MICRO FRONTEND DEVELOPER GUIDE

CAN 7.0

JULY 28, 2023

AVANSEUS TECHNOLOGY PVT. LTD.

REVISION HISTORY

Version	Date	Change Description	Created by	Updated by	Reviewed by
V 1.0	July 2023	Beta Release	Sarikya / Hemanth	Raksha	Chiranjib

Table of Contents

Introduction

MicroFrontend is a type of architecture in which a single monolith web application is divided into different micro-frontend modules. Wherein, each module can use its own technology stack (for example, one remote module can have Java with React, another module can have Python with Angular and so on) and later can be combined with main host module using **webpack** integrated with a plugin called **ModuleFederationPlugin**.

Advantages of Microfrontend Architecture

The advantages of using Micro Frontend architecture are as follows:

- To integrate multiple modules with different code bases. For example, one remote module can have java with react and another module can have python with angular and so on.
- Multiple teams can work on different remote modules with different technology stack that is integrated with the host module.
- Development and deployment is faster when multiple teams work simultaneously on different modules.
- Maintainability becomes easier as they follow divide and conquer approach unlike the monolithic applications.
- It helps in continuous deployment and testing of individual modules.

Procedure to Create Remote Module in CAN

This section is divided into three stages:

1. Add Functionalities from PES.
2. Creating react project and integrating with Webpack.
3. Prerequisites for integration with host module.

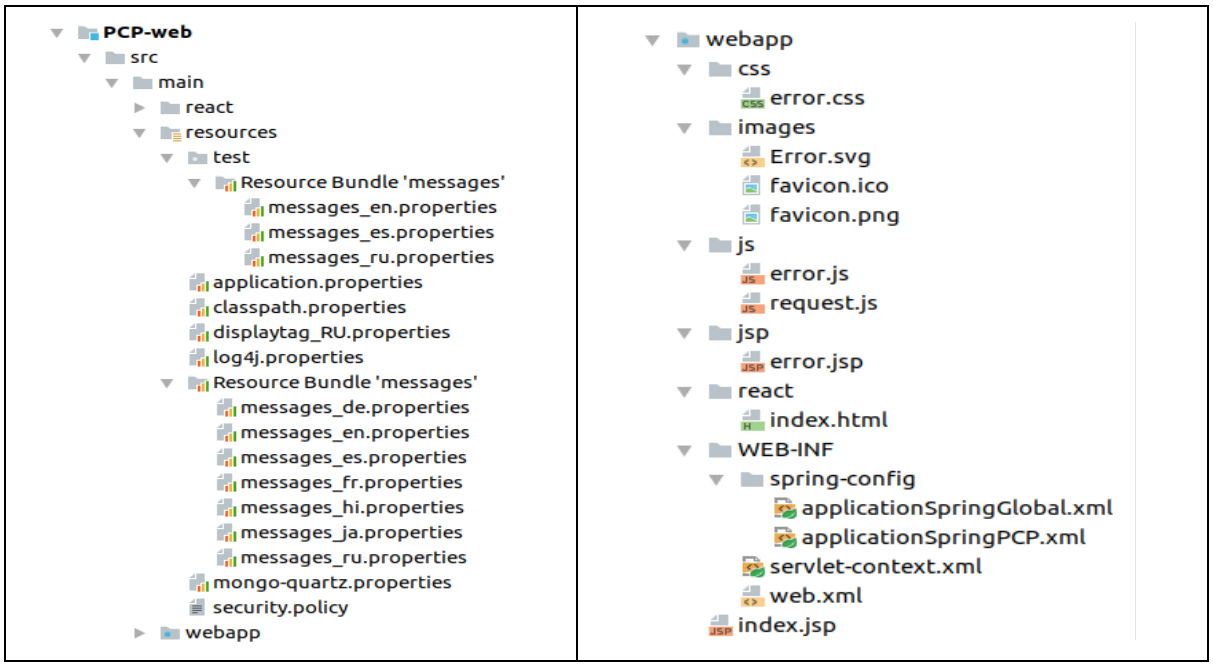
Stage-1: Add Functionalities from PES

The remote module which we are about to create will not have any dependencies on the PES-web module which earlier used to provide with some generic functionalities like Internationalization, Filters, Servlet context, and many more. Hence, we selectively choose some of the required functionalities that needs to be incorporated with the remote module in this section. Follow the steps in the given sequence to integrate the minimum required functionalities.

