

Angular (v \geq 17)

(avec rappels essentiels sur
javascript et typescript)

Table des matières

I - Présentation de Angular.....	5
1. Présentation du framework web Angular.....	5
2. Principaux éléments d'angular.....	13
3. Evolution importante d'angular en 2023/2024.....	13
II - Rappels sur javascript et typescript.....	15
1. Versions de javascript , bases fondamentales.....	15
2. Let, const , boucles for , arrow function.....	17
3. Template string , destructuration , spread operator.....	21
4. Promise es2015 et async/await.....	25
5. Fondamentaux "typescript".....	28
6. Classes , constructor , get/set , generic , interface.....	31
7. Notion de décorateurs "typescript".....	38
8. Modules "es6/es2015".....	39
9. WebComponent.....	44

III - Structure et fondamentaux Angular.....	49
1. Environnement de développement pour Angular.....	49
2. Anatomie élémentaire d'un composant angular.....	62
3. Structure arborescente d'une application Angular 2+.....	63
4. Arborescence de composants (TD).....	64
5. Structure d'une application angular 2+.....	66
6. Modules applicatifs (maintenant facultatifs).....	67
IV - Essentiel sur template, binding, event, pipe.....	72
1. Anatomie d'un composant angular.....	72
2. Templates bindings (property , event).....	75
3. Formatage des valeurs à afficher avec des "pipes".....	85
V - Code flow et signaux.....	88
1. Code flow (version >= 17).....	88
2. Signaux (version >= 16+).....	90
VI - Switch et routing essentiel (navigation).....	96
1. Switch élémentaire de sous composants.....	96
2. Bases élémentaires du routing angular.....	97
VII - Contrôles de formulaires.....	100
1. Contrôle des formulaires (template-driven).....	100
2. Formulaires en mode reactive-form.....	104
VIII - Components et modules (approfondissement).....	107
1. Précisions sur les composants (@Component).....	107
2. Cycle de vie sur composants (et directives).....	115
3. Aperçu sur les directives (angular2+).....	116
IX - Services et injections (essentiel).....	121
1. Services "angular" (concepts et bases).....	121
Autre type de service classique:.....	124
SessionService (avec données de type .username .isConnected ...).....	124
2. Injection de dépendances (bases).....	125
X - Programmation réactive et RxJs.....	128

1. introduction à RxJs.....	128
2. Fonctionnement (sources et consommations).....	129
3. Convention de nom pour "return Observable".....	129
4. Organisation de RxJs.....	130
5. Sources classiques générant des "Observables".....	131
6. Principaux opérateurs (à enchaîner via pipe).....	133
7. Hot Observable.....	135
8. Observable d'ordre 2.....	137
9. Passerelles entre "Observable" et "Promise".....	139
10. Optimisations et précautions.....	140
XI - Appels HTTP (vers api REST).....	141
1. Angular et dialogues HTTP/REST.....	141
2. Api HttpClient (depuis Angular 4.3).....	145
XII - Routing angular (compléments importants).....	153
1. Sous niveau de routage (children).....	153
2. Routes paramétrées et navigation par code.....	154
3. Route conditionnée par gardien.....	156
4. Aperçu sur le routing angular avancé.....	158
XIII - BehaviorSubject et synchronisations.....	161
1. BehaviorSubject (exitant avant les signaux).....	161
2. Autres aspects divers.....	167
XIV - Authentification au sein d' Angular.....	169
1. Sécurisation d'une application "angular".....	169
2. Token pour appels aux Web-services REST.....	170
3. Authentification via OAuth2/OIDC.....	172
XV - Tests unitaires (et ...) avec angular.....	178
1. Différent types de tests autour de angular.....	178
2. Test "end-to-end / cypress".....	179
3. Attention à la cohérence des tests.....	182
4. Tests unitaires élémentaires.....	182
5. Lancement tests "Angular" avec ng test et "karma".....	185
6. Tests unitaires "angular"(composants, service, ...).....	186

XVI - Packaging et déploiement d'appli. angular.....	205
1. Notion de "bundle" pour le déploiement.....	205
2. JIT vs AOT (Ahead-Of-Time) pour angular 2 à 8.....	206
3. ivy (à partir de angular 9).....	207
4. Vite.....	207
5. Mise en production d'une application angular.....	208
6. Application isomorphe et optimisation SSR.....	211
XVII - Annexe – composants avancés et animations.....	215
1. Composants "angular" avancés et animations.....	215
2. ng-content et ng-template.....	219
3. ViewChild / ContentChild.....	224
4. Animations (triggers).....	231
XVIII - Annexe – internationalisation angular (i18n).....	238
1. internationalisation (i18n).....	238
XIX - Annexe – extension @angular/material.....	244
1. Angular-Material (librairie de composants).....	244
2. Essentiel de "flex-layout" (en intégration angular).....	246
3. Quelques composants "angular-material".....	247
XX - Annexe – PWA (Progressive Web App).....	259
1. PWA (Progressive Web App) – aperçu général.....	259
2. Web App Manifest / add to home screen.....	262
3. Service-worker et pwa pour Angular.....	268
XXI - Annexe – mode déconnecté et IndexedDB.....	277
1. Mode "offLine" et indexed-db.....	277
2. IndexedDB et idb.....	278
XXII - Annexe – socket.io.....	283
1. Socket.io.....	283

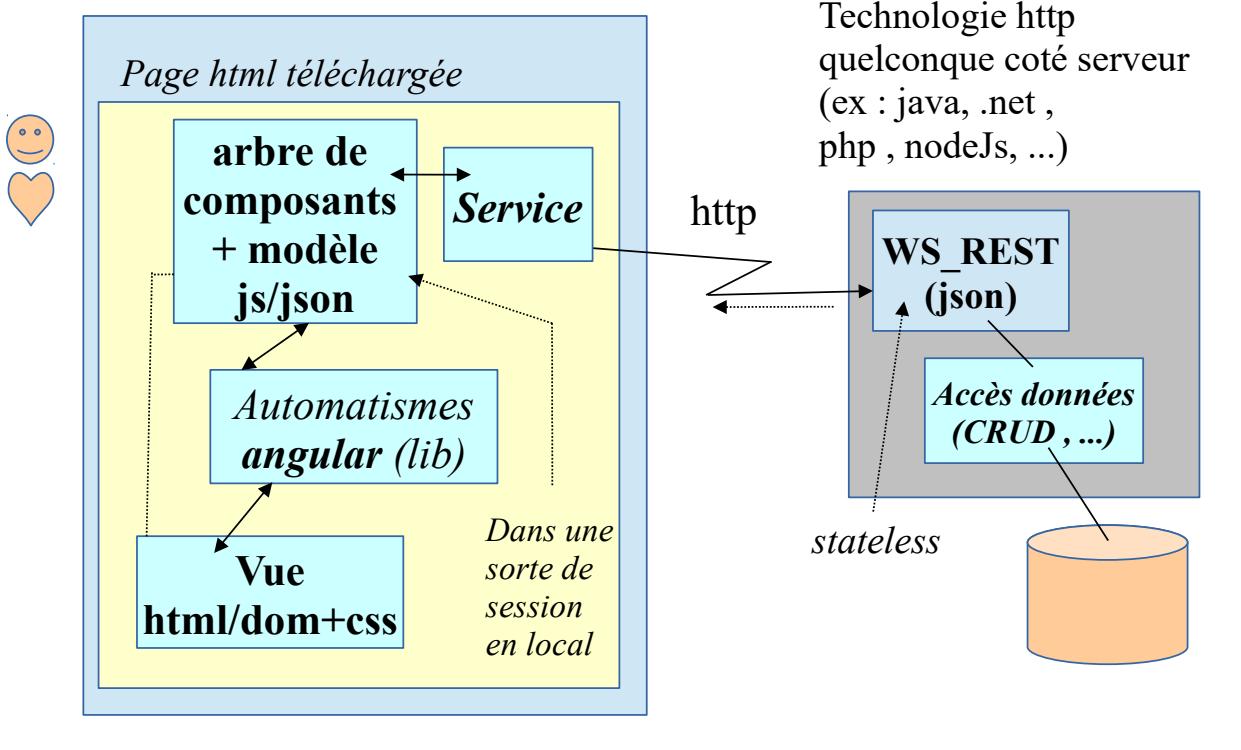
I - Présentation de Angular

1. Présentation du framework web Angular

1.1. Positionnement du framework "Angular"

Angular (positionnement)

Coté client (navigateur)



Angular est un **framework web de Google** qui s'exécute entièrement du coté navigateur et dont la programmation est basée sur le langage **typescript** (version fortement typée de **javascript/es6+**). La récupération de données s'effectue via des services (à programmer) et dont la responsabilité est souvent d'appeler des web services REST (transfert de données JSON via HTTP).

Les principaux intérêts de la technologie angular sont les suivants :

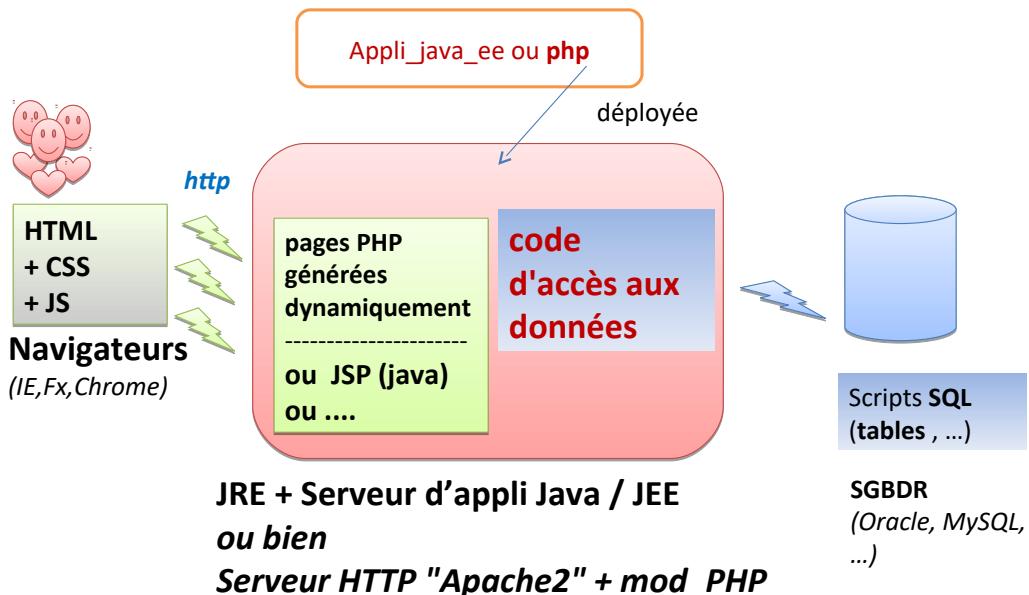
- une grande partie des traitements web s'effectue coté client (dans le navigateur) et le serveur se voit alors déchargé d'une lourde tâche (refabriquer des pages, gérer les sessions utilisateurs, ...) ---> bien pour tenir la charge .
- meilleures performances/réactivités du coté affichage/présentation web (navigateur) : c'est directement l'arbre DOM qui est réactualisé/rendu à partir des modifications apportées sur le modèle typescript/javascript (plus de html à transférer/ré-analyser).
- séparation claire entre la partie "présentation" (js) et la partie "services métiers" (java ou ".net" ou ".php" ou "nodejs" ou ...) . Google présente d'ailleurs parfois angularJs ou Angular2+ comme un framework MVW (Model-View-Whatever) .

1.2. Contexte architectural

Ancienne architecture web prédominante (années 1995-2015) :

Pages HTML générées coté serveur (ex : java/JEE , php, asp, ...) et sessions HTTP coté serveur .

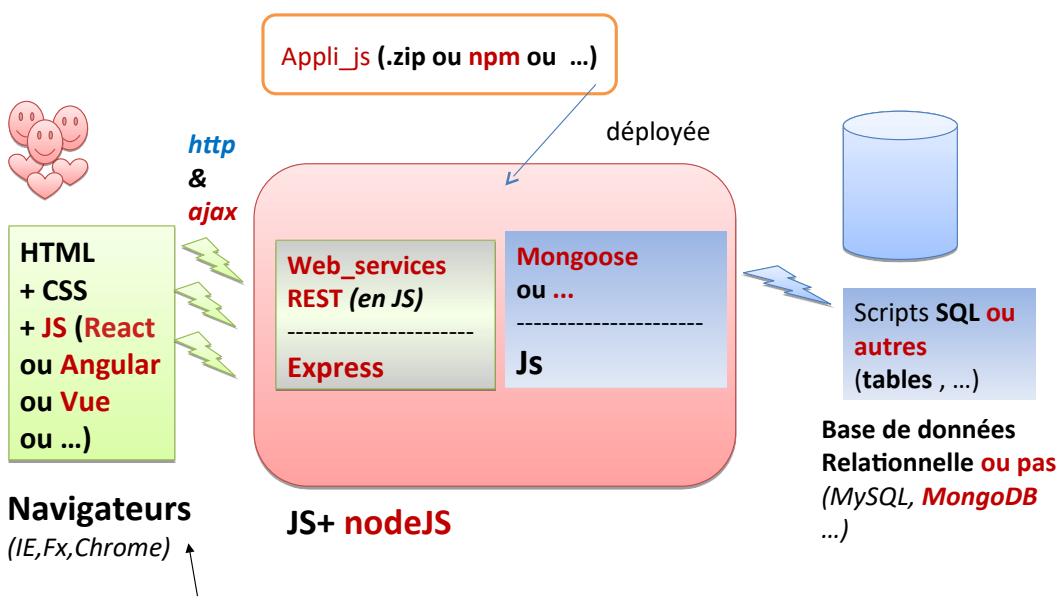
Ancienne architecture web



Nouvelle architecture web prédominante (depuis 2015 environ) :

Front-end (ex : Angular, VueJs , React, ...) en HTML5/CSS3/JS invoquant (via ajax) des Web services REST d'un "backend" serveur quelconque (nodeJs, php , java/JEE/spring , python, ...).

Env exécution NodeJs



souvent SPA (Single Page Application)

--> avantages : meilleures performances (si grand nombre de clients simultanés) et meilleur séparation front-end (affichage standard HTML5/CSS3) / back-end (api rest).

1.3. Evolution du framework "angular" (versions)

Evolution de angular (versions)

angularJs (1.x) 2012 - 2016 — javascript , contrôleur

angular 2 (fin 2016) — typescript , composants,
avec bugs

angular 4 et 5 (2017) — moins bugs , @angular/cli ,
http → httpClient

angular 6, 7 , 8 (2018, 2019) — RxJs et base angular
enfin stable

Angular 9, ... ,15 (2020,2023) — moteur ivy performant

Angular 17,...,19 (2023,2025) — standalone ,
signaux , ssr, ...

NB :

- L'ancienne version 1.x s'appelait **AngularJs**
- Depuis la v2 , le framework à été renommé **Angular** (sans js car typescript)
- La v2 comportait plein de bugs . la v3 n'a jamais existé .
- Les v4 et v5 étaient utilisables (sans bug) mais depuis certaines parties ont été grandement restructurées (http --> httpClient , rxjs , ...)
- **Le framework "angular" s'est enfin stabilisé à partir de la version 6** (les v7 et v8 apportent quelques améliorations sans grand chamboulement) .
- **A partir de la version 9 , le cœur interne d'angular a été refondu (moteur "ivy" plus performant)** et certains aspects avancés ont été améliorés (compacité du code , lazy loading enfin stabilisé , ...)
- **Les versions récentes d'angular (11, 12, 13, ...)** sont maintenant accompagnées d'un langage **typescript** configuré en **mode strict** . Ceci oblige à programmer avec plus de rigueur .
- Les versions **16 , 17 , 18, 19** ont apporté **beaucoup de nouveautés (facultatives)** (**standalone component, signaux , ssr**) qui vont dans le sens "*simplification + performances accrues*" **Certaines nouveautés de la version 17 (et stabilisées depuis la version 19) ont un gros impact sur la structure globale d'une application angular.**

1.4. Binding angular

Composant angular avec binding automatique (*)

(*) m-v-vm (proche mvc)
en javascript / navigateur

product.component.html

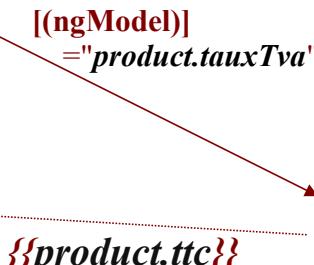
label:	<i>produit xyz</i>
prix ht:	<i>200</i>
tauxTva:	<i>20</i>
actualiser prix ttc	
prix ttc:	<i>240</i>

product.component.ts

```
@Component({ ... })
export class ProductComponent{
  onMajTtc(evt) { ... }
  product = new Product();
}
```

Product (class)

```
.label produit xyz
.ht 200
.tauxTva 20
.ttc 240
```



En s'étant inspiré du design pattern "**MVVM**" (*Model-View-ViewModel*) proche de **MVC**, le framework Angular gère automatiquement une mise à jour de la vue HTML qui s'affiche dans le navigateur en effectuant (quasi-automatiquement) des synchronisations par rapports aux valeurs d'un modèle orienté objet (compatible JSON) qui est géré en mémoire par une **hiérarchie de composants**.

Quelque soit la version d'angular (1, 2, 4 ou +), le **binding automatique** entre valeurs saisies ou affichées et les valeurs des objets "javascript" constitue **une des principales valeurs ajoutées du framework**.

Contrairement à l'ancienne version 1.x, les nouvelles versions 2+ du framework angular sont clairement orientées "composants" (en s'inspirant de la **norme "webComponent"**).

1.5. Structure "Single Page" et routage angular

Single Page Application et switch de sous parties



Coté client (navigateur)

index.html téléchargée (et utilisée longtemps)

**librairies angular
+ main component
avec <router-outlet>**

SWITCH

**sous-composants visuels
(ts/js + html + css)**

Services invisibles (partagés)

Navigation (routage) selon
speudo url (et switch de <div ...>)

singletons

appels
http
(ajax)

De façon à ce que le code javascript (librairies angular + code de l'application) soit conservé en mémoire sur le long terme, une application Angular est constitué d'**une seule grande page "index.html"** qui est **elle même décomposée en une hiérarchie de composants** (ex : *header*, *footer*, *content* ,). On parle généralement en terme de "**SPA : Single Page Application**" pour désigner cette architecture web (très classique).

Le **composant principal** ("main.component" , ".ts" , ".html") **comporte** très souvent une balise spéciale **<router-outlet></router-outlet>** (fonctionnellement proche de **<div>**) dont le **contenu** (interchangeable) sera automatiquement remplacé par **un des sous composants importants** de l'application.

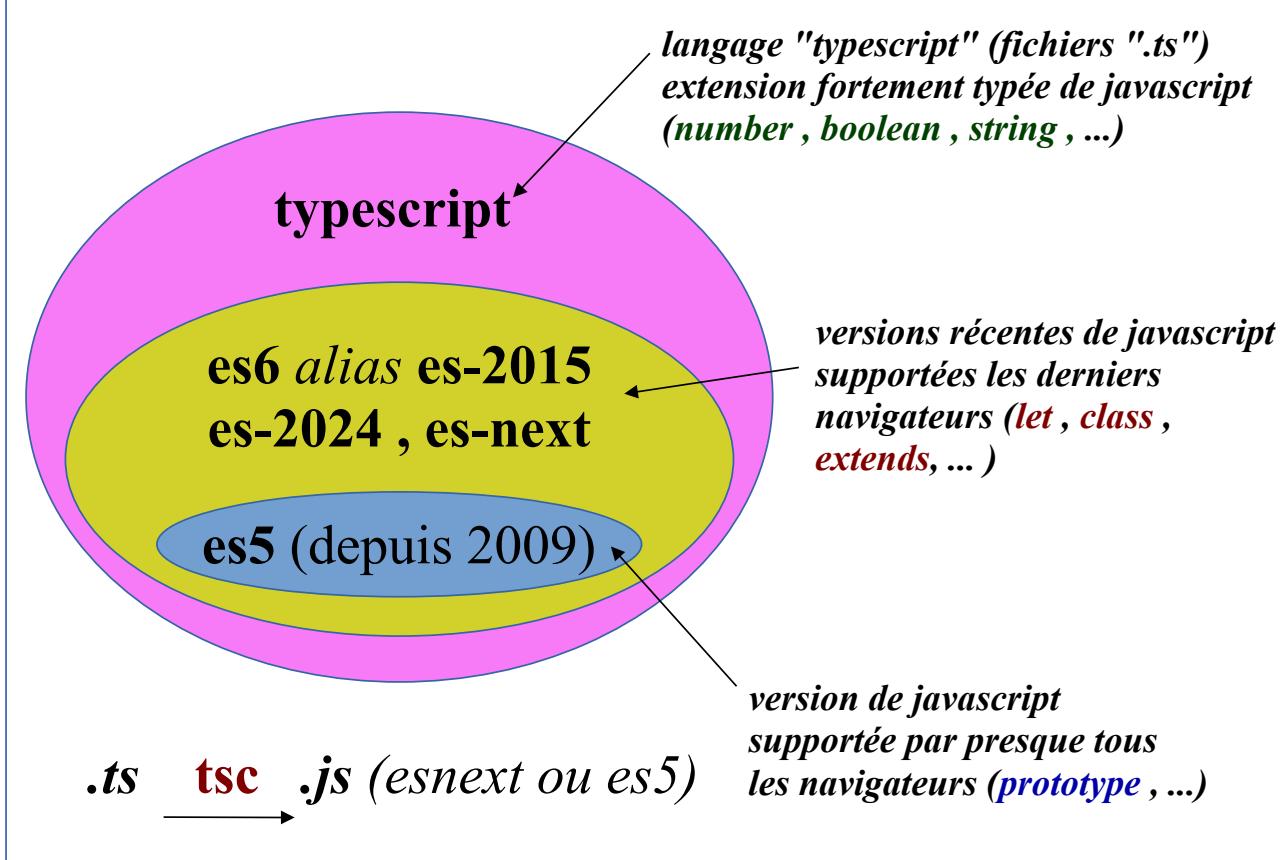
Le **switch de sous composants** sera associé à des **navigations** généralement **paramétrées dans un module de routage** .

En pouvant associer une speudo-URL relative à l'affichage contrôlé d'un certain sous composant précis , il est ainsi possible de mémoriser des "bookmarks / favoris / marques-pages" dans un navigateur .

1.6. Particularités du framework "Angular" (v>=2)

- le code d'une application angular est bien structuré (orienté objet / syntaxe rigoureuse grâce à typescript) et est "très maintenable".
- le framework "angular" est dès le départ très complet (rendu , binding , routage , appels ajax/http , ...) , ce qui n'est volontairement pas le cas de certains autres frameworks concurrents (backbone , react , knockout-js, ...)
- l'environnement de développement est basé (depuis la v2) sur **npm** et **@angular/cli** et est maintenant très complet (tests , génération de bundles , ...)

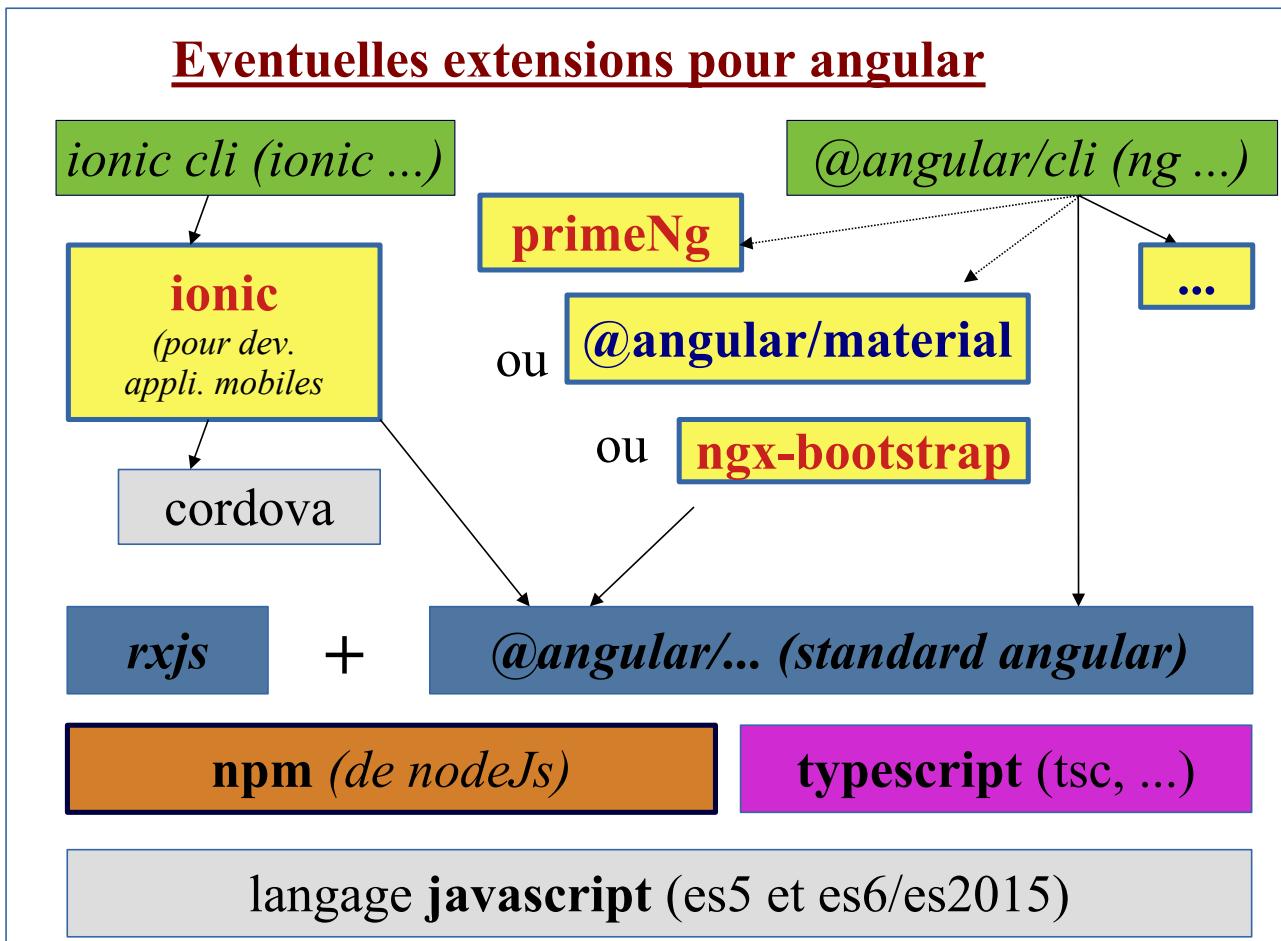
Fonctionnalités "es5" , "es6" et "typescript"



NB :

- Angular Js (1.x) n'était basé que sur javascript/es5 et ne nécessitait aucun environnement de développement sophistiqué (un simple "notepad++" suffisait). En contre partie de cet environnement de développement simpliste, le code d'une application "Angular Js / 1.x" était assez rapidement complexe à maintenir (pas adapté aux applications de grandes tailles).
- Depuis la v2 , "Angular" n'est plus qualifié de "Js" et s'appuie sur un environnement de développement beaucoup plus sophistiqué (npm + @angular/cli + typescript) et très complet (tests , générations de "bundles" ,).
- Depuis la V2 d'angular les composants sont codés en typescript "typé et orienté objet" (.ts)
- A court terme les fichiers ".ts" sont traduits en ".js" (es5 ou es6+) de façon à pouvoir être interprétés par presque tous les navigateurs des années 2010-2018 ou 2018-202x.

1.7. Eventuelles extensions pour angular



primeNg , @angular/material et ngx-bootstrap sont trois extensions concurrentes qui sont constituées d'un ensemble homogène de nouveaux composants graphiques réutilisables (ex : tabs/onglets , menus déroulants , panels , ...)

Attention :

- Certains jolis thèmes de **primeNg** sont des extensions payantes et la programmation de nouveaux thèmes pour "primeNg" n'est pas simple.
- L'extension **ngx-bootstrap** était bien pour angular 8,9,10,11,12 mais devient délicate à utiliser avec angular 13,14,15 (versions de ngx-bootstrap à la traîne)
- L'extension **@angular/material** est l'une des seules qui continue à bien suivre le rythme des évolutions de versions

En utilisant une de ces bibliothèques additionnelles , le développement concret d'une application angular s'appuie sur de nouvelles balises prêtes à l'emploi et facilement paramétrables .

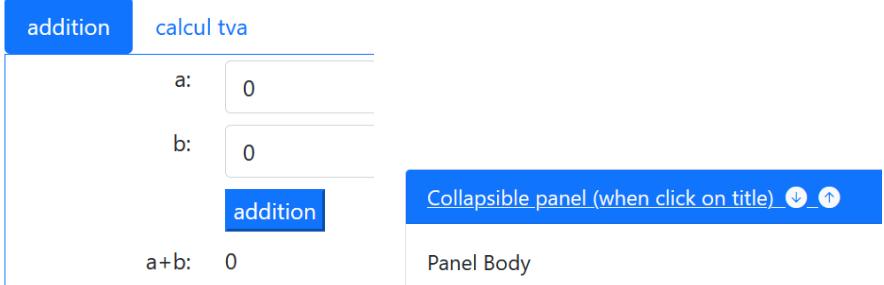
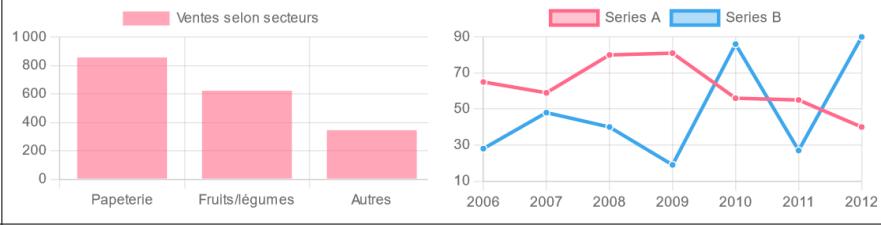
--> avantage : code plus compact et intégration naturelle dans le reste du code applicatif angular.

--> petit inconvénient : balisages et paramétrages assez spécifiques (moins portables que du classique "html5+css3") .

NB : Angular 1.x s'appuyait en interne sur "jquery lite" . Depuis la v2 , Angular ne s'appuie plus du tout sur jquery. Il est vivement déconseillé d'utiliser "jquery" avec angular 2,4,6+

ionic est une extension de angular qui s'appuie en interne sur "apache cordova" et qui permet de développer des applications mobiles hybrides (en partie "web" , en partie native) pour les smartphones "ios/iphone" , "android" , "windows" , ...

Quelques bonnes extensions pour angular (testées/utilisées) :

extensions	utilités																																
ngx-bootstrap ou bien material ou bien primeNg	Composants graphiques évolués réutilisables (panneaux , onglets , ...)																																
	 <p>Collapsible panel (when click on title) ↓ ↑</p> <p>Panel Body</p>																																
ng2-charts	Affichage de diagrammes/graphiques (courbes , ...) en s'appuyant sur chartJs et l'api canvas																																
	 <table border="1"> <caption>Data for Bar Chart: Ventes selon secteurs</caption> <thead> <tr> <th>Secteur</th> <th>Ventes</th> </tr> </thead> <tbody> <tr> <td>Papeterie</td> <td>800</td> </tr> <tr> <td>Fruits/légumes</td> <td>600</td> </tr> <tr> <td>Autres</td> <td>300</td> </tr> </tbody> </table> <table border="1"> <caption>Data for Line Chart: Series A and Series B</caption> <thead> <tr> <th>Année</th> <th>Series A</th> <th>Series B</th> </tr> </thead> <tbody> <tr> <td>2006</td> <td>650</td> <td>300</td> </tr> <tr> <td>2007</td> <td>550</td> <td>450</td> </tr> <tr> <td>2008</td> <td>750</td> <td>350</td> </tr> <tr> <td>2009</td> <td>800</td> <td>200</td> </tr> <tr> <td>2010</td> <td>550</td> <td>850</td> </tr> <tr> <td>2011</td> <td>500</td> <td>250</td> </tr> <tr> <td>2012</td> <td>400</td> <td>900</td> </tr> </tbody> </table>	Secteur	Ventes	Papeterie	800	Fruits/légumes	600	Autres	300	Année	Series A	Series B	2006	650	300	2007	550	450	2008	750	350	2009	800	200	2010	550	850	2011	500	250	2012	400	900
Secteur	Ventes																																
Papeterie	800																																
Fruits/légumes	600																																
Autres	300																																
Année	Series A	Series B																															
2006	650	300																															
2007	550	450																															
2008	750	350																															
2009	800	200																															
2010	550	850																															
2011	500	250																															
2012	400	900																															
ngx-leaflet	Affichage de cartes (en s'appuyant sur openstreetmap ou autres)																																
																																	
ng2-dragula	Drap & drop bien intégré à angular																																
tinymce-angular	Éditeur HTML intégré																																
...	...																																

NB: La plupart des extensions pour angular sont à considérées comme facultatives mais sont très pratiques et permettre d'obtenir un code lisible compact et maintenable dans un style bien "angular" .

2. Principaux éléments d'angular

Par ordre d'importance.

Elements	Fonctionnalités
Component	Composant visuel (sous partie d'une page HTML)
Service	Objet invisible (en arrière plan des composants) pour partager des données et des traitements (ex : session utilisateur ou appels http)
Signal	Element technique encapsulant une donnée variable dont les changements seront automatiquement signalés pour réactualiser un (ou plusieurs) affichages.
Observable	Element technique (de RxJs) encapsulant des données asynchrones et permettant d'enchaîner des traitements intermédiaires (tris , filtrages , transformations, ...) et pouvant conduire à un (ou plusieurs) ré-affichages automatiques .
Directive	Fonctionnalité réutilisable (ex : boucle ou modifications des styles CSS) que l'on peut appliquer sur n'importe quelle balise HTML compatible (ex : p, span, div)
Pipe	Pour appliquer des transformations de format (ex : arrondir à 2 chiffres après la virgule) au moment de l'affichage d'une données au sein des templates HTML
Interceptor	Intercepteur automatique permettant d'enrichir une requête HTTP (en ajoutant par exemple un jeton de sécurité dans la partie "Authorization" de l'entête)
Guard	Gardien permettant de bloquer si besoin une navigation si certaines conditions ne sont pas respectées (pas de login, droits d'accès insuffisants, ...)
Resolver	Elément technique facultatif permettant d'attendre la fin d'un téléchargement de données avant d'atteindre la navigation effective (retardée) vers un composant ayant besoin de ces données au sein de sa phase d'initialisation .
Trigger	Element technique associé à une animation
...	

Généralement trois niveaux dans la maîtrise du framework angular :

1. savoir participer au développement d'une application en codant les éléments fondamentaux (composants applicatifs, services , ...)
2. configurer la sécurité et savoir coder certains tests (e2e/cypress ou unitaires)
3. coder des composants réutilisables et autres aspects avancés

3. Evolution importante d'angular en 2023/2024

A partir de la version 14, le framework angular a recherché la voie de la simplification et de l'efficacité pour devenir plus concurrentiel vis à vis de certains frameworks concurrents plus simples à aborder tels que React ou VueJs .

La version 14 a proposé des composants simplifiés "**standalone component**" sans module. Cette voie a été suivie au sein des versions 15,16 et 17,18,19.

La version 16 a introduit la notion de **signaux** pour simplifier la synchronisations des éléments d'une IHM web/angular . Une bonne partie de l'api "signal angular" était restée en phase "developper preview" au sein des version 17 et 18 et a enfin été stabilisée/finalisée en version 19.

La version 17 a apporté de nouvelles syntaxes alternatives/possibles au sein des templates HTML via des instructions "**code-flow**" telles que `@if() {} @else { ... }`

- **Tout ceci s'utilise de manière facultative .**
 - **On peut toujours utiliser les syntaxes et la structure des versions antérieures (<=14) .**
 - **L'utilisation des nouveautés des versions 16,17,18,19 est néanmoins très conseillée car cela va dans le sens de performances améliorées et de syntaxes recommandées.**
 - La version 17 est quelquefois appelée "*angular reborn*" (renaissance d'angular).
-

Au début 2025 , seules les versions ≥ 17 d'angular bénéficient d'un support (maintenance) gratuite en cas de bug ou de faille de sécurité (<https://endoflife.date/angular>) .

On peut donc (à partir de 2025) affirmer que :

- le coeur d'une application "angular" doit idéalement être en mode "standalone" (sans ancien `@NgModule()` maintenant facultatif et désormais plus recommandé)
- l'usage des signaux et des "code flow" est très recommandé (pour améliorer les performances) et donc moins de `BehaviorSubject` , `@Input` , `@Output` et plus de `signal()` , `input()` , `output()` , ... au sein des projets récents.

II - Rappels sur javascript et typescript

1. Versions de javascript , bases fondamentales

1.1. Rappel: Version de javascript (ES5 ou ES6/es2015)

Les versions standardisées/normalisées de javascript sont appelées **ES** (EcmaScript) .

Les versions modernes sont :

- **ES5** (de 2009) – supporté par quasiment tous les navigateurs actuels ("mobiles" ou "desktop")
- **ES6** (renommé **ES2015** car normalisé en 2015) . ES6/es2015 est petit à petit supporté (partiellement ou complètement) par de plus en plus de navigateurs récents.
ES6/ES2015 apporte quelques nouvelles syntaxes et mots clefs (**class** , **let**, ...) et gère des modules dits "**statics**" via "**import** { ComponentName } **from** 'moduleName' ;" et **export** .
- **ES2017** a apporté **async/await** en tant que nouvelles fonctionnalités importantes
- **ESNext** (dernière version existante (ex : 2020,2021,...))

1.2. Quelques éléments essentiels issus de javascript "es5"

Number("123px") retourne NaN tandis que **parseInt**("123px") retourne 123 .

Remarque importante : une **variable non initialisée** est considérée comme "**undefined**" (notion proche de "null") et ne peut pas être utilisée en tant qu' objet préfixe .

L'opérateur **typeof** *variable* retourne une chaîne de caractère de type "string" , "number" , "boolean" , "undefined" , selon le type du contenu de la variable à l'instant t .

```
var vv ;
if( typeof vv == "undefined" ) {
    console.log("la variable vv n'est pas initialisée") ;
}

if( vv == null ) {
    console.log("la variable vv est soit null(e) soit non initialisée") ;
}
```

L'opérateur **==** (d'origine c/c++/java) retourne true si les 2 expressions ont des valeurs à peu près équivalentes (ex 25 est une valeur considérée équivalente à "25") .

L'opérateur **====** (spécifique à javascript) retourne true si les 2 expressions ont à la fois les mêmes valeurs et le même type ("25" et 25 ne sont pas de même type) .

window.**setTimeout**(chExpr,n) permet d'interpréter l'expression chExpr en différé (n ms plus tard)

window.**setInterval**(chExpr,n) permet de lancer l'interprétation périodique de chExpr toutes les n ms;

```
var jsonString = JSON.stringify(jsObject) ;
```

```
var jsObject2 = JSON.parse(jsonString) ;
```

```
delete tab[i] ; //supprime la valeur de tab[i] qui devient undefined .
```

```
tab.splice(i , 2 , val1 , val2) ; //remplace tab[i] par val1 et tab[i+1] par val2 , etc
```

```
tab.splice(i, 1) ; //remplace tab[i] par rien et donc supprime la case tab[i]
```

```
//sans trou , certains autres éléments sont déplacés (changement d'indice)
```

1.3. LocalStorage et SessionStorage

localStorage et **sessionStorage** sont des objets prédefinis des navigateurs modernes qui permettent de stocker et récupérer des informations depuis une ou plusieurs pages html différentes si nécessaire.

```
localStorage.setItem("clefXy","valeur qui va bien") ;  
var stringValue= localStorage.getItem("clefXy") ;
```

NB : sessionStorage est lié à une session utilisateur et les informations qui y sont stockées sont généralement conservées sur une plus courte durée .

Selon le navigateur , les informations stockées en localStorage sont (ou pas) conservées suite à un redémarrage du navigateur.

Les informations stockées dans localStorage sont conservées suite à un "refresh" d'une page html .

2. Let, const , boucles for , arrow function

2.1. Mots clefs "let" et "const" (es2015)

Depuis longtemps (en javascript) , le mot clef "var" permet de déclarer explicitement une variable dont la portée dépend de l'endroit de sa déclaration (globale ou dans une fonction).

Sans aucune déclaration, une variable (affectée à la volée) est globale et cela risque d'engendrer des effets de bords (incontrôlés) .

Introduits depuis es6/es2015 et typescript 1.4 , les mots clefs **let** et **const** apportent de nouveaux comportements :

- Une variable déclarée via le mot clef **let** a une **portée limité au bloc local** (exemple boucle for) . Il n'y a alors pas de collision avec une éventuelle autre variable de même nom déclarée quelques ligne au dessus du bloc d'instructions (entre {} , de la boucle).
- Une variable déclarée via le mot clef **const** **ne peut plus changer de valeur après la première affectation**. Il s'agit d'une **constante** .

Exemple :

```
const PISur2 = Math.PI / 2;
//PISur2=2; // Error, can't assign to a `const`
console.log("PISur2 = " + PISur2);

var tableau = new Array();
tableau[0] = "abc";
tableau[1] = "def";

var i = 5;
var j = 5;

//for(let i in tableau) {
for(let i=0; i<tableau.length; i++) {
    console.log("*** at index " + i + " value = " + tableau[i] );
}

//for(j=0; j<tableau.length; j++) {
for(var j=0; j<tableau.length; j++) {
    console.log("### at index " + j + " value = " + tableau[j] );
}

console.log("i=" + i); //affiche i=5
console.log("j=" + j); //affiche j=2
```

2.2. "Arrow function" et "lexical this" (es2015)

Rappels (2 syntaxes "javascript" ordinaires) valables en "javascript/es5" :

```
//Named function:  
function add(x, y) {  
    return x+y;  
}  
  
//Anonymous function:  
var myAdd = function(x, y) { return x+y; };
```

Arrow functions (es2015) (alias "Lambda expressions")

Une "**Arrow function**" en javascript/es2015 (à peu près équivalent à une "lambda expression" de java >8) est syntaxiquement introduite via `() => {}`

Il s'agit d'une syntaxe épurée/simplifiée d'une fonction anonyme où les parenthèses englobent d'éventuels paramètres et les accolades englobent le code.

Subtilité du "lexical this" :

La valeur du mot clef "this" est habituellement évaluée lors de l'invocation d'une fonction . Dans le cas d'une "lambda expression" , le mot clef this est évalué dès la création de la fonction et correspond au this du niveau englobant (classe ou fonction).

Exemples de "lambda expressions" :

```
myFct = (tab) => { var taille = tab.length; return taille; }  
//ou plus simplement:  
myFct = (tab) => { return tab.length; }  
//ou encore plus simplement:  
myFct = (tab) => tab.length;  
//ou encore plus simplement:  
myFct = tab => tab.length;
```

```
var numRes = myFct([12,58,69]);  
console.log("numRes=" + numRes); //affiche 3
```

```
myFct2 = (x,y) => { return (x+y) / 2; } //with statement body in {}  
//ou plus simplement:  
myFct2 = (x,y) => (x+y) / 2; //with simple expression
```

NB : les "**Promise**" (es2015) et la technologie "**RxJs**" utilisée par angular>=2 utilisent beaucoup de "Arrow Functions" .

Exemple de "arrow function" combiné ici avec `arrayXy.forEach(callback)` de "javascript/es5" :

```
let array1 = [ 1 , 2 , 3 , 4 , 5 , 6 ];
let eltPairs = [];
array1.forEach( (e) => { if( (e % 2) === 0 )
                           eltPairs.push(e);
                         }
                  );
console.log(eltPairs); // affiche [2,4,6]
```

Suite de l'exemple utilisant nouveauTableau = `arrayXy.map(transform callback)` de es5 :

```
let eltImpairs = eltPairs.map( v => v-1 );
console.log(eltImpairs); // affiche [1,3,5]
```

Exemple montrant "lexical this" (arrow function utilisant `this` de niveau englobant) :

```
var toto = {
  _name: "toto",
  _friends: [ 'titi' , 'tata'],
  printFriends() {
    this._friends.forEach(f =>
      console.log(this._name + " est ami avec " + f));
  }
};
toto.printFriends();
```

Autre comportement à connaître:

Si une "fonction fléchée" / "arrow fonction" est à l'intérieur d'une autre fonction, elle partage alors les arguments/paramètres de la fonction parente .

2.3. avec for...of (es2015) et itérateurs internes

```
var tableau = new Array();  
//tableau.push("abc");  
//tableau.push("def");
```

```
tableau[0] = "abc";  
tableau[1] = "def";
```

Au moins 3 parcours possibles via boucle **for**:

```
var n = tableau.length;  
for(let i = 0; i<n; i++) {  
    console.log(">> at index " + i + " value = " + tableau[i] ) ;  
}
```

```
for(let i in tableau) {  
    console.log("** at index " + i + " value = " + tableau[i] ) ;  
}
```

//**for(index in ...)** existait déjà en es5

//**for(...of ...)** au sens "**for each ... of ...**" est une nouveauté de es2015

```
for( let s of tableau){  
    console.log("## val = " + s) ;  
}
```

NB : la boucle **for...of** est pré définie sur un tableau . il est cependant possible de personnaliser son comportement si l'on souhaite la déclencher sur une structure de données personnalisée. On peut pour cela mettre en oeuvre des itérateurs (et éventuels générateurs de bas niveaux) ---> dans chapitre ou annexe "éléments divers et avancés de es2015" .

3. Template string , destructuration , spread operator

3.1. "template string" es2015 (avec quotes inverses et \${})

```
var name = "toto";
var year=2015;
// ES5
//var message = "Hello " + name + ", happy " + year; // Hello toto , happy 2015
// ES6/ES2015 :
const message = `Hello ${name} , happy ${year}`; // Hello toto , happy 2015
//attention: exception "ReferenceError: name is not defined" si name est undefined
console.log(message);
```

\${} peut éventuellement englober des expressions mathématiques ou bien des appels de fonctions.

```
let x=5 , y=6;
let carre = (x) => x*x ;
console.log(`pour x=${x} et y=${y} , x*y=${x*y} et x*x=${carre(x)}`);
//affiche pour x=5 et y=6 , x*y=30 et x*x=25
```

template-string multi-lignes :

```
/*
//ES5
let htmlPart=
"<select> \
<option>1</option> \
<option>2</option> \
</select>";
*/
```

```
//template multi-lignes ES2015:
let htmlPart=
`<select>
<option>1</option>
<option>2</option>
</select>`;
console.log(htmlPart);
```

3.2. Map , Set

```
// Sets (ensembles sans doublon)
var s = new Set();
s.add("hello").add("goodbye").add("hello");
if(s.size === 2)
    console.log("s comporte 2 elements");
if(s.has("hello"))
    console.log("s comporte hello");
// List ==> Array ordinaire (déjà en es5 , à remplir via .push() ).
```

```
// Maps (table d'association (clef,valeur))
var m = new Map();
m.set("hiver", "froid , neige");
m.set("printemps", "fleur , vert");
m.set("ete", "soleil , plage");
m.set("ete", "chaud , plage"); //la nouvelle valeur remplace l'ancienne .
m.set("automne", "feuilles mortes");
let carateristique_ete = m.get("ete");
console.log("carateristique_ete="+carateristique_ete); //chaud , plage
if(m.has("ete"))
    console.log("Map m comporte une valeur associée à ete");
for(saison of m.keys()){
    console.log("saison "+ saison + " - " + m.get(saison));
}
//m.values() permettrait d'effectuer une boucle sur les valeurs (peu importe les clefs)
for([k,v] of m.entries()){
    console.log("saison "+ k + " -- " + v);
}
m.forEach((val,key)=> console.log("saison "+ key + " --- " + val));
m.clear();
if(m.size==0)
    console.log("map m is empty");

//Bien que ce code soit lisible et explicite, un vieux objet javascript en faisait autant :
var objectMap = {
    hiver : "froid , neige",
    printemps : "fleur , vert",
};
objectMap["ete"]="chaud, plage" ;
console.log("carateristique_hiver="+ objectMap["hiver"]); //froid , neige

//Une des valeurs ajoutées par "Map" (es2015) est la possibilité d'avoir des clefs de n'importe
//quelle sorte possible (ex : window , document , element_arbre_DOM, ...).
```

3.3. "Destructuring" (affectation multiple avec perte de structure)

Destructuring objet : extract object parts in several variables :

```
const p = { nom : 'Allemagne' , capitale : 'Berlin' , population : 83000000, superficie : 357386};
const { nom , capitale } = p;
console.log("nom="+nom+ " capitale=" +capitale);
//nom="?"; interdit car nom et capitale sont considérées comme des variables "const"
```

//NB: les noms "population" et "superficie" doivent correspondre à des propriétés de l'objet
//dont il faut (partiellement) extraire certaines valeurs (sinon "undefined")
//l'ordre n'est pas important

```
const { superficie , population } = p;
console.log("population="+population+ " superficie=" +superficie);
```

==>

nom=Allemagne capitale=Berlin
population=83000001 superficie=357386

utilité concrète (parmi d'autres) : **fonction avec paramètres nommés** :

```
function fxyz_with_named_param( { paramX=0 , a=0 , b=0 , c=0 } = {} ){
    //return ax^2+bx+c
    return a * Math.pow(paramX,2) + b * paramX + c;
}

let troisFois4 = fxyz_with_named_param( { paramX :4 , b : 3 } );
console.log("troisFois4="+troisFois4 );//12
let deuxFois4AuCarreplus6 = fxyz_with_named_param( { paramX :4 , a : 2 , c :6 } );
console.log("deuxFois4AuCarreplus6="+deuxFois4AuCarreplus6 );//38
```

Destructuring iterable (array or ...) :

```
const [ id , label ] = [ 123 , "abc" ];
console.log("id="+id+ " label=" +label);

//const array1Iterable = [ 123 , "abc" ];
//var iterable1 = array1Iterable;
const string1Iterable = "XYZ";
var iterable1 = string1Iterable;
const [ partie1 , partie2 ] = iterable1;
console.log("partie1="+partie1+ " partie2=" +partie2);
==>
id=123 label=abc
partie1=X partie2=Y
```



Autre exemple plus artistique (Picasso) :

3.4. for (..of ..) with destructuring on Array , Map, ...

```
const dayArray = ['lundi', 'mardi', 'mercredi'];
for (const entry of dayArray.entries()) {
    console.log(entry);
}
// [0, 'lundi']
// [1, 'mardi']
// [2, 'mercredi']

for (const [index, element] of dayArray.entries()) {
    console.log(`#${index}. ${element}`);
}
// 0. lundi
// 1. mardi
// 2. mardi
```

```
const mapBoolNoYes = new Map([
    [false, 'no'],
    [true, 'yes'],
]);
for (const [key, value] of mapBoolNoYes) {
    console.log(`#${key} => ${value}`);
}
// false => no
// true => yes
```

3.5. "spread operator ... " sur choses itérables

```
let myArray0 = [ 'e1' , 'e2' , 'e3' ] ;
let monIterable = myArray0 ;
let myArray1 = [ 'e0' , ...monIterable , 'e4' , 'e5' ];
let myArray2 = [ ...monIterable ];
console.log(myArray1); // [ 'e0' , 'e1' , 'e2' , 'e3' , 'e4' , 'e5' ]
console.log(myArray2); // [ 'e1' , 'e2' , 'e3' ]
```

4. Promise es2015 et async/await

4.1. Promise es2015

Lorsque l'on déclenche via un appel de fonction un traitement asynchrone dont le résultat ne sera prêt/connu que dans le futur , on peut retourner immédiatement un objet de type "Promise" qui encapsule l'attente d'une réponse promise.

Le résultat promis qui sera récupéré en différé dans le temps est soit une réponse positive (promesse tenue) soit une réponse négative (erreur / promesse rompue) .

A l'intérieur de la fonction asynchrone appelée, on crée et retourne une promise via l'instruction
return new Promise((resolve,reject)=> { if(...) resolve(...) else reject(...); }) ;

//où *resolve* et *reject* sont des noms logiques de callbacks appelées dans le futur

//pour transmettre l'issue positive ou négatif du traitement asynchrone .

A l'extérieur , l'appel s'effectue via la syntaxe

.then((resolvedValue)=>{} , (rejectedValue)=> { ... }) ;

ou bien

.then((resolvedValue)=>{})

.catch((rejectedValue)=> { ... }) ;

NB: Si à l'intérieur d'un .then(()=>{...}) on appelle et retourne une fonction asynchrone retournant à son tour une autre "Promise" , on peut alors enchaîner d'une manière lisible une séquence d'appel à d'autres .then() qui seront alors exécutés les uns après les autres au fur et à mesure de la résolution des promesses asynchrones :

```
appelAsynchrone1RetournantPromesse1(...)
.then((resPromesse1)=>{ .... ; return appelAsynchrone2RetournantPromesse2(...);})
.then((resPromesse2)=>{ .... ; return appelAsynchrone3RetournantPromesse3(...);})
.then((resPromesse3)=>{ .... ; })
.catch((premiereErreurPromesse1ou2ou3)=>{...});
```

NB : avant d'exister en version normalisée "es2015" , les "Promises" avaient été prises en charge via l'ancienne bibliothèque "q" (*var deferred = Q.defer(); deferred.resolve(data); ... return deferred.promise;*) avec même utilisation *.then(...).then(...).catch(...)* .

Les "Promises" étaient déjà beaucoup utilisées à l'époque de es5/angular-js (avant 2015 et es6/es2015) .

Exemple (avec Promise/es2015) :

```
var stdin = process.stdin;
var stdout = process.stdout;

function ask_(question) {
    return new Promise ((resolve,reject)=> {
        stdin.resume();
        stdout.write(question + ": ");
        stdin.once('data', function(data) {
            data = data.toString().trim();
            if(data=="fin")
                reject("end/reject");
        });
    });
}
```

```

        else
            resolve(data);
        });
    });

var x,y,z;
//calcul (x+y)*z après enchaînement lisible (proche séquentiel) de "saisir x", "saisir y", "saisir z":
ask_("x")
.then((valX)=>{ x=Number(valX); return ask_("y");}
.then((valY)=> { y=Number(valY); let res=x+y ;
                console.log("(x+y)=" +res);
                return ask_("z");
            })
.then((valZ)=> { z=Number(valZ); let res=(x+y)*z ;
                console.log("(x+y)*z=" +res);
                process.exit();
            })
.catch((err)=>{console.log(err);process.exit();});

```

4.2. Async/await es2017

async et **await** sont de nouveaux mots clefs de **es2017** (inspiré de typescript et de l'univers .net/Microsoft) .

Ces nouveaux mots clefs permettent de simplifier les enchaînement de fonctions asynchrones en générant et attendant automatiquement des promesses ("Promise de es2015") .

Principes async/await :

- **return resultat** dans une fonction **async**
est (depuis es2017) équivalent à **return new Promise.resolve(resultat);**
- **throw new Error('erreur')** dans une fonction **async**
est (depuis es2017) équivalent à **return new Promise.reject(new Error('erreur'));**
- **await** permet d'attendre la résolution d'une promesse (liée à un sous appel asynchrone) et de récupérer la valeur dans une variable (équivalent de **.then(...)** automatique) .
NB : await ne peut être utilisé qu'au sein d'une fonction prefixée par async .
- au sein d'une fonction prefixée par **async**, un bloc **try { ... } catch {...}** ordinaire permet de récupérer aussi bien certaines exceptions synchrones que certains échecs liés à des promesses non tenues par des sous appels asynchrones déclenchés via **await** .
Autrement dit : **try { await appel_async1(...) ;**
 await appel_async2(...) } catch {...}
peut remplacer **appel_async1().then(()=>... ; return appel_async2(...);)**
 .then(()=>...)
 .catch((e)=>...) ;

Exemple (async/await) :

```

var stdin = process.stdin;
var stdout = process.stdout;

function ask_(question) {
    return new Promise ((resolve,reject)=> {
        stdin.resume();
        stdout.write(question + ": ");
        stdin.once('data', function(data) {
            data = data.toString().trim();
            if(data=="fin")
                reject("end/reject");
            else
                resolve(data);
        });
    });
}

async function ask_and_compute_x_plus_y(){
    try{
        let x,y;
        const valX = await ask_("x"); x=Number(valX);
        const valY = await ask_("y"); y=Number(valY);
        let xPlusY=x+y ;console.log("(x+y)=" +xPlusY);
        return xPlusY;
    }
    catch(e){
        console.log(e);
        throw new Error("xPlusY-error:"+e);
    }
}

async function x_plus_y_mult_z(){
    try{ /*
        const valX = await ask_("x"); let x=Number(valX);
        const valY = await ask_("y"); let y=Number(valY);
        let xPlusY=x+y ;console.log("(x+y)=" +xPlusY);
    */
        const xPlusY = await ask_and_compute_x_plus_y();
        const valZ = await ask_("z"); const z=Number(valZ);
        let res=xPlusY * z ;console.log("(x+y)*z=" +res);
    }
    catch(e){
        console.log(e);
    }
    process.exit();
}

x_plus_y_mult_z();

```

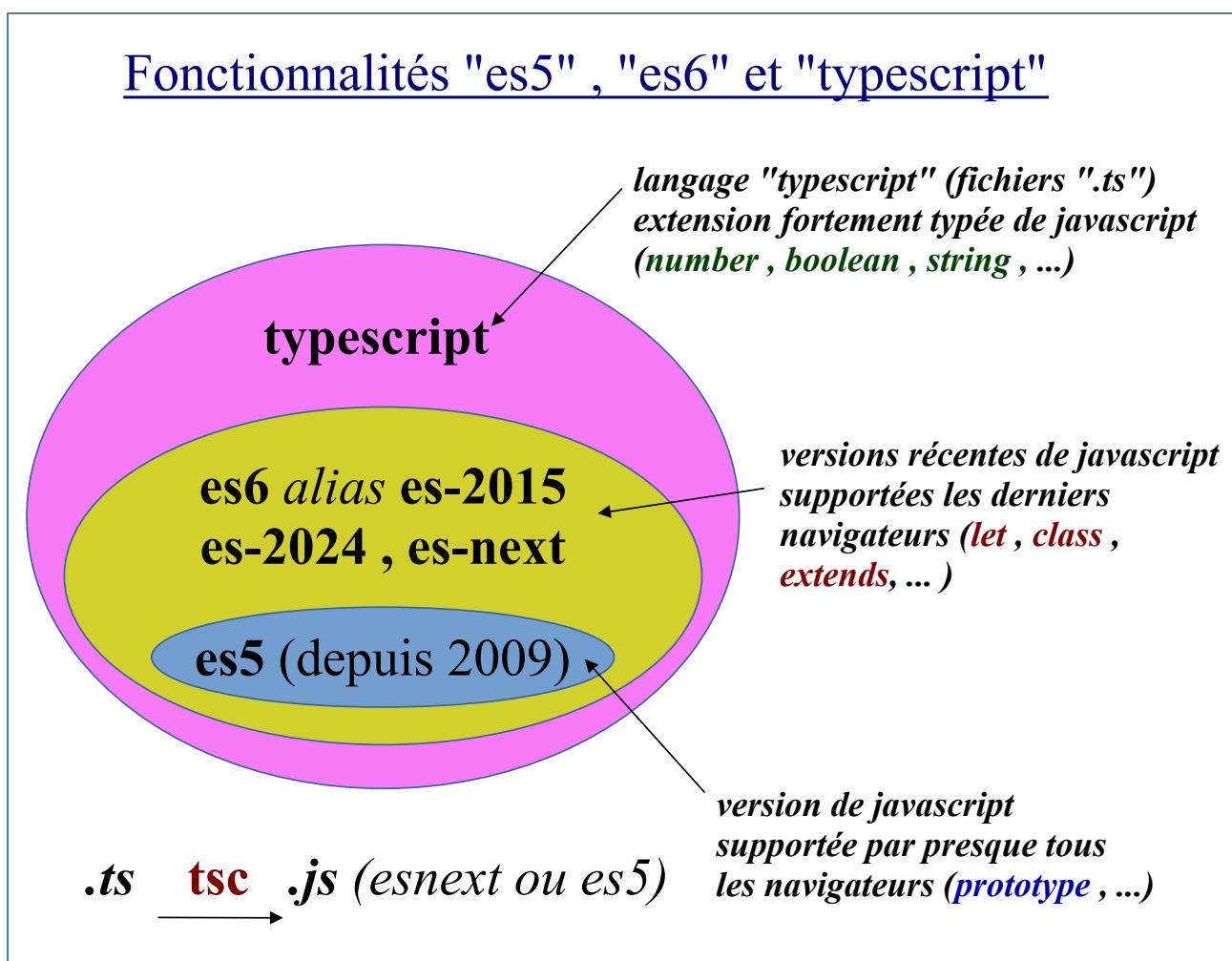
==>

```
x: 5
y: 6
(x+y)=11
z: 3
(x+y)*z=33
```

5. Fondamentaux "typescript"

5.1. Présentation de TypeScript / ts

Fonctionnalités "es5" , "es6" et "typescript"



- **TypeScript** doit être vu comme un **langage un peu virtuel à systématiquement compiler**.
- On gagne en rigueur à écrire le code source en typescript plutôt qu'en javascript.
- C'est toujours **du code transformé en javascript qui s'exécute au runtime** au sein d'un navigateur ou bien de nodeJs .

Installation de typescript :

De façon à télécharger et lancer le compilateur "tsc" , on pourra (entre autres possibilités) s'appuyer sur **nodejs/npm** (à préalablement télécharger/installer si nécessaire en version **LTS** depuis <https://nodejs.org/en/download/>)

```
npm install -g typescript
```

5.2. utilisation directe de tsc sans tsconfig.json

Premiers pas (sans structure de projet) :

f1.ts

```
let a :number ;  
a=6 ;  
console.log(a) ;
```

traduction (transpilation/compilation) de ".ts" vers ".js" via tsc :

tsc f1.ts

génère (par défaut, sans option) dans le même répertoire le fichier suivant :

f1.js

```
var a;  
a = 6;  
console.log(a);
```

Lancement du fichier f1.js (ici via node sans navigateur internet) :

node f1.js

6

5.3. Générer le fichier tsconfig.json (début de projet)

au sein d'un répertoire (vide ou pas) de projet on peut lancer la commande suivante :

tsc --init

---> cela génère un fichier de configuration avec plein de paramètres possibles en commentaires

Lignes importantes de tsconfig.json

```
{  
  "compilerOptions": {  
    "target": "es2017" ou "es5" ou autre  
    "module": "CommonJS" ou "es2015" ou "ESNext" ou autre ,  
    "rootDir": "src",  
    "outDir": "dist/js",  
    "strict": true  
  }  
}
```

Un lancement de tsc (sans argument) permet de compiler d'un seul coup tout un tas de fichiers (en tenant compte des options du fichier tsconfig.json).

