1. **What is React? What are the advantages of using React? What are the limitations of React?**

ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components. It is an open-source, component-based front end library responsible only for the view layer of the application.

**Advantage:**

1. Component-Based Architecture:. This modular structure promotes reusability, maintainability, and scalability, as components can be easily composed and nested within each other.
2. Virtual DOM: React can efficiently update and render only the necessary components that have changed, rather than re-rendering the entire page. This leads to improved performance and a smoother user experience.
3. Simplicity:ReactJS uses JSX file which makes the application simple and to code as well as understand. We know that ReactJS is a component-based approach which makes the code reusable as your need. This makes it simple to use and learn.
4. platforms. This allows for faster development and easier maintenance of mobile apps.
5. Performance Optimization: use of a virtual DOM, memoization, and shouldComponentUpdate lifecycle method, which help in improving the efficiency of React applications.
6. JSX Syntax: React uses JSX (JavaScript XML) syntax, which allows developers to write HTML-like code within JavaScript. This enhances code readability and makes it easier to visualize

**Disadvantage**

**It’s a Library, Not a Framework:** the flexible nature of React reduces uniformity. Flexibility in the organization of code makes it harder to understand code. This can be difficult when analyzing a project you are unfamiliar with.

**It Uses JSX:**it allows you to use a mixture of HTML and JavaScript to write code in componentsJSX adds complexity to learning React, especially for beginner

**Lack of Updated Documentation:** Changes made to the libraries affect the performance of Apps if not updated.

1. **Components**
   1. In component-based approach the entire application is divided into a small logical group of code, which is known as components.
   2. A Component is considered as the core building blocks of a React application. It makes the task of building UIs much easier.
   3. Each component exists in the same space, but they work independently from one another and merge all in a parent component, which will be the final UI of your application.
   4. Every React component have their own structure, methods as well as APIs
   5. The components are rendered to a root element in the DOM using the React DOM library.

**Two types of components**

1. **Functional component** 
   1. only contain a render method and don't have their own state.
   2. They are simply JavaScript functions that may or may not receive data as parameters. We can create a function that takes props(properties) as input and returns what should be rendered
   3. he functional component is also known as a stateless component because they do not hold or manage state.
2. **Class component**
   * + 1. Class components are more complex than functional components. It requires you to extend from React. Component and create a render function which returns a React element.
       2. You can pass data from one class to other class components..
       3. The class component is also known as a stateful component because they can hold or manage local state.
3. **State** 
   1. The state is an updatable structure that is used to contain data or information about the component.
   2. The state in a component can change over time. The change in state over time can happen as a response to user action or system event.
   3. A component with the state is known as stateful components
   4. It determines the behavior of the component and how it will render. They are also responsible for making a component dynamic and interactive.
4. **Props**
   1. Props stand for "Properties." They are read-only components.
   2. It gives a way to pass data from one component to other components
   3. Props are immutable so we cannot modify the props from inside the component
5. **What are React Hooks?**
   1. Hooks is a feature that allow developers to use state and other React features without writing a class component.
   2. It does not work inside classes.

**Rules of Hook**

1. Only call Hooks at the top level

Do not call Hooks inside loops, conditions, or nested functions.

2. Only call Hooks from React functions

1. **What is useState() in React?**

useState is the Hook which needs to call inside a function component to add some local state to it. The useState returns a pair where the first element is the current state value/initial value, and the second one is a function which allows us to update it. After that, we will call this function from an event handler The useState is similar to this.setState in class.

1. **What is the use of useEffect React Hooks?**

The Effect Hook allows us to perform side effects (an action) in the function components. It does not use components lifecycle methods which are available in class components. In other words, Effects Hooks are equivalent to componentDidMount(), componentDidUpdate(), and componentWillUnmount() lifecycle methods.

**Side effects have common features such as:**

* Updating the DOM,
* Fetching and consuming data from a server API,
* Setting up a subscription, etc.

# **useContext Hook**

* 1. React Context is a way to manage state globally.
  2. It can be used together with the useState Hook to share state between deeply nested components more easily than with useState alone.
  3. In nested components. The component at the top and bottom of the stack need access to the state.So To do this we will need to pass the state as "props" through each nested component. So create context is used
  4. Context Provider wrap the tree of components that need the state Context.

Example :

**import React, { createContext, useContext } from 'react';**

**// Create a context**

const MyContext = createContext();

**// A component that provides the context value**

const ParentComponent = () => {

const contextValue = 'Hello, useContext!';

return (

<MyContext.Provider value={contextValue}>

<ChildComponent />

</MyContext.Provider>

);

};

**// A child component that consumes the context value**

const ChildComponent = () => {

const contextValue = useContext(MyContext);

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

};

**// Render the parent component**

const App = () => {

return (

<div>

<ParentComponent />

</div>

);

};

export default App;

1. **Why do React Hooks make useref ?**

useRef hook returns a mutable ref object whose current property can be used to store and access a value.

* + 1. **Storing references to DOM elements or components**: you can create a ref object and attach it to a DOM element to access and manipulate the element or component without triggering a re-render.
    2. **Managing focus, timers, and other imperative operations:**

1. **What are the rules that must be followed while using React Hooks?**

1. Only call Hooks at the top level

Do not call Hooks inside loops, conditions, or nested functions.

2. Only call Hooks from React functions

1. **Explain Strict Mode in React.**
   1. "strict mode" is a development mode feature that helps highlight potential issues and warnings in your application.
   2. It is a wrapper component provided by React that you can use to enable additional checks and stricter mode of rendering for your components.
   3. To catch and address potential problems early on, ensuring a better and more stable React application.
   4. find components with unsafe or deprecated lifecycle methods. give you a warning if you are using methods like componentWillMount, componentWillUpdate
   5. enables warnings for deprecated features and upcoming changes in React.
2. **How to pass data between react components?**
3. **Props** (Properties): The most common way to pass data is through props. You can pass data from a parent component to its child component by assigning values to attributes when rendering the child component. The child component can then access the data through its props.
4. **Callback** **functions**: You can also pass callback functions as props to child components to allow them to communicate with their parent components. The child component can call the callback function, passing any necessary data as arguments.
5. **What are Higher Order Components?**

A higher-order component is a function that takes a component as input and returns a new component with extended functionality. It acts as a wrapper component around the original component, providing additional props, manipulating the component's behavior, or handling certain aspects of its rendering.

Example:

**// A simple component**

const MyComponent = ({ name }) => {

return <div>Hello, {name}!</div>;

};

**// A Higher-Order Component**

const withUpperCaseName = (WrappedComponent) => {

return (props) => {

const { name } = props;

const upperCaseName = name.toUpperCase();

return <WrappedComponent {...props} name={upperCaseName} />;

};

};

**// Wrap the component with the HOC**

const EnhancedComponent = withUpperCaseName(MyComponent);

EnhancedComponent calling withUpperCaseName and passing MyComponent as an argument. This wraps MyComponent and modifies its behavior by providing an uppercase version of the name prop.

1. **What are the different phases of the component lifecycle?**

## **Initial Phase:** It is the **birth** phase of the lifecycle of a ReactJS component. Here, the component starts its journey on a way to the DOM. In this phase, a component contains the default Props and initial State. These default properties are done in the constructor of a component

* **Mounting Phase:** the instance of a component is created and inserted into the DOM
* **Updating Phase:**This phase also allows to handle user interaction and provide communication with the components hierarchy. The main aim of this phase is to ensure that the component is displaying the latest version of itself**.**
* **Unmounting Phase:** It is the final phase of the react component lifecycle. It is called when a component instance is destroyed and unmounted from the DOM

1. **What are the lifecycle methods of React?**

**Mounting Phase:**

**constructor(props):** The constructor is called when a component is being initialized. It is used to set up the initial state and bind event handlers.

