The Microfrontend Revolution: Module Federation with Angular

With Module Federation we can use Angular and its router to lazy load separately compiled and deployed microfrontends.

26.04.2020 | share

Dies ist Beitrag 2 von 8 der Serie *“Module Federation”*

3. Dynamic Module Federation with Angular
4. Building A Plugin-based Workflow Designer With Angular and Module Federation
5. Getting Out of Version-Mismatch-Hell with Module Federation
6. Using Module Federation with (Nx) Monorepos and Angular
7. Multi-Framework and -Version Micro Frontends with Module Federation: The Good, the Bad, the Ugly
8. Pitfalls with Module Federation and Angular

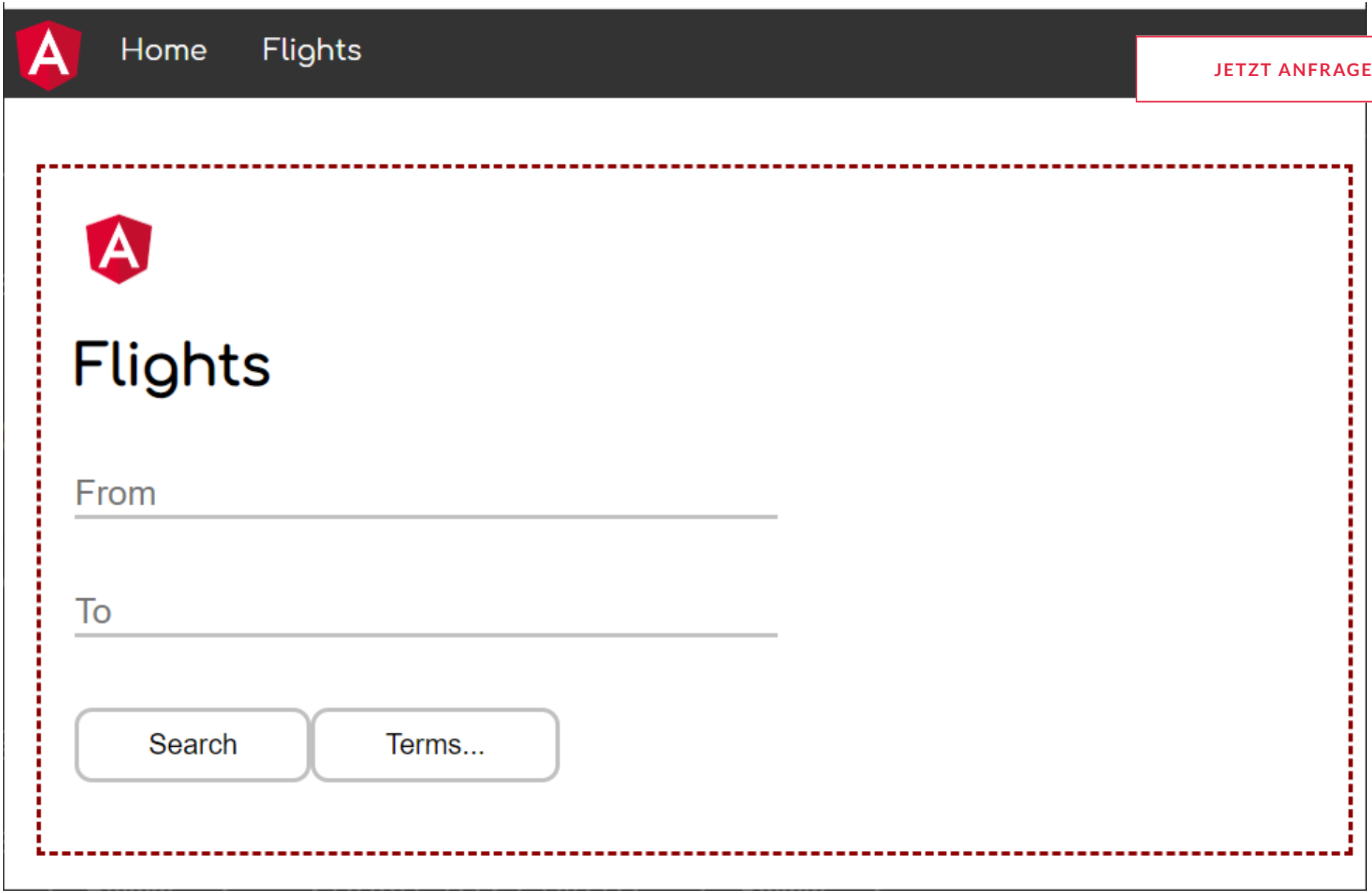
> 2020-10-13: Updated to use webpack 5 and Angular CLI 11.0.0-next.6.
> 2021-08-08: Updated for Angular CLI 12.x
> 2021-12-23: Updated for Angular CLI 13.1.x and above

