



Contexte GestEPI

Introduction

Dans le cadre de ses activités, une société de travaux en hauteur emploie des cordistes pour réaliser des tâches sur des chantiers en bâtiment, voirie ou en espaces verts. Pour travailler en hauteur de manière sécurisée, les cordistes utilisent des EPI. Les Equipements de Protection Individuel sont des outils de sécurité normée et qui se doivent d'être suivis et contrôlé.

Les EPI doivent être suivis et contrôlés de manière fréquente. Pour cela, l'entreprise désigne un gestionnaire d'EPI, ces derniers se doivent de tenir un registre dans lequel ils inscrivent les équipements ainsi que les contrôles qu'ils effectuent.

Pour réaliser ce suivi, le registre est généralement fait sur papier ou sur divers fichiers type tableur. L'idéal serait d'utiliser une application dédiée.

Mission

Votre mission est de développer une application permettant la gestion des EPI en couvrant plusieurs fonctionnalités principales :

- Saisie d'EPI

- Saisie des contrôles
- Consultation des EPI et leur historique de contrôle
- Consulter le détail d'un contrôle
- Alerte pour les contrôles à venir

Modélisation d'un EPI

Dans l'univers des cordistes, les EPI les plus courants sont :

- les cordes, sangles et longes
- les baudriers
- les casques
- les systèmes d'assurage et mousquetons

Un EPI est identifiable au travers de sa marque, son modèle et son numéro de série. Cependant les gestionnaires veulent pouvoir saisir un identifiant unique personnalisé.

Pour certains EPI, on souhaite enregistrer la taille et la couleur (par exemple pour les cordes ou les baudriers)

On doit pouvoir également enregistrer la date d'achat, la date de fabrication et la date de mise en service. Il faut savoir que pour les EPI textiles (baudriers, cordes, sangles ou longes), la loi impose le renouvellement de l'EPI tous les 10 ans quelque soit son état. Pour le matériel métallique, le gestionnaire des EPI peut le mettre à disposition des cordistes tant que l'état de celui-ci est correct.

Un EPI peut avoir un lien avec plusieurs contrôles et un contrôle pointe vers un unique EPI.

Chaque EPI a également une périodicité de contrôle qui lui est propre, il va être obligatoire de l'enregistrer pour la fonctionnalité d'alerte sur les prochains contrôles

Modélisation d'un contrôle

Pour le gestionnaire d'EPI, il faut qu'il puisse saisir dans un contrôle d'EPI plusieurs informations :

- la date du contrôle
- Le gestionnaire d'EPI qui effectue le contrôle
- l'EPI contrôlé
- le statut de l'EPI après le contrôle
 - Opérationnel
 - A réparer
 - Mis au rebut
- Des remarques

