

Ibrahima BERETE
Brayan KUTLAR

Step 1: Créer une application de composants partagée

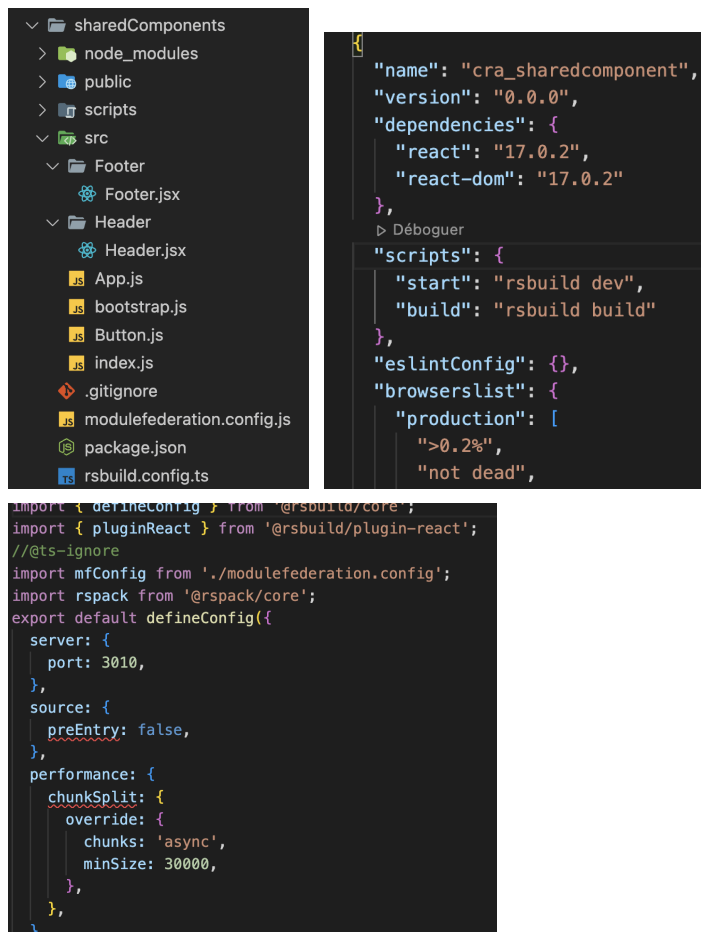
On a copié le projet "remote" et l'appeler "sharedComponent".

Dans sharedComponent/package.json on a renommé l'application en "cra_sharedcomponent"

Dans sharedComponent/rsbuild.config.ts on a changé le port de l'application en 3010

Dans cra/package.json, ajouter le nouveau dossier dans workspace, on l'a nommé sharedComponent

On a créé 2 nouveaux fichiers Header et Footer, dans sharedComponent, qui sont des composants partagés sur les différentes interfaces.



Header :

```
import React from "react";

const Header = () => {
  return (
    <header style={{ backgroundColor: "green", color: "white", padding: "20px", textAlign: "center" }}>
      <h1>Bienvenue sur notre projet Micro Front End</h1>
    </header>
  );
}

export default Header;
```

Footer :

```
cra > sharedComponents > src > Footer > Footer.jsx > ...
1  import React from 'react';
2
3  const Footer = () => {
4    return (
5      <footer
6        style={{ backgroundColor: 'blue', color: 'white', padding: '20px', textAlign: 'center' }}
7      >
8        <p>&copy; Fait par Brayan et Ibrahima en 2024</p>
9      </footer>
10   );
11 };
12
13 export default Footer;
14
```

Interface Admin :

Bienvenue sur notre projet Micro Front End

Basic Host-Remote

Host Component Title

Host

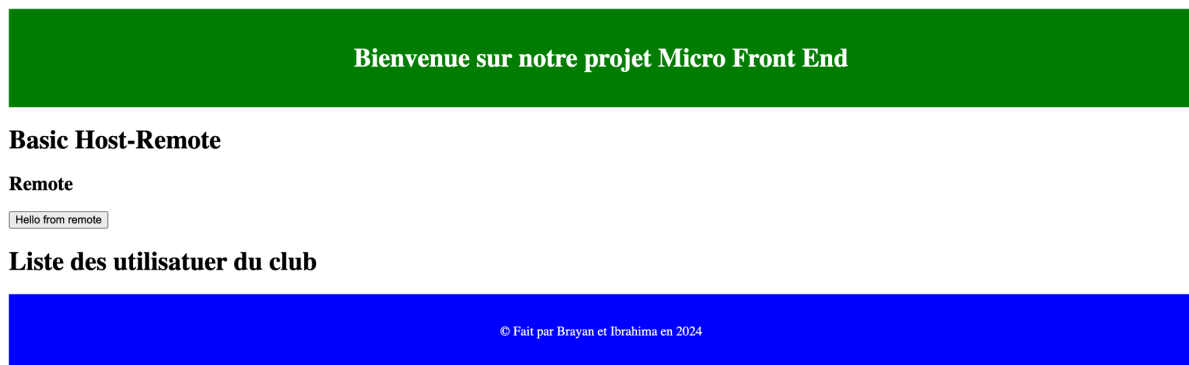
Hello from remote

Choisir un fichier | psm2quiz.pdf

Upload

© Fait par Brayan et Ibrahima en 2024

Interface Utilisateur :



Step 2 :

On a créé un composant unique pour la partie host et remote, en passant des paramètres.

