

Angular

-

aspects avancés



Table des matières

I - Quelques repères sur le framework angular.....	4
1. Framework web Angular (cadre).....	4
II - composants , formulaires , animations.....	13
1. Composants "angular" avancés et animations.....	13
2. Zones et Change-detection.....	17
3. ng-content et ng-template.....	24
4. ViewChild / ContentChild.....	26

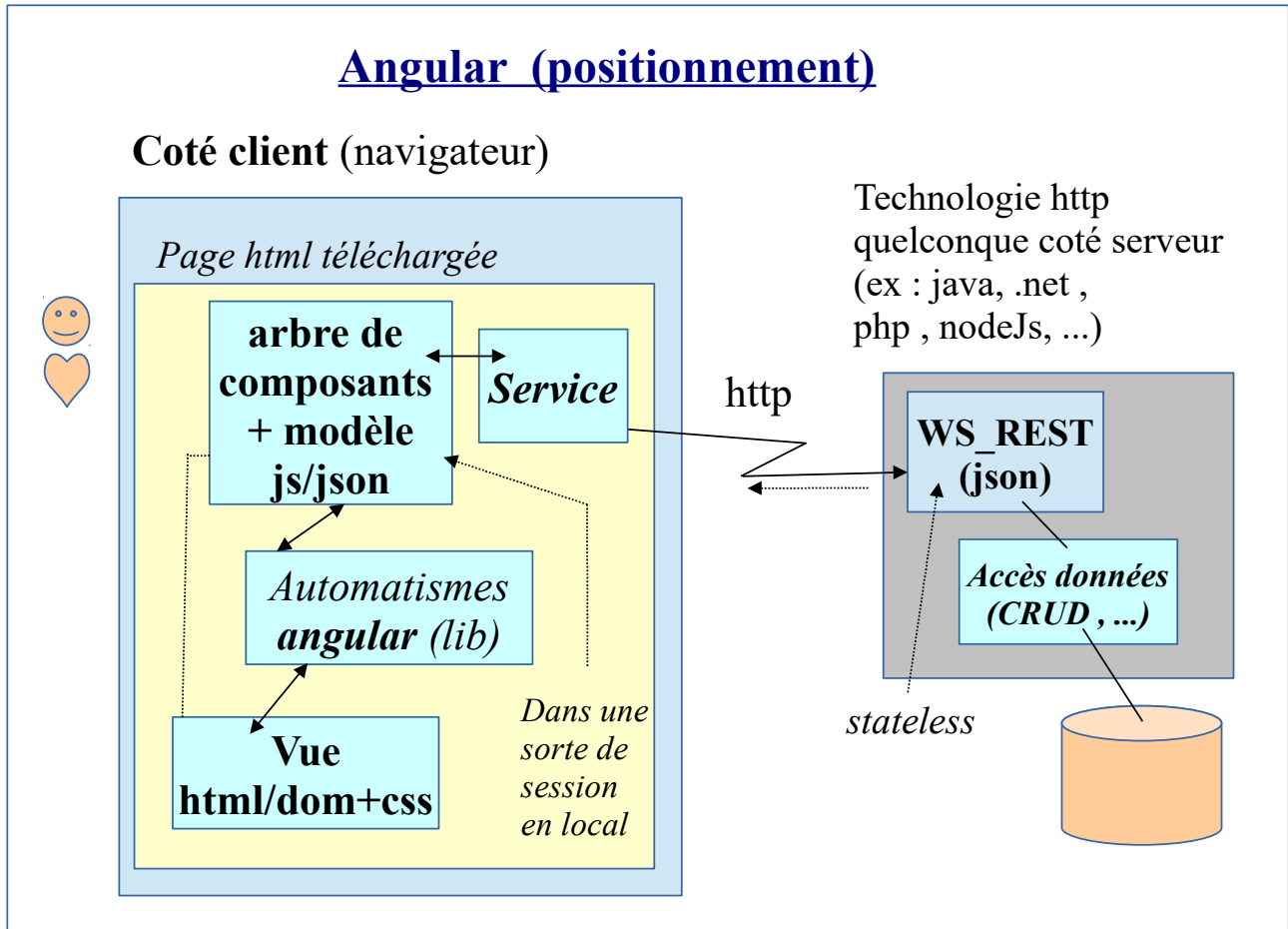
5. Les directives (angular2).....	31
6. Animations (triggers).....	36
7. Contrôle des formulaires.....	42
III - Rxjs , services , injections.....	48
1. Injection de dépendances "Angular".....	48
2. Aspects plus ou moins avancés de RxJs et des services angular.....	56
3. Jetons "JWT" et intercepteurs de sécurité.....	57
IV - modules & routing (avancés).....	62
1. Rappels sur les bases du routing angular.....	62
2. fallback route (**) in last position.....	65
3. Sous niveau de routage (children).....	65
4. Routes conditionnées (via gardiens).....	66
5. router-outlet annexe.....	68
6. Resolver et autres gardiens.....	69
7. Lazy loading.....	72
V - aot, ivy, internationalisation.....	76
1. JIT vs AOT (Ahead-Of-Time) pour angular 4,6,8.....	76
2. ivy (à partir de angular 9).....	77
3. internationalisation (i18n).....	78
VI - Tests unitaires et "end-to-end" angular.....	84
1. Différent types de tests autour de angular.....	84
2. Test "end-to-end / e2e".....	85
3. Contexte et Config tests "karma" / jasmine.....	87
4. Attention à la cohérence des tests.....	89
5. Tests unitaires élémentaires.....	90
6. Lancement tests "Angular" avec ng test et "karma".....	93
7. Tests unitaires "angular"(composants, service, ...)......	94
VII - PWA , Services workers, mode déconnecté.....	110
1. Mode "offLine" et indexed-db.....	110
2. IndexedDB et idb.....	111
3. PWA (Progressive Web App) – aperçu général.....	115
4. Service Worker (essentiel).....	118
5. Web App Manifest / add to home screen.....	124

6. Service-worker et pwa pour Angular.....	131
VIII - Ecosystème angular et extensions.....	140
1. Ecosystème angular (extensions).....	140
2. Différentiation "dev" et "prod" via environnement.....	141
3. Introduction à "Angular-universal".....	143
IX - Divers aspects avancés de angular.....	157
1. Divers Aspects avancés de Angular.....	157
X - Annexe – ngx-bootstrap.....	164
1. Extension "ngx-bootstrap" pour angular.....	164
2. Angular-Material (bibliothèque de composants).....	169
3. Essentiel de "flex-layout" (en intégration angular).....	171
4. Quelques composants "angular-material".....	172
XI - Annexe – socket.io.....	184
1. Socket.io.....	184
XII - Annexe – Ionic/cordova et NativeScript.....	192
1. Cordova (utilisé en interne par ionic).....	192
2. Présentation de ionic.....	194
3. Installation de Ionic.....	196
4. Utilisation pratique de Ionic.....	197
5. Structure d'une application ionic.....	200
6. NativeScript angular (pour applications mobiles).....	203

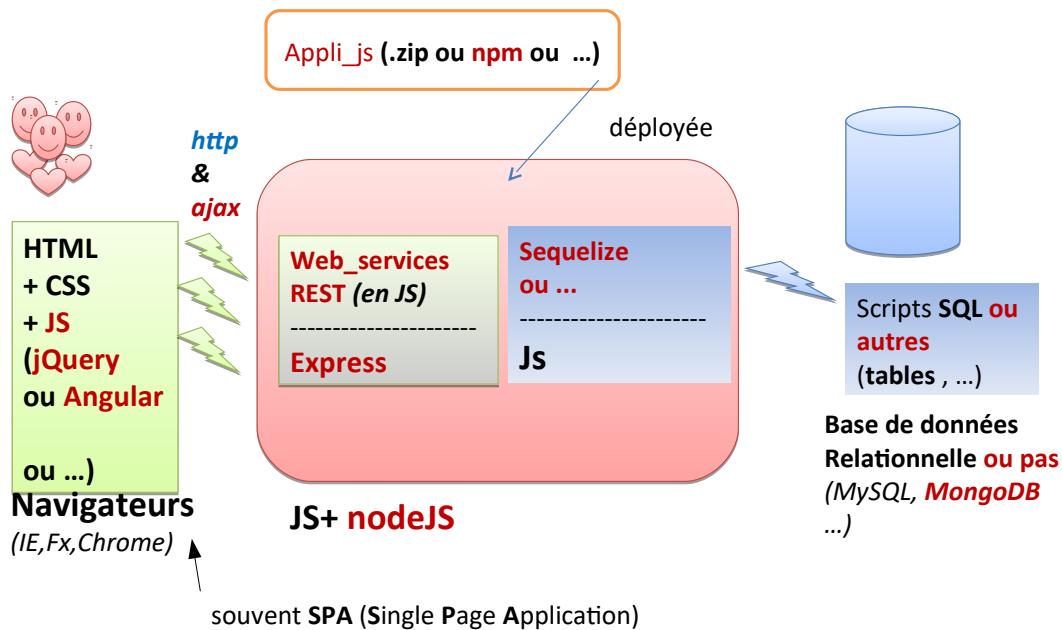
I - Quelques repères sur le framework angular

1. Framework web Angular (cadre)

1.1. Positionnement du framework "Angular"



Env exécution NodeJs



Evolution de angular (versions)

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

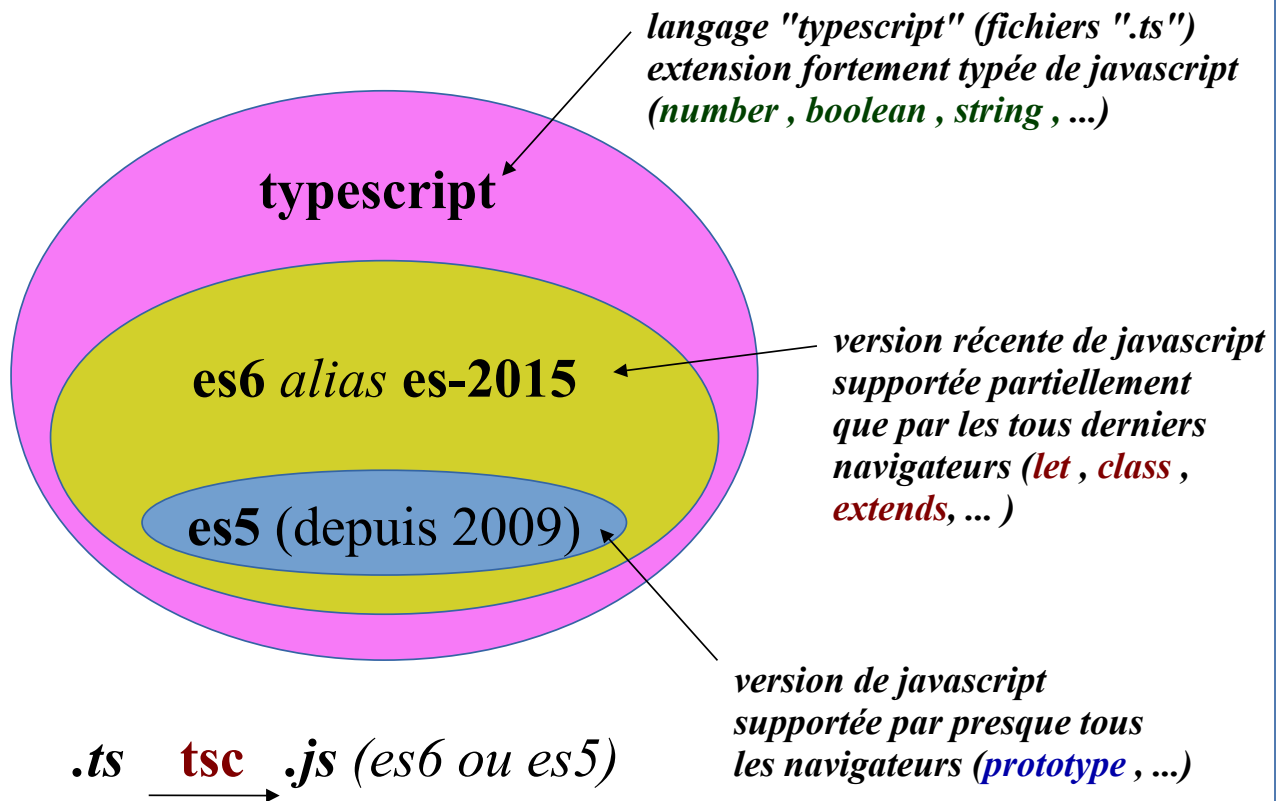
angular 2 (fin 2016) — typescript , composants, avec bugs

angular 4.0 à 4.2 (début 2017) — moins bugs , angular-cli

angular 4.3 et 5.x (fin 2017) — ~~http~~ --> httpClient

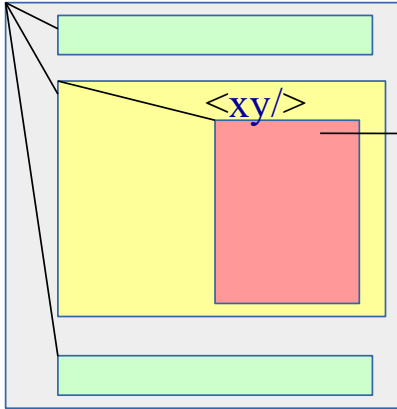
angular 6 , 7 , 8 (2018, 2019) — RxJs et angular enfin stable

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



1.2. Binding angular

Page "angular" constituée d'une hiérarchie 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

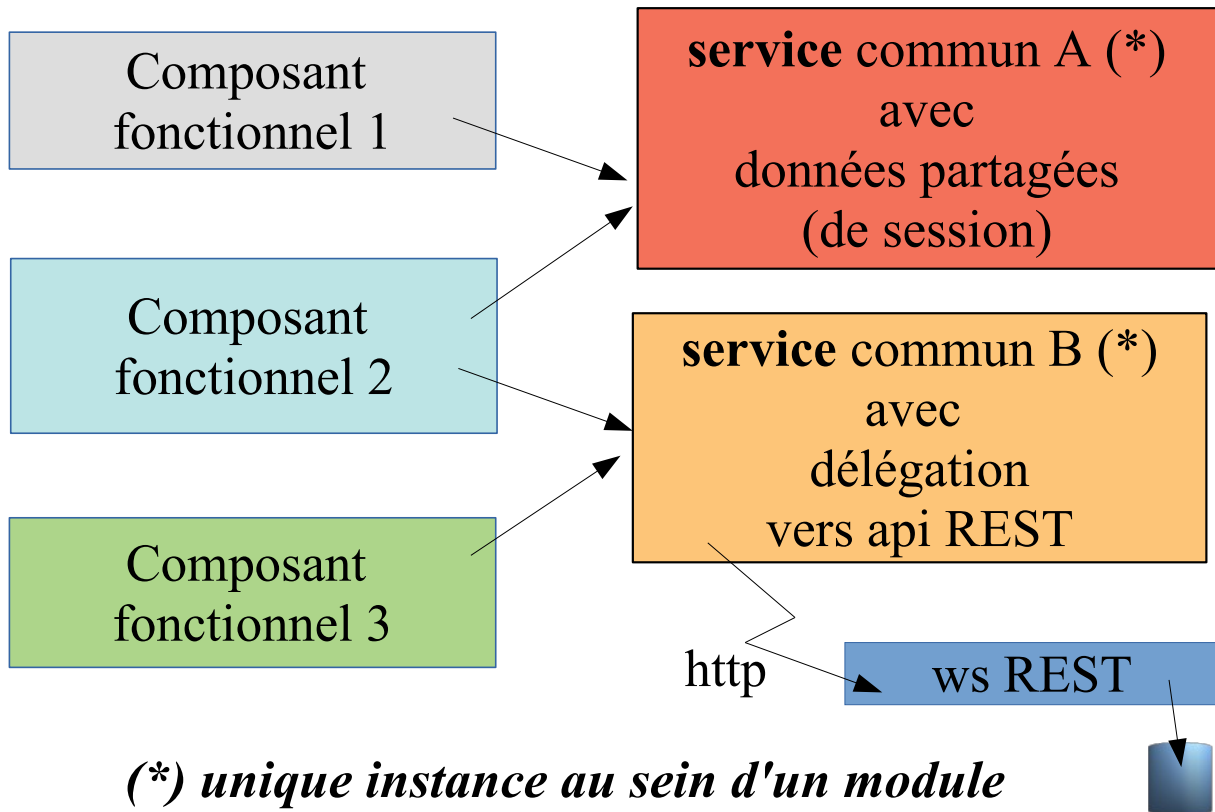
```
<button (click)=
  "onNewXy($event)">
<p>{{data.label}}</p>
<input [(ngModel)]
  ="data.value">
```

(Modèle orienté objet)

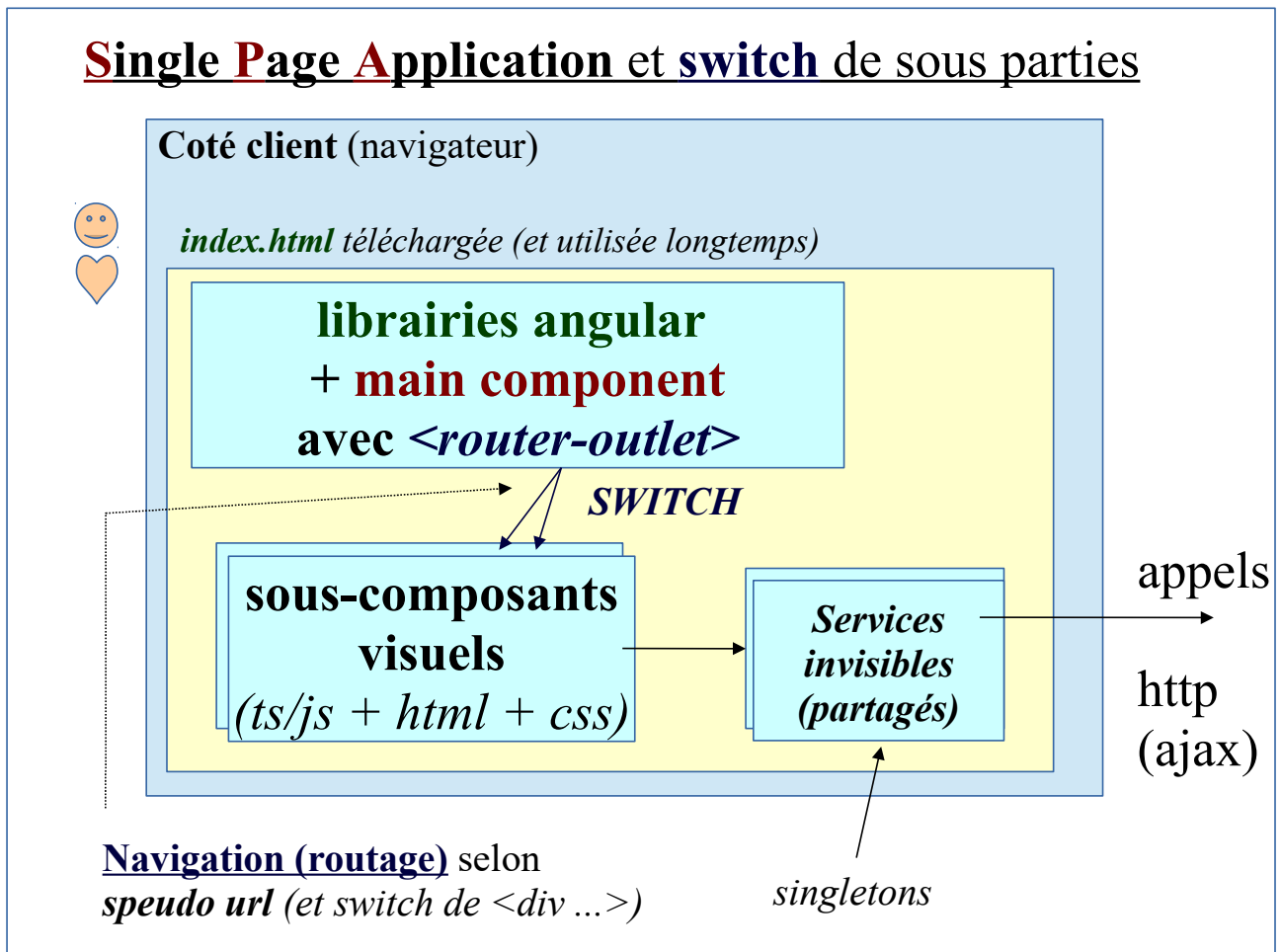
```
data.id
.label
.value
```

1.3. Services

Services pour composants fonctionnels



1.4. Structure "Single Page" et routage angular



De façon à ce que le code javascript (librairies angular + code de l'application) soit converté 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 .

1.5. Angular-cli

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 --prod -- construction de bundles (pour déploiement)

ng ...

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

Il serait possible (durant la phase de développement) de :

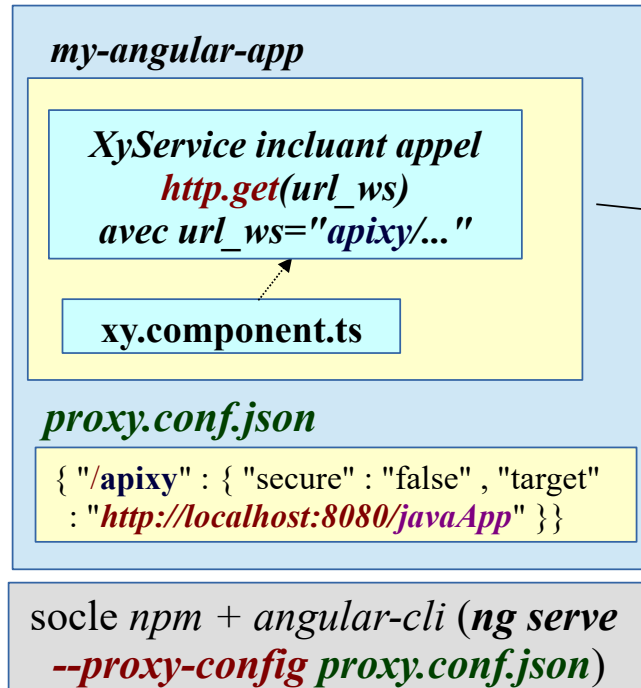
- utiliser des URLs absolues (ex : `http://localhost:8282/xxx/yyy`) pour appeler des Web services "REST" (java ou php ou nodeJs/express ou ...) via angular et ajax
- paramétrer des autorisations "CORS" du côté serveur (code java ou js/express ou ...)
- configurer des switchs d'URL côté angular

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

- toujours utiliser des URLs relatives (ex : `xxx/yyy`) 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`) .

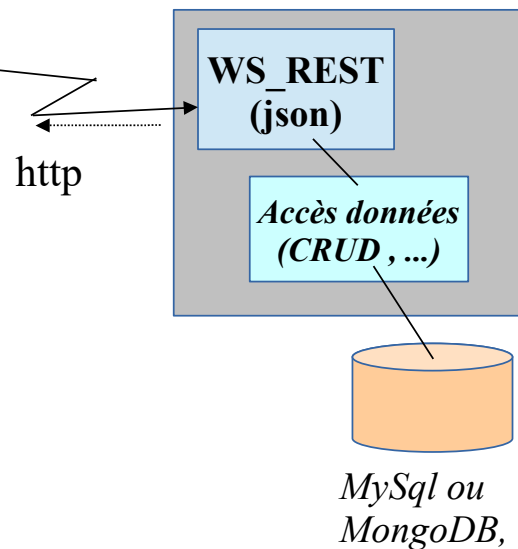
Environnement de développement Angular (v2,v4,...)

partie cliente "Angular"
([http://localhost:4200/...](http://localhost:4200/))



Partie "back-end" / WS-REST

[*http://localhost:8080/javaApp/apixy*](http://localhost:8080/javaApp/apixy)
ou bien
[*http://localhost:8082/nodeApp/apixy*](http://localhost:8082/nodeApp/apixy)



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

avec **proxy.conf.json**

```
{ "/apixy" : { "secure" : "false" ,  
              "target" : "http://localhost:8080/javaApp" }  
}
```

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

ou bien

avec **proxy.conf.json**

```
{ "/apixy" : { "secure" : "false" ,  
              "target" : "http://localhost:8282/nodeApp" }  
}
```

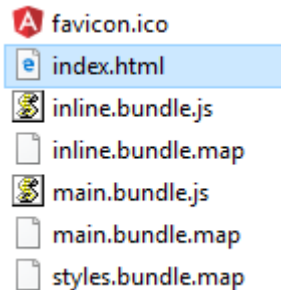
pour que les requêtes d'urls relatives **apixy/..** soient redirigées vers une appli. nodeJs/express .

Ne fonctionnant qu'en mode "développement" (via **ng serve**) , certains équivalents de **-proxy-config** en mode "production" seront exposés au sein des paragraphes ci-après.

1.7. Bases de la mise en production d'une application angular

ng build (et **ng build --prod**) génère des fichiers dans le répertoire **my-app/dist** .

Contenu du répertoire **my-app/dist** après la commande "**ng build**" (par défaut en mode **--dev**) :



ng build --prod est quelquefois accompagné de quelques "bugs" avec certaines anciennes versions de angular-cli .

Lorsque le mode "**--prod**" fonctionne , les fichiers "bundle" générés sont compressés au format ".gz".

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'utilisateur 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** .

II - composants , formulaires , animations

1. Composants "angular" avancés et animations

1.1. Brefs rappels sur les bases des composants "angular"

@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)

1.2. Cycle de vie précis d'un composant angular :

Interfaces (à facultativement implémenter)	Méthodes (une par interface)	Moment où la méthode est appelée automatiquement par angular2
(1) OnChanges	ngOnChanges()	dès changement de valeur d'un "input binding" (exemple : "propriété initialisée selon niveau parent")
(2) OnInit	ngOnInit()	à l'initialisation du composant et après les premiers éventuels ngOnChanges() et après constructeur et injections
(3) DoCheck	ngDoCheck()	permet éventuellement d'indiquer des changements si Angular ne les a pas détecté (avant un futur réaffichage)
(4) AfterContentInit	ngAfterContentInit()	est déclenchée à l'initialisation après la projection de contenu (*) (après ngOnInit() et avant ngAfterViewInit())
(5) AfterContentChecked	ngAfterContentChecked()	déclenchée après la détection de changement dans le contenu projeté
(6) AfterViewInit	ngAfterViewInit()	déclenchée après l'initialisation de la vue du composant et après l'initialisation des vues des composants enfants.
(7) AfterViewChecked	ngAfterViewChecked()	est déclenchée après détection d'un changement dans la vue du composant et dans les vues des composants enfants.
(8) OnDestroy	ngOnDestroy()	juste avant destruction d'un composant

Ces méthodes sont appelées dans l'ordre 1,2,...7 lors de l'initialisation et du premier affichage d'un composant . Par la suite seules certaines méthodes seront redéclenchées après une détection de valeurs modifiées (ngOnChanges() , ngDoCheck() , ngAfterContentChecked() , ngAfterViewChecked())

(*) NB : on appelle "**projection de contenu**" la prise en compte (au niveau d'un sous composant) d'un contenu (choisi par le parent) imbriqué entre début et fin de balise du sous composant

exemple :

```
<tabset>
  <tab heading="calcul tva">
    <app-tva></app-tva> <!-- contenu projeté dans composant tab -->
  </tab>
...
```

NB : La potentielle erreur "**ExpressionChangedAfterItHasBeenCheckedError**" est déclenchée en développement seulement pour indiquer qu'une erreur de conception risque de compromettre la détection de changement et empêcher la mise à jour correcte de la vue. (ex : this.updatedValue = 'new_value'; n'est pas à faire dans ngAfterContentChecked())

Pour approfondir le sujet (mécanismes internes de angular) :

<https://cdiese.fr/angular-change-detection/>

ngDoCheck() n'est à coder que dans des cas ultra-pointus (en s'appuyant sur `KeyValueDiffers` ou `IterableDiffers`)

Lien pour approfondir le sujet (si nécessaire) :

<https://www.concretepage.com/angular/angular-ngdocheck>

1.3. Détails sur ngOnChanges():

Cette *callback* est exécutée si le composant contient des propriétés en entrée (notamment avec le décorateur `@Input()`).

