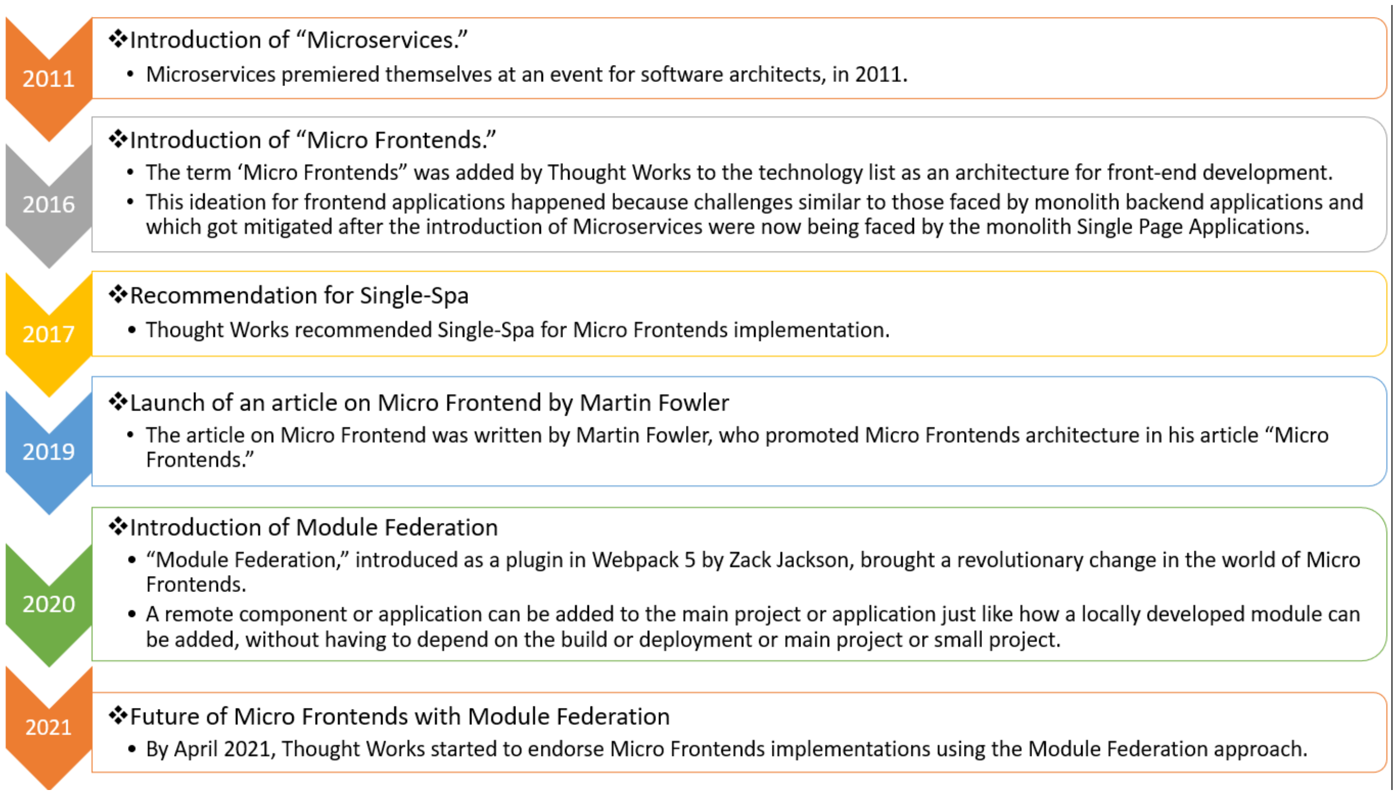
# 1. Introduction

## 1.1 History of Micro FrontEnds



2011

❖Introduction of “Microservices.”

- Microservices premiered themselves at an event for software architects, in 2011.

2016

❖Introduction of “Micro Frontends.”

- The term ‘Micro Frontends” was added by Thought Works to the technology list as an architecture for front-end development.
- This ideation for frontend applications happened because challenges similar to those faced by monolith backend applications and which got mitigated after the introduction of Microservices were now being faced by the monolith Single Page Applications.

2017

❖Recommendation for Single-Spa

- Thought Works recommended Single-Spa for Micro Frontends implementation.

2019

❖Launch of an article on Micro Frontend by Martin Fowler

- The article on Micro Frontend was written by Martin Fowler, who promoted Micro Frontends architecture in his article “Micro Frontends.”

2020

❖Introduction of Module Federation

- “Module Federation,” introduced as a plugin in Webpack 5 by Zack Jackson, brought a revolutionary change in the world of Micro Frontends.
- A remote component or application can be added to the main project or application just like how a locally developed module can be added, without having to depend on the build or deployment or main project or small project.

2021

❖Future of Micro Frontends with Module Federation

- By April 2021, Thought Works started to endorse Micro Frontends implementations using the Module Federation approach.

## 1.2 Micro FrontEnd Architecture

The Micro Frontend architecture has a strong resemblance to the microservices architecture.

In this Micro Frontend architecture of front-end web applications, instead of building one single monolithic front-end application, the UI of the web applications necessarily becomes a composition of Micro Frontends.

For example, take into consideration an eCommerce site. Features like product listing, cart, etc., will be developed as two independent front-end applications. These applications represent a small or micro portion of the wholesome applications, referred to as Micro Frontend applications.

A micro frontend can hence be

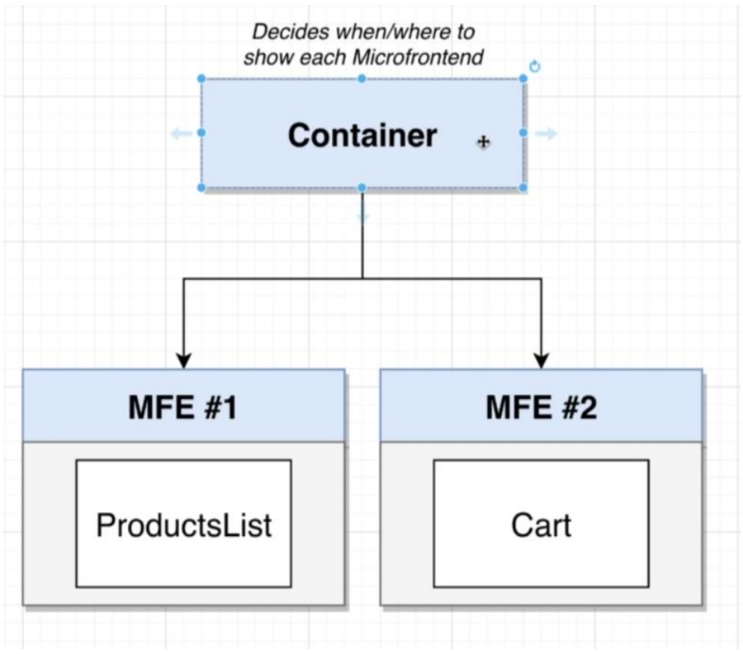
- a complete page (e.g., a product list page or a cart page which is part of the final front-end application) or
- individual fragments/portions of the same page (e.g., search bars, ads, headers, etc.)

The main website can, therefore, be split up by type of page or feature or use-case, and each type can be assigned to an independent team to work on.

The independent team will be responsible for end-to-end development, code quality, user experience, and deployment of the respective Micro Frontend application.

In short, the Micro-Frontends architecture consists of 3 main components:

- Independent Micro Frontends, which can also be called remote applications
- A Host/Container application where the independent Micro Frontends are composed or assembled
- A Micro Frontend integration approach

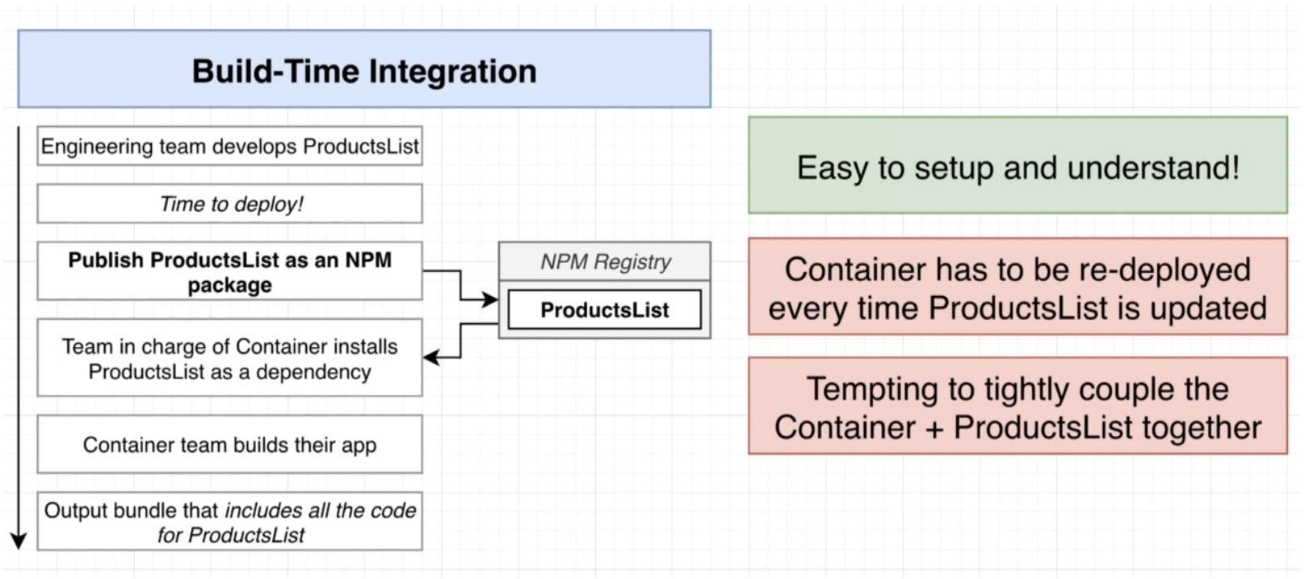


### 1.3 Micro FrontEnd Approaches

There are two main approaches in which independent Micro Frontend applications may be integrated are:

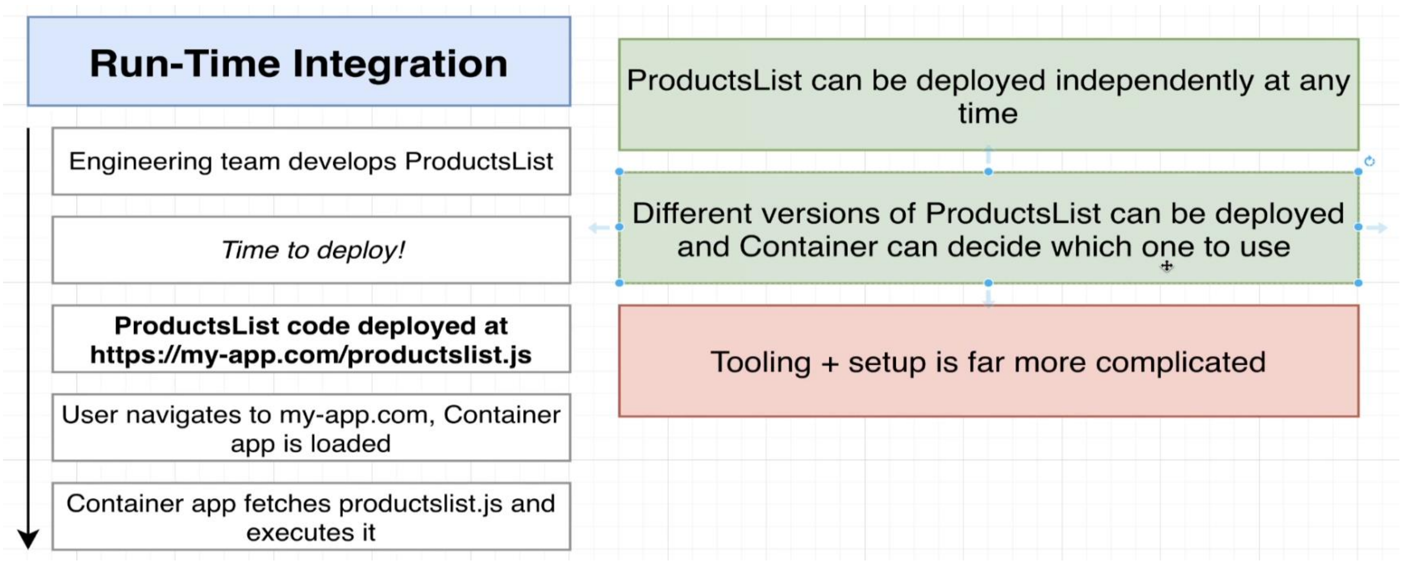
#### 1.3.1 Build-Time integration

In the build-time integration way of developing Micro Frontends, the apps are independently created and published as Node packages or modules. This is then added as a dependency for the container application in its package.json. The container is then deployed.


