
Angular

~~v2,v4,v5~~, v6 - v15
(avec npm et typescript)

Table des matières

I - Présentation de Angular.....	5
1. Présentation du framework web Angular.....	5
II - Environnement de développement Angular.....	14
1. Environnement de développement pour Angular.....	14
2. Anatomie élémentaire d'un composant angular.....	25
3. Arborescence de composants (TD).....	26
III - Langage typescript.....	28
1. Renvois selon le contexte de la formation.....	28
2. javascript pour angular.....	29
3. typescript pour angular.....	29
IV - Essentiel sur templates , bindings , events.....	30

1. Anatomie d'un composant angular.....	30
2. Templates bindings (property , event).....	33
V - Switch et routing essentiel (navigation).....	43
1. Switch élémentaire de sous composants.....	43
2. Bases élémentaires du routing angular.....	44
VI - Contrôles de formulaires (bases essentielles).....	47
1. Contrôle des formulaires (template-driven).....	47
VII - Components (angular).....	51
1. Structure d'une application angular 2+.....	51
2. Les modules applicatifs.....	53
3. Précisions sur les composants (@Component).....	56
4. Cycle de vie sur composants (et directives).....	63
5. Formatage des valeurs à afficher avec des "pipes".....	64
VIII - Services et injections (essentiel).....	66
1. Services "angular" (concepts et bases).....	66
Autre type de service classique:.....	67
SessionService (avec données de type .username .isConnected ...)	67
2. Injection de dépendances (bases).....	68
IX - Appels de W.S. REST (Observable, ...).....	70
1. Angular et dialogues HTTP/REST.....	70
2. Api HttpClient (depuis Angular 4.3).....	72
X - Routing angular (compléments importants).....	78
1. Sous niveau de routage (children).....	78
2. Routes paramétrées et navigation par code.....	79
3. Route conditionnée par gardien.....	81
4. Aperçu sur le routing angular avancé.....	83
XI - Directives.....	84
1. Aperçu sur les directives (angular2+).....	84
XII - Packaging et déploiement d'appli. angular.....	88
1. Notion de "bundle" pour le déploiement.....	88

2. reverse proxy en mode développement (ng serve).....	89
3. JIT vs AOT (Ahead-Of-Time) pour angular 2 à 8.....	91
4. ivy (à partir de angular 9).....	92
5. Mise en production d'une application angular.....	93
XIII - Aspects divers (BehaviorSubject,pipe,...).....	96
1. BehaviorSubject.....	96
2. Autres aspects divers.....	102
XIV - Tests unitaires (et ...) avec angular.....	104
1. Différent types de tests autour de angular.....	104
2. Test "end-to-end / cypress".....	105
3. Attention à la cohérence des tests.....	108
4. Tests unitaires élémentaires.....	108
5. Lancement tests "Angular" avec ng test et "karma".....	111
6. Tests unitaires "angular"(composants, service, ...).....	112
XV - Sécurité – application Angular.....	131
1. Sécurisation d'une application "angular".....	131
2. Sécurisation des appels aux Web-services REST.....	132
XVI - Annexe – RxJs.....	135
1. introduction à RxJs.....	135
2. Fonctionnement (sources et consommations).....	136
3. Convention de nom pour "return Observable".....	136
4. Organisation de RxJs.....	137
5. Sources classiques générant des "Observables".....	138
6. Principaux opérateurs (à enchaîner via pipe).....	140
7. Passerelles entre "Observable" et "Promise".....	142
8. Optimisations et précautions.....	142
XVII - Annexe – Model Driven Forms.....	143
1. Contrôle des formulaires.....	143
XVIII - Annexe – ngx-bootstrap.....	146
1. Extension "ngx-bootstrap" pour angular.....	146
XIX - Annexe – extension @angular/material.....	149

1. Angular-Material (librairie de composants).....	149
2. Essentiel de "flex-layout" (en intégration angular).....	151
3. Quelques composants "angular-material"	152

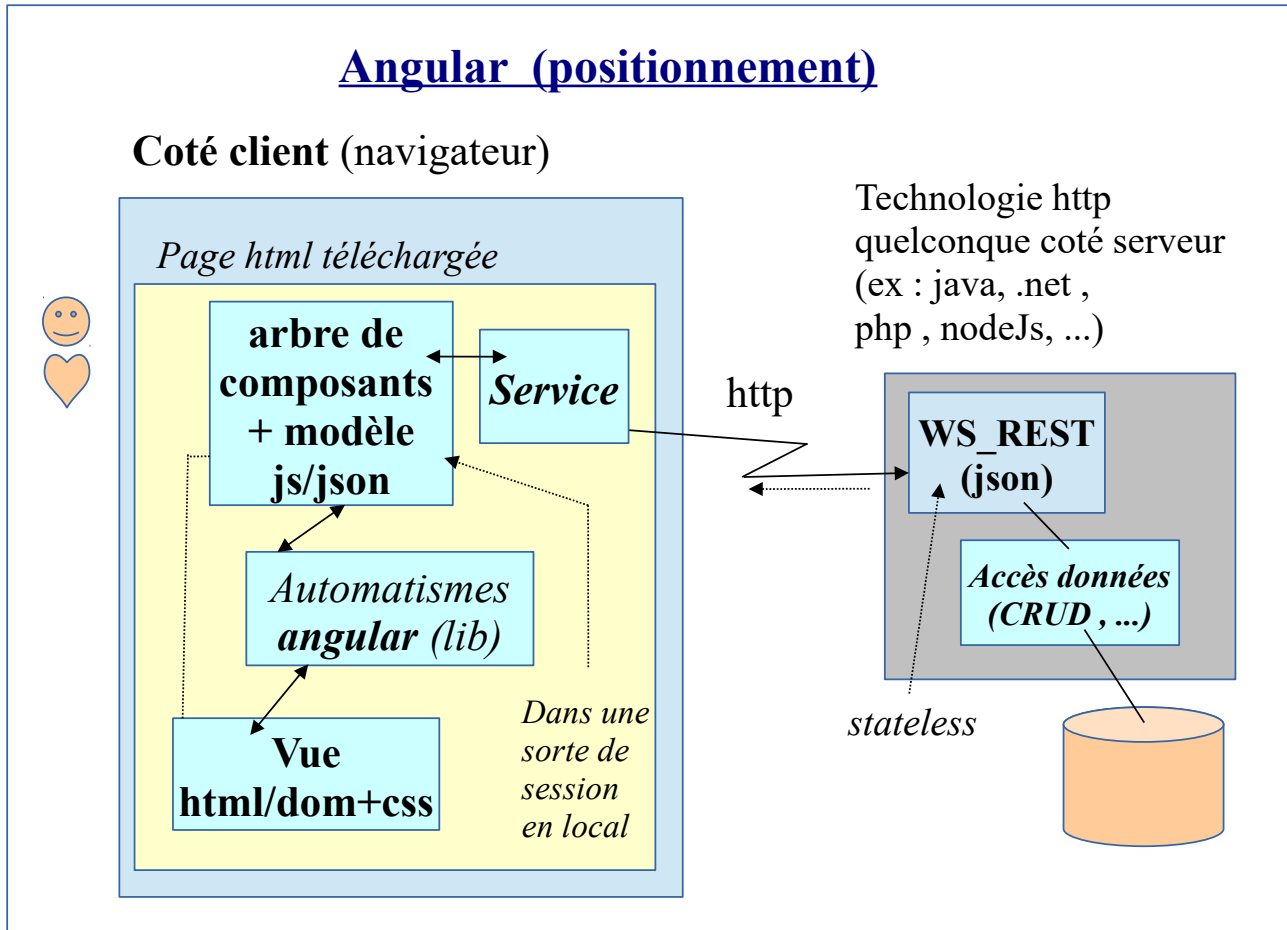
XX - Annexe – PWA (Progressive Web App).....	164
---	------------

1. PWA (Progressive Web App) – aperçu général.....	164
2. Web App Manifest / add to home screen.....	167
3. Service-worker et pwa pour Angular.....	173
4. Mode "offLine" et indexed-db.....	182
5. IndexedDB et idb.....	183
6. Socket.io.....	187

I - Présentation de Angular

1. Présentation du framework web Angular

1.1. Positionnement du framework "Angular"



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 :

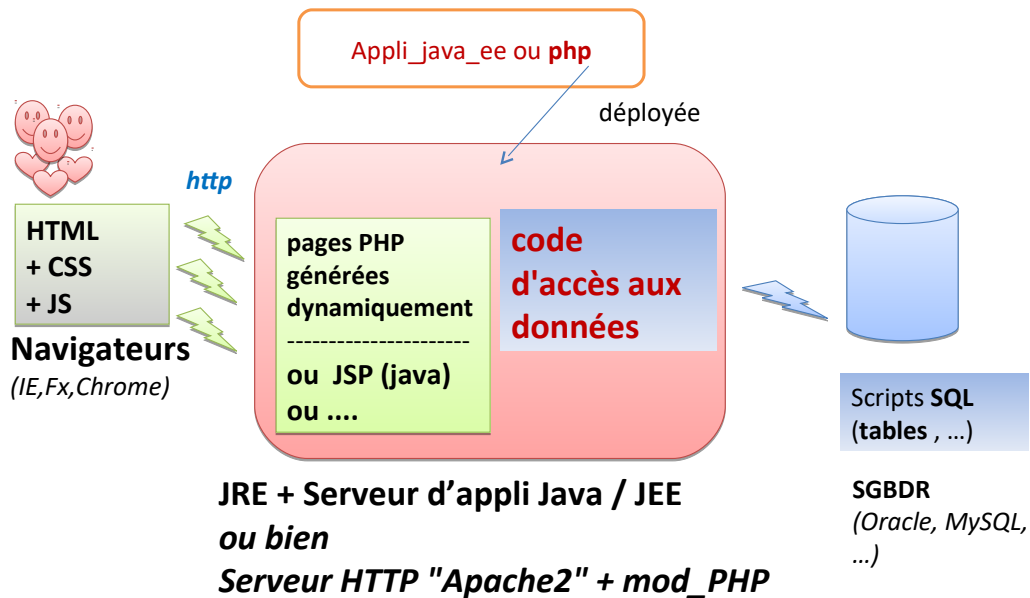
- une grande partie des traitements web s'effectue coté client (dans le navigateur) et le serveur se voit alors déchargé d'une lourde tâche (refabriquer des pages, gérer les sessions utilisateurs, ...) ---> bien pour tenir la charge.
- meilleurs performances/réactivités du coté affichage/présentation 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 .

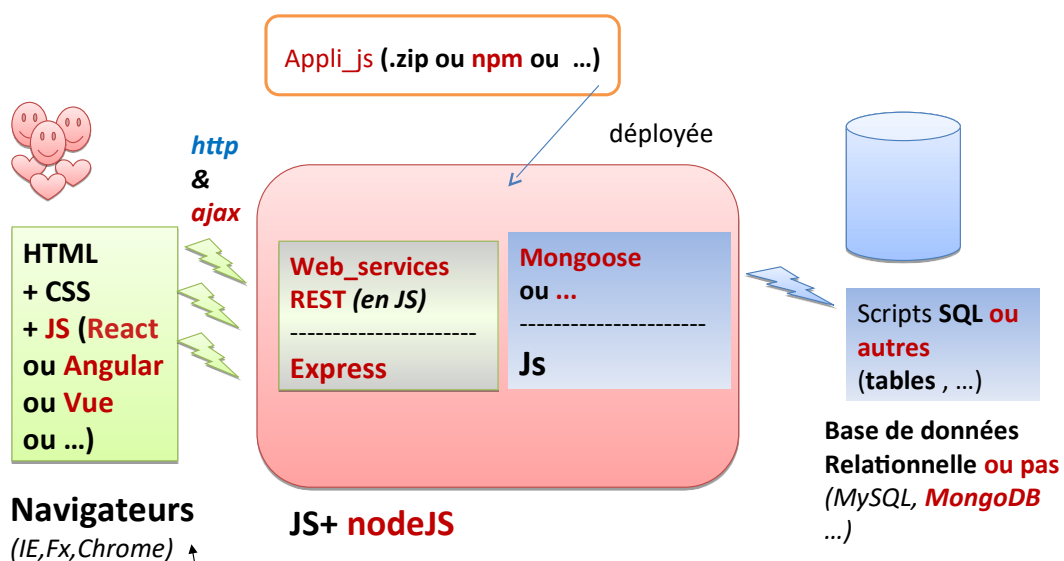
Ancienne architecture web



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 : meilleurs 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)

<i>angularJs</i> (1.x) 2012 - 2016	— <i>javascript</i> , contrôleur
angular 2 (fin 2016)	— typescript , composants, avec bugs
angular 4.0 à 4.2 (début 2017)	— moins bugs , angular-cli
angular 4.3 et 5.x (fin 2017)	— http --> httpClient
angular 6 , 7 , 8 (2018, 2019)	— RxJs et base angular enfin stable
Angular 9, ... ,13 (2020,2022)	— moteur ivy performant

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 coeur 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 .**

1.4. Binding angular

Composant angular avec **binding** automatique (*)

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


product.component.ts

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


product.component.html

label: produit xyz

prix ht: 200

tauxTva: 20 

actualiser prix ttc

prix ttc: 240 

~~[(ngModel)]~~
= "product.tauxTva"

Product (*class*)

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

{{product.ttc}}

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** .

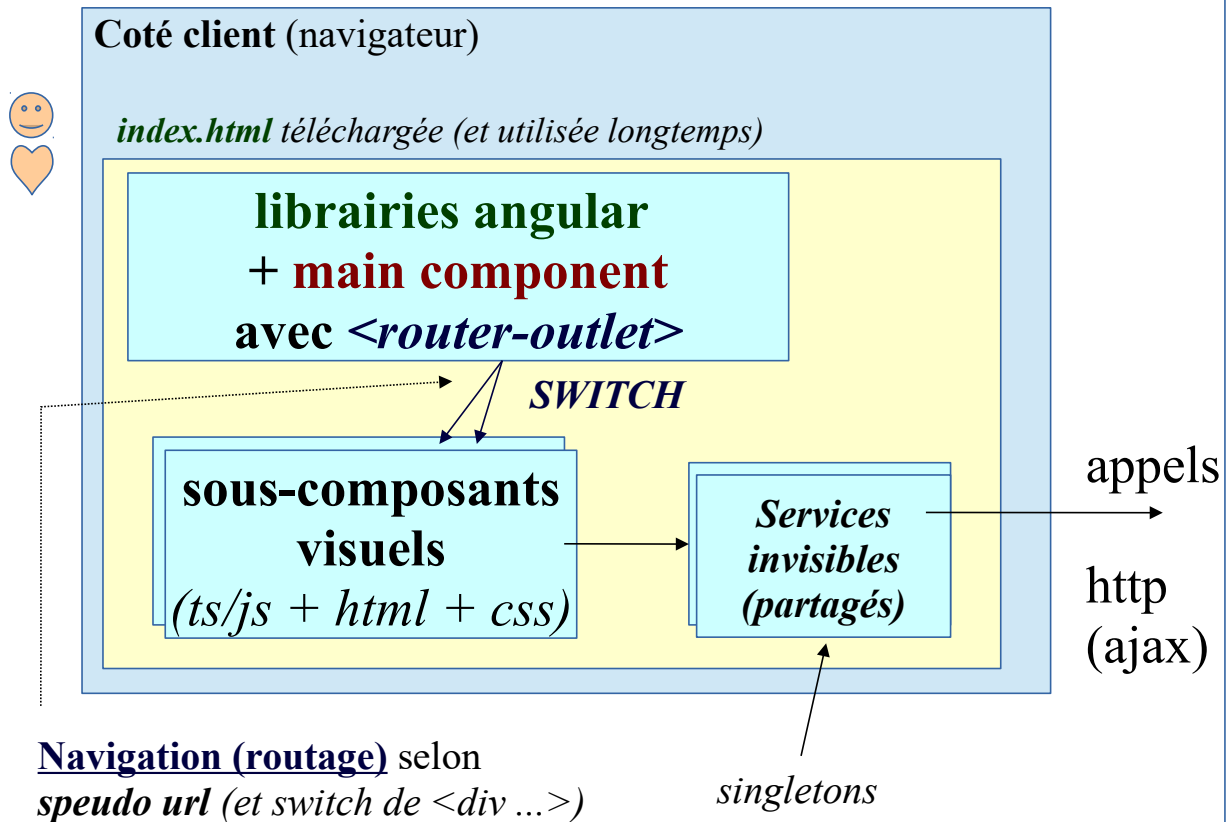
Quelque soit la version d'angular (1 , 2, 4 ou +) , le **binding automatique** entre valeurs saisies ou affichées et les valeurs des objets "javascript" constitue **la principale valeur ajoutée du framework**.