Cette *callback* peut être implémentée avec un argument de type `SimpleChanges`:

```
void ngOnChanges(changes: SimpleChanges): void {}
```

Elle sera alors déclenchée une seule fois (globalement pour toutes les propriétés modifiées)

Exemple :

```
ngOnChanges(changes: SimpleChanges){  
  console.log( "changes=" + JSON.stringify(changes));  
}
```

--> premier appel (avec valeurs initiales choisies par le parent) :

```
changes={"compteur":{"currentValue":1,"firstChange":true},  
  "message":{"currentValue":"msg","firstChange":true},  
  "donnees":{"currentValue":{"num":1,"label":"label"},"firstChange":true}}
```

--> appels ultérieurs (suite à changements opérés par le parent) :

```
changes={"compteur":{"previousValue":1,"currentValue":2,"firstChange":false}}
```

puis

```
changes={"message":{"previousValue":"msg","currentValue":"msg*","firstChange":false}}
```

@Input et ngOnChanges() selon modifications effectuées

template d'un composant parent :

```
counter : {{counter}} <button (click)="counter=counter+1">++</button> ,
msg : {{msg}} <button (click)="msg=msg+'*'">++</button> ,
data : ({{data.num}} <button (click)="data.num=data.num+1">++</button>,
      {{data.label}} <button (click)="data.label=data.label+'*'">++</button>
    )
<my-child [compteur]="counter" [message]="msg" [donnees]="data"> </my-child>
```

Affichage initial :

```
values in this (parent) component: counter : 1 ++ , msg : msg ++ , data : (1 ++ , label ++ )
with 3 @input/property links / [compteur]="counter" [message]="msg" [donnees]="data" :
values in this (child) component: compteur : 1 ++ , message : msg ++ , donnees : (1 ++ , label ++ )
```

Après modifications effectuées sur le parent (et automatiquement répercutées sur "child" avec appels de ngOnChanges() sur "child") :

```
values in this (parent) component: counter : 2 ++ , msg : msg* ++ , data : (1 ++ , label ++ )
with 3 @input/property links / [compteur]="counter" [message]="msg" [donnees]="data" :
values in this (child) component: compteur : 2 ++ , message : msg* ++ , donnees : (1 ++ , label ++ )
```

Après modifications effectuées uniquement sur le sous composant "child" .

pas de remontées de modifications vers le parent tant que pas de @Output / event

```
values in this (parent) component: counter : 2 ++ , msg : msg* ++ , data : (1 ++ , label ++ )
with 3 @input/property links / [compteur]="counter" [message]="msg" [donnees]="data" :
values in this (child) component: compteur : 4 ++ , message : msg*** ++ , donnees : (1 ++ , label ++ )
```

Dans cet exemple l'objet data (comportant lui même des sous parties modifiables) est initialement passé par référence (via **[donnees]="data"**) .

Cet objet de données (se retrouvant partagé entre les composants parent et enfant) conduit à un affichage automatiquement synchronisé dans les 2 sens (sans aucun appel à ngOnChanges() tant que l'on modifie les sous parties de cet objet sans changer l'objet data lui même) .

```
values in this (parent) component: counter : 2 ++ , msg : msg* ++ , data : (5 ++ , label*** ++ )
with 3 @input/property links / [compteur]="counter" [message]="msg" [donnees]="data" :
values in this (child) component: compteur : 4 ++ , message : msg*** ++ , donnees : (5 ++ , label*** ++ )
```


2. Zones et Change-detection

Un composant angular a techniquement 2 parties complémentaires :

- une partie "DOM" (conforme à l'api des navigateurs)
- une partie "données et mécanismes angular"

Les mécanismes internes du framework "Angular" utilisent la bibliothèque "**Zone.js**" pour gérer contextuellement certaines modifications de données au niveau de l'application prise en charge par le navigateur .

Une "Zone" est un élément abstrait qui peut être vu comme proche de "ThreadLocal" et qui correspond à "un contexte d'exécution interchangeable et durable " ou "zone mémoire bien conservée" permettant de faire un lien entre plusieurs micro-exécutions de tâches potentiellement asynchrones .

Les évènements extérieurs à Angular qui peuvent subvenir et qui sont automatiquement gérés par Zone.js et la zone angular sont:

- Les évènements Javascript comme les *clicks*, *keydown*, *submit*, *input* etc...
- Les évènements XHR provenant de l'objet [XmlHttpRequest](#),
- Les évènements provenant de *Timers* comme *setTimeout()* ou *setInterval()*.

Cette gestion automatique des événements par Zone.js et par le framework angular permet de :

- déterminer (par algorithme sophistiqué) , les modifications qui doivent ou pas se traduire par des ré-affichages actualisés
- détecter et répercuter les changements de certaines données
- assurer la cohérence de l'ensemble de l'interface graphique de l'application
- bien gérer le binding (data --- vue) .

Les articles suivants (très techniques , très complets) permettent si besoin d'approfondir le sujet :

- <https://cdiese.fr/angular-change-detection/>
- <https://cdiese.fr/angular-customize-change-detection/>
- <https://blog.thoughttram.io/angular/2016/02/01/zones-in-angular-2.html>

2.1. NgZone et le contournement contrôlé des détections de changements

En injectant **NgZone** dans un constructeur de composant angular , on peut contrôler par programmation si une modification de données sera effectuée ou pas au dehors de la zone "angular" .

Sur **this.zone** (de type **NgZone**) on peut appeler :

- **runOutsideAngular()** pour exécuter du code en dehors de la détection de changements Angular.
- **run()** pour exécuter du code dans une zone Angular. La détection de changements sera exécutée.

La variante **runTask()** permet d'exécuter du code dans une zone Angular de façon asynchrone. La détection de changements sera exécutée.

2.2. Exemple avec runOutsideAngular:

```
<p>inside-outside-zone works!</p>
cpt=<span class="enEvidence">{{cpt}}</span>
<input type="button" value="reset to 0" (click)="onResetCpt()" /><br/>
<input type="button" value="++to10 inside" (click)="onIncrCpt()" /> <br/>
<input type="button" value="++to10 outside" (click)="onIncrCptOutside()" /> <br/>
```

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

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

  public cpt : number = 0;

  constructor(private zone: NgZone) { }

  public recursIncrCpt(doneCallback: () => void){
    this.cpt++;
    console.log(`Current cpt: ${this.cpt}`);
    if (this.cpt < 10) {
      window.setTimeout(() => {
        this.recursIncrCpt(doneCallback);
      }, 100);
    } else {
      doneCallback();
    }
  }

  public onIncrCpt(){ this.recursIncrCpt(() => console.log('inside Done!')); }
  public onResetCpt(){ this.cpt = 0; }

  public onIncrCptOutside(){
    this.zone.runOutsideAngular(() => {
      //this.recursIncrCpt(() => console.log('Done!')); //no display final 10
      this.recursIncrCpt(
        () => this.zone.run(() => {
          console.log('outside Done!'); //indirectly update UI (to final 10)
        })
      );
    });
  } //end of this.zone.runOutsideAngular

} //end of onIncrCptOutside
```

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

Comportement / résultats :

cpt=10 reset to 0
 ++to10 inside
 ++to10 outside

En cliquant sur le premier bouton (déclenchant `onIncrCpt()`) , l'appel normal/ordinaire de la méthode récursive **recursIncrCpt()** s'effectue par défaut dans la zone angular et chaque mise à jour de cpt (de 1 à 10 toutes les 100 ms) est automatiquement détectée par angular et conduit à une réactualisation de l'affichage : cpt=1 puis cpt=2 puis cpt=3 puis cpt=10 .

En cliquant sur le deuxième bouton (déclenchant `onIncrCptOutside()`) , l'appel de la méthode récursive **recursIncrCpt()** s'effectue au dehors de la zone d'angular et chaque mise à jour de cpt (de 1 à 10 toutes les 100 ms) n'est pas automatiquement détectée par angular et ne conduit pas à une réactualisation de l'affichage .

Il s'affiche toujours cpt=0 avec la callback `() => console.log('Done!')`.

```
public onIncrCptOutside(){
  this.zone.runOutsideAngular(() => {
    this.recursIncrCpt(() => console.log('Done!')); //no display final 10 , cpt=0
  });
}
```

Cependant en ajustant la callback avec un appel explicite à `() => this.zone.run(...)`

```
public onIncrCptOutside(){
  this.zone.runOutsideAngular(() => {
    this.recursIncrCpt(
      () => this.zone.run(() => {
        console.log('outside Done!'); //indirectly update UI (to final 10)
      })
    );
  });
}
```

on termine via un retour en zone angular et cela provoque l'affichage de cpt=10

On passe ainsi directement de l'affichage cpt=0 à cpt=10 (les incrémentation intermédiaires 1, 2 , 3 ... 9 ne sont affichées qu'à la console selon un appel explicite) .

Cet exemple "cas d'école" montre le principe de fonctionnement.

Au sein d'un vrai projet angular , on peut par exemple effectuer tout un tas de prétraitement hors zone angular pour éviter des affichages inutiles et gagner en performance .

Seul le dernier traitement pourra être effectué au sein de la zone d'angular et conduire à un affichage intéressant .

2.3. Variante possible avec ApplicationRef et tick()

Inversement , après des modifications de données effectuées "hors zone angular" et ne conduisant normalement pas à des ré-affichages (car changements alors non détectés) , on peut effectuer un appel explicite de la méthode **tick()** et un élément injecté de type **ApplicationRef**.

Ceci permet de demander explicitement une prise en compte des données modifiées au niveau de l'ensemble des composants de l'application .

```
<p>inside-outside-zone works!</p>
cpt=<span class="enEvidence">{{cpt}}</span>
...
<input type="button" value="++to10 outside with explicit tick"
      (click)="onIncrCptOutsideWithExplicitTick()" /> <br/>
```

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

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

  public cpt : number = 0;

  constructor(private zone: NgZone, private applicationRef: ApplicationRef) {}

  public recursIncrCptWithExplicitTick(doneCallback: () => void){
    this.cpt++;
    this.applicationRef.tick(); //explicit global check/detection of changes to display
    console.log(' Current cpt: ${this.cpt}`);

    if (this.cpt < 10) {
      window.setTimeout(() => {
        this.recursIncrCptWithExplicitTick(doneCallback);
      }, 100);
    } else {
      doneCallback();
    }
  }

  public onIncrCptOutsideWithExplicitTick(){
    this.zone.runOutsideAngular(() => {
      this.recursIncrCptWithExplicitTick( () => {
        console.log('outside Done with explicit ticks !');
      });
    });
  }
  ...
}
```

Comportement :

- Sans l'appel à tick() , l'affichage reste bloqué à cpt = 0
- Avec l'appel explicite à **this.applicationRef.tick()**; (effectuer ici récursivement au sein de ce cas d'école) , l'affichage est régulièrement réactualisé (cpt=1 puis cpt=2 puis cpt=3 puis cpt=10 .)

2.4. Variante rare et déconseillée avec affichage natif direct

```
<input type="button" value="++to10 outside direct display"
      (click)="onIncrCptOutsideDirectDisplay()" /> <br/>
cpt (#e_cpt)=<span #e_cpt class="enEvidence">{{cpt}}</span>
```

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

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

  public cpt : number = 0;

  @ViewChild('e_cpt', { static: true })
  e_cpt: ElementRef<HTMLElement>

  constructor(private zone: NgZone) { }

  public recursIncrCpt(doneCallback: () => void){
    this.cpt++;
    console.log(`Current cpt: ${this.cpt}`);

    if (this.cpt < 10) {
      window.setTimeout(() => {
        this.recursIncrCpt(doneCallback);
      }, 100);
    } else {
      doneCallback();
    }
  }

  public onIncrCptOutsideDirectDisplay(){
    this.zone.runOutsideAngular(() => {
      this.recursIncrCpt( () => {
        this.e_cpt.nativeElement.innerText = ""+this.cpt;
        //BUG : cannot reset after , cannot restore automatic update
        console.log('outside Done with direct display !'); //indirectly update UI (to final 10)
      });
    });
  }
}
```

```
}  
  
...  
  
}
```

Comportement :

via la décoration `@ViewChild('e_cpt', { static: true })` on peut initialiser côté .ts une variable `e_cpt` de type `ElementRef<HTMLElement>` correspondant à la balise `<p>` , `` ou `<div>` marquée via `#e_cpt` au sein du template html .

Finalement , l'instruction `this.e_cpt.nativeElement.innerHTML = ""+this.cpt;` déclenchée hors de la zone angular conduit à un affichage direct natif ce qui permet d'actualiser immédiatement l'affichage .

Cette solution n'est malheureusement pas parfaite car ça perturbe la bonne interprétation ultérieure de `{{cpt}}` au sein de la balise marquée ici via `#e_cpt` .

2.5. Composant avec "change detection" personnalisé

NB: on peut éventuellement **définitivement désactiver l'exécution automatique de la détection sur un composant** via

```
@Component({  
  selector: ...,  
  templateUrl: ...,  
  changeDetection: ChangeDetectionStrategy.OnPush  
})
```

mais c'est pas très souple !!!

En injectant l'élément technique **ChangeDetectorRef** au sein d'un composant , on peut contrôler finement l'activation ou la désactivation de la détection automatique des changements :

- Désactiver l'exécution automatique de la détection : **ChangeDetectorRef.detach()**,
- Ré-activer la détection automatique : **ChangeDetectorRef.reattach()**,
- Forcer l'exécution de la détection : **ChangeDetectorRef.detectChanges()**,
- Vérifier si des changements sont détectés : **ChangeDetectorRef.checkNoChanges()**
- Marquer la vue du composant pour que la détection de changements soit effectuée : **ChangeDetectorRef.markForCheck()**

Exemple :

```
<p>custom-detection works with counter={{counter}}</p>
<p><input type='button' (click)='disableChangeDetection()' value='Disable change detection' /></p>
<p><input type='button' (click)='enableChangeDetection()' value='Enable change detection' /></p>
```

```
import { Component, OnInit, ChangeDetectorRef } from '@angular/core';
@Component({
  selector: 'app-custom-detection',
  templateUrl: './custom-detection.component.html',
  styleUrls: ['./custom-detection.component.scss']
})
export class CustomDetectionComponent implements OnInit {
  public counter : number = 0;
  constructor(private changeDetectorRef: ChangeDetectorRef) { }

  ngOnInit(): void { setInterval(() => { this.counter++; }, 1000); }
  disableChangeDetection(): void { this.changeDetectorRef.detach(); }
  enableChangeDetection(): void { this.changeDetectorRef.reattach(); }
}
```

Comportement :

En cliquant sur le premier bouton , conduisant à l'instruction `this.changeDetectorRef.detach()`; les changements ne sont plus détectés et l'affichage reste bloqué à une certaine valeur (ex ; 2)

custom-detection works with counter=2

Disable change detection

Enable change detection

En cliquant sur le deuxième bouton , conduisant à l'instruction `this.changeDetectorRef.reattach()`; les changements sont de nouveau détectés et l'affichage est régulièrement réactualisé (ex ; 47 puis 48 puis 49)

custom-detection works with counter=49

Disable change detection

Enable change detection

3. ng-content et ng-template

3.1. Projection des éléments imbriqués

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

Au sein du template HTML d'un composant le (ou le paquet de) sous-composant(s)/sous-balises sera vu via la balise spéciale **<ng-content></ng-content>** .

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

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

```
import { Component, Input } from '@angular/core';
@Component({
  selector: 'toggle-panel',
  templateUrl: './toggle-panel.component.html',
  styleUrls: ['./toggle-panel.component.css']
})
export class TogglePanelComponent {
  toggleP : boolean =false;
  @Input()
  title : string = 'default panel title';
  constructor() { }
}
```

```
<div class="card">
  <!-- <div class="card-header bg-info"> -->
  <h4 class="card-header my-bg-primary">
    <a class="text-light" (click)="toggleP = !toggleP" >{{title}}
      <span class="fa" [class.fa-chevron-circle-down]="!toggleP"
        [class.fa-chevron-circle-up]="toggleP"></span>
    </a></h4>
  <!-- </div> -->
  <div class="card-body collapse " [class.show]="toggleP">
```



```
        <ng-content></ng-content>
    </div>
</div>
```

Exemple d'utilisation :

```
<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>
```

<div>part 1 (basic.module) ▼</div>	<div>part 1 (basic.module) ▲</div> <div>binding @Input/@Output forms pipes service</div> <div>raw date: Sun Jan 21 2018 20:34:08 GMT+0100</div> <div>date with date : Jan 21, 2018</div>
------------------------------------	---

4. ViewChild / ContentChild

Vocabulaire (issu de la norme "web component") :

Le **Shadow DOM** - est un DOM interne de votre composant défini par vous (en tant que créateur du composant) et caché de l'utilisateur final. Par exemple:

```
@Component ({
  selector: 'some-component',
  template: `
    <h1>I am Shadow DOM!</h1>
    <h2>Nice to meet you :)</h2>
    <ng-content></ng-content>
  `;
})
class SomeComponent { /* ... */ }
```

Le **Light DOM** - est un DOM qui est un utilisateur final de votre composant . Par exemple:

```
@Component ({
  selector: 'another-component',
  directives: [SomeComponent],
  template: `
    <some-component>
      <h1>Hi! I am Light DOM!</h1>
      <h2>So happy to see you!</h2>
    </some-component>
  `;
})
class AnotherComponent { /* ... */ }
```

NB :

@ViewChild et **@ViewChildren** recherchent des éléments dans **Shadow DOM** (autrement dit dans la structure du template du composant actuel)

tandis que

@ContentChild et **@ContentChildren** les recherchent dans **Light DOM** (autrement dit dans le contenu qui sera projeté via le composant parent)

4.1. Exemple avec @ViewChild et @ViewChildren

a.child.component.ts

```
... @Component({
  selector: 'app-a-child', ...
})
export class AChildComponent {
  @Input()
  name : string ="aChild";

  public sayHello() :string{
    return "hello_from_"+this.name;
  }
  ... }
```

a.child.component.html

```
<p>a-child with name={{name}}</p>
```

with.child.component.html

```
<h3>with-child works!</h3>
message (from child 1 and 2)={{message}} <br/>
message (from all children)={{messageV2}}
<hr/>
<app-a-child name="child1"></app-a-child>
<app-a-child #c2 name="child2"></app-a-child>
<app-a-child name="child3"></app-a-child>
<p #p #p1><b>p1</b></p>
<p #p><i>p2</i></p>
<p #p>p3</p>
```

with-child works!

message (from child 1 and 2)=hello_from_child1 hello_from_child2
 message (from all children)= hello_from_child1 hello_from_child2 hello_from_child3 p1 p2 p3

a-child with name=child1

a-child with name=child2

a-child with name=child3

p1

p2

p3

with.child.component.ts

```
import { Component, OnInit, ViewChild, ViewChildren,
        QueryList, ElementRef } from '@angular/core';
import { AChildComponent } from './a-child/a-child.component';

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

  message:string="";
  messageV2:string="";

  @ViewChild(AChildComponent)
  first_achild : AChildComponent; // refer to first subcomponent of type AChildComponent

  @ViewChild('c2', { static: true })
  achild_c2 : AChildComponent; // refer to subcomponent/subElement with #c2 in .html
```

```

@ViewChildren(AChildComponent)
allchilds: QueryList<AChildComponent>; // refer to subcomponents of type AChildComponent

@ViewChildren('p', { read: ElementRef }) pList: QueryList<ElementRef>; // all elt with #p
@ViewChild('p1', { read: ElementRef }) p1 : ElementRef; // elt with #p1

ngAfterViewInit() {
  // call method .sayHello() of child sub component :
  let msgChild1= this.first_achild?this.first_achild.sayHello():"";

  let msgChild2= this.achild_c2?this.achild_c2.sayHello():"";
  let allMsg = msgChild1 + " " + msgChild2;

  let allMsgV2 = "";
  for(let c of this.allchilds){
    allMsgV2 = allMsgV2+" " + c.sayHello();
  }

  for(let p of this.pList){
    allMsgV2 = allMsgV2+" " + p.nativeElement.innerText;
  }

  //this.message= allMsg;!-->ExpressionChangedAfterItHasBeenCheckedError"
  setTimeout(=> {
    this.message=allMsg;
    this.messageV2=allMsgV2;
  }, 0);
  console.log(allMsg);
  console.log("in p1:" + this.p1.nativeElement.innerHTML);
}
...
}

```

4.2. Exemple avec @ContentChild et @ContentChildren

with-content.component.html

```

<p>with-content works!</p>
<app-my-tabset>
  <app-my-tab title="tab1">
    <div #tabContent >div_a</div>
  </app-my-tab>
  <app-my-tab title="tab2">
    <div #tabContent >div_b</div>
  </app-my-tab>
  <app-my-tab title="tab3">
    <div #tabContent >div_c</div>
  </app-my-tab>
</app-my-tabset>

```

with-content works!

TAB1 TAB2 TAB3

div_c

with-content works!

TAB1 TAB2 TAB3

div_a

my-tab.component.html

```
<span class="aTab" (click)="onClick()" [style.fontWeight]="selected?'bold':'normal'">
  {{title}}</span>
```

my-tab.component.css

```
.aTab { display: inline-block; margin : 2px; padding: 2px;
  border-style: solid; border-color: blue; border-width: 1px;}
```

my-tab.component.ts

```
import { Component, OnInit, Input, AfterContentInit, ContentChild,
  ElementRef, EventEmitter, Output } from '@angular/core';
@Component({
  selector: 'app-my-tab',
  templateUrl: './my-tab.component.html',
  styleUrls: ['./my-tab.component.scss']
})
export class MyTabComponent implements AfterContentInit {

  @Input() title:string="defaultTabTitle";

  @Input() selected:boolean=false;

  @Output()
  public selectionChange : EventEmitter<{value:MyTabComponent}> =
    new EventEmitter<{value:MyTabComponent}>();

  @ContentChild("tabContent")
  contentElementRef : ElementRef; // refer to elt with #tab_content in projected content

  onClick(){
    this.selected=!this.selected;
    if(this.selected)
      this.selectionChange.emit({value:this});
  }

  constructor() { }
  ngAfterContentInit(): void {
    console.log(this.contentElementRef.nativeElement.innerText)
  }

  public tabContentAsString(){
    if(this.contentElementRef==undefined) return "";
    else return this.contentElementRef.nativeElement.innerText
  }
}
```

my-tabset.component.html

```
<ng-content></ng-content> <!-- will display tabs titles-->
<br/>
{{selectedTabContent}}
```

my-tabset.component.ts

```
import { Component, OnInit, ContentChildren, QueryList, AfterContentInit } from '@angular/core';
import { MyTabComponent } from '../my-tab/my-tab.component';

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

  selectedTabContent : string = "";

  @ContentChildren(MyTabComponent)
  tabs: QueryList<MyTabComponent>

  onSelectionChange(newSelectedTab : MyTabComponent ){
    this.tabs.forEach(tab => { tab.selected=(tab===newSelectedTab);
      if(tab === newSelectedTab )
        this.selectedTabContent = newSelectedTab.tabContentAsString();
    })
  }

  ngAfterContentInit() {
    let lastTab = this.tabs.last;
    this.tabs.forEach(tab => { console.log(tab.title);
      tab.selectionChange.subscribe(
        (evt)=>{ this.onSelectionChange(evt.value); }
      );
      tab.title=tab.title.toUpperCase();
      tab.selected=(tab===lastTab); //last tab selected by default
    })
    this.selectedTabContent = lastTab.tabContentAsString();
  }
  ...
}
```

Cet exemple "rudimentaire" de composant "onglet" est bien évidemment améliorable . Il permet cependant de montrer ce qu'il est possible de faire avec **@ContentChildren** et **@ContentChild** . En annexe (*ngx-bootstrap , angular-material*) , on trouvera des composants onglets prédéfinis.

5. Les directives (angular2)

5.1. 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>)

5.2. 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';

  constructor(private el: ElementRef) {
  }

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

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

  private _highlight(color: string)
```



```
{ this.el.nativeElement.style.backgroundColor = color; }
}
```

NB :

Sans argument, @Input() fait que la propriété exposée a le même nom que celle de la classe (public ou get / set).

Avec un argument , @Input permet de préciser un alias sur la propriété exposée (ex : 'myHighlight' plutôt que highlightColor).

@HostListener permet d'associer des noms d'événements (déclenchés sur l'élément DOM courant) à une méthode événementielle .

Utilisation :

```
<p [myHighlight]=" 'yellow' " >Highlight me!</p>
```

ou bien (plus simplement) :

```
<p myHighlight=" yellow " >Highlight me!</p>
```

Pick a highlight color

☐ Green ☐ Yellow ☐ Cyan

ou bien (avec un choix dynamique) :

```
<h4>Pick a highlight color</h4>
<div> <input type="radio" name="colors" (click)="color='lightgreen' " id="r1" />
    <label for="r1">Green</label>
    <input type="radio" name="colors" (click)="color='yellow' " id="r2" />
    <label for="r2">Yellow</label>
    <input type="radio" name="colors" (click)="color='cyan' " id="r3" />
    <label for="r3">Cyan</label>
</div>
<span [myHighlight]="color"> Highlight with choosen color</span> <br/>
```

Version encore améliorée via @HostBinding

via **@HostBinding()** on peut directement associé une propriété de la classe de directive à une propriété de style ou autre de l'élément "host" sur laquelle la directive sera appliquée.

Syntaxes possibles :

@HostBinding('style.xyz') **@HostBinding('attr.xyz')** **@HostBinding('class.xyz')**

HostBinding('value') myValue; est à l'intérieur d'une classe de directive un équivalent de **[value]="myValue"** que l'on écrirait dans un template html .

et

HostListener('click') myClick() { } est à l'intérieur d'une classe de directive un équivalent de **(click)="myClick()"** que l'on écrirait dans un template html .

