In the past year (2019) I spent a lot of time talking, writing, podcasting and just chatting with people about micro-frontends.
I'm truly passionate about the topic, mainly because it's still an unexplored land where challenges arise every single day.

During all my talks, workshops and webinars I was building step by step a mental model that allowed me to simplify my thoughts for making them approachable by people listening to me or reading my contents.
I'm sure there is still a lot of work to do but from the feedback I received so far, I was able to explain in a reasonable way what I had in my mind.

Recently I had the pleasure to do a 2h workshop in Bergen (Norway) about this topic and during my presentation (literally on the stage) I realized that I was very close to simplify even further the decisions process for embracing a micro-frontends architecture.
In *JavaScriptland* we are used to choosing a framework and getting along with that for the entire lifecycle of a project, sometimes we change it in favor of another one but many times we stick with it for many months/years till the end of life for a specific project.
Every time we use a JavaScript framework, someone else made architectural decisions that we live with (or trying to) and we focus on what matters the most from a business point of view: the features of an application.
This doesn't mean we have an easy challenge in front of us at all, we still have an essential part of this "game" taking the right design decisions for delivering the project, but first, we need to focus on having a solid base that allows us to build on top of it, aka the architecture.

Martin Fowler defines software architecture as:

> "Software architecture is those decisions which are both important and hard to change"

Based on this definition, focusing on the key things, even better, the most important decisions for defining a micro-frontends architecture I came up with 4 pillars we have

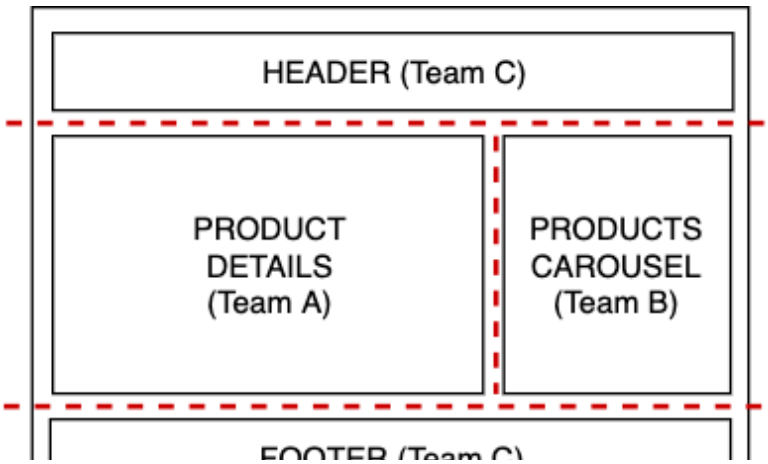
to decide upfront when we start architecting a micro-frontends project.
Those pillars can be summarized in:

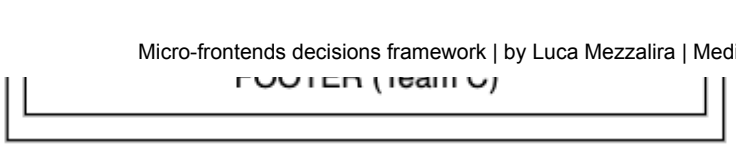
1. Definition
2. Composition
3. Route
4. Communication

Each of them plays a fundamental role in the good outcome of a micro-frontends project.
After taking those 4 decisions there will be many others to take along the journey but with those cornerstones, we should be able to cascade vast majority of the other decisions without any need of evaluating again our architecture.
Bear in mind that those decisions could change over the lifecycle of a project but changing them will require a considerable development effort in order to achieve a coherent project's architecture.

