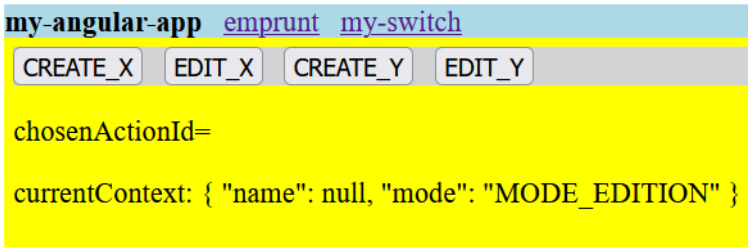
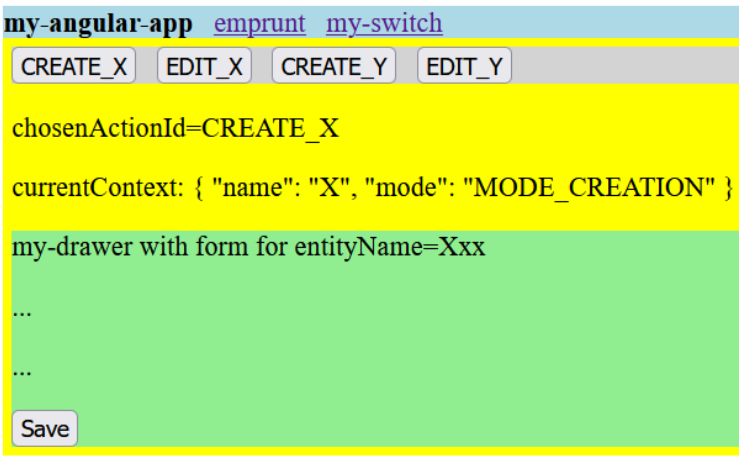
Enoncés de TPs complémentaires Angular/GCA

1. Tp stratégie avec contexte et signaux

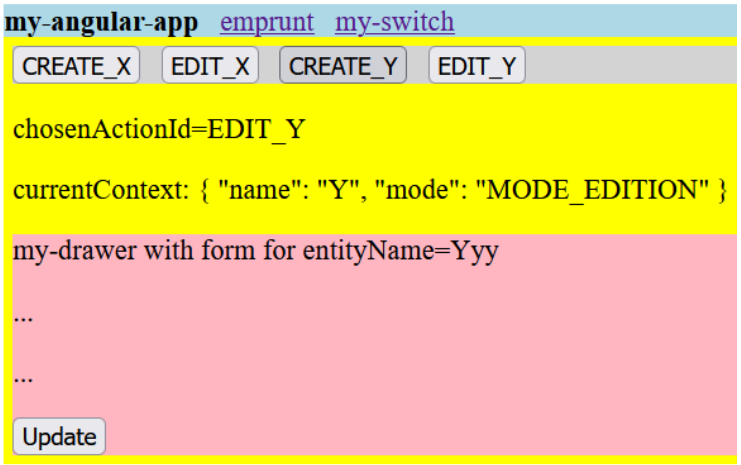
Si aucune action sélectionnée/déclenchée ,
rien ne s'affiche au niveau du drawer :



Si action "CREATE_X" sélectionnée/déclenchée :
le drawer affiche le résultat de la strategy X → entityName=Xxx et couleur de fond verte
et bouton "Save" si currentContext.mode == "MODE_CREATION"



Si action "EDIT_Y" sélectionnée/déclenchée :
le drawer affiche le résultat de la strategy Y → entityName=Yyy et couleur de fond rose
et bouton "Update" si currentContext.mode == "MODE_EDITION"



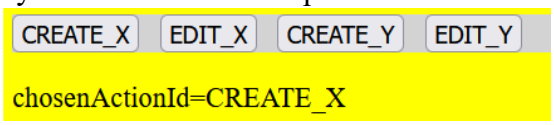
- Tout ce qui sera codé dans ce TP pourra éventuellement être placé dans un nouveau répertoire (ex : src/app/withActionsAndContext)

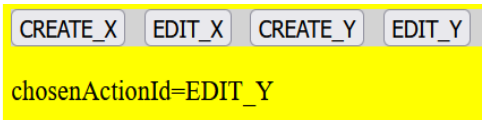
- Le composant principal de ce Tp pourra éventuellement être nommé "my-switch" et sera intégré d'une manière ou d'une autre au composant principal src/app/component