NB : Lancé avec l'option -w, le transpilateur tsc va automatiquement compiler tous les fichiers ".ts" fraîchement modifiés et sauvegardés.

Ce fonctionnement met en place une tâche de fond surveillant l'état d'un paquets de fichiers dans quelques répertoires d'un projet . Une fois lancé " **tsc -w**" ne rend plus la main dans une console "CMD" ou "powershell" ou "sh" . Il vaut mieux donc lancer **tsc -w** dans un terminal dédié .

5.4. Types élémentaires de typescript

boolean	<code>var isDone: boolean = false;</code>
number	<code>var height: number = 6; var size : number = 1.83 ;</code>
string	<code>var name: string = "bob"; name = 'smith' ;</code>
array	<code>var list1 : number[] = [1, 2, 3]; var list2 : Array<number> = [1, 2, 3];</code>
enum	<code>enum Color {Red, Green, Blue}; // start at 0 by default // enum Color {Red = 1, Green, Blue}; var c: Color = Color.Green; //display as "1" by default var colorName: string = Color[1]; // "Green" if "Red" is at [0] // Color["Green"] return 1</code>
any	<code>var notSure :any = 4; notSure = "maybe a string instead"; notSure = false;</code>
unknown	<code>var something :unknown = 4; something = "string value"; var eh : string = something;</code>
void	<code>function warnUser(): void { alert("This is my warning message"); }</code>
object	(objet quelconque : plus précis que "any" , moins précis qu'un nom de classe). <code>var obj : object = { id : 2 , label : "cahier" } ; obj = { prenom : "jean" , nom : "Bon" } ; //structure objet différente acceptée .</code>

6. Classes , constructor , get/set , generic , interface

6.1. Classe et instances (typescript)

```
class Compte{
    numero : number = 0;
    label : string = " ?";
    solde : number = 0.0;

    debiter(montant : number) : void {
        this.solde -= montant; // this.solde = this.solde - montant;
    }

    crediter(montant : number) : void {
        this.solde += montant; // this.solde = this.solde + montant;
    }
}
```

```
var c1 = new Compte(); //instance (exemplaire) 1
console.log("numero et label de c1: " + c1.numero + " " + c1.label);
console.log("solde de c1: " + c1.solde);

var c2 = new Compte(); //instance (exemplaire) 2
c2.solde = 100.0;
c2.crediter(50.0);
console.log("solde de c2: " + c2.solde); //150.0
```

NB: Sans initialisation explicite (via constructeur ou autre) , les propriétés internes d'un objet sont par défaut à la valeur "**undefined**" mais lorsque **tsconfig.json** comporte la ligne d'option "**strict**": **true**, ceci ne fonctionne qu'avec l'option complémentaire "**strictPropertyInitialization**": **false** .

6.2. constructor

Un constructeur est une méthode qui sert à initialiser les valeurs internes d'une instance dès sa construction (dès l'appel à new).

En langage typescript le constructeur se programme comme la méthode spéciale "**constructor**" (mot clef du langage) :

```
class Compte{
    numero : number;
    label: string;
    solde : number;

    constructor(numero:number, libelle:string, soldeInitial:number){
        this.numero = numero;
        this.label = libelle;
        this.solde = soldeInitial;
    }

    //...
}
```

```
var c1 = new Compte(1,"compte 1",100.0);
c1.crediter(50.0);
console.log("solde de c1: " + c1.solde);
```

NB: il n'est pas possible d'écrire plusieurs versions du constructeur :

```
constructor(numero:number, libelle:string, soldeInitial:number){
    this.numero = numero;
    this.label = libelle;
    this.solde = soldeInitial;
}

constructor(){
    this.=0;
    this.label=?";
    this.=0.0;
}
```

Il faut donc quasi systématiquement utiliser la syntaxe = **valeur_par_defaut** sur les arguments d'un constructeur pour pouvoir créer une nouvelle instance en précisant plus ou moins d'informations lors de la construction :

```
class Compte{
    numero : number;
    label: string;
    solde : number;

    constructor(numero:number=0, libelle:string=?, soldeInitial:number=0.0){
        this.numero = numero;
```

```

        this.label = libelle;
        this.solde = soldeInitial;
    } //...
}
```

```

var c1 = new Compte(1,"compte 1",100.0);
var c2 = new Compte(2,"compte 2");
var c3 = new Compte(3);
var c4 = new Compte();
```

Remarque : en plaçant le mot clef **readonly** devant un attribut (propriété) , la valeur de celui ci doit absolument être initialisée dès le constructeur et ne pourra plus changer par la suite.

6.3. Propriété "private"

```

class Animal {
    private _size : number;
    name:string;
    constructor(theName: string = "default animal name") {
        this.name = theName;
        this._size = 100; //by default
    }
    move(meters: number = 0) {
        console.log(this.name + " moved " + meters + "m." + " size=" + this._size);
    }
}
```

```

var a1 = new Animal("favorite animal");
a1._size=120; //erreur détectée '_size' est privée et seulement accessible depuis classe 'Animal'.
a1.move();
```

Remarques importantes :

- **public par défaut .**
- En cas d'erreur détectée sur "private / not accessible" , le fichier ".js" est (par défaut) tout de même généré par "tsc" et l'accès à ".size" est tout de même autorisé / effectué au runtime.
⇒ private génère donc des messages d'erreurs qu'il faut consulter (pas ignorer) !!!

6.4. Accesseurs automatiques `get xxx()` /`set xxx()`

```
class Animal {
    private _size : number;
    public get size() : number{ return this._size;
    }
    public set size(newSize : number){
        if(newSize >=0) this._size = newSize;
        else console.log("negative size is invalid");
    }
...} //NB : le mot clef public est facultatif devant "get" et "set" (public par défaut)
```

```
var a1 = new Animal("favorite animal");
a1.size = -5; // calling set size() → negative size is invalid (at runtime) , _size still at 100
a1.size = 120; // calling set size()
console.log("size=" + a1.size); // calling get size() → affiche size=120
```

6.5. Mixage "structure & constructor" avec public ou private

```
class Compte{
    numero : number;
    label: string;
    solde : number;

    constructor(public numero : number=0,
                public label : string="?",
                public solde : number=0.0){
        this.numero = numero;
        this.label = label; this.solde = solde;
    } //...
}
```

```
var c1 = new Compte(1,"compte 1",100.0);
c1.solde = 250.0; console.log(c1.numero + ' ' + c1.label + ' ' + c1.solde );
```

Remarque importante : via le mof clef "**public**" ou "**private**" ou "**protected**" (au niveau des paramètres du constructeur) , certains paramètres passés au niveau du constructeur sont automatiquement transformés en attributs/propriétés de la classe .

Autrement dit, toutes les lignes "barrées" de l'exemple précédent sont alors générées implicitement (automatiquement) .

6.6. Héritage et valeurs par défaut pour arguments:

```
class AnimalDomestique {
    name:string;
    constructor(name: string = "defaultAnimalName") { this.name= name;}
    decire(){ console.log("AnimalDomestique nom=" + this.name );}
    parler(){ console.log("...") ; }
}
```

```
class Chat extends AnimalDomestique {
    nbHeuresSommeil /*:number */ = 14 ;
    constructor(name: string = "defaultCatName") { super(name); }
    decire() {
        console.log("Je suis un chat qui dort " +this.nbHeuresSommeil + " h");
        super.decire();
    }
    parler() { console.log("miaou miaou") ; }
    ronronner() { console.log("ronron...") ; }
}
```

```
class Chien extends AnimalDomestique {
    fonction : string | undefined ;
    constructor(name: string = "defaultDogName") { super(name); }
    decire() {
        console.log("Je suis un chien , fonction=" + this.fonction );
        super.decire();
    }
    parler() { console.log("whaouf whaouf") ; }
    monterLaGarde() { console.log("je monte la garde ...") ; }
}
```

```
var a = new AnimalDomestique(); //var a = new AnimalDomestique("animal");
a.decire(); //AnimalDomestique nom=defaultAnimalName

var chat1 = new Chat("malo"); //var chat1 = new Chat();
chat1.ronronner(); //ronron
chat1.decire(); // Je suis un chat qui dort 14h AnimalDomestique nom=malo
chat1.parler(); // miaou miaou

var a2: AnimalDomestique = new Chien("Rantanplan");
a2.montLaGarde();
if(a2 instanceof Chien)
    (<Chien> a2 ).monterLaGarde();
a2.decire(); //Je suis un chien fonction=undefined AnimalDomestique nom=Rantanplan
a2.parler(); // whaouf whaouf //polymorphisme (Chien = sorte de Animal)
```

6.7. Interfaces en typescript

person.ts

```
interface Person {
    firstname: string;
    lastname: string;
}

function greeterPerson(person : Person) {
    return "Hello, " + person.firstname + " " + person.lastname;
}

//var user = {name: "James Bond", comment: "top secret"};
//incompatible avec l'interface Person (erreur détectée par tsc)

var user = {firstname: "James", lastname: "Bond" , country: "UK"};
//ok : compatible avec interface Person

msg = greeterPerson(user);
console.log(msg);

class Student {
    fullname : string;
    constructor(public firstname, public lastname, public schoolClass) {
        this.fullname = firstname + " " + lastname + "[" + schoolClass + "]";
    }
}

var s1 = new Student("cancre", "Ducobu" , "Terminale"); //compatible avec interface Person
msg = greeterPerson(s1); console.log(msg);
```

Rappel important : via le mot clef "**public**" ou "**private**" (au niveau des paramètres du constructeur), certains paramètres passés au niveau du constructeur sont automatiquement transformés en attributs/propriétés de la classe (ici "Student") qui devient donc compatible avec l'interface "Person". Au sein du langage "typescript", une **interface** correspond à la notion de "**structural subtyping**". Le véritable objet qui sera compatible avec le type de l'interface pourra avoir une structure plus grande.

Interface pour type précis d'objets :

```
interface ClockInterface {
    currentTime: Date;
    setTime(d: Date);
}

class Clock implements ClockInterface {
    currentTime: Date;
    setTime(d: Date) {
        this.currentTime = d;
    }
    //...
}
```

6.8. Exemple de "Generic" en typescript :

MyStack<T> (generic)

```
class MyStack<T> {
    private _items : T[] = [];

    constructor() {}

    public push(element:T){
        this._items.push(element);
    }

    public isEmpty() : boolean{
        return this._items.length == 0;
    }

    public pop() : T | undefined{
        if (this.isEmpty())
            throw "underflow / empty stack";
        return this._items.pop(); //remove and return top stack value
    }

    public peek(): T | undefined{
        if (this.isEmpty())
            throw "underflow / empty stack";
        return this._items[this._items.length - 1]; //return top stack value without remove it
    }
}
```

```
const stackOfNumber = new MyStack<number>();
stackOfNumber.push(1); stackOfNumber.push(4);
stackOfNumber.push(9); stackOfNumber.push(25);
```

```
const topVal : number | undefined = stackOfNumber.pop();
console.log("topVal="+topVal); //25
```

```
const stackOfString = new MyStack<string>();
stackOfString.push("aaa"); stackOfString.push("bbb");
stackOfString.push("ccc"); stackOfString.push("ddd");
```

```
const sTopVal : string | undefined = stackOfString.pop();
console.log("sTopVal="+sTopVal); //ddd
```

7. Notion de décorateurs "typescript"

Les "décorateurs typescript" correspondent à une fonctionnalité expérimentale de typescript et de es7 mais qui est *déjà utilisée par Angular >=2*.

Syntaxiquement introduit via la syntaxe **@decorateurXy()** , un décorateur "typescript" correspond à la fois :

- à une *variante du design pattern "décorateur"* (fonctionnalités/comportements enrichis)
- à une *variante des "annotations java"* (*programmation déclarative*)

Plus concrètement , un décorateur "typescript" :

- se code comme une fonction technique ayant une signature très particulière
- s'applique de manière déclarative en placant **@decorateurXy ou @decorateurXy()** ou **@decorateurXy(valParam1,valParam2)** au dessus d'un élément à enrichir ou paramétriser (classe , méthode, propriété , argument d'une méthode) .
- s'exécute au runtime comme une fonction de pré-traitement permettant d'altérer par métaprogrammation/réflexion le code ordinaire (constructeur , méthode, ...) appelé .

NB1 : La partie "decorator" est depuis longtemps en mode "expérimental" en typescript (et c'est toujours le cas en 2024) .

La normalisation met du temps à se mettre en place (le temps que la norme "EcmaScript" évolue pour intégrer les décorateurs en "pur javascript") : travail en cours ...

Exemple d'application de décorateurs :

```

@myLogClassDecorator()
class Cercle {

    @myLogPropertyDecorator
    public unite:string | undefined;

    constructor(public xC :number =0, public yC :number =0,
               public rayon :number =0){
    }

    @myLogMethodDecocator(true)
    description() : string{
        return "Cercle de centre ("+this.xC+","+this.yC+) et de rayon " + this.rayon;
    }

    @myLogMethodDecocator()
    perimetre() : number { return Math.PI * this.rayon * this.rayon;
    }

}
//Cercle.prototype.unite="m";

```

let c1=new Cercle(100,80,50);

```
//>>> myLogClassDecorator , New: Cercle is created
//>>> myLogPropertyDecorator , set newValue of propertyName=unite : ?
let descriptionC1= c1.description();
console.log("descriptionC1="+descriptionC1);
* >>myLogMethodDecocator intercept call of description() returnValue is CERCLE DE CENTRE (100,80) ET DE RAYON 50 *
//descriptionC1=CERCLE DE CENTRE (100,80) ET DE RAYON 50
c1.unite='cm';
//>>> myLogPropertyDecorator , set newValue of propertyName=unite : cm
//>>> myLogPropertyDecorator , get value of propertyName=unite : cm
console.log("unite de c1="+c1.unite); //unite de c1=CM
let perimetreC1= c1.perimetre();
//>>myLogMethodDecocator intercept call of perimetre() returnValue is 7853.981633974483
console.log("perimetre de c1 =" +perimetreC1); //perimetre de c1 =7853.981633974483
```

NB :

En tant que fonctionnalité expérimentale (et pas encore définitivement normalisée) , l'utilisation des décorateurs nécessite l'option **--experimentalDecorators** de tsc

ou bien "**experimentalDecorators": true** dans **tsconfig.json**

8. Modules "es6/es2015"

8.1. Modules es2015 = norme javascript

Les modules ES6 :

- ont une syntaxe simple et sont basés sur le découpage en fichiers (un module = un fichier),
- sont automatiquement en mode « strict » (rigoureux),
- offrent un support pour un chargement asynchrone et permet de générer des bundles "statiques" via rollup ou webpack .

Les modules doivent exposer leurs variables et méthodes de façon explicite. On dispose donc des deux mots clés :

- **export** : pour exporter tout ce qui doit être accessible en dehors du module,
- **import** : pour importer tout ce qui doit être utilisé dans le module (et qui est donc exporté par un autre module).

Exemple :

math-util.js

```
export function additionner(x , y) {  
    return x + y;  
}  
  
export function multiplier(x, y) {  
    return x * y;  
}
```

ou bien

```
function additionner(x , y) {  
    return x + y;  
}  
  
function mult(x, y) {  
    return x * y;  
}  
  
export { additionner, mult as multiplier };
```

dom-util.js

```
export class DomUtil {  
    static displayInDiv(divId,message){  
        document.querySelector('#'+divId).innerHTML = message;  
    }  
  
    static multilineMessage(...args){  
        //NB: la syntaxe ... permet de récupérer tous (ou bien les derniers) arguments (en nombre variable)  
        //sous forme de tableau . Cette syntaxe est permise en mode "strict" alors que la  
        //syntaxe Dom.multilineMessage.arguments est interdite en mode "strict" (dans module es6)  
        let nb_arg=args.length;  
        let messages=null;  
        if(nb_arg>=1) messages=args[0];  
        for(let i=1;i<nb_arg;i++)  
            messages+="  
        >"+args[i];  
        return messages;  
    }  
}
```

main.js

```
import { additionner as add, multiplier } from "./math-util.js";
import { DomUtil } from "./dom-util.js";

function carre(x){
    return multiplier(x,x) ;
}

/*
var msg1 = "Le carre de 5 est " + carre(5);  console.log(msg1);
var msg2 = "4 * 3 vaut " + multiplier(4, 3);  console.log(msg2);
var msg3 = "5 + 6 vaut " + add(5, 6);  console.log(msg3);
document.querySelector('#divA').innerHTML = msg1 + "<br/>" + msg2 + "<br/>" + msg3;
*/
DomUtil.displayInDiv('divA',
    DomUtil.multilineMessage(
        "Le carre de 5 est " + carre(5),
        "4 * 3 vaut " + multiplier(4, 3),
        "5 + 6 vaut " + add(5, 6)
    )
);
```

```
<html>
    <body>
        <script type="module" src="main.js"></script>
        <div id="divA"></div>
    </body>
</html>
```

Attention :

- `type="module"` est indispensable mais n'est supporté que par les navigateurs récents
- la page html et les modules javascripts doivent être téléchargés via http (par exemple via `http://localhost:3000/` et lite-server).

default export (one per module)

xy.js

```
export function mult(x, y) {  
    return x * y;  
}  
  
//export default function _or _object _or _class (ONE PER MODULE)  
export default {  
    name : "xy",  
    features : { x : 1 , y: 3 }  
}
```

main.js

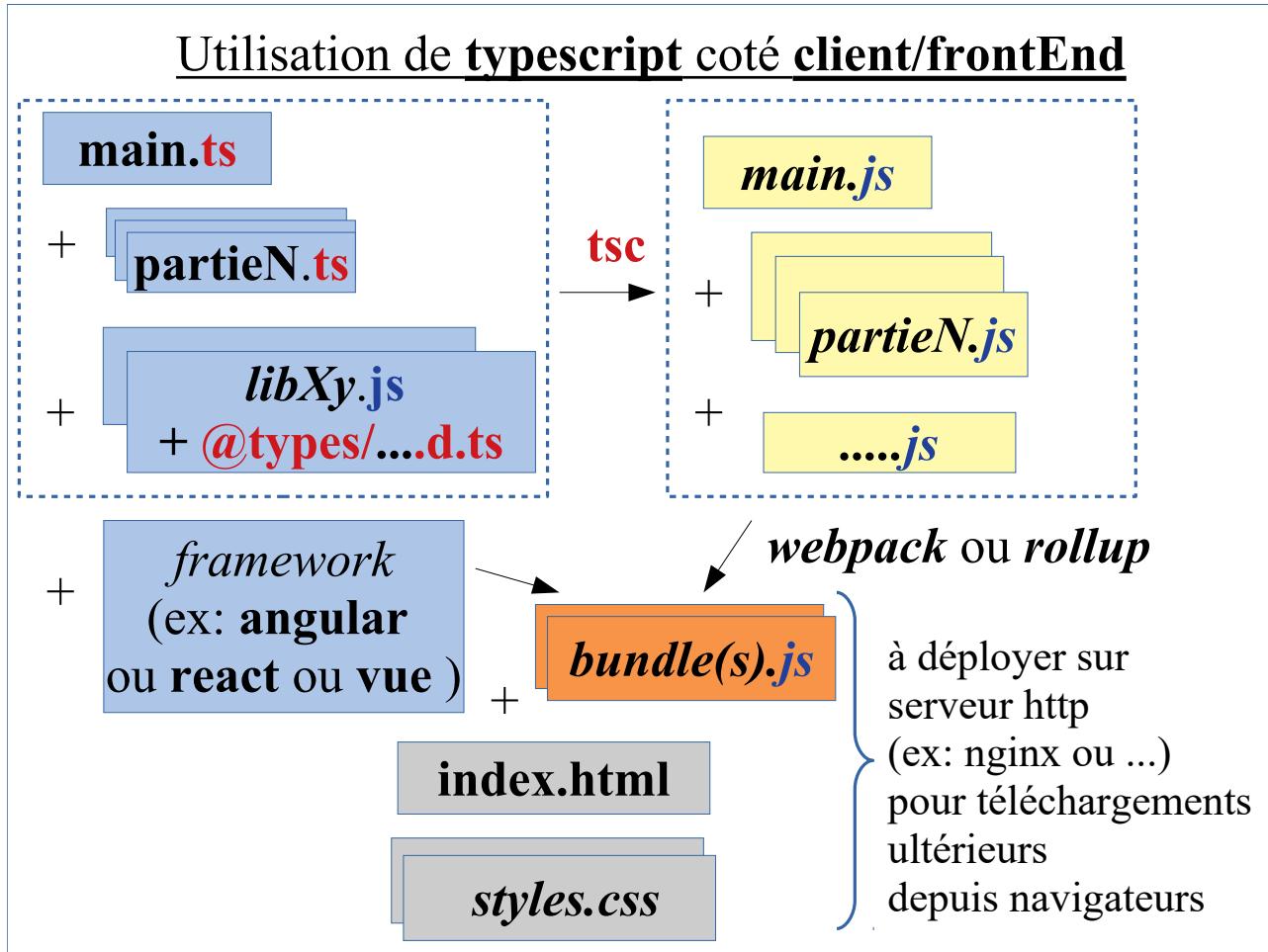
```
import xy , { mult } from "./xy.js";  
...  
let msg = xy.name + "--" + JSON.stringify(xy.features) + "--" + mult(3,4);
```

8.2. Modules en typescript

Seule différence importante par rapport aux modules ".js" .

import { ... } from "./xyz" (sans extension : ni .ts , ni .js)

8.3. Génération de bundles depuis modules (.js ou .ts)



Exemple de configuration de webpack :

webpack.config.js

```

const webpack = require("webpack");
const path = require("path");

//entry: "./src/index.js" ou entry: "./src/main.js" ou ...
let config = {
  entry: "./src/main.js",
  output: {
    path: path.resolve(__dirname, "./dist"),
    filename: "./main-bundle.js"
  }
}
module.exports = config;

```

9. WebComponent

9.1. Notion de WebComponent

Un "WebComponent" est un **composant web** dans un **format se voulant normalisé/standard (W3C)** que l'on peut *utiliser en pur "html/javascript"* sans se soucier de la technologie qui a servie à le développer.

Autrement dit un composant web codé avec angular peut par la suite être utilisé dans une application web qui n'est pas basée sur angular ou vice versa.

9.2. WebComponent élémentaire en pur javascript

hello-world-component.js

```
class HelloWorldComponent extends HTMLElement {  
    constructor() {  
        super();  
    }  
    connectedCallback() {  
        this.innerHTML = `<div style="color:red;font-size:24pt">Hello world!</div>`  
    }  
}  
  
customElements.define("hello-world-component", HelloWorldComponent)
```

with-web-component.html

```
<html>  
  <head>  
    <title>with-web-component</title>  
    <script src="./hello-world-component.js" ></script>  
  </head>  
  
  <body>  
    <h1>with-web-component</h1>  
  
    <!-- NB:cette balise personnalisée correspond au webcomponent  
        élémentaire codé dans le fichier hello-world-component.js -->  
    <hello-world-component></hello-world-component>  
  </body>  
</html>
```

Affichage résultant de ce code dans tous les navigateurs récents (chrome, firefox, edge) :

with-web-component

Hello world!

9.3. Code source des exemples

Référentiel <https://github.com/didier-mycontrib/angular12plus>
et partie `vanilla-js\basic-web-component`

9.4. Web Component avec attributs

`with-attr-component.js`

```
class WithAttrComponent extends HTMLElement {
  constructor() {
    super();
    this.name = "?"; //default value
    this.color = "red"; //default value
  }

  //static get observedAttributes() to define list of observedAttributes
  static get observedAttributes() {
    return ['name', 'color'];
  }

  //attributeChangedCallback(property, oldValue, newValue) callback
  // to update new changed attribute/property value
  attributeChangedCallback(property, oldValue, newValue) {
    if (oldValue === newValue) return;
    this[property] = newValue;
  }

  connectedCallback() {
    this.innerHTML =
      `<div style="color:${this.color};font-size:24pt">Hello ${this.name}!</div>`;
  }
}

customElements.define("with-attr-component", WithAttrComponent)
```

Utilisation :

```
<html><head> ...
  <script src=".//with-attr-component.js" ></script>
</head>
<body>
  <with-attr-component name="toto" color="blue"></with-attr-component>
  <with-attr-component name="titi" color="green"></with-attr-component>
...

```

Hello toto!
Hello titi!

9.5. Web Component avec "shadow DOM" et "slot"

with-shadow-component.js

```
class WithShadowComponent extends HTMLElement {
  constructor() {
    super();
  }

  connectedCallback() {
    const shadow = this.attachShadow({ mode: 'closed' });//only accessible by web component
    //const shadow = this.attachShadow({ mode: 'open' });//accessible outside with Element.shadowRoot
    shadow.innerHTML =
      <style>
        div {
          text-align: center;
          font-weight: normal;
          padding: 1em;
          background-color: #eee;
          border: 1px solid blue;
        }
        .c1 { color: green; }

        :host {
          display: block;
          background-color: lightgreen;
          padding: 1em;
        }
      </style>
      <div class="c1 c2">With-shadow-DOM <slot name="comment"></slot></div>
    '
  }
}

customElements.define("with-shadow-component", WithShadowComponent)

//NB: le selecteur :host permet de cibler la racine du webComponent
//ici la balise <with-shadow-component></...> contenant elle même la <div>...</>
```

Utilisation :

<pre>... <p class="c1 c2">before</p> <with-shadow-component> *** </with-shadow-component> <p class="c1 c2">after</p> et .c1 { color:blue;}</pre>	<p><u>before</u></p> <p>The diagram illustrates the shadow DOM structure. A green rectangular box labeled "before" contains a white rectangular box labeled "With-shadow-DOM ***". This white box is enclosed in a blue border. Below the green box is another white rectangular box labeled "after". The entire structure is enclosed in a red border.</p> <p><u>after</u></p>
--	---

9.6. toggle-panel webComponent (slot , shadow , attribute, ...)

toggle-panel-component.js

```
class TogglePanelComponent extends HTMLElement {
  constructor() {
    super()
    this.label="TogglePanelComponent"; //default value
    this.toggleP = false;
  }

  initComponentInnerHtml(){
    this.componentCssTemplateString = `
      .my-card { margin-top: 0.1em; margin-bottom: 0.1em; }
      .my-card-header { border-top-left-radius: 0.3em; border-top-right-radius: 0.3em;
        padding: 0.1em; margin-bottom: 0px; }
      a { text-decoration: none; }
      .my-card-body { border: 0.1em solid blue; border-bottom-left-radius: 0.3em;
        border-bottom-right-radius: 0.3em; padding: 0.2em; }
      .my-bg-primary { background-color: blue; }
      .my-text-light { color : white; }
      .my-icon { color : blue; background-color: white; margin: 0.2em;
        padding-left: 0.2em; padding-right : 0.2em;
        min-width: 1em; font-weight: bold; }
      .my-collapse { display : none; }
      .my-show { display : block ; }

      this.componentHtmlTemplateString = `
        <div class="my-card">
          <h4 class="my-card-header my-bg-primary" id="myCardHeader">
            <a class="my-text-light" >
              <span class="my-icon" id="myIconShow">+</span>
              <span class="my-icon" id="myIconCollapse" style="display:none">-</span>${this.label}
            </a>
          </h4>
          <div id="myCardBody" class="my-card-body my-collapse">
            <slot></slot>
          </div>
        </div>
      `

      this.componentInnerHtml = `
        <style>
        ${this.componentCssTemplateString}
        </style>

        ${this.componentHtmlTemplateString}

    `

    // component attributes
    static get observedAttributes() {
      return ['label'];
    }
  }
}
```

```

}

// attribute change
attributeChangedCallback(property, oldValue, newValue) {
    if (oldValue === newValue) return;
    this[ property ] = newValue;
}

connectedCallback() {
    const shadow = this.attachShadow({ mode: 'closed' }); //only accessible by web component
    //const shadow = this.attachShadow({ mode: 'open' }); //accessible outside with Element.shadowRoot
    this.initComponentInnerHtml();
    shadow.innerHTML = this.componentInnerHtml;
    shadow.getElementById("myCardHeader").addEventListener("click", function(evt){
        this.toggleP = ! this.toggleP;
        //console.log("toggleP="+this.toggleP);
        const myCardBody = shadow.getElementById("myCardBody");
        myCardBody.setAttribute("class", "my-card-body my-collapse"
            + (this.toggleP?"my-show":""))
        const myIconShow = shadow.getElementById("myIconShow");
        const myIconCollapse = shadow.getElementById("myIconCollapse");
        myIconShow.style.display=this.toggleP?'none':'inline-block';
        myIconCollapse.style.display=this.toggleP?'inline-block':'none';
    });
}
}

customElements.define("toggle-panel-component", TogglePanelComponent)

```

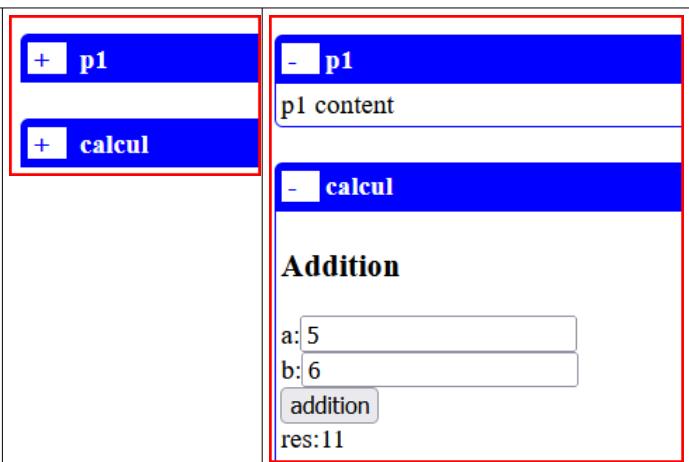
Utilisation :

```

<toggle-panel-component label="p1">
    <div>p1 content</div>
</toggle-panel-component>

<toggle-panel-component label="calcul">
    ...
</toggle-panel-component>

```



III - Structure et fondamentaux Angular

1. Environnement de développement pour Angular

1.1. Environnement de développement minimum (depuis v2)

Environnement de développement Angular 2+

① **ng new my-app**
pour générer *my-app*
comportant
package.json

*Node
Package
Manager
(de NodeJs)*

NPM

② **téléchargement des technologies**
"javascript" (tsc, angular, ...)
via **npm install**

③ écriture du code (.ts , .css , .html , ...)
via un éditeur quelconque

④ utilisation des technologies
téléchargées : "**tsc**" , "**ng-serve**" ,
"webpack" ,
"karma" , ...
pour **tester** le code de l'application

⑤ **packaging** de l'application
sous forme de **bundle(s)** pour une
mise en production sur un vrai
serveur http (ex : **nginx** ou ...)

Bien que Angular soit une technologie qui s'exécute "coté navigateur" , l'environnement de développement s'appuie sur la sous partie "**npm**" de **nodeJs**.

L' **éco-système "npm"** sert essentiellement à télécharger et exécuter les technologies de développement nécessaires pour angular ("tsc" , "@angular/cli" , ...") .

A l'époque des premières "v2" de Angular , le mode de développement préconisé consistait à directement partir d'un fichier "package.json" récupéré par copier/coller et éventuellement adapté pour ensuite lancer "npm install" , etc ...

Bien qu'encore possible actuellement , ce mode de développement basique et direct est de moins en moins utilisé au profit de l'utilitaire en ligne de commande "@angular/cli" (exposé au sein du prochain paragraphe).

1.2. Développement Angular basé sur @angular/cli (ng)

incontournable @angular/cli

S'installant via **npm install -g @angular/cli** , angular CLI est un **utilitaire en ligne de commandes** (s'appuyant sur *npm* et *webpack*) permettant de gérer toutes les phases d'un projet angular :

ng new my-app -- création d'une nouvelle appli angular4+

ng g component cxy -- génération d'un nouveau composant

ng g service sa -- génération d'un nouveau service

ng g ...

ng serve -- build en mémoire + démarrage serveur de test

ng build -- construction de bundles (pour déploiement et production)

ng ...

Angular-CLI est maintenant officiellement préconisé sur le site officiel de Angular. Autant faire comme tout le monde et utiliser cette façon de structurer et construire une application angular.

Installation (en mode global) de angular-cli via npm : **npm install -g @angular/cli**
NB : si besoin , upgrade préalable de npm via npm install npm@latest -g ou bien carrément désinstaller nodeJS et réinstaller une version plus récente.

La création d'une nouvelle application s'effectue via la ligne de commande "**ng new my-app**".
 Cette commande met pas mal de temps à s'exécuter (beaucoup de fichiers sont téléchargés).

Au sein de l'arborescence des répertoires et fichiers créés (voir ci-après) :

* src/assets (ou bien /public) est prévu pour contenir des ressources annexes (images ,) qui seront automatiquement recopiées/packagées avec l'application construite.

À l'époque "angular 11" avec module "app" :

/	dans src	dans src/app
dist e2e node_modules src .editorconfig angular.json package.json package-lock.json proxy.conf.json README.md tsconfig.json tslint.json	app assets environments browserslist favicon.ico index.html karma.conf.js main.ts polyfills.ts styles.scss test.ts tsconfig.app.json tsconfig.spec.json tslint.json	app.component.html app.component.scss app.component.spec.ts app.component.ts app.module.ts app-routing.module.ts

À l'époque "angular 17, 18" en mode standalone et ssr:

/	dans src	dans src/app
node_modules src .editorconfig .gitignore angular.json package.json package-lock.json README.md README_v17.txt server.ts tsconfig.app.json tsconfig.json tsconfig.spec.json et éventuellement public/favicon.ico (v18+)	app assets favicon.ico index.html main.server.ts main.ts styles.scss	app.component.html app.component.scss app.component.spec.ts app.component.ts app.config.server.ts app.config.ts app.routes.ts

Première Installation de @angular/cli

Eventuelle installation de nodeJs et npm (si nécessaire):

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

Eventuelle installation de typescript (si nécessaire):

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

npm install -g typescript

installation de angular-cli :

Si ng -help est une commande non reconnue (dans CMD) alors lancer la commande suivante :

npm install -g @angular/cli

Installer si nécessaire l'IDE Visual Studio code.

Création et lancement d'une application angular

Dans **c:/.../tp-js** ou ailleurs lancer la commande suivante

ng new my-app (par défaut sans `@NgModule`, en standalone depuis v17)
ng new --no-standalone my-app (depuis v17 pour my-app avec `@NgModule`)

- choisir "y" à l'éventuelle question "activer le routing angular"
 - choisir "scss" comme format de feuilles de styles
 - choisir "y" ou pas à l'éventuelle question "activer ssr"

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 .

NB: En phase de développement, le mini serveur démarré par `ng serve` re-génère 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 re-démarrage 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**

Principales lignes de commandes de **ng** (angular-cli) :

ng new my-app , cd my-app	Création d'une nouvelle application "my-app" .
ng serve	Lancement de l'application en mode développement (watch & compile file , launch server,) → URL par défaut : http://localhost:4200
ng build	Construction de l'application (par défaut en mode --prod depuis la version 12)
ng help	Affiche les commandes et options possibles
ng generate ... (ou ng g ...)	Génère un début de code pour un composant , un service ou autre (selon argument précisé)
ng test	Lance les tests unitaires (via karma)
<i>ng e2e (avant version 12) installer et utiliser cypress depuis la v12</i>	Lance les tests "end to end" / "intégration"
...	...

Type de composants que l'on peut générer (début de code) :

Scaffold (échafaudage)	Usage (ligne de commande)
Component	ng g component my-new-component ng g c my-new-component
Directive	ng g directive my-new-directive
Pipe	ng g pipe my-new-pipe
Service	ng g service my-new-service ng g s my-new-service
Class	ng g class my-new-class
Interface	ng g interface my-new-interface
Enum	ng g enum my-new-enum
Module	ng g module my-module

NB : principales options utiles pour **ng g component** sont les suivantes :

- style **css** (ou --style **sass**) ou ...
- flat (pas de sous répertoire)
- skip-tests (ne génère pas de fichier ...spec.ts)
- inline-template (ne génère pas de fichier .html)
- inline-style (ne génère pas de fichier .css , possible en css seulement)

NB : **ng serve** construit l'application entièrement en mémoire pour des raisons d'efficacité / performance (on ne voit aucun fichier temporaire écrit sur le disque) .

ng build génère quant à lui des fichiers dans le répertoire my-app/**dist** .

Contenu du répertoire my-app/**dist**/my-app après la commande "**ng build**" :

3rdpartylicenses.txt	13/03/2022 20:45	Document texte	16 Ko
favicon.ico	23/02/2022 17:17	Fichier ICO	1 Ko
index.html	13/03/2022 20:45	Firefox HTML Doc...	3 Ko
main.e690766f54729005.js	13/03/2022 20:45	Fichier de JavaScript	672 Ko
polyfills.96fd61f4191dac28.js	13/03/2022 20:45	Fichier de JavaScript	37 Ko
runtime.b5b2d38d33a5a587.js	13/03/2022 20:45	Fichier de JavaScript	2 Ko
styles.a5d4ddde1dbf3b1d.css	13/03/2022 20:45	Fichier CSS	159 Ko

Migration globale de angular-cli vers version plus récente :

```
npm uninstall -g angular-cli  
npm cache verify  
npm install -g @angular/cli@latest
```

Migration locale du seul projet courant vers version plus récente :

```
ng update @angular/cli @angular/core [ --to=13.3.0 ]
```

Ajout de bibliothèque(s) javascript :

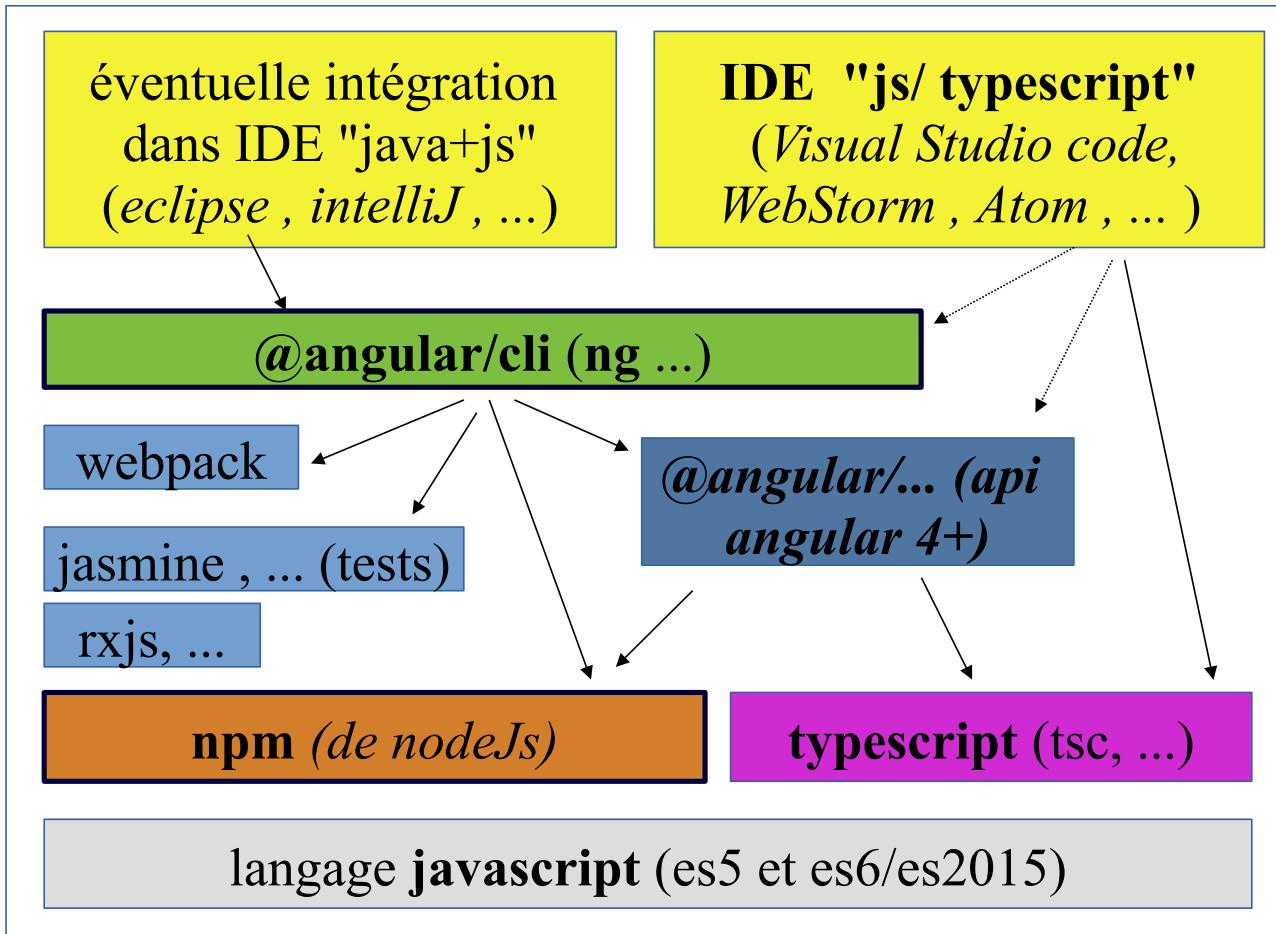
NB : si l'on souhaite utiliser conjointement certains fichiers javascripts complémentaires (exemple pas conseillé mais quelquefois compatible : "jquery...js" et "bootstrap.min.js") on peut :

- 1) placer un répertoire "js" à côté de node-modules (ou bien utiliser un jquery récupéré par npm)
- 2) adapter le fichier **angular.json** de la façon suivante :

```
"scripts": [ "./js/jquery-3.2.1.min.js" ,  
           "./js/bootstrap.min.js"],
```

NB : bien que techniquement possible , l'ajout de jquery.js à un projet angular est **très déconseillé !!!**

1.3. IDE et éditeurs de code pour angular



1.4. Configuration "npm" et "@angular/cli" pour Angular

Voici un exemple de fichier "**package.json**" généré par "**ng new my-app**" :

```
{
  "name": "my-app",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "watch": "ng build --watch --configuration development",
    "test": "ng test",
    "serve:ssr:my-app": "node dist/my-app/server/server.mjs"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "^18.2.0",
    "@angular/cdk": "^18.2.8",
    "@angular/common": "^18.2.0",
    "@angular/compiler": "^18.2.0",
    "@angular/core": "^18.2.0",
    "@angular/forms": "^18.2.0",
    "@angular/material": "^18.2.8",
    "@angular/platform-browser": "^18.2.0",
    "@angular/platform-browser-dynamic": "^18.2.0",
    "@angular/platform-server": "^18.2.0",
    "@angular/router": "^18.2.0",
    "@angular/ssr": "^18.2.7",
    "bootstrap": "^5.3.3",
    "bootstrap-icons": "^1.11.3",
    "express": "^4.18.2",
    "rxjs": "~7.8.0",
    "tslib": "^2.3.0",
    "zone.js": "~0.14.10"
  },
  "devDependencies": {
    "@angular-devkit/build-angular": "^18.2.7",
    "@angular/cli": "^18.2.7",
    "@angular/compiler-cli": "^18.2.0",
    "@types/express": "^4.17.17",
    "@types/jasmine": "~5.1.0",
    "@types/node": "^18.18.0",
    "jasmine-core": "~5.2.0",
    "karma": "~6.4.0",
    "karma-chrome-launcher": "~3.2.0",
    "karma-coverage": "~2.2.0",
    "karma-jasmine": "~5.1.0",
    "karma-jasmine-html-reporter": "~2.1.0",
    "typescript": "~5.5.2"
  }
}
```

NB : au sein de cet exemple , l'extensions *bootstrap* a été ajoutée via *npm install -s*

Voici un exemple de fichier **tsconfig.json** généré :

```
{
  "compileOnSave": false,
  "compilerOptions": {
    "baseUrl": "./",
    "outDir": "./dist/out-tsc",
    "forceConsistentCasingInFileNames": true,
    "strict": true,
    "noImplicitOverride": true,
    "noPropertyAccessFromIndexSignature": true,
    "noImplicitReturns": true,
    "noFallthroughCasesInSwitch": true,
    "sourceMap": true,
    "declaration": false,
    "downlevelIteration": true,
    "experimentalDecorators": true,
    "moduleResolution": "bundler",
    "importHelpers": true,
    "target": "ES2022",
    "module": "ES2022",
    "lib": [ "ES2022", "dom" ]
  },
  "angularCompilerOptions": {
    "enableI18nLegacyMessageIdFormat": false,
    "strictInjectionParameters": true,
    "strictInputAccessModifiers": true,
    "strictTemplates": true
  }
}
```

Ce fichier servira à paramétrer le comportement du pré-processeur (ou pré-compilateur) "**tsc**" permettant de transformer des fichiers ".ts" en fichiers ".js" .

Fichier "**main.ts**" (généré et ré-exploité par @angular/cli) pour charger et démarrer le code de l'application :

En version --no-standalone (avec AppModule) :

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { NgModule } from './app/app.module';

platformBrowserDynamic().bootstrapModule(AppModule, {
  ngZoneEventCoalescing: true
})
  .catch(err => console.error(err));
```

En version standalone (sans module) :

```
import { bootstrapApplication } from '@angular/platform-browser';
import { appConfig } from './app/app.config';
import { AppComponent } from './app/app.component';

bootstrapApplication(AppComponent, appConfig)
  .catch((err) => console.error(err));
```

Page principale **index.html** (*qui sera retraitée/enrichie via **ng serve** ou **ng build***)

```
<html lang="en">
<head>
<meta charset="utf-8">
<title>myApp</title>
<base href="/">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
<app-root></app-root>
</body>
</html>
```

NB : les fichiers index.html et styles.css peuvent être un petit peu personnalisés mais le gros du code de l'application sera placé dans les sous-réertoires "app" et autres .

1.5. Exemple de code élémentaire pour une application angular

app/app.component.ts

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

@Component({
  selector: 'app-root',
  template: `<h1>My First {{message}} .. </h1>
    <table border="1">
      <tr> <th>i</th> <th>i*i</th> </tr>
      <tr *ngFor="let i of values" >
        <td>{{i}}</td> <td>{{i*i}}</td>
      </tr>
    </table>`}

export class AppComponent {
  message: string;
  values: number[] = [1,2,3,4,5,6,7,8,9];
  constructor(){
    this.message = "Angular 2 App";
  }
}
```

My First Angular 2 App ..

i	i*i
11	
24	
39	
416	
525	
636	
749	
864	
981	

NB : dans **@Component()**,

- soit **templateUrl: 'app.component.html'** (*template html dans fichier annexe*)
- soit **template: `<h1>... {{message}}</h1> ...`** (*contenu direct d'un petit template html entre quotes inverses*)

à lancer et tester via **ng serve**

et **http://localhost:4200**

...

1.6. Styles css globaux et styles spécifiques à un composant .

xyz.component.ts

```
...
@Component({
  selector: 'xyz',
  templateUrl: './xyz.component.html',
  styleUrls: ['./xyz.component.css']
})
export class XyzComponent implements OnInit {
...
}
```

xyz.component.css

```
input.ng-valid[required] {
  border-left: 5px solid #42A948; /* green */
}
input.ng-invalid {
  border-left: 5px solid #a94442; /* red */
}
.errMsg{
  font-style: italic;
}
```

Ces classes de styles css ne sont utilisées que par le composant "xyz" . Il n'y aura pas d'effet de bord (pas d'éventuels conflits ou perturbations) avec d'autres composants .

Pour utiliser des styles css au niveau global (toute l'application) , on peut dans un contexte "angular-cli" les référencer dans la partie "styles" de *.angular.json*

```
...
"styles": [
  "src/styles.css", "src/public/css/xyz.css"
],
...
```

NB : Les chemins sont à exprimer de façon relative à **la racine de l'appli angular** (la où est placé *package.json*) .

1.7. Lien classique entre angular 12+ et bootstrap-css 4 ou 5

Avertissement préalable :

Bootstrap-css vient assez récemment de basculer de la version 4.x à la nouvelle version 5.x
Bootstrap-css est tout à fait facultatif, on peut s'en passer en s'appuyant sur des flex-box .

Téléchargement (et installation dans l'appli) de bootstrap-css via npm :

npm install -s bootstrap

et éventuellement (pour paquets d'icônes) :

npm install -s bootstrap-icons (pour V5)

ou bien

npm install -s @fortawesome/fontawesome-free (pour V4 ou ...)

NB : par défaut , la dernière version stable de bootstrap-css est téléchargée (actuellement 5+).
on peut (si besoin) préciser la version de bootstrap souhaitée lors du "npm install" .

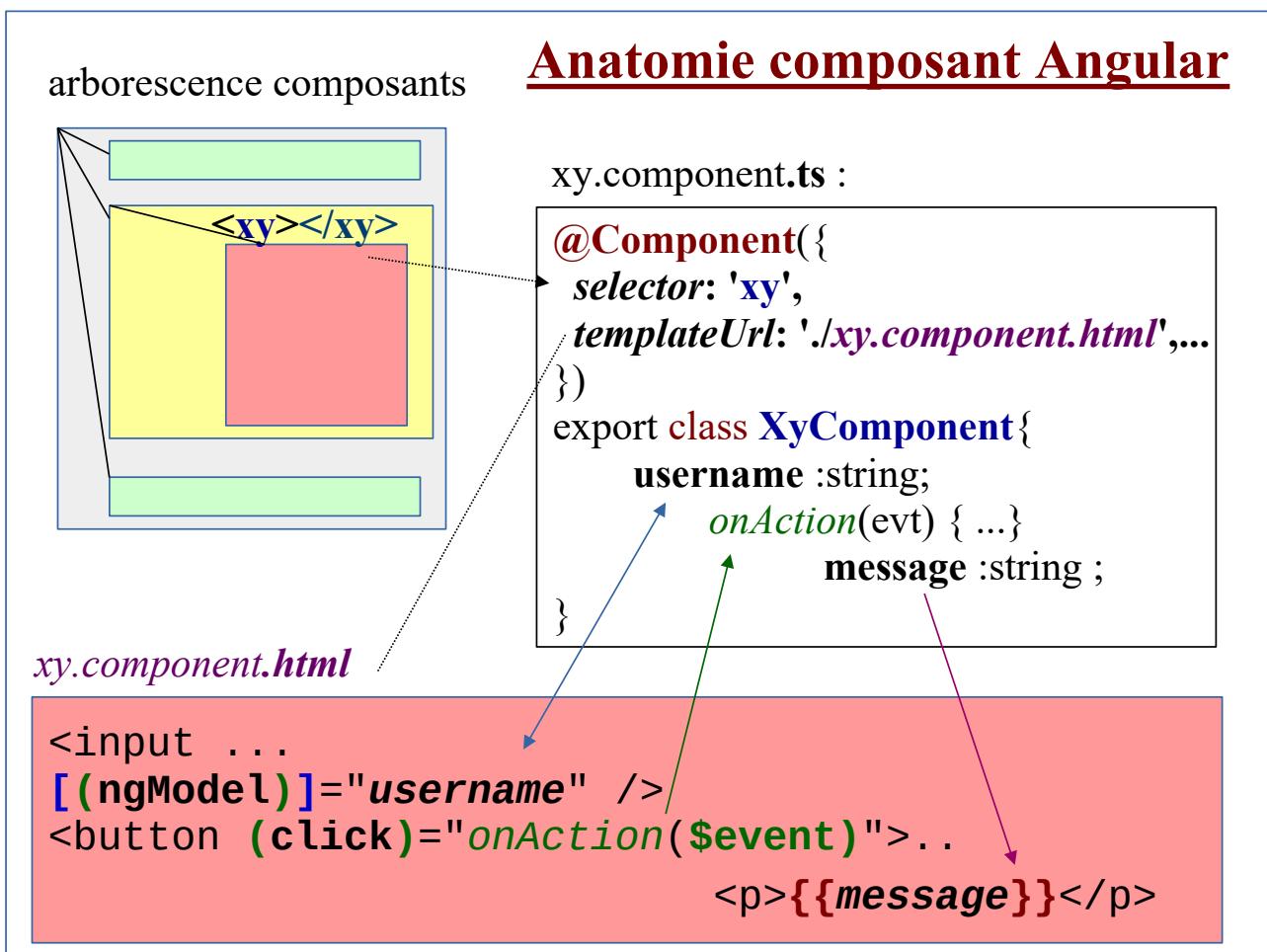
dans angular.json :

```
"styles": [  
    "src/styles.scss" ,  
    "node_modules/bootstrap/dist/css/bootstrap.min.css",  
    "node_modules/bootstrap-icons/font/bootstrap-icons.css" ou bien  
    "node_modules/@fortawesome/fontawesome-free/css/all.min.css"  
,]
```

Petit test dans un ...component.html:

```
<input type="button" value="ok" class="btn btn-primary" />  
<i class="fa fa-heart" aria-hidden="true" style="color: red;"></i>  
ou bien  
<i class="bi-arrow-down-circle-fill"></i>  
  
<!-- bi-... pour bootstrap-icons et fa-.... pour fontawesome -->
```

2. Anatomie élémentaire d'un composant angular



Chaque **composant** (visuel) d'une application Angular est constitué de plusieurs fichiers complémentaires (par défaut rangés dans un même sous-répertoire) :

- **xy.component.ts** (classe TypeScript du composant)
- **xy.component.html** ("template" HTML du composant)
- **xy.component.css** (ou .scss) (styles CSS spécifiques au composant)
- **xy.component.spec.ts** (spécifications pour tests unitaires)

Le fichier **xy.component.ts** correspond à la **structure orientée objet du composant** et comporte généralement des propriétés / attributs / données internes et des méthodes événementielles.

Remarque : la valeur de **selector :** (dans la décoration `@Component()` du fichier `xy.component.ts`) correspond au nom de la nouvelle balise qui sera associée à ce composant et qui permettra d'accrocher / insérer ce composant dans le template HTML d'un composant parent.

Le fichier `xy.component.html` appelé "template" HTML correspond à la **représentation HTML + Angular du composant** et correspondra, après analyse et traitement par Angular, à une partie de l'arbre DOM de la page HTML.

Un "template" HTML Angular comporte, en plus des syntaxes HTML standardisées, des

paramétrages spécifiques au framework Angular :

- Expressions : `{ { . . . } }`
- "Bindings" de propriétés : `[(ngModel)] = " . . . "`
- Déclenchement de méthodes événementielles : `(click) = "onAction()"`

La racine de l'arborescence des composants est la balise `<app-root></app-root>` de la page `src/index.html`. Ainsi, `app-root` est la valeur du sélecteur du composant principal (`AppComponent` dans `app.component.ts`, `app.component.html`, ...)

Attention : Pour qu'il soit pris en compte, chaque composant de l'application doit être **enregistré** quelque part (ex : dans la partie "`imports: [. . .]`" d'un composant parent ou dans `src/app/app.route.ts` ou ailleurs).

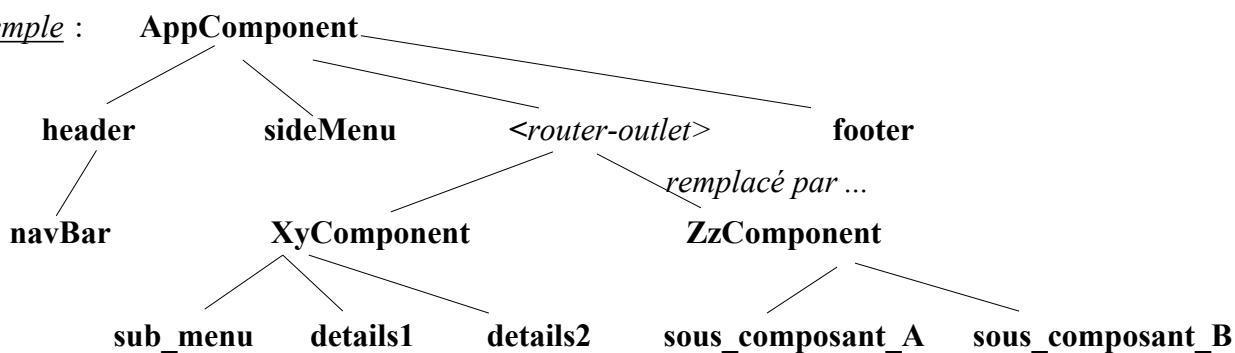
On pourra éventuellement prévoir un sous répertoire spécial (ex : "`src/app/common`") permettant de ranger tous les éléments partagés au niveau de l'ensemble des composants de l'application :

<pre> <common> > data > directive > gard > interceptor > pipe > service > util > validator </pre>	<p><i>src/app/common/data</i> (ou <i>common/model</i>) permet de ranger des classes de données qui sont à la fois manipulées par des composants (ex : formulaires de saisies) et à la fois manipulés par des services dans le cadre des appels de WS-REST (DTO JSON) .</p> <p><i>src/app/common/service</i> permet de ranger des services angular qui sont par nature partagés/partageables .</p>
---	--

3. Structure arborescente d'une application Angular 2+

Une application "angular2+" est structurée comme un **arbre de composants**.

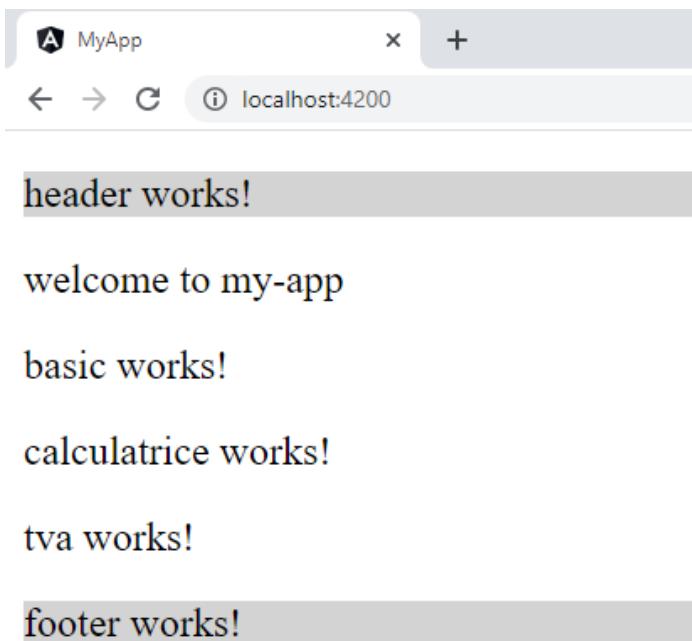
Exemple :



4. Arborescence de composants (TD)

Ce TD va permettre de créer de nouveaux composants Angular et de les rattacher entre eux.