C'est le principal apport d'Angular par rapport à une application simplement basé sur jquery .

NB: AngularJs (v1.x) utilisait un binding systématiquement bidirectionnel et assez peu performant. Angular (v2, v4, ...) utilise maintenant un binding mieux contrôlé (soit unidirectionnel, soit bidirectionnel) et est plus performant.

1.5. Structure "Single Page" et routage angular

Single Page Application et switch de sous parties



De façon à ce que le code javascript (librairies angular + code de l'application) soit converti en mémoire sur le long terme, une application Angular est constituée 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.

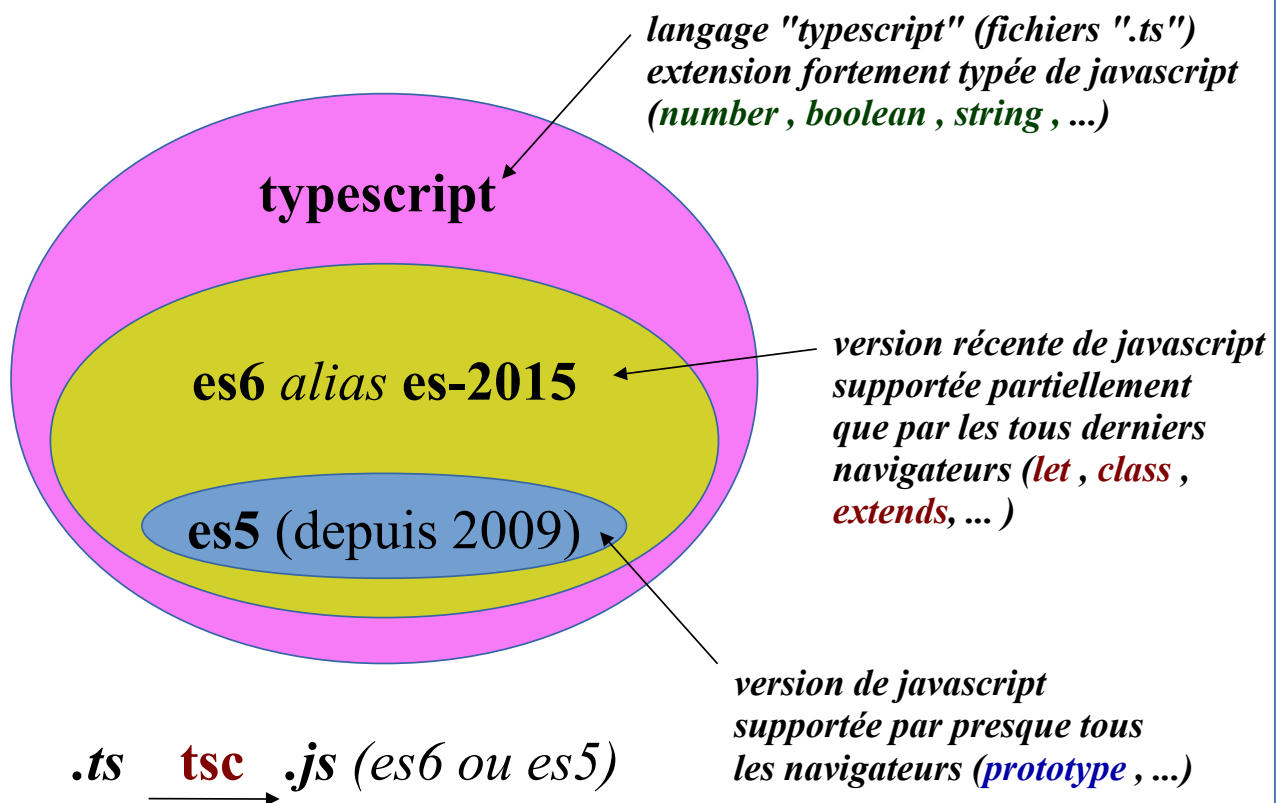
Le **switch de sous composants** sera associé à des **navigations** généralement **paramétrées dans un module de routage**.

En pouvant associer une pseudo-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" (v2, v8, ... , v14, ...)

- 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. Orientation "composants"

Contrairement à l'ancienne version 1.x , les nouvelles versions 2+ du framework angular sont clairement orientées "composants" . Il s'agit là d'une évolution récente des technologies web .

Web Component :

"Web Component" est une **spécification** (récente) du "W3C" .

"Web component" est un ensemble d'API WEB permettant de programmer et utiliser des composants personnalisés au sein de pages (ou applications) HTML.

Les 4 api fondamentales des "web component" sont :

- **Custom elements** (*pour créer et enregistrer de nouveaux éléments HTML et les faire reconnaître par le navigateur*)
- **shadow dom** (encapsulation (private/public) de "js et css")
- **es module / html imports** (modularité / packaging)
- **html template** (*squelettes/modèles de nouveau éléments HTML instanciables*)

Shadow DOM :

- Fragment d'un arbre dom (isolé de l'arbre DOM principal) .

Nouvelles balises "html" associées aux "Web Component":

```
<template>  
<slot>  
...
```

NB :

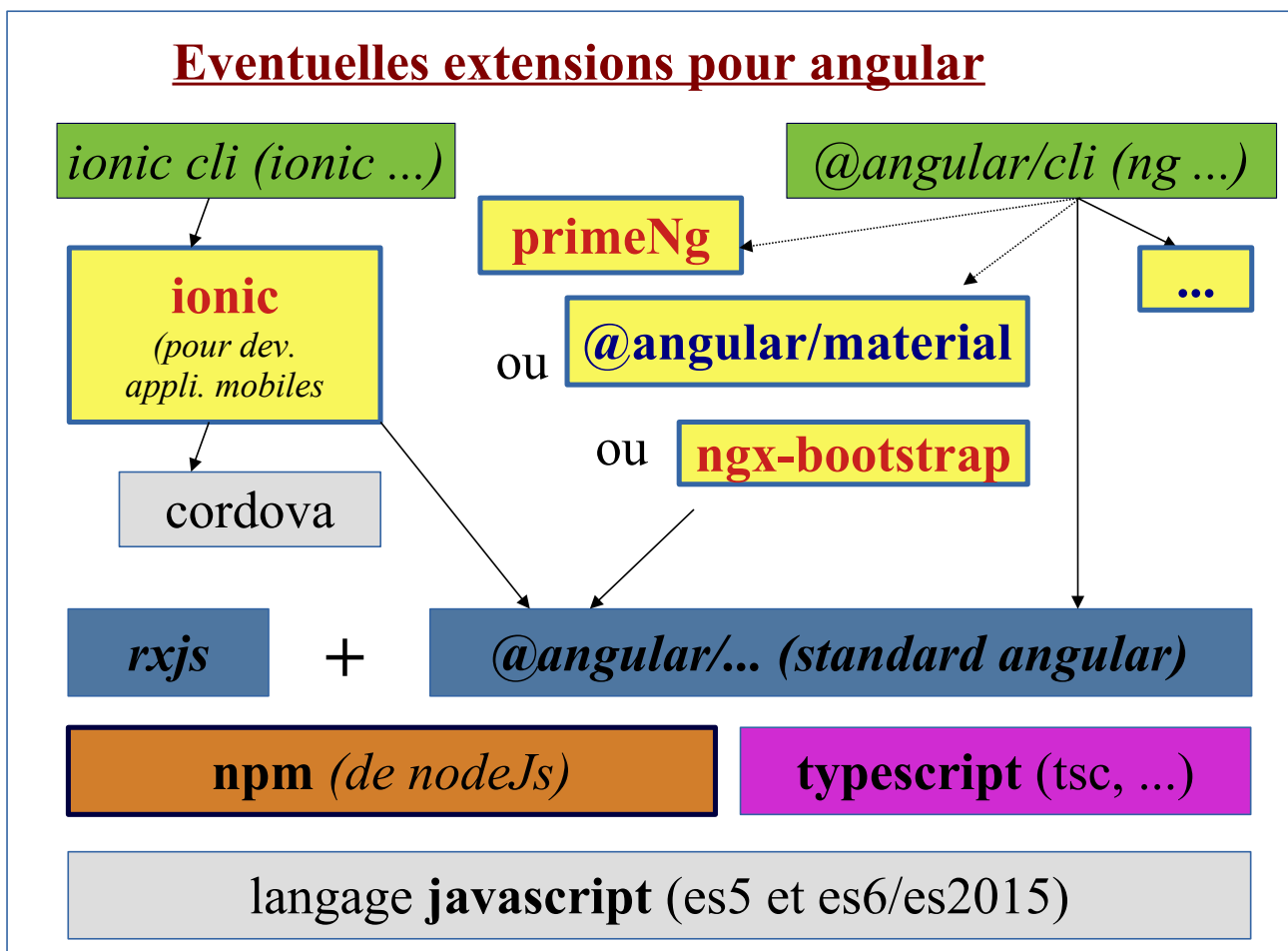
*Cette normalisation (récente) peut éventuellement encore un peu évoluer.
C'est supporté que par certains navigateurs récents.*

Quelques URLs pour approfondir le sujet :

<https://css-tricks.com/an-introduction-to-web-components/>
https://developer.mozilla.org/fr/docs/Web/Web_Components
<https://www.webcomponents.org/introduction>
<https://developers.google.com/web/fundamentals/web-components/>
...

-
- Le framework "Angular" met en oeuvre (à sa façon, sans absolument tenir compte de cette norme) la plupart des fonctionnalités des "web component" .
 - Une petite application angular de type "composant réutilisable" peut éventuellement être packagé comme un web-component portable (respectant la norme) de manière à être ensuite utilisé dans un cadre "html/javascript" classique (sans framework).
 - Dans la plupart des cas , le framework angular sera utilisé pour programmer une application complète constituée d'un assemblage de composants spécifiques "angular" .

1.8. 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) :

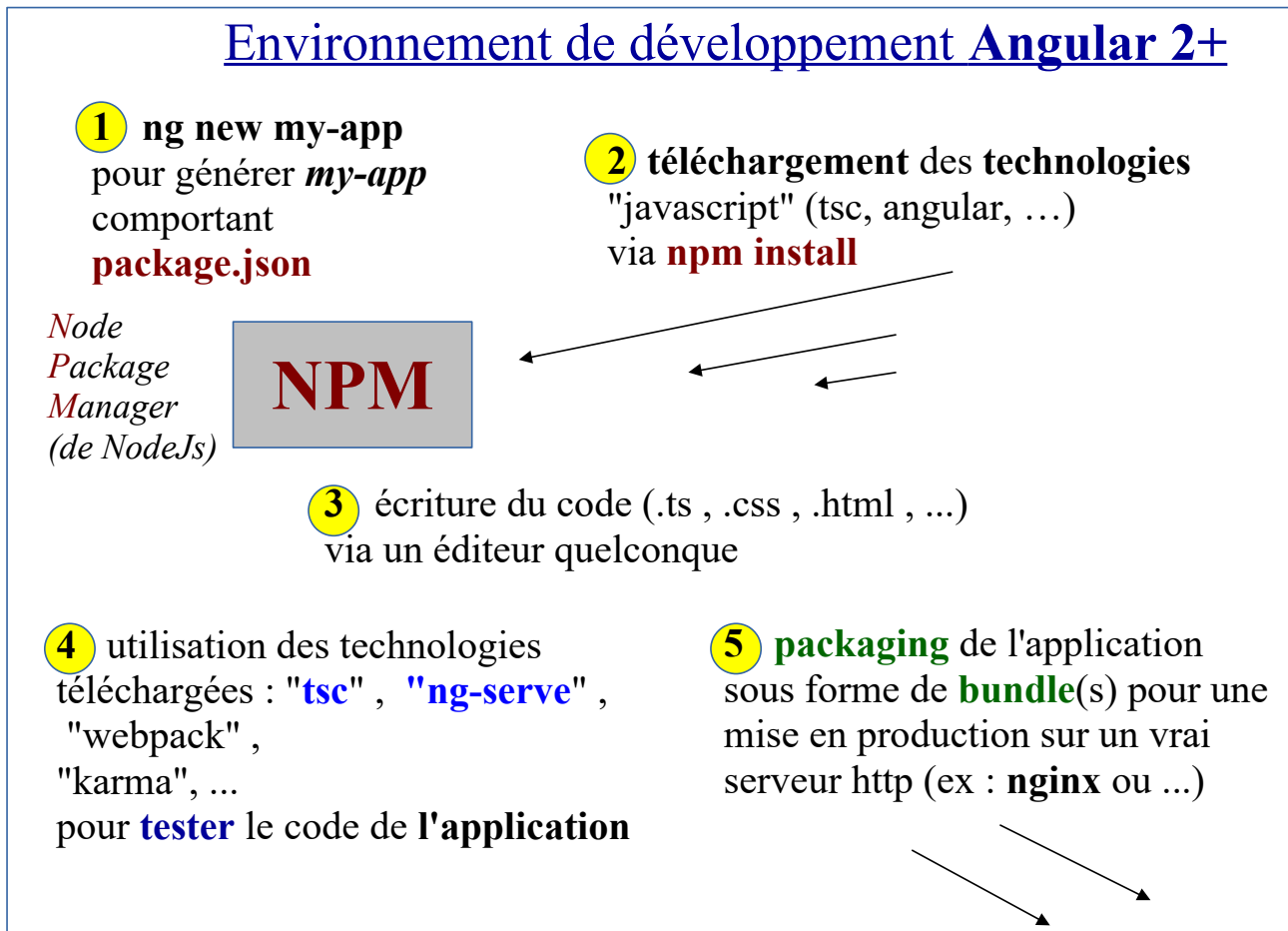
extensions	utilités
ngx-bootstrap ou bien material ou bien primeNg	Composants graphiques évolués réutilisables (paneaux , onglets , ...) <div> <div>addition</div> <div>calcul tva</div> <div> a: <input type="text" value="0"/> </div> <div> b: <input type="text" value="0"/> </div> <div>addition</div> <div>a+b: 0</div> <div> Collapsible panel (when click on title) </div> <div>Panel Body</div> </div>
ng2-charts	Affichage de diagrammes/graphiques (courbes , ...) en s'appuyant sur chart.js et l'api canvas <div> <div>Ventes selon secteurs</div> <div>  </div> <div> <div>Series A</div> <div>Series B</div>  </div> </div>
ngx-leaflet	Affichage de cartes (en s'appuyant sur openstreetmap ou autres) <div>  </div>
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 permettent d'obtenir un code lisible compact et maintenable dans un style bien "angular" .

II - Environnement de développement Angular

1. Environnement de développement pour Angular

1.1. Environnement de développement minimum (depuis v2)



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

/	dans src	dans src/app
<ul style="list-style-type: none"> dist e2e node_modules src .editorconfig angular.json package.json package-lock.json proxy.conf.json README.md tsconfig.json tslint.json 	<ul style="list-style-type: none"> 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 	<ul style="list-style-type: none"> app.component.html app.component.scss app.component.spec.ts app.component.ts app.module.ts app-routing.module.ts

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

ng new <i>my-app</i> , <i>cd my-app</i>	Création d'une nouvelle application " <i>my-app</i> " .
ng serve	Lancement de l'application en mode développement (watch & compile file , launch server,) → URL par défaut : <i>http://localhost:4200</i>
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)
ng e2e (avant version 12) installer et utiliser cypress depuis la v12	Lance les tests "end to end" / "intégration"
...	...

