


Vue.Js


Interview Questions

Q: What is Vue.js?




A: Vue.js is a progressive JavaScript framework used for building user interfaces. It is designed to be incrementally adoptable, meaning you can use it to enhance existing projects or build complex single-page applications (SPAs) from scratch.

Q: What are the core features of Vue.js?




A: The core features of Vue.js include a reactive data binding system, a component-based architecture, a virtual DOM for efficient rendering, directives for manipulating the DOM, and the Vue Router for managing navigation in SPAs.

Q: Explain the Vue instance.




A: The Vue instance is the root of a Vue application. It is created using the Vue `constructor` and controls the app's data and behavior. The Vue instance connects the data `model` with the DOM, allowing for reactive updates when the data changes.

Q: What is the difference between v-if and v-show?




A: v-if is a directive that conditionally renders elements based on a `boolean` expression. If the expression evaluates to false, the element is destroyed and re-created when true. v-show, on the other hand, toggles the visibility of the element using CSS display property, so it is always rendered but hidden.

Q: How does Vue.js implement two-way data binding?




A: Vue.js **implements** two-way data binding using the v-model directive. **This** directive creates a binding between form input elements and a data property in the Vue instance, allowing for automatic synchronization of data and UI.

Q: What are Vue `components` ?




A: Vue `components` are reusable instances with a name. They can be defined as JavaScript objects or single-file `components` (.vue files). Each `component` has its own data, methods, and lifecycle `hooks` , making it an essential building block for Vue applications.

Q: What is Vue Router?




A: Vue Router is the official `routing` library for Vue.js. It enables navigation between different `components` and views in a single-page application. It allows you to define routes, manage navigation, and create dynamic route matching.

Q: What are lifecycle `hooks` in Vue.js?




A: Lifecycle `hooks` are special methods in Vue `components` that allow you to run code at different stages of a `component`'s lifecycle, such as creation, mounting, updating, and destruction. Common `hooks` include `created`, `mounted`, `updated`, and `beforeDestroy`.

Q: How do you manage `state` in Vue.js applications?




A: `State` management in Vue.js applications can be achieved using Vuex, a `state` management library specifically for Vue. Vuex provides a centralized store for all `components` in an application, allowing for predictable `state` changes and better debugging.

Q: What are directives in Vue.js?




A: Directives are special tokens in the markup that tell the library to do something to a DOM element. Examples include v-bind for binding data to attributes, v-model for two-way data binding, and v-for for rendering lists.

Q: What is the purpose of the key attribute in Vue.js lists?




A: The key attribute is used to uniquely identify elements in a list rendered with v-for. It helps Vue optimize rendering by keeping track of which items have changed, been added, or removed, ensuring efficient updates to the DOM.

Q: How can you pass data from a parent `component` to a child `component` in Vue.js?




A: Data can be passed from a parent `component` to a child `component` using props. `Props` are custom attributes that can be defined in the child `component` 's `props` option and bound to data in the parent component.

Q: Explain the concept of slots in Vue.js.




A: Slots are a way to create reusable `components` with customizable content. They allow you to pass markup from a parent `component` into a child `component` , giving flexibility in how the `component` is used `while` maintaining its structure.

Q: What is the Vue CLI, and why is it used?




A: The Vue CLI is a command-line **interface** tool for scaffolding Vue.js projects. It provides a standard project structure, built-in tools for **development** , **testing** , and **deployment** , and simplifies the setup process for **new** Vue applications.

Q: How can you handle form submissions in Vue.js?




A: Form submissions in Vue.js can be handled using the v-on directive to listen for the submit event on a form element. You can then define a method to process the form data when the form is submitted.

Q: What is the reactivity system in Vue.js?




A: The reactivity system in Vue.js is based on an observer pattern. It makes use of getters and setters to track changes to the data properties. When a property changes, Vue automatically updates the DOM to reflect the new state.

Q: How can you create mixins in Vue.js?




A: Mixins are a flexible way to distribute reusable functionality across Vue components. You can create a mixin as a JavaScript **object** containing data, methods, and lifecycle **hooks** , and then include it in a **component** using the mixins option.

Q: What is the difference between computed properties and methods in Vue.js?




A: Computed properties are cached based on their dependencies and only re-evaluated when those dependencies change, `while` methods are executed every time they are called. Computed properties are generally used for derived `state`, `while` methods are for actions.

