

Enoncés des TDs et TPs / Angular (v ≥ 17)

1. Préparations et installations

Eventuelle installation de nodeJs et npm (si nécessaire):

Si **node -v** et **npm -v** se sont pas des commandes reconnues (dans un terminal CMD ou autre) alors télécharger et installer **nodeJs** (pour windows 64 bits ou autre en version LTS). Relancer ensuite un nouveau terminal et lancer **npm -v** pour vérifier.

Eventuelle installation de typescript (si nécessaire):

Si **tsc -v** est une commande inconnue (dans un terminal), alors lancer la commande suivante :

npm install -g typescript

Installation de angular-cli :

Si **ng -help** est une commande non reconnue , alors lancer la commande suivante :

npm install -g @angular/cli

NB : par défaut , c'est la dernière version stable de @angular/cli qui sera téléchargée.

Si l'on souhaite utiliser une ancienne version, il faut :

- éventuellement désinstaller une autre version (**npm uninstall -g @angular/cli**)

- préciser la version souhaitée : **npm install -g @angular/cli@19.2.0**

Installer si nécessaire l'IDE **Visual Studio code**.

2. TD "création d'un projet Angular"

Dans c:/.../tp-angular ou ailleurs on pourrait éventuellement lancer la commande

```
ng new --no-standalone my-app-with-module
```

mais il vaut mieux lancer la commande suivante :

```
ng new my-app
```

- l'option **--no-standalone** n'est disponible qu'au sein des versions récentes (17, 18 , 19, ...) et permet d'obtenir la même structure "classique" de l'application créée qu'au sein des versions antérieures d'angular (6,...,15,...)
- choisir **"scss"** via **saas** comme format de feuilles de styles
- choisir plutôt **"no"** à l'éventuelle question **"enable SSR"** car cette extension peut être ajoutée plus tard (si besoin)

NB: La commande **ng new my-app** met généralement **beaucoup de temps à s'exécuter** et elle **sollicite beaucoup de réseau (nombreux téléchargements)**. Dans certains cas (heureusement très rares) , il faut lancer la commande une seconde fois si la première tentative n'a pas fonctionné.

Se placer dans le répertoire my-app (cd my-app)

Lancer la commande

```
ng serve -o
```

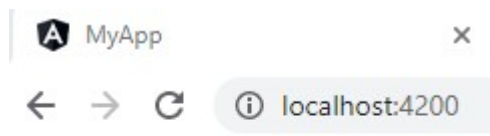
Vérifier le fonctionnement initial de l'application construite via l'url <http://localhost:4200> à charger dans un navigateur .

Via l'éditeur **Visual Studio code** , ouvrir le "folder" **c:...\tp-angular\my-app**

Modifier le fichier **src/app/app.component.html** en supprimant tout l'ancien contenu et en y plaçant à la place

```
<p> welcome to {{title}} </p>
<!-- <router-outlet></router-outlet> -->
```

Vérifier les changements (<http://localhost:4200>)



welcome to my-app

NB: En phase de développement, le mini serveur démarré par **ng serve** régénère automatiquement une nouvelle version à jour de l'application dès que le code .ts ou .html de l'application est modifié. Cependant , certains bugs temporaires ou **certaines modifications importantes de l'application** (configuration, restructuration de composants, ...) **nécessitent quelquefois un redémarrage de ng serve.**

Pour arrêter ng serve avant de le redémarrer, il faut se placer dans le terminal (souvent dédié) où ng serve a été préalablement lancé et taper **Ctrl-C**

Visualiser (au sein de l'éditeur "Visual Studio Code") les fichiers importants suivants (pour comprendre la structure du code source de l'application):

- **my-app/package.json** (repérer les dépendances d'angular en version 17, 18 ou autres)
- **my-app/src/index.html** (repérer la balise `<app-root></app-root>`)
- **my-app/src/app/app.component.ts** (repérer le sélecteur "app-root")

3. TD "création d'un arbre de composants angular"

Ce TD va montrer comment **créer de nouveaux composants "angular"** et les **rattacher entre eux**. Revenir (si besoin) sur le projet **my-standalone-app** (angular) via un *open folder* de *visual-studio-code* et relancer (si besoin) **ng serve** au sein d'un terminal (nouveau ou pas).

Au sein d'un **nouveau terminal** de visual-studio-code lancer (dans l'ordre) les commandes suivantes:

- **ng g component header**
- **ng g component footer**
- **ng g component basic**

se placer dans **src/app/basic** (via **cd**) et lancer les commandes suivantes

- **ng g component calculatrice**
- **ng g component tva**

Lire les messages affichés par la commande **ng g component ...** et visualiser (au sein de visual studio code):

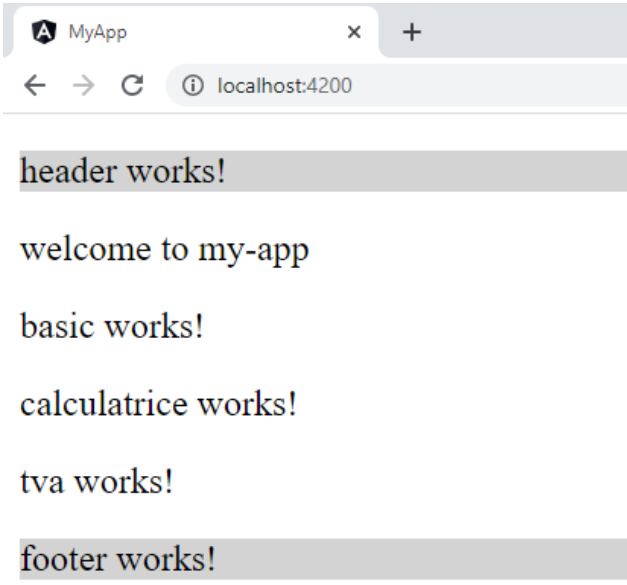
- les nouveaux répertoires créés (dans **src/app**)
- les nouveaux fichiers créés (.ts , .scss , .html, -spec.ts)

Repérer dans **src/app/header/header.component.ts** la valeur **app-header** du sélecteur

Ajouter les sous composants `<app-header></...>` `<app-basic></...>` `<app-footer></...>` dans **src/app/app.component.html**

De la même façon **ajouter** les **sous-sous-composants** *calculatrice* et *tva* dans **src/app/basic/basic.component.html**

Modifier éventuellement certaines couleurs de fond (via .scss) pour bien distinguer les sous-composants.

Tester via http://localhost:4200	Etats des principaux fichiers à ce stade
	<pre>src/app/header/header.component.html <p class="entete">header works!</p> src/app/header/header.component.css .entete { background-color: lightgrey;} src/app/basic/basic.component.html <p>basic works!</p> <app-calculatrice></app-calculatrice> <app-tva></app-tva> src/app/app.component.html <app-header></app-header> <p> welcome to {{title}} </p> <!-- <router-outlet></router-outlet> --> <app-basic></app-basic> <app-footer></app-footer></pre>

NB: en mode standalone, il faudra prévoir des choses de type `imports: [RouterOutlet , HeaderComponent , FooterComponent]` au sein de `@Component()` .

4. TD "calculatrice angular"

Revenir (si besoin) sur le projet **my-standalone-app** (angular) via un *open folder* de *visual-studio-code*

Relancer (si besoin) **ng serve** au sein d'un terminal (nouveau ou pas)

Ouvrir le fichier **src/app/basic/calculatrice/calculatrice.ts** (créé lors d'un TD antérieur) et y ajouter le code suivant:

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-calculatrice',
  imports: [FormsModule],
  templateUrl: './calculatrice.component.html',
  styleUrls: ['./calculatrice.component.scss']
})
export class CalculatriceComponent {

  a : number = 0;
  b : number = 0;
  res : number = 0;

  montrerHisto : boolean = true;
  historiqueCalculs :string[] = [];

  onCalculer(op:string){
    switch(op){
      case "+" :
        this.res = Number(this.a) + Number(this.b); break;
      case "-" :
        this.res = Number(this.a)- Number(this.b); break;
      case "*" :
        this.res = Number(this.a) * Number(this.b); break;
      default:
        this.res = 0;
    }
    this.historiqueCalculs.push(`${this.a} ${op} ${this.b} = ${this.res}`)
  }

  //coordonnées relatives de la souris qui survole une div
  x:number=0;
  y:number=0;

  onMouseMove(evt : MouseEvent){
    let currentDiv : HTMLElement = <HTMLElement> evt.target;
    this.x = evt.pageX - currentDiv.offsetLeft;
    this.y = evt.pageY - currentDiv.offsetTop;
  }

  onMouseLeave(evt : MouseEvent){
    this.x=0; this.y=0;
  }

  constructor() { }
}
```

NB: ajouter si besoin **imports: [FormsModule]** dans **@Component({standalone:true, ...})**

Ouvrir le fichier `src/app/basic/calculatrice/calculatrice.html` et y ajouter le code suivant:

```
<div class="c1">
  <h3>calculatrice angular</h3>

  <label>a :</label> <input type="number" [(ngModel)]="a" /> <br/>
  <label>b :</label> <input type="number" [(ngModel)]="b" /> <br/>
  <label>operation :</label>
    <input type="button" value="+" (click)="onCalculer('+')" />
    &nbsp; <input type="button" value="-" (click)="onCalculer('-')" />
    &nbsp; <input type="button" value="*" (click)="onCalculer('*')" />
  <br/>
  <label>resultat:</label>
  <span [style.font-weight]="res>0?'bold':'normal'"
    [class.negatif]="res<0" >
    {{res}}</span> <br/>
  <input type="checkbox" [(ngModel)]="montrerHisto" /> voir l'historique des
calculs<br/>
  @if(montrerHisto){
    <ul>
      @for(h of historiqueCalculs; track h){
        <li>{{h}}</li>
      }
    </ul>
  }
  <div class="c2" (mousemove)="onMouseMove($event)"
    (mouseout)="onMouseLeave($event)">
    Zone à survoler à la souris .<br/>
    x={{x}} , y={{y}}
  </div>
</div>
```

Ouvrir le fichier `src/app/basic/calculatrice/calculatrice.scss` et y ajouter le code suivant:

```
.c1 { background-color: rgb(244, 252, 142)}
.c2 { background-color: rgb(178, 235, 252)}
li { font-style: italic;}
label { display: inline-block; width: 6em;}
.negatif { color : red; font-style: italic;}
```

`ngModel` est une **directive d'attribut prédéfinie** dans le module `FormsModule`. Ainsi, l'utilisation de `[(ngModel)]` nécessite l'importation de `FormsModule` dans `app.module.ts` ou bien dans `@Component()` en mode standalone :

```
import { FormsModule } from '@angular/forms';
....

