[1. Project Setup 3](#_Toc156226078)

[1.1 Create applications 3](#_Toc156226079)

[1.2 Install bootstrap all apps 3](#_Toc156226080)

[2 Add Module Federation 3](#_Toc156226081)

[2.1 Add Module Federation 3](#_Toc156226082)

[2.2 Add module and pages for demo 4](#_Toc156226083)

[2.2.1 Add layout for host-app 4](#_Toc156226084)

[2.2.2 Add functions in products-app 4](#_Toc156226085)

[2.2.3 Add functions in carts-app 4](#_Toc156226086)

[3. Load micro-frontend in host-app 4](#_Toc156226087)

[3.1 Configure module federation 4](#_Toc156226088)

[3.1.1 Config webpack for Shell app “host-app” 4](#_Toc156226089)

[3.1.2 Config webpack for products-app 6](#_Toc156226090)

[3.1.3 Config webpack for carts-app 6](#_Toc156226091)

[3.1.4 Display products and carts app from host-app 6](#_Toc156226092)

[4. Create Shared Library 7](#_Toc156226093)

[4.1 Initialize the shared library 7](#_Toc156226094)

[4.2 Create LanguageService 7](#_Toc156226095)

[4.3 Use the Shared Library Locally 8](#_Toc156226096)

[4.3.1 Use LanguageService in all 3 applications 8](#_Toc156226097)

[5 Implement i18n 9](#_Toc156226098)

[5.1 Implement i18n for host-app 9](#_Toc156226099)

[5.1.1 Add ngx-translate library 9](#_Toc156226100)

[5.1.2 Implement i18n 9](#_Toc156226101)

[5.2 Implement i18n for products-app and carts-app 12](#_Toc156226102)

[5.2.1 Add ngx-translate library 12](#_Toc156226103)

[5.2.2 Implement i18n 12](#_Toc156226104)

[5.3 Cache Language 14](#_Toc156226105)

[Appendix - Highlights 15](#_Toc156226106)

[1. 404 Error – Unknown Configuration Issue in webpack.config.js 15](#_Toc156226107)

[2. i18n Support Cross Applications 17](#_Toc156226108)

> cd C:\Dev\works\poc\microfront\module\_fed\mfe-module-fed-2

## 1. Project Setup

### 1.1 Create applications

> cd mfe-module-fed-2

> ng new host-app

> ng new products-app

> ng new carts-app

### 1.2 Install bootstrap all apps

> npm install bootstrap

Update angular.json for both apps

"styles": [

  "src/styles.css",

    "node\_modules/bootstrap/dist/bootstrap.min.css"

],

scripts": [

    "./node\_modules/bootstrap/dist/js/bootstrap.bundle.min.js"

],

Go to <https://bootswatch.com/cerulean/> to get styles

## 2 Add Module Federation

### 2.1 Add Module Federation

> cd host-app

> ng add @angular-architects/module-federation --port 4200

> cd products-app

> ng add @angular-architects/module-federation --port 4201

> cd carts-app

> ng add @angular-architects/module-federation --port 4202

**AND**

Downgrade ngx-build-plus to ^16.0.0

> npm install for both

### 2.2 Add module and pages for demo

#### 2.2.1 Add layout for host-app

> cd host-app

> ng g c layout --skip-tests

And update corresponding files

#### 2.2.2 Add functions in products-app

> cd products-app

> ng g m products --route products --module app.module

Complete products page

#### 2.2.3 Add functions in carts-app

> cd carts-app

> ng g m carts --route carts --module app.module

Complete carts page

## 3. Load micro-frontend in host-app

### 3.1 Configure module federation

#### 3.1.1 Config webpack for Shell app “host-app”

host-app/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;

const sharedMappings = new mf.SharedMappings();

sharedMappings.register(

  path.join(\_\_dirname, 'tsconfig.json'),

  [/\* mapped paths to share \*/]);

module.exports = {

  output: {

    uniqueName: "hostApp",

    publicPath: "auto",

    scriptType: 'text/javascript'

  },

  optimization: {

    runtimeChunk: false

  },

  resolve: {

    alias: {

      ...sharedMappings.getAliases(),

    }

  },

  experiments: {

    outputModule: true

  },

  plugins: [

    new ModuleFederationPlugin({

        library: { type: "module" },

        name: "hostApp",

        shared: share({

          "@angular/core": { singleton: true, strictVersion: true, requiredVersion: 'auto' },

