



InterviewBit

Vue JS Interview Questions



To view the live version of the page, [click here](#).

© Copyright by Interviewbit

Contents

Vue js Interview Questions for Freshers

1. What is a Single-Page Application?
2. What are the key features of Vue.js?
3. What is a Vue instance? How can we create a Vue instance?
4. What is a component in Vue.js? How can we create a component?
5. What are mixins in Vue.js? Explain with an example. Also, its advantages and Disadvantages.
6. What is Virtual Dom in Vue.js?
7. Can you explain the life cycle hook in vue.js?
8. What are Hooks in Vue.js? Also, list various built-in Hooks.
9. What is Data Binding in Vue.js? And how many types of Data Binding are there in Vue.js?
10. How do you pass data between components in Vue.js?
11. What is the difference between one-way data flow and two-way data binding in Vue.js?
12. What is a filter in Vue.js? Provide an example.
13. Explain the difference between 'v-if' and 'v-show' in Vue.js?
14. What is a template in Vue.js? How does it differ from regular HTML?
15. What are Routers?
16. What is the difference between \$emit and \$on in Vue.js?
17. What are the global and local components in Vue.js?
18. What are custom key modifier aliases? How do you define it?

Vue Js Interview Questions for Experienced

Vue Js Interview Questions for Experienced

(.....Continued)

19. What are the lifecycle methods of Vue JS?
20. Explain Directives. Also, explain the purpose of directives in vue.js.
21. What is server-side rendering in Vue.js and how is it different from client-side rendering?
22. How would you optimize the performance of a large Vue.js application?
23. What is Vuex and when would you use it in a Vue.js application?
24. What is the difference between synchronous and asynchronous components in Vue.js?
25. What is the difference between a functional component and a regular component in Vue.js?
26. How would you integrate a third-party library into a Vue.js application?
27. How would you implement lazy-loading in a Vue.js application?
28. Can you tell the difference between Vue.js, React, and Angular?
29. How do you handle errors in Vue.js? Explain with an example.
30. How would you handle authentication and authorization in a Vue.js application?
31. What is a watcher in Vue.js? When would you use one?
32. What is the difference between mounted and created hooks in Vue.js?

Frequently Asked Questions

33. What is Vue js good for?
34. Is Vue js front end or back end?
35. How much does a Vue JS developer earn?
36. Does Vue JS have a future?

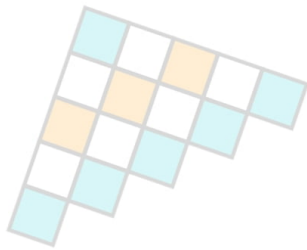
Frequently Asked Questions

(.....Continued)

- 37. What can Vue js be used for?
- 38. How to explain your projects in (Vue.js) interviews?

Tips to Crack Vue Js Interview

- 39. Interview Tips



InterviewBit

Let's get Started

Introduction

Vue.js is a popular JavaScript framework. It is used for building dynamic user interfaces and single-page applications. It was created by Evan You and has gained a significant following among developers due to its simplicity, flexibility, and performance. Since Vue.js continues to gain popularity, it is also becoming a popular topic in job interviews for front-end developer positions.

So, if you're preparing for a Vue.js job interview, it is very important to have a good understanding of the framework's concepts and features. To help you prepare for an interview, In this article, we have provided a list of common **Vue.js interview questions and answers**, along with tips on how to answer them. These questions cover a wide range of topics, from the basics of Vue.js to more advanced concepts such as component communication and Vuex state management. By studying these questions and practising your responses, you'll be better prepared to showcase your Vue.js skills during your job interview.

What is Vue.js?

Vue.js is an advanced JavaScript framework used for building user interfaces and single-page applications. One of the key features of Vue.js is its reactivity system, which enables developers to build dynamic and responsive user interfaces with ease. It also uses a template-based syntax which helps developers to declaratively render the UI component based on the application state. Vue.js has taken inspiration from **MVVM (Model - View - ViewModel)** Pattern and uses this to build a high-quality single-page application.

Vue.js is designed to be incrementally adoptable, which means that we can use as much or as little of the framework needed in their projects. This flexibility makes Vue.js a great choice for both small and large-scale projects.

Vue js Interview Questions for Freshers

1. What is a Single-Page Application?

A Single Page Application (SPA) is a lightweight application that functions within a single web page, with no full-page reloading as the user interacts with the application. As the user consumes SPA with interactions or requests, the DOM's page content is dynamically updated by sending or executing said requirements asynchronously.

2. What are the key features of Vue.js?

Vue.js have several features, some of the key features of Vue.js are:

- **Declarative Rendering:** Vue.js is a template-based syntax that allows rendering the DOM declaratively based on the underlying data model. This makes it easier in reasoning about the application and helps in manipulating the state of the application.
- **Two-way Data Binding:** Vue.js makes it simpler to synchronize the data model with the DOM. This means that changes to the data model are reflected in the display automatically, and vice versa.
- **Component-based Architecture:** Vue.js promotes the creation of user interfaces as a tree of reusable components, each of the components encapsulates its state and behaviour. This makes it simple to handle complicated user interfaces and reuse code.
- **Directives:** Vue.js includes a set of built-in directives that allow to bind data to DOM elements declaratively, alter the DOM, and respond to events.
- **Reactive:** Vue.js automatically tracks data property dependencies and modifies the DOM accordingly. This simplifies the development of reactive, data-driven applications.
- **Virtual DOM:** When the state of the application changes, Vue.js employs a virtual DOM to efficiently update the actual DOM. This reduces the number of DOM modifications, and results in improved performance.
- **Lightweight:** Vue.js is a lightweight framework with a minimal footprint and quick execution. As a result, it is suitable for developing high-performance online applications.

