**Raaghu MFE Documentation**

**Project Structure Overview**

Host Project:

The project is a monorepo of multiple angular micro front ends. The host micro front end is the shell app within which the other micro front ends will be loaded.

A screen shot of a computer

Description automatically generated with low confidence

The webpack.config.js file at the root of the host application has the configuration for remote MFE’s. Alternatively, the MFE’s can also be loaded dynamically using loadRemoteModule() api provided by @angular-architects/module-federation as we will see later on.

The host project is a simple angular project with module federation installed. Installing module federation brings out the webpack.config.js file which allows us to extend the underlying webpack configuration which is usually hidden.

Libraries:

The libraries are stored in the libs folder. Library is not a micro frontend on its own but can be utilized inside other micro frontends. This allows us to reuse common components and share functionality among micro frontends. Services can be registered inside common library and can be used through dependency injection.

All the libraries are registered in the root tsconfig.ts file for it to be shared in all the MFE’s.

Text

Description automatically generated

1. **rds-elements**

Rds elements is derived from raaghu-elements repo and is an open source library of ui elements. Here is the github repo for the project.

Github: <https://github.com/Wai-Technologies/raaghu-elements>

This library consists of all the rds elements (each one wrapped into an individual module) and exported via the public-api.ts file. All the components are exported here.

Text

Description automatically generated

rds-elements has storybook preinstalled within it. These individual elements can be inspected inside the storybook environment. The entire source code is available to the user and he is free to edit the elements according to his needs.

1. **shared**

The shared library consists of all the functionality which can be used in multiple MFE’s. All the common services and base classed should go in here and will be exported through its public-api.ts present at the root of the library.

The MfeBaseComponent is a base component which any component from an MFE can extend. It provides some functionality like setting and retrieving values from local storage and also handles inter-mfe communication. The inter component communication uses RxJs subjects internally.

Text

Description automatically generated Text

Description automatically generated

The ComponentLoaderComponent is used to render an angular component exposed via module federation. It takes care of fetching the exposed module and creating a dynamic component using the ComponentFactoryResolver. It can be used in the following way inside an MFE.

Text

Description automatically generated

The config acts as an input property to pass the configuration needed to dynamically load the component. The inputs and outputs can be passed directly via the configuration object as shown below.

Text

Description automatically generated

1. **state-management**

The state management library consists of a sample implementation for maintaining global state of a feature. It is implemented using @ngrx/store which is an industry standard for angular state management. All of the actions, reducers and selectors go inside this library.

Text

Description automatically generated

Please make sure to register the StoreModule using the forRoot() method only once in the host application. Registering it in more than one module creates multiple instances of the store which will make state management futile. Same goes for the EffectsModule and the StoreDevtoolsModule

Text

Description automatically generated

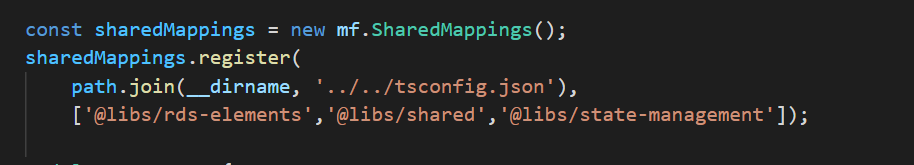
Build Scripts:

The framework comes with some built in scripts to make scaffolding new libraries and mfe’s easier.

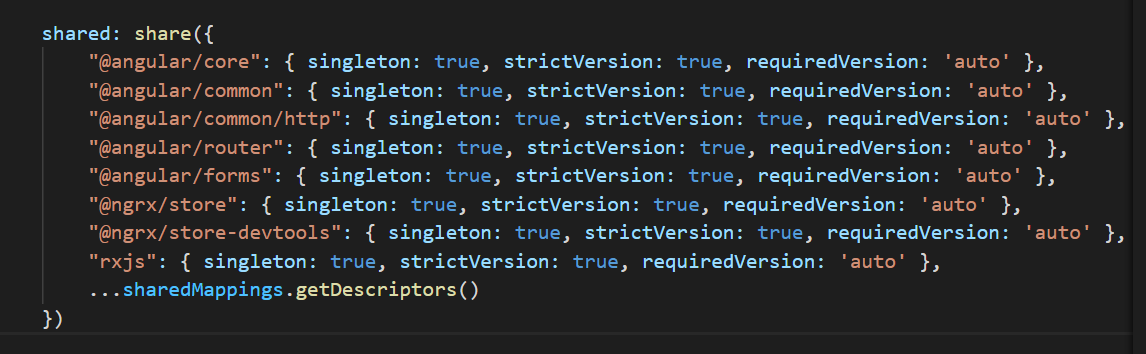
**npm run mfe <mfe-name> <mfe-port>**

This command will create a new MFE project inside the project's directory. It will also add the @angular-architects/module-federation package which will create the webpack.config.json file.

