result. It provides a cleaner and more concise way to group elements without adding extra nodes to the DOM.

### Key Points

* **No Extra Nodes**: Both React.Fragment and the short syntax do not create additional DOM nodes.
* **Keys with Fragments**: When rendering a list of elements using fragments, you can provide a key attribute to the fragments, just like you would with other elements. This is especially useful for lists where keys are necessary.

jsx

Copy code

importReactfrom'react';

constListWithFragments = ({ items }) => {

return (

<>

{items.map((item, index) => (

<React.Fragmentkey={index}>

<dt>{item.term}</dt>

<dd>{item.description}</dd>

</React.Fragment>

))}

</>

);

};

exportdefaultListWithFragments;

In this example, each term-description pair is wrapped in a React.Fragment with a key attribute to ensure each fragment is uniquely identified.

**22. What are the hooks in React.js? Explain some hooks used in your project.**

**Hooks** are functions that allow you use state and lifecycle features in functional components.

**Hooks** are functions that allow you to manage state and lifecycle and other features in functional component in React.

* useState: Manages state.
* useEffect: Handles side effects.
* useContext: Consumes context values.
* useReducer: Manages more complex state logic.

**23. Explain following hooks.**

* **useState:** Declares state variables in functional components.

javascript

Copy code

const [count, setCount] = useState(0);

* **useLocation:** Returns the current location object.

javascript

Copy code

const location = useLocation();

* **useNavigate:** Returns a function to navigate programmatically.

javascript

Copy code

const navigate = useNavigate();

navigate('/home');

* **useRef:** Returns a mutable ref object.

javascript

Copy code

constmyRef = useRef(null);

* **useMemo:**Memoizes a value.

javascript

Copy code

constmemoizedValue = useMemo(() =>computeExpensiveValue(a, b), [a, b]);

* **useCallback:**Memoizes a function.

javascript

Copy code

constmemoizedCallback = useCallback(() => {

doSomething(a, b);

}, [a, b]);

* **useDispatch:** Returns the dispatch function from the Redux store.

In React.js, we use useDispatch to send actions to the Redux store. It allows functional components to update the state through Redux, making the application's state management systematic and predictable.

javascript

Copy code

const dispatch = useDispatch();

* **useContext:** Consumes a context.

javascript

Copy code

const value = useContext(MyContext);

* **useSelector:** Selects state from the Redux store.

javascript

Copy code

const value = useSelector((state) =>state.value);

* **useEffect:** Handles side effects.

javascript

Copy code

useEffect(() => {

// Effect

return () => {

// Cleanup

};

}, [dependencies]);

**24. In which lifecycle method would you like to make an HTTP call?**

componentDidMount is ideal for making HTTP calls in class

components.

**25. Where to make an API/HTTP/network call in functional components?**

Use the useEffect hook to make API calls in functional components.

**26. Write a code using useEffect to implement mounting, updating, and unmounting in functional components.**

javascript

Copy code

import React, { useEffect } from 'react';

function MyComponent() {

useEffect(() => {

console.log('Component mounted');

return () => {

console.log('Component unmounted');

};

}, []);

useEffect(() => {

console.log('Component updated');

});

return <div>Hello World</div>;

}

**27. Can we update props?**

No, props are read-only and cannot be updated by the child component.

**28. How to update parent state data from child component?**

Pass a function from the parent to the child component and call it from the child to update the parent’s state.

javascript

Copy code

function ParentComponent() {

const [data, setData] = useState('');

return <ChildComponentupdateData={setData} />;

}

function ChildComponent({ updateData }) {

return <button onClick={() =>updateData('New Data')}>Update Data</button>;

}

**29. What is context API in React.js?**

The **Context API** allows passing data through the component tree without having to pass props down manually at every level.

**30. What is Redux? Explain the components of Redux (Store, Action, Reducer).**

**Redux** is a state management library for React apps.

* **Store:** Holds the entire state of the application.
* **Action:** A plain object that describes the intention to change the state.
* **Reducer:** A function that takes the current state and an action and returns a new state.