**static getDerivedStateFromProps(nextProps, prevState):** This static method is invoked before rendering, whenever new props are received. It allows the component to update its internal state based on changes in props. It returns an object representing the updated state or null.

**render():** The render method is responsible for rendering the component's JSX markup. It returns a React element representing the component's UI.

**componentDidMount():** This method is called immediately after the component has been mounted (inserted into the DOM tree). It is commonly used to initiate network requests, set up subscriptions, or perform other initialization tasks.

**Updating Phase:**

**ComponentWillUpdate(nextProps, prevState):** Similar to the mounting phase, this method is called when new props are received. It allows the component to update its internal state based on changes in props.

**shouldComponentUpdate(nextProps, nextState):** This method determines if the component should re-render or not. By default, it returns true.

**render():** The render method is called to update the component's UI.

**componentDidUpdate(prevProps, prevState):** This method is called after the component has updated and re-rendered. It is commonly used to perform side effects, such as fetching new data based on prop changes**.**

**Unmounting Phase:**

**componentWillUnmount():** This method is called right before the component is unmounted and destroyed. It is used to perform cleanup tasks, such as canceling network requests, unsubscribing from event listeners, or releasing resources.

1. **Differentiate React Hooks vs Classes.**

|  |  |  |
| --- | --- | --- |
|  | Hooks | classes |
| Functionality: | provide a more focused and composable way to manage state, side effects, and component logic. | classes lifecycle methods (componentDidMount, componentDidUpdate, etc.), instance variables, and inheritance. |
| syntax | Regular JavaScript functions that can be used directly within functional components | Class components require the use of this keyword and have a specific structure with lifecycle methods. |
| State Management | useState hook, which allows functional components to have their own state | this.state object and setState method to manage component state |
| Lifecycle Methods | React hooks provide equivalents to some of these lifecycle methods using the useEffect hook. useEffect allows you to perform side effects and handle lifecycle events within functional components. | Class components have lifecycle methods like componentDidMount, componentDidUpdate, and componentWillUnmount, which allow you to hook into specific moments of a component's lifecycle |
|  |  |  |

1. **What is Redux?**

* Redux is an open-source JavaScript library used to manage application state.
* React uses Redux for building the user interface.
* It follows the principles of Flux and provides a predictable state container for javascript apps that helps manage the state of an application in a centralized and predictable manner.

**The main reason to use React Redux are:**

* State transfer between components is pretty messy in react as it is hard to keep track of which compinent data is coming It becomes really complicated if users are working with a large

number of states within an application. **Redux** solvesit by storing all te state in a place called store

* Every component in application can then directly access the required state from the store
* It implements many performance optimizations internally, which allows to components re-render only when it actually needs.

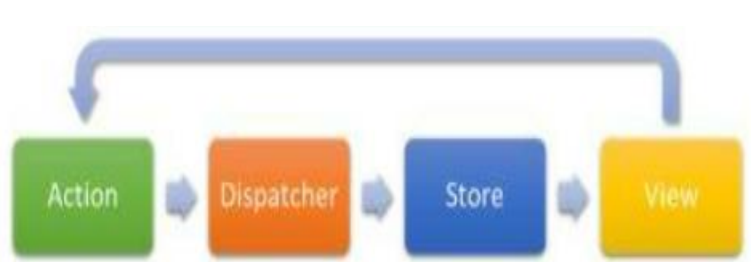
**Principles of Redux**

* The state of your whole application is stored in an

object tree within a single-store.

* A single state tree makes it easier to debug or inspect an application It gives you a faster development cycle by enabling you to persist

**The components of Redux architecture are explained below.**

* **Store:** The store is a single JavaScript object that holds the complete state of the application. It is the central hub for all state-related operations in Redux. The store is created using the createStore() function provided by Redux.
* **Actions:** Actions are plain JavaScript objects that describe events or user interactions in the application. They have a type property that describes the type of action and can optionally contain additional data. Actions are dispatched using the dispatch() function, which triggers the state update process.
* **Reducers:** Reducers are pure functions that specify how the state should change in response to dispatched actions. They take the current state and an action as input and return a new state. Reducers should not mutate the original state, but instead create a new state object.
* **Dispatch**: Dispatching an action is the process of sending an action to the store. It is done using the dispatch() function provided by Redux. When an action is dispatched, it flows through the reducers, which update the state accordingly.
* 

1. **How do you style React components?**

**Inline CSS**: In inline styling basically we create objects of style. And render it inside the components in style attribute using the React technique to incorporate JavaScript variable inside the JSX (Using ‘{ }’ )

**Normal CSS:** In the external CSS styling technique, we basically create an external CSS file for each component and do the required styling of classes. and use those class names inside the component

**CSS in JS:** It uses javascript objects to describe styles in a declarative way using ‘createUseStyles’ method of react-jss and incorporate those styles in functional components using className attribute.

**EXAMPLE:**

import {createUseStyles} from 'react-jss'

const styles = createUseStyles({

student : {

border : '2px solid green',

width: '40%',

listStyleType:'none'

},

studentDetails : {

color : 'blue',

fontSize : '23px'

}

})

const StudentList = (props) => {

const classes = styles()

const {name, classNo, roll, addr} = props

return(

<ul className={classes.student}>

<li className={classes.studentDetails}>Name : {name}</li>

)}

**Styled Components** : It is a library for React. It uses enhance CSS for styling React component systems in your application, which is written with a mixture of JavaScript and CSS.

1. **What is React Router? Why do we need to React Router? How is React routing different from conventional routing? How do you implement React routing?**

* React Router is a popular routing library for React applications. It provides a declarative way to handle routing and navigation in a React application, allowing you to create dynamic, single-page applications with multiple views or pages.
* React Router helps in mapping specific URLs to corresponding components, enabling navigation and rendering different components based on the current URL.
* It provides a set of components and hooks that can be used to define routes, navigate between views, and access URL parameters.

**Here are some key features and components provided by React**

**BrowserRouter**: It provides the foundation for routing in a React application.

**Route**: A component used to define a mapping between a URL path and a component to render when that path is matched. It can be nested to create nested routes and handle complex routing scenarios.

**Switch**: A component that renders the first matching Route or Redirect within its children. It helps in rendering only the first matching route, useful when defining multiple routes.

**Link** :Components that provide declarative navigation by rendering HTML anchor tags (<a>) that automatically handle updating the URL and triggering the corresponding route rendering.

**Need to React Router**

**Client-Side Routing:** React Router enables client-side routing, allowing the creation of single-page applications (SPAs) where navigation occurs without refreshing the entire page. It uses JavaScript to handle routing on the client-side, resulting in a faster and smoother user experience.