The script will also look through the angular.json and find out the library projects. Then it will provide those within the webpack.config.json file as shared libraries to be used inside the newly created project.



This command will also add the shared libraries inside the webpack.config.json



The command will also expose the app-module of the newly generated project by default as can be seen in the webpack file. It will also replace the forRoot() methid in the routing file with the forChild() method since this MFE will be used inside the host project.

Replacing the forRoot() with forChild() prevents creating multiple instances of RouterService



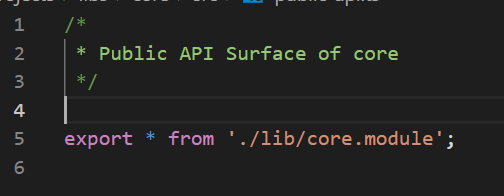
Eg. npm run mfe sample-mfe 3000

This command will create an mfe project with the name sample-mfe and will host it on port 3000 upon running the project.

**npm run lib <library-name>**

This command will create a shared library inside the libs folder and will also update the angular.json projects section with the library.

Since this is not an MFE project, there is no webpack.config.js file in the library. The public-api.ts file exposes the modules and functionalities exposed from the library project.

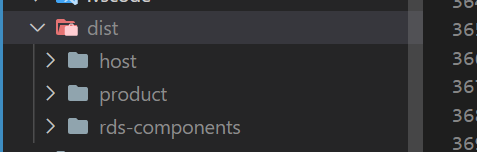


Eg. npm run lib sample-library

This command will create a library with the name sample-library

**npm run build-all**

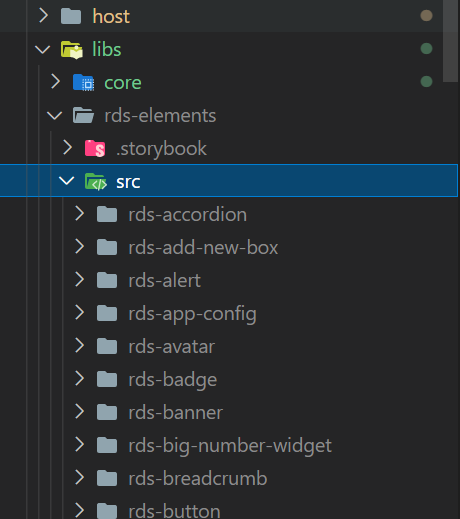
This command will build all the mfe’s as well as the host project and generate the build files inside the dist directory.



These builds can now be deployed on independent servers and all the mfe’s will get loaded into the host project

**npm run sync-elements**

This command add the raaghu-elements into the raaghu-mfe project within the rds-elements library. Note: This command will delete all the previous elements and rewrite them with the updated elements from the github repo



**npm run page <mfe-name> <page-name>**

This command will generate a page inside an mfe specified within the command. If the mfe does not exists, then it will throw an error in the console. The page created will be within the app folder inside of the mfe by default and will be registered into the app.module.ts within the same mfe (which is the root module).

**Project Architecture for ASPNetZero project**

The mfe architecture mirros the module structure of a standard asp net zero project.

App Module (Host project)

The root module forms the base of the angular application. It consists of a single <router-outlet> where all the other pages will get rendered.

All of the singleton services are registered into the root module like HttpClient and Authentication Services.

Account MFE

The Account MFE consists of all the pages related to user login and sign up. It contains the Login component, register component, forgot password component etc.

The user need not be authenticated to access these pages. An unauthenticated user will be automatically redirected to the login page.

Admin MFE

The Admin MFE consists of all the pages related to administration like managing users, roles and organization units. This module will only be accessible to users having the admin role. The admin will have permission to modify users, permissions, roles and also create roles, users and new permissions.

Dashboard MFE

The Dashboard MFE contains of the dashboard pages for all the different roles.

Service Proxy Common Lib

The service proxy is a shared library which consists of the api calls and which can be consumed in all the other MFE’s.