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
```

- choisir "y" à la question "activer le routing angular"
- choisir "scss" comme format de feuilles de styles

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**

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

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








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

- `--style css` (ou `--style scss`) 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.a5d4dde1dbf3b1d.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 :

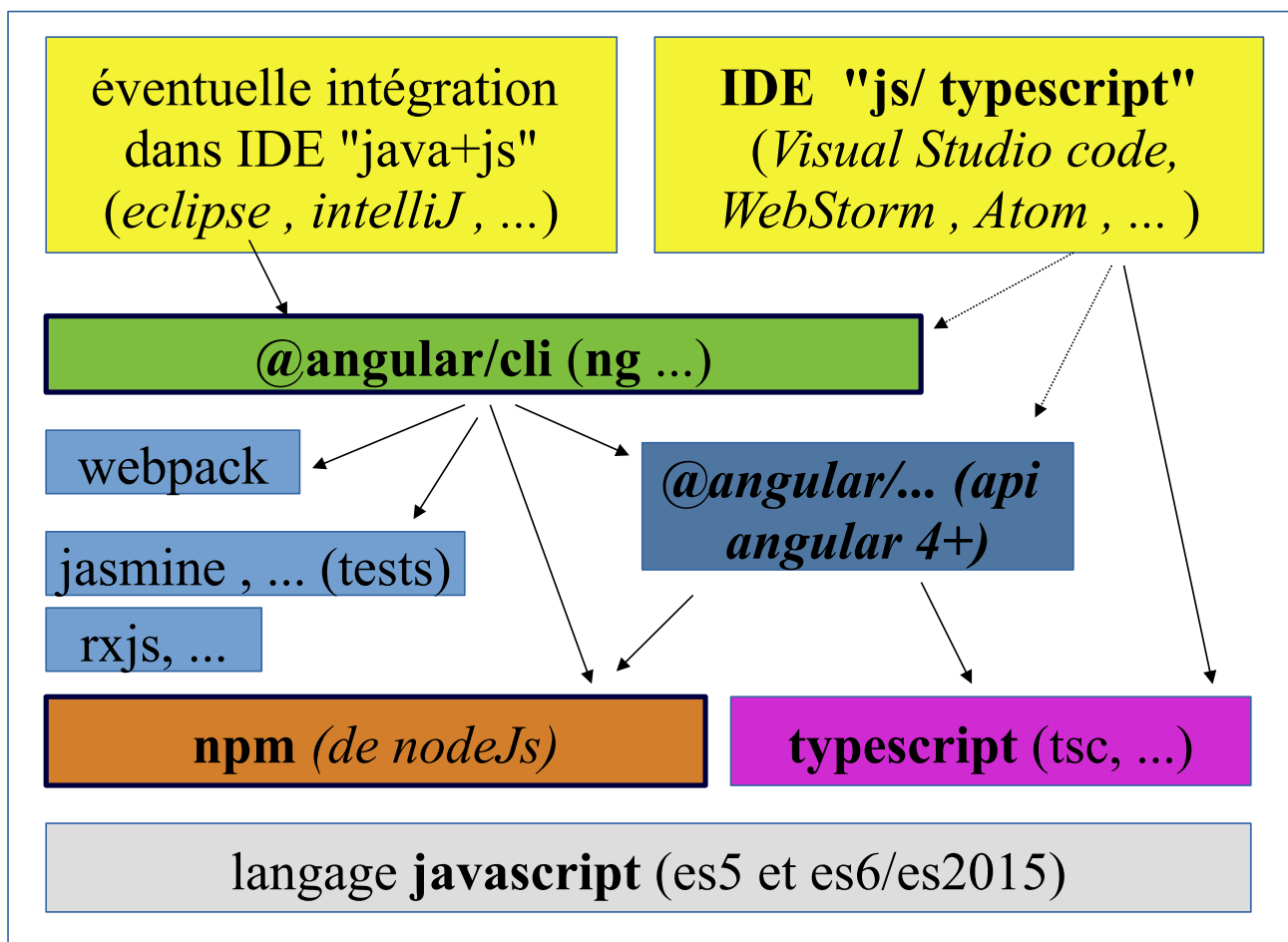
NB : si l'on souhaite utiliser conjointement certains fichiers javascripts complémentaires (exemple pas conseillé mais quelquefois compatible : "jquery...js" et "bootstrap.min.js") on peut :

- 1) placer un répertoire "js" a coté 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"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "~13.0.0",
    "@angular/common": "~13.0.0",
    "@angular/compiler": "~13.0.0",
    "@angular/core": "~13.0.0",
    "@angular/forms": "~13.0.0",
    "@angular/platform-browser": "~13.0.0",
    "@angular/platform-browser-dynamic": "~13.0.0",
    "@angular/router": "~13.0.0",
    "bootstrap": "^5.1.3",
    "ngx-bootstrap": "^8.0.0",
    "rxjs": "~7.4.0",
    "tslib": "^2.3.0",
    "zone.js": "~0.11.4"
  },
  "devDependencies": {
    "@angular-devkit/build-angular": "~13.0.4",
    "@angular/cli": "~13.0.4",
    "@angular/compiler-cli": "~13.0.0",
    "@types/jasmine": "~3.10.0",
    "@types/node": "^12.11.1",
    "jasmine-core": "~3.10.0",
    "karma": "~6.3.0",
    "karma-chrome-launcher": "~3.1.0",
    "karma-coverage": "~2.0.3",
    "karma-jasmine": "~4.0.0",
    "karma-jasmine-html-reporter": "~1.7.0",
    "typescript": "~4.4.3"
  }
}
```

NB : au sein de cet exemple , les extensions *bootstrap* et *ngx-bootstrap* ont été ajoutées 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": "node",
    "importHelpers": true,
    "target": "es2017",
    "module": "es2020",
    "lib": [
      "es2020",
      "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 :

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.log(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-répertoires "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
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81

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)

app/app.module.ts

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

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

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

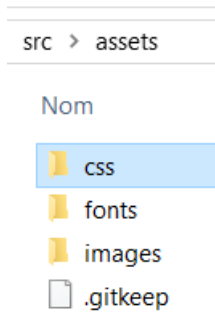
```
input.ng-valid[required] {
  border-left: 5px solid #42A948; /* green */
}
input.ng-invalid {
  border-left: 5px solid #a94442; /* red */
}
.errMsg{
  font-style: italic;
}
```

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/assets/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 :

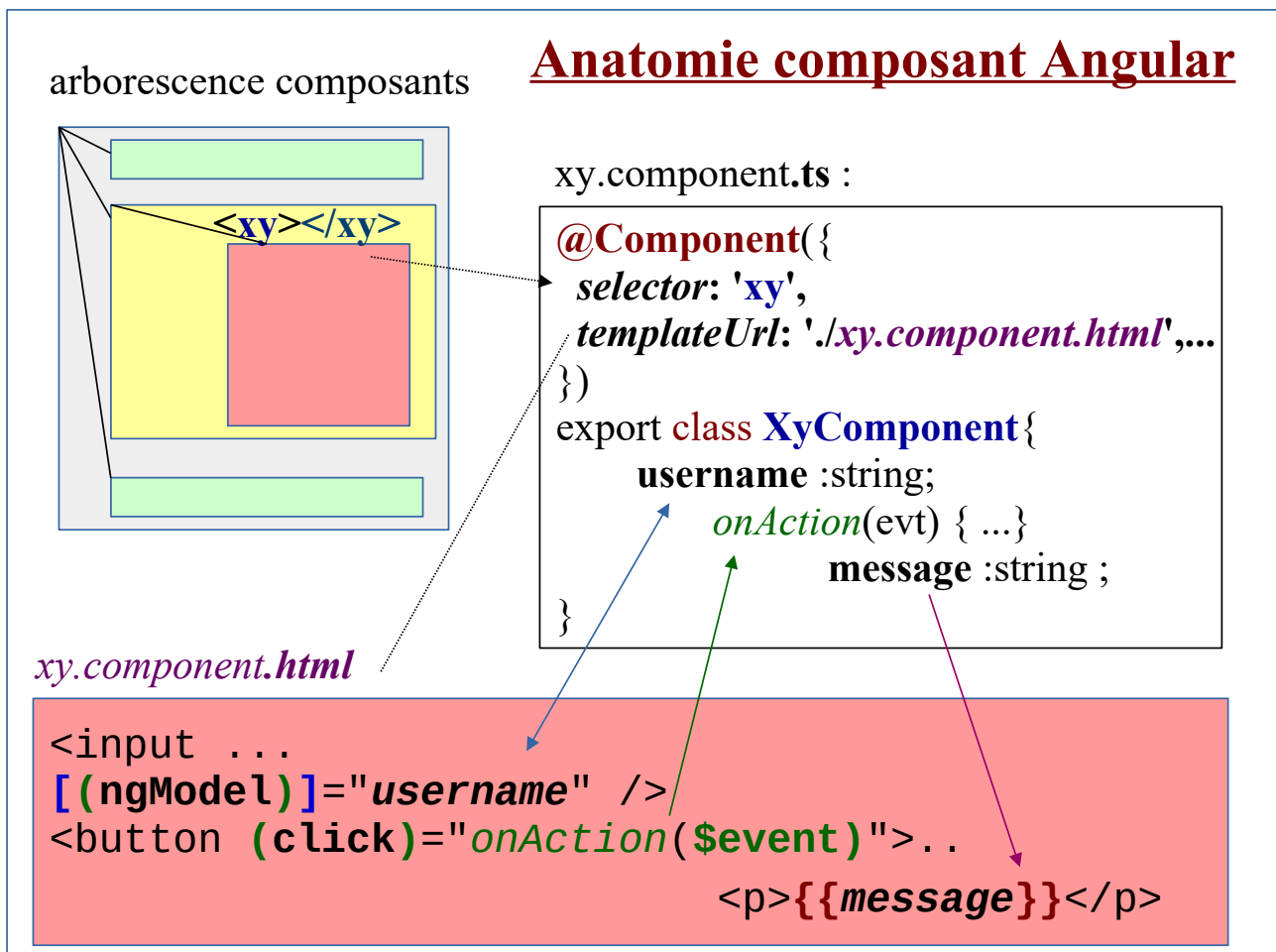
```
"styles": [
  "src/styles.scss",
  "node_modules/bootstrap/dist/css/bootstrap.min.css",
  "node_modules/bootstrap-icons/font/bootstrap-icons.css" ou bien
  "node_modules/@fortawesome/fontawesome-free/css/all.min.css"
],
```

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`, ...)

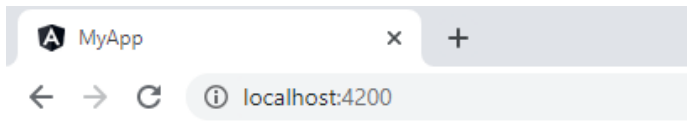
Attention : Pour qu'il soit pris en compte, chaque composant de l'application doit être **enregistré** dans un des modules (souvent dans la partie `"declarations: [...]"` du module principal `src/app/app.module.ts`).

3. 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`)
 - Les modifications apportées dans le fichier `src/app/app.module.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

- Tester via <http://localhost:4200>



header works!

welcome to my-app

basic works!

calculatrice works!

tva works!

footer works!

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 (sélecteurs 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*)

III - Langage typescript

1. Renvois selon le contexte de la formation

Selon la variante de la formation angular, les connaissances fondamentales sur javascript et typescript sont :

- soient vues (dans l'essentiel , dans les grandes lignes) en début de formation
- soient considérées comme déjà acquises (par exemple via d'autres formations préalables)

→ Il faut utiliser la documentation complémentaire typescript.pdf (avec de nombreuses annexes sur javascript) pour si besoin apprendre ou approfondir le langage typescript .

2. javascript pour angular

Parties de javascript à idéalement bien connaître pour le développement angular :

- bases du langage (null, undefined , boucle for avec mots clefs in et of , ...)
- format JSON (JSON.stringify() , JSON.parse() , ...)
- tableaux
- template string (entre quotes inverses)
- localStorage et sessionStorage
- fonctions fléchées (alias lambda expressions)
- modules es2015 (export , import { ... } from '....')

Parties de javascript moins utiles pour le développement angular :

- api DOM
- Promise es2015 , async/await es2017
- ...

3. typescript pour angular

Parties de typescript à idéalement bien connaître pour le développement angular :

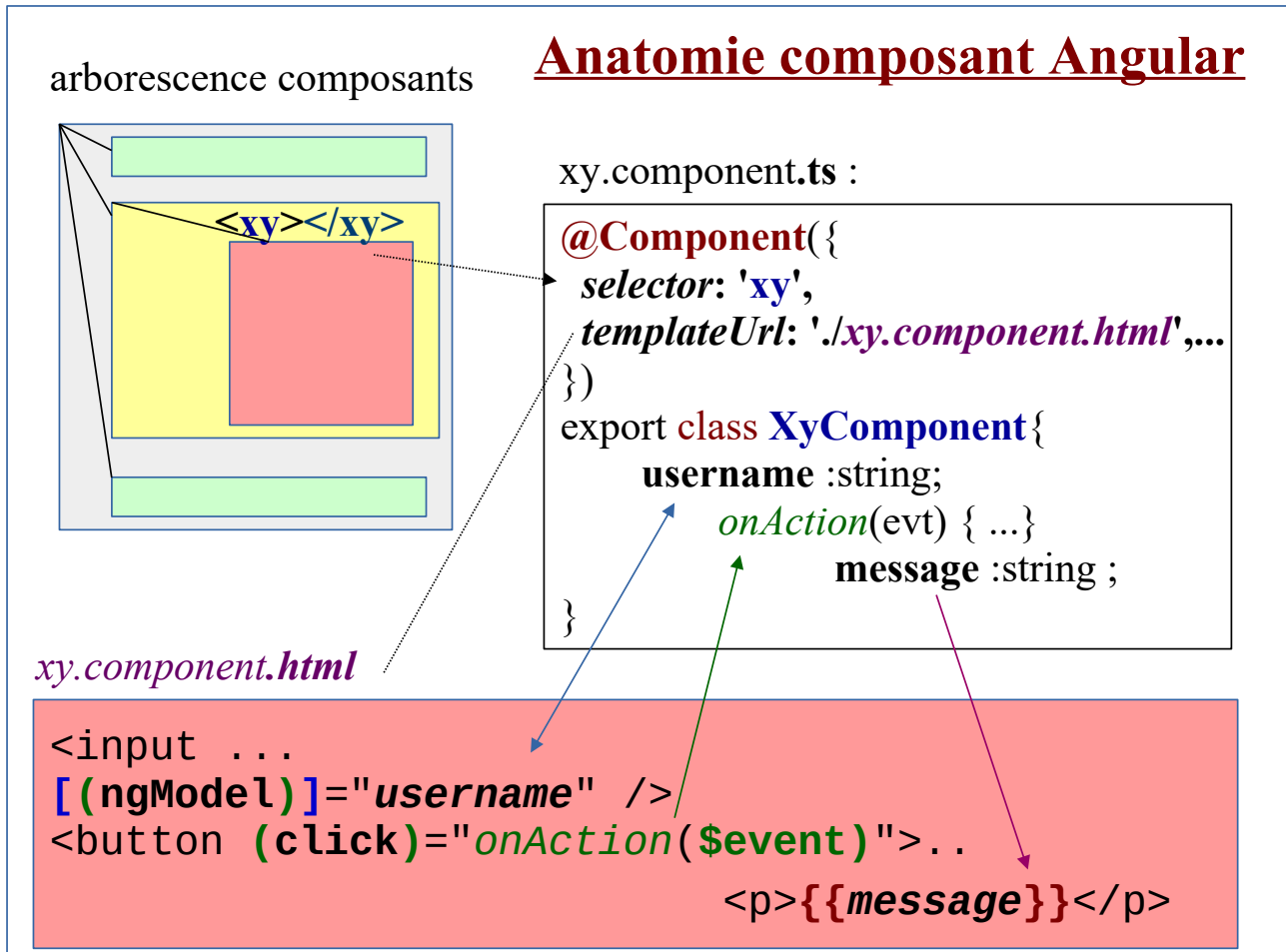
- bases du langage (tsc , tsconfig.json , types élémentaires , syntaxes strictes, ...)
- programmation orientée objet (class , constructor , public/private , get/set , static , héritage)
- constructor() avec mots clefs public ou private (c'est très important)
- modules en version "typescript" (export , import { ... } from '....')
- generics <T>
- un minimum de connaissances sur les interfaces
- ...

Parties de typescript moins utiles pour le développement angular :

- librairie de définitions (d.ts)
- aspects avancés sur interfaces
- ...

IV - Essentiel sur templates , bindings , events

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 `[(ngModel)]="..."`) ou bidirectionnel (via `[(ngModel)]="..."`).