Build-Time Integration
Engineering team develops ProductsList
Time to deploy!
Publish ProductsList as an NPM package
NPM Registry
ProductsList
Team in charge of Container installs ProductsList as a dependency
Container team builds their app
Output bundle that includes all the code for ProductsList
Easy to setup and understand!
Container has to be re-deployed every time ProductsList is updated
Tempting to tightly couple the Container + ProductsList together

### 1.3.2 Run-Time integration

Each Micro Frontend deploys its JavaScript run-time packages to a container like an AWS S3 bucket in run-time integration. The host/container application uses/downloads those components at run time and uses them to render content.


Run-Time Integration
Engineering team develops ProductsList
Time to deploy!
ProductsList code deployed at https://my-app.com/productslist.js
User navigates to my-app.com, Container app is loaded
Container app fetches productslist.js and executes it
ProductsList can be deployed independently at any time
Different versions of ProductsList can be deployed and Container can decide which one to use
Tooling + setup is far more complicated

### 1.3.3 Comparison of various integration approaches

| **Application Development** | **Monolithic Frontend** | **Micro Frontend with build-time integration** | **Micro Frontend with run-time integration** |
|---|---|---|---|
| Approach | No integration, as entire codebase is available in one single repository | Each dependent web application is npm installed by the root or host application | Each independently deployed dependent web application is dynamically loaded by the root or host application |
| Complexity in setting up | Easy | Medium | Advanced |
| Code repositories | Single | Single | May be separate |
| Application Build | Single build. May need to rebuild as a whole when needed | Separate build | Separate build |

|  |  |  |  |
|---|---|---|---|
| Application deployment | Single deployment. May need to redeploy as a whole when needed | Each Micro Frontend can be independently deployed without depending on others. | Each Micro Frontend can be independently deployed without depending on others. |
| Advantages | Simple to create | Each application is built separately before it's published to npm | Each Micro Frontend is developed and deployed independently. Provides better scalability |
| Disadvantages | The build will be slow. Deployments are all tied up. | If any Micro Frontend is modified, the host has to reinstall, rebuild and redeploy itself. | Knowledge of the Micro Frontends and their relationships to the root application is required to be known. |

### 1.4 Frameworks Available for Micro FrontEnds

There are several frameworks in the market today that can help develop Micro Frontends. Some of the popular frameworks are:

- Webpack Module Federation Plugin:

Module Federation is a JavaScript architecture that permits JavaScript applications to get the code dynamically imported from another application during runtime. The highlight of Module Federation is an easy dependency management and less code duplication in the host application.

Webpack, very popular in Frontend application development, has added a new plugin for Module Federation called Webpack Module Federation Plugin in Webpack5. This makes it even easier to build Micro Frontends as Micro Frontend applications can now dynamically import code from other applications at runtime from different URLs.

- Single SPA framework:

Single SPA framework allows placing different Micro Frontends into a single DOM by integrating them at runtime in the host application. The specialty of Single SPA compared to traditional SPA is that there is no restriction to stick to a specific JavaScript library or library. The applications need to be equipped to co-exist with other applications. The host application will have a DOM placeholder for each remote application, and the remote application can be added to the respective DOM placeholder meant for it.

- Open Components

OpenComponents is an open-source framework that helps teams quickly build and deploy front-end components. It can contain logic as a server-side Node.js application. It can also be used to include HTML and CSS. It highlights is it doesn't restrict developers to just JavaScript but can also use libraries like Ruby, PHP, etc. This adds more flexibility in working with existing environments of the project rather than rebuilding them.

- Bit

Bit provides a product-ready solution for building Micro Frontends. It can take separate components and mold them into Micro Frontends. It gives a CICD pipeline as well. It works similarly to Webpack, just that it works at build time. However, with the Module Federation plugin, Webpack can give an advantage of run-time integration, bringing in more decoupling.

Other frameworks available are:

- SystemJS
- Mosaic 9
- PuzzleJS
- Piral, etc

## 1.5 Factors to consider before shortlisting a framework

- Team Size

If the proposed development team is tiny, then some of the critical benefits of Micro Frontend architecture will be lost. The overhead involved with Micro Frontend architecture can make it hard for one team to manage multiple different components effectively. As the developer switches his work from coding to deployment for the various component slices, the purpose of deploying Micro Frontends is defeated.

- Component Communication

In Micro Frontend architecture, every component must be wrapped inside a Custom Element. As per the architecture, DOM has to be leveraged DOM for information sharing among different parts for which element attributes can be used for facilitating component communications. Without this declarative approach, object references may get lost in transition. This component communication is a key that needs to be considered when the Micro Frontend frameworks are chosen.

- Thinking Micro Frontend from Monolithic

When building Micro Frontend from a Monolithic by splitting it, care should be taken to avoid slicing it into very thin Micro Frontends.

- Usability

Define the goals before getting started with Micro Frontend implementation. This gives grounds for choosing a framework that is more aligned with the end goal of the application. Hence, looking at the application's usability aspect is essential before deciding on the framework.

- The choice between SSR or CSR

If search engine optimization(SEO) is a matter of concern for the application, Server-Side Rendering (SSR) is suitable. However, if SEO is not a concern, Client-Side Rendering (CSR) is a better choice. Frameworks can be accordingly chosen then.

Initially, only server-side composition was the possible approach. With NPM, build-time compositions are now possible. Also, with advances in Webpack, run-time compositions on the client are now possible. Hence, developers prefer to consider Single-SPA and Module Federation, which support run-time integration of Micro Frontends.

If assembling is a matter of concern, Single-SPA may be chosen.

If splitting a heavy Monolithic into manageable Micro Frontends, Module Federation may be preferred.

For legacy websites, however, it is preferred to use Server-side or Build-time composition while including them in the newly created Micro Frontend.

## 2. Module Federation

### 2.1 What is Module Federation?

- Module Federation is a JavaScript architecture that Zack Jackson invented.
- Module Federation simplifies our app architecture by making it easy to utilize and share code and dependencies between codebases of two or more different applications.
- The architecture permits JavaScript applications to get the code dynamically imported from another application during runtime. In case any dependency goes missing, the host application will take care of downloading the dependency. This introduces the possibility of significantly less code duplication in the application.
- With the introduction of this cool feature, developers who were previously using frameworks like single-spa, etc., for developing MicroFrontEnd Apps have moved to the Module Federation approach.
- With new Webpack5, which also contains the Module Federation plugin, the MicroFrontApps can now be built more easily using Module Federation as the collaboration between the master-mind Zack and Webpack team resulted in the introduction of the Module Federation Plugin into Webpack5.

### 2.2 Advantages of Module Federation

- As Module Federation is not a framework but a plugin that is added to Webpack, it gives complete freedom and flexibility to build the project the way the developer wants.
- Module Federation integrates components at run-time; hence, there is no need to worry about deployments or dependencies of other micro frontends.
- Solves the challenges associated with dependency between codes and larger bundle sizes for the MicroFrontEnds by facilitating dependency sharing.
  As an example: If the React source is already present, then with Module Federation in place, you won't need to reimport React code again. The module will reuse the React source that is already available and import the component code.
  Also, for Microfrontends built with React, React. lazy and React. suspense can be used to offer a fallback in case the imported code fails for some reason, ensuring there won't be any disruption to the user experience because of the failing build.

### 2.3 When to use Module Federation?

Module Federation can be of great use when:

- The project in hand is quite large, and there is a possibility of this application being built as a monolith by following traditional architectures.
- It is possible to divide the app into self-contained systems.
- Multiple, independent teams may develop the application
- The development teams may agree on a frontend framework (like Angular/React) for harmonized development and maintenance.

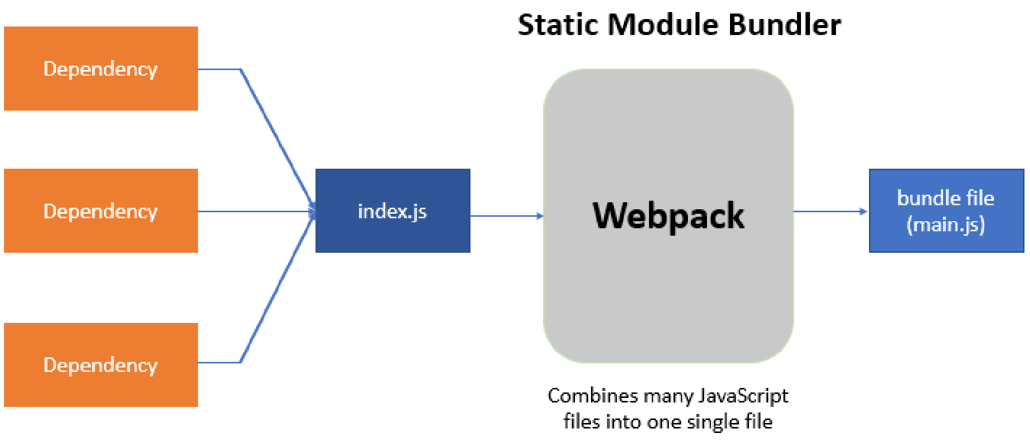
### 2.4 When NOT to use Module Federation?

An alternate choice over Module Federation can be considered when:

- The project in hand may not contain an exhaustive list of features and may not be expanding in future
- The individual teams may want to develop individual micro frontends different frameworks and major versions. A lot of benefits of Module Federation will be lost in this context
- There is less control over frameworks and their versions being used.
- Expecting a framework change for one of Microfrontend shortly.
- Webpack is not being used as Module Bundler.
- Older versions of Angular like <v11 are being used, which don't have support for Webpack5.

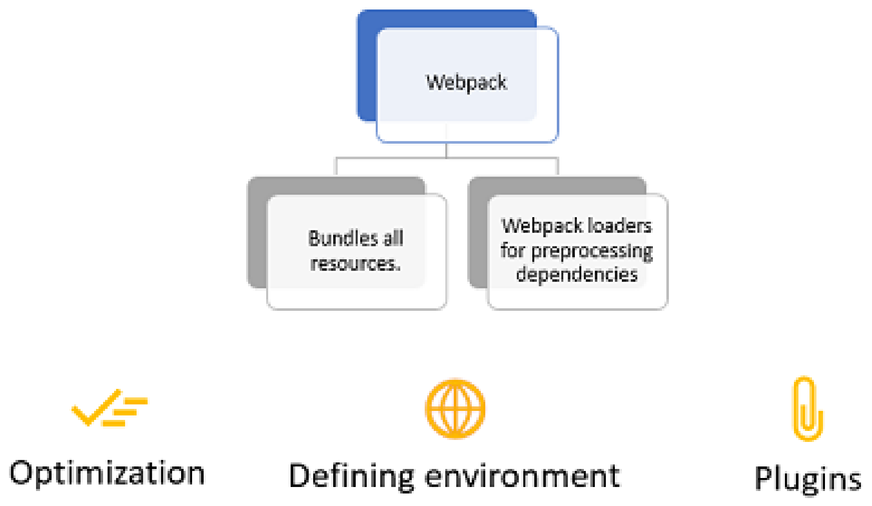
### 2.5 Webpack Overview

Webpack is a static module bundler for modern JavaScript applications.



While bundling, it repeatedly builds a hierarchy of modules used in the application. Based on this created hierarchy, it makes one or more JavaScript files at the end of bundling.

Suppose you build a complex front end with many static assets like images, CSS, fonts, etc. Webpack will offer you great benefits. React JS, Angular CLI, and Vue CLI uses Webpack to bundle their assets in the application in the background. Webpack follows its parsing logic and creates one or more bundles. It is very flexible to accommodate a variety of JS files and bundles formats like ejs, ES modules, etc.



**HTML Webpack Plugin**

HTML Webpack Plugin simplifies the creation of HTML files for loading your webpack bundles.

Install the plugin using the command:

```
npm install html-webpack-plugin --save-dev
```

Import the package in webpack.config.js:

```
var HtmlWebpackPlugin = require('html-webpack-plugin');
```

Using this plugin, there are two approaches by which the bundles can be served:

Option 1: Either let the plugin create the HTML template.