3. What is a Vue instance? How can we create a Vue instance?

The Vue instance is the root of your application. It is a Vue class instance that represents the ViewModel in the Model-View-ViewModel (MVVM) architecture paradigm. The Vue instance is responsible for generating and managing the component hierarchy, as well as handling data, state management, event, and lifecycle hook.

To create a Vue instance, we simply create a new instance of the Vue class and pass in an options object that defines the properties and behavior of the instance.

Consider below the example of creating a basic Vue instance:

```
var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue!'
  }
})
```

In this example, we have created a new Vue instance and passed in an options object with two properties:

- **‘el’**: specifies the DOM element that the Vue instance should be mounted to.
- **‘data’**: specifies the data object that contains the application's state.

Once the Vue instance is created, it will take over the specified DOM element and replace its contents with the template and data bindings defined in the Vue instance.

4. What is a component in Vue.js? How can we create a component?

In Vue.js, a component is a reusable and modular piece of code that encapsulates the template, style, and behaviour of a particular feature or UI element in the application. A component can be composed of other components and can communicate with other components through props and events.

To create a component in **‘Vue.js’**, we can use the **‘Vue.component()’** method or define it as an object with a template, data, props, methods, computed, watch and other options.

Consider the below code to understand this:

```
Vue.component('my-component', {
  template: '<div>{{ message }}</div>',
  data: function () {
    return {
      message: 'Hello from my component!'
    }
  }
})
```


In this example, we define a component called '**my-component**' and specify its template and data options. The template option defines the HTML markup that should be rendered when the component is used, and the data option defines the initial state of the component's data.

To use this component, we can reference the component tag name at the place where we want to use it. **Example** - We can add my component as a tag. Consider the below snippet.

```
<div id="app">
  <my-component></my-component>
</div>
```

5. What are mixins in Vue.js? Explain with an example. Also, its advantages and Disadvantages.

In Vue.js, mixins are a way to reuse code across multiple components. A mixin is essentially an object with component options that can be merged with the options of other components. When a component uses a mixin, it inherits all the properties and methods defined in the mixin.

Consider the below code to understand this better -

```
// Define a mixin with a common method
const greetingMixin = {
  methods: {
    greet() {
      console.log("Hello, world!");
    }
  }
};

// Define a component that uses the mixin
Vue.component("my-component", {
  mixins: [greetingMixin],
  template: `
    <div>
      <h1>My Component</h1>
      <button @click="greet()">Greet</button>
    </div>
  `
});

// Create a Vue instance with the component
new Vue({
  el: "#app"
});
```

In the above code, we have created a mixin with the name - '**greetingMixin**'. And in the component, we are consuming it with (**mixin : []**). Similarly, it can be utilized at multiple components reducing the code repeatability.

Mixins have certain advantages, but it is also having certain disadvantages. Some of them include the following:

Advantages	Disadvantages
It Promotes code reusability.	Naming conflicts can occur.
It reduces code duplication.	It can make it harder to understand the behavior of a component.
It helps keep code organized and modular.	It can lead to unexpected behavior if used improperly.
It can make it easier to maintain and update code.	It can increase code complexity if too many mixins are used.
It can be used to extend and customize existing functionality.	It may not be the best approach for all use cases.

6. What is Virtual Dom in Vue.js?

The Virtual DOM (VDOM) in Vue.js is a lightweight in-memory representation of the actual Document Object Model (DOM) of a web page. When a Vue.js component is created or updated, it creates a VDOM representation of its current state, which is then compared to the previous VDOM representation to determine the minimum set of changes that need to be made to the actual DOM to reflect the updated state.

The VDOM is used as an optimization technique to minimize the number of direct manipulations of the DOM, which can be expensive in terms of performance. Instead of updating the actual DOM every time a component's state changes, Vue.js uses the VDOM to calculate the most efficient way to update the DOM and then applies those changes in a batched manner. This can result in significant performance improvements, particularly in large and complex applications.

7. Can you explain the life cycle hook in vue.js?

The lifecycle hooks in Vue.js are special methods that are called at various stages of a component's lifecycle. These hooks allow the execution of code at specific points in the component's lifecycle, such as before the component is created before it is mounted, after it is updated, and before it is destroyed.

- **beforeCreate:** This hook is called before the component instance is created. It is useful for initializing data and setting up event listeners.
- **created:** This hook is called after the component instance is created. It is useful for performing initial setups such as fetching data from an API.
- **beforeMount:** This hook is called before the component is mounted to the DOM. It is useful for performing any necessary DOM manipulations before the component is rendered.
- **mounted:** This hook is called after the component is mounted to the DOM. It is useful for performing any necessary DOM manipulations after the component is rendered.
- **beforeUpdate:** This hook is called before the component is updated due to changes in its props or state. It is useful for performing any necessary pre-update tasks.
- **updated:** This hook is called after the component is updated due to changes in its props or state. It is useful for performing any necessary post-update tasks.
- **beforeUnmount:** This hook is called before the component is unmounted from the DOM. It is useful for performing any necessary cleanup tasks before the component is removed from the DOM.
- **unmounted:** This hook is called after the component is unmounted from the DOM. It is useful for performing any necessary cleanup tasks after the component is removed from the DOM.

By using these lifecycle hooks in your Vue.js components, you can execute code at specific points in the component's lifecycle, allowing you to perform initialization, cleanup, and other tasks as needed.

8. What are Hooks in Vue.js? Also, list various built-in Hooks.

Hooks are a new feature introduced in **Vue.js 3**. This allows adding logic and state management into components in a more organized and reusable way. Hooks are similar to lifecycle hooks, but they allow for encapsulation and reuse of logic across multiple components.

