MICRO-FRONTEND WITH ANGULAR

➢ *What are Microfrontends?*

Microfrontends or Module Federation is a front-end web development pattern in which a single application may be build from disparate builds. It is analogous to a microservices approach but for client-side single-page applications written in JavaScrips.

➢ *Core Ideas behind Microfrontends*

Be Technology Agnostic : Each team should be able to choose and upgrade their stack without having to coordinate with other teams.

Isolate Team Code : Don’t share a runtime, even if all teams use the same framework. Build independent apps that are self contained. Don’t rely on shared state or global variables.

Establish Team Prefixes : Agree on naming conventions where isolation is not possible yet. Namespace CSS, Events, Local storage and Cookies to avoid collisions and clarify ownership.

Favor Native Browser Features over custom APIs : Use Browser Events for communication instead of building a global Pubsub system. If you really have to build a cross team API, try keeping it as simple as possible.

Build a Resilient site : Your feature should be useful, even if JavaScript failed or hasn’t executed yet. Use Universal Rendering and Progressive Enhancement to improve perceived performance.

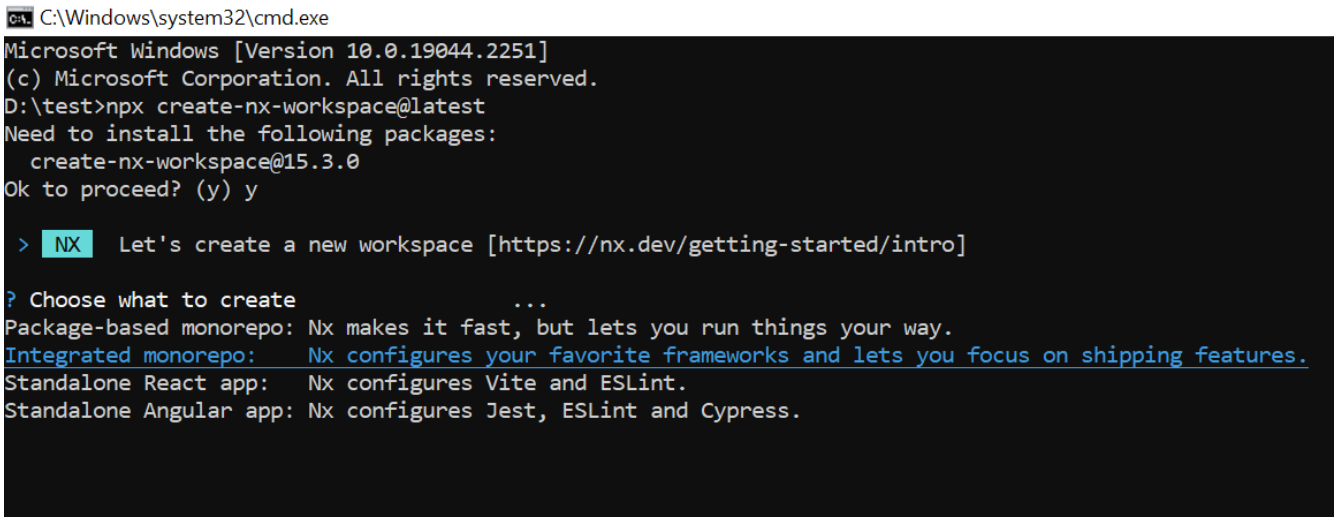
Step by Step Guide

***In this guide we will use Nx to create workspace which is just a such a mono-repository in which we can store multiple apps and libraries.*