@NgModule_ou_@Component({
  imports:      [ ... , FormsModule ],
  ...
})
export class AppModule_ou_XyzComponent { }
```

Attention : sans ajout de `FormsModule` dans la partie `imports: []` la syntaxe `[(ngModel)]="..."` ne fonctionne pas !!! .

Faire fonctionner le code du composant ci-dessus (<http://localhost:4200>) pour bien comprendre son comportement.

calculatrice angular

a :

b :

operation :

resultat: -2

☒ voir l'historique des calculs

- $6 + 8 = 14$
- $6 * 8 = 48$
- $6 - 8 = -2$

Zone à survoler à la souris .

x=204 , y=34

- Analyser chaque ligne de code de l'exemple ci dessus (coté .ts, .html , .scss)
- Effectuer éventuellement des ajustements pour voir comment ça réagit
- Le Tp qui va suivre ressemblera partiellement à cet exemple

5. Tp "calcul de tva avec listes déroulantes"

Consignes du Tp :

- **coder de A à Z le composant src/app/basic/tva**
- il faudra calculer (via l'appel d'une seule fonction) les montants **tva** et **ttc** à partir d'un montant **ht** saisi et d'un **taux** de tva sélectionné .
- le **taux** de tva (5.0 , 10.0 ou 20.0) sera **choisi** à travers une **liste déroulante**.
- après une première version temporaire où le déclenchement du calcul sera effectué via un bouton poussoir, on codera une version améliorée (sans bouton) où la méthode qui réactualise les valeurs "tva" et "ttc" à calculer sera déclenchée via les traitements d'événements suivants:
 - **(input)="..."** sur **zone de saisie** du montant "ht"
 - **(change)="..."** sur **changement de sélection** du taux .

Rappel: l'événement **input** est déclenché sur une zone de saisie lorsque le texte saisi a changé (après avoir tapé un nouveau caractère).

résultats attendus:

Calcul de tva

ht: 200

taux: 5 ▼

tva= 10

ttc= 210

Suite facultative du TP :

Ajouter du côté *tva.component.ts* une map (table d'association) entre taux de tva et liste des principales catégories de produits associés à ce taux.

Initialiser cette map avec quelques valeurs au sein du constructeur.

```
mapTauxCategorieProd= new Map<number,string[]>();
tauxSel : number | undefined = undefined; //taux sélectionné
listeCategoriePourTauxSel = [];
constructor() {
  this.mapTauxCategorieProd.set(20 , [ "services" , "outils" , "objets"]);
  this.mapTauxCategorieProd.set(10 , [ "transports" , "hotels" , "restaurants" , "spectacles" , "médicaments"]);
  this.mapTauxCategorieProd.set(5 , [ "aliments" , "énergies" , "livres" ]);
}
```

Du côté *tva.component.html* , ajouter (en bas) un affichage des différents taux de tva sous forme d'une liste de puces (avec @for() comportant) .

Via un traitement d'événement adéquat, faire ensuite en sorte qu'en cliquant sur un des taux de tva :

- ça actualise (en dessous ou à côté) l'affichage de la liste des catégories de produits associés .
- ce taux de tva sélectionné soit affiché avec une classe de style css différent (pour le mettre en évidence). On pourra pour cela utiliser la syntaxe [class.nomClasseCss]='conditionPourActivation'.

principaux taux de tva: principales catégories pour le taux sélectionné:

- 5% **aliments,énergies,livres**
- 10%
- 20%

6. TP utilisation de "pipe" pour la mise en forme

Au sein de *tva.component.html* , on ajoutera l'utilisation d'au moins un pipe (tel que | **number : "..."**) de manière à ce que les résultats des calculs de tva soient toujours affichés avec 2 chiffres au maximum après la virgule.

Après avoir utilisé le pipe prédéfini "number" d'angular (qui a un comportement moyen d'un point de vue "affichage équivoque des grandes valeurs avec séparateur entre millier et centaine ressemblant à une virgule française") , on pourra se programmer un pipe personnalisé "**toFixed**" qui n'affichera pas de séparateur entre milliers et centaines .

Résultats attendus :

Sans pipe	Avec number:'1.0-2'	Avec toFixed:'2'
ht: <input type="text" value="123456.7897"/> tauxTva(%): <input type="text" value="20%"/> tva= 24691.35794 ttc= 148148.14763999998	ht: <input type="text" value="123456.7897"/> tauxTva(%): <input type="text" value="20%"/> tva= 24,691.36 ttc= 148,148.15	ht: <input type="text" value="123456.7897"/> tauxTva(%): <input type="text" value="20%"/> tva= 24691.36 ttc= 148148.15

Indications :

en mode standalone , besoin d'importer DecimalPipe dans @Component .

créer si besoin le sous répertoire src/app/**common/pipe**

ng g pipe toFixed

7. ajustements du composant tva avec signaux

Sauvegarder des copies du composant tva (sans signaux) :

tva.component.html.v1.txt

tva.component.ts.v1.txt

Améliorer le code de tva.component.ts et tva.component.html en s'appuyant sur des signaux.

Indications :

- Les éléments à saisir (ex : ht) ou à choisir (ex : taux) peuvent être déclarés comme des signaux ordinaires (via signal()) .
- Les éléments calculés (ex : tva , ttc) peuvent être matérialisés comme des signaux calculés (via computed())
- la partie tauxPossibles = [5, 10, 20] peut rester simple/ordinaire (sans signal)

8. TD routing/navigation élémentaire

Ce **TD très important** va nous permettre de mettre en place de "routing angular". Grâce à cela, les composants "basic" , "welcome" et "login" ne s'afficheront pas tous en même temps mais à tour de rôle selon les choix effectués dans un **menu**.

Revenir (si besoin) sur le projet **my-app** (angular) via un *open folder* de *visual-studio-code* et relancer (si besoin) **ng serve** au sein d'un terminal (nouveau ou pas)

Générer un nouveau composant "welcome" (WelcomeComponent) de premier niveau en lançant la commande suivante au sein d'un terminal (*en étant placé , via cd, dans src/app ou bien à la racine du projet my-app*):

```
ng g component welcome
```

Générer également (si besoin) un composant "login" :

```
ng g component login
```

Paramétrer quelques routes simples dans `src/app/app.route.ts` (anciennement `src/app/app-routing.module.ts`) (ex: `ngr-welcome/WelcomeComponent`, `ngr-basic/BasicComponent`, `ngr-login/LoginComponent`):

```
import { Routes } from '@angular/router';
import { WelcomeComponent } from '../welcome/welcome.component';
import { LoginComponent } from '../login/login.component';
import { BasicComponent } from '../basic/basic.component';

export const routes: Routes = [
  { path: 'ngr-welcome', component: WelcomeComponent },
  { path: '', redirectTo: '/ngr-welcome', pathMatch: 'full' },
  { path: 'ngr-login', component: LoginComponent },
  { path: 'ngr-basic', component: BasicComponent },
  { path: '**', redirectTo: '/ngr-welcome', pathMatch: 'full' }
];

//NB: les path peuvent éventuellement commencer par "ngr-" .
//ceci permet en production, un éventuel filtrage d'url à un niveau
//intermédiaire (ex: nginx)
```

NB: certaines lignes de type `import { ... } from '...'` peuvent quelquefois être générées par Visual-Studio-Code après un click sur une erreur et un click sur une ampoule jaune ou bleu .

(ré-)intégrer le routage de premier niveau dans `src/app/app.component.html` : en remplaçant

```
<app-basic></app-basic>
```

par

```
<router-outlet></router-outlet>
```

Ajouter quelques liens hypertextes dans `src/app/header/header.component.html` de type ` welcome `;

```
<p class="entete">header
  <a routerLink='/ngr-welcome'> welcome</a> &nbsp;
  <a routerLink='/ngr-basic'> basic</a> &nbsp;
  <a routerLink='/ngr-login'> login</a> &nbsp;
</p>
```

NB: en mode standalone, il faudra prévoir `imports: [RouterOutlet, ...]` dans `app.component.ts` et `imports: [RouterLink, ...]` dans `header.component.ts`

Tester le tout (<http://localhost:4200>) s

9. TD formulaire de login (template-driven)

Le framework Angular offre **deux possibilités** pour **effectuer des contrôles de saisies** au sein des **formulaires**:

- une **approche** dite **"template-driven"** où la plupart des **paramétrages** sont effectués du **côté ".html"** avec des syntaxes normalisées HTML5
- une **approche** dite **"model-driven"** où la plupart des **paramétrages** sont effectués du **côté ".ts"** (d'une manière très spécifique à angular)

Le formulaire de login sera basée sur l'approche "template-driven" à travers ce TD.

Un TP ultérieur permettra la mise en œuvre le l'approche model-driven/reactive-form .

Attention : sans ajout de FormsModule dans la partie imports: [] de @NgModule de app.module.ts , la gestion des formulaire ne fonctionne pas !!! .

Pour **bien organiser le code** , créer *dans src/app* un **nouveau "folder"** **src/app/common** et un **sous répertoire** **src/app/common/data**

Après s'être placé dans **src/app/common/data** (via cd), générer une nouvelle classe de données **Login** via la commande **ng g class login**

Au sein de la classe **Login**, **ajouter 3 attributs** de type **string** nommés **username** , **password** et **roles** .

src/app/common/data/login.ts

```
export class Login {
  constructor(
    public username : string = "",
    public password : string = "",
    public roles : string = "" ){}
}
```

Au sein de la classe **LoginComponent** , **ajouter un attribut** **login** de type **Login** et ayant = **new Login()** comme valeur par défaut.

Ajouter également dans LoginComponent (coté .ts):

- une propriété **message** de type **string**
- une méthode **onLogin()** qui dans un premier temps se contentera de mettre à jour le message en fonction des valeurs saisies cotés .html :

src/app/login/login.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Login } from '../common/data/login'

@Component({
  selector: 'app-login',
  imports: [FormsModule],
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss']
})
export class LoginComponent {

  public login : Login = new Login();
  public message /* :string */ = "";
  public onLogin(){
    this.message = "donnees saisies = " + JSON.stringify(this.login);
  }
}
```

```

}

constructor() { }
}

```

Dans **LoginComponent.html** ajouter de quoi saisir username, password et roles avec des [(ngModel)] paramétrés sur **login.username** etc :

Ajouter également un bouton poussoir **login** déclenchant **onLogin()**:

src/app/login/login.component.html

```

<h3>login</h3>
<label>username:</label>
  <input [(ngModel)]="login.username" /> <br/>
<label>password:</label>
  <input [(ngModel)]="login.password" /> <br/>
<label>roles:</label>
  <input [(ngModel)]="login.roles" /> <br/>
<input type="button" value="login" (click)="onLogin()" /> <br/>
message: {{message}}

```

Lancer **ng serve** dans un terminal libre de *Visual Studio Code*

Visualiser un résultat de ce type dans un navigateur internet (<http://localhost:4200>):

login

username:

password:

roles:

message: donnees saisies = {"username":"user1","password":"pwduser1",

NB: Le framework **angular**, analyse les **attributs/propriétés** (selon syntaxes normalisées **HTML5**) des **éléments d'un formulaire** (du côté template HTML) et **affecte automatiquement certaines classes de styles** selon les **circonstances suivantes** :

Etat	flag (booléen)	Css class si true	Css class si false
Champ visité (souris entrée et sortie)	touched	ng-touched	ng-untouched
Valeur du champ modifiée	dirty	ng-dirty	ng-pristine
Valeur du champ valide	valid	ng-valid	ng-invalid

NB:

- Le framework **angular** **active ou désactive automatiquement** les classes de styles ng-valid , ng-invalid , etc dont les noms sont prédéfinis (convenus à l'avance) au niveau des champs d'un formulaire .
- C'est néanmoins au **développeur** que revient le soin d'**associer une mise en forme souhaitée** à ces **classes de styles** dans le fichier global **src/styles.scss** ou bien dans le fichier **...component.scss** .

A faire en TD: ajouter les lignes suivantes dans **src/styles.scss** ou ailleurs:

```

input.ng-valid {
  border-left: 5px solid #42A948; /* green */
}

input.ng-invalid {
  border-left: 5px solid #a94442; /* red */
}

```

Rappels des principales contraintes de saisies d'HTML5:

- **required** : le champ est requis (valeur obligatoire)
- **minlength="3"** : il faut saisir au moins 3 caractères
- **pattern="^[a-zA-Z].+"** : la valeur saisie doit correspondre à une expression régulière (ici ça doit commencer par un caractère alphabétique puis contenir au moins un autre caractère quelconque)
- ...