Vue.js provides several built-in hooks that we can use to add functionality to the components. Some of the most commonly used hooks include:

- **'setup'**: This is the main hook that you use to add logic and state management to the component. The setup function is called before the component is created, and it allows the definition of reactive data, methods, computed properties, and other logic that can be accessed by the component's template.
- **'onMounted'**: This hook is called after the component is mounted to the DOM. We can use this hook to perform any necessary DOM manipulations or data fetching operations.
- **'onUpdated'**: This hook is called after the component is updated due to changes in its props or state. We can use this hook to perform any necessary post-update tasks.
- **'onUnmounted'**: This hook is called before the component is unmounted from the DOM. We can use this hook to perform any necessary cleanup tasks before the component is removed from the DOM.

9. What is Data Binding in Vue.js? And how many types of Data Binding are there in Vue.js?

Data binding is the process of synchronizing data between the Vue instance and the DOM. It allows us to establish a connection between the data and the UI, so that any changes to the data are automatically reflected in the UI, and any changes to the UI are automatically reflected in the data.

There are three types of data binding in Vue.js:

1. **Interpolation:** This is a one-way binding that allows us to embed the value of a data property into the content of an HTML element. It is denoted by double curly braces ‘{{ }}’. For example, ‘{{ message }}’ would bind the value of the message data property to the content of the HTML element.
2. **Property binding:** This is a one-way binding that allows us to set the value of an HTML element's attribute to the value of a data property. It is denoted by the ‘**v-bind**’ directive. For example, (v-bind:href="url") would bind the value of the ‘url’ data property to the ‘href’ attribute of an HTML element.
3. **Two-way binding:** This allows us to bind the value of a form input element to a data property, so that changes to the input elements are reflected in the data, and vice versa. It is denoted by the ‘v-model’ directive. For example, (v-model="message") would bind the value of the ‘message’ data property to the value of a form input element.

10. How do you pass data between components in Vue.js?

We can pass data between components in the following ways-

- **Props:** It is like an argument that is given to the component. The term used to define this is an attribute. This is registered on the component and it allows to pass of the data from the parent component to the child component. To pass data through props, we can define the prop on the child component, and then bind the data to the prop on the parent component.

For example:

```
<!-- Parent component template -->
<template>
  <child-component :message="parentMessage"></child-component>
</template>

<!-- Child component script -->
<script>
export default {
  props: {
    message: String
  }
}
</script>
```

- **Event bus:** An event bus is a Vue instance that you can use to emit and listen to events across different components. To pass data using an event bus, we can simply emit an event with the data from one component, and then listen for the event and receive the data in another component.

For example:

```
// Event bus instance
export const bus = new Vue()

// Emitting an event with data in Component A
bus.$emit('my-event', data)

// Listening for the event and receiving data in Component B
bus.$on('my-event', data => {
  // Do something with the data
})
```

- **Vuex store:** Vuex is a state management pattern and library for Vue.js applications. It allows us to define a centralized store for managing the state of your application and provides a way to access and mutate the state from different components. To pass data using a Vuex store, we can simply define a state property and mutation method in the store, and then dispatch the mutation with the data from one component, and then access the state property in another component.

For example:


```
// Store instance with state property and mutation
export const store = new Vuex.Store({
  state: {
    message: ''
  },
  mutations: {
    setMessage (state, payload) {
      state.message = payload
    }
  }
})

// Dispatching the mutation with data in Component A
store.commit('setMessage', data)

// Accessing the state property in Component B
this.message = this.$store.state.message
```

11. What is the difference between one-way data flow and two-way data binding in Vue.js?

	One-way data flow	Two-way data binding
Definition	Data flows only from parent to child components.	Data flows between parent and child components in both directions.
Syntax	Data is passed down from parent to child components through props.	Data is bound to form input elements using the 'v-model' directive.
Example	<pre><child-component :message="parentMessage"> </child-component></pre>	<pre><input v- model="message"></pre>
Benefits	<ol style="list-style-type: none"> 1. Simpler data flow, easier to track data changes. 2. Improved performance due to unidirectional data flow. 	<ol style="list-style-type: none"> 1. Easier to handle user input and form data. 2. Reduces the amount of code needed to handle data synchronization.
Drawbacks	<ol style="list-style-type: none"> 1. More complex to implement a two-way data flow. 2. Can make it harder to track data changes. 	<ol style="list-style-type: none"> 1. Can make it harder to track data changes. 2. More complex data flow can reduce performance.

12. What is a filter in Vue.js? Provide an example.

In Vue.js, filters are functions that can be used to transform data in a template expression. Filters allow us to apply a specific formatting or transformation to a piece of data before it is displayed in the user interface.

Filters are defined as functions that take a value as their input, perform some transformation on that value, and then return the transformed value.

Filters can be added to a Vue.js application using the **Vue.filter()** method, and can then be used in template expressions using the **|** character.

Example -

```
Vue.filter('reverse', function(value) {  
  // Reverse the characters in the string  
  return value.split('').reverse().join('')  
})  
  
new Vue({  
  el: '#app',  
  data: {  
    message: 'Hello, world!',  
  },  
})
```

In this example, we have defined a reverse filter that takes a string as its input, then it reverses the characters in the string, and then returns the reversed string. We have then created a new Vue instance, and then set the message data property to 'Hello, world!'.