- Revenir, si besoin, sur le projet **my-app** (Angular) via un File/Open Folder de Visual Studio Code
- Relancer, si besoin, ng serve au sein d'un terminal (nouveau ou pas)
- Au sein d'un **nouveau terminal** de Visual Studio Code, lancer les commandes suivantes dans l'ordre indiqué ci-dessous :
 - **ng g component header**
 - **ng g component footer**
 - **ng g component basic**
- Se placer dans **src/app/basic** (via **cd**) pour lancer les commandes suivantes :
 - **ng g component calculatrice**
 - **ng g component tva**
- Lire les messages affichés par les commandes précédentes **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**)
- Dans **src/app/header/header.component.ts**, repérer la valeur **app-header** du sélecteur
- Ajouter les sous-composants **<app-header></...>**, **<app-basic></...>** et **<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
- Ajouter si besoin les dépendances inter-composants dans les parties imports:[] de **@Component()** et Tester via <http://localhost:4200>



Etats des principaux fichiers à ce stade :

- Fichier `src/app/header/header.component.html`

```
<p class="entete">header works!</p>
```

- Fichier `src/app/header/header.component.css`

```
.entete {background-color: lightgrey;}
```

- Fichier `src/app/basic/basic.component.html`

```
<p>basic works!</p>
<app-calculatrice></app-calculatrice>
<app-tva></app-tva>
```

- Fichier `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>
```

Arborescence générée (selecteurs et composants) :

index.html

app-root (`app.component`)
app-header (`header.component`)
app-basic (`basic.component`)
 app-calculatrice (`calculatrice.component`)
 app-tva (`tva.component`)
app-footer (`footer.component`)

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

5. Structure d'une application angular 2+

5.1. Programmation "orientée objet" et "modularité par classe"

Selon une logique proche de celle de nodeJs , une application "Angular" peut s'appuyer sur les mots clefs "**export**" et "**import**" (de TypeScript et de ES2015) de façon à être construite sur une base **modulaire** .

Chaque composant élémentaire est un fichier à part. (dans le cas d'un composant visuel , il pourra y avoir des fichiers annexes ".html" , ".css") .

Il **exporte quelque-chose** (classe , interface , données , ...).

Un composant angular doit importer un ou plusieurs autre(s) module(s) ou classe(s) / composant(s) s'il souhaite **avoir accès aux éléments exportés** et les utiliser. Le référencement d'un autre composant s'effectue généralement via un chemin relatif commençant par "./" .

Le cœur d'**angular** est codé à l'intérieur de **librairies de composants prédefinis regroupés en modules**.

Les modules des "librairies" prédefinies d'angular sont par convention préfixés par "@angular" .

Exemples :

```
import {Component}      from '@angular/core';
import {HttpClient}    from '@angular/common/http';
import {Observable}    from 'rxjs';
import {BrowserModule} from '@angular/platform-browser'
import {OnInit}        from '@angular/core';
import {Router, RouteParams}   from '@angular/router';
...
...
```

5.2. Rappels des principaux types de composants "angular":

Module fonctionnel maintenant facultatif	Ensemble de composants et/ou de services (généralement placés dans un même répertoire) et chapeauté par une classe décorée via @NgModule .
Component <i>(composant visuel)</i>	Composant visuel de l'application graphique (associé à une vue basée elle même sur un template) – généralement spécifique à une application <u>NB:</u> un composant angular est souvent vu comme une nouvelle balise/sélecteur (ex : <app-header></app-header>)
Directive	Elément souvent réutilisable permettant de générer ou altérer dynamiquement une partie de l'arbre DOM . NB: une fois programmée, une directive peut s'utiliser sur tout un tas de balise/élément html (ex: sur <p>, sur <div>, sur ...) Une directive s'utilise souvent comme un nouvel attribut spécial (ex : [highlight]=""yellow"" , *ngIf="...") 2 types de directives : "structurelles" , "attribut"
Service <i>(invisible , potentiellement</i>	Un service est un élément invisible de l'application (qui rend un certain service) en arrière plan (ex : accès aux données , configuration,

<i>partagé)</i>	<i>calculateur , ...)</i>
------------------	---------------------------

6. Modules applicatifs (maintenant facultatifs)

NB :

- **Au sein des versions récentes d'angular (17, 18, ...) les modules sont devenus facultatifs. Une application sans module est qualifiée "standalone" .**
- **Au sein des anciennes versions d'angular (2,...,16) , les modules étaient obligatoires (au moins un module principal app.module.ts)**

Dans le contexte d'une application angular , on appelle "**module angular**" une **sous partie importante de l'application** (correspondant généralement à un répertoire ou sous répertoire).

Chaque **module angular** comporte un fichier principal **xyz.module.ts** comportant la décoration **@NgModule**.

Dans certains cas techniques , un **module (secondaire ou annexe)** correspond à un seul fichier (comportant une décoration @NgModule) : module auxiliaire pour configurer le routage (app-routing.module.ts , ...)

Une grosse application peut comporter plusieurs grands modules complémentaires (ex : partie principale publique "app" , partie réservée à l'administrateur "admin" , partie/espace réservé(e) à un utilisateur connecté et ayant tel rôle .

Un module peut également permettre de rassembler un ensemble de services communs spécifiques à l'application (ex : Authentication , Session , SharedData , ...)

Finalement, un module peut rassembler un ensemble de composants réutilisables (à la manière des extensions "material" , "ionic" ou "primeNG").

La **classe principale** d'un module Angular doit être (par convention) placée dans un fichier **xyz/xyz.module.ts** où **xyz** est le **nom du module** (ex : "app").

Cette classe principale doit être décorée par **@NgModule** .

providers: liste explicite des "composants services" qui seront rendus accessible à l'ensemble des composants de ce module. (NB : depuis angular 6 et le paramétrage <code>{ providedIn: 'root' }</code> de <code>@Injectable()</code> tous les services du module courant sont <u>implicitement rendus accessibles</u> à tous les composants du module)	<pre>import { NgModule } from '@angular/core'; import { BrowserModule } from '@angular/platform-browser'; @NgModule({ imports: [BrowserModule, FormsModule], providers: [Logger], declarations: [AppComponent, XyComponent], exports: [], bootstrap: [AppComponent] }) export class XyzModule { }</pre>
declarations : liste des classes plutôt visuelles (composants, directives,pipes) appartenant au module	
exports : sous ensemble des "déclarations" qui seront visibles par d'autres modules	

imports: liste de modules (prédéfinis/techniques , extensions , ...) dont le composants internes sont rendus accessibles au éléments du module courant

bootstrap: vue principale ("root", "main" , ...) (pour module principal seulement)

6.1. Module applicatif unique "app" pour petite application angular

NB: par défaut , une application angular "avec module" (≤ v16 ou bien générée par **ng new --no-standalone** ...) ne comporte qu'un seul module applicatif (répertoire **src/app** et fichier **app.module.ts**)

Une petite application angular peut se contenter d'un seul module.

Une application angular de grande taille devrait idéalement être décomposée en plusieurs modules complémentaires (chargés dynamiquement en mémoire en mode "lazy") de façon à démarrer rapidement .

6.2. Modules multiples pour grande application angular

Dès que l'on a besoin de développer une application angular de moyenne ou grande taille, on a tout intérêt à décomposer l'application en modules complémentaires de façon à :

- s'y retrouver dans une structure bien carrée et modulaire.
- optimiser les performances au démarrage (chargement du code au fur et à mesure des navigations si "lazy loading") .

Structure classique (avec variantes) :

src/app/

core (ou "**root**" : module principal/noyau/racine avec singltons , home/welcome component, ...)

shared (module partagé de choses qui seront réutilisées par d'autres modules)

xxx (features_module , ex: *bases*)

yyy (features_module , ex: *devises*)

Chacun de ces modules pourra avoir la sous structure suivante :

.../

components

gards

pipes

models (or data)

directives

services

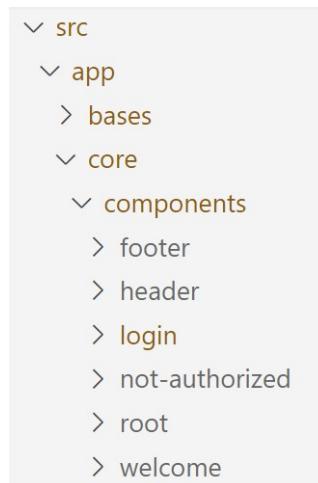
... (**utils**, **configs**, ...)

src/app/app.module peut devenir **src/app/core/core.module** with **CoreModule** in *src/main.ts*

src/app/app.routing.module peut devenir **src/app/core/core.routing.module**

src/app/app.component peut devenir **src/app/core/components/root/root.component.ts** with

core-root in *index.html*



Convention de noms pour les sélecteurs : **moduleName-ComponentName** (ex : *core-root*, *core-welcome*, *core-header*, ...)

dans *src/app/shared/shared.module.ts*

```

...
@NgModule({
  imports: [  CommonModule , FormsModule , HttpClientModule , ... ],
  exports: [  TogglePanelComponent , ... ],
  declarations: [  TogglePanelComponent , ... ],
...
  
```

dans *xyz/xyz-routing.modules.ts*

```

...
{ path: '' , redirectTo: 'xyz' , pathMatch: 'prefix' }
...
  
```

dans *core/core-routing.modules.ts*

```

...
{ path: 'ngr>Welcome' , component: WelcomeComponent },
{ path: '' , redirectTo: '/ngr>Welcome' , pathMatch: 'full' },
{ path: 'ngr>Login' , component: LoginComponent },
{ path: 'ngr>xxx' , loadChildren: () => import('..//xxx/xxx.module').then(m => m.XxxModule) },
{ path: 'ngr>yyy' , loadChildren: () => import('..//yyy/yyy.module').then(m => m.YyyModule) },
...
  
```

NB : il suffit de rectifier les chemins relatifs au niveau des *import { ... } from '...'* ; pour restituer une cohérence dans le cas où certains réertoires sont renommés ou déplacés lors d'un refactoring.

6.3. Standalone application (sans module)

NB: ceci a longtemps été impossible au sein des anciennes versions d'angular (de 2 à 14,15 ou 16). Ce n'est possible et vraiment au point que depuis la version 17.

Via `bootstrapApplication(MyMainComponent);` de `@angular/core`

on peut démarrer une application angular sans module à partir d'un simple composant en mode standalone.

src/main.ts en version standalone

```
import { bootstrapApplication } from '@angular/platform-browser';
import { appConfig } from './app/app.config';
import { AppComponent } from './app/app.component';

bootstrapApplication(AppComponent, appConfig)
  .catch((err) => console.error(err));
```

src/app/app.config.ts

```
import { ApplicationConfig } from '@angular/core';
import { provideRouter } from '@angular/router';

import { routes } from './app.routes';
import { provideClientHydration } from '@angular/platform-browser';

export const appConfig: ApplicationConfig = {
  providers: [provideRouter(routes), provideClientHydration()]
};
```

src/app/app.routes.ts

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

export const routes: Routes = [
  { path: 'welcome', component: WelcomeComponent },
  { path: '', redirectTo: '/welcome', pathMatch: 'full' },
];
```

Changement de comportement de angular-cli depuis v17

`ng new simpleApp (standalone par défaut depuis v17)`

`ng new --no-standalone appWithModules`

NB: au sein d'une application "standalone" , chaque composant est autonome (@Component avec { standalone : true , ...}) et doit lui même renseigner finement ses dépendances (ex : RouterLink , FormsModule , DecimalPipe , NgFor , ...)

6.4. Standalone components

En mode **standalone**, un composant n'appartient plus à un module ; il est complètement indépendant et défini lui même tous les importations dont il a besoin.

```
@Component({
  standalone: true, ...
})
export class MyAngularStandaloneComponent {}
```

NB : Le paramétrage standalone n'existe pas avant la version 14 (comme si il avait la valeur false par défaut). Au sein des versions 15 à 18 , il faut explicitement préciser standalone : true .

A partir de la version 19, le paramétrage standalone est maintenant implicitement avec la valeur true par défaut et on est donc obligé de spécifier *standalone : false* dans les cas où c'est utile (en mode module à l'ancienne).

Exemples d'importations classiques :

```
imports: [RouterOutlet , HeaderComponent, FooterComponent],  
imports: [RouterLink , DatePipe , AsyncPipe , DecimalPipe , JsonPipe , UppercasePipe],  
imports: [FormsModule , NgIf , NgFor],
```

6.5. Utilisation possible d'un "standalone component" dans un module

Un "Standalone component" peut être importé (comme d'autres modules) au sein d'un module de composants . Ceci permet (surtout à court/moyen terme) l'utilisation de composant "standalone" au sein de modules classiques (d'applications existantes).

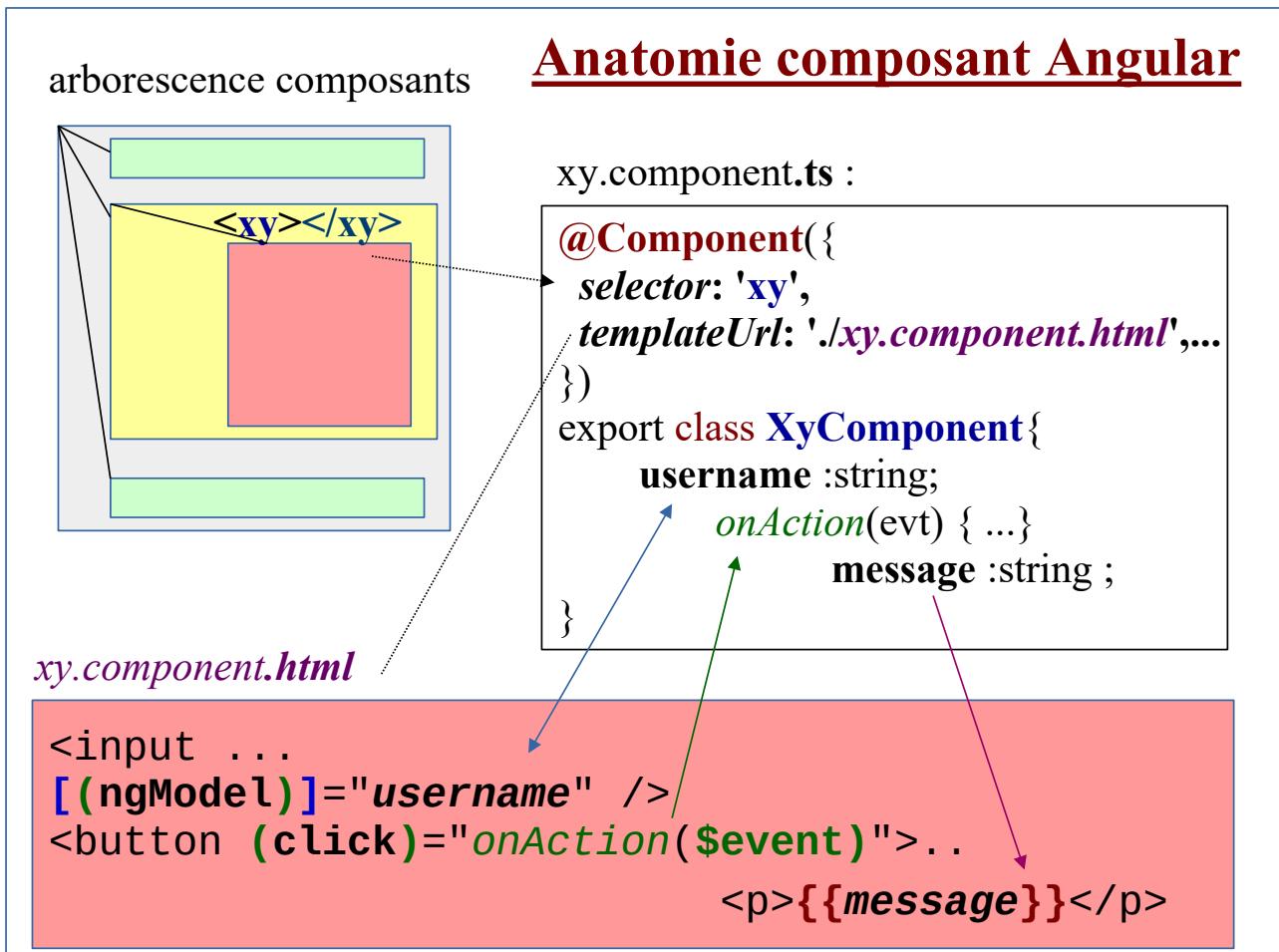
Exemple :

```
@NgModule({
  declarations: [AlbumComponent],
  exports: [AlbumComponent],
  imports: [PhotoGalleryComponent],
})
export class AlbumModule {}
```

Et inversement , un composant "standalone" peut éventuellement importer un module regroupant plusieurs éléments.

IV - Essentiel sur template, binding, event, pipe

1. Anatomie d'un composant angular



Un "**template**" HTML Angular comporte, en plus des syntaxes HTML standardisées, des paramétrages spécifiques au framework Angular :

- Expressions : `{{ . . . }}`
- "Bindings" de propriétés : `[(ngModel)]="..."`
- Déclenchement de méthodes événementielles : `(click)="onAction()"`

Le principal intérêt du framework Angular réside dans les **liaisons automatiques** établies entre les parties d'un modèle de vue HTML ("template") et les données d'un modèle orienté objet en mémoire dans le composant.

Ce "**mapping**" ou "**binding**" peut être unidirectionnel (via `{}{...}` ou `[. . .]="..."`) ou bidirectionnel (via `[(ngModel)]="..."`).

Le déclenchement de méthodes événementielles, via la syntaxe `(evtName) = "on... ($event)"`, constitue également un paramétrage du "mapping" Angular.

1.1. Exemple ultra simple

username:

message:

username:

message: **Hello superUser**

Fichier **basic.component.ts**

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

@Component({
  selector: 'app-basic',
  templateUrl: './basic.component.html',
  styleUrls: ['./basic.component.scss']
})
export class BasicComponent{

  username : string ="";
  message : string ="";

  onAction(){
    this.message ="Hello " + this.username;
  }

  constructor() { }
}
```

Fichier **basic.component.html**

```
username: <input [(ngModel)]="username" > <br>
<button (click)="onAction()">hello</button> <br>
message: <b>{{message}}</b>
```

Explications :

- L'expression `{ {message} }` (côté .html) permet d'**afficher automatiquement la valeur de l'attribut message** de l'instance de la classe du composant TypeScript
- Cet affichage sera automatiquement réactualisé par le framework Angular dès que la valeur de l'attribut `.message` changera dans la zone mémoire du composant
- La syntaxe `(click)="onAction()"` sur le bouton du template HTML permet de demander un appel automatique à la méthode `onAction()` du composant dès que l'utilisateur cliquera sur le bouton
- La syntaxe `[(ngModel)]="username"` sur la zone de saisie correspond à un **binding bidirectionnel** entre la zone `input` et l'attribut `username` du composant TypeScript. Par conséquence :
 - La zone `input` affichera toujours une valeur actualisée de l'attribut `username` en mémoire
 - L'attribut `username` sera automatiquement modifié en mémoire dès que l'utilisateur saisira une nouvelle valeur dans la zone de saisie

NB : l'utilisation de `[(ngModel)]` nécessite l'importation de **FormsModule** dans `app.module.ts` ou bien dans `@Component({ standalone:true , imports : [] })` :

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { AppComponent }  from './app.component';

@NgModule({
  imports:  [ BrowserModule , FormsModule ],
  declarations: [ AppComponent ],
  bootstrap:  [ AppComponent ]
})
export class AppModule { }
```

Attention : Sans ajout de **FormsModule** dans la partie `imports: []` de `@NgModule` de `app.module.ts` ou bien de `@Component({ standalone:true , ... })`, la syntaxe `[(ngModel)]="..."` ne fonctionnera pas !!!

2. Templates bindings (property , event)

2.1. Binding Angular

Page "angular" constituée d'une arborescence de composants

Binding Angular

Anatomie de chaque composant :

```
@Component({
  selector: 'xy',
  templateUrl: 'app/xy.component.html',
})
export class XyComponent{
  data : DataTypeZz ;
  ...
  onNewXy = function(evt) { ... }
}
```

xy.component.html

```
<button (click)="onNewXy($event)">
<p>{{data.label}}</p>
<input [(ngModel)]="data.value">
```

(Modèle orienté objet)
 data.id
 .label
 .value

Le "mapping" ou "binding" Angular peut être unidirectionnel (via `{{...}}` ou `[...]="..."`) ou bidirectionnel (via `[(ngModel)]="..."`).

Comme le montre le schéma ci-dessus, ce "binding" peut éventuellement être appliqué sur des sous-objets du composant (exemple : `{{data.label}}`).

2.2. Exemples d'interpolations / expressions `{}{}`

`<p>My current hero is {{currentHero.firstName}}</p>`

`<p>The sum of 1 + 1 is {{a + b}}</p>`

`<p>The sum of 1 + 1 is not {{a + b + computeXy()}}</p>`

`<!-- computeXy() appelé sur composant courant associé au template -->`

2.3. Syntaxe des liaisons entre vue/template et modèle objet

Syntaxes (template HTML)	Effets/comportements
<p>Hello {{ponyName}}</p>	Affiche Hello poney1 si ponyName vaut poney1
<p>Employer : {{employer?.companyName}}</p>	Pas exception (affichage ignoré) si l'objet (facultatif/optionnel) est un "undefined"
<p>Card No.: {{cardNumber myCreditCardNumberFormatter}}</p>	Pipe(s) pour préciser un ou plusieurs(s) traitement(s) avant affichage
<button [disabled]="mode=='existing'">	Affecte la valeur de l'expression (ici booléenne) à la propriété disabled (one-way)
<div title="Hello {{ponyName}}">	équivalent à: <div [title]="'Hello' + ponyName">
<div [style.width.px]="mySize">	Affecte la valeur de l'expression mySize à une partie de style css (ici width.px)
<button (click)= "readRainbow(\$event)">	Appelle la méthode readRainbow () en passant l'objet \$event en paramètre lorsque l'événement click est déclenché sur le composant courant (ou un de ses sous composants)
<input [(ngModel)]="userName">	Liaison dans les 2 sens (lecture/écriture). <u>NB</u> : ngModel est ici une directive d'attribut prédéfinie dans le module FormsModule.
<my-cmp [(title)]="name"> possible mais très rare	"two-way data binding" sur (sous-)composant. équivalent à: <my-cmp [title]="name" (titleChange)="name=\$event">

2.4. Principales directives prédéfinies (angular2/common)

<section *ngIf="showSection">	Rendu conditionnel (si expression à true) . Très pratique pour éviter exception {{obj.prop}} lorsque obj est encore à "undefined" (pas encore chargé)
<li *ngFor="let item of list">	Elément répété en boucle (forEach)
<div [ngClass]="{active: isActive, disabled: isDisabled}">	Associe (ou pas) certaines classes de styles CSS selon les expressions booléennes .

2.5. Précision (vocabulaire) "attribut HTML , propriété DOM"

<input type="text" value="Bob"> est une syntaxe HTML au sein de laquelle l'attribut **value** correspond à **la valeur initiale de la zone de saisie**.

Lorsque cette balise HTML sera interprétée , elle sera transformée en sous arbre DOM puis affichée/rendue par le navigateur internet.

Lorsque l'utilisateur saisira une nouvelle valeur (ex : "toto") :

- la valeur de l'attribut HTML *value* sera inchangée (toujours même valeur initiale/par défaut) .
- La valeur de la propriété "value" attachée à l'élément de l'arbre DOM aura la nouvelle valeur "toto" .

Autrement dit, un attribut HTML ne change pas de valeur, tandis qu'une propriété de l'arbre DOM peut changer de valeur et pourra être mise en correspondance avec une partie d'un composant "angular2" .

NB : Au sein de l'exemple ci-dessous , la propriété "**disabled**" de l'élément de l'arbre DOM est valuée à partir de la propriété "**isUnchanged**" du composant courant .

```
<button [disabled]="isUnchanged">Save</button>
```

et la propriété "**disabled**" de l'élément DOM a (par coïncidence non systématique) le même nom que l'attribut "**disabled**" de la balise HTML button .

Exception qui confirme la règle :

Dans le cas, très rare , où une propriété d'un composant "angular" doit être associé à la valeur d'un attribut d'une balise HTML , la syntaxe prévue est **[attr.nomAttributHtml]="expression"** .

Exemple: <td **[attr.colspan]**="1 + 1">One-Two</td>

2.6. Style binding

```
<button [style.color] = "isSpecial ? 'red' : 'green'">Red</button>
<button [style.backgroundColor] = "canSave ? 'cyan' : 'grey'">Save</button>
```

Au sein des exemples ci-dessus, un seul style css n'était dynamiquement contrôlé à la fois.

De façon à contrôler dynamiquement d'un seul coup les valeurs de plusieurs styles css on pourra préférer l'utilisation de la directive **ngStyle** :

```
setStyles() {
  return {
    // CSS property names
    'font-style': this.canSave ? 'italic' : 'normal', // italic
    'font-weight': !this.isUnchanged ? 'bold' : 'normal', // normal
    'font-size': this.isSpecial ? 'x-large' : 'smaller', // larger
  }
}

<div [ngStyle]="setStyles()">
  This div is italic, normal weight, and x-large
</div>
```

2.7. CSS Class binding

La classe CSS "special" (.special { ... }) est dans l'exemple ci dessous associée à l'élément <div> de l'arbre DOM si et seulement si l'expression "isSpecial" est à true au sein du composant .

```
<div [class.special] = "isSpecial">The class binding is special</div>
```

Cette syntaxe est appropriée et conseillée pour contrôler l'application conditionnée d'une seule classe de style CSS.

De façon à contrôler dynamiquement l'application de plusieurs classe CSS , on préférera la directive **ngClass** spécialement prévue à cet effet :

```
setClasses() {
  return {
    saveable: this.canSave, // true
    modified: !this.isUnchanged, // false
    special: this.isSpecial, // true
  }
}
```

pour un paramétrage selon la syntaxe suivante :

```
<div [ngClass] = "setClasses()">This div is saveable and special</div>
```

2.8. Bindings particuliers

Cas particulier [ngValue] pour une liste de sélection d'objet :

```
...
<select [(ngModel)]="selectedPublication" size="8" style="width:100%">
  (change)="onChangeSelectedPublication($event)">
    <option *ngFor="let publication of tabPublications"
      [ngValue]="publication" >{ {essentielPublicationString(publication)} }</option>
</select>
...
```

Pour une cache à cocher (input de type=checkbox), le binding unidirectionnel (ts ---> template) peut s'effectuer via [checked]="nomProprieteBooleene" .

Via [(ngModel)]="propXy" la valeur de la propriété propXy est automatiquement mise à jour suite à une saisie ou sélection de nouvelle valeur . Il est cependant possible en complément de demander à appeler juste après une méthode particulière pour déclencher un éventuel traitement utile via la syntaxe (ngModelChange)="onMajZzAfterXyChange()"

2.9. Exemple 1 (calculatrice)

calculatrice.component.ts

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

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

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

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

```
//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() { }
ngOnInit(): void {}
}
```

Quelques explications :

- Au moment où la méthode `onCalculer()` sera appelée, les valeurs saisies pour les zones de saisie `a` et `b` seront normalement déjà répercutées en mémoire dans `this.a` et `this.b` via les syntaxes `[(ngModel)]="a"` et `[(ngModel)]="b"` côté `.html`
- Lorsque le code de la méthode `onCalculer()` modifiera la valeur de `this.res`, cette valeur sera automatiquement réaffichée / actualisée côté `.html` via la syntaxe `{{res}}`
- Le code de la méthode `onMouseMove()` est un peu plus technique. Cette méthode montre comment accéder aux informations de l'événement `click` géré par un navigateur internet.

`calculatrice.component.html` :

```
<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>

<hr>

<div class="c2" (mousemove)="onMouseMove($event)"
      (mouseout)="onMouseLeave($event)">
Zone à survoler à la souris .<br>
x={{x}} , y={{y}}
</div>
</div>
```

Quelques explications :

- Au niveau de `(click)="onCalculer('+')`, on passe ici un paramètre simple

(nom de l'opération mathématique) à la fonction événementielle qui sera appelée

- Au niveau de `(mousemove)="onMouseMove($event)"`, on passe ici un paramètre technique (objet événement `click` géré par le navigateur, vu ici comme la syntaxe spéciale `$event` du framework Angular)
- `[style.font-weight]="res>0?'bold':'normal'"` permet d'affecter, au style CSS `font-weight`, la valeur `bold` si la valeur de `this.res` est positive et `normal` dans les autres cas.
- `[class.negatif]="res<0"` permet d'activer la classe `.negatif` (à définir dans un fichier `.css`) dès que la valeur de `this.res` sera négative

calculatrice.component.scss :

```
.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;}
```

calculatrice angular (v1)

a :	<input type="text" value="2"/>
b :	<input type="text" value="5"/>
operation :	<input type="button" value="+"/> <input type="button" value="-"/> <input type="button" value="*"/>
resultat:	7

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

Zone à survoler à la souris .

x=204 , y=34

2.10. Autre exemple (plus sophistiqué)

catégorie à selectionner:

- cd
- dvd
- other

catégorie à selectionner:

- **cd**
- dvd
- other

produits de la catégorie cd :

ref	label	prix
p1	CD1	5.6
p2	CD2	9.6

catégorie à selectionner:

- cd
- **dvd**
- other

produits de la catégorie dvd :

ref	label	prix
p3	DVD a	15.6
p4	DVD b	19.6

zz.component.ts

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

class Produit{
  constructor(public ref:string="?",
    public label : string ="?",
    public prix : number =0){}
}

@Component({
  selector: 'app-zz',
  templateUrl: './zz.component.html',
  styleUrls: ['./zz.component.scss']
})
export class ZzComponent implements OnInit {

  listeCategories = [ "cd" , "dvd" , "other"];
  categorie : string | undefined;//à choisir
  mapCategorieProduits= new Map<string,Produit[]>();
  listeProduits : Produit[] | undefined ; //selon categorie choisie

  onSelectCategorie(categorieChoisie:string){
    this.categorie=categorieChoisie;
    console.log("categorieChoisie="+this.categorie)
    this.listeProduits=this.mapCategorieProduits.get(this.categorie) ;
    console.log("listeProduits="+JSON.stringify(this.listeProduits))
  }
}
```

```

constructor() {
    this.mapCategorieProduits.set("cd",
        [ new Produit('p1','CD1',5.6) , new Produit('p2','CD2',9.6)]
    );
    this.mapCategorieProduits.set("dvd",
        [ new Produit('p3','DVD a',15.6) , new Produit('p4','DVD b',19.6)]
    );
    this.mapCategorieProduits.set("other",
        [ new Produit('p5','smartPhone',255.6) , new Produit('p6','TV',567.6)]
    );
}

ngOnInit(): void {
}
}

```

zz.component.html

```

<div class="myWrapFlexbox">
    <div class="myColItem">
        <h3>catégorie à selectionner:</h3>
        <ul>
            <li *ngFor="let c of listeCategories"
                (click)="onSelectCategorie(c)"
                [class.selected]="c==categorie">{{c}}</li>
        </ul> <br/>
    </div>
    <div class="myColItem" *ngIf="categorie">
        <h3>produits de la catégorie {{categorie}} :</h3>
        <table border="1" >
            <tr> <th>ref</th> <th>label</th> <th>prix</th> </tr>
            <tr *ngFor="let p of listeProduits" >
                <td>{{p.ref}}</td> <td>{{p.label}}</td> <td>{{p.prix}}</td>
            </tr>
        </table>
    </div>
</div>

```

NB : en mode "standalone", l'utilisation de *ngIf et *ngFor nécessite
imports : [... , NgIf , NgFor , ...] au sein de **@Component**

zz.component.css

```

.selected { color : blue; font-weight: bold;}
.myWrapFlexbox { padding : 1em; display: flex; flex-flow: row wrap; }
.myColItem { flex: 1 1 auto; text-align: left; padding: 2px; margin: 8px; }

```

2.11. TP tva

Si ce n'est pas déjà fait:

- créer un nouveau composant **TvaComponent** via **ng g component tva**
- accrocher ce composant à son composant parent (**<app-tva></app-tva>**)
- vérifier la présence de **FormsModule** dans la partie **imports:[]** de **app.module.ts**

Coder ensuite (en 1 ou 2 versions successives) de composant **TvaComponent** de manière à atteindre l'objectif suivant :

calcul de tva

ht: **200**

tauxTva: **20%**

tva: **40**

ttc: **240**

Suggestions :

- On pourra coté .ts préparer un tableau de taux de tva possibles avec par exemple les valeurs = [**5** , **10** , **20**];
- Une seule méthode **onCalculTvaTtc()** devrait suffire à recalculer tva et ttc
- On pourra éventuellement coder une pré version au sein de laquelle la méthode **onCalculTvaTtc()** est déclenchée par un bouton poussoir temporaire
- Dans la version finale sans bouton poussoir, on pourra traiter l'événement (**input**) sur la zone de saisie et l'événement (**change**) sur la liste déroulante .
- Via une directive ***ngIf** on affichera les valeurs calculées tva et ttc que si tva est >0 .

2.12. Optimisations et approfondissements autour de *ngFor

Récupérer également la valeur de l'index (0,1,...n-1) au sein d'une boucle ***ngFor** :

```
<tr *ngFor="let item of tab ; let i=index">
  {{item}}... {{i}}
</tr>
```

Optimiser l'identification des éléments d'un tableau de l'arbre DOM (améliore performance):

```
identifyProduct(index:unknown, item:Product){
  return item.code;
}
```

```
<tr *ngFor="let p of tabProducts ; let i=index; trackBy: identifyProduct">...</tr>
```

3. Formatage des valeurs à afficher avec des "pipes"

3.1. Pipes prédefinis

Exemples:

```
{ { montant | number:'1.0-2' } } <!-- arrondi à 2 chiffres après virgule
                                         number:'minIntegerDigit.minFractionDigit-maxFractionDigit' -->
```

```
<div> {{ birthday | date:"MM/dd/yy" }} </div>
```

```
<!-- pipe with configuration argument => "February 25, 1970" -->
<div>Birthdate: {{currentHero?.birthdate | date:'longDate'}}</div>
```

```
<div>{{ title | uppercase }}</div> <div>{{ title | lowercase }}</div>
{{taux | percent }} <!-- affiche 5% si taux vaut 0.05 -->
```

```
 {{ birthday | date | uppercase }}
```

```
 {{ birthday | date:'fullDate' | uppercase }}
```

```
 {{ tva | currency:'EUR': 'symbol': '1.0-2' }} €240.27
```

il existe encore d'autres pipes prédefinis:

"json pipe" pour (temporairement) déboguer un "binding" :

```
<div>{{ currentHero | json }}</div>
<!-- Output:
  {
    "firstName": "Hercules",
    "lastName": "Son of Zeus",
    "birthdate": "1970-02-25T08:00:00.000Z",
    "url": "http://www.imdb.com/title/tt0065832/",
    "rate": 325,
    "id": 1
  }
-->
```

NB1: Dans une application angular >=2 , les tris et filtrages ne sont généralement pas effectués par des pipes "angular" (pas de | sort ni de |filter) mais par des opérateurs de RxJs (à placer coté .ts dans .pipe() avant .subscribe()) --> voir annexe "RxJs" .

NB2: Si un élément à afficher est de type "Observable" (de RxJs) , on peut alors utiliser le pipe "async" : {{ myObservableCounter\$ | async }}

Dépendances vers "pipe" en mode "standalone" :

Si l'application angular est en mode "standalone" (sans module), il faut alors placer des dépendances au sein de @Component :

```
import { AsyncPipe, UpperCasePipe, ... } from '@angular/common';

@Component({
...
imports: [FormsModule, AsyncPipe, UpperCasePipe, JsonPipe, DecimalPipe, DatePipe],
...
})
```

Pipe selon "locale" (fr , en-US ,)

```
{{ montant | number:'1.0-2' :'fr' }} <!-- affiche 20 000,55 -->
{{ montant | number:'1.0-2' :'en-US' }} <!-- affiche 20,000.55 -->
```

NB : l'argument :'**fr**' nécessite l'enregistrement de localFr dans **app.module.ts** :

```
import localeFr from '@angular/common/locales/fr';
import { registerLocaleData } from '@angular/common';
registerLocaleData(localeFr);

@NgModule({
...})
```

On peut éventuellement définir ':fr-FR' comme **default "locale"** dans **app.module.ts** :

```
import { LOCALE_ID, NgModule } from '@angular/core';
import localeFr from '@angular/common/locales/fr';
import { registerLocaleData } from '@angular/common';
registerLocaleData(localeFr);

@NgModule({
...
providers: [
  { provide: LOCALE_ID, useValue: 'fr-FR' },
],
...})
```

avec

```
{{ montant | number:'1.0-2' }} <!-- affichant alors 20 000,55 -->
```

3.2. Custom pipe :

```
cd src/app/common/pipe
```

ng g pipe exponential (avec si nécessaire option `--skip-import` en cas de problème)

src/app/common/pipe/exponential.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';
/* Raise the value exponentially
 * Example:
 * {{ x | exponential:3}} , pour x= 5 , affiche 5 puissance 3 = 5*5*5=125 .
 */
@Pipe({ name: 'exponential'})
export class ExponentialPipe implements PipeTransform {

  transform(value: unknown, ...args: unknown[]): unknown {
    let val : number = <number> value;
    let p : number = <number> args[0] || 1;
    return Math.pow(val,p);
  }
}
```

Déclaration du pipe personnalisé dans app.module.ts :

```
import { ExponentialPipe } from './common/pipe/exponential.pipe';
.....
@NgModule({
  declarations: [ AppComponent, .... , ExponentialPipe ] , ....
})
```

Template de Composant utilisant le pipe personnalisé :

....html

```
<p> Super power boost: {{2 | exponential: 10}} </p>
```

The screenshot shows a component template with the following code:

```
<p> Power Booster </p>
<----> Super power boost: 1024
```

The output of the pipe is visible in the browser preview, showing the value 1024.

V - Code flow et signaux

1. Code flow (version >= 17)

Depuis angular 17, il est possible d'utiliser de nouvelles syntaxes (appelées "code flow") au sein des templates html.

Principaux intérêts:

- **pas besoin de imports:[NgIf , NgFor]**
- **performances améliorées**

Ainsi, au lieu d'utiliser `*ngIf="..."` ou bien `*ngFor="let v of values"` placées sur des balises de type "`<p>`" ou "`<div>`" ou autres on peut directement utiliser `@if(...)` `@else{...}` dans le texte du template (au dehors des balises html) .

1.1. @if(. . .){ . . . }@else{...}

```
<input type="checkbox" [(ngModel)]="withDetails"> withDetails
<hr/>
@if(withDetails) {
  <div>
    <p>details ...</p>
  </div>
} @else {
  <p>no details</p>
}
```

1.2. @for(. . .){ . . . }@empty{...}

```
<table border="1">
<tr><th>v</th><th>v*v</th></tr>
@for(v of values; track v){
  <tr><td>{v}</td><td>{v*v}</td></tr>
}
@empty {
  <tr><td>?</td><td>?</td></tr>
}
</table>
```

NB :

- La partie `@empty { }` est facultative et n'est utilisée que si le tableau est vide .
- La partie `; track v` (ou bien `; track obj.id`) est obligatoire et permet de préciser l'identifiant

de chaque entrée de la collection . C'est un point clef pour obtenir de bonnes performances.

1.3. `@switch(...){ @case(...){...}}`

```
<select [(ngModel)]="detailLevel">
  <option>none</option>
  <option>low</option>
  <option>high</option>
</select> <br/>
@switch (detailLevel) {
  @case("none"){ <div>detailLevel=none [no details]</div>}
  @case("low"){ <div>detailLevel=low , few details</div>}
  @case("high"){ <div>detailLevel=high , much details , ...</div>}
}
```

NB: il existe `@default { }`

2. Signaux (version >= 16+)

NB: Les signaux sont apparus avec la version 16 d'angular . Ils ont été en phase "developper-preview" au sein des versions 17,18 . L'essentiel est enfin stabilisé depuis angular 19.

Les signaux forment un **mécanisme d'actualisation réactive synchrone**. Il sont vus comme des **fonctions d'accès à des données** . Les signaux peuvent être composés de **sous-signaux** (vus comme des sous fonctions).

Principales sortes de signaux :

Types de signaux	Fonctions de construction	Caractéristiques
WritableSignal	signal()	Sous objet de données évolutives avec signalement automatique des mises à jour
Signal	computed() , effect() en void	Signal de haut niveau basé (par calcul ou autre) sur des signaux de bas niveau
En remplacement de @Input() , @Output() , ...	input() , output() , model()	Signaux en remplacement de @Input() , @Output() de manière à obtenir de meilleures performances

2.1. Signaux ordinaires (*WritableSignal* , *signal()*)

Exemple :

```
import { Component, Signal, computed, effect, signal } from '@angular/core';
```

```
@Component({
  selector: 'app-with-signal',
  standalone: true,
  imports: [],
  templateUrl: './with-signal.component.html',
  styleUrls: ['./with-signal.component.scss'
})
export class WithSignalComponent {
  //sCount=signal<number>(0);
  sCount = signal(0); //new WritableSignal With initial value to 0
  //other possible signal types : String, boolean , object , array , ...

  public onIncrement(){
    //NB: signalName as function call to get value ,
    // .set() to update/change value with synchronization
    this.sCount.set(this.sCount() + 1);
  }

  public onDecrement(){
    //.set(newValue or .update(currentValue->newValue)
    //this.sCount.set(this.sCount() - 1);
    this.sCount.update ( count => count-1);
  }

  public onSCountChange(event : Event){
    const input = event.target as HTMLInputElement;
    this.sCount.set(Number(input.value));
  }
}
```

```
{
}
```

```
<span [style.color]="sCountColor()">sCount={{sCount0}}</span>
  <button (click)="onIncrement()">++</button>
  <button (click)="onDecrement()">--</button> <br/>
<hr/>
sCount = <input type="number" [value]="sCount()" (change)="onSCountChange($event)" />
```

sCount=15 ++ refresh --

sCount = 15

- **avec signaux --> meilleures performances , meilleure détection** automatique des changements et réactualisation automatique des affichages .

2.2. Signal avec ngModel

```
@Component({
  imports: [FormsModule,UpperCasePipe , ...],...
})
export class WithSignalComponent {
  name1="abc";
  name2AsSignal=signal("abc");
  name3AsSignal=signal("abc");
}
```

```
name1:<input [(ngModel)]="name1" /> {{name1 | uppercase}} (without signal)<br/>
name3AsSignal:<input [ngModel]="name3AsSignal()" (ngModelChange)="name3AsSignal.set($event)" />
{{name3AsSignal() | uppercase}} <br/>
<!-- more simple equivalent with some recent angular version : -->
name2AsSignal:<input [(ngModel)]="name2AsSignal" /> {{name2AsSignal() | uppercase}} <br/>
```

name1: abcd ABCD (without signal)
 name3AsSignal: abcd ABCD
 name2AsSignal: abcd ABCD

→ la tendance actuelle de l'évolution d'angular consiste à petit à petit s'éloigner de l'ancien système de détection des changements (avec zoneJs) et d'utiliser le mécanisme des signaux à la place.

2.3. Signaux calculés (computed())

Exemple :

```
import { Component, Signal, computed, effect, signal } from '@angular/core';

...
export class WithSignalComponent {
  sCount = signal(0);

  //NB: la fonction computed() (de @angular/core) permet de définir un nouveau signal calculé
  //à partir d'autres signaux → réactualisation automatique .

  sSquare /* :Signal<Number> */ = computed(() => this.sCount() * this.sCount());
  sCountColor /* :Signal<String> */ = computed(() => this.sCount()>=0?'green':'red');
}
```

```
...
sSquare={{sSquare}} <br/>
```

sCount=8 ++ refresh --

sCount = 8

sSquare=64

2.4. Effets (à utiliser qu'avec parcimonie)

Exemple :

```
import { Component, Signal, computed, effect, signal } from '@angular/core';
...
export class WithSignalComponent {
  message="";
  sCount = signal(0); //new WritableSignal With initial value to 0

  //NB: la fonction effect() (de @angular/core) permet d'enregistrer une callback
  //qui sera automatiquement appelée dès qu'un signal changera de valeur
  logsCountEffect = effect(()=>{ this.message ="sCount="+this.sCount();
    console.log(this.message);});
}
```

```
<span [style.color]="sCountColor()">sCount={{sCount}}</span> &nbsp;
<button (click)="onIncrement()">++</button> &nbsp; <button (click)="onDecrement()">--</button> <br/>
<hr/> message(as an effect)={{message}}
```

message(as an effect)=[5] sCount=4

2.5. Ré-évaluation des signaux si changements

Pour des raisons d'optimisation, les effets et les rafraîchissements/réactualisations d'affichage ne sont effectués que lorsqu'un signal change réellement de valeur.

Autrement dit, appeler `signalXy.set(meme_valeur)` ne déclenche rien !

Conséquences :

- les signaux fonctionnent parfaitement bien avec des objets "immutables" (string, number, ...)
- besoin de re-créer quelquefois des instances pour déclencher des réactualisations dans certains cas pointus .

```
// Refreshable<T> wrapper for handling immutability of signal value
class Refreshable<T>{
  public timestamp: Date;
  constructor(public value:T){
    this.timestamp=new Date(); //with time
  }
  time(){ return this.timestamp.getTime(); }
  refresh(){ return new Refreshable<T>(this.value); }
}
```

Exemple :

```
export class WithSignalComponent {
  message=""; //sera actualisé et affiché via un "effect"
  messageCount=0; //nombre de fois où l'effet sera déclenché
  refreshMessage=""; //sera actualisé et affiché via un "effect"
  refreshMessageCount=0; //nombre de fois où l'effet sera déclenché
  sCount = signal(0);
  sRefreshableCount = signal(new Refreshable<number>(0));
  ...
  logsCountEffect = effect(()=>{ this.message ="["+ (++this.messageCount)
    + "] sCount=" +this.sCount(); console.log(this.message);});

  logsRefreshableCountEffect = effect(()=>{
    this.refreshMessage ="["+ (++this.refreshMessageCount) + "] sRefreshableCount="
      +this.sRefreshableCount().value + " with .timestamp=" + this.sRefreshableCount().time();
    console.log(this.refreshMessage);
  });
  ...
  public onRefresh(){
    //signal.update(val) is only useful if val change
    //refreshable wrapper instance is changed when .refesh() is call
    //with a new timestamp) but inner .value may still remain same value
    this.sRefreshableCount.set(this.sRefreshableCount().refresh());
      //change .time value (new instance of wrapper)

    this.sCount.set(this.sCount());
    //if same value (no change) --> no effect call , no update of display !!!
  }
}
```

sCount=5

sCount =

sSquare=25

message(as an effect)=[6] sCount=5

refreshMessage(as an effect)=[11] sRefreshableCount=5 with .timestamp=1745914056552

2.6. input(), output(), model()

en tant que signaux pour interactions avec sous-composants .

sub.component.ts

```
import { Component, input, model, output } from '@angular/core';
import { FormsModule } from '@angular/forms';

@Component({
  selector: 'app-sub',
  imports: [FormsModule],
  standalone: true,
  templateUrl: './sub.component.html',
  styleUrls: ['./sub.component.scss'
})
export class SubComponent {
  public ic = input(0); //input counter as input signal
  public oc = output<number>(0); //output counter as output signal
  public mc = model(0); //model (input+output) counter as model signal

  public ocVal=0;

  onOcIncr(){
    this.ocVal++;
    this.oc.emit(this.ocVal);
  }
}
```

sub.component.html

```
<div class="sc">
<span>sub component</span> <br/>
<span>ic (input counter signal): {{ic}}</span>&nbsp;&nbsp;
<button (click)="onOcIncr()">increment oc (output counter signal) : {{ocVal}}</button> <br/>
mc = <input [(ngModel)]="mc" /> (model counter signal)
</div>
```

parent...html

```
...
<app-sub [(mc)]="pMcS" [ic]="sCount()" (oc)="onOc($event)" ></app-sub>
<span>pOcVal={{pOcVal}}</span> <br/>
<span>pMcS:<input [(ngModel)]="pMcS" /> {{pMcS}} </span>
```

parent...ts

```
pMcS= signal(100); //parentMcSignal value
pOcVal=0; //parent oc value
...
public onOc( ocVal: number){
  this.pOcVal=ocVal;
}
```

```

sub component
ic (input counter signal): 5    increment oc (output counter signal) : 11
mc = 10555                      (model counter signal)
pOcVal=11
pMcS: 10555                     10555

```

NB (pour cas pointus/complexes):

si besoin d'ajuster des types de données en typescript , il est possible de s'appuyer sur une syntaxe de ce genre (à adapter au cas pas cas) :

```

//liste des objets à afficher :
tabObjectsRef :InputSignal<object[] | null>
= input(null,{transform: (tabObjects)=> <any> tabObjects });

//objet sélectionné (null au début):
public selectedObjectRef :ModelSignal<any> = model(null);

```

2.7. Signaux avec services et RxJs

NB:

- Des signaux partagés peuvent être placés dans des services

Signal (angular 16+) vs BehaviorSubject(RxJs)

- Les signaux (synchrone) sont généralement mieux (plus simples, plus performants) dans la majorité des cas.
- Les "Subjects" (de RxJs) sont plus complexes/perfectionnés (asynchrones, multiples abonnés, ...) et sont à réserver/utiliser pour les cas pointus.

```

import { interval} from 'rxjs';
import { toSignal} from '@angular/core/rxjs-interop';
...
myIntervalObs = interval(1000); //observable emmitting 1,2,3,.. every 1000ms
myIntervalSignal = toSignal(this.myIntervalObs, { initialValue: 0 } )

```

```

...
from observable : {{ myIntervalObs | async}} <br/>
from signal : {{ myIntervalSignal()}}

```

from observable : 10
from signal : 10

VI - Switch et routing essentiel (navigation)

1. Switch élémentaire de sous composants

Attention: il ne s'agit ici que d'une présentation préliminaire de quelques possibilités d'angular de manière à montrer (par comparaison) les valeurs ajoutées du véritable "routing" angular .

1.1. rare switch (local) de sous-templates (sous-composants):

```
<div [ngSwitch]="variableXy">
<composant1 *ngSwitchCase=""case1Exp"">...</composant1>
<composant2 *ngSwitchCase=""case2LiteralString"">...</composant2>
<div_ou_composant3 *ngSwitchDefault>...</div_ou_composant3>
</div>
```

NB : Attention à bien placer des "simples quotes" dans les "doubles quotes" .

Ceci permet simplement de switcher de "détails à afficher" (et donc souvent de sous-composant). On reste dans un même composant parent principal . Pas de changement d'URL .

Attention : Le switch/basculement de composant s'effectue par remplacement d'instance (et éventuelle perte de l'état de l'ancien composant remplacé) .

--> pas de show/hide ni display none/block mais recharge complet d'un nouveau composant !!!

1.2. éventuel pseudo switch visuel (en apparence)

On peut éventuellement se bricoler facilement un "*pseudo switch visuel*" en jouant sur le style *display* de plusieurs <div ...> dont une seule est active à l'instant t :

```
<div class="panel panel-info" >
  <div class="panel-body" [style.display]=""subpart=='c1'? 'block': 'none'">
    <composant1></composant1>
  </div>
  <div class="panel-body" [style.display]=""subpart=='c2'? 'block': 'none'">
    <composant2></composant2>
  </div>
...
</div>
```

--> dans ce cas l'instance développée ou pas (visible ou pas) d'un sous composant est conservée (ainsi que l'état de ses variables d'instance) .

2. Bases élémentaires du routing angular

2.1. Navigation avec changement d'url et configuration

De façon à naviguer efficacement (tout en pouvant enregistrer des "bookmarks" sur une des parties de l'application) , il faut utiliser le "routeur" d'angular 4+ qui sert à basculer de composants (ou sous composant) tout en gérant bien les URLs/Paths relatifs .

Le service de routage est prise en charge par le module "*RouterModule*" .

Lors de la création d'une nouvelle application angular (via **ng new my-app**) , certaines questions sont posées telles qu'entre-autres :

"voulez vous activer le routing angular ?"

Si l'on répond "oui" à cette question le fichier **app-routing.module.ts** est alors créé au sein du répertoire **src/app** et ce module annexe est également relié au module principal **app.module.ts** .

Si l'on a répondu "non" , il faut alors ajouter soit même le fichier **app-routing.module.ts** et le relier au fichier **app.module.ts** selon l'exemple suivant :

app-routing.module.ts

```
import { NgModule }           from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { WelcomeComponent } from './welcome.component'; ...
const routes: Routes = [
  { path: 'welcome', component: WelcomeComponent },
  { path: '', redirectTo: '/welcome', pathMatch: 'full'},
  { path: 'login', component: LoginComponent }
];
@NgModule({
  imports: [ RouterModule.forRoot(routes) ],
  exports: [ RouterModule ]
})
export class AppRoutingModule {}
```

Liaison au sein du module principal **app.module.ts** :

```
import { AppRoutingModule } from './app-routing.module';
....
@NgModule({
  imports:  [ BrowserModule , FormsModule , HttpClientModule, AppRoutingModule , ... ],
  ...
})
export class AppModule { }
```

2.2. router-outlet

router-outlet = partie d'un template qui changera automatiquement de contenu selon la route courante .

Exemple: app.component.html

```
<div class="container-fluid">
<app-header [titre]="title"></app-header>

<!-- la balise spéciale router-outlet de angular
    sera automatiquement remplacée par le composant associé
    à la route courante/sélectionnée -->
<router-outlet></router-outlet>

<app-footer></app-footer>
</div>
```

Dans la plupart des cas simple/ordinaire comme celui-ci , un seul router-outlet (sans nom) suffit.

Dans certains cas sophistiqués et bien structurés , il est possible qu'un des sous-composants comporte en lui un autre <router-outlet> (à un niveau imbriqué) pour ainsi pouvoir switcher de sous-sous-composant .

Dans des cas très complexes, il est possible de configurer des "router-outlet" annexes (avec des noms) et de leurs associer des contenus variables selon un suffixe particulier placé en fin d'URL.

2.3. Routage simple (sans paramètres)

Au sein de **app-routing.module.js**

```
const routes: Routes = [
  { path: 'welcome', component: WelcomeComponent },
  { path: '', redirectTo: '/welcome', pathMatch: 'full'},
  { path: 'login', component: LoginComponent },
  { path: '**', redirectTo: '/welcome', pathMatch: 'full' }
];
```

suffit pour se retrouver automatiquement redirigé de index.html vers l'URL .../welcome (d'après la seconde règle avec redirectTo:) .

La première route associe le composant "WelcomeComponent" à la fin d'url "welcome" et dans ce cas la balise <router-outlet></router-outlet> sera remplacé par le contenu (template) du composant "WelcomeComponent" .

Finalement un click sur un lien hypertexte dont l'url relative est "login" (ou bien une navigation équivalente) déclenchera automatiquement un basculement de sous composant (le template de "LoginComponent" sera affiché au niveau de <router-outlet></router-outlet>).

La route spéciale (de path valant '**') permet de naviguer par défaut vers /welcome en cas de route mal exprimée (exemple: fin d'url inconnue suite à un changement de version de l'application) .

2.4. Déclenchement d'une navigation angular par lien hypertexte

Par exemple dans *header.component.html*

```
<a routerLink="/welcome">welcome</a> &nbsp;  
<a routerLink="/login">login</a> &nbsp;
```

Avec ici "*welcome*" et "*login*" qui correspondent aux valeurs des *path* de *app-routing.module.js*

VII - Contrôles de formulaires

1. Contrôle des formulaires (template-driven)

1.1. les différentes approches (template-driven , model-driven,...)

Approches	Caractéristiques
template-driven	Simples paramétrages dans le templates HTML, selon standard HTML5, pas ou très peu de code typescript/ javascript
model-driven (alias reactive-forms)	Manière plus précise de paramétrier le comportement des validations de formulaire (moins de paramétrages coté HTML) , plus de code typescript
via Form-builder API	Variante sophistiquée de model-driven / reactive-forms

L'approche la **plus simple** et la **plus classique** est "**template-driven**".

L'approche "**model-driven**" (un plus complexe et très différente) sera étudiée ultérieurement pour ne pas apporter de confusion.

Rappel (configuration nécessaire dans le module) :

app.module.ts

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';

...
@NgModule({
  imports:    [ BrowserModule, FormsModule, ... ],
  declarations: [ AppComponent,  ],
  providers:  [ ... ],
  bootstrap:  [ AppComponent ]
})
export class AppModule { }
```

1.2. 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)
- ...

Exemples :

```
... <input [(ngModel)]="login.username" required pattern="^ [a-zA-Z].+/">
... <input [(ngModel)]="login.password" required minlength="3"/>
.... <input [(ngModel)]="login.roles" required />
```

1.3. Activations automatiques de classes css (ng-invalid , ...)

En mode "template driven", le framework *Angular analyse les contraintes de validation du standard HTML5 et active ou désactive automatiquement certaines classes de styles css* :

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

Exemples de **styles.css**

```
.ng-valid[required] {
  border-left: 5px solid #42A948; /* green */
}

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

Name

Dr IQ

temp class name: form-control ng-pristine ng-valid **ng-touched**

(si visité)

temp class name: form-control **ng-untouched** ng-pristine ng-valid

(si pas visité)

Name

Hercule

temp class name: form-control ng-valid ng-touched **ng-dirty**

lorsque modifié

Name

temp class name: form-control ng-touched **ng-dirty** **ng-invalid**

si invalide

1.4. Exemples de paramétrages HTML (ngForm), template-driven

Attention:

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 coté ng serve ou bien du coté console du navigateur .

NB: un formulaire est globalement considéré comme valide que lorsque tous les champs de celui ci sont valides .

Bouton "submit" grisé tant que l'ensemble du formulaire n'est pas encore entièrement valide :

```
<form #formXy="ngForm">
  <label for="name">Name</label>
  <input type="text" name="name"
    [(ngModel)]="model.name" required > <br/>
  ....
  <button type="button" [disabled]="!formXy.form.valid"

```

NB : L'intérêt d'écrire explicitement

<form #formXy="ngForm" > est de pouvoir écrire plus bas

<button type="button" [disabled]="!formXy.form.valid">Submit</button>

<!-- déclencheur d'action grisé tant que formulaire pas globalement valide -->

Eventuels messages d'erreurs montrés ou cachés:

En déclarant une variable locale de référence associée à l'objet du champ de saisie via la syntaxe **#nameFormCtrl="ngModel"** , on peut afficher de façon conditionnée certains messages d'erreurs :

```
<input type="text" class="form-control"
[(ngModel)]="model.name" #nameFormCtrl="ngModel" required />

<div [hidden]="nameFormCtrl.valid" class="alert alert-danger">
    Name is required
</div>
```

nameFormCtrl.valid (true or false)

nameFormCtrl.dirty (true or false)

nameFormCtrl.touched (true or false)

Soumission d'un formulaire:

```
<div [hidden]="submitted">
<h1>Coords Form</h1>
<form #formXy="ngForm">
.....
<button type="button" (click)="onSubmit()"
        [disabled]="!formXy.form.valid">Submit</button>
</form>
</div>

<div [hidden]!="submitted">
... <!-- actions au cas par cas après la soumission du formulaire -->
</div>
```

```
.....
export class CoordsFormComponent {
.....
    submitted = false;
    onSubmit() { this.submitted = true; // ou autre }
.....
}
```

2. Formulaires en mode reactive-form

2.1. Approche "model-driven" / "reactive-form"

dans composant angular :

```
import { FormGroup, FormControl, Validators } from '@angular/forms';
...
class ModelFormComponent implements OnInit {
  myForm: FormGroup;
  constructor() {
    this.myForm = new FormGroup({
      firstName: new FormControl('', Validators.required ),
      lastName: new FormControl('default_name', [ Validators.required ,
                                                    Validators.pattern("[A-Z].+") ]),
      email: new FormControl('', [ Validators.required, Validators.email ]),
      password: new FormControl( '',[ Validators.required, Validators.minLength(8) ] )
    });
  }
}
```

dans app.module.ts ou @Component() :

```
import { ReactiveFormsModule } from '@angular/forms';
...
imports: [ ... FormsModule, ReactiveFormsModule, ... ],
...
```

dans template HTML :

```
<form novalidate [formGroup]="myForm">
  <div class="form-group">
    <label>First Name</label>
    <input type="text" class="form-control" formControlName="firstName" >
  </div>
  <div class="form-group">
    <label>Last Name</label>
    <input type="text" class="form-control" formControlName="lastName" >
  </div>
  <div class="form-group">
    <label>Email</label>
    <input type="email" class="form-control" formControlName="email" >
  </div>
  ...
  <input type="button" value="submit" (click)="onSubmit()" 
        [disabled]=""!myForm.valid"" />
  <br/>
</form>
```

NB :

- **novalidate** (dans <form ...>) **signifie pas de validation HTML5 automatiquement effectuée par le navigateur mais simplement par l'application angular .**
- **en mode "model-driven" / "reactive-form"**, pas besoin de `[(ngModel)]="xxx.yyy"` mais on récupère (dans `onSubmit()` ou ...) les données saisies au sein de **`myForm.value`** .
- Pour transformer **`myForm.value`** en une instance d'une classe de données (data / model / ...) on peut par exemple s'appuyer sur un bon constructeur :

```
public onSubmit(){
  const r = this.myForm.value; //rawFormValuesObject
  this.reservation = new Reservation(r.firstName , r.lastName , r.telephone ,
                                      r.email , this.dateTimeFromDateAndLocalTime(r.date,r.time) );
  this.message="résa effectuée=" + JSON.stringify(this.reservation);
}
```

Accès aux détails d'un champ d'un formulaire contrôlé par angular :

`myForm.controls['formControlName'].errors // .dirty , .valid , ...`

Exemple (à adapter):

```
myFieldErrorMessage(fieldName:string, fieldExpectation:string=""){
  const myForm= this.xyzForm;
  //myForm.controls[fieldName].invalid && (myForm.controls[fieldName].dirty || myForm.controls[fieldName].touched)
  let errMsg="";
  const vErrors : ValidationErrors | null = myForm.controls[fieldName].errors;
  if(vErrors!=null){
    console.log(JSON.stringify(vErrors));
    if(vErrors['required']==true)
      errMsg=`${fieldName} is required`;
    else if(vErrors['minlength'])
      errMsg=`${fieldName}.minLength=${vErrors['minlength'].requiredLength}`;
    else if(vErrors['email']==true)
      errMsg=`${fieldName} should be a valid email with ...@...`;
    else
      errMsg=`${fieldName} ${fieldExpectation}`;
  }
  return errMsg;
}
```

Utilisation du côté .html :

<div class="f-align smallError">{{myFieldErrorMessage('lastName')}}</div>
ou bien

`{{myFieldErrorMessage('lastName','must start with uppercase and other character')}}`

2.2. Avec l'aide de FormBuilder

En mode *model-driven / reactiveForm*, de façon à construire plus simplement le paramétrage d'un FormGroup avec FormControl et Validateurs imbriqués, on peut éventuellement s'appuyer sur FormBuilder :

Exemple :

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';

@Component({ ... })
export class AppComponent implements OnInit {
  myForm: FormGroup;
  constructor(private _formBuilder: FormBuilder) {
    this.myForm = this._formBuilder.group({
      name: ['default_name', [Validators.required, Validators.pattern("[A-Z].+")]],
      email: ['', [Validators.required, Validators.email]],
      message: ['', [Validators.required, Validators.minLength(15)]]
    });
  }
}
```

Attention aux double `[]` :

- un premier pour `['valeur_par_defaut'`, validateur(s)]
- un second imbriqué pour la [liste des Validateurs](#)

2.3. Exemple de Validateur personnalisé / spécifique :

url.validator.ts

```
import { AbstractControl } from '@angular/forms';

export function ValidateUrl(control: AbstractControl) {
  if (!control.value.startsWith('https') || !control.value.includes('.io')) {
    return { invalidUrl: true }; //return { errorKeyname : true } if invalid
  }
  return null; //return null if ok (no error)
}
```

Utilisation :

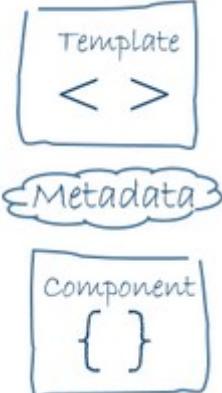
```
this.myForm = this._formBuilder.group({
  userName: ['', Validators.required],
  websiteUrl: ['', [Validators.required, ValidateUrl ]],
});
```

VIII - Components et modules (approfondissement)

1. Précisions sur les composants (@Component)

1.1. Anatomie d'un composant de angular 2+

Rappels:

	<p>Un composant "angular 2+" est essentiellement constitué d'une classe codant le la structure et le comportement de celui ci (par exemple : réactions aux événements).</p> <p>Les métadonnées sont introduites par une ou plusieurs décorations placées au dessus de la classe (ex : <code>@Component</code>). <u>Vocabulaire typescript</u> : décoration plutôt qu'annotation .</p> <p>La vue graphique générée par un composant est essentiellement structurée via un template HTML/CSS comportant des syntaxes spécifiques "angular 2+" (<code>*ngFor</code>, <code>{{ }}</code>).</p>
--	---

<p>Les liaisons/correspondances gérées par le framework angular2+ entre les éléments du composant "orienté objet" et les éléments de la vue "web/HTML/DOM" issue du template sont appelées "data binding".</p> <p><u>Nuances :</u></p> <ul style="list-style-type: none"> [propertyBinding] (eventBinding) [(two-way data binding)] 	<p>DOM/Template/User</p> <p>Component (Object/Memory)</p> <pre> <----- {{xy}} <----- [propXy]="xy" ----- (eventZz)="onZz(\$event)" ----- <----- [(ngModel)]="xy" ----- </pre>
---	--

1.2. Vue d'ensemble sur input et output

input() et output() pour composants réutilisables

dans composant parent

```
<c1 [p1]="'valeurP1'" (eventA)="onEvtA($event)"></c1>
```

```
@Component({selector :'c1',...})
SousComposant 1 {

  p1 = input('defaultValue')

  eventA = output<...>();

  onInternalEvt(){
    this.eventA.emit(...);
  }
}
```

et éventuelles
répercussions
sur

Autre(s)
sousComposant(s)

Principe fondamental (commun au framework concurrent "react") :

Au sein d'une arborescence de composants ,

- les ordres/directives/paramétrages descendant
- les événements remontent

1.3. @Input() avant angular 17 (moins bien que input())

@Input (du coté sous-composant) permet de récupérer des valeurs (fixes ou variables et dynamiques) qui sont spécifiées par un composant parent/englobant .

Exemple élémentaire : header.component.ts

```
import { Component , Input } from '@angular/core';
@Component({
  selector: 'app-header',
  templateUrl: './header.component.html'
})
export class HeaderComponent {
  @Input()
  titre /*: string*/ ="titreParDefaut";

  @Input()
  infos /*: string*/ ="";
}
```

header.component.html

```
<h3>MyHeader {{titre}} .. {{infos}}</h3>
```

Exemple d'utilisation depuis un composant parent : app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  title /* :string */ = 'titre1';
  message: string = "Welcome to my Angular 2+ App";
}
```

app.component.html

```
<app-header [titre]="title" infos="***" ></app-header>
<h1>{{message}} .. </h1> ...
```

MyHeader titre1 .. **

date:Mon Dec 12 2016 11:54:09 GMT+0100 (CET)

My First Angular 2 App ..