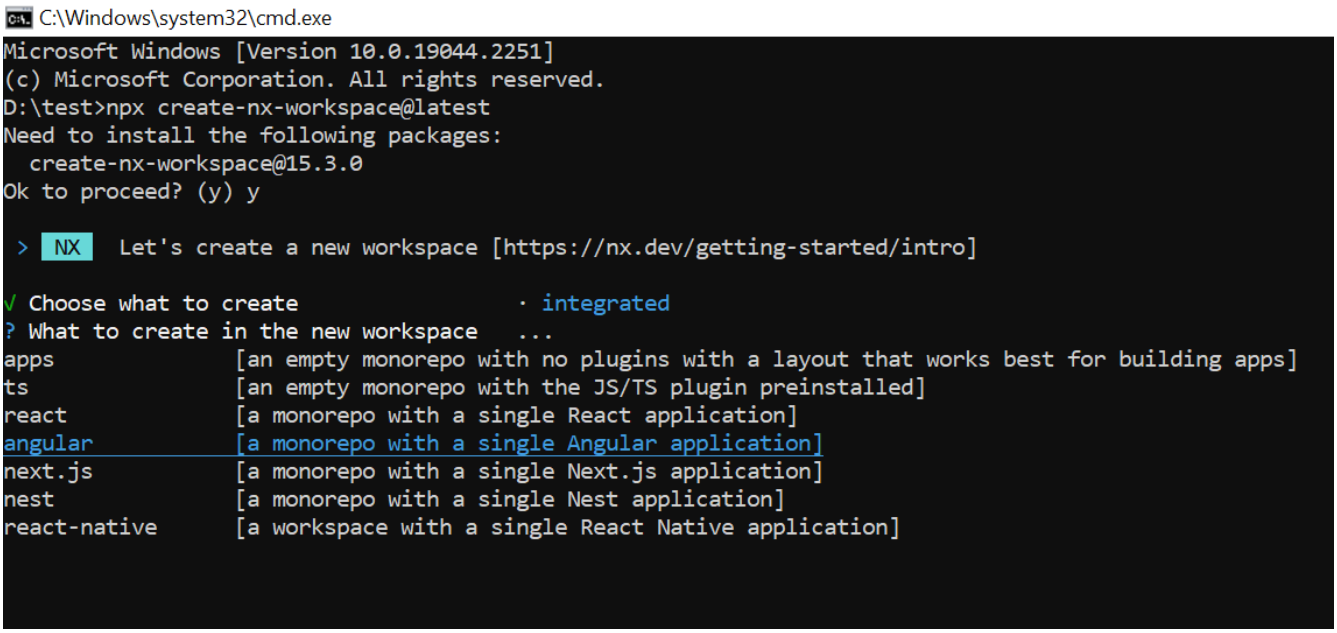
STEPS

1. ***Create a Nx workspace.***

   ```
   npx create-nx-workpace@latest
   ```



*you will get a prompt in which you can select **integrated monorepo***



*another prompt for what to create in workspace, choose **angular***

```
npm install
Microsoft Windows [Version 10.0.19044.2251]
(c) Microsoft Corporation. All rights reserved.
D:\test>npx create-nx-workspace@latest
Need to install the following packages:
  create-nx-workspace@15.3.0
Ok to proceed? (y) y

 > NX   Let's create a new workspace [https://nx.dev/getting-started/intro]

√ Choose what to create                 · integrated
√ What to create in the new workspace   · angular-monorepo
√ Repository name                       · microfrontend
√ Application name                      · shell
√ Default stylesheet format             · css
√ Enable distributed caching to make your CI faster · Yes

 > NX   Nx is creating your v15.3.0 workspace.

   To make sure the command works reliably in all environments, and that the preset is applied correctly,
   Nx will run "npm install" several times. Please wait.

- Installing dependencies with npm
```

give repository name, application name and select style

2. *Install Nx*

```
npm install -g nx
```

3. *Create an Angular application "remote"*

```
nx generate @nrwl/angular:application remote
```

```
C:\Windows\system32\cmd.exe
D:\test>npm install g -nx

added 1 package, and audited 2 packages in 1s

found 0 vulnerabilities

D:\test>cd microfrontend

D:\test\microfrontend>nx generate @nrwl/angular:application remote

 > NX   Generating @nrwl/angular:application

√ Would you like to configure routing for this application? (y/N) · true
[NX] Angular devkit called `writeWorkspace`, this may have created 'workspace.json' or 'angular.json
[NX] Double check workspace configuration before proceeding
Skipping remote since apps\remote\project.json already exists.
```

4. *Add Module Federation in both the application "remote & shell"*

first install module federation using below command.

```
npm install –save-dev @angular-architects/module-federation
```

Then add module-federation using below command. You will have to run this for each

```
nx g @angular-architects/module-federation:init
```