To use the reverse filter in a template expression, we can simply pipe the message property through the filter using the **|** character. Consider the below snippet to understand it.

```
<div id="app">  
  <p>{{ message | reverse }}</p>  
</div>
```

13. Explain the difference between 'v-if' and 'v-show' in Vue.js?

‘v-if’ and ‘v-show’ are both directives that allow conditional rendering of elements in the user interface. However, they differ in how they work and when they should be used. The differences are:

Feature	v-if	v-show
Initial render	Element is not included in the DOM.	Element is included in the DOM but hidden.
Re-render	Element is added or removed from the DOM.	Element's display style property is toggled.
Performance	More efficient for elements that are rarely used.	More efficient for elements that are frequently used.
Conditional logic	It can be used with v-else and v-else-if.	It cannot be used with v-else or v-else-if.
Use case	It is best for elements that are rarely shown or hidden.	It is best for elements that are frequently shown or hidden.

14. What is a template in Vue.js? How does it differ from regular HTML?

A template is a piece of HTML that defines the structure and layout of a component in a Vue.js application. These are very similar to regular HTML code, but they also include Vue-specific syntax and directives. This allows binding data to the user interface, handling user input, and conditionally rendering content based on data and user interactions.

There are differences between a Vue.js template and regular HTML. The differences are -

- **Vue-specific syntax:** Templates include special syntax and directives. This allows binding data to the user interface and performing other dynamic operations. **For example** - We can use double curly braces (`{{ }}`) to output data values. Also, we can use `v-bind` to bind HTML attributes to data properties.
- **Reactivity:** Templates are reactive. It means that the template can automatically update in response for changing in data and user interactions. When the data changes in a Vue.js application, the corresponding templates are automatically re-renders to reflect the new state of the application.
- **Directives:** Vue.js templates include several built-in directives that allow us to perform complex operations. **For Example** - We can render data values conditionally using (`v-if`, `v-else`, `v-show`), we can easily loop over arrays and objects using (`v-for`), and handle user input using (`v-on`).
- **Components:** Templates in vue.js can also be used to define the components. Components can be reused and can compose UI elements that can be utilized in the entire application. Components are defined using a combination of a template, script, and style, and can be nested inside other components to create complex UI structures.

15. What are Routers?

The Vue.js router is a library that allows the implementation of client-side routing in Vue.js applications. Here routing means the process of mapping URLs to specific components or views in the application. Routing allows users to navigate between different pages or views without refreshing a full page reload. So when the entire page doesn't refresh it feels like using a mobile / desktop application. Since the entire page didn't get reloaded, it improves the performance also.

16. What is the difference between \$emit and \$on in Vue.js?



\$emit	\$on
It is called on a child component to emit an event.	It is called on a parent component to listen for an event.
It can pass data as an optional second parameter.	It does not take any parameters.
It can emit events with custom names.	It listens for events with a specific name.
It can be used to communicate from child to parent components.	It can be used to communicate from child to parent or between sibling components.
This is typically used in the child component's methods section.	This is typically used in the parent component's created or mounted hook.
It is used to trigger an action in the parent component in response to a user interaction or data change in the child component.	It is used to respond to events emitted by child components and update the parent component's state or trigger other actions.
Example: <pre>this.\$emit('button-clicked', 'Hello from the child component')</pre>	Example: <pre>this.\$on('button-clicked', this.onButtonClicked)</pre>

17. What are the global and local components in Vue.js?

Components are the fundamental building blocks of an application. This contains a piece of HTML, CSS, and JavaScript code and is encapsulated into a single function and shares this code across different parts of the application. By using components, we can avoid code duplication, And we can reuse the component as many times as it is required in the application. In vue.js, components are of two types -

- **Global components** - Global components are defined at the top and can be used anywhere in the application. Mostly, we define global components in the **main.js file** or in a separate file and then import them into the application. We can register global components using the **Vue.component()** method.

Example-

```
// Defining a global component
Vue.component('global-component', {
  template: `
    <div>
      <h1>Global Component</h1>
      <p>This is a global component.</p>
    </div>
  `
})

// Using the global component in a Vue instance
new Vue({
  el: '#app',
  template: `
    <div>
      <global-component></global-component>
    </div>
  `
})
```

- **Local components**, on the other hand, are defined within a specific component and can only be used within that component or its child components. They are registered using the components option of the parent component.

Example-


```
// Define a local component
const LocalComponent = {
  template: `
    <div>
      <h1>Local Component</h1>
      <p>This is a local component.</p>
    </div>
  `
}

// Use the local component in a parent component
Vue.component('parent-component', {
  components: {
    'local-component': LocalComponent
  },
  template: `
    <div>
      <local-component></local-component>
    </div>
  `
})
```

In this example, LocalComponent is a local component that is defined within the parent component. It can only be used within parent-component or child components.

18. What are custom key modifier aliases? How do you define it?

Key modifiers are used to handle keyboard events. They allow you to listen to specific key events and perform actions in response. By default, Vue.js provides some key modifiers such as **(.enter, .tab, .delete, etc)**. However, you can define your own custom key modifier aliases using the **Vue.config.keyCodes** object.

Example for defining a custom key modifier alias:

```
Vue.config.keyCodes.f2 = 113; // define a key modifier alias for the F2 key
```

In this example, we are defining a new key modifier alias for the F2 key with code 113. Once the custom key modifier alias has been defined, you can use it in your Vue.js templates like any other key modifier.