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

```
@Directive({
```

```
  selector: '[myHighlight]
```

```
})
```

```
export class HighlightDirective {
```

```
  @Input('myHighlight') public highlightColor: string;
```

```
  private _defaultColor = 'red';
```

```
  @HostBinding('style.backgroundColor') backgroundColor: string;
```

```
  @HostListener('mouseenter')
```

```
    onMouseEnter() { this._highlight(this.highlightColor || this._defaultColor); }
```

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

```
  private _highlight(color: string) {      this.backgroundColor = color;    }
```

```
}
```

5.3. Directive structurelle

Une directive structurelle (ajoutant ou retirant des sous éléments dans l'arbre DOM) se programme de façon très semblable à une directive d'attribut (même décoration @Directive , même syntaxe (avec crochets) pour le sélecteur CSS) . La principale différence tient dans les éléments injectés dans le constructeur :

- TemplateRef correspond à la branche des sous éléments imbriqués (à supprimer ou ajouter ou ...)
- ViewContainerRef permet de contrôler dynamiquement le contenu via des méthodes prédéfinies telles que .clear() ou .createEmbeddedView()

Exemple "myUnless" (tiré du tutoriel officiel) :

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

@Directive({ selector: '[myUnless]' })
export class UnlessDirective {

  constructor( private _templateRef: TemplateRef<any>,
               private _viewContainer: ViewContainerRef ) {}

  @Input() set myUnless(condition: boolean) {
    if (!condition) { this._viewContainer.createEmbeddedView(this._templateRef); }
    else { this._viewContainer.clear(); }
  }
}
```

Utilisation (comme *ngIf) :

age: <input type='text' [(ngModel)]="age" />

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

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

</p>

<p *myUnless="age<18">

MAJEUR . condition "age<18" is false and myUnless is true.

</p>

6. Animations (triggers)

6.1. Animations css ordinaires (déclenchées via :hover ou ...)

Au sein d'un fichier css, une animation se configure en "keyframes" (ayant un nom logique).

On y définit les étapes de l'animation (par exemples à 0 %, 50 % , 100 % du temps) :

@keyframes *monanimation*

```
{
  0% { transform: translateX(0px); }
  50% { transform: translateX(40px) rotate(-15deg) ;}
  100% { transform: translateX(80px) rotate(15deg); }
}
```

@keyframes *monrebond*

```
{
  0% { transform: translateY(0px) translateX(0px);}
  50% { transform: translateY(8px) translateX(2px);}
  100% { transform: translateY(0px) translateX(0px); }
}
```

On applique ensuite une animation via la propriété "animation" :

```
a:hover { color: blue; transition: color 1s linear; animation : monrebond 1s;}
button:hover { color: blue; animation : monrebond 1s;}

p:hover { background-color: #93db83; /* vert*/; animation: monanimation 2s ; }
```

6.2. Animations "angular" déclenchées sur changement d'état

Une **animation angular** se code comme un **trigger réutilisable** ayant un nom logique (ex : **'changeDivSize'**) et étant généralement structuré par différents états (ex : "smaller" , "normal").

Chaque état sera associé à une association de styles et transformations css (ex : scale , translation , ...) .

On définit également des transitions entre états via la syntaxe suivante

transition('etat1=>état2', animate(...)) .

Par la suite , au sein d'un composant angular utilisant un trigger , une syntaxe spéciale **[@nomTrigger]='nomEtatSelon ValeursDuComposant'** permettra de déclencher une animation angular .

Exemple (avec effet de rétrécissement/grossissement):

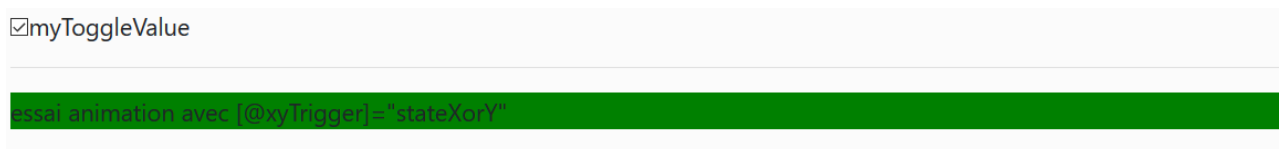
changeDivSizeTrigger.ts

```
import { trigger, state, style, transition, animate } from '@angular/animations';
export const changeDivSizeTrigger =
  trigger('changeDivSize', [
    state('smaller', style({
      backgroundColor: 'lightgreen',
      transform: 'scale(0.9)'
    })),
    state('normal', style({
      backgroundColor: 'green',
      transform: 'scale(1.0)'
    })),
    transition('smaller=>normal', animate('800ms')),
    transition('normal=>smaller', animate('400ms'))
  ]);

/* exemple d'utilisation:
@Component({...,  animations: [ changeDivSizeTrigger , ... ] })
<div [@changeDivSize]="!myToggleValue?'smaller':'normal'">....</div>
<div [@changeDivSize]="taux<20?'smaller':'normal'">essai animation
  avec [@xyTrigger]="stateXorY", taux={{taux}} </div>
*/
```



changement de taille (grossissement) et changement de couleur progressifs .



Exemple avec les alias "enter" et ":leave" :

NB : Au sein des définitions de transition entre état(s) ,

* signifie *n'importe quel état*

void signifie *aucun état connu* et (**void** => *) correspond donc à un **début quelconque**

et (* => **void**) correspond donc à une **fin quelconque**

:enter est un alias/raccourci pour **void** => *

et **:leave** est un alias/raccourci pour * => **void**

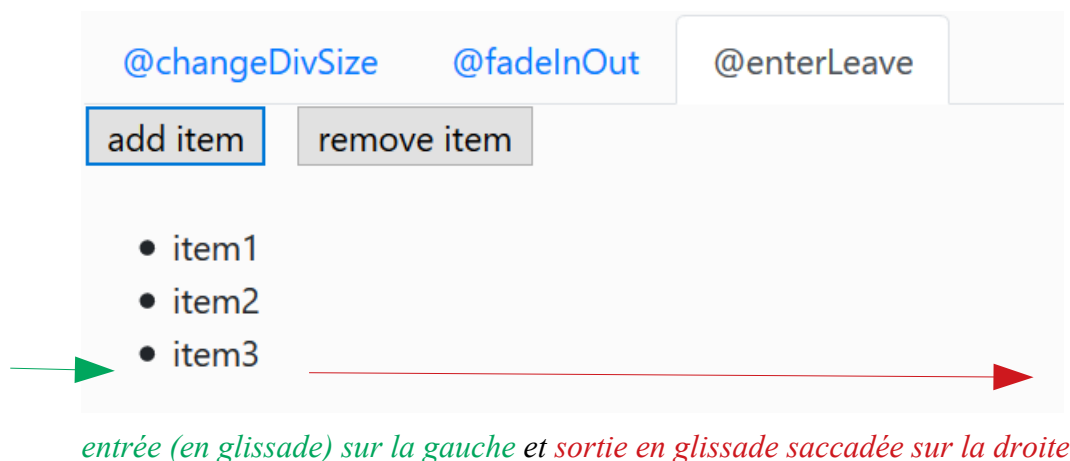
enterLeaveTrigger.ts

```
import { trigger, state, style, transition, animate } from '@angular/animations';

export const enterLeaveTrigger =
trigger('enterLeave', [
  state('flyIn', style({ transform: 'translateX(0)' })),
  transition(':enter', [
    style({ transform: 'translateX(-100%)' }),
    animate('0.5s 300ms ease-in')
  ]),
  transition(':leave', [
    animate('0.3s ease-out', style({ transform: 'translateX(100%)' }))
  ])
]);
```

Exemple d'utilisation :

```
<li *ngFor="let item of myList" [@enterLeave]='flyIn' ">
  {{item}}
</li>
```



Autre exemple avec un seul état anonyme et début et fin indifférenciés :

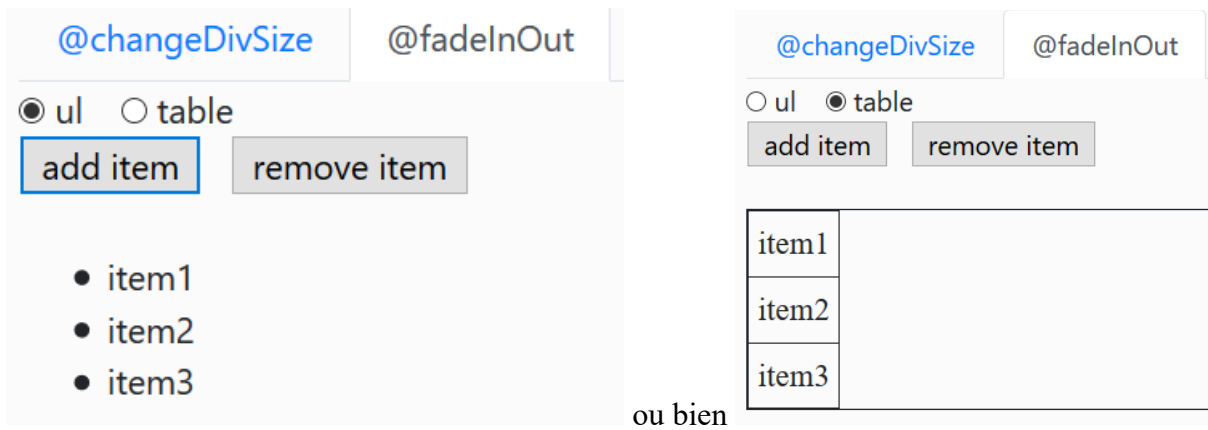
fadeInOutTrigger.ts

```
import { trigger, state, style, transition, animate } from '@angular/animations';

export const fadeInOutTrigger =
  trigger('fadeInOut', [
    state('void', style({
      opacity: 0
    })),
    transition('void <=> *', animate(1000)),
  ]);

/* exemple d'utilisation:
@Component({...,
  animations: [ fadeInOutTrigger , ... ]
})

<li *ngFor="let list of listItem" [@fadeInOut]>
  {{list}}
</li>
*/
```



apparition progressive d'un nouvel élément et disparition progressive d'un élément retiré .

Exemples d'utilisations détaillées :

WithAnimationsComponent.ts

```
import { Component, OnInit } from '@angular/core';
import { changeDivSizeTrigger } from 'src/app/common/animation/changeDivSizeTrigger';
import { enterLeaveTrigger } from 'src/app/common/animation/enterLeaveTrigger';
import { fadeInOutTrigger } from 'src/app/common/animation/fadeInOutTrigger';

@Component({
  selector: 'app-with-animations',
  templateUrl: './with-animations.component.html',
  styleUrls: ['./with-animations.component.scss'],
  animations: [ changeDivSizeTrigger , enterLeaveTrigger, fadeInOutTrigger ]
})
export class WithAnimationsComponent implements OnInit {

  myToggleValue : boolean = false;
  ulOrTable : string ="ul";
  myList : string[] = [ "item1" , "item2"];
  lastItemNumber : number = 2;

  onAddItem(){
    this.lastItemNumber++;
    this.myList.push(`item${this.lastItemNumber}`);
  }
}
```



```
onRemoveItem(){
    if(this.lastItemNumber>0){
        this.lastItemNumber--;
        this.myList.splice(this.lastItemNumber,1);
    }
}

constructor() { }

ngOnInit() {
}
}
```

with-animations-component.html

[illegible]

```

    </tr>
  </table>
</tab>
<tab heading="@enterLeave">
  <input type="button" value="add item" (click)="onAddItem()" /> &nbsp;
  <input type="button" value="remove item" (click)="onRemoveItem()" /><br/>
<br/>
<ul>
  <li *ngFor="let item of myList" [@enterLeave]='flyIn' ">
    {{item}}
  </li>
</ul>
</tab>
</tabset>

```

7. Contrôle des formulaires

7.1. 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**".

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 {  
}
```

7.2. 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 , ...

7.3. 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)]
    });
  }
}
```

7.4. 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 ]],
});
```

7.5. Approche mixte (à expérimenter) :

Au sein de l'approche template driven , tous les contrôles de saisies sont paramétrés du coté HTML (essentiellement selon les syntaxes normalisées de HTML5) .

Au sein de l'approche "model-driven / reactive-form" , tous les contrôles de saisies sont paramétrés du coté "code typescript du composant" .

Bien que "pas officielle" , une approche mixte (un peu de paramétrage HTML pour les cas simples/ordinaires et un peu de "code typescript" pour les cas évolués) est éventuellement envisageable :

Configuration de contrôles de saisies spécifiques:

NB : pour configurer des contrôles de saisies plus sophistiqués/spécifiques , on peut (entre autres solutions) adopter ce style de code (qui deviendra plus limpide après avoir lu le paragraphe suivant).

```
...
import { NgForm, Validators } from "@angular/forms";
...

export class XyzComponent implements OnInit {
  ...
  @ViewChild('formXy') form : NgForm ; //pour accéder/manipuler <form #formXy="ngForm"

  /*
  <form #formXy="ngForm" [hidden]="submitted"
    (mouseenter)="onFormInit()" ....>
  */
  onFormInit(){
    console.log("onFormInit() called")
    //NB: dans ngOnInit() : trop tôt , this.form.controls['...'] undefined
    // l'événement (mouseenter) peut convenir (entre autres solutions).
    this.form.controls['age'].setValidators(
      [Validators.required ,
       Validators.min(0),
       Validators.max(150)]);

    this.form.controls['email'].setValidators(
      [Validators.email]);

  }
}
```


III - Rxjs , services , injections

1. Injection de dépendances "Angular"

1.1. Concepts et terminologie

Provider = Implémentation fournie (ex : Classe "typescript" enregistrée quelquepart) pour un type plus ou moins abstrait (ex : interface , classe abstraite , classe concrète) .

Injecteur = élément technique qui instancie (en mode "singleton") et qui retourne un élément d'un certain type à injecter.

Dépendance = dépendance entre un type (offrant service) et un autre (consommant service) .

Une instance du fournisseur de service doit absolument être initialisée avant de pouvoir être référencée et utilisée par une instance du consommateur de service .

Dans beaucoup de cas il peut y avoir plusieurs niveaux de dépendances et donc une arborescence d'éléments à injecter les uns dans les autres.

Le type de chose à injecter est au sens large appelé "token" . Il s'agit souvent au niveau du code source, d'une classe typescript . A l'exécution , ça deviendra un identifiant de type (string/token/...) .

```
const injector = new Injector([XyService]);
const s1 = injector.get(XyService);
const s2 = injector.get(XyService);
```

// ==> s2 est la même instance que s1.

A un niveau donné (ex : scope d'un @NgModule), un injecteur construit si besoin et retourne toujours la même instance (pour un type donné) .

---> comportement de type "Singleton" et map <token_or_type, instance_construite> en interne au niveau d'un injecteur .

1.2. 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: [ ConfigService, ClientService ],
bootstrap: [ AppComponent ]
})
export class AppModule {
```

Remarque importante :

Lorsqu'un service est déclaré au niveau d'un module (via providers : de @NgModule ou autrement) , il sera géré par un injecteur partagé par tous les composants du module et **il y aura donc une seule instance du service injecté/partagée dans tous les composants de ce module.**

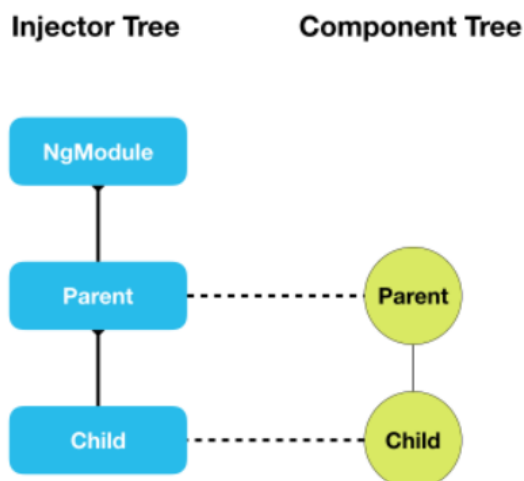
Si un élément potentiellement injectable n'est pas, globalement, enregistré au niveau global (@NgModule) , il pourra éventuellement être déclaré, localement, au niveau des "providers" spécifiques d'un composant (cas rare):

```
@Component({
  selector: 'my-app',
  template: `...`,
  providers: [HeroService]
})
export class AppComponent { ...
}
```

Remarque importante :

Lorsqu'un service est déclaré au niveau d'un composant spécifique (via providers : de @Component) , il sera géré par un injecteur dédié qu'à ce composant et **il y aura donc une instance du service qui sera construite et qui sera dédiée exclusivement à ce composant** (cette instance du service pourra éventuellement exister avec d'autres instances de la même classe de service gérée par d'autres injecteurs de niveau "composant" ou autre) .

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 , ...)



A ne surtout pas faire (involontairement):

Ne pas déclarer un provider de service à la fois au niveau d'un module et à la fois au niveau d'un composant --> car 2 instances différentes et debug très délicat .

providers: [LoggerService , ...]

est un raccourci vis à vis de

providers:[{ **provide:** LoggerService, **useClass:** LoggerService } , ...]

Variantes :

```
{ provide: LoggerService, useClass: LoggerService }  
{ provide: AliasLoggerService, useExisting: LoggerService }  
{ provide: Xyz, useValue: anObject } { provide: Xyz, useValue: new SimpleXy() }  
{  
  deps: [XyzConfig]  
  provide: XyzService,  
  useFactory: (config: XyzConfig) => new XyzService(config)  
}
```

NB : Attention au new direct, ne l'utiliser que sur des classes simples car en général beaucoup de mécanismes techniques (proxies, décorateurs, ...) sont à mettre en oeuvre (à ne pas court-circuiter) .

{ provide: **LoggerService**, **useClass:** LoggerServiceV1 } ne fonctionne qu'avec **LoggerService** en tant que classe (abstraite ou pas) mais pas avec LoggerService en tant qu'interface , car les interfaces n'existent qu'au niveau du code source typescript et pas au runtime et donc le test if (... instance InterfaceName) retourne l'erreur 'InterfaceName' only refers to a type, but is being used as a value here.

Exemple :

.../service/MyStringService.ts

```
export abstract class MyStringService{  
  public abstract withPrefix(s:string):string;  
  public abstract withSuffix(s:string):string;  
}
```

.../service/MyStringServiceV1.ts

```
import { MyStringService } from './MyStringService';  
  
export class MyStringServiceV1 extends MyStringService {  
  public withPrefix(s:string):string {  
    return "***"+s;  
  }  
  public withSuffix(s:string):string {  
    return s+"***";  
  }  
}
```

}

.../service/MyStringServiceV2.ts

```
import { MyStringService } from './MyStringService';

export class MyStringServiceV2 extends MyStringService {
  public withPrefix(s:string):string{
    return "##"+s;
  }
  public withSuffix(s:string):string{
    return s+"##";
  }
}
```

with-injection.commpoent.html

```
<div [style.backgroundColor]="lightgrey">
  {{commentWithPrefix}} <br/> {{commentWithSuffix}}
</div>
```

with-injection.commpoent.ts

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

@Component({....})
export class WithInjectionComponent implements OnInit {
  comment : string = "comment";
  commentWithPrefix : string = null; commentWithSuffix : string = null;

  constructor(private stringService : MyStringService) {
    this.commentWithPrefix = stringService.withPrefix(this.comment);
    this.commentWithSuffix = stringService.withSuffix(this.comment);
  }
}
```

dans **app.module.ts**

```
@NgModule({
  ...
  providers: [
    //{ provide: MyStringService, useClass: MyStringServiceV1 }
    { provide: MyStringService, useClass: MyStringServiceV2 }
  ]
})
export class AppModule { }
```

```
**comment for exp_adv_app  
comment for exp_adv_app**
```

avec V1 et

```
##comment for exp_adv_app  
comment for exp_adv_app##
```

avec v2

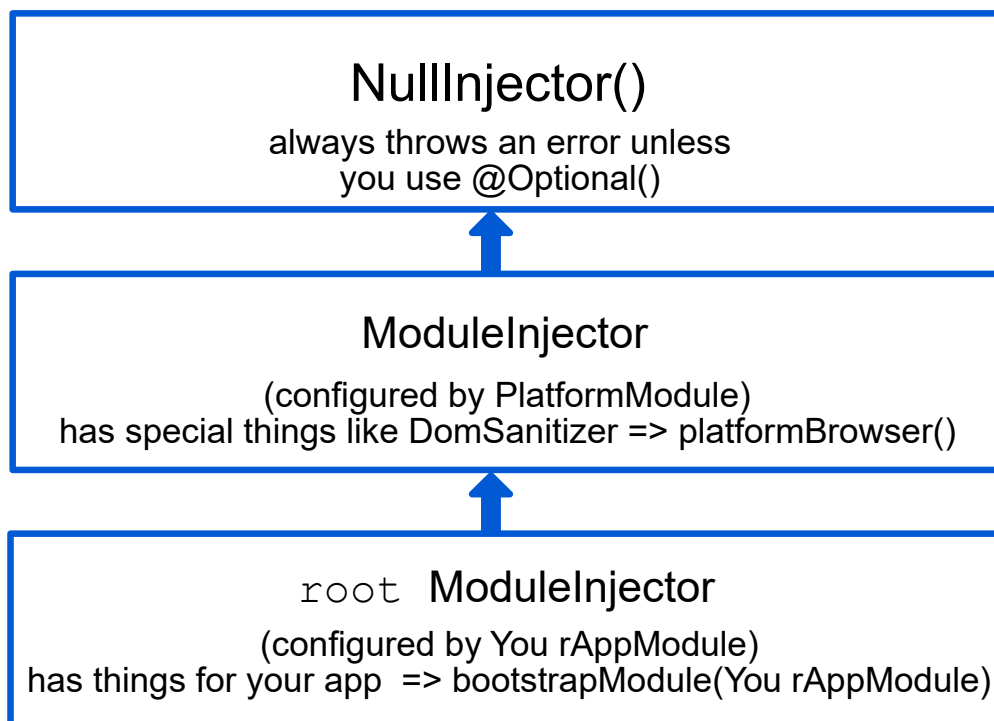
1.3. Nouveauté/évolution à partir de la version 6

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) .



Autres valeurs possibles pour `providedIn` (depuis angular 9):

- **"any"** (pas de singleton pour toute l'application mais des instances distinctes au niveau de chaque "child injector" de niveau module (par exemple injecteur associé à un "lazy loaded (sub-)module"). Cependant différents composants d'un même sous module partageront un accès vers un singleton de niveau "sous-module"
- **"platform"** (potentiellement partagé par plusieurs applications angular chargées depuis même page – cas très rare)

1.4. Injection de dépendance via constructeur "typescript"

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

    // this._heroService (de type HeroService) 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 HeroService). 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)

NB :

- Si paramètre du constructeur pas typé alors @Inject(ClassComponent) permet de préciser le type de composant à injecter
- Un service peut utiliser un autre service (injecté) , etc.
- **placer** (private ou public refService : TypeService) **comme paramètre d'un constructeur** correspond à une **déclaration de dépendance nécessaire**.
C'est le framework "angular" qui va analyser les dépendances à tous les niveaux connus , pour ensuite initialiser et injecter les éléments les uns dans les autres dans le bon ordre.

1.5. Décorateurs @Optional et @Host au niveau d'un constructeur

- Le décorateur **@Optional** de la propriété à injecter dit à Angular de renvoyer null (sans erreur) lorsqu'il ne peut pas trouver la dépendance.
- Le décorateur **@Host** de la propriété arrête la recherche ascendante sur le l'injecteur du composant hôte. Le composant hôte est généralement le composant demandant la dépendance et quelquefois le composant parent
--> pas de recherche via d'autres injecteurs (de niveau module par exemple)

1.6. Décorateurs @Self et @SkipSelf au niveau d'un constructeur

- Le décorateur **@Self** (très proche de @Host) de la propriété n'effectue une recherche qu'au

niveau de l'injecteur associé au composant courant.

- Le décorateur **@SkipSelf** de la propriété effectue inversement une recherche avec les injecteurs qui ne sont pas associés au composant courant (mais ceux du niveau module par exemple) .

1.7. Ancien "ReflectiveInjector" (deprecated puis V5)

```
import { Injector, ReflectiveInjector } from '@angular/core';
...
var injector : Injector = ReflectiveInjector.resolveAndCreate([ HeroService ]);
                                // ou [ HeroService , S2, S3, ... ]
this._heroService = injector.get(HeroService);
ou bien (selon contexte) :
var hService = injector.get(HeroService);
```

NB1 : Cette ancienne technologie d'injection de dépendance a été inventée à l'époque du début d'angular 2 (où l'on pensait pouvoir utiliser au choix le langage javascript , typescript ou dart) .

NB2 : Pour savoir qui dépend de qui (ex : Service en utilisant un autre) , il fallait utiliser la décoration @Inject au niveau d'un constructeur :

```
class B {}
class A { constructor(@Inject(B) b) {} }
//or class A { constructor(private b :B) {} }

const i = ReflectiveInjector.resolveAndCreate([A, B]);
const a = i.get(A);
...
```

depuis v5 : ReflexiveInjector deprecated , StaticInjector instead

1.8. Présentation de StaticInjector (depuis angular 5)

Plus absolument besoin de @Inject(X) dans les constructeurs pour préciser les interdépendances d'un service à l'autre , mais besoin d'expliciter (en contrepartie) , les interdépendances au moment de l'activation dynamique d'un injecteur :

```
class B {}
class A { constructor(b) {} }
const i = Injector.create([{provide: A, useClass: A, deps: [B]}]);
const a = i.get(A);
```

Principal avantage du nouvel "StaticInjector" : plus rapide et nécessitant moins de génération de code .

1.9. inject anything : InjectionToken

Lorsque l'on injecte des valeurs élémentaires (chaînes de caractères , dates , ...) , le type de chose à injecter est bien géré au runtime en tant que token javaScript (identifiant unique) .

src/app/common/token/global-tokens.ts

```
import { InjectionToken } from '@angular/core';
export const TITLE = new InjectionToken('title');
```

src/app/app.module.ts

```
import { TITLE } from '../common/token/global-tokens';
...
providers: [
  { provide: TITLE,      useValue: 'Hero of the Month' },
  ... ]
```

et utilisations au sein des composants avec (**@Inject(TITLE) title :string**) dans **constructeurs** pour *injections* .

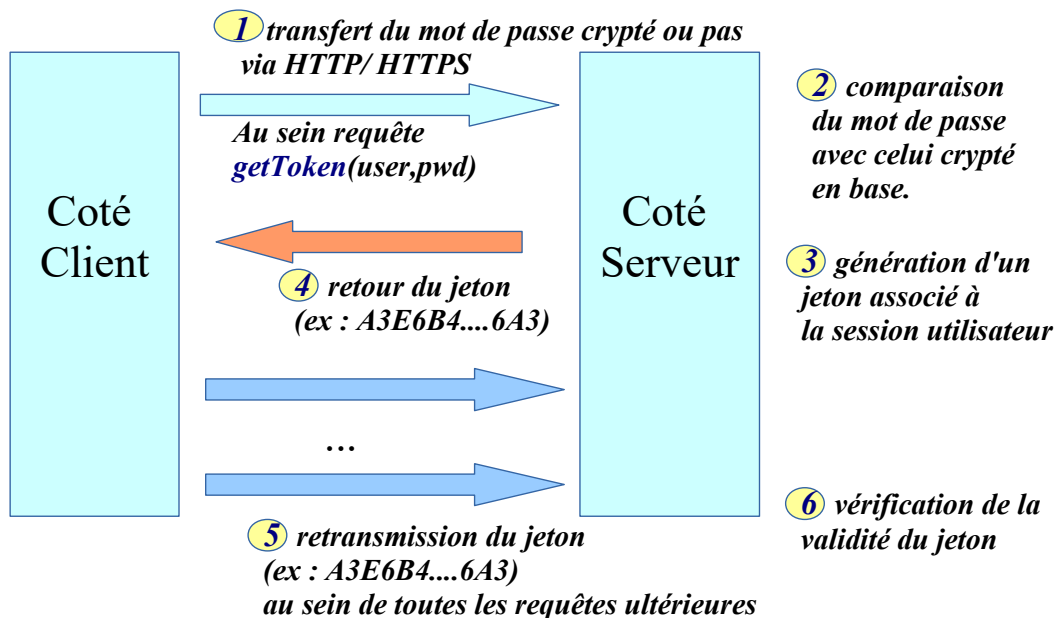
2. Aspects plus ou moins avancés de RxJs et des services angular

....