*select **remote** to add it on remote application and **shell** to add in shell and enter the port number while prompted i.e. **4201** for remote and **4200** for shell.*

```
C:\Windows\system32\cmd.exe - "C:\Users\ashishkumars\nvm\nodejs\bin\\node.exe"  "C:\Users\ashishkumars\nvm\nodejs\bin\\node_
D:\test\microfrontend>npm install --save-dev @angular-architects/module-federation

added 24 packages, removed 16 packages, changed 2 packages, and audited 1565 packages in 12s

175 packages are looking for funding
  run `npm fund` for details

4 high severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.

D:\test\microfrontend>nx g @angular-architects/module-federation:init

> NX  Generating @angular-architects/module-federation:init

? Project name (press enter for default project) ...
remote-e2e
shell-e2e
remote
shell
```

```
D:\test\microfrontend>nx g @angular-architects/module-federation:init

> NX  Generating @angular-architects/module-federation:init

√ Project name (press enter for default project) · remote
√ Port to use · 4201
Using Nx builders!
√ Packages installed successfully.
CREATE apps/remote/webpack.config.js
CREATE apps/remote/webpack.prod.config.js
CREATE apps/remote/src/bootstrap.ts
UPDATE tsconfig.base.json
UPDATE apps/remote/tsconfig.app.json
UPDATE workspace.json
UPDATE package.json
UPDATE apps/remote/src/main.ts

? Project name (press enter for default project) ...
remote-e2e
shell-e2e
remote
shell
```

```
D:\test\microfrontend>nx g @angular-architects/module-federation:init

> NX  Generating @angular-architects/module-federation:init

√ Project name (press enter for default project) · remote
√ Port to use · 4201
Using Nx builders!
√ Packages installed successfully.
CREATE apps/remote/webpack.config.js
CREATE apps/remote/webpack.prod.config.js
CREATE apps/remote/src/bootstrap.ts
UPDATE tsconfig.base.json
UPDATE apps/remote/tsconfig.app.json
UPDATE workspace.json
UPDATE package.json
UPDATE apps/remote/src/main.ts

√ Project name (press enter for default project) · shell
√ Port to use · 4200
Using Nx builders!
| Installing packages (npm)...
```

CONFIGURING REMOTE APPLICATION

So now we will create a simple *AbcModule* which only routes to blank Abc component. Then we will expose this module by adjusting the *webpack.config.js*

1. Creating AbcModule and Routing

```
ng generate module abc –project=remote –routing
```

C:\Windows\system32\cmd.exe

```
D:\test\microfrontend\apps\remote\src\app>ng generate module abc --project=remote --routing

> NX  Generating @schematics/angular:module

CREATE apps/remote/src/app/abc/abc-routing.module.ts
CREATE apps/remote/src/app/abc/abc.module.ts
```

2. Create Abc Component

```
ng generate component abc –project=remote
```

Note:- I have created one more component "xyz Component" which will be the part of AbcModule

C:\Windows\system32\cmd.exe

```
D:\test\microfrontend\apps\remote\src\app>ng generate component abc --project=remote

> NX  Generating @nrwl/angular:component

CREATE apps/remote/src/app/abc/abc.component.html
CREATE apps/remote/src/app/abc/abc.component.spec.ts
CREATE apps/remote/src/app/abc/abc.component.ts
CREATE apps/remote/src/app/abc/abc.component.css
UPDATE apps/remote/src/app/abc/abc.module.ts
UPDATE tsconfig.base.json
```

3. Configure abc-routing.module.ts

abc-routing.module.ts U

apps > remote > src > app > abc > TS abc-routing.module.ts > routes > component

```ts
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { AbcComponent } from './abc.component';
import { XyzComponent } from './xyz/xyz.component';

const routes: Routes = [
  {
    path: 'abc',
    component: AbcComponent
  },
  {
    path: 'xyz',
    component: XyzComponent
  }
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class AbcRoutingModule { }