Q: What are the advantages of using Vue.js over other frameworks?




A: Advantages of using Vue.js include its simplicity and ease of integration, a gentle learning curve, a component-based architecture, excellent documentation, a flexible ecosystem, and strong community support.

Q: What is Vue.js?




A: Vue.js is a progressive JavaScript framework used for building user interfaces. It is designed to be incrementally adoptable, meaning that if you have an existing project, you can introduce Vue.js into it without having to rewrite the entire codebase.

Q: What are the core features of Vue.js?




A: Core features of Vue.js include its reactive data binding, component-based architecture, a virtual DOM for efficient rendering, directives for extending HTML, and a rich ecosystem of libraries and tools.

Q: How does Vue's reactivity system work?




A: Vue's reactivity system is based on getters and setters, which are used to track changes to data properties. When a data property is changed, Vue automatically re-renders the `components` that depend on that property, keeping the UI in sync with the underlying data.

Q: What are Vue `components` ?



A: Vue `components` are reusable instances with a name. They can be used as custom elements inside other `components` ' templates. `Components` allow for encapsulation of functionality and for better code organization.

Q: Explain the lifecycle `hooks` in Vue.js.



A: Vue.js provides lifecycle `hooks` that allow developers to run code at specific stages of a `component`'s lifecycle. Key `hooks` include 'created', 'mounted', 'updated', and 'destroyed', which correspond to the `component`'s creation, insertion into the DOM, reactivity updates, and destruction, respectively.



25%


You're 25% through! Keep going! Success is built one step at a time.

Q: What is the difference between `props` and data in Vue.js?




A: In Vue.js, 'data' is used to define the internal `state` of a `component`, while '`props`' are used to pass data from a parent `component` to a child component. `Props` are read-only within the child component.

Q: How can you manage `state` in a Vue.js application?




A: `State` management in a Vue.js application can be handled using Vuex, which is a `state` management library specifically for Vue. Vuex provides a centralized store to manage the application `state` and ensures that `components` reactively update when the `state` changes.

Q: What are directives in Vue.js?




A: Directives are special tokens in the markup that tell the library to do something to a DOM element. For example , 'v-bind' is used to bind an attribute to an expression, and 'v-if' is used for conditional rendering.

Q: What is the purpose of computed properties in Vue.js?




A: Computed properties are used to define properties that depend on other reactive data properties. They are cached based on their dependencies and are only recalculated when their dependencies change, making them efficient for performance.

Q: How do you handle events in Vue.js?




A: Events in Vue.js are handled using the 'v-on' directive. You can listen for DOM events and execute methods when those events occur. For example, 'v-on:click' can be used to call a method when an element is clicked.

Q: What is Vue Router?




A: Vue Router is the official `routing` library for Vue.js that enables navigation between different `components` or views in a Vue application. It supports dynamic `routing` , nested routes, route parameters, and guards.

Q: How do you create a Vue instance?




A: A Vue instance is created using the 'new Vue()' constructor. You can pass an options **object** that defines the data, methods, computed properties, lifecycle **hooks** , and other configurations for the instance.

Q: What is the purpose of the 'key' attribute in Vue.js lists?




A: The 'key' attribute is used in Vue.js lists to give each item a unique identity. **This** helps Vue to efficiently update and re-render only the items that have changed, rather than re-rendering the entire list.

Q: Explain the concept of slots in Vue.js.




A: Slots are a way to create reusable `components` with flexible content. They allow you to pass content from a parent `component` into a child component. Named slots can be used to specify different sections of content.

Q: How can you conditionally `render` `components` in Vue.js?




A: You can conditionally `render` `components` using the 'v-if', 'v-else-if', and 'v-else' directives. These directives allow you to specify conditions under which certain `components` should be rendered.

Q: What is the difference between v-show and v-if?




A: The 'v-if' directive adds or removes elements from the DOM based on the condition, `while` 'v-show' toggles the visibility of an element by using CSS display property. 'v-if' is more performance-intensive than 'v-show' since it involves DOM manipulation.

Q: How can you handle form inputs in Vue.js?




A: Form inputs in Vue.js can be handled using the 'v-model' directive, which creates two-way data binding between form elements and the Vue instance data. **This** means that changes to the input fields automatically update the data properties.