1. Create a maven project with “Core” and “Web” as the submodules under the remote module and add the required dependencies in their respective pom.xml file.
2. The backend code (usually Java) should be placed under the “Core” submodule and the front-end code (usually React) should be placed under the “Web” submodule.
3. Create a directory named “resources” in the remote module (<remote-web>/src/main/resources). Add the necessary resources inside it like classpath.properties, application.properties, log4j.properties, mongo-quartz.properties, messages_en.properties, etc.
4. Create a directory named “webapp” in the remote module (<remote-web>/src/main/webapp). Add the following files to your remote module’s webapps directory by copying them from PES module.
    a. react/index.html
    b. js/request.js: remove the addition of flowExecutionKey in the form headers.
    c. js/error.js.
    d. images/Error.svg.
    e. images/favicon.png and images/favicon.ico
    f. WEB-INF/servlet-context.xml
    g. WEB-INF/web.xml (add only needed configurations here)
    h. WEB-INF/spring-config/applicationSpring<remote>.xml (add the necessary beans here) and WEB-INF/spring-config/applicationSpringGlobal.xml
5. Create a file named “index.jsp” in your remote module’s webapps directory (<remote-web>/src/main/webapps) and paste the contents from index.jsp.

6. Create a file named “error.jsp” in <remote-web>/src/main/webapp/jsp/ directory and paste the contents from error.jsp .

NOTE: If the above mentioned 6 steps of stage-1 is followed correctly, then your remote module’s project structure of web project and its webapp directory contents will look like the below project structure:



Stage-2: Creating React Project and Integrating with Webpack

1. Create a react application named “react-app” in <remote-web>/src/main. The command to create the react application is:

```
npx create-react-app react-app
```

2. A folder named “react-app” with the react application files is created in <remote-web>/src/main/. Rename this folder to “react”.
3. Create 2 webpack configuration files in <remote-web>/src/main/react:
   a. One for production needs- **webpack.prod.js** (used for taking the build for production).
   b. Another for development needs- **webpack.dev.js** (used for taking the build for development purposes).

   NOTE: The contents of these two files along with the explanation can be found in a section named “Webpack Integration” in the later part of this document.

4. In **package.json** file of the remote module’s react project, add the below mentioned configurations in dev-dependencies and scripts section respectively.

Dev-dependencies:

"@babel/core": **"^7.22.5"**,
"@babel/plugin-proposal-private-property-in-object": **"^7.21.11"**,

"@babel/preset-env": **"^7.22.5"**,
"@babel/preset-react": **"^7.22.5"**

```
"ajv": "^8.12.0",
"babel-loader": "^8.0.4",
"css-loader": "^6.7.3",
"eslint-webpack-plugin": "^4.0.1",
"html-webpack-plugin": "^5.5.0",
"webpack": "^5.76.3",
"webpack-cli": "^5.0.1",
"webpack-dev-server": "^4.13.1"
```

Scripts:

```
"start": "webpack serve",
"build": "webpack build --config webpack.prod.js",
"dev-build": "webpack build --config webpack.dev.js"
```

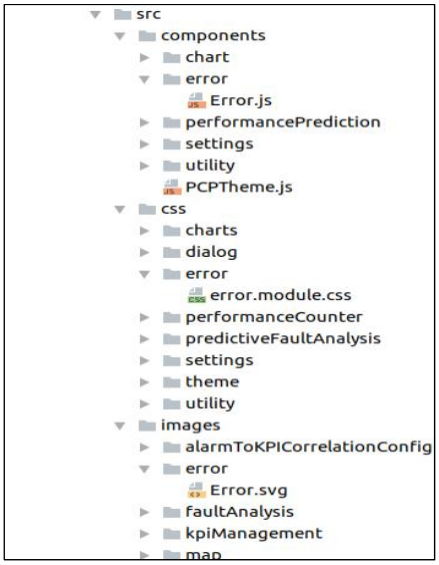
The production and development builds can be taken by executing the following commands in <remote-web>/src/main/react folder.

a. **npm run build:** It is used to take build for production which takes the webpack configuration present in *webpack.prod.js* which is configured in build using --config parameter.
b. **npm run dev-build:** It is used to take build for development which takes the webpack configuration present in *webpack.dev.js* which is configured in the dev-build using --config parameter.

5. Add the below mentioned files to *"<remote-web>/src/main/react/src"* folder by copying them from *"CAN-web/src/main/react/src"* directory. These files are required to redirect user to error page in the event of exception.
   a. components/error/Error.js
   b. css/error/error.module.css
   c. images/error/Error.svg

   In the theme.js of your remote module wrap the react components inside Error component by importing the above copied files (Please refer PCPTheme.js file). Also, ensure that remote module's theme.js should not be exposed as a component to Host module.

