**How To Implement Micro-Frontend Architecture With Angular**



Modern web applications are becoming big and more complex and sometimes managed by different teams. Your application might have features developed by different teams and you want to release only certain features into production before delivering the entire application. How do you manage different teams, different release timelines if you have one repo?

Most of these complex apps live on the client-side which makes it harder to maintain. There are some other issues as well with this monolithic big fat application.

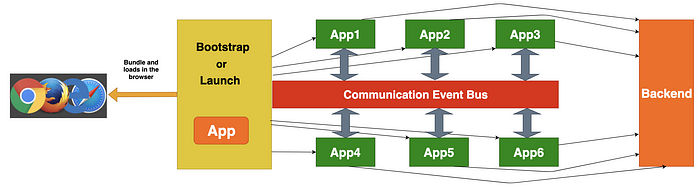
* ***Introduction***
* ***A Journey Into Mirco-Frontends***
* ***Advantages of Mirco-Frontends***
* ***Features of Mirco-Frontends***
* ***How Do We Split Apps***
* ***Different Approaches to Micro-Frontends***
* ***Micro-Frontend Frameworks***
* ***Example Micro-frontend Project With Angular***
* ***Summary***
* ***Conclusion***

**Introduction**

Micro-frontends are small applications mostly divided by subdomain or functionality working together to deliver a larger application. Before diving into Mirco-frontends, we will understand what are micro frontends and why we are talking about those.

Usually, projects come in different sizes and different requirements. If your project is simple enough to have two or three pages and can be maintained by a single team, you don’t have to think about these Micro-frontends. You can just implement with any of your chosen framework such as Angular, React, or Vuejs.

But, this is not the case all the time. Sometimes your frontend app might be a small part of another big application or your app consists a lot of sections and features which are developed by different teams or your app is being released into production feature by feature with developed by separate teams. If you are in one of these situations then you need to think about Micro Frontends. Let’s look at it in the below diagram.



**Micro Frontends Architecture**

As you see in the above diagram, we have 6 frontend apps working together to deliver the large application. The communication between these apps can be done with an event bus, window object, or publish/subscribe methods. Each app can be implemented by a separate team and any framework. Each app can talk to their backends or endpoints individually. There is a bootstrap/launch app that loads all the apps and mounts and unmounts in the DOM depending on the user interaction or routing.

**A Journey Into Mirco-Frontends**

We will first understand why do we need this in the first place. Let’s have a look at the present technologies that we use for user faced web applications.

coming soon!

**Advantages of Mirco-Frontends**

Here are the advantages of this architecture.

**Apps are small**

Obviously, apps become small when we split the large application by sections, pages or features.

**Apps are independent**

Since all the apps are divided and developed separately these are independent of each other

**Apps are easier to understand**

Apps are easier to understand because they are small and developed by a single team.

**Apps are easier to develop and deploy**

As these apps are small in nature and developed by a single team its very easy to develop and deploy. We can even deploy independently.

**Apps are easier to test**

We have to write thousands and thousands of unit tests for larger applications and takes forever to run. This makes out deployment process slower. When it comes to micro frontends each app has few unit tests and executes its own unit tests and can be run independently.

**Apps development becomes faster**

The whole development becomes faster and easier because of separate teams.

**CI/CD Becomes easier**

Each app can be integrated and deployed separately and this makes the CI/CD process a lot easier. When we fix the app or introduce a new feature we don’t have to worry about the entire application since all the features are independent.

**Independent Stacks and versions**

We can choose our own stack for each app but this doesn’t happen so often But, we can have different versions of the same stack. For example, Some teams have the flexibility and time to introduce and test newer versions of the same stack.

**No Shared Code**

In large applications, we tend to share code across features but, this doesn’t scale well and introduce a lot of bugs and interdependency as the app grows bigger and bigger. This doesn’t apply with the Micro-frontends as we do not share the code unless it is a dumb component.

**Can change architecture easily without touching old one**

Sometimes we have to extend the old architecture but we might not have the developers to implement or extend the architecture. With the Micro frontends approach, we can develop the new feature with the latest stack and deliver independently.