**Declarative Routing**: React Router provides a declarative approach to define routes using components and props. This makes it easier to understand and maintain the routing logic

|  |  |  |
| --- | --- | --- |
|  | **React routing** | **conventional routing** |
| **Client-Side Routing vs. Server-Side** | Client-side routing, meaning the routing is handled entirely within the client's browser using JavaScript. The routing logic and rendering of different components occur without a full page reload. | primarily performed on the server-side, where the server determines the appropriate response based on the requested URL |
| **Single-Page Applications (SPAs):** | React routing is often used in the context of building single-page  SPAs dynamically update the content based on user interactions and routing | Typically used in traditional multi-page applications, where navigating between pages involves a full page reload. |
| **URL** | URLs are used to represent application states or views. The URL can be used to determine which components to render and how to display specific content within the application. | URLs primarily represent server-side resources or endpoints. URLs often correspond to physical files or routes on the server. |
| **Routing** | React routing libraries, such as React Router, provide built-in history management. They allow you to manage browser history, handle back and forward navigation, and programmatically navigate between different routes within the application without triggering a full page reload. | It doesn’t provide that |

1. **What is flux**

* It is neither a library nor a framework.
* It is an architectural pattern that was introduced by Facebook as a way to manage the flow of data in complex web applications.
* It follows the concept of Unidirectional Data Flow model.
* It is useful when the project has dynamic data, and we need to keep the data updated in an effective manner.
* It reduces the runtime errors.

**Flux applications have three major roles in dealing with data:**

* **Dispatcher**: The Dispatcher is responsible for receiving actions from various sources and dispatching them to registered stores. It acts as a central hub for coordinating data flow in the application
* **Actions**: Actions represent user interactions or system events that trigger changes in the application's state. They encapsulate the data related to the event and are dispatched to the Dispatcher.
* **Stores**: It primarily contains the application state and buisness logic It is used for maintaining a particular state within the application, updates themselves in response to an action, and emit the change event to alert the controller view.
* **Views** (React components): It is located at the top of the chain to store the logic to generate actions and receive new data from the store.

1. **What is JSX?**

* JSX (JavaScript XML) is an extension to the JavaScript language syntax used in React applications. It allows you to write HTML-like code directly within JavaScript, making it easier to define the structure and appearance of components.
* JSX combines the power of JavaScript with the flexibility of HTML-like syntax. It provides a concise and intuitive way to create React elements and define the user interface of a component.

1. **What are the differences between functional and class components?**

|  |  |  |
| --- | --- | --- |
|  | Functional | class |
| Syntax | They are simpler and have a more concise syntax compared to class components. | lass components are defined as ES6 classes that extend the base React.Component class. They have a more verbose syntax compared to functional components. |
| State Management | hey couldn't manage their own state. However, with the advent of Hooks, functional components can now use state and other React features like lifecycle methods. | Class components have built-in state management. They can define and manage their own state using the setState() method provided by React. |
| Lifecycle Methods | functional components did not have access to lifecycle methods. However, with the introduction of Hooks, functional components can now use the useEffect() hook to perform side effects and mimic lifecycle behavior. | Class components have access to lifecycle methods like componentDidMount(), componentDidUpdate(), and componentWillUnmount(). These methods allow developers to handle component initialization, state updates, and cleanup. |
| Code Reusability | Functional components promote code reusability as they are pure functions. They only rely on the props passed to them and do not have any internal state. | same |
| Performance: | Functional components are generally more performant because they are lightweight and do not require the overhead of the additional class component features. | Class components support the use of ref to access and interact with the underlying DOM elements or child co  mponents. |

1. **What is the virtual DOM? How does react use the virtual DOM to render the UI?**

It is a lightweight representation of the actual DOM (Document Object Model) tree structure.

The virtual DOM allows React to efficiently update and render components in response to changes in the application state.

**Here's how the Virtual DOM works in React:**

**Initial** **Render**: When a React component is rendered for the first time, it creates a virtual representation of the DOM tree called the virtual DOM.

**Component** **Updates**: When the state or props of a component change, React re-renders the component. It generates a new virtual DOM representation of the component based on the updated state/props.

**Diffing** : React performs a process called "diffing" to compare the new virtual DOM with the previous virtual DOM. It identifies the differences between the two representations, known as the "diffs" or "changes."

**Reconciliation:** After identifying the differences, React applies only the necessary changes to the actual DOM, rather than updating the entire DOM tree. This process is known as reconciliation. By updating only the specific parts of the DOM that have changed

1. **What is render() function**

render() function is a required method that is used to define the structure and content of a component's UI. It determines what should be displayed on the screen when the component is rendered.

**Here are the key points about the render() function**:

**Class Components:** The render() function is specific to class components in React. It is defined within a class that extends the React.Component base class.

**Return Value:** The render() function must return a single React element, which represents the component's UI. This element can be a single HTML-like element or a fragment containing multiple elements.

**JSX Syntax:** The render() function typically uses JSX (JavaScript XML) syntax to describe the UI. JSX allows you to write HTML-like code within JavaScript, making it easier to define the structure and appearance of the U

1. **What is Babel**

Babel is a popular JavaScript compiler that is widely used in modern web development, particularly in the context of building applications It allows developers to write code using the latest JavaScript syntax and transpile it into a version of JavaScript that is compatible with older browsers and environments.

1. **How do you create a React app?**
   1. **Set Up Development Environment**: install Node.js which comes bundled with npm.
   2. **Create a New React App**: Open terminal and navigate to the directory where you want to create your React app. Then, run

**npx create-react-app my-app**

* 1. **Navigate to App Directory**: Change your current directory to the newly created app directory by

**cd my-app**

* 1. **Start the Development Server**: by running

**npm start**

* 1. This command will build the app and open it in your default browser at http://localhost:3000.
  2. **Explore and Modify the App**: start exploring and modifying the code in the src directory. The main file you'll work with is src/App.js, which contains the root component of the app.
  3. **Add Components**: create components, import them into your App.js file,

1. **What is the map() function?**

It is used to iterate over an array and transform each element in the array based on a function. The map() function returns a new array with the transformed elements, without modifying the original array.

const numbers = [1, 2, 3, 4, 5];

const doubledNumbers = numbers.map((num) => num \* 2);

console.log(doubledNumbers); // Output: [2, 4, 6, 8, 10]

1. **What are some of the advantages of Angular over other frameworks?**

**MVC Architecture**: Angular follows the Model-View-Controller (MVC) or Model-View-ViewModel (MVVM) architectural pattern. It provides a structured way to organize code and promote code reusability.

**Two-Way Data Binding**:In Angular changes in the model automatically reflect in the view and vice versa. This simplifies data synchronization and reduces the need for manual DOM manipulation.

**Dependency Injection (DI**): Angular manage dependencies between components. DI promotes code modularity, testability, and reusability by allowing components to declare their dependencies

**TypeScript Language**: Angular is built with TypeScript, TypeScript adds features such as static typing, classes, interfaces, and modules to JavaScript.

**Rich Set of Features**: It includes features like routing, form handling, validation, server communication, and component-based architecture.

1. **What is two way data binding**

* Two-way data binding allows automatic synchronization of data between the model (component) and the view (template). It enables changes made in the model to be reflected in the view, and vice versa, without the need for explicit data manipulation code.
* It is achieved using the ngModel directive. The ngModel directive binds a form control (such as an input or select element) to a property in the component. It establishes a two-way communication between the control and the component's property, ensuring that any changes made to the control's value are immediately reflected in the component, and any changes made in the component are propagated to the control.
* EXAMPLE:

export class AppComponent {

name: string = 'John Doe';

}

<input type="text" [(ngModel)]="name">

<p>Welcome, {{ name }}!</p>

Any changes made in the input element will update the "name" property in the component, and any changes made to the "name" property will update the input element's value.

* Two-way data binding simplifies the synchronization between the model and the view, reducing the need for manual event handling or property updates.
* It enhances productivity and readability by reducing the amount of boilerplate code required for data manipulation in Angular applications.

