

Identifying micro-frontends in our applications

I've always spent a lot of time reading, attending conferences, researching different topics and those learnings really helped me shape my career, one of them is definitely *Domain Driven Design* (DDD).

Let's take a step back first, *why am I talking about DDD?*

One thing that always puzzled me in this industry is the lack of learnings from different technology communities, for instance, we can find quite a lot food for thoughts when we examine the principles behind microservices more than focusing on the bare implementation.

On the backend side, there are often practices, methodologies, more in general ideas, that are totally applicable on the frontend too, but often we don't think how to do it.

Often just taking a step back, understanding why someone implemented a pattern over another, allows us to open up a world of opportunities that we would never think about because "it's not the standard way to do things".

Contaminations from different industries or technologies allow us to see the world from a different perspective, creating new possibilities not explored enough (or at all sometimes), giving us the possibility to apply concepts and mental model to our day to day work.

DDD key concepts

Ok, now we can explain why DDD is mentioned in a post where I talk about micro-frontends. DDD brings on the table some of the key concepts for defining a micro-frontends because it helps our organisation to align the business with the tech side, unifying de facto 2 main areas of our companies: product and tech.

DDD starts with the idea of identifying parts of our application that represents a subdomain of the final application.

Usually, an application is focused on a core domain, for instance, Netflix core domain is streaming movies anywhere at any time, considering the domain is usually a complex proposition, DDD suggests to split the domain into multiple subdomains allowing a company to understand how to structure the company as well as the project.

Some examples of subdomains could be the authentication, customer support inventory management and so on.

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

Close and accept

that deliver values above anything else

. **Supporting Subdomains:** subdomains related to the core ones but not key differentiators, those subdomains could support the core subdomains but at the same time are not essential for delivering the real value to our users

. **Generic Subdomains:** those are needed subdomains used for completing the platform and often the companies decide to go with off the shelf software because not strictly related to their domain, for instance, authentication or payments management, more in general anything that is not related to our code business

Sponsored Video

Watch to learn more

SPONSORED BY ADVERTISING PARTNER

[REPORT THIS AD](#)

Inside each subdomain, tech and product teams should identify a *ubiquitous language*, or rather a way where business meets tech using the same language for identifying functionalities, objects but more, in general, the domain model.

Think about it, how often we speak with a product owner that defines part of an application in a completely different way from the techies!

Ubiquitous language is not a static language, should evolve with the business and the applications running alongside it.

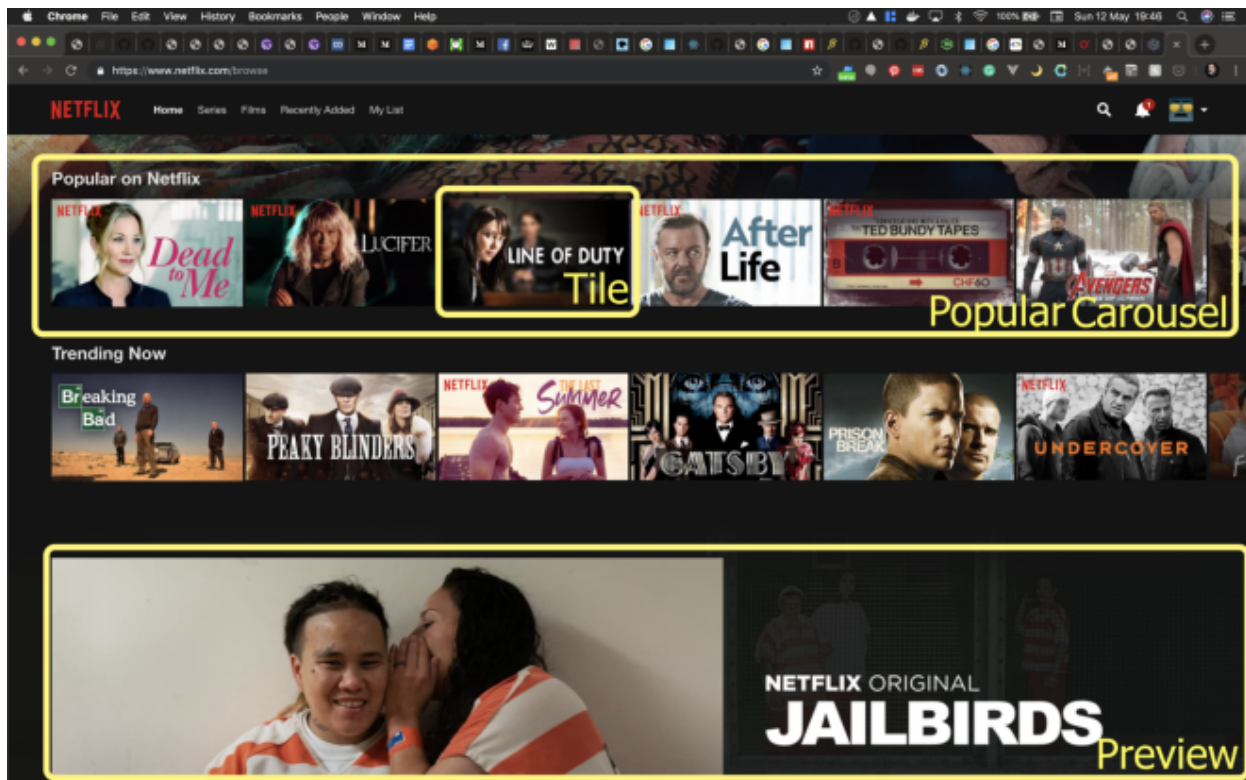
In this way, we would be able to define a domain-model similar to what we discuss day to day with the domain experts, constantly up-to-date.