**Features of Mirco-Frontends**

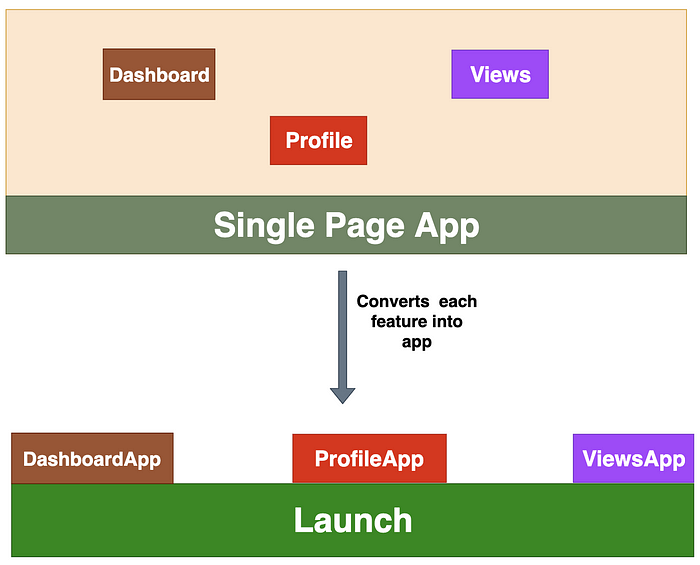
* Each frontend represents a specific feature or subdomain of the entire application
* Each frontend can be implemented with a separate team.
* Each frontend can be implemented with different technology.
* They cannot share logic and its independent of each other.
* Each Frontend can be owned by a single team.

**How Do We Split Apps**

Let’s see how we can split large applications into micro frontends. There are no specific criteria to divide the apps and we can separate in a number of ways depending on our needs. We will see all the possible ways that we could divide apps.

**By Feature**

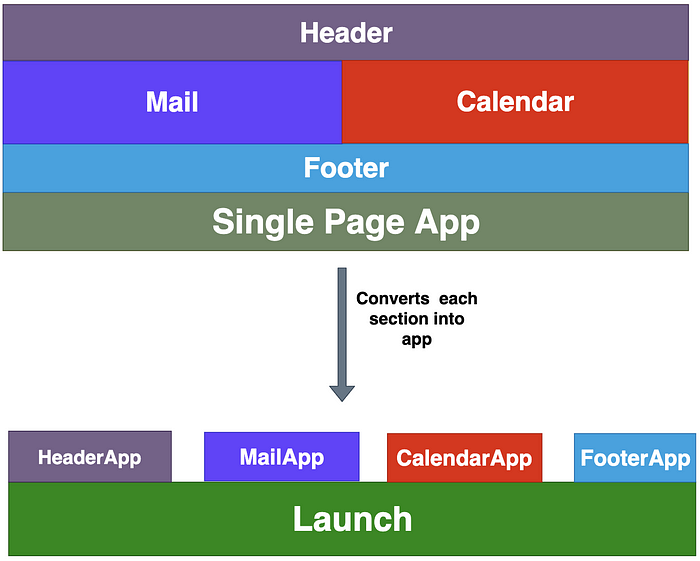
This is the most common method since we can easily divide the features of the app. For example, if there are three features for the app Dashboard, Profile and views we can make each feature as a separate app and mounts and unmounts in the DOM with the help of ***Launch.js.***This Launch.js can be a separate app or just a simple javascript app.



