

# Enoncés de TPs complémentaires Angular 19+

## 1. Nouveau petit projet angular

- Vérifier si besoin l'installation de node/npm , @angular/cli et VisualStudioCode
- se créer un nouveau répertoire de tp (ex : tp-dateQuiVaBien)
- au sein de ce répertoire , créer un nouveau projet angular (ex : "my-app" ) via ng new
- ouvrir le répertoire de ce projet avec visual studio code
- lancer "ng serve" au sein d'un terminal (intégré ou pas à VSCode) de manière à vérifier le bon fonctionnement de l'application (http://localhost:4200)
- Au sein de app.component.html supprimer toutes les choses inutiles et ne garder que l'essentiel utile {{title}} et éventuellement <router-outlet />

## 2. TP "composant emprunt avec signaux"

- Créer un nouveau composant angular de nom Emprunt via ng g component
- intégrer de manière libre ce nouveau composant au sein du composant principal (soit via une nouvelle route , soit par imbrication directe, soit autrement)
- Le nouveau composant "Emprunt" devra avoir les fonctionnalités suivantes :

my-angular-app emprunt

**emprunt (calcul mensualité constante pour emprunt , avec signaux)**

montant=  
tauxInteretAnnuel= (en %)  
durée (nbMois)=

mensualite=**425.40263368479225**

NB :

- les éléments de base (à saisir) **montant**, **nbMois**, **tauxInteretAnnuelPct** seront (du coté .ts) codés comme des signaux (à construire via la fonction **signal(valeur\_initiale)** .
- On pourra utiliser [(ngModel)] du coté .html (en pensant à l'importation nécessaire FormsModule du coté .ts)
- le signal **mensualite** sera idéalement construit à partir des autres signaux de base via la fonction **computed ( )=> ...)**

NB :

si l'on souhaite respecter la formule exacte de calcul d'une mensualité d'emprunt constante, on pourra éventuellement s'appuyer sur une fonction de ce genre :

```
calculMensualite(montant:number,nbMois:number,tauxInteretAnnuelPct:number){  
  let mensualite=0;  
  const tauxInteret = (tauxInteretAnnuelPct/12) / 100;  
  if(tauxInteret!=0)  
    mensualite = montant * tauxInteret / ( 1 - Math.pow(1+tauxInteret,- nbMois))  
  else  
    mensualite = montant / nbMois  
  return mensualite  
}
```

Suite facultative du Tp :

montant=   
tauxInteretAnnuel=  (en %)  
durée (nbMois)=

**mensualite=175.27760053243995**  
*emprunt de longue duree. besoins complémentaires*

Via un **effect()** , mettre à jour le signal **message** (par défaut vide)

de la manière à ce qu'il s'affiche "*emprunt de longue duree. besoins complémentaires*" si la valeur de nbMois est supérieure ou égale à 60 .

### 3. Tp signaux input(), output(), model() sur "task"

**my-angular-app** [emprunt](#) [my-switch](#) [task-crud](#)

#### Select and Edit Tasks

title	state	taskRequest	taskResponse
2plus2	done	combien vaut 2+2 ?	4
5moins4	todo	combien vaut 5-4 ?	
3fois0	todo	combien vaut 3*0 ?	

title: 2plus2  
request: combien vaut 2+2 ?  
response:

---

Ce TP portera sur le concept de "task" (avec états "todo" et "done").

On pourra organiser la structure du projet avec par exemple un nouveau répertoire `src/app/task`

**L'objectif principal de ce TP** consiste à développer un **nouveau composant "task-crud"** qui sera composé par 2 sous-composants indépendants : "generic-selector" et "task-editor"

- Le sous composant "generic-selector" permettra de **sélectionner une des tâches disponibles**.
- Le sous composant "task-editor" servira à modifier les valeurs d'une tâche sélectionnée.
- Le composant global "task-crud" comportera des propriétés "taskList" et "selectedTask" qui seront transmises au sous composants via des signaux en mode *input()* ou bien *model()* selon les besoins

### 3.1. Mise en place de la structure composée :

- Par exemple au sein du répertoire `src/app/task` , créer les 3 nouveaux composants **TaskCrudComponent** , **GenericSelectorComponent** et **TaskEditorComponent**
- Imbriquer `<app-generic-selector>` et `<app-task-editor>` au sein de `task-crud.component.html`
- Incorporer le nouveau composant **TaskCrudComponent** dans l'application angular d'une manière ou d'une autre (via routing ou via imbrication directe au sein de **AppComponent**)
- Par exemple au sein du répertoire `src/app/task`, ajouter un nouveau fichier **task.ts** ayant un contenu de ce genre :

```
export type TaskState = "todo" | "done"

export class Task{
  constructor(public title :string="my-task",
              public taskRequest : any = "combien vaut 2+2 ?",
              public taskResponse : any = "",
              public state :TaskState="todo"
            ) {}
}
```

Au sein de **TaskCrudComponent** , ajouter un début de code de ce genre :

```
initialTaskList = [ new Task("2plus2","combien vaut 2+2 ?"),
                   new Task("5moins4","combien vaut 5-4 ?"),
                   new Task("3fois0","combien vaut 3*0 ?")
]
```

- puis ajouter les propriétés **taskList** et **selectedTask**.
- **taskList** pourra être de type **Task[]** , soit **directement** , soit en tant que **signal()** , soit en tant que **model()**
- **selectedTask** pourra être de type **Task|null**, soit **directement** , soit en tant que **signal()** , soit en tant que **model()**
- **model()** ne serait utile que si le composant **TaskCrudComponent** aurait ultérieurement besoin de partager les informations **taskList** et **selectedTask** avec un composant parent.
- **signal<T>()** (**WritableSignal<T>**) semble un bon type de données pouvant facilement être transformé en **model()** si les besoins évoluent dans le futur .