Let's take Netflix for instance, I think we are all familiar with this famous streaming platform, a subdomain of Netflix might be the catalogue, inside it we can identify multiple areas with specific functionalities, those could be directly connected to backend APIs related to a subset of the entire application.

In Netflix case, they are using Backend For Frontend pattern (BFF) nevertheless the principles remain the same.

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

Close and accept



Netflix web platform

In this screenshot, we can identify some components, those might be linked to some microservices like the personalisation service, the catalogue per country, the most popular contents and so on.

Despite the technical integration that could be via backend for frontend, GraphQL, Server Side Rendering and so on, ***the most important thing to understand is that those areas are all linked to the same subdomain.***

Therefore those microservices, as well as the frontend, should be encapsulated in a unique subdomain with its own ubiquitous language.

Sponsored Video

Watch to learn more

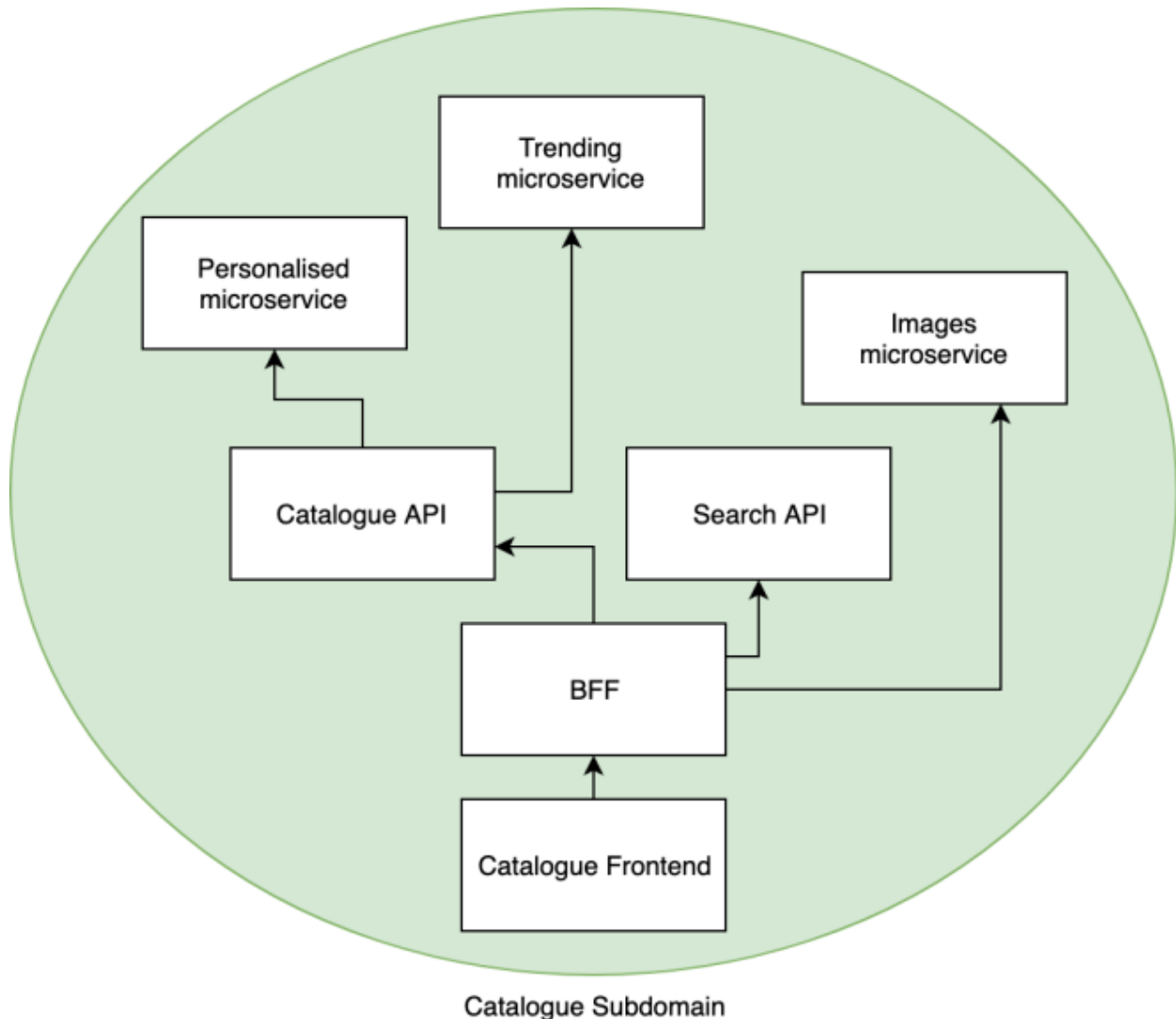
SPONSORED BY ADVERTISING PARTNER

REPORT THIS AI

Following just an example to understand what a subdomain should contain:

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

Close and accept



Catalogue Subdomain

An example of subdomain based on the Netflix platform

This is a fundamental step for identifying how to “slice” our application, understanding that the frontend is part of the subdomain allows us to think holistically about our web application. If we then extend the concept to the infrastructure too, we finally see all the components for developing a subdomain *in the hands of a team that can own Frontend, Backend and Infrastructure end to end* without too many external dependencies.

Up to now, DDD was applied to the backend layer but not very often to the frontend as well, extending these concepts to the frontend allow us to easily identify our micro-frontends. I'd like to highlight another important concept, a ***subdomain cannot (and shouldn't) be recognised as a component in a page***, it's true that in each UI we can find links or graphical elements related to different subdomains but at the same time we need to understand that they are not standalone identifying a subdomain and they need to have teams owning a subdomain end to end as we are going to see in the next few paragraphs.

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

Close and accept

Following the DDD principles, identifying a micro-frontend becomes quite trivial.

Usually, there are 2 main scenarios to deal with on a day to day based: **greenfield projects**, usually very exciting for any developer but also more complicated because we don't have real information about our user base and how they would consume our content; **legacy projects**, where we have a tons of information (if we have diligently tracked our users behaviours using Google Analytics or similar tools) and therefore it's easier to rationalise a logical identification of bounded context across the entire platform *following our users' behaviours*