**31. What is the difference between useState and useReducer?**

* **useState:** Best for simple state logic.
* **useReducer:** Best for complex state logic involving multiple sub-values or when the next state depends on the previous one.

**32. What is call, bind, apply in JS?**

* **call:** Invokes a function with a specified this context and arguments.
* **apply:** Similar to call, but arguments are passed as an array.
* **bind:** Returns a new function, permanently bound to a specific this context.

**33. Difference between context API and Redux?**

* **Context API:** Suitable for small to medium applications with simpler state management.
* **Redux:** Suitable for larger applications with more complex state logic and better debugging capabilities.

**34. What is RTK?**

**Redux Toolkit (RTK)** is a set of tools that help simplify the process of writing Redux logic, reducing boilerplate.

**35. What is the use of async thunk (createAsyncThunk)?**

**createAsyncThunk** is a utility from Redux Toolkit to handle asynchronous logic in Redux.

javascript

Copy code

constfetchUserById = createAsyncThunk(

'users/fetchById',

async (userId, thunkAPI) => {

const response = await userAPI.fetchById(userId);

return response.data;

}

);

**36. What is memo in ReactJS?**

React.memo is a higher-order component that memoizes the rendered output, preventing unnecessary re-renders.

**37. What is JSX?**

**JSX** is a syntax extension for JavaScript that allows writing HTML-like code inside JavaScript files.

**38. What is reconciliation?**

**Reconciliation** is the process by which React updates the DOM with changes detected by comparing the Virtual DOM and the real DOM.

**39. What are controlled and uncontrolled components?**

* **Controlled Component:** Form inputs that are controlled by React state.
* **Uncontrolled Component:** Form inputs that are managed by the DOM itself.

**40. What does shouldComponentUpdate do and why is it important?**

shouldComponentUpdate allows you to prevent a component from re-rendering if the props or state have not changed, optimizing performance.

**41. Describe how events are handled in React.**

Events in React are handled using camelCase syntax and are passed as functions.

javascript

Copy code

<button onClick={handleClick}>Click Me</button>

**42. What is the second argument that can optionally be passed to setState and what is its purpose?**

The second argument is a callback function that is executed after setState has completed and the component has re-rendered.

**43. List down the variations of setState() in ReactJS?**

* Passing an object.
* Passing a function with previous state and props.

**44. What is props.children?**

props.children is used to pass nested elements directly into a component.

**45. Does React use HTML?**

React uses JSX, which is syntactic sugar for JavaScript and XML/HTML, but it gets transpiled to JavaScript.

**46. Tell me the most significant drawback of ReactJS.**

One significant drawback is the frequent need for additional libraries for state management, routing, and other functionalities, increasing the complexity of the setup.

**47. Explain React Router.**

**React Router** is a library used for routing in React applications, allowing navigation between different components.

**48. How to modify each request and response (Interceptor) in React.js?**

Use libraries like **Axios** which provide interceptors to modify requests and responses.

javascript

Copy code

axios.interceptors.request.use((config) => {

// Modify request config

return config;

});

**49. What is REST API? How to call API from React?**

A **REST API** is a web service that uses HTTP methods to access and manipulate data. Use fetch or Axios to call APIs in React.

javascript

Copy code

fetch('https://api.example.com/data')

.then(response =>response.json())

.then(data => console.log(data));

**50. How to send a file in a request from React?**

Use FormData to send files in a request.

javascript

Copy code

constformData = new FormData();

formData.append('file', file);

fetch('https://api.example.com/upload', {

method: 'POST',

body: formData,

});

**51. Why should we not update the state directly?**

Directly updating the state bypasses React's internal mechanisms, leading to unpredictable state changes and issues with re-rendering.

**52. What is the purpose of callback function as an argument of setState()?**

It allows executing code after the state has been updated and the component has re-rendered.

**53. How to bind method and event handlers in JSX?**

Use the bind method or arrow functions.

javascript

Copy code

<button onClick={this.handleClick.bind(this)}>Click Me</button>

<button onClick={() =>this.handleClick()}>Click Me</button>

**54. What are synthetic events in ReactJS?**

**Synthetic Events** are cross-browser wrappers around the browser's native event, providing a consistent API across different browsers.