Eventuels paramétrages avancés (pour cas très pointus) :

```
@Input('title') // pour <myheader title='titre 1' /> (vue externe)
titre : string ; //pour this.titre en interne dans le sous composant
```

@Attribute dans constructeur ressemble un peu à @Input

```
export class Child {
  isChecked;
  constructor(@Attribute("checked") checked) {
    this.isChecked = !(checked === null);
  }
}
```

avec cette utilisation potentielle :

```
<child checked></child>
<child checked='true'></child>
<child></child>
```

1.4. input() signal (alternative à @Input)

Au sein des versions récentes d'angular , on peut remplacer @Input par la **fonction input()** produisant un signal.

Principal avantage: meilleure synchronisation (plus performante).

Code du coté "sous composant" :

```
import { Component, inject, input } from '@angular/core';
...
export class HeaderComponent {
  titre = input("titre_par_defaut"); //input<string or ...>(...)
  ...
}
```

```
... {{titre0}}...
```

Pas de changement du coté "utilisation externe".

Variantes d'utilisation :

```
<app-header></app-header> <!-- avec valeur par défaut -->
<app-header titre="monAppli"></app-header> <!-- avec valeur fixe -->
<app-header [titre]="'monAppli'"></app-header> <!-- syntaxe complexe possible -->
<app-header [titre]="title"></app-header> <!-- avec valeur variable
(copie dynamique de title) -->
```

1.5. @Output() avant angular 17 (assez complexe)

@Output (au niveau d'un sous-composant) permet de déclarer un **événement** qui sera potentiellement remonté et géré par un composant parent/englobant.

Exemple :*reglette.component.ts*

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

@Component({
  selector: 'app-reglette',
  templateUrl: './reglette.component.html',
  styleUrls: ['./reglette.component.scss']
})
export class RegletteComponent implements OnInit {

  @Input()
  width/*:string*/ = "100"; //largeur paramétrage (100px par défaut)

  @Output()
  changeEvent = new EventEmitter<{value:number}>();

  onCurseur(event : Event){
    const evt : MouseEvent = <MouseEvent> event;
    const valX = evt.offsetX;
    const pctCurseur = (valX / Number(this.width)) * 100 ; //en %
    this.changeEvent.emit({value:pctCurseur});
  }
}
```

reglette.component.html

```
<div class="reglette" (click)="onCurseur($event)"
  [style.width.px] = "width">
</div>
```

reglette.component.css

```
.reglette {
  width : 100px;  height : 40px;
  background: linear-gradient(to right, white, blue);
  border-style : solid;  border-width : 2px;  border-color : blue;
}
```

Utilisation au niveau d'un composant parent :

```
...
export class DemoComponent implements OnInit {
  valeurCurseur /*:number*/ =0;

  onChangeCurseur(event : any){
    const evt : {value:number} = event;
    this.valeurCurseur = evt.value;
```

```
}
```

```
<app-reglette width="200" (changeEvent)="onChangeCurseur($event)"></app-reglette>
curseur = {{valeurCurseur}}
```



curseur = 49

si click dans le milieu de la réglette



curseur = 1

si click dans le début de la réglette (coté gauche)



curseur = 97

si click dans la fin de la réglette (coté droit)

1.6. output() en tant que signal depuis angular 17

Depuis angular 17 on peut utiliser output() sous forme de signal plutôt que @Output(). Ainsi , l'exemple précédent se simplifie de cette manière :

```
import { Component, input, output } from '@angular/core';
@Component({...})
export class RegletteComponent {

width /*:string*/ = input("100"); //largeur paramétrage (100px par defaut)

//@Output()
//changeEvent = new EventEmitter<{value:number}>();
changeEvent = output<{value:number}>();

onCurseur(event : Event){
  const evt : MouseEvent = <MouseEvent> event;
  const valX = evt.offsetX;
  const pctCurseur = (valX / Number(this.width)) * 100 ; //en %
  //console.log("pctCurseur="+pctCurseur);
  this.changeEvent.emit({value:pctCurseur});
}

}
```

1.7. Projection d'éléments imbriqués (<ng-content>)

Un composant angular peut (en tant que nouvelle balise telle que *toggle-panel* ici) , incorporer à son tour certains sous éléments (ex : <div ...> ou autre sous-sous composant angular).

Exemple :

```
<toggle-panel title="panel1">
    <app-part1></app-part1> <!-- ou ... , vu comme ng-content dans toggle-panel -->
</toggle-panel>

<toggle-panel title="panel2">
    <div>contenu du panneau 2</div> <!-- vu comme ng-content dans toggle-panel....html -->
</toggle-panel>
```

Le (ou le paquet de) sous-composant(s)/sous-balises imbriqué au niveau de l'utilisation d'un composant réutilisable sera vu via la balise spéciale `<ng-content></ng-content>` au sein du template HTML (code interne) de ce composant.

Cette fonctionnalité était appelée "transclusion" au sein des directives "angular Js 1.x" , elle est maintenant appelée "**projection de contenu**" au sein des composants angular 2+ .

Exemple concret : composant réutilisable "togglePanel" basé sur des styles *bootstrap* et fontes *bootstrap-icons* :

```
import { Component, input , model } from '@angular/core';
@Component({
  selector: 'toggle-panel',
  templateUrl: './toggle-panel.component.html' , styleUrls: ['./toggle-panel.component.scss']
})
export class TogglePanelComponent {
  panelOpenState=model(false);

  title /* : string */ = input( 'default panel title');
  constructor() { }
}
```

version du template html basée sur les classes css bootstrap 5 :

```
<div class="card mb-2">
  <div class="card-header text-white bg-primary"
      (click)="panelOpenState.set(!panelOpenState())" >
    {{title}} <i [class.bi-arrow-down-circle-fill]="!panelOpenState()"
               [class.bi-arrow-up-circle-fill]="panelOpenState()" ></i>
  </div>
  <div class="card-body collapse" [class.show]="panelOpenState()">
    <ng-content></ng-content> <!-- remplacé par le contenu imbriqué -->
  </div>
</div>
```

version du template html basée sur les classes css spécifiques :

```
<div class="my-card">
  <h4 class="my-card-header my-bg-primary">
    <a class="my-text-light" (click)="panelOpenState.set(!panelOpenState())" >
      <span class="my-icon" [style.display]="panelOpenState()?'none':'inline-block'">+</span>
      <span class="my-icon" [style.display]="panelOpenState()?'inline-block':'none'">-</span>
    {{title}}
```

```

        </a>
</h4>
<div class="my-card-body my-collapse" [class.my-show]="panelOpenState()">
    <ng-content></ng-content> <!-- remplace par le contenu imbriqué -->
</div>
</div>

```

toggle-panel.component.scss

```

.my-card { margin-top: 0.1em; margin-bottom: 0.1em; }
.my-card-header { border-top-left-radius: 0.3em; border-top-right-radius: 0.3em;
    padding: 0.1em; margin-bottom: 0px; }
a { text-decoration: none; }
.my-card-body {border: 0.1em solid blue; border-bottom-left-radius: 0.3em;
    border-bottom-right-radius: 0.3em; padding: 0.2em; }
.my-bg-primary { background-color: blue; }
.my-text-light { color : white; }
.my-icon { color : blue; background-color: white; margin: 0.2em;
    padding-left: 0.2em; padding-right : 0.2em;
    min-width: 1em; font-weight: bold; }
.my-collapse { display : none; }
.my-show { display : block ;}

```

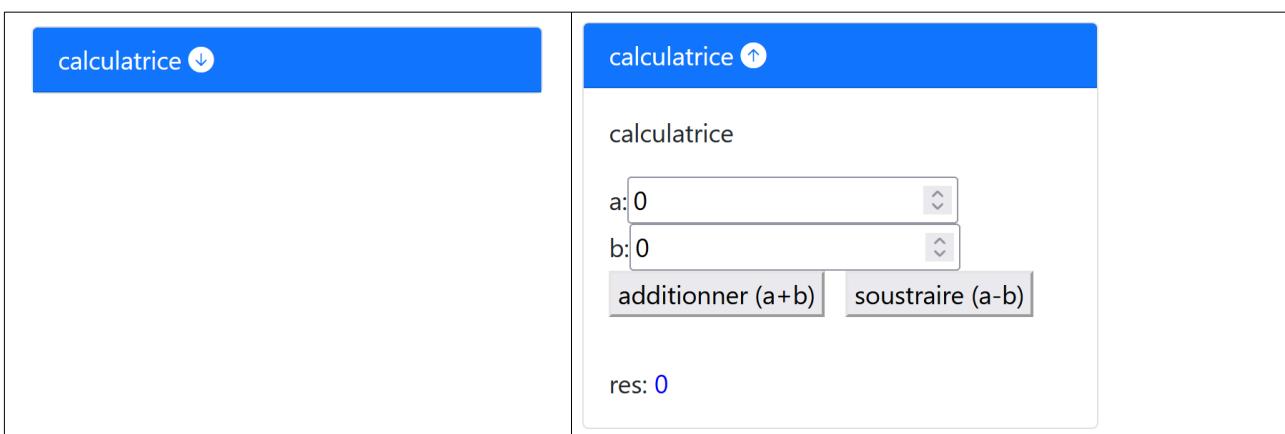
Exemple d'utilisation :

```

<toggle-panel title="calculatrice">
    <app-calculatrice></app-calculatrice> <!-- ou ... , vu comme ng-content dans toggle-panel -->
</toggle-panel>

<toggle-panel title="calcul tva">
    <app-tva></app-tva> <!-- vu comme ng-content dans toggle-panel....html -->
</toggle-panel>

```



2. Cycle de vie sur composants (et directives)

Interfaces (à facultativement implémenter)	Méthodes (une par interface)	Moment où la méthode est appelée automatiquement par angular2
OnChanges	ngOnChanges()	Dès changement de valeur d'un "input binding" (exemple : "propriété initialisée selon niveau parent")
OnInit	ngOnInit()	A l'initialisation du composant et après les premiers éventuels ngOnChanges() et après constructeur et injections
DoCheck AfterContentInit AfterContentChecked AfterViewInit AfterViewChecked	ngDoCheck() ngAfterContentInit() ngAfterContentChecked() ngAfterViewInit() ngAfterViewChecked()	pour cas très pointus avec détection spécifique des changements à afficher ou pas,
OnDestroy	ngOnDestroy()	Juste avant destruction du composant

Exemple :

liste-comptes.component.ts

```
import { Component, input, OnInit } from '@angular/core';
import { PreferencesService } from './common/service/preferences.service';

@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.scss']
})
export class HeaderComponent /* implements OnInit */{

  titre=input("titreQuiVaBien");

  constructor() { console.log("dans constructeur de HeaderComponent , titre=" + this.titre) }

  ngOnInit(): void { console.log("dans ngOnInit() de HeaderComponent , titre=" + this.titre) }
}
```

Dans la console du navigateur :

dans constructeur de HeaderComponent , titre=titreQuiVaBien

dans ngOnInit() de HeaderComponent , titre=myApp

ngOnInit() de angular ressemble un peu à *@PostConstruct* de java/jee et permet de programmer

des initialisations au bon moment (pas trop tôt) .

3. Aperçu sur les directives (angular2+)

3.1. Les 3 types/niveaux de directives d'angular2:

Attribute Directive	Change l'apparence ou le comportement d'un (souvent seul) élément de l'arbre DOM (exemple <code>ngStyle</code>)
Structural Directive	Change la structure de l'arbre DOM (et donc des éléments affichés) en ajoutant ou supprimant des sous éléments dans l'arbre DOM (exemple : <code>*ngIf</code> , <code>*ngFor</code> , <code>ngSwitch</code>)
Component	élément composite de l'arbre DOM (selon structure du template HTML associé)

Au final , `@Component` peut être vu comme un cas particulier (et assez fréquent) de directive.

3.2. Directive (de niveau "attribut")

Exemple (tiré du "tutoriel officiel") :

app/highlight.directive.ts

```
import {Directive, ElementRef, Input} from '@angular/core';

@Directive({ selector: '[myHighlight]' })
export class HighlightDirective {
    constructor(el: ElementRef) {
        el.nativeElement.style.backgroundColor = 'yellow';
    }
}
```

Le paramétrage le plus important est la décoration `@Directive` .

Le nom du **sélecteur** CSS doit être encadré par des **crochets** lorsqu' il s'agit d'une directive.

`el` de type `ElementRef` correspond à un élément de l'arbre DOM dont il faut mettre à jour le rendu.

Exemple d'utilisation :

app/app.module.ts

```
import { NgModule }      from '@angular/core';
...
import {HighlightDirective} from './highlight.directive'

@NgModule({
  imports:  [ BrowserModule , FormsModule ],
  declarations: [ AppComponent , MyHeaderComponent , HighlightDirective ],
  providers:  [ ],
  bootstrap:  [ AppComponent ]
})
export class AppModule { }
```

app/app.component.ts

```
import {Component} from 'angular2/core';
@Component({
  selector: 'my-app',
  template: ' ... <span myHighlight>Highlight me!</span>'
})
export class AppComponent { }
```

Résultat:

My First Angular 2 App

Highlight me!

Version améliorée (avec paramètre via `@Input` et gestion d'événements) :

```
import {Directive, ElementRef, HostListener, Input} from '@angular/core';
@Directive({ selector: '[myHighlight]'})
export class HighlightDirective {
  constructor(private el: ElementRef) { }
  private _defaultColor = 'red';

  @Input('myHighlight')
  public highlightColor: string = this._defaultColor;

  @Input() set defaultColor(colorName:string){
    this._defaultColor = colorName || this._defaultColor;
  }
}
```

```

@HostListener('mouseenter')
onMouseEnter() { this._highlight(this.highlightColor || this._defaultColor); }

@HostListener('mouseleave')
onMouseLeave() { this._highlight(null); }

private _highlight(color: string | null)
  { this.el.nativeElement.style.backgroundColor = color; }
}

```

NB :

Sans argument, `@Input()` fait que la propriété exposée a le même nom que celle de la classe (public ou get / set).

Avec un argument , `@Input` permet de préciser un alias sur la propriété exposée (ex : 'myHighlight' plutôt que `highlightColor`).

`@HostListener` permet d'associer des noms d'événements (déclenchés sur l'élément DOM courant) à une méthode événementielle .

Utilisation :

```
<p [myHighlight]="'yellow'" [defaultColor]="'violet'">Highlight me!</p>
```

ou bien (plus simplement) :

```
<p myHighlight="yellow" defaultColor="violet">Highlight me!</p>
```

Pick a highlight color

Green Yellow Cyan

ou bien (avec un choix dynamique) :

```

<h4>Pick a highlight color</h4>
<div> <input type="radio" name="colors" (click)="color='lightgreen'" id="r1" />
    <label for="r1">Green</label>
    <input type="radio" name="colors" (click)="color='yellow'" id="r2" />
    <label for="r2">Yellow</label>
    <input type="radio" name="colors" (click)="color='cyan'" id="r3" />
    <label for="r3">Cyan</label>
</div>
<span [myHighlight]="color"> Highlight with choosen color</span> <br/>

```

Version plus sophistiquée via @HostBinding

via **@HostBinding()** on peut directement associer une propriété de la classe de directive à une propriété de style ou autre de l'élément "host" sur laquelle la directive sera appliquée.

Syntaxes possibles :

@HostBinding('style.xyz') **@HostBinding('attr.xyz')** **@HostBinding('class.xyz')**

HostBinding('value') myValue; est à l'intérieur d'une classe de directive un équivalent de **[value] = "myValue"** que l'on écrirait dans un template html .

et

HostListener('click') myClick(){} est à l'intérieur d'une classe de directive un équivalent de **(click) = "myClick()"** que l'on écrirait dans un template html .

```
import { Directive, Input, HostListener, HostBinding } from '@angular/core';

@Directive({
  selector: '[myHighlight]'
})

export class HighlightDirective {
  private _defaultColor = 'red';

  @Input('myHighlight') public highlightColor: string = this._defaultColor;

  @HostBinding('style.backgroundColor') backgroundColor: string;

  @HostListener('mouseenter')
  onMouseEnter() { this._highlight(this.highlightColor || this._defaultColor); }

  @HostListener('mouseleave')
  onMouseLeave() { this._highlight(null); }

  private _highlight(color: string | null) { this.backgroundColor = color ? color : 'white'; }
}
```

3.3. Directive structurelle

Une directive structurelle (ajoutant ou retirant des sous éléments dans l'arbre DOM) se programme de façon très semblable à une directive d'attribut (même décoration `@Directive`, même syntaxe (avec crochets) pour le sélecteur CSS). La principale différence tient dans les éléments injectés dans le constructeur :

- `TemplateRef` correspond à la branche des sous éléments imbriqués (à supprimer ou ajouter ou ...)
- `ViewContainerRef` permet de contrôler dynamiquement le contenu via des méthodes prédéfinies telles que `.clear()` ou `.createEmbeddedView()`

Exemple "myUnless" (tiré du tutoriel officiel) :

```
import {Directive, Input} from '@angular/core';
import {TemplateRef, ViewContainerRef} from '@angular/core';

@Directive({ selector: '[myUnless]' })
export class UnlessDirective {

  constructor( private _templateRef: TemplateRef<any>,
    private _viewContainer: ViewContainerRef ) {}

  @Input() set myUnless(condition: boolean) {
    if (!condition) { this._viewContainer.createEmbeddedView(this._templateRef); }
    else { this._viewContainer.clear(); }
  }
}
```

Utilisation (comme *ngIf) :

```
age: <input type='text' [(ngModel)]="age" /><br/>
<p *myUnless="age>=18">
  condition "age>=18" is false and myUnless is true.
</p><br/>

<p *myUnless="!(age>=18)">
  condition "age>=18" is true and myUnless is false.
</p>
```

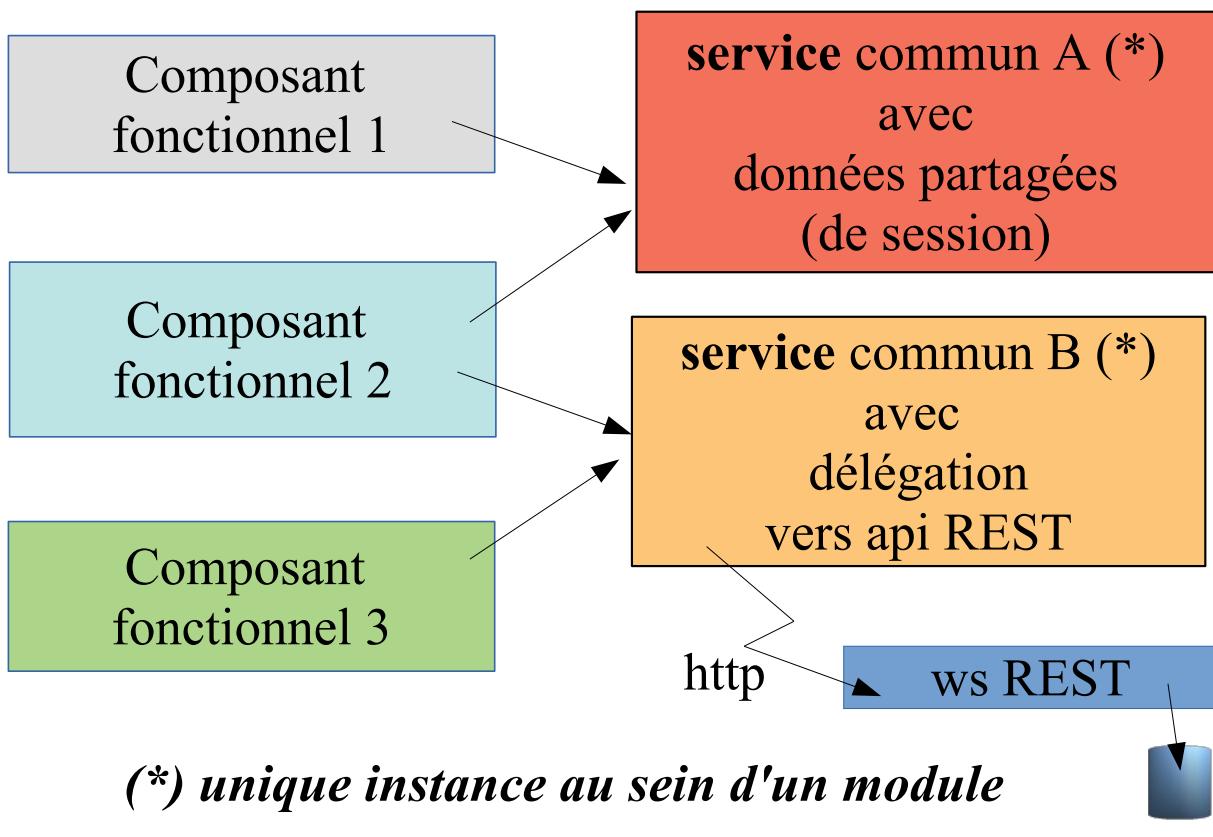
IX - Services et injections (essentiel)

1. Services "angular" (concepts et bases)

Un service est un module de code "invisible" comportant des traitements "ré-utilisables" et souvent "partagés" tels que :

- des accès aux données (*indirectement via ajax/http et ws rest/json*)
- des mises à jours de données (*indirectement via ajax/http et ws rest/json*)
- données partagées (par plusieurs composants) en mémoire *dans l'appli angular (coté navigateur)*
- gestion de la session utilisateur (authentification , ...)
- traitements réutilisables (calculs , traductions , ...)
- ...

Services pour composants fonctionnels



NB : Pour synchroniser plusieurs composants visuels sur l'affichage de données communes/partagées on pourra :

- éventuellement injecter (publiquement) un même service dans ces différents composants visuels puis utiliser des expressions telles que `{{serviceDataXy.subData.pXy}}`
- et/ou utiliser "**BehaviorSubject**" (cas particulier de "**Observable**") (*voir autre chapitre "aspect divers et avancés"*).

- Au sein des versions récentes d'angular, on peut également utiliser les **signaux** comme alternative à BehaviorSubject/Observable

Exemple simple:

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);

  //public couleurFondPreferee :string = 'lightgrey'; //v1 : public
  private _couleurFondPreferee :string = "?"; //v2 : private + get/set + localStorage

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

REMARQUES TRES IMPORTANTES:

- Après une navigation d'un composant vers un nouveau , les données internes au composant précédent sont généralement perdues.
- En plaçant les données importantes dans un service partagé (qui est maintenu en mémoire tout le temps de fonctionnement de l'application) , les données importantes sont conservées plus longtemps en mémoire (plus perdues après navigation).
- En cas de "refresh" complet de l'application déclenchable au niveau du navigateur, toute l'application est réinitialisée et toutes les valeurs en mémoire dans un service seraient également perdues (sauf si l'on stocke et récupère celles ci dans la zone "localStorage" du navigateur) . Si besoin , on peut stocker et récupérer des blocs de données complets dans "localStorage" ou "sessionStorage" en utilisant JSON.stringify() et JSON.parse() .

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 pourvoir 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() { }
}
```

Autre type de service classique:

SessionService (avec données de type .username .isConnected ...)

2. Injection de dépendances (bases)

A l'époque du framework "angular 1" , l'injection de dépendances consistait à automatiquement relier entre eux deux composants via des correspondances entre "nom de service" et nom d'un paramètre d'une fonction "contrôleur" .

Angular 2+ gère l'injection de dépendances de manière plus typée et plus orientée objet.

2.1. Enregistrement des éléments qui pourront être injectés:

L'injection de dépendances gérée par Angular2+ passe par un enregistrement des fournisseurs de choses à potentiellement injecter.

Ceci s'effectue généralement au moment du "bootstrap" et se configure au niveau de la partie ("providers :") de la décoration **@NgModule** d'un module applicatif .

Exemples :

```
...
@NgModule({
  ...
  providers: [ PreferencesService, SessionService ],
  bootstrap: [ AppComponent ]
})
export class AppModule {
```

Cas très rare : Si un élément potentiellement injectable n'est pas, globalement, enregistré au niveau global (@NgModule) , il pourra éventuellement être déclaré, localement, au niveau des "providers" spécifiques d'un composant (**Attention : dans ce cas pas de partage/singleton!!!**) :

```
@Component({
  selector: 'my-app',
  template: `...`,
  providers: [ MyNotSharedService ]
})
export class AppComponent{ ... }
```

La documentation officielle d'Angular 2+ parle en terme de "**root_injector**" (pour de niveau @NgModule) et de "**child_injector**" (pour les sous niveaux : @Component , ...)

2.2. Nouveauté/évolution à partir de la version 6

A partir de la version 6 d'angular , la décoration `@Injectable()` comporte un paramètre important nommé "`providedIn`" .

```
@Injectable({
  providedIn: 'root'
})
export class XyService { ... }
```

La valeur de `providedIn` correspond souvent à '`root`' (au sens "root injector" de niveau module ou principal (en mode "standalone") et dans ce cas le service "XyService" sera automatiquement considéré comme fourni par le contexte (app.module.ts ou autre) .

Autrement dit , via ce nouveau paramétrage, plus absolument besoin de placer XyService dans la partie providers : [] de @NgModule() , le service XyService sera automatiquement fourni à tous les composants qui en auront besoin (ceux qui auront paramétré une injection de dépendance) .

2.3. Injection de dépendance via constructeur

```
@Component({
  selector: 'my-app',
  template: `...` })
export class AppComponent {
  ...
  constructor(public sessionService: SessionService) {
    // this.sessionService (de type SessionService) est automatiquement initialisé
    // par injection si métadonnées introduites via @Component ou @Injectable ou ...
  }
  ...
}
```

Angular2+ initialise automatiquement les éléments passés en argument des constructeurs lorsqu'il le peut (ici par injection du service de type `SessionService`). Cet automatisme n'est déclenché que si la classe du composant est décorée par `@Component()` ou `@Injectable()` ou ...

Pour que des injections de dépendances puissent être gérées au niveau du constructeur de la classe courante :

- au minimum `@Injectable()` (*au sens "sous-composants , sous-services automatiquement injectables"*)
- assez souvent `@Component()` (qui peut plus peut moins)

si paramètre du constructeur pas typé alors `@Inject(ClassComponent)` permet de préciser le type de composant à injecter

2.4. Injection via fonction inject() de @angular/core

Attention, ceci n'est possible qu'avec une version récente d'angular .
La fonction inject() est apparue au sein de la version 14 et est couramment utilisée au sein des versions 16,17,18+ .

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

@Component(...)
export class XyzComponent {

  /*
  constructor(public preferencesService: PreferencesService){
    //dependency injection with constructor
  }
  */

  //better dependency injection (in case of inheritance or if used in function)
  public preferencesService = inject(PreferencesService);

  ...
}
```

NB: Cette nouvelle manière de paramétriser une injection de dépendance est surtout très pratique lorsque l'on a besoin d'injecter un service angular dans des "intercepteurs" ou "gardiens" qui sont codés comme des fonctions (plutôt que comme des classes) au sein des versions récentes d'angular.

X - Programmation réactive et RxJs

1. introduction à RxJs

1.1. Principes de la programmation réactive

La programmation réactive consiste essentiellement à programmer un enchaînement de traitements asynchrones pour réagir à un flux de données entrantes .

Un des intérêts de la programmation réactive réside dans la souplesse et la flexibilité des traitements fonctionnels mis en place :

- En entrée, on pourra faire librement varier la source des données (jeux de données statiques , réponses http , input "web-socket" , événement DOM/js ,)
- En sortie, on pourra éventuellement enregistrer plusieurs observateurs/consommateurs (si besoin de traitement en parallèles . par exemple : affichages multiples synchronisés) .

1.2. RxJs (présentation / évolution)

RxJs est une bibliothèque **javascript** (assimilable à un mini framework) spécialisée dans la programmation réactive .

Il existe des variantes dans d'autres langages de programmation (ex : RxJava , ...) .

RxJs s'est largement inspiré de certains éléments de programmation fonctionnelle issus du langage "**scala**" (map ,flatMap , filter , reduce , ...) et a été à son tour une source d'inspiration pour "**Reactor**" utilisable au sein de Spring 5 .

RxJs a été dès 2015/2016 mis en avant par le framework Angular qui à fait le choix d'utiliser "Observable" de RxJs plutôt que "Promise" dès la version 2.0 (Angular 2) .

Depuis, le framework "Angular" a continuer d'exploiter à fond la bibliothèque RxJs . Cependant , les 2 frameworks ont beaucoup évolué depuis 2015/2016 .

La version **4.3** de **Angular** a apporter de grandes simplifications dans les appels de WS-REST via le service **HttpClient** (rendant *obsolète* l'ancien service *Http*) .

La version 6 de Angular a de son coté été *restructurée* pour intégrer les gros changements de **RxJs 6** . Heureusement, moins de bouleversement dans les versions ultérieures (mais cependant quelques petits changements au niveau des import et des "deprecated") .

La version 6 de RxJs s'est restructurée en profondeur sur les points suivants :

- changement des éléments à importer (nouvelles syntaxes pour les import { } from "")
- changements au niveau des opérateurs à enchaîner (plus de préfixe , pipe() , ...) .

1.3. Principales fonctionnalités d'un "Observable"

Observable est la structure de données principale de l'api "RxJs" .

- Par certains cotés , un "Observable" ressemble beaucoup à un objet "Promise" et permet d'enregistrer élégamment une suite de traitements à effectuer de manière asynchrone .
- En plus de cela , un "Observable" peut facilement être manipulé / transformé via tout un tas d'opérateurs fonctionnels prédéfinis (ex : map , filter , sort , ...)
- En outre , comme son nom l'indique , un "Observable" correspond à une mise en oeuvre possible du design pattern "observateur" (différents observateurs synchronisés autour d'un même sujet observable).

2. Fonctionnement (sources et consommations)

Source_configurée_et_initialisée

```
.pipe(
  callback_fonctionnelle_1 ,
  callback_fonctionnelle_2 ,
  ...
) .subscribe({
  next : callback_success ,
  error : callback_error ,
  terminate : callback_terminate
})
```

Source_configurée_et_initialisée

```
.pipe(
  callback_fonctionnelle_1 ,
  callback_fonctionnelle_2 ,
  ...
) .subscribe(callback_success );
```

NB: la partie `.pipe()` est **facultative** et ne sert qu'à enchaîner une éventuelle série de traitements intermédiaires .

NB : les parties `error : callback_error` et `terminate : callback_terminate` de `.subscribe()` sont **facultatives** et peuvent donc être omis si elles ne sont pas utiles en fonction du contexte.

NB : il faut (en règle général , sauf cas particulier/indication contraire) appeler `.subscribe()` pour que la chaîne de traitement puisse commencer à s'exécuter .

3. Convention de nom pour "return Observable"

Certains développeurs angular/RxJs ont pris l'habitude de mettre le **suffixe \$ à la fin des noms de méthodes qui retourne en interne un objet Observable** .

Exemple partiel:

```
public getAllDevises$() : Observable<Produit[]>{
  //const url = `${this.publicBaseUrl}/devise`;
  //console.log( "url = " + url);
  //return this._http.get<Devise[]>(url);
  return of(this.tabProduit) ;
}
```

4. Organisation de RxJs

4.1. Imports dans projet Angular / typescript

```
import { Observable , of } from 'rxjs';
import { map , mergeMap ,toArray ,filter} from 'rxjs/operators';
```

exemple d'utilisation (dans classe de Service) :

```
public rechercherProduitSimu$(prixMaxi : number) : Observable<Produit[]> {
    let tabProduit = [
        { numero : 1 , label : "produit 1" , prix : 50 } ,
        { numero : 2 , label : "produit 2" , prix : 30 } ,
        { numero : 3 , label : "produit 3" , prix : 80 } ,
        { numero : 4 , label : "produit 4" , prix : 500 }
    ]
    return of(tabProduit)
        .pipe(
            mergeMap(pInTab=>pInTab) ,
            filter((p) => p.prix <= prixMaxi) ,
            map((p : Produit)=>{p.label = p.label.toUpperCase(); return p;}) ,
            toArray()
        );
}
```

4.2. Imports "umd" dans fichier js (navigateur récent)

EssaiRxjs.html

```
... <body>
    Essai Rxjs (Observable, pipe , subscribe)
    <hr/>
    <p>ouvrir la console web du navigateur</p>
    <script src="lib/rxjs.umd.min.js"></script>
    <script src="js/essaiRxjs.js"></script>
</body>...
```

avec `rxjs.umd.min.js` récupéré via <https://unpkg.com/rxjs/bundles/rxjs.umd.min.js> ou bien via une URL externe directe (CDN) `<script src="https://unpkg.com/rxjs/bundles/rxjs.umd.min.js"></script>`

NB : "The global namespace for rxjs is rxjs"

essaiRxJs.js

```
console.log('essai rxjs');
const { range , map, filter } = rxjs;
range(1, 10).pipe(
    filter(x => x ≥ 5),
    map(x => x * x)
).subscribe(x => console.log(x));
// affiche 25 , 36 , 49, 64 , 81, 100
```

```
console.log('essai via .rxjs as prefix');
rxjs.range(1, 10).pipe(
    rxjs.filter(x => x ≥ 5),
    rxjs.map(x => x * x)
).subscribe(x => console.log(x));
// affiche 25 , 36 , 49, 64 , 81, 100
```

5. Sources classiques générant des "Observables"

5.1. Données statiques (tests temporaires , cas très simples)

```
let jsObject = { p1 : "val1" , p2 : "val2" } ;
of(jsObject)...subscribe(...);

let tabObj = [ { ...} , { ... } ] ;
of(tabObj)...subscribe(...);

of(value1 , value2 , ... , valueN)...subscribe(...);
```

5.2. Données numériques : range(startValue,endValue)

```
range(1, 10).subscribe(x => console.log(x)) ;
1
2
...
10
```

5.3. source périodique en tant que compteur d'occurrence

```
const obsvI1 = interval(1000 /*ms*/);
//subscriptionObjsvI1 is the result of .subscribe() call
const subscriptionObjsvI1 = obsvI1.subscribe(n =>
  { console.log(`the number of interval occurrence (starting at 0) is ${n} `);
    if(n>=5) {
      subscriptionObjsvI1.unsubscribe(); //stop if n>=5
    }
  });
});
```

5.4. source en tant qu'événement js/DOM

```
import { fromEvent } from 'rxjs';

const el = document.getElementById('my-element');

// Create an Observable that will publish mouse movements
const mouseMoves = fromEvent(el, 'mousemove');

// Subscribe to start listening for mouse-move events
const subscription = mouseMoves.subscribe((evt /*: MouseEvent */ ) => {
  // Log coords of mouse movements
  console.log(`Coords: ${evt.clientX} X ${evt.clientY}`);

  // When the mouse is over the upper-left of the screen,
  // unsubscribe to stop listening for mouse movements
});
```

```
if (evt.clientX < 40 && evt.clientY < 40) {  
    subscription.unsubscribe();  
}  
});
```

5.5. source en tant que réponse http (sans angular HttpClient)

```
import { ajax } from 'rxjs/ajax';  
  
// Create an Observable that will create an AJAX request  
const apiData = ajax('/api/data');  
// Subscribe to create the request  
apiData.subscribe(res => console.log(res.status, res.response));
```

5.6. source en tant que données reçues sur canal web-socket

```
const subject = webSocket('ws://localhost:8081');  
...  
  
// https://rxjs.dev/api/webSocket/webSocket pour approfondir
```

6. Principaux opérateurs (à enchaîner via pipe)

Rappel (syntaxe générale des enchaînements) :

Source_configurée_et_initialisée

```
.pipe(
  callback_fonctionnelle_1 ,
  callback_fonctionnelle_2 ,
  ....
) .subscribe({
  next : callback_success ,
  error : callback_error
})
```

avec plein de variantes possibles

Principaux opérateurs :

map	Transformations quelconques (calculs , majuscules , tri ,)
flatMap , mergeMap	Tableau réactif devient flux réactif des éléments du tableau
toArray	flux réactif d' éléments devient Tableau réactif
filter	Filtrages (selon comparaison,)
tap	Traitements supplémentaires en parallèle sur les données du flux réactif mais sans changer la valeur de retour

6.1. map() : transformations

En sortie , résultat (retourné via return) d'une modification effectuée sur l'entrée .

Exemple 1:

```
const obsNums = of(1, 2, 3 ,4 ,5);
const squareValuesFunctionOnObs = map((val) => val * val);
const obsSquaredNums = squareValuesFunctionOnObs(obsNums);
obsSquaredNums.subscribe(x => console.log(x));
// affiche 1 4 9 16 25
```

Exemple 2:

```
const obsStrs = of("un" , "deux" , "trois");
obsStrs.pipe(
  map( (s) => s.toUpperCase() )
)
.subscribe(s => console.log(s));
// affiche UN DEUX TROIS
```

6.2. mergeMap() et toArray()

De façon à itérer une séquence d'opérateurs sur chaque élément d'un tableau tout en évitant une imbrication complexe et peu lisible de ce type :

```
observableSurUnTableau
.pipe(
  map((tableau)=>{
    return tableau.map(
      (itemInTab)=>{itemInTab.label = itemInTab.label.toUpperCase();
      return itemInTab;}
    );
  });
)
```

on pourra placer une séquence d'opérateurs qui agiront sur chacun des éléments du tableau entre mergeMap() et toArray() :

```
observableSurUnTableau
.pipe(
  mergeMap(itemInTab=>itemInTab),
  filter((itemInTab) => itemInTab.prix <= 300 ),
  map(( itemInTab )=>{ ... }),
  toArray()
).subscribe( (tableau) => { ... } );
```

6.3. filter()

```
const obsVals = of(12 , -15 , 30 , -8 , 40);
obsVals.pipe(
  filter( (v) => v >= 0)
)
.subscribe(v => console.log(v));
//affiche 12 30 et 40
```

```
range(1,10).pipe(
  filter( (v) => v % 2 === 0 )
)
.subscribe(v => console.log(v +" est une valeur paire"));
```

6.4. map with sort on array

```
const obsTab = of([
  {numero:1,label:'produit1',prix:40.0},
  {numero:2,label:'produit2',prix:30.0},           {numero:3,label:'produit3',prix:35.0},
  {numero:4,label:'produit4',prix:15.0},           {numero:5,label:'produit5',prix:35.0}
]);
obsTab.pipe(
  map( (tab) => tab.sort( (p1,p2) => (p1.prix - p2.prix) ) )
  // map( (tab) => tab.sort( (p1,p2) => p1?p1.label.localeCompare(p2.label):0 ) )
)
.subscribe(t => console.log( JSON.stringify(t) ));
```

7. Hot Observable

Type d'observable	comportement
cold (by default) observable	Flux pas partagé / pas synchronisé . Une instance du flux observable pour chaque consommateur
hot observable	Flux partagé et synchronisé . Si plusieurs consommateurs (en mode "multicast"), ceux-ci reçoivent alors les mêmes données . Un "unsubscribe" commun/global sera automatiquement différé (en attendant la fin de la dernière consommation)

Exemple :

type of observable:

viewer1:	viewer2 (delay=2s):	viewer3 (delay=4s):
3	1	

type of observable:

viewer1:	viewer2 (delay=2s):	viewer3 (delay=4s):
3	3	

fonction qui enregistre un consommateur "viewer 1_ou_2_ou_3" vis à vis d'un observable \$obs en effectuant cet enregistrement en décalé dans le temps : *delay* ms

et qui programme un unsubscribe *subscription timeout* ms après le début de l'enregistrement :

```
subscription_for_viewer=(viewerId,obs$,delay,subscription_timeout)=>{
    setTimeout(_ =>{
        const subscription = obs$.subscribe(
            (v)=> {
                let msg = viewerId+": "+v;
                console.log(msg);
                document.getElementById(viewerId).innerHTML=v;
            }
        );
        setTimeout(_ => {
            subscription.unsubscribe();
        }, subscription_timeout);
    }, delay);
}
```

```

document.getElementById("btnStart").addEventListener("click" , ()=>{
    ...

const coldObs$ = new rxjs.Observable((observer) => {
    //initialize producer:
    let currentNumber = 1;
    const producer = setInterval( _ => {
        document.getElementById("spanLastProduced").innerHTML=currentNumber;
        observer.next(currentNumber++); //envoi la valeur (++) de currentNumber toutes les 1s
    }, 1000);

    //returning function that will be called when unsubscribe:
    return ()=> {
        console.log("end of subscription");
        clearInterval(producer);
    }
});

//build hot observable from cold :
let hotObs$ = coldObs$.pipe(rxjs.share());

let coldOrHot =document.getElementById("selectColdOrHot").value;
let obs$=(coldOrHot=="hot")?hotObs$:coldObs$;

subscription_for_viewer("viewer1",obs$,0,7000);//sans délai
subscription_for_viewer("viewer2",obs$,2000,7000); //démarrage différé dans 2s
subscription_for_viewer("viewer3",obs$,4000,7000);//démarrage différé dans 4s

});

```

Comportement :

- en version "cold" , plusieurs instances du flux , lorsque le viewer1 reçoit 3 , le viewer 2 (au démarrage décalé) reçoit 1
- en version "hot" , une seule instance du flux , lorsque le viewer1 reçoit 3 , le viewer 2 reçoit également 3

8. Observable d'ordre 2

"observables of observables" = "High order observables"

```
outerObservable.pipe(
  concatMap_or_xyzMap(innerObservable),
  ....)
.subscribe(... données globalement récupérées ...);
```

Variantes	Comportements
concatMap (innerObs)	concaténation après un déclenchement d'une boucle sur le outer_observable et bufferisation . Ordre contrôlé au sein du résultat global (groupBy sur outer observable)
mergeMap (innerObs)	fusion avec sous déclenchements en parallèles au fur et à mesure (multiple sous-subscribes simultanés) Ordre incontrôlé au sein du résultat global
switchMap (innerObs)	Sous déclenchements séquentiels (subscribe , unsubscribe/cancel_new, subscribe, unsubscribe/cancel_new , ...) Interruption du traitement du sous niveau courant dès qu'arrive un nouvel élément du niveau principal
exhaustMap (innerObs)	comme switchMap mais ignore nouvelle occurrence du niveau principal si le traitement du sous niveau courant n'est pas encore terminé

Exemple :

```
let msg = "";
let wsUrlArray = [ 'https://catfact.ninja/fact' , 'https://geo.api.gouv.fr/communes?codePostal=78000',
  'https://geo.api.gouv.fr/communes?codePostal=76000' , 'https://geo.api.gouv.fr/communes?codePostal=80000'
];
let mode =document.getElementById("selectMode").value;
let mapFunction;
switch(mode){
  case "mergeMap" : mapFunction=rxjs.mergeMap; break;
  case "switchMap" : mapFunction=rxjs.switchMap; break;
  case "exhaustMap" : mapFunction=rxjs.exhaustMap; break;
  case "concatMap" :
    default: mapFunction=rxjs.concatMap;
}

rxjs.of(... wsUrlArray)
.pipe(
  //in this pipe : url as Observable<string> : first level of observable things :outer observable
  mapFunction(
    url => rxjs.ajax.ajax(url)
    //http.get(url) or ajax(url) return a second level of observables things (http responses) : inner observable
  ),
  rxjs.map((rxjsAjaxResp)=> "status=" + rxjsAjaxResp.status
    + " jsonData=" + JSON.stringify(rxjsAjaxResp.response))
)
.subscribe({
  next : x => { msg += (x+'<br/>'); } ,
  error : e => console.log(e) ,
  complete : () => dualDisplay("spanExHighOrderObs",msg)
});
```

Au sein de cet exemple , le premier niveau d'observable (produit via of(...)) est un tableau d'url .

Le sous niveau d'observable (construit via ajax(...)) est une réponse à une api rest (ici en mode get

par défaut).

Comportements de l'exemple :

mode	résultat	comportement
concatMap	<pre>status=200 jsonData={"fact":"A cat can travel at a top speed of approximately 31 mph (49 km) over a short distance.", "length":86} status=200 jsonData=[{"nom":"Versailles","code":"78646","codeDepartement":"78","siren":"217806462","codeEpci":"247800584","codeRegion":"11","codesPostaux":["78000"],"population":83587}] status=200 jsonData=[{"nom":"Rouen","code":"76540","codeDepartement":"76","siren":"217605401","codeEpci":"200023414","codeRegion":"28","codesPostaux":["76000","76100"],"population":114083}] status=200 jsonData=[{"nom":"Amiens","code":"80021","codeDepartement":"80","siren":"218000198","codeEpci":"248000531","codeRegion":"32","codesPostaux":["80000","80080","80090"],"population":133625}]</pre>	Même ordre que déclenchement
mergeMap	<pre>status=200 jsonData=[{"nom":"Versailles","code":"78646","codeDepartement":"78","siren":"217806462","codeEpci":"247800584","codeRegion":"11","codesPostaux":["78000"],"population":83587}] status=200 jsonData=[{"nom":"Rouen","code":"76540","codeDepartement":"76","siren":"217605401","codeEpci":"200023414","codeRegion":"28","codesPostaux":["76000","76100"],"population":114083}] status=200 jsonData=[{"nom":"Amiens","code":"80021","codeDepartement":"80","siren":"218000198","codeEpci":"248000531","codeRegion":"32","codesPostaux":["80000","80080","80090"],"population":133625}] status=200 jsonData={"fact":"Relative to its body size, the clouded leopard has the biggest canines of all animals' canines. Its dagger-like teeth can be as long as 1.8 inches (4.5 cm).", "length":156}</pre>	ordre différent que celui du déclenchement car api geo plus rapide que catfact
switchMap	<pre>status=200 jsonData=[{"nom":"Amiens","code":"80021","codeDepartement":"80","siren":"218000198","codeEpci":"248000531","codeRegion":"32","codesPostaux":["80000","80080","80090"],"population":133625}]</pre>	Premier(s) pas rapide(s) interrompu(s) et récupération du/des dernier(s)
exhaustMap	<pre>status=200 jsonData={"fact":"During the time of the Spanish Inquisition, Pope Innocent VIII condemned cats as evil and thousands of cats were burned. Unfortunately, the widespread killing of cats led to an explosion of the rat population, which exacerbated the effects of the Black Death.", "length":259}</pre>	Attente du premier (pas rapide) et autres appels pas gérés

9. Passerelles entre "Observable" et "Promise"

9.1. Source "Observable" initiée depuis une "Promise" :

```
import { from } from 'rxjs';

// Create an Observable out of a promise :
const observableResponse = from(fetch('/api/endpoint'));

observableResponse.subscribe({
  next(response) { console.log(response); },
  error(err) { console.error('Error: ' + err); },
  complete() { console.log('Completed'); }
});
```

9.2. Convertir un "Observable" en "Promise"

observableXy.toPromise().then(...).catch(...)
is now deprecated.

Use `lastValueFrom()` or `firstValueFrom()` instead .

Exemple :

```
async onConvertirV2(){
  try{
    this.montantConverti = await firstValueFrom(this._deviseService.convertir$(this.montant,
      this.codeDeviseSource,
      this.codeDeviseCible));
  }catch(err){
    console.log(err);
  }
}
```

NB: `firstValueFrom()` déclenchera automatiquement un "unsubscribe" dès que possible .

10. Optimisations et précautions

```
myObservable.pipe(take(1)).subscribe(x => { console.log(x);});
```

Seul ou bien en fin de pipe() , l'opérateur **take(1)** permet un déclenchement automatique d'un "**unsubscribe**" pour éviter au mieux les risques de fuite de mémoire .

Par défaut, **toSignal()** se désabonne automatiquement de l'Observable lorsque le composant ou le service qui le crée est détruit .

Attention : pas de unsubscribe automatique avec seulement .pipe(shareReplay(1)) (pour hot observable partagé) , besoin du pipe "async" ou autre .

NB : le pipe **async** d'angular **déclenche également automatiquement un "unsubscribe"** lorsque le composant est détruit ce qui en fait une bonne pratique très recommandée dans la plupart des cas :

```
@Component({ ... imports: [AsyncPipe , ...], ...})
export class XxxComponent {
  montantConvertiObs$!: Observable<number>;
  ...
  onConvertir() {
    this.montantConvertiObs$ = this._deviseService.convertir$(this.montant,
      this.codeDeviseSource, this.codeDeviseCible) ;
  }
}
```

```
<input type="button" (click)="onConvertir()" value="convertir" /> <br/>
montantConverti = {{montantConvertiObs$ | async}} <br/>
```

XI - Appels HTTP (vers api REST)

1. Angular et dialogues HTTP/REST

1.1. Contexte des appels HTTP/REST et CORS

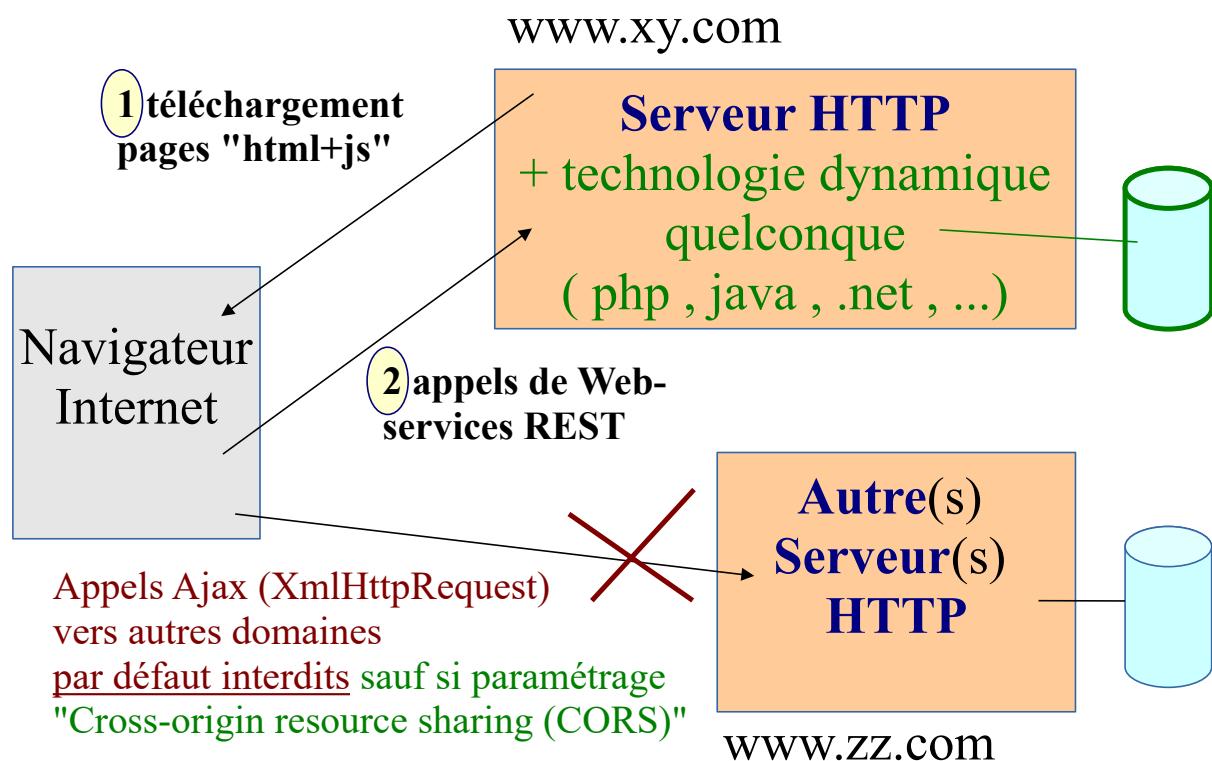
Une application Angular s'exécute au sein d'un navigateur Web . Lorsque celle-ci doit effectuer un appel HTTP/ajax , elle est soumise aux restrictions habituelles du navigateur :

- appels en file:/ pas toujours autorisés (ok avec firefox , par défaut refusés avec chrome)
- les URLs des WS-REST appelés doivent commencer par le même nom de domaine que celui qui a servi à télécharger index.html (ex : <http://localhost:4200> ou <http://www.xyz.com> ou ...). Dans le cas contraire des autorisations "CORS" sont nécessaires.

Rappel important : si les appels HTTP/ajax sont effectués vers un autre nom de domaine il faudra prévoir des autorisations "**CORS**" du coté des web-services REST coté serveur (ex : nodeJs/Express ou Java/JEE/JaxRS ou SpringMvc) .

D'autre part, beaucoup de web-services REST nécessitent des paramètres de sécurité pour pouvoir être invoqués (ex : jetons/tokens, ...) . Une adaptation au cas par cas sera souvent à prévoir.

Cadre des appels "html/js/ajax" vers services REST



```
// Exemple : CORS enabled with express/node-js :
app.use(function(req, res, next) {
  res.header("Access-Control-Allow-Origin", "*"); // "*" ou "xy.com , ... "
  res.header("Access-Control-Allow-Methods", "POST, GET, PUT, DELETE,
OPTIONS"); // default: GET, ...
  res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type,
Accept, Authorization");
  next();
});
```

1.2. Simulation d'invocation de WS REST via of() de rxjs

of(...) permet de retourner immédiatement un jeu de données (en tant que simulation d'une réponse à un appel HTTP en mode get) .

D'autres solutions sont disponibles pour simuler des appels HTTP ... (voir chapitre/annexe sur test unitaire)

1.3. Configuration d'un reverse-proxy http en mode dev

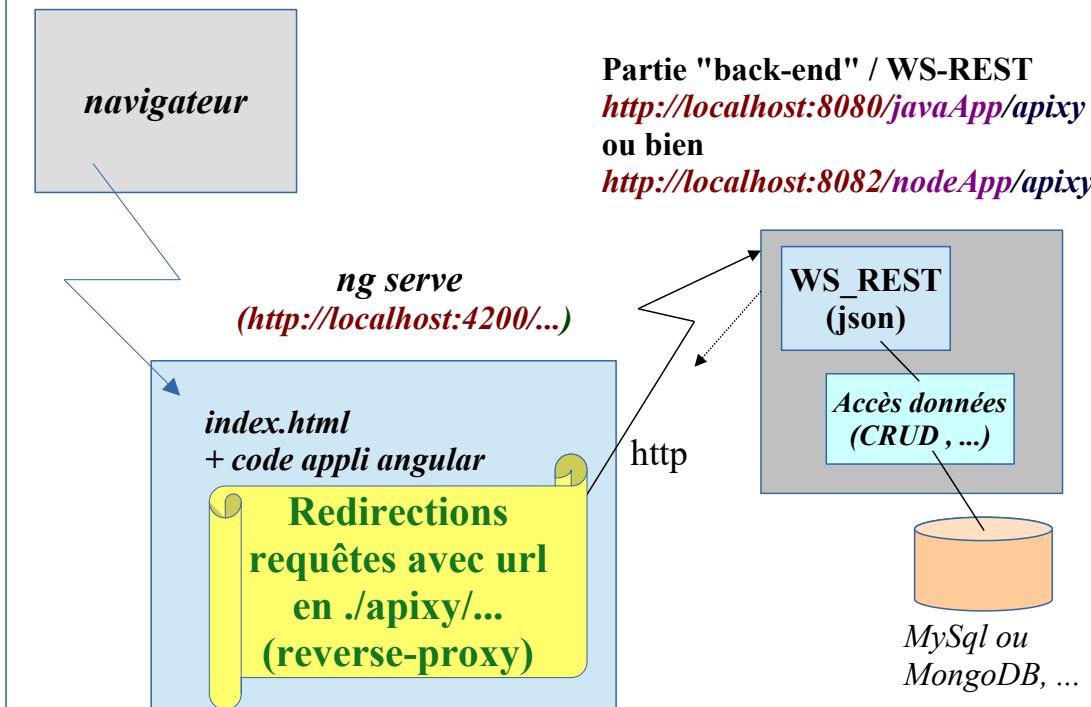
Il serait possible (durant la phase de développement) de :

- utiliser des URLs absolues (ex : http://localhost:8282/apiXy/xy) pour appeler des Web services "REST" (java ou php ou nodeJs/express ou ...) via angular et ajax
- paramétrier des autorisations "CORS" du coté serveur (code java ou js/express ou ...)

De façon à éviter toutes ces choses (aujourd'hui déconseillées) , on peut :

- toujours utiliser des URLs relatives (ex : apiXy/xy) pour appeler des Web services "REST" (java ou php ou nodeJs/express ou ...) via angular et ajax dès la phase de développement
- ne pas systématiquement avoir besoin de configurer des autorisations "CORS" du coté serveur
- configurer un proxy http au sein d'angular-CLI (uniquement exploitable avec ng serve) .

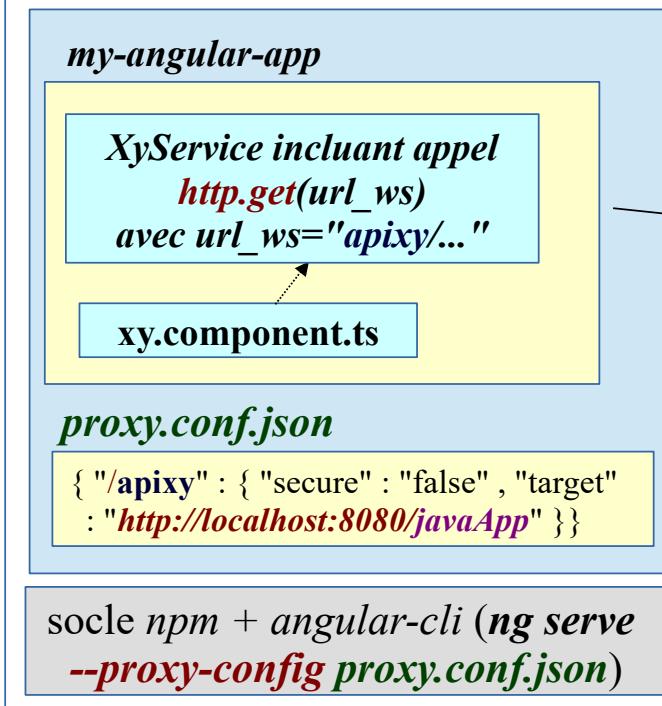
ng serve avec reverse proxy



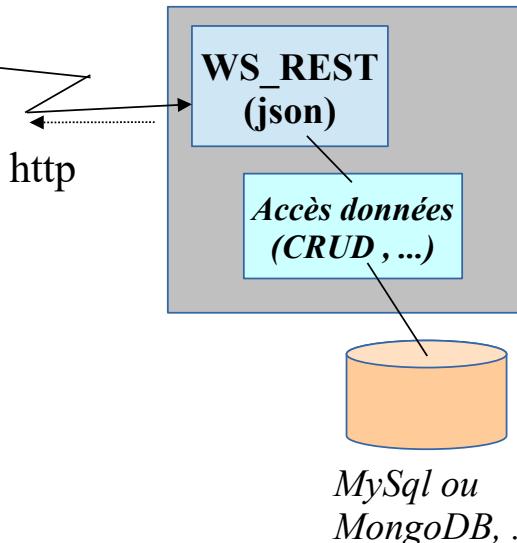
- <http://localhost:4200/index.html> fait que le serveur lancé par `ng serve` retourne directement le code de l'application angular .
- <http://localhost:4200/apixy/xy> fait que le serveur lancé par `ng serve` (et bien configuré via l'option `--proxy-config proxy.conf.json`) effectue une redirection de l'appel HTTP vers le backend en arrière plan angular .
- Et comme le navigateur envoie toutes les requêtes au même endroit (sans être au courant de la redirection effectuée en arrière plan), il n'y a pas de besoin d'autorisations CORS.

Environnement de développement Angular (v2,v4,...)

partie cliente "Angular"
([http://localhost:4200/...](http://localhost:4200/))



Partie "back-end" / WS-REST
<http://localhost:8080/javaApp/apixy>
ou bien
<http://localhost:8082/nodeApp/apixy>



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

avec **proxy.conf.json**

```
{
  "/apixy": {
    "secure": false,
    "changeOrigin": true,
    "target": "http://localhost:8080/javaApp"
  }
}
```

pour que les requêtes d'urls relatives **apixy/..** soient redirigées vers une application java/tomcat.
ou bien

avec **proxy.conf.json**

```
{
  "/apixy": {
    "secure": false,
    "changeOrigin": true,
    "target": "http://localhost:8282/nodeApp"
  }
}
```

pour que les requêtes d'urls relatives **apixy/..** soient redirigées vers une appli. nodeJs/express .

Astuce :

En plaçant cette option (pas très loin de la ligne 75) du fichier **angular.json** on pourra démarrer **ng serve** facilement (*sans repreciser l'option --proxy-config à chaque redémarrage*).

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

2. Api HttpClient (depuis Angular 4.3)

L'api **HttpClient** de la partie `@angular/common/http` de **Angular >= 4.3** est une **version améliorée** (avec meilleur typage , généréricité , code d'utilisation plus compacte, ...) de l'ancien service technique **Http** disponible depuis Angular 2 .

2.1. Configuration du module HttpClientModule avant v18

Pour utiliser HttpClient, il faut commencer par importer le module technique "**HttpClientModule**" dans un module fonctionnel de l'application (ex : `app.module.ts`) :

```
import { HttpClientModule } from '@angular/common/http';
//à la place de import { HttpModule } from "@angular/http";
...

@NgModule({
  declarations: [ AppComponent , ... ],
  imports: [ BrowserModule,    HttpClientModule ],
  providers: [ ... ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

2.2. Variante de configuration depuis angular 17 et 18

Depuis angular 18 , l'utilitaire "ng new" génère une application qui peut **potentiellement être en mode "standalone"** avec ou pas l'optimisation "SSR" lors d'un futur démarrage en production.

Pour s'adapter à ces nouvelles configurations possibles, le module **HttpClientModule** est maintenant considéré comme obsolète (deprecated) depuis angular 18.

Pour assurer une configuration équivalente dans un module d'angular 18+ , il faut maintenant paramétrier les choses de la façon suivante :

`app.module.ts`

```
...
import { provideHttpClient } from '@angular/common/http';
@NgModule({
  declarations: [ AppComponent , ... ],
  imports: [ BrowserModule, FormsModule , ... ],
  providers: [ ... , provideHttpClient() ],
  bootstrap: [AppComponent]
})
```

ou bien

```
export const appConfig: ApplicationConfig = {
  providers: [ ... , provideRouter(routes), provideClientHydration(),provideHttpClient()]
};
```

si application en mode "standalone" sans module

2.3. Utilisation du service technique HttpClient dans un service :

(ou éventuellement directement depuis un composant) :

```
import { HttpClient } from '@angular/common/http';
...
constructor(private http: HttpClient){ } //injection de dépendance
...
//utilisation rare (non typée) / ici directement depuis un composant :
ngOnInit(): void {
    this.http.get('https://api.github.com/users/didier-tp')
        .subscribe(data => { console.log(data); });
}
```

==> En mode "HttpClient" l'appel à http.get() retourne directement les données de la réponse Http au format Observable<any> et non pas un Observable<Response> brute à analyser/décortiquer .