          "@angular/common": { singleton: true, strictVersion: true, requiredVersion: 'auto' },

          "@angular/common/http": { singleton: true, strictVersion: true, requiredVersion: 'auto' },

          "@angular/router": { singleton: true, strictVersion: true, requiredVersion: 'auto' },

          ...sharedMappings.getDescriptors()

        })

    }),

    sharedMappings.getPlugin()

  ],

};

#### 3.1.2 Config webpack for products-app

products-app/webpack.config.js

const { withModuleFederationPlugin } = require('@angular-architects/module-federation/webpack');

module.exports = withModuleFederationPlugin({

  name: 'products',

  filename: "remoteEntry.js",

  exposes: {

    './ProductsModule': './src/app/views/products/products.module',

  },

  shared: {

    "@angular/core": { singleton: true, strictVersion: false, requiredVersion: "auto" },

    "@angular/common": { singleton: true, strictVersion: false, requiredVersion: "auto" },

    "@angular/common/http": { singleton: true, strictVersion: false, requiredVersion: "auto" },

    "@angular/router": { singleton: true, strictVersion: false, requiredVersion: "auto" }

  }

});

#### 3.1.3 Config webpack for carts-app

carts-app/webpack.config.js

Similar as products-app

#### 3.1.4 Display products and carts app from host-app

host-app/src/app/app-routing.module.ts

const routes: Routes = [

  {

    path: 'products',

    loadChildren: () =>

         loadRemoteModule({

            type: 'module',

            remoteEntry: 'http://localhost:4201/remoteEntry.js',

            exposedModule: './ProductsModule'

        })

        .then(m => m.ProductsModule)

  },

  {

    path: 'carts',

    loadChildren: () =>

         loadRemoteModule({

            type: 'module',

            remoteEntry: 'http://localhost:4202/remoteEntry.js',

            exposedModule: './CartsModule'

        })

        .then(m => m.CartsModule)

  }

];

host-app/src/app/app.component.html

  <a [routerLink]="['/products']" class="btn btn-primary">Products</a>

  <a [routerLink]="['/carts]" class="btn btn-primary">Carts</a>

## 4. Create Shared Library

### 4.1 Initialize the shared library

> cd mfe-module-fed-2

> ng new mfe-shared --create-application=false

> cd mfe-shared

> ng generate library mfe-shared-lib

### 4.2 Create LanguageService

> cd mfe-shared

> ng generate service services/language --skip-tests --project=mfe-shared-lib

@Injectable({

  providedIn: 'root'

})

export class LanguageService {

  private currentLang$ = new BehaviorSubject<string>('en');

  setLanguage(lang: string) {

    this.currentLang$.next(lang);

  }

  getLanguage() {

    return this.currentLang$.asObservable();

  }

}

public-api.ts:

export \* from './lib/services/language.service';

export \* from './lib/mfe-shared-lib.module';

> ng build mfe-shared-lib

> cd dist/mfe-shared-lib

> npm link

**Note:**

Remember that any time you make changes to the library, you need to rebuild it for those changes to take effect in any applications that use the library.

### 4.3 Use the Shared Library Locally

#### 4.3.1 Use LanguageService in all 3 applications

> cd host-app

> npm link mfe-shared-lib

Import LanguageService in app.module.ts

import { MfeSharedLibModule } from './../../../mfe-shared/projects/mfe-shared-lib/src/lib/mfe-shared-lib.module';

imports: [

MfeSharedLibModule

]

Do the same for products-app and carts-app as well

## 5 Implement i18n

### 5.1 Implement i18n for host-app

#### 5.1.1 Add ngx-translate library

Add ngx-translate for host-app, products-app and carts-app as below

> cd host-app

> npm install @ngx-translate/core @ngx-translate/http-loader --save

#### 5.1.2 Implement i18n

**1. webpack.config.js**

shared: share({

          … …

          "@ngx-translate/core": { singleton: true, strictVersion: true, requiredVersion: "15.0.0" },

          "@ngx-translate/http-loader": { singleton: true, strictVersion: true, requiredVersion: "8.0.0" }

        })