## Defining Micro-Frontends

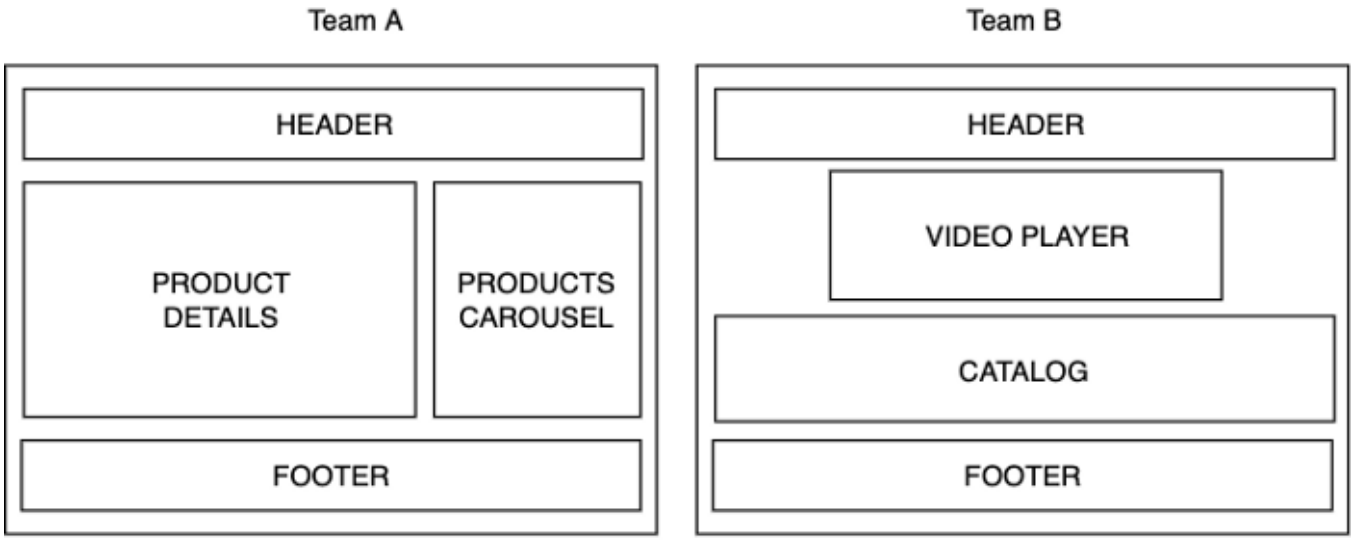
The first decision to take is how we identify a micro-frontend, is it a part of a view (horizontal split) or the entire view (vertical split)?
Splitting horizontally would mean identifying micro-frontends inside the same view, therefore multiple teams are taking care of the page composition coordinating themselves for the final result presented to a user.





HORIZONTAL SPLIT

Splitting vertically, instead, would assign a specific view, or group of views, to a team allowing that team mastering a specific area of the application.