The plugin will create an HTML5 file with script tags containing all the webpack bundles added to the body. For this, it is needed to add the following code under the plugin array in the webpack.config.js file:

```
plugins:[
      new HtmlWebpackPlugin()
   ]
```

This will generate an index.html file in the public folder as follows:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>webpack App</title>
  </head>
  <body>
    <script src="bundle.js"></script>
  </body>
</html>
```

Option 2: Supply the template.

There will be scenarios where it is necessary to add the bundles to an existing HTML file. This can be done by specifying the template path in HTML Webpack plugin options as shown below:

```
plugins:[
      new HtmlWebpackPlugin({
         template:'./public/index.html'
      })
   ]
```

- template: path to the template.
- filename: Specifies the filename to write the HTML to. Defaults to index.html. A subdirectory can also be mentioned. For example, ./public/index.html

### 2.6 How the Module Federation approach works with Micro Frontends?

JobRecruitmentPlatfo
rm.zip

The below steps give a quick walkthrough of how Module Federation works:

1. Label one app as host and one as remote.
    - First, designate one application as a host and one as a remote.
    - The host is the application that is trying to make use of code from another project.
    - The remote is the project that makes that code available to other projects.

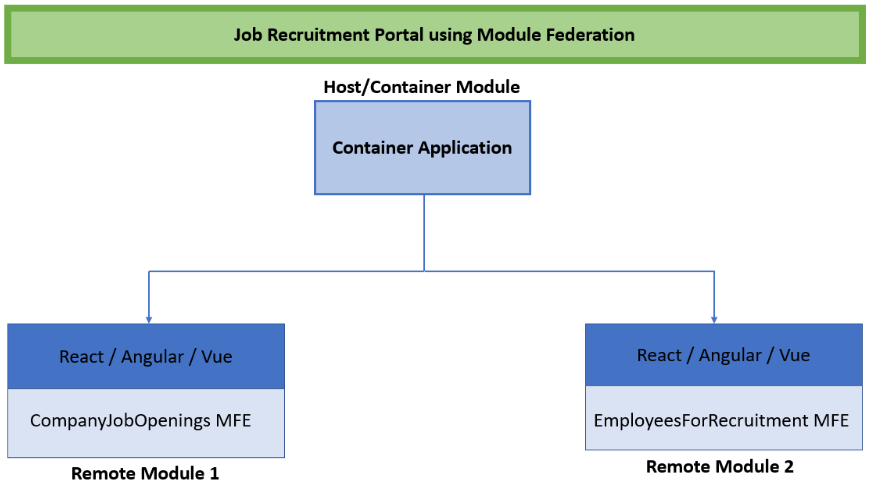
2. In the remote, decide which modules you want to make accessible to other projects.
3. Configure Module Federation in the remote to expose the files or modules selected.
    - Use the Module Federation plugin from Webpack to configure this.
    - So, in the webpack.config.js of the remote,

      require the module federation plugin,
      then in the plugins section add new ModuleFederationPlugin, which has a name, remoteEntry file name, and exposes object.

4. In the host, decide which files you want to get from the remote.
5. Configure Module Federation in the host to get the files or modules needed.
    - Use the Module Federation plugin from Webpack to configure this.
    - So in the webpack.config.js of the host, require the module federation plugin, then in the plugins section, add a new ModuleFederationPlugin, which has a name. This remote object can point to the remote federated module.
6. In the host, refactor the entry point to load asynchronously.
7. In the host, you can now import whatever files you need from the remote

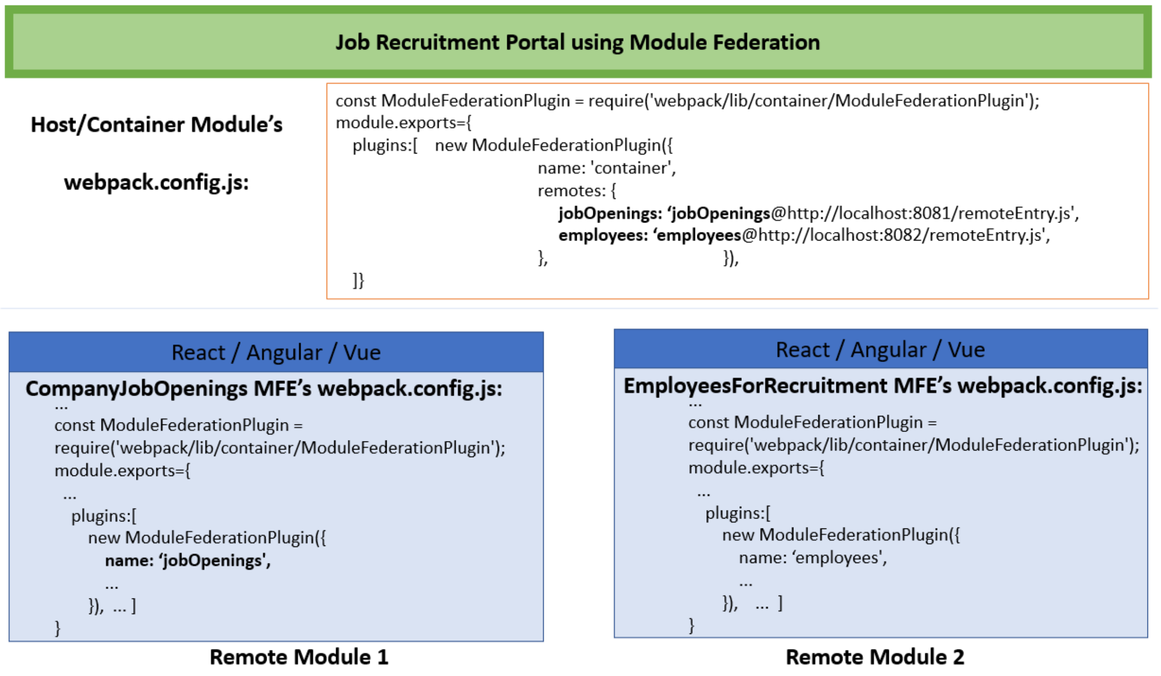
Consider the scenario where CompanyJobOpenings and EmployeesForRecruitment Microfrontend apps need to be consumed by another micro frontend app, Container.

With the Module Federation approach in place, the CompanyJobOpenings and EmployeesForRecruitment apps need to be configured as Remote Modules, and Container needs to be configured as the Host container Module.



In the webpack.config.js of the CompanyJobOpenings app, using Module Federation Plugin, the module can be exposed with the name ‘jobOpenings’. Likewise, for EmployeesForRecruitment, using Module Federation Plugin, the module can be told with the word ‘employees’.

Now in the webpack.config.js of the Container, using Module Federation Plugin, the remote modules to be used can be configured for consumption posts, which can be imported at required places within the Container app.



```
const ModuleFederationPlugin = require('webpack/lib/container/ModuleFederationPlugin');
module.exports={
   plugins:[    new ModuleFederationPlugin({
                                    name: 'container',
                                    remotes: {
                                        jobOpenings: ‘jobOpenings@http://localhost:8081/remoteEntry.js',
                                        employees: ‘employees@http://localhost:8082/remoteEntry.js',
                                    },                              }),
   ]}
```

**CompanyJobOpenings MFE’s webpack.config.js:**

```
   …
   const ModuleFederationPlugin =
   require('webpack/lib/container/ModuleFederationPlugin');
   module.exports={
    …
     plugins:[
       new ModuleFederationPlugin({
          name: ‘jobOpenings',
          …
       }), … ]
   }
```

**EmployeesForRecruitment MFE’s webpack.config.js:**

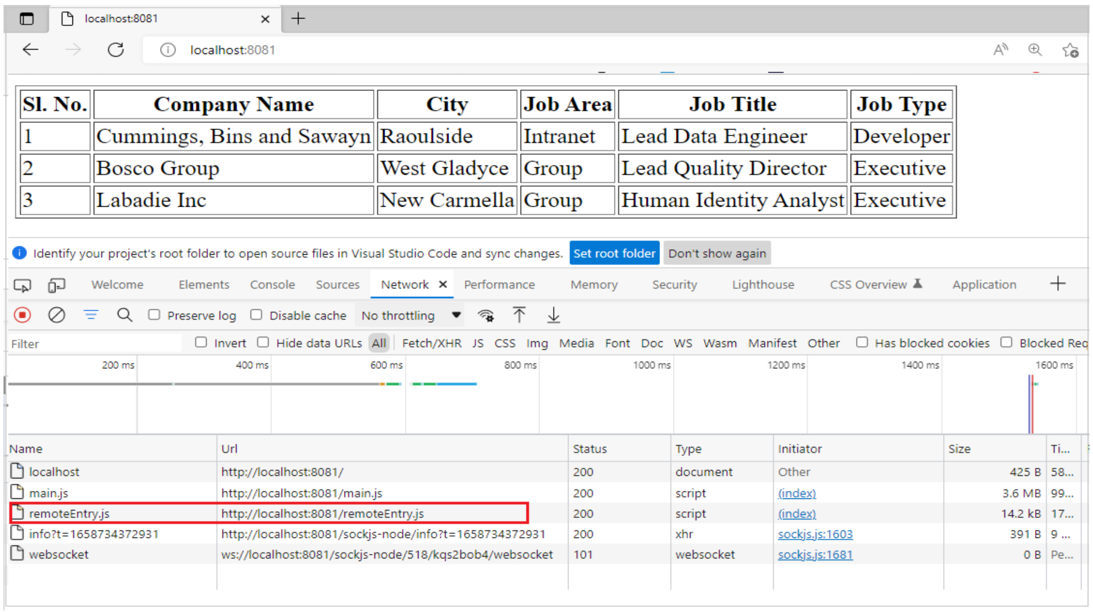
```
   …
   const ModuleFederationPlugin =
   require('webpack/lib/container/ModuleFederationPlugin');
   module.exports={
    …
     plugins:[
       new ModuleFederationPlugin({
          name: ‘employees',
          …
       }),   … ]
   }
```

### 2.7 Configuring Module Federation in Microfrontends in sample app

Open the webpack.config.js of CompanyJobOpenings.
Add import statement for Module Federation.
Now, add a new module federation plug-in and configurations in the plugins array.

- First, add a name of 'jobOpenings':
  This will be the name to access this micro frontend from the container.
- Next, A file name of remote entry.js:
  This will reference the manifest file/ the remote file that is being created.
- Next, exposes:
  This will be an object with the key of ./jobsIndex and the value of ./src/index.
  The exposes object contains which modules or files inside the CompanyJobOpenings project we wish to expose to the outside world. If there were many different files inside of CompanyJobOpenings, it's possible to export all of them, or just one of them, or as many as needed.
  Here, a module called JobsIndex is exposed. If anyone tries to import something called JobsIndex, then the src/index file will be given.
  So, in the container, to use the CompanyJobOpenings, the JobsIndex can be imported. Webpack will get the remoteEntry file, it will find the JobsIndex.

Restart the CompanyJobOpenings and observe the network tab at http://localhost:8081. Notice the remoteEntry.js being generated.



localhost:8081
localhost:8081
Identify your project's root folder to open source files in Visual Studio Code and sync changes.
Set root folder
Don't show again
Welcome Elements Console Sources Network Performance Memory Security Lighthouse CSS Overview Application
Preserve log
Disable cache
No throttling
Filter
Invert
Hide data URLs
All Fetch/XHR JS CSS Img Media Font Doc WS Wasm Manifest Other
Has blocked cookies
Blocked Req
200 ms 400 ms 600 ms 800 ms 1000 ms 1200 ms 1400 ms 1600 ms

| Sl. No. | Company Name | City | Job Area | Job Title | Job Type |
|---|---|---|---|---|---|
| 1 | Cummings, Bins and Sawayn | Raoulside | Intranet | Lead Data Engineer | Developer |
| 2 | Bosco Group | West Gladyce | Group | Lead Quality Director | Executive |
| 3 | Labadie Inc | New Carmella | Group | Human Identity Analyst | Executive |