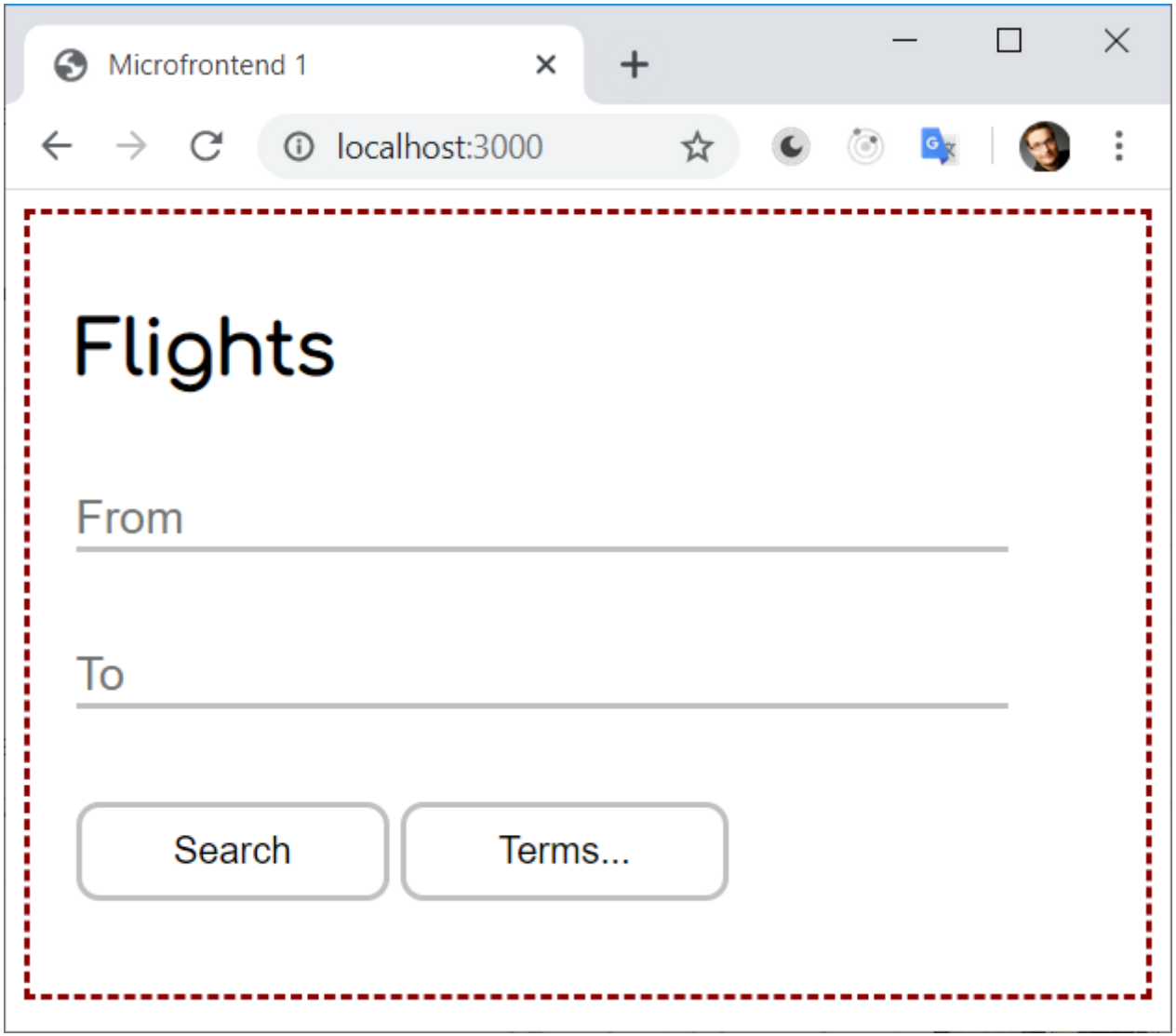
> Big thanks to Zack Jackson, the mastermind behind Module Federation, who helped me bypassing some pitfalls.