1. **List out differences between AngularJS and Angular?**

|  |  |  |
| --- | --- | --- |
|  | **AngularJS** | **Angular** |
| **Architecture** | based on the Model-View-Controller (MVC) architecture, | follows the Component-Based Architecture. |
| **Design** | uses two-way data binding and scope hierarchy for data flow | uses a unidirectional data flow and relies on reactive data binding with properties and events |
| **Language** | built with JavaScript and supports ES5 syntax. | built with TypeScript, |
| **Performance** | suffers from performance issues due to its two-way data binding and digest cycle. | addresses these performance concerns with a more optimized change detection mechanism, making it faster and more efficient. |
| **Component-Based Architecture** | does not have native support for components and relies on directives for similar functionality. | application is built using reusable components. Components |

1. **What are directives in Angular?**

* Directives are defined as classes that can add new behavior to the elements in the template or modify existing behavior.
* used to extend HTML with custom behaviors.
* The purpose of Directives in Angular is to modify the DOM, be it by adding new elements to DOM or removing elements and even changing the appearance of the DOM elements. Declared with decorator @Directive.

**Component Directives** : They combine an HTML template with a TypeScript class to create reusable UI components. Components encapsulate their own logic, data, and presentation, making it easier to build and maintain complex user interfaces.

**Attribute directives :** It modify the behavior or appearance of an element or component For example, ngStyle( applying styles) or ngClass(applying CSS classes).

**Structural Directives:** Structural directives change the structure of the DOM by adding, removing, or manipulating elements An example would be \*ngIf or \*ngFor

1. **What are AngularJS controller?**

* AngularJS controllers are used to control the flow of data of AngularJS application.
* A controller is defined using ng-controller directive.
* A controller is a JavaScript object containing attributes/properties and functions.
* Each controller accepts $scope as a parameter which refers to the application/module that controller is to control.

1. **Explain Components, Modules and Services in Angular**

* Components : component represents a part of the user interface (UI) and encapsulates the template, styles, and behavior related to that UI.
* Components are responsible for rendering the view and handling user interactions.
* They consist of three main parts:

**Template**:

* The HTML markup that defines the structure and layout of the component's view.
* It represents the visual part of the component and is responsible for displaying data, handling user interactions, and rendering dynamic content.

**Styles:** The CSS styles that control the appearance and presentation of the component.

**Class:** The TypeScript class that contains the component's logic and data.

* Components are reusable and can be composed together to build complex user interfaces.
* They communicate with other components and services using inputs, outputs, and events.

**Modules:** It is a container for the different parts of your application like controller, services, filters, directives etc.

* A module is used as a Main() method. Controller always belongs to a module.
* Modules help in modularity, code organization, lazy loading, and managing application-level dependencies.

**Services**: Service is a piece of reusable code with a focused purpose.

* Services provide a way to centralize common operations, such as data fetching, data manipulation, and communication with external APIs.
* By separating reusable logic into services, you can achieve code reusability, testability, and maintainability.
* Common use cases for services include data fetching from APIs, authentication, caching, state management, and sharing data between components.

1. **What exactly is the router state?**

* the router state refers to the current state of the application's routing system.
* It represents the active route and its associated parameters,
* The router state is managed by the Angular Router, which is responsible for navigating between different views and updating the application's URL.
* It keeps track of the current route and allows you to access the router state to retrieve information about the current route and its parameters.

1. **What are router links?**

Router links are used to navigate between different routes in an application.

They are implemented using the <a> element and the routerLink attribute. The routerLink attribute specifies the target route or route path

1. **What are lifecycle hooks in Angular? Explain a few lifecycle hooks.**

* component lifecycle starts when Angular instantiates the component class and renders the component view and its child views.
* The lifecycle continues with change detection
* The lifecycle ends when Angular destroys the component instance and removes its rendered template from the DOM.
* **Hooks** allow you to act on a component or directive instance

**ngOnChanges:** This hook is called when one or more input properties of the component change.

**ngAfterViewInit(**): This hook is called after the component's view and child views are initialized. It is used to perform any initialization tasks that rely on the component's view, such as accessing and manipulating DOM element

**ngOnInit:** It is commonly used for initialization tasks like fetching data from a service or initializing component properties.

**ngDoCheck:** This hook is called during every change detection cycle and allows you to implement custom change detection logic. It is triggered whenever Angular checks for changes in the component's properties or its child components.

**ngOnDestroy:** This hook is called just before the component is destroyed and removed from the DOM. I

1. **What does Angular Material mean?**

* Angular Material is a UI component library that provides a set of pre-built and ready-to-use UI components
* Angular Material offers a wide range of components such as buttons, cards, forms, dialogs, menus, tooltips, sliders, tabs, and more

1. **What is transpiling in Angular?**

transpiling refers to the process of converting TypeScript code into JavaScript code that can be understood and executed by web browsers**.**

**Steps:**

* 1. The TypeScript compiler checks the type annotations
  2. Transpiling: The TypeScript compiler converts the TypeScript code into equivalent JavaScript code. It transforms the TypeScript features, such as classes, interfaces, decorators, and TypeScript-specific syntax, into their JavaScript equivalents.
  3. Bundling: The transpiled JavaScript code is bundled together into one or more JavaScript bundles
  4. Minification and Optimization: The bundled JavaScript code can be further minified and optimized to reduce its size

1. **What is AOT compilation? What are the advantages of AOT?**

AOT (Ahead-of-Time) compilation is a compilation technique to convert the application's templates and components into JavaScript code during the build process.

With AOT compilation, the templates are compiled and converted into JavaScript before the application is deployed, rather than being compiled at runtime in the browser.

**Advantages of AOT**

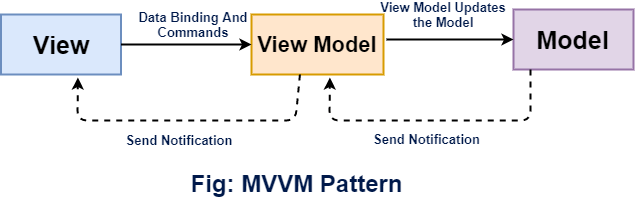
**Improved Performance**: As templates are already compiled and optimized during the build process. This eliminates the need for template compilation at runtime, resulting in improved runtime performance.

**Smaller Bundle Size**: AOT compilation removes unused code from the final bundle. This results in smaller bundle sizes, leading to faster downloads.

**Detect Template Errors**: This allows you to catch potential errors early and ensures a more stable and reliable application.

**Template Security**: AOT compilation removes the Angular-specific template syntax from the final JavaScript bundle,which improves the security of the application.

1. **Explain MVVM architecture**



**Model**: The Model represents the data and business logic of the application. It encapsulates the data structures, entities, and operations related to the application's functionality. It is responsible for retrieving, manipulating, data. The Model notifies the ViewModel of any changes to the data.

**View**: The View represents the user interface (UI) of the application. It defines how the data from the ViewModel is presented to the user and how user interactions are captured.

**ViewModel**: The ViewModel acts as a bridge between the Model and the View. It exposes the data and operations from the Model to the View, providing a way for the View to access and manipulate the data. It handles the communication between the View and the Model, updating the Model based on user actions or changes in the View.

**BENEFITS**

The MVVM pattern separates the data, UI logic, and user interaction concerns, making the application easier to understand, maintain, and test.

**Testability**: With clear separation of responsibilities, the ViewModel and Model can be unit tested independently without a dependency on the UI.

