Variations

In this lesson, we'll look at some variations to the approaches we've discussed already in this chapter.

WE'LL COVER THE FOLLOWING

Asset server

The example uses a Node project for the shared assets. An alternative option is an asset server that stores the assets. Since the assets are static files loaded via HTTP, an asset server can just be a simple web server.

Over time, the assets will change. For the asset project from the example, a new version of the asset project would have to be created. The new version must be integrated in every microservice. This sounds like unnecessary work, but this way each application can be tested with a new version of the assets.

Therefore, even with an asset server, a new version of the assets should not simply be put into production, but the applications should be adjusted to the new version and also tested with the assets. The version of the assets can be included in the URL path. Thus, version 3.3.7 of Bootstrap might be found under `/css/bootstrap-3.3.7-dist/css/bootstrap.min.css`. A new version would be available under a different path – for example, `/css/bootstrap-4.0.0-dist/css/bootstrap.min.css`.

Simpler JavaScript code

The JavaScript code in the example is quite flexible and can also deal with the failure of a service. A primitive alternative is shown in the SCS jQuery Project.

In essence, it uses the following JavaScript code:

```
$(document).ready(function() {
  $("a.embeddable").each(function(i, link) {
    $("<div />").load(link.href, function(data, status, xhr) {
      $(link).replaceWith(this);
    });
  });
});
```

The code uses jQuery to search for hyperlinks (`<a ...>`) with the CSS class `embeddable`, and then replaces the link with the content that link refers to.

This demonstrates how simple it is to implement a client integration with JavaScript. At minimum, it is just seven lines of jQuery code.

Of course, it is also possible that each microservice has its own code for transclusion. This seems like a redundant implementation at first, however, it is not uncommon for each microservice to have its own code to parse, e.g., a JSON data structure. So custom code per microservice for transcluding HTML can also be an option.

Integration using other frontend integrations

Of course, client-side integration and links with server-side integration (see chapter 5) can be combined. Both approaches have different benefits:

- Web pages with server-side integration originate entirely from the server. Thus, the integration makes sense when the web page can be correctly displayed only if all the content is integrated.
- With client-side integration, transclusion is not executed when the other server is not available. The web page is still displayed, just without transclusion. This can be the better option because it improves resilience. However, the webpage would need to be usable without the transcluded content.

However, a server-side integration requires additional infrastructure. For client-side integration, this is not necessary. Therefore, it makes sense to start with client-side integration and to supplement server-side integration when necessary.

Other integrations

Synchronous communication (chapter 9) or asynchronous communication (chapter 6) enable the communication of the backend system and therefore can be combined, of course, with client-side frontend integration.

QUIZ

1 In what case does server-side integration make sense?

COMPLETED 0%

1 of 2

In the next lesson, we'll look at some experiments you can conduct on links and client-side integration.