Vue Js Interview Questions for Experienced

19. What are the lifecycle methods of Vue JS?

VueJS has several lifecycle methods that allow developers to handle certain events that occur during the lifecycle of a Vue component. The lifecycle methods of VueJS can be divided into three categories:

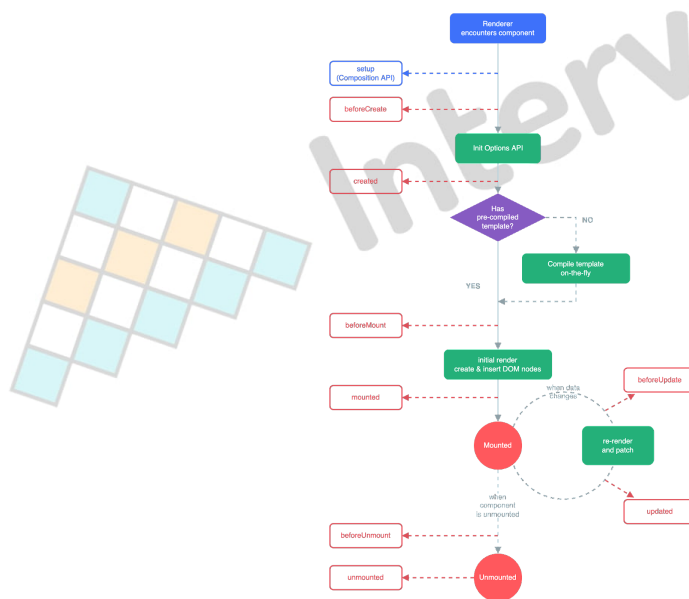


Image Reference: Github

1. **Creation:** These methods are called when components are created for the first time.
 - **beforeCreate():** This method is called before a component is created, and data has not yet been made reactive.
 - **created():** This method is called after a component has been created, and data has been made reactive.
2. **Mounting:** Before a component gets inserted into the DOM, various mounting methods come into play.
 - **beforeMount():** Prior to the component's mounting to the DOM, the beforeMount() method is invoked
 - **mounted():** method is triggered post-component mounting to the DOM.
3. **Updating:** Upon the modification of a component's reactive data, the aforementioned approaches get invoked.
 - **beforeUpdate():** Prior to the update of a component's reactive data, this method is invoked.
 - **updated():** Once the reactive data of a component has been updated, this method is invoked.

Two additional lifecycle methods are called when a component is destroyed:

- **beforeDestroy():** When the components are about to destroy, this method is called.
- **destroyed():** When the components are destroyed this method is invoked.

20. Explain Directives. Also, explain the purpose of directives in vue.js.

Directives are special attributes that can be applied to DOM elements. The 'v-' prefix easily distinguishes these utilitarian properties and allows elements to demonstrate reactive and declarative behaviour ascribed to them.

Vue.js directives are used to apply common DOM transformations and data bindings declaratively, removing the need for specialized JavaScript code. They make it simple to apply behaviour to DOM components, improving code readability, understanding, and maintainability. Directive customization and extension can provide give additional functionality.

Some common directives in Vue.js include:

- **‘v-if and v-show’:** It is used for conditionally rendering the elements based on a Boolean expression.
- **‘v-for:’** It is used for looping through an array or object and rendering a template for each item.
- **‘v-bind’:** It is used for dynamically binding the data to an element's attributes, properties, or styles.
- **‘v-on’:** It is used for attaching the event listeners to DOM elements and call methods when the event is triggered.
- **‘v-model’:** It is used for creating two-way data bindings between form input elements and data property.

Consider the below code -

```
<template>
  <div>
    <div v-if="showText">This text will only be displayed if showText is true.</div>

    <div v-show="showText">This text will be hidden if showText is false.</div>

    <ul>
      <li v-for="(item, index) in items" :key="index">{{ item }}</li>
    </ul>

    <div v-bind:class="{ 'active': isActive }">This element will have the 'active' class</div>

    <button v-on:click="doSomething">Click me!</button>

    <input v-model="message" type="text">
    <p>You typed: {{ message }}</p>
  </div>
</template>

<script>
export default {
  data() {
    return {
      showText: true,
      items: ['apple', 'banana', 'orange'],
      isActive: false,
      message: ''
    }
  },
  methods: {
    doSomething() {
      // Code to execute when the button is clicked
    }
  }
};
</script>
```

21. What is server-side rendering in Vue.js and how is it different from client-side rendering?

Server-side rendering (SSR) is a web dev trick that involves generating HTML content on the server and shipping it off to the client as a fully-formed HTML page. Vue.js backs up SSR and can be used along with client-side rendering to whip up all-purpose apps.

- In the rendering on the client side, the browser snags the basic webpage codes and designs and kicks off the page view on the user side. And when the user fiddles with the page, the side handled by the user reaches out to get fresh data from the server and tweaks the core HTML tags as per the requirements. Going this way has advantages - like cracking open the page in a dash, making some great interactive changes, and making the job of coders a whole lot easier. Yet, it also has disadvantages - like when it comes to SEO, there are going to be some issues, and the page's time-to-content is going to drag a bit.
- On the other side, SSR does the HTML content baking on the server end and sends it out to the client as a full HTML page. It takes more time for the page to first show up, but once it does, it's already functioning perfectly. Besides that, SSR has some advantages over-client-side renderings like better search engine optimization, quicker content loading time, and more stable performance on weak devices.

To view SSR in Vue.js, we have to use Node.js-based server to spit out the first HTML content. When a user asks for a page, the code on the server side creates the HTML content using Vue.js goodies and sends it to the player. Once the page is ready, the client-side code takes the snippets and updates the DOM as needed. This provides a seamless user experience.