Q: What are mixins in Vue.js?




A: Mixins are a flexible way to distribute reusable functionalities across components. A mixin can contain data, methods, lifecycle `hooks` , and more, which can be included in a `component` , thereby allowing for code reuse.

Q: How can you optimize `performance` in a Vue.js application?




A: `Performance` in a Vue.js application can be optimized by using the 'v-once' directive for static content, lazy loading `components`, using computed properties instead of methods for derived `state`, and leveraging the Vue devtools for profiling.

Q: What is Vue.js?




A: Vue.js is a progressive JavaScript framework used for building user interfaces. It is designed to be incrementally adoptable, which means that you can use it to enhance existing applications or build **new** applications from scratch.

Q: What are the main features of Vue.js?




A: The main features of Vue.js include its reactive data binding, component-based architecture, virtual DOM implementation, directives for DOM manipulation, and an ecosystem that includes Vue Router for `routing` and Vuex for `state` management.

Q: What is the Vue instance?




A: The Vue instance is the root of every Vue application. It is created using the Vue `constructor` and serves as the entry point for Vue's reactivity system. The instance is created by passing an options `object` that contains data, methods, and lifecycle hooks.

Q: What is the difference between a `component` and an instance in Vue?




A: A Vue instance is a standalone `object` that manages its own data and behavior, `while` a `component` is a reusable piece of code that encapsulates its own template, logic, and styles. `Components` are instances of Vue that can be reused throughout the application.

Q: What are `props` in Vue.js?




A: `Props` are custom attributes that can be passed from a parent `component` to a child `component` in Vue.js. They allow data to be passed down the `component` tree and enable communication between components.

Q: How does Vue.js handle two-way data binding?




A: Vue.js **implements** two-way data binding primarily through the v-model directive. **This** directive binds the value of an input element to a data property, allowing for automatic synchronization between the data and the view.

Q: What is the purpose of the Vue Router?




A: Vue Router is the official router for Vue.js. It enables navigation between different views or `components` within a Vue application, allowing for a single-page application (SPA) experience. It supports dynamic `routing` and nested routes.

Q: What are Vue directives?




A: Vue directives are special tokens in the markup that tell the library to do something to a DOM element. Some common directives include v-bind for binding data to attributes, v-if for conditional rendering, and v-for for iterating over lists.

Q: What is Vuex?




A: Vuex is a `state` management library for Vue.js applications. It provides a centralized store for managing `state` across `components`, which helps in maintaining a predictable `state` across the application, especially in larger applications.

Q: How can you create a custom directive in Vue.js?



A: You can create a custom directive in Vue.js using the `Vue.directive()` method, where you define the directive's behavior, such as binding, updating, and unbinding functions to handle DOM manipulation.

Q: What are lifecycle `hooks` in Vue.js?




A: Lifecycle `hooks` are special methods in Vue `components` that allow you to run code at specific stages of a `component`'s lifecycle, such as creation, mounting, updating, and destruction. Common lifecycle `hooks` include `created`, `mounted`, `updated`, and `destroyed`.

A green circular progress indicator with a white border, showing 50% completion. The text "50%" is centered in white.

50%


Halfway there! Every expert was once a beginner.

Q: What is the purpose of the v-for directive?




A: The v-for directive in Vue.js is used to `render` a list of items by iterating over an `array` or object. It can be used to create a list of `components` or elements dynamically based on the data provided.

Q: How can you handle events in Vue.js?




A: In Vue.js, you can handle events using the v-on directive, which allows you to listen for DOM events and execute methods defined in your component. You can also use the shorthand '@' for v-on.

Q: What is the difference between computed properties and methods in Vue.js?




A: Computed properties are cached based on their dependencies, meaning they only re-evaluate when their dependencies change, `while` methods are called every time they are invoked. Computed properties are typically used for transforming data displayed in the template.

Q: What is the purpose of the key attribute in Vue.js lists?




A: The key attribute is used in Vue.js to give each item in a list a unique identifier, which helps Vue efficiently update and re-render the list when items are added, removed, or reordered.

Q: What are mixins in Vue.js?




A: Mixins are a flexible way to distribute reusable functionalities across `components` in Vue.js. A mixin can contain data, methods, lifecycle `hooks`, and more, and when mixed into a `component`, its properties are merged into that `component`'s options.