**Reusability**: The ViewModel can be reused across multiple Views, allowing for better code reuse.

1. **How does one share data between components in Angular?**

**Input and Output properties:** Use @Input() and @Output() decorators to define input and output properties on the child component. The parent component can bind data to the child component using the input property, and the child component can emit events using the output property to communicate back to the parent component.

**Shared** **service**: Create a shared service that acts as a central data store or communication channel between components.

**Route parameters**: If the data needs to be shared between components based on the current route, you can pass data through route parameters

1. **What are the different data types present in javascript?**
2. **Explain Hoisting in javascript.**
3. **Difference between “ == “ and “ === “ operators.**
4. **Difference between var and let keyword in javascript.**
5. **Explain Implicit Type Coercion in javascript.**
6. **Is javascript a statically typed or a dynamically typed language?**
7. **What is NaN property in JavaScript?**
8. **Explain passed by value and passed by reference.**

**Passed by Value:** When a primitive data type (such as numbers, strings, booleans) is assigned to a variable or passed as an argument to a function, it is passed by value. This means that a copy of the value is created and assigned to the new variable or parameter. Any modifications made to the new variable or parameter do not affect the original value.

let num1 = 5;

let num2 = num1; **// Passed by value, a copy of the value is assigned**

num2 = 10;

console.log(num1); // Output: 5 (original value is unchanged)

console.log(num2); // Output: 10 (new value)

**Passed by Reference**: When objects (including arrays and functions) are assigned to a variable or passed as an argument to a function, they are passed by reference. This means that instead of copying the entire object, a reference or pointer to the original object is passed. Modifying the new variable or parameter will affect the original object.

let arr1 = [1, 2, 3];

let arr2 = arr1; // Passed by reference, a reference to the original array is assigned

arr2.push(4);

console.log(arr1); // Output: [1, 2, 3, 4] (original array is modified)

console.log(arr2); // Output: [1, 2, 3, 4] (new reference)

// Similarly, when passing objects to functions

function modifyArray(array) {

array.push(5);

}

modifyArray(arr1);

console.log(arr1); // Output: [1, 2, 3, 4, 5] (original array is modified)

1. **What is an Immediately Invoked Function OR self invoking function in JavaScript?**
   1. a JavaScript function that is executed immediately after it is defined.
   2. It is a way to create a self-contained scope for the code inside the function and to avoid polluting the global scope.
   3. The primary purpose of using an IIFE is to encapsulate variables and functions within a local scope.
   4. Variables defined inside the IIFE are not accessible from the outside
2. **What do you mean by strict mode in javascript and characteristics of javascript strict-mode?**

Strict mode enables a stricter set of rules and stricter error handling in the JavaScript runtime.

When strict mode is enabled, certain silent errors are converted into thrown errors**.**

**Characteristics:**

**No implicit global variables:** assigning a value to an undeclared variable results in a ReferenceError.

**Strict function scope:** the value of this inside a function is undefined if the function is not called as a method or constructor.

**No duplicate parameter** it results in a SyntaxError. This helps avoid potential naming conflicts and ambiguity.

**No Octal literals:** Octal literals are considered syntax errors.

**Assigning to read-only properties and variables:** TypeError**.**

**Deleting variables, functions, or function arguments**:. Deleting them results in a SyntaxError.

1. **Explain Higher Order Functions in javascript.**

In JavaScript, higher-order functions are functions that can take other functions as arguments or return functions as their results.

**Accepting functions as arguments:**

function higherOrderFunction(callback) {

// Do some processing

}

function callbackFunction() {

console.log('Callback function executed');

}

higherOrderFunction(callbackFunction);

**Returning functions as results:**

function createMultiplier(factor) {

return function (number) {

return number \* factor;

};

}

const double = createMultiplier(2);

console.log(double(5)); // Output: 10

const triple = createMultiplier(3);

console.log(triple(5)); // Output: 15

1. **Explain “this” keyword.**
   1. this keyword refers to the context in which a function is executed.
   2. It is a special variable that holds a reference to the object that owns or calls the function.
   3. The value of this is determined dynamically at runtime and can vary depending on how a function is invoked.

**Global context**: When a function is invoked in the global scope (outside of any object or function), it results window in browsers

**Object method**: When a function is called as a method of an object, this refers to the object itself.

const obj = {

name: 'John',

sayHello: function() {

console.log(`Hello, ${this.name}!`);}};

obj.sayHello(); // Output: Hello, John!

1. Explain call(), apply() and, bind() methods.

the call(), apply(), and bind() methods are used to explicitly set the value of **this** when invoking a function.

They provide a way to control the context in which a function is executed and allow you to pass arguments to the function.

**call() method**: The call() method is used to invoke a function with a specified this value the first argument to call() sets the value of this, and arguments are passed as parameters to the function being called.

function greet(message) {

console.log(`${message}, ${this.name}!`);

}

const person = {

name: 'John'

};

greet.call(person, 'Hello'); // Output: Hello, John!

**apply() method**: The apply() method is similar to call(), but it accepts the function arguments as an array or an array-like object.

function greet(message, punctuation) {

console.log(`${message}, ${this.name}${punctuation}`);

}

const person = {

name: 'John'};

greet.apply(person, ['Hello', '!']); // Output: Hello, John!

**bind() method:** It allows you to bind a function to a specific context without invoking it immediately. The bind() method returns a new function that can be called later.

1. **What is the difference between exec () and test () methods in javascript?**

**Exec():**

* The exec() method is used to search for a match in a given string.
* It returns an array containing information about the first match found, or null if no match is found.
* The returned array contains additional properties like index (the starting index of the match in the string) and input (the original input string).

const str = 'Hello, World!';

const regex = /W(or)ld/;

const result = regex.exec(str);

console.log(result); // Output: ["World", "or", index: 7, input: "Hello, World!"]

**test():**

* The test() method is used to check if a pattern matches a given string.
* It returns a boolean value (true or false) indicating whether a match is found.
* It does not provide detailed information about the match or capture groups.
* It is typically used when you only need to check if a pattern exists in a string, without requiring further details about the match.

const str = 'Hello, World!';

const regex = /W(or)ld/;

const result = regex.test(str);

console.log(result); // Output: true

1. **What is currying in JavaScript?**

* The process of transforming a function with multiple arguments into a curried function is called currying.
* It enables us to create new functions based on existing ones with preset values for some arguments.
* Currying can be achieved manually by creating nested functions

function add(a) {

return function(b) {

return a + b;

}}

const addCurried = add(5); // Currying the add function

console.log(addCurried(3)); // Output: 8

1. **Explain Scope and Scope Chain in javascript.**

**Scope** refers to the accessibility and visibility of variables, functions, and objects in a particular part of your code during runtime. It determines where variables and functions are defined and where they can be accessed.

**The scope chain** is formed based on the nesting of functions or blocks. Each function or block creates its own scope, and when a variable is not found in the current scope, JavaScript looks for it in the next outer scope until it finds the variable or reaches the global scope.

1. **Explain Closures in JavaScript.**

* A closure is a function that retains access to variables from its outer (enclosing) scope even after the outer function has finished executing.
* It allows access its lexical scope, including variables, even when the function is invoked outside its original scope.
* The closure keeps the reference to outerVariable intact, allowing innerFunction() to access and use it even when executed outside the scope of outerFunction().

function outerFunction() {

var outerVariable = 'I am from the outer function';

function innerFunction() {

console.log(outerVariable);

}

return innerFunction;

}

var closure = outerFunction();

closure(); // Output: I am from the outer function

1. **What are object prototypes?**

* Prototypes provide a mechanism for object inheritance, allowing objects to inherit properties and methods from other objects.
* his prototype object acts as a blueprint or template for the object, defining the shared properties and methods that the object can access.