**55. What are inline conditional expressions?**

They are expressions used directly within JSX to conditionally render elements.

{condition ? <ComponentA /> : <ComponentB />}

**56. What is key props and what is the benefit of using it in arrays of elements?**

**Key Props** help React identify which items have changed, are added, or are removed, improving the efficiency of re-rendering lists.

**57. What is the use of refs?**

Refs are used to access DOM nodes or React elements directly.

**58. How to create refs?**

Use React.createRef or useRef in functional components.

javascript

Copy code

Const myRef = React.createRef();

Const myFuncRef = useRef(null);

**59. What are forward refs?**

**Forward Refs** allow passing refs through components to access DOM nodes of child components.

javascript

Copy code

constFancyButton = React.forwardRef((props, ref) => (

<button ref={ref} className="FancyButton">

{props.children}

</button>

));

These explanations and examples should provide a comprehensive understanding of these React concepts.

**1. What is OOP?**

**OOP (Object-Oriented Programming)** is a programming paradigm that uses "objects" – data structures consisting of fields, and methods together with their interactions – to design applications and computer programs. Key concepts of OOP include:

* **Class**: A blueprint for creating objects.
* **Object**: An instance of a class.
* **Inheritance**: A mechanism where a new class inherits the properties and behavior of another class.
* **Polymorphism**: The ability to present the same interface for different underlying data types.
* **Encapsulation**: The bundling of data and the methods that operate on that data.
* **Abstraction**: Hiding the complex implementation details and showing only the necessary features of the object.

**2. What is the difference between a framework and a library?**

* **Framework**: A framework provides a structure and set of rules for developing software applications. It usually includes pre-written code that developers can use, modify, or extend. Frameworks dictate the architecture and flow of application development (e.g., React, Angular, Django).
* **Library**: A library is a collection of pre-written code that developers can call upon to perform common tasks. Libraries offer specific functionality and are used when needed without imposing a specific structure (e.g., Lodash, jQuery).

**Example**:

* **Library**: Using lodash to manipulate arrays in JavaScript.

javascript

Copy code

const \_ = require('lodash');

let arr = [1, 2, 3, 4];

let doubled = \_.map(arr, n => n \* 2);

console.log(doubled); // [2, 4, 6, 8]

* **Framework**: Using React to build a UI component.

javascript

Copy code

import React from 'react';

import ReactDOM from 'react-dom';

function App() {

return <h1>Hello, World!</h1>;

}

ReactDOM.render(<App />, document.getElementById('root'));

**3. What is Express?**

**Express** is a minimal and flexible Node.js web application framework that provides a robust set of features to develop web and mobile applications. It facilitates the management of HTTP requests and responses, middleware, and routing.

**4. Difference between Express and Node**

* **Node.js**: A runtime environment for executing JavaScript code server-side. It is built on Chrome's V8 JavaScript engine and allows the creation of scalable network applications.
* **Express.js**: A web application framework built on top of Node.js that simplifies the server creation and routing process. It provides a higher-level API to handle web requests and responses.

**Example**:

* **Node.js** without Express:

javascript

Copy code

const http = require('http');

const server = http.createServer((req, res) => {

res.statusCode = 200;

res.setHeader('Content-Type', 'text/plain');

res.end('Hello World\n');

});

server.listen(3000, () => {

console.log('Server running at http://127.0.0.1:3000/');

});

* **Node.js** with Express:

javascript

Copy code

const express = require('express');

const app = express();

app.get('/', (req, res) => {

res.send('Hello World');

});

app.listen(3000, () => {

console.log('Server running on port 3000');

});

**5. Folder structure of any framework**

The folder structure can vary between frameworks, but a common structure for a web application might look like this:

lua

Copy code

/my-app

|-- /node\_modules

|-- /public

| |-- /css

| |-- /js

| |-- /images

|-- /src

| |-- /controllers

| |-- /models

| |-- /routes

| |-- /views

| |-- /middlewares

| |-- index.js

|-- .gitignore

|-- package.json

|-- README.md

**6. What is MVC?**

**MVC (Model-View-Controller)** is a design pattern used for developing web applications. It divides the application into three inter-connected components:

* **Model**: Manages data and business logic.
* **View**: Displays data and sends user actions to the controller.
* **Controller**: Handles user input, manipulates the model, and updates the view.