22. How would you optimize the performance of a large Vue.js application?

Optimizing the performance of a large Vue.js application requires several solutions that address different areas of the application. Although adopting specific optimization procedures, combining techniques has proven to deliver the highest return. Consider the following strategies: -

- **Lazy Loading:** Instead of loading everything at once, we can divide the application into smaller chunks and load them on demand using lazy loading. This split-up approach can significantly reduce initial load time and enhance overall application performance.
- **Code Splitting:** We can split the code into smaller chunks. And can load only the required component. Using this we can reduce the size of the application and can improve its performance.
- **Minification and Compression:** We can squeeze the app size and use the minified version of Javascript and CSS files (like jquery.min.js). This speeds up downloads and boosts performance.
- **Caching:** We can cache stuff like pics, fonts, and API responses so we don't put the load on the network as much and make our app run faster.
- **Use Vuex for State Management:** Vue.js provides Vuex (a state management library). This allows management of the state of the application in a centralized store. Using Vuex, we can cut down the number of API calls and make our Vue.js application faster.
- **Use Async Components:** Async components allows to load of components on demand. So using this we can reduce the unnecessary calls/request of components at the initial time.
- **Optimize Vue.js Directives:** We can use v-if instead of v-show for conditionally rendering elements. Also, we can use v-for with the key attribute, and use v-cloak to hide uncompiled Vue.js templates.
- **Server-Side Rendering:** Using Server-Side-Rendering, we can preload some of the necessary HTML, CSS, and JavaScript files. This can improve performance in terms of SEO.
- **Use Performance Analysis Tools:** There are various performance analysis tools that we can use like - Google Chrome Developer tools, Vue.js Developer tools, etc. This helps in debugging the complexity so that we can reduce it and optimize the performance.

23. What is Vuex and when would you use it in a Vue.js application?

Vuex is a state management pattern and library for Vue.js applications. It provides a centralized store to manage the state of the application and allows synchronization of data between components and predictably manages complex stateful logic.

Here are some common use cases for using Vuex in a Vue.js application:

- **Large Applications:** Vuex can be useful in large applications with many components that need to share data or communicate with each other. By centralizing the state management in a Vuex store, we can avoid passing props and events between components and simplify the code.
- **Complex State Management:** Vuex can help manage complex stateful logic by providing a clear separation of concerns between the state management code and the view components. This can help improve the maintainability and readability of the code.
- **Debugging:** Vuex provides a centralized store that makes it easier to debug the state of the application. By using the Vuex DevTools, we can easily monitor the state changes and debug issues related to state management.
- **Time Travel:** Vuex supports time travel debugging, which allows us to step forward and backwards through the state changes of the application. This can be useful for debugging complex stateful logic or reproducing issues in the application.
- **Server-Side Rendering:** Vuex can be used in conjunction with server-side rendering to provide a consistent state between the client and server. This can help improve the performance and SEO of the application.

24. What is the difference between synchronous and asynchronous components in Vue.js?

	Synchronous Components	Asynchronous Components
Loading Time	Loaded during app initialization.	Loaded only when needed.
Import Statement	'import' statement in the main file.	'import' statement in the parent component.
Bundle Size	Increases the initial bundle size.	Reduces the initial bundle size.
Performance Impact	Can impact the initial load time and performance of the app.	Improves the initial load time and performance of the app.
Component Definition	Defined synchronously in the parent component.	Defined asynchronously using a factory function.
Usage in Templates	Can be used directly in templates.	Must be wrapped in a <component> tag with is an attribute.
Dynamic Component	Not suited for dynamically rendering components.	Suited for dynamically rendering components.
Code Splitting	Not possible to code split.	Can be easily code split.

25. What is the difference between a functional component and a regular component in Vue.js?

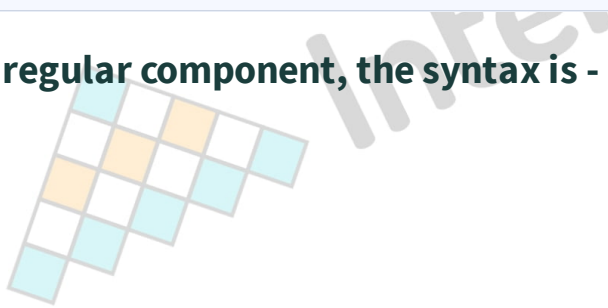
	Functional Components	Regular Components
Component Type	Stateless and simpler.	Stateful and complex.
Instance Lifecycle Hooks	No instance creation or hooks.	All instance hooks are available.
Render Function	Defined as a single render function using the 'functional' option or shorthand syntax.	Defined with a template or render function.
Data	No data or methods.	Can have data and methods.
Performance	Better performance with less overhead.	Lower performance overhead.
Communication with Parents	Must use props and emit events.	Can use props, events, and parent methods.

To Create a functional component, the syntax is -

```
// Using the `functional` option
Vue.component('my-functional-component', {
  functional: true,
  render: function (createElement, context) {
    // Render function logic here
  }
})

// Using the shorthand syntax
export default {
  functional: true,
  render: (h, context) => {
    // Render function logic here
  }
}
```

To Create a regular component, the syntax is -



```
// Using a template
Vue.component('my-regular-component', {
  template: `
    <div>
      <!-- Template markup here -->
    </div>
  `,
  data() {
    return {
      // Data properties here
    }
  },
  methods: {
    // Methods here
  },
  // Other instance options and hooks here
})

// Using a render function
Vue.component('my-regular-component', {
  render: function (createElement) {
    // Render function logic here
  },
  data() {
    return {
      // Data properties here
    }
  },
  methods: {
    // Methods here
  },
  // Other instance options and hooks here
})
```

26. How would you integrate a third-party library into a Vue.js application?

To integrate a third-party library into a Vue.js application, we can follow the general steps:

1. **Install the library:** We can use a package manager like npm or yarn to install the library as a dependency.
2. **Import the library:** Import the library in the Vue.js application's entry point (e.g., main.js) using the appropriate syntax for the library. This may involve importing the library as a module, loading a script file from a CDN, or using a special plugin for the library.
3. **Use the library in your components:** Depending on the library, we may need to configure it or initialize it before we can start using it in our Vue.js components. Then, we can use the library's APIs and functions in component methods, computed properties, or lifecycle hooks.