Ajouter les contraintes de saisies suivantes dans `src/app/login/login.component.html`

```

<h3>login</h3>
<label>username:</label>
  <input [(ngModel)]="login.username" required pattern="^[a-zA-Z].+" />
  <br/>
<label>password:</label>
  <input [(ngModel)]="login.password" required minlength="3"/>
  <br/>
<label>roles:</label>
  <input [(ngModel)]="login.roles" required />
  <br/>
<input type="button" value="login" (click)="onLogin()" />
  <br/>
message: {{message}}

```

Tester le nouveau comportement avec un navigateur:

login

username: 3aa

password: pw

roles:

login

message:

login

username:

password:

roles:

On va encore **peaufiner le formulaire** en faisant en sorte que **le bouton "login" reste grisé (disabled) tant que l'ensemble des champs de saisies ne sont pas tous valides.**

Il faut pour cela , englober les champs du formulaire via `<form #formXy="ngForm" >` et `</form>`

Le principal intérêt d'écrire explicitement `<form #formXy="ngForm" >` est de pouvoir écrire plus bas :

```
<input type="button ou submit" value="..." (click)="..."
      [disabled]="!formXy.form.valid" />
<!-- déclencheur d'action grisé tant que formulaire pas globalement valide -->
```

NB: Chaque champ du formulaire doit absolument avoir un nom de renseigné (via **name="..."**) dès qu'il se trouve encadré par `<form ...>` et `</form>`. Sinon : erreur du côté ng serve (ou du côté console du navigateur) .

A faire en TD:

Ajouter tous les éléments vus précédemment dans `src/app/login/login.component.html`

```
<h3>login</h3>
<form #formLogin="ngForm">
  <label>username:</label>
  <input [(ngModel)]="login.username" name="username"
    required pattern="[a-zA-Z].+" />
  <br/>
  <label>password:</label>
  <input [(ngModel)]="login.password" name="password"
    type="password" required minlength="3"/>
  <br/>
  <label>roles:</label>
  <input [(ngModel)]="login.roles" name="roles" required />
  <br/>
  <input type="button" value="login" (click)="onLogin()"
    [disabled]="!formLogin.form.valid" />
  <br/>
</form>
message: {{message}}
```

Bien tester bien le tout (via <http://localhost:4200>):

login

username: a
password:
roles: admin

message:

login

username: admin1
password:
roles: admin

message: donnees saisies = {"username":"admin1","password":"pwdadm

10. TD facultatif : intégration de bootstrap-css

10.1. Intégration de bootstrap-css dans angular

Lancer les commandes suivantes au sein d'un terminal (en étant placé à la racine de l'application angular: là où est package.json):

```
npm install --save bootstrap  
npm install --save bootstrap-icons
```

Visualiser les dépendances ajoutées dans package.json:

```
...  
"bootstrap": "^5.3.3",  
"bootstrap-icons": "^1.11.3",  
...
```

Ajouter les chemins qui mènent à `./node_modules/bootstrap/dist/css/bootstrap.min.css` et autres dans la partie **styles** de la partie **architect/build/options** de **angular.json** (près de la ligne 31):

```
"styles": [
  "src/styles.scss" ,
  "node_modules/bootstrap/dist/css/bootstrap.min.css",
  "node_modules/bootstrap-icons/font/bootstrap-icons.css"
]
```

Attention à ne pas introduire d'erreur en plaçant mal une virgule .

10.2. Test de l'intégration bootstrap/angular:

Effectuer des ajouts de classes "bootstrap" et "fontawesome" dans src/app/login/login.component.html en adaptant si besoin cet exemple:

```
...
<button class="btn btn-primary" (click)="onLogin()"
  [disabled]="!formLogin.form.valid" >
  login <span class="badge badge-primary">
    <i class=""bi bi-person"></i>
  </span>
</button>
...
```

Arrêter (si besoin) et relancer **ng serve** et visualiser les résultats via <http://localhost:4200>



10.3. Ajustements "container-fluid" ou "container":

NB: Le fait d'intégrer "bootstrap css" à l'application Angular fait que malheureusement les marges sur les cotés sont peut être disparues.

Pour les rétablir on peut ajouter **<div class="container-fluid">** et **</div>** dans le composant principal src/app/app.component.html :

```
<div class="container-fluid">
  <app-header></app-header>
  ...
  <app-footer></app-footer>
</div>
```

10.4. Formulaire responsive avec champs bien alignés:

On va maintenant terminer ce TD en **ajoutant** les classes de styles de bootstrap css (ex: "row", "col-sm-4", ...) au sein de **login.component.html** pour obtenir un formulaire **responsive** avec des **champs bien alignés**:

```
<h3>login</h3>
<form #formLogin="ngForm">
  <div class="form-group row">
```

```

<label class="col-sm-4">username:</label>
<div class="col-sm-8">
  <input [(ngModel)]="login.username" name="username"
    class="form-control" required pattern="[a-zA-Z].+" />
</div>
</div>
<div class="form-group row">
  <label class="col-sm-4">password:</label>
  <div class="col-sm-8">
    <input [(ngModel)]="login.password" name="password"
      type="password" class="form-control" required minlength="3"/>
  </div>
</div>
<div class="form-group row">
  <label class="col-sm-4">roles:</label>
  <div class="col-sm-8">
    <input [(ngModel)]="login.roles" class="form-control"
      name="roles" required />
  </div>
</div>
<div class="form-group row">
  <div class="col-sm-8 offset-4">
    <button class="btn btn-primary" (click)="onLogin()"
      [disabled]="!formLogin.form.valid" > login </button>
  </div>
</div>
</form>
message: {{ message }}

```

```

input.ng-valid.form-control { width:100%; margin-top: 2px; border-left: 5px solid #42A948; }
input.ng-invalid.form-control { width:100%; margin-top: 2px ; border-left: 5px solid #a94442; }

```

username:	<input type="text" value="user1"/>
password:	<input type="password" value="pwd1"/>
roles:	<input type="text" value="user"/>
<input type="button" value="submit/login"/>	

11. TD alternatif facultatif : custom css (flexbox)

Ne pas utiliser bootstrap-css mais se construire des classes de styles basés sur les "flexbox" de manière à obtenir un bon comportement "responsive" au niveau des formulaires.
On pourra facultativement s'appuyer sur des directives personnalisées.

Exemples :

styles.scss

```

.f-form-group-wrap {
  display: flex;
  flex-flow: row wrap;
}

```



```
label.f-align {
  width: 12em;
  display: inline-block;
  margin-left: 0.3em;
  flex: 0 0 12em;
}
```

```
.f-align {
  margin-left: 0.3em;
  flex: 1 1 auto;
}
```

```
.f-max-size {
  width: 98%;
}
```

```
<form #formLogin="ngForm">
  <div class="f-form-group-wrap">
    <label class="f-align">username:</label>
    <div class="f-align">
      <input [(ngModel)]="login.username" class="f-max-size"
        name="username" required pattern="^[a-zA-Z].+"/>
    </div>
  </div>
  ...
```

12. Tp (facultatif) reactive-form

- Générer un nouveau composant "reservation" via ng component ...
- ajouter ce nouveau composant dans la liste des routes du app-routing.module.ts ou app.routes.ts et au sein du menu de navigation (dans header.component.html)

Après s'être placé dans src/app/common/data (via cd), générer une nouvelle classe de données *Login* via la commande **ng g class reservation**

Coder la classe **Reservation** comme ceci :

src/app/common/data/reservation.ts

```
export class Reservation {
  constructor(
    public firstName : string ="prenom",
    public lastName : string ="Nom",
    public telephone : string ="0605040302",
    public email : string ="prenom.Nom@xyz.com" ,
    public dateTime : Date = new Date()){}
```

Coder les cotés .ts et .html de ReservationComponent

reservation

firstName:	<input type="text"/>	firstName is required
lastName:	<input type="text" value="Nom"/>	
telephone:	<input type="text" value="060504030"/>	telephone.minLength=10
email:	<input type="text" value="prenom.Nomxyz.com"/>	email should be a valid email with ...@...
date:	<input type="text" value="2024-10-17"/>	
time:	<input type="text" value="11:0"/>	time HH:MM

13. TP input() sur composant header

Ajouter (via **input()**) un titre paramétrable au sein du composant HeaderComponent

Afficher la valeur de `{{titre()}}` dans `header.component.html`

Au sein de `app.component.html`, donner une valeur fixe au titre paramétrable de `<app-header>`

Tester le tout

Donner ensuite une valeur dynamique au titre paramétrable de `<app-header>` en demandant une copie de la valeur de `AppComponent.title`.

Tester le nouveau comportement.

`my-app` `welcome` `basic`

welcome



footer , date=13/09/2024

14. TP simpleSelector via output() puis model()

- Générer un nouveau composant appelé "*SimpleSelector*" et intégrer le quelque part (par exemple au sein de *basic.component.html*)
- coder une première version de ce composant (avec input() et output()) de manière à ce que l'on puisse paramétrer :
 - * un *title* variable
 - * une liste variable *values* de valeurs (de type string[]) à choisiret que l'on puisse récupérer la valeur choisie sous la forme d'un événement *choix*

Exemple de look souhaité pour le composant "SimpleSelector" ici en fond gris:

choix couleur

- red
- **green**
- blue

couleurChoisie = green

Exemple d'utilisation souhaitée de ce composant depuis un parent :

```
<app-simple-selector  
  title="choix couleur"  
  [values]="[ 'red' , 'green', 'blue' ]"  
  (choix)="onChoixCouleur(.....)"></app-simple-selector> <br/>  
couleurChoisie = {{couleurChoisie}}
```

Suite facultative du Tp :

- Sauvegarder des copies de simple-selector.component.html et .ts (ex : .v1.txt)
- Puis au sein de simple-selector.component.ts , transformer *choix* de output() en **model()** de manière à ce que le parent puisse (s'il en a envie) fixer un choix par défaut (ex : 'green')
- rendre tout cela cohérent et tester le nouveau comportement (choix par défaut paramétrable) dans une version adaptée du composant parent (après une éventuelle copie de l'ancienne version en commentaires)

15. TD projection dans composant réutilisable

Préparer l'arborescence src/app/common/component et se placer dans ce répertoire .

ng g component toggle-panel

Coder le contenu du composant **toggle-panel** en s'inspirant du code du support de cours (chapitre

"composants et modules / approfondissements" et sous partie "projection d'éléments imbriquées").
avec `<ng-content></ng-content>` et avec ou sans l'utilisation de bootstrap-css

Ajuster le contenu de `basic.component.html` de façon à ce que les composants "calculatrice" et "tva" soient imbriqués dans des panneaux déployables/rétractables :

```
<app-toggle-panel title="calculatrice">
  <app-calculatrice></app-calculatrice>
</app-toggle-panel>

<app-toggle-panel title="calcul tva">
  <app-tva></app-tva>
</app-toggle-panel>
```

calculatrice ↑

calculatrice

a: 5

b: 6

+ - * /

res= 11

Zone à survoler à la souris .
x=0 , y=0

calcul tva ↓

16. TD (facultatif) : onglet de @angular/material

Intégration de l'extension @angular/material au sein du projet angular :

ng add @angular/material

>>> Choisir un des thèmes proposés et accepter les animations

Importation du module "MatTabsModule" (onglets de material) :

Ajouter dans `src/app/app.module.ts`

```
import {MatTabsModule} from '@angular/material/tabs';
et
imports: [
  ...,
  MatTabsModule,
  ],...
```

Utilisation possible au sein de `basic.component.html` :

```
<mat-tab-group>
  <mat-tab label="calculatrice">
    <app-calculatrice></app-calculatrice>
  </mat-tab>
  <mat-tab label="calcul tva">
```

```
<app-tva></app-tva>
</mat-tab>
</mat-tab-group>
<hr/>
```

calculatrice

calcul tva

calcul tva

ht: 0

tauxTva(%): 20% ▼

Un redémarrage de ng serve sera peut être nécessaire ...