Le déclenchement de méthodes événementielles, via la syntaxe `(eventName) = "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 :

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

2.4. Principales directives prédéfinies (angular2/common)

<code><section *ngIf="showSection"></code>	Rendu conditionnel (si expression à true) . Très pratique pour éviter exception <code>{{obj.prop}}</code> lorsque <i>obj</i> est encore à "undefined" (pas encore chargé)
<code><li *ngFor="let item of list"></code>	Élément répété en boucle (forEach)
<code><div [ngClass]="{active: isActive, disabled: isDisabled}"></code>	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 évaluée 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 case 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>?'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 :

b :

operation :

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 sélectionner:

- cd
- dvd
- other

catégorie à sélectionner:

- **cd**
- dvd
- other

produits de la catégorie cd :

ref	label	prix
p1	CD1	5.6
p2	CD2	9.6

catégorie à sélectionner:

- 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="myCollItem">
    <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="myCollItem" *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>

```

zz.component.css

```

.selected { color : blue; font-weight: bold;}
.myWrapFlexbox { padding : 1em; display: flex; flex-flow: row wrap; }
.myCollItem { 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:

tauxTva:

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>
```

V - 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 rechargement 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; &nbsp;  
<a routerLink="/login">login</a> &nbsp; &nbsp;
```

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

VI - Contrôles de formulaires (bases essentielles)

1. Contrôle des formulaires (template-driven)

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

Approches	Caractéristiques
template-driven	Simple paramétrage dans le templates HTML, selon standard HTML5, pas ou très peu de code typescript/ javascript
model-driven (alias reactive-forms)	Manière plus précise de paramétrer le comportement des validations de formulaire (moins de paramétrage côté 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 peu 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
Champ visité (souris entrée et sortie)	<i>touched</i>	ng-touched	ng-untouched
Valeur du champ modifiée	<i>dirty</i>	ng-dirty	ng-pristine
Valeur du champ valide	<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 côté ng serve ou bien du côté 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">Submit</button>
</form>
```

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 }
  ....
}
```

VII - Components (angular)

1. Structure d'une application angular 2+

1.1. 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 "./" .

Exemple :

app/app.component.ts

```
...
export class AppComponent { ... }
```

app/app.module.ts

```
import {AppComponent} from './app.component'
...
@NgModule({
  ...
  bootstrap: [ AppComponent ]
})
...
```

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

Les modules des "**librairies**" prédéfinies 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';
...
```

1.2. Deux niveaux de modularité: fichiers/classes et modules

Une **classe** (composant visuel , directive , service, ...) est un composant de petite taille (généralement de niveau **fichier**).

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 petite application peut comporter un seul grand module fonctionnel (ex : "app") .

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 : Authentification , Session , SharedData , ...)

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

1.3. Principaux types de composants "angular":

Module fonctionnel (répertoire)	Ensemble de composants et/ou de services (généralement placés dans un même répertoire) et chapeauté par une classe décorée via @NgModule .
Module auxiliaire (paramétrages / config.)	Simple fichier auxiliaire de paramétrage (ex : app-routing.module.ts) regroupant certains éléments de configuration.
Component (composant visuel)	Composant visuel de l'application graphique (associé à une vue basée elle même sur un template) – généralement spécifique à une application NB: un composant angular est souvent vu comme une nouvelle balise/sélecteur (ex : <code><app-header></app-header></code>)
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 <code><p></code> , sur <code><div></code> , sur <code></code> ...) Une directive s'utilise souvent comme un nouvel attribut spécial (ex : <code>[highlight]="yellow"</code> , <code>*ngIf="..."</code>) 2 types de directives : "structurelles" , "attribut"
Service (invisible , potentiellement partagé)	Un service est un élément invisible de l'application (qui rend un certain service) en arrière plan (ex : accès aux données , configuration, calculateur , ...)

2. Les modules applicatifs

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 accessibles à l'ensemble des composants de ce module.
(NB : depuis angular 6 et le paramétrage `{ providedIn: 'root' }` de `@Injectable()` tous les services du module courant sont implicitement rendus accessibles à tous les composants du module)

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)

```
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 { }
```

NB: par défaut , une application angular (générée par `ng new ...`) ne comporte qu'un seul module applicatif (répertoire `src/app` et fichier `app.module.ts`)

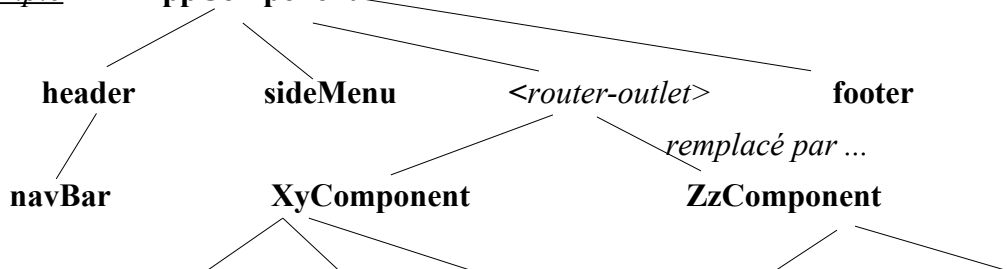
Une petite application angular peut se contenter d'un seul module.

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 . C'est un aspect avancé qui sera généralement présenté dans une annexe du cours .

2.1. Structure arborescente d'une application Angular 2+

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

Exemple : **AppComponent**



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

Une application angular créée via l'assistant "*ng new my-app*" ne comporte qu'un seul module applicatif appelé "**app**". Cette structure simple suffit pour une application de petite taille.

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

<div> <div>▼ common</div> <div> <div>> data</div> <div>> directive</div> <div>> gard</div> <div>> interceptor</div> <div>> pipe</div> <div>> service</div> <div>> util</div> <div>> validator</div> </div> </div>	<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>
--	---

2.3. 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 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 :

.../

components

gards

pipes

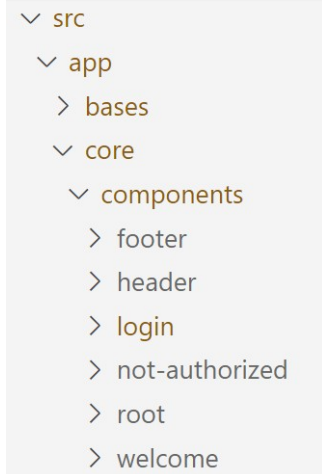
models (or *data*)

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`

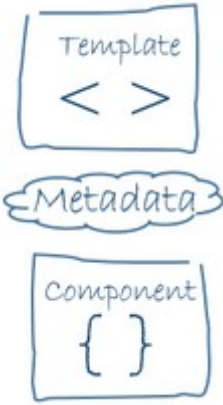
```
...
{ path: 'ngr-welcome', component: WelcomeComponent },
{ path: '', redirectTo: '/ngr-welcome', pathMatch: 'full' },
{ path: 'ngr-login', component: LoginComponent },
{ path: 'ngr-xxx', loadChildren: () => import('../xxx/xxx.module').then(m => m.XxxModule) },
{ path: 'ngr-yyy', loadChildren: () => 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 répertoires sont renommés ou déplacés lors d'un refactoring.

3. Précisions sur les composants (@Component)

3.1. Anatomie d'un composant de angular 2+

Rappels:

	<p>Un composant "angular 2+" est essentiellement constitué d'une classe codant 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 gérée 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> <p>[propertyBinding]</p> <p>(eventBinding)</p> <p>[(two-way data binding)]</p>	<p><i>DOM/Template/User</i></p> <p style="text-align: right;">Component (Object/Memory)</p> <pre> <----- {{xy}} <----- [propXy]="xy" ----- (eventZz)="onZz(\$event)" ----> <----- [(ngModel)]="xy" -----> </pre>
--	--

3.2. 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 {
```

```
  @Input()
```

```
  p1
```

```
  @Output
```

```
  eventA
```

```
  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 descendent
- les événements remontent

3.3. Simple et efficace @Input()

@Input (du côté 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*/ ="";

  constructor() {}
}
```

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 ..

⏏ ⏪ ⏩ ⏹

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) -->

```

Eventuels et rares paramétrages avancés (pour cas très pointus) :

```

@Input('titre') // pour <myheader titre='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>

```

3.4. @Output() assez complexes pour besoins avancés

@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}>();
}

```

```

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

```

reglette.component.html

```

<div class="reglette" (click)="onCursur($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 {
  valeurCursur /*:number*/ =0;

  onChangeCursur(event : any){
    const evt : {value:number} = event;
    this.valeurCursur = evt.value;
  }
  ...
}

```

```

<app-reglette width="200" (changeEvent)="onChangeCursur($event)"></app-reglette>
cursur = {{valeurCursur}}

```



cursur = 49

si click dans le milieu de la réglette



cursur = 1

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



cursur = 97

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

3.5. 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 } 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>
```

<div style="background-color: #007bff; color: white; padding: 5px; text-align: center;">calculatrice ⬇</div>	<div style="background-color: #007bff; color: white; padding: 5px; text-align: center;">calculatrice ⬆</div> <div style="padding: 10px;"> <p>calculatrice</p> <p>a: <input type="text" value="0"/></p> <p>b: <input type="text" value="0"/></p> <div style="display: flex; justify-content: space-around;"> additionner (a+b) soustraire (a-b) </div> <p>res: 0</p> </div>
--	--

4. 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 {
  @Input()
  titre /*: string*/ ="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) .

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

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 prédéfinis:

"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 }
-->
```

NB : 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" .

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 -->
```

NB : il est possible de programmer de nouveaux pipes personnalisés et réutilisables .

Cet aspect avancé est généralement présenté dans une annexe du cours .

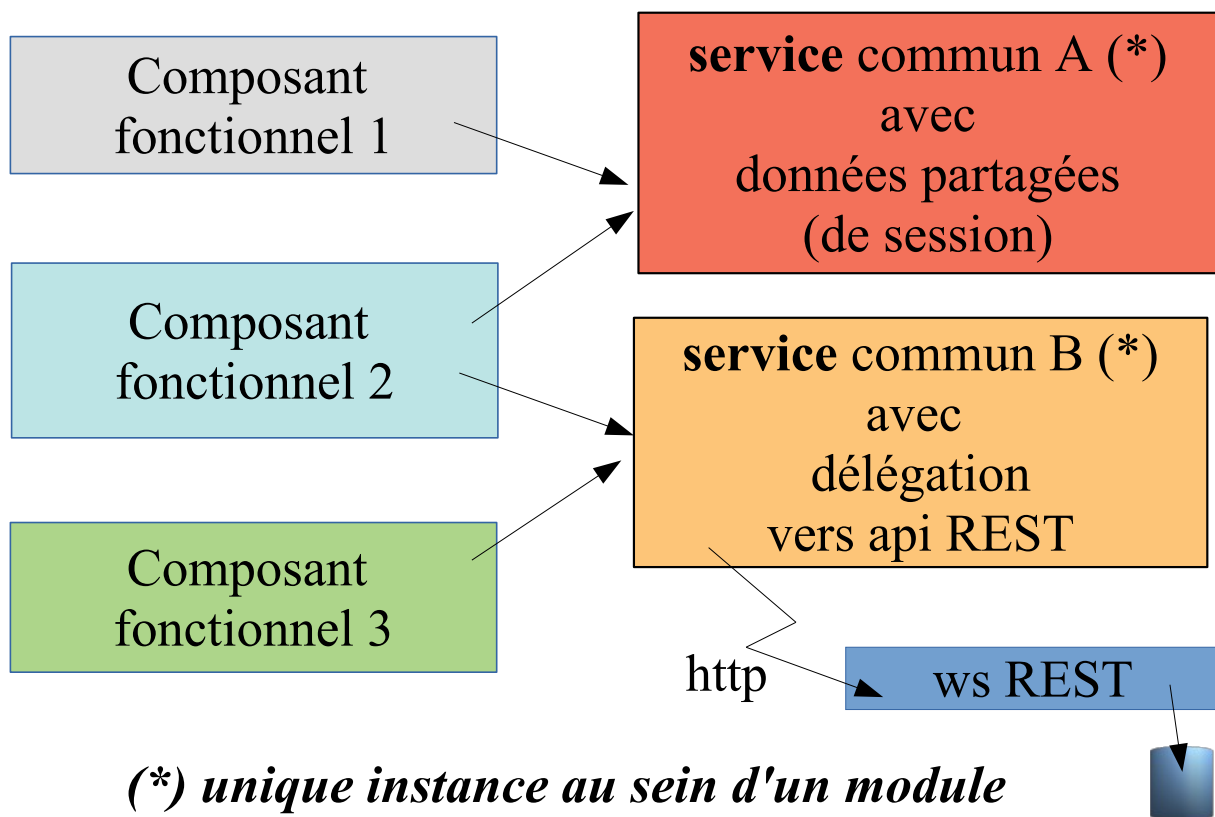
VIII - Services et injections (essentiel)

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"*) .

Exemple simple:

preferences.service.ts

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

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

  //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);
  }

  constructor() {
    const c = localStorage.getItem('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() .

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) et dans ce cas le service "XyService" sera automatiquement considéré comme fourni par le module (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 du module qui en auront besoin (ceux qui auront paramétré une injection de dépendance) .

2.3. Injection de dépendance via constructeur

```
@Component({
  selector: 'my-app',
  template: '...' /*,
  providers: [HeroService] nécessaire que si pas déjà enregistré globalement dès le NgModule() */
})
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

IX - Appels de W.S. REST (Observable, ...)

1. Angular et dialogues HTTP/REST

1.1. Contexte des appels HTTP/REST et CORS

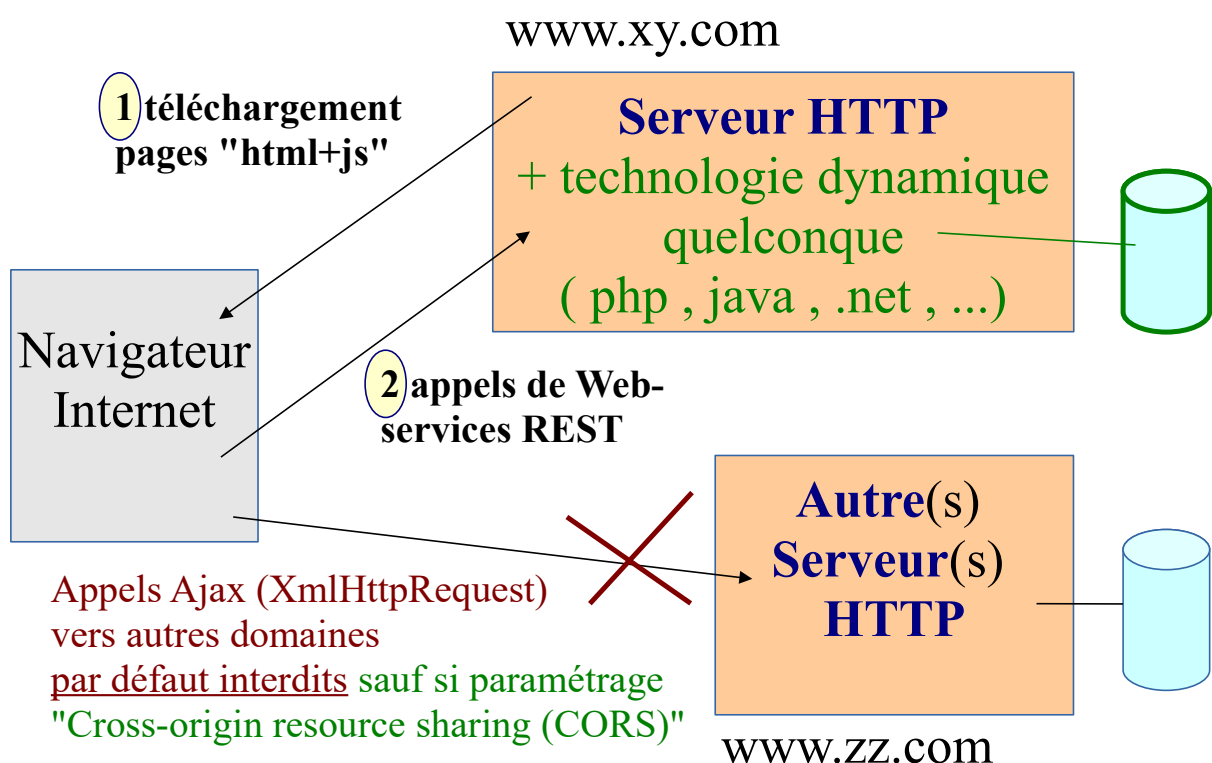
Une application Angular s'exécute au sein d'un navigateur Web . Lorsque celle-ci doit effectuer un appel HTTP/ajax , elle est soumise aux restrictions habituelles du navigateur :

- 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 côté des web-services REST côté 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. Configuration d'un reverse-proxy http

De façon à écrire et tester du code simple , stable et portable (de la phase de développement/test jusqu'à la phase de production) , on aura intérêt à :

- **utiliser des URLs relatives**
- **configurer un reverse proxy http** (option *--proxy-config* de *ng serve* développée dans le chapitre "packaging / déploiement")

1.3. 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)

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, généricité, code d'utilisation plus compacte, ...) de l'ancien service technique **Http** disponible depuis Angular 2.

2.1. Configuration du module HttpClientModule

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 { HttpClientModule } from "@angular/http";
...

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

2.2. 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 { HttpResponse } from "@angular/common/http";

export function messageFromError(err : HttpResponse , 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: HttpResponse) => { this.message = messageFromError(err,"echec get"); }
  });
```

2.3. 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.