**Example**: In a Node.js application using Express and a template engine:

* **Model** (e.g., User.js):

javascript

Copy code

const mongoose = require('mongoose');

constUserSchema = new mongoose.Schema({

name: String,

email: String

});

module.exports = mongoose.model('User', UserSchema);

* **View** (e.g., index.ejs):

html

Copy code

<html>

<head>

<title>Users</title>

</head>

<body>

<h1>Users List</h1>

<ul>

<% users.forEach(user => { %>

<li><%= user.name %> - <%= user.email %></li>

<% }); %>

</ul>

</body>

</html>

* **Controller** (e.g., userController.js):

javascript

Copy code

const User = require('../models/User');

exports.getUsers = async (req, res) => {

const users = await User.find();

res.render('index', { users });

};

**7. HTTP vs HTTPS**

* **HTTP (Hypertext Transfer Protocol)**: An application protocol for distributed, collaborative, hypermedia information systems. HTTP is used to fetch resources such as HTML documents but is not secure.
* **HTTPS (HTTP Secure)**: An extension of HTTP. It uses SSL/TLS to encrypt data between the client and server, ensuring data integrity and security.

**8. Difference between PUT and PATCH**

* **PUT**: Replaces the entire resource with the new data provided. It is idempotent, meaning multiple identical requests should have the same effect as a single request.
* **PATCH**: Applies partial modifications to a resource. It is not necessarily idempotent as multiple identical requests may have different effects.

**Example**:

* **PUT** request to update a user's data:

http

Copy code

PUT /users/1

{

"name": "John Doe",

"email": "john.doe@example.com"

}

* **PATCH** request to update a user's email:

http

Copy code

PATCH /users/1

{

"email": "john.new@example.com"

}

**9. SQL Queries**

* **HAVING Clause**: Used to filter records after an aggregation (e.g., GROUP BY).

**Example**:

sql

Copy code

SELECT department, COUNT(\*)

FROM employees

GROUP BY department

HAVING COUNT(\*) > 5;

* **WHERE Clause**: Used to filter records before any groupings.

**Example**:

sql

Copy code

SELECT \* FROM employees

WHERE salary > 50000;

* **Aggregate Functions**: Perform calculations on multiple rows and return a single result (e.g., SUM, AVG, COUNT).

**Example**:

sql

Copy code

SELECT AVG(salary) FROM employees;

**10. Explain MERN Stack**

**MERN Stack** is a JavaScript stack for building full-stack web applications. It consists of:

* **MongoDB**: A NoSQL database.
* **Express.js**: A web application framework for Node.js.
* **React**: A front-end library for building user interfaces.
* **Node.js**: A runtime environment for executing JavaScript on the server side.

**11. XML vs JSON**

* **XML (eXtensible Markup Language)**: A markup language designed to store and transport data, with a focus on what data is.

**Example**:

xml

Copy code

<person>

<name>John</name>

<age>30</age>

</person>

* **JSON (JavaScript Object Notation)**: A lightweight data interchange format that's easy for humans to read and write and easy for machines to parse and generate.

**Example**:

json

Copy code

{

"name": "John",

"age": 30

}

**12. Why JSON is used nowadays**

* **Lightweight and easy to read**: JSON's syntax is simpler and more compact than XML.
* **Easier to parse**: JSON parsers are readily available in most programming languages.
* **JavaScript compatibility**: JSON is native to JavaScript, making it a natural choice for web development.

**13. Pros of XML**

* **Self-descriptive**: XML is verbose and self-descriptive, which makes it easier to understand the data structure.
* **Extensible**: Custom tags can be defined, providing flexibility.
* **Widely used in configurations and data interchange**: XML is widely used in web services (SOAP) and configuration files.

