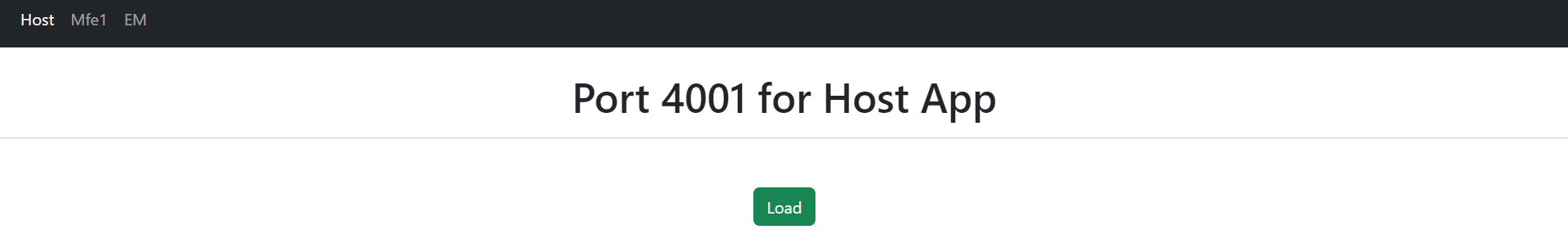
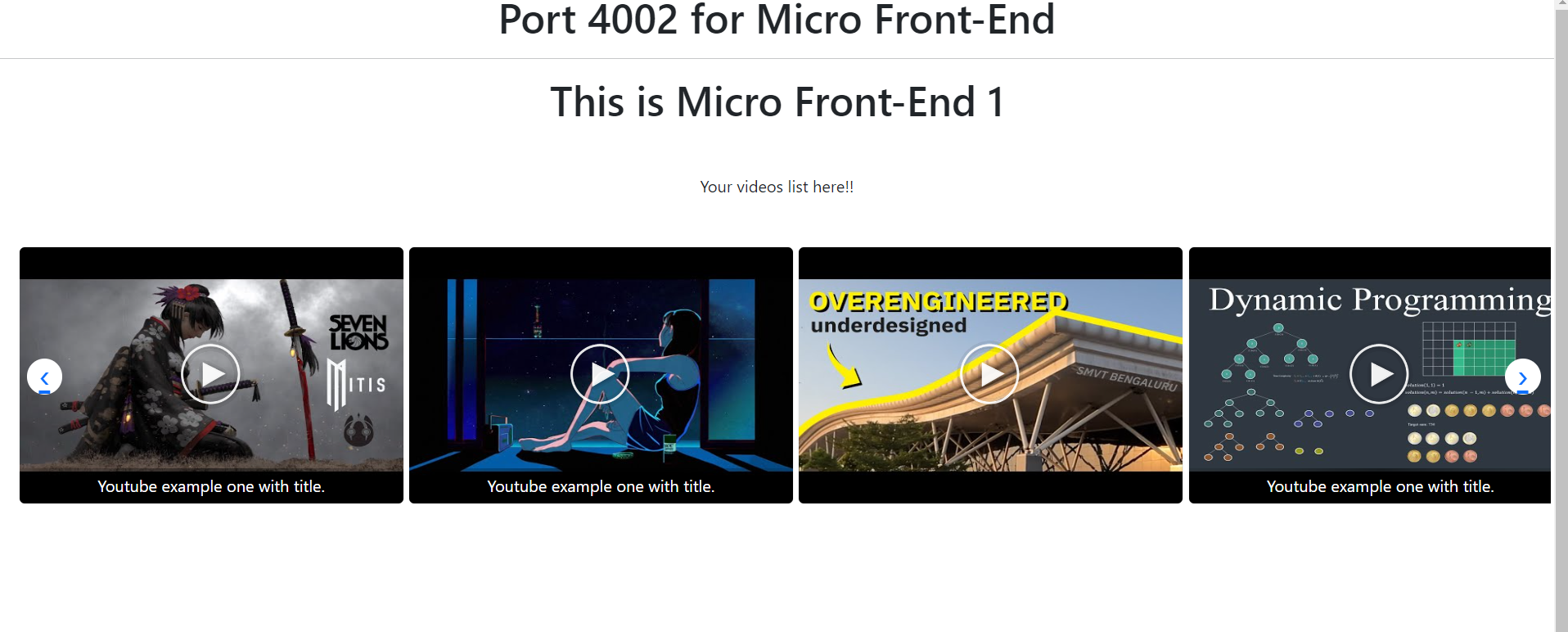
# Micro Frontends Using Module Federation