2.4. Déclenchement du coté 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)"); }
        });
  }
}

```

2.5. 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 InscriptionService {
  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.

2.6. Exemple de suppression (delete):

```
public deleteDeviseServerSide$(deviseCode):Observable<any>{
  let deleteUrl : string = this.basePrivateUrl + "/" + deviseCode ;
  console.log("deleteUrl= " + deleteUrl );
  return this.http.delete(deleteUrl );
}
```

2.7. 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 { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';
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 { }
```

2.8. 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);  
}
```

...

X - 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 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{
  yyId: 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 },
```

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 :

```
...
{ 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)},
...
```

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

guards

pipes

models (or *data*)

directives

services

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

XI - Directives

1. Aperçu sur les directives (angular2+)

1.1. 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 <i>ngStyle</i>)
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 : <i>ngIf</i> , <i>ngFor</i> , <i>ngSwitch</i>)
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.

1.2. 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 {
  @Input('myHighlight')
  public highlightColor: string;

  private _defaultColor = 'red';

  @Input() set defaultColor(colorName:string){
    this._defaultColor = colorName || this._defaultColor;
  }

  constructor(private el: ElementRef) {
```

```

}

@HostListener('mouseenter')
onMouseEnter() { this._highlight(this.highlightColor || this._defaultColor); }

@HostListener('mouseleave')
onMouseLeave() { this._highlight(null); }

private _highlight(color: string)
{ 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/>

```

1.3. 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,
               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" />

<p *myUnless="age>=18">

condition "age>=18" is false and myUnless is true.

</p>

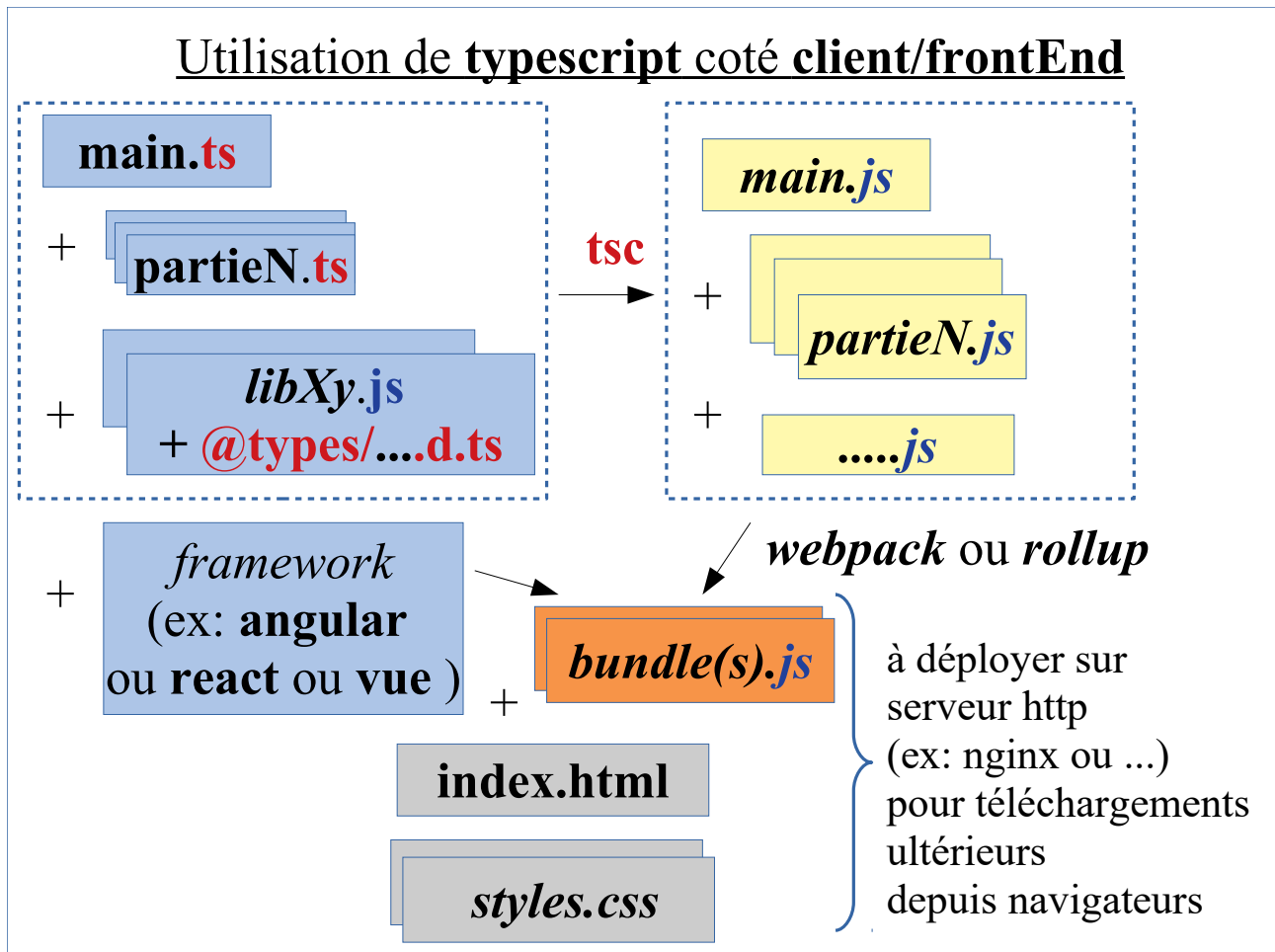
<p *myUnless="!(age>=18)">

condition "age>=18" is true and myUnless is false.

</p>

XII - Packaging et déploiement d'appli. angular

1. Notion de "bundle" pour le déploiement



> 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** 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. reverse proxy en mode développement (ng serve)

L'essentiel de @angular/cli a déjà été présenté en début du cours.

ng serve	Lancement de l'application en mode développement (watch & compile file , launch server,) → URL par défaut : <i>http://localhost:4200</i>
-----------------	--

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) .

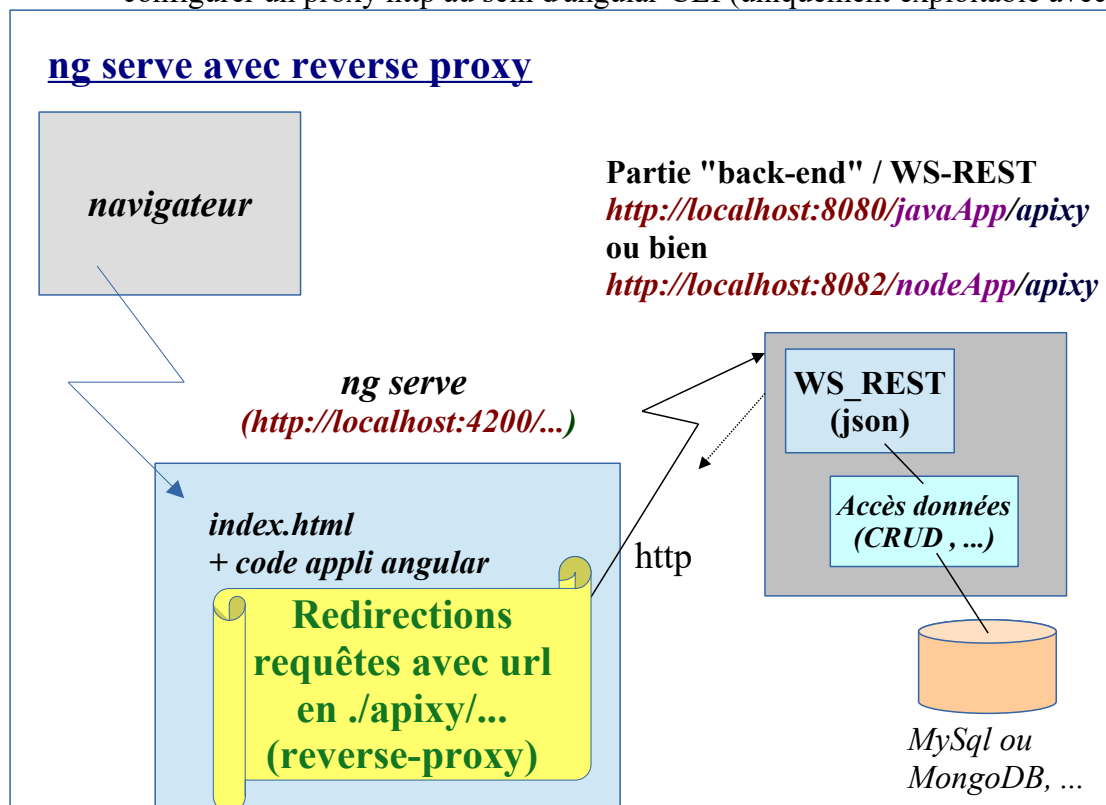
2.1. proxy http pour appels ajax/WS-REST en mode dev

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
- paramétrer des autorisations "CORS" du côté serveur (code java ou js/express ou ...)

De façon à éviter toutes ces choses (aujourd'hui déconseillées) , on peut :

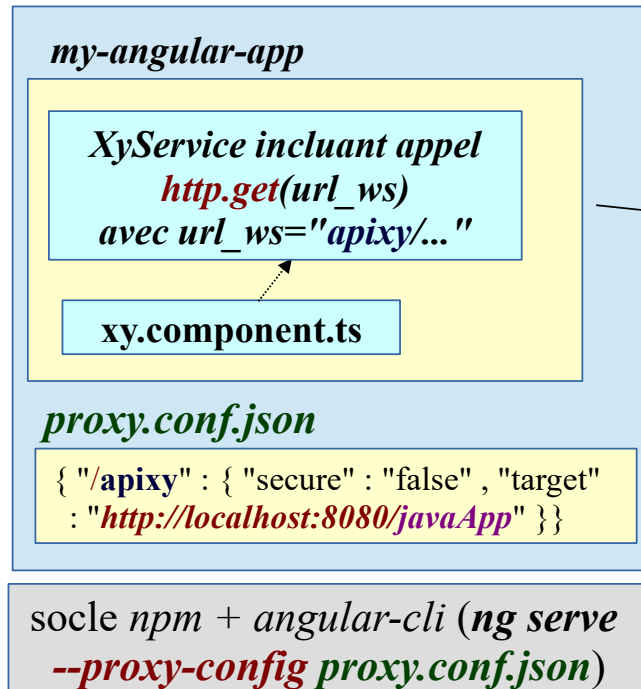
- toujours utiliser des URLs relatives (ex : `apiXy/xy`) pour appeler des Web services "REST" (java ou php ou nodeJs/express ou ...) via angular et ajax dès la phase de développement
- ne pas systématiquement avoir besoin de configurer des autorisations "CORS" du côté serveur
- configurer un proxy http au sein d'angular-CLI (uniquement exploitable avec ng serve) .



- <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/...>)

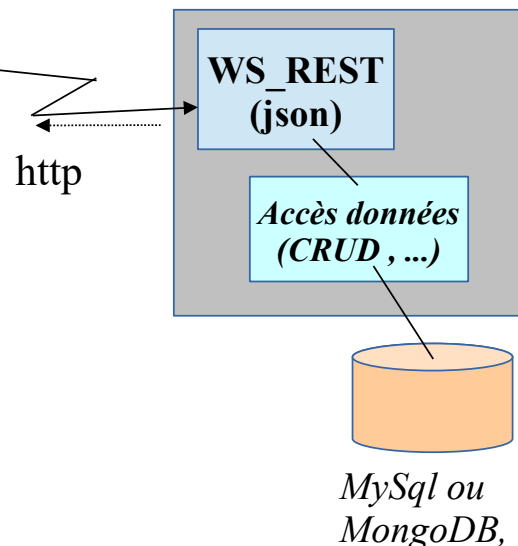


Partie "back-end" / WS-REST

<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" ,  
              "target" : "" }  
}
```

pour que les requêtes d'urls relatives **apixy/..** soient redirigées vers une application java/tomcat.
ou bien

avec **proxy.conf.json**

```
{ "/apixy" : { "secure" : "false" ,  
              "target" : "" }  
}
```

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 préciser l'option **--proxy-config** à chaque redémarrage) .

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

3. 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 "**j**it" (*just in time*)
- par le compilateur "**a**ot" (*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 j it	avec a ot
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.

4. 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

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:

> 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

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

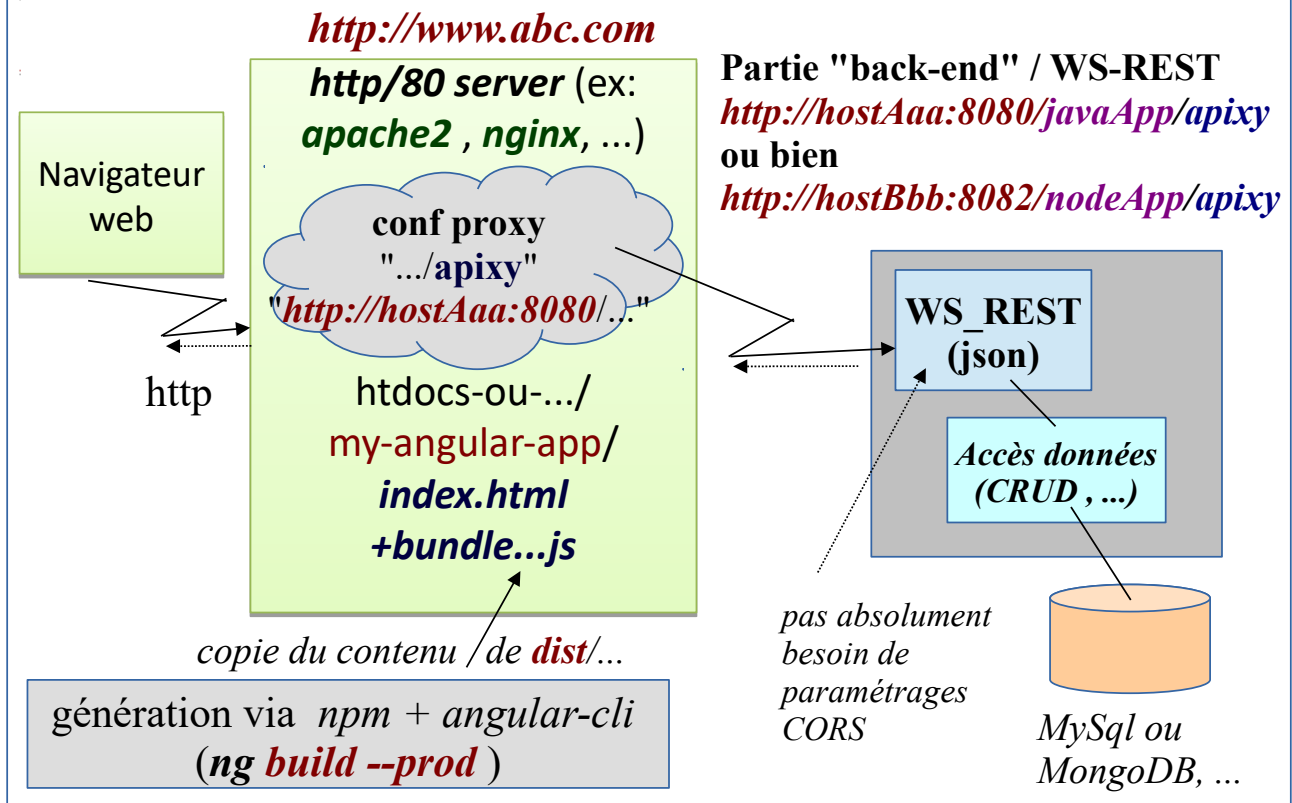
http-server --proxy http://localhost:8282 -c-1 dist/my-app

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>) .

Environnement de prod compatible Angular (v2,v4,...)



5.1. 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
# virtualy 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
# virtualy 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, ...)!!!!

XIII - Aspects divers (BehaviorSubject,pipe,...)

1. BehaviorSubject

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.

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(...).