Important: This article is written for Angular and **Angular CLI 13.1** and higher. Make sure you have a fitting version if you try out the examples outlined here! For more details on the differences/ migration to Angular

version 5 to implement microfrontends. This article brings Angular into play and shows how to create an Angular-based microfrontend shell using the router to lazy load a separately compiled, and d[illegible] microfrontend.

Besides using Angular, the result looks similar as in the previous article:


Home
Flights
JETZT ANFRAGEN!
Flights
From
To
Search
Terms...



Activating Module Federation for Angular Projects

The case study presented here assumes that both, the shell and the microfrontend are projects in the same Angular workspace. For getting started, we need to tell the CLI to use module federation when building them. However, as the CLI shields webpack from us, we need a custom builder.

The package @angular-architects/module-federation provides such a custom builder. To get started, you can just "ng add" it to your projects:

```
1 ng add @angular-architects/module-federation --project shell --port 5000
2 ng add @angular-architects/module-federation --project mfe1 --port 3000
```

While it's obvious that the project `shell` contains the code for the `shell`, `mfe1` stands for *Micro Frontend 1*.

The command shown does several things:

- Generating the skeleton of an `webpack.config.js` for using module federation

Please note that the `webpack.config.js` is only a **partial** webpack configuration. It only cont[illegible] control module federation. The rest is generated by the CLI as usual.

The Shell (aka Host)

Let's start with the shell which would also be called the host in module federation. It uses the router to lazy load a `FlightModule`:

```
1  export const APP_ROUTES: Routes = [
2    {
3      path: '',
4      component: HomeComponent,
5      pathMatch: 'full'
6    },
7    {
8      path: 'flights',
9      loadChildren: () => import('mfe1/Module').then(m => m.FlightsModule)
10   },
11 ];
```

To ease the TypeScript compiler, we need a typing for it:

```
// decl.d.ts
declare module 'mfe1/Module';
```

Also, we need to tell webpack that all paths starting with `mfe1` are pointing to an other project. This can be done by using the `ModuleFederationPlugin` in the generated `webpack.config.js`:

```
const ModuleFederationPlugin = require("webpack/lib/container/ModuleFederationPlugin");
const mf = require("@angular-architects/module-federation/webpack");
const path = require("path");
const share = mf.share;

[...]

module.exports = {
 output: {
  uniqueName: "shell",
  publicPath: "auto"
```

```
},
resolve: {
 alias: {
  ...sharedMappings.getAliases(),
 }
},
experiments: {
  outputModule: true
},
plugins: [
 new ModuleFederationPlugin({
  library: { type: "module" },
  remotes: {
    "mfe1": "mfe1@http://localhost:3000/remoteEntry.js",
  },

  shared: share({
   "@angular/core": { singleton: true, strictVersion: true, requiredVersion: 'auto' },
   "@angular/common": { singleton: true, strictVersion: true, requiredVersion: 'auto' },
   "@angular/router": { singleton: true, strictVersion: true, requiredVersion: 'auto' },
   "@angular/common/http": { singleton: true, strictVersion: true, requiredVersion: 'auto' },

   ...sharedMappings.getDescriptors()
```

```
41 |   sharedMappings.getPlugin(),
42 |  ],
43 | };
```


The `experiments` section is temporarily needed to activate a bug fix in webpack. The `remotes` section maps the internal name `mfe1` to the same one defined within the separately compiled microfrontend. It also points to the path where the remote can be found — or to be more precise: to its remote entry. This is a tiny file generated by webpack when building the remote. Webpack loads it at runtime to get all the information needed for interacting with the microfrontend.