```

4. Configure app.module.ts with routes

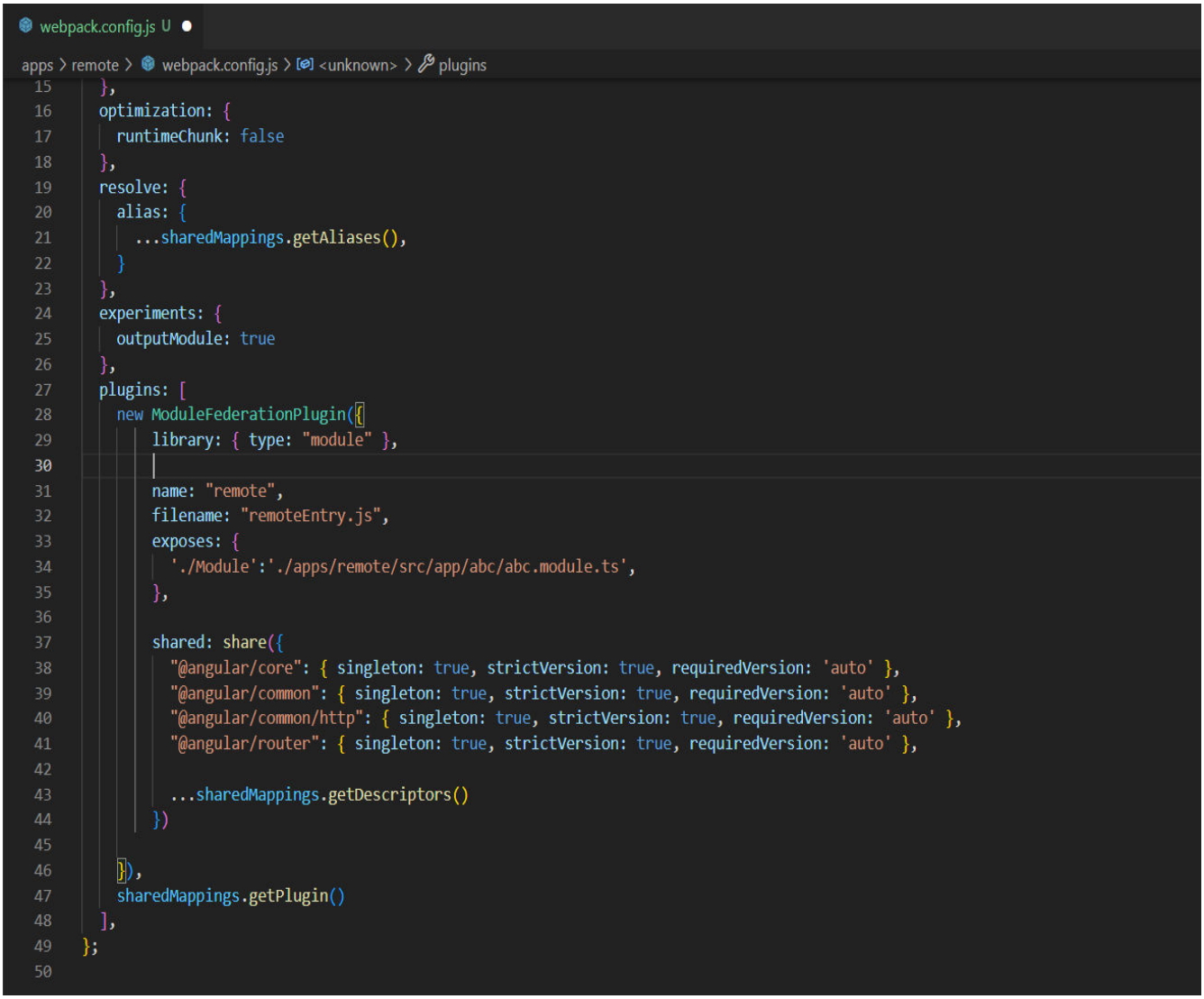
app.module.ts U

apps > remote > src > app > TS app.module.ts > AppModule

```ts
import { AppComponent } from './app.component';
import { NxWelcomeComponent } from './nx-welcome.component';
import { RouterModule, Routes } from '@angular/router';
import { AppRoutingModule } from './app.routes';
// import { appRoutes } from './app.routes';

const routes: Routes = [
  {
    path: '',
    loadChildren: () => import('./abc/abc.module').then( m => m.AbcModule)
  }
]

@NgModule({
  declarations: [AppComponent, NxWelcomeComponent],
  imports: [
    BrowserModule,
    RouterModule.forRoot(routes, { initialNavigation: 'enabledBlocking' }),
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}

```

5. Expose AbcModule in webpack.config.js

Only add remote configurations under "plugin object" changes are from line 31 to 35



*The **remoteEntry.js** file is auto-generated and needed to give instructions to outside applications containing the webpackconfig configurations. It exposes tells which mappings should be shared and which things it exposes. If you run the remote application you could also just look into this file by adding it to the url like this: **https://localhost:4200/remoteEntry.js***