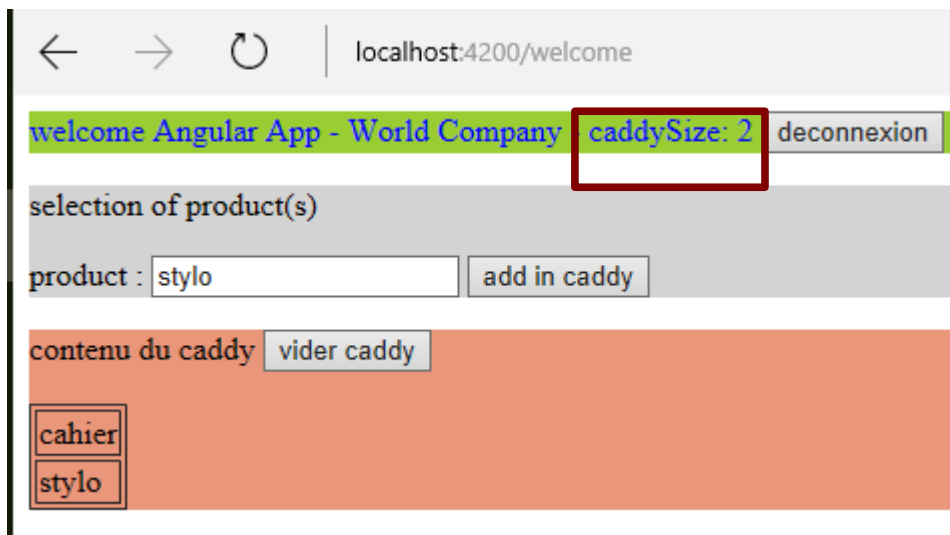
Principe de fonctionnement :

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 <table> <thead> <tr> <th>numero</th> <th>label</th> <th>prix</th> </tr> </thead> <tbody> <tr> <td>6</td> <td>PRODUIT 6 20</td> <td></td> </tr> <tr> <td>2</td> <td>PRODUIT 2 30</td> <td></td> </tr> <tr> <td>1</td> <td>PRODUIT 1 50</td> <td></td> </tr> <tr> <td>3</td> <td>PRODUIT 3 80</td> <td></td> </tr> </tbody> </table>	numero	label	prix	6	PRODUIT 6 20		2	PRODUIT 2 30		1	PRODUIT 1 50		3	PRODUIT 3 80		seuilMax 60 nbProd : 3 list-prod <table> <thead> <tr> <th>numero</th> <th>label</th> <th>prix</th> </tr> </thead> <tbody> <tr> <td>6</td> <td>PRODUIT 6 20</td> <td></td> </tr> <tr> <td>2</td> <td>PRODUIT 2 30</td> <td></td> </tr> <tr> <td>1</td> <td>PRODUIT 1 50</td> <td></td> </tr> </tbody> </table>	numero	label	prix	6	PRODUIT 6 20		2	PRODUIT 2 30		1	PRODUIT 1 50		seuilMax 200 nbProd : 5 list-prod <table> <thead> <tr> <th>numero</th> <th>label</th> <th>prix</th> </tr> </thead> <tbody> <tr> <td>6</td> <td>PRODUIT 6 20</td> <td></td> </tr> <tr> <td>2</td> <td>PRODUIT 2 30</td> <td></td> </tr> <tr> <td>1</td> <td>PRODUIT 1 50</td> <td></td> </tr> <tr> <td>3</td> <td>PRODUIT 3 80</td> <td></td> </tr> <tr> <td>5</td> <td>PRODUIT 5 120</td> <td></td> </tr> </tbody> </table>	numero	label	prix	6	PRODUIT 6 20		2	PRODUIT 2 30		1	PRODUIT 1 50		3	PRODUIT 3 80		5	PRODUIT 5 120	
numero	label	prix																																													
6	PRODUIT 6 20																																														
2	PRODUIT 2 30																																														
1	PRODUIT 1 50																																														
3	PRODUIT 3 80																																														
numero	label	prix																																													
6	PRODUIT 6 20																																														
2	PRODUIT 2 30																																														
1	PRODUIT 1 50																																														
numero	label	prix																																													
6	PRODUIT 6 20																																														
2	PRODUIT 2 30																																														
1	PRODUIT 1 50																																														
3	PRODUIT 3 80																																														
5	PRODUIT 5 120																																														

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. 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>
```

Power Booster

Super power boost: 1024

2.2. 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>
```

2.3. 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

XIV - 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 "gradle"

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 ...

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

Après un lancement de `cypress open`, il faut en théorie choisir un test à lancer au sein de la fenêtre de sélection. Nous devons cependant écrire préalablement notre propre test. À arrêter (en fermant cette fenêtre) et 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. `npmx 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)
```

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 :

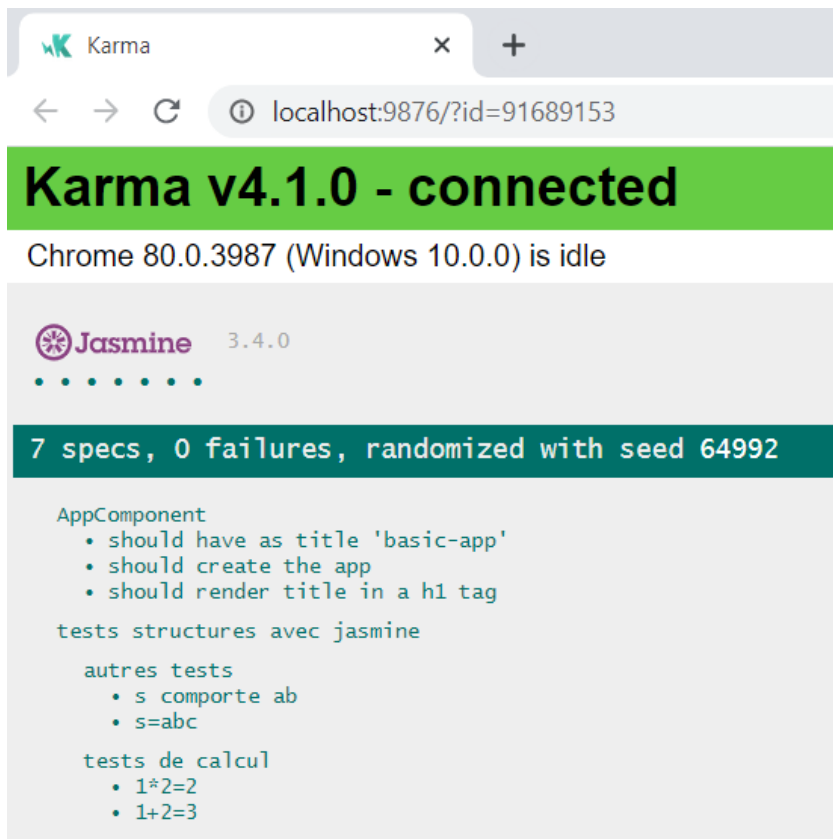
- **beforeEach()** et **afterEach()** sont appelées avant et après l'exécution de chaque **it()** *pour les describe() de même niveau.*
- Souvent , un niveau de **describe()** correspond à un **objet/composant/sujet à tester** et la fonction **it()** signifie "*it should behave like that*" .

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 :



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 était 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 baseUrl = './devise-api/public/devise';
  private baseUrlPrivate = './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', () => {
        //excepted 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.apiUrl="./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 { AppRoutingModuleModule } 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 , AppRoutingModuleModule ] ,
      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 `async` :

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 paramétrer

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_synchrone1() ;
    appel_synchrone2() ;
    appel_asynchrone_retournant_promise().then( () => {
        instructions_de_la_callback_asynchrone ;
        done();
    });
});
```

L'ancienne fonction utilitaire `async()` de [@angular/core/testing](https://angular.io/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_synchrone1() ;
    appel_synchrone2() ;
    let resAsync = await appel_asynchrone_retournant_promise() ;
    instructions_après_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:

tauxTva:

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 = 220.000000000000003

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_synchrone_1() ;
    appel_synchrone_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_synchrone_1() ;
    appel_synchrone_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_synchrone_1() ;
    appel_synchrone_2() ;
    appel_asynchrone_Xy_retournant_promise_ou_observable() ;
    tick() ; // attente au sein de fakeAsync()
    fixture.detectChanges();
    expect(...).toBe(...);
}

```

sont censés être **à peu près équivalents** 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.

XV - Sécurité – application 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 compilée, 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", ...)

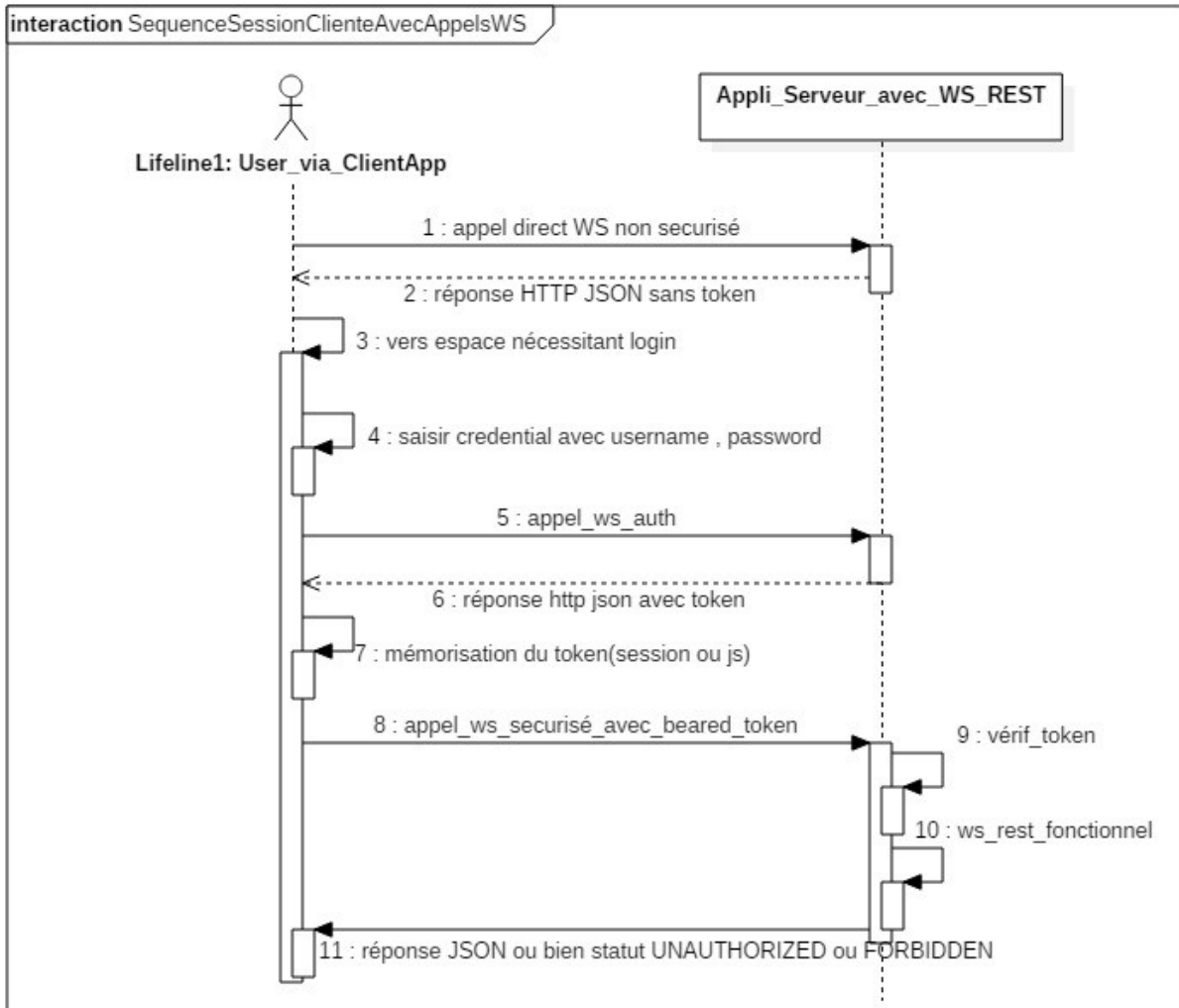
--> le côté serveur ne doit avoir qu'une confiance limitée aux requêtes angular.
Il est bon de vérifier fréquemment "token" et autres.

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. Sécurisation des 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
New key	Value

Status: 200 OK

si ok

Status: 403 Forbidden

ou bien

si faux jeton (invalid token)

Status: 401 Unauthorized

ou bien

si pas de jeton

ANNEXES

XVI - Annexe – 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 a fait le choix d'utiliser "Observable" de RxJs plutôt que "Promise" dès la version 2.0 (Angular 2) .

Depuis, le framework "Angular" a continué d'exploiter à fond la bibliothèque RxJs .
Cependant , les 2 frameworks ont beaucoup évolué depuis 2015/2016 .

La version **4.3** de **Angular** a apporté 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 côté 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
})
```

NB : les parties *error : callback_error* et *terminate : callback_terminate* de .subscribe() sont **facultatifs** 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 } = rxjs;
const { map, filter } = rxjs.operators;
range(1, 10).pipe(
  filter(x => x >= 5),
  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*/);
//subscriptionObsvI1 is the result of .subscribe() call
const subscriptionObsvI1 = obsvI1.subscribe(n =>
{ console.log( the number of interval occurrence (starting at 0) is ${n} `);
  if(n>=5) {
    subscriptionObsvI1.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

...

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	Traitement supplémentaire 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. Passerelles entre "Observable" et "Promise"

7.1. 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'); }
});
```

7.2. 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);
  }
}
```

8. 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 .

XVII - Annexe – Model Driven Forms

1. Contrôle des formulaires

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

Approches	Caractéristiques
template-driven	Simple paramétrage dans le templates HTML, pas ou très peu de code typescript/ javascript
model-driven (alias reactive-forms)	Meilleure façon de paramétrer le comportement (moins de paramétrage côté 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**" a normalement été déjà étudiée .

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

dans composant angular :

```
import { FormGroup, FormControl , Validators } from '@angular/forms';
....
class ModelFormComponent implements OnInit {
  myform: FormGroup;

  ngOnInit() {
    myform = new FormGroup({
      name: new FormGroup({
        firstName: new FormControl('', Validators.required ),
        lastName: new FormControl('default_name', Validators.required),
      }),
      email: new FormControl('', [ Validators.required,
        Validators.pattern("^[ @]*@[^ @]*") ] ),
      password: new FormControl( '',[ Validators.required, Validators.minLength(8) ] ),
      language: new FormControl()
    });
  }
}
```

dans module :

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

dans template HTML :

```
<form novalidate [formGroup]="myform">
  <fieldset formGroupName="name">
    <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>
  </fieldset>
  <div class="form-group">
    <label>Email</label>
    <input type="email" class="form-control"
      formControlName="email" >
  </div>
  ...
</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** .

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

myForm.get(*formControlName*).errors // .dirty , .valid , ...

1.3. 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) {}

  ngOnInit() {
    this.myForm = this._formBuilder.group({
      name: ['default_name', Validators.required],
      email: [', [Validators.required, Validators.pattern('[a-z0-9.@]*')]],
      message: [', [Validators.required, Validators.minLength(15)]]
    });
  }
}
```

1.4. 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 ]],
});
```

XVIII - Annexe – ngx-bootstrap

1. Extension "ngx-bootstrap" pour angular

1.1. Présentation de ngx-bootstrap

L'extension "**ngx-bootstrap**" (principalement développée par "*Valor Software*") permet de bien intégrer **bootstrap-css** (3 ou 4 ou 5) au sein d'une application angular 4,6 ou + .

Rappel :

- Une grande partie de bootstrap-css correspond essentiellement à des styles css prédéfinis et ne nécessite pas absolument de code javascript (un *npm install bootstrap* pourrait suffire dans de tel cas).
- Certains aspects de bootstrap-css sont dynamiques (ex : menus déroulants , popups, basculement d'onglets,) et nécessitent un peu de code javascript pour fonctionner. A l'origine bootstrap-css était très souvent accompagné par jquery et du plugin "bootstrap/jquery" .
- Une application "angular" à sa propre dynamique et n'est normalement pas accompagnée par jquery. On a donc besoin d'une adaptation particulière "angular" pour les aspects dynamiques de bootstrap-css et c'est principalement à cela que sert l'extension ngx-bootstrap

NB : initialement nommé "~~ng2-bootstrap~~" (à l'époque angular 2 et 4) , le projet a été ensuite renommé ngx-bootstrap . Il existe également un projet concurrent (et très ressemblant) intitulé "~~ng-bootstrap~~" (~~sans x~~) qui est beaucoup moins téléchargé/utilisé . Autant donc suivre la plus grande communauté de développeurs et utiliser "ngx-bootstrap" .

NB : A partir de 2021 et la sortie de bootstrap 5 (ne s'appuyant plus sur jQuery) , l'extension ngx-bootstrap est devenue un peu moins utile qu'avant .

D'autre part , l'extension ngx-bootstrap est un peu à la traîne vis des toutes dernières versions d'angular (13,14,15,...) et il faut quelquefois forcer son installation (avec des éventuels problèmes d'incompatibilités) .

1.2. Intégration de ngx-bootstrap dans un projet angular :

Dans la procédure valable pour toutes les versions d'angular, il est nécessaire de :

1. télécharger "ngx-bootstrap" via "**npm install -s**" (en spécifiant une version compatible)
2. ajuster le fichier angular.json de façon à configurer les css
3. ajuster le fichier app.module.ts (ou un autre module)

package.json

