INTERVIEW QUESTIONS

**1.**

**Explain the difference between *slots* and *scoped slots*.**

Hide answer

A slot is a placeholder in a child component that is filled with content passed from the parent. Content of a regular slot is compiled in the parent’s scope and then passed to the child component.

Thus you can’t use child component properties in a slot’s content. But *scoped slots* allow you to pass child component data to the parent scope and then use that data in slot content.

**2.**

**Explain Vue.js reactivity and common issues when tracking changes.**

Hide answer

All properties defined in a Vue instance’s data option are *reactive*, meaning that if they change, the component is automatically updated and re-rendered as needed.

All such properties are converted to getters and setters during initialization, thus allowing Vue to detect when those properties are accessed or changed.

The following limitations must be accounted for when designing a Vue app:

* Vue cannot detect object property addition or deletion due to a JavaScript limitation, so the Vue.set method must be used to add new root-level reactive properties.
* Similarly, Vue cannot detect when an array item is modified using an index. Vue.set must be used here as well.

**3.**

**What are mixins? Describe their benefits and drawbacks.**

Hide answer

Mixin support is a feature that allows code reuse between components in a Vue.js application and a software composition tool.

A mixin is a JavaScript object that can contain any option that a component can contain. All mixin content is merged with a component’s options when that component uses a mixin.

Mixins help with following the DRY (don’t repeat yourself) principle. A mixin can even be applied globally to every component instance. In that case, it’s called a global mixin.

Mixins are a powerful tool, but some caution is needed while using them. As with all injected code, we should be careful to avoid maintenance issues and unexpected behavior.

It helps to implement mixins using pure functions that don’t modify anything outside their own scope.

Global mixins should be avoided, as affecting every single component can lead to maintenance issues as an application grows. Injecting specific mixins to components as needed leads to more maintainable code.

**Apply to Join Toptal's Development Network**

and enjoy reliable, steady, remote [Freelance Vue.js Developer Jobs](https://www.toptal.com/freelance-jobs/developers/vue-js)

[**Apply as a Freelancer**](https://www.toptal.com/talent/apply)

**4.**

**What is a single-file component?**

Hide answer

A single-file component is a file with a .vue extension that contains a Vue component. It contains the component’s template, logic, and styles all bundled together in one file. It consists of one <script> block, optional <template> and <style> blocks, and possible additional custom blocks.

To use one, you need to set up Vue Loader for parsing the file (usually done as part of a webpack building pipeline). But this then also supports using non-default languages such as Sass or HTML templating languages with pluggable pre-processors.

**5.**

**Describe data flow between components in a Vue.js app.**

Hide answer

Vue.js uses what’s called a *one-way data flow*. Data is passed to child components from a given parent component using a prop or a custom attribute that becomes a property on the child component instance.

When the parent component updates a prop value, it’s automatically updated in the child component. Mutating a property inside a child component should not be done. Also, it does not affect the parent component (unless it is an object or array).

The child component can communicate back to the parent via an event. The parent can assign a handler to any event emitted by the child component instance and data can be passed back to the parent. The child component can emit a special event for updating the props passed to it.

**6.**

**List the most common cause of memory leaks in Vue.js apps and how they can be solved.**

Hide answer

Memory leaks in Vue.js applications often come from using third-party libraries that create their own instances and/or manipulate the DOM. The v-if directive and the Vue Router destroy Vue component instances; however, cleaning up after any third party library should be done manually in the beforeDestroy() lifecycle hook.

For example, let’s say we use a fictional library, PowerGraph.js, inside our component. It creates a graph instance that displays some data on the page:

**mounted**() {

this.chart = **new** **PowerGraph**();

}

We need to call the graph instance’s destroy() method if it’s provided or implement our own cleanup method:

**beforeDestroy**() {

this.chart.**destroy**();

}

If cleanup is not done before our component gets destroyed, then that memory is never going to be released. Hence, a memory leak.

**7.**

**What is the virtual DOM and how is it beneficial?**

Hide answer

The virtual DOM is a tree-like data structure (or a collection) of JavaScript objects representing DOM nodes that are managed by Vue.js and that should be rendered on the page. These objects are called “virtual nodes” or VNodes for short.

The main purpose of the virtual DOM is faster and more efficient DOM manipulation. When there are lots of nodes in the DOM, updating them can be expensive in terms of processing power and resources required.

