Micro Frontends

Micro frontends have been the buzzword for couple of years. It's similar to micro services and it promises that we can decompose the UI into blocks. The basic idea is that you want to take this big massive monolith, that you have for your front end code base and break it down to a smaller micro apps that can be deployed independently, making it easy for all of us. We want to move fast, sustain the pace, reduce our code complexity, reduce our technical debt and reduce our internal communication overhead. If you manage to have all of these factors, you will have a good code base. You will be able to move fast and make awesome products.
Let's look at the pros and cons here:

PROS:

Separate Deployment
It means once your development task is complete, you can immediately deploy it. No blocking as there is no dependency on another team. You don't have to wait until the other team completes their tasks. You can just publish it and user will see the updated app immediately.

Different technology

You can mix and match technologies. Not because you want to, because you have to when you are evolving a system of 20 years. You cannot predict the future and it is very likely that in some years a new technology will show up that fits better for your use case there.

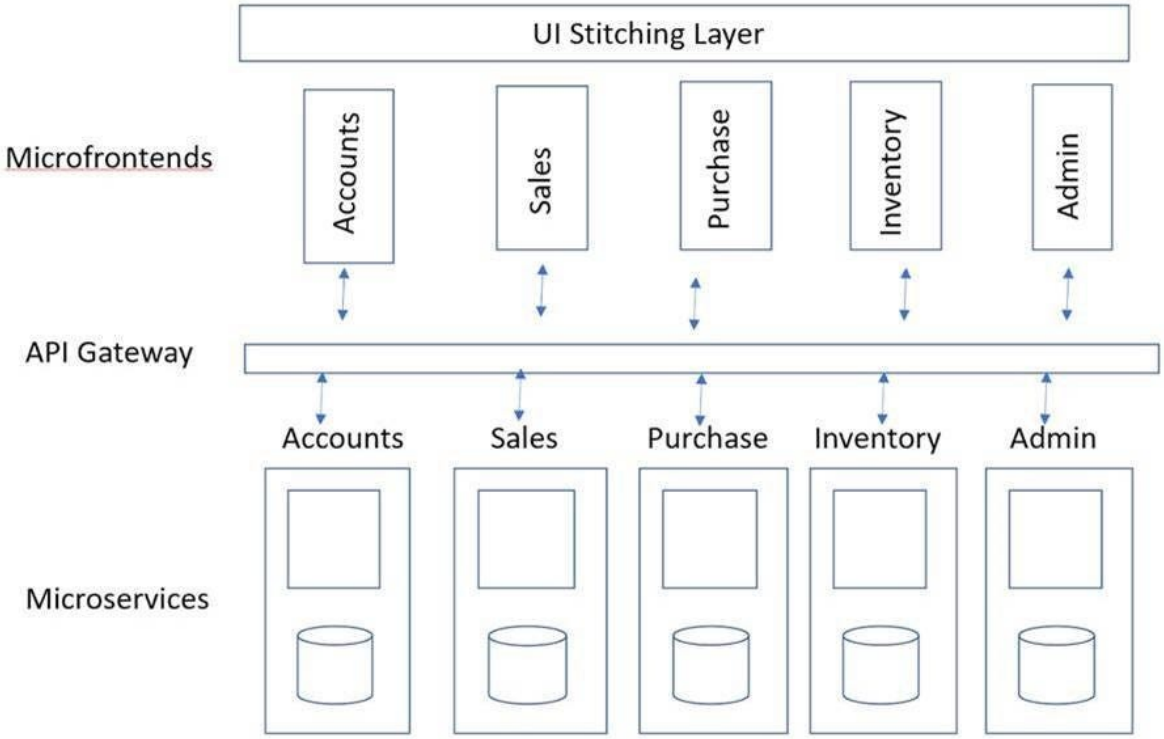
Better coordination
Bigger teams can't coordinate easily, as it is hard to communicate and work with a lot of people. It is a fact that in most cases working in a small team is always more efficient.

Less complexity
Having a tiny system leads to less complexity as the team can focus only on their micro apps without worrying about the big monolith app.