Use cases

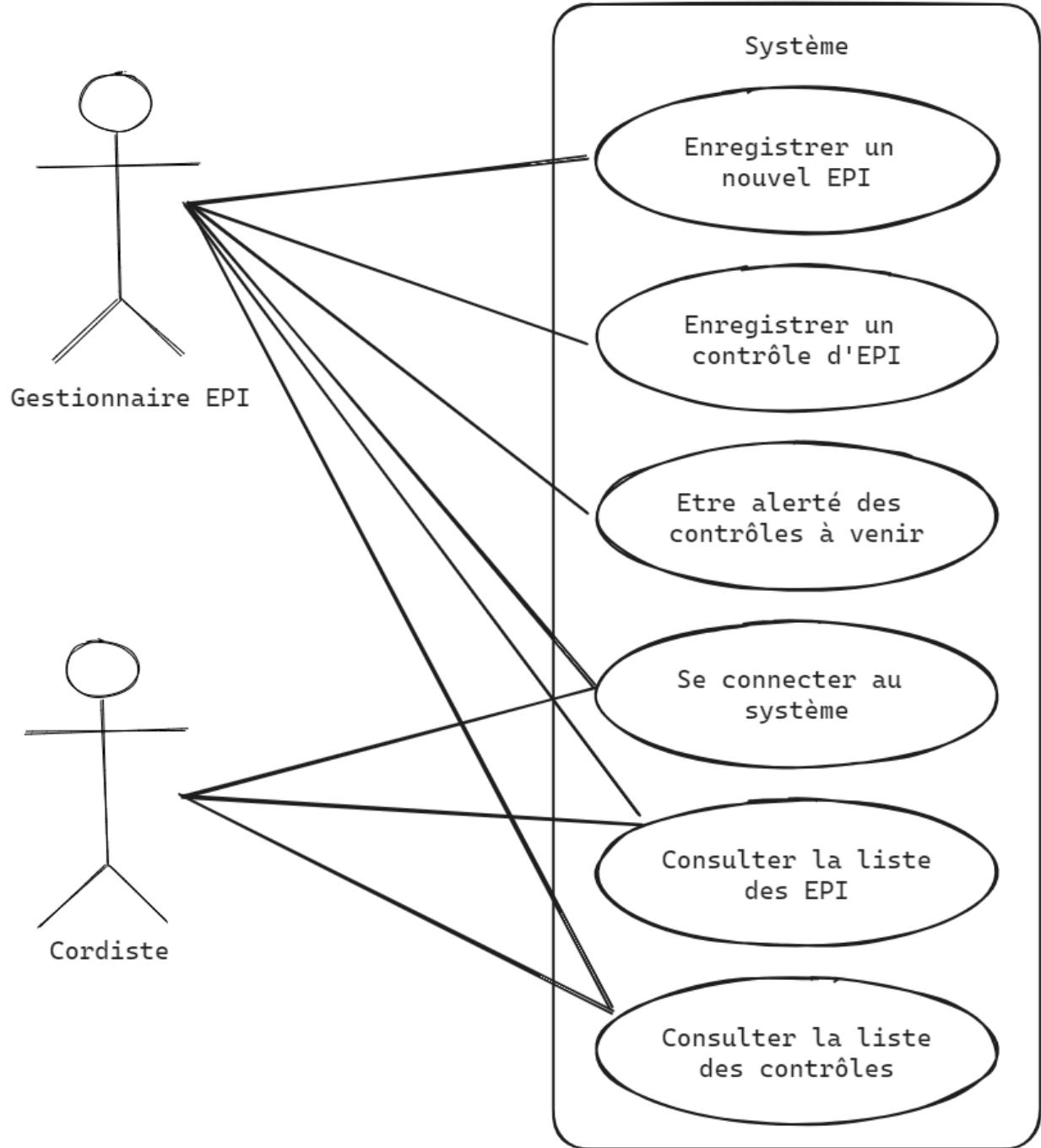


Diagramme de cas d'utilisation

Front

Template de projet

```

npm install -g @romgio/create-react-typescript-mui-app
npm install -g yarn # Pas besoin de refaire cette commande si vous l'avez déjà fa
npx @romgio/create-react-typescript-mui-app GestEPIFront

```

Maquette de l'application

GestEPI											
		Liste des EPI									
		Liste des contrôles									
+ AJOUTER EPI											
		Actions	ID interne	N° de série	Marque	Modèle	Type	Taille	Couleur	Date d'achat	Date de fab...
				CAS1	1029384756	Beal	Joker	CORDE	80m	rouge	29/02/2024
				CAS2	1029384756	Beal	Joker	CORDE	80m	rouge	05/03/2024
				CAS3	1029384756	Beal	Joker	CORDE	80m	rouge	05/03/2024
				CAS5	1029384756	Beal	Joker	CORDE	80m	rouge	05/03/2024
				CAS6	1029384756	Beal	Joker	CORDE	80m	rouge	05/03/2024

Page de gestion des EPI

Actions	EPI	Date du contrôle	Status	Contrôleur
	CAS1 (Beal Joker)	17/03/2021	Conforme	Roméo Giorgio
	CAS1 (Beal Joker)	17/03/2022	Conforme	Roméo Giorgio
	CAS1 (Beal Joker)	17/03/2023	Conforme	Roméo Giorgio
	CAS3 (Beal Joker)	25/03/2024	Conforme	Roméo Giorgio
	CAS6 (Beal Joker)	25/03/2024	Conforme	Roméo Giorgio
	CAS5 (Beal Joker)	25/03/2024	Conforme	Roméo Giorgio
	CAS3 (Beal Joker)	25/03/2024	A réparer	Roméo Giorgio
	CAS2 (Beal Joker)	25/03/2024	Conforme	Roméo Giorgio

Page de gestion des contrôles

Back

Template de projet

```
npm install -g @romgio/create-express-ts-mariadb-api
npm install -g yarn # Pas besoin de refaire cette commande si vous l'avez déjà fa
npx @romgio/create-express-ts-mariadb-api GestEPIBack
```

Interfaces

Afin d'utiliser les mêmes types dans les deux briques Back et Front, il est pertinent de les définir dans un package que l'on publiera sur npm afin de l'importer dans nos projets en tant que dépendance.

