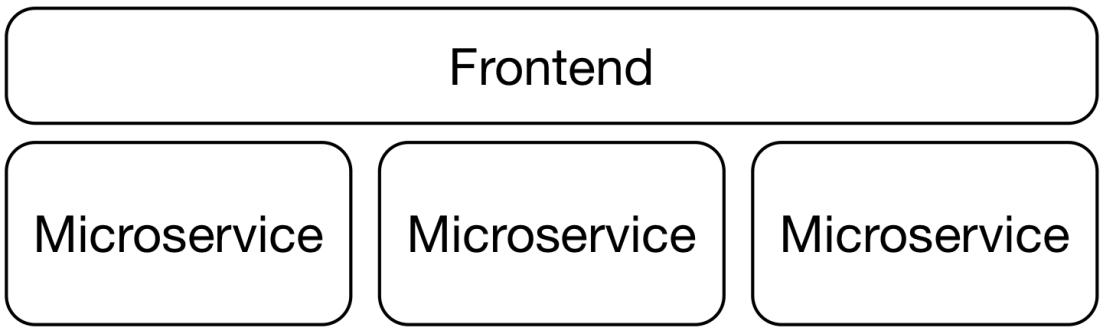
Frontend: Monolith or Modular?

In this lesson, we'll debate the pros and cons of a monolithic frontend and a modular frontend.

WE'LL COVER THE FOLLOWING

The following drawing shows a frontend monolith which serves as the frontend for multiple microservices.



Frontend Monolith with Microservices Backend

Option: monolithic frontend and backend

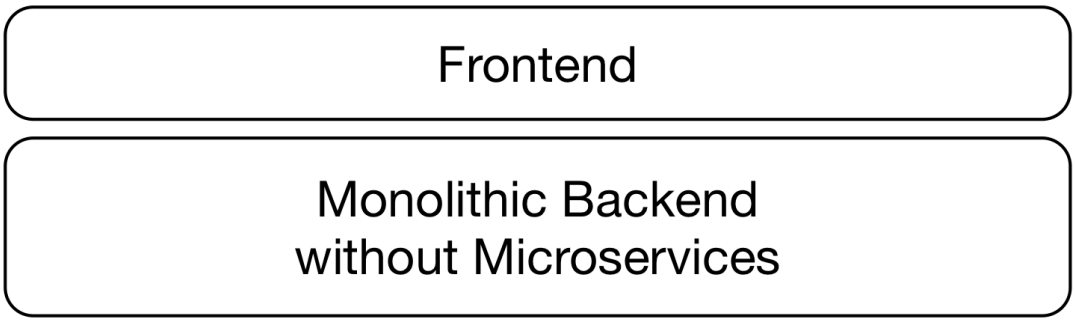
Doing without modularization in the frontend is inconsistent. On the one hand, a backend is modularized into microservices which involves a great

deal of effort.

On the other side is a monolithic frontend, which is an architecture that must be questioned. The result does not have to be a modularized frontend. It may also turn out that a monolithic frontend together with a monolithic backend also meets the requirements. Then you can save the effort for the modularization of the backend into microservices.

This can be the case, for example, if the advantage of separate deployment and further decoupling in other areas is not important after all.

The following drawing shows a frontend monolith which serves as frontend for a monolithic backend.



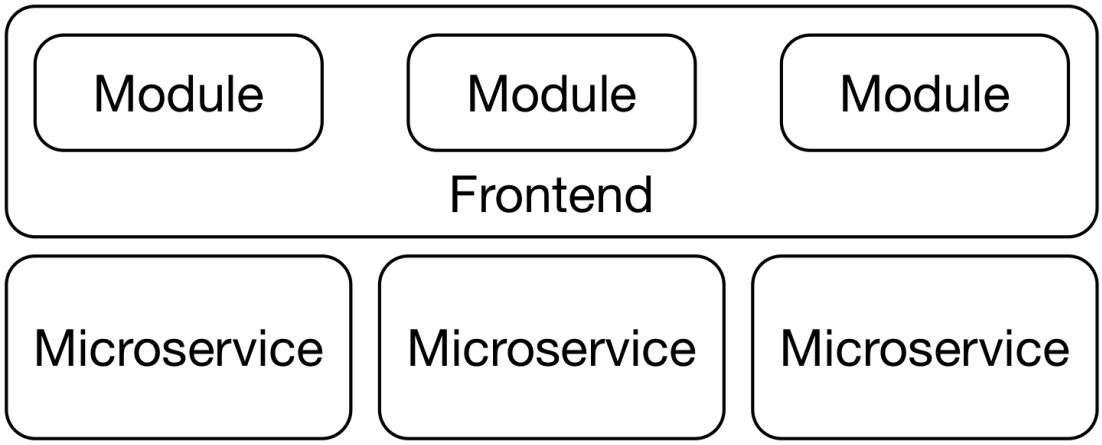
Frontend Monolith with Backend Monolith

Option: modularly developed frontend

Of course, a **frontend deployed as a monolith can be developed modularly**. Unfortunately, experience shows that modular development often still leads to an unmaintainable, non-modular system in the end because the boundaries between the modules dissolve over time. The boundaries between modules that can be deployed separately, such as microservices, are not easy to circumvent so that modularization is secured in the long term.