CONS

Duplicate Code
Some parts of you app might have lots of end points and those have to communicate with each other. It means that each of the micro apps, has to communicate with those API endpoints separately, which can cause duplicate code in each of the micro apps. Whereas, in big UI code base approach, all of the API communications might happen via a common UI service layer.

UI Composition
For us, it is better to have multiple micro applications that solves a specific business problem. End users, however is not interested in working with multiple applications; they want to have one common app that they can use. This means that we have to provide a thin app layer which stitches the micro apps together. This would allow the user to think that it is a one big application.
Image 2: Micro Frontend architecture diagram



Therefore, let's define the objective first. The objective is very simple in theory at least. All your front end code runs into browser on a single web page, now what you want to do is, take all of the micro apps you have created and bring them together into the same web page and give a seamless user experiences. On top of the UI create a stitching layer that effectively take all your micro apps and bundles them together before serving them to the user.

UI composition solutions

When we talk about UI composition there are several solutions for this:

UI composition with Hyperlinks

Hyperlink is a quite a mature technology. They have been here for over 30 years and we haven't heard a single bad word about hyperlinks. In this hyperlink approach you will end up with several single page applications with several micro apps. You can navigate back and forth using those links, and even though this sounds awkward this could be a right solution for your needs. An example is the office 365 home page, which allows users to navigate different apps like PowerPoint, Word, Excel, OneNote etc.
Let's look at the pros and cons here:
PROS

- Browser / Server is your stitching layer
- Well supported by all browser
- No jS/CSS conflicts. Each App is separate

CONS

- Complete page loads when switching between micro apps, losing states
- Each app loads a complete bundle every time.
- Chances of missing consistent look and feel.

UI composition with Iframe

Sometimes Iframe is the right solution for your needs, not because it is not the most popular stack but because it provides isolation. Isolation means applications running under one iframe cannot harm another application which is running another iframe. This is important if you have a plugin system, for a third-party vendor, and you want to include it in your application. In this scenario you will end up loading each micro apps in iframe. Stitching layer shows the iframe according to the route being opened.
Let's look at the pros and cons here:

PROS

- Seamless switching between micro apps
- No JS/CSS conflicts
- Each app in its own iframe

CONS

- Route need to be configured in the container
- Each iframe creates an extra browser process
- Each app loads complete bundle every time

UI composition with Single SPA

If you don't need an iframe, you can boot strap several single page applications. Which means one index.html, where you can load multiple single page application and you can consider the lazy loading for on demand support. You can use popular library called single-spa. Its open source and it works similarly to It loads each single page applications into the DOM.
The way it works, is that it takes the root element and places it in the document body and initializes them as they require. In order to get this running, you need to integrate this library to your code base so it has recipes for angular, react, view and whatever else you are using. Actually you can run all of these frameworks side by side if you want, and it's pretty easy. It does not take too long. You have to create a separate container which essentially links all of this and renders everything in that said container.
Let's look at the pros and cons here:

PROS

- Micro apps loads on a single DOM (no extra refresh)
- Relatively easy setup, most complexity handled by the library

CONS

- All application shared the global namespaces
- And same global browser object like moment.js
- Each app loads a complete bundle every time
- Deployment needs custom tooling

UI composition with Web Component

The final approach and this is slightly different. It is the web component approach. The idea here is very simple, you take all of your JavaScript code (can be any framework) and compile it down to custom elements. Now additional advantages you will get here is you can start sharing your micro apps within the same view.

A web component provides a unified API, you can data bind information to your properties and you can get out events. This is something all modern frameworks are doing.
I know the web component is not the first thing you are thinking about, when are talking about micro apps, because web component is a tiny part of an application and micro app is bigger. It is a big application, but to be honest, just think about angular, view, react. All these frameworks based application is a component consisting of multiple components which in turns has components inside it and so it is not that unnatural when we consider web components.
Besides, this you can create web components which wraps different technology inside this, and you can load it on demand.
Web component is a framework independent component which means you can write it in a framework and use it in another framework or you can use it in vanilla JS. One nice thing about using web component is that they can very easily load into browser in a dynamic fashion.
Let's look at the pros and cons here:

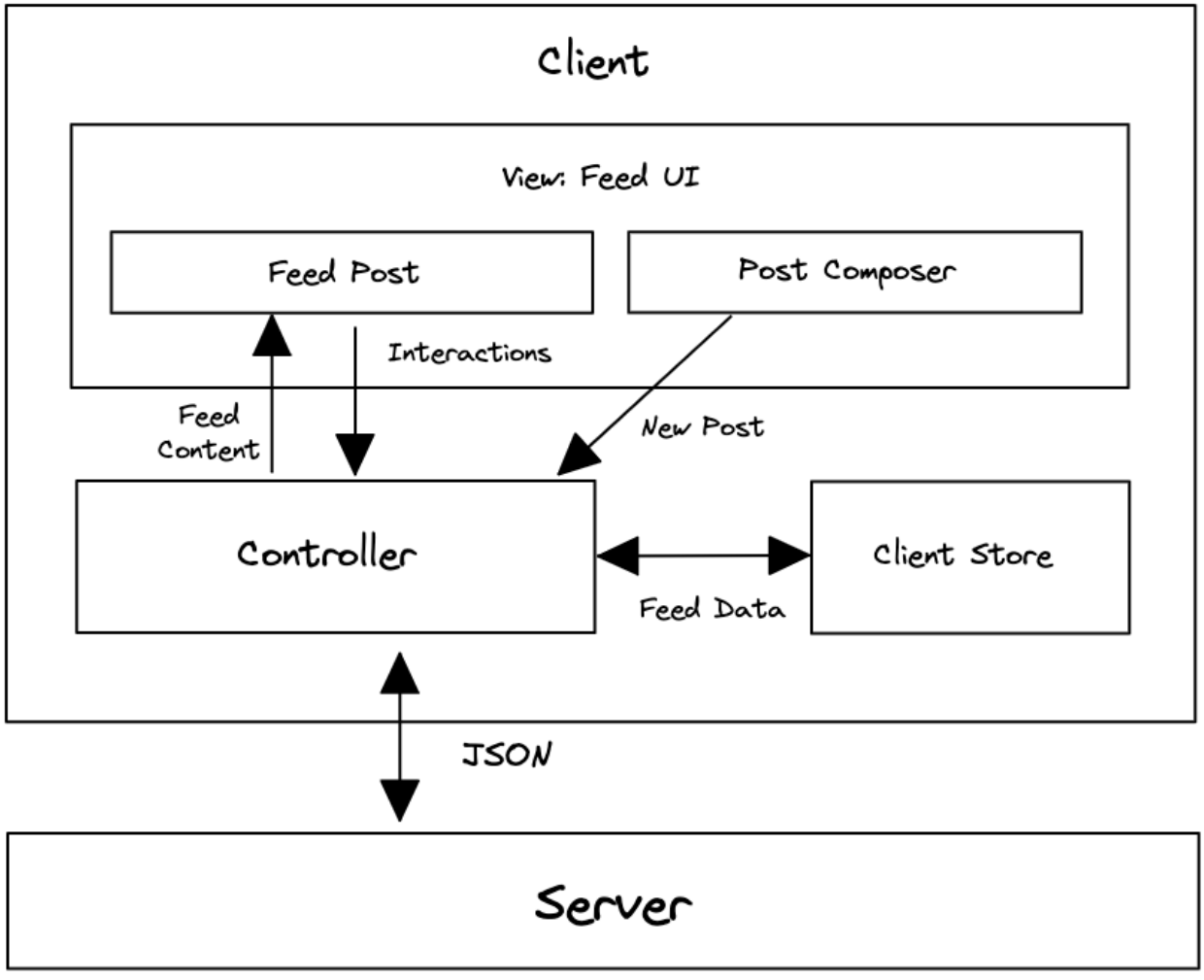
PROS

- Multiple micro apps shared the same view
- Everything compiles to web components, so there is no JS framework restriction

CONS

- Shared JavaScript environment
- Chances of duplicate code across different component.

ARCHITECTURE



UI Wireframe