Création du package

Aller dans le répertoire GestEPI qui contient déjà les briques Front et Back.

```
rgio@MacBook-Pro-de-Romeo workspace % cd GestEPI
rgio@MacBook-Pro-de-Romeo GestEPI % ls
GestEPIBack    GestEPIFront    README.md
rgio@MacBook-Pro-de-Romeo GestEPI % █
```

Créer le répertoire GestEPIInterfaces et l'initialiser comme un projet node avec npm.

! Au moment de l'initialisation, npm demande plusieurs informations sur le package que l'on est en train de créer.

```
mkdir GestEPIInterfaces
cd GestEPIInterfaces
npm init
```

Installer Typescript

```
npm install typescript --save-dev
```

Configurer Typescript en créant un fichier `tsconfig.json` à la racine du répertoire `GestEPIInterfaces`.

```
1  {
2      "compilerOptions": {
3          "target": "ES6",
4          "module": "CommonJS",
5          "declaration": true,
6          "outDir": "./dist",
7          "strict": true
8      },
9      "include": ["src/**/*"]
10 }
```



- `declaration: true` → génère les fichiers `.d.ts` (fichiers de définition de types) pour chaque fichier `.ts`.
- `outDir: "./dist"` → indique où placer les fichiers compilés (Javascript et `.d.ts`).
- `include: ["src/**/*"]` → indique que tous les fichiers dans le dossier `src` sont inclus dans la compilation.

Configurer le package avec le fichier `package.json`.



- Modifier la propriété main pour faire pointer vers `dist/index.d.ts` .
- Ajouter la propriété types pour exposer les types du package en la faisant pointer vers `dist/types.d.ts` .
- Penser à modifier le nom du dépôt en y ajoutant un trigramme par exemple (pour éviter les conflits).
- Ajouter un script `"build"` .
- Ajouter la propriété `"files"` avec la valeur `["dist"]` afin de s'assurer que seul le dossier `dist` sera inclus dans le package publié (utile pour éviter d'inclure les fichiers sources `src` ou d'autres fichiers inutiles).

```
{  
  "name": "gestepiinterfacesrgio",  
  "version": "1.0.0",  
  "main": "dist/index.d.ts",  
  "types": "dist/types.d.ts",  
  "scripts": {  
    "build": "tsc"  
  },  
  "files": [  
    "dist"  
  ],  
  "keywords": [],  
  "author": "Roméo Giorgio",  
  "license": "ISC",  
  "description": "",  
  "devDependencies": {  
    "typescript": "^5.6.3"  
  }  
}
```

Création des types

Créer les interfaces et enum utiles au projet dans le fichier `/src/types.ts`.

The screenshot shows the VS Code interface with the following details:

- EXPLORER** view: Shows the project structure under `GESTEPI`, including `GestEPIBack`, `GestEPIFront`, `GestEPIInterfaces` (selected), `dist`, `node_modules`, and `src` (selected). Inside `src`, `index.ts` and `types.ts` are listed.
- package.json**: Standard package configuration.
- App.tsx**: A component file.
- index.ts**: A file containing the following code:

```
1 export enum UserType {
2     ADMIN = 1,
3     MANAGER = 2,
4     USER = 3,
5 }
6
7 export interface User {
8     id: number;
9     firstName: string;
10    lastName: string;
11    type: UserType;
12    phone?: string;
13    mail?: string;
14 }
15
16 export interface EPIType {
17     id: string;
18 }
```

- types.ts**: A file containing the following code:

```
1 export enum UserType {
2     ADMIN = 1,
3     MANAGER = 2,
4     USER = 3,
5 }
6
7 export interface User {
8     id: number;
9     firstName: string;
10    lastName: string;
11    type: UserType;
12    phone?: string;
13    mail?: string;
14 }
15
16 export interface EPIType {
17     id: string;
18 }
```

Comme `src/index.ts` est le point d'entrée, c'est lui qui expose le contenu. On peut donc exporter les types. C'est utile pour exposer seulement certains types et garder les autres pour un usage interne.