1. **What are callbacks?**

a callback is a function that is passed as an argument to another function and is invoked or called by that function at a specific time or in response to a particular event.

1. **What is the use of a constructor function in javascript?**

* a constructor function is a special type of function that is used to create and initialize objects.
* It serves as a blueprint or template for creating multiple objects of the same type, often referred to as instances.
* The primary purpose of a constructor function is to define the properties and methods that will be shared by all instances created from it.
* When a constructor function is invoked with the new keyword, it creates a new object and sets the this keyword to refer to that newly created object.

1. **What is DOM? What do you mean by BOM?**

* **DOM** is a programming interface for web documents that represents the structure of an HTML or XML document as a tree-like model**.**
* DOM consists of a hierarchy of nodes that represent different parts of an HTML or XML document.
* Each node in the DOM tree corresponds to an element, attribute, or text content in the document.

**BOM** is an API provided by web browsers that allows JavaScript code to interact with and control the browser window and its various components.

**Some of the key components of the BOM include:**

* Window Object: The window object represents the browser window or tab.
* Document Object: The document object represents the web page loaded in the browser window
* History Object: The history object allows navigation through the user's browsing history
* Location Object: The location object represents the URL of the current web page.

1. **What is the distinction between client-side and server-side JavaScript?**

|  |  |
| --- | --- |
| **Client side** | **Server side** |
| runs on the client's web browser. | runs on the server, typically using a JavaScript runtime environment like Node.js. |
| used for enhancing the user interface and interactivity of web pages. | used for server-side scripting and handling server-related tasks. |
| used for form validation, handling user events, making AJAX requests, | interact with databases, perform file operations, handle HTTP requests, and implement server-side logic |
|  | used for building web servers, APIs, real-time applications, and server-side rendering. |

1. Differences between declaring variables using var, let and const.
2. **What is the rest parameter and spread operator?**

**Rest Parameter:**

The rest parameter is denoted by three dots (...) followed by a parameter name in a function declaration. It allows a function to accept an arbitrary number of arguments as an array.

function sum(...numbers) {

let total = 0;

for (let number of numbers) {

total += number;

}

return total;

}

console.log(sum(1, 2, 3, 4, 5)); // Output: 15

**Spread Operator:**

The spread operator is also denoted by three dots (...), but it is used in a different context. It allows an iterable (such as an array or string) to be expanded into individual elements**.**

const array1 = [1, 2, 3];

const array2 = [4, 5, 6];

const combinedArray = [...array1, ...array2];

console.log(combinedArray); // Output: [1, 2, 3, 4, 5, 6]

const string = 'Hello';

const characters = [...string];

console.log(characters); // Output: ['H', 'e', 'l', 'l', 'o']

1. **In JavaScript, how many different methods can you make an object?**

**Object literals:** Objects can be created using object literals by enclosing key-value pairs in curly braces.

const person = { name: 'John', age: 25 };

**Constructor function:**

function Person(name, age) {

this.name = name;

this.age = age;

}

const person = new Person('John', 25);

**Object.create():**

**const personPrototype = {**

greet() {

console.log('Hello!');

}

};

const person = Object.create(personPrototype);

person.greet(); // Output: Hello!

**ES6 Class:**

class Person {

constructor(name, age) {

this.name = name;

this.age = age;

}

greet() {

console.log('Hello!');

}

}

const person = new Person('John', 25);

person.greet(); // Output: Hello!

1. **What is the use of promises in javascript?**

Promises in JavaScript are a way to handle asynchronous operations Promises represent the eventual completion or failure of an asynchronous operation and allow you to attach callbacks that will be executed when the operation is successful (resolved) or encounters an error (rejected).

function getData() {

return new Promise((resolve, reject) => {

// Perform asynchronous operation

// If successful, call resolve() with the result

// If there's an error, call reject() with the error

});

}

getData()

.then((result) => {

// Handle successful operation

})

.catch((error) => {

// Handle error

});

1. What are classes in javascript?

Classes are essentially a blueprint for creating objects with shared properties and methods.

**Here are some key points about classes in JavaScript:**

**Class Definition:** Classes are defined using the class keyword followed by the class name.

**Constructor Method:** The constructor is called automatically when you create a new instance of the class using the new keyword. It is used to initialize the object's properties and can accept parameters.

**Properties**: Class instances can have their own properties, which are defined within the constructor or other methods using the this keyword. Properties hold the state or data of the object.

**Methods**: Methods define the behavior or actions that objects of the class can perform.

**Inheritance**: Classes support inheritance, allowing you to create a new class based on an existing class.

1. **What is Object Destructuring?**

Object destructuring allows to extract properties from an object and assign them to variables

It provides a shorthand syntax for accessing and extracting values from objects.

const person = {

name: 'John',

age: 30,

city: 'New York'

};

// Destructuring assignment

const { name, age, city } = person;

console.log(name); // Output: John

console.log(age); // Output: 30

console.log(city); // Output: New York

1. **What is Node.js? Where can you use it?**

Node.js is an open-source, server-side JavaScript runtime environment. It allows developers to run JavaScript code outside of a web browser, enabling them to build server-side applications,

**Uses and why:**

* 1. With the help of frameworks like Socket.IO, Node.js is popular for building real-time applications, including chat applications,
  2. Node.js finds applications in the IoT domain due to its lightweight nature and support for event-driven, non-blocking I/O.
  3. Node.js is also widely used for building command-line tools and scripts. It provides access to the file system, network
  4. Node.js brings JavaScript to the server-side, allowing developers to use a single programming language (JavaScript) across both the front-end and back-end of a web application.
  5. Node.js applications can handle a large number of requests with minimal overhead,
  6. Node.js comes with npm. npm provides access to thousands of open-source packages and libraries,

1. **Explain callback in Node.js.**

In Node.js callbacks is used to handle asynchronous operations. Asynchronous operations in Node.js, such as file I/O, network requests, and database queries, often take time to complete. Instead of blocking the execution of code until the operation finishes, Node.js allows you to provide a callback function that will be invoked once the operation is complete.

The callback function takes two parameters: an err parameter and a result/data parameter. If an error occurs during the operation, the err parameter will contain the error object. If the operation is successful, the data parameter is used to pass the result or data of the asynchronous operation to the callback function.

1. **Explain the difference between frontend and backend development?**

**Frontend development** involves building the client-side of a web application, which is what users interact with directly in their web browsers. It focuses on the presentation layer and user interface (UI) of the application.

**User Interface (UI):** in frontend the designing and implementing the visual and interactive elements of a website is done

**User Experience (UX):** it considers the overall user experience, ensuring that the application provides a smooth interaction for users.

**Browser Compatibility:** Frontend developers must ensure that the web application works consistently across different web browsers and devices.

**Responsive** **Design**: need to create responsive designs that adapt to different screen sizes.

**Backend development** focuses on building the server-side of a web application. It involves handling the logic, data storage, and communication between the client-side and server-side of the application.

**Database Management**: Backend developers work with databases to store and retrieve data required by the application.

**API Development**: Backend developers create APIs to llow other external services to interact with the applicatio

1. **What is NPM?**

NPM (Node Package Manager) is a package manager for Node.js applications and libraries. It is a command-line tool that allows developers to discover, install, manage, and publish reusable code packages written in JavaScript. NPM is the default package manager for Node.js and is bundled with the Node.js runtime environment.