While specifying the remote entry's URL that way is convenient for development, we need a more dynamic approach for production. Fortunately, there are several options for doing this. One option is presented in a below sections.

The property `shared` contains the names of libraries our shell shares with the microfrontend(s). The combination of `singleton: true` and `strictVersion: true` makes webpack emit a runtime error when the shell and the micro frontend(s) need different incompetible versions (e. g. two different major versions). If we skipped `strictVersion` or set it to `false`, webpack would only emit a warning at runtime. More information about dealing with version mismatches can found in a further article of this series.

> The helper function share used in this generated configuration replaces the value `'auto'` version found in your `package.json`.

In addition to the settings for the `ModuleFederationPlugin`, we also need to place some options in the `output` section.

The `uniqueName` is used to represent the host or remote in the generated bundles. By default, webpack uses the name from `package.json` for this. In order to avoid name conflicts when using monorepos with several applications, it is recommended to set the `uniqueName` manually.

The Microfrontend (aka Remote)

The microfrontend — also referred to as a *remote* with terms of module federation — looks like an ordinary Angular application. It has routes defined within in the `AppModule`:

```
1 export const APP_ROUTES: Routes = [
2   { path: '', component: HomeComponent, pathMatch: 'full'}
```

Also, there is a FlightsModule:


```
@NgModule({
 imports: [
  CommonModule,
  RouterModule.forChild(FLIGHTS_ROUTES)
 ],
 declarations: [
  FlightsSearchComponent
 ]
})
export class FlightsModule { }
```

This module has some routes of its own:

```
export const FLIGHTS_ROUTES: Routes = [
  {
    path: 'flights-search',
    component: FlightsSearchComponent
  }
];
```

```
const ModuleFederationPlugin = require("webpack/lib/container/ModuleFederationPlugin");
const mf = require("@angular-architects/module-federation/webpack");
const path = require("path");

const share = mf.share;

[...]

module.exports = {
 output: {
  uniqueName: "mfe1",
  publicPath: "auto"
 },
 optimization: {
  runtimeChunk: false
 },
 resolve: {
  alias: {
   ...sharedMappings.getAliases(),
  }
 },
 experiments: {
```

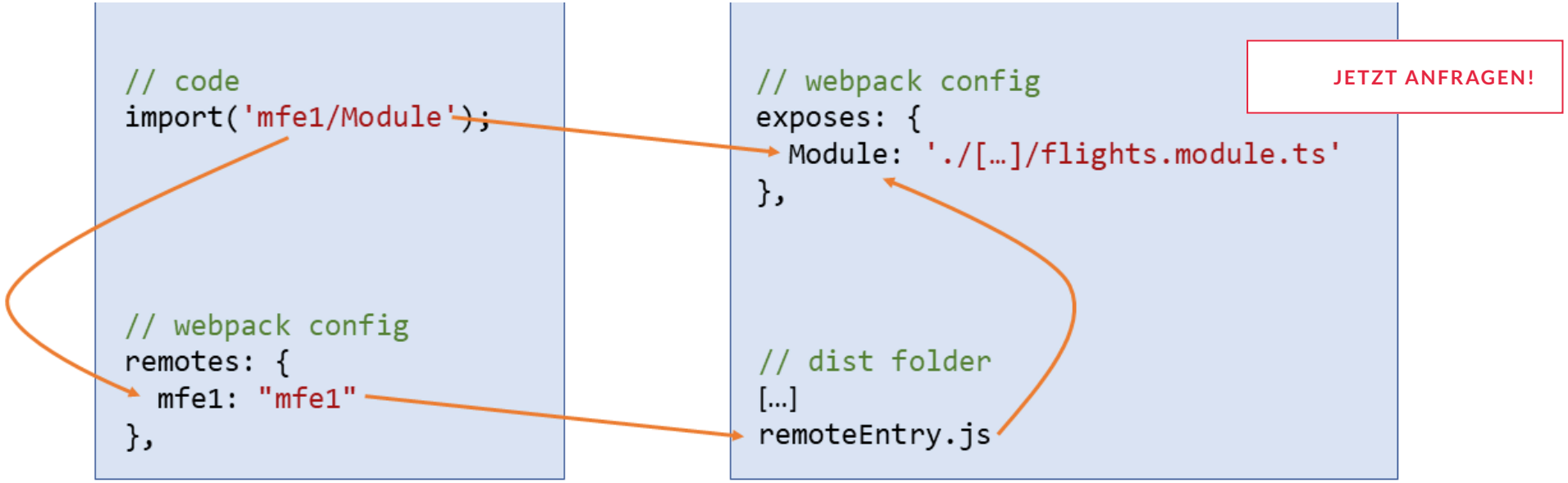
```
26    new ModuleFederationPlugin({
27      library: { type: "module" },
28
29      name: "mfe1",
30      filename: "remoteEntry.js",
31      exposes: {
32        './Module': './projects/mfe1/src/app/flights/flights.module.ts',
33      },
34      shared: share({
35        "@angular/core": { singleton: true, strictVersion: true, requiredVersion: 'auto' },
36        "@angular/common": { singleton: true, strictVersion: true, requiredVersion: 'auto' },
37        "@angular/router": { singleton: true, strictVersion: true, requiredVersion: 'auto' },
38        "@angular/common/http": { singleton: true, strictVersion: true, requiredVersion: 'auto' },
39
40        ...sharedMappings.getDescriptors()
41      })
42
43    }),
44    sharedMappings.getPlugin(),
45  ],
46 };
```