Q: How do you perform AJAX requests in Vue.js?




A: You can perform AJAX requests in Vue.js using libraries like Axios or the `Fetch` API. You typically make these requests in lifecycle `hooks` such as `mounted` or `created` to `fetch` data and update the `component`'s `state` accordingly.

Q: What is the difference between `props` and data in Vue.js?




A: `Props` are used to pass data from a parent `component` to a child `component`, `while` data is used to define internal `state` within a component. `Props` are read-only in the child `component`, whereas data can be modified.

Q: How do you implement conditional rendering in Vue.js?




A: You can implement conditional rendering in Vue.js using the v-if, v-else-if, and v-else directives. These directives allow you to `render` elements conditionally based on the truthiness of an expression.

Q: What is Vue.js?




A: Vue.js is a progressive JavaScript framework used for building user interfaces and single-page applications. It is designed to be incrementally adoptable, meaning you can use it as a library to enhance a web page or as a full-fledged framework for a complex application.

Q: What are the core features of Vue.js?




A: Some core features of Vue.js include a reactive data binding system, a component-based architecture, a virtual DOM for efficient rendering, directives for DOM manipulation, and a powerful ecosystem with tools and libraries for `routing` and `state` management.

Q: Explain the Vue instance.




A: The Vue instance is the root of every Vue application. It is created by calling the Vue `constructor` with an options `object` that can include data, methods, computed properties, lifecycle `hooks` , and more. The Vue instance is responsible for managing the data and behaviors of the application.

Q: What are `components` in Vue.js?




A: `Components` are reusable Vue instances with a name. They can accept `props`, emit events, and encapsulate their own `state` and behavior. `Components` help in organizing the application into smaller, manageable parts, making it easier to maintain and reuse code.

Q: What is the difference between v-if and v-show?




A: v-if and v-show are both directives used for conditional rendering in Vue.js. v-if actually removes or adds the element to the DOM based on the condition, `while` v-show simply toggles the visibility of the element (using CSS display property). v-if is more performance-heavy when toggling frequently, `while` v-show is better for scenarios where the element needs to be shown/hidden often.

Q: How do you pass data from a parent `component` to a child `component` ?




A: Data can be passed from a parent `component` to a child `component` using props. The parent `component` declares the `props` on the child `component` and binds data to them using the v-bind directive or shorthand colon notation.

Q: What are directives in Vue.js?




A: Directives are special tokens in the markup that tell the library to do something to a DOM element. Vue.js comes with built-in directives like v-bind, v-model, v-if, v-for, and v-show, which provide functionality for data binding, conditional rendering, and iteration over data.

Q: Explain the lifecycle `hooks` in Vue.js.



A: Lifecycle `hooks` are methods that provide visibility into the various stages of a Vue `component` 's lifecycle. Common lifecycle `hooks` include `created`, `mounted`, `updated`, and `destroyed`. Each hook allows you to run your own code at specific points in the `component` 's existence, such as when it is created, added to the DOM, updated, or removed.

Q: What is Vue Router?




A: Vue Router is the official `routing` library for Vue.js. It allows you to create single-page applications by managing the navigation between different views or components. It supports features like nested routes, route parameters, dynamic `routing` , and route guards for authentication.

Q: What is Vuex?




A: Vuex is a `state` management pattern + library for Vue.js applications. It serves as a centralized store for all the `components` in an application, allowing `components` to share `state` in a predictable way. It follows a unidirectional data flow and provides features like `state`, getters, mutations, and actions.

Q: How do you handle events in Vue.js?




A: Events in Vue.js can be handled using the v-on directive. You can listen for DOM events and execute methods when those events occur. For example, v-on:click can be used to listen for a click event on an element, and you can bind it to a method defined in the Vue instance.

Q: What is a computed property in Vue.js?




A: Computed properties are properties that are derived from other data properties. They are reactive and will automatically re-evaluate when their dependencies change. Computed properties are useful for encapsulating complex logic and keeping the template clean.

Q: What is the purpose of the 'key' attribute in Vue.js?




A: The 'key' attribute is used to assign a unique identifier to elements in lists rendered with v-for. It helps Vue optimize rendering by keeping track of changes, enabling it to efficiently re-render only the elements that have changed rather than the entire list.