   Copy the above three files under the same sub-directories as shown in the below image. (Sub-directories must be created manually before pasting the required files)



6. **<Remote>-web pom.xml:** Add the content of profiles provided in the "Profiles" section provided in the later part of the document in the "<remote-web>" pom.xml file.

Stage-3: Prerequisites for Integration with Host Module

The prerequisites that the remote module must satisfy before integrating itself with the host module (CAN) are:

1. When the Host module (Example: CAN) integrated with Remote module (Example: PCP) is accessed in the browser, the below mentioned calls occur from host module to the remote module.
    - **moduleEntry.js:** The URL to load the moduleEntry.js is provided in the remotes section of ModuleFederationPlugin of the webpack files of host module.
    - **bundle.js:** The URL to load the bundle.js is provided in the output section of ModuleFederationPlugin of the webpack files of remote module.
    - Axios calls that occur from remote module.

    All the above-mentioned calls are made from host module to the remote module. A unique prefix must be attached to each of the above calls, so that all these calls can be identified as remote module calls and bypass the filters present in the host module (like DoubleSubmitFilter, XSS Filter, Validation Filter etc.) and redirect to the remote module.

    Syntax of the URL paths of the above-mentioned calls should be given as:

    - **moduleEntry.js:** <remoteModuleName>@<uniqueRemoteModulePrefix>/<remoteModuleName>ModuleEntry/react/

      Example: **PCP@PCPRemoteModule/PCPModuleEntry/react/**
    - **bundle.js:**

      <uniqueRemoteModulePrefix>/<remoteModuleName>Bundle/react/

      Example: **PCPRemoteModule/PCPBundle/react/**
    - Each axios call in remote module must be prefixed with <uniqueRemoteModuleAxiosPrefix>

      Example: **PCPModuleAxiosRequest**

    NOTE:

    1. For the moduleEntry.js and bundle.js calls to bypass the DoubleSubmitFilter, add the /<uniqueRemoteModulePrefix> in the excludedUrls of DoubleSubmitFilter in web.xml of PES-web module. Example for adding the URL is as shown below.

```
<filter>
   <filter-name>csrfFilter</filter-name>
   <filter-class>com.avanseus.security.csrf.DoubleSubmitFilter</filter-class>
   <init-param>
      <param-name>excludedUrls</param-name>
      <param-
value>/index.jsp,/react,/jsp,/dialog,/js,/css,/images,/fonts,/CanPredictedFaultsProvider,/PC
PRemoteModule,/PCPServiceProvider</param-value>
   </init-param>
</filter>
<filter-mapping>
   <filter-name>csrfFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

    The moduleEntry.js and bundle.js calls are of type “GET” request method. For these to surpass the ValidationFilter add the /<uniqueRemoteModulePrefix>/ in **httpGetAllowedRequestURL** file in CAN-web/src/main/resources.

    Example: **/PCPRemoteModule/**

2. Webpack does not support eager loading. Create a file named bootstrap.js (inside the folder <remote-web>/src/main/react/src/), move the code present in index.js (<remote-web>/src/main/react/src/) into bootstrap.js and dynamically import bootstrap.js into index.js file. Hence, the content of index.js will be as shown below.

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

3. To check the availability of remote modules functionalities, it can be accessed through the window object.

   Syntax:

```
window.[remoteModuleName]
```

   Example: window.PCP

4. The key value pairs used for internationalisation must be present in both remote and host module message properties file when integrated with the host module.

Optional Steps based on the developer needs

1. When remote module make API calls to the host module to leverage some generic functionalities (like dataload, filter configuration etc.,) that are present in Host module, it must bypass the filters provided in the host module by adding a custom URL pattern like "**/<remoteModuleName>ServiceProvider"** in the excludedUrls inside web.xml of PES-web module (PES-web/src/main/webapp/WEB-INF/).

```
<filter>
   <filter-name>csrfFilter</filter-name>
   <filter-class>com.avanseus.security.csrf.DoubleSubmitFilter</filter-class>
   <init-param>
      <param-name>excludedUrls</param-name>
      <param-
value>/index.jsp,/react,/jsp,/dialog,/js,/css,/images,/fonts,/CanPredictedFaultsProvider,/PC
PRemoteModule,/PCPServiceProvider</param-value>
   </init-param>
</filter>
<filter-mapping>
   <filter-name>csrfFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

   A security:intercept-url should also be added in applicationSpringKeycloak.xml as shown below in PES-web module (PES-web/src/main/webapp/WEB-INF/spring-config/).

```
<security:intercept-url pattern="/PCPServiceProvider/**" access="permitAll()"/>
```

2. If redux is used in the remote module, register the store in the component that is exposed to host module.
3. If the remote module needs to access "username" of the logged in person, it can be accessed in the remote module as shown below.

```
URLDecoder.decode(httpServletRequest.getHeader("userName"), "UTF-8")
```

Integration with Host Module (CAN)

To integrate the Remote module (Ex: PCP, RCA) with the Host module (CAN) follow the steps below.