1. **What are the modules in Node.js?**

modules are used to organize and encapsulate code into reusable units. They provide a way to split your codebase into smaller, manageable files, making it easier to maintain and reuse code across different parts of your application.

**There are three types of module**

**Built-in Core Modules:** These modules are available globally and can be used in any Node.js application without the need for additional installation. Examples of core modules include fs (file system), http (HTTP server), path (file path), util (utilities), and events (event handling).

**Local Modules**: Local modules are custom modules created by developers within the application. They encapsulate specific functionality and can be created in separate files. using the module.exports or exports object, and then import them into another file using the require functio

**Third-Party Modules:**created by other developers and are not built into Node.js. These modules extend the functionality of Node.js by providing additional features and utilities. They are typically published on the NPM

1. **What is the purpose of the module .Exports?**

It is used to define the public functions of a module, allowing functions, objects, or values to be exported and made available for use in other modules.

The purpose of module.exports is to control what is exposed or accessible outside of a module. By assigning values to module.exports, you specify which functions, objects, or values should be visible to other modules when they require or import the module.

1. **What is the difference between Angular and Node.js?**

|  |  |
| --- | --- |
| Angular | NodeJs |
| Angular is a frontend JavaScript framework used for building dynamic single-page applications (SPAs) and mobile web applications. | Node.js is a runtime environment that allows you to run JavaScript on the server-side. |
| offers a set of features and tools for building complex UI components, managing application state, handling data binding, routing, and performing client-side rendering. | Node.js is primarily used for building server-side applications, web servers, and APIs. It is efficient for handling concurrent requests and performing I/O operations. |
| provides a structured framework for developing web applications using TypeScriptIt follows the Model-View-Controller (MVC) architectural pattern. | Node.js has a vast ecosystem of libraries and modules making it easy to use in external dependencies into applications. |
| executed in the browser and interact with backend services or APIs to fetch and manipulate data. It can consume data from various sources, including Node.js backend APIs. | Node.js can be used with various frameworks, such as Express.js, to create robust server-side applications. |

1. **Which database is more popularly used with Node.js?**

MongoDB: MongoDB is a popular NoSQL database that is often used with Node.js. It is a document database that stores data in a flexible, JSON-like format called BSON (Binary JSON). MongoDB is known for its scalability, high performance, and easy integration with Node.js. It is particularly suitable for applications with rapidly changing or unstructured data and where the flexibility of a schema-less database is desired.

1. **What are some of the most commonly used libraries in Node.js?**

**Express** : Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

**Mongoose**: Mongoose is an object modeling library for MongoDB. It provides a straightforward way to define data schemas, perform CRUD operations, and interact with MongoDB databases using an intuitive API. Mongoose simplifies working with MongoDB in Node.js applications.

**Socket.IO:** Socket.IO is a library that enables real-time bidirectional communication between the server and clients using WebSocket and fallback mechanisms. It is commonly used for building chat applications, real-time collaboration tools, and applications that require instant data updates.

1. **What is the command used to import external libraries?**

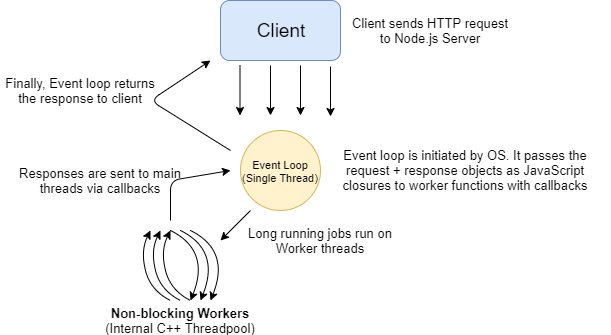
Npm install “library name”

To import external libraries in a Node.js application, you need to use the require function. The require function is a built-in function in Node.js that allows you to import modules or libraries into your code.

1. **Process Model**

The Node.js process model differs from traditional web servers in that Node.js runs in a single process with requests being processed on a single thread. One advantage of this is that Node.js requires far fewer resources. When a request comes in, it will be placed in an event queue. Node.js uses an event loop to listen for events to be raised for an asynchronous job. The event loop continuously runs, receiving requests from the event queue.

There are two scenarios that will occur depending on the nature of the request. If the request is non-blocking, it does not involve any long-running processes or data requests, the response will be immediately prepared and then sent back to the client. In the event the request is blocking, requiring I/O operations, the request will be sent to a worker thread pool. The request will have an associated call-back function that will fire when the request is finished and the worker thread can send the request to the event loop to be sent back to the client. In this way, when the single thread receives a blocking request, it hands it off so that the thread can process other requests in the meantime. In this way Node.js is inherently asynchronous.



The combination of the asynchronous nature of Node.js plus the reduced resource consumption of the single-threaded process leads to a significant increase in performance. It should be noted, however, that Node.js does not excel with CPU-intensive operations such as image processing and computationally-expensive work.

1. **What are the two types of API functions in Node.js?**

**Synchronous API Functions**: it blocks the execution of the program until it completes its task and returns the result. These functions wait for the operation to finish before proceeding to the next line of code.

Example:

const fs = require('fs');

const data = fs.readFileSync('file.txt', 'utf8');

console.log(data);

**It first reads the contents of the file.txt file and returns the data, which is then logged to the console.**

**Asynchronous API Functions:** it initiates the operation and immediately moves to the next line of code without waiting for the operation to complete. The result is obtained later through a callback function or by using promises or async/await.

Example

fs.readFile('file.txt', 'utf8', (err, data) => {

if (err) {

console.error(err);

return;

}

console.log(data);

});

**It reads the contents of the file.txt file, but instead of returning the data directly, it takes a callback function as an argument. The callback function is called once the operation completes, providing the error (if any) and the data.**

1. **What is the package.json file?**

It provides metadata about the project and its dependencies.

**Project Information**: The package.json file includes information about the project, such as its name, version, description, author, license, repository

**Dependencies:** lists the external libraries and packages required for the project to run.

**Dev Dependencies**: lists the dependencies that are only needed during the development process

**Scripts**: The scripts section contains custom scripts that can be executed using the NPM command line like start , build

1. **How would you use a URL module in Node.js?**

**url.parse()** function allows you to parse a URL string and extract its different components, such as protocol, host, pathname, query parameters, and more.

**url.format()** function enables you to construct a URL string from its individual components.

**url.resolve()** function helps in resolving a relative URL against a base URL.

1. **What is the Express.js package?**

Express.js is a popular web application framework for Node.js that simplifies the process of building robust and scalable web applications and APIs. It provides flexible approach to web development, allowing developers to handle routing, middleware, request/response handling, and more with ease.

* npm install expres
* const express = require('express');
* const app = express();

1. **How do you create a simple Express.js application?**

const express = require('express');

const app = express();

**// Define a route for the root URL**

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

res.send('Hello, World!');

});

**// Define a route for '/about'**

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

res.send('This is the About page');

});

**// Start the server**

app.listen(3000, () => {

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

});

1. **What are streams in Node.js?**

Streams in Node.js are a powerful feature that allow for efficient processing of data, especially when dealing with large amounts of data or when data is being received or sent in chunks. Streams provide an abstraction to handle data as a continuous flow, making it possible to read or write data piece by piece, rather than loading it all into memory at once.