Nevertheless, each microservice can, for example, be assigned a module in the frontend in order to decouple and parallelize the development. However, in that case, the frontend monolith still has to be deployed as a whole. Separate deployment is one advantage of microservices over other modularizations.

The following drawing shows a frontend monolith that is divided into modules.



Frontend Monolith Divided into Modules

Reasons for a frontend monolith

A frontend monolith can be a good choice in some circumstances.

Native mobile & rich client apps

Native mobile applications or *rich client applications* are **always deployment monoliths**. In other words, they can be delivered only as a whole.

Mobile applications even need to pass a review process in the app store in case of an update, so deployment takes even longer. However, this mechanism can be "undermined" to a certain extent. An app can display web pages. These can be provided by a microservices system that uses web frontend integration.

Frameworks such as Cordova even allow web applications to take advantage of proprietary mobile phone features or to offer a web application for download from the app store. This means that compromises can be implemented between native applications and web applications.

SPAs

Single page apps (SPAs) **can also only be deployed as a whole**. There are enough alternatives to SPAs for web applications to completely modularize an application in the frontend.

The boundaries are fluid; SPAs can contain links to other sites or other SPAs and display HTML generated by a different system. In this way, SPAs can be integrated into the frontend. But in practice, **SPAs typically lead to a**

integrated into the frontend. But in practice, **SPAs typically lead to a frontend deployment monolith** despite these theoretical possibilities.

Single team

One reason to develop a monolithic frontend would be if there was *a team* that was dedicated to frontend development. Each team should be responsible for one component.

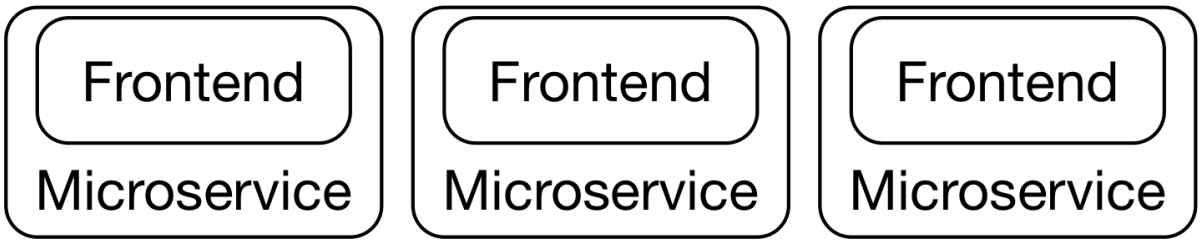
If you do not want to break up the frontend team because it is too big an organizational change, or because the team is working in a different location than the other teams, a monolithic frontend can be the best approach.

Migration

Finally, the *migration* of an existing system might be particularly easy if the monolithic frontend remains intact.

Modularized frontend

The alternative to a monolithic frontend is a fully modularized one. The drawing below shows a modularized frontend where each microservice has its own frontend. Like the backend, the frontend is part of the separately deployable microservices.



Modularized Frontend

Advantages

Such a modularization of the frontend has many advantages.

- Microservices and self-contained systems can be **independent concerning their domain logic**.
- If the microservices contain a part of the modularized frontend, then **a change in a domain can be implemented by modifying and deploying just one microservice**, even if changes are necessary at the frontend.
  - If the UI is a monolith, on the other hand, many changes to the

- If the UI is a monolith, on the other hand, many changes to the domain logic require modifications to the UI monolith so that the UI monolith becomes a main focus of change

Modularized frontend and frontend integration

To combine the separately deployed frontends into a complete system, the **frontends must be integrated**.

Modularization is intended to decouple development. Nevertheless, an integrated system must be created. In other words, a frontend modularized into different microservices is only possible if an approach for frontend integration has been chosen.

For this, different technical approaches are possible, which are the focus of this and the following chapters.

QUIZ

1 What is the argument against a monolithic frontend for multiple microservices?

COMPLETED 0%

1 of 4

In the next lesson, we'll discuss a few options for frontend integration.