**By Feature**

**By Section**

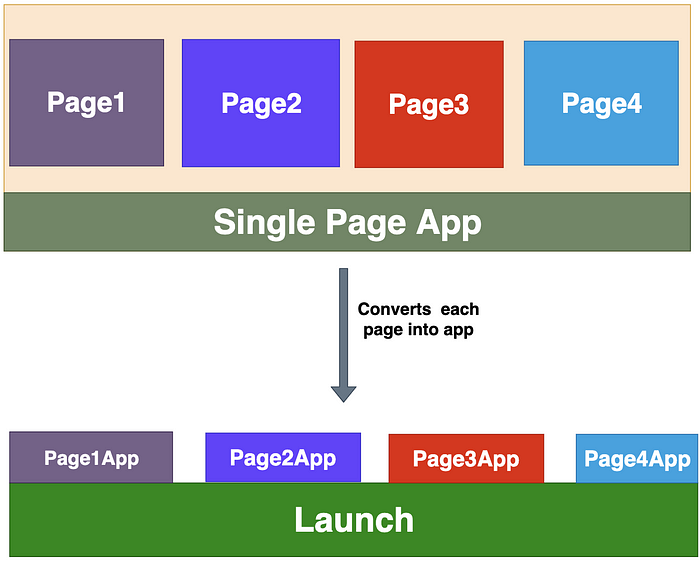
Some of the apps have so much functionality with each section, for example, coinbase, Gmail, etc. We can implement each section as a new app in that scenario.



**By Section**

**By Page**

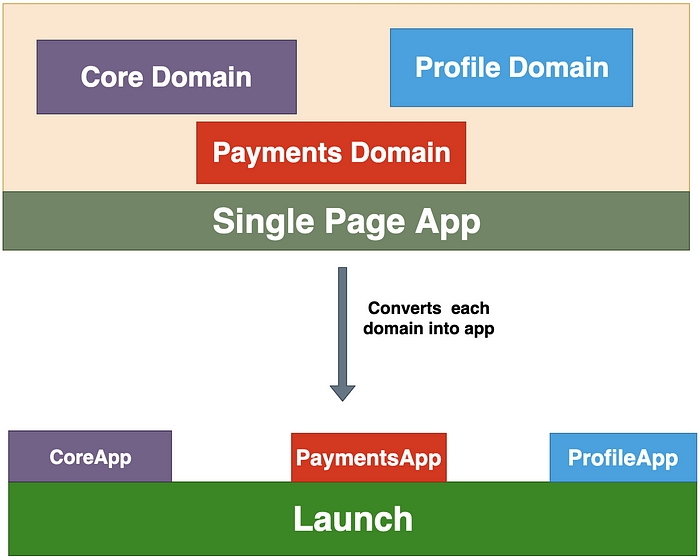
Some app’s functionalities are divided by page. Each page has some functionality that can be independent. We can divide these apps by page We have four pages in the below diagram. we can make four apps out of this.



**By Page**

**By Domain**

Splitting app based on the domain is also one of the most common approaches.



**By Domain**

**Different Approaches to Micro-Frontends**

There are many approaches that you could implement Micro-Frontends.

* ***Iframes***
* ***Through NGINX***
* ***Web Components/Angular Elements***
* ***Angular Libraries***
* ***Monorepos***
* ***Customized Orchestrator***

**Micro-Frontend Frameworks**

Micro frontends have been implemented for at least two years and it is still a green field. Have you ever wonder are there any frameworks or libraries to implement these and makes our job easier. The answer is yes and there are a couple of libraries or frameworks

* [single-spa](https://single-spa.js.org/)
* [frint.js](https://frint.js.org/)

**single-spa**

single-spa is a javascript framework for front-end microservices and can be implemented with all three popular frameworks/libraries such as Angular, React and Vue.js. It can lazy load the applications based on the need and You can check out their website for more information.

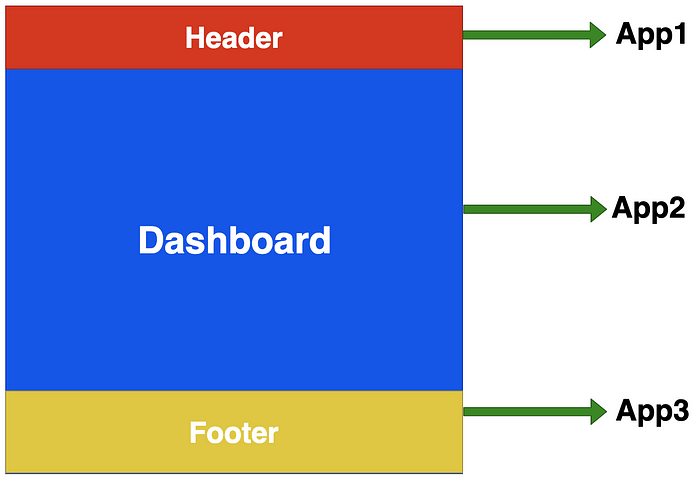
**frint.js**

frint.js is a Modular JavaScript framework for building Scalable & Reactive applications. It doesn’t support Angular yet but it supports React. If you are building a reactive application from scratch and you are just starting, this is the framework for you. You can check out their website for more information.

