**Linking nodes with D3**

**Project Progress**

Currently we can differentiate 03 phases of the project, which each one is distinguished by the development method**\***, to understand this, we need to review the libraries used.

**D3:**

Just now the project is using the version 5.

Surfing on the internet we can find D3 projects using version 3, 4, 5 and 6.

Version 3 is the oldest one. It had too programmatic and algorithmic ways of implementing the graph. Version 4 is the most used. Nevertheless, we chose the version 5, because is the next version. There exist few examples made with version 6, also it has limited documentation.

Therefore, by the amount of documentation, easy implementation, more examples, and the idea of not using the old-fashioned, we opt for the version 5.

**How does it work D3?**

Basically, it selects an element from a HTML document, from this, we add SVG objects that according of the input data (internal or external), it will draw a graph.

The SVG objects of the graph can detect events.

**What are the main problems of D3?**

As we mentioned before, it selects an element of an HTML document, this means, the graph could not be generated until the HTML document has been created. For this reason, it is needed to work correctly with the project lifecycle, an aspect solved by Vue3js.

We need to consider that once a graph is created, if we want to add new information to the graph, D3 offers a way of updating the graph (you can run it constantly, but it will require a lot of RAM consumption). But this update can be triggered by an event because it could not be triggered automatically when we add new information.

**Vue3js:**

Now, we are implementing Vue3js Composition API and Options API.

In the beginning we used only Options API, which consisted in using the lifecycle of the components, internal status (data), properties (props), etc.

**What was the problem with Options API?**

The graph object upon receiving constant information either by mouse event or an update of input data, the code which resolve this is turned complex and long.

Using the graph updating offered by D3 once we get information was not an option to choose, because we would not be applying the reactivity concepts that the app requires. This means, we would not be taking advantage of the Vue3Js benefits causing spaghetti code.

**What did we use to solve it?**

We used Composition API, that is another way of programming with Vue3, and it is more suitable for this case of reactivity.

The D3 graphic is a component that requires a lot of reactivity, because it receives too much information to be processed.

Then, this project had several changes in the code as much of Vue3Js as the D3 implementation, because when you delegate to Vue3Js update, it separates the manipulation of the information to Vue3 and graphic logic for D3.

**(\*)** The first implementation was to graphic **elements** linked with other elements from given information and loaded internally.

The second implementation was to link **nodes** through links from given information and loaded internally.

The third implementation was to linked nodes through given and watched information, once the information is changed, the graph changes, the information is not loaded internally then the graphic is **instantly**.

References:

[Vue 3 - The Composition API (Part 1) -- newline](https://www.newline.co/@kchan/vue-3-the-composition-api-part-1--afbd9dbf)

[5 simple rules to data visualization with Vue.js and D3.js | by Lampros Papadimitriou | Medium](https://medium.com/@lambrospd/5-simple-rules-to-data-visualization-with-vue-js-and-d3-js-f6b2bd6a1d40)

[Using Vue 3's Composition API with D3 - DEV Community](https://dev.to/muratkemaldar/using-vue-3-with-d3-composition-api-3h1g)