1. **What is garbage collection?**

* The Garbage Collector(GC) finds the unused objects and deletes them to reclaim the memory.
* In short the data, function, object, etc which is not used and capturing the space so at that we collect those unused object, data, etc this is known as garbage collection.

1. **What is memory leakage issue and how to overcome from that?**

* Memory leak occurs when programmers create a memory in heap and forget to delete it.
* Memory leak is the memory that is no longer used by the system process and is not released in time. When the memory occupancy becomes higher and higher, the system performance will be affected slightly, and the process will collapse heavily.
* **Several situations of memory leak in VUE=>** Memory leak is a cumulative process, only when the page life cycle is slightly longer, frequent interaction can speed up the cumulative process, partial display of pages is difficult to expose such problems (so-called refresh can revive full blood). So many times, we are passive waiting for the problem to be exposed and then we do the investigation. Active analysis is usually difficult. Vue pages are mostly single-page applications, which are highly interactive and stay for a long time, and memory leaks are easy to occur if they are not handled properly.
* To overcome with the memory leakage, we can use some of the function provide by vue and other libraries. i.e.
  + destroy()
  + beforeDestroy()
  + deActivated()

and many more methods or properties are there to achieve this.

* Also, by making the global variable we increase the chances of memory leakage so to overcome that when we need at that only we have to create a global variable and after done with the uses delete that variable also using the methods and properties.
* **Reference Link:**
  + <https://vuejs.org/v2/cookbook/avoiding-memory-leaks.html>
  + <https://developpaper.com/location-and-repair-of-memory-leak-in-vue-single-page-application>

1. **What is server-side rendering?**

* In the server-side rendering, the JavaScript file is compiled on the server not on the browser. Thus, straight and meaningful html takes place directly in the browser.
* In this it runs webpack as a watcher, builds server bundle and restarts app, then it changed
* **When we want to use SSR.**
* There are two main reasons for this; Better SEO and Faster time-to-content.
* Performing SSR makes us stronger for SEO. Because Google Spiders (or other bots) can understand straight and meaningful html output.
* Also, if we use ssr, our site opens faster. Because we make the compilation on the server side so we build a more user-friendly website.
* **Solution or how to achieve SSR in SPA.**

1. Using Express:

* Run the given command on the terminal to set it up i.e. ‘npm install vue vue-server-renderer express — save’
* The renderer functionality in this package is important to us. Vue instance is compiling on the server side with renderer function.
* Now we create a create ‘index.js’ and import all necessary libraries

Code:

**const Vue = require('vue')**

**const server = require('express')()**

**const renderer = require('vue-server-renderer').createRenderer()**

* Now, we create vue instance in express.

Code:

**server.get('\*', (req, res) => {**

**const app = new Vue({**

**data: {},**

**template: ``**

**})**

**})**

**server.listen(8000)**

* Now, we set our port as 8000 and let’s run index.js file on the node.

1. Using Nuxt.js

* For this we required JavaScript environment to render our web page and also required node.js to be configured to execute our Vue.js application.
  + **SSR steps for Nuxt.js:**

**Step 1: Browser to server.**

When a browser sends the initial request, it will hit the Node.js internal server. Nuxt.js will h=generate the HTML and send it back to the browser with results from executed function.

**Step 2: Server to browser.**

The browser receives the render page from the server with the generated HTML. The content is displayed and the Vue.js hydration kicks in, making it reactive. After this process, page is interactive.

**Step 3: Browser to Browser.**

Navigation between pages with <NuxtLink> is done on the client side so we don’t hit the server again unless we hit the hard reload.

* **Reference link**:
* <https://medium.com/swlh/how-to-server-side-rendering-ssr-with-vue-js-71cedce7c299>
* <https://nuxtjs.org/docs/2.x/concepts/server-side-rendering>

1. **What is Computed properties?**

* In-template expressions are very convenient, but they are meant for simple operations. Putting too much logic in your templates can make them bloated and hard to maintain.
* This is used because it decreases the load time of the app if we perform some complicated task or logic. By using this it will store that value in the cache an get the value from there till we don’t change the value.
* **Computed Caching vs Methods:**
  + Instead of a computed property, we can define the same function as a method. For the end result, the two approaches are indeed exactly the same. However, the difference is that **computed properties are cached based on their reactive dependencies.** A computed property will only re-evaluate when some of its reactive dependencies have changed.
* **Code:**
* <template>
* <!--Here i bind the non-props property to this tag. -->
* <h1 v-bind="$attrs">{{ greet }}</h1>
* <h2>
* Name: <u>{{ user.Name }}</u> Age: {{ user.age }}
* </h2>
* <button v-on:click="showAlert()">Click</button>
* <div :class="applyStyle" v-on:mousemove="classStyle = !classStyle">
* <h2>Hover me.</h2>
* </div>
* <hr />
* <h3>Ref Example.</h3>
* <input type="number" ref="demoRef" v-model.number="textValue"/><br>
* {{textValue}}
* <br>
* <button v-on:click="getRefvalue()">Submit</button>
* </template>
* <script>
* export default {
* data() {
* return {
* classStyle: true,
* textValue:''
* };
* },
* props: {
* greet: String,
* user: Object,
* showAlert: Function,
* },
* computed: {
* applyStyle() {
* return {
* font\_size: this.classStyle,
* };
* },
* },
* methods: {
* getRefvalue() {
* if (this.$refs.demoRef.value.length < 6) {
* this.$refs.demoRef.style.color = "red";
* } else {
* this.$refs.demoRef.style.color = "green";
* }
* console.log(this.$refs.demoRef.value);
* },
* },
* };
* </script>
* <style scoped>
* .font\_size {
* font-size: 20px;
* color: crimson;
* }
* </style>
* **Reference Link:** 
  + [**https://v3.vuejs.org/guide/computed.html#computed-properties**](https://v3.vuejs.org/guide/computed.html#computed-properties)

**Q) What is watcher?**

* It generally used to observe the property and take the action when we needed. This is what actually watcher means.
* **Reference Link:**
  + [**https://v3.vuejs.org/guide/computed.html#watchers**](https://v3.vuejs.org/guide/computed.html#watchers)