```
Name                   Url                                                   Status  Type       Initiator      Size     Ti...
localhost              http://localhost:8081/                                200     document   Other          425 B    58...
main.js                http://localhost:8081/main.js                         200     script     (index)        3.6 MB   99...
remoteEntry.js         http://localhost:8081/remoteEntry.js                  200     script     (index)        14.2 kB  17...
info?t=1658734372931   http://localhost:8081/sockjs-node/info?t=1658734372931 200    xhr        sockjs.js:1603 391 B    9 ...
websocket              ws://localhost:8081/sockjs-node/518/kqs2bob4/websocket 101    websocket  sockjs.js:1681 0 B      Pe...
```

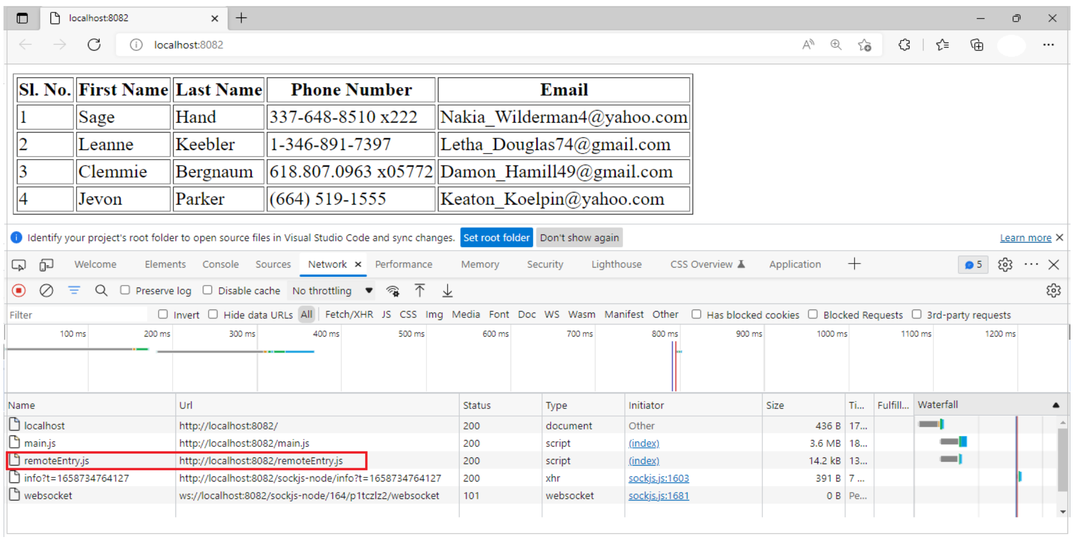
Configuring the remote: EmployeesForRecruitment MFE

Open the webpack.config.js of EmployeesForRecruitment.
Now, add a new module federation plugin and configurations in the plugins array.

- First, adding a name of 'employees':
  This will be the name to access this micro frontend from the container.
- Next, A file name of remote entry.js:
  This will be the name to reference the manifest file/ the remote file created.
- Next, exposes:
  This will be an object with the key of ./employeesIndex'and the value of ./src/index.

The exposes object contains which modules or files inside the EmployeesForRecruitment project we wish to expose to the outside world. If there were many different files inside of EmployeesForRecruitment, it's possible to export all of them, or just one of them, or as many as needed.
Here, a module called EmployeesIndex is exposed. If anyone tries to import something called EmployeesIndex, then the src/index file will be given.
So, in the container, to use the EmployeesForRecruitment, the EmployeesIndex can be imported. Webpack will get the remoteEntry file; it will find the EmployeesIndex.

Restart the EmployeesForRecruitment and observe the network tab at http://localhost:8082. Notice the remoteEntry.js being generated.



| Sl. No. | First Name | Last Name | Phone Number | Email |
|---|---|---|---|---|
| 1 | Sage | Hand | 337-648-8510 x222 | Nakia_Wilderman4@yahoo.com |
| 2 | Leanne | Keebler | 1-346-891-7397 | Letha_Douglas74@gmail.com |
| 3 | Clemmie | Bergnaum | 618.807.0963 x05772 | Damon_Hamill49@gmail.com |
| 4 | Jevon | Parker | (664) 519-1555 | Keaton_Koelpin@yahoo.com |

Identify your project's root folder to open source files in Visual Studio Code and sync changes.

| Name | Url | Status | Type | Initiator | Size | Ti... | Fulfill... |
|---|---|---|---|---|---|---|---|
| localhost | http://localhost:8082/ | 200 | document | Other | 436 B | 17... | |
| main.js | http://localhost:8082/main.js | 200 | script | (index) | 3.6 MB | 18... | |
| remoteEntry.js | http://localhost:8082/remoteEntry.js | 200 | script | (index) | 14.2 kB | 13... | |
| info?t=1658734764127 | http://localhost:8082/sockjs-node/info?t=1658734764127 | 200 | xhr | sockjs.js:1603 | 391 B | 7 ... | |
| websocket | ws://localhost:8082/sockjs-node/164/p1tczlz2/websocket | 101 | websocket | sockjs.js:1681 | 0 B | Pe... | |

Configuring the host: Container MFE

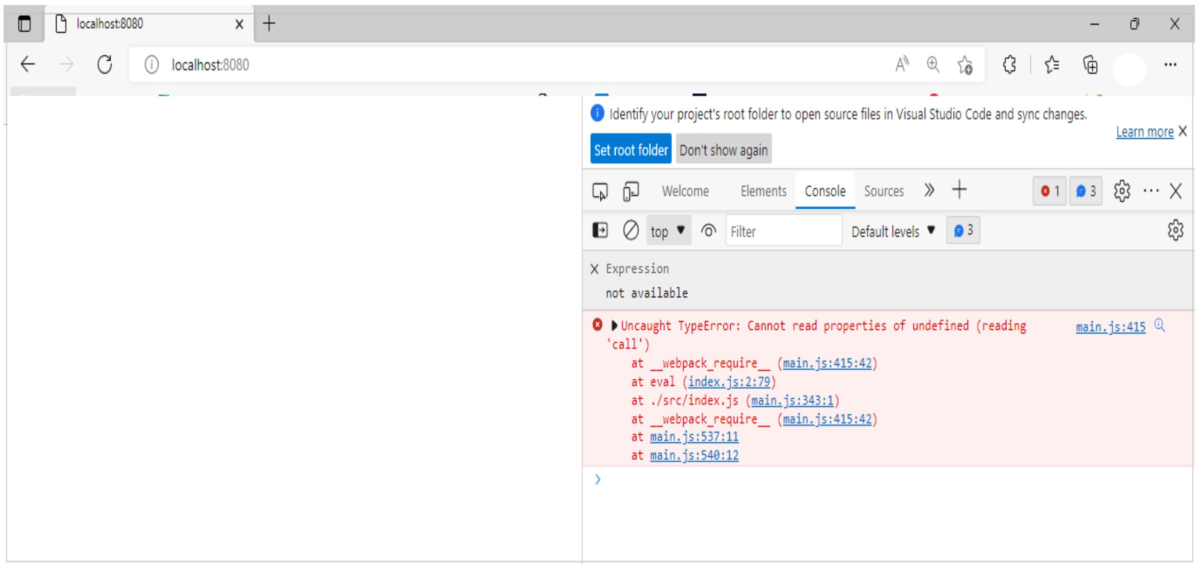
Open the webpack.config.js of Container.
Add import statement for Module Federation.
Now, add a new module federation plug-in and configurations in the plugins array.

- First, add the name of the container.
  This will be the name used to access this micro frontend.
- Next is remotes:
  This contains the list of projects containers that can search for additional code.
  The remote controls, how webpack will decide to load up that remote entry js file.
  If the jobOpenings key is used, Webpack will understand that the expectation is for a module that begins with the word jobOpenings. If webpack cant find the jobOpenings dependency inside the node_modules folder, then webpack will look at the remotes section of the module federation plugin. It will check for the presence of a key named 'jobOpenings' inside the remotes. This means the webpack must load up the remoteEntry file configured here.
  Likewise for the employees critical as well.

Using the remotes inside the container

To use CompanyJobOpenings and EmployeesForRecruitment  MFEs inside the container, import the files that are needed from the remotes in the Container MFE.

Open index.js of Container. Import 'jobOpenings/jobsIndex' and 'employees/employeesIndex'.
That's it! Remotes have been accessed in the host by now. Restart the container and notice the output in the browser.



Identify your project's root folder to open source files in Visual Studio Code and sync changes.

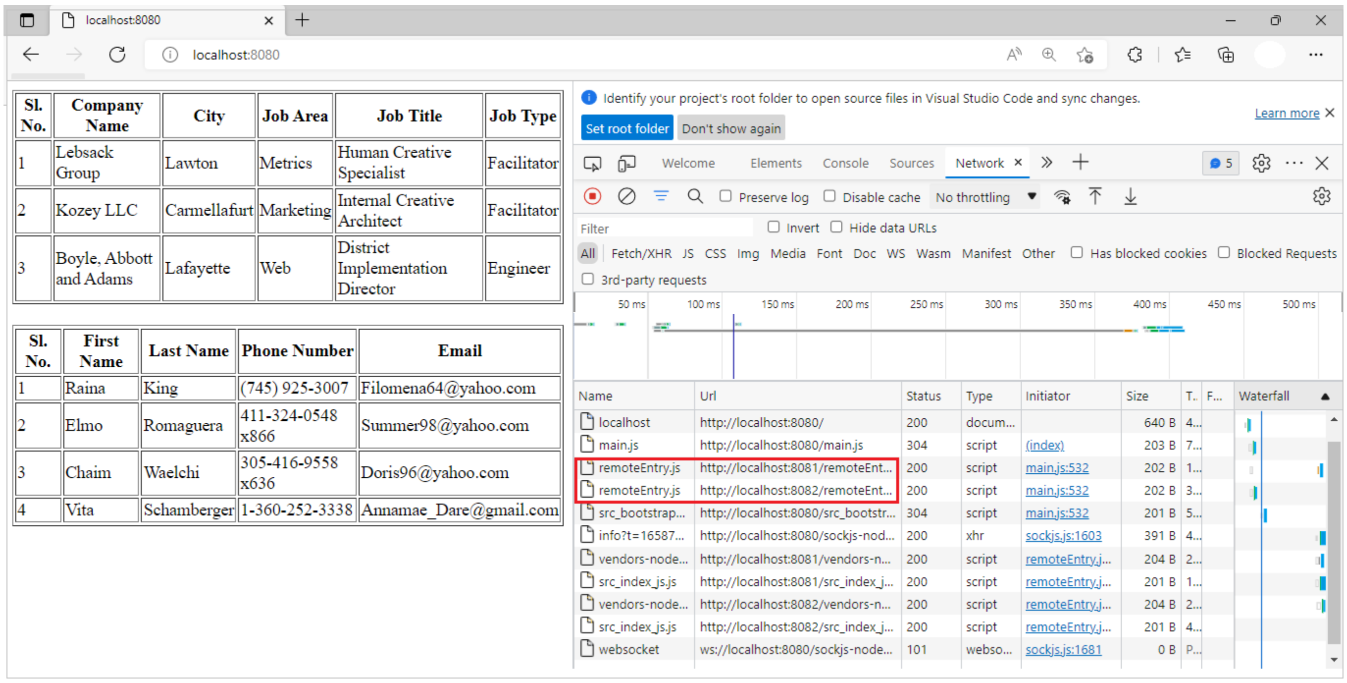
```
Expression
not available