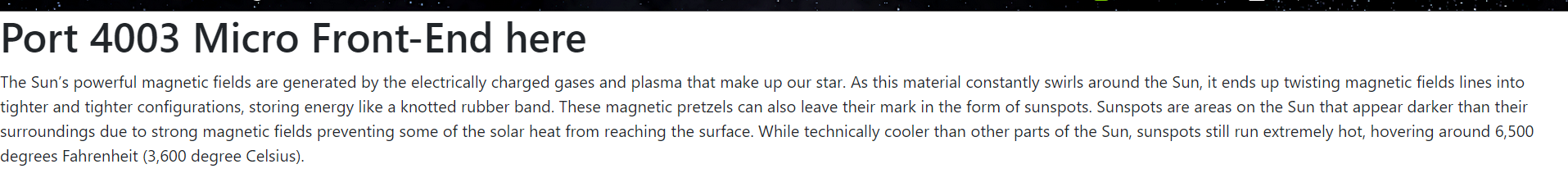
Host-APP



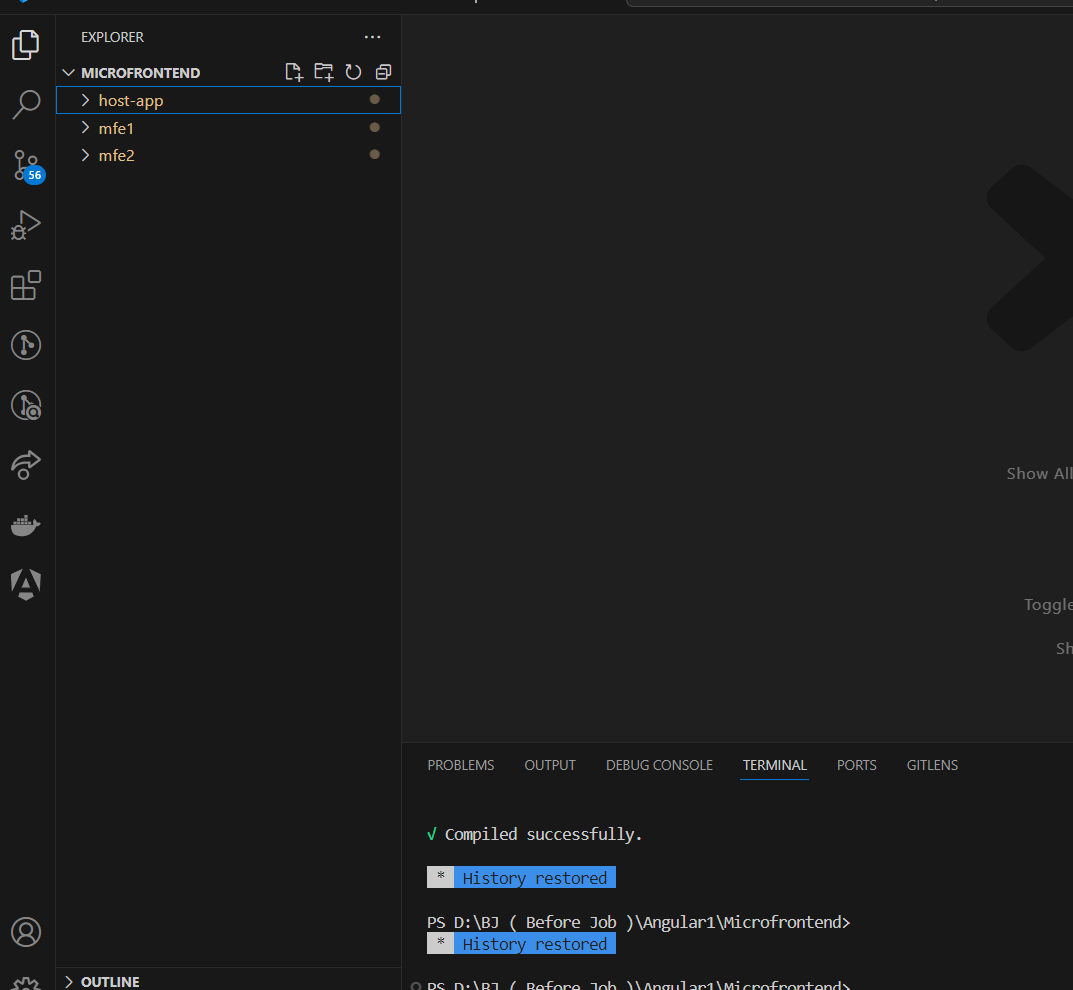
MFE1



MFE2



1. Create one folder which contain 3 apps in which 1 will be host all and the others will be remote app



ng new projectname

1. Now install angular-architects module federation in each one of them

ng g @angular-architects/module-federation@15 --project shell --port 4200 --type host

ng g @angular-architects/module-federation@15 --project mfe1 --port 4201 --type remote

This code makes two new file one of which is important for change known as webpack.config.js it contains details for host it is consuming and for remotes(or mfes) it is exposing.

One more change It just looks like this:

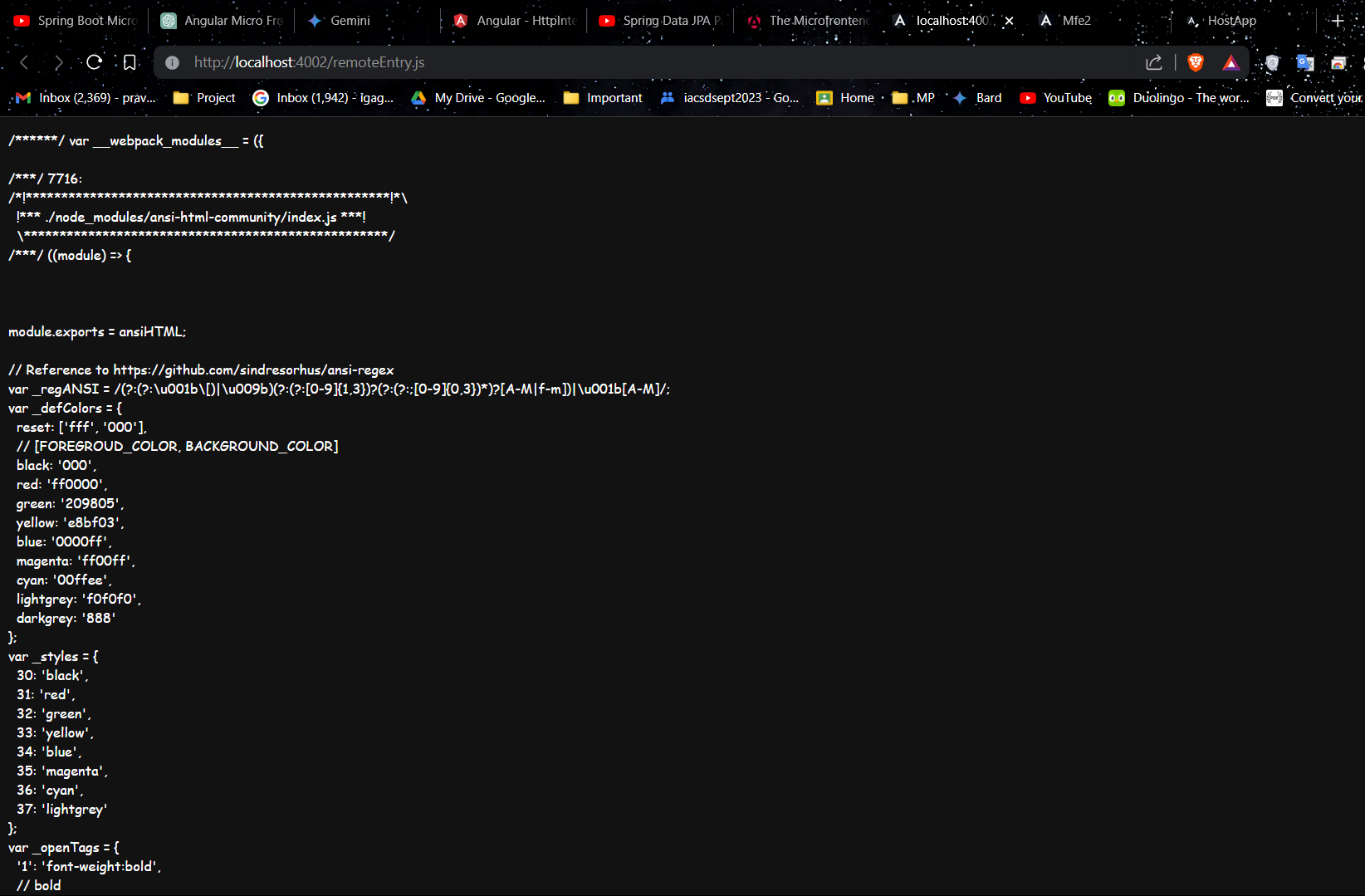
import('./bootstrap')

.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.