Récupération fine des causes d'erreurs via seconde partie facultative de subscribe :

```
src/app/common/util/util.ts
import { HttpErrorResponse } from "@angular/common/http";

export function messageFromError(err: HttpErrorResponse, myMsg /*: string*/ = ""){
    let message="";
    if (err.error instanceof Error) {
        console.log("Client-side error occurred." + JSON.stringify(err));
        message = myMsg;
    } else {
        console.log("Server-side error occurred : " + JSON.stringify(err));
        let detailErrMsg = (err.error && err.error.message)?":"+err.error.message:"";
        message = myMsg + " (status="+ err.status + ":" + err.statusText + ") " + detailErrMsg ;
    }
    return message;
}
```

```
import { messageFromError } from '../common/util/util';

this.http.get('https://api.github.com/users/didier-tp')
    .subscribe({
        next : data => { console.log(data); },
        error : (err: HttpErrorResponse) => { this.message = messageFromError(err,"echec get"); }
    });
```

2.4. Appels typés :

Plus rigoureusement , un appel à **http.get<DataClass>(url , ...)** retourne des données au format **Observable<DataClass>** plutôt qu'au format **Observable<any>**

Exemple :

```
public getTabInscriptions$(): Observable<Client[]> {
  const inscriptionUrl : string = this._inscriptionUrlBase ;
  console.log( "inscriptionUrl = " + inscriptionUrl);
  return this.http.get<Client[]>(inscriptionUrl );
}
```

Idem pour les appels en mode post,put,delete,... .

Exemple :

```
this.http.post<TypeRetour>(url , dataToSend ) ;
```

retournant **Observable<TypeRetour>** avec assez souvent un identifiant auto-incrémenté.

2.5. Déclenchement du côté composant

```

@Component({...})
export class XyzComponent implements OnInit {
  tabClients : Client[] = [];//vu et affiché du coté .html
  clientTemp : Client = new Client(); //[ngModel] = "clientTemp.name", ....
  message /*: string*/ = "";

  constructor(private xyzService : XyzService) {
    //injection de dépendance via constructeur
  }

  ngOnInit(): void {
    this.xyzService.getTabInscriptions$()
      .subscribe({ next: (tabCli)=>{ /*this.tabClients = tabCli;*/  

        this.initAfterFetch(tabCli) },  

        error: (err)=>{ this.message = messageFromError(err,  

          "echec rechercherClients(get)"); }  

      });
  }

  onUpdate(){
    this.xyzService.putInscription$(this.clientTemp)
      .subscribe(  

        { next: (updatedCustomer)=>{ this.message="client bien mis à jour";  

          this.updateClientSide(updatedCustomer); } ,  

        error: (err)=>{ this.message = messageFromError(err,"echec update (put)"); }  

      );
  }
}

```

2.6. Mode post avec header http personnalisé :

```

import { HttpClient, HttpHeaders } from "@angular/common/http";
import { Observable } from "rxjs";
import { Client } from "./client";

@Injectable({
  providedIn: 'root'
})
export class InscriptionService {
  constructor(private _http : HttpClient) { }

  private _headers = new HttpHeaders({'Content-Type': 'application/json'});

  public postInscriptions$(cli : Client):Observable<Client> {
    let inscriptionUrl : string = "xy-api/inscription" ; //avec ng serve --proxy-config proxy.conf.json
    return this._http.post<Client>(inscriptionUrl ,
      cli,
      {headers: this._headers} );
  } ...
}

```

Etant donné que '**application/json**' est le '**Content-Type**' par défaut , le code précédent peut se simplifier en le suivant :

```
export class IncriptionService {
  constructor(private _http : HttpClient) { }

  public postInscriptions$(cli : Client):Observable<Client> {
    let inscriptionUrl : string = "xy-api/inscription" ; //avec ng serve --proxy-config proxy.conf.json
    return this._http.post<Client>(inscriptionUrl , cli );
  }
}
```

Autre exemple (avec post-traitement annexe via .pipe() et tap from 'rxjs/operators') :

```
postAuth$(auth : AuthRequest):Observable<AuthResponse>{
  return this._http.post<AuthResponse>(this._authBaseUrl ,auth )
  .pipe(
    //tap( other async task without transforming result)
    tap( authResponse )=> { this.storeAuthResponseAndToken(authResponse); }
  );
}

private storeAuthResponseAndToken(authResponse:AuthResponse){
  ... = authResponse.token ; sessionStorage.setItem('authToken',...);
}
```

Mode "put" (variante du mode "post") :

```
putInscription$(cli : Client):Observable<Client>{
  return this.http.put<Client>(this.basePrivateUrl ,cli);
}
```

Rappel :

- les conventions "api REST" recommande le mode "PUT" pour les mises à jour (update) d'entités/ressources existantes.

2.7. Exemple de suppression (delete):

```
public deleteDeviseServerSide$(deviseCode):Observable<any>{
  let deleteUrl : string = this.basePrivateUrl + "/" + deviseCode ;
  console.log("deleteUrl= " + deleteUrl );
  return this.http.delete(deleteUrl );
}
```

2.8. intercepteurs :

Disponible à partir de la version 4.3 , les intercepteurs sont surtout utiles pour renvoyer automatiquement un jeton d'authentification au sein des requêtes envoyées au serveur.

```
cd src/app/common/interceptor
```

```
ng g interceptor MyAuth
```

src/app/common/interceptor/my-auth.interceptor.ts (avec code complété)

```
import { Injectable } from '@angular/core';
import { HttpEvent, HttpInterceptor, HttpHandler, HttpRequest } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable()
export class MyAuthInterceptor implements HttpInterceptor {
  intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {
    const token = sessionStorage.getItem('authToken');
    const authReq = request.clone({
      headers: request.headers.set('Authorization', 'Bearer ' + token)
    });
    return next.handle(authReq);
  }
}
```

avec la déclaration suivante dans app.module.ts :

```
...
import { HTTP_INTERCEPTORS } from '@angular/common/http';

import { AppComponent } from './app.component';
import { MyAuthInterceptor } from './my-auth.interceptor';

@NgModule({
  declarations: [ AppComponent ],
  imports: [
    BrowserModule,
    HttpClientModule
  ],
  providers: [
    {
      provide: HTTP_INTERCEPTORS,
      useClass: MyAuthInterceptor,
      multi: true
    },
    bootstrap: [AppComponent]
  ]
})
export class AppModule {}
```

NB : pour angular 18+ , **provideHttpClient(withInterceptorsFromDi())** dans providers:[]

NB : il existe des intercepteurs sous forme de fonction fléchées. Ceux ci sont plutôt prévus pour le

mode standalone (sans @NgModule).

```
import { inject, PLATFORM_ID } from "@angular/core";
import { isPlatformBrowser } from "@angular/common";
import { Injectable } from '@angular/core';
import { HttpEvent, HttpInterceptor, HttpHandler, HttpRequest } from '@angular/common/http';
import { Observable } from 'rxjs';

export const myAuthInterceptor: HttpInterceptorFn = (req, next) => {
  const platform = inject(PLATFORM_ID);
  let token : string | null = "";
  if(isPlatformBrowser(platform)) {
    token = sessionStorage.getItem('access_token');
  }
  const authReq = req.clone({
    headers: req.headers.set('Authorization', 'Bearer ' + token)
  });
  return next(authReq);
};
```

et

enregistrement via :

```
providers : [ ... , provideHttpClient( withInterceptors([myAuthInterceptor]) ) ]
```

2.9. Accès possible à toute la réponse Http pour cas pointus

```
getXxxResponse$(): Observable<HttpResponse<Xxx>> {
  const httpOptions = {
    /* headers: new HttpHeaders({ 'Content-Type': 'application/json' }), */
    observe: 'response'
  };
  return this.http.get<Xxx>( this.xxxUrl, httpOptions);
}
```

...

2.10. Attente du téléchargement avec Resolver

Pour éviter une navigation immédiate suivie d'un affichage progressif des données au fur et à mesure de leur téléchargement on peut utiliser un "Resolver" qui va différer la navigation pour que celle ci ne soit déclenchée que lorsque les données seront prêtes .

Exemple (à créer via **ng g resolver ...**) :

```
import { inject } from '@angular/core';
import { Observable } from 'rxjs';
import { Devise } from './data/devise';
import { DeviseService } from './service/devise.service';
import { ResolveFn } from '@angular/router';

//in odler angular version: DevisesResolver implements Resolve<Devise[]>
//as a service with .revolve() returning Observable<Devise[]>

export const devisesResolver: ResolveFn<Devise[]> =
  (route, state): Observable<Devise[]> => {
  const deviseService = inject(DeviseService);
  console.log("prefetch Devises via devisesResolver")
  return deviseService.getAllDevises$();
  //return deviseService.getXyz(route.params['xyz']);
};
```

Utilisation au niveau d'une route :

```
{ path: 'xyz', component: XyzComponent ,resolve : { devises : devisesResolver} } , ...
```

Récupération des données au bout de la route suivie :

```
import { ActivatedRoute, Data } from '@angular/router';
...
export class XyzComponent {
constructor(private route: ActivatedRoute) {
  console.log("XyzComponent");
  this.route.data.subscribe(
    (data: Data) =>{
      let tabDevises = data['devises'];
      // ...
    });
}}
```

Astuce pour visualiser temporairement en phase de développement l'aspect différé de la navigation:

```
return this._http.get<Devise[]>(url).pipe(
  delay(800) // temp 800ms wait (to see Resolver effect)
);
```

XII - Routing angular (compléments importants)

1. Sous niveau de routage (children)

Une application angular peut comporter plusieurs niveaux de composants et sous composants.

Un <router-outlet> dans un composant principal permet par exemple de switcher de sous composants Xxx , Yyy , Zzz et un de ces sous composants (par exemple Yyy) peut à son tour comporter une balise <router-outlet> pour switcher de sous-sous-composants (Aa ou Bb).

Exemple d'url avec sous niveaux :

xxx/

yyy/**aa**

yyy/**bb**

zzz/

Exemple de configuration de routes avec sous niveau:

```
...
const routes: Routes = [
  { path: 'welcome', component: WelcomeComponent },
  { path: '', redirectTo: '/welcome', pathMatch: 'full' },
  { path: 'xxx', component: XxxComponent },
  { path: 'yyy', component: YyyComponent,
    children: [
      { path: 'aa', component: AaComponent },
      { path: 'bb', component: BbComponent },
      { path: '', redirectTo: 'aa', pathMatch: 'prefix' }
    ]
  };
];
```

2. Routes paramétrées et navigation par code

2.1. Configuration de routes avec paramètres logiques

```
// dans app-routing.module.ts  (:nomParametreLogique )
const routes: Routes = [...  
  { path: 'xx/:categorie' ,  component: XxComponent }  
  { path: 'yy/:yyId' ,  component: YyComponent }  
];
```

2.2. Navigation avec paramètres et via lien hypertexte

```
<a [routerLink]="'/yy', numYy'" ...> vers yy </a>
```

et

```
<a [routerLink]="'/xx', categorieXx'" ...> vers xx </a>
```

où **numYy** et **categorieXx** sont des attributs/propriétés (avec des *valeurs variables*) de la classe du composant à l'origine de la navigation (là où sont les liens hypertextes).

Si peu de catégories on peut envisager ce type de lien hypertexte:

```
<a [routerLink]="'/xx', 'categorieA'" ...> vers xx de categorie a </a>
```

```
<a [routerLink]="'/xx', 'categorieB'" ...> vers xx de categorie b </a>
```

2.3. Déclenchement d'une navigation par code (coté .ts)

....html

```
<button (click)="onNavigateYy()"> vers yy </button>
```

.....ts

```
import {Router} from '@angular/router';  ...  
export class .....Component {  
  numYy : number = 1;  
  categorieXX :string = "categorieA";  
  constructor(private _router: Router){  
  }  
  onNavigateYy():void {  
    let link = ['/yy', this.numYy]; //ou link = ['/zz'] ; si pas de paramètre  
    this._router.navigate( link );  
    // ou bien this._router.navigateByUrl('/yy/${this.numYy}') ; //avec quote inverse `...` !!!  
  }  
}
```

2.4. Récupération du paramètre accompagnant la navigation :

On s'intéressera dans ce paragraphe à tenir compte de la valeur d'un paramètre passé au bout de route activée .

Remarque très importante (pour comprendre et ne pas devenir fou) :

- (Cas "a") Suite à la série de navigation suivante :

yyy , xxx , yyy , des composants des classes Yyy , Xxx et Yxx seront ré-instanciés à chaque changement d'URL (et l'état des variables d'instance sera perdu) .

- (Cas "b") Suite à la série de navigation suivante :

detail_produit/1 ,detail_produit/2 , detail_produit/3 (où seule change la valeur du paramètre en fin d'url) , l'instance de la classe DetailProduit est conservée et la callback enregistrée (via subscribe) sur this._route.params est automatiquement ré-appelée pour prendre en compte le changement de numéro de produit à détailler .

NB:

Dans le cas "a" , où les instances ne sont pas conservées , on peut coder simplement l'unique récupération d'un paramètre en fin de route activée via la notion de snapshot (valeurs à l'instant t):

Exemple :

```
class MyComponent {
  constructor(route: ActivatedRoute) {
    const yyId: number = Number(route.snapshot.params['yyId']);
    // NB le nom logique du paramètre (ici yyId ) doit correspondre à celui
    // de la route { path: 'yy:yyId' , component: YyComponent } configurée dans
    //app-routing.module.ts
    ...
  }
}
```

Dans le cas "b" , on a besoin d'enregistrer une callback potentiellement réappelée plusieurs fois (à chaque changement de valeur du paramètre) :

```
import {Component , OnInit} from '@angular/core';
import { ActivatedRoute, Params} from '@angular/router';
import { Location }  from '@angular/common';
...
export class YyComponent implements OnInit{
  yId: number = 0;
  y : Yy | undefined; message : string = "...";
  constructor(private _yyService : YyService,
  private _route: ActivatedRoute,
  private _location: Location){
  }
}
```

```

ngOnInit() {
    this._route.params.subscribe(
        (params: Params) => {
            this.yId = Number(params['yyId']); //où 'yyId' est le nom du paramètre
                // dans l'expression de la route avec path: 'yy/:yyId'
                // du fichier app-routing.module.ts
            this.fetchYy();
        }
    );
}

fetchYy(){
    this._yyService.getYyObservable(this.yId).subscribe(yy=>this.y = yy ,
        error => this.message = <any>error);
}

goBack(): void { this._location.back(); /* retour arrière dans historique des navigations */ }
}

```

3. Route conditionnée par gardien

Il est possible de créer (et enregistrer dans un module) des services techniques (ex : CanActivateRouteGuard) et implémentant une interface "Can...." (ex : CanActivate) pour contrôler si une route peut ou pas être activée selon (par exemple) une authentification utilisateur effectuée ou pas .

NB : en cas de route bloquée, il n'y a pas d'automatisme de type lien hypertexte grisé ou invisible. Une telle fonctionnalité doit éventuellement/potentiellement être codée en complément .

3.1. Gardien basique (bloquant sans explication)

```

import { Injectable } from '@angular/core';
import { CanActivate,
    ActivatedRouteSnapshot,
    RouterStateSnapshot } from '@angular/router';

import { AuthService } from './auth.service';

@Injectable()
export class CanActivateRouteGuard implements CanActivate {
    constructor(private auth: AuthService) {}

    canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {

```

```
    return this.auth.isUserAuthenticated());
}
}
```

Dans la définition des routes , on pourra déclarer une liste de gardiens à utiliser :

```
...
{ path: 'dashboard',
  component: DashboardComponent,
  canActivate: [CanActivateRouteGuard]
}, ...
```

3.2. Redirection si route bloquée

Depuis Angular 7.1 , la méthode canActivate() d'un gardien peut non seulement retourner un booléen mais aussi un objet de type **UrlTree** pour effectuer une redirection vers un composant explicatif de type "*NotAuthorizedComponent*" .

Exemple :

```
@Injectable({
  providedIn: 'root'
})
export class CanActivateAdminRouteGuard implements CanActivate {
  constructor(private _authService: AuthService,
    private router: Router) {}

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean | UrlTree {
    if(this._authService.isUserAuthenticatedWithRole("admin")){
      return true;
    }
    else{
      //return false;

      //NB: since v7.1 , canActivate can return a UrlTree for redirecting (and not only blocking):
      return this.router.parseUrl('/not-authorized');
      //or this.router.createUrlTree(['/not-authorized']);
    }
  }
}
```

avec par exemple *not-authorized.component.html*

```
<h3>not-authorized</h3>
<p>login obligatoire pour accéder à la partie demandée</p>
<a routerLink="/login">login</a>
```

et *app-routing.module.ts* comportant

```
{ path : "not-authorized" , component : NotAuthorizedComponent },
```

Gardien sous forme de fonction :

//au sein des versions récentes (proches de 17) d'angular :

```
import { inject } from '@angular/core';
import { CanActivateFn, Router } from '@angular/router';

export const verifAuthGuard: CanActivateFn = (route, state) => {
  const router = inject(Router);
  let token = sessionStorage.getItem('access_token');
  if(token != "")
    return true;
  else
    //return false;
  return router.parseUrl('/ngr-login_or_not-authorized');
};
```

4. Aperçu sur le routing angular avancé

4.1. Aspects avancés du "routing angular" :

router-outlet annexe/secondaire	A un niveau donné, il peut exister d'autres <router-outlet> secondaires avec des noms. Ceci permet de changer le contenu de plusieurs <div> d'un seul coup (ex : contenu et menu latéral)
LazyLoading	Au lieu de charger dès le démarrage (dans le navigateur) tous les composants de l'application (ce qui peut quelquefois être long/lent), on peut organiser la structure du code en différents modules qui ne seront chargés (en différé) que lors d'une navigation (ex : activation d'un lien hypertexte associé au routage)

Ces divers aspects avancés sont souvent traités dans une annexes ou bien un cours "angular avancé"

4.2. Syntaxe de déclenchement d'un lazyLoading avec des modules (pas en mode standalone) :

...

```
{
  path: 'ngr-welcome', component: WelcomeComponent },
  { path: '', redirectTo: '/ngr-welcome', pathMatch: 'full' },
  { path: 'ngr-login', component: LoginComponent },
  { path: 'ngr-xxx', loadChildren: () => import('../xxx/xxx.module').then(m => m.XxxModule) },
  { path: 'ngryyy', loadChildren: () => import('../yyy/yyy.module').then(m => m.YyyModule) },
...
}
```

Simple à déclencher mais besoin d'une réflexion pour identifier des parties relativement indépendantes d'une grosse application à décomposer en modules .

Rappel d'une structure possible avec différents modules :

src/app/
core (ou "root" : module principal/noyau/racine avec singletons , home/welcome component, ...)
shared (module partagé de choses qui seront réutilisées par d'autres modules)
xxx (features_module , ex: *bases*)
yyy (features_module , ex: *devises*)

Chacun de ces modules pourra avoir la sous structure suivante (répertoires) :

components
gards
pipes
models (or data)
directives
services
... (utils, configs, ...)

4.3. Lazy loading en mode standalone (sans module)

```
//LAZY LOADING of a whole subpart (in this standaloneApp :
// ANNEXE1 "hyper_component" with childrens)
{ path: 'ngr-annexe1',
  loadChildren: () => import('./p-annexe1/p-annexe1.routes')
    .then(m => m.ANNEXE1_ROUTES) },
```

p-annexe1.routes.ts

```
...
export const ANNEXE1_ROUTES: Routes = [
  {path: '', component: Annexe1Component,
  providers: [ CalculService ],
  children: [
    { path: 'p2', component: P2Component },
    { path: 'p2b', component: P2bComponent },
    { path: '', redirectTo: 'p2', pathMatch: 'full' },
    { path: 'reservation', component: ReservationComponent }
  ]};
```

p-annexe1.routes.html

```
<p>partie_annexe1</p>
<router-outlet></router-outlet>
```

4.4. QueryParams at end of angular route url (since angular 16+)

```
RouterModule.forRoot([], {  
  bindToComponentInputs: true  
})
```

QueryParams at end of angular route url , retrieved as @Input :

```
http://localhost:4200/articles?articleId=001
```

and

```
@Input() articleId?: string;
```

in ArticleComponent .

XIII - BehaviorSubject et synchronisations

1. BehaviorSubject (exitant avant les signaux)

NB: un objet de type **BehaviorSubject**<...> doit avoir une valeur initiale dès sa construction.

C'est une chose "**Observable**" depuis plusieurs composants de l'application.

Dès que la valeur sera modifiée , tous les observateurs seront automatiquement synchronisés.

NB : Maintenant qu'il existent des signaux qui font à peu près la même chose mais de manière plus simple et légère, il est préférable d'utiliser signal() et computed() à la place de BehaviorSubject .

Il s'agit d'une implémentation RxJs/Angular du design pattern "observateur" .

BehaviorSubject<T> Observable

Composant_1

```
sxy.changeAaVal(newVal) ;
```

Composant_2_3_n

```
constructor ou ngOnInit() :  
sxy.bsDataAa.subscribe(  
(newV)=>  
this.updateFromA(newV)  
)  
  
updateFromA(newVal) {  
this.updateBfromNewA(...);  
}
```

service commun Sxy

```
private _bsDataAa :BehaviorSubject<Aa>  
= new BehaviorSubject<Aa>(initVal) ;  
  
public changeAaVal(aa){  
this._bsDataAa.next(aa) ;  
}  
  
public get bsDataAa(){  
return this._bsDataAa ;  
}
```

_bsDataAa.next(...) redéclenche partout les callbacks préalablement enregistrées via _bsDataAa.subscribe(...) .

Principe de fonctionnement :

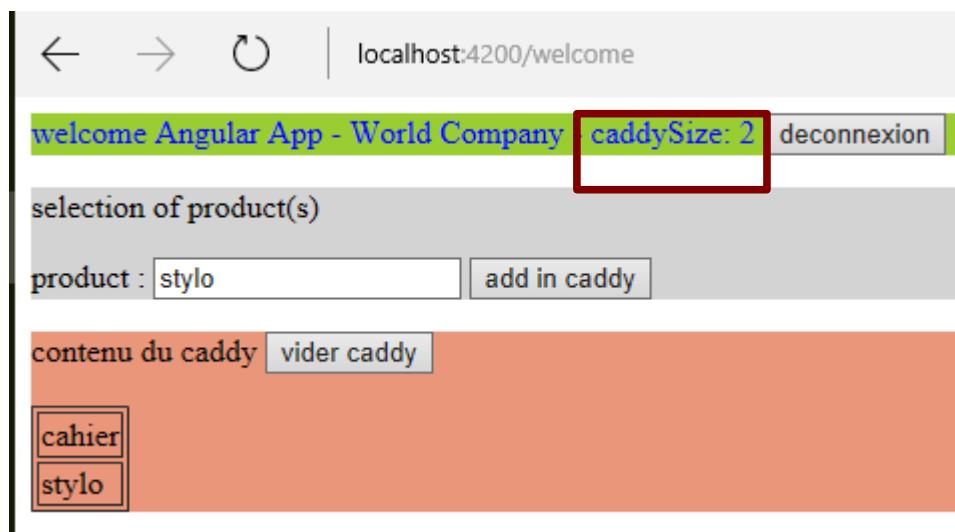
1. initialisations : chaque composant enregistre une callback (qui sera ultérieurement appelée une ou plusieurs fois) via un appel à `sxy.bsDataAa.subscribe(...)`
2. Lors de la première initialisation du BehaviorSubject et à chaque appel de `this._bsDataAa.next(aa)` ; les automatismes du framework RxJs vont redéclencher toutes les callbacks préalablement enregistrées
3. Via des callbacks spécifiquement adaptées à chacun des composants, chaque composant peut ainsi tenir compte de la nouvelle valeur de aa qui vient de changer pour réagir de

manière adéquate et bien synchronisée (soit simplement recopier/afficher la valeur de aa qui vient de changer, soit par traitement actualiser une donnée B,C ou D devant rester cohérente avec celle de aa).

1.1. Exemples d'application (BehaviorSubject)

Exemple d'application 1:

Via un service comportant un BehaviorSubject lié à un panier/caddy, on peut faire en sorte que dès qu'un composant change le contenu du caddy , d'autres composants réagissent aussitôt en affichant par exemple le nouveau nombre d'éléments de ce caddy :



Exemple d'application 2 (avec code):

seuilMax 100 nbProd : 4	list-prod	seuilMax 200 nbProd : 5																																													
<table border="1"> <thead> <tr> <th>numero</th> <th>label</th> <th>prix</th> </tr> </thead> <tbody> <tr> <td>6</td> <td>PRODUIT 620</td> <td></td> </tr> <tr> <td>2</td> <td>PRODUIT 230</td> <td></td> </tr> <tr> <td>1</td> <td>PRODUIT 150</td> <td></td> </tr> <tr> <td>3</td> <td>PRODUIT 380</td> <td></td> </tr> </tbody> </table>	numero	label	prix	6	PRODUIT 620		2	PRODUIT 230		1	PRODUIT 150		3	PRODUIT 380		<table border="1"> <thead> <tr> <th>numero</th> <th>label</th> <th>prix</th> </tr> </thead> <tbody> <tr> <td>6</td> <td>PRODUIT 620</td> <td></td> </tr> <tr> <td>2</td> <td>PRODUIT 230</td> <td></td> </tr> <tr> <td>1</td> <td>PRODUIT 150</td> <td></td> </tr> </tbody> </table>	numero	label	prix	6	PRODUIT 620		2	PRODUIT 230		1	PRODUIT 150		<table border="1"> <thead> <tr> <th>numero</th> <th>label</th> <th>prix</th> </tr> </thead> <tbody> <tr> <td>6</td> <td>PRODUIT 620</td> <td></td> </tr> <tr> <td>2</td> <td>PRODUIT 230</td> <td></td> </tr> <tr> <td>1</td> <td>PRODUIT 150</td> <td></td> </tr> <tr> <td>3</td> <td>PRODUIT 380</td> <td></td> </tr> <tr> <td>5</td> <td>PRODUIT 5120</td> <td></td> </tr> </tbody> </table>	numero	label	prix	6	PRODUIT 620		2	PRODUIT 230		1	PRODUIT 150		3	PRODUIT 380		5	PRODUIT 5120	
numero	label	prix																																													
6	PRODUIT 620																																														
2	PRODUIT 230																																														
1	PRODUIT 150																																														
3	PRODUIT 380																																														
numero	label	prix																																													
6	PRODUIT 620																																														
2	PRODUIT 230																																														
1	PRODUIT 150																																														
numero	label	prix																																													
6	PRODUIT 620																																														
2	PRODUIT 230																																														
1	PRODUIT 150																																														
3	PRODUIT 380																																														
5	PRODUIT 5120																																														

Au sein de cet exemple , un composant "seuil" permet de fixer le prix maximal de produits que l'on a envie d'acheter. Via le mécanisme observable/observateurs (BehaviorSubject) , dès que le seuil "bsSeuilMaxi" est changé via un appel à .next(nouvelleValeur) deux autres composants se synchronisent immédiatement :

- le composant listProd affiche un tableau avec la liste des produits existants dont le prix est inférieur ou égal au seuil maximum qui vient de changer.
- le composant demo calcule et affiche le nouveau nombre de produits qui respectent le nouveau prix maximal.

produit.service.ts

```

import { Injectable } from '@angular/core';
import { BehaviorSubject, Observable , of } from 'rxjs';
import { map ,toArray ,filter, mergeMap , take} from 'rxjs/operators';

export interface ProduitV2 {
  numero : number;
  label : string;
  prix : number;
}

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

  private bsSeuilMaxi = new BehaviorSubject<number>(100); //seuil (pour prixMaxi)

  public changerSeuil(nouveauSeuilMaxi : number){
    this.bsSeuilMaxi.next(nouveauSeuilMaxi);
    //l'appel à next(90_ou_80) va provoquer le rédéclenchement de toutes les callbacks
    //(dans tous les composants)
  }

  /*dans un composant angular utilisant ce service on aura
  produitService.seuilMaxiObservable.subscribe(
    (seuilQuiVientChanger => ....)
  )
  */

  public get seuilMaxiObservable() : Observable<number>{
    return this.bsSeuilMaxi;
  }

  private tabProduit = [
    { numero : 5 , label : "produit 5" , prix : 120 } ,
    { numero : 1 , label : "produit 1" , prix : 50 } ,
    { numero : 2 , label : "produit 2" , prix : 30 } ,
    { numero : 3 , label : "produit 3" , prix : 80 } ,
    { numero : 4 , label : "produit 4" , prix : 500 },
    { numero : 6 , label : "produit 6" , prix : 20 },
  ]

  public rechercherNombreProduitSimu$(prixMaxi : number) : Observable<number> {

```

```

return this.rechercherProduitSimu$(prixMaxi).pipe(
  map(tabProd=>tabProd.length)
);
}

//convention de nommage : nom de methode se terminant par $
//pour indiquer type de retour de type Observable
public rechercherProduitSimu$(prixMaxi : number) : Observable<ProduitV2[]> {

  return of(this.tabProduit)
    .pipe(
      mergeMap(pInTab=>pInTab),
      map((p : ProduitV2)=>{p.label = p.label.toUpperCase(); return p;}) ,
      filter((p) => p.prix <= prixMaxi),
      toArray(),
      map( tabP => tabP.sort( (p1,p2) => p1.prix – p2.prix)),
      take(1)
    );
}

constructor() { }
}

```

seuil.component.ts

```

...
export class SeuilComponent {
  public seuilMax=100; //à saisir

  onSeuilChange(){
    this._produitService.changerSeuil(this.seuilMax);
  }

  constructor(private _produitService : ProduitService) { }
}

```

seuil.component.html

```

<label>seuilMax</label>
<input type="number" [(ngModel)]="seuilMax" (ngModelChange)="onSeuilChange()" />

```

demo.component.ts

```

...
export class DemoComponent implements OnInit {
  nbProdPrixInferieurSeuilMaxi /*: number*/ = 0;

  actualiserNbProd(prixMaxi : number){
    this._produitService.rechercherNombreProduitSimu$(prixMaxi)
      .subscribe((nbProd) => { this.nbProdPrixInferieurSeuilMaxi = nbProd;});
  }
}

```

```
constructor(private produitService : ProduitService) {
  this.produitService.seuilMaxiObservable.subscribe(
    (nouveauSeuil)=>{ this.actualiserNbProd(nouveauSeuil);}
  );
}
}
```

demo.component.html

```
<app-seuil></app-seuil>
<p>nbProd : {{nbProdPrixInferieurSeuilMaxi}}</p>
<hr/><app-list-prod></app-list-prod>
```

list-prod.component.ts

```
...
export class ListProdComponent {
  listeProduits : ProduitV2[] = [];

  constructor(private _produitService : ProduitService) {
    this._produitService.seuilMaxiObservable
      .subscribe( seuilQuiVientChanger =>
        this.actualiserListeProduitSelonSeuilMaxi(seuilQuiVientChanger))
  }
}

actualiserListeProduitSelonSeuilMaxi(seuilMaxi : number){
  this._produitService.rechercherProduitSimu$(seuilMaxi)
    .subscribe((listP : ProduitV2[])=> { this.listeProduits = listP})
}
}
```

list-prod.component.html

```
<p>list-prod </p>
<table border="1">
<tr><th>numero</th><th>label</th><th>prix</th></tr>
<tr *ngFor="let prod of listeProduits">
  <td>{{prod.numero}}</td>  <td>{{prod.label}}</td>  <td>{{prod.prix}}</td>
</tr>
</table>
```

1.2. Eventuelle combinaison "BehaviorSubject + LocalStorage"

En cas de "refresh" déclenché (quelquefois involontairement) pas l'utilisateur (via F5 ou autre), tout le contenu en mémoire de l'application angular est réinitialisé et potentiellement perdu .

Pour ne pas perdre le contenu (caddy ou autre) dans un tel cas , le "localStorage" d'HTML5 peut éventuellement être une solution.

caddy.service.ts

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

@Injectable()
export class CaddyService {
  private _caddyContent : string[] = [];

  public bsCaddyContent : BehaviorSubject<string[]>
    = new BehaviorSubject<string[]>(this._caddyContent);

  constructor() {
    this.tryReloadCaddyContentFromLocalStorage();
    this.subscribeCaddyStoringInLocalStorage();
  }
  ...

  //Attention: localStorage = moyennement sécurisé / confidentiel
  private subscribeCaddyStoringInLocalStorage(){
    this.bsCaddyContent.subscribe( caddyContent =>
      localStorage.setItem("caddyContent",JSON.stringify(caddyContent)));
  }

  private tryReloadCaddyContentFromLocalStorage(){
    // code à améliorer (en tenant compte des exceptions):
    let caddyContentAsString = localStorage.getItem("caddyContent");
    if(caddyContentAsString){
      this._caddyContent = JSON.parse(caddyContentAsString);
      this.bsCaddyContent.next( this._caddyContent);
    }
  }
}
```

2. Autres aspects divers

2.1. Syntaxes alternatives :

bind-target = "expression" est équivalent à **[target]** = "expression"

on-target = "expression" est équivalent à **(target)**="expression"

bindon-target = "expression" est équivalent à **[(target)]="expression"**

```
<hero-detail *ngFor="#hero of heroes" [hero]="hero"></hero-detail>
```

est équivalent à

```
<template ngFor #hero [ngForOf]="heroes">
  <hero-detail [hero]="hero"></hero-detail>
</template>
```

2.2. Divers

Rappel :

The null or not hero's name is `{{nullHero?.firstName}}`

`{{a?.b?.c?.d}}` is ok if a or b or c is null or undefined

2.3. @defer (since angular 17)

Automatic lazy (deferred) loading

Exemple :

```
<p>with-defer</p>

<hr/>
<h3>Timer(3s)</h3>
@defer (on timer(3s)) {
  <div class="greenBackground">Visible after 3s</div>
}
@placeholder {
  <div class="yellowBackground">Placeholder (before 3s)</div>
}

<hr/>
<div #aPagePart>aPagePart</div>
```

```
@defer (on viewport(otherPagePart)) ; prefetch on idle {
  <p class="greenBackground">final render</p>
} @placeholder(minimum 2s) {
  <div class="yellowBackground">...placeholder.. (before final rendering with lazy loading
...)</div>
}
<hr/>
**<br/>**<br/>**<br/>**<br/>**<br/>**<br/>**<br/>**<br/>**<br/>**<br/>
<div #otherPagePart>otherPagePart</div>
```

Timer(3s)

Placeholder (before 3s)

Timer(3s)

Visible after 3s

aPagePart

...placeholder.. (before final rendering with lazy loading ...)

**
**
**
**

aPagePart

final render

**
**
**
**
**
**
**
**
**
otherPagePart

NB :

- for user experience , @defer and @placeholder are important
- for developper/debug , @loading and @error may be useful

triggers:

<div #otherPagePart>otherPagePart</div>

@defer (on viewport(otherPagePart)) when #otherPagePart is visible on viewport (visible part of html page)

@defer (on hover(otherPagePart)) when mouse/... over #otherPagePart

@defer (on interaction(aControlOfTheTemplate)) when user interact with aControlOfTheTemplate

@defer (when aBooleanExpression) when aBooleanExpression is true (after changing by code)

XIV - Authentification au sein d' Angular

1. Sécurisation d'une application "angular"

1.1. Quelques considérations générales sur la sécurité

Même transformé de ".ts" en "js" , une application Angular n'est pas véritablement compilé , son code source est accessible et éventuellement sujet à interception / modification .

--> jamais d'éléments confidentiels dans le code de l'application (pas de "salt " , pas de "default_password" , ...)

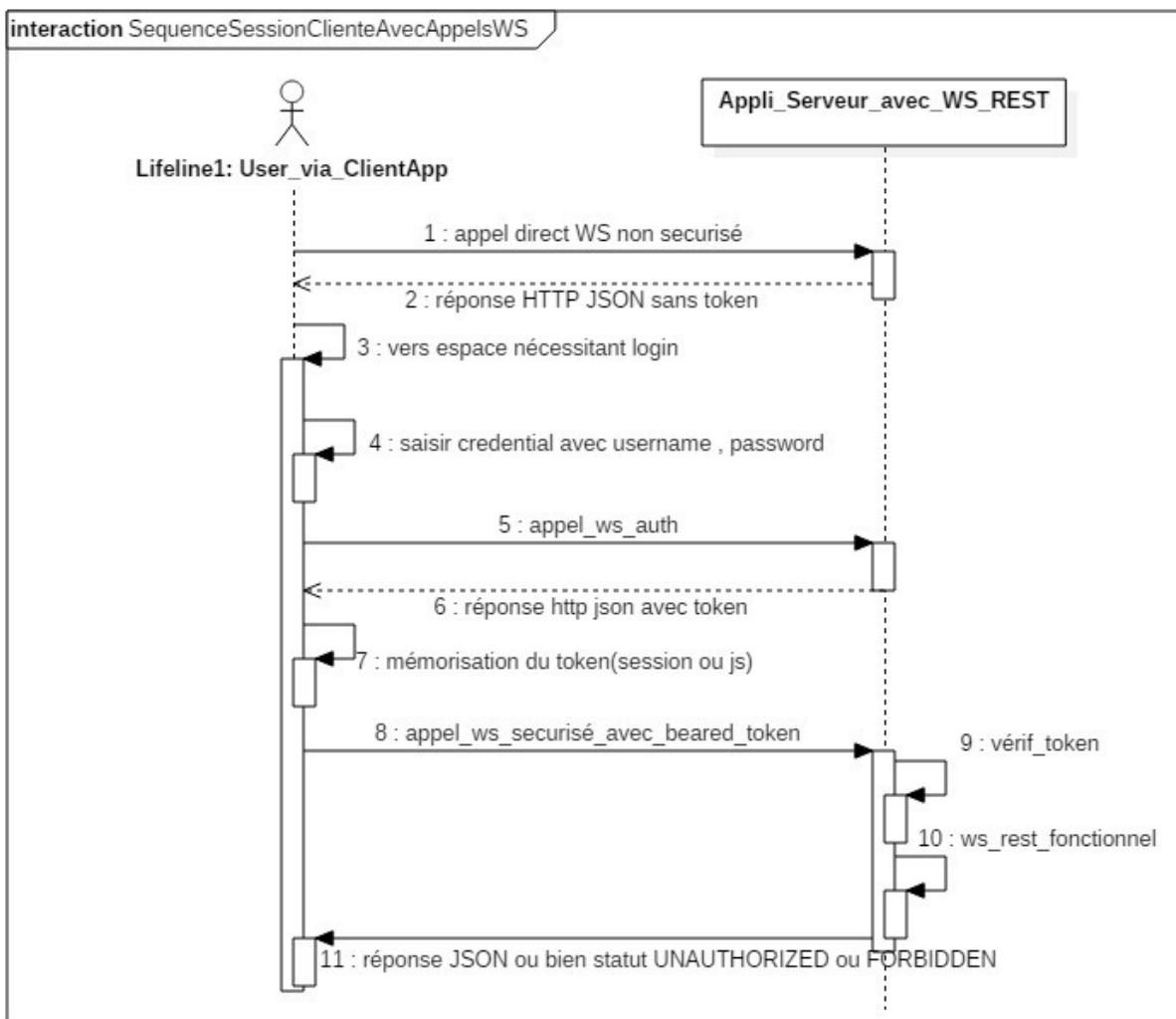
--> le coté serveur ne doit avoir qu'une confiance limitée aux requêtes angular .
Il est bon de revérifier fréquemment "token" et autres .

1.2. Conseils sur la structure d'une application angular sécurisée

- HTTPS / SSL dès qu'il faut échanger des informations confidentielles (username, password) , ...
- Un service d'authentification
- Des gardiens pour certaines routes

2. Token pour appels aux Web-services REST

2.1. Pseudo session avec "token" plutôt que cookie :



Des jetons de sécurité ("token") sont généralement employés pour gérer l'authentification d'un utilisateur et d'une application dans le cadre d'une communication sous forme de WS-REST.

Le jeton de sécurité est généré si le couple (username,password) transmis est correctement vérifié coté serveur.

Ce "token" (véhiculé au format "string") pourra prendre la forme d'un uuid (universal unique id, exemple: e51cd176-a522-454c-9c0a-36ca74cdb2d0) ou bien être conforme au format JWT (Json Web Token).

Dans le cas d'un token sophistiqué de type jwt, le token généré comporte déjà en lui (de manière cryptée/extractible) certaines informations utiles (subject, roles,) et donc pas besoin de map coté serveur.

Le protocole HTTP a normalisé la façon dont le token doit être retransmis au sein des requêtes

émises du client vers le serveur (après l'authentification préalablement effectuée) :

Il faut pour cela utiliser le champ "Authorization :" de l'entête HTTP pas en mode "Basic" mais en mode "Bearer" (signifiant "au porteur" en français).

exemple (postman) :

Authorization	Headers (1)	Body	Pre-request Script	Tests
Key				Value
Authorization				Bearer e51cd176-a522-454c-9c0a-36ca74cdb2d0
Never				Always

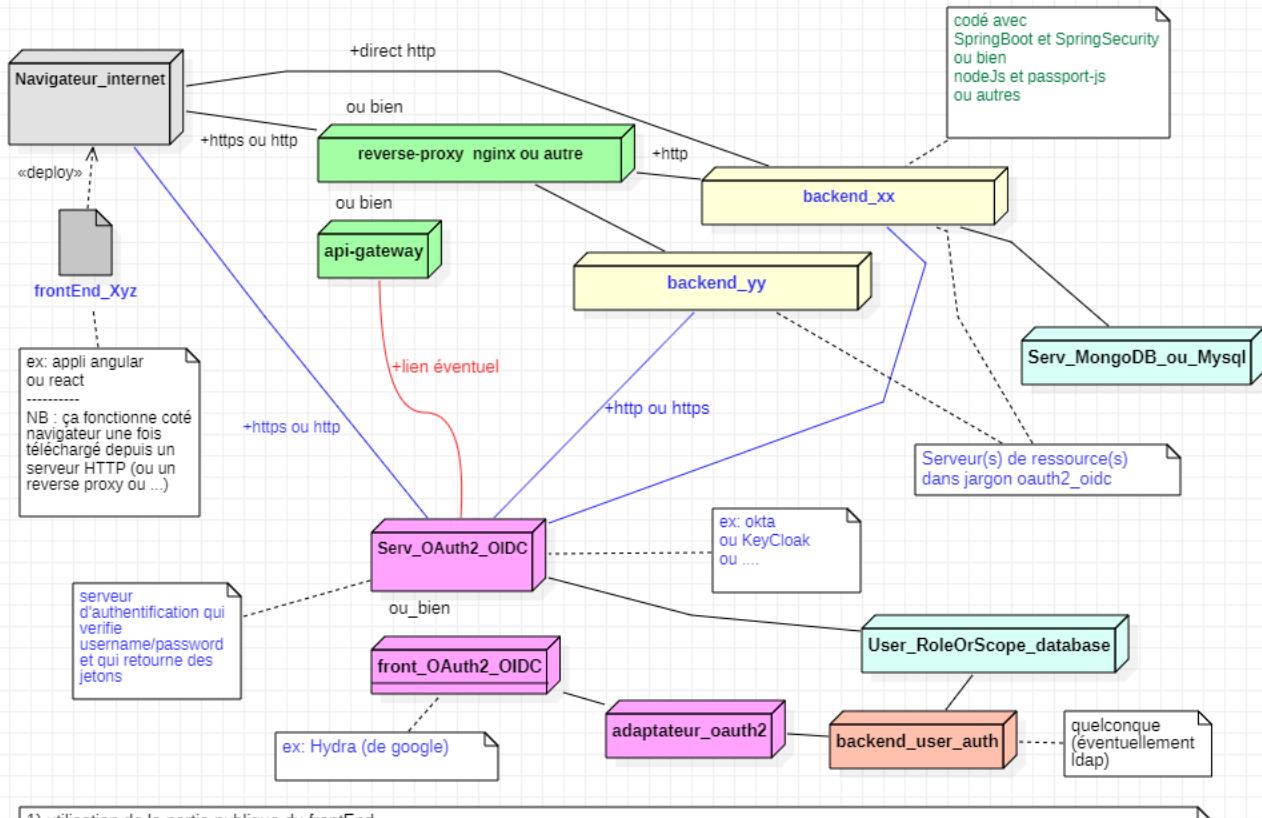
Status: 200 OK
si ok

ou bien Status: 403 Forbidden
si faux jeton (invalid token)

ou bien Status: 401 Unauthorized
si pas de jeton

3. Authentification via OAuth2/OIDC

Authentification avec OAuth2 et/ou OIDC



- 1) utilisation de la partie publique du frontEnd
- 2) le frontEnd (souvent dans une popup) délègue le login (et éventuels consentements) au serveur d'authentification OAuth2_OIDC
- 3) le serveur OAuth2_OIDC présente un formulaire de login (avec éventuels choix de scopes) et retourne si ok des jetons après échange de code
- 4) le frontEnd mémorise le ou les jeton(s) (dans session utilisateur ou sessionStorage ou autre)
- 5) le frontEnd retransmet le jeton dans tous les appels de Web-Services privés ("bearer token" Http)
- 6) le serveur "backend_xx_ou_yy" (appelé "serveur de ressources") communique si besoin avec le serveur OAuth2_OIDC pour vérifier la validité du jeton
- 7) en fonction d'éventuels scopes (droits d'accès) contenus dans le jeton le serveur "backend_xx_ou_yy" accepte ou pas de répondre à la requête

L'entreprise "Okta" propose un service "OAuth2_OIDC" clef en main à distance (en mode saas). Le serveur "Keycloak" (basé sur jboss-wildfly) est un serveur "OAuth2_OIDC" open source avec une console d'admin correcte. Certaines extension de Spring servent à établir simplement une communication entre un serveur de ressources basé sur java/springBoot et un serveur d'authentification OAuth2OIDC. Certaines extensions pour angular (telles que "angular-oauth2-oidc") servent à établir une délégation de login entre un frontEnd angular et un serveur d'authentification OAuth2OIDC.

3.1. Accès à un serveur d'autorisation depuis angular

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

src/silent-refresh.html

```
<html>
  <body>
    <script>
      var checks = [/\[?\&#]code=/, /\[?\&#]error=/, /\[?\&#]token=/, /\[?\&#]id_token=/];

      function isResponse(str) {
        if (!str) return false;
        for(var i=0; i<checks.length; i++) {
          if (str.match(checks[i])) return true;
        }
        return false;
      }

      var message = isResponse(location.hash) ? location.hash : '#' + location.search;

      if (window.parent && window.parent !== window) {
        // if loaded as an iframe during silent refresh
        window.parent.postMessage(message, location.origin);
      } else if (window.opener && window.opener !== window) {
        // if loaded as a popup during initial login
        window.opener.postMessage(message, location.origin);
      } else {
        // last resort for a popup which has been through redirects and can't use window.opener
        localStorage.setItem('auth_hash', message);
        localStorage.removeItem('auth_hash');
      }
    </script>
  </body>
</html>
```

angular.json

```
...
"assets": [
  ...
  "src/silent-refresh.html"
],
...
```

src/app/common/data/user_in_session.ts

```
export class UserInSession {
  constructor(public username = "?",
             public authenticated=false,
             public roles=?",
             public grantedScopes : string[] = []) {}
}
```

Dans une application angular avec module, ajouter

OAuthModule.forRoot() dans la partie `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';

src/app/common/service/session.service.ts

```
import { Injectable } from '@angular/core';
import { Router } from '@angular/router';
import { AuthConfig, OAuthErrorEvent, OAuthInfoEvent,
        OAuthService, OAuthSuccessEvent } from 'angular-oauth2-oidc';
import { UserInSession } from './data/user_in_session';
import { Location } from '@angular/common';

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

  private _userInSession = new UserInSession();

  public get userInSession() { return this._userInSession; }

  public set userInSession(u:UserInSession){
    this._userInSession=u;
    sessionStorage.setItem("session.userInSession",JSON.stringify(this._userInSession));
  }

  constructor(private oauthService: OAuthService , private router : Router,private location: Location) {
    this.initOAuthServiceForCodeFlow();
    let sUser = sessionStorage.getItem("session.userInSession");
    if(sUser) {
      this._userInSession = JSON.parse(sUser);
    }
  }

  initOAuthServiceForCodeFlow(){
    const authCodeFlowConfig: AuthConfig = {
      // Url of the Identity Provider
      issuer: 'https://www.d-defrance.fr/keycloak/realm/sandboxrealm',

      // URL of the SPA to redirect the user to after login
      //redirectUri: window.location.origin + "/ngr-loggedIn",

      silentRefreshRedirectUri: window.location.origin + this.location.prepareExternalUrl("/silent-refresh.html"),
      useSilentRefresh: true,
    }

    postLogoutRedirectUri : window.location.origin + this.location.prepareExternalUrl("/ngr-logInOut"),
    //ou /ngr-welcome ou ...

    // The SPA's id. The SPA is registered with this id at the auth-server
    clientId: 'anywebappclient',
    //clientSecret if necessary (not very useful for web SPA)
    //dummyClientSecret is required if client not public (client authentication: on + credential in keycloak)
    //dummyClientSecret: 'DMzPzIV2OQaphSbR84D7ebwxjrUNBgq5',
    responseType: 'code',

    // set the scope for the permissions the client should request
    // The first four are defined by OIDC.
    // Important: Request offline_access to get a refresh token
    // The api scope is a usecase specific one
    scope: 'openid profile resource.read resource.write resource.delete',
    showDebugInformation: true,
  };
}
```

```

this.oauthService.configure(authCodeFlowConfig);
this.oauthService.oidc = true; // ID_Token

this.oauthService.setStorage(sessionStorage);

this.oauthService.loadDiscoveryDocumentAndTryLogin();

this.oauthService.events.subscribe(
  event => {
    if (event instanceof OAuthSuccessEvent) {
      //console.log("OAuthSuccessEvent: "+JSON.stringify(event));
      console.log("OAuthSuccessEvent: "+event.type);
      this.manageSuccessEvent(event);
    }
    if (event instanceof OAuthInfoEvent) {
      // console.log("OAuthInfoEvent: "+JSON.stringify(event));
      console.log("OAuthInfoEvent: "+event.type);
    }
    if (event instanceof OAuthErrorEvent) {
      // console.error("OAuthErrorEvent: "+JSON.stringify(event));
      console.log("OAuthErrorEvent: "+event.type);
    } else {
      console.warn(event.type);
    }
  };
}//end of initOAuthServiceForCodeFlow

manageSuccessEvent(event : OAuthSuccessEvent){
if(event.type=="token_received") {
  console.log("***** token_received *****")
  this._userInSession.authenticated = true;
  let grantedScopesObj : object = this.oauthService.getGrantedScopes();
  this._userInSession.grantedScopes <any> grantedScopesObj;
  console.log("grantedScopes="+JSON.stringify(this._userInSession.grantedScopes));

  var claims : any = this.oauthService.getIdentityClaims();
  console.log("claims="+JSON.stringify(claims))
  if (claims) this._userInSession.username= claims.preferred_username + "("+ claims.name + ")";

  if(this.oauthService.silentRefreshRedirectUri != null){
    //this.router.navigateByUrl("/ngr-loggedIn");
    this.router.navigateByUrl("/ngr-welcome"); //with message from this SessionService if successfull login
  }
  }
}

delegateOidcLogin(){
  this.oauthService.initLoginFlowInPopup();
}

oidcLogout(){
  this.oauthService.logOut(); //clear tokens in storage and redirect to logOutEndpoint
  //this.oauthService.revokeTokenAndLogout(); //warning : problems if no CORS settings !!!!

  //delete old cookies (oauth2/oidc/keycloak):
  document.cookie = "AUTH_SESSION_ID=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/";
  document.cookie = "AUTH_SESSION_ID_LEGACY=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/";
}

public get accessTokenString() : string {
  return this.oauthService.getAccessToken();
}

```

src/app/common/interceptor/my-auth.interceptor.ts

```
import { Injectable } from '@angular/core';
import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable()
export class MyAuthInterceptor implements HttpInterceptor {

  constructor() {}

  intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {
    let access_token = sessionStorage.getItem('access_token');//NB: access_token for angular-oauth2-oidc or ...
    if(access_token && access_token!=""){
      const authReq = request.clone({headers:
        request.headers.set('Authorization', 'Bearer ' + access_token)});
      return next.handle(authReq);
    }
    else
      return next.handle(request);
  }
}
```

+ configuration classique de l'intercepteur (voir chapitre HTTP/Appel REST)

log-in-out.component.ts

```
import { Component, OnInit } from '@angular/core';
import { SessionService } from '../common/service/session.service';

@Component({
  selector: 'app-log-in-out',
  templateUrl: './log-in-out.component.html',
  styleUrls: ['./log-in-out.component.scss']
})
export class LogInOutComponent {

  constructor(public sessionService : SessionService) {}

  public login() { this.sessionService.delegateOidcLogin(); }

  public logout() { this.sessionService.oidcLogout(); }
}
```

log-in-out.component.html

```
<a href="https://www.d-defrance.fr/keycloak/realm/sandboxrealm/.well-known/openid-configuration"
target="_new">openid-configuration</a> &nbsp;<span class="redClass">(delete old cookies if necessary (login
error) !!!)</span> <br/>
LogInOut COMPONENT (login with OAuth2/OIDC keycloak server)<br/>
<p>
  <button (click)="login()" class="btn btn-primary">Login</button> &nbsp;&nbsp;
  <button (click)="logout()" class="btn btn-primary">Logout</button> &nbsp;<span
  class="redClass">(if logout error, go back and try login)</span>
```

</p>

footer.component.html

```
...
username=<b>{ {sessionService.userInSession.username} }</b> &nbsp;
authenticated=<b>{ {sessionService.userInSession.authenticated} }</b><br/>
grantedScopes=<b>{ {sessionService.userInSession.grantedScopes} }</b>
```

my-app welcome basic userAccount conversion oauth2/oidc loginOut devise (crud)

[openid-configuration](#) (delete old cookies if necessary (login error) !!!)

LoginOut COMPONENT (login with OAuth2/OIDC keycloak server)

[Login](#) [Logout](#) (if logout error, go back and try login)

exemples of good Username/Password:

- mgr1/pwd1 (MANAGE_RW : resource.read , resource.write)
- admin1/pwd1 (ADMIN_CRUD : resource.read , resource.write , resource.delete)
- user1/pwd1 (USER : resource.read)

[direct/standalone login \(without oauth2/oidc server\)](#)

footer - couleurFondPreferee: lightgreen ▾ username=? authenticated=false

roles=? grantedScopes=?

my-app welcome basic userAccount conversion oauth2/oidc loginOut devise (crud)

[openid-configuration](#) (delete old cookies if necessary (login error) !!!)

LoginOut COMPONENT (login with OAuth2/OIDC keycloak server)

[Login](#)

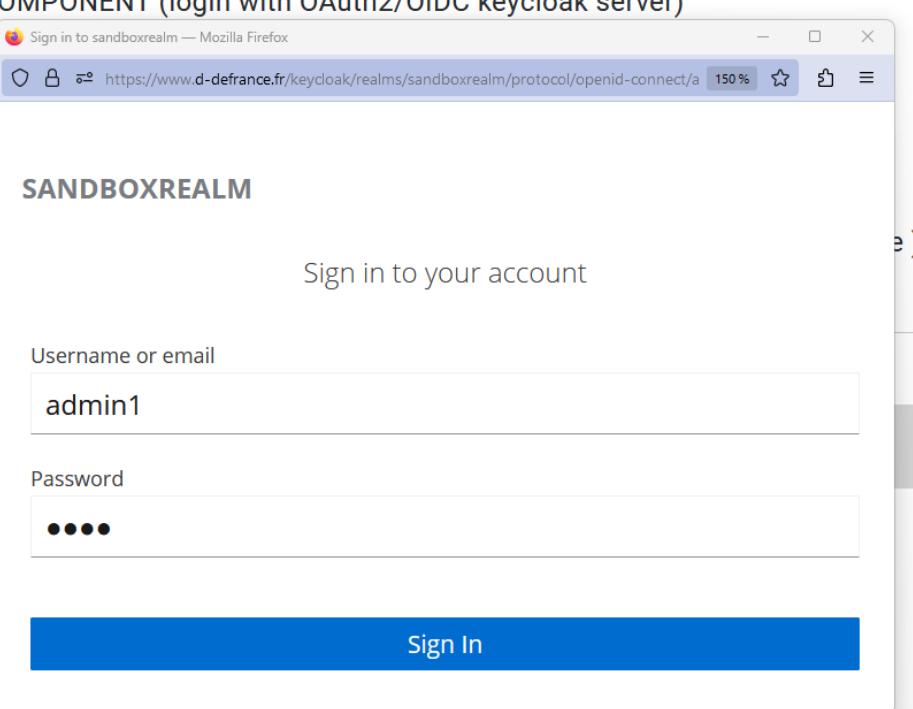
exemples de login

- mgr1/
- admin1/
- user1/

[direct/standalone login](#)

footer - couleurFondPreferee: lightgreen ▾

roles=? grantedScopes=?



footer - couleurFondPreferee: lightgreen ▾ username=admin1(jean Bon) authenticated=true
roles=? grantedScopes=openid,profile,resource.delete,resource.read,email,resource.write

XV - Tests unitaires (et ...) avec angular

1. Différent types de tests autour de angular

1.1. Vue d'ensemble / différents types et technologies de tests

Rappel du contexte: une application "Angular2+" correspond avant tout à la partie "Interface graphique" d'une architecture n-tiers et s'exécute au sein d'un navigateur web avec interprétation d'un code "typescript" transformé en "javascript".

Les principales fonctionnalités à tester sont les suivantes :

- **test unitaire d'un service** (avec éventuel "mock" sur accès aux données)
- **test unitaire d'un composant graphique** (avec éventuel mock sur "service")
- **test d'intégration complet (end to end)** englobant un dialogue HTTP/ajax/XHR avec des web services REST en arrière plan et sans avoir à connaître la structure interne de l'application (vue comme un boîte noire , vue de la même façon que depuis l'utilisateur final).

Pour tester du code javascript , la technologie de référence est "**jasmine**" . Avec quelques extensions pour angularJs ou Angular2+, cette technologie pourrait suffire à mettre en place des tests unitaires simples .

Etant donné que l'on souhaite également tester unitairement des composants graphiques (en javascript) qui s'exécutent dans un navigateur, on a également besoin d'une technologie de test qui puisse interagir avec un navigateur (chrome, firefox, ...) et c'est là qu'intervient "**karma**" .

Pour effectuer des tests globaux en mode "**end-to-end**" / "**boîte noire**" , on pouvait utiliser jusqu'à la version 11 ou 12 d'angular la technologie spécifique "**protractor**" qui permettait d'intégrer ensemble "**selenium**" et "**angular**" à travers des tests faciles à lancer.

A partir de Angular 12 ou 13, la technologie protractor n'est plus intégrée dans Angular et il est conseillé d'utiliser la technologie **cypress** (encore plus simple) à la place .

On a souvent besoin d'automatiser certaines étapes lors du lancement des tests. Une technologie annexe de script telle que "**Grunt**" ou "**gulp**" peut alors être intéressante (nb: dans le cas particulier de "angular CLI" , beaucoup de choses sont déjà automatisées derrière le lancement de "ng test" et "grunt" ou "gulp" n'est pas indispensable).

Dans la plupart des cas, le coeur de la configuration du projet est basé sur npm / package.json (avec éventuellement "angular_cli") et la configuration autour des tests est globalement la suivante :

package.json (npm)

- **grunt** ou **gulp** ou **angular_cli** (scripts)
 - **karma** ou **protractor** (interaction navigateur)
 - **jasmine** (tests codés en javascript)
 - et extensions pour angular

A titre de comparaison, dans le monde "java" :

l'équivalent de "npm" + "grunt" ou "gulp" correspond à "maven" , "ant" ou "graddle"
l'équivalent de "karma" ou "protractor" correspond à "selenium_driver" ou autre

l'équivalent de "jasmine" correspond à "JUnit" (+ extensions "mockito", "...").

2. Test "end-to-end / cypress"

2.1. Présentation de cypress

Cypress est une technologie JavaScript un peu plus moderne que Selenium permettant de déclencher des tests "end-to-end".

Les principales différences entre Cypress et Selenium sont que :

- Selenium fonctionne au-dehors d'un navigateur et les ordres sont donnés au navigateur via l'intermédiaire d'un WebDriver
- Cypress fonctionne directement en JavaScript dans un navigateur : il est donc beaucoup plus rapide et peut beaucoup plus facilement interagir avec l'arbre DOM et XHR / Ajax

Selenium a cependant été utilisé massivement par un très grand nombre de testeurs. Selenium reste une RÉFÉRENCE incontournable.

Le "Test Runner" Cypress est open-source et fourni sous licence MIT

2.2. Installation de cypress

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 ...

Eventuels 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 pertube pas les tests unitaires d'angular :

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

2.3. 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

Apres 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 à relancer ultérieurement.

2.4. Écriture d'un nouveau test Cypress

Au sein du répertoire .../cypress/integration ou .../cypress/e2e selon version , créer le nouveau fichier **myTest.spec.js** ou bien **myTest.spec.cy.js** (*selon version*) comportant un code de ce genre (à adapter) :

```
//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')

  })
})
```

2.5. Lancement d'un test Cypress

1. Lancer d'abord l'application web à tester (via **ng serve** ou **lite-server** ou autre) et d'éventuels "backends" en arrière plan.
- 2. npx cypress open** puis sélectionner le test à lancer

ou bien

```
npx cypress run --spec "cypress/e2e/myTest.spec.cy.js" --browser firefox >test_report.txt
```

Rapport test_report.txt généré : ...

