Challenges

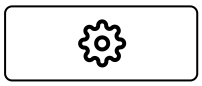
In this lesson, we'll look at some challenges pertaining to the use of self-contained systems.

We'll cover the following

SCS describes an architectural approach that is more narrowly defined than microservices. Therefore, **SCS cannot be the approach for solving all problems**.

Limitation to web applications

The first limitation of SCSs is that **they are web applications**. Thus, SCSs are not a solution when a web UI is not required.



However, some aspects of SCSs can still be implemented in a scenario that does not involve web applications: the clear separation of domains and the focus on asynchronous communication.

If a system is to be developed that offers only an API, an architecture can be created that provides at least some of the advantages of SCSs.

Single page app (SPA)

A **single page app (SPA)** is usually an application written in JavaScript that runs in the browser and it often implements complex UI logic. Applications such as Google Maps or Gmail are examples of applications that are highly complex and must be very interactive. SPAs are ideal for these cases.

Disadvantages

However, SPAs also have **disadvantages**:

- Since logic can be implemented in an SPA, in practice **business logic often also leaks into the UI**. This makes further development difficult, because logic is now implemented on the server and the client and thus at two different points in two, usually different, programming languages.
- The **load times of an SPA can be higher than the load times of a simple website**. Not only must HTML be displayed, but the JavaScript code must be loaded and started. Loading times are very important in some areas like e-commerce, because the user behavior of customers depends on loading times.

SCS based challenges

SCS based challenges

With SCSs, these problems are complemented by **further challenges** such as:

- An **SPA for the entire system** would mean that there is a **common UI**. This is forbidden in SCSs.
- An **SPA per SCS** is one possibility for providing each SCS with its own UI.
    - In this case, switching from one SCS to another means starting and loading a new SPA, which can take some time.
    - Moreover, it is not easy to split one SCS into multiple SCSs, because the SPA also needs to be split up. However, a division can be important for the development of the system in order to adapt to the architecture.

Due to the high popularity of SPAs among developers, the contradictions between SPAs and SCSs in practice are a major reason for difficulties in implementing SCSs.

An alternative is ROCA (https://www.educative.io/collection/page/10370001/5441945024331776/5676079995944960). It establishes rules that stand for classic web applications and are much easier to use with the SCS idea.

Mobile applications

SCSs are not suitable as a backend for mobile applications. The mobile application is a separate UI from the backend so that UI and logic is separated and the concepts of SCS are violated.

Of course, it is still possible to implement an SCS with a web UI that also provides a backend for a mobile application.

There are different **alternatives**. Let's look at a few.

Web application

A *web application* can be developed instead of a mobile application. It can be responsive so that the layout adapts to a desktop, tablet, or smartphone.

Compared to a native app, the advantage is that there is no need to download and install an app from the app store. The number of apps that a typical mobile user has is quite low. Therefore, the installation of an app can be a hurdle.

Thanks to HTML5, JavaScript applications can now use many of the mobile phone features. Websites such as https://caniuse.com/ (https://caniuse.com/) show which features are provided by which browser. When the decision is made in favor of a web interface, real SCSs can be implemented.

Web application with framework

A *web application can be implemented with a framework* such as Cordova (https://cordova.apache.org/) that takes advantage of the smartphone's specific features.

There are other solutions based on Cordova. Therefore, you can still create a real SCS, but have the app be downloaded from the app store, use all the features of the smartphone, and have it behave like a native app.

Of course, it is possible to implement parts of an app with such a framework and implement the rest as a truly native app.

Native app

Finally, a *native app* can be created where, for example, the backends can offer a REST interface. If the backends do not also implement a web

offer a REST interface. If the backends do not also implement a web interface, they are not SCSs. Even in this case, it is possible to divide the logic into largely independent services and use asynchronous communication between the services in order to achieve many advantages of the SCS architecture.

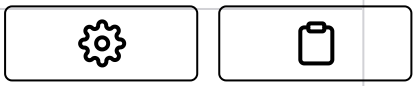
Look and feel

Dividing a system into several web applications quickly raises the question of a uniform look and feel. **A uniform look and feel can only be achieved with a macro architecture decision**. This also applies to SCSs, of course.

QUIZ

1 Suppose you are developing an app that has no UI. Is it possible to migrate it from a monolithic system to one that leverages the advantages of an SCS-based one?

- A) Yes
- B) No, SCSs strictly need a UI.



Submit Answer

Question 1 of 2
0 attempted

Reset Quiz

In the next lesson, we'll study the benefits of using SCSs.

← Back

SCSs and Microservices

Next →

Benefits & Variations

Mark as Completed

Report an Issue