Remember there are 2 one is host where module which contains components to be hosted is to be shown at a url or can be rendered using button on main page and the other one is remote of microfrontend(mfe) which is on different port but can be hosted on the port of host app.

1. Expose the mfe module which has to be shown always know whatever you expose it gets exposed on the same post and on the relativeurl of (“your\_port\_url/remoteEntry.js”) eg http:localhost:4002/remoteEntry.js



1. Dependencies can also be shared like this (Not done Practically)

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

module.exports = withModuleFederationPlugin({

// Explicitly share packages:

shared: share({

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

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

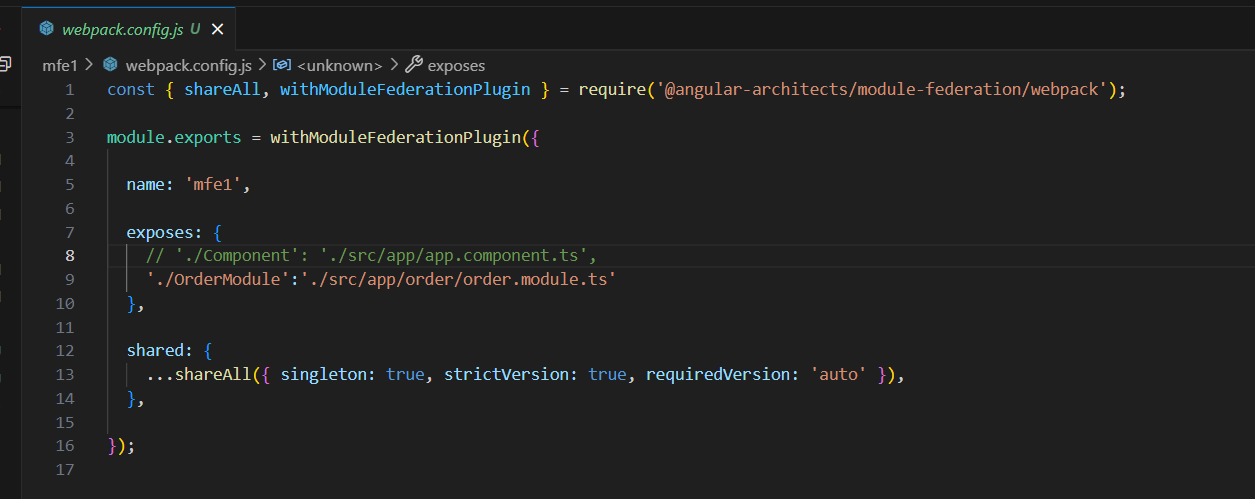
"@angular/common/http": { singleton: true, strictVersion: true, requiredVersion: 'auto' },

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

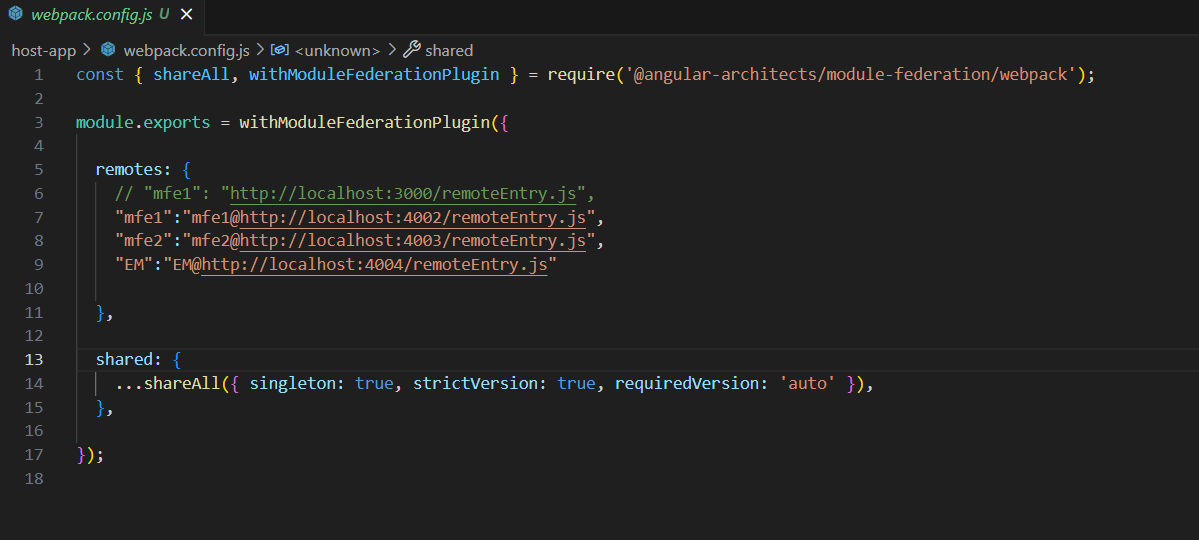
}),

});

