Chapter Conclusion

That's it for this chapter! Here's a quick summary.

WE'LL COVER THE FOLLOWING

This example system is deliberately presented in the first chapter on frontend integration. It shows how much is already possible with a simple integration via links.

Only for the *postbox* do you need some JavaScript. Before using the advanced technologies for frontend integration, you should understand what is already possible with such a simple approach.

The example integrates different technologies. In addition to the `Node.js` systems, there is also a Java/Spring Boot application that seamlessly integrates into the system. This demonstrates that a **frontend integration results in only a few limitations regarding the technology choice**.

ROCA

ROCA helps especially with this type of integration. The **microservices can be accessed via links, making integration very easy**. At the same time, the applications are largely decoupled in terms of deployments and technology.

An application can easily be deployed in a new version and not affect the other applications. The applications can also be implemented in different technologies.

At the same time, the ROCA UI is comfortable and easy to use. Compared to a single page app (SPA), there are no compromises in user comfort.

Assets

Finally, the application shows how to handle assets, in this case by using a common `Node.js` project. As a result, **each application can decide for itself when to adopt a new version of the assets**. This is important because otherwise a change of assets is automatically rolled out to all applications and might cause problems in the applications.

However, several versions of the assets should be used only temporarily on the web page. After all, **the design, look, and feel should be uniform**. Dealing with the asset project in such a way that not all services always use the current version is only meant to minimize the risk of an update but **must not lead to long-term inconsistencies**.

However, the asset project also ensures that all web pages contain jQuery in the version that the asset project uses. Thus, the asset project limits the freedom of individual projects with regards to JavaScript libraries.

Self-contained systems

Unlike self-contained systems (see chapter 2), this solution uses a **common backend**. With an SCS, the logic should also be part of the respective SCS and not be implemented in another system.

However, it is still possible to use some SCS ideas even if all systems share a common backend. The systems do not have to deal with logic and storing data as much as an SCS, because they are in the backend. Thus, this system shows **how a well-modularized portal for a monolithic backend can be implemented**.

But this approach also presents challenges. Any changes to the system will probably affect one of the frontend applications and also the backend. Therefore, the development and deployment of the two components must be coordinated.

Benefits

- Loose coupling
- Resilience
- No additional server components
- Low technical complexity
- Links often enough

Challenges

- Uniform look and feel

That's it for this chapter! Let's discuss server-side integration using edge side includes (ESI) in the next chapter.