Sponsored Video

Watch to learn more

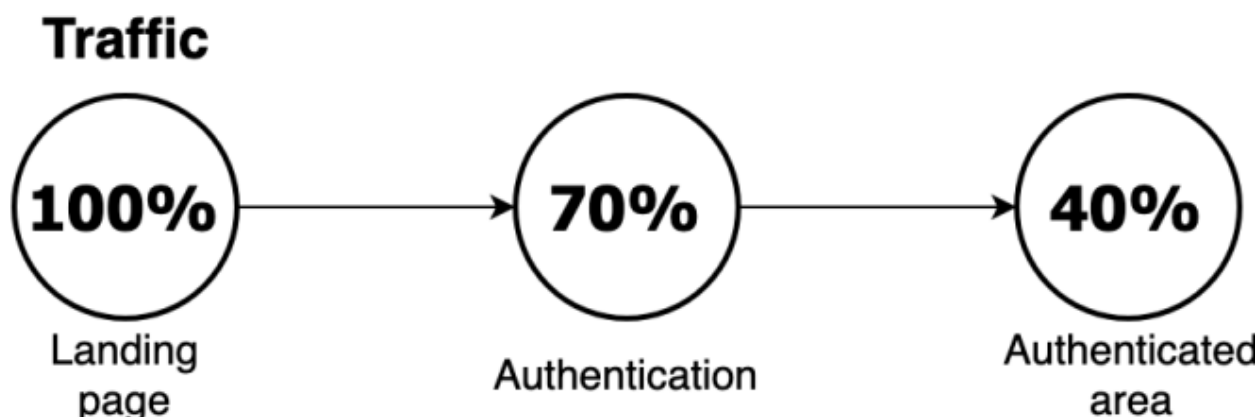
SPONSORED BY ADVERTISING PARTNER

See More

[REPORT THIS AD](#)

Having data to consult is one of the best situations we can aim for, understanding the users' behaviours allow us to easily identify the subdomains of our applications.

Let's assume we see a huge amount of traffic consulting the landing page, then 70% of those users are moving to the authentication journey (sign in, sign up, payment...), from here only 40% of the traffic subscribes to a service or use their credentials for accessing the service.



Users' behaviours example in an application

Those are good indications about our users' behaviours in our platform, DDD would suggest starting from the domain model of our application identifying the subdomains and their related bounded context and having behavioural data supports us on how to "slice" the frontend applications,

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.

To find out more, including how to control cookies, see here: [Cookie Policy](#)

Close and accept

have a faster experience because they won't download the entire application immediately and the 30% of users who won't move forward to the authentication area will have just enough code downloaded for understanding our service.

Obviously, mobile devices with slow connections can only benefit from this approach for multiple reasons: less KB to download, less memory used, less Javascript to parse and execute and a faster first interaction of the page.

Greenfield projects are a bit more complicated to manage, identifying micro-frontends upfront without knowing how our users interact with the platform could result in bad experiences but nevertheless, we have to find a way for structuring our micro-frontends architecture.

In this case, working closely with the product team or the subject experts could make a huge difference.

In my experience, any startup or medium-large organization have always a team or a person that has a clear idea of how the platform should behave, that knows inside out the core domain of an organization.

This person or team is key for understanding how the user should behave and therefore how to identify the domain model of our application as well as our micro-frontends.

It's essential to understand that a subdomains evolves with the business, never assume that is immutable!

REPORT THIS AD

In DAZN we have decided to split our Single Page Application into multiple subdomains based on the data retrieved in the past years, ending up with 5 different micro-frontends with a few components developed by external teams and embedded as dependencies in one micro-frontend.

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

Close and accept

- . *Catalogue*
- . *Playback*
- . *Sports data*
- . *User account*
- . *Help*
- . *Chat*

For instance, Playback and Sports Data are components living inside the catalogue micro-frontends, the complexity of those 2 subdomains lead us to assign a team dedicated to each of those subdomains.

Those components are published in an NPM private repository and are treated as an external dependency for the catalogue micro-frontend.

All the others are SPAs or single pages loaded by our **client side orchestrator**.

The power of local decisions

Working with subdomains allow us to assign a team to a specific area of our application, now stop for a moment and think how powerful could be this concept...

One of the key thing that I've always envy to startups is how fast they are capable to move and how quickly they take decisions on architecture, design or UX even.

When they need to take a decision, it's a matter of minutes or hours but not weeks like in large organizations where we need to have a quorum of people agreeing on the solution. If we think even further, a startup can react very quickly because they can take **local decisions**, in their case a local decision is a company decision, but if we extend this concept to medium-large organizations, dividing an application by subdomains allow us to have "multiple startups" inside an organization, therefore, empowering a team to take local decisions will allow to speed up the delivery, reduce the frustration and brings on the table interesting concepts like independent builds and deployments, less external dependencies, less frustration and more innovation.

The outcome of using DDD for identifying a ubiquitous language and subdomains would be creating a cross-functional team composed by frontend developers, backend developers, manual QAs and dev-in-test working closely to their product team/subject expert and being able to take a wide range of local decisions, from product decisions to infrastructure decisions, being responsible of the subdomain end to end.