3. Jetons "JWT" et intercepteurs de sécurité

3.1. Tokens : principes et transmission via HTTP

Jeton ("token") d'authentification valide le temps d'une session utilisateur



Plusieurs sortes de jetons/tokens

Il existe plusieurs sortes de jetons (normalisés ou pas).

Dans le cas le plus simple, un **jeton** est **généré aléatoirement** (ex : **uuid** ou ...) et sa **validation** consiste essentiellement à **vérifier son existence** en tentant de le récupérer quelque part (en mémoire ou en base) et éventuellement à vérifier une date et heure d'expiration.

JWT (Json Web Token) est un **format particulier de jeton** qui **comporte 3 parties** (une entête technique , un paquet d'informations en clair (ex : username , email , expiration, ...) au format JSON et une signature qui ne peut être vérifiée qu'avec la clef secrète de l'émetteur du jeton.

Bearer token (jeton au porteur) et transmission

Le champ **Authorization**: normalisé d'une entête d'une requête HTTP peut comporter une valeur de type **Basic ...** ou bien **Bearer ...**

Le terme anglais "**Bearer**" signifiant "**au porteur**" en français indique que la simple possession d'un jeton valide par une application cliente devrait normalement , après transmission HTTP, permettre au serveur d'autoriser le traitement d'une requête (après vérification de l'existence du jeton véhiculé parmi l'ensemble de ceux préalablement générés et pas encore expirés).

NB: Les "bearer token" sont utilisés par le protocole "O2Auth" mais peuvent également être utilisés de façon simple sans "O2Auth" dans le cadre d'une authentification "sans tierce partie" pour API REST.

NB2 : un "bearer token" peut éventuellement être au format "JWT" mais ne l'est pas toujours (voir rarement) en fonction du contexte.

3.2. JWT (Json Web Token)



Structure jeton "JWT / Json Web Token"



Exemple:

`eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJkb3B0YWwuc29tIiwiaXNjaXNjaXNDI2NDIwODAwLCJodHRwOi8vdG9wdGFsLmNvbS9qd3RfY2xhaW1zL2l2ZXZkbWluIj0cnVILCJjb21wYW55IjoibG9wdGFsIiwiaXNjaXNDI29tZSI6dHJ1ZX0.yRQYnWzskCZUxPwaQupWkiUzKELZ49eM7oWxAQK_ZXw`

NB: "iss" signifie "issuer" (émetteur) , "iat" : issue at time
 "exp" correspond à "date/heure expiration" . Le reste du "payload" est libre (au cas par cas) (ex : "company" et/ou "email" , ...)

3.3. intercepteurs "angular" :

Disponible à partir de la version 4.3 , les intercepteurs sont surtout utiles pour renvoyer un jeton d'authentification :

```
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 (req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    let token = localStorage.getItem('token');
    const authReq = req.clone({
      headers: req.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 './myauth.interceptor';

@NgModule({
  declarations: [ AppComponent ],
  imports: [
    BrowserModule, HttpClientModule ],
  providers: [{
    provide: HTTP_INTERCEPTORS,
    useClass: MyAuthInterceptor,
    multi: true
  }],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

3.4. détails sur jeton jwt avec angular

via une dépendance supplémentaire dans *package.json*

```
...
"@auth0/angular-jwt": "^3.0.0",
```

app.module.ts

```

import { JwtModule } from "@auth0/angular-jwt";

...

//this sub function will be used by automatic internal
//interceptor of JwtModule (@auth0/angular-jwt)
export function myTokenGetter() {
  return /*localStorage.* sessionStorage.getItem("authToken");
}
//NB: no need of MyAuthInterceptor if JwtModule has already
//his internal interceptor

@NgModule({
  ...
  imports: [
    ... ,
    JwtModule.forRoot({
      config: { tokenGetter: myTokenGetter }
    })
  ],
  ...
})
export class AppModule { }

```

Exemple partiel de service d'authentification stockant un jeton jwt en sessionStorage :

```

import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs';
import { AuthRequest } from '../data/auth-request';
import { AuthResponse } from '../data/auth-response';
import { tap } from 'rxjs/operators';

@Injectable({
  providedIn: 'root'
})
export class AuthService {
  public lastAuthResponse : AuthResponse = null; //last succeeding response only

  constructor(private _http : HttpClient) { }
  private _headers = new HttpHeaders({'Content-Type': 'application/json'});
  private _authBaseUrl = "../login-api/public/auth" ; //avec ng serve --proxy-config proxy.conf.json

  postAuth(auth : AuthRequest):Observable<AuthResponse>{
    return this._http.post<AuthResponse>(this._authBaseUrl ,auth,{headers: this._headers} )
    .pipe(
      //tap( other async task without transforming result)
    )
  }
}

```

```

    tap( (authResponse) => { this.storeAuthResponseAndToken(authResponse); })
  );
}

private storeAuthResponseAndToken(authResponse : AuthResponse){
  if(authResponse.status){
    this.lastAuthResponse = authResponse;
    sessionStorage.setItem("authToken",authResponse.token);
  }
  else{
    this.lastAuthResponse = null;
    sessionStorage.setItem("authToken",null);
  }
}
}
}

```

quelquepart :

```

import { JwtHelperService } from '@auth0/angular-jwt';
...
constructor(....., public jwtHelper: JwtHelperService) {
  }

logJwtToken(authToken:string=null){
  let token = authToken;
  if(token==null)
    token = sessionStorage.getItem("authToken");
  console.log("isTokenExpired="+this.jwtHelper.isTokenExpired(token)); // true or false
  console.log("tokenExpirationDate="+this.jwtHelper.getTokenExpirationDate(token)); // date
  console.log("jwt payload="+this.jwtHelper.decodeToken(token)); // token
  }
...

```

IV - modules & routing (avancés)

1. Rappels sur les bases du routing angular

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: 'identification', component: IdentificationComponent },
  { path: 'clientIdentification/:clientId', component: IdentificationComponent }
];
@NgModule({
  imports: [ RouterModule.forRoot(routes) ],
  exports: [ RouterModule ]
})
export class AppRoutingModule {}
```

NB : le fichier src/app/**app-routing.module.ts** est souvent créé dès le début avec par exemple la nouvelle option **ng new my-app --routing** (depuis la version 8.1)

et d'un point de vue routage, dans le fichier principal du module **app.module.ts** il ne reste plus que :

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

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

1.1. Déclenchement d'une navigation avec ou sans paramètre

```
// dans app-routing.module.ts
const routes: Routes = [
  { path: 'xx',    component: XxComponent }
  { path: 'yy/:yyId',    component: YyComponent }
];
```

Solution 1 (par méthode événementielle) :

```
import {Router} from '@angular/router';
... <button (click)="onNavigate()" > vers yy </button>
et
export class XxComponent {
  numYy : number;
  constructor(private _router: Router){
  }
  onNavigate():void {
    let link = ['/yy', this.numYy]; //ou link = ['/xx'] ; sans paramètre
    this._router.navigate( link );
    // ou bien this._router.navigateByUrl(`/yy/${this.numYy}`);
    //avec quote inverse `...` !!!
  }
}
```

Solution 2 (via paramètre de la directive routerLink) :

```
<a [routerLink]="['/yy', numYy ]" ...> vers yy </a>
```

```
<a [routerLink]="['/xx' ]" ...> vers xx </a>
```

1.2. Récupération du paramètre accompagnant la navigation :

```
import {Component , OnInit} from '@angular/core';
import { ActivatedRoute, Params} from '@angular/router';
import { Location }    from '@angular/common';
...
export class YyComponent implements OnInit{
```

```

message : string = "...";
yId: number ;
y : Yy;
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
                                     // (fichier app-routing.module.js)

      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 */ }
}

```

Remarque très importante (pour comprendre et ne pas devenir fou) :

- (Cas "a") Suite à la série de navigation suivante :
aaa/yyy , **aaa/xxx** , **aaa/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 :
bbb/detail_produit/1 , **bbb/detail_produit/2** , **bbb/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 simplifier 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']);
    ...
  }
}
```

2. fallback route (**) in last position

Dans certains cas , d'une version à l'autre d'une application , certaines routes (urls) peuvent changer. Si un utilisateur a mémorisé une ancienne url précise (via un favoris/bookmark) et qu'il essaie ensuite d'activer directement cette ancienne route maintenant invalide dans la nouvelle version de l'application , il va recevoir un "404 not found" pas très "user friendly" .

Dans le tableau des routes configurées, on peut placer en dernière position une route spéciale :

```
{ path: "**" , component : PageNotFoundComponent or HomeComponent }
```

Cette route permettra d'effectuer une redirection vers un composant personnalisé en cas d'erreur d'URL

3. Sous niveau de routage (children)

Une application de grande taille peut comporter plusieurs niveaux de composants et sous composants.

Un `<router-outlet>` dans un composant main 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 .

Exemple d'url avec sous niveaux :

xxx/

yyy/1

yyy/1/details_prod/3

yyy/1/details_prod/4

zzz/

yyy sera intitulé *subspace-main* (sous espace principal) dans l'exemple qui suit .

Exemple de configuration de routes avec sous niveau:

```

...
const routes: Routes = [
  { path: 'welcome', component: WelcomeComponent },
  { path: '', redirectTo: '/welcome', pathMatch: 'full' },
  { path: 'login', component: LoginComponent },
  { path: 'subspace-main/:id', component: SubspaceMainComponent
children: [
  { path: 'prodList', component: ProdListComponent },
  { path: 'details/:num', component: ProdDetailComponent },
  { path: '', redirectTo: 'prodList', pathMatch: 'prefix' }
]
}
];

```

4. Routes conditionnées (via gardiens)

4.1. Gardien basique

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 :

```

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]
}, ...
```

4.2. 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.currentSecureMode===false)
      return true;
    else{
      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('/ngr/not-authorized');
        //or this.router.createUrlTree(['/ngr/not-authorized']);
      }
    }
  }
}
```

4.3. Eventuelles données accompagnant un gardien

dans `app-routing.module.ts`

```
...
{
```

```

path: 'super-user-stuff',
component: SuperUserStuffComponent,
canActivate: RoleGuard,
data: {roles: ['SuperAdmin', ...]}
}

```

dans RoleGuard.ts :

```

...
canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot)
: Observable<boolean> | Promise<boolean> | boolean | UrlTree {
  let roles = route.data.roles as Array<string>;
  ...
}

```

5. router-outlet annexe

A un niveau donné (principal ou "child") , il peut y avoir des <router-outlet> annexes (avec des noms précis) .

Ceci permettra (suite à une navigation) , de switcher à la fois :

- le composant principal
- un ou plusieurs(x) panneau(x) annexe(s) (exemple : menu latéral ou autre).

Exemple (dans app.component.html):

```

<div class="row">
  <div class="col-sm-3">
    <router-outlet name="aside"></router-outlet> <!-- secondary outlet with name -->
  </div>
  <div class="col-sm-9">
    <router-outlet></router-outlet>
    <!-- main outlet without specified name , default_name="primary" -->
  </div>
</div>

```

Dans **app-routing.module.ts**

```

const routes: Routes = [

{ path: 'asidePanelA', component: AsidePanelAComponent , outlet : 'aside'},
{ path: 'asidePanelB', component: AsidePanelBComponent , outlet : 'aside'},

{ path: 'ngr/welcome', component: WelcomeComponent },
{ path: '', redirectTo: '/ngr/welcome(aside:asidePanelA)', pathMatch: 'full'},
...

```

URL complète pour naviguer (dans barre d'url):

`http://localhost:4200/ngr/welcome(aside:asidePanelA)`

`http://localhost:4200/ngr/welcome(aside:asidePanelB)`

Navigation via lien angular :

```
<a [routerLink]="[ { outlets: { primary : ['/ngr/welcome'], aside : ['asidePanelB'] } } ]">
welcome</a>
```

Navigation via code typescript :

```
router.navigate([ { outlets: { primary : ['/ngr/welcome'], aside : ['asidePanelB'] } } ])
```

(à éventuellement adapter) .

6. Resolver et autres gardiens

<i>interfaces de gardiens</i>	<i>rôles/utilités</i>
CanActivate	activation possible d'une route ?
CanActivateChildren	activation possible des "sub-routes" ?
CanDeactivate	pour forcer à ne pas bouger tant que modifications pas sauvegardées
Resolve	permet de différer l'activation d'une route tant que certaine données ne sont pas chargées
CanLoad	gardien pour modules asynchrones

6.1. Resolver

Un "resolver" est une **abstraction** et correspondant à un **petit objet technique** dont le rôle est de **préparer un chargement de données** avant de pourvoir les afficher au sein d'un composant.

Sans resolver :

Via une navigation angular , on arrive sur un composant qui affiche une liste d'élément vide au départ . La méthode `ngOnInit()` du composant ensuite à un service de récupérer certaines données et celles-ci seront affichées en différé (après téléchargement et éventuelles consolidations et autres post-traitements) .

Avec resolver :

La route est associée à un "resolver" qui va :

- déléguer le chargement des données à un service (via HttpClient)
- effectuer éventuellement certains post-traitements (restructuration des données , appels secondaires , ...)
- bloquer l'activation de la route tant que tout n'est pas prêt .

Ainsi , lorsque que le "resolver" aura fini son travail, les données récupérées/préparées seront alors associées à la route activée et le composant associé à cette route aura juste besoin de les afficher.

Exemple simple :

```
import { Resolve, ActivatedRouteSnapshot } from '@angular/router';
...
@Injectable({
  providedIn: 'root'
})
export class NewsResolver implements Resolve<Publication[]> {
  constructor(private _publicationService : PublicationService) {}
  resolve(): Observable<Publication[]> {
    console.log("via NewsResolver ...") ;
    return this._publicationService.getListePublicationObservable();
  }
}
```

dans *app-routing.module.ts* :

```
...
{ path: 'ngr/news', component: NewsComponent , resolve: { publications: NewsResolver } } , ...
```

dans *news.components.ts* :

```
...
constructor(private route: ActivatedRoute) {}

ngOnInit() {
  this.route.data.subscribe(
    ( data: { publications: Publication[] } ) => {
      this.tabPublicationNews=data.publications; //...
    } ,
    (error) => { console.log( " error : " + error ) ; }
  );
}
```

...

Limitation des resolvers :

Un "resolver" va permettre d'attendre le téléchargement des données JSON (depuis une Api REST) pour celles-ci soient bien prêtes lorsqu'un composant en aura besoin.

Ceci dit , si ces données JSON comportent des références (url relatives) vers de gros fichiers "images" ou "vidéos" , ces fichiers ne seront traités (téléchargés et affichés progressivement) qu'au niveau du composant angular (après l'action du resolver et sans aucun lien avec celui-ci) .

7. Lazy loading

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).**

Une application angular bien structurée comportera généralement :

- **un module applicatif principal** (`src/app/app.module.ts`)
avec `RouterModule.forRoot(routes)` dans `src/app/app-routing.module.ts`
- **plusieurs sous modules applicatifs** (quelquefois dénommés "*features modules*" ou encore "*domain modules*")
(ex : `src/app/mylazyN/mylazyN.module.ts`)
avec `RouterModule.forChild(routes)` dans `src/app/mylazyN/mylazyN-routing.module.ts`
- **d'éventuels modules de composants réutilisables** (quelquefois dénommés "*widget modules*" ou encore "*shared modules*")

(ex : `src/my-util/...`)

7.1. Exemple de (sous-)module en mode lazy :

En mode "non automatisé" au sein des anciennes versions d'angular :

```
ng g module myLazy
cd src/app/my-lazy
ng g component xx
ng g component yy
```

my-lazy.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
...
import { Routes, RouterModule } from '@angular/router';
const childRoutes: Routes = [
  { path: 'xx', component: XxComponent },
  { path: 'yy', component: YyComponent },
  { path: '', redirectTo: 'xx', pathMatch: 'prefix' }
];
@NgModule({
  imports: [ CommonModule,
    RouterModule.forChild(childRoutes)
  ],
  declarations: [ XxComponent, YyComponent ]
})
export class MyLazyModule { }
```

Et par exemple pour naviguer relativement de xx vers yy (et vice versa) :

```
<a [routerLink]="'[..../yy']"> vers yy </a>
```


En mode "automatisé" au sein des versions récentes d'angular :

```
ng g module myLazy2 --route mylazy2 --module app.module
```

automatise toutes les opérations suivantes :

```
CREATE src/app/my-lazy2/my-lazy2-routing.module.ts (349 bytes)
CREATE src/app/my-lazy2/my-lazy2.module.ts (358 bytes)
CREATE src/app/my-lazy2/my-lazy2.component.html (23 bytes)
CREATE src/app/my-lazy2/my-lazy2.component.spec.ts (636 bytes)
CREATE src/app/my-lazy2/my-lazy2.component.ts (283 bytes)
CREATE src/app/my-lazy2/my-lazy2.component.scss (0 bytes)
UPDATE src/app/app-routing.module.ts (1017 bytes)
```

avec

```
...
@NgModule({
  declarations: [MyLazy2Component],
  imports: [ CommonModule, MyLazy2RoutingModule ]
})
export class MyLazy2Module { }
```

dans *my-lazy2.module.ts*

avec

```
...
const routes: Routes = [
  { path: "", component: MyLazy2Component }
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class MyLazy2RoutingModule { }
```

dans *my-lazy2-routing.module.ts*

et avec l'ajout de

```
{ path: 'mylazy2',
  loadChildren: () => import('./my-lazy2/my-lazy2.module').then(m => m.MyLazy2Module) }
```

dans les routes de *src/app/app-routing.module.ts*

7.2. Référencement d'un sous module en mode lazy

Référencement du ou des sous module(s) "lazy" depuis le module principal de l'application :

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
const routes: Routes = [
  { path: 'mylazy1',
    loadChildren: () => import('./my-lazy/my-lazy.module').then(m => m.MyLazyModule) ,
  { path: 'mylazy2',
    loadChildren: () => import('./my-lazy2/my-lazy2.module').then(m => m.MyLazy2Module)}
];
@NgModule({
  imports: [RouterModule.forRoot(routes)], exports: [RouterModule]
})
export class AppRoutingModuleModule { }
```

Point clef:

pas de imports : [...] explicite/systématique de sous module mais chemin **loadChildren** exprimé de la façon suivante :

au sein d'anciennes versions de angular :

```
{ path: 'mylazy1', loadChildren: './my-lazy/my-lazy.module#MyLazyModule' },
```

avec un suffixe en **"#LazySubModuleName"** .

au sein des versions récentes d'angular :

```
{ path: 'mylazy1',
  loadChildren: () => import('./my-lazy/my-lazy.module').then(m => m.MyLazyModule)}
```

Dans un des composants du module principal :

```
<a [routerLink]="['/mylazy1']"> lazy loading part 1 ...</a> <br/>
```

Pour visualiser l'effet lazy-loading on peut éventuellement surveiller le **chargement différé** (au moment de la navigation) de ***-chunk.js** dans la partie **"network"** de **"Chrome Dev Tools / F12"**.

7.3. Eventuel "pré-chargements en tâche de fond" (background preloading)

```
import { PreloadAllModules } from '@angular/router';
...
RouterModule.forRoot(
  appRoutes,
  {
    preloadingStrategy: PreloadAllModules
  }
)
```

dans **app-routing.module.ts**

7.4. Organisation des services et modules

src/app/common/service/global.service.ts (partagé pour toute l'appli)
....

src/app/lazy1/common/service/for.lazy1.service.ts (partagé entre composants
de ce sous module 'lazy1')
....

src/app/lazy2/common/service/for.lazy2.service.ts (partagé entre composants
de ce sous module 'lazy2')
...

NB:

- si **GlobalService** est en { **providedIn** : 'root' } alors une seule instance partagée par toute l'application (singleton global)
- si **GlobalService** est en { **providedIn** : 'any' } alors une instance partagée par tous les composants d'un même (sous-)module . Et instances différentes pour module principal *app.module* et sous modules *lazy1.module* et autres

V - aot, ivy, internationalisation

1. JIT vs AOT (Ahead-Of-Time) pour angular 4,6,8

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 peut 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) sont révélées lors d'un lancement de ng serve --aot

ou bien **ng build --prod** . C'est alors le moment de peaufiner encore un peu le code de l'application.

2. ivy (à partir de angular 9)

En version "preview" au sein de angular8 et maintenant bien intégré dans angular 9 et 10 , **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

3. internationalisation (i18n)

$i18n = i + 18 \text{ caractères} + n$

3.1. paramétrage au niveau d'un template html

```
<h1 i18n="main header|Friendly welcoming message">Welcome!</h1>
```

```
<h1 i18n="main header|Friendly welcoming message@@welcome">Welcome!</h1>
```

i18n (attribut sans valeur, sans indication) demande simplement à effectuer une traduction du texte de la balise.

i18n="texte_description" demande une traduction en fonction d'une description

i18n="meaning | description" demande une traduction en fonction d'une signification et d'une description.

i18n="@@custom_translation_id" ou

i18n="description@@custom_translation_id" ou

i18n="meaning | description@@custom_translation_id" permet en plus de contrôler la valeur de l'id qui sera généré dans le fichier de traduction en lui affectant une valeur parlante/significative et non changeante (pas partiellement aléatoire).

Selon les paramétrages des fichiers de traduction, toutes des traductions associées à la même signification auront par défaut la même valeur.

Souvent utilisé au niveau d'une véritable balise html (ex : <p>, <h3>, ...) l'attribut i18n peut également être utilisé sur <ng-container> ce qui aura pour effet de ne générer que le texte traduit (sans balise html englobante)

Exemple: <ng-container i18n>I don't output any element</ng-container>

Application d'une traduction au niveau d'un attribut :

```
<img [src]="logo" i18n-title title="Angular logo" />
```

Via **i18n-attributeName**, la valeur de l'attribut est alors d'abord traduite avant d'être utilisée comme une propriété dans l'arbre DOM.

3.2. Fichiers de traduction

Format par défaut : [XML Localization Interchange File Format](#) (XLIFF, version 1.2)

Autres formats supportés : XLIFF 2 et XMB (Xml Message Bundle).

Un fichier xliif a généralement l'extension .xlf

```
ng xi18n --out-file src/locale/messages.xlf
```

```
ng xi18n --i18n-locale fr --out-file src/locale/messages.fr.xlf
```

//attention, le contenu du fichier messages.fr.xlf reste à traduire manuellement ou automatiquement !!!

options de la commande : --format=xlf ou --format=xlf2 ou --format=xmb

messages.de.xlf (généré avec `il8n` sans valeur)

```
<?xml version="1.0" encoding="UTF-8" ?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
  <file source-language="de" datatype="plaintext" original="ng2.template">
    <body>
      <trans-unit id="e27c4e1996f81811e5f18c03bfc30af4db0650cf" datatype="html">
        <source>welcome</source>
        <context-group purpose="location">
          <context context-type="sourcefile">
            src/app/advanced/with-traduction/with-traduction.component.html</context>
          <context context-type="linenumber">2</context>
        </context-group>
      </trans-unit>
    </body>
  </file>
</xliff>
```

--> baser des paramétrage de traduction uniquement sur des numéros de ligne qui peuvent changer, ce n'est pas très viable.

```
<h3 il8n="welcome|my simple welcome message">welcome</h3>

<button il8n="ok">ok</button> <br/>
```

==>

```
...
<trans-unit id="50c1a51a1c7bd808f40a497ca24543b5b6612169" datatype="html">
  <source>welcome</source>
  <target>bienvenu</target> <!-- a ajouter (pas trop tot) -->
  <context-group purpose="location">... </context-group>
  <note priority="1" from="description">my simple welcome message</note>
  <note priority="1" from="meaning">welcome</note>
</trans-unit>
<trans-unit id="e644f6231bef1a54e64391b9c6856d3b92822a8f" datatype="html">
  <source>ok</source>
  <context-group purpose="location">....</context-group>
  <note priority="1" from="description">ok</note>
</trans-unit>
...
```

Attention, un ajout manuel de **<target>traduction</target>** est écrasé en re-générant le fichier via la commande **ng xi18n** et l'option **--out-file**.

NB : depuis **angular 9**, l'utilisation de l'internationalisation nécessite de lancer la commande suivante dans le projet angular :

```
ng add @angular/localize
```

3.3. Utilitaire pour automatiser les traductions

Etant donné que le format xliif est une norme , il existe tout un tas de programmes utilitaires (éditeurs , traducteurs,) permettant de travailler sur des fichiers **.xlf**

npm install -g xlf-translator

Installé en mode global via npm, l'utilitaire xlf-translator permet de générer automatique certaines traduction via "google-traduction" .

La configuration s'effectue via le fichier

translator.config.json (à placer à la racine du projet)

```
{
  "source": "/src/locale/messages.xlf",
  "outputPath": "/src/locale",
  "fromLanguage": "en",
  "toLanguage": [ "it", "en", "de", "fr", "es" ]
}
```

```
ng xli8n --out-file src/locale/messages.xlf
```

translate

==> génère src/locale/**translations/csv/messages.fr.csv** ,

et src/locale/**messages/messages.fr.xlf** , messages.it.xlf ,

avec des traductions récupérées via google-traduction .

NB : en modifiant manuellement un fichier **.csv** et en relançant la commande **translate** on peut très simplement personnaliser les traductions et re-générer un fichier tel que **messages.fr.xlf** .