1. Add the remote module's moduleEntry URL in the remotes section of ModuleFederationPlugin of webpack.dev.js and webpack.prod.js of the host module.

   Syntax:

   ```
   <Placeholder>:`<remoteModuleName>@<remoteModuleName>ModuleEntry/react/`
   ```

   Example:

   a. PCPModule: `PCP@PCPRemoteModule/PCPModuleEntry/react/`,
   b. RCAModule: `RCA@RCARemoteModule/RCAModuleEntry/react/`

2. Modify the json files that are used to render the navigation tab contents in host module (like user-header.json, administrator-header.json or adaption-menu.json). Add your remote module name in "moduleName" and set "isRemoteModule" to true where the remote module is present. If the required remote module json is not present, add new json in the respective file with the above-mentioned configurations.

   For example:

   ```
   {
     "backgroundImage": "performance-counter-icon",
     "text": "Performance Prediction",
     "url": "performance-counter",
     "isAdminModule":true,
     "isCircleManagerModule":true,
     "isZoneLeadModule":true,
     "moduleName": "PCP",
     "isRemoteModule": true
   }
   ```

3. Import the remote module in the required component of the host module. As webpack supports lazy loading, the import must be provided with the help of React.lazy.

   Syntax:

   ```
   const <moduleName> = React.lazy(() =>
   import("<Placeholder>/<exposedComponentPlaceholderFromRemoteModule>"))
   ```

   Example:

   ```
   const PerformancePrediction = React.lazy(() => import("PCPModule/PCP"));
   ```

4. In the render() function of the component where the remote module is imported, the imported remote module must be enclosed in <React.Suspense fallback={null}></React.Suspense>. <React.Suspense> supports lazy loading of the remote module component in the host module. Fallback attribute shows an alternative component until the remote module component is loaded (like a spinner etc.).

   Example:

```
<React.Suspense fallback={null}>
        {this.state.moduleComponent}
</React.Suspense>
```

5. Add the config entries, **avanseus.app.<remoteModule>.host** and **avanseus.app.<remoteModule>.domain** with host and domain values respectively in config.properties in the tomcat where the remote module is deployed.
6. In MicroServiceFilter.java of PES-core module, add a condition in doFilter() method to forward (or proxy) all the requests of your remote module from host module to remote module as shown below.

```
else if(servletPath.startsWith("/<uniqueRemoteModulePrefix>") ||
servletPath.startsWith("/<uniqueRemoteModuleAxiosPrefix>")) {
   RemoteModuleRouter remoteModuleRouter = new RemoteModuleRouter();

remoteModuleRouter.routeToRemoteModule("<remoteModuleName>","<remoteModul
eHostConfigkey>", prefix, moduleEntryFileName, httpServletRequest,
httpServletResponse);
   return;
}
```

7. Take the build of PES, host and remote module and deploy the host and remote modules in the same tomcat for local development environment.

   NOTE: Now Host and Remote module's build can be taken in two different ways.
   a. mvn clean install –P development
   b. mvn clean install –P production (or just mvn clean install)

   Refer to Profiles sections for more details.

Resources

JSPs to be copied

index.jsp

```
<?xml version="1.0" encoding="UTF-8" ?>
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1">
<%@ page import="java.util.Locale" %>
<%@ page import="com.avanseus.i18n.locale.Language" %>
<%@ page import="com.avanseus.config.Config" %>

<%
    Config conf = Config.getInstance();
    String appName = conf.getConfig("avanseus.app.name");
    Locale locale = request.getLocale();
    Language language = Language.getDisplayLanguage(locale.getDisplayLanguage());
%>
<head>
    <link rel="shortcut icon" href="images/favicon.png"/>
    <title><%=appName%></title>
    <script src="js/language.js"></script>
    <script src="js/request.js"></script>
    <script type="text/javascript">
        function getLabel(key) {
            var label = <%=language.getCode()%>[key];
            if (label == null || label == undefined) {
                label = "** Text not found **";
            }
            return label;
        }
    </script>
</head>
<%@include file="react/index.html"%>
</html>
```