Sponsored Video

Watch to learn more

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

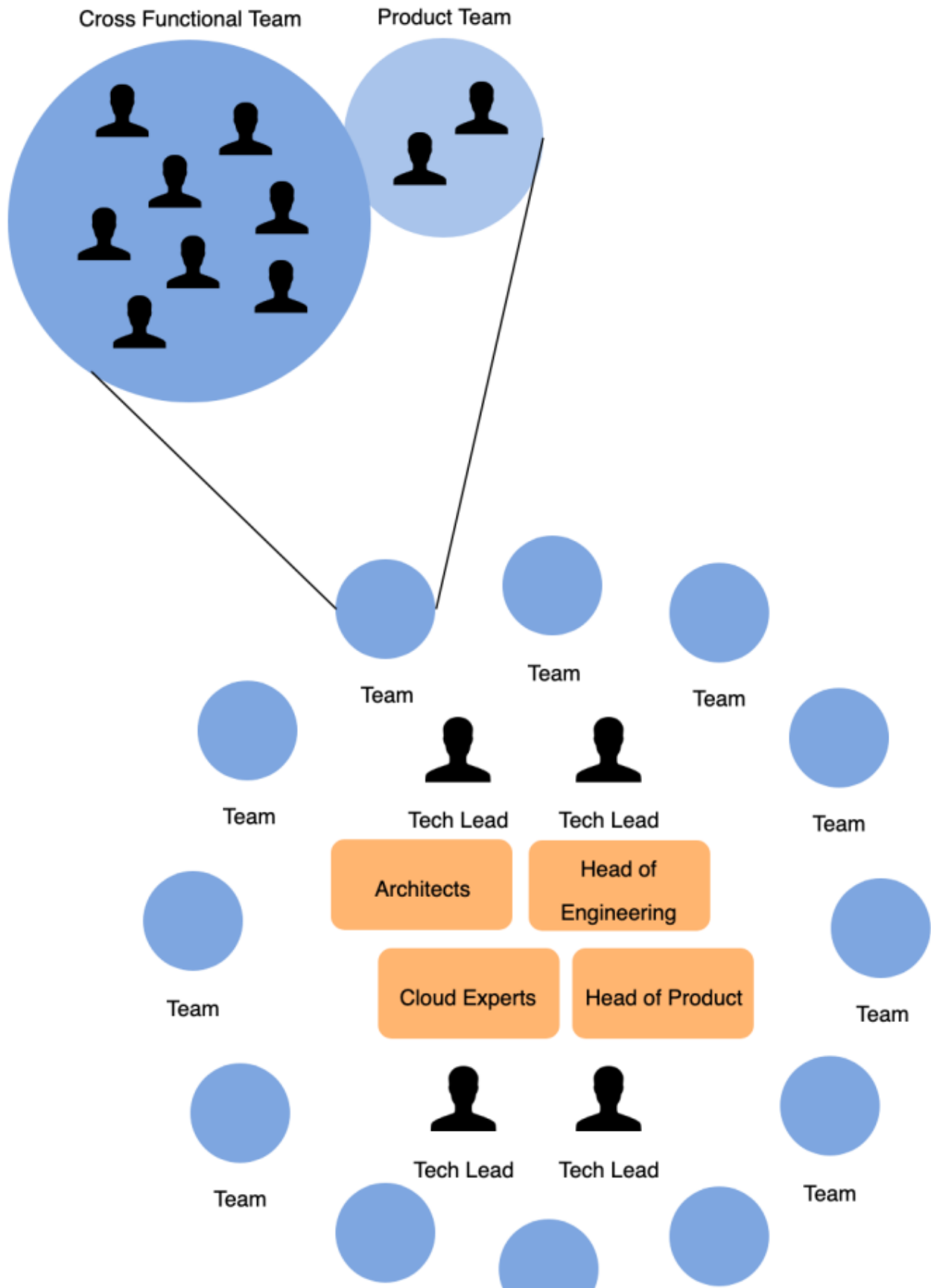
Close and accept

cision is taken locally, these teams have to collaborate with the rest of the organization using

services like architects, cloud experts and other functions inside the organization following the boundaries created by the heads of the technical department.

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#).

Close and accept



Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

Close and accept

In the past years, I read a lot about DDD and I found an interesting box inside *Domain Driven Design Distilled* book that caught my attention and I think is worth to share in this post to enforce the concepts explained in this paragraph:

Bounded Contexts, Teams, and Source Code Repositories

There should be one team assigned to work on one Bounded Context. There should also be a separate source code repository for each Bounded Context. It is possible that one team could work on multiple Bounded Contexts, but multiple teams should not work on a single Bounded Context. [...]