Working with the virtual DOM JavaScript object is significantly faster. Subsequently, Vue.js organizes DOM updates in batches for more efficiency.

**8.**

**Consider the following code (**index.html**is omitted for brevity.) What is it going to output in the browser? Please mention any notable console output as well.**

**const** **MockComponent** = {

props: {

showMe: {

type: **Boolean**,

required: true,

},

},

template: `

<div v-if="showMe">

This is a test component

</div>

`,

};

**new** **Vue**({

el: '#app',

components: {

**MockComponent**,

},

template: `

<div>

<MockComponent showMe="" />

</div>

`,

});

Hide answer

A Boolean prop works as boolean attribute in HTML. If a boolean prop is present and has no value (<MockComponent showMe/>), or is equal to the empty string (as in this example) or its own name in [kebab case](http://wiki.c2.com/?KebabCase), its value is true. Therefore this example will output This is a test component in the browser, and no error messages in the console.

Bonus points for mentioning that required doesn’t make much sense here, as omitting this prop (i.e. <MockComponent/>) is the proper way of setting its value to false.

**9.**

**Consider the following code (**index.html**is omitted for brevity). What is it going to output in the console?**

**const** **MockComponent** = {

**render**() {

**return** this.$slots.default;

},

**data**() {

**return** {

status: '',

};

},

watch: {

status: {

**handler**(newVal) {

console.**log**('Status update: ' + newVal);

},

immediate: true,

},

},

**beforeCreate**() {

this.status = 'initializing';

},

**mounted**() {

this.status = 'online';

},

**beforeDestroy**() {

this.status = 'offline';

},

};

**new** **Vue**({

el: '#app',

components: {

**MockComponent**,

},

template: `

<div>

<MockComponent v-if="showComponent" />

</div>

`,

**data**() {

**return** {

showComponent: false,

};

},

**mounted**() {

this.showComponent = true;

window.setTimeout(() => {

this.showComponent = false;

}, 1000);

},

});

Hide answer

It will output:

Status **update**:

Status **update**: online

When the watcher runs immediately it uses the initial value, an empty string. The watcher isn’t triggered when beforeCreate is fired, as reactivity hasn’t been initialized yet. Watchers are destroyed before beforeDestroy is called.

**10.**

**Why would you choose Vue.js over other front-end frameworks?**

Hide answer

When a developer chooses a tech stack they should display an understanding of the pros and cons of alternative solutions. Vue.js should be compared with the most popular front-end frameworks, React and Angular.

Performance

All three frameworks display similar performance. No significant difference here. However, Angular apps with the same base features and functionality have a larger compiled size than two other frameworks.

Ease of Use

React is widely considered to have a steep learning curve. Developers need to learn JSX, ES6, and a build system (like webpack) before they can be productive with React. [Create React App](https://github.com/facebook/create-react-app) assumes that you are building a single-page app thus adopting React for other scenarios would require extra learning time.

Angular effectively requires knowing TypeScript to get started with it. Angular framework design targets building enterprise-scale applications and is quite complex. Developers need to familiarize themselves with an extensive framework API and its concepts before starting to be productive.

One design goal of Vue.js was for it to be *incrementally adoptable*. This means you can drop it into legacy projects and start using it without rebuilding the whole app from the ground up. Vue.js uses core web technologies that web developers are already familiar with: HTML, CSS, and plain JavaScript (ES5). Using build tools is not required: You can include Vue with just a <script> tag.

Suitability for Large-scale Apps

React and Angular were developed by Facebook and Google, respectively, so they have been enterprise-grade frameworks from the start. While Vue.js started as a one-man project, it’s come a long way since then, attracting a large number of supporters and developers—its core team now has more than 20 developers.

More to the point, Vue.js has been adopted by [a number of companies](https://github.com/vuejs/awesome-vue#projects-using-vuejs) including the likes of Adobe and Alibaba.

Thus, all three frameworks are well-suited for building large-scale apps.

Ecosystem

React is the most popular framework at the moment, having a considerably richer ecosystem than both Vue and Angular, with a large number of third-party libraries.

To sum up, Vue.js is on equal footing with other frameworks when building large-scale apps, and it excels when you need to work with legacy applications or jump in and get something done fast. But it’s worth checking ecosystem support for your project’s particulars first.