error.jsp

```
<%@ page import="com.avanseus.config.Config" %>
<%@ page import="java.util.Locale" %>
<%@ page import="com.avanseus.i18n.locale.Language" %>
<%@ page import="com.avanseus.i18n.resource.bundle.ResourceBundle" %>
<%@ page import="com.avanseus.logger.ApplicationLogger" %>
<%@ page import="com.avanseus.exception.ApplicationException" %>
<%@ page import="com.avanseus.exception.code.impl.GenericErrorCode" %>

<%
   Config conf = Config.getInstance();
   String protocol = conf.getConfig("avanseus.protocol");
   String domain = conf.getConfig("avanseus.app.domain");
   String cas = conf.getConfig("avanseus.app.cas.domain");
   String appName = conf.getConfig("avanseus.app.name");
   Locale locale = request.getLocale();
   Language language = Language.getDisplayLanguage(locale.getDisplayLanguage());
   ResourceBundle resourceBundle = ResourceBundle.getInstance();
   String rootPath = protocol + "://" + domain + "/" + appName;
   Throwable throwable = (Throwable) request.getAttribute("javax.servlet.error.exception");
   Integer statusCode = (Integer) request.getAttribute("javax.servlet.error.status_code");
   String servletName = (String) request.getAttribute("javax.servlet.error.servlet_name");
   if (servletName == null) servletName = "Unknown";
   String requestUri = (String) request.getAttribute("javax.servlet.error.request_uri");
   if (requestUri == null) requestUri = "Unknown";
   ApplicationLogger logger = ApplicationLogger.getLogger(requestUri);
   ApplicationException applicationException;
   if (throwable != null) {
      applicationException = new
ApplicationException(GenericErrorCode.GEN_JSP_EXCEPTION, throwable);
      logger.error(applicationException);
   }
   boolean restrictedAccess = (request.getAttribute("RESTRICETD_ACSESS") != null) ?
true : false;
   if (!restrictedAccess) {
%>
<link rel="stylesheet" type="text/css" href="/<%=appName%>/css/error.css"/>
<script type="text/javascript" src="/<%=appName%>/js/error.js"></script>
<div id="parentErrorDiv">
   <div id="errorBorderDiv">
      <div id="errorBorder">
         <div id="errorDiv">
            <div id="errorImage">
               <image src="/<%=appName%>/images/Error.svg" alt="NOT
FOUND"></image>
            </div>
            <div id="oops"><%=resourceBundle.getLabel("error.oops", language)%>
            </div>
            <div class="description">
               <div
class="description"><%=resourceBundle.getLabel("error.defaultErrorMessage",
language)%>
               </div>
            </div>
         </div>
      </div>
   </div>
</div>
```

```
<%
} else {
%>
<div class="restrictedAccessParentDiv" style="width:100%; height:100vh; background-
color: whitesmoke">
   <center>
      <div style="display: table-cell; width: 500px; height: 500px; vertical-align: middle; text-
align: center;">
     <pre><%=resourceBundle.getLabel("error.restrictedAccess", language)%>
       <a
href="javascript:redirect('<%=protocol%>://<%=cas%>/CAS/logout')"><%=resourceBundle.
getLabel("error.restrictedAccess.click", language)%></a>
<%=resourceBundle.getLabel("error.restrictedAccess.logout", language)%>
     </pre>
       </div>
   </center>
</div>
<%
   }
%>
```

Webpack Integration

Webpack.prod.js

```
const HtmlWebpackPlugin = require("html-webpack-plugin");
const ModuleFederationPlugin = require("webpack/lib/container/ModuleFederationPlugin");
const ESLintPlugin = require('eslint-webpack-plugin');
const path = require("path");
const { dependencies } = require("./package.json");

console.log("ENTERING PRODUCTION BUILD")
module.exports = {
    entry: "./src/index",
    mode: "production",
    devtool: 'source-map',
    output: {
        path: path.resolve(__dirname, 'build'),
        publicPath: "PCPRemoteModule/PCPBundle/react/",
        filename: '[name].[contenthash].js',

        clean: true

    },
    module: {
        rules: [
            {
                test: /\.(js|jsx)?$/,
                exclude: /node_modules/,
                use: [
                    {
                        loader: "babel-loader",
                        options: {
                            presets:  [
                                "@babel/preset-env",
                                "@babel/preset-react"
                            ]
                        }
                    }
                ]
            },
            {
                test: /\.css$/i,
                exclude: /\.module\.css$/i,
                use: ['style-loader', 'css-loader']
            },
            {
                test: /\.module\.css$/i,
                use: [
                    'style-loader',
                    {
                        loader: 'css-loader',
                        options: {
                            modules : {
                                localIdentName : "PCP__[name]_[local]__[hash:base64:5]"
                            }
                        }
                    }
                ]
            },
            {
                test: /\.(gif|png|jpe?g|svg)$/,
                type: 'asset/resource'
            }
        ]
    },
```

```
    plugins: [
        new HtmlWebpackPlugin({
            template: "./public/index.html"
        }),
        new ModuleFederationPlugin({
            name: "PCP",
            filename: "moduleEntry.js",
            exposes: {
                "./PCP": "./src/components/performancePrediction/performancePrediction",
                "./KPIManagement": "./src/components/settings/kPIManagement/kPIManagement",
                "./AlarmToKPICorrelation":
"./src/components/settings/alarmToKPICorrelationConfiguration/alarmToKPICorrelation"
            },
            shared: {
                ...dependencies,
                react: {
                    singleton: true,
                    requiredVersion: dependencies["react"]
                },
                "react-dom": {
                    singleton: true,
                    requiredVersion: dependencies["react-dom"]
                }
            }
        }),
        new ESLintPlugin()
    ],
    resolve: {
        extensions: [".js", ".jsx"]
    },
    target: "web"
};
```