1. The mfe1 module exposed at remoteEntry.js webpoint

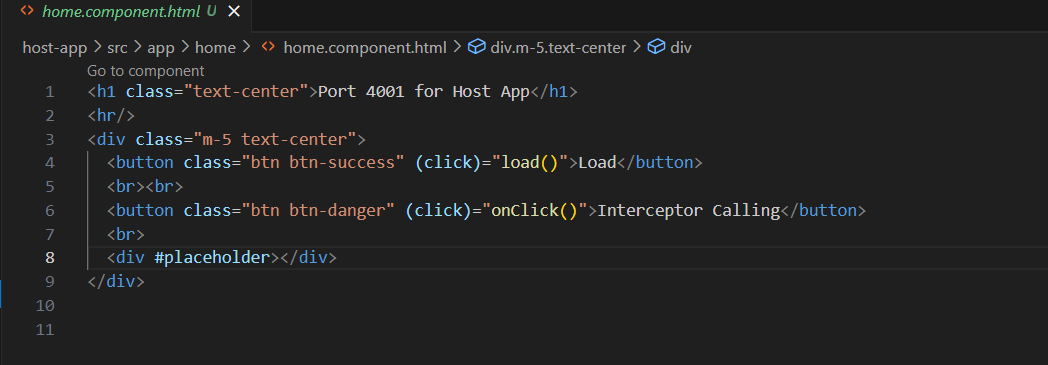


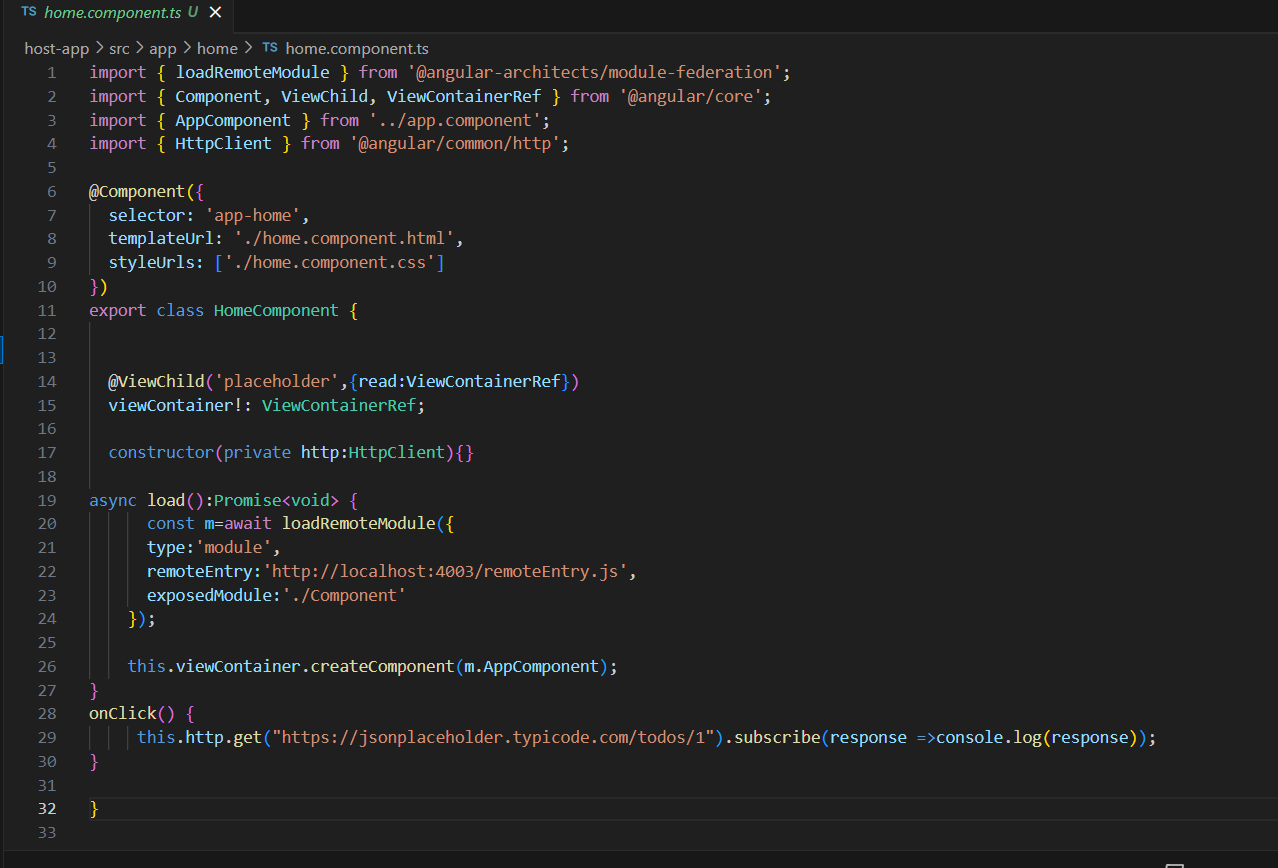
1. The host app consuming the modules which are exposed at the webpoints of other ports



1. At host App we can create those mfes to the web endpoints or thorugh the button(Button method)

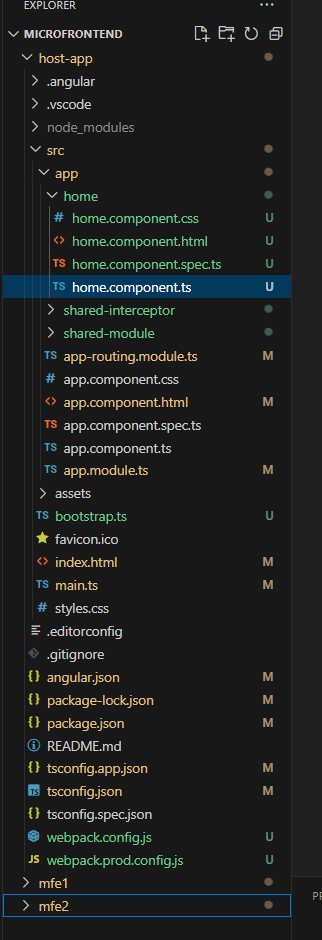
Here is the html page which contains load function onclick of button Load and #placeholder where the mfe will load





Here in load method which is of type Promise<void> and loads remote module on click of Load button

And View Container Ref is used to render that component to the host-app using @ViewChild