17. TP (facultatif) directive "highlight-over"

Au sein du composant "welcome" , placer une série de 3 ou 4 petites images.

On va maintenant coder et utiliser une **directive personnalisée "border-over"** qui va automatiquement mettre en évidence une des images survolées (en ajoutant une bordure par défaut rouge ou bien d'une couleur paramétrable).

Indications :

créer si besoin le sous répertoire src/app/common/directive
ng g directive border-over

18. TD "PreferencesService" et injection,synchro

Ce TD permettra de mettre en place une première version simple d'un service commun pour partager des données "utilisateur".

Créer le sous **folder** `src/app/common/service` pour bien ranger les services.

Dans un terminal, se placer dans `src/app/common/service` (via `cd` ou bien le menu contextuel *open in integrated terminal* de *Visual-Studio-Code*) et générer un nouveau service via la commande suivante:

```
ng g service preferences
```

Au sein de `PreferencesService`, ajouter (en public) la propriété `couleurFondPreferee` (de type `string`) avec `'lightgrey'` comme valeur par défaut:

`src/app/common/service/preferences.service.ts`

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class PreferencesService {

  public couleurFondPreferee :string = 'lightgrey';

  constructor() { }
}
```

Injecter le service `PreferencesServices` (en public) **dans les constructeurs** de `HeaderComponent` et `FooterComponent`.

NB: Si le Tp est effectué avec une version récente d'angular, on pourra préférer effectuer une injection avec la fonction `inject()` :

```
public preferencesService = inject(PreferencesService) ;
```

Faire en suite en sorte que l'on puisse changer la valeur de `preferencesService.couleurFondPreferee` via une liste déroulante au sein de `FooterComponent`:

`src/app/footer/footer.component.ts` :

```
import { Component, inject } from '@angular/core';
import { PreferencesService } from '../common/service/preferences.service';
import { FormsModule } from '@angular/forms';

@Component({
  selector: 'app-footer',
  imports: [FormsModule],
  templateUrl: './footer.component.html',
  styleUrls: ['./footer.component.scss']
})
export class FooterComponent {

  listeCouleurs : string[] = [ "lightyellow", "white",
    "lightgrey" , "lightgreen" , "lightpink" , "lightblue" ] ;

  //injection moderne via inject()
  public preferencesService = inject(PreferencesService) ;

  /*
```

```
//ancienne injection par constructeur
constructor(public preferencesService : PreferencesService) { }
*/
}
```

src/app/footer/footer.component.html :

```
<p class="piedPage">footer - couleurFondPreferee:
  <select [(ngModel)]="preferencesService.couleurFondPreferee" >
    @for(c of listeCouleurs; track c){
      <option [ngValue]="c">{{c}}</option>
    }
  </select>
</p>
```

Faire en suite en sorte que la nouvelle valeur choisie soit utilisée en tant que couleur de fond de HeaderComponent:

src/app/header/header.component.ts :

```
...
export class HeaderComponent implements OnInit {
  public preferencesService = inject(PreferencesService) ;
  ...
}
```

src/app/header/header.component.html :

```
<p class="entete"
  [style.background-color]="preferencesService.couleurFondPreferee">header
  <a routerLink='/ngr-welcome'> welcome</a> &nbsp;
  <a routerLink='/ngr-basic'> basic</a> &nbsp;
  <a routerLink='/ngr-login'> login</a> &nbsp;
</p>
```

Résultat attendu:

header welcome basic login

welcome to my-app

welcome works!

footer - couleurFondPreferee: lightgreen ▼

Changer de sélection de couleurFondPreferee dans la liste déroulante du pied de page. La couleur de fond de l'entête devrait normalement être bien synchronisée.

En étant positionné sur une couleur de fond autre que celle par défaut ('lightgrey') , effectuer un "refresh" de la page au sein du navigateur. La valeur de couleurFondPreferee est alors réinitialisée et reprend sa valeur par défaut.

Si l'on souhaite de pas perdre la valeur choisie de couleurFondPreferee au moment d'un "refresh" de la page au sein du navigateur, on peut améliorer le code du service de la façon suivante:

src/app/common/service/**preferences.service.ts**

```
import { Injectable } from '@angular/core';
//import { MyStorageUtilService } from './my-storage-util.service';

@Injectable({
  providedIn: 'root'
})
export class PreferencesService {

  //readonly myStorageUtilService = inject(MyStorageUtilService);

  private _couleurFondPreferee :string ;

  public get couleurFondPreferee(){
    return this._couleurFondPreferee;
  }

  public set couleurFondPreferee(c:string){
    this._couleurFondPreferee=c;
    localStorage.setItem('preferences.couleurFond', c);
    // this.myStorageUtilService.setItemInLocalStorage('preferences.couleurFond',c);
  }

  constructor() {
    //let c :string | null = this.myStorageUtilService.getItemInLocalStorage('preferences.couleurFond');
    let c = localStorage.getItem('preferences.couleurFond');
    this._couleurFondPreferee = c?c:'lightgrey';
  }
}
```

Attention (éventuel ajustement avec angular 17ou18 et en mode SSR) :

Si une application Angular récente est activée en mode SSR , il faut s'assurer que le code fonctionne bien coté navigateur (en appelant `isPlatformBrowser()`) de manière à pouvoir utiliser l'objet `localStorage` qui n'est disponible que du coté navigateur (et pas lors d'un pré-rendu coté serveur) :

Ce besoin étant récurrent, autant le coder dans un sous service réutilisable :

```
import { Injectable } from '@angular/core';
import { inject, PLATFORM_ID } from "@angular/core";
import { isPlatformBrowser, isPlatformServer } from "@angular/common";

@Injectable({
  providedIn: 'root'
})
export class MyStorageUtilService {
  private readonly platform = inject(PLATFORM_ID);

  public setItemInLocalStorage(key:string, value: string | null | undefined){
    if (isPlatformBrowser(this.platform)) {
      localStorage.setItem(key,value??'');
    }
  }

  public setItemInSessionStorage(key:string, value: string | null | undefined){
    if (isPlatformBrowser(this.platform)) {
```



```

    sessionStorage.setItem(key,value??'');
  }
}

public getItemInLocalStorage(key:string):string|null {
  return isPlatformBrowser(this.platform)?localStorage.getItem(key):null
}

public getItemInSessionStorage(key:string):string|null {
  return isPlatformBrowser(this.platform)?sessionStorage.getItem(key):null
}
constructor() { }
}

```

19. TD/TP simulation d'appels ajax avec Rxjs

Ce TD très important va nous permettre de mettre en place la structure classique d'un appel http/ajax asynchrone (dans un premier temps simulé) depuis une application angular.

19.1. Préparation de la classe de données "Devise"

Réouvrir le projet angular **my-app** dans l'éditeur "Visual-Studio-Code".

Créer (via new File ...) une **nouvelle classe** de données **Devise** dans **src/app/common/data** et comportant les propriétés **.code** (string) , **.name** (string) , **.change** (number)

src/app/common/data/devise.ts

```

export class Devise{
  constructor(public code:string="?",
               public name:string="?",
               public change:number=1){}
}

```

19.2. Préparation du Service "DeviseService"

Générer un nouveau service **DeviseService** dans **src/app/common/service** (et menu contextuel "open in integrated Terminal ...") via la commande

```
ng g service devise
```

Placer le code suivant dans **src/app/common/service/devise.service.ts** :

```

import { Injectable } from '@angular/core';
import { Devise } from '../data/devise';
import { Observable, of } from 'rxjs';
import { delay, map } from 'rxjs/operators';

@Injectable({
  providedIn: 'root'
})
export class DeviseService {

```

```

//jeux de données (en dur) pour pré-version (simulation asynchrone)
private devises : Devise[] = [
  new Devise('EUR','euro',1.0),
  new Devise('USD','dollar',1.1),
  new Devise('GBP','livre',0.9)
];

public getAllDevises$() : Observable<Devise[]>{
  return of(this.devises) //version préliminaire (cependant asynchrone)
    .pipe(
      delay(111) //simuler une attente de 111ms
    );
}

public convertir$(montant: number,
  codeDeviseSrc : string,
  codeDeviseTarget : string
) : Observable<number> {
  let coeff = (codeDeviseSrc==codeDeviseTarget)?1:Math.random();
  //coefficient aleatoire ici (simple simulation)
  let montantConverti = montant * coeff;
  if(montant < 0)
    return throwError( ()=>new Error("montant négatif invalide") )
  return of(montantConverti) //version temporaire (cependant asynchrone)
    .pipe(
      delay(222) //simuler une attente de 222ms
    );
}
}

```

19.3. Codage du composant "ConversionComponent"

Générer dans **src/app** (et menu contextuel "open in integrated Terminal ..."), un nouveau composant **ConversionComponent** via la commande

```
ng g component conversion
```

Ajouter la ligne suivante dans le tableau des **routes** de **src/app/app-routes.ts** ou bien **app-routing.module.ts**:

```
{ path: 'ngr-conversion', component: ConversionComponent }
```

Ajouter le lien hypertexte suivant dans **src/app/header/header.component.html**:

```
<a routerLink='/ngr-conversion'> conversion</a> &nbsp;   
```

Injecter le service **DeviseService** dans **ConversionComponent**.

Placer le code suivant dans **ConversionComponent** de façon à ce que l'on puisse effectuer **des conversions de monnaies**:

src/app/conversion/conversion.component.ts

```

import { Component, OnInit } from '@angular/core';
import { DeviseService } from '../common/service/devise.service'
import { Devise } from '../common/data/devise'

@Component({

```

```

selector: 'app-conversion',
imports: [FormsModule],
templateUrl: './conversion.component.html',
styleUrls: ['./conversion.component.scss']
})
export class ConversionComponent {

  montant: number = 0;
  codeDeviseSource: string = "?";
  codeDeviseCible: string = "?";
  montantConverti: number = 0;

  listeDevises : Devise[] = []; //à choisir dans liste déroulante.

  constructor(private _deviseService : DeviseService) { }

  onConvertir(){
    console.log("debut de onConvertir")
    this._deviseService.convertir$(this.montant,
                                   this.codeDeviseSource,
                                   this.codeDeviseCible)
      .subscribe({
        next : (res :number) => { this.montantConverti = res;
                                console.log("resultat obtenu en différé")} ,
        error : (err) => { console.log("error:"+err)}
      });
    console.log("suite immédiate (sans attente) de onConvertir");
    //Attention : sur cette ligne , le résultat n'est à ce stade pas encore connu
    //car appel asynchrone non bloquant et réponse ultérieure via callback
  }

  initListeDevises(tabDevises : Devise[]){
    this.listeDevises = tabDevises;
    if(tabDevises && tabDevises.length > 0){
      this.codeDeviseSource = tabDevises[0].code; //valeur par défaut
      this.codeDeviseCible = tabDevises[0].code; //valeur par défaut
    }
  }

  //ngOnInit() est automatiquement appelée par le framework après le constructeur
  //et après la prise en compte des injections et des éventuels @Input
  ngOnInit(){
    this._deviseService.getAllDevises$()
      .subscribe({
        next: (tabDev : Devise[])=>{ this.initListeDevises(tabDev); },
        error: (err) => { console.log("error:"+err)}
      });
  }
}