**2. Create custom translation loader**

import { TranslateLoader } from '@ngx-translate/core';

import { HttpClient } from '@angular/common/http';

import { forkJoin, Observable } from 'rxjs';

import { map } from 'rxjs/operators';

export class CustomTranslateLoader implements TranslateLoader {

  constructor(private http: HttpClient, private resources: string[]) {}

  public getTranslation(lang: string): Observable<any> {

    const requests = this.resources.map(resource =>

      this.http.get(`${resource}/${lang}.json`)

    );

    return forkJoin(requests).pipe(

      map(response => {

        return response.reduce((acc, current) => {

          return { ...acc, ...current };

        }, {});

      })

    );

  }

}

**3. Configure ngx-translate** in Angular Module (app.module.ts), and **load remote translations**:

export function HttpLoaderFactory(http: HttpClient) {

  return new CustomTranslateLoader(http, [

    'http://localhost:4200/assets/i18n',

    'http://localhost:4201/assets/i18n',

    'http://localhost:4202/assets/i18n'

  ]);

}

@NgModule({

  declarations: [ ],

  imports: [

… existing imported modules

    TranslateModule.forRoot({

      loader: {

        provide: TranslateLoader,

        useFactory: HttpLoaderFactory,

        deps: [HttpClient]

      }

    })

  ],

  exports: [ ]

})

export class AppModule { }

**4. Initialize Translation:**

host-app/src/app/app.component.ts

export class AppComponent implements OnInit{

  destroyRef = inject(DestroyRef);

  constructor(private translate: TranslateService,

    private languageService: LanguageService) {  }

  ngOnInit(): void {

    this.initLanguage();

  }

  initLanguage(): void {

    this.languageService.getLanguage().pipe(

      takeUntilDestroyed(this.destroyRef)

    ).subscribe(lang => {

      this.translate.setDefaultLang(lang);

      this.translate.use(lang)

    })

  }

}

**5. Create en.json and fr.json to store label keys.**

**6. Apply i18n in HTML template**

host-app/src/app/layout/layout.component.ts

  constructor(private languageService: LanguageService,

    private translateService: TranslateService) {}

  switchLanguage(language: string): void {

    this.languageService.setLanguage(language);

  }

  showLangSwitchFor(lang: string): boolean {

    return this.translateService.currentLang !== lang;

  }

host-app/src/app/layout/layout.component.html

      <ul class="navbar-nav ms-auto"> <!-- Note the use of ms-auto for right alignment -->

        <li class="nav-item">

          <a \*ngIf="showLangSwitchFor('en')" class="nav-link" href="#" (click)="switchLanguage('en')">

            {{ 'host.layout.nav.lang.en' | translate }}

          </a>

        </li>

        <li class="nav-item">

          <a \*ngIf="showLangSwitchFor('fr')" class="nav-link" href="#" (click)="switchLanguage('fr')">

            {{ 'host.layout.nav.lang.fr' | translate }}

          </a>

        </li>

      </ul>

### 5.2 Implement i18n for products-app and carts-app

#### 5.2.1 Add ngx-translate library

See 4.4.1

#### 5.2.2 Implement i18n

**1. webpack.config.js**

See 4.4.1 - 1

**2. Import ngx-translate module and configuration**:

Create shared module and import ngx-translate for sub modules ( **TranslateModule.forChild()** )

producst\src\app\shared\shared.module.ts

@NgModule({

  declarations: [],

  imports: [

    CommonModule,

    TranslateModule.forChild()

  ],

  exports: [

    TranslateModule

  ]

})

export class SharedModule { }

products\src\app\pp.module.ts: **TranslateModule.forRoot() – This is for running products-app standalone**

export function HttpLoaderFactory(http: HttpClient) {

  return new TranslateHttpLoader(http);

}

@NgModule({

  declarations: [ ],

  imports: [

… existing imported modules

    TranslateModule.forRoot({

      loader: {

        provide: TranslateLoader,

        useFactory: HttpLoaderFactory,

        deps: [HttpClient]

      }

    })

  ],

  exports: [ ]

})

export class AppModule { }

**3. Initialize Translation:**

products-app/src/app/app.component.ts:

See 4.4.4

**4. Create en.json and fr.json to store label keys.**

**5. Apply i18n in HTML template**

products-app/src/app/products/products.component.html

{{ 'products.list.btn.add.to.Cart' | translate }}

### 5.3 Cache Language

To keep the users’ language preference and avoid language setup lost due to page reload, cache the language in browser localStorage