⇒ dès que possible *ng g component mySwitch* dans src/app/withActionsAndContext
et fond jaune et intégration dans l'application (via routing ou imbrication directe)

1.1. composant actions-header avec input() et output()

Créer un nouveau composant "***ActionsHeaderComponent***" (ci dessous en fond gris)
ayant ce look et ce comportement :





- Le composant ActionsHeaderComponent comporte en entrée via **input()** une ***liste de nom/id de boutons d'actions*** de nom "**actionsList**"
  (ex : ***myActionsList*** = ['CREATE_X' , 'EDIT_X', 'CREATE_Y', "EDIT_Y"]) qui seront automatiquement générés
- Le composant ActionsHeaderComponent comporte en sortie via **output()** un événement personnalisable ***actionClick*** remontant l'**id du bouton d'action sélectionné/déclenché**

Pour tester le bon fonctionnement du composant "***ActionsHeaderComponent***" on pourra l'intégrer au sein du composant parent "MySwitchComponent" avec un code (coté .html) de ce genre :

```
<app-actions-header (actionClick)="onActionClick($event)"
                    [actionsList]="myActionsList" />
<p>chosenActionId={{chosenActionId}} </p>
```

Au sein de la première version , la méthode événementielle **onActionClick()** de MySwitchComponent se contentera de mettre à jour la propriété chosenActionId à afficher.

Quelques indications :

Au sein de *ActionsHeaderComponent* on pourra (coté .ts) facultativement s'appuyer sur :

- **input<String[]>([])** et sur **output<string>()**
- **this.actionClick.emit(buttonId);**
- **fireActionEvent(event : MouseEvent){ let buttonId = (<HTMLElement>event.target).id; … }**

Et du coté .html on pourra s'appuyer sur une boucle **@for(… actionsList()…)** pour faire mieux que
<!-- <button id="EDIT_X" (click)="fireActionEvent($event)">EDIT_X</button> →

1.2. signal currentContext dans MyContextService

- Dans src/app/withActionsAndContext par exemple, créer un nouveau service angular **MyContext**Service
- Dans le haut du fichier my-context.service.ts (ou ailleurs) placer le nouveau type de donnée typescript MyContextMode et la classe Mycontext avec un code de ce genre par exemple :

```
type MyContextMode = "MODE_CREATION" | "MODE_EDITION"

export class MyContext{
   constructor(public name :string|null=null,
               public mode :MyContextMode="MODE_EDITION"){}
}
```

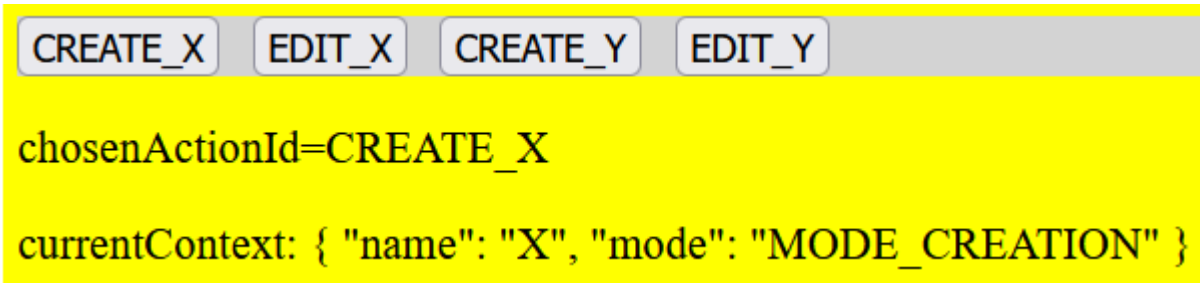
- ajouter ensuite ***currentContext*** à construire via ***signal<MyContext>(…)*** à l'intérieur de la classe ***MyContextService*** .