Trying it out

To try everything out, we just need to start the shell and the microfrontend:

```
1 ng serve shell -o
2 ng serve mfe1 -o
```

Then, when clicking on `Flights` in the shell, the micro frontend is loaded:



> **Hint:** To start several projects with one command, you can use the npm package concurrently.

A little further detail

Ok, that worked quite well. But have you had a look into your `main.ts`?

```
1 import('./bootstrap')
2   .catch(err => console.error(err));
```


The code you normally find in the file `main.ts` was moved to the `bootstrap.ts` file loaded here. All of this was done by the `@angular-architects/module-federation` plugin.

While this doen't seem to make a lot of sense at first sight, it's a typical pattern you find in Module Federation-based applications. The reason is that Module Federation needs to decide which version of a shared library to load. If the shell, for instance, is using version 12.0 and one of the micro frontends is already built with version 12.1, it will decide to load the latter one.

To look up the needed meta data for this decision, Module Fedaration squeezes itself into dynamic imports like this one here. Other than the more tradtional static imports, dynamic imports are asynchronous. Hence, Module Federation can decide on the versions to use and actually load them.

More details on this can be found in another article of this series.

Conclusion and Evaluation

be shared and strategies for dealing with incompatible versions can be configured.

It is also interesting that the microfrontends are loaded by Webpack under the hood. There in the source code of the host or the remote. This simplifies the use of module federation and the resulting source code, which does not require additional microfrontend frameworks.

However, this approach also puts more responsibility on the developers. For example, you have to ensure that the components that are only loaded at runtime and that were not yet known when compiling also interact as desired.