VERTICAL SPLIT

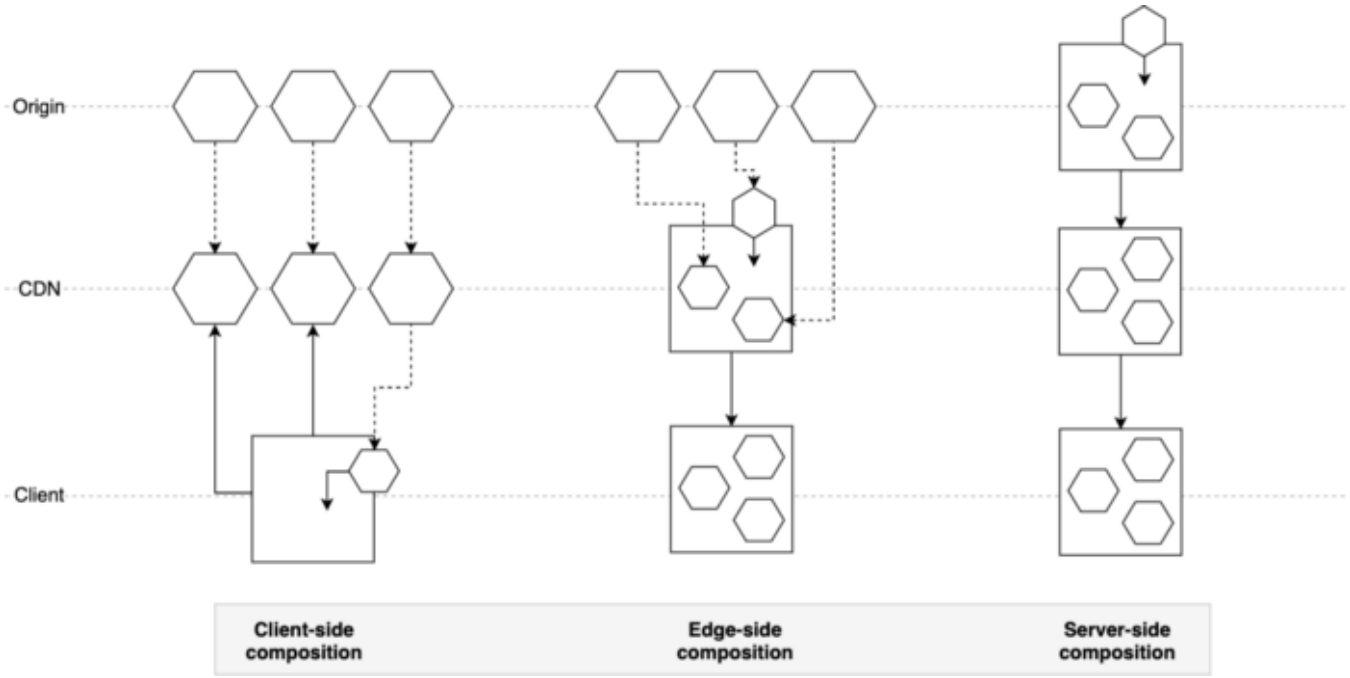
In general, deciding to approach a micro-frontends project with a vertical slicing will simplify many decisions to take further down the line, because it's closer the way a frontend developer is used to work so many practices can be easily applied to this approach.
Also, the coordination between teams should be easier considering each team is owning a vertical slice of the application and not multiple parts spread across the application.

There are additional challenges when we decide to go with the horizontal split, anyway I described more in-depth how to identify micro-frontends in another post available on this website.

## Micro-Frontends Composition

The second decision is understanding where we want to compose micro-frontends, here we have 3 options to choose from:

- client-side composition
- edge-side composition
- server-side composition



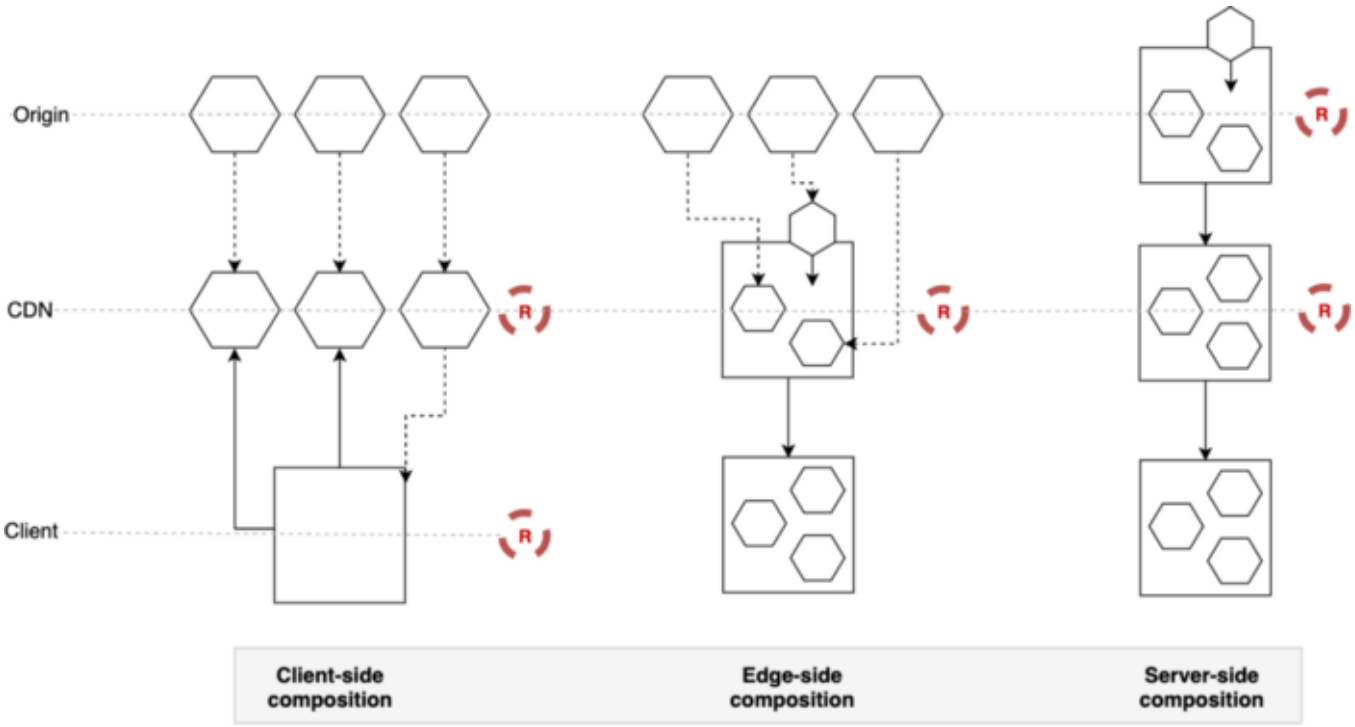

Client-side means implementing techniques like an App Shell loading single page applications, for instance, check out Single-SPA project, with iframes like Luigi framework or via a library using client-side transclusion technique.
Implementing over the edge would mean using CDNs capabilities like Edge-side includes (not always available in all the CDN providers or not fully implemented) or via computation happening on/near the edge (bear in mind there are restrictions in how much logic we could run with a solution like lambda@edge for instance).
Finally, on the server-side, there are plenty of frameworks like Ara Framework, Open Components, Piral or Tailor.js and many others.
When we decide to go with a vertical split we should aim for a client-side composition using Single-SPA or similar mechanisms like we do in DAZN.
Instead, when we decide to identify our micro-frontends as a single part of a view we have all the options available (client-side, edge-side and server-side).

## Micro-Frontends Routing

After defining our micro-frontends composition strategy we need to decide how we want to route our views.

This time the decision is not mutually exclusive, we can decide to route client-side and adding logic on the edge or having the routing logic on the client or server only.
In this case, my suggestion is to be coherent with the composition part if you have decided to go with a client-side composition, use the app shell for mapping the routing logic, if you have chosen the server-side composition, use the webserver for managing the routing logic.
In the case we decided to go with the edge-side composition we will rely on the URL associated with a page.

In the following diagram, you can easily discover the different routing possibilities associated with a composition technique.




I wrote an article about this topic a few months ago that I recommend reading if you are interested in this topic