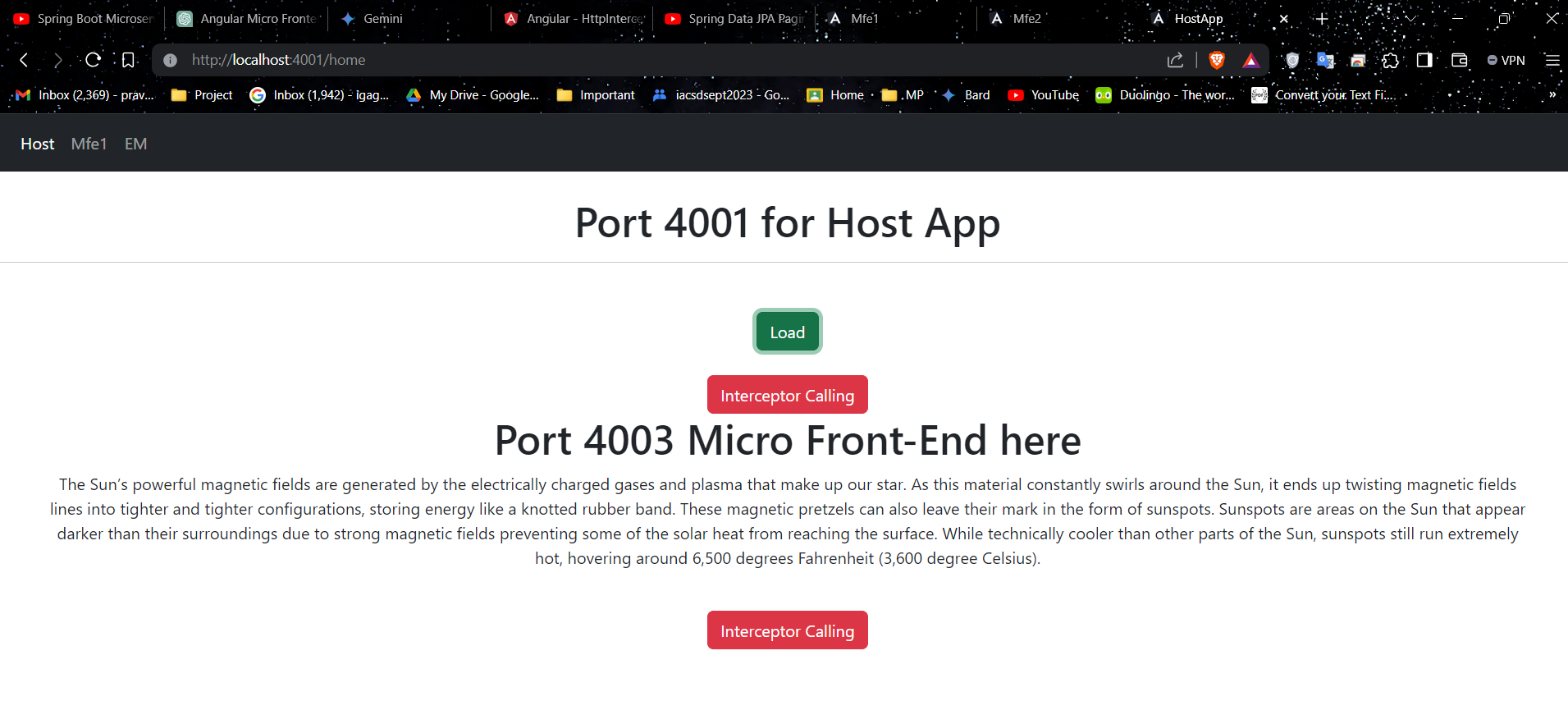


Here home folder contains home component which is then include in appmodule and used in app component using router-outlet



