Bonnes pratiques

1. Gestion des Formulaires Angular

- **Organisation et création :**
  - Organiser les champs de formulaire dans un même fichier avec un formGroup.
  - **Déclaration d'un FormGroup :**

```
formGroup: FormGroup;
```

  - **Association des FormControls :**

```
this.mainInfosFormGroup = new FormGroup({
  lastName: new FormControl("", [Validators.required]),
  firstName: new FormControl("", [Validators.required]),
  roles: new FormControl("", [Validators.required]),
  email: new FormControl("", [Validators.required, Validators.email]),
  isArchived: new FormControl(false),
});
```

- **Liaison et validation :**
  - Lier les formControls aux champs du formulaire et gérer les erreurs.
  - **Exemple de template avec gestion d'erreur :**

```
<div class="flex flex-col grow">
  <mat-label class="label" for="lastName">{{ tr.language().USER_FORM_LASTNAME }}</mat-
  <input class="h-12" formControlName="lastName" id="lastName" matInput type="text" />
  @if (checkError(mainInfosFormGroup, "lastName", "required")) {
    <mat-error class="text-sm text-red-500">
      {{ tr.language().USER_LASTNAME_REQUIRED }}
    </mat-error>
  }
</div>
```

- **Mise à jour dynamique :**
  - Utiliser les signaux pour mettre à jour la page en temps réel.
  - Préférer l'usage de `await firstValueFrom` de rxjs plutôt que `subscribe`.

---

2. Architecture et Factorisation du Code

- Factoriser au maximum les écrans et composants pour réutiliser le code.
- Utiliser `#region` pour organiser et commenter le code de manière structurée.
- Préférer des noms de variables et de fonctions longs et explicites pour améliorer la lisibilité.

---

3. Conventions de Nommage et Bonnes Pratiques de Développement

- **Nommage :**
  - Respecter l'écriture en CamelCase.
- **Commits :**
  - Inclure le nom de la branche dans le commit pour une meilleure traçabilité.
- **Débogage :**
  - Utiliser les points d'arrêt (breakpoints) plutôt que d'insérer des `console.log()` dans le code.

---

4. Gestion des Styles et du CSS

- Utiliser des variables CSS pour une meilleure maintenabilité et cohérence.
- **Exemple d'utilisation des variables dans le CSS :**

```
.ag-cell-label-container {
  @apply gap-2;
  justify-content: start;
  --ag-accent-color: #{$primary};
}
```

---

5. Optimisation des Requêtes API

- Optimiser les retours des requêtes pour limiter le nombre d'appels à l'API.
- **Exemple :** Faire en sorte qu'une requête POST retourne l'état courant complet (et non seulement l'ID) afin d'éviter un nouvel appel API.

---

6. Génération de Services via ng-open api et Décoration des Contrôleurs .NET

- **Pour générer uniquement des services JSON :**
  - Ajouter les décorateurs suivants aux contrôleurs .NET :

```
[Produces("application/json")]
[Consumes("application/json")]
```

- **Pour des routes spécifiques :**
  - Surcharger localement si une route nécessite un autre type de contenu, par exemple :

```
[HttpPost]
[Consumes("multipart/form-data")]
```