## Communicating between Micro-Frontends

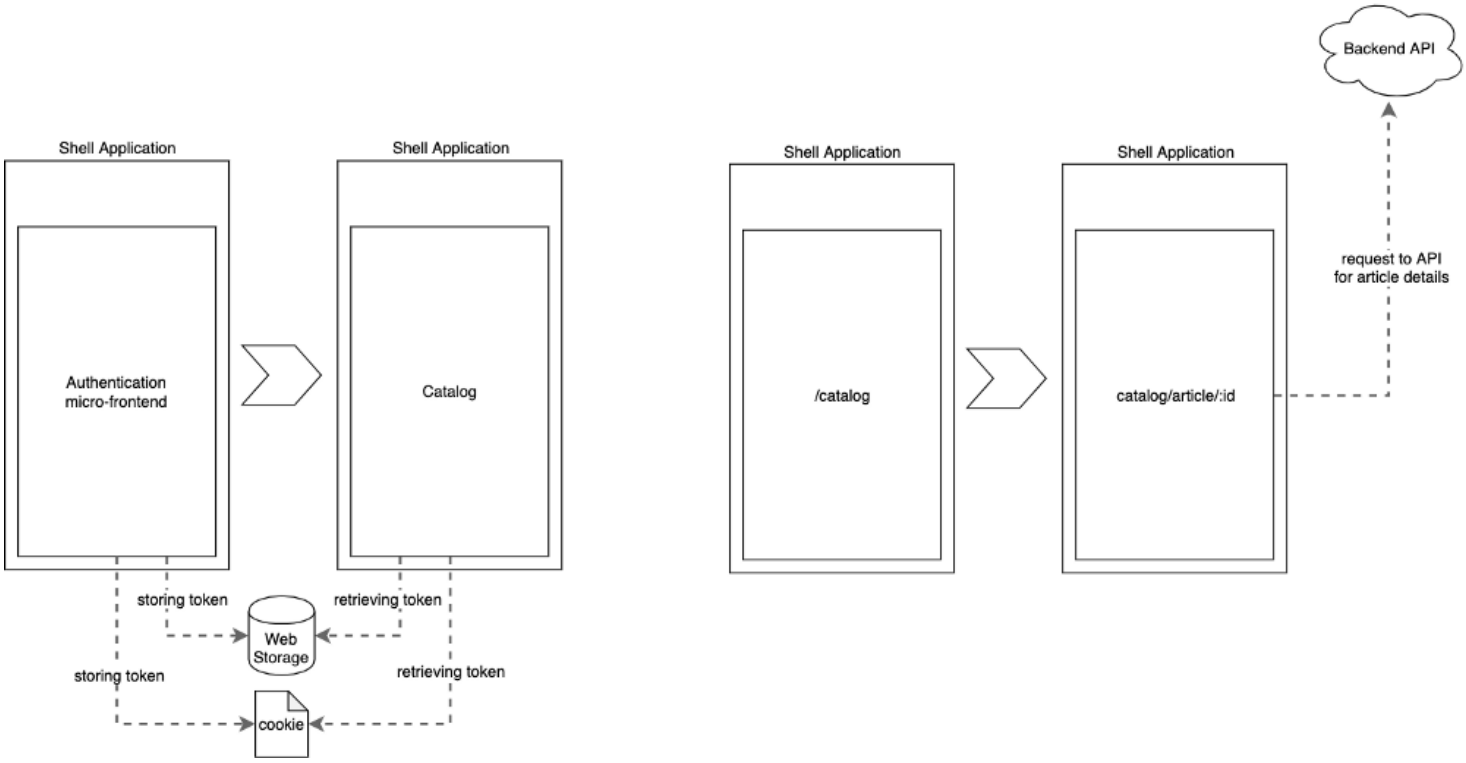
The final decision is finding a good way of communicating between micro-frontends bearing in mind that are independent units that should be completely decoupled between each other.
When we decide to have multiple micro-frontends on the same page we need to decide how (and if) a micro-frontend should notify when an event or a user interaction

happens for coordinating changes in other micro-frontends.

In this case, we could use custom events or an event emitter library (there are many available) so our micro-frontends just emit/dispatch an event and who is listening reacts to that specific event.

However, either we identify multiple micro-frontends per page or we consider a micro-frontend an entire view, we need to decide how a view would exchange data with another view.

We can decide to use a web-storage solution (local and/or session storage) where every time a new view is loaded looks inside the web-storage to find the information needed or a URL routing where the application request to the backend the user state as displayed in the following diagram.




As we have seen, there are some decisions to make when we want to approach a micro-frontends architecture, those will shape the other challenges will be faced alongside a micro-frontends project like how to create a seamless user experience, how to define our CI/CD pipelines, how to optimize our micro-frontends reducing our JavaScript bundles and so on.

However, those 4 decisions made upfront will provide a strong guideline for facing all the other challenges and restrict the number of options to choose from.