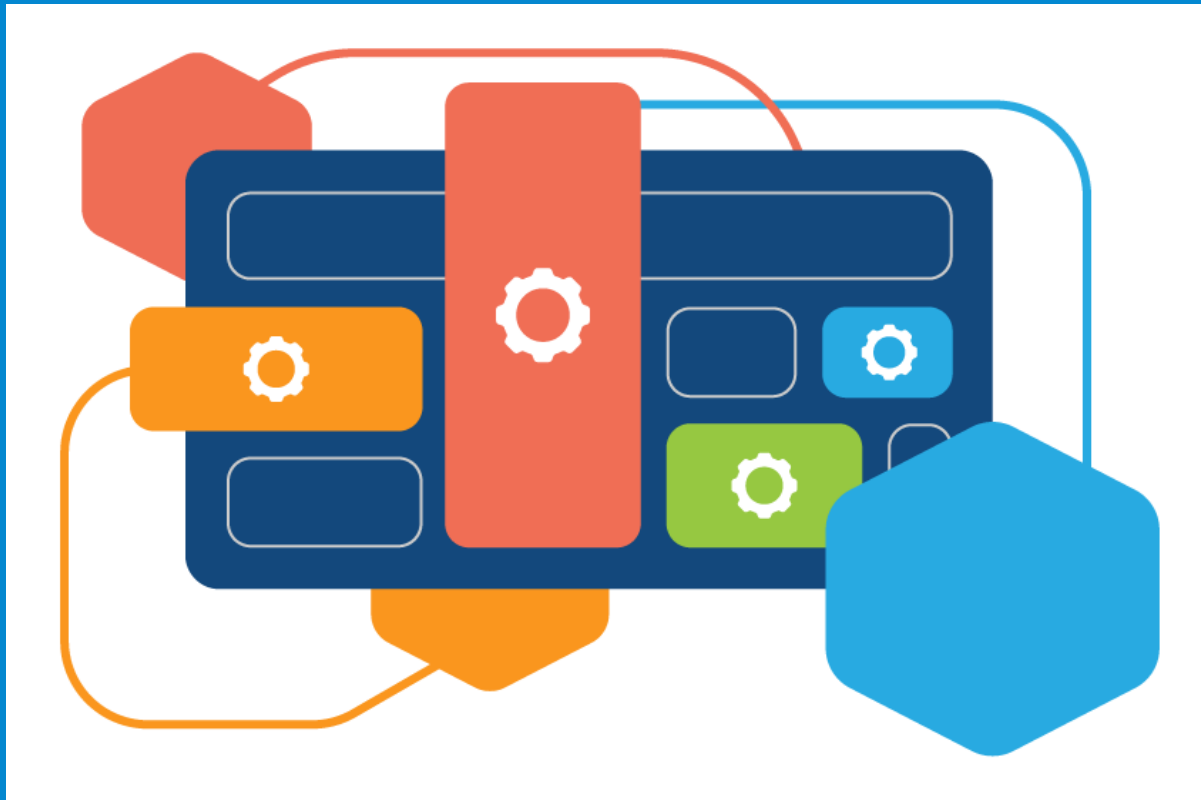


# Création d'un Microfrontend avec Module Federation



# Microfrontend ?

