Author: Henry Ponce

1. What's a closure? Where in the code is there a closure?

A closure is a method that remembers and maintains access variables from outside its context.
It keeps reference to those variables, even after its creation.
This allows encapsulation, state retention and modular design.
For example, it helps build factory functions, utility functions and reusable code.

In my code, there are many examples, inside Vue components:

- In List.vue, the method **fetchJokes()** maintains access to ref constants from the store.
- In Pager.vue, the method **getPageNextNavigatorClass()**, maintains access to computed constant and ref constant.
- In List.vue, the watcher accesses the **props.total** that was defined outside its scope.

2. Which are the potential side-effects in any function? Could you point out any of these cases in

your code? Are they expected? Can they be avoided?

In my code, I'm well aware that I could've added tests (unit & integration). Error handling could also be improved (I didn't want to spend more time on that).

I had to make modifications to the joke api. Otherwise it wasn't possible to implement pagination, sorting or filtering. I know the modifications to the API aren't the optimal, but I didn't want to spend much time (more than necessary) in the modifications to the API.

I didn't implement rating, nor remove/add more jokes, since it involved modifying the API. It relies on a json (in-memory) file to load all jokes. Persistence could be improved by using a database.

I didn't implement any routing, since it wasn't necessary based on the specs of Challenge. But I'm well aware of how it can be added to the project.

I made use of a Pinia Store to manage filtering. This allows the information to be accessed very easily in different components. And adding 2 Pager.vue very quickly (and easily).

Some level of persistence could be accomplished by using browsers local storage. But ideally, the use of a database (in backend) would be optimal.