3.4. Configuration des traductions au niveau du projet angular

Dans **angular.json**

```
{
  "projects": {
    "myapp": {
      "architect": {
        "build": {
          "configurations": {
            "fr": {
              "aot": true,
              "outputPath": "dist/myapp-fr/",
              "i18nFile": "src/locale/messages/messages.fr.xlf",
              "i18nFormat": "xlf",
              "i18nLocale": "fr",
              "i18nMissingTranslation": "error"
            },
            "de": {
              "aot": true,
              "outputPath": "dist/myapp-de/",
              "i18nFile": "src/locale/messages/messages.de.xlf",
              "i18nFormat": "xlf",
              "i18nLocale": "de"
            }
          },
          "production": {
            "baseHref": "/fr/"
          }
        }
      }
    }
  }
}
```

- Sans "i18nMissingTranslation": "error" ou "ignore", des "warning" par défaut.
- Pour la production, un éventuel paramétrage supplémentaire de type "baseHref": "/fr/" peut quelquefois être utile (par exemple pour switcher de langue via des routes commençant par "/en/..." ou "/fr/..." ou autre).

et plus bas dans **angular.json** :

```
{
  "projects": {
    "myapp": {
      "architect": {
        "serve": {
          "builder": "@angular-devkit/build-angular:dev-server",
          "options": {
            "browserTarget": "myapp:build"
          }
        },
        "configurations": {
          "production": {
            "browserTarget": "myapp:build:production"
          },
          "fr": {
            "browserTarget": "myapp:build:fr"
          },
          "de": {
            "browserTarget": "myapp:build:de"
          }
        }
      }
    }
  }
}
```

dans **package.json**

```
...
"scripts": {
  "ng": "ng",
  "start": "ng serve",
  "start:fr": "ng serve --configuration=fr",
  "start:de": "ng serve --configuration=de",
  "build": "ng build",
  "test": "ng test",
  "lint": "ng lint",
  "build:fr": "ng build --configuration=fr",
  "build:de": "ng build --configuration=de",
  "e2e": "ng e2e"
},
...
```

Ceci permet de lancer des commandes de ce type

```
npm run start:fr -- --port=4201
```

```
npm run start:de -- --port=4202
```

http://localhost:4200 --> sans traduction

http://localhost:4201 --> traductions françaises

http://localhost:4202 --> traductions allemandes

3.5. aspects avancés / internationalisation

Avec ou sans "s" (ou bien afficher "un" ou "plusieurs")

--> pluralization :

```
<span i18n>Updated {minutes, plural, =0 {just now} =1 {one minute ago} other
{{{minutes}}} minutes ago}</span>
```

avec une interprétation spéciale dite "ICU" sur la pluralité de la variable minutes (sorte de switch/case sur la valeur de minutes)

avec dans le fichier de traduction :

```
<trans-unit id="5a134dee893586d02bffc9611056b9cadf9abfad" datatype="html">
  <source>{VAR_PLURAL, plural, =0 {just now} =1 {one minute ago} other {<x
id="INTERPOLATION" equiv-text="{{minutes}}"/> minutes ago} }</source>
  <target>{VAR_PLURAL, plural, =0 {à l'instant} =1 {il y a une minute} other
{il y a <x id="INTERPOLATION" equiv-text="{{minutes}}"/> minutes} }</target>
</trans-unit>
```

Alternatives :

```
<span i18n>The author is {gender, select, male {male} female {female} other {other}}</span>
```

avec dans le fichier de traduction :

```
<trans-unit id="eff74b75ab7364b6fa888f1cbfae901aaaf02295" datatype="html">
  <source>{VAR_SELECT, select, male {male} female {female} other {other}}
</source>
  <target>{VAR_SELECT, select, male {un homme} female {une femme} other {autre}}
</target>
</trans-unit>
```

3.6. Choix d'une langue

Principe :

- générer plusieurs versions "fr" , "en" , "de" de l'application angular avec "baseHref": "/"fr/" ou "baseHref": "/"en/" , ...
- dans un menu principal de l'application (à base de liste déroulante par exemple) , faire en sorte que l'on puisse déclencher des liens hypertextes en "/"en/" ou "/"fr/" ,

Exemple :

```
<ul>
  <li *ngFor="let language of languageList">
    <a href="/{{language.code}}/">
      {{language.label}}
    </a>
  </li>
</ul>
```

Pour que ça fonctionne, il faut bien paramétrer une configuration cohérente au niveau du serveur HTTP (ex : nginx) vis à vis des répertoires et des "baseHref" .

VI - Tests unitaires et "end-to-end" 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 peut utiliser la technologie spécifique "**protractor**" qui permet d'intégrer ensemble "**selenium**" et "**angular**" à travers des tests faciles à lancer.

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 / e2e"

Les tests e2e de angular sont exécutés via la technologie "**protractor**".
 Cette technologie (spécifique à angular) est en interne basée sur "**selenium**" et "**jasmine**".

e2e/app.po.ts

```
import { browser, element, by } from 'protractor';

export class ENgPage {
  navigateTo() {
    return browser.get('/');
  }

  getParagraphText() {
    //return element(by.css('app-root h1')).getText();
    return element(by.css('app-root #mainHeader h3')).getText();
  }
}
```

La méthode `getParagraphText()` de l'exemple ci-dessus permet d'accéder à l'élément suivant du template html du composant principal de l'application :

```
...
@Component({
  selector: 'app-root',
  template: `
    <header id="mainHeader" role="banner">
      <h3>Minibank App</h3>
    </header>
  `,
  ...
```

e2e/app-e2e-spec.ts

```
import { ENgPage } from './app.po';

describe('e-ng App', function() {
  let page: ENgPage;

  beforeEach(() => {
    page = new ENgPage();
  });

  it('should display title Minibank App', () => {
    page.navigateTo();
    expect(page.getParagraphText()).toEqual('Minibank App');
  });
});
```

Lancement d'un test e2e (angular 2) :

- 0) vérifier la connexion réseau (pour téléchargement automatique de certains pilotes)
- 1) se placer dans le répertoire my-app (ou ...) contenant package.json
- 2) lancer d'abord l'application via **ng serve** (ou un équivalent)
- 3) lancer (éventuellement dans une autre console) **ng e2e**

```
> protractor "./protractor.conf.js"

[19:28:53] I/direct - Using ChromeDriver directly...
[19:28:53] I/launcher - Running 1 instances of WebDriver
Spec started

e-ng App
✓ should display title Minibank App

Executed 1 of 1 spec SUCCESS in 4 secs.
[19:29:12] I/launcher - 0 instance(s) of WebDriver still running
[19:29:12] I/launcher - chrome #01 passed
All end-to-end tests pass.
```

Autres fonctionnalités de protractor :

...

3. Contexte et Config tests "karma" / jasmine

package.json

```
{
  "name": "my-app",
  "version": "0.0.0",
  "license": "MIT",
  "angular-cli": {},
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "test": "ng test",
    "pre2e": "webdriver-manager update --standalone false --gecko false",
    "e2e": "protractor"
  },
  "private": true,
  "dependencies": {
    "@angular/common": "^2.3.1",
    "@angular/compiler": "^2.3.1",
    "@angular/core": "^2.3.1",
    "@angular/forms": "^2.3.1",
    "@angular/http": "^2.3.1",
    "@angular/platform-browser": "^2.3.1",
    "@angular/platform-browser-dynamic": "^2.3.1",
    "@angular/router": "^3.3.1",
    "angular-in-memory-web-api": "latest",
    "core-js": "^2.4.1",
    "rxjs": "^5.0.1",
    "ts-helpers": "^1.1.1",
    "zone.js": "^0.7.2"
  },
  "devDependencies": {
    "@angular/compiler-cli": "^2.3.1",
    "@types/jasmine": "2.5.38",
    "@types/node": "^6.0.42",
    "angular-cli": "1.0.0-beta.26",
    "codifyer": "~2.0.0-beta.1",
    "jasmine-core": "2.5.2",
    "jasmine-spec-reporter": "2.5.0",
    "karma": "1.2.0",
    "karma-chrome-launcher": "^2.0.0",
    "karma-cli": "1.0.1",
    "karma-jasmine": "1.0.2",
    "karma-remap-istanbul": "^0.2.1",
    "karma-jasmine-html-reporter": "^0.2.2",
    "protractor": "~4.0.13",
    "ts-node": "1.2.1",
    "tslint": "^4.3.0",
    "typescript": "~2.0.3"
  }
}
```

karma.conf.js

```
// Karma configuration file, see link for more information
// https://karma-runner.github.io/0.13/config/configuration-file.html

module.exports = function (config) {
  config.set({
    basePath: '',
    frameworks: ['jasmine', 'angular-cli'],
```

```

plugins: [
  require('karma-jasmine'),
  require('karma-chrome-launcher'),
  require('karma-remap-istanbul'),
  require('angular-cli/plugins/karma'),
  require('karma-jasmine-html-reporter')
],
files: [
  { pattern: './src/test.ts', watched: false }
],
preprocessors: {
  './src/test.ts': ['angular-cli']
},
mime: {
  'text/x-typescript': ['ts', 'tsx']
},
remapIstanbulReporter: {
  reports: {
    html: 'coverage',
    lcovonly: './coverage/coverage.lcov'
  }
},
angularCli: {
  config: './angular-cli.json',
  environment: 'dev'
},
reporters: config.angularCli && config.angularCli.codeCoverage
  ? ['progress', 'karma-remap-istanbul', 'kjhtml']
  : ['progress', 'kjhtml'],
port: 9876,
colors: true,
logLevel: config.LOG_INFO,
autoWatch: true,
browsers: ['Chrome'],
singleRun: false
});
};

```

NB: le rapporteur 'karma-jasmine-html-reporter' ('kjhtml') permet d'afficher le résultat des tests en mode HTML en cliquant sur l'icône **DEBUG** de la page HTML principale de karma .

src/test.ts

```

// This file is required by karma.conf.js and loads recursively all the .spec and
// framework files

import './polyfills.ts';

import 'zone.js/dist/long-stack-trace-zone';
import 'zone.js/dist/proxy.js';
import 'zone.js/dist/sync-test';
import 'zone.js/dist/jasmine-patch';
import 'zone.js/dist/async-test';
import 'zone.js/dist/fake-async-test';

```



```

import { TestBed } from '@angular/core/testing';
import {
  BrowserDynamicTestingModule,
  platformBrowserDynamicTesting
} from '@angular/platform-browser-dynamic/testing';

// Unfortunately there's no typing for the `__karma__` variable.
// Just declare it as any.
declare var __karma__: any;
declare var require: any;

// Prevent Karma from running prematurely.
__karma__.loaded = function () {};

// First, initialize the Angular testing environment.
getTestBed().initTestEnvironment(
  BrowserDynamicTestingModule,
  platformBrowserDynamicTesting()
);
// Then we find all the tests.
const context = require.context('./', true, /\.spec\.ts$/);
// And load the modules.
context.keys().map(context);
// Finally, start Karma to run the tests.
__karma__.start();

```

4. 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

5. Tests unitaires élémentaires

5.1. 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": "~3.3.8",
  "@types/jasminewd2": "~2.0.3",
  "jasmine-core": "~3.4.0",
  "jasmine-spec-reporter": "~4.2.1",
  "karma": "~4.1.0",
  "karma-chrome-launcher": "~2.2.0",
  "karma-coverage-istanbul-reporter": "~2.0.1",
  "karma-jasmine": "~2.0.1",
  "karma-jasmine-html-reporter": "^1.4.0",
  ...
}
```

Les **assertions** de jasmine s'expriment via **expect(...).to...**

(BDD syntax : Behavior Driven Development) :

```
expect(object_or_var)
  .toEqual(expected)
  .toContain(val)
  .toBe(value)
  .toBeDefined()
  .toBeNull()
  .toBeTruthy()/.toBeFalsy()
  .toHaveBeenCalled()
  ...
```

5.2. 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
```

5.3. 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:

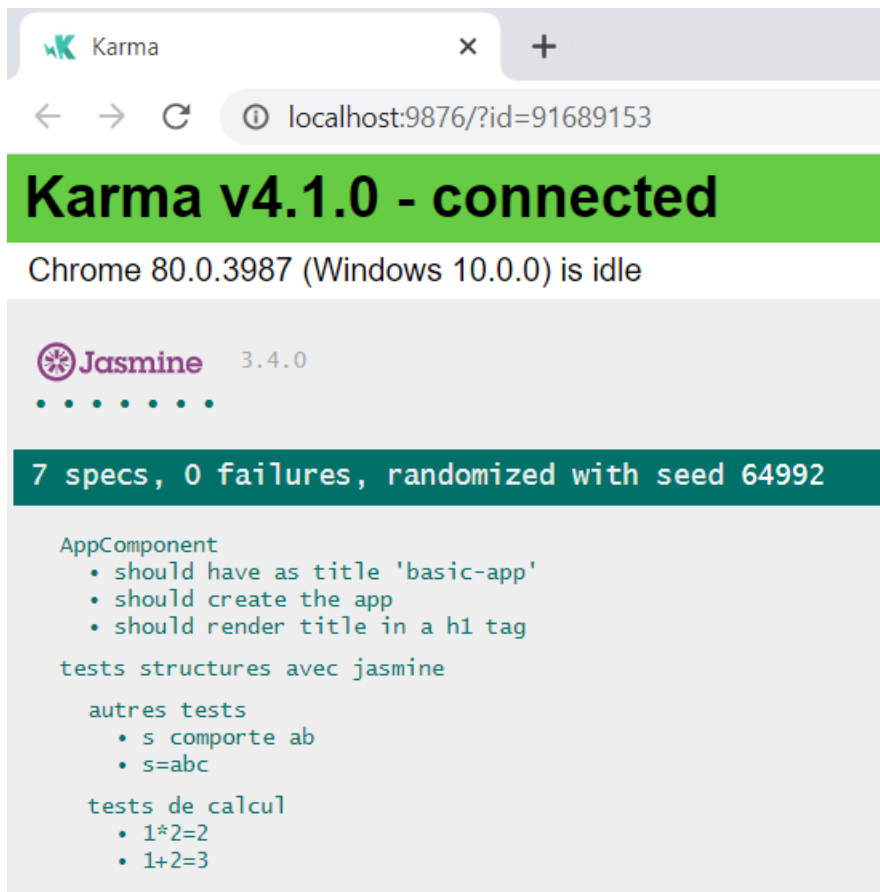
- **beforeEach()** et **afterEach()** sont appelées avant et après l'exécution de chaque **it()** *pour les describe() de même niveau.*
- Souvent , un niveau de **describe()** correspond à un **objet/composant/sujet à tester** et la fonction **it()** signifie "*it should behave like that*".

6. 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 (--single-run)**
- **générer un rapport de couverture de test (--code-coverage)**

Exemple :

```
ng test --single-run --code-coverage
```

7. Tests unitaires "angular"(composants, service, ...)

7.1. 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.

7.2. 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.get(ComputeService);
    //NB: à partir de angular 9 , TestBed.inject(ComputeService) ;
    //est mieux que TestBed.get() dont le type de retour est any
    expect(service).toBeTruthy();
  });

  it('20%tva sur 200 ht = 40', () => {
    const service: ComputeService = TestBed.get(ComputeService);
    expect(service.vat(200,20)).toBe(40);
  });
});
```

7.3. 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<ResConv>
{
  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 );
}

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 result AND mock HTTP ResponseContent :
        let convResult = {source:"EUR",target:"USD",amount:200, result:217.3913}

        //excepted method behavior (just subscribe , deffered) :
        service.convertir(200,'EUR', 'USD').subscribe(res => {
            expect(res).toBe(convResult);
        });

        //expected HTTP request built by convertir() method :
        const req = backend.expectOne({
            url: './devise-api/public/convert?amount=200&source=EUR&target=USD',
            method: 'GET'
        });

        //déclenchement méthode avec mock http response:
        req.flush(convResult, { status: 200, statusText: 'ok' });
    });

    afterEach( inject([HttpTestingController], (httpMock: HttpTestingController) => {
        httpMock.verify(); //requete bien terminée?
    })
    );
});

```

7.4. Test unitaire de composant angular avec TestBed

Exemple :

app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'my-app',
  template: `
    <h1>{{title}}</h1>
    <nav>
      <a routerLink="/dashboard">Dashboard</a>
      <a routerLink="/heroes">Heroes</a>
    </nav>
    <router-outlet></router-outlet>
  `
})
export class AppComponent {
  title = 'Tour of Heroes';
}
```

Exemple de test :

app.component.spec.ts

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { By } from '@angular/platform-browser';
import { DebugElement } from '@angular/core';

import { RouterTestingModule } from '@angular/router/testing';
// for <router-outlet></router-outlet> in template
import { AppComponent } from './app.component';

describe('AppComponent (inline template)', () => {

  let comp: AppComponent;
  let fixture: ComponentFixture<AppComponent>;
  let de: DebugElement;
  let el: HTMLElement;

  //initialisations (avant chaque test)
  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [ RouterTestingModule ], // for <router-outlet> in template
      declarations: [ AppComponent ], // declare the test component
    });

    fixture = TestBed.createComponent(AppComponent);
    comp = fixture.componentInstance; // AppComponent test instance

    // query for the title <h1> by CSS element selector
    de = fixture.debugElement.query(By.css('h1'));
    el = de.nativeElement;
    //variante (v2):
    //const compiledComponentNativeElement = fixture.debugElement.nativeElement;
    //elv2 = compiledComponentNativeElement.querySelector('h1');

    // fixture.detectChanges();
  });

  it('should have a defined component', () => { expect(comp).toBeDefined(); });

  it('no title in the DOM until manually call `detectChanges`', () => {
    expect(el.textContent).toEqual(''); // test scolaire/pédagogique seulement
    // en pratique , premier appel à
    // fixture.detectChanges();
    // à la fin du beforeEach() sans async()
  });
});
```

```

it('should display original title', () => {
  fixture.detectChanges();
  console.log("title:" + el.textContent);
  expect(el.textContent).toContain(comp.title);
});

it('should display a different test title', () => {
  comp.title = 'Test Title';
  fixture.detectChanges();
  expect(el.textContent).toContain('Test Title');
});
});

```

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()** 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.

Attention : sur un vrai projet, il est déconseillé de tester le composant principal "`app.component`" car celui-ci comporte trop de dépendances vers les autres composants et services.

Un test unitaire devrait idéalement ne porter que sur un sous composant bien précis de l'application.

7.5. Ajustements à effectuer pour un template html externe

Dans le cas très fréquent où le template html (ou les styles css) n'est (ou ne sont pas) pas en inline mais dans un fichier ".html" ou ".css" annexe, le code du test doit être ajusté de façon à que certains mécanismes asynchrones d'angular aient le temps de lire, compiler et intégrer les fichiers externes.

Ligne supplémentaire (dans le haut de app.component.spec.ts):

```
import { async } from '@angular/core/testing'; //si templateUrl externe
```

Deux beforeEach() au lieu d'un seul (dans app.component.spec.ts):

```
//first beforeEach with async() for template and css initialization :
beforeEach(async() => {
  TestBed.configureTestingModule({
    imports: [ RouterTestingModule ], // if necessary
    declarations: [ AppComponent ], // declare the test component
  })
  .compileComponents(); // compile asynchronously template and css , return promise
});

//second beforeEach without async() for fixture, component, debugElement:
beforeEach(() => {

  fixture = TestBed.createComponent(AppComponent);

  comp = fixture.componentInstance; // AppComponent test instance

  // query for the title <h1> by CSS element selector
  de = fixture.debugElement.query(By.css('h1'));
  el = de.nativeElement;
});
```

Explications autour de **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();
    });
});

```

La fonction utilitaire **async()** de [@angular/core/testing](#) est une alternative simplifiée permettant de ne pas avoir à appeler explicitement *done()* du standard jasmine. Le *then / done* est caché à l'intérieur.

NB: dans le cas particulier de "Angular_CLI" et *webPack*, l'équivalent de *.compileComponents()* semble être automatiquement effectué lors du packaging et il semble être possible de tester un composant angular comportant un template externe sans avoir à appeler *.compileComponents()* au sein d'un *beforeEach(async(...))*.

Paramètre **NO_ERRORS_SCHEMA** pour éviter erreur suite à dépendances indirectes :

```

 TestBed.configureTestingModule({
   imports: [ FormsModule ],
   declarations: [ XyComponent ],
   schemas: [ NO_ERRORS_SCHEMA ]
 })
 .compileComponents();

```

Le paramètre **NO_ERRORS_SCHEMA** permet au compilateur d'ignorer les éléments non reconnus : un composant déclaré peut utiliser d'autres composants. Ces dépendances indirectes n'auront pas d'utilité dans notre test, mais provoqueraient une erreur d'exécution sans cette option.

7.6. Tester un composant angular utilisant un service simple

Exemple de composant à tester (avec service simple injecté) :

vat.component.ts

```

import { Component } from '@angular/core';
import { ComputeService } from './compute.service';

@Component({
  selector: 'temp-test',
  template: `
    <h3>temp test / value added tax</h3>
    price excluded of tax: <input type='text'
                          id='price_excluded_of_tax_input'
                          [(ngModel)]='price_excluded_of_tax'
                          (input)='onRefresh($event)' /> <br/>
    value added tax rate (%): <input type='text'
                                  id='v_a_tax_rate_pct_input'
                                  [(ngModel)]='v_a_tax_rate_pct'

```

```

        (input)='onRefresh($event)' /> <br/>
        price inclusive of tax :
        <span id='price_inclusive_of_tax_span'>
        {{price_inclusive_of_tax}} </span><br/>
    },
    })
    export class VatComponent {
        price_excluded_of_tax:number;
        v_a_tax_rate_pct:number;
        price_inclusive_of_tax:number;

        constructor(private computeService: ComputeService) { }

        onRefresh(evt : KeyboardEvent){
            this.price_inclusive_of_tax = Number(this.price_excluded_of_tax)
            + Number(this.computeService.vat(this.price_excluded_of_tax,
            this.v_a_tax_rate_pct));
        }
    }
}

```

temp test / value added tax

price excluded of tax:

value added tax rate (%):

price inclusive of tax : 240

Exemple de test :

vat.component.spec.ts

```

import { ComponentFixture, TestBed, async } from '@angular/core/testing';
import { By } from '@angular/platform-browser';
import { DebugElement } from '@angular/core';

import { VatComponent } from './vat.component';
import { ComputeService } from './compute.service';
import { FormsModule } from '@angular/forms';

describe('VatComponent (using simple service)', () => {

    let comp: VatComponent;
    let fixture: ComponentFixture<VatComponent>;
    let deEot, deIot, deRate: DebugElement;
    let elIot: HTMLElement;
    let elEot, elRate: HTMLInputElement;

    let computeServiceWithinTest : ComputeService;

    beforeEach(async(() => {
        //stub Service for test purposes (will be cloned and injected)
        let computeServiceStub = {
            vat(excl_tax : number, vat_pct : number ) : number{
                return excl_tax * vat_pct / 100;
            }
        };
    });

```

```

TestBed.configureTestingModule({
  imports: [ FormsModule ], // FormsModule is for [(ngModel)]
  providers: [ {provide: ComputeService,
                useValue: computeServiceStub } ],
  declarations: [ VatComponent ],
}).compileComponents();
}));

beforeEach( () => {
  fixture = TestBed.createComponent(VatComponent);
  computeServiceWithinTest =
    fixture.debugElement.injector.get(ComputeService);

  comp = fixture.componentInstance; // VatComponent test instance

  deEot = fixture.debugElement.query(
    By.css('#price_excluded_of_tax_input'));
  elEot = deEot.nativeElement;
  deRate = fixture.debugElement.query(
    By.css('#v_a_tax_rate_pct_input'));
  elRate = deRate.nativeElement;
  deIot = fixture.debugElement.query(
    By.css('#price_inclusive_of_tax_span'));
  elIot = deIot.nativeElement;
});

it('20% , 200 -> 240 from model', () => {
  comp.price_excluded_of_tax=200;
  comp.v_a_tax_rate_pct=20; //20%
  comp.onRefresh(null /*not used event*/);
  fixture.detectChanges();
  console.log("from model, price_inclusive_of_vat:"
    +elIot.textContent);
  expect(elIot.textContent).toContain('240');
});

it('test computeServiceWithinTest', () => {
  expect(computeServiceWithinTest).toBeDefined();
  expect(computeServiceWithinTest.vat(100, 20)).toBe(20);
});
});

```

Remarque importante :

Bien que dans cet exemple extra-simple, on aurait pu utiliser en direct le réel service de calcul via `TestBed.configureTestingModule({`

```

  providers: [ ComputeService ] ,
...} ;)

```

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 computeServiceStub = {
  vat(excl_tax : number, vat_pct : number ) : number{
    return excl_tax * vat_pct / 100;
  }
};

```

```
TestBed.configureTestingModule({
  imports: [ FormsModule ], // FormsModule is for [(ngModel)]
  providers: [ {provide : ComputeService,
                useValue : computeServiceStub } ],
  declarations: [ VatComponent ],
}).compileComponents();
```

Simuler/déclencher événement via `HTMLInputElement.dispatchEvent()` :

```
it('10% , 300 -> 330 from input', () => {
  elEot.value="300"; //saisir 300 dans zone input eot/ht
  elEot.dispatchEvent(new Event('input')); //déclencher evt input
  fixture.detectChanges();
  expect(Number(component.price_excluded_of_tax)).toBe(300);

  elRate.value="10"; //saisir 10 au sens 10% dans zone input rate/taux
  elRate.dispatchEvent(new Event('input')); //déclencher evt input
  fixture.detectChanges();
  expect(Number(component.v_a_tax_rate_pct)).toBe(10);

  console.log("from html, price_inclusive_of_vat:" + elIot.textContent);
  expect(elIot.textContent).toContain('330');
});
```

7.7. 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 { ResConv } from '../common/data/res-conv';

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

  public devises=null;

  public source : string ;
  public cible : string ;
```



```

public montantSource : number = 100;
public montantCible : number;

constructor(private deviseService : DeviseService) { }

ngOnInit() {
  this.deviseService.getDevises().subscribe(
    (devises)=>{this.devises = devises; this.initDefault(); },
    (err)=>{console.log(err);}
  );
}

private initDefault(){
  if(this.devises && this.devises[0] ){
    this.source=this.devises[0].code;
    this.cible=this.devises[0].code;
  }
}

public onConvertir(evt:any){
  //console.log(` source=${this.source} cible=${this.cible} montant=${this.montantSource}`)
  this.deviseService.convertir(this.montantSource,this.source,this.cible)
    .subscribe((resConv:ResConv)=> { this.montantCible = resConv.result;
    console.log("resConv="+JSON.stringify(resConv)); });
}
}

```

conversion.component.html