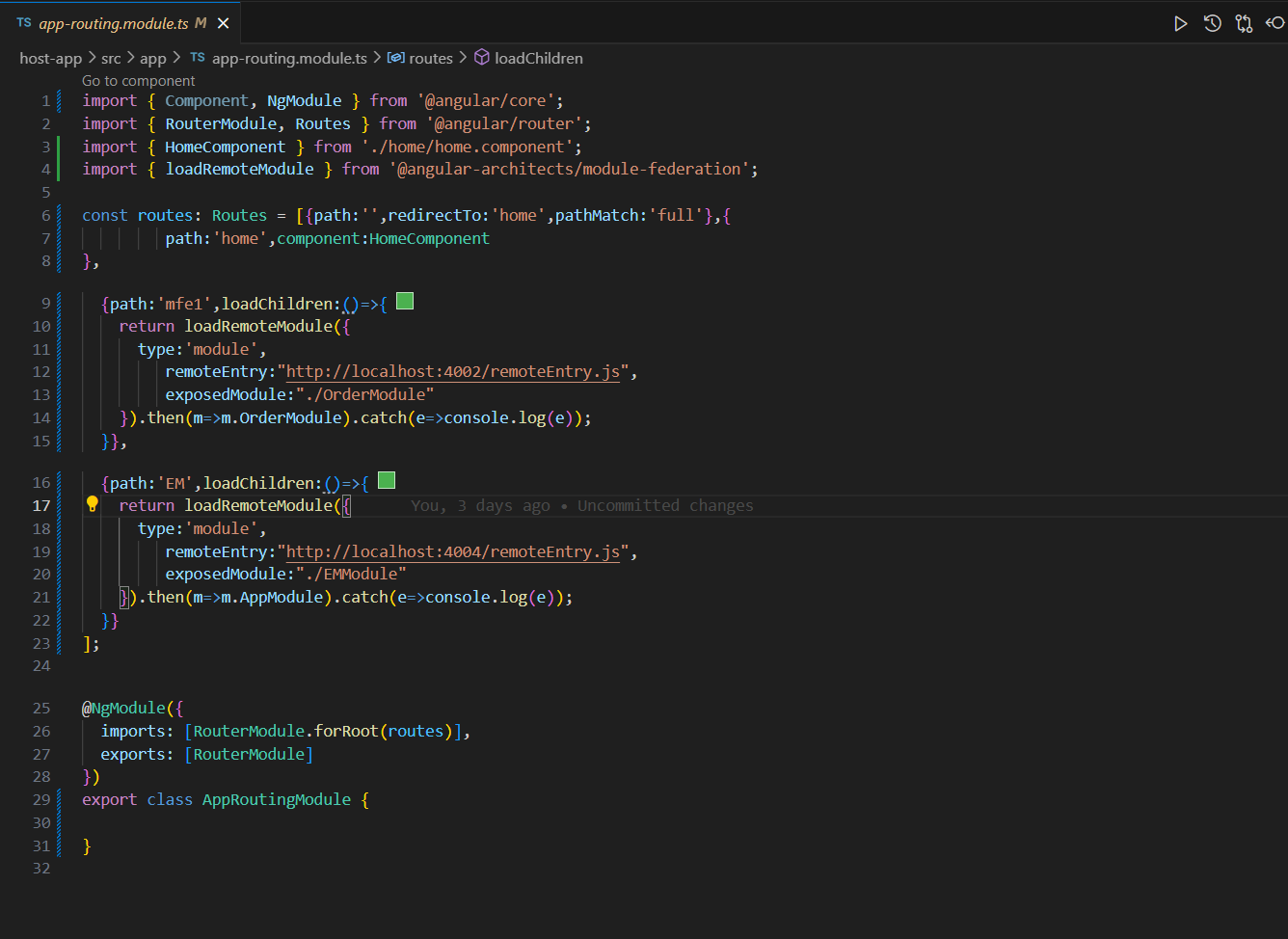
Include home component here

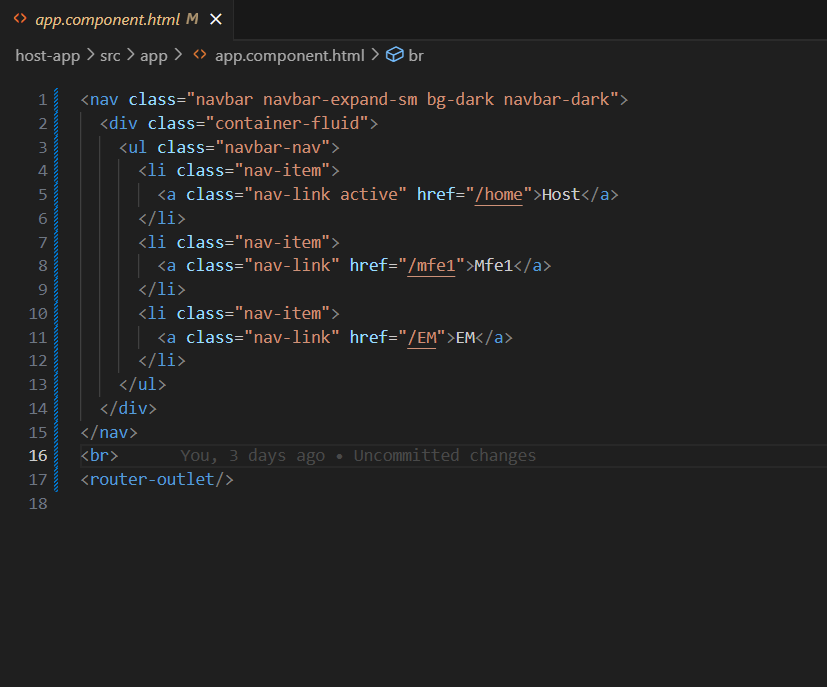
Here Component is loaded which the load button is called.



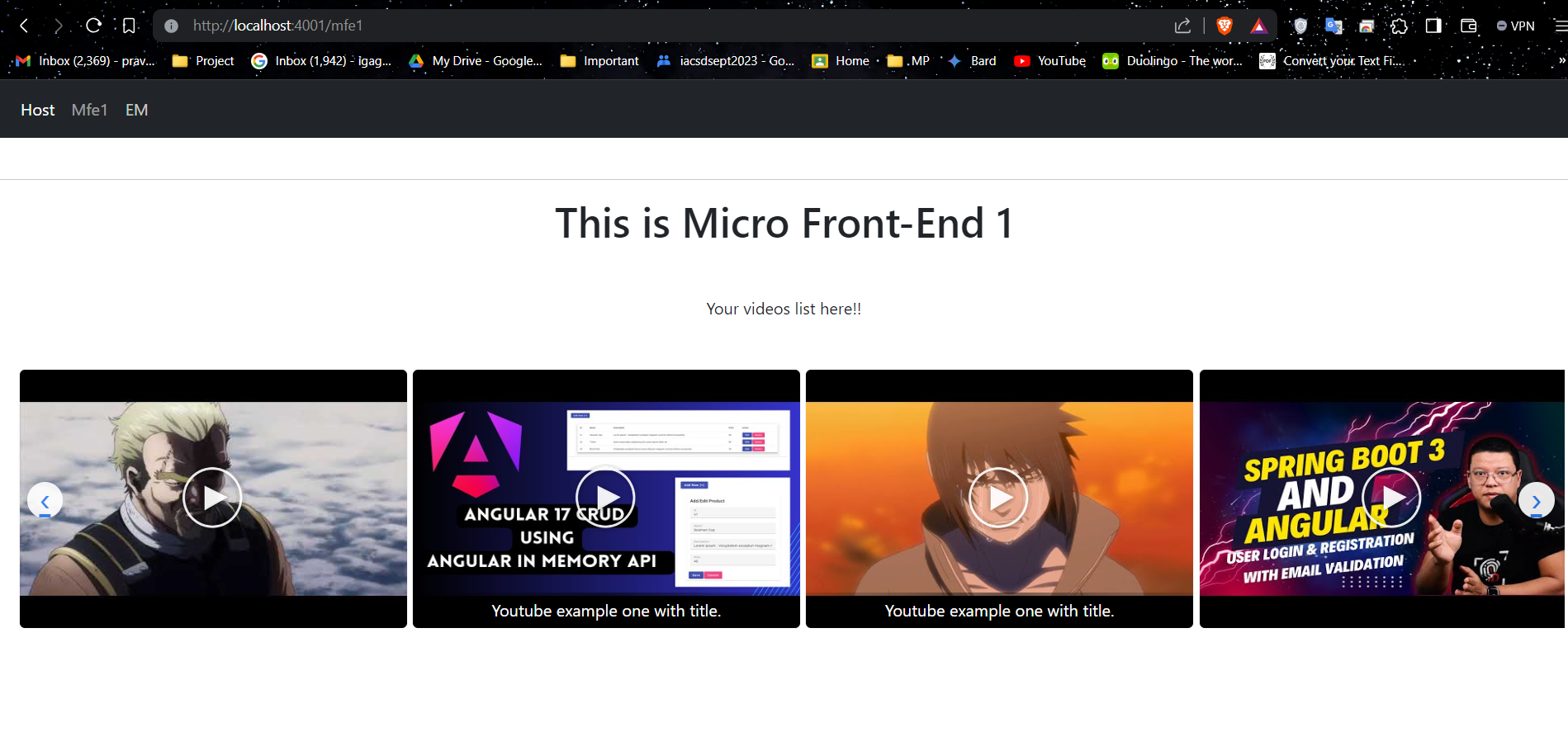
(At the Web EndPoint)

Here is the app-routing-module which renders one of the mfes on the relative path of the hostapp which is http://localhost:4001/mfe1