```
...
"dependencies": {
  "@angular/core": "~8.2.0",
  ... ,
  "bootstrap": "4.1.1",
  "chart.js": "^2.8.0",
  "font-awesome": "^4.7.0",
  "ng2-charts": "^2.3.0",
  "ngx-bootstrap": "^5.1.1",
```

```
...
}
```

angular.json

```
...
"styles": [
  "./node_modules/bootstrap/dist/css/bootstrap.min.css",
  "./node_modules/ngx-bootstrap/daterangepicker/bs-daterangepicker.css",
  "./node_modules/font-awesome/css/font-awesome.min.css",
  "src/styles.scss"
],
...
```

src/app/app.module.ts

```
import { TabsModule } from 'ngx-bootstrap/tabs';
import { BsDatepickerModule } from 'ngx-bootstrap/daterangepicker';
import { CarouselModule } from 'ngx-bootstrap/carousel';
...
imports: [
  BrowserModule, FormsModule, HttpClientModule, AppRoutingModule, ChartsModule,
  BrowserAnimationsModule,
  TabsModule.forRoot(), BsDatepickerModule.forRoot(), CarouselModule.forRoot()
]
...
```

Exemple d'installation encore valable en 2022 :

ng add ngx-bootstrap@10.0.0 (si ça fonctionne)

ou bien

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

```
npm install -s @fortawesome/fontawesome-free
```

```
npm install ngx-bootstrap@9.0.0 --save
```

ou bien

```
npm install ngx-bootstrap@10.0.0 --save
```

ou bien

```
npm install ngx-bootstrap --save --force
```

après git clone ou git pull , npm install ou bien npm install --force

NB: --force uniquement nécessaire si petites incompatibilités

1.3. Site de référence pour ngx-bootstrap

<https://valor-software.com/ngx-bootstrap/#/>

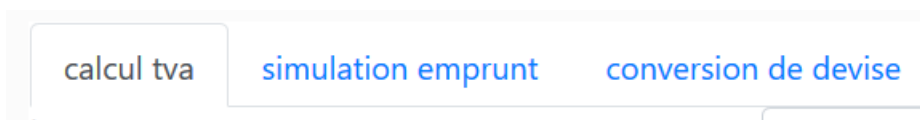
<https://valor-software.com/ngx-bootstrap/#/documentation>

<https://valor-software.com/ngx-bootstrap/#/documentation#getting-started>

1.4. Quelques composants de ngx-bootstrap

Onglets (tabs) :

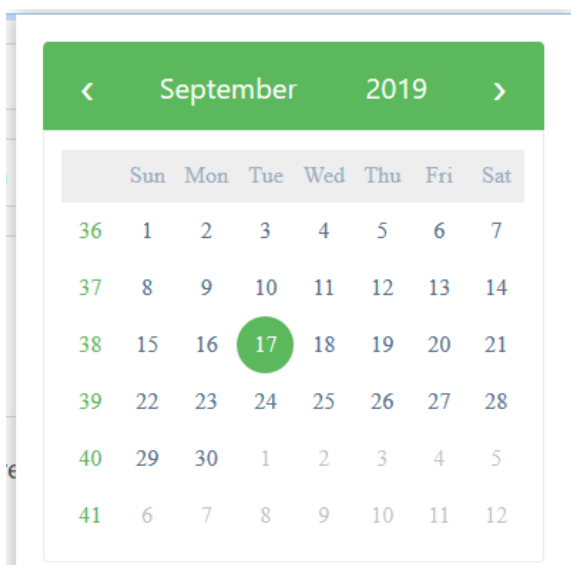
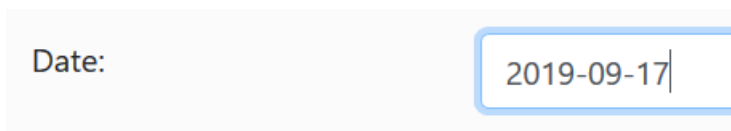
```
<tabset>
  <tab heading="calcul tva">
    <app-tva></app-tva>
  </tab>
  <tab heading="simulation emprunt">
    <app-simu-emprunt></app-simu-emprunt>
  </tab>
  <tab heading="conversion de devise">
    <app-conversion></app-conversion>
  </tab>
</tabset>
```



Calendrier javascript (datePicker) :

```
<input placeholder="yyyy-mm-dd" name="date"
  [(ngModel)]="datePublication" bsDatepicker
  [bsConfig]="{ dateInputFormat: 'YYYY-MM-DD' }" />
```

avec `datePublication : Date = new Date();` du côté .ts



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):

divA (25%) divB (50%) divC

empilement (si moins de 960px):

divA (25%)
divB (50%)
divC

breakpoints	size(px)	breakpoints	size(px)	breakpoints	size(px)
xs (extra small)	599 ou moins	lt-sm (less than small)	moins que 600		
sm (small or medium)	600 à 959	lt-md (less than md)	moins que 960	gt-xs (greater than xs)	600 ou plus
md (medium)	960 à 1279	lt-lg (less than large)	moins que 1280	gt-sm (greater than sm)	960 ou plus
lg (large)	1280 à 1919	lt-xl (less than xl)	moins que 1920	gt-md (greater than md)	1280 ou plus
xl (extra large)	1920 à 5000			gt-lg (greater than large)	1920 ou plus

3. Quelques composants "angular-material"

Card (panneau d'encadrement) :



```
<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; border-style:solid; border-color:blue ; padding : 0 }

.my-card-header {
  background-color: rgb(23, 23, 112); 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>preferredColor:</mat-label>
      <input matInput name="preferredColor" type="color" placeholder="preferredColor" [(ngModel)]= "person.preferredColor" />
    </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-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>

```

full name: *

didier defrance

kind: ☒ Male ☐ Female

email:

didier@d-defrance.fr

☒ crazy (as checkbox)

☐ crazy (as slide-toggle)

birthday:

7/11/1969

childNumber:

2

preferredColor:

password:

country:

F

France

Finlande

birthday:

7/11/1969

JUL 1969

S M T W T F S

JUL

1 2 3 4 5

6 7 8 9 10 11 12

13 14 15 16 17 18 19

20 21 22 23 24 25 26

27 28 29 30 31

preferredColor:

Couleurs

Couleurs de base :

Couleurs personnalisées :

Teinte : 160 Rouge : 0

Satur : 240 Vert : 0

Lum : 120 Bleu : 255

OK Annuler Ajouter aux couleurs personnalisées

....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 "boîte 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>

```

```

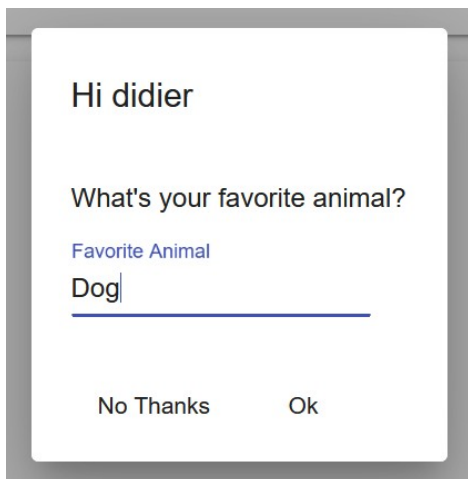
    <mat-form-field>
      <mat-label>What's your name?</mat-label>
      <input matInput [(ngModel)]="name">
    </mat-form-field>
  </li>
  <li>
    <button mat-raised-button (click)="openDialog()">open dialog</button>
  </li>
  <li *ngIf="animal">
    You chose: <i>{{animal}}</i>
  </li>
</ol>

```

What's your name?

1. didier

2. open dialog



What's your name?

1. didier

2. open dialog

3. You chose: Dog

Composants "step",

isLinear (step by step, cannot skip step)

1 Fill out your name — 2 Fill out your address — 3 Done

Name *

Next

isLinear (step by step, cannot skip step)

cre Fill out your name — cre Fill out your address — 3 Done

You are now done. (data: { "firstStep": { "name": "didier" }, "secondStep": { "address": "123 rue Elle 75000 Par ici" } })

Back Reset

Composants "mat-table" avec dataSource

table of countries (click on column header to sort)

Filter

<input type="checkbox"/>	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: 50 1 – 5 of 5 |< < > >|

Attention: la selection globale et calcul du total ne tient pas compte de la pagination (par defaut)

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 générés 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 default)

 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 : <i>cordova</i> et/ou <i>ionic</i>)	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.

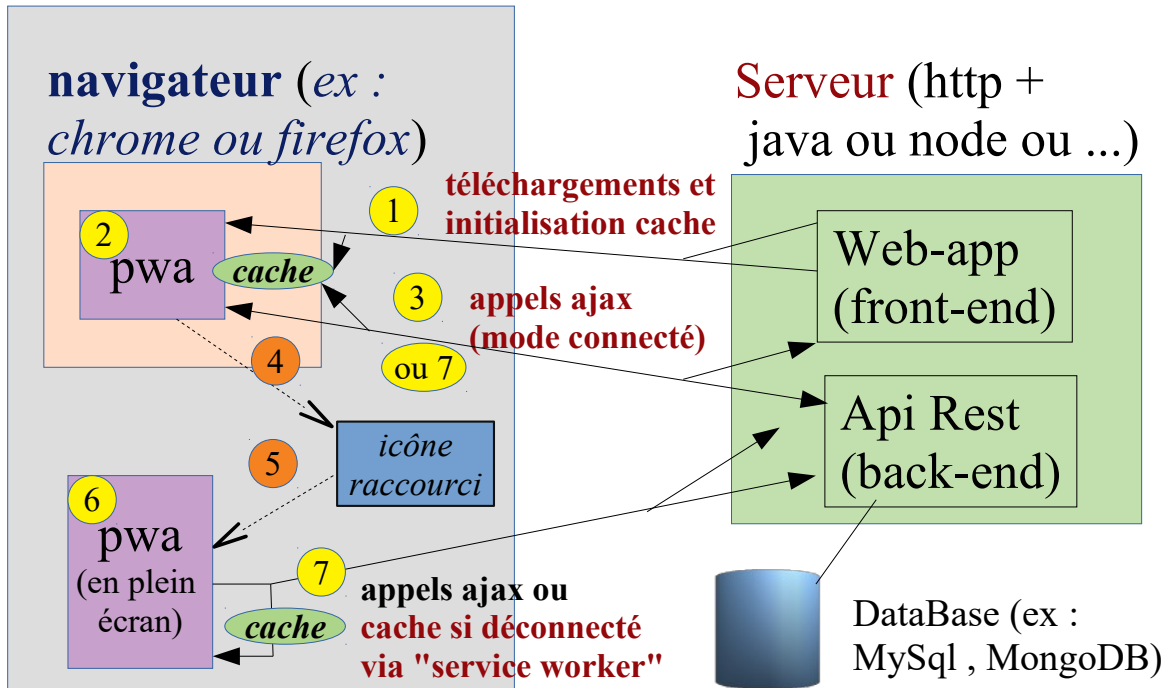
Le concept de "pwa" est très moderne et pour l'instant assez peu répandu en 2018/2019. L'essor prévu des "pwa" dépendra essentiellement du support offert par les navigateurs des smartphones. Pour l'instant "android, chrome, firefox" avancent clairement dans cette direction. Apple (ios/iphone) comporte 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 :

- **navigation 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**

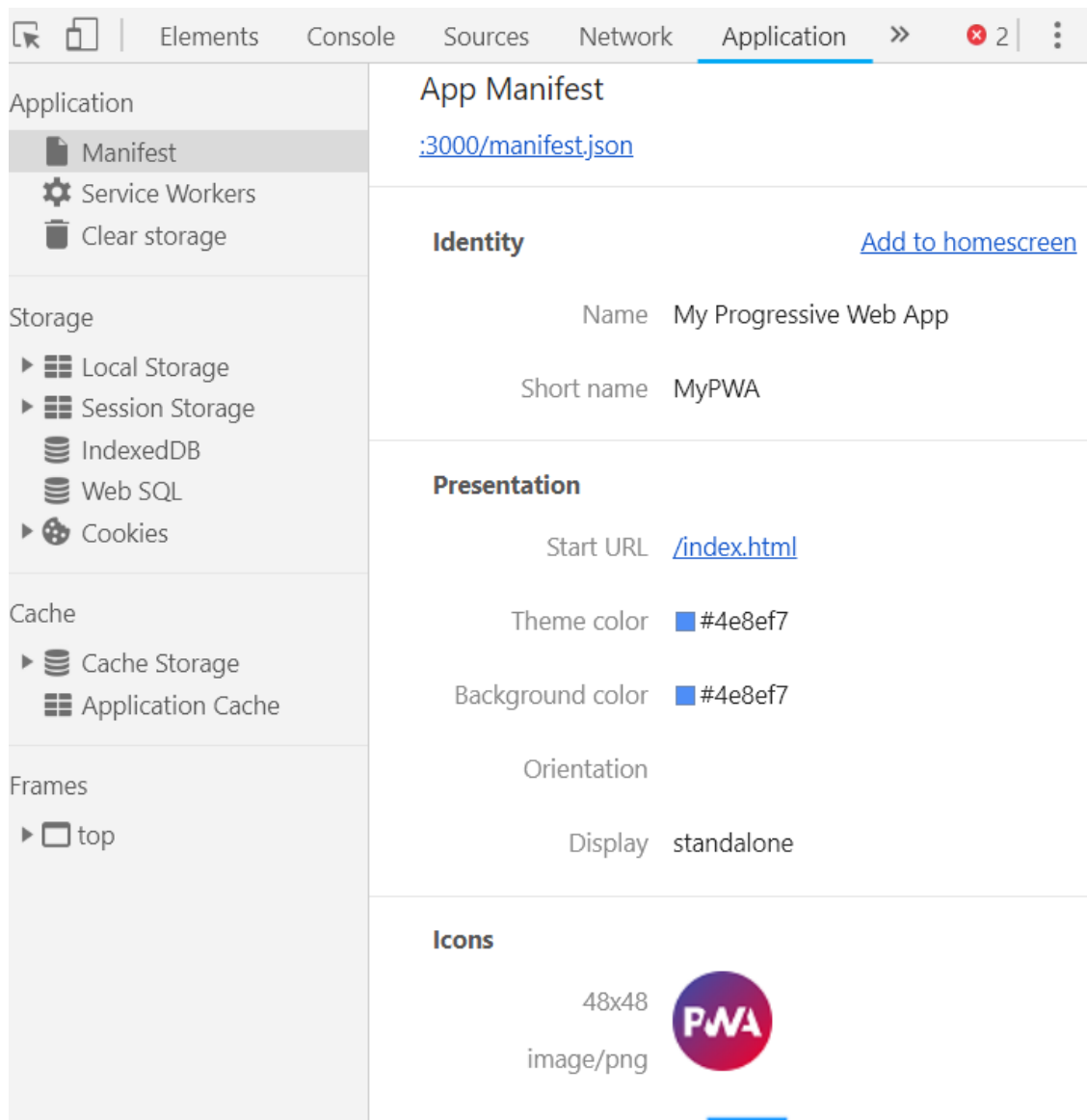
```
<!DOCTYPE html>
```

```

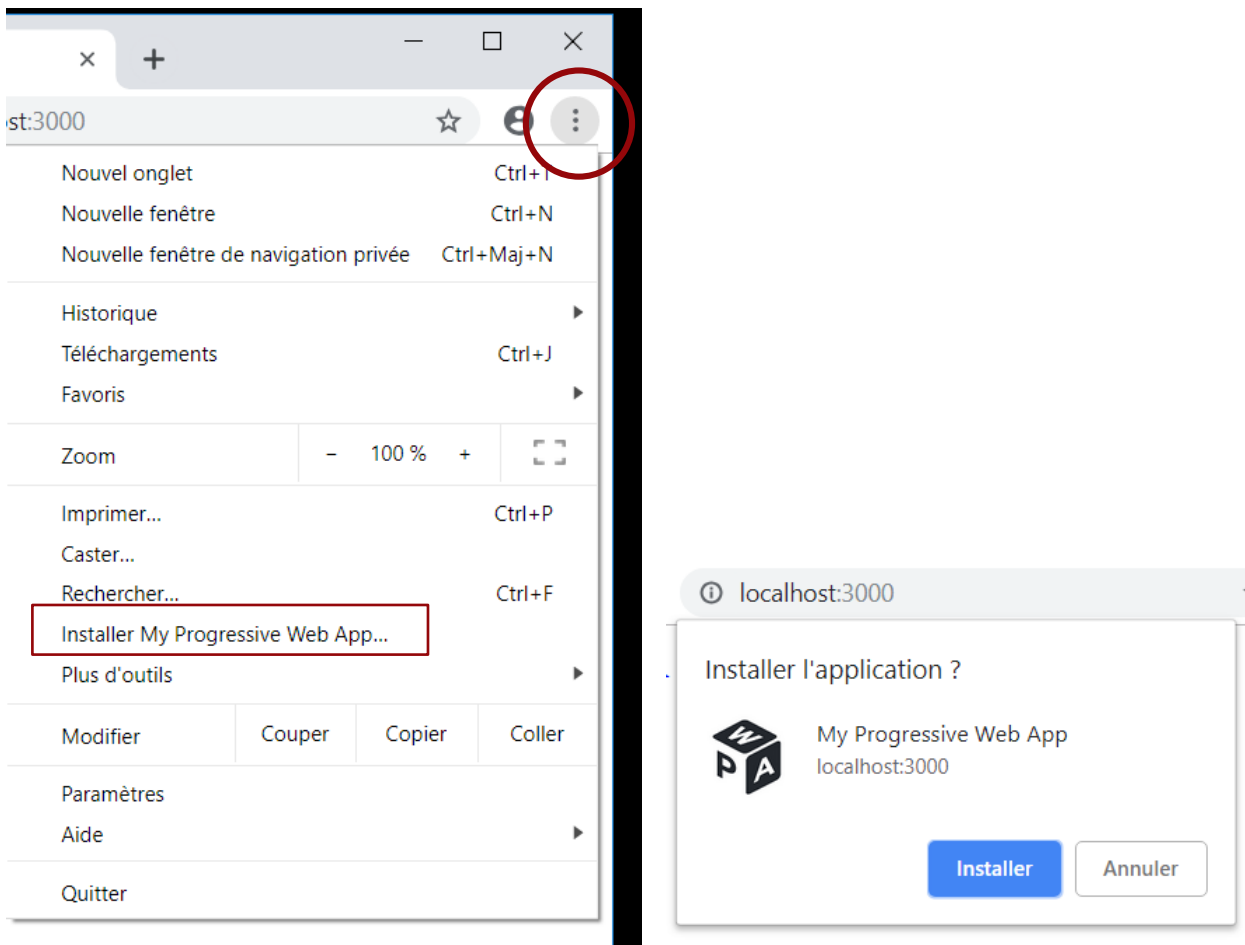
<html> <head>
  <meta charset="UTF-8">
  <title>My progressive Web App</title>
  <link rel="manifest" href="manifest.json">
</head>
<body>    ... </body> </html>