Uncaught TypeError: Cannot read properties of undefined (reading    main.js:415
'call')
    at __webpack_require__ (main.js:415:42)
    at eval (index.js:2:79)
    at ./src/index.js (main.js:343:1)
    at __webpack_require__ (main.js:415:42)
    at main.js:537:11
    at main.js:540:12
```

Notice that the required call to index.js has failed. This is because, in index.js, the container expects content from jobsIndex and employeesIndex. This content needs to be made available before loading the index.js. This means there is a need to give Webpack extra space and time to find and load the remotes and then load the rest of the content. There is hence a need for asynchronous loading of index.js.

To update the code of the Host/Container MFE to load the entry point file, i.e., index.js of the remote MFEs asynchronously:

- Open index.js of the container and rename this to bootstrap.js.
- Next, create a new file called index.js in the same directory and add an import function call to the bootstrap.js file.



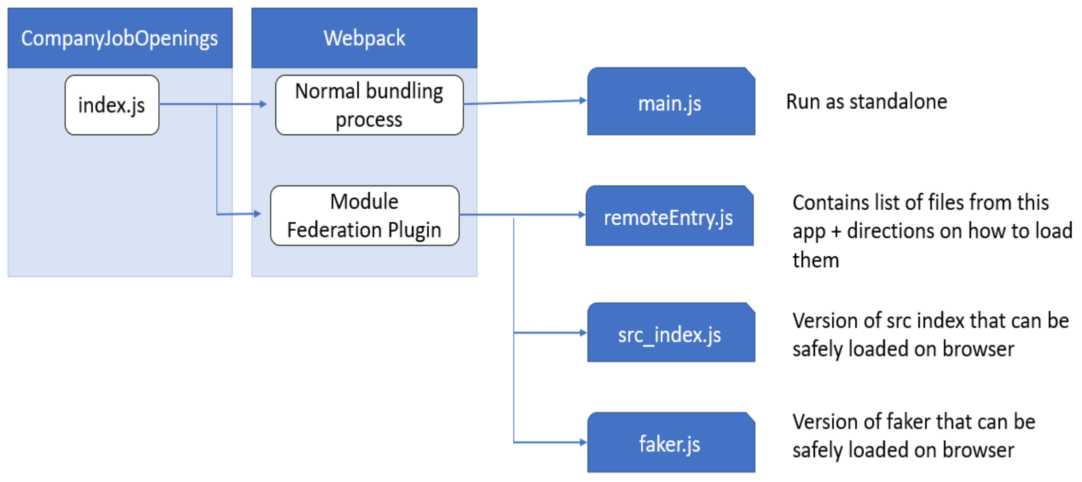
| Sl. No. | Company Name | City | Job Area | Job Title | Job Type |
|---|---|---|---|---|---|
| 1 | Lebsack Group | Lawton | Metrics | Human Creative Specialist | Facilitator |
| 2 | Kozey LLC | Carmellafurt | Marketing | Internal Creative Architect | Facilitator |
| 3 | Boyle, Abbott and Adams | Lafayette | Web | District Implementation Director | Engineer |

| Sl. No. | First Name | Last Name | Phone Number | Email |
|---|---|---|---|---|
| 1 | Raina | King | (745) 925-3007 | Filomena64@yahoo.com |
| 2 | Elmo | Romaguera | 411-324-0548 x866 | Summer98@yahoo.com |
| 3 | Chaim | Waelchi | 305-416-9558 x636 | Doris96@yahoo.com |
| 4 | Vita | Schamberger | 1-360-252-3338 | Annamae_Dare@gmail.com |

Identify your project's root folder to open source files in Visual Studio Code and sync changes.

| Name | Url | Status | Type | Initiator | Size | T... |
|---|---|---|---|---|---|---|
| localhost | http://localhost:8080/ | 200 | docum... | | 640 B | 4... |
| main.js | http://localhost:8080/main.js | 304 | script | (index) | 203 B | 7... |
| remoteEntry.js | http://localhost:8081/remoteEnt... | 200 | script | main.js:532 | 202 B | 1... |
| remoteEntry.js | http://localhost:8082/remoteEnt... | 200 | script | main.js:532 | 202 B | 3... |
| src_bootstrap... | http://localhost:8080/src_bootstr... | 304 | script | main.js:532 | 201 B | 5... |
| info?t=16587... | http://localhost:8080/sockjs-nod... | 200 | xhr | sockjs.js:1603 | 391 B | 4... |
| vendors-node... | http://localhost:8081/vendors-n... | 200 | script | remoteEntry.j... | 204 B | 2... |
| src_index_js.js | http://localhost:8081/src_index_j... | 200 | script | remoteEntry.j... | 201 B | 1... |
| vendors-node... | http://localhost:8082/vendors-n... | 200 | script | remoteEntry.j... | 204 B | 2... |
| src_index_js.js | http://localhost:8082/src_index_j... | 200 | script | remoteEntry.j... | 201 B | 4... |
| websocket | ws://localhost:8080/sockjs-node... | 101 | webso... | sockjs.js:1681 | 0 B | P... |

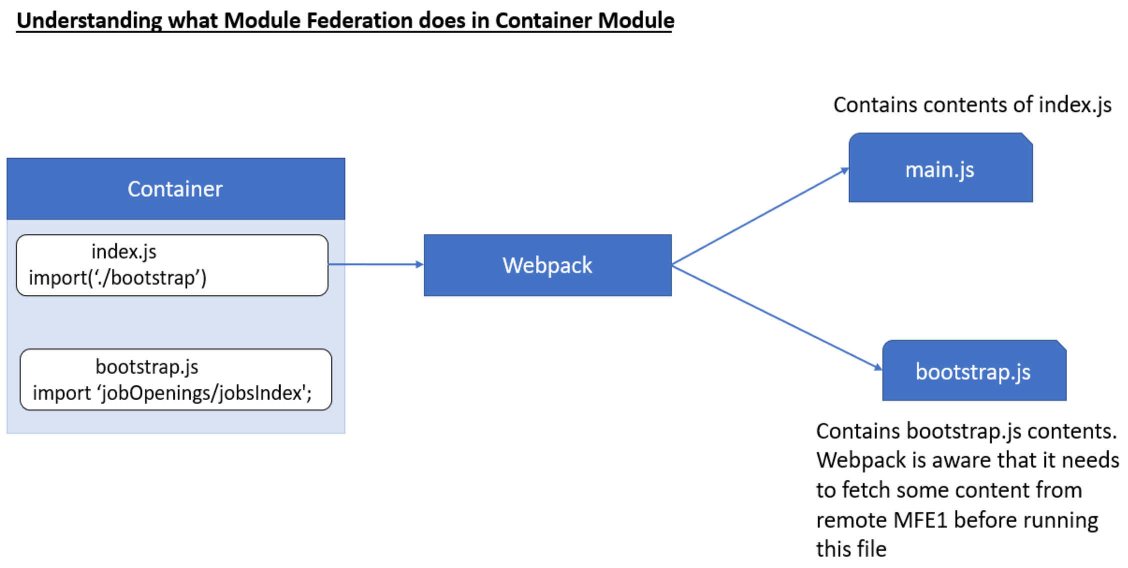
Understanding what Module Federation does in Remote Module


CompanyJobOpenings
index.js
Webpack
Normal bundling process
Module Federation Plugin
main.js
Run as standalone
remoteEntry.js
Contains list of files from this app + directions on how to load them
src_index.js
Version of src index that can be safely loaded on browser
faker.js
Version of faker that can be safely loaded on browser

- The first file, remote entry js, is a JavaScript file containing a manifest.

  It lists all the other files emitted by the module federation plugin and contains the directions on how to use them.
  The remote entry JS file can be considered a set of directions for other projects like the container.
  This will inform the set of directions to the container/host application on how to load up some source code from the products project.
  That means, the remote entry file can tell the container the directions on how to go and fetch the index.js if needed.
- The source index and faker files emitted here are almost similar to the original index.js of the CompanyJobOpenings app and faker. Since webpack processed it, it will take all this code and convert it so we can use modules like import statements and things like that inside the browser.
- So eventually, the version of the source index and faker files emitted by the module federation plugin of webpack, can be safely loaded and executed in the browser.

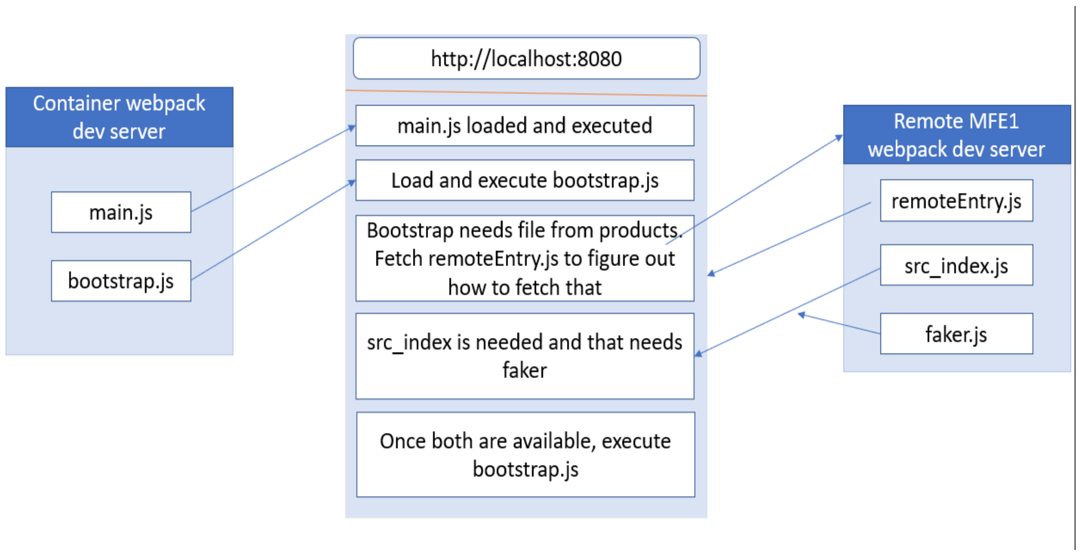
Understanding what Module Federation does in Container Module


Understanding what Module Federation does in Container Module
Container
index.js
import('./bootstrap')
bootstrap.js
import 'jobOpenings/jobsIndex';
Webpack
Contains contents of index.js
main.js
bootstrap.js
Contains bootstrap.js contents. Webpack is aware that it needs to fetch some content from remote MFE1 before running this file

When integrating the CompanyJobOpenings (Remote MFE1) with the container, the original code for importing the Remote MFE, i.e., import 'jobOpenings/jobsIndex' was moved from the index.js of the Container to another file called bootstrap.js file. After this, the bootstrap file was imported inside index.js. So why was this done?

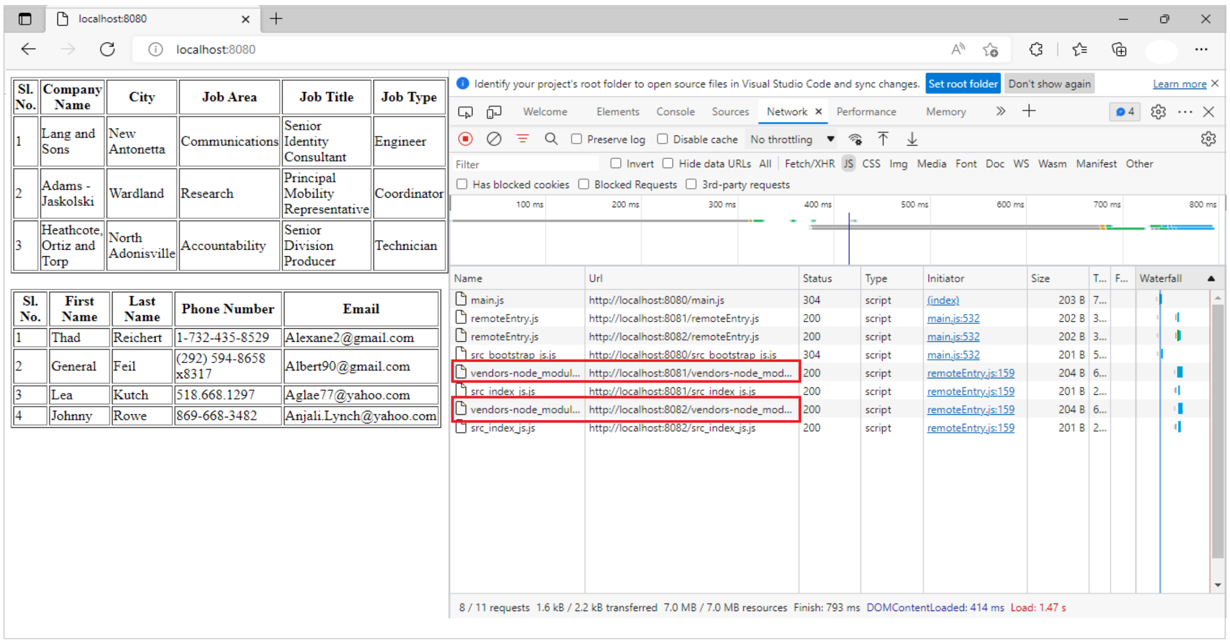
- By default, when webpack launches index.js, the expectation is that the content of the file and all the different dependencies it requires are already available for access and execution. However, executing the index.js file directly would result in an error, saying, ‘Sorry, but I don't have any code for ‘jobOpenings’. This is because it is still unclear where to get the code of ‘jobOpenings’ as it is not yet available since it has to come from a remote source.
- By moving the code of index.js to bootstrap.js and then importing the bootstrap from within the index file, Webpack will get an opportunity inside the browser to go and fetch some dependencies from jobOpenings before actually executing the code.

Hence index.js was modified this way to load the bootstrap file asynchronously. This helps ensure that webpack gets an opportunity to look for some additional JavaScript and ensure that the jobOpenings code is ready along with the faker code before trying to do something with it when the actual contents of the bootstrap js file get executed.


Container webpack dev server
main.js
bootstrap.js
http://localhost:8080
main.js loaded and executed
Load and execute bootstrap.js
Bootstrap needs file from products. Fetch remoteEntry.js to figure out how to fetch that
src_index is needed and that needs faker
Once both are available, execute bootstrap.js
Remote MFE1 webpack dev server
remoteEntry.js
src_index.js
faker.js

Sharing of dependencies

Observe the network tab of the container application running on the 8080 port.


localhost:8080

| Sl. No. | Company Name | City | Job Area | Job Title | Job Type |
|---|---|---|---|---|---|
| 1 | Lang and Sons | New Antonetta | Communications | Senior Identity Consultant | Engineer |
| 2 | Adams - Jaskolski | Wardland | Research | Principal Mobility Representative | Coordinator |
| 3 | Heathcote, Ortiz and Torp | North Adonisville | Accountability | Senior Division Producer | Technician |

| Sl. No. | First Name | Last Name | Phone Number | Email |
|---|---|---|---|---|
| 1 | Thad | Reichert | 1-732-435-8529 | Alexane2@gmail.com |
| 2 | General | Feil | (292) 594-8658 x8317 | Albert90@gmail.com |
| 3 | Lea | Kutch | 518.668.1297 | Aglae77@yahoo.com |
| 4 | Johnny | Rowe | 869-668-3482 | Anjali.Lynch@yahoo.com |

Identify your project's root folder to open source files in Visual Studio Code and sync changes. Set root folder Don't show again Learn more

| Name | Url | Status | Type | Initiator | Size | T... | F... |
|---|---|---|---|---|---|---|---|
| main.js | http://localhost:8080/main.js | 304 | script | (index) | 203 B | 7... | |
| remoteEntry.js | http://localhost:8081/remoteEntry.js | 200 | script | main.js:532 | 202 B | 3... | |
| remoteEntry.js | http://localhost:8082/remoteEntry.js | 200 | script | main.js:532 | 202 B | 3... | |
| src_bootstrap_js.js | http://localhost:8080/src_bootstrap_js.js | 304 | script | main.js:532 | 201 B | 5... | |
| vendors-node_modul... | http://localhost:8081/vendors-node_mod... | 200 | script | remoteEntry.js:159 | 204 B | 6... | |
| src_index_js.js | http://localhost:8081/src_index_js.js | 200 | script | remoteEntry.js:159 | 201 B | 2... | |
| vendors-node_modul... | http://localhost:8082/vendors-node_mod... | 200 | script | remoteEntry.js:159 | 204 B | 6... | |
| src_index_js.js | http://localhost:8082/src_index_js.js | 200 | script | remoteEntry.js:159 | 201 B | 2... | |

8 / 11 requests 1.6 kB / 2.2 kB transferred 7.0 MB / 7.0 MB resources Finish: 793 ms DOMContentLoaded: 414 ms Load: 1.47 s

Notice that the faker is loaded twice, once for the MFE1 running at 8081 and once for the MFE2 running at 8082. The size of the library is already significant, and loading it twice would create unnecessary overhead.

The significant advantage of Module federation is sharing of common dependencies.

With the Module Federation plugin, this can also be very quick. The container fetches both remotes and notices the shared dependency. It then loads one copy of the shared dependency and makes the exact copy available to both remotes.

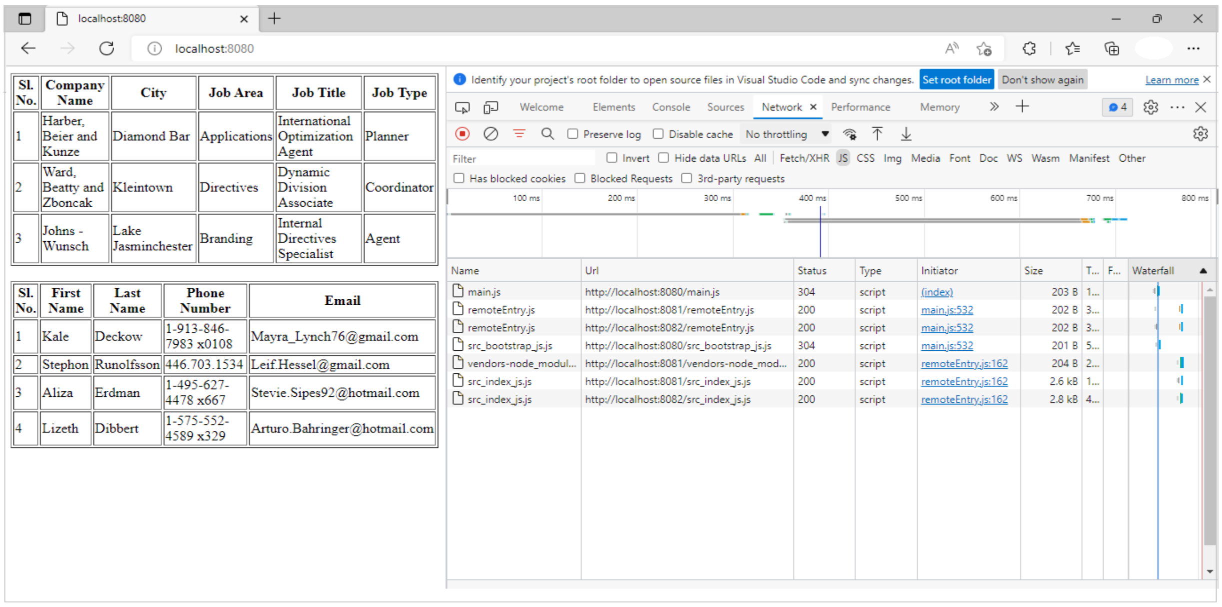
Module federation assists in this decision by enabling the shared option in the Module federation configuration object.

webpack.config.js of MFE1:

```
plugins:[
    new ModuleFederationPlugin({
      name:'jobOpenings',
      filename: 'remoteEntry.js',
      exposes: {
        './JobsIndex':'./src/index'
      },
         shared:['faker']
    }),
]
```

webpack.config.js of MFE2:

```
plugins:[
    new ModuleFederationPlugin({
      name:'employees',
      filename: 'remoteEntry.js',
      exposes: {
        './employees':'./src/index'
      },
         shared:['faker']
    }),
]
```



Identify your project's root folder to open source files in Visual Studio Code and sync changes.

| Sl. No. | Company Name | City | Job Area | Job Title | Job Type |
|---|---|---|---|---|---|
| 1 | Harber, Beier and Kunze | Diamond Bar | Applications | International Optimization Agent | Planner |
| 2 | Ward, Beatty and Zboncak | Kleintown | Directives | Dynamic Division Associate | Coordinator |
| 3 | Johns - Wunsch | Lake Jasminchester | Branding | Internal Directives Specialist | Agent |

| Sl. No. | First Name | Last Name | Phone Number | Email |
|---|---|---|---|---|
| 1 | Kale | Deckow | 1-913-846-7983 x0108 | Mayra_Lynch76@gmail.com |
| 2 | Stephon | Runolfsson | 446.703.1534 | Leif.Hessel@gmail.com |
| 3 | Aliza | Erdman | 1-495-627-4478 x667 | Stevie.Sipes92@hotmail.com |
| 4 | Lizeth | Dibbert | 1-575-552-4589 x329 | Arturo.Bahringer@hotmail.com |

| Name | Url | Status | Type | Initiator | Size | T... | F... | Waterfall |
|---|---|---|---|---|---|---|---|---|
| main.js | http://localhost:8080/main.js | 304 | script | (index) | 203 B | 1... | | |
| remoteEntry.js | http://localhost:8081/remoteEntry.js | 200 | script | main.js:532 | 202 B | 3... | | |
| remoteEntry.js | http://localhost:8082/remoteEntry.js | 200 | script | main.js:532 | 202 B | 3... | | |
| src_bootstrap_js.js | http://localhost:8080/src_bootstrap_js.js | 304 | script | main.js:532 | 201 B | 5... | | |
| vendors-node_modul... | http://localhost:8081/vendors-node_mod... | 200 | script | remoteEntry.js:162 | 204 B | 2... | | |
| src_index_js.js | http://localhost:8081/src_index_js.js | 200 | script | remoteEntry.js:162 | 2.6 kB | 1... | | |
| src_index_js.js | http://localhost:8082/src_index_js.js | 200 | script | remoteEntry.js:162 | 2.8 kB | 4... | | |

Version management for dependencies shared using module federation.

Fallback to different shared library versions are not always a perfect solution. With multiple versions in place, unexpected effects may happen when the application works in integration. This can happen mainly when different versions of UI frameworks and libraries are used in different MFEs integrated into the container.

To deal with such situations, Module Federation allows defining libraries as Singletons.

Singletons

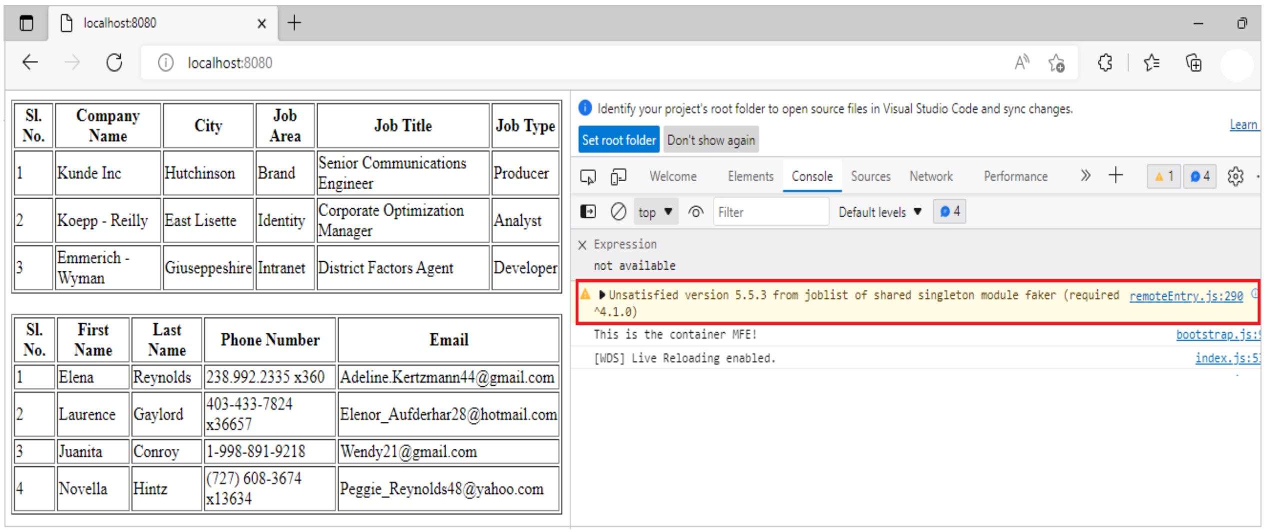
Modules configured as Singletons can be loaded only once.
If compatible versions are already available as per semantic versioning, then Module Federation will decide on the highest compatible one as usual. However, if a version mismatch exists, singletons will prevent Module Federation from falling back to the other library version.

Configuring singletons

Consider MFE1 CompanyJobOpenings and MFE2 EmployeesForRecruitment have the below configurations:

| CompanyJobOpenings MFE1 | EmployeesForRecruitment MFE2 |
|---|---|
| **package.json:** | **package.json:** |
| "dependencies": { | "dependencies": { |
| "faker": "^5.1.0", | "faker": "^4.1.0", |
| .. | .. |
| } | } |
| **webpack.config.js** | **webpack.config.js:** |
| new ModuleFederationPlugin({ | new ModuleFederationPlugin({ |
| name:'jobOpenings', | name: 'employees', |
| filename: 'remoteEntry.js', | filename: 'remoteEntry.js', |
| exposes: { | exposes: { |
| './jobsIndex':'./src/index' | './employeesIndex': './src/index', |
| }, | }, |
| shared:{ | shared:{ |
| 'faker': {singleton:true} | 'faker': {singleton:true} |
| } | } |
| }), | }), |

In order to prevent loading of several versions of the singleton package, Module Federation always decides to load the highest available version of the shared library which it is aware of during the initialization phase. In this case, version 5.



localhost:8080

| Sl. No. | Company Name | City | Job Area | Job Title | Job Type |
|---|---|---|---|---|---|
| 1 | Kunde Inc | Hutchinson | Brand | Senior Communications Engineer | Producer |
| 2 | Koepp - Reilly | East Lisette | Identity | Corporate Optimization Manager | Analyst |
| 3 | Emmerich - Wyman | Giuseppeshire | Intranet | District Factors Agent | Developer |

| Sl. No. | First Name | Last Name | Phone Number | Email |
|---|---|---|---|---|
| 1 | Elena | Reynolds | 238.992.2335 x360 | Adeline.Kertzmann44@gmail.com |
| 2 | Laurence | Gaylord | 403-433-7824 x36657 | Elenor_Aufderhar28@hotmail.com |
| 3 | Juanita | Conroy | 1-998-891-9218 | Wendy21@gmail.com |
| 4 | Novella | Hintz | (727) 608-3674 x13634 | Peggie_Reynolds48@yahoo.com |

Identify your project's root folder to open source files in Visual Studio Code and sync changes.

```
Expression
not available
⚠ ▶Unsatisfied version 5.5.3 from joblist of shared singleton module faker (required    remoteEntry.js:290
^4.1.0)
This is the container MFE!                                                               bootstrap.js:
[WDS] Live Reloading enabled.                                                            index.js:5
```

Version 5 is not compatible with version 4 as per semantic versioning. Hence a warning is shown on the browser. The federated application may still continue to work even with this mismatch.

With singleton configured, only a warning detailing version mismatch is shown, and it's assumed the apps would work. However, it is also possible that with the new version 5, some breaking changes may be introduced because the application might fail to work when accessed. In these situations, it is always advisable to fall faster when detecting a mismatch, possibly showing an error immediately on the initial launch, rather than getting an error when the respective MFE is accessed.

For enabling this, Module Federation has a setting for strictVersion for singletons, which can be set to true.

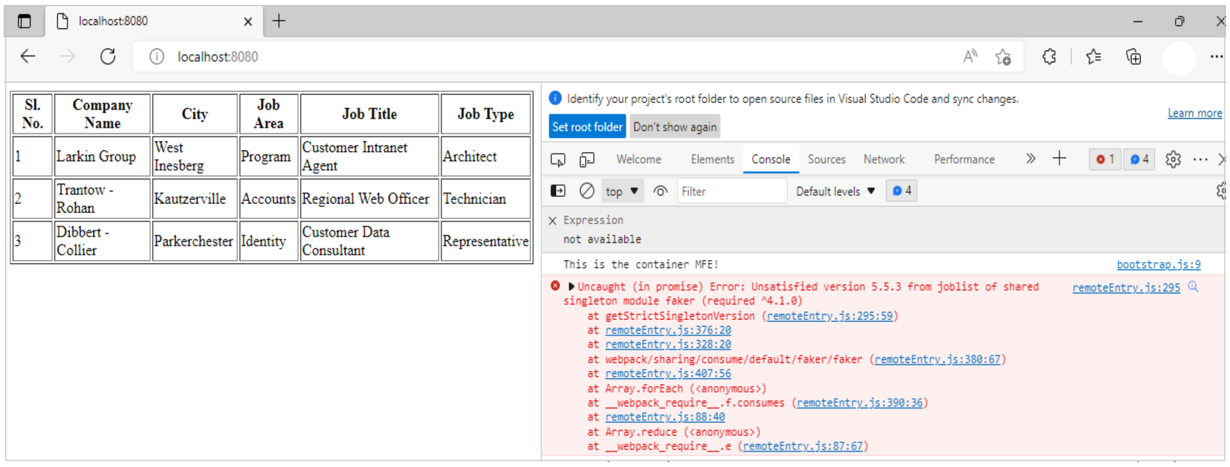
Modify the webpack config files of MFEs to include strictVersion to true.

**MFE1**

```
webpack.config.js
 new ModuleFederationPlugin({
         name:'jobOpenings',
         filename: 'remoteEntry.js',
         exposes: {
            './jobsIndex':'./src/index'
         },
         shared:{
            'faker':
{singleton:true,strictVersion:true}
            }
      }),
```

**MFE2**