It is especially important to be clear that one team works on a single Bounded Context. This completely eliminates the chances of any unwelcome surprises that arise when another team makes a change to your source code. Your team owns the source code and the database and defines the official interfaces through which your Bounded Context must be used. It's a benefit of using DDD.

from Domain Driven Design Distilled—[chapter 2](#)

5 suggestions for dividing your frontend monolith

Last but not least, I think would be helpful having some takeaways of this post based on my experience and what I saw so far:

1. **Gather data:** if you have a legacy project you can use Google Analytics or similar services for understanding how your users are interacting with your application, you will find a clear idea how your user base is interacting with your application.
For greenfield projects, engage with your product team or customer, add GA or similar tools in your web application and via data validate the initial assumptions.
Remember, *bounded context and subdomains evolve with your business, are not defined once and set in stone!*
2. **Talk with the domain experts:** invest time with your product team or the domain experts in your company, understand their point of view, their roadmap, how they think to evolve

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

Close and accept

lowing DDD there are some teams crossing multiple subdomains, *make the bold decision re-viewing the internal organization, the entire business will benefit from it!*

4. **A micro-frontend could be a single page or a SPA or SSR:** as long you are following DDD for identifying your subdomains, a micro-frontend may end up to be represented by a single page like in the case of a landing page, or a more complex solution based on a Single Page Application architecture or a Server Side Rendering one.

Components risk being not representative of a subdomain because tightly linked to the container where they are nested, therefore the overlap of multiple contexts could cause more issues than benefits.

5. **Invest the right amount of time at the beginning of your project:** designing an architecture upfront is not the best way for starting a project, usually an architecture should work iteratively, therefore we should start designing “just enough” and slowly but steady we enhance the design based on additional information we found engaging with the product team, developers and users.

When you are identifying the different subdomains of your application invest enough time because this decision could impact how to structure the tech teams as well as how much communication overhead your company is going to spend due to dependencies between teams

Advertisements

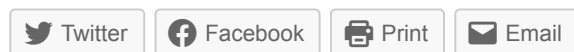
S

V

SF

REPORT THIS AD

SHARE THIS:



Like

Be the first to like this.

Orchestrating micro-frontends

April 12, 2019

In "Javascript"

Building Micro-Frontends... the book

February 24, 2020

Micro-frontends, the future of Frontend architectures

April 1, 2019

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use. To find out more, including how to control cookies, see here: [Cookie Policy](#)

Close and accept

PUBLISHED BY

**lucamezzalira**

I'm the VP of Architecture at DAZN with more than 15 years of experience, a Google Developer Expert on Web Technologies and the London Javascript community Manager I had the chance to work on cutting-edge projects for mobile, desktop, web, TVs, set top boxes and embedded devices. I am currently managing DAZN, a sports video platform based on the cloud with millions of users that are watching live and on-demand contents. I'm the author of Front-End Reactive Architectures published by APress:

<https://goo.gl/ywAmsx> I think the best way to use any programming language is mastering their models, that's why I spent a lot of time studying and researching on topics like OOP, Functional and Reactive programming. In my spare time, I wrote for national and international technical magazines and editors, I'm also a technical reviewer for APress, Packt Publishing, Pragmatic Bookshelf and O'Reilly. I was speaker at: O'Reilly media webinars, O'Reilly Software Architecture (San Francisco & London), O'Reilly Fluent (San Jose), O'Reilly Oscon (London), Google Developers Summit (Krakow), Google DevFest (London), Frontend Devs Love (Amsterdam), Voxxed Days (Belgrad & Bristol), JeffConf (Milan), International Javascript Conference (Munich & London), JS Poland (Warsaw), Code Europe (Wroclaw), JSDay (Verona), CybercomDev (Łódź), Jazoon Conference (Bern), JDays (Göteborg), Codemotion (Milan), FullStack Conference (London), React London UG (London), Scrum Gathering (Prague), Agile Cymru (Cardiff), Scotch on the rocks (Edinburgh & London), 360Max (San Francisco), PyCon (Florence), Lean Kanban Conference (London), Adobe Creative Suite CS 5.5 - Launch event (Milan), Mobile World Congress (Barcelona)

[View all posts by lucamezzalira →](#)

📅 May 21, 2019 👤 lucamezzalira 📁 HTML5, Javascript, micro-frontends, Microservices 🔖 best practices, coding, frontend, javascript, micro-frontends, Microservices, web, web applications



Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

Close and accept