CONFIGURING SHELL APPLICATION

1. Adjust and configure remotes in webpack.config.js in shell application

**you only need to add remotes as its shown at line 31*

webpack.config.js U

apps > shell > webpack.config.js > <unknown> > plugins

```
1  const ModuleFederationPlugin = require("webpack/lib/container/ModuleFederationPlugin");
2  const mf = require("@angular-architects/module-federation/webpack");
3  const path = require("path");
4  const share = mf.share;
5
6  const sharedMappings = new mf.SharedMappings();
7  sharedMappings.register(
8    path.join(__dirname, '../../tsconfig.base.json'),
9    [/* mapped paths to share */]);
10
11 module.exports = {
12   output: {
13     uniqueName: "shell",
14     publicPath: "auto"
15   },
16   optimization: {
17     runtimeChunk: false
18   },
19   resolve: {
20     alias: {
21       ...sharedMappings.getAliases(),
22     }
23   },
24   experiments: {
25     outputModule: true
26   },
27   plugins: [
28     new ModuleFederationPlugin({
29         library: { type: "module" },
30
31         remotes: {},
32
33         shared: share({
34           "@angular/core": { singleton: true, strictVersion: true, requiredVersion: 'auto' },
35           "@angular/common": { singleton: true, strictVersion: true, requiredVersion: 'auto' },
36           "@angular/common/http": { singleton: true, strictVersion: true, requiredVersion: 'auto' },
37           "@angular/router": { singleton: true, strictVersion: true, requiredVersion: 'auto' },
```

2. Load the microfrontend lazily when the route is activated.

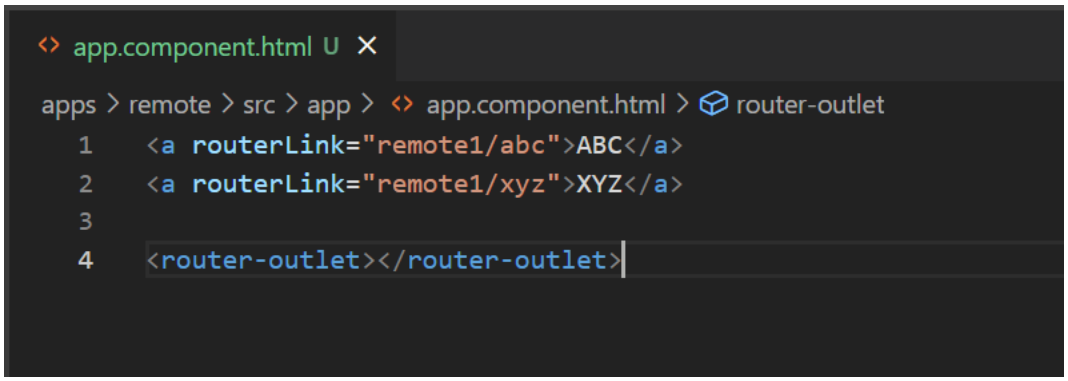
update your ***app.routes.ts*** *as shown below and import* ***AppRoutingModule*** *in your* ***app.module.ts***

app.routes.ts U

apps > remote > src > app > app.routes.ts > routes

```
1  import { Route, RouterModule, Routes } from '@angular/router';
2  import { loadRemoteModule } from '@angular-architects/module-federation'
3  import { NgModule } from '@angular/core';
4
5  export const routes: Routes = [
6      {
7          path: 'remote1',
8          loadChildren: () => loadRemoteModule({
9              remoteEntry: 'http://localhost:4201/remoteEntry.js',
10             type: 'module',
11             exposedModule: './Module',
12         })
13         .then( m => m.AbcModule)
14     }
15 ];
16
17 @NgModule({
18     imports: [
19         RouterModule.forRoot(routes)
20     ],
21     exports: [RouterModule]
22 })
23 export class AppRoutingModule {}
```

3. Adapt app.component.html to have router-outlet and a routerLink.



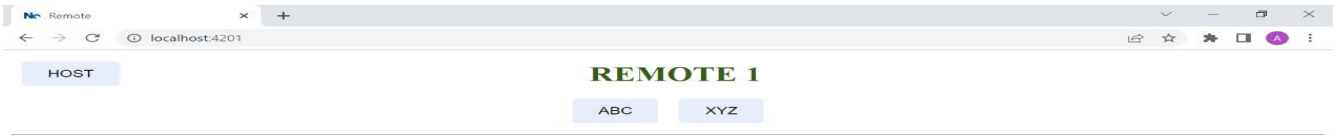
Everything is set to run now and should work. Let's start both applications, but keep in mind that we should start the microfrontends before the shell.