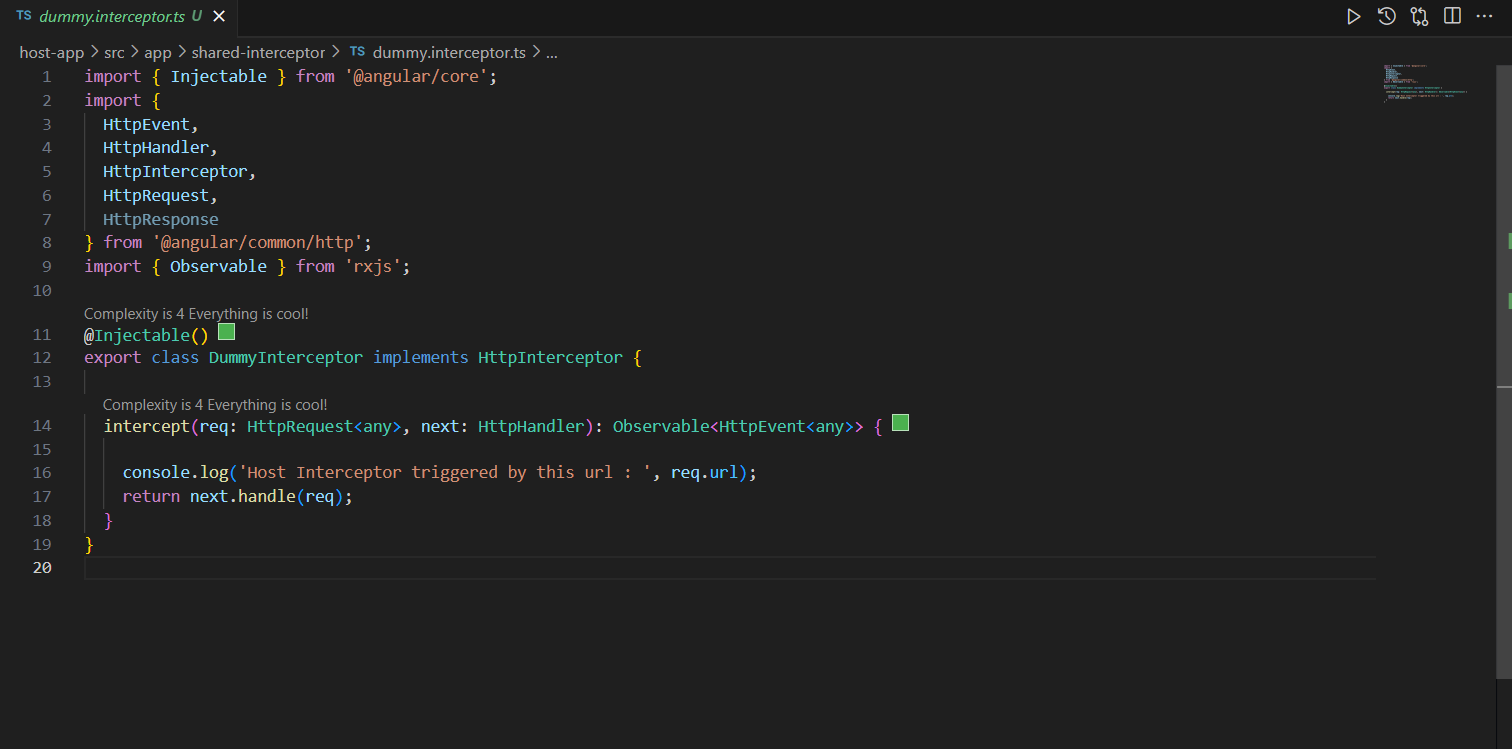
Here is the html where a anchor tag takes to the link of mfe1 .



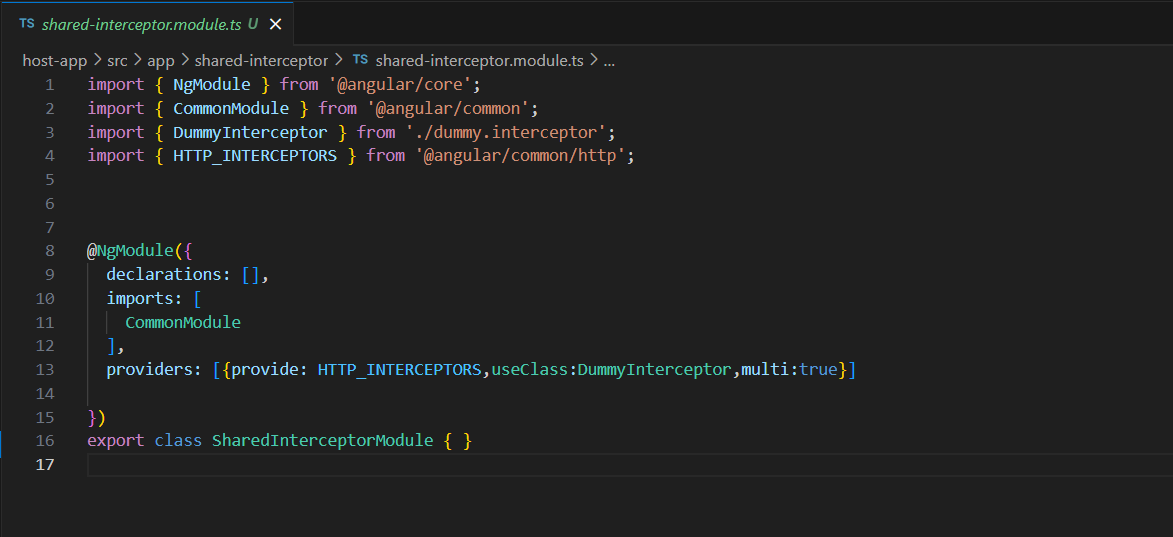
So this was your micro frontends in working.

# Interceptors in Micro Frontends

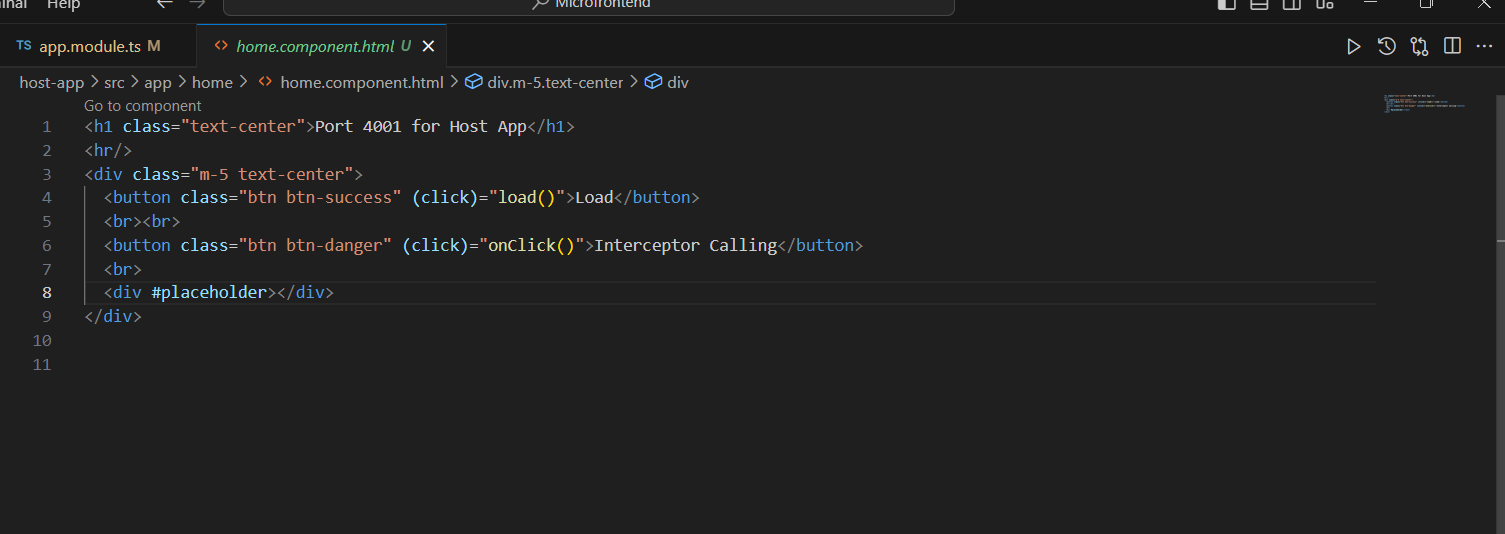
1. Create a interceptor using ng g interceptor