1. Readable Streams: Readable streams are used for reading data from a source.
2. Writable Streams: Writable streams are used for writing data to a destination.
3. Duplex Streams: Duplex streams are both readable and writable. They represent a two-way flow of data
4. Transform Streams: Transform streams are a type of duplex stream that allow for data transformation
5. **How do you install, update, and delete a dependency?**

npm install express

npm update express

npm uninstall express

1. **How do you create a simple server in Node.js that returns Hello World?**

const http = require("http");

http.createServer(function (req, res) {

res.writeHead(200);

res.end("hello");

}).listen(8080);

1. **How do we implement async in Node.js?**

async function myFunction() {

try {

const result = await asyncFunction();

// Handle the result

} catch (error) {

// Handle the error

}

1. **What is a callback function in Node.js?**

asyncFunction((error, result) => {

if (error) {

// Handle the error

} else {

// Handle the result

}

});

1. **What is REPL in Node.js?**

REPL (Read-Eval-Print Loop) environment. REPL is an interactive programming environment that allows you to execute JavaScript code, line by line, and see the results immediately.

To access the Node.js REPL, you can open a command prompt or terminal and type node.

$ node

> console.log('Hello, World!');

Hello, World!

exit the Node.js REPL by typing .exit, pressing Ctrl + C twice, or pressing Ctrl + D.

1. **What is the control flow function?**

It is a function or a utility that helps manage the flow of asynchronous operations or control the execution order of code in an asynchronous environment.

It provides a way to handle multiple asynchronous tasks and ensure that they are executed in a specific sequence or with desired concurrency.

It address challenges related to asynchronous programming, such as nested callbacks or managing dependencies between asynchronous operations. They offer mechanisms for structuring and organizing asynchronous code, making it more readable, maintainable, and manageable.

**There are several popular control flow functions**

* passing **callbacks** to asynchronous functions, you can control the execution flow and specify what should happen once an operation completes.
* **Promises** provide a built-in control flow mechanism that represents the eventual completion (or failure) of an asynchronous operation.

.then() to specify what should happen next when a promise is fulfilled, and use .catch() to handle errors

* **Async/Await** : The async keyword is used to define an asynchronous function, and the await keyword is used to pause the execution of the function until a Promise is resolved.

1. **How do you open a file in Node.js?**

const fs = require('fs');

**// Open a file for reading**

fs.open('path/to/file.txt', 'r', (error, fileDescriptor) => {

if (error) {

console.error('Error opening the file:', error);

return;

**// Close the file after use**

fs.close(fileDescriptor, (error) => {

if (error) {

console.error('Error closing the file:', error);

}

});

});

1. **What are the different types of HTTP requests?**

**GET:** used to retrieve or fetch a resource from the server.

**POST:** used to submit data to the server to create a new resource.

**PATCH:** used to partially update an existing resource on the server.

**DELETE** is used to delete a specified resource on the server.

**HEAD:** similar to the GET method but retrieves only the response

headers, without the actual response body.

**OPTIONS**: used to retrieve the communication options available for a given resource.

1. **How would you connect a MongoDB database to Node.js?**

var mongo = require("mongo").mongo;

var url = "mongodb://0.0.0.0:27017";

mongo.connect(url, function (err, db) {

if (err) throw err;

var dbo = db.db("hello");

dbo.createCollectio("hi"),

function (err, res) {

if (err) throw err;

};

});

1. **What is a first-class function in Javascript?**

first-class functions refer to the concept where functions are treated as values and can be assigned to variables, passed as arguments to other functions, and returned as values from other functions.

**Functions can be assigned to variables**

const greet = function() {

console.log('Hello!');

};

**Functions can be passed as arguments**: This allows you to pass behavior or functionality to other functions dynamically. For example:

function execute(callback) {

callback();

}

execute(greet);

**Functions can be returned as values**:. return new functions based on certain conditions or parameters. For example:

function createGreeter() {

return function() {

console.log('Hello!'); };}

const greet = createGreeter();

greet();

**Functions can be stored in data structures:**

const functions = [greet, someOtherFunction, anotherFunction];

1. **How do you manage packages in your Node.Js project?**

you can manage packages and dependencies using npm (Node Package Manager). npm is the default package manager for Node.js

Initialization package.json file : **npm init**

Install packages: **npm install package-name**

Install multiple Packages: **npm install package1 package2 package3**

1. **What is MongoDB ?**
2. **What are some of the advantages of MongoDB?**
3. **What is a Document in MongoDB?**
4. **What is a Collection in MongoDB?**
5. **What are Databases in MongoDB?**
6. **What is the Mongo Shell?**
7. **How does Scale-Out occur in MongoDB?**
8. **How to add data in MongoDB?**
9. **How do you Update a Document?**
10. **How do you Delete a Document?**
11. **How to perform queries in MongoDB?**
12. **What are the data types in MongoDB?**
13. **When to use MongoDB?**
14. **How is Querying done in MongoDB?**
15. **Explain the term “Indexing” in MongoDB.**
16. **What is a Document in MongoDB?**
17. **What is a Collection in MongoDB?**
18. **What are Databases in MongoDB?**
19. **What is the Mongo Shell?**
20. **How does Scale-Out occur in MongoDB?**
21. **What are some features of MongoDB?**
22. **How to add data in MongoDB?**
23. **How do you Update a Document?**
24. **How do you Delete a Document?**
25. **How to perform queries in MongoDB?**
26. **What are the data types in MongoDB?**
27. **When to use MongoDB?**
28. **How is Querying done in MongoDB?**
29. **Explain the term “Indexing” in MongoDB.**
30. **What are some differences between BSON documents used in MongoDB and JSON documents in general?**

JSON (JavaScript Object Notation) offers several advantages as a data interchange format. Here are some of the key advantages of JSON:

**Human-Readable Format**: JSON is designed to be easy to read and understand by both humans and machines. Its syntax is clear and concise, making it simple to create and modify JSON documents manually.

**Wide Language Support**: JSON has native support in many programming languages. It can be parsed and generated easily, making it accessible for developers working in different programming languages. This allows for seamless data interchange between different systems and platforms.

**Lightweight and Efficient:** JSON has a compact structure, resulting in smaller file sizes and efficient data transfer over networks. Its simplicity and lack of unnecessary markup contribute to its lightweight nature.

**Supports Complex Data Structures**: JSON supports complex data structures such as nested objects and arrays. This flexibility allows for representing hierarchical data in a concise and organized manner.

**Well-Suited for Web Applications**: JSON aligns well with JavaScript, the widely used scripting language for web development. It can be directly parsed into JavaScript objects, making it easy to consume and manipulate data within web applications.

**Easy to Integrate**: JSON can be easily integrated with various APIs and web services. It is the preferred format for many APIs, enabling seamless communication and data exchange between systems.

**Platform Independence**: JSON is platform-independent, meaning it can be used across different operating systems and environments without compatibility issues. This makes it highly portable and versatile for data exchange.

**JSON Schema Validation**: JSON supports schema-based validation, where a JSON schema can define the structure, data types, and validation rules for JSON documents. This enables data validation and helps maintain data integrity when exchanging JSON data.

1. **What is the difference between the $all operator and the $in operator?**
2. **What are NoSQL databases? What are the different types of NoSQL databases?**
3. **How can you achieve Primary Key - Foreign Key relationships in MongoDB?**
4. **How do one perform the SQL JOIN equivalent in MongoDB?**
5. **How is data stored in MongoDB?**
6. **How to query MongoDB with %like%?**