I am writing this documentaion to assist the others finding out about State Management in Micro Frontend.

Step 1: Create your Store APPLICATION !

Before you think about any states or Context or Redux or etc, you must create an app for your store (npx create-mf-app).

Step 2: Create the files in your Store App

You can create a .js file which would be called “store” and create your Redux Store in it.

Install Rdux, React Redux, Redux toolkit.

Then do it in the code:

```
import { configureStore, createSlice } from "@reduxjs/toolkit";

import React from "react"
import { Provider, useSelector, useDispatch } from "react-redux"

////////////////// Making The Store
export const counterSlice = createSlice({
    name: counter,
    initialState: {counter: 0},
    reducers: {
        inc(state) {
            state.counter++
        },

        makeNull(state) {
            state.counter = 0
        },
    }
})
const store = configureStore({
    reducer: {
        counter: counterSlice.reducer
    }
})
```

Step 3: Make an object of your States and Actions to export

It’s Micro Frontend. You have to export the states and actions through the Applications not components. So in this case, make sure you build an object of your states and actions:

```
const {inc, makeNull} = counterSlice.actions;
const counter = useSelector(state => state.counter)

////////////////// Time to expose
export function useStore() {
    return {
        counter,
        inc: () => useDispatch(inc()),
        makeNull: () => useDispatch(makeNull())
    }
}
```

So we gathered all of them in “useStore”.

Also we have to do the exact same thing to the Provider :

```
export function StateProvider({children}) {
    return <Provider store={store}>{children}</Provider>
}
```

Step 4: Expose your Store App

Expose Loader in Webpack has a duty to make your module global.

Go to your webpack.config.js :

```
plugins: [
  new ModuleFederationPlugin({
    name: "store",
    filename: "remoteEntry.js",
    remotes: {},
    exposes: {
      "./counterStore" : "./src/counterStore"
    },
```

{“./what do you want to be called” : “the local url”}.

And now we can use it in other Applications but how?

Step 5: Read The Exposes in Other Applications

Go to the other applications which you want to show or change the states.

Example: You want to show the states in Header Application. Then go to webpack.config.js in Header Application to read the exposes :

```
plugins: [
  new ModuleFederationPlugin({
    name: "header",
    filename: "remoteEntry.js",
    remotes: {
      counterStore: "store@http://localhost:3011/remoteEntry.js"
    },
```

remotes: {**1**: "**2**@localhost(**3**)/remoteEntry.js"}

1: the exposed name(name in the exposed obj)

2: the name(name in the exposed plugin)

3: the port of the exposed app.

What is **remoteEntry.js** ? If you added "/remoteEntry.js" to the url of the exposed app, you would see the file exposes that other applications can use.

Step 6: Show the States in the app you used "remotes"

As we said before in the example, in header app we went to the webpack.config and used "remotes" to get the exposed app which is store.

And now let's go to the App.js in Header App:

```
import { useStore, StateProvider } from "counterStore/counterStore"; // 1

const App = () => {
  const {counter, inc} = useStore(); // 2

  return (
    (
      <React.Fragment>
        {/* {count} /////// 3 */}
        <Header app={{ name: "Micro-Hamid" }} />
        <div className="mt-10 text-3xl mx-auto max-w-6xl">
          <div>Name: header</div>
          <div>Framework: react</div>
          <div>Language: JavaScript</div>
          <div>CSS: Tailwind</div>
        </div>
      </React.Fragment>
    )
  )
}

ReactDOM.render(
  // 4
<StateProvider>
    <App />,
</StateProvider>,
document.getElementById("app"));
```

1: You import the object we made in the store, which was full of states and actions.

2: split them as a single variable.

3: You can show the states.

3.5 : Also you can use the actions, for example:

<button onClick={inc}> Do it </button>

4: wrap all the app with the provider.

Some warning:

- Pay attention that we use Redux and share the states in other places. SO we have to lunch the store sonner than the other apps, if not : error
- It’s better to import the states and actions in each components, rather than we pass it through the props.