Utilisation de MyContextService au sein de MySwitchComponent :

- injecter le service MyContextService dans la classe MySwitchComponent
- au sein de la méthode onActionClick(actionId:string){…} de l'étape précédente du Tp ajouter un appel à cette nouvelle méthode :

```
setContextFromActionId(actionId:string){
   let newCurrentContext : MyContext | undefined;
   switch(actionId){
     case 'EDIT_X': newCurrentContext =new MyContext('X','MODE_EDITION'); break;
     case 'CREATE_X':newCurrentContext =new MyContext('X','MODE_CREATION'); break;
     case 'EDIT_Y': newCurrentContext =new MyContext('Y','MODE_EDITION'); break;
     case 'CREATE_Y':newCurrentContext =new MyContext('Y','MODE_CREATION'); break;
   }
   if(newCurrentContext)
      this.myContextService.currentContext.set(newCurrentContext)
 }
```

Tester le bon nouveau comportement en ajoutant un affichage du contenu su signal *this.myContextService.currentContext* dans la partie .html de MySwitchComponent .



1.3. MyDrawerComponent avec strategy

Dans *src/app/withActionsAndContext* par exemple, on pourra **ajouter** un **fichier *my-strategy.ts*** comportant par exemple un code de ce genre :

```
import { WritableSignal } from "@angular/core";

//vision abstraite du composant "drawer" sur laquelle la stratégie sera appliquée :
export interface IDrawer{
     entityName :WritableSignal<string>
     bgColor :WritableSignal<string>
}
//classe abstraite de la stratégie avec méthode de déclenchement :
export abstract class MyStrategy{
      abstract init(drawer : IDrawer):void;
}
//stratégie concrète "XStrategy" fixant le signal entityName à "Xxx" et bgColor à "lightgreen"
export class XStrategy extends MyStrategy{
    init(drawer : IDrawer){
      drawer.entityName.set("Xxx")
      drawer.bgColor.set("lightgreen")
    }
```

```
}

//stratégie concrète "YStrategy" fixant le signal entityName à "Yyy" et bgColor à "lightpink"
export class YStrategy extends MyStrategy{
    init(drawer : IDrawer){
      drawer.entityName.set("Yyy")
      drawer.bgColor.set("lightpink")
    }
}

//factory élémentaire pour fabriquer une stratégie selon un nom ("X" ou "Y" ou ...)
export class MyStrategyFactory{
   getStrategy(name:string):MyStrategy|null{
      switch(name){
         case "X" : return new XStrategy();
         case "Y" : return new YStrategy();
         default : return null;
      }
   }
}
```

Dans *src/app/withActionsAndContext* par exemple, coder ensuite le nouveau composant **MyDrawer**Component selon ces indications :

- la classe ***MyDrawerComponent*** devra implémenter l'interface ***IDrawer*** et sera par conséquence obligée de comporter les signaux ***entityName*** et ***bgColor*** .
- Ajouter au sein de la classe MyDrawerComponent une injection du service myContextService et une instance de la classe MyStrategyFactory (que l'on pourra par exemple créer à coup de new …) .
- Ajouter également displayMode=signal("none") au sein de MyDrawerComponent
- Ajouter enfin un **effect()** au sein de classe MyDrawerComponent qui :
  * partira de ctx = this.myContextService.currentContext()
  * affichera la valeur de ctx dans la console en mode JSON
  * fixera la valeur du signal displayMode à "block" si ctx.name n'est pas null
  ou bien à "none" sinon
  * déclenchera une récupération de la stratégie à appliquer selon le nom ctx.name
  * appliquera la stratégie à sur le drawer courant "this"

Exemple de code partiel :

```
const strategy = this.myStrategyFactory.getStrategy(ctx.name??"?");
strategy?.init(this);
```

Dans le .html de *MyDrawerComponent* on :

- afficher la valeur du signal ***entityName*** initialisé par la stratégie
- fixera la valeur du signal ***displayMode*** en tant que valeur de **[style.display]** sur une englobante . De la même manière on tiendra compte de la valeur du signal ***bgColor*** initialisé par la stratégie pour fixer la valeur de la couleur de fond
- on affichera soit un bouton "Update" soit un bouton "Save" selon la valeur récupérée de myContextService.currentContext().mode ("MODE_EDITION" ou "MODE_CREATION")

⇒ résultat attendu : ressemblant au copies d'écran (2 ou 3pages avant celle ci) du début de ce TP.