Webpack.dev.js

```
const HtmlWebpackPlugin = require("html-webpack-plugin");
const ModuleFederationPlugin = require("webpack/lib/container/ModuleFederationPlugin");
const ESLintPlugin = require('eslint-webpack-plugin');
const path = require("path");
const { dependencies } = require("./package.json");

console.log("ENTERING DEVELOPMENT BUILD")
module.exports = {
    entry: "./src/index",
    mode: "development",
    devtool: 'source-map',
    output: {
        path: path.resolve(__dirname, 'build'),
        publicPath: "/PCP/react/",
        filename: '[name].[contenthash].js',

        clean: true
    },
    module: {
        rules: [
            {
```

```
            test: /\.(js|jsx)?$/,
            exclude: /node_modules/,
            use: [
               {
                  loader: "babel-loader",
                  options: {
                     presets:  [
                        "@babel/preset-env",
                        "@babel/preset-react"
                     ]
                  }
               }
            ]
         },
         {

            test: /\.css$/i,
            exclude: /\.module\.css$/i,
            use: ['style-loader', 'css-loader']
         },
         {
            test: /\.module\.css$/i,
            use: [
               'style-loader',
               {
                  loader: 'css-loader',
                  options: {
                     modules : {
                        localIdentName : "PCP__[name]_[local]__[hash:base64:5]"
                     }
                  }
               }
            ]
         },
         {
            test: /\.(gif|png|jpe?g|svg)$/,
            type: 'asset/resource'
         }
      ]
   },
   plugins: [
      new HtmlWebpackPlugin({
         template: "./public/index.html"
      }),
      new ModuleFederationPlugin({
         name: "PCP",
         filename: "moduleEntry.js",
         exposes: {
            "./PCP": "./src/components/performancePrediction/performancePrediction",
            "./KPIManagement": "./src/components/settings/kPIManagement/kPIManagement",
            "./AlarmToKPICorrelation":
"./src/components/settings/alarmToKPICorrelationConfiguration/alarmToKPICorrelation"
         },
         shared: {
            ...dependencies,
            react: {
               singleton: true,
               requiredVersion: dependencies["react"]
            },
            "react-dom": {
               singleton: true,
```

```
                requiredVersion: dependencies["react-dom"]
            }
        }
    }),
    new ESLintPlugin()
],
resolve: {
    extensions: [".js", ".jsx"]
},
target: "web"
};
```

The configurations provided in the webpack files are explained below.

entry: The path of entry file of the react application from where the execution starts, must be specified here.

mode: The deployment mode must be specified here. It can be either production or development. By default, Production is considered.

devtool: This option controls how the source-maps must be generated. This configuration has many parameters that can be provided. Source-map is used to provide the uncompressed files in the developer console so that it is easier for debugging purpose.

output: It is a Json entry where we can provide multiple parameters. In host and remote modules, we are using four parameters namely path, publicPath, filename and clean.

1. **Path** is provided as **path.resolve(__dirname, 'build')** where __dirname resolves to the current directory and followed by build directory. Hence, the above provided path resolves to:

   <remote>/<remote-web>/src/main/react/build

   The files that are generated after the react build is completed are put in the above folder that can be used for the deployment purpose.

2. **PublicPath** specifies the public URL of the output directory (bundle.js) when referenced in a browser. It is the path where the bundle files (or output files) are present.

   The publicPath is provided as:

   <uniqueRemoteModulePrefix>/<remoteModuleName>Bundle/react/: in production. In this case, the bundle.js call is sent to the host module that redirects it to the remote module.

   /<uniqueRemoteModulePrefix>/react/: in development. In this case, the bundle.js call is sent directly to the remote module to load the output or the bundle.js files in the browser.

