Step - 1

CREATE THREE APPLICATIONS

- A. CONTAINER APPLICATION - `NPX CREATE-REACT-APP CONTAINER --TEMPLATE=TYPESCRIPT`
- B. HOMEPAGE - `NPX CREATE-REACT-APP HOMEPAGE --TEMPLATE=TYPESCRIPT`
- C. COURSES - `NPX CREATE-REACT-APP COURSES --TEMPLATE=TYPESCRIPT`

• CONTAINER APPLICATION ACTS AS THE HOST APPLICATION FOR TWO OTHER REMOTE APPLICATIONS

Step - 2

CREATE MODULEFEDERATION.CONFIG.JS FILE IN BOTH THE REMOTE AND CONTAINER APPLICATIONS AND COPY THE BELOW CODE

```
const { dependencies } = require("./package.json");
module.exports = {
name: "",
filename: "remoteEntry.js",
remotes: {},
exposes: {},
}
```

Line 1: dependencies are imported from the package.json file
Line 4: The name: “mainpage" property is the name of the current micro-frontend. This name is used when
this micro-frontend is consumed by others.

LINE 5: THE FILENAME: "REMOTEENTRY.JS" PROPERTY IS THE NAME OF THE FILE THAT WILL BE CREATED BY WEBPACK.

THIS FILE WILL CONTAIN INFORMATION ABOUT THE EXPOSED MODULES AND SHARED DEPENDENCIES.

LINE 6 : THE REMOTE: {} IS USED TO SPECIFY WHICH REMOTE MICRO-FRONTENDS IT CONSUMES.

LINE 7: THE EXPOSES: {} IS USED TO SPECIFY WHICH MODULES THIS MICRO-FRONTEND EXPOSES TO OTHERS

ADD THE BELOW CONFIGURATION IN HOMEPAGE AND COURSES MICRO-FRONTENDS UNDER REMOTES PROPERTY

```
exposes: {
    "./HomePage": "./src/App",
  },
```

Here App component and count-reducer is exposed from the homepage
micro-frontends.

Remotes property lists all the remote application that a container application can import module from

- Key is the name of the remote application
- Value is the URL where the remote application's remoteEntry.js file can be found

```
remotes: {
    mainpage: "mainpage@http://localhost:3010/remoteEntry.js",
    coursepage: "coursepage@http://localhost:3009/remoteEntry.js",
  },
```

ADD THE ABOVE CONFIGURATION IN MODULEFEDERATION.CONFIG.JS FILE IN CONTAINER APPLICATION

```
shared: {
    ...dependencies,
    react: {
      singleton: true,
      import: "react",
      shareScope: "default",
      requiredVersion: dependencies.react,
    },
    "react-dom": {
      singleton: true,
      requiredVersion: dependencies["react-dom"],
    },
  },
```

LINE 1: SHARED PROPERTY IS USED TO LIST THE DEPENDENCIES THAT ARE SHARED AMONG THE MICRO-FRONTENDS. HERE IT SHARES DEPENDENCIES LISTED IN PACKAGE.JSON AND REACT AND REACT-DOM PACK

LINE 4 & 10: SINGLETON PROPERTY ENSURES THAT ONLY ONE VERSION OF REACT AND REACT-DOM PACKAGES ARE LOADED EVEN IF MULTIPLE MICRO-FRONTENDS LOAD THEM. HAVING MULTIPLE VERSIONS CAN LEAD TO CONFLICTS

LINE 5: THE IMPORT: "REACT" PROPERTY UNDER REACT SPECIFIES THE MODULE THAT SHOULD BE SHARED.

LINE 6: THE SHARESCOPE: "DEFAULT" PROPERTY UNDER REACT DEFINES THE SHARED SCOPE. THIS IS THE NAME UNDER WHICH THE SHARED MODULES WILL BE STORED IN THE CONSUMING APPLICATION.

LINE 7 & 11: REQUIREDVERSION PROPERTY SPECIFIES THE VERSION OF THE MODULE THAT HOST EXP

1. ADD REMOTETYPES.D.TS FILE IN CONTAINER APPLICATION IN SRC FOLDER AND PASTE THE BELOW CODE.

```
///<reference types="react" />
```

2. THIS DEFINES THE TYPE OF MODULE THAT ARE BEING IMPORTED FROM OTHER REMOTE APPLICATIONS.

```
declare module "coursepage/Course" {
  const Course: React.ComponentType;
  export default Course;
}

declare module "mainpage/HomePage" {
  const HomePage: React.ComponentType;
  export default HomePage;
}
```