```
| Cypress: 12.2.0
| Browser: Firefox 108 (headless)
| Specs: 1 found (myTest.spec.cy.js)
| Searched: my-app\cypress\e2e\myTest.spec.cy.js
```

Running: myTest.spec.cy.js

My Angular Tests

```
    ✓ good conversion (1519ms)
```

1 passing (3s)

NB : pour éviter un conflit entre les tests cypress et les tests unitaires d'angular, il faut ajouter ceci en bas de **tsconfig.json** :

```
,  
  "exclude": [  
    "cypress.config.ts",  
    "cypress/**/*.ts"  
  ]
```

3. Attention à la cohérence des tests

Les commandes d'angular-cli (ng new , ng g component , ...) génèrent par défaut des fichiers de tests pour chaque composant et chaque service .

Si on ne s'intéresse jamais aux tests (on ne les code pas, on ne les lance pas) , des fichiers xyz.spec.ts inutilisés (et souvent devenus incohérents par rapport au code des composants) ne sont que des fichiers inutiles .

Si par contre, on tient à lancer quelques tests unitaires, il vaut mieux que ceux-ci soient maintenus de manière cohérente avec le code , sinon --> erreurs de tous les cotés (manque import , ...) .

Coder et maintenir à jour un fichier xyz.spec.ts représente un gros travail (il faut y passer du temps).

Conséquence :

- il vaut mieux développer sans test , qu'avec des tests incohérents et plein de bugs .
- ne générer les fichiers xyz.spec.ts que si on les code et on les maintien à jour .
- une restructuration (*refactoring*) d'une application angular peut conduire à beaucoup de tests à retoucher pour que tout reste globalement cohérent (bon courage!!!) .

NB : Pour générer un nouveau composant xyz sans test unitaire au départ on peut utiliser l'option **--skipTests=true** de **ng g component xyz** .

Et l'on pourra ajouter le fichier xyz.spec.ts ultérieurement si besoin

4. Tests unitaires élémentaires

4.1. Tests unitaires en javascript

Pour écrire un test unitaire en javascript , les 2 technologies les plus utilisées sont :

- **jasmine**
- **mocha + chai**

Angular a fait le choix d'utiliser **jasmine** .

package.json :

```
...
"devDependencies": {
...
  "@types/jasmine": "~4.0.0",
  "jasmine-core": "~4.1.0",
  "karma": "~6.3.0",
  "karma-coverage": "~2.2.0",
  "karma-chrome-launcher": "~3.1.0",
  "karma-jasmine": "~5.0.0",
  "karma-jasmine-html-reporter": "^1.7.0",
}
```

...

Les assertions de jasmine s'expriment via `expect(...).to...`
(BDD syntax : Behavior Driven Development) :

```
expect(object_or_var)
.toEqual(expected)
.toContain(val)
.toBe(value)
.toBeCloseTo(value,delta)
.toBeDefined()
.toBeNull()
.toBeTruthy()/toBeFalsy()
.toHaveBeenCalled()
...
```

4.2. Test unitaire élémentaire (jasmine)

basicTest.spec.ts

```
describe('premiers tests', () => {
  it('1+1=2', () => expect(1+1).toBe(2));
  it('2+2=4', () => expect(2+2).toBe(4));
});
```

Ces spécifications de tests au format "jasmine" correspondent à un fichier d'extension ".spec.ts" (ou ".spec.js") et chaque partie "`() => expect(...).toBe(...)`" correspond à une assertion à vérifier.

Lancement d'un test unitaire jasmine (sans angular) :

```
npm install -g jasmine (si nécessaire)
tsc --skipLibCheck
jasmine dist\out-tsc\src\app\basicTest.spec.js
```

--> affiche (en cas de succès) :

```
Started
..
2 specs, 0 failures
Finished in 0.014 seconds
ou bien (ici en cas d'échec volontaire via expect(2+2).toBe(6)) :
```

```
Failures:
1) premiers tests 2+2=4
  Message:
    Expected 4 to be 6.
  Stack:
    Error: Expected 4 to be 6.
      at <Jasmine>
        at UserContext.it (D:\tp\local-git-mycontrib-repositories\tp_angular8+\basic-app\dist\out-tsc\src\app\basicTest.spec.js:3:37)
          at <Jasmine>
2 specs, 1 failure
```

Finished in 0.013 seconds

4.3. Tests structurés (avec jasmine) :

```
describe('tests structures avec jasmine', () => {
  let s : string;
  let x,y,z ;

  beforeAll((()=>{
    console.log("beforeAll called once");  s="abc";
  }));

  beforeEach((()=>{
    console.log("beforeEach called again");  x=1;y=2;
  }));

  describe('tests de calcul', () => {
    it('1+2=3', () => expect(x+y).toBe(3));
    it('1*2=2', () => expect(x*y).toBe(2));
  });

  describe('autres tests', () => {
    it('s=abc', () => expect(s).toBe('abc'));
    it('s comporte ab', () => expect(s).toContain('ab'));
  });

  afterEach((()=>{  console.log("afterEach called again"); });
  afterAll((()=>{  console.log("afterAll called once"); });
});
```

NB :

- **beforeEach()** et **afterEach()** sont appelées avant et après l'exécution de chaque **it()** pour les **describe()** de même niveau.
- Souvent , un niveau de **describe()** correspond à un **objet/composant/sujet à tester** et la fonction **it()** signifie "*it should behave like that*" .

5. Lancement tests "Angular" avec ng test et "karma"

ng test

Cette commande (de angular CLI) permet de lancer tous les tests unitaires "angular/jasmine" (fichiers `xyz.spec.ts`) via karma (de façon à interagir avec le navigateur).

Les résultats des tests déclenchés s'affichent de cette façon :

```

Karma v4.1.0 - connected
Chrome 80.0.3987 (Windows 10.0.0) is idle

 Jasmine 3.4.0
• • • • •

 7 specs, 0 failures, randomized with seed 64992

 AppComponent
  • should have as title 'basic-app'
  • should create the app
  • should render title in a h1 tag
 tests structures avec jasmine
 autres tests
  • s comporte ab
  • s=abc
 tests de calcul
  • 1*2=2
  • 1+2=3

```

Par défaut, **`ng test`** détecte tous les tests unitaires (`.spec.ts`) et les lance en mode « **`watch`** ».

Chaque modification enregistrées d'un code source provoque une ré-exécution des tests.

On peut compléter la commande "ng test" avec des options, comme :

- n'exécuter le test qu'une fois (`--watch=false`)
- générer un rapport de couverture de test (`--code-coverage`)

Exemple :

```
ng test --watch=false
```

et

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

pour ne lancer que les tests unitaires sur les services

6. Tests unitaires "angular"(composants, service, ...)

6.1. Rare test unitaire isolé (sans extension Angular)

Exemple :

Service élémentaire à tester (calcul de tva):

compute.service.ts

```
import { Injectable }      from '@angular/core';
@Injectable()
export class ComputeService {
  public vat(excl_tax : number, vat_pct : number ) : number{
    return excl_tax * vat_pct / 100;
  }
}
```

Test unitaire :

compute.service.spec.ts

```
import { ComputeService } from './compute.service';
// Isolated unit test = Straight Jasmine test
// no imports from Angular test libraries
describe('ComputeService without the TestBed', () => {
  let service: ComputeService;

  beforeEach(() => { service = new ComputeService(); });

  it('20%tva sur 200 ht = 40', () => {
    expect(service.vat(200,20)).toBe(40);
  });
});
```

ng test --> , 0 failures

ComputeService without the TestBed
• 20%tva sur 200 ht = 40

NB : Ce service de calcul à tester étant extrêmement simple, l'instance à tester a pu être créée par un simple new . Ce cas est très rare. Bien souvent , les services à tester ont des dépendances vis à vis d'autres services ou éléments du framework angular.

6.2. Test de service simple avec TestBed

Le test précédent peut être ré-écrit en s'appuyant sur "**TestBed**" . Ceci permet d'obtenir des instances d'éléments à tester bien initialisés par le framework angular et ses mécanismes d'injection de dépendances :

```
compute.service.spec.ts
import { TestBed } from '@angular/core/testing';

import { ComputeService } from './compute.service';

describe('ComputeService', () => {
  beforeEach(() => TestBed.configureTestingModule({
    /* providers: [ ComputeService ] already provided in root via @Injectable() */
  }));

  it('should be created', () => {
    const service: ComputeService = TestBed.inject(ComputeService);
    //NB: avant angular 9 , TestBed.get(ComputeService) ;
    //était moins bien que TestBed.inject() car le type de retour etait any
    expect(service).toBeTruthy();
  });

  it('20%tva sur 200 ht = 40', () => {
    const service: ComputeService = TestBed.get(ComputeService);
    expect(service.vat(200,20)).toBe(40);
  });
});
```

6.3. Test de service (avec mocks sur appels Ajax/http)

Exemple de service à tester :

common/service/devise.service.ts

```
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { ResConv } from '../data/res-conv';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Devise } from '../data/devise';

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

  //avec ng serve --proxy-config proxy.conf.json
  private basePublicUrl = "./devise-api/public/devise";
  private basePrivateUrl = "./devise-api/private/role_admin/devise";
```

```

private _headers = new HttpHeaders({'Content-Type': 'application/json'});

public convertir( montant :number, source :string , cible :string) : Observable<number> {
    let convertirUrl : string = null;
    convertirUrl = "./devise-api/public/convert?amount=" + montant
        +"&source=" + source + "&target=" + cible ;
    console.log( "convertirUrl = " + convertirUrl);
    return this.http.get<ResConv>(convertirUrl )
        .pipe(
            map( (res:ResConv) => res.result)
        );
}

postDevise(dev : Devise):Observable<Devise>{
    return this.http.post<Devise>(this.basePrivateUrl ,dev,{headers: this._headers} );
}

putDevise(dev : Devise):Observable<Devise>{
    return this.http.put<Devise>(this.basePrivateUrl ,dev,{headers: this._headers} );
}

public deleteDeviseServerSide(deviseCode):Observable<any>{
    console.log("will deleting devise of code = " + deviseCode );
    let deleteUrl : string = this.basePrivateUrl + "/" + deviseCode ;
    return this.http.delete(deleteUrl );
}

public getDevises() : Observable<Devise[]> {
    let deviseUrl : string = null;
    deviseUrl = this.basePublicUrl ;
    return this.http.get<Devise[]>(deviseUrl );
}

constructor(private http : HttpClient) { }
}

```

common/data/devise.ts

```

export class Devise {
    public code : string; // EUR ou USD ou ...
    public name : string; // Euro ou Dollar ou ...
    public change : number; //nb ... pour 1 dollar
}

```

common/data/res-conv.ts

```

export class ResConv {
    public source : string;
    public target : string;
    public amount : number;
    public result : number;
}

```

common/service/devise.service.spec.ts

```

import { TestBed , async , inject } from '@angular/core/testing';
import { DeviseService } from './devise.service';
import { HttpClientTestingModule, HttpTestingController }  

    from '@angular/common/http/testing';
import { HttpClient } from '@angular/common/http';

describe('DeviseService with mock http request/response', () => {

let service, http, backend;

beforeEach(() => TestBed.configureTestingModule({
  imports: [ HttpClientTestingModule ],
  /* providers: [ DeviseService ] already provided in root */
}));  
  

//injections pour le test à préparer
beforeEach(inject([DeviseService, HttpClient, HttpTestingController],  

  ( s: DeviseService, _h: HttpClient, _b: HttpTestingController) =>  

  {   service = s;   http = _h;   backend = _b; }  

));

it('should be created', () => {   expect(service).toBeTruthy();  

});  
  

it('should get good conversion', () => {
  //expected method behavior (just subscribe , deffered) :
  service.convertir(200,'EUR', 'USD').subscribe(res => {
    expect(res.toBeCloseTo(217.4 , 0.1));
  });
});  
  

//expected HTTP request built by convertir() method :
const req = backend.expectOne({
  url: './devise-api/public/convert?amount=200&source=EUR&target=USD',
  method: 'GET'
});  
  

//mock HTTP ResponseContent :
let convResult = {source:"EUR",target:"USD",amount:200, result:217.3913}
//déclenchement méthode avec mock http response:
req.flush(convResult, { status: 200, statusText: 'ok' });
});  
  

afterEach( inject([HttpTestingController], (httpMock: HttpTestingController) => {
  httpMock.verify(); //requete bien terminée?
  })
);
});
});
```

6.4. Test réel de Service http sans mock (mais avec "backEnd" accessible en arrière plan)

devise.service.spec.ts

```
import { HttpClient, HttpClientModule } from '@angular/common/http';
import { TestBed } from '@angular/core/testing';
import { fail } from 'assert';

import { DeviseService } from './devise.service';

describe('DeviseService', () => {
  let service: DeviseService;

  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [HttpClientModule],
      providers: [HttpClient] /* providers: [DeviseService] already provided in root */
    });
    service = TestBed.inject(DeviseService);

    /*NB : pour service.apiBaseUrl=".devise-api"; en relatif (comme ng serve et proxy.conf.json)
     il faut ajouter l'option suivante dans le fichier karma.conf.js :
     proxies: {
       '/devise-api': {
         'target': 'http://localhost:8282/devise-api',
         'changeOrigin': true
       }
     }
    */
  });

  it('should be created', () => {
    expect(service).toBeTruthy();
  });

  it('should convert EUR to USD correctly with backend WS ', (done) => {
    service.convertir$(200,"EUR","USD").subscribe(
      (montantConverti) => {
        console.log("montantConverti=" + montantConverti);
        expect(montantConverti).toBeCloseTo(217.4,0.1);
        done();
      },
      (err) => { fail("convert error :" + JSON.stringify(err));}
    )
  });
});
```

// ng test --include=**/service/*.spec.ts pour ne lancer que les tests unitaires sur les services

karma.conf.js (à placer à la racine de l'application angular)

```
// Karma configuration file, see link for more information
// https://karma-runner.github.io/1.0/config/configuration-file.html

module.exports = function (config) {
  config.set({
    basePath: '',
    frameworks: ['jasmine', '@angular-devkit/build-angular'],
    plugins: [
      require('karma-jasmine'),
      require('karma-chrome-launcher'),
      require('karma-jasmine-html-reporter'),
      require('karma-coverage'),
      require('@angular-devkit/build-angular/plugins/karma')
    ],
    client: {
      jasmine: {
        // you can add configuration options for Jasmine here
      },
      clearContext: false // leave Jasmine Spec Runner output visible in browser
    },
    jasmineHtmlReporter: {
      suppressAll: true // removes the duplicated traces
    },
    coverageReporter: {
      dir: require('path').join(__dirname, './coverage/my-app'),
      subdir: '',
      reporters: [
        { type: 'html' },
        { type: 'text-summary' }
      ]
    },
    reporters: ['progress', 'kjhtml'],
    port: 9876,
    colors: true,
    LogLevel: config.LOG_INFO,
    autoWatch: true,
    browsers: ['Chrome'],
    singleRun: false,
    restartOnFileChange: true,
    proxies: {
      '/devise-api': {
        'target': 'http://localhost:8282/devise-api',
        'changeOrigin': true
      }
    });
};
```

6.5. Test unitaire de composant angular avec TestBed

Exemple :

calculatrice.component.ts

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

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

  modeChoisi : string = "simple"; // "simple" ou "sophistiquee"

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

  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;
    }
  }

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

  ngOnInit(): void {
  }
}
```

calculatrice.component.html

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

  <label>a :</label> <input type="number" name="a" [(ngModel)]="a" > <br>
  <label>b :</label> <input type="number" name="b" [(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('*')"
      [style.visibility]="modeChoisi=='sophistiquee'? 'visible': 'hidden'" >
    <br>
  <label>resultat:</label>
  <span [style.font-weight]="res>0?'bold':'normal'"
    [class.negatif]="res<0" id="spanRes">
    {{res}}</span> <br>
</div>
```

Exemple de test :

calculatrice.component.spec.ts

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { FormsModule } from '@angular/forms';
import { AppRoutingModule } from 'src/app/app-routing.module';

import { CalculatriceComponent } from './calculatrice.component';

describe('CalculatriceComponent', () => {
  let component: CalculatriceComponent;
  let fixture: ComponentFixture<CalculatriceComponent>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [FormsModule, AppRoutingModule],
      declarations: [CalculatriceComponent]
    })
    .compileComponents(); // compile asynchronously template and css , return promise

    fixture = TestBed.createComponent(CalculatriceComponent);
    component = fixture.componentInstance;
    fixture.detectChanges(); //ici sauf si initialisations asynchrones dans ngOnInit()
  });

  it('5+6=11 from model', () => {
    component.a=5;
    component.b=6;
    component.onCalculer('+');//à ne pas oublier d'appeler si pas de dispatchEvent
    fixture.detectChanges();
    const compNativeElt = fixture.debugElement.nativeElement;
    let spanResElt = compNativeElt.querySelector('#spanRes');
    console.log("from model, res:" + spanResElt.innerText);
    expect(spanResElt.innerText).toContain('11'));
  });

  it('10-3=7 from IHM', () => {
    const compNativeElt = fixture.debugElement.nativeElement;
    let aInputElt = compNativeElt.querySelector("input[name='a']");
    aInputElt.value=10;
    aInputElt.dispatchEvent(new Event('input'));

    let bInputElt = compNativeElt.querySelector("input[name='b']");
    bInputElt.value=3;
    bInputElt.dispatchEvent(new Event('input'));

    let moinsButtonElt =
      compNativeElt.querySelector("input[type='button'][value='-']");
    //moinsButtonElt.dispatchEvent(new Event('click'));
    moinsButtonElt.click();
    fixture.detectChanges();
    expect(component.a).toBe(10);
    expect(component.b).toBe(3);
    expect(component.res).toBe(7);
    let spanResElt = compNativeElt.querySelector('#spanRes');
    console.log("from IHM, res:" + spanResElt.innerText);
    expect(spanResElt.innerText).toContain('7'));
  });
});
```

// ng test --watch=false --include=**/calculatrice/*.spec.ts

La librairie `@angular/core/testing` comporte entre autres la classe fondamentale **TestBed** qui permet de créer un module/environnement de test (de type `@NgModule`) à partir de la méthode `configureTestingModule()` qui admet des paramètres d'initialisation très semblables à ceux de la décoration `@NgModule` que l'on trouve par exemple dans `app.module.ts`.

Il est conseillé d'appeler cette méthode à l'intérieur de `beforeEach()` de façon à ce que chaque test soit indépendant des autres (avec un contexte ré-initialisé) .

Seulement après une bonne et définitive configuration, la méthode `TestBed.createComponent()` permet de créer une chose technique de type `ComponentFixture<ComponentType>` sur laquelle on peut invoquer :

`.componentInstance` de façon à accéder à l'instance du composant à tester

`.debugElement.nativeElement` de façon à accéder au nœud d'un arbre DOM lié au template du composant à tester.

NB : la méthode `fixture.detectChanges()` permet d'explicitement (re)synchroniser la vue HTML en fonction des changements effectués au niveau du modèle.

Il est éventuellement possible d'importer le service automatique

```
import { ComponentFixtureAutoDetect } from '@angular/core/testing';
```

et de le déclarer à l'initialisation de `TestBed` via ce code :

```
TestBed.configureTestingModule({
  declarations: [ AppComponent ],
  providers: [
    { provide: ComponentFixtureAutoDetect, useValue: true }
  ]
})
```

cependant cette invocation automatique et implicite de `detectChanges()` étant asynchrone , il y a beaucoup de cas où l'appel explicite sera nécessaire pour immédiatement tenir compte d'un changement au niveau des lignes de code séquentielles de l'écriture d'un test.

Simuler/déclencher événement via `HTMLInputElement.dispatchEvent()` :

```
it('...', () => {
  const compNativeElt = fixture.debugElement.nativeElement;
  let htInputElt = compNativeElt.querySelector("input[name='ht']");
  htInputElt.value=200;
  htInputElt.dispatchEvent(new Event('input'));
  fixture.detectChanges();
  expect(...).toBeCloseTo(...);
});
```

NB: `moinsButtonElt.click();` est équivalent à `moinsButtonElt.dispatchEvent(new Event('click'));`

6.6. Explications autour de `async()` et `done` :

Le framework jasmine lance automatiquement en boucle les fonctions `beforeEach()` et `it()` ;
le code interne des méthodes `beforeEach()` et/ou `it()` est :

- soit entièrement synchrone et aucune attente n'est à prévoir
- soit en partie asynchrone et certaines attentes sont à paramétrier

Le framework "**jasmine**" offre de façon standard une fonction **`done()`** permettant de l'avertir de la fin d'une fonction appelée `beforeEach()` ou `it()` :

```
beforeEach_or_it(..., (done) => {
    appel_synthronique1();
    appel_synthronique2();
    appel_asynchrone_retournant_promise().then( () => {
        instructions_de_la_callback_asynchrone;
        done();
    });
});
```

L'ancienne fonction utilitaire **`async()`** de [@angular/core/testing](#) est une alternative simplifiée permettant de ne pas avoir à appeler explicitement `done()` du standard jasmine. Le `then / done` est caché à l'intérieur.

TypeScript et javascript es2017+ comportent maintenant (de manière standardisée les mots clefs **`async`** et **`await`**).

Au sein d'un projet moderne, il vaut mieux utiliser le mot clef standard **`async`** plutôt que l'ancienne fonction de même nom et dans les 2 cas , plus besoin d'appeler `done()` .

```
beforeEach_or_it(..., async () => {
    appel_synthronique1();
    appel_synthronique2();
    let resAsync = await appel_asynchrone_retournant_promise();
    instructions_apres_recup_resultat_appel_asynchrone;
});
```

6.7. Tester un composant angular utilisant un service simple

Exemple de composant à tester (avec service simple injecté) :

tva-with-service.component.ts

```
import { Component, OnInit } from '@angular/core';
import { TvaService } from 'src/app/common/service/tva.service';

@Component({
  ...
})
export class TvaWithServiceComponent implements OnInit {
  tabTaux :number[] = [ 5 , 10 , 20 ];
  ht /*:number*/ = 0;
  tauxTva /*:number*/ = 20; //en %
  tva /*:number*/ =0;
  ttc /*:number*/ =0;

  onCalculTvaTtc(){
    this.tva = this.tvaService.tva(this.ht,this.tauxTva);
    this.ttc = this.ht + this.tva ;
  }

  constructor(private tvaService : TvaService) { }
  ...
}
```

tva.service.ts

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

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

  public tva(ht : number, tauxTvaPct : number ) : number{
    return ht * tauxTvaPct / 100;
  }
}
```

tva-with-service.component.html

```
<p id="p1">calcul de tva (avec service simple)</p>
<label>ht:</label> <input type="number" name="ht" [class.siZero]="ht==0"
  [(ngModel)]="ht" (input)="onCalculTvaTtc()" /> <br/>
<label>tauxTva:</label>
<select [(ngModel)]="tauxTva" name="tauxTva" (change)="onCalculTvaTtc()">
  <option *ngFor="let t of tabTaux" [ngValue]="t" >{{t}}%</option>
```

```
</select>
<div *ngIf="tva>0"> ...
  ttc: <span id="spanTtc" class="enEvidence">{ {ttc | number:'1.0-2'} }</span> <br/>
</div>
```

calcul de tva (avec service simple)

ht: 200

tauxTva: 20%

tva: 40

ttc: 240

Exemple de test :

tva-with-service.component.spec.ts

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { FormsModule } from '@angular/forms';
import { TvaService } from 'src/app/common/service/tva.service';
import { TvaWithServiceComponent } from './tva-with-service.component';

describe('TvaWithServiceComponent', () => {
  let component: TvaWithServiceComponent;
  let fixture: ComponentFixture<TvaWithServiceComponent>;

  beforeEach(async () => {
    let tvaServiceStub = {
      tva(ht : number, tauxTvaPct : number) : number{
        return ht * tauxTvaPct / 100;
      }
    };

    await TestBed.configureTestingModule({
      imports: [ FormsModule ], /* providers : [TvaService] ,*/
      providers: [ {provide : TvaService,
                    useValue : tvaServiceStub } ],
      declarations: [ TvaWithServiceComponent ]
    })
    .compileComponents();
  });

  fixture = TestBed.createComponent(TvaWithServiceComponent);
  component = fixture.componentInstance;
  fixture.detectChanges();
});

it('10% , 200 -> 220 from IHM', () => {
  const compNativeElt = fixture.debugElement.nativeElement;
  let htInputElt = compNativeElt.querySelector("input[name='ht']");
  htInputElt.value=200;
  htInputElt.dispatchEvent(new Event('input'));
```

```

let tauxTvaSelectElt = compNativeElt.querySelector("select[name='tauxTva']");
tauxTvaSelectElt.value=tauxTvaSelectElt.options[1].value; //0: 5.5% , 1: 10% , 2: 20%
tauxTvaSelectElt.dispatchEvent(new Event('change'));
fixture.detectChanges();
expect(component.ht).toBe(200);
expect(component.tauxTva).toBe(10);
let ttcElt = compNativeElt.querySelector('#spanTtc');
console.log("from ihm, ttc:" + ttcElt.innerText);
expect(ttcElt.innerText).toContain('220');
});

});

// ng test --watch=false --include=**/tva-with-service/*.spec.ts
// ng test --include=**/tva-with-service/*.spec.ts

```

Remarque importante :

Bien que dans cet exemple extra-simple, on aurait pu utiliser en direct le réel service de calcul via `TestBed.configureTestingModule({`

```

    providers: [ TvaService ] ,
...} );
```

il est en général conseillé de demander à injecter un service de type "stub" ou "mock" (simulant le comportement du réel service et permettant de se focaliser sur le composant angular à tester) :

```

//stub Service for test purposes (will be cloned and injected)
let tvaServiceStub = {
  tva(ht : number, tauxTvaPct : number ) : number{
    return ht * tauxTvaPct / 100;
  }
};

await TestBed.configureTestingModule({
  imports: [ FormsModule ] ,      /* providers : [TvaService] ,*/
  providers: [ {provide : TvaService,
    useValue : tvaServiceStub } ],
  declarations: [ TvaWithServiceComponent ]
})
.compileComponents();
```

6.8. Tester un composant angular utilisant un service asynchrone

Exemple de composant à tester (utilisant un service asynchrone) :

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',
  templateUrl: './conversion.component.html',
  styleUrls: ['./conversion.component.scss']
})
export class ConversionComponent implements OnInit {

  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;},
        error : (err:any) => { console.log("error:"+err)}
      });
  }

  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(): void {
    this._deviseService.getAllDevises$()
      .subscribe({
        next: (tabDev : Devise[])=>{ this.initListeDevises(tabDev); },
        error: (err) => { console.log("error:"+err)}
      });
  }
}
```

conversion.component.html

```
<h3>conversion de devise</h3>
montant : <input [(ngModel)]="montant" name="montant" /><br/>
code devise source : <select [(ngModel)]="codeDeviseSource" name="codeDevSource">
  <option *ngFor="let d of listeDevises">{{d.code}}</option>
</select> <br/>
code devise cible : <select [(ngModel)]="codeDeviseCible" name="codeDevCible">
  <option *ngFor="let d of listeDevises">{{d.code}}</option>
</select> <br/>
<input type="button" (click)="onConvertir()" value="convertir" /> <br/>
montantConverti = <span id="montantConverti">{{montantConverti}}</span>
```

conversion de devise

montant :

code devise source :

code devise cible :

montantConverti = 220.00000000000003

Mock du service asynchrone (spyOn(...).and...)

En règle générale un service asynchrone effectue un dialogue HTTP/XHR avec un web service REST distant (fonctionnant sur un autre ordinateur en Php , java , nodeJs ou autre).

Cette dépendance externe n'étant pas facile à gérer durant les tests unitaires, la stratégie préconisée dans la plupart des cas consiste à préparer/initialiser le test en :

- injectant le réel service dans le composant à tester (paramétrage providers : [ServiceXy] de TestBed)
- redéfinissant ponctuellement/localement la fonction du service qui sera appelée (via spyOn(...).and.returnValue() ou bien spyOn(...).and.callFake(function(...){ })) .

Exemple :

conversion.component.spec.ts

```
import { HttpClientModule } from '@angular/common/http';
import { ComponentFixture, fakeAsync, TestBed,
         tick, waitForAsync } from '@angular/core/testing';
import { FormsModule } from '@angular/forms';
import { DeviseService } from './common/service/devise.service';
import { Observable, of } from 'rxjs';
import { delay } from 'rxjs/operators';
import { ConversionComponent } from './conversion.component';
import { Devise } from './common/data/devise';

describe('ConversionComponent', () => {
  let component: ConversionComponent;
```

```

let fixture: ComponentFixture<ConversionComponent>;
let deviseServiceWithinTest : DeviseService;
let spyAndFakeGetAllDevises : jasmine.Spy;
let spyAndFakeConvertir : jasmine.Spy;

beforeEach(waitForAsync(() => {
  TestBed.configureTestingModule({
    imports: [ FormsModule , HttpClientModule] ,
    declarations: [ ConversionComponent ]
    /* providers: [ DeviseService ] already provided in root via @Injectable() */
  })
  .compileComponents();
}));

beforeEach(() => {
  fixture = TestBed.createComponent(ConversionComponent);
  component = fixture.componentInstance;
  //fixture.detectChanges(); //NOT HERE BECAUSE ngOnInit() contains async call(s)
  deviseServiceWithinTest = fixture.debugElement.injector.get(DeviseService);

  let stubDevises : Devise[] = [
    new Devise('EUR','euro',1.0),
    new Devise('USD','dollar',1.1),
    new Devise('GBP','livre',0.9)
  ];
  spyAndFakeGetAllDevises = spyOn(deviseServiceWithinTest, 'getAllDevises$')
    .and.returnValue(of(stubDevises).pipe(delay(44)/*simu 44ms*/));
}

spyAndFakeConvertir = spyOn(deviseServiceWithinTest, 'convertir$')
  .and.callFake((montant : number, source :string , cible : string):Observable<number> =>{
  let convResult = 0;
  if(source==='EUR'&&cible==='USD')
    convResult=220.0;
  else if(source==cible)
    convResult=montant;
  return of(convResult).pipe(delay(44)/*simu 44ms*/);
});
});

it('should display good conversion result with fakeAsync', fakeAsync(() => {
  fixture.detectChanges(); //waiting for start of initial callback and bindings (ngOnInit , injections, ...)
  expect(spyAndFakeGetAllDevises.calls.any())
    .withContext('getAllDevises() should be called').toBe(true);
  console.log("before first tick(), component.codeDeviseSource="+component.codeDeviseSource);
  console.log("before first tick(), component.codeDeviseCible="+component.codeDeviseCible);
  tick(50); //waiting for async result of async calls in ngOnInit
  fixture.detectChanges(); //waiting for callback and bindings
  console.log("after first tick(), component.codeDeviseSource="+component.codeDeviseSource);
  console.log("after first tick(), component.codeDeviseCible="+component.codeDeviseCible);

  const compNativeElt = fixture.debugElement.nativeElement;
})
);

```

```

let montantInputElt = compNativeElt.querySelector("input[name='montant']");
montantInputElt.value=200;
montantInputElt.dispatchEvent(new Event('input'));
//component.codeDeviseSource='EUR'; // pre-version
let codeDevSourceSelectElt = compNativeElt.querySelector("select[name='codeDevSource']");
codeDevSourceSelectElt.value='EUR';
codeDevSourceSelectElt.dispatchEvent(new Event('change'));

//component.codeDeviseCible='USD'; // pre-version
let codeDevCibleSelectElt = compNativeElt.querySelector("select[name='codeDevCible']");
codeDevCibleSelectElt.value='USD';
codeDevCibleSelectElt.dispatchEvent(new Event('change'));

let convButtonElt = compNativeElt.querySelector("input[type='button'][value='convertir']");
//convButtonElt.dispatchEvent(new Event('click'));
convButtonElt.click();
fixture.detectChanges(); //waiting for callback and bindings

expect(component.montant)
.withContext('component.montant should be 200 after input')
.toBeCloseTo(200,0.0001);

expect(component.codeDeviseSource)
.withContext('component.codeDeviseSource should be EUR after selection')
.toBe('EUR');

expect(component.codeDeviseCible)
.withContext('component.codeDeviseCible should be USD after selection')
.toBe('USD');

expect(spyAndFakeConvertir.calls.any())
.withContext('convertir$() should be called').toBe(true);

tick(50); //waiting for async result of async calls in onConvertir
fixture.detectChanges(); //waiting for callback and bindings

let spanResElt = compNativeElt.querySelector('#montantConverti');
console.log("from IHM, montantConverti:" + spanResElt.innerText);
expect(spanResElt.innerText)
.withContext('spanResElt.innerText (#montantConverti) should be 220')
.toBeCloseTo(220 , 0.1);
});

it('should display good conversion result with async/await fixture.whenStable()', async () => {
  fixture.detectChanges();//waiting for start of initial callback and bindings (ngOnInit , injections, ...)
  expect(spyAndFakeGetAllDevises.calls.any())
  .withContext('getAllDevises$() should be called').toBe(true);
  console.log("before await fixture.whenStable(), component.codeDeviseSource='"+component.codeDeviseSource);
  console.log("before await fixture.whenStable(), component.codeDeviseCible='"+component.codeDeviseCible);
  await fixture.whenStable();
  fixture.detectChanges()//waiting for callback and bindings
  console.log("after await fixture.whenStable(), component.codeDeviseSource='"+component.codeDeviseSource);
  console.log("after await fixture.whenStable(), component.codeDeviseCible='"+component.codeDeviseCible);

```

```

const compNativeElt = fixture.debugElement.nativeElement;
let montantInputElt = compNativeElt.querySelector("input[name='montant']");
montantInputElt.value=200;
montantInputElt.dispatchEvent(new Event('input'));
//component.codeDeviseSource='EUR'; // pre-version
let codeDevSourceSelectElt = compNativeElt.querySelector("select[name='codeDevSource']");
codeDevSourceSelectElt.value='EUR';
codeDevSourceSelectElt.dispatchEvent(new Event('change'));

//component.codeDeviseCible='USD'; // pre-version
let codeDevCibleSelectElt = compNativeElt.querySelector("select[name='codeDevCible']");
codeDevCibleSelectElt.value='USD';
codeDevCibleSelectElt.dispatchEvent(new Event('change'));

let convButtonElt = compNativeElt.querySelector("input[type='button'][value='convertir']");
//convButtonElt.dispatchEvent(new Event('click'));
convButtonElt.click();
fixture.detectChanges(); //waiting for callback and bindings
expect(component.montant)
.withContext('component.montant should be 200 after input')
.toBeCloseTo(200,0.0001);
expect(component.codeDeviseSource)
.withContext('component.codeDeviseSource should be EUR after selection')
.toBe('EUR');
expect(component.codeDeviseCible)
.withContext('component.codeDeviseCible should be USD after selection')
.toBe('USD');
expect(spyAndFakeConvertir.calls.any())
.withContext('convertir$() should be called').toBe(true);
await fixture.whenStable(); //waiting for async result of async calls in onConvertir
fixture.detectChanges(); //waiting for callback and bindings
let spanResElt = compNativeElt.querySelector('#montantConverti');
console.log("from IHM, montantConverti:" + spanResElt.innerText);
expect(spanResElt.innerText)
.withContext('spanResElt.innerText (#montantConverti) should be 220')
.toBeCloseTo(220 , 0.1);
});

});

// ng test --watch=false --include=**/conversion/*.spec.ts
// ng test --include=**/conversion/*.spec.ts

```

NB : le **.withContext("...")** de jasmine permet d'indiquer/afficher un message de contexte dans le rapport de test . Ce message s'affichera à coté de l'assertion non vérifiée.

NB : En théorie ,

```
async ()=> {
    appel_synthronique_1();
    appel_synthronique_2();
    appel_asynchrone_Xy_retournant_promise_ou_observable();
    fixture.whenStable().then(() => {
        //zone de test (attente automatique de toute promise ou observable
        //indirectement déclenchée)
        fixture.detectChanges();
        expect(...).toBe(...);
    });
}
```

pouvant s'écrire (de manière plus moderne)

```
async ()=> {
    appel_synthronique_1();
    appel_synthronique_2();
    appel_asynchrone_Xy_retournant_promise_ou_observable();
    await fixture.whenStable();
    //zone de test (attente automatique de toute promise ou observable indirectement déclenchée)
    fixture.detectChanges();
    expect(...).toBe(...);
}
```

et

```
fakeAsync()=> {
    appel_synthronique_1();
    appel_synthronique_2();
    appel_asynchrone_Xy_retournant_promise_ou_observable();
    tick(); // attente au sein de fakeAsync()
    fixture.detectChanges();
    expect(...).toBe(...);
}
```

sont censés être **à peu près équivalent** et **permettre d'attendre la fin d'une opération asynchrone** indirectement déclenchée par un composant angular et un service.

En pratique, certains bugs, limitations techniques et autres problèmes font qu'un test d'appel asynchrone est assez délicat à mettre au point.

Plus en détails :

tick(25) ; effectue une attente de 25ms .

Deux appels consécutifs à tick(25) effectuent globalement une attente de 50 ms .

Sans argument, tick() est équivalent à tick(0).

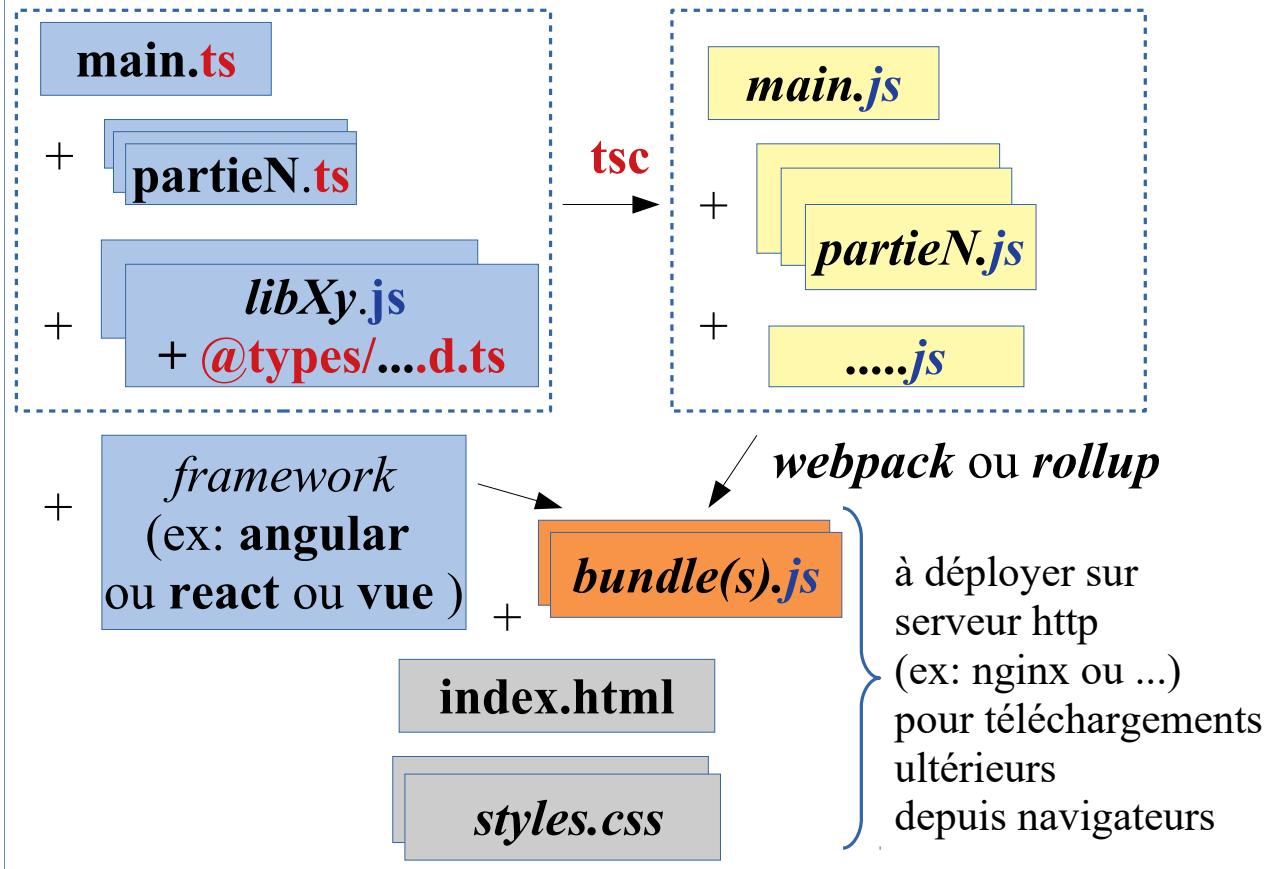
Depuis angular 4.3 , flush() peut quelquefois être utilisé à la place de tick().

A priori , fakeAsync() et les mécanismes de "mock" d'angular mettent en place des "timeout" internes et un appel à tick() allonge le temps d'attente tandis qu'un appel à flush() récupère le temps d'attente une fois l'opération terminée.

XVI - Packaging et déploiement d'appli. angular

1. Notion de "bundle" pour le déploiement

Utilisation de **typescript** coté **client/frontEnd**



> tp-angular > j4 > my-app > dist > my-app

Rechercher dans : my-app

Nom	Modifié le	Type	Taille
assets	31/12/2022 11:01	Dossier de fichiers	
3rdpartylicenses.txt	31/12/2022 11:01	Document texte	22 Ko
favicon.ico	07/06/2022 18:49	Fichier ICO	1 Ko
index.html	31/12/2022 11:01	Firefox HTML Doc...	4 Ko
main.6d795a0e21eff236.js	31/12/2022 11:01	JSFile	279 Ko
polyfills.e9d7fa7bfa9afaf3.js	31/12/2022 11:01	JSFile	33 Ko
runtime.397c3874548e84cd.js	31/12/2022 11:01	JSFile	2 Ko
styles.38c9f60f2e08c552.css	31/12/2022 11:01	Fichier source CSS	249 Ko

construits via **ng build**, les fichiers dist/my-app/main.....js et runtime....js sont des **bundles**. ng build utilise en interne la technologie **webpack** ou bien **vite/rollup** pour générer les bundles en analysant finement les **import { ... } from '...'** entre modules de typescript/javascript (*tree shaking*). Au final, tout le code de l'application angular tient en quelques fichiers rapides à télécharger.

2. JIT vs AOT (Ahead-Of-Time) pour angular 2 à 8

Une application angular est principalement constituée de fichier ".ts" et ".html".

Au moment de l'exécution du code au sein d'un navigateur, même les templates ".html" sont transformés en code javascript au niveau des mécanismes internes.

Cette "compilation/transpilation" (.ts + .html) => "..bundle.js" peut être effectuée de 2 manières :

- par le compilateur "**jit**" (*just in time*)
- par le compilateur "**aot**" (*ahead-of-time*)

Le choix du mode de compilation pouvait à l'époque s'effectuer en plaçant ou pas l'option **--aot** au niveau de **ng serve** ou **ng build** :

Lancement par défaut avec "jit" :

ng serve
ng build

Lancement avec "aot" :

ng serve --aot
ng build --aot

Nb : avec l'option --prod , ng build utilise par défaut --aot :

lancement avec --aot implicite :

ng build --prod

Effets/comportements :

	avec jit	avec aot
bundle .js construit	comporte le code de "jit" pour transformer ".html" juste avant exécution	ne comporte pas "jit" mais le ".js" déjà construit à partir des ".html"
temps de compilation	assez rapide	beaucoup plus lent
temps d'exécution	moyen	plus rapide
taille des bundles à télécharger	gros	petit

aot offre également plus de sécurité (via à vis de l'injection de code ".js" rendue plus difficile) .

Restrictions (rigueurs ajoutées) par "aot" :

La compilation en mode "aot" des templates ".html" s'effectue de manière rigoureuse (avec quelques restrictions "typescript") . Voir éventuellement la documentation officielle (<https://angular.io/guide/aot-compiler>) pour approfondir le sujet.

Beaucoup de petites erreurs (de cohérence ".html" / ".ts") passées comme inaperçues lors du développement ordinaire (avec **ng serve) étaient alors révélées lors d'un lancement de **ng serve --aot** ou bien **ng build --prod**** . C'était alors le moment de peaufiner encore un peu le code de l'application.

3. ivy (à partir de angular 9)

En version "preview" au sein de angular8 , intégré dans angular 9 et 10 et bien optimisé dans les versions ultérieures (11,12,13,14, 15, ...) , **ivy** est le nom de code du **nouveau moteur de compilation et de rendu d'Angular** .

Principaux apports de ivy

- **compilation plus rapide en aot** (*gain d'environ 40%*)
- **poids des bundles "js" générés réduit d'environ 15 %** (*grâce au "Tree shaking"*) .
- **plus de rapidité au niveau des tests**
- **debug plus facile** (*car code généré plus clair*)

Impacts de Ivy sur le développement "angular" (v9, v10, ...)

- certaines fonctionnalités avancées (angular-universal, ...) ont mis du temps à s'adapter à ivy (utiliser les versions les plus récentes possibles)
- réels gains de tous les cotés en production
- étant donné que le temps de compilation "aot" a été réduit de 40 % , le "**ng serve**" des **versions 9 et 10 d'angular utilise maintenant "aot" par défaut plutôt de "jit"** .
Avantages : moins d'incohérences laissées inaperçues , compilation plus rigoureuse
Inconvénients : "**ng serve**" **plus lent** (*surtout au premier lancement*) et un peu plus besoin de arrêter et relancer "ng serve" suite à des modifications importantes de l'application.

Depuis v9 , "aot" par défaut (avec **ng serve** et avec **ng build**) .

Pour un lancement *quelquefois* un petit plus rapide (en mode "*jit*") de "ng serve" en V9, v10 , il faut lancer :

```
ng serve --aot=false
```

4. Vite

A partir des versions 16 et 17 , Angular-CLI utilise en interne la technologie Vite (avec rollup) plutôt que webpack . Ceci permet entre autre d'obtenir un gain en performance (compilation plus rapide, démarrage plus rapide que l'ancien ng serve , ...)

5. Mise en production d'une application angular

NB : l'ancienne option `--prod` n'existe plus sur `ng build`, elle est *maintenant implicite (déclenchée d'office)*.

ng build génère des fichiers dans le répertoire `my-app/dist/my-app` (`dist` comme "à distribuer")

Selon la version d'angular, le contenu du répertoire `my-app/dist/my-app` après la commande "`ng build`" est à peu près le suivant:

Nom	Modifié le	Type	Taille
assets	31/12/2022 11:01	Dossier de fichiers	
3rdpartylicenses.txt	31/12/2022 11:01	Document texte	22 Ko
favicon.ico	07/06/2022 18:49	Fichier ICO	1 Ko
index.html	31/12/2022 11:01	Firefox HTML Doc...	4 Ko
main.6d795a0e21eff236.js	31/12/2022 11:01	JSFile	279 Ko
polyfills.e9d7fa7bfa9afaf3.js	31/12/2022 11:01	JSFile	33 Ko
runtime.397c3874548e84cd.js	31/12/2022 11:01	JSFile	2 Ko
styles.38c9f60f2e08c552.css	31/12/2022 11:01	Fichier source CSS	249 Ko

NB :

- **le code généré dans le répertoire dist ne fonctionne qu'avec un accès "http" (pas file:).**
- **il est possible de recopier le code du répertoire "dist" vers le répertoire d'une application simple nodeJs/express pour effectuer une sorte de mixage compatible** (code nodeJs/express pour WS-REST et code "angular" recopié dans sous répertoire "**front-end**" servi statiquement par nodeJs/express via `app.use(express.static('front-end'))` ;
- Une mise en production évoluée passera souvent par l'utilisation d'un **véritable serveur http** (tel que **apache 2.x ou nginx**) . On pourra déposer le code "static" angular à cet endroit et configurer des "reverse-proxy" vers des WS-REST java ou php ou nodeJs/express .

Pour effectuer des petits tests :

ng build

npm install -g http-server

http-server --proxy http://localhost:8282 -c-1 dist/my-app

http-server est un petit serveur http (comme lite-server).

Son option `-c-1` permet de désactiver les caches. Son url est par défaut <http://localhost:8080>

Via l'option `--proxy`, ce petit serveur peut (en tant que reverse proxy) rediriger toutes les requêtes qu'il ne sait pas gérer tout seul vers un serveur en arrière plan (ex : backend nodeJs/express ou springBoot d'url <http://localhost:8282>).

Eventuelles adaptations pour les versions récentes d'angular (17,18,...) :

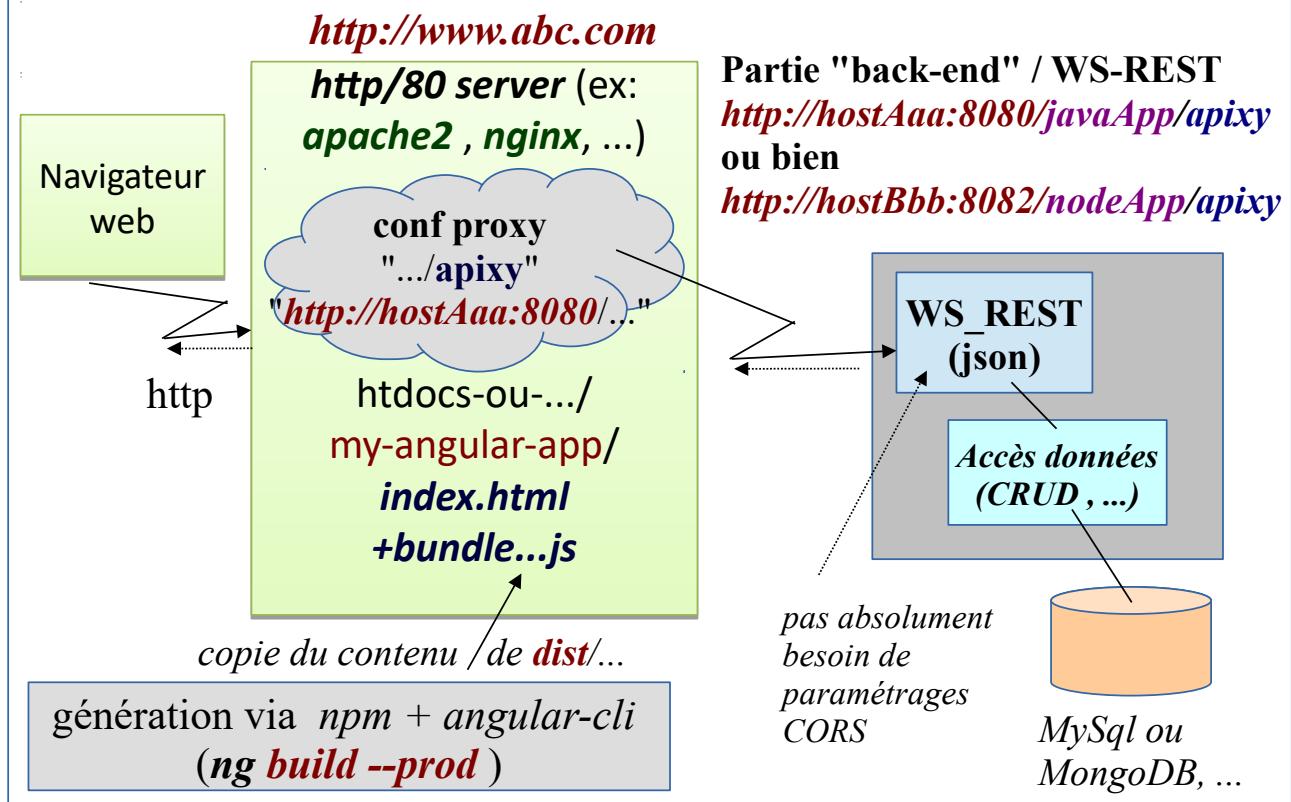
Une application angular récente peut éventuellement comporter le mode "ssr" (proposé dès la création d'une nouvelle appli avec **ng new**).

Dans ce cas, pour générer un build sans ssr ni rendu, il faut lancer ng build avec les options suivantes : **ng build --ssr=false --prerender=false**

Et d'autre part, le répertoire généré peut s'appeler /dist/my-app/browser plutôt que /dist/my-app.

5.1. Déploiement dans un serveur web/http :

Environnement de prod compatible Angular (v2,v4,...)



5.2. Configuration nginx pour angular et redirection WS-REST.

Le démarrage de **nginx** sous windows s'effectue sans aucune option (*double click sur nginx.exe*)
L'arrêt du serveur s'effectue via **nginx -s stop**

répertoire par défaut des pages statiques : **html** (ou l'on peut placer un sous répertoire my-angular-app)

configuration par défaut (sous windows) : **conf/nginx.conf** (avec copie de sauvegarde conseillée dans *original.nginx.conf.txt*)

Configuration importante au sein de **conf/nginx.conf** :

```
server {  
    listen      80;  
    server_name localhost;  
  
    # NB1 : dans nginx.conf, l'ordre des règles est important  
    # et selon ~ ou = ou ^~ les autres règles sont utilisées ou pas  
    # NB2 : les "location" sont exprimés avec des expressions  
    # régulières ^AuDebut , aLaFin$ , contenu de( ) récupéré par $1,...  
  
    # REMARQUE IMPORTANTE pour APPLI ANGULAR:  
    # on peut déposer le code d'une appli angular (contenu du répertoire "dist")  
    # dans un sous répertoire "my-angular-app" ou "appxy" de nginx/html  
    # que si la valeur de <base href="/"> est ajustée en <base href="/appxy/"> ou ...  
  
    # proxy mvc/api part of my-java-app to tomcat on 127.0.0.1:8080  
    # virtually seen as a "api" subpart of /my-angular-app (in html)  
  
    location ~ ^/my-angular-app/api/(.*){  
        proxy_pass http://127.0.0.1:8080/my-java-app/mvc/api/$1?$args;  
    }  
  
    # proxy minibank api part of nodeJs app to 127.0.0.1:8282  
    # virtually seen as a "minibank" subpart of /appxy (in html)  
  
    location ~ ^/appxy/minibank/(.*){  
        proxy_pass http://127.0.0.1:8282/minibank/$1?$args;  
    }  
  
    location /{  
        root  html;  
        index index.html index.htm;  
    }  
}
```

Attention, Attention : il faut absolument placer 127.0.0.1 (ou autre ip ou) dans la config mais pas localhost pour que ça fonctionne partout (sous windows , sous linux, ...)!!!!

6. Application isomorphe et optimisation SSR

Autrefois appelée "**angular-universal**" et maintenant rebaptisée "ssr" dans les versions 17+, l'extension "ssr" pour angular permet d'**obtenir un démarrage plus rapide ainsi qu'un bien meilleur référencement naturel ou SEO (Search Engine Optimization)**.

La nouvelle version "ssr" est plus élaborée/performante et elle est basée sur de l'hydratation.

6.1. Objectif de "ssr"

Pour être visité par un grand nombre d'utilisateurs, un site web se doit de remplir deux conditions essentielles.

- La page d'accueil soit s'afficher le plus rapidement possible.
- L'application angular doit être bien référencé par les moteurs de recherche.

La technique qui permet de le faire porte un nom.

- Le **Rendu Côté Serveur** ou **Server Side Rendering (SSR)** en anglais.

Les moteurs de recherche ont à l'heure actuelle du mal à interpréter le javascript.

Par défaut une application angular est de type "SPA (Single Page Application)" et "CSR" (Client Side Rendering) . Le titre principal de la page d'accueil est caché dans un bundle javascript produit par "ng build" mais n'apparaît pas dans index.html .

--> On a donc besoin d'ajouter un mécanisme SSR pour obtenir un bon référencement .

On ne peut cependant pas remettre en cause tous la logique de fonctionnement "SPA/CSR" d'angular qui fonctionne très bien pour naviguer rapidement d'un sous composant à l'autre et qui offre un très bon comportement dynamique (réactions rapides).

"Angular SSR" est en fait une technologie dite d'**isomorphisme** :

Au lieu de n'utiliser qu'un rendu coté serveur , "Angular SSR" va servir à générer coté serveur une sorte de pré-rendu "HTML/CSS" immédiatement/rapidement affichable par un navigateur de la page principale , son entête , son pied de page et son sous composant d'accueil (ex : WelcomeComponent) .

Le "pré-rendu" SSR n'est qu'un point de départ de l'application (très rapidement téléchargé et affiché) . Les mécanismes "javascript/SPA/CSR" sont téléchargés et chargés en mémoire en tâche de fond (le temps que l'utilisateur passe à admirer le pré-rendu affiché) .

Par la suite, les mécanismes SPA/CSR prennent le relai et l'application angular fonctionne comme d'habitude en dynamique javascript coté client et sans SSR.

Depuis la version 16 d'angular , cette prise de relai a été beaucoup optimisée et elle passe par une "**Hydration**" "javascript/eventListener" des pages statiques préalablement pré-rendues.

Ce "pré-rendu" SSR est en tout point identique à celui qui serait générer en pur javascript par les mécanismes habituels CSR , on parle alors en terme d'**isomorphisme** : même forme prise par les rendus SSR et CSR .

6.2. Positionnement de angular-ssr

La technologie angular-ssr est à la fois :

- liée au projet angular (add-on)
- vouée à être exécutée coté serveur via **nodeJs/express**

6.3. intégration de angular ssr

Au sein d'un projet existant angular , on lancera la commande suivante de façon à ajouter la fonctionnalité "ssr" :

ng add @nguniversal/express-engine pour anciennes versions

ng add @angular/ssr depuis angular 17+

Cette commande va ajouter certains fichiers et modifier certains fichiers existants de l'application angular. Il se peut que l'extension ssr soit déjà ajoutée dès la création de l'application (proposition de ng new).

6.4. Utilisation de Angular -SSR

npm run build (alias npm pour "**ng build**")

génère la sous arborescence suivante :

dist/my-app/browser/index-csr.html etjs (partie habituellement générée par ng build)

dist/my-app/server/server.mjs (partie supplémentaire générée en mode ssr)

En mode "extension ssr activée" , la fonctionnalité **"Static Site Generation"** (SSG) ou **"Prerendering"** génère directement (dans le répertoire *dist/my-app/browser*) des versions **".html sans javascript"** pour la page d'accueil et les principales routes "non lazy" .

Lancement direct de l'application angular générée en mode "ssr":

npm run serve:ssr:my-app

ou bien (directement sans un des alias de package.json) :

node dist/my-app/server/server.mjs

6.5. Pièges à éviter en mode SSR :

- Le rendu "SSR" (coté serveur "sans navigateur") n'est pas capable d'utiliser les objets globaux qui n'existeront que plus tard au runtime dans votre navigateur comme *document*, *window*, *localStorage*, etc.

Fausse solution : désactiver "ssr" (ok que si on ne veut pas s'en servir).

On pourrait ajouter "ssr": false, "prerender": false

dans angular.json près des lignes 72,73

```
"development": {
  "optimization": false,
  "extractLicenses": false,
  "sourceMap": true,
  "ssr": false,
  "prerender": false
}
```

Bonne solution : Tester le contexte (CSR ou SSR) avant d'utiliser un objet global du navigateur (document ou localStorage, ...) via un bloc de code de ce genre :

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

export class XyzComponent {
  private readonly platform = inject(PLATFORM_ID);
  private readonly document = inject(DOCUMENT);

  constructor() {
    if (isPlatformBrowser(this.platform)) {
      console.warn("browser / csr");
      // Safe to use document, window, localStorage, etc. :-)
    }

    if (isPlatformServer(this.platform)) {
      console.warn("server / ssr");
      // Not smart to use document here, however, we can inject it ;-)
    }
  }
}
```

Points à principalement encadrer comme cela :

- *localStorage.setItem(..., ...)* et *.getItem(...)*
- appels automatiques de WS-REST en mode GET au sein de *ngOnInit()* ou autres

ANNEXES

XVII - Annexe – composants avancés et animations

1. Composants "angular" avancés et animations

1.1. Cycle de vie précis d'un composant angular :

Interfaces (à facultativement implémenter)	Méthodes (une par interface)	Moment où la méthode est appelée automatiquement par angular2
(1) OnChanges	ngOnChanges()	dès changement de valeur d'un "input binding" (exemple : "propriété initialisée selon niveau parent")
(2) OnInit	ngOnInit()	à l'initialisation du composant et après les premiers éventuels ngOnChanges() et après constructeur et injections
(3) DoCheck	ngDoCheck()	permet éventuellement d'indiquer des changements si Angular ne les a pas détecté (avant un futur réaffichage)
(4) AfterContentInit	ngAfterContentInit()	est déclenchée à l'initialisation après la projection de contenu (*) (après ngOnInit() et avant ngAfterViewInit())
(5) AfterContentChecked	ngAfterContentChecked()	déclenchée après la détection de changement dans le contenu projeté
(6) AfterViewInit	ngAfterViewInit()	déclenchée après l'initialisation de la vue du composant et après l'initialisation des vues des composants enfants.
(7) AfterViewChecked	ngAfterViewChecked()	est déclenchée après détection d'un changement dans la vue du composant et dans les vues des composants enfants.
(8) OnDestroy	ngOnDestroy()	juste avant destruction d'un composant

Ces méthodes sont appelées dans l'ordre 1,2,...7 lors de l'initialisation et du premier affichage d'un composant . Par la suite seules certaines méthodes seront redéclenchées après une détection de valeurs modifiées (ngOnChanges() , ngDoCheck() , ngAfterContentChecked() , ngAfterViewChecked())

(*) NB : on appelle "**projection de contenu**" la prise en compte (au niveau d'un sous composant) d'un contenu (choisi par le parent) imbriqué entre début et fin de balise du sous composant

exemple :

```
<mat-tab-group>
  <mat-tab label="calcul tva">
    <app-tva></app-tva> <!-- contenu projeté dans composant mat-tab -->
```

```
</mat-tab>
```

```
...
```

NB : La potentielle erreur "**ExpressionChangedAfterItHasBeenCheckedError**" est déclenchée en développement seulement pour indiquer qu'une erreur de conception risque de compromettre la détection de changement et empêcher la mise à jour correcte de la vue. (ex : this.updatedValue = 'new_value'; n'est pas à faire dans ngAfterContentChecked())

Pour éviter l'erreur précédente , il suffit souvent d'englober les instructions à problème au sein d'un bloc **setTimeout(()=>{....},1)** ;

Pour approfondir le sujet (mécanismes internes de angular) :

<https://cdiese.fr/angular-change-detection/>

ngDoCheck() n'est à coder que dans des cas ultra-pointus (en s'appuyant sur **KeyValueDiffers** ou **IterableDiffers**)

Lien pour approfondir le sujet (si nécessaire) :

<https://www.concretewebpage.com/angular/angular-ngdocheck>

1.2. Détails sur **ngOnChanges()**:

Cette *callback* est exécutée si le composant contient des propriétés en entrée (notamment avec le décorateur **@Input()**).