27. How would you implement lazy-loading in a Vue.js application?

Lazy loading is basically dividing the code. We can do this cool trick with Webpack, which is just a fancy way of bundling stuff for Vue CLI projects. Here are the general steps to implement lazy-loading in a Vue.js application:

- **Configure webpack:** In the vue.config.js file of your project, you need to set up webpack so that it chops your app into smaller sections by utilizing splitChunks or **dynamicImport**.
- **Define a route-based chunk:** In the Vue router, we can define some piece of code for a specific route by using the component property and a function. This will return an import statement dynamically for the component.
- **Use the lazy-loaded component:** In your Vue templates, use the component element and bind the attribute to the lazy-loaded component name.

Consider the below example for the implementation of this -

```
<!-- Template -->
<template>
  <div>
    <!-- Use v-lazy directive to lazy load the image -->
    <img v-lazy="imageSrc" alt="Image">
  </div>
</template>

<script>
import Vue from 'vue';
import VueLazyload from 'vue-lazyload';

export default {
  data() {
    return {
      imageSrc: 'path/to/image.jpg'
    }
  },

  mounted() {
    // Register the VueLazyload plugin with Vue
    Vue.use(VueLazyload, {
      // The factor by which to preload images, relative to the viewport height
      preload: 1.3,
      // The image to display when there is an error loading the image
      error: 'path/to/error.png',
      // The image to display while the lazy image is loading
      loading: 'path/to/loading.gif',
      // The number of times to attempt loading the image before giving up
      attempt: 1
    });
  }
}
</script>
```

28. Can you tell the difference between Vue.js, React, and Angular?

	Vue.js	React	Angular
Language	It uses javascript.	It also uses javascript.	It uses typescript, a superset of JavaScript that adds features like static typing.
Template	It uses HTML-Based templates.	It uses JSX. JSX allows writing HTML Codes inside javascript.	It also uses HTML-Based templates.
Size	It is a small and compact framework. Size is less than 30KB.	It is also a small and compact framework. Size is less than 30KB.	Angular is a larger framework, with a size of more than 500 KB.

29. How do you handle errors in Vue.js? Explain with an example.

There are several ways to handle errors in vue.js depending on the context and the type of error. Here are some common approaches:

- **Error Handling in Components:**

To handle errors in Vue components, we can use the **try-catch block**. If an error occurs in the component, it will be caught by the catch block, and you can take appropriate action to handle the error.

For example, you can display an error message to the user, log the error, or even send it to a server for further analysis. Consider the below example code.

```
<template>
  <div>
    <button @click="handleClick">Click Me</button>
  </div>
</template>

<script>
export default {
  methods: {
    handleClick() {
      try {
        // Code that may cause an error
      } catch (error) {
        // Handle the error
      }
    }
  }
}
</script>
```

- **Global Error Handling:**

To handle errors globally, we can use the **errorHandler** property of the Vue configuration object. This allows us to catch and handle errors that occur anywhere in your application. Consider the below example code to understand better.


```
import Vue from 'vue'
Vue.config.errorHandler = function (error, vm, info) {
  // Handle the error
}
```

- **Error Handling in Promises:**

For using the Promises in the Vue.js application, we can use the catch method to handle errors that occur during asynchronous operations. Consider the below example code that demonstrates how we can use it.

```
this.$http.get('/api/data')
  .then(response => {
    // Handle the response
  })
  .catch(error => {
    // Handle the error
  })
```

30. How would you handle authentication and authorization in a Vue.js application?

Authentication and authorization are critical aspects of web application development, and Vue.js provides several approaches to handle them. Some commonly used methods are:

- **Authentication:** To handle authentication in a Vue.js application, we can use an authentication library like **JWT** or **OAuth2.0**. Once the user logs in, the authentication library will generate a token that you can store in the browser's local storage or cookies. We can then include this token in all subsequent requests to the server to verify the user's identity.
 - **For example** - we can use the vue-authenticate library to implement OAuth2.0 authentication in the Vue.js application. Once the user logs in using their credentials, the library will retrieve an access token that we can use to authenticate subsequent requests.
- **Authorization:** To handle authorization in a Vue.js application, we can implement **role-based access control (RBAC)** or **attribute-based access control (ABAC)**. With RBAC, we can define the roles that correspond to different levels of access, and we assign these roles to users. With ABAC, we can define policies that specify which users are authorized to perform specific actions.
 - **For example** - we can use the vue-acl library to implement RBAC in the Vue.js application. This library provides a middleware that we can use to restrict access to specific routes or components based on the user's role.

31. What is a watcher in Vue.js? When would you use one?

In Vue.js, a watcher is a special object that allows us to watch for changes in a specific data property, and perform some action when that property changes. Watchers are a key part of Vue.js's reactivity system, which automatically triggers the event and updates the view when data changes.

Watchers are particularly useful when we need to perform some action in response to changes in data that cannot be accomplished with computed properties or methods.

For example, We might use a watcher to update a chart or graph in response to changes in a data source. Or maybe to trigger an API call when a specific data property changes.

32. What is the difference between mounted and created hooks in Vue.js?

Throughout their lifecycle, components experience numerous stages, where both created and mounted hooks function as tools to execute relevant actions. Although there is some difference between them. Those are -

When a component is made, the **"created"** hook is instantly called and provides access to the component's data for tweaking. This hook can be utilized for organizational tasks such as configuring methods, data, and events that are necessary for the overall functionality of the component

As soon as the component's template has been compiled, rendered, and inserted into the Document Object Model (DOM), the **"Mounted"** hook comes into action. This hook is essential for executing any tasks that require the usage of DOM. This could involve setting up event listeners or kickstarting third-party libraries.