RemoteComponent :

```
// RemoteComponent.jsx
import React from 'react';

function RemoteComponent({ info }) {
  return (
    <h1>
      Liste {info}
    </h1>
  );
}

export default RemoteComponent;
```

HostComponent :

```
// HostComponent.jsx
import React from 'react';

function HostComponent({ title }) {
  return <p>{title}</p>;
}

export default HostComponent;
```

Résultat interface communiquer dans la première partie.

Step 3: Host App development (Client View)

On a créé un api test avec mock.api puis on a fait appel à cette requête.

Voici le code pour fetch l'api :

```

const [data, setData] = useState([]);
const [loading, setLoading] = useState(true);

useEffect(() => {
  async function fetchData() {
    try {
      const response = await fetch(
        'https://65c380e839055e7482c10bde.mockapi.io/microfront/api/v1/test',
      );
      const jsonData = await response.json();
      setData(jsonData);
      setLoading(false);
    } catch (error) {
      console.error('Error fetching data:', error);
    }
  }
  fetchData();
}, []);

```

Voici le style d'affichage des données :




```

{loading ? (
  <p>Loading data...</p>
) : (
  <Table>
    <TableHead>
      <TableRow>
        <TableCell>Avatar</TableCell>
        <TableCell>Name</TableCell>
        <TableCell>Front</TableCell>
      </TableRow>
    </TableHead>
    <TableBody>
      {data.map(item => (
        <TableRow key={item.id}>
          <TableCell>
            <img src={item.avatar} alt="Avatar" />
          </TableCell>
          <TableCell>{item.name}</TableCell>
          <TableCell>{item.front}</TableCell>
        </TableRow>
      ))}
    </TableBody>
  </Table>
)

```

Voici le résultat :

Liste des utilisateur du club

Avatar	Name	Front
	Justin Upton	Shoes
	Evan Skiles	Wagon
	Muriel Koss PhD	Health

Step 4: Ajouter l'upload de fichiers (Admin View)

Utilisation de l'Api AWS S3 pour uploader des fichiers (n'importe quel type de fichier, pdf, images etc) vers le cloud public OVH.

Nous avons tout d'abord créé le compte, afin de créer une base de données pour stocker nos fichiers upload.

Voici le code pour uplaod le fichier :

```

import React, { useState } from 'react'; // 7.4k (gzipped: 3k)
import AWS from 'aws-sdk';

const UploadFile = () => {
  const [selectedFile, setSelectedFile] = useState(null);

  const handleFileChange = (event) => {
    setSelectedFile(event.target.files[0]);
  };

  const handleUpload = async () => {
    if (!selectedFile) {
      alert('Please select a file');
      return;
    }

    const s3 = new AWS.S3({
      accessKeyId: '893452ce6d0a4c53b6932efca9ccd08e',
      secretAccessKey: 'Y 6733517fd8e84e8d98579efff530a14f',
      region: 'SBG',
    });

    const params = {
      Bucket: 'microcontainer',
      Key: selectedFile.name,
      Body: selectedFile,
    };

    try {
      await s3.upload(params).promise();
      alert('File uploaded successfully');
    } catch (error) {
      console.error('Error uploading file:', error);
      alert('Error uploading file. Please try again.');
```

```

    };

    try {
      await s3.upload(params).promise();
      alert('File uploaded successfully');
    } catch (error) {
      console.error('Error uploading file:', error);
      alert('Error uploading file. Please try again.');
```

Voici l'affichage visuel :

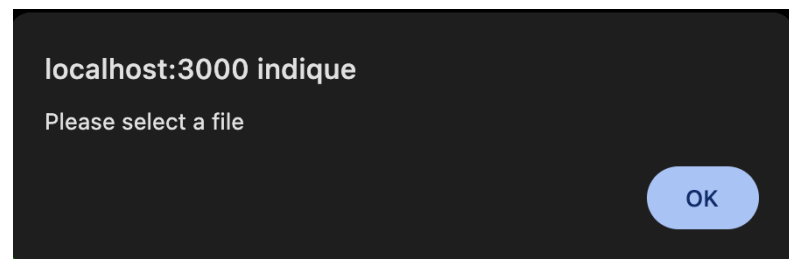
- Nous avons un bouton qui permet de sélectionner le fichier à upload



- Après avoir choisi le fichier nous avons le message qui change et qui affiche le nom du fichier choisis



- Dans la situation, où nous avons pas de fichier et qu'on upload nous avons l'alert suivant:



- Pour effectuer l'upload nous disposons d'un bouton upload



Nous avons des erreurs de cors, voici l'alert qu'on reçoit :

