

RBAC Implementation - Frontend Strategy

1. Context & Objective

- Each company can define its own roles (e.g. *Fleet Manager, Supervisor*).
- Roles differ between companies but are stored in a single shared database.
- We need a **flexible, dynamic RBAC system** that controls frontend behavior.
- **Goal:** Enforce access control at the **UI and route level** based on permissions.

2. Frontend Permissions Model

- The backend returns the user's allowed actions on get profile api:
The structure should be like : {
 ...userInfos,
 "Permissions":[
 {
 module_name:"Vehicle",
 Permissions:["create","delete","read","update"]
 }
 }...
]
}
- Store this permissions in global state (redux)
- Create a dynamic Hook that check the permission
- This hook accept 2 parameters (module,action)
- If the user is allowed we return true, otherwise we return false

```
useCan.tsx 2, U X
src > hooks > fleet > roles > useCan.tsx > default
1  import { useSelector } from "react-redux"
2  import { RootState } from "../../store"
3
4  const useCan = (module: string, action: string) => {
5    const { current } = useSelector((state: RootState) => state.user)
6    const userPermissions = current!.permissions
7
8    const modulePermission = userPermissions?.find((p) => p.module_name === module)
9    if (!modulePermission) {
10     return false
11   }
12
13   const isAllowed = modulePermission.includes(action)
14   return isAllowed
15 }
16
17 export default useCan
```

•

3. Frontend Rendering Small Ui

- Before displaying the small ui (Buttons,actionMenus) we call the general hook (we name it useCan for example)
- If the user is allowed we display the ui, otherwise we never show it
- Because this verification is repetitive action we will create a general component that verify the rendering of those small Ui. we pass to this component 3 required props which are : "Children" (The small ui to be displayed),"Module", "action"
- In this component we call our useCan hook . if user is allowed we return the children,otherwise we return null

```
src > components > fleet > roles > Can.tsx > default
1  import React from 'react'
2  import { useCan } from "../../hooks/fleet/_index"
3
4  interface Props {
5    children: React.ReactNode,
6    module: string,
7    action: string
8  }
9
10 const Can: React.FC<Props> = ({ children, module, action }) => {
11
12   const isAllowed = useCan(module, action)
13   return isAllowed ? children : null
14 }
15
16 export default Can
```

•

```

src > views > fleet > vehicle > all > list > VehiclesPage.tsx > [0] default
You, 1 second ago | 5 authors (badreddine mehenni and others)
1 import { Box, Typography } from "@mui/material";
2 import { useTranslation } from "react-i18next";
3 import { BiPlus } from "react-icons/bi";
4 import { PrimaryButton } from "../../components/fleet/buttons/_index";
5 import { VehiclesTable } from "../../components/fleet/tables/_index";
6 import { Link } from "react-router-dom";
7 import ImportExportActionsMenu from "../../components/fleet/menus/ImportExportActionsMenu";
8 import Can from "../../components/fleet/roles/Can";
9
10 const VehiclesPage: React.FC = () => {
11   const { t } = useTranslation("global");
12
13   return <>
14     <Box sx={{ display: { xs: 'block', md: 'flex' }, justifyContent: 'space-between' }}>
15       <Typography sx={{ fontWeight: 600, fontSize: 24 }} component='h1'>{t("Vehicles")}</Typography>
16
17       <Box sx={{ display: 'flex', gap: 2, mt: { xs: 2, md: 0 } }}>
18         <ImportExportActionsMenu exportPath="vehicles" importPath="vehicles" isVehicles={true} />
19         <Can module="vehicle" action="create">
20           <Link to="/fleet/vehicles/create">
21             <PrimaryButton text={t('Add vehicle')} icon={<BiPlus color='white' size={20} />} />
22           </Link>
23         </Can>
24       </Box>
25     </Box>
26     <VehiclesTable />
27   </>
28 }
29
30 export default VehiclesPage;
badreddine mehenni, 16 months ago • Add auth pages ...

```

4. Frontend Routes

- **ProtectedRoute** – restricts page access by module/action.
- Create a custom component that verify if the user is allowed to go to that page (for example create vehicle)
- If the user is allowed he continues, otherwise redirect to forbidden page

```

// Vehicles
{
  path: "vehicles",
  children: [
    { index: true, element: <VehiclesPage /> },
    { path: ":id", element: <VehicleDetailsPage /> },
    {
      path: "create", element: <ProtectedRoute module="vehicle" action="create">
        <CreateVehiclePage />
      </ProtectedRoute>
    },
    { path: 'edit/:id', element: <EditVehiclePage /> },
  ],
},

```

5. Resume

- User logs in Backend returns user profile api (already exists including his permissions)

- Frontend stores user profile in the redux (already exists)
- All concerned components and routes the general components validations (**Can** for small Ui and **protectedRoute** form entire route) and who in turn use the useCan to verify the permission
- When role/permissions change (from settings page) :
Refresh permissions by calling the user profile again