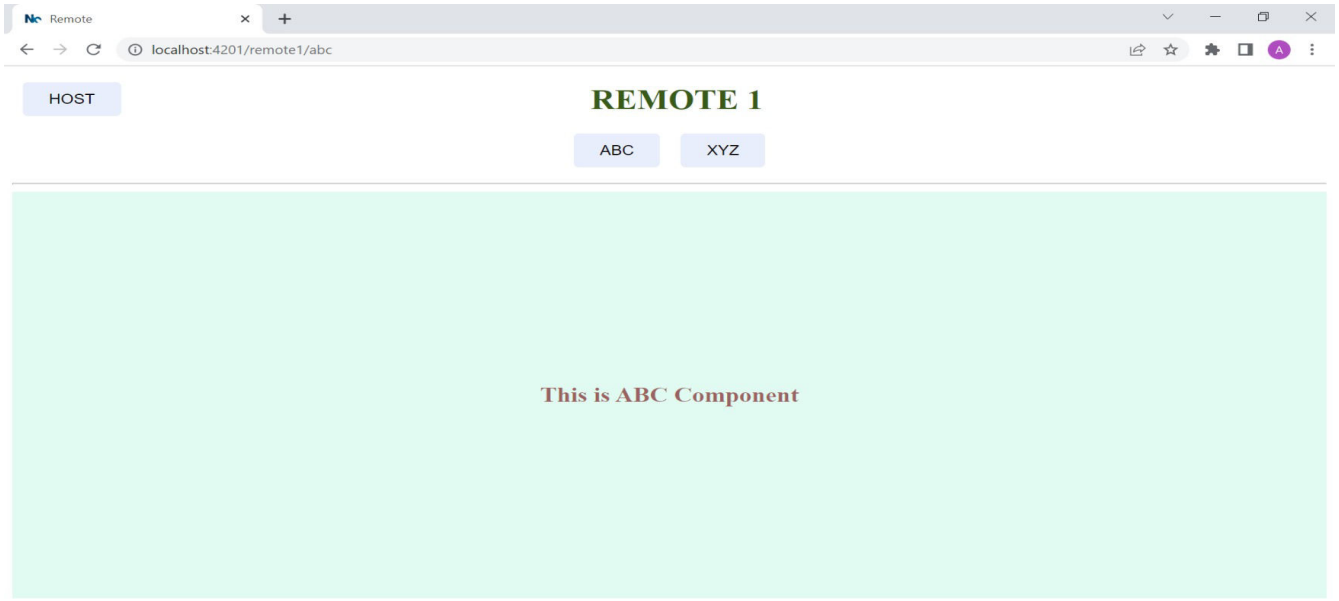
```
to run remote use : nx serve remote
to run shell use  : nx serve shell
```

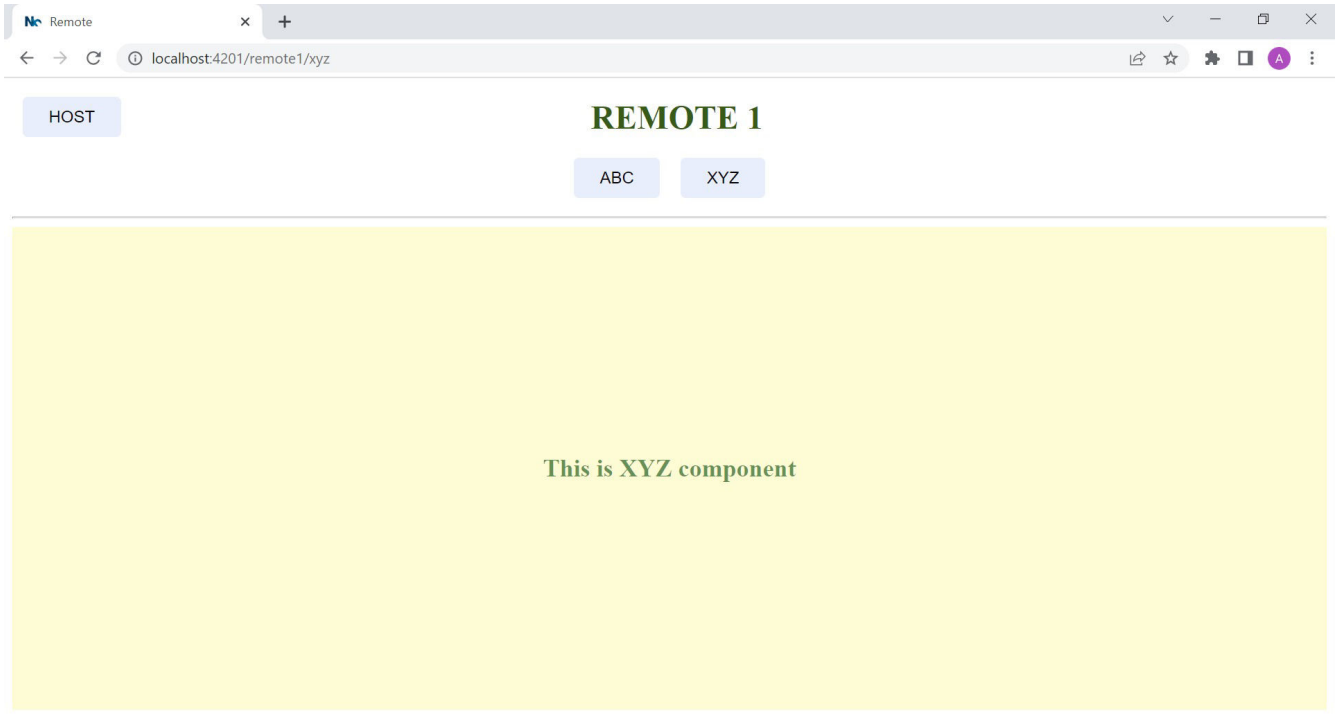
after successful compilation you can navigate to ***http://localhost:4200***