```

<div>
<form>
  <label>montant à convertir:</label>
  <input name="montantSource" [(ngModel)]="montantSource" />
  <br/>
  <label>monnaie source:</label>
  <select name="source" [(ngModel)]='source'>
    <option *ngFor="let d of devises">{{d.code}}</option>
  </select>
  <br/>
  <label>monnaie cible:</label>
  <select name="cible" [(ngModel)]='cible'>
    <option *ngFor="let d of devises">{{d.code}}</option>
  </select>
  <br/>
  <input type='button' id="btnConv" value='convertir' (click)="onConvertir($event)" /> <br/>
</form>
<hr/>
montant converti : <b>{{montantCible | number:'1.0-2'}}</b>
</div>

```

montant à convertir:

monnaie source:

monnaie cible:

montant converti : **217.39**

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 { async, ComponentFixture, TestBed, fakeAsync, tick } from '@angular/core/testing';

import { ConversionComponent } from './conversion.component';
import { FormsModule } from '@angular/forms';
import { DeviseService } from '../common/service/devise.service';
import { HttpClientModule } from '@angular/common/http';
import { of } from 'rxjs';
import { DebugElement } from '@angular/core';
import { By } from '@angular/platform-browser';

describe('ConversionComponent', () => {
  let component: ConversionComponent;
  let fixture: ComponentFixture<ConversionComponent>;
  let deSourceSelect : DebugElement;    let elSourceSelect : HTMLSelectElement;
  let deCibleSelect : DebugElement;    let elCibleSelect : HTMLSelectElement;
  let deMontantSource : DebugElement;  let elMontantSource : HTMLInputElement;
  let deBtnConv : DebugElement;        let elBtnConv : HTMLInputElement;
  let deviseServiceWithinTest : DeviseService;
  let spy : jasmine.Spy;

  beforeEach(async() => {
    TestBed.configureTestingModule({
      imports: [ FormsModule , HttpClientModule ],
      declarations: [ ConversionComponent ],
      providers: [ DeviseService ]
    })
  })
```

```

.compileComponents();
}));

beforeEach(() => {
  fixture = TestBed.createComponent(ConversionComponent);
  deviseServiceWithinTest = fixture.debugElement.injector.get(DeviseService);

  /*
  let stubConvResult = {source:"EUR",target:"USD",amount:200, result:217.3913}
  spy = spyOn(deviseServiceWithinTest, 'convertir')
    .and.returnValue(of(stubConvResult));
  */

  spy = spyOn(deviseServiceWithinTest, 'convertir')
    .and.callFake(function(montant : number, source :string , cible : string){
      let convResult = {source:source,target:cible,amount:montant, result:0};
      if(source=='EUR'&&cible=='USD')
        convResult.result=217.3913;
      else if(source==cible)
        convResult.result=montant;
      return of(convResult);
    });

  component = fixture.componentInstance;
  deMontantSource = fixture.debugElement.query(By.css('input[name=montantSource]'));
  elMontantSource = deMontantSource.nativeElement;
  deSourceSelect = fixture.debugElement.query(By.css('select[name=source]'));
  elSourceSelect = deSourceSelect.nativeElement;
  deCibleSelect = fixture.debugElement.query(By.css('select[name=cible]'));
  elCibleSelect = deCibleSelect.nativeElement;
  deBtnConv = fixture.debugElement.query(By.css('#btnConv'));
  elBtnConv = deBtnConv.nativeElement;
  fixture.detectChanges();
});

it('should create', () => { expect(component).toBeTruthy();
});

it('should display good conversion result', async() => {
  /*
  component.montantSource=200;
  component.source='EUR';
  component.cible='USD';
  component.onConvertir(null);
  */

  // wait for async activities (observable/promise/event/...)
  fixture.whenStable().then(() => {
    component.devises= [
      { code : "EUR" , name : "Euro" , change : 1.0} ,
      { code : "GBP" , name : "Livre" , change : 0.9} ,
      { code : "JPY" , name : "Yen" , change : 123.1} ,
    ]
  })
});

```

```

    { code : "USD" , name : "Dollar" , change : 1.1 }
  ];
  fixture.detectChanges(); //in order to synchronize options of select
  elMontantSource.value="200"; elMontantSource.dispatchEvent(new Event('input'));
  elSourceSelect.value=elSourceSelect.options[0].value;//"EUR";
  elSourceSelect.dispatchEvent(new Event('change'));
  elCibleSelect.value=elSourceSelect.options[3].value;//"USD";
  elCibleSelect.dispatchEvent(new Event('change'));
  elBtnConv.dispatchEvent(new Event('click'));//déclencher evt click
  fixture.detectChanges();
  expect(Number(component.montantSource)).toBe(200);
  expect(component.source).toBe('EUR');
  expect(component.cible).toBe('USD');
  // fixture.detectChanges();
  expect(spy.calls.any()).toBe(true, 'convertir should be called');
  console.log("conversion - component.montantCible:"+component.montantCible);
  expect(component.montantCible).toBeCloseTo(217.3913,0.0001);
  });
  });

  it('conversion result=montantSource if cible=source', fakeAsync() => {
    component.montantSource=200;
    component.source='EUR';
    component.cible='EUR';
    component.onConvertir(null);
    tick();//waiting inside fakeAsync for observable or promise result
    fixture.detectChanges();
    expect(spy.calls.any()).toBe(true, 'convertir should be called');
    console.log("conversion - component.montantCible:"+component.montantCible);
    expect(component.montantCible).toBeCloseTo(200,0.0001);
  });
});

```

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(...);
  });
)

```

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 équivalent** et **permettre d'attendre la fin d'une opération asynchrone** indirectement déclenchée par un composant angular et un service.

En pratique, certains bugs, limitations techniques et autres problèmes font qu'un test d'appel asynchrone est assez délicat à mettre au point.

Plus en détails :

tick(25) ; effectue une attente de 25ms .

Deux appels consécutifs à tick(25) effectuent globalement une attente de 50 ms .

Sans argument, tick() est équivalent à tick(0).

Depuis angular 4.3 , flush() peut quelquefois être utilisé à la place de tick().

A priori , fakeAsync() et les mécanismes de "mock" d'angular mettent en place des "timeout" internes et un appel à tick() allonge le temps d'attente tandis qu'un appel à flush() récupère le temps d'attente une fois l'opération terminée.

VII - PWA , Services workers, mode déconnecté

1. Mode "offLine" et indexed-db

1.1. Gestion de online/offline par les navigateurs

La plupart des navigateurs détectent et gère le mode "déconnecté" de la manière suivante :

- la propriété booléen **window.navigator.onLine** est automatiquement fixée par le navigateur pour indiquer si la connexion à internet est établie ou coupée .

Les événements "**online**" et "**offline**" sont automatiquement déclenchés par le navigateur en cas de basculement / changement d'état .

1.2. exemple de service angular "OnlineOfflineService"

```
...
@Injectable({ providedIn: 'root'})
export class OnlineOfflineService {
  public connectionChanged = new BehaviorSubject<boolean>(window.navigator.onLine);
  get isOnline() { return window.navigator.onLine; }
}

constructor() {
  window.addEventListener('online', () => this.updateOnlineStatus());
  window.addEventListener('offline', () => this.updateOnlineStatus());
}

private updateOnlineStatus() {
  console.log("onLine="+window.navigator.onLine);
  this.connectionChanged.next(window.navigator.onLine);
}
}
```

Exemple d'utilisation :

```
export class FooterComponent implements OnInit {
  private onLine:boolean;
  constructor( private onlineOfflineService: OnlineOfflineService) {}
  ngOnInit() { this.onlineOfflineService.connectionChanged
    .subscribe( (onLine)=>{this.onLine = onLine;})
  }
}
```

et onLine={{onLine}} coté .html

2. IndexedDB et idb

2.1. Présentation de 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

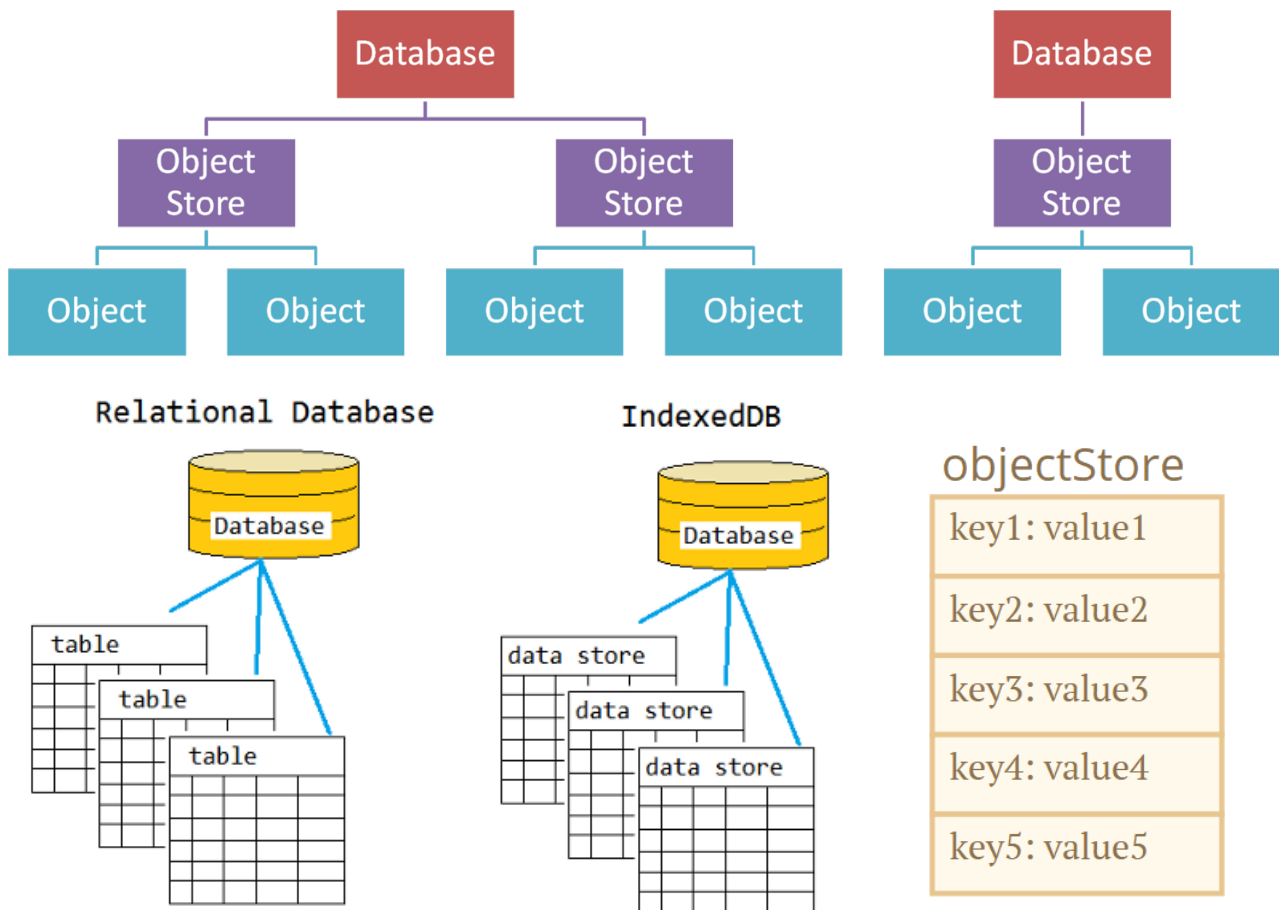
bien expliqués:

- <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-3d047dddd313>

2.2. Structure de IndexedDB



2.3. Utilisation de IndexedDB via idb (avec "Promise") :

```
if (!('indexedDB' in window)) {
  console.log('This browser doesn\'t support IndexedDB');
  return;
}
```

npm install -s idb

(ex : *version 4.0.4*)

Les exemples de code (partiels) ci-après seront en typescript et intégrés au framework "angular" .


```
import { Observable, of, from } from 'rxjs';
import { openDB , 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 async 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();
  }
}

```

3. PWA (Progressive Web App) – aperçu général

3.1. 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 :

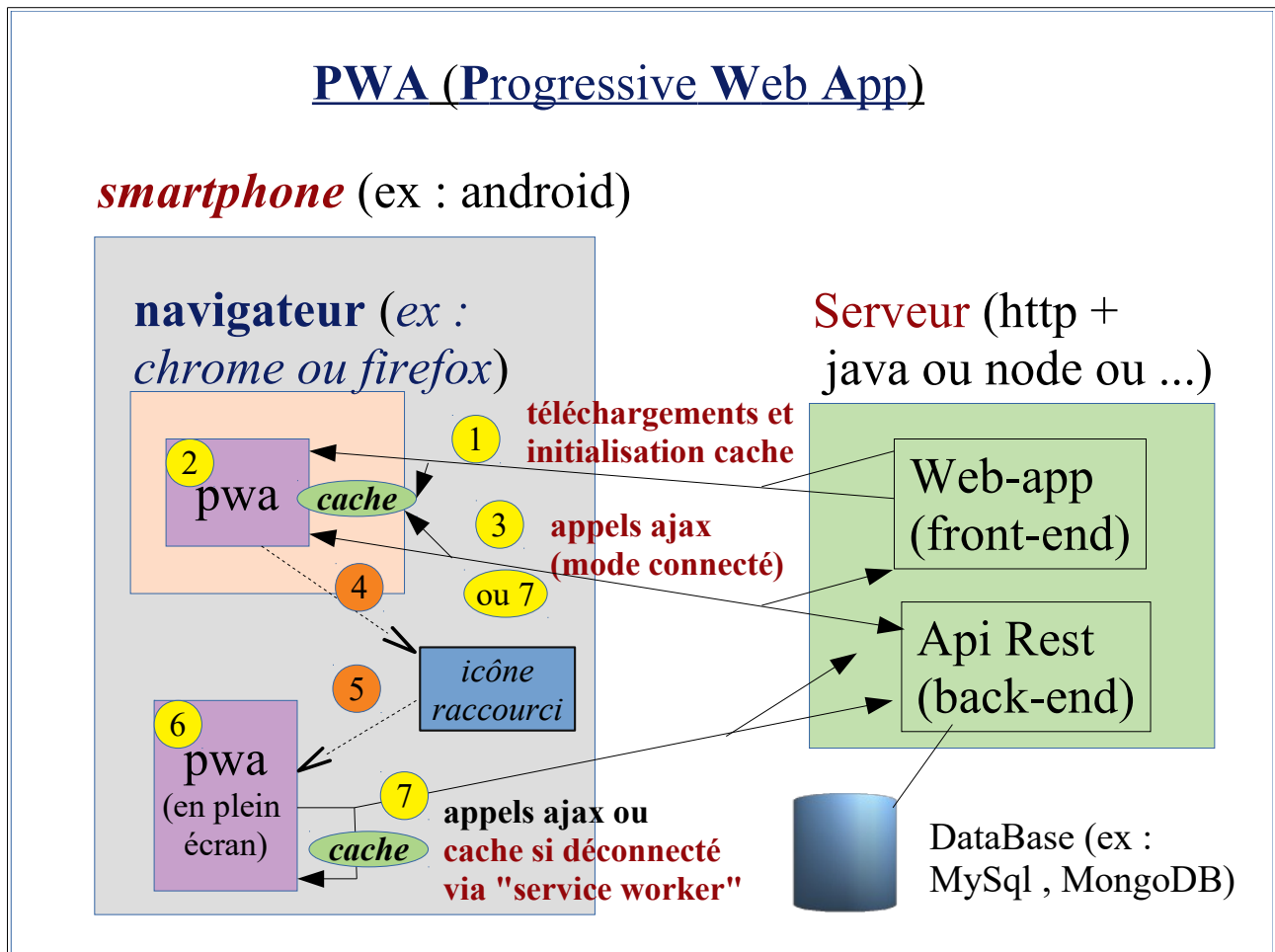
	Développement	Exécution
Application mobile native (ex : java pour Android)	Langage et api spécifique à une plateforme mobile (ex : java et android-sdk ou bien ios/iphone/swift)	application mobile (à télécharger depuis un "store") et à exécuter sur un type précis de smartphone (android ou iphone)
Application 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).



3.2. 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*
-

3.3. 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) .

3.4. 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
- ...

3.5. 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
-

4. Service Worker (essentiel)

4.1. Présentation des "service worker"

Un "service worker" est un composant complètement invisible qui travaille en tâche de fond et qui n'interagit jamais directement avec l'interface utilisateur (pas de DOM, ...).

Bien que pas directement associé à la structure d'une page HTML , un "service worker" est associé à une partie d'une certaine application web et selon sa portée il sera fonctionnel et accessible depuis une ou plusieurs pages HTML de l'application web (html5/css3/javascript) .

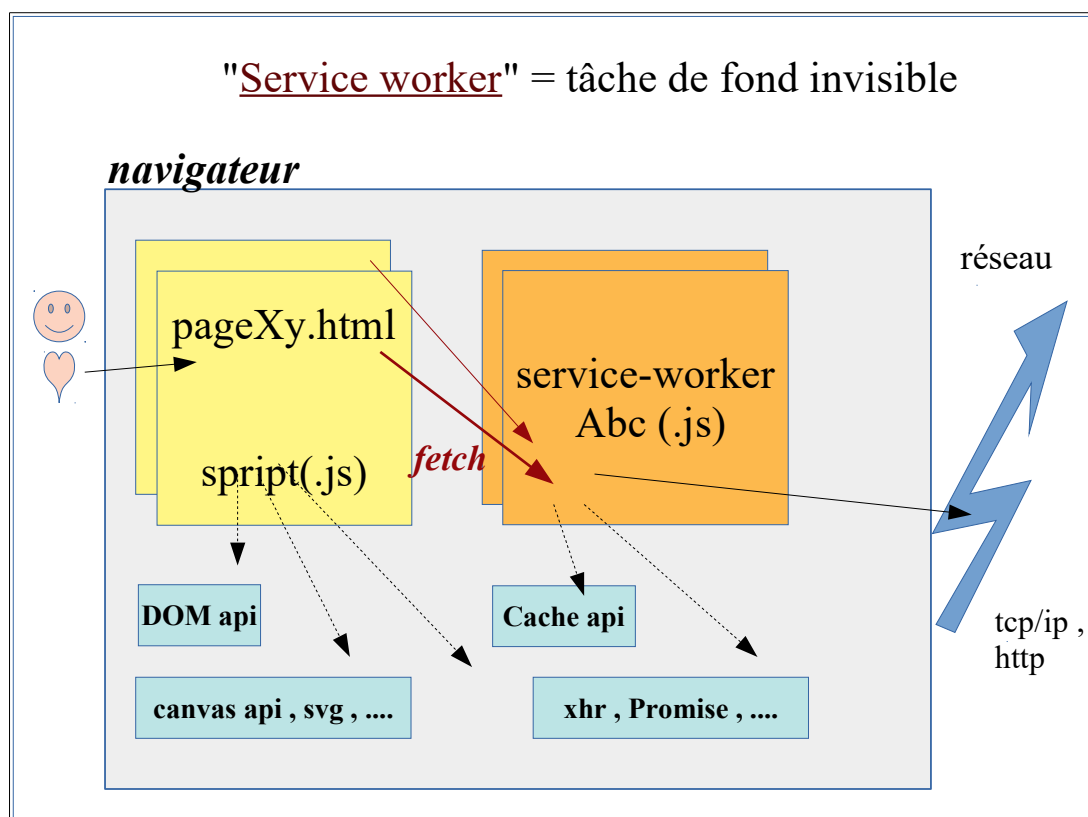
Un "service worker" agira essentiellement comme un proxy (intermédiaire transparent mais à programmer) entre la partie "front end" cliente de l'application web et la fréquente autre partie "back end" (comportant généralement quelques web services "rest" à invoquer via AJAX) .

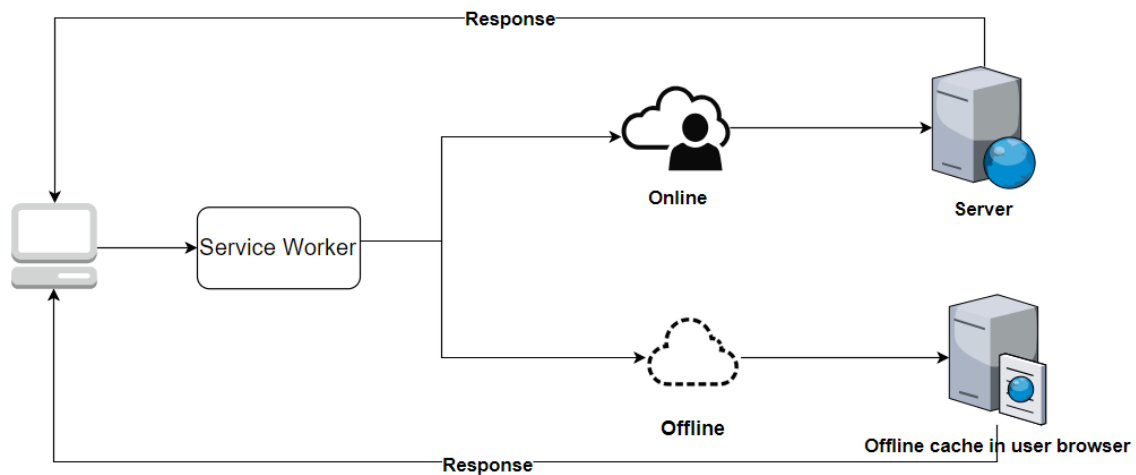
Ce "service worker" agissant comme un proxy pourra par exemple:

- récupérer certaines données distantes en mode "connexion ouverte"
- les stocker en cache
- restituer ces données en mode "déconnecté / off-line" pour que l'application puisse au moins afficher quelques choses et fonctionner partiellement lorsque le réseau 4G ou Wifi est inaccessible .

Caractéristiques principales d'un "service-worker" :

- invisible en arrière plan (pas d'accès au DOM)
- asynchrone et non-bloquant (Promise ou ...)
- normalement associé à HTTPS (http déconseillé)





4.2. Portée et enregistrement d'un service-worker

Bien que pas directement associé à une page html spécifique , un service-worker a une **portée** qui est directement liée à l'arborescence des URLs d'un domaine ou d'une application .

Par exemple , un service-worker dont l'enregistrement aura été déclenché par *http(s)://www.xyz.com/my-app/index.html* pourra ensuite être utilisé pour intercepter toutes les requêtes issues des pages suivantes :

http(s)://www.xyz.com/my-app/page2.html

http(s)://www.xyz.com/my-app/foo/pagefoo.html

L'enregistrement d'un service-worker est donc assez souvent effectué depuis la page principale "index.html" .

Exemple d'enregistrement (au sein de index.html) :

```

<script>
if ('serviceWorker' in navigator) {
  window.addEventListener('load', function() {
    navigator.serviceWorker.register('service-worker.js').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);
    });
  });
}
</script>

```

Attention : ne pas confondre "enregistrement / registration" et "(re-)installation" .

La phase d'enregistrement ci-dessus ne fait que demander au navigateur de prendre en compte

l'existence du service-worker et de lui associer un domaine (ou portée) .

Dans le cas d'un premier enregistrement , la phase d'installation sera généralement enclenchée juste après par le navigateur.

Dans la cas d'un ré-enregistrement, la phase de (ré-)installation ne sera enclenchée par le navigateur que si le code du fichier "service-worker.js" a changé depuis l'enregistrement précédent .

NB : il est éventuellement possible d'ajuster la portée (le scope relatif) d'un service worker lors de son enregistrement :

```
navigator.serviceWorker.register('/worker.js', {scope: '/'})
```

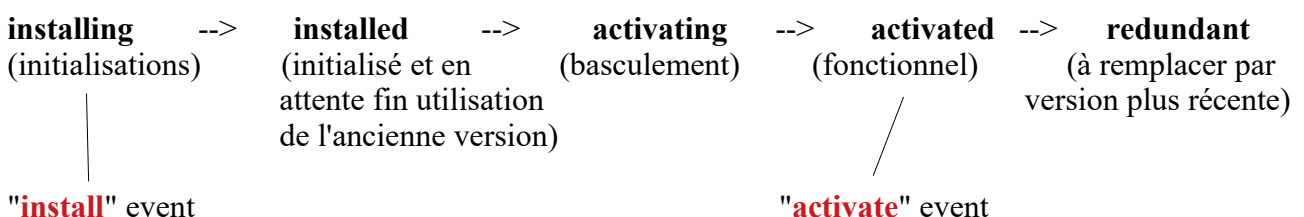
NB : il ne peut y avoir qu'un seul "service-worker" actif et utilisé par une certaine page html.

Dans le cas où plusieurs "workers" sont enregistrés (à différents niveaux de l'arborescence) , ce sera le "worker" au plus près de la page active qui sera utilisé .

Un fichier xy-worker.js (enregistré depuis une certaine page) peut cependant faire référence à des sous fichiers "sub-worker-x.js" , "..." via **importScripts()** et d'autre part , un worker peut éventuellement fournir de multiples gestionnaires (handler) pour un certain événement (ex : "fetch")

4.3. Cycle de vie d'un "service worker" (événements)

Etant donné que le code d'un "service worker" pourra évoluer au cours du temps (versions de plus en plus récente) , un cycle de vie élaboré (géré/contrôlé par le navigateur) permettra dans certains cas d'obtenir un basculement contrôlé entre un ancien et un nouveau "service worker" : Attente de la fin des utilisations en cours de l'ancien "service worker" pour pouvoir basculer sur le nouveau .



Phase d'installation (ou de ré-installation) :

Un service worker est un fichier javascript dont l'installation initiale est généralement déclenchée par du code lié à une page html importante (ex : index.html) .

Pour garantir un code régulièrement à jour , un re-téléchargement sera retenté périodiquement (ex : toutes les 24h) et si le code a changé une ré-installation sera alors automatiquement lancée.

Une ré-installation pourra également être déclenchée (si nécessaire , si code modifié) suite à un rechargement de la page html principale .

C'est généralement durant cette phase d'installation (à la réception de l'événement "install") que le service-worker stocke en cache certaines ressources de l'application permettant un ultérieur

fonctionnement en mode déconnecté .

Ces ressources peuvent soit être statiques (.html, .css, .js,) soit être générées dynamiquement par du code coté serveur (ex : WS-REST java , php ou nodeJs retournant des données "json" ou "...") .

Le nom logique du cache devrait idéalement être de type "my-site-cache-v1" ou "...-v2" , ... de manière à distinguer la version "n" de l'ancienne version "n-1" .

NB : dans le cas d'une ré-installation , la nouvelle version installée (en parallèle) peut temporairement coexister avec l'ancienne version (potentiellement toujours en phase d'exécution) .

Exemple (partie de service-worker.js) :

```
var CACHE_NAME = 'my-site-cache-v1';
var urlsToCache = [
  '/css/styles.css',
  '/js/main.js',
  '/json/sampleData.json'
];

self.addEventListener('install', function(event) {
  console.log('[ServiceWorker] install');
  // Perform install steps
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(function(cache) {
        //console.log('Opened cache before addAll urlsToCache');
        return cache.addAll(urlsToCache);
      })
  );
});
```

Phase d'activation :

Soit juste après la phase d'installation soit après l'attente de la fin des exécutions des pages html utilisant une ancienne version d'un "service-worker" , la réception de l'événement "**activate**" marque le début de la phase de fonctionnement du "service-worker" .

C'est généralement au moment de la réception de l'événement "activate" que le service-worker pourra supprimer l'ancien cache devenu inutile (en version "n-1") .

Exemple (partie de la version 2 de service-worker.js) :

```
self.addEventListener('activate', function(event) {
  var cacheWhitelist = ['my-site-cache-v2', 'my-other-cache-v2'];
  // delete all existing caches that are not in cacheWhitelist :
  event.waitUntil(
    caches.keys().then(function(cacheNames) {
      return Promise.all(
        cacheNames.map(function(cacheName) {
          if (cacheWhitelist.indexOf(cacheName) === -1) {
```

```

        return caches.delete(cacheName);
    }
}
);
});
);
});

```

4.4. Fonctionnement d'un "service-worker" activé

Lorsqu'un service-worker a été préalablement installé et activé au sein d'un navigateur , il fonctionnera comme une tâche de fond invisible .

Le principal rôle d'un "service-worker" est d'agir comme un "**proxy**" ou "**intercepteur transparent**" au niveau des échanges (requêtes http) qui sont effectués entre une page html et le coté serveur .

Un "service-worker" pourra ainsi intercepter :

- des téléchargements de styles css , images , vidéos , ...
- des téléchargements de fichiers "javascript" (bibliothèques , bundles , scripts,)
- des téléchargements de données "json" (statiques ou bien générées dynamiquement via des web services REST)
- autres échanges ("post" , "push" ,)

Lorsque du code javascript lié à une page html effectue un appel ajax (via XMLHttpRequest) ou bien lorsque suite à l'analyse d'une page html le navigateur a besoin de télécharger des ressources annexes (.css , .js , .json , .jpeg , .png , ...) , un navigateur récent (ex : Firefox ou Chrome) va alors automatiquement déclencher l'événement "**fetch**" au niveau d'un service-worker associé au domaine (ou site) de la page html .

Lorsque l'événement "**fetch**" est reçu par le service-worker , celui ci a alors (en tant qu'intercepteur) la possibilité de :

- retransmettre (ou pas) la requête au coté serveur (en mode connecté)
- renvoyer directement des fichiers préalablement stockés dans un cache (en mode déconnecté)
- effectuer d'autres tâches

Exemple simple:

Le service-worker élémentaire ci dessous va renvoyer un élément du cache s'il existe ou bien retourner la réponse du coté serveur sinon :

```

self.addEventListener('fetch', function(event) {
  event.respondWith(
    caches.match(event.request)
      .then(function(response) {
        // Cache hit - return response
        if (response) {
          return response;
        }
      })
    return fetch(event.request);
  )
});

```

```
)
);
});
```

Exemple un peu plus sophistiqué :

Cette version un peu plus sophistiquée (mais encore beaucoup améliorable) agrandi le contenu du cache au fur et à mesure des appels effectués vers le serveur :

```
self.addEventListener('fetch', function(event) {
    console.log('[ServiceWorker] Fetch', event );
    event.respondWith(
        caches.match(event.request)
        .then(function(response) {
            // Cache hit - return response from cache
            console.log("cache hit for fetch request " + JSON.stringify(event.request));
            if (response) {
                return response;
            }

            //else standard (network) http request/fetch .