Step - 3

CREATE WEBPACK.CONFIG.JS FILE IN CONTAINER AND REMOTE APPLICATIONS IN THE ROOT FOLDER AND PASTE THE BELOW CODE

```
const { ModuleFederationPlugin } = require('webpack').container;
const fs = require('fs');
const path = require('path');

const webpackConfigPath = 'react-scripts/config/webpack.config';
const webpackConfig = require(webpackConfigPath);

const override = (config) => {
const mfConfigPath = path.resolve(__dirname, 'moduleFederation.config.js');

if (fs.existsSync(mfConfigPath)) {
const mfConfig = require(mfConfigPath);
config.plugins.push(new ModuleFederationPlugin(mfConfig));
config.output.publicPath = 'auto';
}
return config;};
```

LINE 1: OVERRIDE FUNCTION MODIFIES A GIVEN CONFIGURATION OBJECT TO USE WITH WEBPACK

LINE 2: MODULEFEDERATION CONFIGURATION FILE PATH IS CONFIGURED

LINE 4-8: MODULEFEDERATION CONFIGURATION OBJECT IS ADDED TO THE PLUGINS ARRAY OF WEBPACK CONFIGURATION OF CREATE-REACT-APP

LINE 10: OVERRIDE FUNCTION RETURNS THE MODIFIED CONFIGURATION OBJECT

```
require.cache[require.resolve(webpackConfigPath)].exports = (env) =>
override(webpackConfig(env));

module.exports = require(webpackConfigPath);
```

LINE 1-2: THE CODE IS MANIPULATING NODE.JS'S REQUIRE.CACHE TO MODIFY THE EXPORTS OF A WEBPACK CONFIG MODULE

REQUIRE.RESOLVE METHOD IS USED TO FIND THE ABSOLUTE PATH OF THE WEBPACK CONFIGURATION FILE

THE CODE THEN ASSIGNS A NEW FUNCTION TO THE EXPORTS PROPERTY OF THE MODULE OBJECT IN THE REQUIRE.CACHE.

IN ESSENCE, THIS CODE MODIFIES THE DEFAULT WEBPACK CONFIGURATION FROM CREATE-REACT-APP

Step-4

IN THE NEXT STEP, CREATE A FOLDER NAMED SCRIPTS WITH ONE FILE; START.JS IN THE ROOT FOLDER OF ALL THE APPLICATIONS
PASTE THE BELOW CODE IN START.JS

```
process.env.NODE_ENV = process.env.NODE_ENV || "development";
require("../webpack.config");
require("react-scripts/scripts/start");
```

Line 1: setting the environment variable

Line 2: importing the custom configuration object from webpack.config

Line 3: importing the start.js file from the react-scripts package

In essence, this code starts a react application in development environment

Step - 5

CREATE BOOTSTRAP.TSX FILE IN SRC FOLDER OF ALL THE THREE APPLICATIONS AND COPY ALL CODE FROM INDEX.TSX TO BOOTSTRAP.TSX

UPDATE INDEX.TSX WITH BELOW CODE. IT LOADS BOOTSTRAP.TSX ON DEMAND

```
IMPORT("./BOOTSTRAP");
EXPORT {};
```

Step - 6

MODIFY THE PACKAGE.JSON START SCRIPTS AS SHOWN BELOW

```
"START": "SET PORT=3008 && NODE ./SCRIPTS/START.JS",
```

UPDATE THE PORT NUMBER FOR EACH APPLICATION

ONCE DONE, RUN THE APPLICATION USING NPM START COMMAND

```
IMPORT './APP.CSS';
IMPORT { BROWSERROUTER AS ROUTER, ROUTE, ROUTES, LINK } FROM 'REACT-ROUTER-DOM';
IMPORT HOMEPAGE FROM 'MAINPAGE/HOMEPAGE';
IMPORT COURSES FROM 'COURSEPAGE/COURSE';

FUNCTION APP() {
  RETURN (
    <ROUTER>
      <DIV>
        <NAV CLASSNAME='NAVBAR'>
          <UL>
            <LI>
              <LINK TO="/">HOME</LINK>
            </LI>
            <LI>
              <LINK TO="/COURSES">COURSES</LINK>
            </LI>
          </UL>
        </NAV>
        <ROUTES>
          <ROUTE PATH="/" ELEMENT={<HOMEPAGE />} />
          <ROUTE PATH="/COURSES" ELEMENT={<COURSES />} />
        </ROUTES>
      </DIV>
    </ROUTER>
  );
}

EXPORT DEFAULT APP;
```