3. **Filename** is the filename of the output file. The top-level output key contains a set of options instructing webpack on how and where it should output your bundles, assets, and anything else you bundle or load with webpack. [contenthash] changes when the content of a particular asset changes.
4. **Clean** is used to clean the build directory every time before the new assets is populated.

Module rules: It is an array of objects containing the rules to parse various types of files. The loaders used to parse different types of files are listed below.

- For js and jsx files, babel-loader is used.
- For pure css files, css-loader and style-loader are used. The css module files are excluded from parsing using the exclude attribute.
- For css modules, css-loader and style-loader are used but with different options. localIdentName is used for naming conventions the classnames in the build.
- For images, type is provided as asset/resource that allows the parsing of image files during the build.

Plugins: An array of plugins are provided here.

- **HtmlWebpackPlugin** simplifies creation of HTML files to serve the webpack bundles. This is especially useful for webpack bundles that include a hash in the filename that changes every compilation.
- **ESLintPlugin** is used for emitting the es6 errors in javascript files during the build time.
- **ModuleFederationPlugin** allows a build to provide or consume modules with other independent builds at runtime.

name: This is the name of the module.

filename: It is the name of the file which provides the access to that module.

exposes: This is a json that is provided in the remote module which contains the components that needs to be exposed to the host module.

Syntax:

```
{
  "<exposedComponentPlaceholderFromRemoteModule>": "<pathOfComponentTobeExposed>"
}
```

remotes: This is a json that is provided in the host module which contains the moduleEntry url of the remote modules along with the name of the remote module.

Syntax:

```
{
  "<Placeholder>": "<remoteModuleName>@ <remoteModuleName>ModuleEntry/react/"
}
```

shared: This is used to share the versions of the dependencies between the host and remote modules.

resolve: It is used to configure how modules are resolved. Attempt to resolve these extensions in order.

target: It instructs webpack to generate runtime code for a specific environment. Web is the default configuration that compiles to usage in browser like environment.

Webpack.prod.js	**Webpack.dev.js**
Mode is production.	Mode is development.
Devtool is not provided. In this case, the files are compressed and are not accessible in their original form.	Devtool is provided as source-map. In this case, the files are accessible in their original format that makes it easy for debugging purpose during the development phase.
Warnings are masked in the developer console in production mode.	Warnings are present in the developer console in development mode.