```
webpack.config.js:
new ModuleFederationPlugin({
     name: 'employees',
     filename: 'remoteEntry.js',
     exposes: {
       './employeesIndex': './src/index',
     },
          shared:{
            'faker': {singleton:true,
strictVersion:true}
          }
   }),
```



| Sl. No. | Company Name | City | Job Area | Job Title | Job Type |
|---|---|---|---|---|---|
| 1 | Larkin Group | West Inesberg | Program | Customer Intranet Agent | Architect |
| 2 | Trantow - Rohan | Kautzerville | Accounts | Regional Web Officer | Technician |
| 3 | Dibbert - Collier | Parkerchester | Identity | Customer Data Consultant | Representative |

```
This is the container MFE!                                              bootstrap.js:9
Uncaught (in promise) Error: Unsatisfied version 5.5.3 from joblist of shared    remoteEntry.js:295
singleton module faker (required ^4.1.0)
    at getStrictSingletonVersion (remoteEntry.js:295:59)
    at remoteEntry.js:376:20
    at remoteEntry.js:328:20
    at webpack/sharing/consume/default/faker/faker (remoteEntry.js:380:67)
    at remoteEntry.js:407:56
    at Array.forEach (<anonymous>)
    at __webpack_require__.f.consumes (remoteEntry.js:390:36)
    at remoteEntry.js:88:40
    at Array.reduce (<anonymous>)
    at __webpack_require__.e (remoteEntry.js:87:67)
```

For situations where a higher version is backward compatible even though it doesn’t need to be concerning semantic versioning. Here, the Module Federation can be made to accept a defined version range, by configuring requiredVersion for the shared library.

Modify the webpack config files of MFE2 to include strictVersion and requiredVerion.

**MFE2**

```
webpack.config.js:
new ModuleFederationPlugin({
     name: 'employees',
     filename: 'remoteEntry.js',
     exposes: {
       './employeesIndex': './src/index',
     },
           shared:{
             'faker': {singleton:true, strictVersion:true, requiredVersion: ">=4.1.0
<5.6.0"}
           }
   })
```

## 2.8 Micro Frontends Using Webpack's Module Federation with React

When Module Federation needs to be added, the configuration needs to be done in webpack.config.js of the application.
Most of the times, create-react-app (CRA) is used for creating React applications where webpack.config.js is not available straight away due to the abstraction provided by CRA. Surely, we may not wish to eject the create-react-app configuration.
In this scenario, there are two options in which can be followed to create a React application where Module Federation will be used:

- Pull up a React application structure manually:

  Create a basic Node application
  Installing requried dependencies (babel, webpack, htmlWebPackPlugin, React)
  Add React files inside folders (public -> index.html, src-> App.js)
  Configure webpack and babel
  Add scripts in package.json
  Now start your application using npm start.

- Another alternative is to use CRACO.

  Craco is the Create React App Configuration Override, an easy and comprehensible configuration layer. It can also enable usage module federation in CRA application.

We will follow option1 – i.e., pull up a React application structure manually and add required configurations.

1. Create a new folder named micro-frontend-1, open it in VS Code terminal and run the following command.

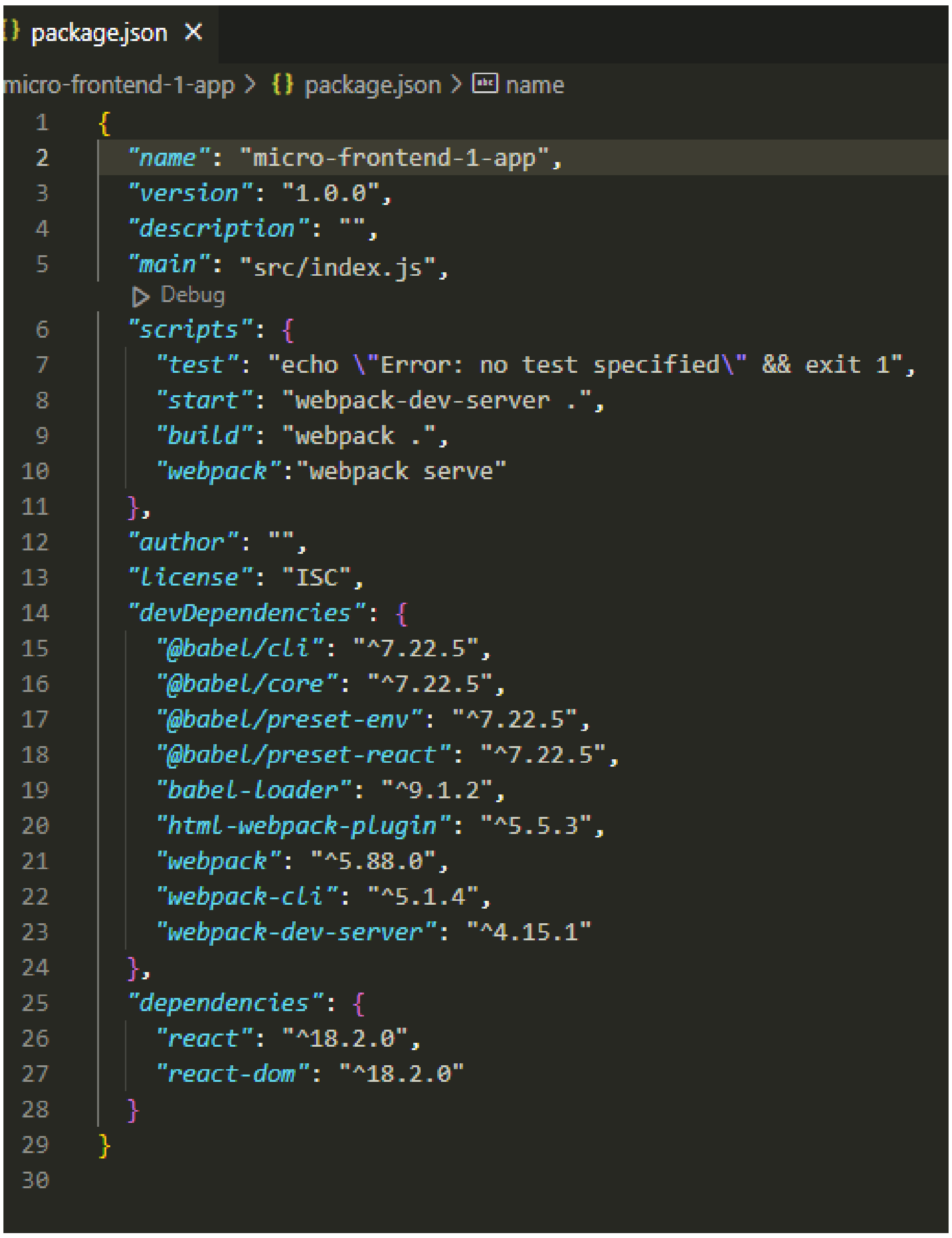
```
npm init
```

2. Install the required dependencies : Babel, Webpack, HtmlWebpackPlugin and React dependencies.

Use the below commands one by one.

```
npm install --save-dev @babel/core babel-loader @babel/cli @babel/preset-env @babel/preset-react
npm install --save-dev @babel/core babel-loader @babel/cli @babel/preset-env @babel/preset-react
npm install --save-dev html-webpack-plugin
npm install react react-dom
```

3. After installing all the dependencies, the package.json will appear as shown below



```
{
  "name": "micro-frontend-1-app",
  "version": "1.0.0",
  "description": "",
  "main": "src/index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "webpack-dev-server .",
    "build": "webpack .",
    "webpack":"webpack serve"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "@babel/cli": "^7.22.5",
    "@babel/core": "^7.22.5",
    "@babel/preset-env": "^7.22.5",
    "@babel/preset-react": "^7.22.5",
    "babel-loader": "^9.1.2",
    "html-webpack-plugin": "^5.5.3",
    "webpack": "^5.88.0",
    "webpack-cli": "^5.1.4",
    "webpack-dev-server": "^4.15.1"
  },
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0"
  }
}
```

4.Create two folders – public, src. Create index.html inside public folder, App.js and index.js inside src.

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>React App1</title>
</head>
<body>
   <div id="root"></div>
</body>
</html>
```

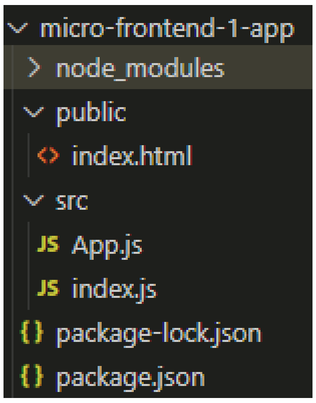
App.js:

```
import React from "react";
const App = () =>{
   return (
      <h1>
         Hello world! I am using React
      </h1>
   )
}
export default App;
```

index.js

```
import React from 'react'
import  { createRoot }  from 'react-dom/client';
import App from './App.js'
const container = document.getElementById('root');
const root = createRoot(container);
root.render(<App/>);
```

Now the folder structure of the application will be:



```
micro-frontend-1-app
  node_modules
  public
    index.html
  src
    App.js
    index.js
  package-lock.json
  package.json
```

5. Configure Babel

Create a file named .babelrc in the root folder and add the following code:

```
{
   "presets": ["@babel/preset-env","@babel/preset-react"]
}
```

6. Configure Webpack

Create a file named webpack.config.js and add the following code:

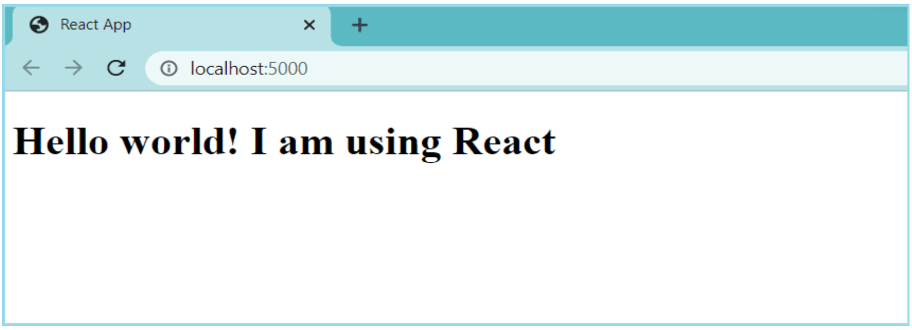
```
const HtmlWebpackPlugin = require('html-webpack-plugin');
const path = require('path');
module.exports = {
  entry: './src/index.js',
  mode: 'development',
  output: {
    path: path.resolve(__dirname, './dist'),
    filename: 'index_bundle.js',
  },
  target: 'web',
  devServer: {
    port: '5000',
    static: {
      directory: path.join(__dirname, 'public')
},
    open: true,
    hot: true,
    liveReload: true,
  },
  resolve: {
    extensions: ['.js', '.jsx', '.json'],
  },
  module: {
    rules: [
      {
        test: /\.(js|jsx)$/,
        exclude: /node_modules/,
        use: 'babel-loader',
      },
    ],
  },
  plugins: [
    new HtmlWebpackPlugin({
      template: path.join(__dirname, 'public', 'index.html')
    })
  ]
};
```

7. Add scripts to package.json:

```
"scripts": {
    "start": "webpack-dev-server .",
    "build": "webpack .",
"webpack":"webpack serve"
  }
```

8. Start your application

Run npm start to start the application



Hello world! I am using React

Converting the React application to a React Micro Frontend (Remote) with Module Federation

To convert the created React application to a Micro Frontend Remote with Module Federation, required configurations need to be added in webapck:

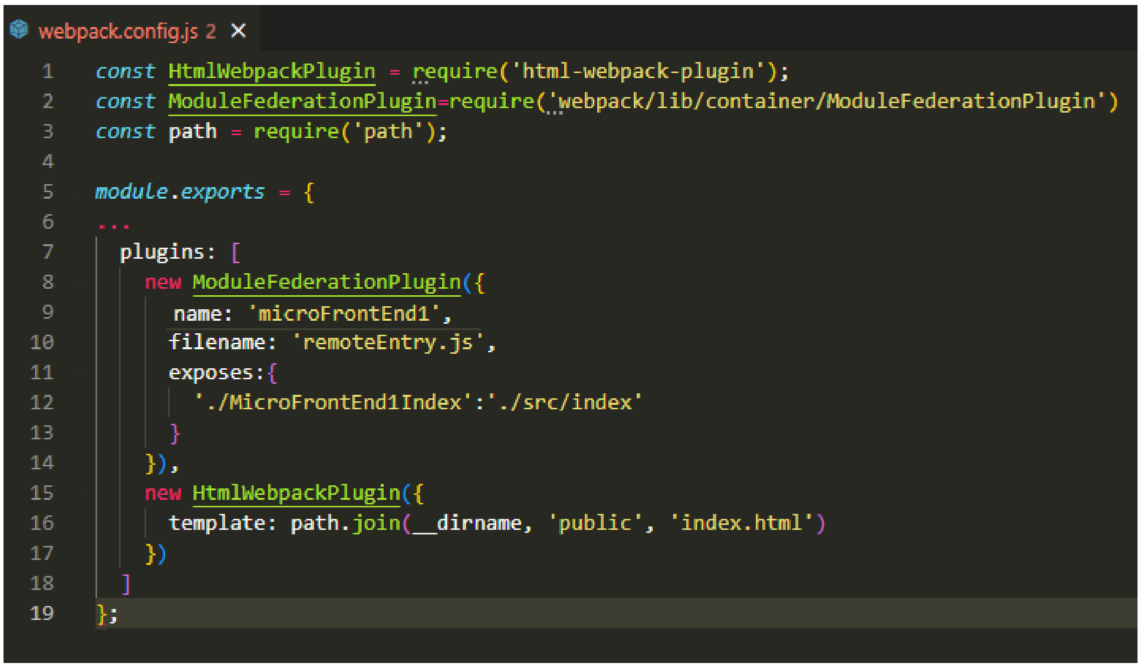
Create a file named bootstrap.js inside src and add the below code:

```
import React from 'react';
import  { createRoot }  from 'react-dom/client';
import App from './App';
const root = createRoot(document.getElementById('react-microfrontend-1'));
root.render(
   <App />);
```