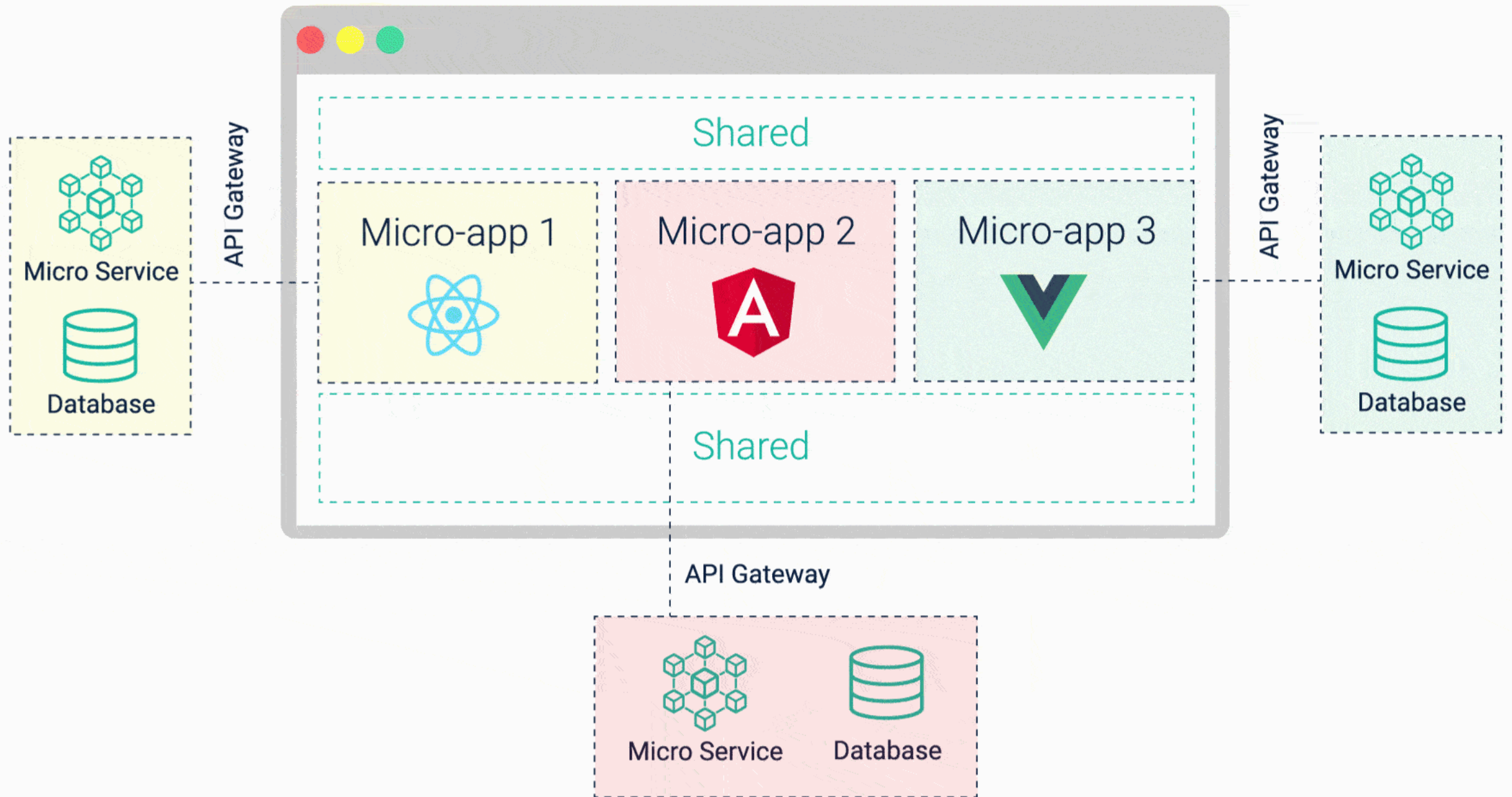
## Microservices

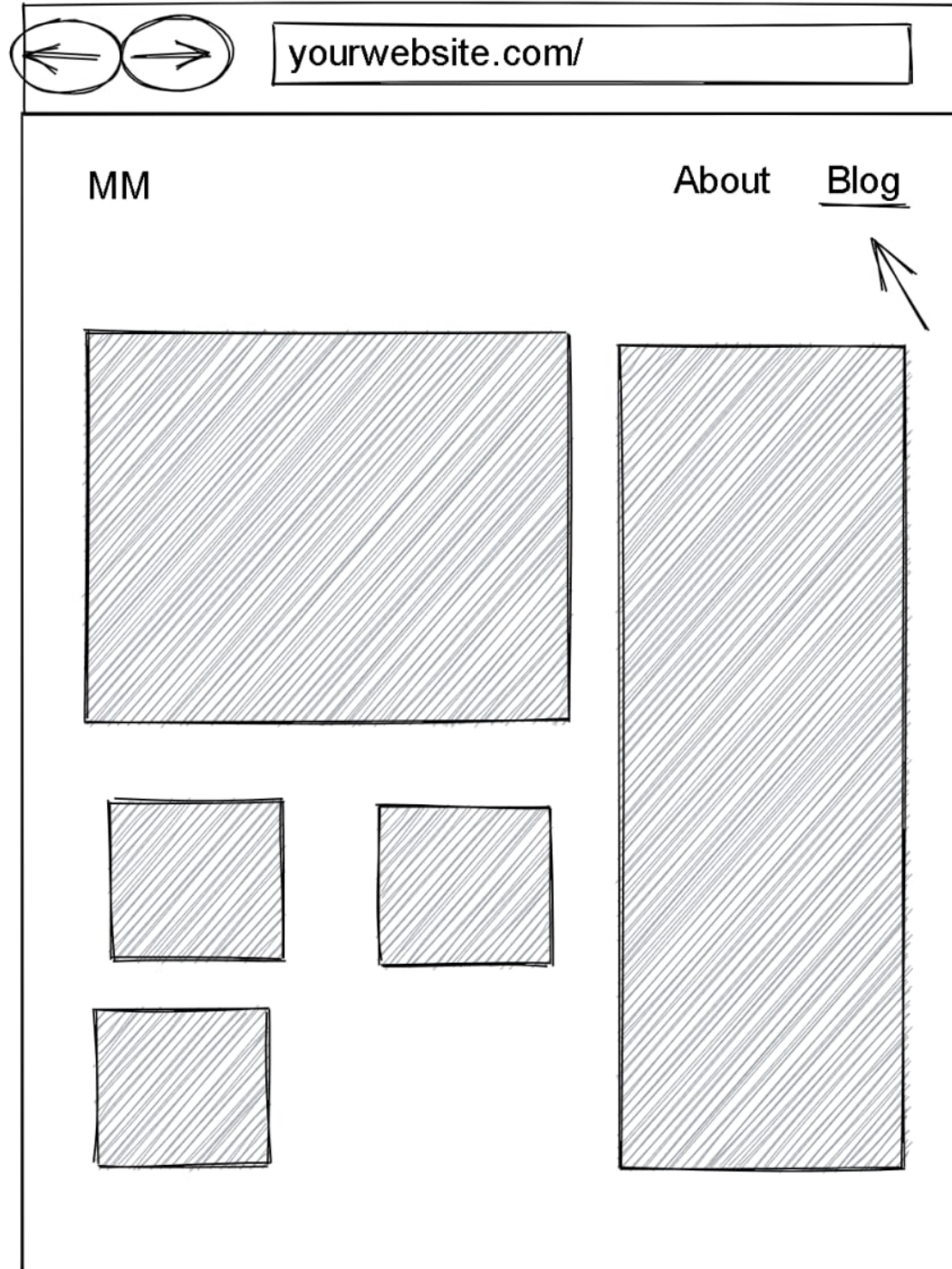
Définition :

“ An architectural style where independently deliverable frontend applications are composed into a greater whole.

*Martin Fowler*

”