**14. In MVC, what is a Controller?**

A **Controller** in MVC acts as an intermediary between the Model and the View. It processes incoming requests, manipulates data using the Model, and returns the final output to the View.

**Example**: In a Node.js application using Express:

* **Controller** (e.g., userController.js):

javascript

Copy code

const User = require('../models/User');

exports.getUser = async (req, res) => {

const user = await User.findById(req.params.id);

res.render('user', { user });

};

exports.createUser = async (req, res) => {

constnewUser = new User(req.body);

await newUser.save();

res.redirect('/users');

};

**15. SQL vs NoSQL**

* **SQL (Structured Query Language)**: Used with relational databases. Data is stored in tables with fixed schemas.

**Example**: MySQL, PostgreSQL.

sql

Copy code

SELECT \* FROM users WHERE age > 30;

* **NoSQL**: Used with non-relational databases. Data can be stored in various formats like documents, key-value pairs, graphs, or wide-column stores.

**Example**: MongoDB, CouchDB.

javascript

Copy code

db.users.find({ age: { $gt: 30 } });

**16. MongoDB vs MySQL**

* **MongoDB**:
  + NoSQL database.
  + Uses JSON-like documents.
  + Schema-less.
  + Scales horizontally.
* **MySQL**:
  + SQL relational database.
  + Uses tables with fixed schemas.
  + Schema-based.
  + Scales vertically.

**17. JWT**

**JWT (JSON Web Token)** is a compact, URL-safe means of representing claims to be transferred between two parties. It is commonly used for authentication and information exchange.

**Example**:

json

Copy code

{

"alg": "HS256",

"typ": "JWT"

}

{

"sub": "1234567890",

"name": "John Doe",

"iat": 1516239022

}

**18. What is Crypto?**

**Crypto** refers to cryptography, which is the practice and study of techniques for securing communication and data. In web development, it is often used for encrypting data, hashing passwords, and ensuring secure communication.

**Example**: Using the crypto module in Node.js to hash a password:

javascript

Copy code

const crypto = require('crypto');

const hash = crypto.createHash('sha256').update('password').digest('hex');

console.log(hash); // Outputs the hashed password

**19. When to use MySQL/MongoDB databases?**

* **MySQL**: Use when your application requires complex transactions, relationships between entities, and structured data with fixed schemas.
* **MongoDB**: Use when your application needs to handle large volumes of unstructured or semi-structured data, requires horizontal scaling, or when the schema might evolve over time.

**20. What is Schema?**

A **Schema** defines the structure of data in a database. In SQL databases, schemas define tables, fields, types, and relationships. In NoSQL databases, schemas can define the structure of documents.

**21. ACID Properties**

**ACID** properties ensure reliable processing of database transactions:

* **Atomicity**: Transactions are all-or-nothing.
* **Consistency**: Transactions bring the database from one valid state to another.
* **Isolation**: Transactions are executed in isolation from other transactions.
* **Durability**: Once a transaction is committed, it remains so, even in the event of a system failure.

**22. What is a Transaction?**

A **Transaction** is a sequence of operations performed as a single logical unit of work. A transaction must be all-or-nothing: either all of its operations are completed successfully, or none are.

**23. Normalization**

**Normalization** is the process of organizing data in a database to reduce redundancy and improve data integrity. It involves dividing large tables into smaller tables and defining relationships between them.

**Example**:

* **Unnormalized table**:

sql

Copy code

CREATE TABLE orders (

order\_id INT,

customer\_name VARCHAR(255),

product\_name VARCHAR(255),

product\_price DECIMAL

);

* **Normalized tables**:

sql

Copy code

CREATE TABLE customers (

customer\_id INT PRIMARY KEY,

customer\_name VARCHAR(255)

);

CREATE TABLE products (

product\_id INT PRIMARY KEY,

product\_name VARCHAR(255),

product\_price DECIMAL

);

CREATE TABLE orders (

order\_id INT PRIMARY KEY,

customer\_id INT,

product\_id INT,

FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id),

FOREIGN KEY (product\_id) REFERENCES products(product\_id)

);