# Angular Module Federation Steps (non-dynamic)

1. Add this to your *package.json* right above the dependencies sections:

```
"resolutions": {
  "webpack": "^5.0.0"
},
```

2. Configure Ng CLI to use yarn using this command in your command line:

```
ng config cli.packageManager yarn
```

3. Add the *@angular-architects/module-federation package* to both your shell Angular application and your remote Angular applications using this command:

```
ng add @angular-architects/module-federation --project <<NAME_OF_SHELL> --port 5000
```

```
ng add @angular-architects/module-federation --project <<NAME_OF_REMOTE>> --port 3000
```

   a. Doing this will complete the following tasks for you:
      i. Generates the skeleton of a partial webpack.config.js for using module federation (sits on top of the Angular webpack config that the CLI handles
      ii. Installing a custom builder making webpack within the CLI use the generated webpack.config.js.
      iii. Assigning a new port for ng serve so that several projects can be served simultaneously.
      iv. Sets up your project's bootstrap.ts and main.ts files for dynamic imports

4. Add a route to the shell router for the new remote module like this :

```
export const APP_ROUTES: Routes = [
[…]
  {
    path: 'vii',
    loadChildren: () => import(vii/Module').then(m => m.HomeModule)
  },
[…]
```

5. The shell will not have any awareness of what vii/Module is so we need to add a module declaration. Add a *decl.d.ts* file to the root of your shell app and put a line in it that looks like this:

```
declare module 'vii/Module';
```

# Angular Module Federation Steps (non-dynamic)

6. In the shell app's *webpack.config.js* add any remotes you wish to reference and add any npm packages you would like shared:

```
  plugins: [
    new ModuleFederationPlugin({
        remotes: {
            vii: "vii@http://localhost:7000/remoteEntry.js"
        },
        shared: {
          "@angular/core": { singleton: true, strictVersion: true },
          "@angular/common": { singleton: true, strictVersion: true },
          "@angular/router": { singleton: true, strictVersion: true },
    […]
  ],
[…]
```

   a. NOTES:
      i. Defining a shared package as a singleton will make it only exist in memory 1 time instead of being loaded separately by each app, resulting in smaller bundle size
      ii. Setting strictVersion to true means that both apps are required to use the same version of that package in their *package.json*. If this is false or omitted Webpack will attempt to use the highest version available.

7. In the remote app's *webpack.config.js* add the definition for that remote and what modules or components it exposes and add any npm packages you would like shared:

```
plugins: [
    new ModuleFederationPlugin({
        // For remotes (please adjust)
        name: "vii",
        filename: "remoteEntry.js",
        exposes: {
            './vii: './angular/vii/src/app/home/home.module.ts',
        },
        shared: {
          "@angular/core": { singleton: true, strictVersion: true },
          "@angular/common": { singleton: true, strictVersion: true },
          "@angular/router": { singleton: true, strictVersion: true },
        […]
    ],
  […]
```