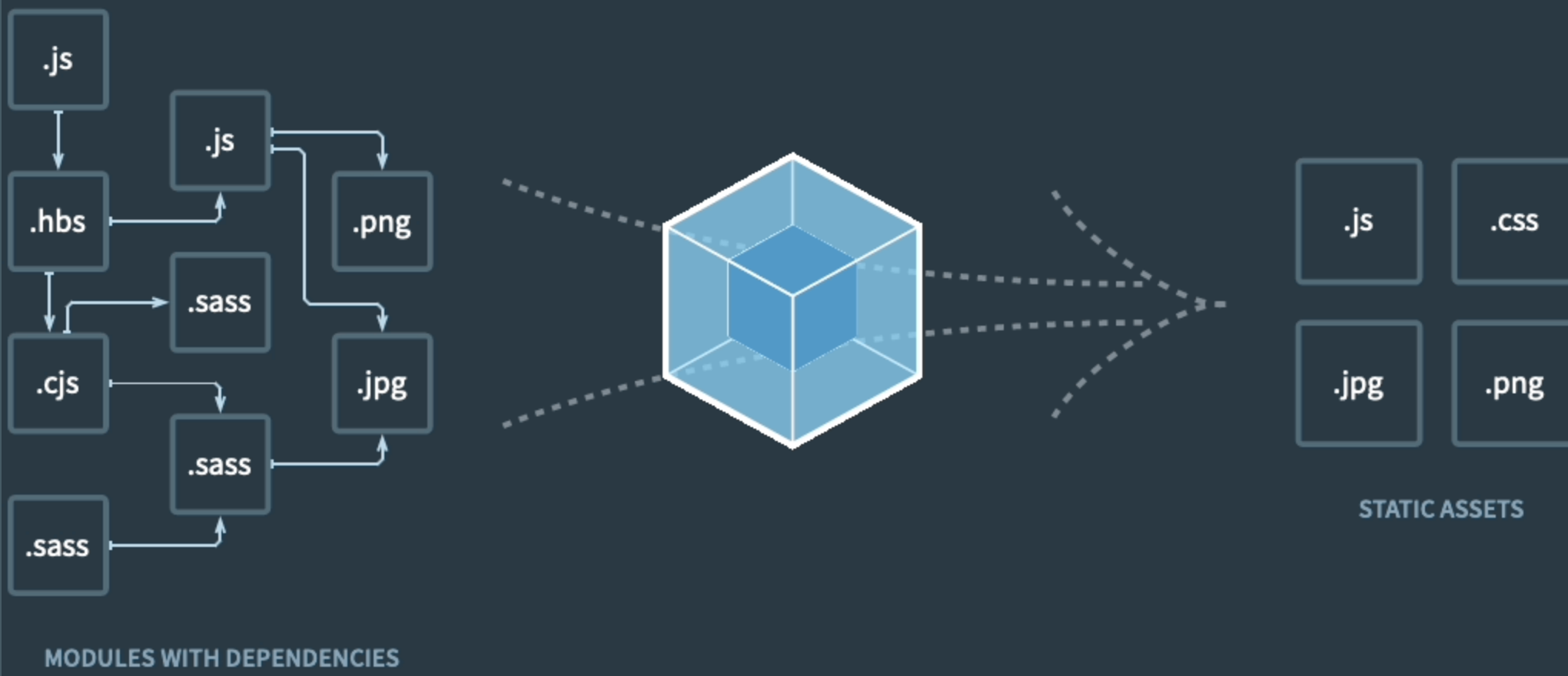


# Comment faire ?

- Avec Module Federation
- Avec des Web Components
- Avec des IFrames

# Webpack

bundle your assets



# Module Federation

- Plugin Webpack (Webpack 5)
- Chargement asynchrone de modules distants (pas dans le code de l'application).
  - Le code est chargé dynamiquement à l'exécution, avec les **dépendances** si nécessaire.
- Plus large que le Microfrontend, peut aussi être utilisé côté backend

# Comment faire avec React ?

**Tout dépend de la façon dont on souhaite construire notre application !**

- En utilisant Webpack directement
- Create React App
- Create React App Rewired
- Vite
- Next

La documentation officielle fournit des [exemples](#)

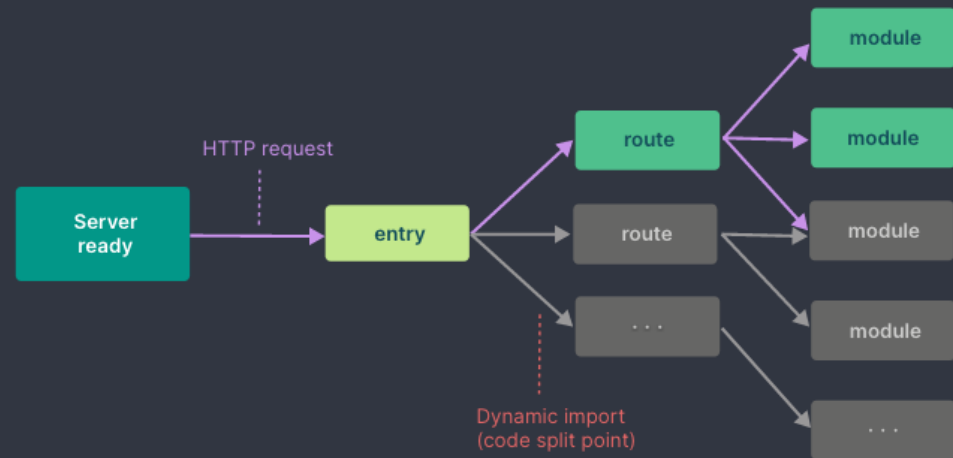


# Vite

Outil front-end JS pour améliorer la rapidité de développement avec une compilation optimisée pour la production.

