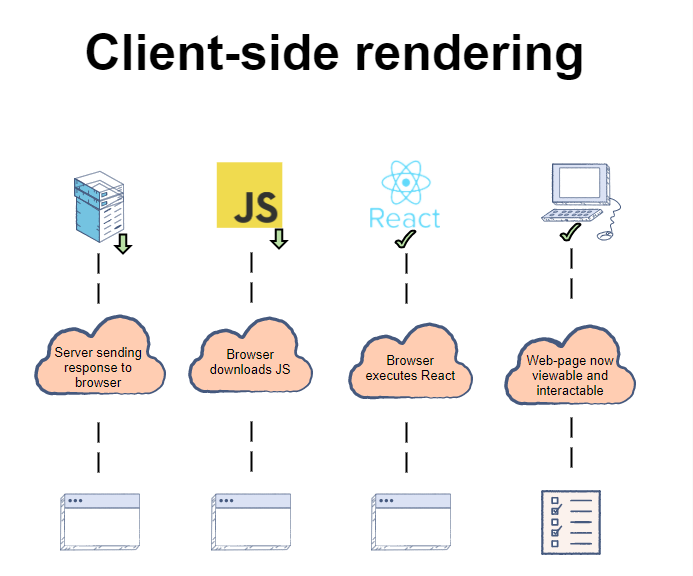
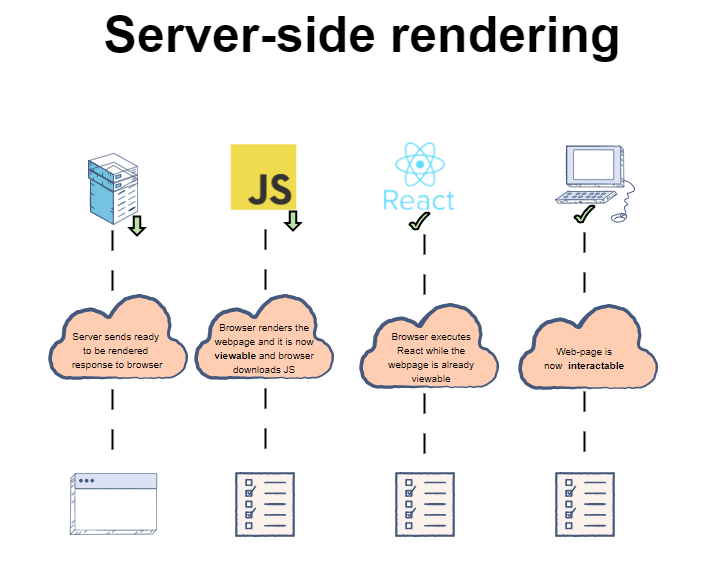
**1) What is server side rendering?**

* **Server-side rendering (SSR)**, is the ability of an application to contribute by displaying the web-page on the server instead of rendering it in the browser. Server-side sends a fully rendered page to the client; the client’s JavaScript bundle takes over and allows the SPA framework to operate. There is also **client-side rendering** which slows down the procedure of viewing and interacting with the web page.



* reference link:-<https://www.educative.io/edpresso/what-is-server-side-rendering>

**2) Garbage Collection:-**

* Garbage collection is a type of memory management. It automatically cleans up unused objects and pointers in memory, allowing the resources to be used again. It can then add instructions to automatically deallocate those resources from memory.

**3) What is memory leakage issue and how to overcome from that:-**

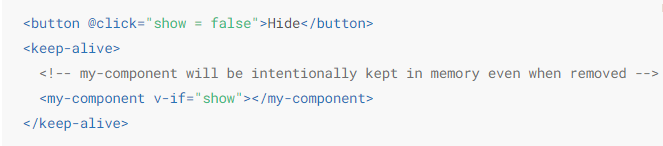
* **Memory Leak**:-
* This issue is especially important in Single Page Applications (SPAs) because by design, users should not have to refresh their browser when using an SPA, so it is up to the JavaScript application to clean up components and make sure that garbage collection takes place as expected.
* **general reasons**:-

1. addEventListener. This is the most common one. Call removeEventListener to clean it up.
2. setTimeout / setInterval. If you create a recurring timer (e.g. to run every 30 seconds), then you need to clean it up with clearTimeout or clearInterval. (setTimeout can leak if it’s used like setInterval – i.e., scheduling a new setTimeout inside of the setTimeout callback.)
3. IntersectionObserver, ResizeObserver, MutationObserver, etc. These new-ish APIs are very convenient, but they are also likely to leak. If you create one inside of a component, and it’s attached to a globally-available element, then you need to call disconnect() to clean them up. (Note that DOM nodes which are garbage-collected will have their listeners and observers garbage-collected as well. So typically, you only need to worry about global elements, e.g. the <body>, the document, an omnipresent header/footer element, etc.)
4. Promises, Observables, EventEmitters, etc. Any programming model where you’re setting up a listener can leak memory if you forget to stop listening. (A Promise can leak if it’s never resolved or rejected, in which case any .then() callbacks attached to it will leak.)
5. Global object stores. With something like Redux the state is global, so if you’re not careful, you can just keep appending memory to it and it will never get cleaned up.
6. Infinite DOM growth. If you implement an infinite scrolling list without virtualization, then the number of DOM nodes will grow without bound.

* Ref-link:-<https://nolanlawson.com/2020/02/19/fixing-memory-leaks-in-web-applications/>
* **How to overcome**:-

1) Use <keep-alive> around dynamic component when toggle between different components.

* Reference-link:-<https://learnvue.co/2019/12/an-overview-of-vue-keep-alive/#:~:text=This%20means%20that%20Vue%20does,show%20up%20as%20a%20component>.
* It includes code and output difference between before <keep-alive> and after <keep-alive around components.



2) Destroy any calling library before destroy component.

* Ref-link:-<https://vuejs.org/v2/cookbook/avoiding-memory-leaks.html>
* The following example shows a memory leak caused by using the [Choices.js](https://github.com/jshjohnson/Choices) library.
* Ex:-Before handling memory leak:- <https://codepen.io/freeman-g/pen/qobpxo>
* After handling memory leak:- <https://codepen.io/freeman-g/pen/qobpxo>

3) Check when use router in web app (when switch components frequently.)

* Just like the **v-if** directive, **vue-router** removes elements from the virtual DOM and replaces those with new elements when a user navigates around your application. The Vue **beforeDestroy()** [lifecycle hook](https://vuejs.org/v2/guide/instance.html#Lifecycle-Diagram) is a good place to solve the same sort of issue in a vue-router based application.
* Reference link:-<https://forum.vuejs.org/t/how-to-destroy-mounted-element-with-its-parent/3232>

**4) What we can do to achieve server side rendering in single page application?**

* To accomplish server-side rendering of an SPA, **javascript code is executed in NodeJS** to generate the initial HTML. In the browser, the same javascript code is executed during a "hydration" process, which attaches event listeners to the HTML. Most popular UI Frameworks (Vue, React, Angular, etc) are capable of executing in both NodeJS and the browser, and offer APIs for both generating the server HTML and hydrating it in the browser.
* Reference links:-
  + <https://vuejs.org/v2/guide/ssr.html>
  + <https://ssr.vuejs.org/> (This link provides full guidance for implement SSR in VueJS Framework.)