Modify the content of index.js to below code:

```
import('./bootstrap');
```

Add Module Federation Plugin. In the webpack-config.js file:



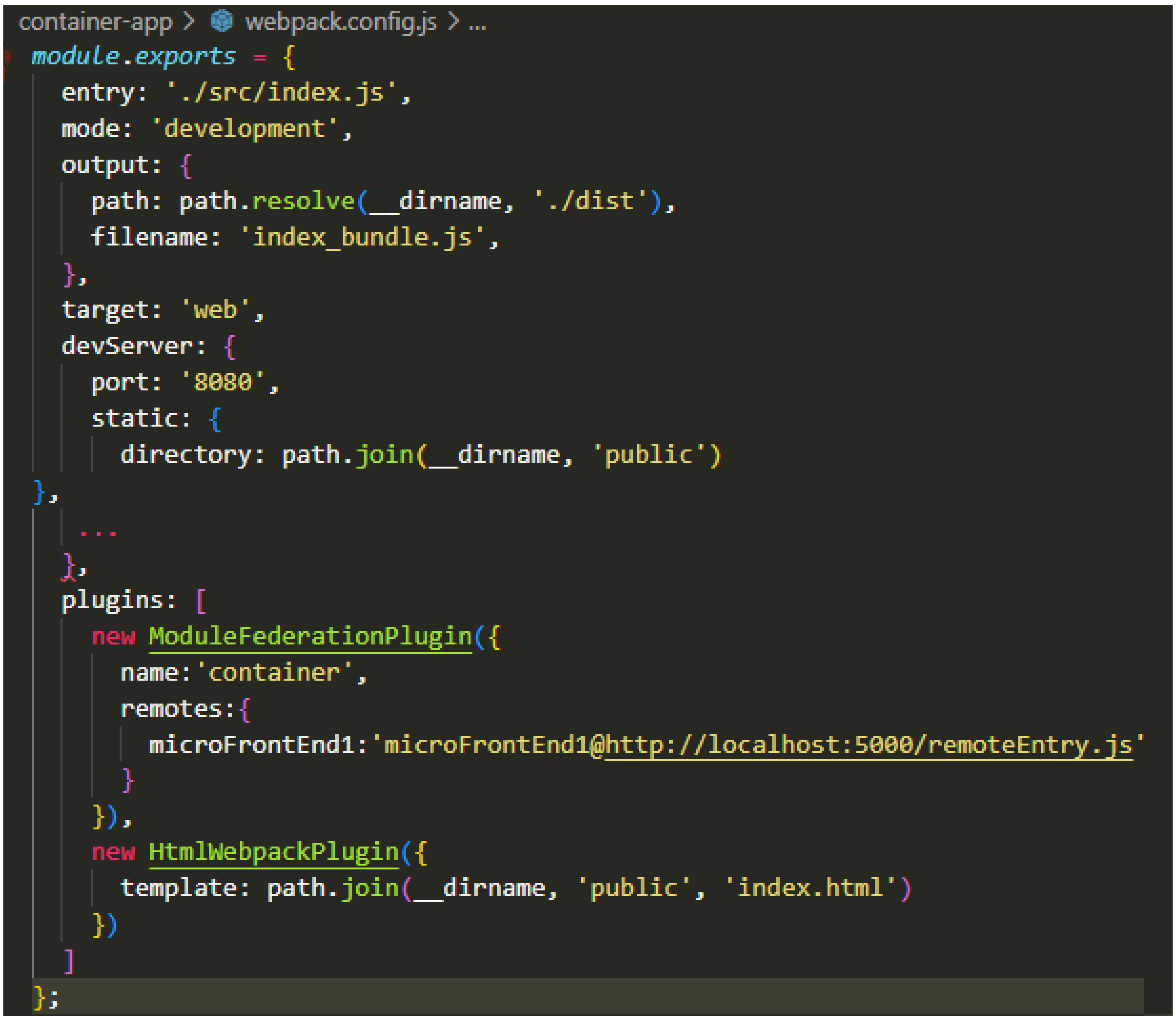
```
const HtmlWebpackPlugin = require('html-webpack-plugin');
const ModuleFederationPlugin=require('webpack/lib/container/ModuleFederationPlugin')
const path = require('path');

module.exports = {
...
  plugins: [
    new ModuleFederationPlugin({
      name: 'microFrontEnd1',
     filename: 'remoteEntry.js',
     exposes:{
       './MicroFrontEnd1Index':'./src/index'
     }
    }),
    new HtmlWebpackPlugin({
     template: path.join(__dirname, 'public', 'index.html')
    })
  ]
};
```

Creating a host Micro Frontend to consume the React remote, using Module Federation

Create a container React application (without using CRA). Follow the steps mentioned earlier, for this.

Add Module Federation Plugin. In the webpack-config.js file of the host container:



```
module.exports = {
  entry: './src/index.js',
  mode: 'development',
  output: {
    path: path.resolve(__dirname, './dist'),
    filename: 'index_bundle.js',
  },
  target: 'web',
  devServer: {
    port: '8080',
    static: {
      directory: path.join(__dirname, 'public')
},
   ...
  },
  plugins: [
    new ModuleFederationPlugin({
      name:'container',
      remotes:{
        microFrontEnd1:'microFrontEnd1@http://localhost:5000/remoteEntry.js'
      }
    }),
    new HtmlWebpackPlugin({
      template: path.join(__dirname, 'public', 'index.html')
    })
  ]
};
```

Create a file named bootstrap.js inside src and add the below code:

```
import 'microFrontEnd1/MicroFrontEnd1Index';
console.log("container");
const root = createRoot(document.getElementById('react-microfrontend-1'));
```

Modify App.js:

```
import React from "react";
const App = () =>{
   return (
      <h1>
         Hello world! I am using React
      </h1>
   )
}
export default App
```

Modify the content of index.js to below code:

```
        import('./bootstrap');
```

Modify index.html of container application to below code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>React App</title>
</head>
<body>
  <div id="root"></div>
  <div id="react-microfrontend-2"></div>
</body>
</html>
```

Add Module Federation Plugin. In the webpack-config.js file:

```
const HtmlWebpackPlugin = require('html-webpack-plugin');
const ModuleFederationPlugin=require('webpack/lib/container/ModuleFederationPlugin');
const path = require('path');
module.exports = {
  entry: './src/index.js',
  mode: 'development',
  output: {
    path: path.resolve(__dirname, './dist'),
    filename: 'index_bundle.js',
  },
  target: 'web',
  devServer: {
    port: '8080',
    static: {
      directory: path.join(__dirname, 'public')
},
    open: true,
    hot: true,
    liveReload: true,
  },
  resolve: {
    extensions: ['.js', '.jsx', '.json'],
  },
  module: {
    rules: [
      {
       test: /\.(js|jsx)$/,
       exclude: /node_modules/,
       use: 'babel-loader',
      },
    ],
  },
  plugins: [
    new ModuleFederationPlugin({
      name:'container',
      remotes:{
        microFrontEnd1:'microFrontEnd1@http://localhost:5000/remoteEntry.js'
      }
    }),
    new HtmlWebpackPlugin({
      template: path.join(__dirname, 'public', 'index.html')
    })
  ]
};
```

This will import the code from remote MFE and populate the div in container’s index.html. The container is now configured.

## 2.9 CSS in Micro Frontends

CSS is needed in Micro Frontends as it guarantees consistent design, appropriate layout thereby guaranteeing seamless user experience across the application.

In a micro frontend architecture, CSS may be available as separate modules distributed among various MFEs. Even though every Micro Frontend might have its own CSS files, a consistent and integrated design system is a must to maintain a consistent user interface

CSS in Micro Frontends can be approached in several ways:

- Shared CSS Files: Here, a central CSS file containing styling related code for all MFEs can be created and this file can be shared and used by all the MFEs so that all have uniform appearance and feel. This however, may introduce connectivity across the various MFEs thus resulting in some sort of coupling which inturn can impact the autonomous creation and deployment of independent MFEs. This in turn defies the concept of Micro Frontend.
- Scoped CSS: Scoping CSS within each Micro Frontend module is another solution. Here, care is taken to scope the CSS styles within certain component or module border so as to prevent styles from leaking or interfering with other modules.
  Scoped CSS can be achieved using CSS-in-JS libraries. With this method modularity will continue to be supported and it also lowers the likelihood of style clashes.
- CSS Libraries: Creating a design system which is a collection of reusable components, patterns, and rules that can help in maintaining uniformity across application is yet another option. With a shared design system or component library in Micro Frontends, a consistent UI is easy to achieve.
- For example: Using standard libraries like Bootstrap or Material UI across all MFEs allows for consistent styling and promotes code standardization and reusability.

### 2.9.1 How to overcome CSS conflicts between MFEs?

**CSS Modules**

CSS Modules are the CSS files that include class names or animation names that are by default within local scope. CSS Modules compile to ICSS, or "Interoperable CSS," a low-level interchange format which is used as JS objects for further processing and security. All the imports (@imports) and URLs (url(...)) used in CSS are in module request format. For example,

- "/<filename.ext>" and "../filename.ext" means relative,
- "filename" and "<foldername>/<filename>" means in node_modules

While importing the CSS Module in a JS Module, it exports an object that contains all the class and animation names from local names to global names. CSS Modules are popular as they automatically generate unique class and animation names, eliminating the need to worry about selector name conflicts.

Advantages

- Class name collisions may be prevented by using CSS modules since when they are loaded on the browser, they generate random class names.
- They help in organizing and maintaining a clean and easy CSS files.
- As the scope of CSS classes remain local, numerous unintentional side effects will be avoided.

Disadvantages

- Combining CSS Modules with global CSS classes is inconvenient.

**CSS in JS**

In the situation that we have any media queries, keyframes, or pseudo classes in general, inline styles will be unreliable.
CSS-in-JS would be our rescue in such cases where, we write our CSS within our JavaScript in a slightly different way, allowing our styling to be injected dynamically.
There are various libraries available that will help us with the process of adding CSS-in-JS. The most popular libraries in this field are Styled-Components, Radium, Aphrodite, JSS, and Styletron, however we will focus on Styled-Components because it is widely used by developers.
Let us now understand CSS-in-JS in detail by implementing it for our previous demo.
Let us add styled-components library to our project.

Advantages

In a React component, <BoxStyle> is more semantically pleasant than <div className={style.boxStyle}>
As larger projects include conditional rendering, maintenance becomes easier since styles are contained within the components.
CSS-in-JS can dynamically clean up CSS code which are no longer being used.
Disadvantages

In most of the projects, CSS files are being cached. As our styling is included within our JS, separate caching of CSS cannot be done.
Styled components parsing, adding styles to the DOM and browser interpretation of styles require time, which might cause slower rendering.
We can't use SCSS, Less, Postcss, stylelint in our CSS in JS.
Slightly time consuming for newer team mates to adapt to the codebase.

Advantages

- In a React component, <BoxStyle> is more semantically pleasant than <div className={style.boxStyle}>
- As larger projects include conditional rendering, maintenance becomes easier since styles are contained within the components.
- CSS-in-JS can dynamically clean up CSS code which are no longer being used.

Disadvantages

- In most of the projects, CSS files are being cached. As our styling is included within our JS, separate caching of CSS cannot be done.
- Styled components parsing, adding styles to the DOM and browser interpretation of styles require time, which might cause slower rendering.
- We can't use SCSS, Less, Postcss, stylelint in our CSS in JS.
- Slightly time consuming for newer team mates to adapt to the codebase.

### 2.9.2 CSS Best Practises

- Create a Shared Design System or Component Library: Following a common shared library or following the same component library can provide uniformity and makes CSS integration easier among Micro Frontend.
- Use Scoped CSS: To isolate styles within module boundaries, its best to make use scoped CSS techniques such as CSS modules or CSS-in-JS libraries so as to minimize clashes.
- Communicate and Collaborate: Make sure to promote consistent communication and collaboration among working teams of various Micro Frontends. This can help in aligning CSS styles, resolving conflicts, and keeping a consistent user interface.
- Optimize Performance: Make sure to use optimization approaches like code splitting, CSS minification, and bundling so as to ensure that CSS file delivery is done fastly thereby improving overall application performance.