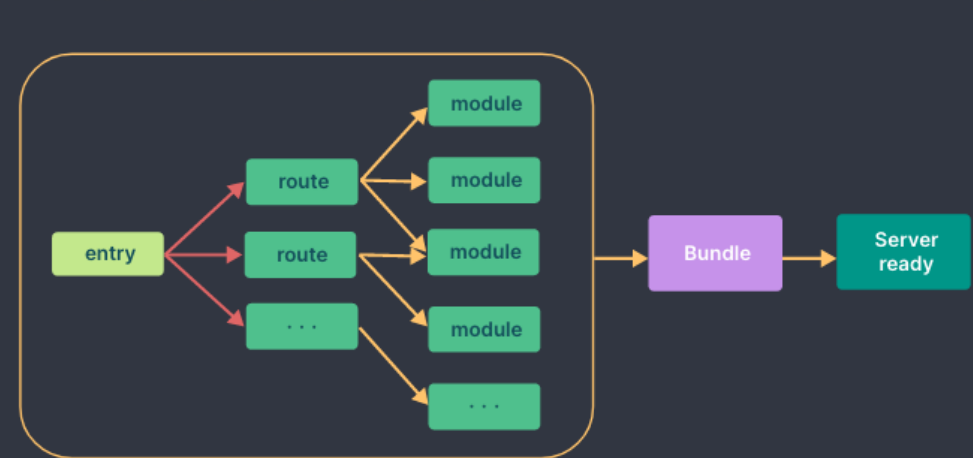
## Webpack

### Native ESM based dev server



## Vite

### Bundle based dev server



# On commence ?

Toutes les ressources sont disponibles sur le dépôt github [ddecrulle/workshop-module-federation](https://github.com/ddecrulle/workshop-module-federation).

**Vous pouvez commencer par forker le dépôt.**

Le starter est très simple et contient essentiellement de la CI et des outils pour faire du monorepo.

# Création des projets

Créons 2 projets vite

```
yarn create vite  
  # Project Name : host  
  # Framework React/Typescript
```

```
yarn create vite  
  # Project Name : remote  
  # Framework React/Typescript
```

# Lancement des applications

```
npx lerna bootstrap #Télécharge les dépendances  
yarn dev #Lance les 2 applications  
yarn build #Build les 2 applications
```

Fixons les ports de lancement, host : 5000, remote 5001.

```
"scripts": {  
  "dev": "vite --port 5000 --strictPort",  
  "build": "tsc && vite build",  
  "lint": "eslint src --ext ts,tsx --report-unused-disable-directives --max-warnings 0",  
  "preview": "vite preview --port 5000 --strictPort"  
}
```

# Modifications

Pour commencer et éviter de mélanger host et remote, modifions le titre en ajoutant **Host** ou **Remote** dans `App.tsx`.

Je vous propose ensuite de customiser le bouton du **remote**.

Pour ce faire créons un composant Button dans `components/Button.tsx` et importons le dans notre `App.tsx`.

# Button.tsx

```
import './Button.css';

import { useState } from 'react';

export const Button = () => {
  const [state, setState] = useState(0);
  return (
    <div>
      <button
        id="click-btn"
        className="shared-btn"
        onClick={() => setState((s) => s + 1)}
      >
        Click me: {state}
      </button>
    </div>
  );
};
```

## Button.css

```
.shared-btn {  
  background-color: skyblue;  
  border: 1px solid white;  
  color: white;  
  padding: 15px 30px;  
  text-align: center;  
  text-decoration: none;  
  font-size: 18px;  
}
```

Dans App.tsx remplaçons le bouton par celui que nous venons de créer

```
import { Button } from "../components/Button"

...
{/* remplacer */}

<button onClick={() => setCount((count) => count + 1)}>
    count is {count}
</button>
{/* par */}

<Button />
```

Le code jusqu'à [cette étape](#).