Cette *callback* peut être implémentée avec un argument de type **SimpleChanges**:

```
void ngOnChanges(changes: SimpleChanges): void {}
```

Elle sera alors déclenchée une seule fois (globalement pour toutes les propriétés modifiées)

Exemple :

```
ngOnChanges(changes: SimpleChanges){  
  console.log( "changes=" + JSON.stringify(changes));  
}
```

--> premier appel (avec valeurs initiales choisies par le parent) :

```
changes={"compteur":{"currentValue":1,"firstChange":true},  
         "message":{"currentValue":"msg","firstChange":true},  
         "donnees":{"currentValue":{"num":1,"label":"label"},"firstChange":true}}
```

--> appels ultérieurs (suite à changements opérés par le parent) :

```
changes={"compteur":{"previousValue":1,"currentValue":2,"firstChange":false}}
```

puis

```
changes={"message":{"previousValue":"msg","currentValue":"msg*","firstChange":false}}
```


@Input et ngOnChanges() selon modifications effectuées

template d'un composant parent :

```
counter : {{counter}} <button (click)="counter=counter+1">++</button> ,
msg : {{msg}} <button (click)="msg=msg+'*'">++</button> ,
data : ({{data.num}}) <button (click)="data.num=data.num+1">++</button> ,
    {{data.label}} <button (click)="data.label=data.label+'*'">++</button>
)
<my-child [compteur]="counter" [message]="msg" [donnees]="data"> </my-child>
```

Affichage initial :

values in this (parent) component: counter : 1 ++ , msg : msg ++ , data : (1 ++ , label ++)

with 3 @input/property links / [compteur]="counter" [message]="msg" [donnees]="data" :

values in this (child) component: compteur : 1 ++ , message : msg ++ , donnees : (1 ++ , label ++)

Après modifications effectuées sur le parent (et automatiquement répercutées sur "child" avec appels de ngOnChanges() sur "child") :

values in this (parent) component: counter : 2 ++ , msg : msg* ++ , data : (1 ++ , label ++)

with 3 @input/property links / [compteur]="counter" [message]="msg" [donnees]="data" :

values in this (child) component: compteur : 2 ++ , message : msg* ++ , donnees : (1 ++ , label ++)

Après modifications effectuées uniquement sur le sous composant "child" .

pas de remontées de modifications vers le parent tant que pas de @Output / event

values in this (parent) component: counter : 2 ++ , msg : msg* ++ , data : (1 ++ , label ++)

with 3 @input/property links / [compteur]="counter" [message]="msg" [donnees]="data" :

values in this (child) component: compteur : 4 ++ , message : msg*** ++ , donnees : (1 ++ , label ++)

Dans cet exemple l'objet data (comportant lui même des sous parties modifiables) est initialement passé par référence (via **[donnees]="data"**).

Cet objet de données (se retrouvant partagé entre les composants parent et enfant) conduit à un affichage automatiquement synchronisé dans les 2 sens (sans aucun appel à ngOnChanges() tant que l'on modifie les sous parties de cet objet sans changer l'objet data lui même).

values in this (parent) component: counter : 2 ++ , msg : msg* ++ , data : (5 ++ , label*** ++)

with 3 @input/property links / [compteur]="counter" [message]="msg" [donnees]="data" :

values in this (child) component: compteur : 4 ++ , message : msg*** ++ , donnees : (5 ++ , label*** ++)

2. ng-content et ng-template

2.1. Projection des éléments imbriqués

Un composant angular peut (en tant que nouvelle balise telle que *toggle-panel* ici) , incorporer à son tour certains sous éléments (ex : <div ...> ou autre sous-sous composant angular).

Exemple :

```
<toggle-panel title="panel1">
  <app-part1></app-part1> <!-- ou ... , vu comme ng-content dans toggle-panel -->
</toggle-panel>

<toggle-panel title="panel2">
  <div>contenu du panneau 2</div> <!-- vu comme ng-content dans toggle-panel....html -->
</toggle-panel>
```

Le (ou le paquet de) sous-composant(s)/sous-balises imbriqué au niveau de l'utilisation d'un composant réutilisable sera vu via la balise spéciale **<ng-content></ng-content>** au sein du template HTML (code interne) de ce composant.

Cette fonctionnalité était appelée "transclusion" au sein des directives "angular Js 1.x" , elle est maintenant appelée "**projection de contenu**" au sein des composants angular 2+ .

Exemple concret : composant réutilisable "togglePanel" basé sur des styles *bootstrap* et fontes *bootstrap-icons* :

```
import { Component, Input } from '@angular/core';
@Component({
  selector: 'toggle-panel',
  templateUrl: './toggle-panel.component.html' , styleUrls: ['./toggle-panel.component.scss']
})
export class TogglePanelComponent {
  toggleP /* : boolean */ =false;
  @Input()
  title /* : string */ = 'default panel title';
  constructor() { }
}
```

version du template html basée sur les classes css bootstrap 5 :

```
<div class="card mb-2">
  <div class="card-header text-white bg-primary" (click)="toggleP = !toggleP" >
    {{title}} <i [class.bi-arrow-down-circle-fill]="!toggleP"
      [class.bi-arrow-up-circle-fill]="toggleP" ></i>
  </div>
  <div class="card-body collapse" [class.show]="toggleP">
    <ng-content></ng-content> <!-- remplacé par le contenu imbriqué -->
  </div>
</div>
```

version du template html basée sur les classes css spécifiques :

```
<div class="my-card">
  <h4 class="my-card-header my-bg-primary">
    <a class="my-text-light" (click)="toggleP = !toggleP" >
      <span class="my-icon" [style.display]="toggleP?'none':'inline-block'">+</span>
      <span class="my-icon" [style.display]="toggleP?'inline-block':'none'">-</span>{{title}}
    </a>
  </h4>
  <div class="my-card-body my-collapse" [class.my-show]="toggleP">
    <ng-content></ng-content> <!-- remplacé par le contenu imbriqué -->
  </div>
</div>
```

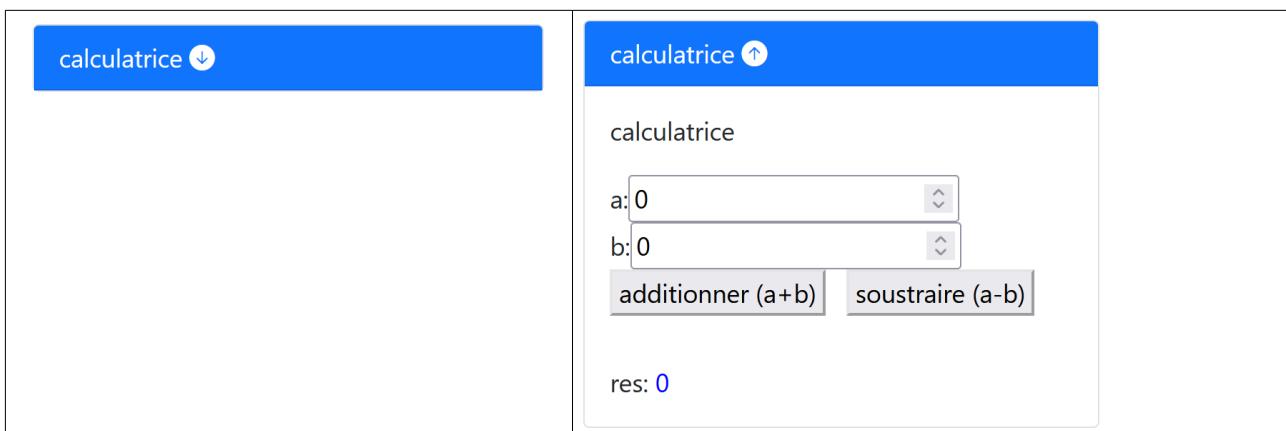
toggle-panel.component.scss

```
.my-card { margin-top: 0.1em; margin-bottom: 0.1em; }
.my-card-header { border-top-left-radius: 0.3em; border-top-right-radius: 0.3em;
  padding: 0.1em; margin-bottom: 0px; }
a { text-decoration: none; }
.my-card-body {border: 0.1em solid blue; border-bottom-left-radius: 0.3em;
  border-bottom-right-radius: 0.3em; padding: 0.2em; }
.my-bg-primary { background-color: blue; }
.my-text-light { color : white; }
.my-icon { color : blue; background-color: white; margin: 0.2em;
  padding-left: 0.2em; padding-right : 0.2em;
  min-width: 1em; font-weight: bold; }
.my-collapse { display : none; }
.my-show { display : block ; }
```

Exemple d'utilisation :

```
<toggle-panel title="calculatrice">
  <app-calculatrice></app-calculatrice> <!-- ou ... , vu comme ng-content dans toggle-panel -->
</toggle-panel>

<toggle-panel title="calcul tva">
  <app-tva></app-tva> <!-- vu comme ng-content dans toggle-panel....html -->
</toggle-panel>
```



2.2. ng-template

Au sein d'un template HTML angular, un bloc `<ng-template #nomTemplate>...</ng-template>` est un bloc qui n'est pas affiché par défaut et qui le sera potentiellement par la suite.

```
...
export class WithNgTemplateComponent {
  ready=false;
...
}
```

```
<input type="checkbox" [(ngModel)]="ready" />ready <br/>
<!-- *ngIf="booleanExpression else alternativeTemplate" -->
<div *ngIf="ready else notReady">
  ready / done !
</div>

<ng-template #notReady>
  waiting (not ready)
</ng-template>
```

ready
ready / done !

ready
waiting (not ready)

Affichage via ng-container et *ngTemplateOutlet

```
<!-- ng-container peut être vu comme une div virtuelle avec possibilité de *ngIf="..." ou autre sans niveau d'imbrication supplémentaire -->

<ng-template #loading>
  <p>loading...</p>
</ng-template>

<ng-template #details>
  <p>details...</p>
</ng-template>

<ng-container *ngTemplateOutlet="loading"></ng-container>
<!--affiche à cet endroit le template #loading-->
```

ng-template avec contexte :

```
....  
export class WithNgTemplateComponent {  
...  
  ctxSemaine = {code: 1, label: "semaine"};  
  ctxWeekEnd = {code: 2, label: "weekend"};  
}
```

```
<h4>templateAvecContexte</h4>  
<ng-template #templateAvecContexte let-code="code" let-label="label">  
  <!-- via let-code="code" au sein de ce template la variable code  
       aura la valeur de la sous partie ".code" d'un objet javascript  
       interchangeable appelé context -->  
  <div> code={{code}} label={{label}}</div>  
</ng-template>  
  
<ng-container *ngTemplateOutlet="templateAvecContexte;context:ctxSemaine">  
</ng-container>  
  
<ng-container *ngTemplateOutlet="templateAvecContexte;context:ctxWeekEnd">  
</ng-container>
```

templateAvecContexte

code=1 label=semaine

code=2 label=weekend

ng-template avec TemplateRef :

```
import { Component, Input, TemplateRef } from '@angular/core';
...
export class PanelWithTemplateRefComponent {
  @Input()
  headerTemplate: TemplateRef<any> | undefined;

  @Input()
  footerTemplate: TemplateRef<any> | undefined;
}
```

panel-with-template-ref.component.html

```
<ng-template #defaultHeader>
  <div>defaultHeader</div>
</ng-template>

<ng-template #defaultFooter>
  <div>defaultFooter</div>
</ng-template>

<div class="withBorder">
  <ng-container *ngTemplateOutlet="headerTemplate?headerTemplate:defaultHeader"></ng-container>
  <ng-content></ng-content>
  <ng-container *ngTemplateOutlet="footerTemplate?footerTemplate:defaultFooter"></ng-container>
</div>
```

```
<h4>sous composant avec TemplateRef</h4>
<app-panel-with-template-ref>
  panel with defaultHeader and defaultFooter
</app-panel-with-template-ref>
<br/>
<app-panel-with-template-ref [headerTemplate]="details" [footerTemplate]="loading">
  panel with details template as Header and loading template as Footer
</app-panel-with-template-ref>
```

sous composant avec TemplateRef

```
defaultHeader
panel with defaultHeader and defaultFooter
defaultFooter
```

```
details...
```

```
panel with details template as Header and loading template as Footer
loading...
```

3. ViewChild / ContentChild

Vocabulaire (issu de la norme "web component") :

Le **Shadow DOM** - est un DOM interne de votre composant défini par vous (en tant que créateur du composant) et caché de l'utilisateur final. Par exemple:

```
@Component({
  selector: 'some-component',
  template: `
    <h1>I am Shadow DOM!</h1>
    <h2>Nice to meet you :)</h2>
    <ng-content></ng-content>
  `
})
class SomeComponent { /* ... */ }
```

Le **Light DOM** - est un DOM qui est un utilisateur final de votre composant . Par exemple:

```
@Component({
  selector: 'another-component',
  directives: [SomeComponent],
  template: `
    <some-component>
      <h1>Hi! I am Light DOM!</h1>
      <h2>So happy to see you!</h2>
    </some-component>
  `
})
class AnotherComponent { /* ... */ }
```

NB :

@ViewChild et @ViewChildren recherchent des éléments dans **Shadow DOM**
(autrement dit dans la structure du template du composant actuel)

tandis que

@ContentChild et @ContentChildren les recherchent dans **Light DOM**
(autrement dit dans le contenu qui sera projeté via le composant parent)

3.1. Exemple avec @ViewChild et @ViewChildren

a.child.component.ts

```
... @Component({
  selector: 'app-a-child', ...
})
export class AChildComponent {
  @Input()
  name : string ="aChild";

  public sayHello() :string{
    return "hello_from_"+this.name;
  }
...
}
```

a.child.component.html

```
<p>a-child with name={{name}}</p>
```

with.child.component.html

```
<h3>with-child works!</h3>
message (from child 1 and 2)={{message}} <br/>
message (from all children)={{messageV2}}
<hr/>
<app-a-child name="child1"></app-a-child>
<app-a-child #c2 name="child2"></app-a-child>
<app-a-child name="child3"></app-a-child>
<p #p1><b>p1</b></p>
<p #p2><i>p2</i></p>
<p #p3>p3</p>
```

with-child works!

message (from child 1 and 2)=hello_from_child1 hello_from_child2
 message (from all children)= hello_from_child1 hello_from_child2 hello_from_child3 p1 p2 p3

a-child with name=child1

a-child with name=child2

a-child with name=child3

p1

p2

p3

with.child.component.ts

```
import { Component, OnInit, ViewChild, ViewChildren,
         QueryList, ElementRef } from '@angular/core';
import { AChildComponent } from './a-child/a-child.component';

@Component({
  selector: 'app-with-child',
  templateUrl: './with-child.component.html',
  styleUrls: ['./with-child.component.scss']
})
export class WithChildComponent {

  message:string="";
  messageV2:string="";

  @ViewChild(AChildComponent)
  first_achild!: AChildComponent; // refer to first subcomponent of type AChildComponent

  @ViewChild('c2', { static: true })
  achild_c2!: AChildComponent; // refer to subcomponent/subElement with #c2 in .html
```

```

@ViewChildren(AChildComponent)
allchilds!: QueryList<AChildComponent>; // refer to subcomponents of type AChildComponent

@ViewChildren('p', { read: ElementRef }) pList!: QueryList<ElementRef>; // all elt with #p
@ViewChild('p1', { read: ElementRef }) p1!: ElementRef; // elt with #p1

ngAfterViewInit() {
    // call method .sayHello() of child sub component :
    let msgChild1= this.first_achild?this.first_achild.sayHello():"";
    let msgChild2= this.achild_c2?this.achild_c2.sayHello():"";
    let allMsg = msgChild1 + " " + msgChild2;

    let allMsgV2 = "";
    for(let c of this.allchilds){
        allMsgV2 = allMsgV2+" " + c.sayHello();
    }

    for(let p of this.pList){
        allMsgV2 = allMsgV2+" " + p.nativeElement.innerText;
    }

    //this.message= allMsg;//-->ExpressionChangedAfterItHasBeenCheckedError"
    setTimeout(()=> {
        this.message=allMsg;
        this.messageV2=allMsgV2;
    }, 0);
    console.log(allMsg);
    console.log("in p1:" + this.p1.nativeElement.innerHTML);
}
...
}

```

3.2. Exemple avec @ContentChild et @ContentChildren

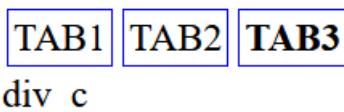
with-content.component.html

```

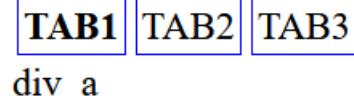
<p>with-content works!</p>
<app-my-tabset>
    <app-my-tab title="tab1">
        <div #tabContent>div_a</div>
    </app-my-tab>
    <app-my-tab title="tab2">
        <div #tabContent>div_b</div>
    </app-my-tab>
    <app-my-tab title="tab3">
        <div #tabContent>div_c</div>
    </app-my-tab>
</app-my-tabset>

```

with-content works!



with-content works!



my-tab.component.html

```
<span class="aTab" (click)="onClick()" [class.selectedTab]="selected" >{{title}}</span>
```

my-tab.component.css

```
.aTab { display: inline-block; margin : 2px; padding: 2px;
        border-style: solid; border-color: blue; border-width: 1px; }
.selectedTab { font-weight: bold; background-color: lightgrey; }
```

my-tab.component.ts

```
import { Component, OnInit, Input, AfterContentInit, ContentChild,
         ElementRef, EventEmitter, Output } from '@angular/core';
@Component({
  selector: 'app-my-tab',
  templateUrl: './my-tab.component.html',
  styleUrls: ['./my-tab.component.scss']
})
export class MyTabComponent implements AfterContentInit {
  @Input() title:string="defaultTabTitle";
  @Input() selected:boolean=false;
  @Output()
  public selectionChange : EventEmitter<{value:MyTabComponent}> =
    new EventEmitter<{value:MyTabComponent}>();
  @ContentChild("tabContent")
  contentElementRef! : ElementRef; // refer to elt with #tab_content in projected content
  onClick(){
    this.selected=!this.selected;
    if(this.selected)
      this.selectionChange.emit({value:this});
  }
  constructor() { }
  ngAfterContentInit(): void {
    console.log(this.contentElementRef.nativeElement.innerText)
  }
  public tabContentAsString(){
    if(this.contentElementRef==undefined) return "";
    else return this.contentElementRef.nativeElement.innerText
  }
}
```

my-tabset.component.html

```
<ng-content></ng-content> <!-- will display tabs titles-->
<br/>
{{selectedTabContent}}
```

my-tabset.component.ts

```
import { Component, OnInit, ContentChildren, QueryList, AfterContentInit } from '@angular/core';
import { MyTabComponent } from './my-tab/my-tab.component';

@Component({
  selector: 'app-my-tabset',
  templateUrl: './my-tabset.component.html',
  styleUrls: ['./my-tabset.component.scss']
})
export class MyTabsetComponent implements OnInit, AfterContentInit {

  selectedTabContent : string ="";

  @ContentChildren(MyTabComponent)
  tabs: QueryList<MyTabComponent>

  onSelectionChange(newSelectedTab : MyTabComponent ){
    this.tabs.forEach(tab => { tab.selected=(tab==newSelectedTab);
      if(tab == newSelectedTab )
        this.selectedTabContent = newSelectedTab.tabContentAsString();
    })
  }

  ngAfterContentInit() {
    let lastTab = this.tabs.last;
    this.tabs.forEach(tab => { console.log(tab.title);
      tab.selectionChange.subscribe(
        (evt)=>{ this.onSelectionChange(evt.value); }
      );
      tab.title=tab.title.toUpperCase();
      tab.selected=(tab==lastTab); //last tab selected by default
    })
    this.selectedTabContent = lastTab.tabContentAsString();
  }
  ...
}
```

Cet exemple "rudimentaire" de composant "onglet" est bien évidemment améliorable . Il permet cependant de montrer ce qu'il est possible de faire avec **@ContentChildren** et **@ContentChild**.

En annexe (**ngx-bootstrap** , **angular-material**) , on trouvera des composants onglets prédefinis.

Version améliorée des onglets personnalisés avec TemplateRef :

my-tab-component.html

```
<span class="aTab" (click)="onClick()" [class.selectedTab]="selected">{{title}}</span>
<ng-template>
  <ng-content></ng-content>
</ng-template>
```

```
import { Component, OnInit, Input, AfterContentInit, ContentChild, ElementRef, EventEmitter, Output, TemplateRef, ContentChildren, ViewChild } from '@angular/core';
...
export class MyTabComponent implements OnInit {
  @Input() title:string="defaultTabTitle";
  @Input() selected:boolean=false;

  @Output()
  public selectionChange : EventEmitter<{value:MyTabComponent}>=
    new EventEmitter<{value:MyTabComponent}>();

  @ViewChild(TemplateRef)
  tabTemplate: TemplateRef<any> | undefined ;

  onClick(){
    this.selected=!this.selected;
    if(this.selected)
      this.selectionChange.emit({value:this});
  }
  ngOnInit(): void { }
  constructor() { }

  public tabContentAsTemplateRef() : TemplateRef<any> | undefined{
    return this.tabTemplate;
  }
}
```

my-tab-set.component.html

```
<ng-template #defaultTabContent>
  <div>defaultTabContent</div>
</ng-template>

<ng-content></ng-content> <!-- will display tabs titles -->
<br/>
<ng-container
  *ngTemplateOutlet="selectedTabTemplateRef?selectedTabTemplateRef:defaultTabContent">
</ng-container>
```

my-tabset.component.ts

```
import { Component, OnInit, ContentChildren, QueryList, AfterContentInit, TemplateRef, AfterViewInit } from '@angular/core';
import { MyTabComponent } from './my-tab/my-tab.component';

...
export class MyTabsetComponent implements OnInit , AfterViewInit {

  selectedTabTemplateRef : TemplateRef<any> | undefined;
```

```

@ContentChildren(MyTabComponent)
tabs!: QueryList<MyTabComponent>

onSelectionChange(newSelectedTab : MyTabComponent ){
    this.tabs.forEach(tab => {
        tab.selected=(tab==newSelectedTab);
        if(tab == newSelectedTab )
            this.selectedTabTemplateRef = newSelectedTab.tabContentAsTemplateRef();
    })
}

ngAfterViewInit(){
    let lastTab = this.tabs.last;
    this.tabs.forEach(tab => { console.log(tab.title);
        tab.selectionChange.subscribe(
            (evt)=>{ this.onSelectionChange(evt.value); }
        );
        setTimeout(()=>{
            tab.selected=(tab==lastTab);
            //setTimeout to avoid error message "Expression has changed after it was checked"
            },1)
    })
    this.selectedTabTemplateRef = lastTab.tabContentAsTemplateRef();
}

constructor() { }
ngOnInit(): void { }
}

```

```

<app-my-tabset>
    <app-my-tab title="tab1" >
        <div>div_a</div>
    </app-my-tab>
    <app-my-tab title="tab2" >
        <div>div_b</div>
    </app-my-tab>
    <app-my-tab title="tab3">
        <div>div_c</div>
    </app-my-tab>
    <app-my-tab title="tab4" >
        <app-welcome></app-welcome>
    </app-my-tab>
</app-my-tabset>

```

tab1 **tab2** **tab3** **tab4**

welcome to exp-adv-app

4. Animations (triggers)

4.1. Animations css ordinaires (déclenchées via :hover ou ...)

Au sein d'un fichier css, une animation se configure en "keyframes" (ayant un nom logique).

On y définit les étapes de l'animation (par exemples à 0 %, 50 % , 100 % du temps) :

```
@keyframes monanimation
```

```
{  
 0% { transform: translateX(0px); }  
 50% { transform: translateX(40px) rotate(-15deg) ;}  
 100% { transform: translateX(80px) rotate(15deg); }  
}
```

```
@keyframes monrebond
```

```
{  
 0% { transform: translateY(0px) translateX(0px);}  
 50% { transform: translateY(8px) translateX(2px);}  
 100% { transform: translateY(0px) translateX(0px); }  
}
```

On applique ensuite une animation via la propriété "animation" :

```
a:hover { color: blue; transition: color 1s linear; animation : monrebond 1s;}
```

```
button:hover { color: blue; animation : monrebond 1s;}
```

```
p:hover { background-color: #93db83; /* vert*/; animation: monanimation 2s ; }
```

4.2. Animations "angular" déclenchées sur changement d'état

Une **animation angular** se code comme un **trigger réutilisable** ayant un nom logique (ex : `'changeDivSize'`) et étant généralement structuré par différents états (ex : "smaller", "normal").

Chaque état sera associé à une association de styles et transformations css (ex : scale , translation , ...) .

On définit également des transitions entre états via la syntaxe suivante
transition('état1=>état2', animate(...)) .

Par la suite , au sein d'un composant angular utilisant un trigger , une syntaxe spéciale
`[@nomTrigger]='nomEtatSelonValeursDuComposant'` permettra de déclencher une animation angular .

Exemple (avec effet de rétrécissement/grossissement):

`changeDivSizeTrigger.ts`

```
import { trigger, state, style, transition, animate } from '@angular/animations';
export const changeDivSizeTrigger =
  trigger('changeDivSize', [
    state('smaller', style({
      backgroundColor: 'lightgreen',
      transform: 'scale(0.9)'
    })),
    state('normal', style({
      backgroundColor: 'green',
      transform: 'scale(1.0)'
    })),
    transition('smaller=>normal', animate('800ms')),
    transition('normal=>smaller', animate('400ms'))
  ]);

/* exemple d'utilisation:
@Component({...,   animations: [ changeDivSizeTrigger , ... ]  })
<div [@changeDivSize]="!myToggleValue?'smaller':'normal">....</div>
<div [@changeDivSize]="taux<20?'smaller':'normal">essai animation
  avec [@xyTrigger]="stateXorY", taux={{taux}}</div>
*/
```

```
@changeDivSize    @fadeInOut    @enterLeave
 myToggleValue
```

essai animation avec [@xyTrigger] = "stateXorY"

changement de taille (grossissement) et changement de couleur progressifs .

myToggleValue

essai animation avec [@xyTrigger] = "stateXorY"

Exemple avec les alias "enter" et ":leave" :

NB : Au sein des définitions de transition entre état(s) ,

* signifie **n'importe quel état**

void signifie **aucun état connu** et (**void => ***) correspond donc à un **début quelconque**
et (* => **void**) correspond donc à une **fin quelconque**

:enter est un alias/raccourci pour **void => ***

et **:leave** est un alias/raccourci pour * => **void**

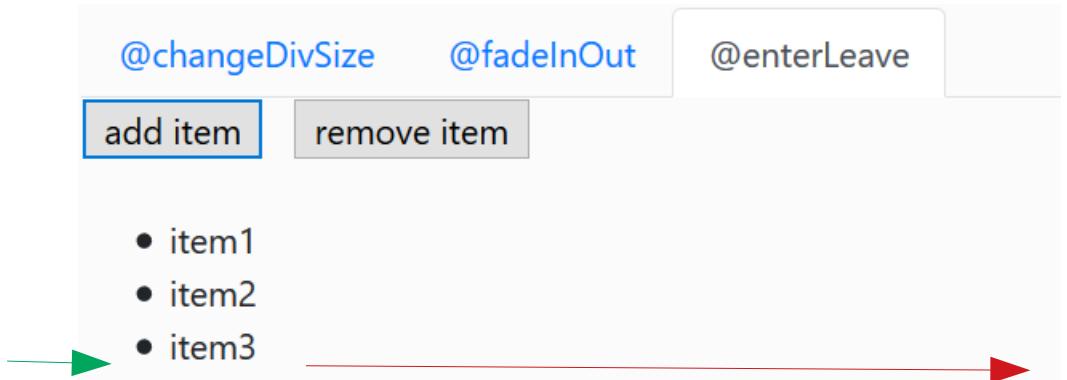
enterLeaveTrigger.ts

```
import { trigger, state, style, transition, animate } from '@angular/animations';

export const enterLeaveTrigger =
trigger('enterLeave', [
  state('flyIn', style({ transform: 'translateX(0)' })),
  transition(':enter', [
    style({ transform: 'translateX(-100%)' }),
    animate('0.5s 300ms ease-in')
  ]),
  transition(':leave', [
    animate('0.3s ease-out', style({ transform: 'translateX(100%)' }))
  ])
]);
```

Exemple d'utilisation :

```
<li *ngFor="let item of myList" [@enterLeave]="'flyIn' ">
  {{item}}
</li>
```



entrée (en glissade) sur la gauche et sortie en glissade saccadée sur la droite

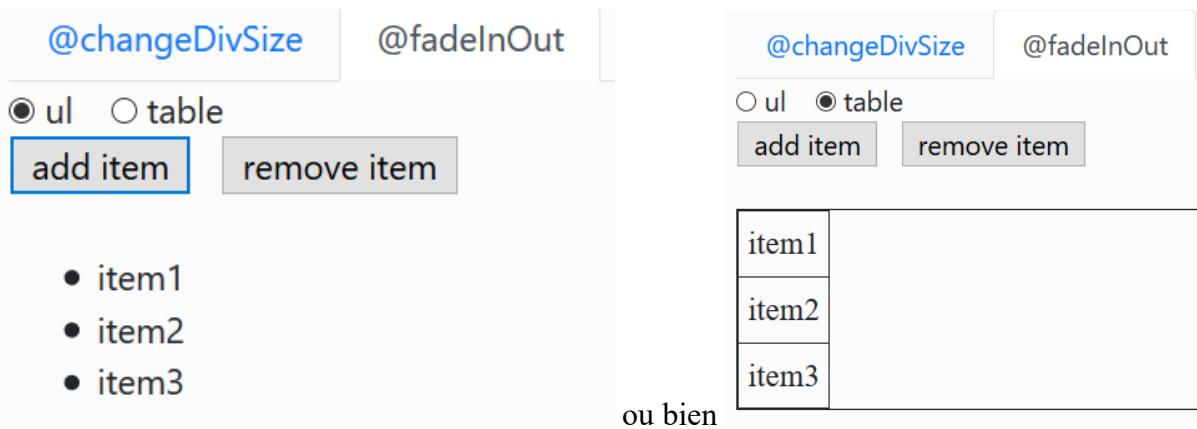
Autre exemple avec un seul état anonyme et début et fin indifférenciés :

fadeOutTrigger.ts

```
import { trigger, state, style, transition, animate } from '@angular/animations';

export const fadeInOutTrigger =
  trigger('fadeOut', [
    state('void', style({
      opacity: 0
    })),
    transition('void <=> *', animate(1000)),
  ]);

/* exemple d'utilisation:
@Component({...
  animations: [ fadeInOutTrigger , ... ]
})
*/
<li *ngFor="let listItem" [@fadeOut]>
  {{list}}
</li>
*/
```



apparition progressive d'un nouvel élément et disparition progressive d'un élément retiré .

Exemples d'utilisations détaillées :

WithAnimationsComponent.ts

```
import { Component, OnInit } from '@angular/core';
import { changeDivSizeTrigger } from 'src/app/common/animation/changeDivSizeTrigger';
import { enterLeaveTrigger } from 'src/app/common/animation/enterLeaveTrigger';
import { fadeInOutTrigger } from 'src/app/common/animation/fadeInOutTrigger';

@Component({
  selector: 'app-with-animations',
  templateUrl: './with-animations.component.html',
  styleUrls: ['./with-animations.component.scss'],
  animations: [ changeDivSizeTrigger , enterLeaveTrigger, fadeInOutTrigger ]
})
export class WithAnimationsComponent implements OnInit {

  myToggleValue : boolean = false;
  ulOrTable : string ="ul";
  myList : string[] = [ "item1" , "item2"];
  lastItemNumber : number = 2;

  onAddItem(){
    this.lastItemNumber++;
    this.myList.push(`item${this.lastItemNumber}`);
  }
}
```

```

onRemoveItem(){
  if(this.lastItemNumber>0){
    this.lastItemNumber--;
    this myList.splice(this.lastItemNumber,1);
  }
}

constructor() { }

ngOnInit() {
}
}

```

with-animations-component.html

```

<h3>with-animations</h3>
<mat-tab-group>
  <mat-tab label="@changeDivSize">
    <input type="checkbox" [(ngModel)]="myToggleValue" />myToggleValue<br/>
    <hr/>
    <div [@changeDivSize]="!myToggleValue?'smaller':'normal'">essai animation avec
    [@xyTrigger]="stateXorY"</div>
  </mat-tab>
  <mat-tab label="@fadeInOut">
    <input type="radio" name="ulOrTable" [(ngModel)]="ulOrTable" [value]="'ul'" /> ul
    &nbsp;&nbsp;
    <input type="radio" name="ulOrTable" [(ngModel)]="ulOrTable" [value]="'table'" /> table
    <br/>
    <input type="button" value="add item" (click)="onAddItem()" /> &nbsp;
    <input type="button" value="remove item" (click)="onRemoveItem()" /><br/>
    <br/>
    <ul [style.display]="ulOrTable=='ul'?'block':'none'">
      <li *ngFor="let item of myList" [@fadeInOut]>
        {{item}}
      </li>
    </ul>
    <table border="1" [style.display]="ulOrTable=='table'?'block':'none'">
      <tr *ngFor="let item of myList" [@fadeInOut]>
        <td>{{item}}</td>

```

```
</tr>
</table>
</mat-tab>
<mat-tab label="@enterLeave">
  <input type="button" value="add item" (click)="onAddItem()" /> &nbsp;
  <input type="button" value="remove item" (click)="onRemoveItem()" /><br/>
  <br/>
  <ul>
    <li *ngFor="let item of myList" [@enterLeave]="'flyIn' ">
      {{item}}
    </li>
  </ul>
</mat-tab>
</mat-tab-group>
```

XVIII - Annexe – internationalisation angular (i18n)

1. internationalisation (i18n)

$i18n = i + 18\text{caractères} + n$

1.1. extension nécessaire

Depuis la version 9 de angular, il faut ajouter l'extension @angular/localize

`ng add @angular/localize`

1.2. Mode opératoire

1. Par ajout de **i18n="..."** au sein des **templates html** , *marquer les zones ayant besoins d'être traduites* (*le texte original doit idéalement être en anglais*)
2. générer le fichier de traduction **src/locale/messages.xlf** via la commande
ng extract-i18n --output-path src/locale

1.3. paramétrage au niveau d'un template html

`<h1 i18n="main header|Friendly welcoming message">Welcome!</h1>`

`<h1 i18n="main header|Friendly welcoming message@@welcome">Welcome!</h1>`

i18n (attribut sans valeur , sans indication) demande simplement à effectuer une traduction du texte de la balise.

i18n="texte_description" demande une traduction en fonction d'une description

i18n="meaning | description" demande une traduction en fonction d'une signification et d'une description .

i18n="@@custom_translation_id" ou

i18n="description@@custom_translation_id" ou

i18n="meaning | description@@custom_translation_id" permet en plus de contrôler la valeur de l'id qui sera généré dans le fichier de traduction en lui affectant une valeur parlante/significative et non changeante (pas partiellement aléatoire).

Selon les paramétrages des fichiers de traduction, toutes des traductions associées à la même signification auront par défaut la même valeur .

Souvent utilisé au niveau d'une véritable balise html (ex : `<p>` , `<h3>` , ...) l 'attribut i18n peut également être utilisé sur `<ng-container>` ce qui aura pour effet de ne générer que le texte traduit (sans balise html englobante)

Exemple: `<ng-container i18n>I don't output any element</ng-container>`

Application d'une traduction au niveau d'un attribut :

``

Via **i18n-AttributeName**, la valeur de l'attribut est alors d'abord traduite avant d'être utilisée comme une propriété dans l'arbre DOM.

1.4. Fichiers de traduction

Format par défaut : [XML Localization Interchange File Format \(XLIFF, version 1.2\)](#)

Autres formats supportés : XLIFF 2 et XMB (Xml Message Bundle).

Un fichier xliff a généralement l'extension .xlf

NB : l'ancienne commande "~~ng xi18n --out-file src/locale/messages.xlf~~" (époque angular 8) a été remplacée par "**ng extract-i18n --output-path src/locale**".

ng extract-i18n --output-path src/locale

options de la commande : --format=xlf ou --format=xlf2 ou --format=xmb

→ fichier généré : **src/locale/messages.xlf**

Début de variante avec traductions :

src/locale/messages.fr.xlf (que l'on peut créer par copier/coller)

//attention, le contenu du fichier messages.fr.xlf reste à traduire manuellement ou automatiquement !!!

messages.de.xlf (généré par copier/coller ou autrement)

```
<?xml version="1.0" encoding="UTF-8" ?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
  <file source-language="de" datatype="plaintext" original="ng2.template">
    <body>
      <trans-unit id="e27c4e1996f81811e5f18c03bfc30af4db0650cf" datatype="html">
        <source>welcome</source>
        <context-group purpose="location">
          <context context-type="sourcefile">
            src/app/advanced/with-traduction/with-traduction.component.html</context>
          <context context-type="linenumber">2</context>
        </context-group>
      </trans-unit>
    </body>
  </file>
</xliff>
```

--> baser des paramétrage de traduction uniquement sur des numéros de ligne qui peuvent changer, ce n'est pas très viable.

```
<h3 i18n="generic.welcome|my simple welcome message">welcome</h3>
```

```
<button i18n="ok">ok</button> <br/>
```

==>

...

```
<trans-unit id="50c1a51a1c7bd808f40a497ca24543b5b6612169" datatype="html">
  <source>welcome</source>
  <target>bienvenu</target> <!-- a ajouter (pas trop tot) -->
  <context-group purpose="location">... </context-group>
  <note priority="1" from="description">my simple welcome message</note>
  <note priority="1" from="meaning">generic.welcome</note>
</trans-unit>
<trans-unit id="e644f6231bef1a54e64391b9c6856d3b92822a8f" datatype="html">
  <source>ok</source>
  <context-group purpose="location">....</context-group>
  <note priority="1" from="description">ok</note>
</trans-unit>
...
```

Attention, un ajout manuel de **<target>traduction</target>** peut être potentiellement écrasé en re-générant le fichier via certaines commandes

1.5. Eventuelles traductions via xlf-translate

npm install -g xlf-translate

Si au sein des templates html , les "*meaning*" ont été précisés
 via **i18n="generic.welcome|welcome message"**
 on peut alors se faire aider par l'utilitaire **xlf-translate** de manière à remplir les parties manquantes
`<target> ... </target>` d'un fichier **messages.fr.xlf** ou autre .

Après **ng extract-i18n --output-path src/locale**
 et après avoir créer `src/locale/messages.fr.xlf` comme une **copie** de `src/locale/messages.xlf`
 on peut :

créer un fichier de traduction tel que

translate.fr.yml

generic:

welcome : bienvenu
ok : ok
cancel : annuler

lancer_traduction.bat

```
REM npm install -g xlf-translate
xlf-translate --lang-file translate.fr.yml messages.fr.xlf
pause
```

Ceci permet de générer toutes les parties manquantes `<target> ... </target>` dans le fichier **messages.fr.xlf** qui est modifié sur place.

Par défaut, seules les parties manquantes sont ajoutées sans écrasement des anciennes valeurs (sauf si option **--force**)

1.6. Configuration des traductions au niveau du projet angular

Dans le haut de **angular.json**

```
{
  "$schema": "./node_modules/@angular/cli/lib/config/schema.json",
  "version": 1,
  "newProjectRoot": "projects",
  "projects": {
    "myapp": {
      "projectType": "application",
      "schematics": {
        "@schematics/angular:component": {
          "style": "scss"
        }
      },
      "root": "",
      "sourceRoot": "src",
      "prefix": "app",
      "i18n": {
        "sourceLocale": "en",
        "locales": {
          "fr": {
            "translation": "src/locale/messages.fr.xlf"
          },
          "de": {
            "translation": "src/locale/messages.de.xlf"
          }
        }
      },
      "architect": {
        ...
      }
    }
  }
}
```

Dans le milieu de **angular.json**

```
{"projects": {
  "myapp": {
    ...
    "architect": {
      "build": {
        ...
      },
      "configurations": {
        ...
        "fr": {
          "localize": ["fr"],
          "outputPath": "dist/exp-adv-app-fr/",
          "i18nMissingTranslation": "warning"
        },
        "de": {
          "localize": ["de"],
          "outputPath": "dist/exp-adv-app-de/",
          "i18nMissingTranslation": "warning"
        },
        ...
      },
      "production": {
        ...
      }
    }
  }
}}
```

- Sans "i18nMissingTranslation": "error" ou "ignore" , des "warning" par défaut .
- Pour la production, un éventuel paramétrage supplémentaire de type "baseHref": "/fr/" peut

quelquefois être utile (par exemple pour switcher de langue via des routes commençant par "/en/..." ou "/fr/..." ou autre) .

et encore plus bas dans **angular.json** :

```
{"projects": {
  "myapp": {
    ...
    "architect": {
      ...
      "serve": {
        "builder": "@angular-devkit/build-angular:dev-server",
        "options": {
          "browserTarget": "myapp:build"
        },
        "configurationsfrmyapp:build:fr"
          },
          "demyapp:build:de"
          }
        }
      }
    }
  }
}
```

dans **package.json**

```
...
"scripts": {
  "ng": "ng",
  "start": "ng serve",
  "start:frng serve --configuration=fr",
  "start:deng serve --configuration=de",
  "build": "ng build",
  "test": "ng test",
  "lint": "ng lint",
  "build:frng build --configuration=fr",
  "build:deng build --configuration=de",
  "e2e": "ng e2e"
},
...
```

Ceci permet de lancer des commandes de ce type

```
npm run start:fr -- --port=4201
```

```
npm run start:de -- --port=4202
```

http://localhost:4200 --> sans traduction

http://localhost:4201 --> traductions françaises

http://localhost:4202 --> traductions allemandes

1.7. aspects avancés / internationalisation

Avec ou sans "s" (ou bien afficher "un" ou "plusieurs")

--> pluralization :

```
<span i18n>Updated {minutes, plural, =0 {just now} =1 {one minute ago} other {{minutes}} minutes ago}</span>
```

avec une interprétation spéciale dite "ICU" sur la pluralité de la variable minutes (sorte de switch/case sur la valeur de minutes)

avec dans le fichier de traduction :

```
<trans-unit id="5a134dee893586d02bffc9611056b9cadf9abfad" datatype="html">
  <source>{VAR_PLURAL, plural, =0 {just now} =1 {one minute ago} other {<x id="INTERPOLATION" equiv-text="{{minutes}}"/> minutes ago} </source>
  <target>{VAR_PLURAL, plural, =0 {à l'instant} =1 {il y a une minute} other {il y a <x id="INTERPOLATION" equiv-text="{{minutes}}"/> minutes} </target>
</trans-unit>
```

Alternatives :

```
<span i18n>The author is {gender, select, male {male} female {female} other {other}}</span>
```

avec dans le fichier de traduction :

```
<trans-unit id="eff74b75ab7364b6fa888f1cbfae901aaaf02295" datatype="html">
  <source>{VAR_SELECT, select, male {male} female {female} other {other}} </source>
  <target>{VAR_SELECT, select, male {un homme} female {une femme} other {autre}} </target>
</trans-unit>
```

1.8. Choix d'une langue

Principe :

- générer plusieurs versions "fr" , "en" , "de" de l'application angular avec "baseHref": "/fr/" ou "baseHref": "/en/" , ...
- dans un menu principal de l'application (à base de liste déroulante par exemple) , faire en sorte que l'on puisse déclencher des liens hypertextes en "/en/" ou "/fr/" ,

Exemple :

```
<ul>
  <li *ngFor="let language of languageList">
    <a href="/{{language.code}}"/>
      {{language.label}}
    </a>
  </li>
</ul>
```

Pour que ça fonctionne, il faut bien paramétrer une configuration cohérente au niveau du serveur HTTP (ex : nginx) vis à vis des répertoires et des "baseHref" .

XIX - Annexe – extension @angular/material

1. Angular-Material (librairie de composants)

Angular-Material est une librairie de composants graphiques qui sont

- destinés à être intégrés au framework **angular**
- basés sur le look "material" (projet transversal mis en avant par "google" entre autres)

Angular-Material offre des composants intéressants tels que les onglets , les boîtes de dialogue , ...

Ces composants sont pour certains agrémentés d'un redimensionnement automatique (comportement "responsive").

Angular-material est un concurrent direct de "ngx-bootstrap" et "primeNg" .

NB : La plupart des composants de "angular-material" ne gèrent que très peu l'aspect "disposition / placement" . On a souvent besoin d'une technologie complémentaire pour cela .

Bien qu'étant facultatif , le complément angular "**flex-layout**" est souvent utilisé en accompagnement de "angular-material" .

Remarque : Bien que pas très conseillée pour éviter des juxtapositions de looks différents et hétérogènes , une utilisation conjointe/complémentaire de "angular-material" et d'une autre librairie de composants (telle que ngx-bootstrap) est techniquement possible . Cette idée a d'ailleurs été mise en oeuvre au sein d'un projet (existant mais peu utilisé) baptisé "....." .

1.1. intégration de "angular-material" au sein d'un projet angular

ng add @angular/material

et facultativement :

npm install -s @angular/flex-layout

Effet dans **package.json** (exemple):

```
...
"dependencies": {
  "@angular/animations": "^8.2.14",
  "@angular/cdk": "^8.2.3",
  ...
  "@angular/flex-layout": "^8.0.0-beta.27",
  "@angular/material": "^8.2.3",
}
...
```

Effet ou paramétrages dans **angular.json** :

```
....  
"styles": [  
    "./node_modules/@angular/material/prebuilt-themes/indigo-pink.css",  
    "src/styles.scss"  
],  
....
```

Type d'importations techniques à ajouter directement ou indirectement dans **app.module.ts** :

```
import { ImportMaterialModule } from './common/imports/import-material.module';  
import { FlexLayoutModule } from "@angular/flex-layout";  
...  
@NgModule({  
...  
imports: [  
    BrowserModule, AppRoutingModule, BrowserAnimationsModule,  
    FlexLayoutModule, ImportMaterialModule,  
    FormsModule, ReactiveFormsModule  
],  
...  
})
```

src/app/common/imports/**import-material.module.ts**

```
import { NgModule } from '@angular/core';  
import { MatTabsModule} from '@angular/material/tabs';  
import { MatInputModule } from '@angular/material/input';  
import { MatSelectModule } from '@angular/material/select';  
import { MatIconModule } from '@angular/material/icon';  
import { MatMenuModule } from '@angular/material/menu';  
import { MatButtonModule } from '@angular/material/button';  
import { MatCheckboxModule } from '@angular/material/checkbox';  
import { MatRadioModule } from '@angular/material/radio';  
import { MatCardModule } from '@angular/material/card';  
import { MatToolbarModule } from '@angular/material/toolbar';  
import { MatFormFieldModule } from '@angular/material/form-field';  
import {MatSidenavModule} from '@angular/material/sidenav';  
import {MatListModule} from '@angular/material/list';  
import {MatDatepickerModule} from '@angular/material/datepicker';  
import {MatNativeDateModule} from '@angular/material/core';  
import {MatAutocompleteModule} from '@angular/material/autocomplete';  
import {MatSlideToggleModule} from '@angular/material/slide-toggle';  
import {MatExpansionModule} from '@angular/material/expansion';  
import {MatBadgeModule} from '@angular/material/badge';  
import {MatProgressSpinnerModule} from '@angular/material/progress-spinner';  
import {MatTooltipModule} from '@angular/material/tooltip';  
import {MatDialogModule} from '@angular/material/dialog';  
import {MatTableModule} from '@angular/material/table';  
import {MatTreeModule} from '@angular/material/tree';  
import {MatStepperModule} from '@angular/material/stepper';
```

```

import {MatSortModule} from '@angular/material/sort';
import {MatPaginatorModule} from '@angular/material/paginator';

@NgModule({
  imports: [
    MatTabsModule,      MatCardModule,MatExpansionModule,
    MatIconModule,     MatFormFieldModule,   MatInputModule,    MatSelectModule,
    MatButtonModule,   MatListModule,    MatCheckboxModule,MatSlideToggleModule,
    MatRadioModule,    MatMenuModule,   MatToolbarModule,   MatSidenavModule,
    MatDatepickerModule,MatNativeDateModule, MatAutocompleteModule,
    MatBadgeModule,MatProgressSpinnerModule, MatTooltipModule,  MatDialogModule,
    MatTableModule,    MatTreeModule,   MatStepperModule,   MatSortModule,   MatPaginatorModule
  ],
  exports:[MatTabsModule,      MatCardModule,MatExpansionModule,      MatIconModule,      MatFormFieldModule,
    MatInputModule,    MatSelectModule,   MatButtonModule,    MatListModule,
    MatCheckboxModule,MatSlideToggleModule, MatRadioModule,   MatMenuModule,
    MatToolbarModule,   MatSidenavModule, MatDatepickerModule,MatNativeDateModule,
    MatAutocompleteModule, MatBadgeModule,MatProgressSpinnerModule, MatTooltipModule,  MatDialogModule,
    MatTableModule,    MatTreeModule,   MatStepperModule,   MatSortModule,   MatPaginatorModule
  ]
})
export class ImportMaterialModule {}
```

2. Essentiel de "flex-layout" (en intégration angular)

npm install -s @angular/flex-layout

...html

empilement (si moins de 960px):

```

<div class="container" fxLayout.lt-md="column"
fxLayoutAlign="center" fxLayoutGap="10px" fxLayoutGap.lt-md="2px">
  <div class="a" fxFlex="25%">divA (25%)</div>
  <div class="b" fxFlex="50%">divB (50%)</div>
  <div class="c">divC</div>
</div>
```

empilement (si moins de 960px):

empilement (si moins de 960px):

breakpoints	size(px)	breakpoints	size(px)	breakpoints	size(px)
xs (extra small)	599 ou moins	lt-sm (less than small)	moins que 600	gt-xs (greater than xs)	600 ou plus
sm (small or medium)	600 à 959	lt-md (less than md)	moins que 960	gt-sm (greater than sm)	960 ou plus
md (medium)	960 à 1279	lt-lg (less than large)	moins que 1280	gt-md (greater than md)	1280 ou plus
lg (large)	1280 à 1919	lt-xl (less than xl)	moins que 1920	gt-lg (greater than large)	1920 ou plus
xl (extra large)	1920 à 5000				

3. Quelques composants "angular-material"

Card (panneau d'encadrement) :

Basic

```
<mat-card class="my-card">
  <mat-card-header class="my-card-header">
    <mat-card-title>Basic</mat-card-title>
    <!-- <mat-card-subtitle>angular material</mat-card-subtitle> -->
  </mat-card-header>

  <mat-card-content class="my-card-content">
    ...
  </mat-card-content>
</mat-card>
```

```
.basic { background: white; }
.my-card { border: 1px solid blue; padding: 0 }

.my-card-header {
  background-color: #2196F3; color: white;
  padding-left: 1em; padding-top: 0.5em;
}

.my-card-content { padding: 1em; }
```

Composants "onglets" (tab , tab-group , ...)

```
<mat-tab-group>
  <mat-tab label="tva">
    <app-tva></app-tva>
  </mat-tab>
  <mat-tab label="titre onglet2">
    ...contenu onglet 2...
  </mat-tab>
<mat-tab-group>
```

tva

demo angular flexLayout

Composants élémentaires pour formulaires

```
<div>
<form role="form" class="form-container" >
<mat-form-field [appearance]="settingService.my_mat_appearance">
  <mat-label>ht:</mat-label>
  <input matInput placeholder="ht" name="ht" [(ngModel)]="ht" (input)="onCompute()" />
  <!-- <mat-icon matSuffix>favorite</mat-icon> -->
  <mat-hint>montant hors taxe</mat-hint>
</mat-form-field>

<mat-form-field [appearance]="settingService.my_mat_appearance">
  <mat-label>taux (en%):</mat-label>
  <mat-select placeholder="taux" name="taux" [(ngModel)]="taux"
    (selectionChange)="onCompute()">
    <mat-option *ngFor="let t of listeTaux" [value]="t">{ {t} }</mat-option>
  </mat-select>
</mat-form-field>
</form>
tva: <span>{ {tva | currency:'EUR':symbol:'1.0-2'}}</span> <br/>
ttc: <span>{ {ttc | currency:'EUR':symbol:'1.0-2'}}</span>
</div>
```

....css

```
.form-container { display: flex; flex-direction: column; }
.form-container > * { width: 30%; }
```

Look avec le paramètre [appearance]="" 'standard' " :

ht:
200

montant hors taxe

taux (en%):
20

tva: €40
ttc: €240

Look avec le paramètre [appearance]="" 'outline' " :

ht:

200

montant hors taxe

taux (en%):

20

NB : les valeurs possibles de **appearance** sont "standard" , "outline" , "fill" et "legacy" .

Autres composants de base (exemples):

```
<div>
  <form role="form" class="form-container" >
    <mat-form-field [appearance]="'standard' ">
      <!-- [hideRequiredMarker]="false" [floatLabel] = "auto" by default on mat-form-field -->
      <mat-label>full name:</mat-label>
      <input matInput placeholder="name" name="name"
        minLength="6" maxLength="20" required
        [(ngModel)]="person.name" />
      <mat-hint align="end">full name</mat-hint>
    </mat-form-field>
    <div>
      <label>kind: </label>
      <mat-radio-group placeholder="kind" name="kind" [(ngModel)]="person.kind" >
        <mat-radio-button matInput *ngFor="let k of listeKind" [value]="k">{{k}}</mat-radio-button>
      </mat-radio-group>
      <!-- mat-radio-group cannot be put inside mat-form-field , ??? -->
    </div>

    <mat-form-field [appearance]="'standard' ">
      <mat-label>email:</mat-label>
      <input matInput name="email" type="email" placeholder="email" [(ngModel)]="person.email" />
    </mat-form-field>

    <mat-checkbox name="crazy" [(ngModel)]="person.crazy" >crazy (as checkbox)</mat-checkbox>
    <mat-slide-toggle name="crazy" [(ngModel)]="person.crazy" >crazy (as slide-toggle)</mat-slide-toggle>
    <!-- mat-checkbox cannot be put inside mat-form-field , already has a label -->

    <mat-form-field [appearance]="'standard' ">
      <mat-label>birthday:</mat-label>
      <input matInput name="birthday" [matDatepicker]="#myDatePicker"
        placeholder="birthday" [(ngModel)]="person.birthday" /> <!-- type="date" if no picker -->
      <mat-datepicker-toggle matSuffix [for]="#myDatePicker"></mat-datepicker-toggle>
      <mat-datepicker #myDatePicker></mat-datepicker>
    </mat-form-field>

    <mat-form-field [appearance]="'standard' ">
      <mat-label>childNumber:</mat-label>
      <input matInput name="childNumber" type="number" placeholder="childNumber" [(ngModel)]="person.childNumber" />
    </mat-form-field>
    <mat-form-field [appearance]="'standard' ">
      <mat-label>preferedColor:</mat-label>
      <input matInput name="preferedColor" type="color" placeholder="preferedColor" [(ngModel)]="person.preferedColor" />
    </mat-form-field>
    <mat-form-field [appearance]="'standard' ">
      <mat-label>password:</mat-label>
      <input matInput name="password" type="password" placeholder="password" [(ngModel)]="person.password" />
    </mat-form-field>
    <mat-form-field [appearance]="'standard' ">
      <mat-label>country:</mat-label>
      <input matInput name="country" placeholder="country"
        (ngModelChange)="adjustfilteredCountries()"
        [(ngModel)]="person.country" [matAutocomplete]="#autoCountry" />
      <!-- for reactiveForm , [formControl] = "myControl" and no ngModel -->
      <mat-autocomplete #autoCountry="matAutocomplete">
    </mat-autocomplete>
  </form>
</div>
```

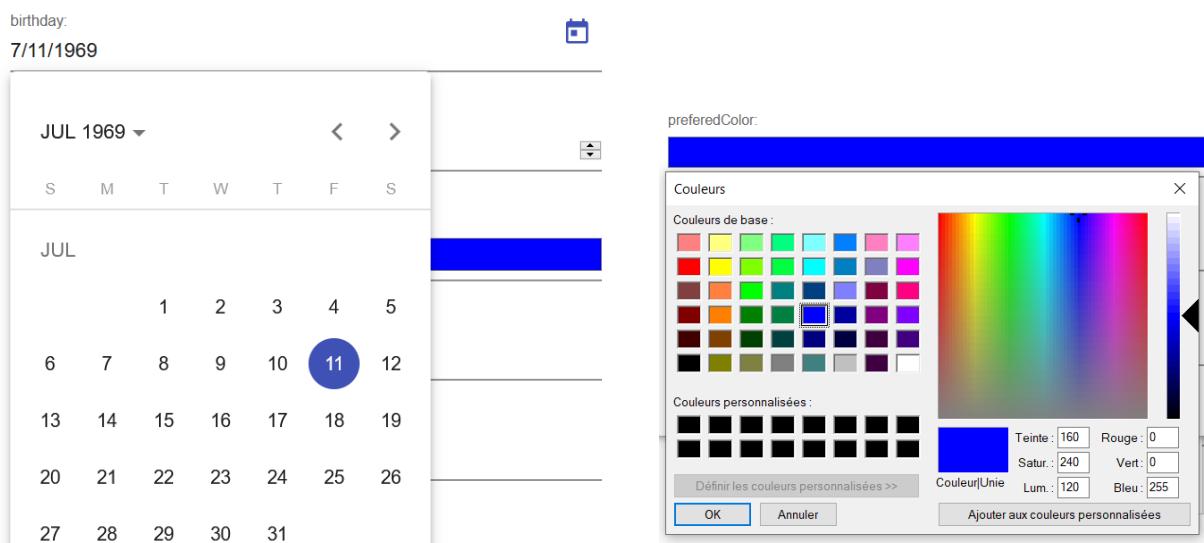
```

<mat-option *ngFor="let c of filteredCountries | async" [value]="c"> {{c}} </mat-option>
</mat-autocomplete>
</mat-form-field>
<mat-form-field [appearance]="'standard' ">
  <mat-label>comment:</mat-label>
  <textarea matInput name="comment" placeholder="comment" [(ngModel)]="person.comment"></textarea>
</mat-form-field>
</form>
person: <span>{{person | json}}</span> <br/>
</div>

```

The screenshot shows a form with the following fields:

- full name:** *
didier defrance
- kind:** Male Female
- email:** didier@d-defrance.fr
- checkbox:** crazy (as checkbox)
- slide-toggle:** crazy (as slide-toggle)
- birthday:** 7/11/1969
- childNumber:** 2
- preferredColor:** A solid blue rectangle.
- password:** **•••**
- country:** F
France
- Finlande**



....ts

```

.....
import { Observable , of} from 'rxjs';
import { map , startWith} from 'rxjs/operators';

@Component({ ...})
export class VariousFormComponent implements OnInit {
  listeKind = [ "Male" , "Female"];
  //myControl = new FormControl(); //if reactiveForm and autocomplete
  countries = [ "France" , "Finlande" , "Allemagne" , "Autriche" , "Italie" , "Espagne" , "..."];
  filteredCountries: Observable<string[]>;
  person : Person = new Person();
  constructor() { }
  ngOnInit(){ }
}

```

```

adjustfilteredCountries() {
  this.filteredCountries = of(this.countries)
    .pipe(
      map(listCountries => this._filterCountriesIgnoreCase(listCountries, this.person.country))
    );
}

private _filterCountriesIgnoreCase(listCountries:string[],value: string): string[] {
  const filterValue = value.toLowerCase();
  return listCountries.filter(c => c.toLowerCase().includes(filterValue));
}
}

```

Composants "boite de dialogue" (exemple , autres variantes possibles)

example-dialog.component.ts

```

import { Component, OnInit, Inject } from '@angular/core';
import { MatDialogRef, MAT_DIALOG_DATA } from '@angular/material/dialog';
import { MyDialogData } from './myDialogData';

@Component({
  selector: 'app-example-dialog',
  templateUrl: './example-dialog.component.html',
  styleUrls: ['./example-dialog.component.scss']
})
export class ExampleDialogComponent implements OnInit {

  /*
  entryComponents: [ExampleDialogComponent],
must be added in @NgModule()
*/

  constructor(
    public dialogRef: MatDialogRef<ExampleDialogComponent>,
    @Inject(MAT_DIALOG_DATA) public data: MyDialogData) {}

  onNoClick(): void { this.dialogRef.close(); }

  ngOnInit() { }
}

```

```

export class MyDialogData {
  name:string;
  animal:string;
}

```

example-dialog.component.html

```

<h1 mat-dialog-title>Hi {{data.name}}</h1>
<div mat-dialog-content>
  <p>What's your favorite animal?</p>
  <mat-form-field>

```

```

<mat-label>Favorite Animal</mat-label>
<input matInput [(ngModel)]="data.animal">
</mat-form-field>
</div>
<div mat-dialog-actions>
  <button mat-button (click)="onNoClick()">No Thanks</button>
  <button mat-button [mat-dialog-close]="data.animal" cdkFocusInitial>Ok</button>
</div>

```

Exemple d'utilisation :

.....ts

```

import { Component, OnInit } from '@angular/core';
import { MatDialog } from '@angular/material/dialog';
import { ExampleDialogComponent } from './example-dialog/example-dialog.component';

@Component({
  selector: 'app-divers',
  templateUrl: './divers.component.html',
  styleUrls: ['./divers.component.scss']
})
export class DiversComponent implements OnInit {
  animal: string;
  name: string;

  constructor(public dialog: MatDialog) {}

  openDialog(): void {
    /*
    entryComponents: [ExampleDialogComponent],
    must be added in @NgModule()
    */
    const dialogRef = this.dialog.open(ExampleDialogComponent, {
      width: '250px',
      data: {name: this.name, animal: this.animal}
    });

    dialogRef.afterClosed().subscribe(result => {
      console.log('The dialog was closed');
      this.animal = result;
    });
  }
}

ngOnInit() {}

```

```

<ol>
  <li>

```

```
<mat-form-field>
  <mat-label>What's your name?</mat-label>
  <input matInput [(ngModel)]="name">
</mat-form-field>
</li>
<li>
  <button mat-raised-button (click)="openDialog()">open dialog</button>
</li>
<li *ngIf="animal">
  You chose: <i>{{animal}}</i>
</li>
</ol>
```

What's your name?

1. didier

2. open dialog



What's your name?

1. didier

2. open dialog

3. You chose: Dog

Composants "step" ,

Composants "mat-table" avec dataSource

table of countries (click on column header to sort)

-	code	name ↑	capital city	area	population
<input checked="" type="checkbox"/>	de	Allemagne	Berlin	357386	83073100
<input checked="" type="checkbox"/>	es	Espagne	Madrid	505911	46934632
<input type="checkbox"/>	fr	France	Paris	632734	67795000
<input type="checkbox"/>	it	Italie	Rome	301336	60359546
<input type="checkbox"/>	en	Royaume-Uni	Londres	246690	65761117
Total			2044057	323923395	
Items per page: <select>50</select> 1 – 5 of 5 < < > >>					

Attention: la selection globale et calcul du total ne tient pas compte de la pagination (par default)
 selected countries : [{ "code": "de", "name": "Allemagne",
 "capital_city": "Berlin", "population": 83073100, "area": 357386,
 "regions": null }, { "code": "es", "name": "Espagne", "capital_city":
 "Madrid", "population": 46934632, "area": 505911, "regions": null }]

NB : bien que l'exemple suivant combine "selection, filtrage , tri , totaux, pagination, ..." , chacun de ces aspects est facultatif et l'on peut dans beaucoup de cas effectuer une mise en oeuvre bien plus simple .