avanseus

Profiles

```
<profiles>
  <profile>
    <id>development</id>
    <build>
      <finalName>PCP</finalName>
      <plugins>
        <plugin>
          <groupId>org.jasig.mojo.jspc</groupId>
          <artifactId>jspc-maven-plugin</artifactId>
          <version>2.0.0</version>
          <executions>
            <execution>
              <phase>compile</phase>
              <goals>
                <goal>compile</goal>
              </goals>
            </execution>
          </executions>
          <configuration>
            <sources>
              <directory>${basedir}/src/main/webapp</directory>
              <includes>
                <include>**/*.jsp</include>
              </includes>
            </sources>
          </configuration>
          <dependencies>
            <dependency>
              <groupId>org.jasig.mojo.jspc</groupId>
              <artifactId>jspc-compiler-tomcat8</artifactId>
              <version>2.0.2</version>
            </dependency>
            <dependency>
              <groupId>javax.servlet.jsp</groupId>
              <artifactId>jsp-api</artifactId>
              <version>2.1</version>
            </dependency>
          </dependencies>
        </plugin>

        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-war-plugin</artifactId>
          <version>2.1.1</version>
          <configuration>
            <webResources>
              <resource>
                <targetPath>react</targetPath>
                <directory>${basedir}/src/main/react/build</directory>
              </resource>
            </webResources>
          </configuration>
        </plugin>
        <plugin>
          <groupId>org.codehaus.mojo</groupId>
          <artifactId>exec-maven-plugin</artifactId>
          <version>1.3.2</version>
          <executions>
```

```
                <execution>
                  <id>npm run reinstall</id>
                  <goals>
                    <goal>exec</goal>
                  </goals>
                  <phase>compile</phase>
                  <configuration>
                    <executable>npm</executable>
                    <arguments>
                      <argument>run</argument>
                      <argument>reinstall</argument>
                      <argument>--prefix</argument>
                      <argument>${basedir}/src/main/react/</argument>

                    </arguments>
                  </configuration>
                </execution>
                <execution>
                  <id>npm run build</id>
                  <goals>
                    <goal>exec</goal>
                  </goals>
                  <phase>compile</phase>
                  <configuration>
                    <executable>npm</executable>
                    <arguments>
                      <argument>run</argument>
                      <argument>dev-build</argument>
                      <argument>--prefix</argument>
                      <argument>${basedir}/src/main/react/</argument>

                    </arguments>
                  </configuration>
                </execution>
              </executions>

              <configuration>
                <environmentVariables>
                  <CI>false</CI>

<NPM_CONFIG_PREFIX>${basedir}/src/main/react/npm</NPM_CONFIG_PREFIX>

<NPM_CONFIG_CACHE>${NPM_CONFIG_PREFIX}/cache</NPM_CONFIG_CACHE>
                  <NPM_CONFIG_TMP>${project.build.directory}/npmtmp</NPM_CONFIG_TMP>
                </environmentVariables>
              </configuration>
            </plugin>
          </plugins>
        </build>
      </profile>
      <profile>
        <id>production</id>
        <activation>
          <activeByDefault>true</activeByDefault>
        </activation>
        <build>
          <finalName>PCP</finalName>
          <plugins>
            <plugin>
```

```
            <groupId>org.jasig.mojo.jspc</groupId>
            <artifactId>jspc-maven-plugin</artifactId>
            <version>2.0.0</version>
            <executions>
              <execution>
                <phase>compile</phase>
                <goals>
                  <goal>compile</goal>
                </goals>
              </execution>
            </executions>
            <configuration>
              <sources>
                <directory>${basedir}/src/main/webapp</directory>
                <includes>
                  <include>**/*.jsp</include>
                </includes>
              </sources>
            </configuration>
            <dependencies>
              <dependency>
                <groupId>org.jasig.mojo.jspc</groupId>
                <artifactId>jspc-compiler-tomcat8</artifactId>
                <version>2.0.2</version>
              </dependency>
              <dependency>
                <groupId>javax.servlet.jsp</groupId>
                <artifactId>jsp-api</artifactId>
                <version>2.1</version>
              </dependency>
            </dependencies>
          </plugin>

          <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-war-plugin</artifactId>
            <version>2.1.1</version>
            <configuration>
              <webResources>
                <resource>
                  <targetPath>react</targetPath>
                  <directory>${basedir}/src/main/react/build</directory>
                </resource>
              </webResources>
            </configuration>
          </plugin>
          <plugin>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>exec-maven-plugin</artifactId>
            <version>1.3.2</version>
            <executions>

              <execution>
                <id>npm run reinstall</id>
                <goals>
                  <goal>exec</goal>
                </goals>
                <phase>compile</phase>
                <configuration>
                  <executable>npm</executable>
```

```
                    <arguments>
                      <argument>run</argument>
                      <argument>reinstall</argument>
                      <argument>--prefix</argument>
                      <argument>${basedir}/src/main/react/</argument>

                    </arguments>
                  </configuration>
                </execution>
                <execution>
                  <id>npm run build</id>
                  <goals>
                    <goal>exec</goal>
                  </goals>
                  <phase>compile</phase>
                  <configuration>
                    <executable>npm</executable>
                    <arguments>
                      <argument>run</argument>
                      <argument>build</argument>
                      <argument>--prefix</argument>
                      <argument>${basedir}/src/main/react/</argument>

                    </arguments>
                  </configuration>
                </execution>
              </executions>

              <configuration>
                <environmentVariables>
                  <CI>false</CI>

<NPM_CONFIG_PREFIX>${basedir}/src/main/react/npm</NPM_CONFIG_PREFIX>

<NPM_CONFIG_CACHE>${NPM_CONFIG_PREFIX}/cache</NPM_CONFIG_CACHE>
                  <NPM_CONFIG_TMP>${project.build.directory}/npmtmp</NPM_CONFIG_TMP>
                </environmentVariables>
              </configuration>
            </plugin>
          </plugins>
        </build>
    </profile>
</profiles>
```

It contains two profiles, with each profile containing id and build parameters. Consider either development or production build by passing the profile id as parameter in mvn clean install.

One profile has the id as development and it is used to take the build for development purpose. It internally uses npm run dev-build that takes the webpack configuration from **webpack.dev.js**.

The command for taking the development build of a module is:

```
mvn clean install -P development
```

Another profile has the id as production and it is used to take the build for production purpose. It internally uses npm run build that takes the webpack configuration from **webpack.prod.js**. By default, the active profile is production.

The command for taking the production build of a module is:

```
mvn clean install -P production
or
mvn clean install
```

NOTE: This developer guide has elaborated the steps of creating a remote module. After the successful creation of remote module, it is the responsibility of the developer to make it production ready by creating necessary files like Docker image, Kubernetes services and Helm chart. Creation of Helm chart and Docker image for a remote module is not included in this document.