***Example Micro-frontend Project With Angular***

With all this information let’s build an example Angular project with the help of a [single-spa](https://single-spa.js.org/) framework. I would like to build a simple app for the demonstration and I will do the full example with all the features in another post.

We will split this app by section as shown in the below diagram. We are going to implement **4** apps in total: HeaderApp, DashboardApp, FooterApp, and root application.

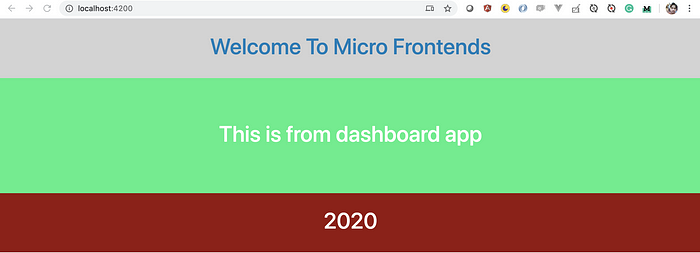


**Micro Frontends**

Here are the four repositories for four apps. You can clone these run separately on your machine.

// root app runs on port 4200  
git clone <https://github.com/bbachi/micro-root.git>  
npm install  
npm start// micro header runs on port 4300  
git clone <https://github.com/bbachi/micro-header.git>  
npm install  
npm start// micro dashboard runs on port 4202  
git clone <https://github.com/bbachi/micro-dashboard.git>  
npm install  
npm start// micro footer runs on port 4201  
git clone <https://github.com/bbachi/micro-footer.git>  
npm install  
npm start

You can access the whole application on [***http://localhost:4200/***](http://localhost:4200/)



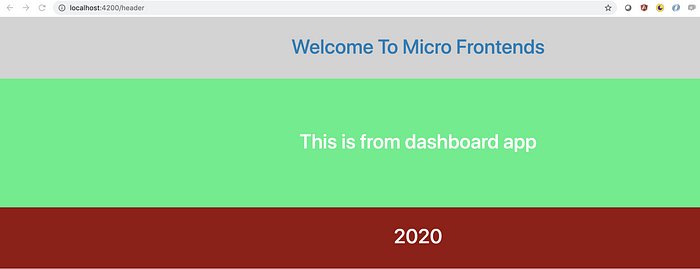
**Running all four apps together to make it one app**

Here is the micro-root app index HTML file. We are importing all the three apps on line 10 and registering the apps with the appropriate name and location. Since we are loading all the apps on page load, we are not defining any specific context paths.

**root app index.html**

You can give /header location path so that the header is loaded when browser URL navigates to /header. Let’s test it out.

**with /header**



**Header only load with the route /header**

***Summary***

* Micro frontends are the implementation of microservices for the front end
* Modern web apps are becoming more and more complex and most of the code lives on the client-side.
* With this Micro frontend approach, we can make it easier to understand, develop, test and deploy larger applications
* There are so many advantages to using this approach. At the same time, don't use it for every app especially small apps.
* Each app can be developed independently, Each app can use a different stack, Each can be owned by a single team or multiple teams and finally, they don’t share logic.
* It’s tempting to use a different framework for each app but not recommended. Remember, this approach should make your life easier.
* We can split the apps by domain, feature, page, and section.
* There are two frameworks currently which implement this architecture: [single-spa](https://single-spa.js.org/) and [frint.js](https://frint.js.org/).
* You don’t need frameworks to implement but, using one of these makes your transition from monolith to micro frontends easier.
* Don’t use it for small apps.