Dans la logique complexe/élaborée de construction/fonctionnement des tableaux "**mat-table**" , les lignes d'entête , de données et de "totaux/pied de tableau" seront automatiquement générées à partir des éléments complémentaires suivants :

- **liste ordonnées des noms de colonnes à afficher** (ex : *displayedColumns coté .ts*)
- définition abstraite de chaque colonnes (`<ng-container matColumnDef="colNamexy">`)
- **source de données** (intermédiaire "**dataSource**" entre données et vue , prenant en compte les **tris** et éventuels filtrages,)

with-table.component.ts

```
import { Component, OnInit, ViewChild } from '@angular/core';
import { GeoService } from '../common/service/geo.service';
import { MatTableDataSource } from '@angular/material/table';
import { MatSort } from '@angular/material/sort';
import { Country } from '../common/data/country';
import { SelectionModel } from '@angular/cdk/collections';
import { MatPaginator } from '@angular/material/paginator';

@Component({
  selector: 'app-with-table',
  templateUrl: './with-table.component.html',
  styleUrls: ['./with-table.component.scss']
})
export class WithTableComponent implements OnInit {

  displayedColumns: string[] = ['select','code', 'name', 'capital_city', 'area', 'population'];
  countries : Country[] = [];
  dataSource = new MatTableDataSource(this.countries); //this.countries; if no sort

  selection = new SelectionModel<Country>(true /*allowMultiSelect*/, [] /*initialSelection*/);

  constructor(private geoService: GeoService) { }

  //sort refer to table with matSort directive in .html
  //useful for get order choice (by name , by population, ...)
  @ViewChild(MatSort, {static: true}) sort: MatSort;

  @ViewChild(MatPaginator, {static: true}) paginator: MatPaginator;

  ngOnInit() {
    this.geoService.getCountries().subscribe(
      (countries)=>{ /*this.dataSource=countries; if no sort*/
        this.countries=countries;
        this.dataSource= new MatTableDataSource(countries);
        this.dataSource.sort=this.sort; //by name or by ...
        this.dataSource.paginator = this.paginator;
      }
    )
  }
}
```

```

this.dataSource.filterPredicate =
  (data: Country, filter: string) => !filter || (data.name.toLowerCase()).includes(filter);
}

};

}

applyFilter(event: Event) {
  const filterValue = (event.target as HTMLInputElement).value;
  this.dataSource.filter = filterValue.toLowerCase();
}

/** Gets the total population and total area of all countries. */
getTotalPopulation() {
  //this.countries.map(..) or this.dataSource.filteredData.map(..)
  return this.dataSource.filteredData.map(c => c.population)
    .reduce((acc, value) => acc + value, 0);
}

 getTotalArea() {
  return this.dataSource.filteredData.map(c => c.area)
    .reduce((acc, value) => acc + value, 0);
}

/** Whether the number of selected elements matches the total number of rows. */
isAllSelected() {
  const numSelected = this.selection.selected.length;
  const numRows = this.dataSource.data.length;
  return numSelected === numRows;
}

/** Selects all rows if they are not all selected; otherwise clear selection. */
masterToggle() {
  this.isAllSelected() ?
    this.selection.clear() :
    this.dataSource.data.forEach(row => this.selection.select(row));
}

/** The label for the checkbox on the passed row */
checkboxLabel(row?: Country): string {
  if (!row) {
    return `${this.isAllSelected() ? 'select' : 'deselect'} all`;
  }
  return `${this.selection.isSelected(row) ? 'deselect' : 'select'} row ${row.code}`;
}

```

with-table.component.css

```

.selected {
  background-color: red;
}

.mat-row.highlighted {

```

```

background: lightblue;
}

table {
  width: 100%;
  overflow-x: auto;
  overflow-y: hidden;
  min-width: 500px;
}

.mat-header-cell, .mat-sort-header {
  background-color: lightblue;
  font-weight: bold;
}

.my-table-container-for-scroll{
  height: 400px;
  overflow: auto;
}

tr.mat-footer-row {
  font-weight: bold;
}

```

with-table.component.html

```

<h4>table of countries (click on column header to sort)</h4>

<mat-form-field>
  <mat-label>Filter</mat-label>
  <input matInput (keyup)="applyFilter($event)">
  placeholder="Ex. i">
</mat-form-field>
<div class="my-table-container-for-scroll">
<table mat-table [dataSource]="dataSource" matSort>

  <!-- Note that these columns can be defined in any order.
       The actual rendered columns are set as a property on the row definition" -->

  <!-- code Column -->
  <ng-container matColumnDef="code">
    <th mat-header-cell *matHeaderCellDef> code </th>
    <td mat-cell *matCellDef="let c"> {{c.code}} </td>
    <td mat-footer-cell *matFooterCellDef>Total</td>
  </ng-container>

  <!-- Name Column -->
  <ng-container matColumnDef="name">
    <th mat-header-cell *matHeaderCellDef mat-sort-header> name </th>
    <td mat-cell *matCellDef="let c"> {{c.name}} </td>
    <td mat-footer-cell *matFooterCellDef></td>
  </ng-container>

  <!-- Capital-city Column -->
  <ng-container matColumnDef="capital_city">

```

```

<th mat-header-cell *matHeaderCellDef> capital city </th>
<td mat-cell *matCellDef="let c"> {{c.capital_city}} </td>
<td mat-footer-cell *matFooterCellDef></td>
</ng-container>

<!-- Population Column -->
<ng-container matColumnDef="population">
  <!-- mat-sort-header only ok if matSort in table -->
  <th mat-header-cell *matHeaderCellDef mat-sort-header> population </th>
  <td mat-cell *matCellDef="let c"> {{c.population}} </td>
  <td mat-footer-cell *matFooterCellDef> {{getTotalPopulation()}}</td>
</ng-container>

<!-- Area Column -->
<ng-container matColumnDef="area">
  <th mat-header-cell *matHeaderCellDef mat-sort-header> area </th>
  <td mat-cell *matCellDef="let c"> {{c.area}} </td>
  <td mat-footer-cell *matFooterCellDef> {{getTotalArea()}}</td>
</ng-container>

<!-- Checkbox selection Column -->
<ng-container matColumnDef="select">
<th mat-header-cell *matHeaderCellDef>
  <mat-checkbox (change)="$event ? masterToggle() : null"
    [checked]="selection.hasValue() && isAllSelected()"
    [indeterminate]="selection.hasValue() && !isAllSelected()"
    [aria-label]="checkboxLabel()">
    </mat-checkbox>
</th>
<td mat-cell *matCellDef="let row">
  <mat-checkbox (click)="$event.stopPropagation()"
    (change)="$event ? selection.toggle(row) : null"
    [checked]="selection.isSelected(row)"
    [aria-label]="checkboxLabel(row)">
    </mat-checkbox>
</td>
<td mat-footer-cell *matFooterCellDef></td>
</ng-container>

<tr mat-header-row *matHeaderRowDef="displayedColumns; sticky: true"></tr>
<tr mat-row *matRowDef="let row; columns: displayedColumns;">
  > <!-- (click)="selection.toggle(row)" --> </tr>
<!-- if mat-footer-row , mat-footer-cell must be defined for each displayedColumns -->
<tr mat-footer-row *matFooterRowDef="displayedColumns;"></tr>
</table>
<mat-paginator [pageSizeOptions]="[50, 20, 12, 6, 3]" showFirstLastButtons></mat-paginator>
</div>
Attention: la selection globale et calcul du total ne tient pas compte de la pagination (par defaut)<br/>
selected countries : {{selection._selected | json}}

```

XX - Annexe – PWA (Progressive Web App)

1. PWA (Progressive Web App) – aperçu général

1.1. Présentation des "progressive web apps"

Les "Progressive Web Apps" (PWA) sont des **applications web modernes (html/css/js)** , **pouvant fonctionner** en mode "*hors connexion*" et axées sur les **mobiles** .
Ces applications "PWA" sont censées rivaliser avec des applications mobiles (natives ou hybrides) .

Quelques comparaisons :

	Développement	Exécution
Application mobile native (ex : java pour Android)	Langage et api spécifique à une plateforme mobile (ex : java et android-sdk ou bien ios/iphone/swift)	application mobile (à télécharger depuis un "store") et à exécuter sur un type précis de smartphone (android ou iphone)
Application mobile hybride (ex : cordova et/ou ionic)	Code source en html/js relativement portable (android , iphone, ...) mais nécessitant une étape de construction qui est moyennement simple avec android et délicate avec ios/iphone	même environnement d'exécution et comportement qu'une application native (avec néanmoins des performances et fonctionnalités quelquefois un peu plus limitées)
PWA (Progressive Web App)	Code source en html/css/js très portable ne nécessitant pas de phase de construction complexe et dépendante d'une plateforme mobile	Une "pwa" s'exécute au sein d'un navigateur de smartphone mais avec le "plein écran" possible et d'autres spécificités. Les fonctionnalités techniques d'une "pwa" seront donc liées aux versions (idéalement récentes) des navigateurs.

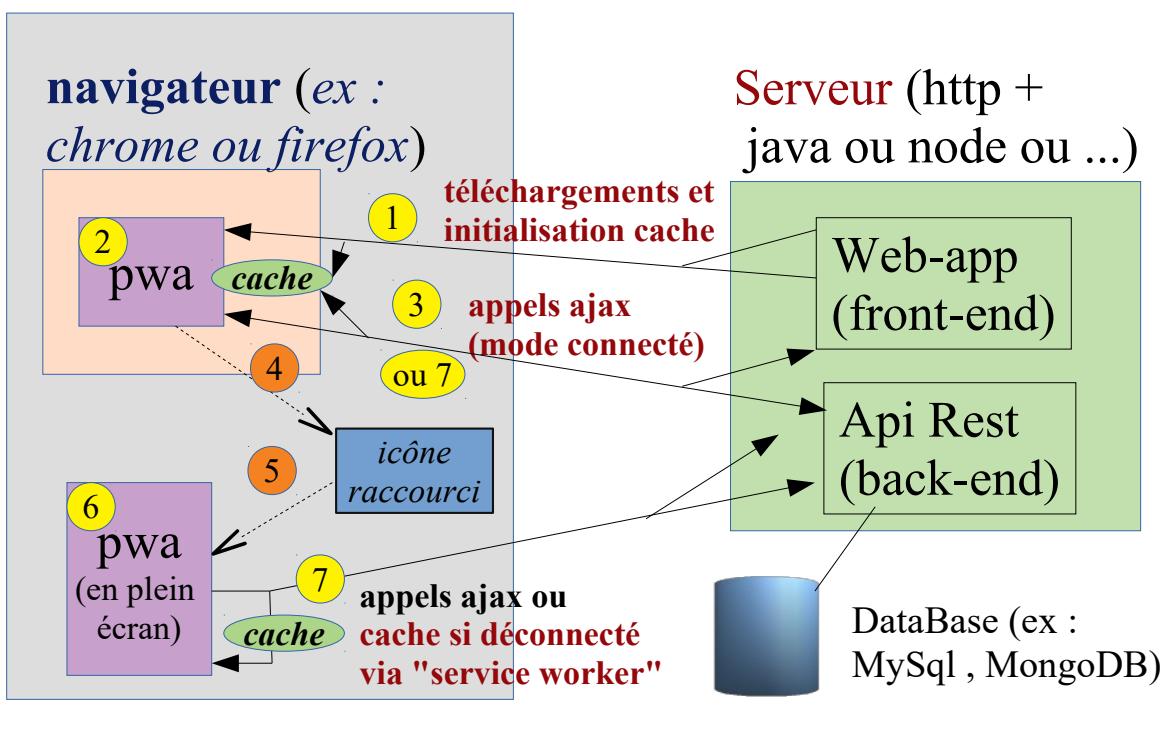
Le concept de "pwa" est très moderne et pour l'instant assez peu répandu en 2018/2019 . L'essor prévu des "pwa" dépendra essentiellement du support offert par les navigateurs des smartphones . Pour l'instant "android , chrome , firefox" avancent clairement dans cette direction . Apple (ios/iphone) comporte un navigateur supportant partiellement les "pwa" . On verra dans l'avenir les choix stratégiques d'Apple (ouverture ou fermeture) .

Dans "Progressive Web App" le terme "progressif" signifie que l'utilisateur d'un smartphone peut de manière très progressive :

- **naviguer sur internet et utiliser l'application web comme un site ordinaire**
- utiliser potentiellement l'application en mode "**plein écran**" (comme une appli mobile)
- **générer un "icône/raccourci" de lancement sur le fond du bureau** (pour relancer ultérieurement cette appli en mode "navigateur pas encore ouvert")
- **utiliser certaines parties de l'application en mode "déconnecté"** grâce à des "**service-workers**" (*tâches de fond* pouvant faire office de "cache" ou "proxy")
- **finalement utiliser l'application web aussi facilement que si elle était native et disposer en prime d'une réactualisation (mise à jour) automatique du code** (vers la dernière version en date).

PWA (Progressive Web App)

smartphone (ex : android)



1.2. Fonctionnalités d'une "progressive web app"

Une "progressive web app" est avant tout une bonne "web app" s'utilisant bien sur divers navigateurs (PC/desktop, smartphone, tablettes) et qui a en plus certaines fonctionnalités spécifiques au contexte mobile (en mode quelquefois déconnecté) .

Fonctionnalités d'une bonne "web app" :

- **Responsive**
- **Lightweight**
- **Géolocalisation (via navigator.geolocation HTML5)**
- etc (intuitive, ergonomique, efficace ,)

Fonctionnalités supplémentaires d'une "progressive web app" :

- **Web App Manifest (Add to Home Screen)**
- **Service Worker**
 - **cache**
 - **notifications**
-

1.3. Paramétrage fondamental pour "responsive" sur smartphone

index.html

```
<html>
<head>
<title>my pwa</title>
<link rel="stylesheet" href=".css/bootstrap.min.css">
<link rel="stylesheet" type="text/css" href="css/styles.css" />
<link rel="icon" href=".img/pwa-icon_48_48.png" /> <!-- favicon.ico by default -->
<link rel="manifest" href="manifest.json" />
<base href="/" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <!-- for responsive bootstrap on mobile device -->
</head>
...
</html>
```

La ligne

```
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

est indispensable pour obtenir un bon comportement "responsive" sur un appareil mobile (ex : smartphone ou tablette) .

Sans celle-ci , ce qui fonctionne bien au sein d'un navigateur sur PC/desktop , ne fonctionne malheureusement pas bien sur un téléphone "android" (c'est tout petit ou bien ça déborde) .

1.4. Quelques exemples d'applications web en mode "PWA"

La plupart des sites web d'actualités fonctionnent aujourd'hui en mode "pwa" (avec cependant plus ou moins de soins apportés au mode "off-line" quelquefois inopérant) :

- le figaro
- le monde
- ...

1.5. Emulateur android permettant de visualiser les effets "pwa" :

- blueStack 4 ne fonctionne pas bien en mode "pwa"
- l'émulateur android proposé par défaut avec "android studio" fonctionne bien
-

2. Web App Manifest / add to home screen

2.1. Génération d'un icône "raccourci" en fond d'écran

De façon à ce qu'un navigateur récent (fonctionnant sur un mobile) puisse proposer la génération d'un icône raccourci sur le fond du bureau , il faut référencer un fichier manifest.json (décrivant l'image de l'icône dans différentes tailles) .

Ce fichier appelé "web app manifest" contiendra quelques informations complémentaires (nom de l'application,) .

2.2. Fichier "web app manifest"

Le manifest JSON offre un moyen de décrire un point d'entrée de l'application, afin de proposer à l'utilisateur une expérience d'application native lors du prochain lancement de la PWA :

- Icône dédiée sur la Home (lien URL)
- Lancement en plein écran,

Ce fichier est à déposer à la racine du serveur (souvent placé à coté du Service Worker).

Exemple :

manifest.json

```
{
  "name": "My Progressive Web App",
  "short_name": "MyPWA",
  "start_url": "/index.html",
  "icons": [
    {
      "src": "img/pwa-icon_48_48.png",
      "sizes": "48x48",
      "type": "image/png"
    },
    {
      "src": "img/pwa_72_72.png",
      "sizes": "72x72",
      "type": "image/png"
    },
    {
      "src": "img/pwa-cube_96_96.png",
      "sizes": "96x96",
      "type": "image/png"
    },
    {
      "src": "img/pwa-cube_144_144.png",
      "sizes": "144x144",
      "type": "image/png"
    },
    {
      "src": "img/pwa-icon_192_192.png",
      "sizes": "192x192",
      "type": "image/png"
    }
  ],
  "display": "standalone",
  "background_color": "#4e8ef7",
  "theme_color": "#4e8ef7"
}
```

Ce fichier doit être référencé au niveau de **index.html**

```
<!DOCTYPE html>
```

```
<html>  <head>
    <meta charset="UTF-8">
    <title>My progressive Web App</title>
    <link rel="manifest" href="manifest.json">
</head>
<body>    . . . </body> </html>
```

Reconnaissance du fichier "*web app manifest*" au sein de la partie "*Application*" de "*outils de développement*" du navigateur "Chrome Desktop" :

The screenshot shows the Chrome DevTools interface with the "Application" tab selected. On the left, there's a sidebar with sections for Application (Manifest, Service Workers, Clear storage), Storage (Local Storage, Session Storage, IndexedDB, Web SQL, Cookies), Cache (Cache Storage, Application Cache), and Frames (top). The "Manifest" item under Application is currently selected.

App Manifest

:3000/manifest.json

Identity

Name: My Progressive Web App
Short name: MyPWA

Presentation

Start URL: /index.html
Theme color: #4e8ef7
Background color: #4e8ef7

Orientation

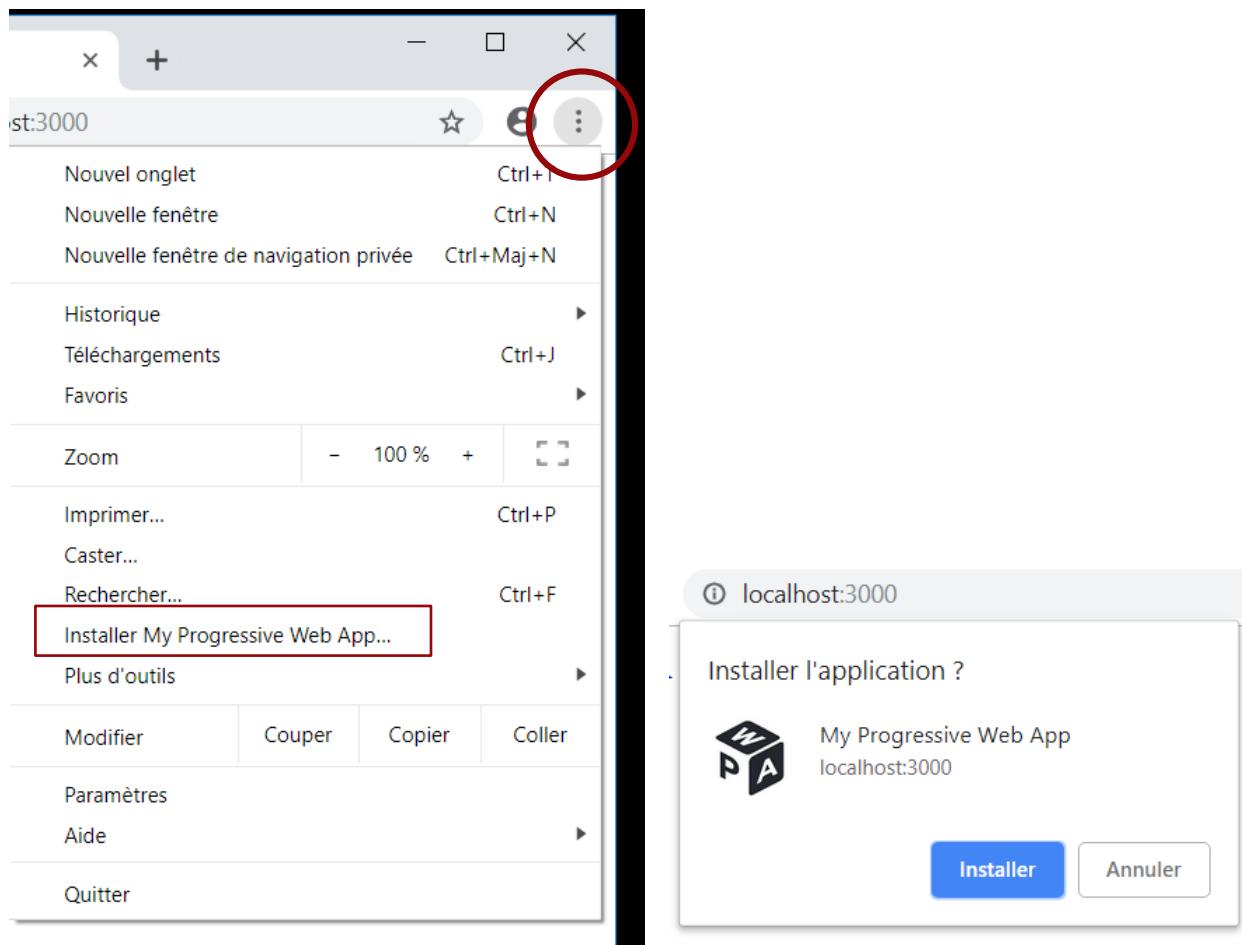
Display: standalone

Icons

48x48
image/png

A circular icon labeled "PWA" is displayed.

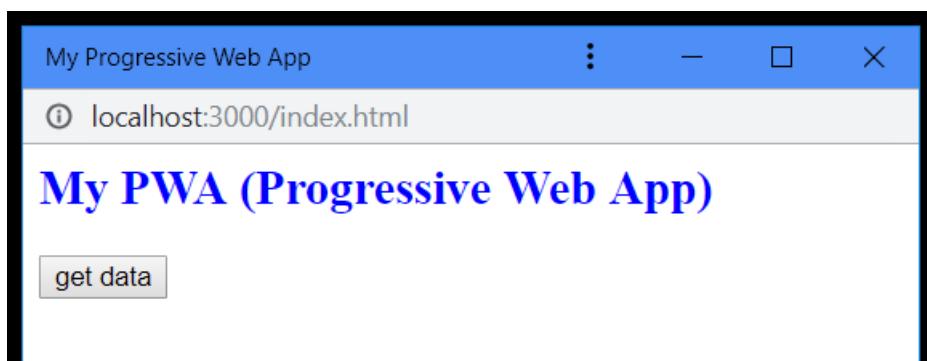
2.3. Effets au sein du navigateur "Chrome Desktop" (tests)



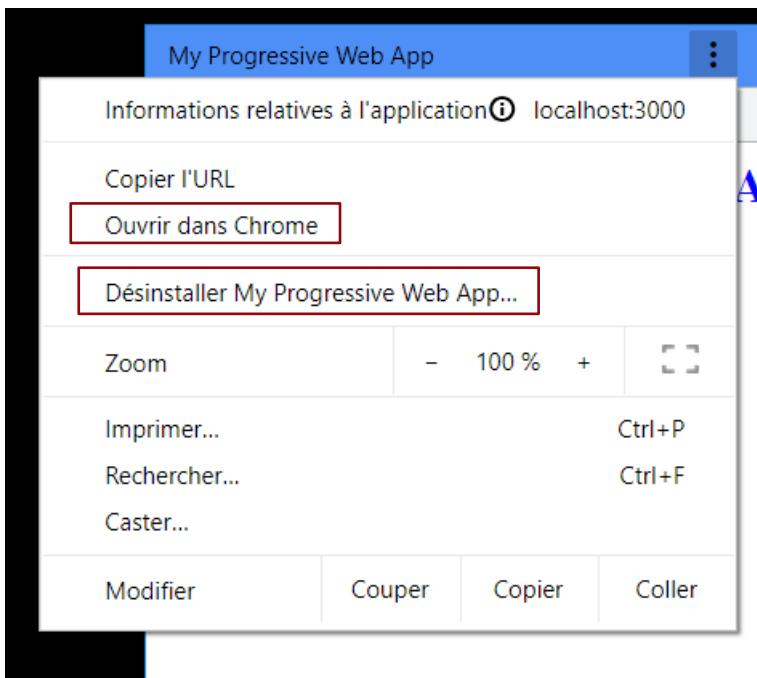
==> cette action génère un icône raccourci sur le fond du bureau :



En cliquant sur ce raccourci , l'application se lance dans une fenêtre spéciale (proche plein-écran) et avec les mêmes fonctionnalités que le navigateur "Chrome" habituel .



Cette application "pwa" comporte un menu contextuel permettant éventuellement la désinstallation de l'application ou bien le lancement de celle ci au sein du navigateur "Chrome" en mode habituel "multi-sites".



2.4. Attention , pas de "add to home screen" sans service-worker enregistré et gérant les événements "install" et "fetch"

En début de développement (ou de "tp") , il faudra au minimum prévoir le code suivant pour que le navigateur propose le menu "add to home screen" ou "installer la pwa" ou "page/créer un raccourci"

dans index.html (ou client.js) :

```
if ('serviceWorker' in navigator) {
    window.addEventListener('load', function() {
        navigator.serviceWorker.register('service-worker.js')
            .then(() => navigator.serviceWorker.ready)
            .then(function(registration) {
                // Registration was successful
                console.log('ServiceWorker registration successful with scope: ', registration.scope);
            }, function(err) {
                // registration failed :
                console.log('ServiceWorker registration failed: ', err);
            });
    });
}
```

dans service-worker.js:

```
self.addEventListener('install', function(event) {
    console.log('[ServiceWorker] install ***');
```

```
});  
  
self.addEventListener('activate', function(event) {  
    console.log('[ServiceWorker] activate ***');  
});  
  
self.addEventListener('fetch', function(event) {  
    console.log('[ServiceWorker] fetch ***');  
});
```

2.5. Effets au sein du navigateur "Chrome pour android"

Lorsqu'une application web est reconnue comme progressive (webmanifest + service-worker + ...) , l'entrée "add to home screen" (ou bien "installer l'application ...") apparaît dans le menu principal du navigateur .

2.6. Effets au sein du navigateur "Firefox pour android"

Lorsqu'une application web est reconnue comme progressive (webmanifest + service-worker + ...) , l'entrée "page / ajouter un raccourcis" (ou bien "....") apparaît dans le menu principal du navigateur . En outre , un icône "maison +" apparaît quelquefois pour signifier un "add to home screen" possible en cliquant dessus .

2.7. Lien ou bouton facilitant l'installation (A2HS)

Pour inviter l'utilisateur à installer l'application en mode A2HS (Add To Home Screen) , il est possible d'ajouter un lien ou bouton qui sera pris en compte par différents navigateurs mobiles (ex : "Chrome pour android" , "Firefox pour android") .

Ce bouton servira simplement à inviter l'utilisateur (via javascript) à gérer l'événement "***beforeinstallprompt***" prévu pour demander l'installation en fond de bureau .

Ce bouton soit idéalement être invisible avant la réception de l'événement .

Exemple de code :

```
<button class="add-button">Add to home screen</button>
```

```
.add-button {
  position: absolute;
  top: 1px;
  left: 1px;
}
```

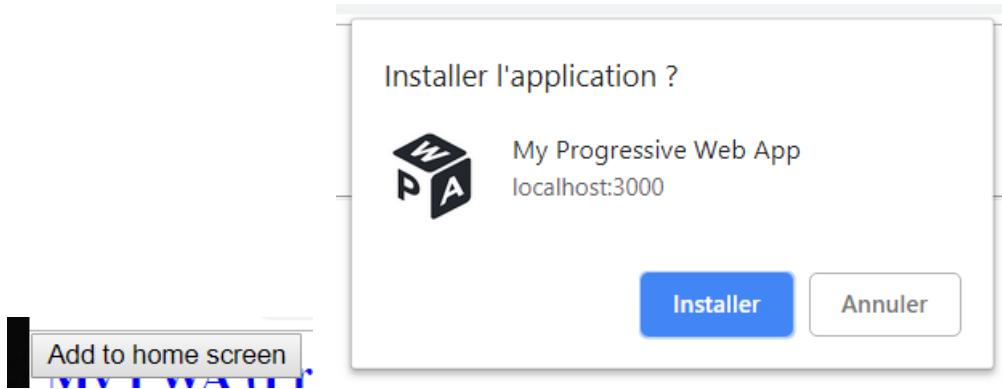
dans un bloc de script en fin de partie "body" :

```
let deferredPrompt;
const addBtn = document.querySelector('.add-button');
addBtn.style.display = 'none';

window.addEventListener('beforeinstallprompt', (e) => {
  // Prevent Chrome 67 and earlier from automatically showing the prompt
  e.preventDefault();
  // Stash the event so it can be triggered later.
  deferredPrompt = e;
  // Update UI to notify the user they can add to home screen
  addBtn.style.display = 'block';

  addBtn.addEventListener('click', (e) => {
    // hide our user interface that shows our A2HS button
    addBtn.style.display = 'none';
    // Show the prompt
    deferredPrompt.prompt();
    // Wait for the user to respond to the prompt
    deferredPrompt.userChoice.then((choiceResult) => {
      if (choiceResult.outcome === 'accepted') {
        console.log('User accepted the A2HS prompt');
      } else {
        console.log('User dismissed the A2HS prompt');
      }
      deferredPrompt = null;
    });
  });
});
```

Effets (en mode "pré-test") avec "Chrome-Desktop" sous windows :

**NB :**

L'événement "**beforeinstallprompt**" ne sera déclenché par un **navigateur mobile récent** que si les conditions suivantes sont réunies :

- L'appli (PWA) n'est pas déjà installée
- "user engagement heuristic" (l'utilisateur a activement utilisé l'appli au moins 30s)
- L'appli (pwa) comporte le fichier "web app manifest".
- L'appli (pwa) est servie par un serveur sécurisé (https).
- Au moins un "service worker" est enregistré avec un gestionnaire pour l'événement fetch .

Nb : A la fin de "pré-test" avec "Chrome-desktop" on pourra éventuellement (en http et pas https) , rendre visible le bouton dès le début (pas de addBtn.style.display = 'none') .

3. Service-worker et pwa pour Angular

Les versions récentes (ex : 6, 7) du framework "Angular" de Google prennent maintenant en charge les aspects "pwa" et "service-worker" d'une manière bien intégrée (via "NGSW") .

Pour tester les aspects "pwa" et "service-worker" du framework Angular , il faudra :

- effectuer les bonnes configuration (génération via ng add @.../pwa , édition ngs-w-config.json,)
- construire une version de production via "ng build --prod"
- installer l'application construite sur un serveur http (ex : nginx ou http-server ou ...)
- effectuer des tests via "Chrome Desktop" ou depuis un navigateur mobile (fonctionnant par exemple sur un système Android réel ou simulé/émulé) .

3.1. initialisation "pwa / sw" appli angular

ng add @angular/pwa

Cette commande (de angular-cli) permet d'ajouter tout un tas d'éléments "pwa" et "service-worker" à une application angular :

- ajout du package **@angular/service-worker** dans **package.json** et configurations associées

- ajout de ***manifest.json*** à la page *index.html* (+ icônes associés dans *src/assets/icons*)
- création du fichier de configuration ***ngsw-config.json*** (paramétrage des caches)
- le flag "***serviceWorker*** : ***true*** positionné dans *angular.json* permet de préciser qu'il faudra utiliser le fichier *ngsw-worker.js* comme worker en mode production avec le fichier de paramétrage *ngsw.json* (qui sera généré lors du "*ng build --prod*").

Attention :

La commande "*ng add @angular/pwa*" fonctionne bien sur une application simple (qui vient d'être générée par exemple par "*ng new myapp*") mais peut ne pas bien fonctionner sur un projet angular complexe (où beaucoup d'ajout personnalisés ont déjà été réalisés : *ng-bootstrap* par exemple).

Il vaut donc mieux déclencher cette commande assez tôt (avant d'ajouter d'autres extensions à *package.json*).

Exemple de dépendance "npm" ajoutée dans *package.json* :

```
"dependencies": {
  ...
  "@angular/pwa": "^0.12.4",
  ...
  "@angular/service-worker": "^7.2.0"
}
```

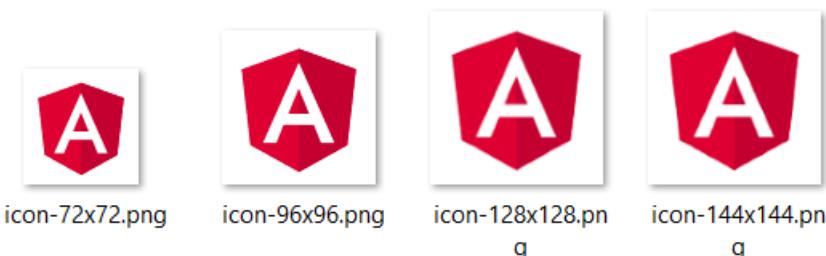
exemple de fichier ***manifest.json*** généré :

manifest.json ou *manifest.webmanifest*

```
{
  "name": "myapp",
  "short_name": "myapp",
  "theme_color": "#1976d2",
  "background_color": "#fafafa",
  "display": "standalone",
  "scope": "/",
  "start_url": "/",
  "icons": [
    {
      "src": "assets/icons/icon-72x72.png",
      "sizes": "72x72",
      "type": "image/png"
    },
    {
      "src": "assets/icons/icon-96x96.png",
      "sizes": "96x96",
      "type": "image/png"
    },
    {
      "src": "assets/icons/icon-128x128.png",
      "sizes": "128x128",
      "type": "image/png"
    },
    {
      "src": "assets/icons/icon-144x144.png",
      "sizes": "144x144",
      "type": "image/png"
    }
  ]
}
```

```
{
  "src": "assets/icons/icon-152x152.png",
  "sizes": "152x152",
  "type": "image/png"
},
{
  "src": "assets/icons/icon-192x192.png",
  "sizes": "192x192",
  "type": "image/png"
},
{
  "src": "assets/icons/icon-384x384.png",
  "sizes": "384x384",
  "type": "image/png"
},
{
  "src": "assets/icons/icon-512x512.png",
  "sizes": "512x512",
  "type": "image/png"
}
]
```

· pwa-app > src > assets > icons



exemple de lien ajouté dans index.html :

```
<link rel="manifest" href="manifest.json">
...
<meta name="theme-color" content="#1976d2">
```

Eléments ajoutés dans le module principal de l'application (*app.module.ts*) :

```
...
import { ServiceWorkerModule } from '@angular/service-worker';

@NgModule({
declarations: [
AppComponent
],
imports: [
```

```
...
ServiceWorkerModule.register('/ngsw-worker.js',
    { enabled: environment.production })
],
```

3.2. Configuration du cache (ngsw-config.json)

Le fichier `src/ngsw-config.json` (pris en compte lors de la construction d'une application via `ng build --prod`) permet de préciser quels sont les éléments à placer en cache et les stratégies de mise à jour.

Dans ce fichier les chemins doivent commencer par "/" et seront interprétés en relatif vis à vis de la racine des fichiers sources (placés dans src puis déplacés dans les bundles de production) .

Sauf indication contraire, les expressions des chemins sont basés sur le format "**glob**" (pas de `regexp`):

- `**` matches 0 or more path segments.
- `*` matches 0 or more characters excluding /.
- `?` matches exactly one character excluding /.
- The `!` prefix marks the pattern as being negative, meaning that only files that don't match the pattern will be included.

Example patterns:

- `/**/*.{html}` specifies all HTML files.
- `/*.{html}` specifies only HTML files in the root.
- `!/**/*.{map}` exclude all sourcemaps.

Groupes d'éléments en cache :

assetGroups	fichiers statiques (pour une version précise de l'application) , ex : icônes , images , ... , avec stratégies "prefetch" ou "lazy"
dataGroups	fichiers (souvent "json") de données fabriqués dynamiquement (souvent via des web services "rest") . paramétrages de type "maxAge" , ...

Modes d'installation en cache (pour un des assetGroups):

prefetch : dès le début (au premier démarrage de l'application) , les ressources sont téléchargées et stockées en cache .

lazy : au fur et à mesure des requêtes déclenchées . Les ressources ne sont placées en cache que si elles ont été téléchargées via le fonctionnement normal de l'application piloté par l'utilisateur .

La valeur par défaut de "**installMode**" est "prefetch" .

Modes de mise à jour du cache (pour un des assetGroups):

prefetch : dès le début (au premier rechargement de l'application modifiée) , les ressources modifiées sont téléchargées et stockées en cache .

lazy : au fur et à mesure des requêtes déclenchées . Les ressources ne sont remplacées en cache que si elles ont été téléchargées via le fonctionnement normal de l'application piloté par l'utilisateur .

La valeur par défaut de "**updateMode**" est la valeur de "**installMode**" .

Expression des chemins :

files=... pour les éléments internes à l'application (ex : icônes et petites images dans assets)

urls=... pour les éléments externes à l'application (ex : CDN pour css, fonts, ... et images téléchargées via http)

Paramétrages pour un des dataGroups :

maxSize	nombre maxi d'éléments (réponses) stockés en cache . éviction si taille dépassée
maxAge	durée maxi de validité en cache (ex : 3d12h)
timeout (paramètre facultatif)	durée d'attente d'une réponse réseau (en mode "online" dégradé) avant d'utiliser le contenu du cache en plan B (ex : 5s30u)
strategy	"performance" (pour données évoluant peu) ou "freshness" (pour données importantes à rafraîchir souvent)

unités :

- d: days
- h: hours
- m: minutes
- s: seconds
- u: milliseconds

Exemple de fichier src/**ngsw-config.json** :

```
{
  "index": "/index.html",
  "assetGroups": [
    {
      "name": "app",
      "installMode": "prefetch",
      "resources": {
        "files": [
          "/favicon.ico",
          "/index.html",

```

```
    "/*.css",
    "/*.js"
  ]
}
},
{
  "name": "assets",
  "installMode": "lazy",
  "updateMode": "prefetch",
  "resources": {
    "files": [
      "/assets/**",
      "/*.(eot|svg|cur|jpg|png|webp|gif|otf|ttf|woff|woff2|ani)"
    ]
  }
}
]
```

ajouts classiques :

```
{  
...  
"dataGroups": [  
{  
  "name": "xy-api",  
  "urls": [  
    "https://www.mycompany.com/xy"  
  ],  
  "cacheConfig": {  
    "maxSize" : 10000,  
    "maxAge" : "7d" ,  
    "strategy" : "freshness" ,  
    "timeout" : "5s"  
  }  
, {  
  .... (avec "strategy" : "performance" et autre(s) url(s))  
}  
]  
}
```

3.3. Mise en "pseudo-production" pour effectuer un test

installation (en mode global) via npm du serveur http-server :

```
npm install -g http-server
```

Génération de l'appli avec ses bundles de production dans le répertoire "dist" :

```
ng build --prod
```

Lancement du serveur http avec la prise en charge de l'application angular :

```
http-server -c-1 dist\myapp\
```

NB : l'option **-c-1** signifie "désactiver les caches coté serveur http"

l'url par défaut est `http://localhost:8080`

l'option `--proxy http://localhost:8282 ./api-rest-xy` permet (si besoin) de configurer une redirection vers une api rest

3.4. Fonctionnalités offertes par ServiceWorkerModule

Les fonctionnalités apportées par le module facultatif *ServiceWorkerModule* sont plutôt secondaires mais peuvent cependant constituer un plus au niveau de l'application .

Via le service **SwUpdate** :

- Savoir si une nouvelle mise à jour est disponible .
- Demander une activation de la mise à jour .
- ...

Via le service **SwPush** :

- ...
- ...

3.5. Notifications sur mises à jour disponibles et activées

```
@Injectable()
export class LogUpdateService {

constructor(updates: SwUpdate) {

  updates.available.subscribe(event => {
    console.log('current (running) version is', event.current);
    console.log('available (waiting) version is', event.available);
  });

  updates.activated.subscribe(event => {
    console.log('old version was', event.previous);
    console.log('new (activated) version is', event.current);
  });
}
}
```

autre exemple :

```
...
export class AppComponent implements OnInit {

constructor(private swUpdate: SwUpdate) {

}

ngOnInit(){
  if(this.swUpdate.isEnabled){
    swUpdate.available.subscribe(event => {
      if ( confirm("new version available , would you like to reload it ?") ) {
        window.location.reload();
      }
    });
  }
}
```

et d'autres fonctionnalités de ce genre à étudier sur le site de référence d'angular
<https://angular.io/guide/service-worker-communications>

3.6. Exemple d'utilisation du service SwPush

...

3.7. Exemple d'enrichissement personnalisé de ngsw

Dans certains cas pointus , il sera peut être nécessaire de faire cohabiter l'implémentation pré définie du service-worker de angular avec certaines extensions personnalisées .

Pour atteindre cet objectif , le mode opératoire (à éventuellement ajuster en fonction des besoins) est le suivant :

sw-custom.js (à ajouter dans src)

```
(function () {
  'use strict';

  self.addEventListener('...', (event) => {
    ...
  });
})();
```

sw-master.js (à ajouter dans src)

```
importScripts('./ngsw-worker.js');
importScripts('./sw-custom.js');
```

Enregistrer le nouveau fichier **sw-master.js** dans les "assets" de "build" de **angular.json** :

```
...
"assets": [
  "src/favicon.ico",
  "src/assets",
  "src/manifest.json",
  "src/sw-master.js"
]
...
```

Référencer **sw-master.js** plutôt que le prédefini **ngsw-worker.js** dans **app.module.ts** :

```
...
ServiceWorkerModule.register('/sw-master.js', { enabled: environment.production })
...
```

XXI - Annexe – mode déconnecté et IndexedDB

1. Mode "offLine" et indexed-db

1.1. Gestion de online/offline par les navigateurs

La plupart des navigateurs détectent et gère le mode "déconnecté" de la manière suivante :

- la propriété booléen `window.navigator.onLine` est automatiquement fixée par le navigateur pour indiquer si la connexion à internet est établie ou coupée .

Les événements "online" et "offline" sont automatiquement déclenchés par le navigateur en cas de basculement / changement d'état .

1.2. exemple de service angular "*OnlineOfflineService*"

```
...
@Injectable({ providedIn: 'root' })
export class OnlineOfflineService {
  public connectionChanged = new BehaviorSubject<boolean>(window.navigator.onLine);
  get isOnline() { return window.navigator.onLine; }
}

constructor() {
  window.addEventListener('online', () => this.updateOnlineStatus());
  window.addEventListener('offline', () => this.updateOnlineStatus());
}

private updateOnlineStatus() {
  console.log("onLine=" + window.navigator.onLine);
  this.connectionChanged.next(window.navigator.onLine);
}
```

Exemple d'utilisation :

```
export class FooterComponent implements OnInit {
  private onLine:boolean;
  constructor( private onlineOfflineService: OnlineOfflineService) {}
  ngOnInit() { this.onlineOfflineService.connectionChanged
    .subscribe( (onLine)=>{this.onLine = onLine;})
  }
}
```

et `onLine={{onLine}}` coté .html

2. IndexedDB et idb

2.1. Présentation de IndexDB , idb et liens webs (documentations)

Tout comme WebSQL/SQLite et localStorage, **IndexedDB** est une **technologie de persistance (base de données) intégrée dans les navigateurs récents (html5)** .

LocalStorage est basique et fonctionne en mode "key-value pairs".

Depuis 2010 WebSQL/SQLite est considéré comme "déconseillé/obsolète" car moins bien que IndexedDB .

IndexedDB permet de directement stocker et recharger des objets "javascript" depuis une zone de stockage persistante gérée par le navigateur .

Documentations sur IndexedDB (de base) fourni sans "Promise" par un navigateur:

- https://developer.mozilla.org/fr/docs/Web/API/API_IndexedDB/Using_IndexedDB
- <https://javascript.info/indexedb>

Bibliothèque "idb" (pour code avec "Promise"):

- <https://www.npmjs.com/package/idb>
- <https://github.com/jakearchibald/idb>

NB: idb est une **api d'un peu plus haut niveau** (basée sur des "**Promise**") qui permet de manipuler plus simplement l'api de bas niveau "IndexedDB" (fourni de façon normalisée par les navigateurs).

Documentation sur IndexedDB (avec api supplémentaire "idb" retournant "Promise") et concepts

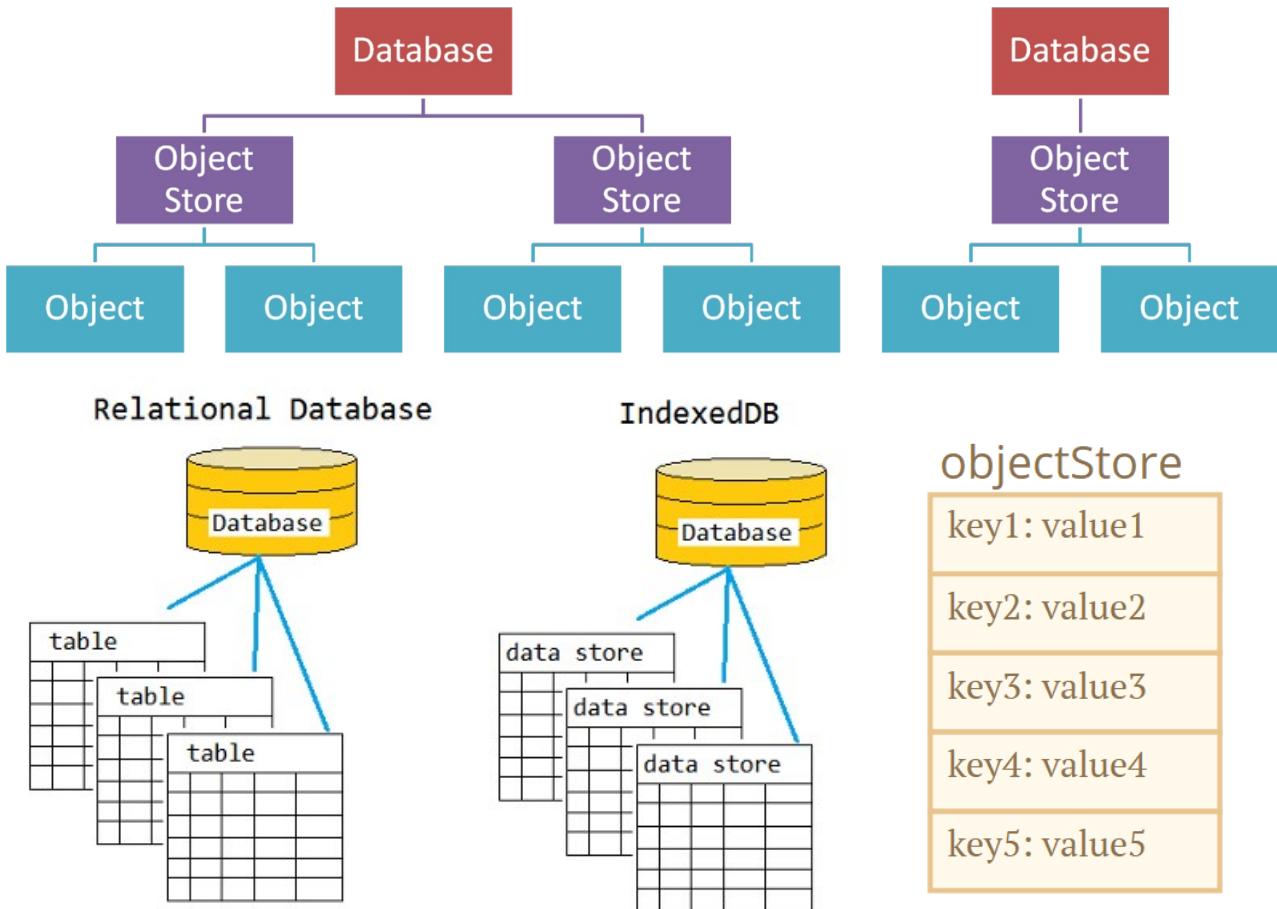
bien expliqués:

- <https://developers.google.com/web/ilt/pwa/working-with-indexedb> (*attention: ancienne version*)

Exemple IndexedDB avec Promises et async/await:

- <https://medium.com/@filipvitas/indexedb-with-promises-and-async-await-3d047ddd313>

2.2. Structure de IndexedDB



2.3. Utilisation de IndexedDB via idb (avec "Promise") :

```
if (!('indexedDB' in window)) {
  console.log('This browser doesn\'t support IndexedDB');
  return;
}
```

```
npm install -s idb
```

(ex : version 4.0.4)

Les exemples de code (partiels) ci-après seront en typescript et intégrés au framework "angular" .

```

import { Observable, of , from} from 'rxjs';
import { openDB , IDBDatabase} from 'idb';

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

    private currentIdb : IDBDatabase = null;

    //méthode asynchrone pour ouvrir la base 'my-idb'
    //en créant si besoin l' objectStore 'products' avec options :

    private openMyIDB() : Promise<IDBDatabase>{
        var dbPromise = openDB('my-idb', 1 /* version */, {
            upgrade(upgradeDb, oldVersion, newVersion, transaction) {
                if (!upgradeDb.objectStoreNames.contains('products')) {
                    upgradeDb.createObjectStore('products', {keyPath: '_id', autoIncrement: false});
                }
            },
        });
        return dbPromise;
    }
}

```

```

//accessMyIDB() return either already open idb or newly open idb if necessary:
// do not call .close() after calling accessMyIDB() !!!
private accessMyIDB() : Promise<IDBDatabase>{
    return new Promise ((resolve,reject)=> {
        if(this.currentIdb !=null){
            resolve(this.currentIdb);
        } else{
            this.openMyIDB().then(
                (db)=>{
                    if(db!=null){
                        this.currentIdb = db; resolve(db);
                    }else{
                        reject("db is null after trying openMyIdb() in accessMyIdb()")
                    }
                } ,
                (err)=>{ console.log(err); reject(err);}
            );
        }
    });
}

```

```

private getAllProductsPromise() : Promise<Product[]>{
    let dbPromise = this.accessMyIDB();
    return dbPromise.then(function(db) {
        var tx = db.transaction('products', 'readonly');
        var store = tx.objectStore('products');

```

```

    return store.getAll();
}
}
```

```

public getProducts() : Observable<Product[]> {
    return from(this.getAllProductsPromise()); //from() to convert Promise to Observable
}
```

```

private addProductInMyIDbPromise(p:Product):Promise<any>{
    let dbPromise = this.accessMyIDB();
    return dbPromise.then(function(db) {
        let tx = db.transaction('products', 'readwrite');
        let store = tx.objectStore('products');
        store.add(p);
        return tx.done;
    });
}
```

```

private updateProductInMyIDbPromise(p:Product):Promise<any>{
    let dbPromise = this.accessMyIDB();
    return dbPromise.then(function(db) {
        let tx = db.transaction('products', 'readwrite');
        let store = tx.objectStore('products');
        store.put(p);
        return tx.done;
    });
}
```

```

private deleteProductInMyIDbPromise(id:string):Promise<any>{
    let dbPromise = this.accessMyIDB();
    return dbPromise.then(function(db) {
        let tx = db.transaction('products', 'readwrite');
        let store = tx.objectStore('products');
        store.delete(id);
        return tx.done;
    });
}
```

```

private memProductlist : Product[] =
[ { _id : "p1" , category : "divers" , price : 1.3 , label : "gomme" , description : "gomme blanche" },
{ _id : "p10" , category : "livres" , price : 12.1 , label : "A la recherche du temps perdu" ,
description : "Marcel Proust" } ];
```

```

private async initMyIdbSampleContent(){
    let db = await this.openMyIDB(); //not accessMyIDB() since .close() at the end of this aync function
    let tx = db.transaction('products', 'readwrite');
    let store = tx.objectStore('products');
    for(let p of this.memProductlist){
        let exitingProdWithSameKey = await store.get(p._id);
        if(exitingProdWithSameKey==null){
            await store.add(p); //reject Promise and tx if p already in store
        }
    }
}
```

```
    await tx.done();  db.close();
}
}
```

XXII - Annexe – socket.io

1. Socket.io

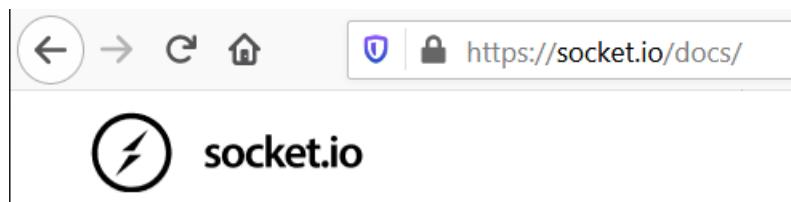
1.1. Présentation de Socket.io

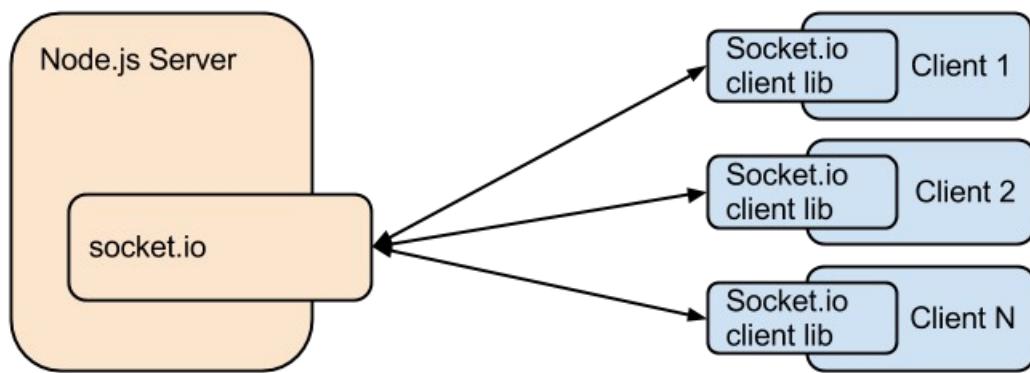
Socket.io est une **api javascript** qui encapsule et améliore la prise en charge des "websockets" .

Rappel : les **websockets** sont une technologie permettant d'établir des ***communications bidirectionnelles*** via un **canal construit au dessus du protocole HTTP** et cela permet au coté serveur d'envoyer spontanément des informations (ou événements) au client (fonctionnant dans un navigateur). C'est une des technologies de "**push**" .

Valeurs ajoutés par la bibliothèque javascript "socket.io" :

websocket (utilisé seul)	socket.io
protocole (lui même basé sur tcp et http) maintenant géré par la plupart des navigateurs et pouvant être manipulé en code javascript de bas niveau	librairie javascript complémentaire (à télécharger et utiliser)
fourni un canal de communication full-duplex de bas niveau	fourni un canal abstrait de communication full-duplex (de plus haut niveau) basé sur des événements .
proxy http et load-balancer ne sont pas gérés par les websockets ==> gros problème / limitation	les communications peuvent êtres établies même en présence de proxy http et de load-balancer (en mettant en oeuvre des mécanismes supplémentaires pour compenser les limitations des "websockets" généralement utilisées en interne)
ne gère pas le broadcasting	gère (si besoin) le broadcasting
pas d'option pour le "fallback"	comporte des options pour le "fallback"





documentation sur site officiel de Socket.io (présentation, concepts):

- <https://socket.io/docs/>

bibliothèque socket.io:

- <https://www.npmjs.com/package/socket.io-client>
- <https://www.npmjs.com/package/@types/socket.io-client>

1.2. chat-socket.io coté serveur (en version "typescript")

package.json

```
{
  "name": "chat-socket-io",
  "version": "0.1.0",
  "dependencies": {
    "ent": "^2.2.0",
    "express": "^4.17.1",
    "socket.io": "^2.2.0"
  },
  "description": "Chat temps réel avec socket.io",
  "devDependencies": {
    "@types/ent": "^2.2.1",
    "@types/express": "^4.17.0",
    "@types/socket.io": "^2.1.2"
  }
}
```

chat-socket-io/src/app.ts

```
import express , { Request, Response } from 'express';
import * as http from 'http';
import * as ent from 'ent';// Permet de bloquer les caractères HTML
import * as sio from 'socket.io';

const app :express.Application = express();
const server = http.createServer(app);
const io = sio.listen(server);

//les routes en /html/... seront gérées par express
//par de simples renvois des fichiers statiques du répertoire "/html"
app.use('/html', express.static(__dirname+"/html"));

app.get('/', function(req :Request, res : Response ) {
  res.redirect('/html/index.html');
});

//events: connection, message , disconnect and custom_event like nouveau_client

io.sockets.on('connection', function (socket:any) {
  // Dès qu'on nous donne un pseudo,
  // on le stocke en variable de session/socket et on informe les autres personnes :
  socket.on('nouveau_client', function(pseudo:string) {
    pseudo = ent.encode(pseudo);
    socket.pseudo = pseudo;
    socket.broadcast.emit('nouveau_client', pseudo);
  });

  // Dès qu'on reçoit un message, on récupère le pseudo de son auteur
  //et on le transmet aux autres personnes
  socket.on('message', function (message:string) {
    message = ent.encode(message);
  });
}
```

```

    socket.broadcast.emit('message', {pseudo: socket.pseudo, message: message});
  });
});

server.listen(8383,function () {
  console.log("http://localhost:8383");
});

```

1.3. chat-socket-io coté client (html/js)

chat-socket-io/dist/lib/socket.io.js (à télécharger)

chat-socket-io/dist/html/index.html

```

<html>
  <head>
    <meta charset="utf-8" />
    <title>Chat temps réel avec socket.io</title>
    <style>
      #zone_chat strong { color: white; background-color: black; padding: 2px; }
    </style>
  </head>

  <body>
    <h1>Chat temps réel avec socket.io (websocket)</h1>
    <p><i>(version adaptée de https://openclassrooms.com/fr/courses/1056721-des-applications-ultra-rapides-avec-node-js/1057959-tp-le-super-chat)</i></p>

    message:<input type="text" name="message" id="message" size="50" autofocus />
    <input type="button" id="envoi_message" value="Envoyer" />

    <div id="zone_chat"></div>

    <script src="lib/socket.io.js"></script>
    <script>
      var zoneChat = document.querySelector('#zone_chat');
      var zoneMessage = document.querySelector('#message');
      // Connexion au serveur socket.io :
      var socket = io.connect('http://localhost:8383');

      // On demande le pseudo, on l'envoie au serveur et on l'affiche dans le titre
      var pseudo = prompt('Quel est votre pseudo ?');
      socket.emit('nouveau_client', pseudo);
      document.title = pseudo + ' - ' + document.title;

      // Quand on reçoit un message, on l'insère dans la page
      socket.on('message', function(data) {
        insereMessage(data.pseudo, data.message)
      });

      // Quand un nouveau client se connecte, on affiche l'information :
      socket.on('nouveau_client', function(pseudo) {

```

```

zoneChat.innerHTML='<p><em>' + pseudo + ' a rejoint le Chat !</em></p>'
+zoneChat.innerHTML;
})

// Lorsqu'on click sur le bouton, on transmet le message et on l'affiche sur la page
document.querySelector('#envoi_message').addEventListener('click',function () {
    var message = zoneMessage.value;
    socket.emit('message', message); // Transmet le message aux autres
    insereMessage(pseudo, message); // Affiche le message aussi sur notre page
    zoneMessage.value=""; zoneMessage.focus(); // Vide la zone de Chat et remet le focus dessus
    return false; // Permet de bloquer l'envoi "classique" du formulaire
});

//fonction utilitaire pour ajouter un message dans la page :
function insereMessage(pseudo, message) {
    zoneChat.innerHTML='<p><strong>' + pseudo + '</strong> ' + message
    +'</p>' +zoneChat.innerHTML;
}
</script>
</body>
</html>

```

localhost:8383 indique

Quel est votre pseudo ?

OK
Annuler

Chat temps réel avec socket.io (websocket)

(version adaptée de <https://openclassrooms.com/fr/courses/1056721-des-applications-ultra-rapides-avec-node-js/1057959-tp-le-super-chat>)

message:

Envoyer

toto cc
titi bb
toto aa

toto a rejoint le Chat !

Chat temps réel avec socket.io (websocket)

(version adaptée de <https://openclassrooms.com/fr/courses/1056721-des-applications-ultra-rapides-avec-node-js/1057959-tp-le-super-chat>)

message:

Envoyer

toto cc
titi bb
toto aa

1.4. Socket.io coté client (intégré dans Angular)

npm install socket.io-client --save

src/app/common/service/chat.service.ts

```
import { Injectable } from '@angular/core';
import * as sio from 'socket.io-client';
import { Observable } from 'rxjs';

@Injectable({ providedIn: 'root' })
export class ChatService {
  private url : string = 'http://localhost:8383';
  //https://github.com/didier-mycontrib/tp_node_js (chat-socket-io)

  private socket;

  constructor() {
    this.socket = sio(this.url);
  }

  public sendMessage(message:string, eventName:string="message") {
    this.socket.emit(eventName, message);
  }

  public getMessagesObservable(eventName :string="message") : Observable<any>{
    return Observable.create((observer)=> {
      this.socket.on(eventName, (message)=> {
        observer.next(message);
      });
    });
  }
}
```

chat.component.ts

```
import { Component, OnInit } from '@angular/core';
import { ChatService } from 'src/app/common/service/chat.service';

interface EventMessageWithPseudo {
  pseudo : string;
  message : string;
}

@Component({
  selector: 'app-chat',
  templateUrl: './chat.component.html',
  styleUrls: ['./chat.component.scss']
})
export class ChatComponent implements OnInit {
  pseudo:string;//undefined by default
  pseudoSent : boolean = false;
  newMessage: string;
  messages: EventMessageWithPseudo[] = [];
}
```

```

constructor(private chatService : ChatService) { }

onSetPseudo() {
  this.chatService.sendMessage(this.pseudo, "nouveau_client");
  this.pseudoSent = true;
}

onSendMessage() {
  this.chatService.sendMessage(this.newMessage); //envoi du message (pour diffusion)
  //push() ajoute à la fin, unshift() ajoute au début :
  this.messages.unshift({pseudo:this.pseudo ,
                           message:this.newMessage});// pour affichage local du message envoyé
  this.newMessage = "";
}

ngOnInit() {
  this.chatService
    .getMessagesObservable("nouveau_client")
    .subscribe((username: string) => {
      this.messages.unshift({pseudo:username , message:'a rejoint le Chat !'});
    });
}

this.chatService
  .getMessagesObservable("message")
  .subscribe((evtMsgWithPseudo: any) => {
    this.messages.unshift(evtMsgWithPseudo);
  });
}
}

```

chat.component.html

```

<p>chat whith socket.io</p>
<div [style.display]="pseudoSent?'none':'block'">
  pseudo:<input [(ngModel)]="pseudo" /> &nbsp;
  <button (click)="onSetPseudo()">set & send</button>
</div>
<div [style.display]="pseudoSent?'block':'none'">
  pseudo : <b>{{pseudo}}</b><br/>
  message:<input [(ngModel)]="newMessage"
                (keyup)="$event.keyCode == 13 && onSendMessage()"/>(press Enter)
  <hr/>
  <div *ngFor="let evtMsgWithPseudo of messages">
    <p><span class="pseudo_chat">{{evtMsgWithPseudo.pseudo}}</span>
       {{evtMsgWithPseudo.message}}</p>
  </div>
</div>

```

chat.component.css

```
.pseudo_chat { color: white; background-color: black; padding: 2px;}
```