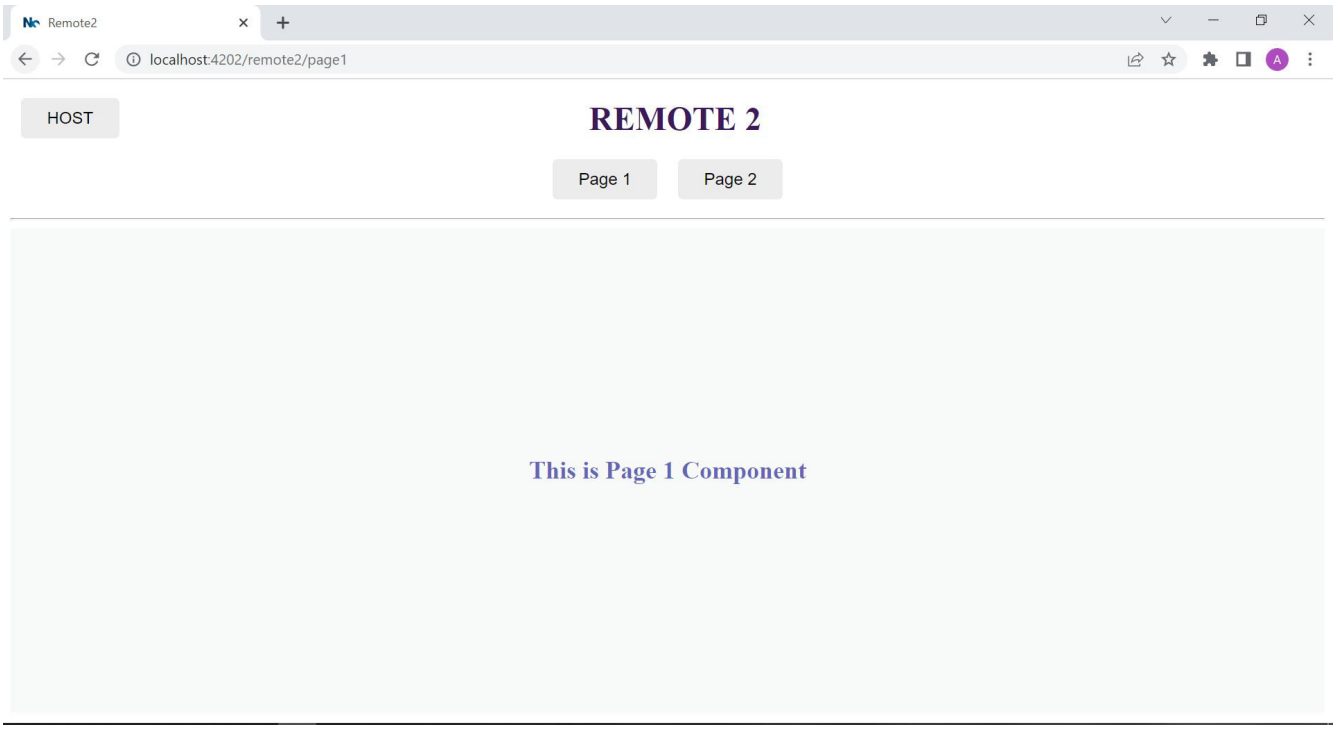
****Note:- I have also added "remote2" application for testing. For this u need to create application in the same way as "remote" and the add module-federation in remote2 by giving port. After that configure you webpack.config.js under remote2. If the configuration is perfect then it should run fine. You can clone the this demo application with the url shared at the end.***