# Configuration de Module Federation

Le plugin Vite : [@originjs/vite-plugin-federation](https://www.npmjs.com/package/@originjs/vite-plugin-federation)

On ajoute la dépendance dans le projet racine car elle est commune à toutes les apps et que c'est une devDependencies.

```
yarn add -D @originjs/vite-plugin-federation -W
```

# Configuration du build

Dans le vite.config.ts (des deux app)

```
export default defineConfig({  
  plugins: [react()],  
  build: {  
    modulePreload: false,  
    target: "esnext",  
    minify: false,  
    cssCodeSplit: false,  
  },  
});
```

# (Optionnel) TsconfigPath

```
yarn add -D vite-tsconfig-paths -W
```

```
import tsconfigPaths from "vite-tsconfig-paths";

export default defineConfig({
  plugins: [
    react(),
    ...,
    tsconfigPaths()
  ],
});
```

# Pour le remote

Cela se passe dans le fichier `vite.config.ts`

```
import federation from "@originjs/vite-plugin-federation";
export default defineConfig({
  plugins: [
    react(),
    federation({
      name: "remote",
      filename: "remoteEntry.js",
      exposes: { "./Button": "./src/components/Button.tsx" },
      shared: ["react", "react-dom"],
    }),
  ],
});
```

# Pour l'Host

```
import federation from "@originjs/vite-plugin-federation";
export default defineConfig({
  plugins: [
    react(),
    federation({
      name: "host",
      remotes: {
        remoteApp: "http://localhost:5001/assets/remoteEntry.js",
      },
      shared: ["react", "react-dom"],
    }),
  ],
});
```

## Import du bouton dans l'host

```
// Static import  
import Button from "remoteApp/Button";  
// Lazy import  
const App = React.lazy(() => import("remoteApp/Button"));
```

## Déclaration du type (pas optimal...)

fichier `custom.d.ts`

```
declare module "remoteApp/*";
```

```
// Dans l'App.tsx  
<Button />
```

# Lancement en local

```
yarn build  
yarn serve
```

L'application host devrait embarquer le bouton du remote !

<http://localhost:5000>

Le remote doit **absolument** utiliser le build lancé via `vite preview`.

L'host peut être en mode développement `vite dev`

Le code jusqu'à [cette étape](#).

# Des améliorations

Variabiliser le passage de l'url du remote.

Créer un fichier .env

```
VITE_REMOTE_URL=http://localhost:5001
```

[Documentation officielle](#)



## Dans le vite.config.ts

```
federation({
  name: "app",
  remotes: {
    remoteApp: {
      external: `Promise.resolve(import.meta.env["VITE_REMOTE_URL"] + "/assets/remoteEntry.js")`,
      externalType: "promise",
    },
  },
  shared: ["react", "react-dom"],
});
```

Ajouter `"baseUrl": "../src"` pour avoir des imports absolus (nécessite **tsConfigPath**).

# IntelliSense pour TypeScript

Pour activer l'IntelliSense sur les variables d'environnements, dans le fichier `vite-env.d.ts` ajouter :

```
interface ImportMetaEnv {  
  readonly VITE_REMOTE_URL: string;  
  // more env variables...  
}  
interface ImportMeta {  
  readonly env: ImportMetaEnv;  
}
```

Le code jusqu'à [cette étape](#).

# C'est cool mais partager un bouton ...

Autant faire une librairie (ou un système de design ) !

Et si on ajoutait l'application remote sur la route `/remote` ?

# Créer le router dans l'host

```
npx lerna add react-router-dom --scope=host
```

routes/root.tsx

```
import { createBrowserRouter } from "react-router-dom";  
import App from "App";  
import RemoteApp from "remoteApp/RemoteApp";  
export const router = createBrowserRouter([  
  { path: "/remote", element: <RemoteApp /> },  
  { path: "/", element: <App /> },  
]);
```

# Utiliser le router

Dans main.tsx

```
import React from "react";
import ReactDOM from "react-dom/client";
import { RouterProvider } from "react-router-dom";
import "./index.css";
import { router } from "routes/root";

ReactDOM.createRoot(document.getElementById("root") as HTMLElement).render(
  <React.StrictMode>
    <RouterProvider router={router} />
  </React.StrictMode>
);
```

# Exposer l'app du remote

Dans vite.config.ts

```
exposes:  
  {  
    "./Button": "./src/components/Button.tsx",  
    "./App": "./src/App.tsx",  
  }
```

# On teste

Comme tout à l'heure

```
yarn build  
yarn serve
```

<https://localhost:5000>

Le code jusqu'à [cette étape](#).



# Vite + React + Remote

Click me: 0

Edit `src/App.tsx` and save to test HMR

Click on the Vite and React logos to learn more



# Quelques points de vigilances

- L'host importe dynamiquement les modules du remote. Il n'y a pas de contrôle sur ce qui est importé au build time. Par conséquent on peut découvrir des **erreurs au runtime** !
  - Bien définir des contrats d'interface !
- L'architecture monorepo est, je pense, à conseiller dans le cas d'un microfrontend. Il faut cependant un gitflow solide.