            // IMPORTANT: Clone the request. A request is a stream and
            // can only be consumed once. Since we are consuming this
            // once by cache and once by the browser for fetch, we need
            // to clone the request.
            var fetchRequest = event.request.clone();

            return fetch(fetchRequest).then(
                function(response) {
                    // Check if we received a valid response
                    if(!response || response.status !== 200 || response.type !== 'basic') {
                        return response;
                    }

                    // IMPORTANT: Clone the response. A response is a stream
                    // and because we want the browser to consume the response
                    // as well as the cache consuming the response, we need
                    // to clone it so we have two streams.
                    var responseToCache = response.clone();

                    caches.open(CACHE_NAME)
                        .then(function(cache) {
                            console.log("put request response in cache " +
                                JSON.stringify(event.request));
                            cache.put(event.request, responseToCache);
                        });

                    return response;
                }
            );//end of fetch(...).then(
        }
    ) //end of caches.matches(...).then(
); //end of event.respondWith
```

});

Principaux événements sur service-worker :

événements	sémantiques
install	début d'installation : le bon moment pour initialiser un cache
activate	début du mode actif : le bon moment pour supprimer ancien cache
fetch	requête à intercepter : on retourne un contenu en cache ou bien on retransmet la requête au serveur ou on effectue une combinaison évoluée .
message	réception d'un message provenant d'un autre script .
sync	synchronisation (maintenant possible , en mode différée) avec le coté serveur lorsque l'on est plus en mode déconnecté : le bon moment pour propager coté serveur des actions effectuées et enregistrées que du coté client .
push	données reçues en mode " <i>push</i> " (provenance=coté serveur) . Souvent des notifications

5. Web App Manifest / add to home screen

5.1. 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,) .

5.2. 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"
}

```

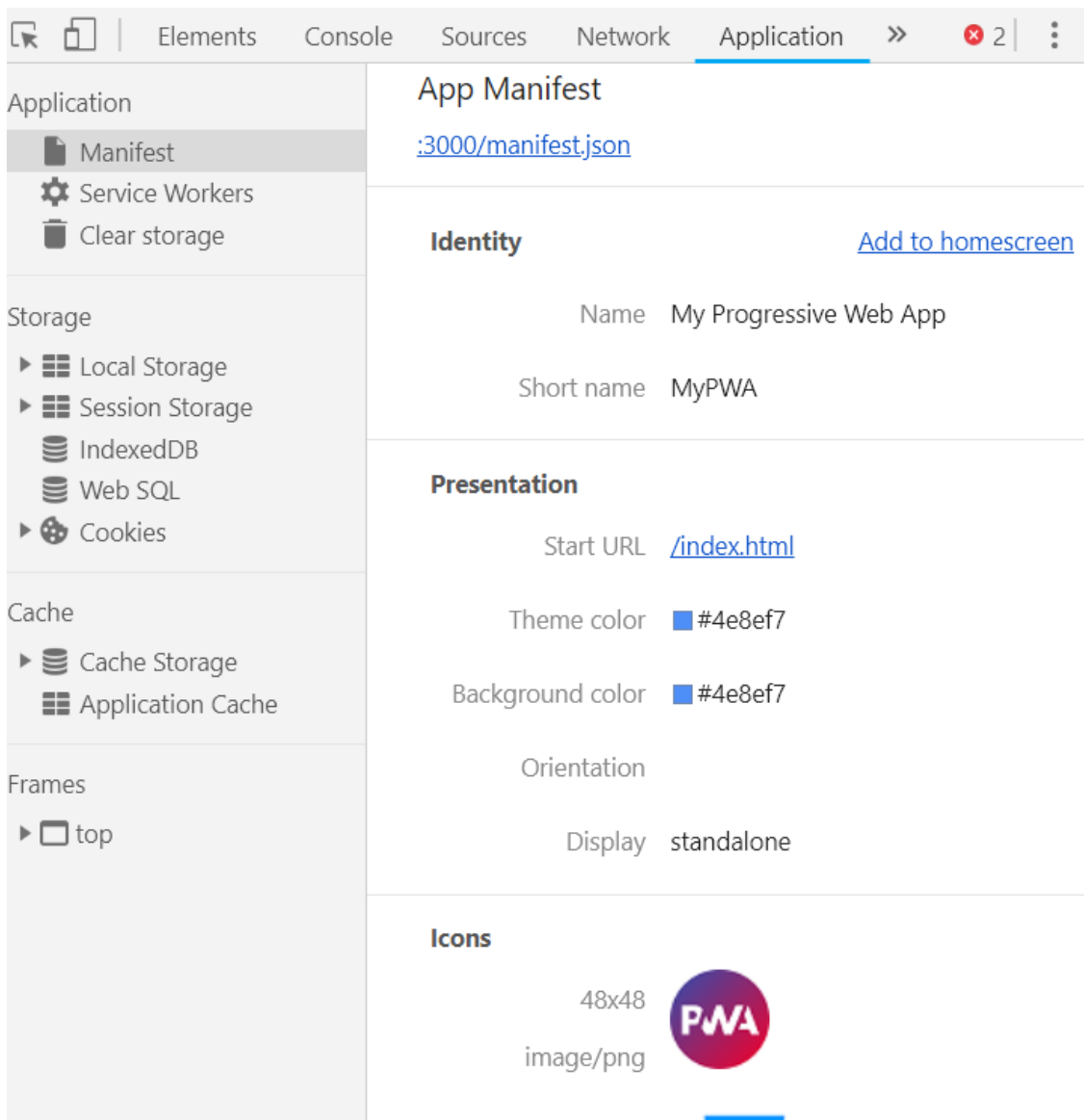
Ce fichier doit être référencé au niveau de **index.html**

```

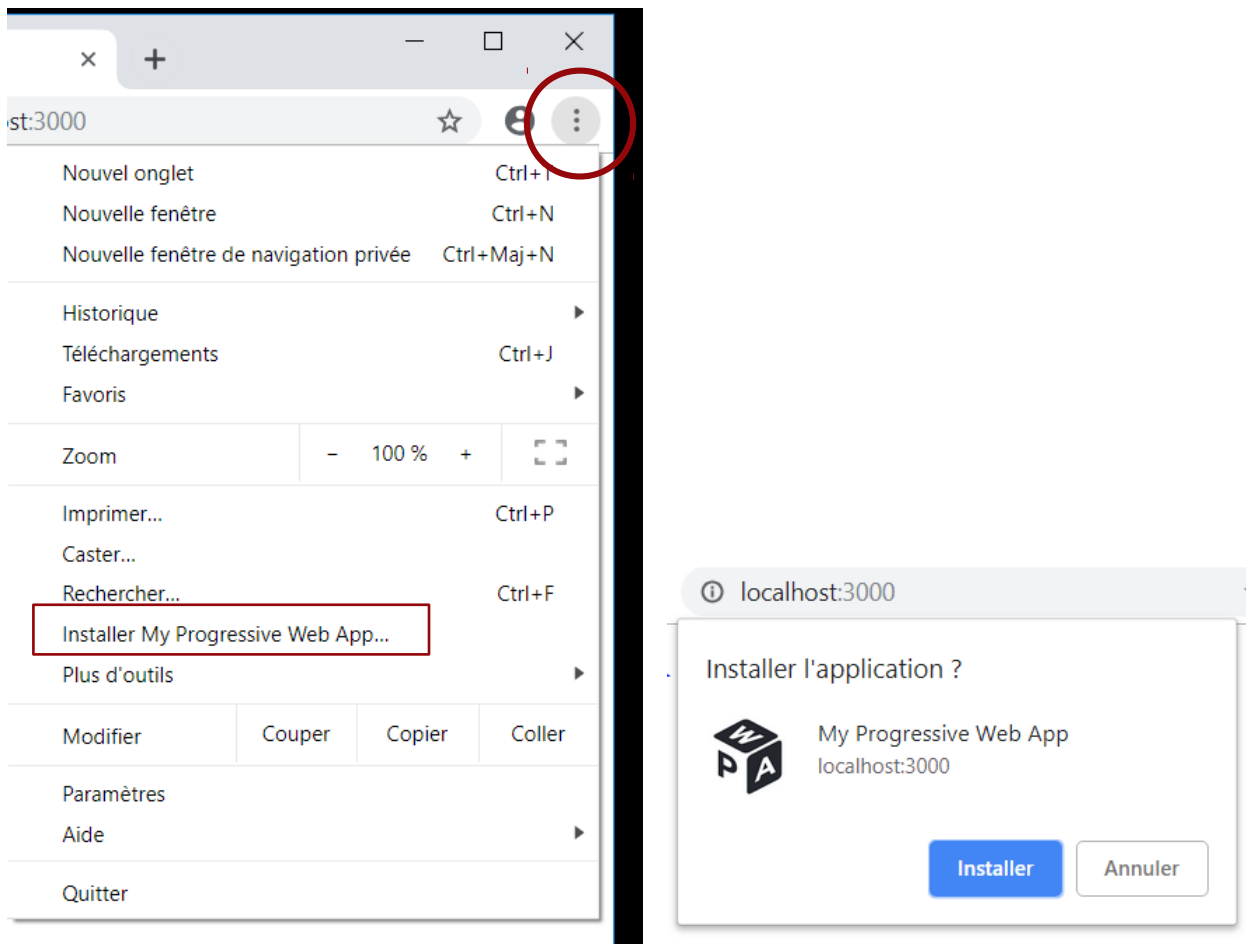
<!DOCTYPE html>
<html> <head>
  <meta charset="UTF-8">
  <title>My progressive Web App</title>
  <link rel="manifest" href="manifest.json">
</head>
<body>    ... </body> </html>

```

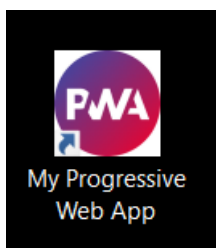
Reconnaissance du fichier "**web app manifest**" au sein de la partie "**Application**" de "**outils de développement**" du navigateur "Chrome Desktop" :



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



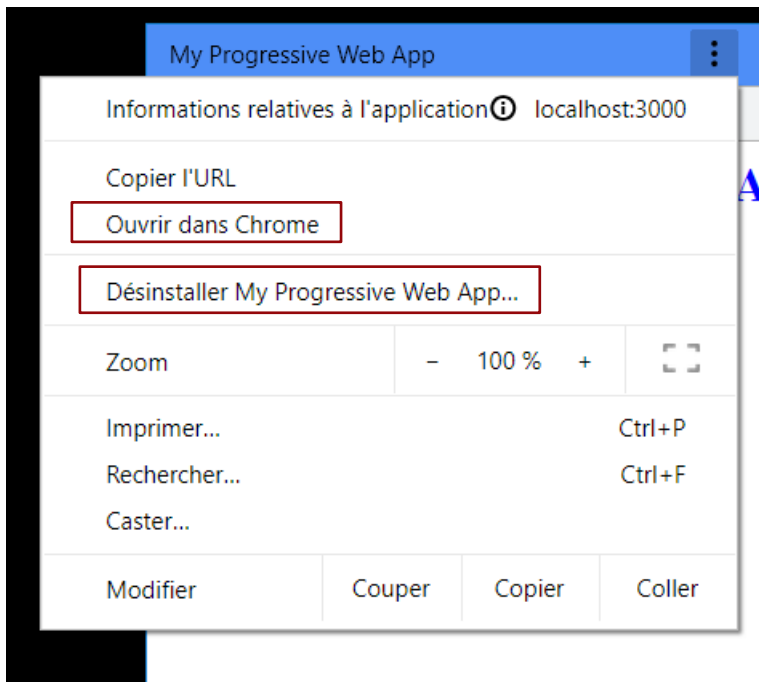
==> cette action génère un icône raccourci sur le fond du bureau :



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" .



5.4. 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 ***');  
});
```

5.5. 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 .

5.6. 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 .

5.7. Lien ou bouton facilitant l'installation (A2HS)

Pour inviter l'utilisateur à installer l'application en mode A2HS (Add To Home Screen) , il est possible d'ajouter un lien ou bouton qui sera pris en compte par différents navigateurs mobiles (ex : "Chrome pour android" , "Firefox pour android") .

Ce bouton servira simplement à inviter l'utilisateur (via javascript) à gérer l'événement **"beforeinstallprompt"** prévu pour demander l'installation en fond de bureau .

Ce bouton soit idéalement être invisible avant la réception de l'événement .

Exemple de code :

```
<button class="add-button">Add to home screen</button>
```

```
.add-button {
  position: absolute;
  top: 1px;
  left: 1px;
}
```

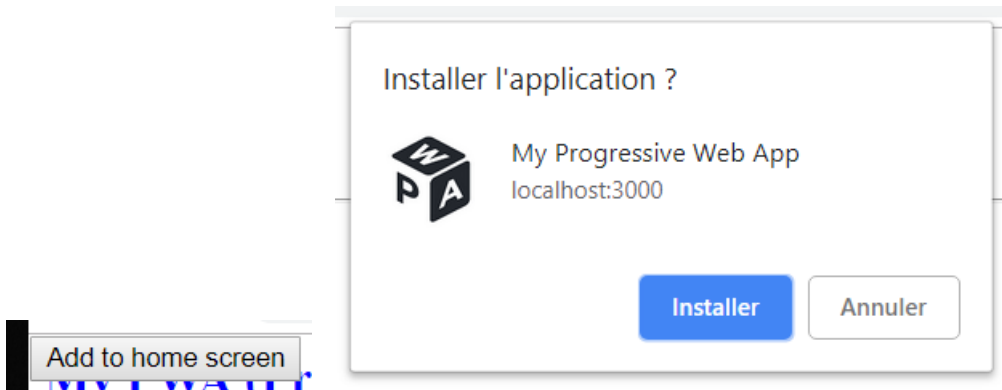
dans un bloc de script en fin de partie "body" :

```
let deferredPrompt;
const addBtn = document.querySelector('.add-button');
addBtn.style.display = 'none';

window.addEventListener('beforeinstallprompt', (e) => {
  // Prevent Chrome 67 and earlier from automatically showing the prompt
  e.preventDefault();
  // Stash the event so it can be triggered later.
  deferredPrompt = e;
  // Update UI to notify the user they can add to home screen
  addBtn.style.display = 'block';

  addBtn.addEventListener('click', (e) => {
    // hide our user interface that shows our A2HS button
    addBtn.style.display = 'none';
    // Show the prompt
    deferredPrompt.prompt();
    // Wait for the user to respond to the prompt
    deferredPrompt.userChoice.then((choiceResult) => {
      if (choiceResult.outcome === 'accepted') {
        console.log('User accepted the A2HS prompt');
      } else {
        console.log('User dismissed the A2HS prompt');
      }
      deferredPrompt = null;
    });
  });
});
```

Effets (en mode "pré-test") avec "Chrome-Desktop" sous windows :



NB :

L'événement "***beforeinstallprompt***" ne sera déclenché par un ***navigateur mobile récent*** que si les conditions suivantes sont réunies :

- L'appli (PWA) n'est pas déjà installée
- "user engagement heuristic" (l'utilisateur a activement utilisé l'appli au moins 30s)
- L'appli (pwa) comporte le fichier "web app manifest".
- L'appli (pwa) est servie par un serveur sécurisé (https).
- Au moins un "service worker" est enregistré avec un gestionnaire pour l'événement fetch .

Nb : A 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'`) .

6. 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) .

6.1. 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*

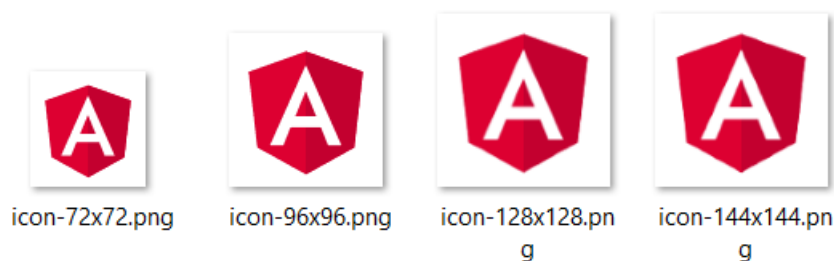
```
{  
  "name": "myapp",  
  "short_name": "myapp",  
  "theme_color": "#1976d2",  
  "background_color": "#fafafa",  
  "display": "standalone",  
  "scope": "/",  
  "start_url": "/",  
  "icons": [  
    {  
      "src": "assets/icons/icon-72x72.png",  
      "sizes": "72x72",  
      "type": "image/png"  
    },  
    {  
      "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 })
],
```

6.2. 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) )
    }
  ]
}

```

6.3. 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

6.4. 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** :

- ...
- ...

6.5. 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>

6.6. Exemple d'utilisation du service SwPush

...

6.7. 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"
]
...
```

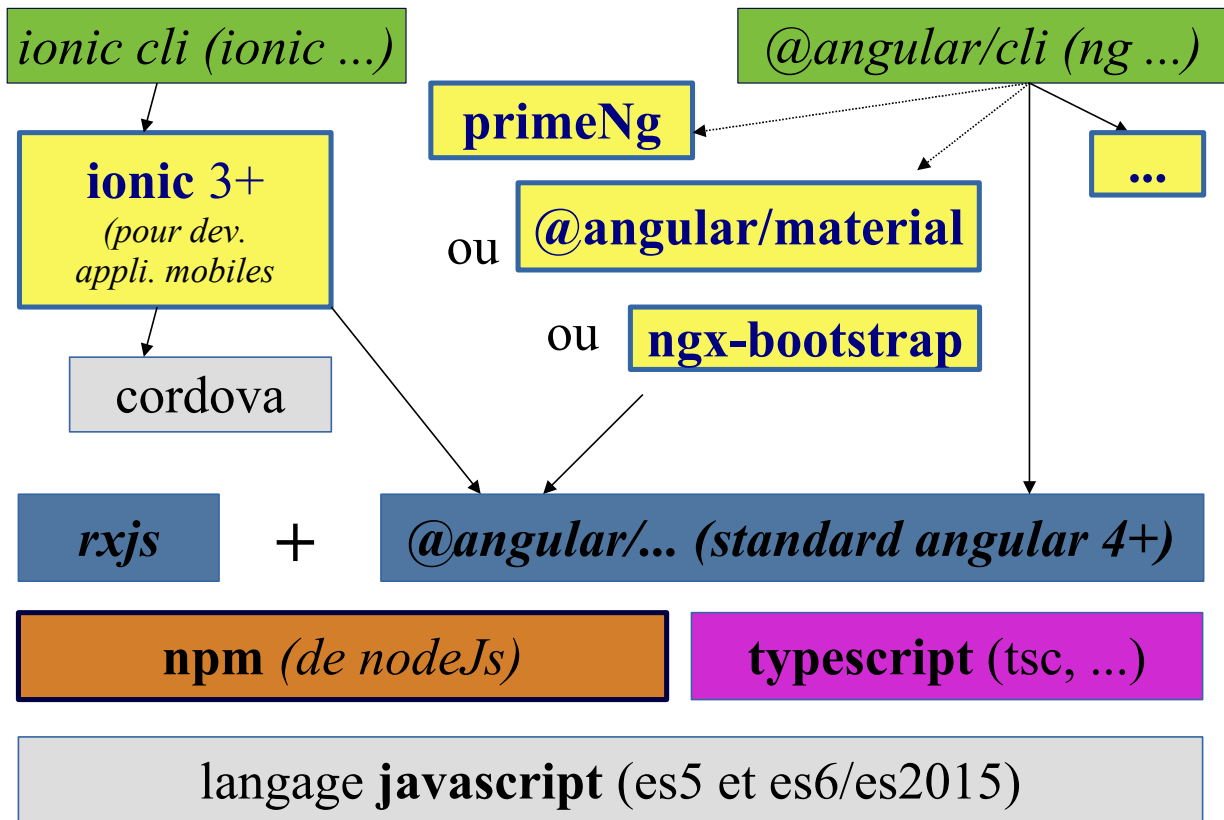
Référencer **sw-master.js** plutôt que le prédéfini **ngsw-worker.js** dans **app.module.ts** :

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

VIII - Ecosystème angular et extensions

1. Ecosystème angular (extensions)

Eventuelles extensions pour angular 4+



2. Différentiation "dev" et "prod" via environnement

Le répertoire `src/environments` d'une application angular comporte les fichiers suivants :

`environment.ts`

```
export const environment = {
  production: false
};
```

`environment.prod.ts`

```
export const environment = {
  production: true
};
```

Comme expliqué dans un commentaire placé dans le haut de `environment.ts` , **le contenu du fichier `environment.ts` est automatiquement remplacé par le contenu du fichier `environment.prod.ts` lorsque l'on génère une version de production de l'application angular via la commande `ng build --prod` .**

`angular.json`

```
....
"configurations": {
  "production": {
    "fileReplacements": [
      {
        "replace": "src/environments/environment.ts",
        "with": "src/environments/environment.prod.ts"
      }
    ],
  },
  ....
```

On peut donc :

1. agrandir de manière cohérente le contenu des ces 2 fichiers
2. importer le paquet de constantes un via un
`import { environment } from '../environments/environment';`

Exemple :

`src/environments/environment.ts`

```
export const environment = {
  production: false,
  application : { name="my-app" , default_username : "power_user" }
};
```

src/environments/environment.prod.ts

```
export const environment = {  
  production: true ,  
  application : { name ="myApp" , default_username : "guest" }  
};
```

src/app/xyz/xyz.component.ts

```
import { environment } from '../../environments/environment';  
  
@Component(...)  
export class XyzComponent implements OnInit {  
  
  name = environment.application.name;  
  
  ...  
  methodeZzz(){  
    if(environment.production){  
      //....  
    }else {  
      //....  
    }  
  }  
}
```

La valeur de name basculera alors automatiquement de "my_app" à "myApp" entre la phase de développement et la phase de production .

3. Introduction à "Angular-universal"

3.1. Objectif de "angular-universal"

Pour être visité par un grand nombre d'utilisateurs, un site web se doit de remplir deux conditions essentielles.

- La page d'accueil soit s'afficher le plus rapidement possible.
- L'application angular doit être bien référencé par les moteurs de recherche.

La technique qui permet de le faire porte un nom.

- Le **Rendu Côté Serveur** ou **Server Side Rendering (SSR)** en anglais.

La technologie **SSR** préconisée par les équipes de Google au niveau d'un projet angular est

- **Angular Universal.**

Cette technologie permettra d'améliorer le **référencement naturel** ou **SEO (Search Engine Optimization)** de notre site.

Les moteurs de recherche ont à l'heure actuelle du mal à interpréter le javascript.

Par défaut une application angular est de type "SPA (Single Page Application)" et "CSR" (Client Side Rendering) . Le titre principal de la page d'accueil est caché dans un bundle javascript produit par "ng build --prod" mais n'apparaît pas dans index.html .

--> On a donc besoin d'ajouter un mécanisme SSR pour obtenir un bon référencement .

On ne peut cependant pas remettre en cause tous la logique de fonctionnement "SPA/CSR" d'angular qui fonctionne très bien pour naviguer rapidement d'un sous composant à l'autre et qui offre un très bon comportement dynamique (réactions rapides).

"**Angular Universal**" est en fait une technologie dite d'**isomorphisme** :

Au lieu de n'utiliser qu'un rendu coté serveur , "Angular Universal" va servir à générer coté serveur une sorte de pré-rendu "HTML/CSS" immédiatement/rapidement affichable par un navigateur de la page principale , son entête , son pied de page et son sous composant d'accueil (ex : WelcomeComponent) . Suite à des enregistrements de "bookmarks/favoris" , angular universal peut éventuellement servir à générer un "pré-rendu" d'une partie spécifique (selon fin d'url et paramétrages du routing angular) .

Le "pré-rendu" SSR n'est qu'un point de départ de l'application (très rapidement téléchargé et affiché) . Les mécanismes "javascript/SPA/CSR" sont téléchargés et chargés en mémoire en tâche de fond (le temps que l'utilisateur passe à admirer la pré-rendu affiché) .

Par la suite, les mécanismes SPA/CSR prennent le relai et l'application angular fonctionne comme d'habitude en dynamique javascript coté client et sans SSR.

Ce "pré-rendu" SSR sera en tout point identique à celui qui serait générer en pur javascript par les mécanismes habituels CSR , on parle alors en terme d'**isomorphisme** : même forme prise par les rendus SSR et CSR .

3.2. Positionnement de angular-universal

La technologie angular-universal est à la fois :

- liée au projet angular (add-on)
- vouée à être exécutée coté serveur via **nodeJs/express**

3.3. intégration de angular universal

site officiel	https://github.com/angular/universal/releases
documentation	https://angular.io/guide/universal
ajout via angular-cli	ng add @nguniversal/express-engine

Au sein d'un projet existant angular , on lancera la commande suivante de façon à ajouter la fonctionnalité "angular-universal" :

ng add @nguniversal/express-engine

Cette commande va ajouter certains fichiers et modifier certains fichiers existants de l'application angular .

Voici à peu près (selon la version exacte d'angular), ce qu'automatise cette commande :

- Installation des nouvelles **dépendances** nécessaires
- Modification du fichier **main.ts**
- Modification du fichier **app.module.ts**
- Modification du fichier **angular.json**
- Création du fichier **src/app/app.server.module.ts**
- Création du fichier **src/main.server.ts**
- Création du fichier **server.ts**
- Création du fichier **tsconfig.server.json**
- Modification du fichier **angular.json**
- Modification du fichier **package.json**

Les nouvelles dépendances ajoutées sont (à peu près) les suivantes :

```
npm install --save @angular/platform-server
npm install --save @nguniversal/express-engine
npm install --save express
npm install --save @nguniversal/builders
npm install --save @types/express
```

et dans **package.json**

```
"dependencies": {
  "@angular/animations": "10.0.3",
  "@angular/common": "10.0.3",
  "@angular/compiler": "10.0.3",
  "@angular/core": "10.0.3",
```



```

"@angular/forms": "10.0.3",
"@angular/platform-browser": "10.0.3",
"@angular/platform-browser-dynamic": "10.0.3",
"@angular/platform-server": "10.0.3",
"@angular/platform-server": "10.0.3",
"@angular/router": "10.0.3",
"@fortawesome/fontawesome-free": "5.13.1",
"@nguniversal/express-engine": "10.0.1",
"bootstrap": "4.5.0",
"express": "4.17.1",
"rxjs": "6.6.0",
"tslib": "2.0.0",
"zone.js": "0.10.3"
},
"devDependencies": {
"@angular-devkit/build-angular": "0.1000.2",
"@angular/cli": "10.0.2",
"@angular/compiler-cli": "10.0.3",
"@nguniversal/builders": "10.0.1",
"@types/express": "4.17.7",
"@types/node": "14.0.22",
"@types/jasmine": "3.5.11",
"@types/jasminewd2": "2.0.8",
"codelyzer": "6.0.0",
"jasmine-core": "3.5.0",
"jasmine-spec-reporter": "5.0.2",
"karma": "5.1.0",
"karma-chrome-launcher": "3.1.0",
"karma-coverage-istanbul-reporter": "3.0.3",
"karma-jasmine": "3.3.1",
"karma-jasmine-html-reporter": "1.5.4",
"protractor": "7.0.0",
"ts-node": "8.10.2",
"tslint": "6.1.2",
"typescript": "3.9.6"
}

```

Autres modifications apportées au fichier **package.json**

```
"scripts": {
  ...
  "dev:ssr": "ng run myapp:serve-ssr",
  "serve:ssr": "node dist/myapp/server/main.js",
  "build:ssr": "ng build --prod && ng run myapp:server:production",
  "prerender": "ng run myapp:prerender"
  ...
}
```

Modifications apportées au sein de `src/main.ts` :

```
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();
}

document.addEventListener('DOMContentLoaded', () => {
  platformBrowserDynamic().bootstrapModule(AppModule)
    .catch(err => console.error(err));
});
```

Modifications apportées au sein de `src/app/app.module.ts` :

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

import { AppComponent } from './app.component';
import { HomeComponent } from './modules/general/home/home.component';
import { AppRoutingModule } from './app-routing.module';

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent
  ],
  imports: [
    BrowserModule.withServerTransition({ appId: 'serverApp' }),
    AppRoutingModule,
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Modifications apportées au sein de `angular.json` :

```
"builder": "@angular-devkit/build-angular:browser",
"options": {
  "outputPath": "dist/myapp/browser",
  ...
}
```

Les propriétés `"server"`, `"serve-ssr"` et `"prerender"` sont rajoutées après la propriété `e2e`.