## 3.2. V1 de generic-selector avec input() et output()

En TP, se débrouiller pour que le composant generic-selector puisse être incorporé dans un parent avec au minimum ce type de paramétrages :

```
<app-generic-selector
  [entities]="taskList()"
  (selectedEntityEvent)="onSelectedEntity($event)" />
```

et éventuellement certains paramétrages secondaires tels que ;

```
<app-generic-selector
  ...
  [mainFieldNames]="mainTaskFieldNames" />
```

**NB:** la syntaxe [xxx]="...." n'est envisageable que si xxx est déclaré en tant que **input(...)** au sein du sous composant GenericSelectorComponent .

**NB:** la syntaxe (xxxEvent)="onXxx(\$event)" n'est envisageable que si xxx est déclaré en tant que **output(...)** au sein du sous composant GenericSelectorComponent .

D'un point de vue comportemental , le sous composant GenericSelectorComponent doit être capable d'afficher une liste d'entités pour que l'on puisse sélectionner l'une d'elles.

Solution 1 simpliste/rapide (via liste) :

```
{ "title": "2plus2", "taskRequest": "combien vaut 2+2 ?", "taskResponse": "4", "state": "todo" }
{ "title": "5moins4", "taskRequest": "combien vaut 5-4 ?", "taskResponse": "", "state": "todo" }
{ "title": "3fois0", "taskRequest": "combien vaut 3*0 ?", "taskResponse": "", "state": "todo" }
```

Solution 2 plus perfectionnée (via tableau) :

title	state	taskRequest	taskResponse
2plus2	todo	combien vaut 2+2 ?	4
5moins4	todo	combien vaut 5-4 ?	

avec peut être besoin d'une propriété (en input()) *mainFieldNames* pour contrôler les noms des champs/colonnes à afficher , par exemple ["title","state","taskRequest","taskResponse"] ou ["title","state"]

Vérification :

Quelque soit la version (plus ou moins évoluée) du composant generic-selector , il faut que le composant parent (ici TaskCrudComponent) puisse récupérer la valeur de la sélection (en output) .

Etant donné que le sous composant app-generic-selector est paramétré par

```
(selectedEntityEvent)="onSelectedEntity($event)"
```

on peut alors considéré que la fonction événementielle *onSelectedEntity()* sera automatiquement appelée dans le composant parent (TaskCrudComponent) dès que la sélection est effectuée.

A l'intérieur de la méthode onSelectedEntity(taskOfEvent:object){}

on pourra mettre à jour this.selectedTask et afficher la nouvelle valeur de this.selectedTask dans la console ou bien temporairement dans le template html de TaskCrudComponent .

---

Type d'informations à normalement visualiser dans la console du navigateur :

*GenericCrudComponent* , *received SelectEntity* event,  
*changed selectedTask*={*"title": "5moins4",*  
*"taskRequest": "combien vaut 5-4 ?", "taskResponse": "", "state": "todo"}*

### 3.3. V2 du sous composant generic-selector avec model()

NB : avant de commencer la version 2 , on pourra éventuellement sauvegarder le code de la v1 dans des copies de fichiers :

*generic-selector-v1.component.html.txt* , *generic-selector-v1.component.ts.txt*  
*task-crud.component-v1.html.txt* , *task-crud.component-v1.ts.txt*

Se débrouiller pour que le composant generic-selector puisse maintenant être incorporé dans un parent avec au minimum ce type de paramétrages :

```
<app-generic-selector  
  [entities]="taskList()"   
  [(selectedEntity)]="selectedTask" />
```

et éventuellement certains paramétrages secondaires tels que ;

```
<app-generic-selector  
  [entities]="taskList()"  [(selectedEntity)]="selectedTask"  
  [mainFieldNames]="mainTaskFieldNames"  
  (selectedEntityChange)="onSelectedTaskChange()" />
```

NB: la syntaxe [(xxx)]="...." n'est envisageable que si xxx est déclaré en tant que **model(...)** au sein du sous composant GenericSelectorComponent .

NB: Si dans cette nouvelle version, selectedEntity est déclaré avec **model()** plutôt que output() cela signifie que l'information **selectedEntity** est véhiculable dans les deux sens et que l'on peut expérimenter une sélection du premier élément qui est potentiellement contrôlable par le composant parent TaskCrudComponent .

Vérification (V2) :

Etant donné que le sous composant app-generic-selector est paramétré par

```
[(selectedEntity)]="selectedTask"
```

on peut alors considéré que l'information **selectedTask** est automatiquement mise à jour dans le composant parent (TaskCrudComponent) dès que la sélection est effectuée.