# Et si le remote a lui aussi un router ?

Essayons de voir ce qu'il se passe !

# Création du router dans le remote

```
npx lerna add react-router-dom --scope=remote
```

routes/root.tsx

```
import { createBrowserRouter } from "react-router-dom";  
import App from "App";  
export const router = createBrowserRouter([  
  { path: "/remote-routes", element: <div>Test router in remote </div> },  
  { path: "/", element: <App /> },  
]);
```

# Utiliser le router

Dans main.tsx

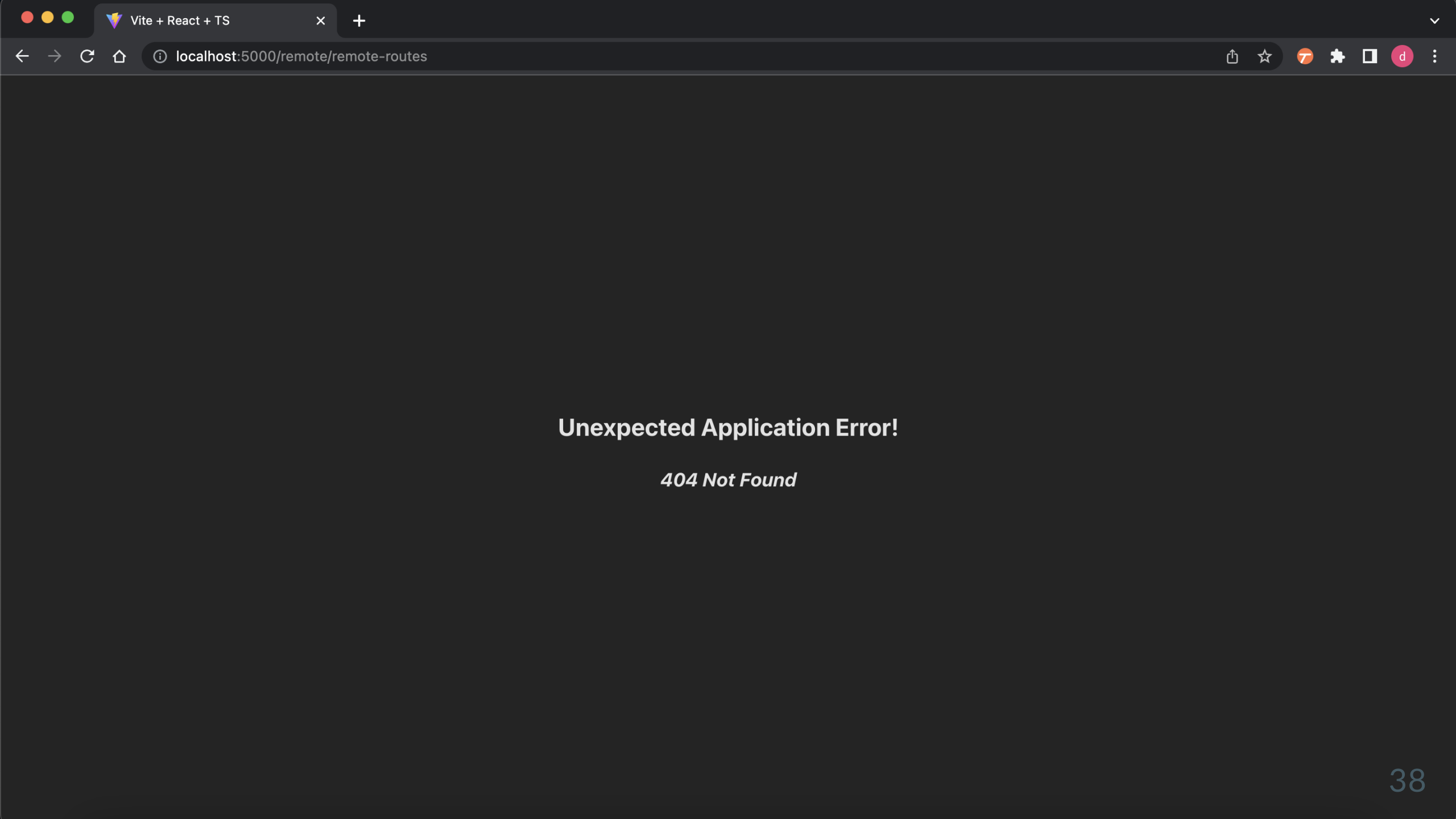
```
import React from "react";
import ReactDOM from "react-dom/client";
import { RouterProvider } from "react-router-dom";
import "./index.css";
import { router } from "routes/root";

ReactDOM.createRoot(document.getElementById("root") as HTMLElement).render(
  <React.StrictMode>
    <RouterProvider router={router} />
  </React.StrictMode>
);
```