```

Reconnaissance du fichier "*web app manifest*" au sein de la partie "*Application*" de "*outils de développement*" du navigateur "Chrome Desktop" :



2.3. Effets au sein du navigateur "Chrome Desktop" (tests)



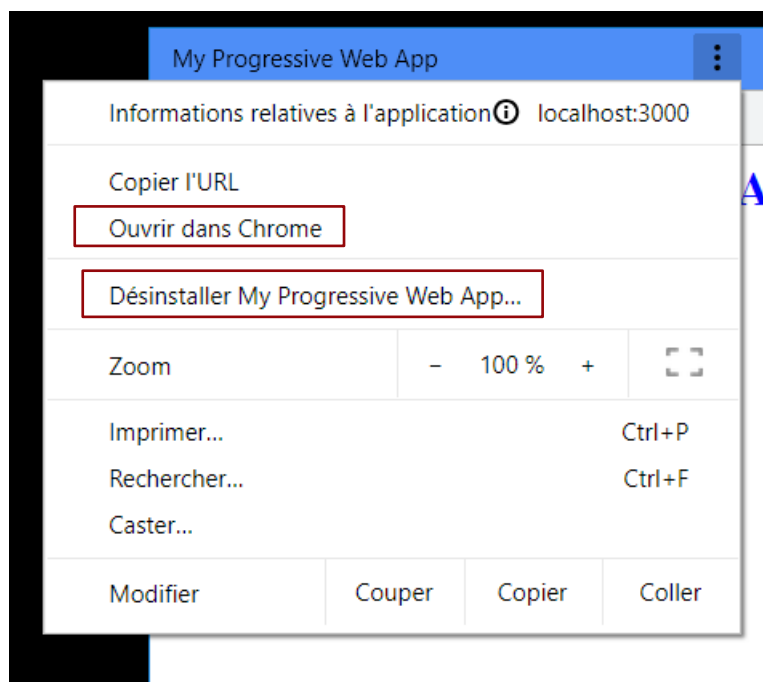
==> cette action génère un icône raccourci sur le fond du bureau :



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 doit 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;
}
```

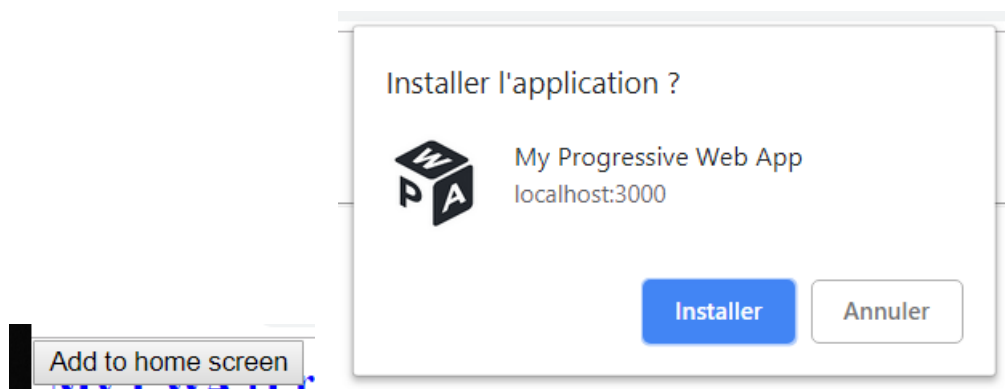
dans un bloc de script en fin de partie "body" :

```
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 des 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 `ngsw-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*

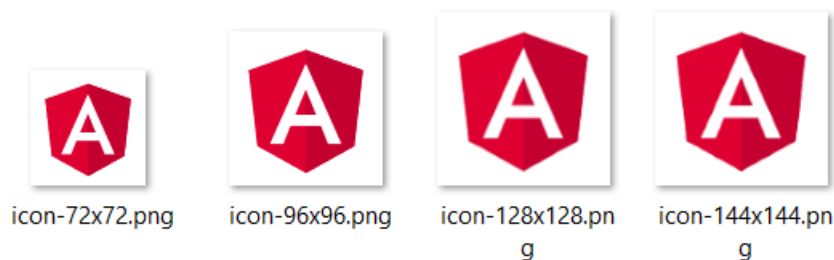
```
{
  "name": "myapp",
  "short_name": "myapp",
  "theme_color": "#1976d2",
  "background_color": "#fafafa",
  "display": "standalone",
  "scope": "/",
  "start_url": "/",
  "icons": [
    {
      "src": "assets/icons/icon-72x72.png",
      "sizes": "72x72",
      "type": "image/png"
    },
    {
      "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 " <i>prefetch</i> " ou " <i>lazy</i> "
dataGroups	fichiers (souvent "json") de données fabriqués dynamiquement (souvent via des web services "rest") . paramétrages de type " <i>maxAge</i> " , ...

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 prédéfinie 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 prédéfini **ngsw-worker.js** dans **app.module.ts** :

```
...
ServiceWorkerModule.register('/sw-master.js', { enabled: environment.production })
...
```

4. Mode "offLine" et indexed-db

4.1. 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 .

4.2. 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

5. IndexedDB et idb

5.1. Présentation de IndexedDB , 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/indexeddb>

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

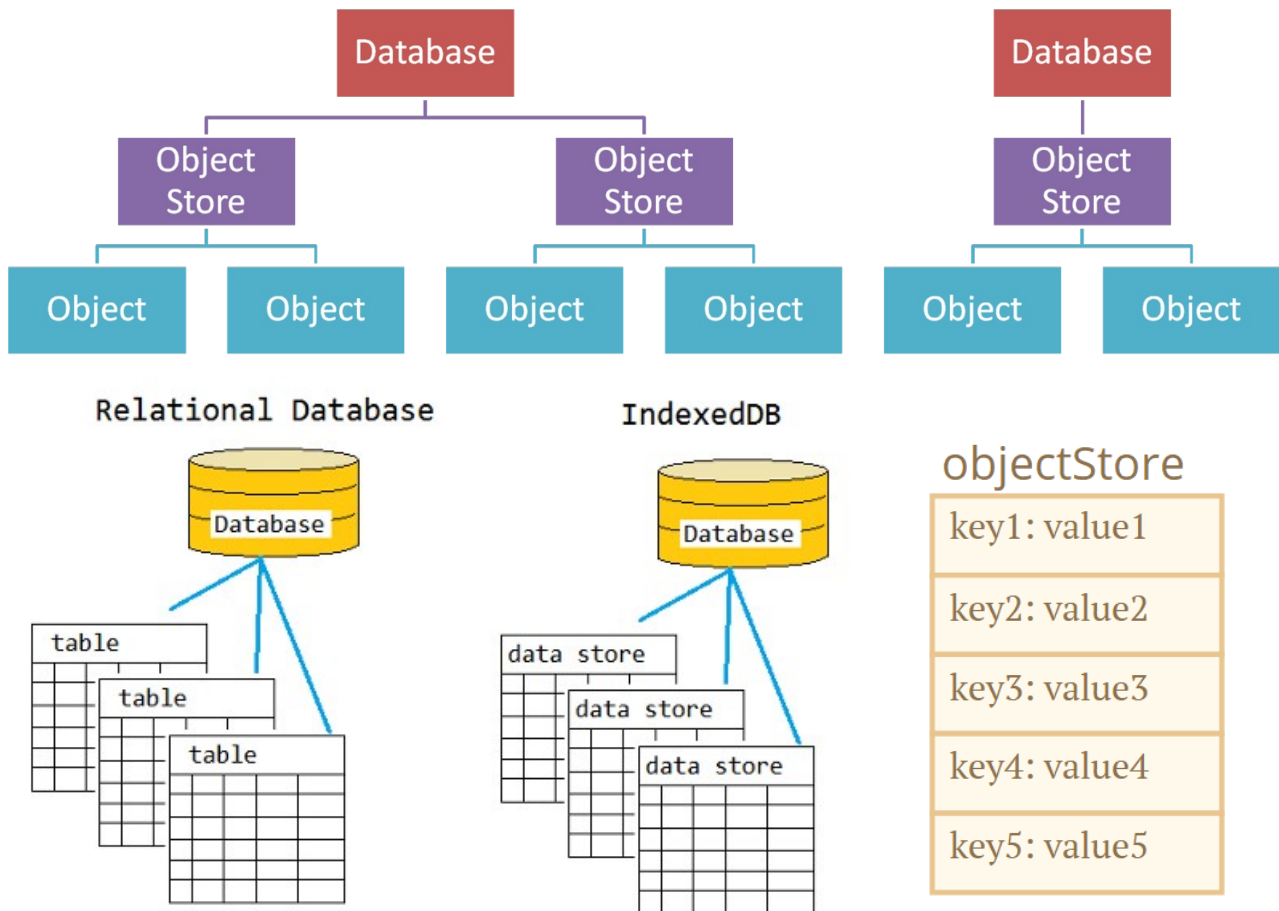
bien expliqués:

- <https://developers.google.com/web/ilt/pwa/working-with-indexeddb> (*attention: ancienne version*)

Exemple IndexedDB avec Promises et async/await:

- <https://medium.com/@filipvitas/indexeddb-with-promises-and-async-await-3d047ddd313>

5.2. Structure de IndexedDB



5.3. 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, IDBPDatabase } from 'idb';
```



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

```
    private currentIdb : IDBPDatabase = null;
```

```
//méthode asynchrone pour ouvrir la base 'my-idb'
```

```
//en créant si besoin l' objectStore 'products' avec options :
```

```
private openMyIDB() : Promise<IDBPDatabase>{
    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<IDBPDatabase>{
    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();
}

```

6. Socket.io

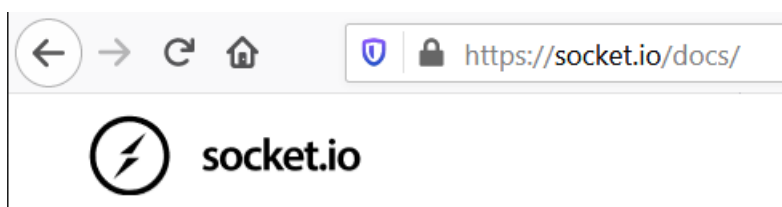
6.1. Présentation de Socket.io

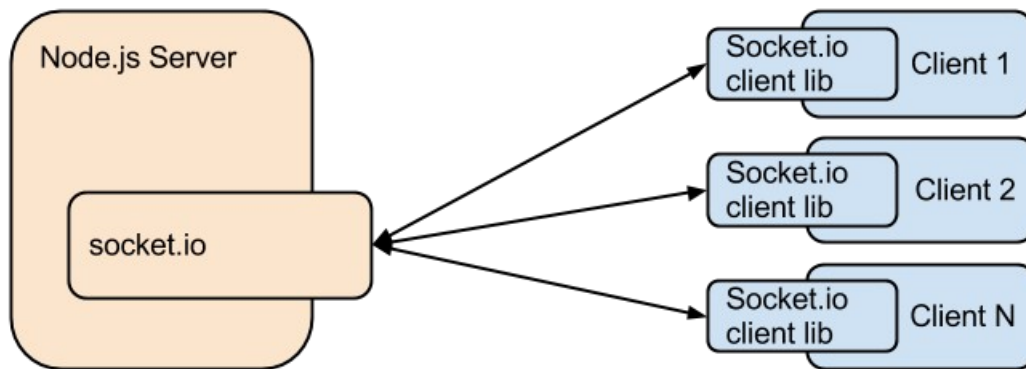
Socket.io est une **api javascript** qui encapsule et améliore la prise en charge des "websockets" .

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>

6.2. 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");
});

```

6.3. 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;

})

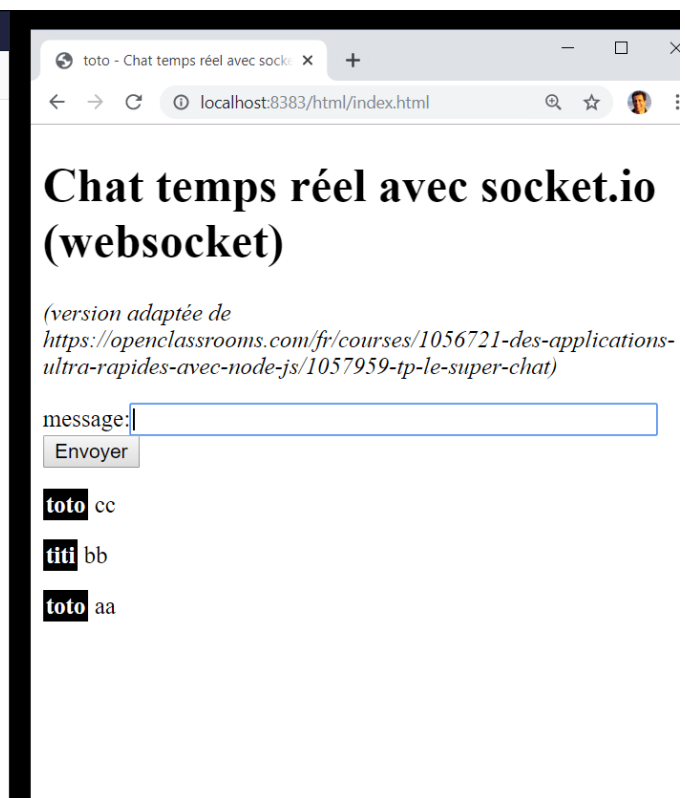
// Lorsqu'on click sur le bouton, on transmet le message et on l'affiche sur la page
document.querySelector('#envoi_message').addEventListener('click',function () {
  var message = zoneMessage.value;
  socket.emit('message', message); // Transmet le message aux autres
  insereMessage(pseudo, message); // 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 ?



6.4. 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 with 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;}

```