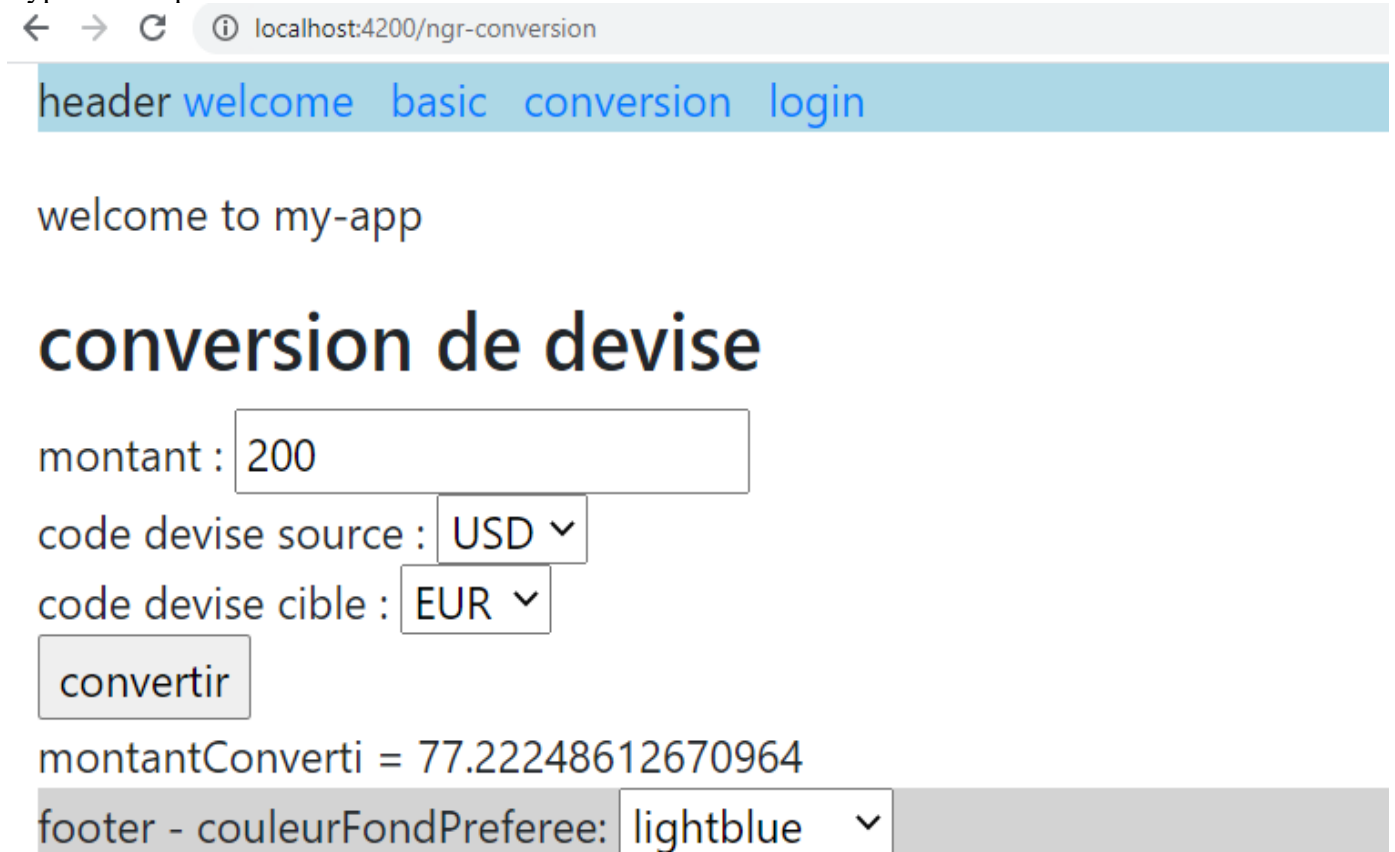
```

src/app/conversion/conversion.component.html

```
<h3>conversion de devise</h3>
montant : <input [(ngModel)]="montant" /> <br/>
code devise source : <select [(ngModel)]="codeDeviseSource" >
  @for(d of listeDevises; track d.code){
    <option>{{d.code}}</option>
  }
</select> <br/>
code devise cible : <select [(ngModel)]="codeDeviseCible" >
  @for(d of listeDevises; track d.code){
    <option>{{d.code}}</option>
  }
</select> <br/>
<input type="button" (click)="onConvertir()" value="convertir" /> <br/>
montantConverti = {{montantConverti}}
```

Lancer un test de l'application angular via la commande **ng serve** et un navigateur (<http://localhost:4200>).

Type de comportement attendu:



Ordre des affichages dans la console du navigateur:

debut de onConvertir	conversion.component.ts:22
suite immédiate (sans attente) de onConvertir	conversion.component.ts:31
resultat obtenu en différé	conversion.component.ts:28

19.4. TD possible (variante avec | async) :

Au sein de **conversion.component.ts** :

- ajouter **AsyncPipe** dans la partie imports:[...] de @Component()
- renommer onConvertir() en onConvertirV1()
- ajouter ceci à l'intérieur de la classe ConversionComponent

```
montantConvertiObservable! : Observable<number>

onConvertir() {
  this.montantConvertiObservable =
    this._deviseService.convertir$(this.montant,
                                    this.codeDeviseSource,
                                    this.codeDeviseCible)
  //à afficher coté .html via {{ montantConvertiObservable | async }}
  //ça nécessite imports:[...,AsyncPipe]
}
```

NB : la syntaxe typescript **variablexyz! : typeVariable** permet de se dispenser d'initialiser une variable d'instance (attribut de la classe) même en mode strict .

Au sein de **conversion.component.html**

remplacer

montantConverti = {{montantConverti}}

par

```
<!-- montantConverti = {{montantConverti}} -->
montantConverti = {{montantConvertiObservable | async }}
```

Version améliorée avec gestion d'erreur :

.smallError{ color:red; font-size: 10pt; margin-left: 6px; } dans .scss

{{message}} dans .html

et dans conversion.component.ts

```
message=""

onConvertir() {
  this.montantConvertiObservable =
    this._deviseService.convertir$(this.montant,
                                    this.codeDeviseSource,
                                    this.codeDeviseCible)
  .pipe(tap((resWithoutErr)=>{this.message=""}),
        catchError((err)=>this.handleError(err)));
}
private handleError(error: any): Observable<never> {
  this.message="erreur de conversion:" + error; //ou mieux
  console.log(this.message);
  //return throwError(()=> new Error('Error ...'));
  return throwError(()=> error);
}
```

→ à tester avec une saisie de montant négatif

NB: il est possible d'automatiser un peu plus la gestion d'erreur avec :

- la partie `.pipe(tap() , catchError())` du côté service
- un message d'erreur global (au sens dernière erreur) dans un service partagé
- un affichage global/factorisé du dernier message d'erreur dans un autre composant toujours visible (ex : partie de HeaderComponent)

19.5. TP facultatif (appel via `async/await` et `firstValueFrom`) :

- Dupliquer le code de la méthode `ngOnInit()`
- renommer `ngOnInitV1` un exemplaire de la versions actuelle (en commentaire ou pas)
- ré-écrire la méthode `ngOnInit()` en s'appuyant sur `async/await` et la fonction `firstValueFrom()` qui permet de transformer un Observable en Promise .

Style de code attendu:

```
async xyz() {  
  try {  
    let resXyz = await firstValueFrom(this._xxxService.zzzzz$());  
    ...  
  } catch (err) {  
    console.log(err);  
    ...  
  }  
}
```

20. TD appel Http/REST en mode GET via HttpClient

Ce TD va permettre de passer d'une version simulée à une version réelle du service **DeviseService** (effectuant en interne des appels **ajax/http** vers une **api REST**).

20.1. Vérification préliminaire le l'api REST à invoquer

Code source : <https://github.com/didier-mycontrib/tp-api>

URL de la version en ligne : <https://www.d-defrance.fr/tp/devise-api/...>

Tester tout particulièrement le comportement des appels suivants:

- <https://www.d-defrance.fr/tp/devise-api/v1/public/devises> devant normalement retourner
[{"name":"Dollar","change":1.109257,"code":"USD"},
{"name":"Yen","change":157.875088,"code":"JPY"},
{"name":"Livre","change":0.844698,"code":"GBP"},
{"name":"Euro","change":1,"code":"EUR"}]
- <https://www.d-defrance.fr/tp/devise-api/v1/public/convert?source=EUR&target=USD&amount=50> devant normalement retourner
{"amount":50,"source":"EUR","target":"USD","result":55.462849999999996}

20.2. Préparation indispensable de l'application angular

Si ancien mode no-standalone (ancienne application angular) :

Ajouter le module technique `HttpClientModule` dans la partie `imports:[...]` de `@NgModule` de `app.module.ts`

Dans application angular récente (en mode standalone) :

Ajouter **`provideHttpClient()`** dans `app.config.ts` (selon l'exemple du chapitre théorique).

20.3. Première version de l'appel ajax/http en mode get

Effectuer idéalement une copie de `src/app/common/service/devise.service.ts` en `src/app/common/service/devise.service.old.without_http.ts.txt` (ancienne version avec simulation sans appel HTTP).

Première version (à copier/coller) d'un code fonctionnel pour `src/app/common/service/devise.service.ts`:

```
import { Injectable } from '@angular/core';
import { Devise } from '../data/devise';
import { Observable, of } from 'rxjs';
import { map } from 'rxjs/operators';
import { HttpClient, HttpParams } from '@angular/common/http';

export interface ConvertRes {
  source :string; //ex: "EUR",
  target :string; //ex: "USD",
  amount :number; //ex: 200.0
  result :number; //ex: 217.3913
};

@Injectable({
  providedIn: 'root'
})
export class DeviseService {

  private _apiBaseUrl ="https://www.d-defrance.fr/tp/devise-api/v1";

  constructor(private _http : HttpClient){}

  public getAllDevises$() : Observable<Devise[]>{
    let url = this._apiBaseUrl + "/public/devises" ;
    console.log( "url = " + url);
    return this._http.get<Devise[]>(url);
  }

  public convertir$(montant: number,
                    codeDeviseSrc : string,
                    codeDeviseTarget : string
                    ) : Observable<number> {

    const url = this._apiBaseUrl + "/public/convert"
      + `?source=${codeDeviseSrc}`
      + `&target=${codeDeviseTarget}&amount=${montant}` ;
    //console.log( "url = " + url);
    return this._http.get<ConvertRes>(url)
      .pipe(
        map( (res:ConvertRes) => res.result)
      )
  }
}
```

```
}  
    }  
};
```

Résultats escomptés (après éventuel relance de **ng serve**) , <http://localhost:4200>

← → ↻ ⓘ localhost:4200/ngr-conversion

header - app - [welcome](#) basic conversion login

welcome to my-app

conversion de devise

montant : 200

code devise source : EUR

code devise cible : USD

convertir

montantConverti = 217.39130434782606

footer - couleurFondPreferee: lightblue

20.4. Configuration du reserve-proxy de ng serve (TD)

Nous allons maintenant appliquer concrètement le reverse-proxy de ng serve au niveau de notre application angular.

Créer et coder à la racine de l'application angular le nouveau fichier **proxy.conf.json** avec le contenu suivant :

```
{ "/tp/standalone-login-api" : { "secure" : false , "changeOrigin" : true,  
                                "target" : "https://www.d-defrance.fr/" } ,  
  "/tp/devise-api" : { "secure" : false , "changeOrigin" : true,  
                      "target" : "https://www.d-defrance.fr/" }  
}
```

Effectuer la modif suivante au sein de `src/app/common/service/devise.service.ts`:

```
//private _apiBaseUrl ="https://www.d-defrance.fr/tp/devise-api/v1";  
  
private _apiBaseUrl ="tp/devise-api/v1";  
// with prefix in proxy.conf.json  
// (ng serve --proxy-config proxy.conf.json)
```

```
// or other config in production mode
```

Repérer le message d'erreur suivant dans la console du navigateur lorsque l'on oublie de relancer ng serve avec l'option `--proxy-config proxy.conf.json` :