Q: How can you create mixins in Vue.js?




A: Mixins are a flexible way to distribute reusable functionalities across components. A mixin is an **object** containing **any** **component** options. You can define a mixin and then include it in a **component** 's mixins option. The **component** will inherit the properties and methods defined in the mixin.

Q: What is the difference between `props` and data in Vue.js?




A: `Props` are custom attributes that can be passed from a parent `component` to a child `component`, allowing data to flow down the `component` tree. Data, on the other hand, is internal `state` managed within the `component` itself. `Props` are read-only in child `components`, while data can be modified within the component.

Q: Explain the concept of watchers in Vue.js.



A: Watchers are a way to react to data changes in Vue.js. They allow you to perform asynchronous or expensive operations in `response` to changing data. A watcher is defined in the Vue instance and can be set to watch a specific data property. When that property changes, the watcher `function` is triggered.

Q: How do you manage forms in Vue.js?




A: Forms in Vue.js can be managed using the v-model directive for two-way data binding. **This** directive automatically synchronizes the input fields with the Vue instance's data properties. You can also use event handlers to validate the form data and manage submission.



75%


You're at 75%! Almost done, push through and finish strong!

Q: What is the purpose of the 'v-model' directive?




A: The v-model directive is used for creating two-way data bindings on form input elements. It automatically updates the bound data property when the input's value changes and vice versa, making it easy to manage user input in forms.

Q: What is Vue.js?




A: Vue.js is a progressive JavaScript framework used for building user interfaces. It is designed to be incrementally adoptable, meaning you can introduce it into existing projects without having to refactor everything.

Q: What are the core features of Vue.js?




A: The core features of Vue.js include a reactive data binding system, a component-based architecture, a virtual DOM for efficient rendering, directives for manipulating the DOM, and a rich ecosystem that includes Vue Router for `routing` and Vuex for `state` management.

Q: Explain the Vue instance.




A: The Vue instance is the root of a Vue application. It is created by instantiating the Vue `class` with the '`new` Vue()' syntax. It connects data, methods, and lifecycle `hooks` to the DOM, allowing for reactive updates and rendering.

Q: What are `components` in Vue.js?




A: `Components` in Vue.js are reusable instances with a name. They encapsulate their own structure, logic, and style. `Components` can be defined globally or locally, and they can communicate with each other through `props` and events.

Q: What is the difference between `props` and data in Vue.js?




A: `Props` are custom attributes that allow data to be passed from a parent `component` to a child `component`, `while` data is local `state` managed within a component. `Props` are read-only and meant for communication, whereas data can be modified within the component.

Q: How do you create a Vue.js component ?




A: A Vue.js component can be created using the `Vue.component` method or by defining a Vue instance with a `components` option in the parent component. Syntax: `Vue.component('my-component', { template: '<div></div>', data() { return {}; } });`

Q: What are directives in Vue.js?




A: Directives are special tokens in the markup that tell the library to do something to a DOM element. Some common built-in directives include 'v-bind' for binding attributes, 'v-model' for two-way data binding, and 'v-for' for rendering lists.

Q: Can you explain the Vue.js lifecycle `hooks` ?




A: Lifecycle `hooks` are methods that allow you to run code at specific stages of a `component` 's lifecycle. Some important `hooks` include 'created' (executed after the instance is created), 'mounted' (executed after the `component` is added to the DOM), and 'destroyed' (executed just before the `component` is destroyed).

Q: What is Vue Router?




A: Vue Router is the official `routing` library for Vue.js. It enables navigation between different `components` and views in a single-page application, allowing for dynamic `routing` and URL management.

Q: What is Vuex?




A: Vuex is a `state` management pattern + library for Vue.js applications. It serves as a centralized store for all `components` in an application, ensuring that the `state` is reactive and can be accessed from `any` component.

Q: How do you handle events in Vue.js?




A: Events in Vue.js can be handled using the 'v-on' directive. You can bind methods to events such as click, input, and submit. For example: `<button v-on:click='methodName'>Click me</button>`.

Q: What is the purpose of the 'v-model' directive?




A: 'v-model' is a directive for creating two-way data bindings on form input elements. It allows you to bind the value of input elements like text fields, checkboxes, and select boxes directly to a `component` 's data property.