The screenshot shows the VS Code interface with the following details:

- EXPLORER** view: Shows the project structure under `GESTEPI`, including `GestEPIBack`, `GestEPIFront`, `GestEPIInterfaces` (selected), `dist`, `node_modules`, and `src` (selected). Inside `src`, `index.ts` and `types.ts` are listed.
- package.json**: Standard package configuration.
- App.tsx**: A component file.
- index.ts**: A file containing the following code:

```
1 export * from "./types";
```

On peut maintenant builder nos types qui apparaîtront dans le dossier `dist`.

```
npm run build
```

Publier le package

Pour publier un package sur npm, il faut se connecter à npm avec la commande `npm login`.

```
rgio@mac GestEPIInterfaces % npm login
npm notice Log in on https://registry.npmjs.org/
Login at:
https://www.npmjs.com/login?next=/login/cli/bfb64fd4-90ca-459e-9fb1-23b9201c7b10
Press ENTER to open in the browser...
```

On peut maintenant publier le package de manière publique (publier de manière privée est payant) avec la commande `npm publish --access public`.

```
rgio@mac GestEPIInterfaces % npm publish --access public
npm notice
npm notice 📦 gestepiinterfacesrgio@1.0.0
npm notice Tarball Contents
npm notice 25B dist/index.d.ts
npm notice 939B dist/types.d.ts
npm notice 313B package.json
npm notice Tarball Details
npm notice name: gestepiinterfacesrgio
npm notice version: 1.0.0
npm notice filename: gestepiinterfacesrgio-1.0.0.tgz
npm notice package size: 638 B
npm notice unpacked size: 1.3 kB
npm notice shasum: 6203f97113b0a9ecdb5ae207238521f31a9b1974
npm notice integrity: sha512-K/ZwaCuoZn0kv[...]ywgcVqiYCn26w==
npm notice total files: 3
npm notice
npm notice Publishing to https://registry.npmjs.org/ with tag latest and public access
+ gestepiinterfacesrgio@1.0.0
```

Utilisation dans le projet

On peut maintenant installer le package dans un autre projet node avec la commande `npm install <nom du package des types>` ou `yarn add <nom du package des types>`.

```

rgio@mac GestEPIInterfaces % cd ../GestEPIFront
rgio@mac GestEPIFront % yarn add gestepiinterfacesrgio
yarn add v1.22.22
warning package.json: License should be a valid SPDX license expression
warning GestEPIFront@1.0.0: License should be a valid SPDX license expression
[1/4] 🔎 Resolving packages...
[2/4] 🚚 Fetching packages...
[3/4] ⚙️ Linking dependencies...
warning " > @testing-library/user-event@13.5.0" has unmet peer dependency "@testing-library/react@^13.5.0".
warning "react-scripts > eslint-config-react-app > eslint-plugin-flowtype@8.0.3" has unmet peer dependency "@babel/plugin-syntax-flow@^7.14.5".
warning "react-scripts > eslint-config-react-app > eslint-plugin-flowtype@8.0.3" has unmet peer dependency "@babel/plugin-transform-react-jsx@^7.14.9".
[4/4] ↵ Building fresh packages...
success Saved lockfile.
warning GestEPIFront@1.0.0: License should be a valid SPDX license expression
success Saved 1 new dependency.
info Direct dependencies
└ gestepiinterfacesrgio@1.0.0
info All dependencies
└ gestepiinterfacesrgio@1.0.0
$ husky
✨ Done in 10.67s.

```

On peut utiliser les types dans l'autre projet node.

The screenshot shows the Visual Studio Code interface with the following sections:

- EXPLORER**: Shows the project structure under **GESTEPI**, including **GestEPIBack**, **GestEPIFront**, **.husky**, **bin**, **node_modules**, **public**, and **src** (which contains **App.css**, **App.test.tsx**, and **App.tsx**).
- PACKAGE MANAGER**: Shows the command `yarn add gestepiinterfacesrgio` with a success message: `success Saved 1 new dependency.`
- EDITOR**: Displays the file **App.tsx** with the following code:

```

GestEPIFront > src > App.tsx > ...
You, 3 hours ago | 2 authors (Romeo Giorgio and one other)
1 import logo from "./logo.svg";
2 import "./App.css";
3 import {User} from "gestepiinterfacesrgio"
4 
5 function App() {
6   console.log("");
7 
8   return (
9     <div className="App">
10       <header className="App-header">
11         <img src={logo} className="App-logo" alt="Logo" />
12       </header>
13       <main className="App-main">
14       </main>
15     </div>
16   );
17 }
18 
19 export default App;

```