```
✖ ▶ GET http://localhost:4200/devise-api/public/devise 404
    (Not Found)
```

Arrêter si besoin ng serve (Ctrl-C) et relancer le via la commande suivante:

ng serve --proxy-config proxy.conf.json

Vérifier le bon fonctionnement de <http://localhost:4200/ngr-conversion>.

Astuce: Pour éviter d'oublier l'option `--proxy-config proxy.conf.json` au démarrage de **ng serve** on peut modifier la partie **serve** du fichier de configuration **angular.json** en y ajoutant l'option **"proxyConfig": "proxy.conf.json"** pour indirectement obtenir de manière automatique les mêmes fonctionnalités.

Configuration partielle de **angular.json** (après la ligne 72):

```
...
"serve": {
  "builder": "@angular-devkit/build-angular:dev-server",
  ...
  "options": {
    "proxyConfig": "proxy.conf.json"
  }
}
...
```

Avec cette configuration, un démarrage simple de **ng serve** (*sans option*) suffit et l'on bénéficie tout de même de la fonctionnalité "reverse proxy".

21. TP appel Http/REST en mode POST

Générer un nouveau service **LoginService** dans **src/app/common/service** en lançant la commande suivante en étant positionné au bon endroit dans un terminal:

ng g service login

Injecter ce service dans LoginComponent (du tp "validation formulaire")

Voici quelques copier/coller des résultats d'un test "PostMan" pour comprendre le fonctionnement du Web-service dont l'URL est <https://www.d-defrance.fr/tp/standalone-login-api/v1/public/auth> qu'il est prévu d'invoquer en mode **POST** avec **"Content-Type" : "application/json"** et avec les données *raw* suivantes:

```
{ "username" : "admin1" , "password" : "wrong-pwd" , "roles" :
"admin" }
```

ou bien

```
{ "username" : "admin1" , "password" : "pwd1" , "roles" :
"admin" }
```

POST ▼ <https://www.d-defrance.fr/tp/standalone-login-api/public/auth>

Params Authorization Headers (8) **Body** ● Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary **JSON** ▼

```
1 { "username": "admin1", "password": "wrong-pwd", "roles": "admin" }
```

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize **JSON** ▼

```
1 {
2   "username": "admin1",
3   "status": false,
4   "message": "login failed (wrong password)",
5   "token": null
6 }
```

POST ▼ <https://www.d-defrance.fr/tp/standalone-login-api/public/auth>

Params Authorization Headers (8) **Body** ● Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary **JSON** ▼

```
1 { "username": "admin1", "password": "pwd1", "roles": "admin" }
```

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize **JSON** ▼

```
1 {
2   "username": "admin1",
3   "status": true,
4   "message": "successful login",
5   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI2NmRjNjZiYTQ1NzQzMdAxNmYyNjgzZmUiLCJwcmVmZXJyZWRfdXNlcr3JpdGUgcmlvZmVzY2UuZGVsZXRLIiwiaWZ2L2Zw5fbmFtZSI6ImplYW4iLCJmYW1pbHIsIm1hdCI6MTcyNjQ0MTQ1MiwiZXhwIjoxNzI2NDg4NjUyLCJpc3MiOiJodHRwczovLjN6Fk5TXDjwUqc8_EUF2hNKzvZbK5IV6Fn0GDBhuerCY",
6   "scope": "resource.read resource.write resource.delete"
7 }
```

Coder ensuite (dans `src/app/common/data`) , une nouvelle classe **LoginResponse** correspondant à la structure de la réponse renvoyée par le web service d'authentification.

Coder ensuite la méthode suivante dans **LoginService**:

```
public postLogin$(login: Login): Observable<LoginResponse>
```

en s'appuyant sur `this._http.post<LoginResponse>(...)`

Appeler ensuite cette méthode **postLogin\$(...)** au sein de **doLogin()** de **LoginComponent** avec un **.subscribe()** et des **callbacks** sous forme de **lambdas/arrow-functions**.

Bien afficher le message (message positif ou message d'erreur, ...) à destination de l'utilisateur.

Afficher également toute la réponse reçue au format **JSON** dans les **logs** du navigateur.

Arrêter et relancer si besoin ng serve et **tester l'application**.

résultats attendus(<http://localhost:4200/ng-login>):

```
loginResponse={"username":"admin1","status":true,"message":"successful  
login","token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXZWQ","scope":"resource.read resource.write  
resource.delete"}
```

login

username:

password:

roles:

successful login

ou bien (si mauvais mot de passe):

login failed (wrong password)

22. TP facultatif "Devise CRUD"

Le Tp "Devise_ CRUD" est un Tp de synthèse facultatif (mais intéressant si on a un peu de temps à y consacrer). L'objectif de ce Tp est de réviser et approfondir (par l'expérience) les principales syntaxes liées au composants et services Angular (avec appels HTTP en mode GET/POST/PUT et DELETE).

22.1. Phase 1) récupération du point de départ du Tp:

1) copier/coller de pour_debut_tp vers votre projet my-app

les parties:

src/app/common/data/devise.ts

src/app/**devise** (tout ce répertoire avec **devise.component.ts** , **.html** , **.scss**)

2) ajouter DeviseComponent dans partie declarations: [] de src/app/app.module.ts
et import qui va avec

3) ajouter la route { path: 'ngr-devise', component: DeviseComponent }
dans src/app/app-routing-module.ts ou app-routes.ts et import qui va avec

4) ajouter devise (crud)
dans src/app/header/header.component.html

ng serve et http://localhost:4200

22.2. Phase 2) Version préliminaire sans accès HTTP

Au sein du fichier devise.component.ts récupéré (en tant que point de départ du Tp) :

- tabDevises correspond à une liste de devises en mémoire qu'il faut afficher dans le tableau html
- selectedDevise correspond à une référence sur une éventuelle sélection de devise (après click sur une des lignes du tableau)
- deviseTemp correspond à une copie (créée par clonage) de la devise sélectionnée.
deviseTemp sera affichée dans le formulaire de saisi et sera soit "réinitialisée" , "ajoutée" ,
"modifiée" ou "supprimée" en fonction du bouton poussoir activé (avec méthodes événementielles
onNew() , onAdd , onUpdate , onDelete())
- message est un message à construire à destination de l'utilisateur
- mode (vaut "existingOne" suite à une sélection de devise existante ou bien "newOne" suite à un
click sur "new") .

L'objectif de cette phase du Tp consiste à coder (par ajout des parties manquantes
sur .devise.component.ts et .html) un comportement "CRUD" complet où les valeurs sont pour
l'instant qu'actualisées en mémoire au niveau de this.tabDevises .

devise sélectionnée (newOne)

code:
name:
change:

new

add

Liste des devises (selon serveur)

code	name	change
GBP	Livre	0.844698
EUR	Euro	1
USD	Dollar	1.109257
JPY	Yen	157.875083

message: **devise bien mise à jour**

devise sélectionnée (existingOne)

code: JPY
name: Yen
change: 157.875083

update ↩

new

delete

Ordre d'amélioration conseillé:

- ajouter si besoin (click)="onSelectDevises(d)" sur <tr *ngFor ... >
- coder si besoin la méthode onSelectDevises()
- ajouter tous les [(ngModel)]="deviseTemp.xyz" manquants dans le côté html
- ajouter (click)="onUpdate()" sur bouton update
- coder méthode onUpdate() via des instructions de type this.selectedDevises.xyz = this.deviseTemp.xyz;
- tester tout cela
- coder le reste (new --> remet this.deviseTemp à vide (nouvelle instance de Devise)
(add --> ajoute this.deviseTemp (ou un clone) dans this.tabDevises)
(delete --> supprime this.selectedDevises du tableau this.tabDevises)
- peaufiner message , mise en évidence de l'élément sélectionné et boutons activables ou pas selon le contexte (ex: [style.visibility]="(mode==='newOne')?'visible':'hidden'" sur certains boutons du côté .html avec this.mode = "newOne" ou bien "existingOne" du côté .ts

22.3. Phase 3) Avec appels HTTP vers api REST

L'objectif de cette phase du Tp consiste à améliorer le code de la phase précédente pour qu'en arrière plan , les actions déclenchées soit synchronisées avec les données d'un serveur/backend REST (URL de base = <https://www.d-defrance.fr/tp/devise-api/v1>) .

URL "public" ou "private" spécifique à ce TP :

```
...
export class DeviseService {

  private _withoutSecurity = false;

  public set withoutSecurity(value:boolean){
    this._withoutSecurity=value;
    this.publicOrPrivateBaseUrl=this._withoutSecurity?this.publicBaseUrl:this.privateBaseUrl;
  }

  public get withoutSecurity():boolean{
    return this._withoutSecurity;
  }
}
```

```

_apiBaseUrl ="tp/devise-api/v1"; //with ng serve --proxy-config proxy.conf.json
publicBaseUrl = `${this._apiBaseUrl}/public`;
privateBaseUrl = `${this._apiBaseUrl}/private`;
publicOrPrivateBaseUrl : string =this.privateBaseUrl; //with security by default

...
}

```

On fera en sorte (au sein de deviseComponent) qu'une case à cocher puisse indirectement contrôler l'état de deviseService.withoutSecurity.

En mode "withoutSecurity" , tous est permis (POST,PUT,DELETE,GET) du coté serveur sans que cela nécessite une authentification/login préalable .

En mode "private" (par défaut sécurisé) , toute action de type POST/PUT/DELETE ne sera acceptée du coté serveur qu'après une authentification préalable (en mode admin , ex : username=admin1) .

NB : on effectuera les premiers tests en mode "withoutSecurity" et on peaufinera que plus tard les aspects sécurisés (authentification, retransmission du token via intercepteur,éventuelle variante en mode OAuth2 , ...).

Changements de code conseillés du coté devise.component.ts:

on s'appuie sur **constructor(private deviseService : DeviseService) {}**

avec une version plus complète (à améliorer) de src/app/common/service/**DeviseService**

avec méthodes du genre :

```

postDevise$(d :Devise): Observable<Devise>{
    const url = `${this.publicOrPrivateBaseUrl}/devises`;
    return this._http.post<Devise>(url,d /*input envoyé au serveur*/);
}

```

exemple (pour Add) dans devise.component.ts:

```

onAdd(){
    this.deviseService.postDevise$(this.deviseTemp)
        .subscribe(
            { next: (savedDevise)=>{ this.message="devise ajoutée";
              this.addClientSide(savedDevise); } ,
            error: (err)=>{ this.message = messageFromError(err,"echec post"); }
        );
}

addClientSide(savedDevise:Devise){
    this.tabDevises.push(savedDevise);
    this.onNew();
}

```

..à peaufiner .. et à compléter sur les autres modes GET/PUT/DELETE/...

23. TP prise en compte de l'authentification

Selon Tp facultatif précédent

Si le Tp "Devise_CRUD" a été effectué complètement (ce qui demande du temps) , on dispose alors déjà d'un composant permettant de modifier une devise.

Si par contre le TP "Devise_CRUD" n'a pas pu être effectué , on compensera alors ceci par un petit ajout de code dans le bas du composant "conversion" :