Si selectedTask a été déclaré dans TaskCrudComponent en tant que signal (via signal() ou model() ) alors on peut placer un effect dans la classe TaskCrudComponent pour afficher dans la console la nouvelle valeur de *selectedTask* dès qu'elle change.

On peut également/alternativement ajuster <app-generic-selector

```
.... [(selectedEntity)]="selectedTask"
```

**(selectedEntityChange)="onSelectedTaskChange()" />** pour que la nouvelle méthode *onSelectedTaskChange()* soit automatiquement appelée coté TaskCrudComponent dès que la valeur de selectedEntity (déclarée en tant que model()) changera dans le sous composant GenericSelector .

Type d'informations à normalement visualiser dans la console du navigateur :

*GenericCrudComponent* , *changed selectedTask*={"title":"5moins4","taskRequest":"combien vaut 5-4 ?", "taskResponse":"","state":"todo"}

Expérimentation temporaire d'une sélection du premier élément :

En ajoutant temporairement de code dans TaskCrudComponent

```
ngOnInit(){
  const firstTask = this.initialTaskList[0];
  this.selectedTask.set(firstTask)
  //this.tempTask.set(cloneTask(this.selectedTask(!)))
}
```

On devrait avoir une sélection du premier élément (ici explicitement demandée par le composant parent) lorsque l'on réinitialise l'application angular (refresh).

Bien que pas souvent utile , cette fonctionnalité montre la valeur ajoutée de model() vis à vis de output() . L'autre impact est dans certains changements de syntaxes :

Avec selectedEntityEvent = <b>output</b> <object>();	Avec selectedEntity = <b>model</b> <object null>();
(selectedEntityEvent)="onSelectedEntity(\$event)"	[(selectedEntity)]= "selectedTask" (selectedEntityChange)="onSelectedTaskChange()"

Désactiver/commenter la méthode ngOnInit() avant de passer à l'étape suivante du TP :

### 3.4. Sous composant task-editor

Se débrouiller en Tp pour que le sous composant TaskEditorComposant ait à peu près ces fonctionnalités :

title: 2plus2  
request: combien vaut 2+2 ?  
response:

avec un paramétrage en input de type <app-task-editor [(task)]= "selectedTask"> avec task déclaré de type model<Task>() .

### 3.5. contrôle des changements de valeurs

Avec ou sans signaux , l'éditeur de tâche (et sa partie formulaire) va agir sur un objet javascript qui peut être :

- soit **directement l'objet sélectionné (passé par référence)**
- soit **une copie de l'objet sélectionné (construit par clonage)**

**Attention :** dans une version simpliste (sans copie/clonage), on s'aperçoit que dès que l'on change la valeur d'un champ de saisie dans le formulaire (ex : réponse modifiée de "" à "4") , cette valeur modifiée s'affiche immédiatement au sein du tableau de sélection (dans l'autre composant GenericSelector) alors que l'on a même pas pris le temps de valider la modif en cliquant sur le bouton "DO" .

**Attention** : le framework angular considère que la valeur d'un signal a changé que si la valeur principale a été changée (soit une valeur primitive de type "string" , "number" , "boolean" , etc , soit une instance d'une classe a été remplacée par une autre instance) . **Si un objet javascript ordinaire (pas immutable) n'a été modifié que une de ses sous parties alors le signal associé à l'ensemble de l'objet javascript est considéré comme inchangé et par conséquent tous les "effect() " et autre écoutes associées ne seront pas déclenchés .**

Exemple (dans TaskEditorComponent):

```
onDoTask(){
  this.task()!.state="done"; //task instance not changed , model signal not changed !!!
  //this.task.set(this.task()) //model signal not changed , same instance !!!
  this.task.set(cloneTask(this.task(!)))
}
```

Pour un bon comportement vis à vis des changements de valeurs bien contrôlés on pourra par exemple :

- ajouter les fonctions **cloneTask()** et **copyValuesOfTask()** dans src/app/task/**task.ts**

```
export function cloneTask(t:Task){
  return new Task(t.title,t.taskRequest,t.taskResponse,t.state);
}

export function copyValuesOfTask(origin:Task,existingTarget:Task){
  existingTarget.state=origin.state;  existingTarget.taskResponse=origin.taskResponse;
  existingTarget.title=origin.title;   existingTarget.taskRequest=origin.taskRequest;
}
```

- et ajouter dans TaskCrudComponent (.html , .ts) :

```
<!-- <app-task-editor [(task)]="selectedTask" /> -->
<app-task-editor [(task)]="tempTask"
  (taskChange)="onTaskChanged()" />
```

```
tempTask = signal<Task|null>(null);
onSelectedTaskChange(){ this.tempTask.set(cloneTask(this.selectedTask(!))) }
onTaskChanged(){ copyValuesOfTask(this.tempTask(!),this.selectedTask(!)) }
```

Suite facultative/possible du TP :

Se débrouiller pour ajouter un bouton "DELETE" quelque part et avec un comportement cohérent à tous les niveaux.