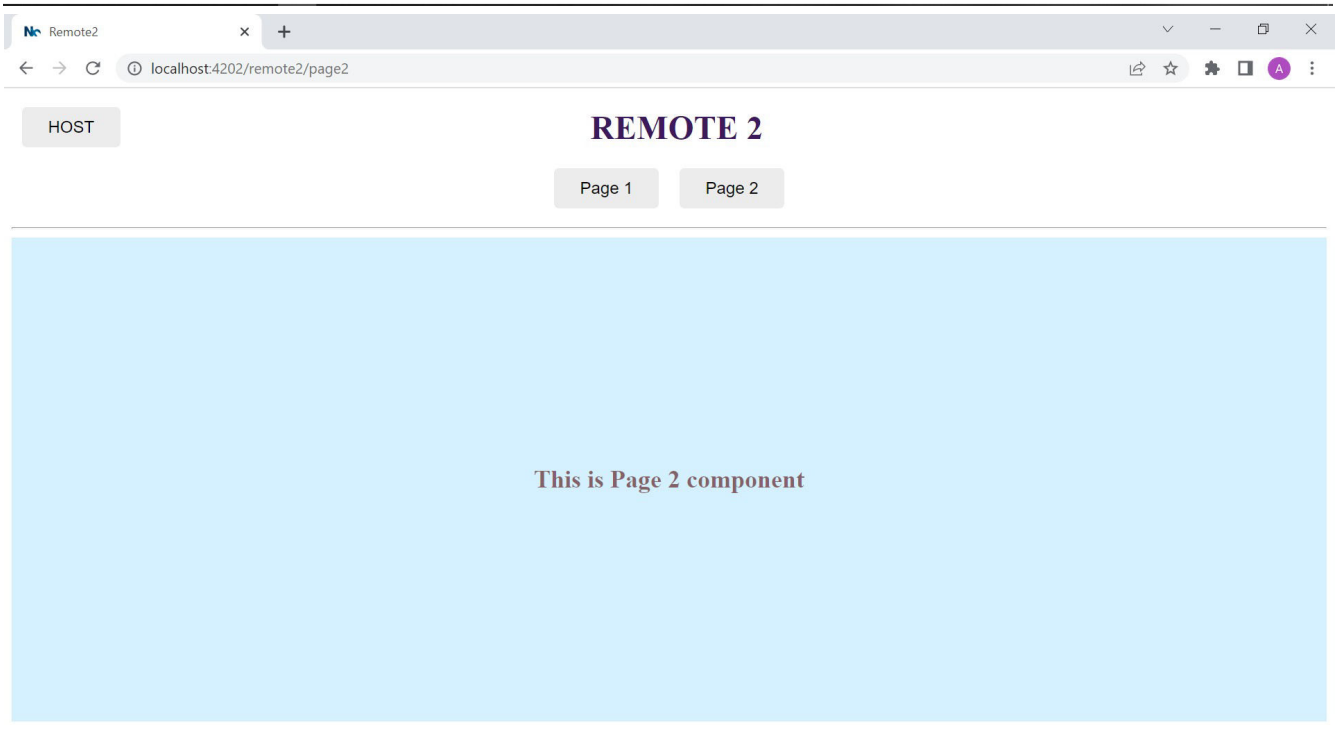
The UI for this application would be like the images shown below. For which you only have to navigate to the url http://localhost:4200 which is the HOST page through which you can check the Remote1 or Remote2 application. If you notice all the three application i.e. shell, remote and remote2 will be running simultaneously having different localhost port, but within the same application as a standalone.


Remote
localhost:4201
HOST
REMOTE 1
ABC
XYZ


Remote
localhost:4201/remote1/abc
HOST
REMOTE 1
ABC
XYZ
This is ABC Component


Remote
localhost:4201/remote1/xyz
HOST
REMOTE 1
ABC
XYZ
This is XYZ component





To get this application running in you machine just run the below commands step by step:-

1. git clone https://github.com/Js-Team-Git/microfrontend.git
2. npm install -g nx
3. npm install
4. nx serve remote (to run the "remote" application)
5. nx serve remote2 (to run the "remote2" application)
6. nx serve shell   (to run the host "shell")

now navigate to the url ***http://localhost:4200****.*
ENJOY!!!! THE APPLICATION WILL BE UP AND RUNNING