```
{ "headers": { "normalizedNames":  
{}, "lazyUpdate": null }, "status": 401, "statusText": "Unauthorized", "url": "http://localhost:4200/tp/devise-api/private/devise?v=true", "ok": false, "name": "HttpErrorResponse", "message": "Http failure response for http://localhost:4200/tp/devise-api/private/devise?v=true: 401 Unauthorized", "error": "Unauthorized" }
```

code devise a modifier: (ex: JPY , GBP , ...)

nouveau change: (ex: 0.9 , 1.1)

Via un code de ce genre :

deviseService	<pre>putDevise\$(d :Devise): Observable<Devise> { //const url = `\${this.publicOrPrivateBaseUrl}/devises/\${d.code}?v=true`; const url = `\${this._apiBaseUrl}/private/devises/\${d.code}?v=true`; return this._http.put<Devise>(url,d /*input envoyé au serveur*/); }</pre>
ConversionComponent.ts	<pre>message="" ; codeToUpdate="?" ; changeToUpdate=1 ; async onUpdate() { try { let d:Devise; let deviseTemp : Devise undefined; for(d of this.listeDevises){ if(d.code===this.codeToUpdate){ deviseTemp = JSON.parse(JSON.stringify(d)); } } if(deviseTemp===null) this.message="pas de devise pour ce code"; else { deviseTemp.change=this.changeToUpdate; await firstValueFrom(this._deviseService.putDevise\$(deviseTemp)); this.message="mise à jour ok"; } } catch (err) { console.log(err); this.message = <string> JSON.stringify(err); } }</pre>
ConversionComponent.html	<pre><hr/>{{message}} <hr/> code devise a modifier: <input [(ngModel)]="codeToUpdate" /> (ex: JPY , GBP)
 nouveau change: <input [(ngModel)]="changeToUpdate" /> (ex: 0.9 , 1.1)
 <button (click)="onUpdate()">update devise</button></pre>

23.1. TP Authentification en mode "standalone" (sans OAuth2)

Depuis le composant "devise" (crud en mode sécurisé) ou bien depuis le composant "conversion", un déclenchement direct d'une modification de devise (sans login préalable) devrait normalement aboutir au message d'erreur : **401/Unauthorized**.

Améliorer le code de **onLogin()** (au sein de **login.component.ts**)

en faisant en sorte que :

- à chaque nouvelle tentative de login, le contenu de "access_token" soit réinitialisé à "" au sein de sessionStorage du navigateur
- que le contenu de "access_token" au sein de **sessionStorage** soit remplacé par **loginResponse.token** à l'issue de la tentative de login

Créer un nouveau répertoire d'organisation **src/app/common/interceptor**

Se placer dedans (au sein d'un terminal texte) et lancer la commande suivante :

```
ng g interceptor my-auth
```

Ajuster le contenu de la fonction d' **interception** avec **paramètres(req,next)**

avec un code de ce genre :

```
//NB: "access_token" plutot que "token" or "authToken" for angular-oauth2-oidc extension compatibility
const token = sessionStorage.getItem('access_token');
if(token && token !== "" && token !== "null"){
  const authReq = req.clone({
    headers: req.headers.set('Authorization', 'Bearer ' + token)
  });
  console.log("MyAuthInterceptor , adding Bearer token="+token)
  return next(authReq); //ou bien next.handle(authReq) dans anciennes versions d'angular
}else
  return next(req); //ou bien next.handle(req) dans anciennes versions d'angular
```

Enregistrer cet intercepteur au sein de **src/app/app-config.ts** ou bien **app.module.ts**

selon les indications du support de cours théorique (fin chapitre HTTP/REST) , selon la version de angular utilisée et selon le contexte (standalone ou pas).

Exemple (pour version récente d'angular) au sein de **src/app/app-config.ts** :

```
...
import { provideHttpClient, withInterceptors } from '@angular/common/http';
import { myAuthInterceptor } from './common/interceptor/my-auth.interceptor';
export const appConfig: ApplicationConfig = {
  providers: [... , provideHttpClient( withInterceptors( [myAuthInterceptor] ) )
  ] };

```

Relancer éventuellement **ng serve** pour une meilleur prise en compte de ces changements.

Nouveau comportement attendu :

- Après un login en échec (ex : mauvais mot de passe) , on ne doit pas pouvoir modifier une devise (401/Unauthorized) . Cette erreur être peut être visible dans la console web du navigateur...
- Après un login réussi (avec compte admin1/pwd1) , on doit pouvoir modifier une devise sans erreur
- Après un login réussi (avec compte user1/pwd1) avec un compte n'ayant pas assez de droits/privilège , on ne doit pas pouvoir modifier une devise (403/Forbidden) . Cette erreur être peut être visible dans la console web du navigateur...

23.2. TD facultatif Authentification en mode OAuth2

En appliquant tout le mode opératoire du paragraphe "accès à un serveur d'autorisation depuis angular" du chapitre "Authentification au sein d' Angular" , on doit pouvoir mettre en place une seconde variante de l'authentification basée sur une délégation d'authentification vers un serveur d'autorisation (ex : "<https://www.d-defrance.fr/keycloak/realms/sandboxrealm>") .

```
npm install angular-oauth2-oidc --save
```

NB : Pour éviter toute confusion entre les versions "standalone" et "oauth2" du login , on pourra éventuellement utiliser les noms suivants :

- `src/silent-refresh.html`
- `src/app/common/data/UserInSession` (avec username, authenticated, roles et grantedScopes)
- `src/app/common/service/OAuth2SessionService` (avec utilisation de angular-oauth2-oidc et méthode `initOAuthServiceForCodeFlow()`)
- `src/app/oauth2-log-in-out (OAuth2LogInOutComponent)`
à créer via `ng g component oauth2LogInOut`

NB: La plupart des fichiers nécessaires se trouvent au sein du répertoire "pour_debut_tp" .
Quelques copier/coller peuvent éventuellement servir à gagner un peu de temps ...

Ajouter une copie de `silent-refresh.html` in `/src`
et ajouter "`src/silent-refresh.html`" dans le premier bloc assets de `angular.json` (près de la ligne 30 ou 40)

```
"assets": [  
  {  
    "glob": "**/*",  
    "input": "public"  
  },  
  "src/silent-refresh.html"  
],
```

En version "avec module" , ajouter `OAuthModule.forRoot()` dans imports :[] de `app.module.ts` avec import `{ OAuthModule } from 'angular-oauth2-oidc'`;

En version "standalone" , ajouter `provideOAuthClient()` dans providers:[] de `app.config.ts` avec import `{ provideOAuthClient } from 'angular-oauth2-oidc'`;

Navigations envisageables:

```
...  
{ path: "ngr-login" , component : LoginComponent},  
{ path: "ngr-oauth2-login-out" , component : OAuth2LogInOutComponent},  
{ path: "ngr-devise" , component : DeviseComponent},  
{ path: "ngr-conversion" , component : ConversionComponent},  
...
```

et

```
<a routerLink="/ngr-login">login (standalone)</a> &nbsp;    
<a routerLink="/ngr-oauth2-login-out">OAuth2 logInOut</a> &nbsp;    
<a routerLink="/ngr-conversion">conversion</a> &nbsp;  
```

```
<a routerLink="/ngr-devise">devise(crud)</a> &nbsp;
```

L'intercepteur est le même que celui déjà configuré en mode standalone .

Un arrêt/redémarrage de ng serve sera nécessaire .

En cas de soucis/bug (dans la communication avec le serveur OAuth2) une manip navigateur "Effacer les cookies et les données du site" peut souvent suffire à remettre les choses en place.

23.3. TP différé – authentification peaufinée avec gardien et message

On essaiera (en temps utile , au sein des futurs Tps facultatifs) de peaufiner les points suivants :

- nouveau service "sessionService" permettant de mémoriser des informations sur l'utilisateur connecté (soit via le login standalone , soit via le login oauth2)
- Affichage des informations sur la personne connectée (username , ...) au sein de footerComponent
- gardien bloquant la navigation vers "devise" (ou a défaut "conversion") lorsque l'utilisateur n'est pas connecté . NB : ce tp sera différé dans l'attente de la théorie sur les gardiens (chapitre "routing évolué").

24. TP facultatif sur BehaviorSubject ou signal

Technologie "BehaviorSubject" conseillée sur ancien projet angular .

Technologie "Signal" (plus moderne, plus performante) conseillée sur projet récent.

- (nouveau) service "sessionService" permettant de mémoriser des informations sur l'utilisateur connecté (soit via le login standalone , soit via le login oauth2)
- Ce service comportera des informations en mode "BehaviorSubject" ou bien "signal"
- Affichage d'un icône "connecté" ou pas "connecté" au sein du composant "header"
- Affichage des informations sur la personne connectée (username , ...) au sein de footerComponent
- Ces affichages seront automatiquement synchronisés via des ".subscribe()" associés au behaviorSubject ou bien via les automatismes des signaux

25. TP approfondissement du "routing angular"

25.1. Sous niveau de routes (en mode "children")

Le composant intermédiaire "basic" (comportant les sous composants "tva" et "calculatrice") a déjà été codé en 3 variantes :

1. avec <app-calculatrice> et <app-tva> toujours affichés
2. avec intégration dans le composant réutilisable <app-toggle-panel>
3. avec basculement via onglets dynamiques de l'extension material

On va maintenant mettre en œuvre une quatrième variante utilisant un sous-niveau "children : []" du routing angular et une nouvelle utilisation de <router-outlet>.

- ajouter <router-outlet></router-outlet> dans **basic.component.html**

et commenter éventuellement les anciens blocs `<app-toggle-panel>` ou `<mat-tab-group><mat-tab>...</>`

ou `<app-tva>` et `<app-calculatrice>`

- ajouter dans **basic.component.html**

```
<a routerLink="/ngr-basic/calculatrice" >calculatrice</a><br/>
<a routerLink="/ngr-basic/tva" >tva</a><br/>
```

NB : ajouter si besoin **RouterLink** , **RouterOutlet** dans la partie **imports:[]** de **BasicComponent** en version standalone

- ajuster un bloc de configuration de ce type dans **app-routes.ts** ou bien **app-routing.module.ts**:

```
{ path: 'ngr-basic', component: BasicComponent ,
  children: [
    { path: 'tva', component: TvaComponent },
    { path: 'calculatrice', component: ....Component },
    { path: '', redirectTo: 'tva', pathMatch: 'prefix' }
  ]
}
```

- tester le nouveau comportement de l'application

25.2. Avec paramètres en fin de routes

Seconde version (avec 2 modes sur calculatrice : "simple" et "sophistiquée"):

- ajouter (dans `calculatrice.component.ts`):

modeChoisi = "simple"; //ou "sophistiquée"

- ajouter (dans `calculatrice.component.html`) un paramétrage pour faire en sorte que le bouton poussoir "**multiplication**" ne soit affiché que si **modeChoisi=='sophistiquée'**

- ajuster les liens hypertextes suivants (`basic.component.html`):

```
<a routerLink="/ngr-basic/calculatrice/simple" >calculatrice simple</a> &nbsp; ;
<a routerLink="/ngr-basic/calculatrice/sophistiquée" >calculatrice sophistiquée</a><br/>
```

- ajuster la route imbriquée au sein `app-routing.module.ts`:

```
children: [ ...,
  { path: 'calculatrice/:mode', component: CalculatriceComponent }, ...
]
```

- ajouter et ajuster (dans `calculatrice.component.ts`):