# On teste ?

Quelqu'un a une idée du comportement ?

```
yarn build  
yarn serve
```



Unexpected Application Error!

*404 Not Found*

# Le Router

On ne peut pas avoir 2 [Browser Router](#) sur une "même" application.

## Solution

- Un Browser Router dans l'application host
- Un [Memory Router](#) dans l'application remote (uniquement en microFrontend)

**Un exemple, utilisant webpack, est  
disponible [ici](#).**

**On essaie de faire pareil dans notre contexte ?**



→ ↺ 🏠 ⓘ localhost:5000/remote

📄 📄

Éléments

Console

Sources

Réseau

Performances

Mémoire

Appli

»

✖ 2

💬 1

⚙️

⋮

⏪ ⏴

🔍

top ▾

👁

Filtrer

Tous les niveaux ▾

1 problème : 💬 1

9 masqués

▶ 📄 11 messages

▶ 👤 7 messages u...

▶ ✖ 2 erreurs

▶ ⚠ 2 avertisseme...

▶ ⓘ 5 infos

▶ 🐛 2 messages d...

✖ ▶ Error: Invalid hook call. Hooks can only be called VM496:react devtools...end compact.js:2367 inside of the body of a function component. This could happen for one of the following reasons:  
1. You might have mismatching versions of React and the renderer (such as React DOM)  
2. You might be breaking the Rules of Hooks  
3. You might have more than one copy of React in the same app  
See <https://reactjs.org/link/invalid-hook-call> for tips about how to debug and fix this problem.  
at Object.throwInvalidHookError (react-dom.development.js:16227:9)  
at useSyncExternalStore (react.development.js:1676:21)  
at RouterProvider (components.tsx:60:28)  
at Xh (index-7f0a9927.js:202:137)  
at Wk (index-7f0a9927.js:325:339)  
at Vk (index-7f0a9927.js:315:393)  
at Uk (index-7f0a9927.js:315:322)  
at Jk (index-7f0a9927.js:315:181)  
at Ok (index-7f0a9927.js:306:88)  
at Hk (index-7f0a9927.js:303:432)

✖ ▶ Uncaught Error: Invalid hook call. Hooks can only be called inside of index-7f0a9927.js:318 the body of a function component. This could happen for one of the following reasons:  
1. You might have mismatching versions of React and the renderer (such as React DOM)  
2. You might be breaking the Rules of Hooks  
3. You might have more than one copy of React in the same app  
See <https://reactjs.org/link/invalid-hook-call> for tips about how to debug and fix this problem.  
at Object.throwInvalidHookError (react-dom.development.js:16227:9)  
at useSyncExternalStore (react.development.js:1676:21)  
at RouterProvider (components.tsx:60:28)  
at Xh (index-7f0a9927.js:202:137)  
at Wk (index-7f0a9927.js:325:339)  
at Vk (index-7f0a9927.js:315:393)  
at Uk (index-7f0a9927.js:315:322)  
at Jk (index-7f0a9927.js:315:181)  
at Ok (index-7f0a9927.js:306:88)  
at Hk (index-7f0a9927.js:303:432)

>

41

# Pour aller plus loin

- Partager des états entre applications
- PWA
  - Offline
- Paramétrage du serveur applicatif pour gérer les CORS
- Déploiement

# Ressources

- [Micro Frontends](#) par Martin Fowler
- Le site [Micro Frontends](#), tiré du livre [Micro Frontends in Action](#)
- [Webpack 5 Module Federation : A game-changer in JavaScript architecture](#) par Zack Jackson (le créateur de Module Federation)
- [The History of Microfrontends](#)
- [Module Federation](#)
- [Vite plugin Federation](#)