@Injectable({

  providedIn: 'root'

})

export class LanguageService {

  DEFAULT\_LANGUAGE: string = 'en';

  LANGUAGE\_CACHE: string = 'mfe-language';

  private currentLang$: BehaviorSubject<string>;

  constructor() {

    const storedLang = localStorage.getItem(this.LANGUAGE\_CACHE);

  this.currentLang$ = new BehaviorSubject<string>(storedLang || this.DEFAULT\_LANGUAGE);

  }

  setLanguage(lang: string) {

    this.currentLang$.next(lang);

    localStorage.setItem(this.LANGUAGE\_CACHE, lang);

  }

  getLanguage() {

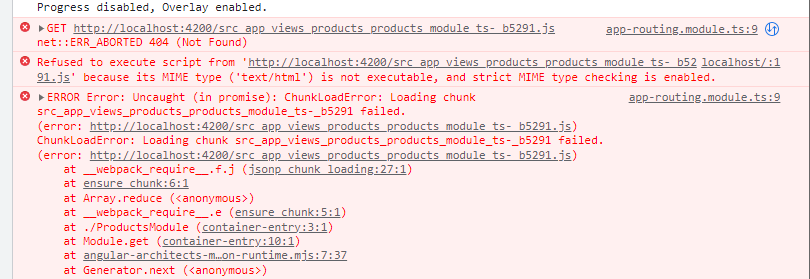
    return this.currentLang$.asObservable();

  }

}

## Appendix - Highlights

### 1. 404 Error – Unknown Configuration Issue in webpack.config.js



**Solution**: Change webpack.config.js

Original webpack.config.js

const ModuleFederationPlugin = require("webpack/lib/container/ModuleFederationPlugin");

module.exports = {

  output: {

    uniqueName: "products",

    publicPath: "auto",

  },

  optimization: {

    runtimeChunk: false,

  },

  experiments: {

    outputModule: true,

  },

  plugins: [

    new ModuleFederationPlugin({

      library: { type: "module" },

      name: "products",

      filename: "remoteEntry.js",

      exposes: {

        './ProductsModule': './src/app/views/products/products.module'

      },

      shared: {

        "@angular/core": { singleton: true, strictVersion: false, requiredVersion: "auto" },

        "@angular/common": { singleton: true, strictVersion: false, requiredVersion: "auto" },

        "@angular/common/http": { singleton: true, strictVersion: false, requiredVersion: "auto" },

        "@angular/router": { singleton: true, strictVersion: false, requiredVersion: "auto" }

      },

    }),

  ],

};

Updated webpack.config.js

const { withModuleFederationPlugin } = require('@angular-architects/module-federation/webpack');

module.exports = withModuleFederationPlugin({

  name: 'products',

  filename: "remoteEntry.js",

  exposes: {

    './ProductsModule': './src/app/views/products/products.module',

  },

  shared: {

    "@angular/core": { singleton: true, strictVersion: false, requiredVersion: "auto" },

    "@angular/common": { singleton: true, strictVersion: false, requiredVersion: "auto" },

    "@angular/common/http": { singleton: true, strictVersion: false, requiredVersion: "auto" },

    "@angular/router": { singleton: true, strictVersion: false, requiredVersion: "auto" },

    "@ngx-translate/core": { singleton: true, strictVersion: true, requiredVersion: "15.0.0" },

    "@ngx-translate/http-loader": { singleton: true, strictVersion: true, requiredVersion: "8.0.0" }

  }

});

### 2. i18n Support Cross Applications

Steps:

1. Apply i18n Support in all applications respectively

2. Share ngx-translate in webpack.config.js of all applications

  shared: {

    … …

    "@ngx-translate/core": { singleton: true, strictVersion: true, requiredVersion: "15.0.0" },

    "@ngx-translate/http-loader": { singleton: true, strictVersion: true, requiredVersion: "8.0.0" }

  }

**Note:**

**Must set strictVersion: true and with specific requiredVersion**

3. Load remote translations to the shell app using custom translate loader

3.1 Create custom translate loader

See section 5.1.2 – 2

3.2 Load remote translations

See section 5.1.2 – 3