Q: How do you perform conditional rendering in Vue.js?




A: Conditional rendering can be performed using the 'v-if', 'v-else-if', and 'v-else' directives. These directives allow you to conditionally `render` elements based on the truthiness of a data property.

Q: What is the purpose of the 'key' attribute in Vue.js lists?




A: The 'key' attribute is used in Vue.js lists to give elements a unique identifier, which helps Vue track changes to the list. **This** improves **performance** and helps Vue optimize rendering by minimizing the **number** of DOM updates.

Q: What is the difference between computed properties and methods in Vue.js?




A: Computed properties are reactive and cached based on their dependencies, meaning they only re-evaluate when their dependencies change. Methods, on the other hand, are invoked every time the `component` re-renders, regardless of whether their dependencies have changed.

Q: How do you pass data from a parent `component` to a child `component` ?




A: Data can be passed from a parent `component` to a child `component` using props. You define `props` in the child `component` and then pass data from the parent `component` using the HTML syntax: `<child-component :propName='data'></child-component>`.

Q: Explain how to implement form validation in Vue.js.




A: Form validation in Vue.js can be implemented using computed properties and methods to check the validity of the form fields. You can also use libraries like VeeValidate or create custom validation logic to provide feedback to users.

Q: What is the difference between 'v-show' and 'v-if'?




A: 'v-if' conditionally renders elements and adds/removes them from the DOM based on the condition, `while` 'v-show' toggles the visibility of elements by changing their CSS 'display' property. 'v-if' is more performance-intensive than 'v-show' because it involves DOM manipulation.

Q: How can you optimize the `performance` of a Vue.js application?



A: `Performance` `optimization` in Vue.js can be achieved by leveraging the virtual DOM, using computed properties for expensive calculations, minimizing watchers, lazy loading `components`, and using `server-side` rendering (SSR) where applicable.

Q: What is Vue.js and what are its core features?




A: Vue.js is a progressive JavaScript framework used for building user interfaces and single-page applications. Its core features include a reactive data binding system, a component-based architecture, a virtual DOM for efficient rendering, and a flexible ecosystem that allows for integration with other libraries or existing projects. Vue also provides an official CLI to streamline project setup and development.

Q: Explain the difference between `props` and `state` in Vue.js.


A: In Vue.js, '`props`' and '`state`' (also known as 'data') serve different purposes. `Props` are used to pass data from a parent `component` to a child `component`, enabling a unidirectional data flow. They are immutable within the child component. On the other hand, '`state`' refers to the internal data of a `component`, which can be modified internally. `This` `state` can be reactive, meaning that `any` changes to it will automatically update the view.

Q: How does Vue.js handle reactivity?




A: Vue.js uses a reactivity system that is based on getters and setters. When you create a Vue instance, it converts the data properties into reactive properties using `Object.defineProperty`. When a property is accessed or modified, Vue tracks these interactions and automatically re-renders the `component` when the `state` changes. `This` ensures that the UI stays in sync with the underlying data model.

Q: What are Vue directives? Can you give some examples?




A: Vue directives are special tokens in the markup that tell the library to do something to a DOM element. Some common directives include 'v-bind' for binding attributes, 'v-model' for two-way data binding with form inputs, and 'v-if' for conditional rendering of elements. For example, 'v-if="isVisible"' will render the element only if the 'isVisible' data property is true.

Q: What is the purpose of Vue Router?



A: Vue Router is an official `routing` library for Vue.js that enables navigation between `components` in a single-page application. It allows developers to define routes that `map` to `components`, handle route parameters, and manage navigation guards for `authentication` and access control. `This` enables building dynamic applications with a seamless user experience, as the browser's history and URL can be manipulated without a full page reload.

Q: Can you explain what Vuex is and how it works?



A: Vuex is a `state` management library for Vue.js applications, designed to manage shared `state` across components. It follows the Flux architecture pattern, using a centralized store to hold all application state. `Components` can access the `state` through getters and can modify it via mutations. Actions can also be used to perform asynchronous operations before committing mutations. `This` structure helps maintain a predictable `state` flow and makes debugging easier.



GM
SUNSHINE

Thank You!

You've completed all the questions.

"Education is not the filling of a pot but the lighting of a fire." — W.B. Yeats