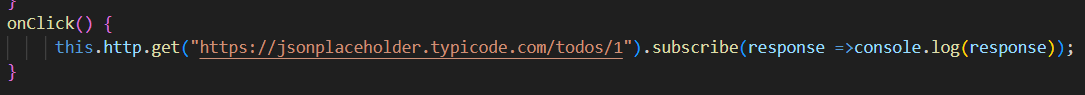


1. Keep it in shared interceptor module

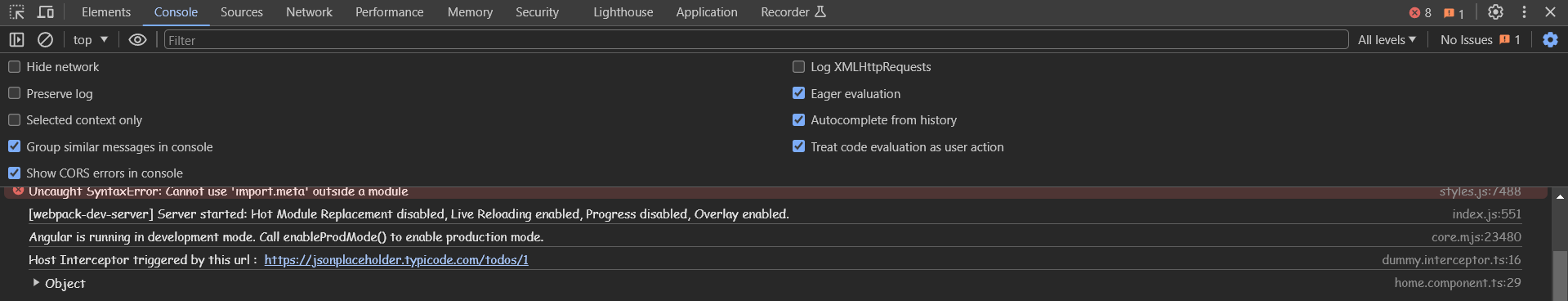


1. Home Component having button to fetch JSON data from a web end point

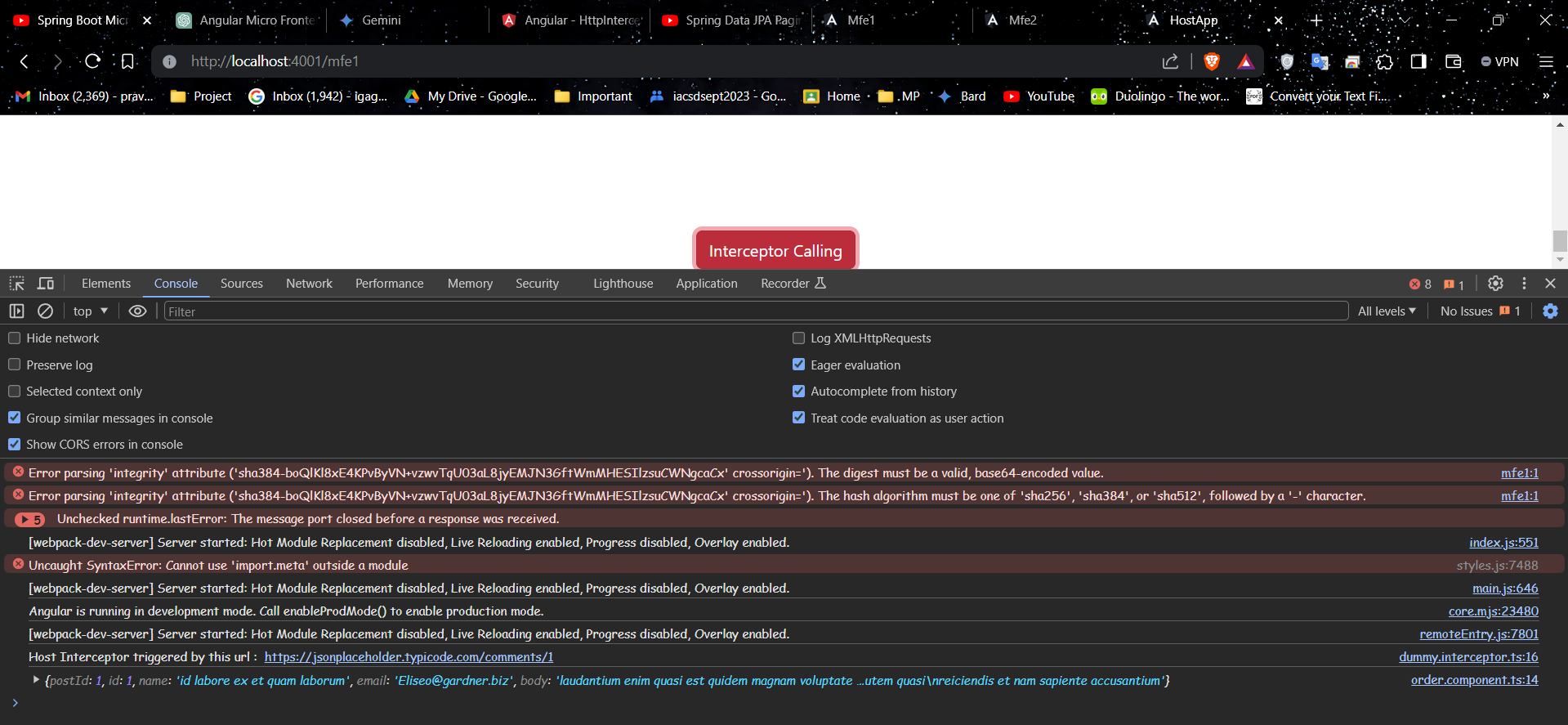




Interceptor working on host-app making http request.



Interceptor working on mfe1 making http request on host app web end point.



So this is how Interceptor works on Micro Front-Ends.