```
constructor(route : ActivatedRoute) {
  //NB: { path: 'calculatrice/:mode', ... },
  route.params.subscribe(
    (params:Params)=> { this.modeChoisi = params['mode'];}
  );
}
```

- tester le nouveau comportement de l'application

25.3. Mettre en œuvre un gardien (Tp facultatif)

- Ce gardien devra bloquer une route tant que l'utilisateur n'est pas connecté
- On effectuera si besoin une redirection vers une page de login ou bien une page d'erreur expliquant le problème

Indications :

ng g component not-authorized

répertoire **src/app/common/guard**

ng g guard auth

choisir "canActivate"

26. TD début de déploiement d'une appli angular

26.1. Petit test en mode "code angular sur serveur http"

npm install -g http-server

ng build

ou bien

ng build --ssr=false --prerender=false (si extension @angular/ssr déjà activée)

http-server --proxy https://www.d-defrance.fr -c-1 dist/my-app/browser

ou bien

http-server --proxy https://www.d-defrance.fr -c-1 dist/my-app (selon version d'angular)

NB : Avec du temps et des moyens suffisants , on faire mieux avec **nginx + docker** .

26.2. Petit test (long et facultatif) en mode SSR/nodeJs

ng add @angular/ssr (si pas déjà fait)

Prise en compte du contexte via éventuels ajouts de *if (isPlatformBrowser(this.platform)) { ...}* si nécessaire

npm run build

npm run serve:ssr:my-app

NB : Avec du temps et des moyens suffisants , on peut intégrer cela dans un contexte **nginx + docker** .

27. TP et TD sur tests au sein d'angular

27.1. TD facultatif : test "end-to-end" avec cypress et angular

Ajustement du code "angular" pour faciliter l'écriture des tests :

- Au sein de `basic.component.html` , faire en sorte qu'il n'y ait qu'une seule version du composant calculatrice (par exemple idéalement via `<router-outlet>` et routing avec url en `/ngr-basic/calculatrice/...` ou bien `<app-calculatrice>` en direct mais pas les deux)
- Au sein de `calculatrice.component.html` ajouter si besoin :
 - `name="a"` sur la balise `<input [(ngModel)]="a" />`
 - `name="b"` sur la balise `<input [(ngModel)]="b" />`
 - `id="spanRes"` sur la balise `` englobant `{{res}}` sans espace

Installation de la technologie cypress dans le projet angular :

```
npm install --save-dev cypress
```

ou bien tout simplement `npm install` si est cypress déjà présent dans le fichier `package.json`

Attention : Prévoir de longues minutes de téléchargement ...

Réglages pour éviter un conflit avec angular :

Au sein d'un projet angular , il faut ajouter ceci à la fin du fichier `tsconfig.json` de manière à ce que l'installation de cypress ne perturbe pas les tests unitaires d'angular :

```
{
  ...
  "compilerOptions": { ....
  },
  "angularCompilerOptions": { ...
  },
  "exclude": [
    "cypress.config.ts",
    "cypress/**/*.ts"
  ]
}
```

Premier lancement de cypress :

```
npx cypress open
```

Remarque : Le premier lancement va permettre la création de `cypress.json` , du sous-répertoire cypress et de certains sous-répertoires avec quelques exemples

Après un lancement de cypress open , il faut en théorie choisir un test à lancer au sein de la fenêtre de sélection. Nous devons cependant écrire préalablement notre propre test. À arrêter (en fermant cette fenêtre et/ou "Ctrl-C") et à relancer ultérieurement.

Au sein du répertoire cypress/e2e , ajouter ce fichier :

myTest.spec.cy.js

```
//NB: il faut préalablement lancer ng-serve ou autre
describe('My Angular Tests', () => {
  it('good conversion', () => {

    //partir de index.html
    cy.visit("http://localhost:4200/index.html")

    //cliquer sur le lien comportant 'basic'
    cy.contains('basic').click()
    cy.wait(50)
    // Should be on a new URL which includes 'basic'
    cy.url().should('include', 'basic')

    cy.get('a[href="/ngr-basic/calculatrice/simple"]').click()

    // Get an input, type data into it
    //and verify that the value has been updated
    cy.get('input[name="a"]')
    .clear()
    .type('9')
    .should('have.value', '9')

    cy.get('input[name="b"]')
    .clear()
    .type('6')
    .should('have.value', '6')

    //declencher click sur bouton soustraction
    cy.get('input[type="button"][value="-"]')
    .click()

    //vérifier que la zone d'id spanRes comporte le texte '3'
    cy.get('#spanRes')
    .should('have.text', '3')

  })
})
```

Ajuster ci besoin le code de ce fichier pour garantir une bonne cohérence avec le code à tester.

* Vérifier que l'application angular est bien démarrée (ex : ng serve préalablement lancé).

* Relancer (si besoin) , **npx cypress open**

puis sélectionner le test à lancer (E2E testing , choix navigateur, choix du test)

27.2. Ajustement du code de TvaComponent pour préparer des tests unitaires cohérents

Au sein de `src/app/common/service`
générer un nouveau service de calcul de tva via
ng g service tva

Ajouter dans **TvaService** les méthodes `tva` et `ttc` suivantes :

```
tva(ht:number ,taux_tva_pct:number){
  return ht * taux_tva_pct/100.0
}

ttc(ht:number ,taux_tva_pct:number){
  return ht * (1 + taux_tva_pct/100.0)
}
```

Injecter ensuite **ServiceTva** au sein de **TvaComponent** (via constructor ou `inject()`)
Puis réajuster les calculs de tva au sein de **TvaComponent** en s'appuyant sur le service injecté.
Par exemple :

```
tvaService = inject(TvaService)

onCalculerTvaTtc(){
  this.tva = this.tvaService.tva(this.ht, this.tauxTvaPct);
  this.ttc = this.tvaService.ttc(this.ht, this.tauxTvaPct);
}
```

Du côté **tva.component.html** :

- ajouter **name="ht"** sur `<input .../>` à côté de `[(ngModel)]="ht"`
- ajouter **name="tauxTvaPct"** sur `<select .../>` ou `<input .../>` à côté de `[(ngModel)]="tauxTvaPct"`
- commenter (via `<!-- -->`) une copie temporaire des affichages avec des pipes (ex : `number` ou `toFixed`)
- englober les affichages de `{{tva}}` et `{{ttc}}` sans pipe par des balises `...`

Exemple :

```
tva: <span id="spanTva">{{tva}}</span> <br/>
ttc: <span id="spanTtc">{{ttc}}</span> <br/>
```

Tester ensuite le bon fonctionnement du composant tva (via `ng serve` , un navigateur internet et une navigation adéquate).

27.3. TP : test d'un service simple (TvaService)

Au sein du fichier **tva.service.spec.ts** :

- ajouter les méthodes de tests suivantes :

```
it('tva(200,20)==40', () => {
  expect(.....).toBe(40);
});
```

```
it('ttc(200,20)==240', () => {  
  expect(.....).toBe(240);  
});
```

Compléter le code et lancer le test unitaire en lançant la commande suivante :

```
ng test --include=**/tva.service.spec.ts
```

Corriger si besoin certaines erreurs de manière à obtenir un résultat de ce genre :

Karma v 6.4.4 - connected; test: complete;

Chrome 137.0.0.0 (Windows 10) is idle

 **Jasmine** 4.6.1

• • •

3 specs, 0 failures, randomized with seed 11818

TvaService
• tva(200,20)==40
• ttc(200,20)==240
• should be created

Ajouter temporairement une erreur dans le code de TvaService.tva (par exemple return -1)
et visualiser ce message d'erreur :

 **Jasmine** 4.6.1

✖ • •

3 specs, 1 failure, randomized with seed 26403

[Spec List](#) | Failures

TvaService > tva(200,20)==40

Expected 0 to be 40.

Restituer le bon code au sein de TvaService.tva() pour revenir au vert

Arrêter ng test via Ctrl-C

27.4. TP : test unitaire d'un composant simple (TvaComponent)

Ne rien ajouter dans `tva.component.spec.ts` (laisser dans l'état le code initialement généré par ng g c tva).

Lancer la commande suivante :

```
ng test --include=**/tva.component.spec.ts
```

Au sein de `tva.component.spec.ts` ajouter et mettre au point (par correction éventuelle) cette première méthode de test :

```
it('tva(200,20)=40 from model', () => {
  component.ht=200;
  component.tauxTvaPct=20;
  component.onCalculerTvaTtc(); //à ne pas oublier d'appeler si pas de dispatchEvent
  fixture.detectChanges();
  const compNativeElt = fixture.debugElement.nativeElement;
  let spanTvaElt = compNativeElt.querySelector('#spanTva');
  console.log("from model, tva:" + spanTvaElt.innerText);
  expect(Number(spanTvaElt.innerText)).toBeCloseTo(40,2);
  // .toBeCloseTo(expectedValue,precision_as_nb_decimal)
});
```

Seulement après avoir mis au point le test précédent, on pourra tenter un second test plus complet de ce genre :

```
it('tva(200,10)=220 from IHM', () => {
  //Saisies de valeurs (via native_elements and DOM api):
  const compNativeElt = fixture.debugElement.nativeElement;
  let htInputElt = compNativeElt.querySelector("input[name='ht']");
  htInputElt.value=200;
  htInputElt.dispatchEvent(new Event('input'));

  /* // Pour version simplifiée avec bouton et sans liste déroulante:
  let tauxTvaPctInputElt = compNativeElt.querySelector("input[name='tauxTvaPct']");
  tauxTvaPctInputElt.value=20;
  tauxTvaPctInputElt.dispatchEvent(new Event('input'));

  let calculButtonElt =
    compNativeElt.querySelector("input[type='button'][value='calculer']");
  //calculButtonElt.dispatchEvent(new Event('click'));
  calculButtonElt.click(); */

  // Pour version sans bouton et avec liste déroulante:
  let tauxTvaPctSelectElt = compNativeElt.querySelector("select[name='tauxTvaPct']");
  let optionElt = null;
  for(let opt of tauxTvaPctSelectElt.children){
    if(opt.innerText=="10%"){
      optionElt=opt;
    }
  }
  console.log("from ihm, optionElt.innerText:" + optionElt.innerText
    + ", optionElt.value:" + optionElt.value);
  tauxTvaPctSelectElt.value=optionElt.value;
  fixture.detectChanges();
  console.log("from ihm, tauxTvaPctSelectElt.value:" + tauxTvaPctSelectElt.value);
  tauxTvaPctSelectElt.dispatchEvent(new Event('change'));
```

```
fixture.detectChanges();
```

```
//Vérifications des valeurs saisies et calculées dans le modèle:
```

```
expect(Number(component.ht)).toBe(200);
```

```
expect(component.tauxTvaPct).toBe(10);
```

```
expect(component.ttc).toBeCloseTo(220,2);
```

```
//Vérifications des valeurs calculées dans la vue (template html):
```

```
let spanTtcElt = compNativeElt.querySelector('#spanTtc');
```

```
console.log("from IHM, res:" + spanTtcElt.innerText);
```

```
expect( Number( spanTtcElt.innerText ) ).toBeCloseTo(220,2);
```

```
});
```

```
});
```

Karma v 6.4.4 - connected; test: complete;

Chrome 137.0.0.0 (Windows 10) is idle

 **Jasmine** 4.6.1
• • •

3 specs, 0 failures, randomized with seed 46122

```
TvaComponent
  • should create
  • tva(200,20)=40 from model
  • tva(200,10)=220 from IHM
```

Dans console/terminal intégré à vscode :

✓ Browser application bundle generation complete.

LOG: 'from model, tva:40'

Chrome 137.0.0.0 (Windows 10): Executed 1 of 3 SUCCESS (0 secs / 0.025 secs)

LOG: 'from ihm, optionElt.innerText:10% , optionElt.value:1: 10'

Chrome 137.0.0.0 (Windows 10): Executed 2 of 3 SUCCESS (0 secs / 0.031 secs)

LOG: 'from ihm, tauxTvaPctSelectElt.value:1: 10'

Chrome 137.0.0.0 (Windows 10): Executed 2 of 3 SUCCESS (0 secs / 0.031 secs)

LOG: 'from IHM, res:220.00000000000003' ...

TOTAL: 3 SUCCESS

NB : s'il faut effectuer un test de CalculatriceComponent et si le constructeur de CalculatriceComponent a besoin du service ActivatedRoute , on est alors obligé d'ajouter un mock (implémentation fournie simplifiée de ActivatedRoute) via ce genre de paramétrage au sein de calculatrice.component.spec.ts :

```
beforeEach(async () => {
  await TestBed.configureTestingModule({
    imports: [CalculatriceComponent],
    providers: [ {
      provide: ActivatedRoute,
      useValue: { params: of({mode: 'simple'}) }
    } ]
  })
  .compileComponents();
```