Conclusion

Vue.js is a popular JavaScript framework that is widely used for building complex, dynamic web applications. Whether you are a seasoned developer or just starting with Vue.js, it's essential to be well-prepared for a Vue.js job interview. This article covered some common Vue.js interview questions that you may encounter, including questions about Vue.js fundamentals, component architecture, routing, state management, and error handling. By studying and practising these questions, you can increase your chances of success and impress your interviewers with your Vue.js knowledge and skills. Remember to keep learning and staying up-to-date with the latest Vue.js features and best practices to become a proficient Vue.js developer. Certainly, here are some tips and tricks that you can follow for answering Vue.js interview questions.

Frequently Asked Questions

33. What is Vue js good for?

Vue.js is good for building user interfaces and complex single-page applications with ease. Thanks to its easy-to-learn syntax, reactivity system, and component-based architecture. It is also lightweight, fast, and flexible, making it an excellent choice for developers who want to create responsive and interactive applications.

34. Is Vue js front end or back end?

Vue js is a front-end javascript library used for creating a lightweight single-page web application. It can communicate with APIs and provides a rich user experience.

35. How much does a Vue JS developer earn?

The average salary for a Vue.js developer in India is around ₹600,000 per year, according to Glassdoor. However, salaries can range from around ₹350,000 to over ₹1,500,000 per year, depending on factors such as the developer's experience level, location, company size, and industry.

36. Does Vue JS have a future?

Yes, We can conclude that Vue.js has a bright future due to its simplicity, flexibility, and performance. Its reactivity system makes it easy to build complex user interfaces with minimal code. Additionally, its growing community and constant updates ensure it will remain a popular choice for front-end development.

37. What can Vue js be used for?

Vue.js can be used to build complex single-page applications, dynamic user interfaces, and reusable components. It can also be used for server-side rendering, mobile app development, and desktop app development. In general, Vue.js is a versatile and flexible framework that can be used for a wide range of web development projects.

38. How to explain your projects in (Vue.js) interviews?

You can start by providing a brief introduction to Vue.js and its core features and how it's relevant to your project. This will help the interviewer understand the context of your project. Other than that, consider some below points that you should keep in mind while answering -

- **Keep it simple:** Avoid getting into unnecessary details and keep your explanation simple and easy to understand.
- **Understand the basics:** Make sure you have a good understanding of the fundamental concepts of Vue.js such as components, directives, and data binding.
- **Know your project structure:** Be familiar with the structure of your project, including the file structure and any relevant dependencies.
- **Explain the functionality:** Clearly explain the main functionality of your project and how it works.
- **Show your code:** Share snippets of your code to demonstrate your understanding and showcase your coding skills.
- **Demonstrate testing:** If applicable, demonstrate any testing methods you used to ensure the functionality of your project.
- **Discuss scalability:** Explain how your project can scale to meet future demands and any techniques you used to ensure scalability.
- **Emphasize your contribution:** Highlight your specific contribution to the project and any challenges you overcame during the development process.

Remember to keep your answers brief and to the point, focusing on the most important aspects of your project.

Tips to Crack Vue Js Interview

39. Interview Tips

Certainly, there are some tips and tricks that you can follow for answering Vue.js interview questions:

- Before the interview, you can take some time to review the Vue.js documentation. This helps you in understanding the key concepts and features of the framework.
- Be familiar with the lifecycle hooks of Vue.js. **(Mostly Asked in Interviews)**. When lifecycle hooks are called. This will help you better understand how components are created and updated.
- Make sure that you are very comfortable with creating components and using directives such as **v-bind** and **v-on** to manipulate them. **(Interviewer might ask to create)**.
- Vue.js templates are a powerful tool for building dynamic user interfaces. Make sure you understand how they work and how to use them effectively. Also, how you have utilized it in your projects.
- If you are applying for a role that requires Vuex knowledge, make sure you are familiar with how to use **Vuex** for state management.
- You should be able to debug Vue.js applications using the Vue.js devtools, as well as the browser's console and debugger.
- Vue.js is constantly evolving, so make sure you keep up to date with the latest changes and updates.
- When answering interview questions, it's important to explain your thought process and the reasoning behind your answers. This will help the interviewer understand how you approach problems and make decisions.
- Practice, practice, practice. The best way to prepare for a Vue.js interview is to practice answering sample interview questions. You can find sample questions in this article or you can create your own based on the job description.
- Be very confident and enthusiastic about your Vue.js skills and experience. Show the interviewer that you are passionate about the framework and excited about the opportunity to work with it. **(This makes the interviewer think that you are the right fit)**.

Links to More Interview Questions

[C Interview Questions](#)

[Php Interview Questions](#)

[C Sharp Interview Questions](#)

[Web Api Interview Questions](#)

[Hibernate Interview Questions](#)

[Node Js Interview Questions](#)

[Cpp Interview Questions](#)

[Oops Interview Questions](#)

[Devops Interview Questions](#)

[Machine Learning Interview Questions](#)

[Docker Interview Questions](#)

[Mysql Interview Questions](#)

[Css Interview Questions](#)

[Laravel Interview Questions](#)

[Asp Net Interview Questions](#)

[Django Interview Questions](#)

[Dot Net Interview Questions](#)

[Kubernetes Interview Questions](#)

[Operating System Interview Questions](#)

[React Native Interview Questions](#)

[Aws Interview Questions](#)

[Git Interview Questions](#)

[Java 8 Interview Questions](#)

[Mongodb Interview Questions](#)

[Dbms Interview Questions](#)

[Spring Boot Interview Questions](#)

[Power Bi Interview Questions](#)

[Pl Sql Interview Questions](#)

[Tableau Interview Questions](#)

[Linux Interview Questions](#)

[Ansible Interview Questions](#)

[Java Interview Questions](#)

[Jenkins Interview Questions](#)