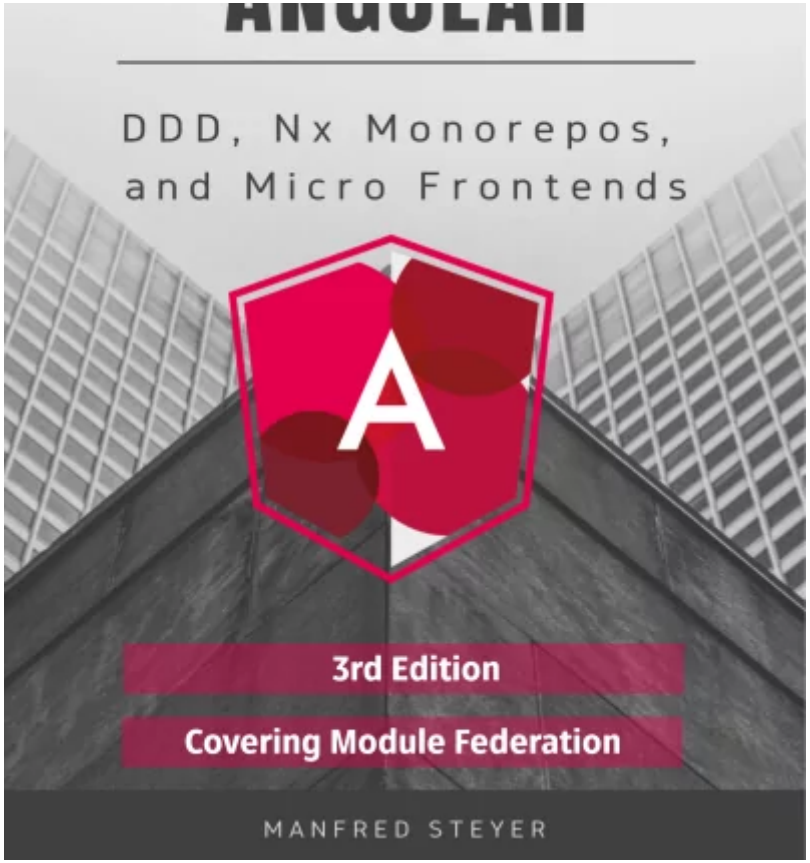
One also has to deal with possible version conflicts. For example, it is likely that components that were compiled with completely different Angular versions will not work together at runtime. Such cases must be avoided with conventions or at least recognized as early as possible with integration tests.

What's next? More on Architecture!

So far, we've seen that Module Federation is a strightforward solution for creating Micro Frontends on top of Angular. However, when dealing with it, several additional questions come in mind:

- Which proven patterns should we use?
- How can we avoid pitfalls when working with Module Federation?
- Which advanced scenarios are possible?

Our free eBook (about 100 pages) covers all these questions and more:



Feel free to download it here now!

Feel free to share it on social media
and subscribe to our newsletter

Don't Miss Anything!

Subscribe to our newsletter to get all the information about Angular.

Business EMail Address*:

Country*

Subscribe*

Unsere Angular-Schulungen

Hot!

Angular Schulung: Strukturierte Einführung

In dieser strukturierten Einführung lernen Einsteiger und Autodidakten alle Building-Blocks…

MEHR INFORMATIONEN

Hot!

Angular Architektur Workshop

In diesem weiterführenden Intensiv-Kurs lernen Sie, wie sich große und…

MEHR INFORMATIONEN

Micro Frontends mit

Lernen Sie große Angular-Lös
cro Frontends zu strukt

MEHR INFORMATIO

Aktuelle Blog-Artikel

ALLE ARTIKEL

VON: MANFRED STEYER, GDE

4 WAYS TO PREPARE FOR ANGULAR'S UPCOMING STANDALONE COMPONENTS

With the introduction of Standalone Components, NgModules will become optional. But how to prepare for…

MEHR ERFAHREN

VON: MANFRED STEYER, GD

STATE-MANAGEMENT MIT

In diesem Tutorial lernen Sie alles über NGRX in Ang
lektoren und Effects…

MEHR ERFAHREN

Nur einen Schritt entfernt!

JETZT ANFRAGEN!

Manfred Steyer

ist Trainer und Berater mit Fokus auf Angular. Er hat berufsbegleitend IT und IT-Marketing in Graz sowie ebenfalls berufsbegleitend Computer Science in Hagen studiert und eine vier-semestrige Ausbildung im Bereich der Erwachsenenbildung abgeschlossen.

JETZT ANFRAGEN

Manfred Steyer

Ludersdorf 219
8200 Gleisdorf
Österreich

manfred.steyer@softwarearchitekt.at

NEWSLETTER ABONNIEREN

Nichts mehr verpassen!

Hiermit erkläre ich mich damit einverstanden, dass der Betreiber dieser Seite meine E-Mail-Adresse zum Zwecke des Versands des Newsletters verarbeiten kann.
DATENSCHUTZ.

Email:

ANGULAR ARCHITECTS INSIDE KNOWLEDGE SOFTWARE ARCHITECT

JETZT ANFRAGEN!