```

"e2e": { "builder": "@angular-devkit/build-angular:protractor",
  "options": {
    "protractorConfig": "e2e/protractor.conf.js",
    "devServerTarget": "myapp:serve"
  },
  "configurations": {
    "production": {
      "devServerTarget": "myapp:serve:production"
    }
  }
},
"server": {
  "builder": "@angular-devkit/build-angular:server",
  "options": {
    "outputPath": "dist/myapp/server",
    "main": "server.ts",
    "tsConfig": "tsconfig.server.json"
  },
  "configurations": {
    "production": {
      "outputHashing": "media",
      "fileReplacements": [
        {
          "replace": "src/environments/environment.ts",
          "with": "src/environments/environment.prod.ts"
        }
      ]
    },
    "sourceMap": false,
    "optimization": true
  }
},
"serve-ssr": {
  "builder": "@nguniversal/builders:ssr-dev-server",
  "options": {
    "browserTarget": "myapp:build",
    "serverTarget": "myapp:server"
  },
  "configurations": {
    "production": {
      "browserTarget": "myapp:build:production",
      "serverTarget": "myapp:server:production"
    }
  }
},
"prerender": {
  "builder": "@nguniversal/builders:prerender",
  "options": {
    "browserTarget": "myapp:build:production",
    "serverTarget": "myapp:server:production",
    "routes": [
      "/"
    ]
  },
  "configurations": {
    "production": {}
  }
}
}

```

Création du nouveau fichier **src/app/app.server.module.ts**:

```
import { NgModule } from '@angular/core';
import { ServerModule } from '@angular/platform-server';

import { AppModule } from './app.module';
import { AppComponent } from './app.component';

@NgModule({
  imports: [
    AppModule,
    ServerModule,
  ],
  bootstrap: [AppComponent],
})
export class AppServerModule {}
```

Création du nouveau fichier **src/main.server.ts**:

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

import { environment } from './environments/environment';

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

export { AppServerModule } from './app/app.server.module';
export { renderModule, renderModuleFactory } from '@angular/platform-server';
```

Création du nouveau fichier fondamental **server.ts** (à la racine du projet angular):

```
import 'zone.js/dist/zone-node';

import { ngExpressEngine } from '@nguniversal/express-engine';
import * as express from 'express';
import { join } from 'path';

import { AppServerModule } from './src/main.server';
import { APP_BASE_HREF } from '@angular/common';
import { existsSync } from 'fs';

// The Express app is exported so that it can be used by serverless Functions.
export function app(): void {
  const server = express();
  const distFolder = join(process.cwd(), 'dist/myapp/browser');
  const indexHtml = existsSync(join(distFolder, 'index.original.html')) ?
    'index.original.html' : 'index';

  // Our Universal express-engine (found @ https://github.com/angular/universal/tree/master/modules/express-engine)
  server.engine('html', ngExpressEngine({
    bootstrap: AppServerModule,
  }));

  server.set('view engine', 'html');
  server.set('views', distFolder);

  // 1. Example Express Rest API endpoints (if in same project)
  // server.get('/api/**', (req, res) => { });
```

```

// 2. Serve static files from /browser
server.get('*..*', express.static(distFolder, {
  maxAge: '1y'
}));

// 3. All regular routes use the Universal engine
server.get('*', (req, res) => {
  res.render(indexHtml, { req,
    providers: [{ provide: APP_BASE_HREF,
      useValue: req.baseUrl }]
  });
});

return server;
}

function run(): void {
  const port = process.env.PORT || 4000; //default port may be changed

  // Start up the Node server
  const server = app();
  server.listen(port, () => {
    console.log(`Node Express server listening on http://localhost:${port}`);
  });
}

// Webpack will replace 'require' with '__webpack_require__'
// '__non_webpack_require__' is a proxy to Node 'require'
// The below code is to ensure that the server is run only when not requiring the bundle.
declare const __non_webpack_require__: NodeRequire;
const mainModule = __non_webpack_require__.main;
const moduleFilename = mainModule && mainModule.filename || '';
if (moduleFilename === __filename || moduleFilename.includes('iisnode')) {
  run();
}

export * from './src/main.server'; //export * means import and re-export !!!

```

Création du nouveau fichier **tsconfig.server.json**

```

/* To learn more about this file see: https://angular.io/config/tsconfig. */
{
  "extends": "./tsconfig.app.json",
  "compilerOptions": {
    "outDir": "./out-tsc/server",
    "target": "es2016",
    "types": [
      "node"
    ]
  },
  "files": [
    "src/main.server.ts",
    "server.ts"
  ],
  "angularCompilerOptions": {
    "entryModule": "./src/app/app.server.module#AppServerModule"
  }
}

```

Selon la complexité du projet, il se peut que quelques petites erreurs soient à corriger dans les portions de codes ajoutées ou modifiées via **"ng add @nguniversal/express-engine"**.

Serveur Http (Express)

- > Moteur de rendu (ex. platform-server ou ngExpressEngine)
- > Application Angular "serveur" (main.server.ts)
- > Module Angular "serveur" (app.server.module.ts)
- > Composant principal Angular (app.component.ts)

3.4. Utilisation de angular avec angular-universal intégré

Développement	<code>npm run start</code> ou bien <code>ng serve</code> <code>http://localhost:4200/</code>
Tests	<code>npm run lint</code> <code>npm run test</code> <code>npm run e2e</code>
Construction (AOT)	<code>npm run build</code>
SSR	<code>npm run build:ssr</code> <code>npm run serve:ssr</code> <code>http://localhost:4000/</code>

npm run build:ssr (alias npm pour "*ng build --prod && ng run myapp:server:production*")

génère la sous arborescence suivante :

dist/myapp/browser/index.html et ...js (partie générée par *ng build --prod*)

dist/myapp/server/main.js (partie générée par *ng run myapp:server:production*)

NB : le fichier **myapp/browser/index.html** (généré par l'habituel *ng build --prod*) comporte toujours `<app-root></app-root>` et doit être rendu/traité par **dist/myapp/server/main.js** .

npm run serve:ssr (alias npm pour **node** dist/universal-app/server/main.js)

lance l'interprétation de main.js via node

(en écoutant par défaut les requêtes sur **http://localhost:4000/**)

npm run prerender (alias npm pour *ng run universal-app:prerender*) lance automatiquement :

- un équivalent de **npm run build:ssr**
- un lancement très particulier (avec options cachées) de main.js qui :
 - * renomme index.html (sans pré-rendu) en **index.original.html**
 - * **génère une nouvelle version de index.html** (avec **pré-rendu SSR**)
 - * éventuellement générer (plus ou moins bien) de rendu de certaines routes
 - * s'arrête immédiatement (pas d'attente de requêtes sur localhost:4000)

NB : npm run prerender ne génère qu'un contenu "static" imparfait (selon api rest en backend plus ou moins accessible) . c'est moins bien que "npm run build:ssr" + "npm run serve :ssr"

NB :

après `npm run start` ou bien `ng serve`, la page HTML (en rendu CSR) au bout de l'url `http://localhost:4200/` est quasiment vide (simplement attachée à du code javascript incompréhensible par les humains et la plupart des moteurs de recherche) .

Par contre, après `npm run build:ssr` et `npm run serve:ssr`, la page HTML (en rendu CSR) au bout de l'url `http://localhost:4000/` est compréhensible par un moteur de recherche ou par un développeur souhaitant visualiser son contenu via une manipulation du type "click droit / code source de la page" .

Incidence du contexte "angular-universal"

Lorsque l'extension "angular_universal" est intégrée dans une application angular, on a maintenant deux fichiers de modules dans `src/app` : `"app.module.ts"` et `"app.server.module.ts"` .

Sans option, la création classique d'un nouveau composant ultérieur lève alors une ambiguïté de type *"More than one module matches. Use the skip-import option to skip importing the component into the closest module or use the module option to specify a module"*.

Pour lever cette ambiguïté, il faut utiliser l'option `--module app` .

```
ng g component xyz --module app
```

3.5. Pièges à éviter (avec certaines versions de angular universal)

- **Universal n'est pas capable d'utiliser les objets globaux** existants au runtime dans votre navigateur comme *document*, *window*, *localStorage*, etc.

NB : si l'on tient absolument à garder des instructions de type `localStorage.setItem(...)` suite à une utilisation indirecte de librairie qui fait ça en interne, on peut éventuellement utiliser le remède suivant (qui fonctionne / débloque la situation):

```
npm i localstorage-polyfill --save
et ajout de ces 2 lignes dans server.ts
```

```
import 'localstorage-polyfill'
global['localStorage'] = localStorage;
```

- De la même façon, **le DOM doit être uniquement manipulé par Angular**, et pas de façon externe (*jQuery*, *vanilla javascript*...), ce qui force à respecter les bonnes pratiques de développement Angular.

Si l'on souhaite connaître le contexte (CSR ou SSR) avant d'utiliser un objet global du navigateur (`document` ou `localStorage`, ...) on peut utiliser un bloc de code de ce genre :

```
import { Inject, PLATFORM_ID } from '@angular/core';
```

```
import { isPlatformBrowser } from '@angular/common';

class MyService {
  constructor(@Inject(PLATFORM_ID) private platformId: Object) {}

  store(key: string, value: string) {
    if(isPlatformBrowser(this.platformId)) {
      localStorage.setItem(key, value);
    }
  }
}
```

On peut utiliser la même technique pour différencier (un tout petit peu) le rendu SSR et CSR :

```
import { Component, Inject, OnInit, PLATFORM_ID } from "@angular/core";
import { isPlatformBrowser } from "@angular/common";

@Component({
  selector: "app-public-page",
  template: `<p>Rendered by {{ renderer }}</p>`,
  styleUrls: ["/public-page.component.scss"]
})
export class PublicPageComponent {
  renderer: string;

  constructor(@Inject(PLATFORM_ID) platformId: any) {
    this.renderer = isPlatformBrowser(platformId) ? "Browser" : "Server";
  }
}
```

Dans **app-routing.module.ts** (*principal*)

```
@NgModule({
  imports: [RouterModule.forRoot(routes, {
    initialNavigation: 'enabled'
  })],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Par contre dans **xyz-routing.module.ts** (*module(s) secondaire(s)*)

```
@NgModule({
  imports: [RouterModule.forRoot(routes, {
    initialNavigation: 'disabled'
  })],
  exports: [RouterModule]
})
export class XyzRoutingModule { }
```


Si vous utilisez les animations, vous devrez faire attention à les activer côté client seulement. Pour cela, le module d'animations BrowserAnimationsModule sera importé dans le module AppBrowserModule seulement, tandis que dans le module AppServerModule, vous importerez le module *NoopAnimationsModule* qui mock le module d'animations.

// *app.server.module.ts*

```
@NgModule({
  imports: [....,
    NoopAnimationsModule
  ], ....
})
export class AppServerModule { }
```

3.6. Paramétrage universel pour lazy module

Avant angular 9 , la syntaxe de *loadChildren* pour les "lazy-modules" était

```
loadChildren : './my-lazy/my-lazy.module#MyLazyModule'
```

et il fallait alors ajouter

```
...
import {ModuleMapLoaderModule} from '@nguniversal/module-map-ngfactory-loader';

@NgModule({
  imports: [
    ...,
    ModuleMapLoaderModule // <-- *Important* to have lazy-loaded routes work
  ],
  ...
})
export class AppServerModule {}
```

dans *app.server.module.ts* pour que angular-universal puisse gérer des routes vers des "lazy-modules" .

A partir de angular 9 , la nouvelle syntaxe de *loadChildren* pour les "lazy-modules" est

```
loadChildren: () => import('./my-lazy/my-lazy.module').then(m => m.MyLazyModule)}
```

et il ne faut rien ajouter de particulier dans *app.server.module.ts*; ça fonctionne bien sans ajout; les routes vers des "lazy-modules" sont automatiquement bien gérées .

3.7. Stratégies de mise en production (avec angular-universal)

Solution 1 déconseillée (pour minuscule projet):

ajouter/mélanger du code "api rest" rattaché à *server.ts* et déployer le tout sur une petite machine exécutant *node_js* en production .

Solution 2 (avec bonne séparation "frontEnd / backEnd"):

- *nodeJs1_angular_universal* en frontal (avec dans *server.ts* des

redirections de requêtes "api-rest" vers d'autres serveurs en arrière plan) .

- nodeJs2,3,n (ou SpringBoot ou ...) prenant en charge api rest
- ...

Solution 3, ... (selon contexte "cloud" ou autre)

variantes de la solution 2 avec en plus **nginx** (ou traefik ou ...) en frontal avec des configurations de type "reverse_proxy" vers :

- nodeJs1 angular_universal
- nodeJs2,3,n (ou SpringBoot ou ...) prenant en charge api rest
- ...

3.8. Eventuelle redirection d'api (via reverse proxy intégré à nodeJs/express) au sein de server.ts

npm install --save express-http-proxy

server.ts (de angular universal)

```
...
import * as expressHttpProxy from 'express-http-proxy';
...

// Example Express Rest API endpoints :
// server.get('/api/**', (req, res) => { });

// Example Express Rest API endpoints with proxy :
server.use('/xyz-api', expressHttpProxy('http://localhost:8282', {
  proxyReqPathResolver: function (req, res) {
    return '/xyz-api' + req.url;
  }
}));
...
```

ou bien

```
server.use('/*-api',
  expressHttpProxy('http://localhost:8282' , {
    proxyReqPathResolver: function (req, res) {
      let pos = req.originalUrl.lastIndexOf(req.url);
      let api_name = req.originalUrl.substring(0,pos);
      //console.log("api_name="+api_name);
      return api_name + req.url;
    }
  })
)
```

ou bien mieux encore avec une analyse d'un fichier de configuration tel que proxy.conf.json utilisé par ng serve avec l'option --proxy-config .

3.9. Angular (universal) SEO

De manière à optimiser le référencement naturel, on peut utiliser (par injection de dépendance), les services **Title** et **Meta** au niveau d'un composant angular (exemple : *HomeComponent* ou *WelcomeComponent* et/ou *AppComponent*).

```
import { Meta, Title } from '@angular/platform-browser';
...

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

  constructor(private title: Title,
               private meta: Meta) {}

  ngOnInit() {

    ....
    // SEO metadata
    this.title.setTitle(this.description);
    this.meta.addTag({name: 'description', content: this.longDescription});
  }
}
```

Certains tag peuvent correspondre à des "**media-crawler**" (descriptions pour "twitter" ou "...")

```
this.meta.addTag({name: 'twitter:title', content: this.description});
this.meta.addTag({name: 'twitter:image',
                  content: 'https://avatars3.githubusercontent.com/u/16628445?v=3&s=200'});
```

3.10. Autre Aide au référencement naturel

src/robots.txt

```
User-agent: *
Disallow:
Sitemap: https://xyz.com/sitemap.xml
```

src/sitemap.xml (à générer via <https://www.xml-sitemaps.com>)

dans angular.json

```
"options": {...
  "assets": [
    "src/favicon.ico",
    "src/assets",
    "src/manifest.webmanifest",
    "src/sitemap.xml",
    "src/robots.txt"
```

],

IX - Divers aspects avancés de angular

1. Divers Aspects avancés de Angular

1.1. concepts et terminologie "redux" du concurrent "react"

Avec le framework react sans redux l'état d'une application est complètement décentralisé à travers tous les composants de celle-ci et les mises à jour des states des principaux composants ne suivent pas d'ordre précis à cause du temps de réponse du serveur côté API, de l'asynchronicité, de la gestion des routes, etc.

Redux (version simplifiée de "flux") sert à **maintenir**, en arrière plan des "vues" de react, **un état global** à l'application bien cohérent et bien synchronisé entre tous les composants .

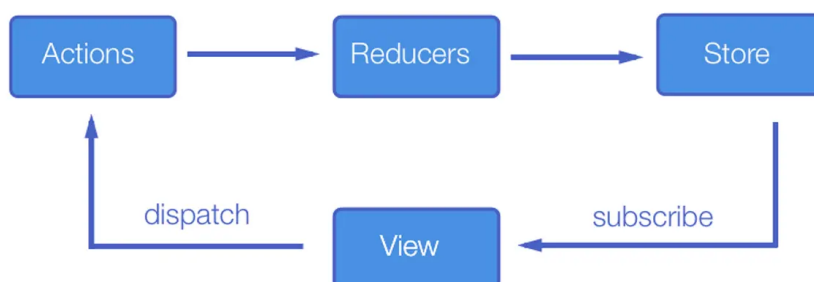
Éléments de redux	Rôles/caractéristiques
global state	objet principal (permettant de mémoriser l'état de toute l'appli) comportant de multiples sous objets <i>immutables</i> et <i>remplacés</i> .
action	objet " demande de traitement " permettant de modifier le "global state" (proche d'un objet " command " du design pattern de même nom)
reducer	objet technique sous forme de fonction pure permettant d'appliquer une action sur tout ou une partie d'un "global state existant" de façon à générer un "nouveau global state" ou une nouvelle sous-partie.
store	là où est stocké le "global state"

Le terme "*reducer*" vient du type de fonction que l'on peut appliquer à [Array.prototype.reduce\(\)](#).

Principales propriétés du global state :

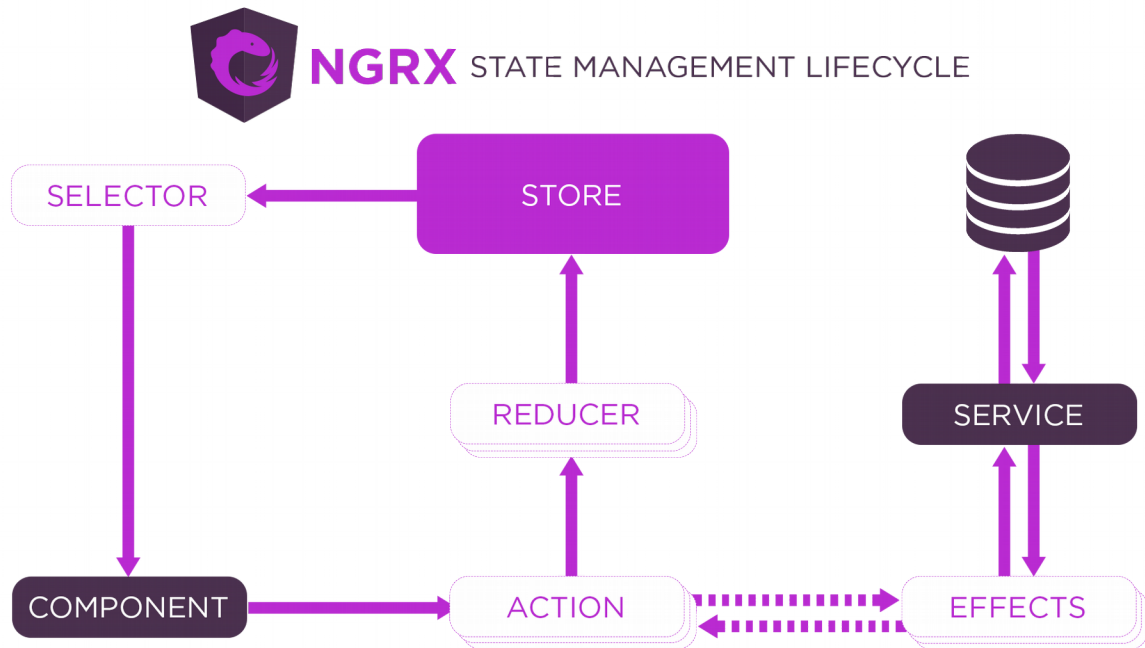
- unique (mais composé de plein de substates complémentaires : un par "reducer")
- en lecture seule (mais dont certaines parties sont remplacée par de nouvelles)

Redux



1.2. @ngrx/store (immutables , ...)

@ngrx/store est une extension permettant d'intégrer le concept de "état global maîtrisé" au sein d'une application angular .



- [Actions](#) describe unique events that are dispatched from components and services.
- State changes are handled by pure functions called [reducers](#) that take the current state and the latest action to compute a new state.
- [Selectors](#) are pure functions used to select, derive and compose pieces of state.
- State is accessed with the [Store](#), an observable of state and an observer of actions.

SHARI principe :

- **Shared**: state that is accessed by many components and services.
- **Hydrated**: state that is persisted and rehydrated from external storage.
- **Available**: state that needs to be available when re-entering routes.
- **Retrieved**: state that must be retrieved with a side-effect.
- **Impacted**: state that is impacted by actions from other sources.

installation de l'extension @ngrx/store :

```
npm install @ngrx/store --save
```

ou mieux encore

```
ng add @ngrx/store@latest
```

ajoutant **StoreModule.forRoot({})** au imports de **app.module.ts**

Exemple (proche du tutoriel officiel) :**counter.actions.ts**

```
import { createAction } from '@ngrx/store';
// just unique names :
export const increment = createAction('[Counter A_Scope] Increment');
export const decrement = createAction('[Counter A_Scope] Decrement');
export const reset = createAction('[Counter A_Scope] Reset');
```

pub.actions.ts

```
import { createAction } from '@ngrx/store';
export const disablePub = createAction('[Pub A_Scope] Disable');
export const enablePub = createAction('[Pub A_Scope] Enable');
export const togglePub = createAction('[Pub A_Scope] Toggle');
```

counter.reducer.ts

```
import { createReducer, on } from '@ngrx/store';
import { increment, decrement, reset } from './counter.actions';

export const initialState = 0;

// define reducer behavior :
const _counterReducer = createReducer(
  initialState,
  on(increment, (state) => state + 1),
  on(decrement, (state) => state - 1),
  on(reset, (state) => 0)
);
// apply reducer on state and action :
export function counterReducer(state, action) {
  return _counterReducer(state, action); //return new state
}
```

pub.reducer.ts

```
import { createReducer, on } from '@ngrx/store';
import { disablePub, enablePub, togglePub } from './pub.actions';

export const initialPub = false;

const _pubReducer = createReducer(
  initialPub,
  on(disablePub, (state) => false),
  on(enablePub, (state) => true),
  on(togglePub, (state) => !state)
);

export function pubReducer(state, action) { return _pubReducer(state, action);}
```

xyz.module.ts

```
...
import { StoreModule } from '@ngrx/store';
import { counterReducer } from './counter.reducer';
...
```

```

@NgModule({
  ...
  imports: [
    ...
    // bind "counterReducer" to "count" (sub-)part
    // and "pubReducer" to "pub" (sub-)part of global state in @ngrx/store
    StoreModule.forRoot({ count: counterReducer , pub : pubReducer })
  ]
})
export class XyzModule { }

```

time.counter.service.ts

```

import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class TimeCounterService {
  public getPeriodicTimeInfo() : Observable<any>{
    return Observable.create(observer => {
      const interval = setInterval(() => {
        observer.next(Date.now())
      }, 5000); //every 5000ms / 5s
    })
  }
  constructor() { }
}

```

xyz/my-counter/my-counter.component.css

```

.positif { color : blue; font-weight: bold;}
.negatif { color : red; font-style: italic;}

```

xyz/my-counter/my-counter.component.ts

```

import { Component, OnInit } from '@angular/core';
import { Store, select } from '@ngrx/store'; import { Observable } from 'rxjs';
import { increment, decrement, reset } from '../counter.actions';
import { disablePub, enablePub , togglePub } from '../pub.actions';
import { TimeCounterService } from '../time-counter.service';

@Component({
  selector: 'my-lazy2-my-counter', templateUrl: './my-counter.component.html',
  styleUrls: ['./my-counter.component.scss']
})
export class MyCounterComponent implements OnInit {
  count$: Observable<number>
  count_value : number;
  pub$: Observable<boolean>

  constructor(private store: Store<{ count: number ,
                                     pub : boolean }>,
    private timeCounterService : TimeCounterService) {

```



```

this.count$ = store.pipe(select('count'));
this.count$.subscribe((c)=>{this.count_value=c})
this.pub$ = store.pipe(select('pub'));

//reception reguliere de timeInfo toutes les 5s :
timeCounterService.getPeriodicTimeInfo().subscribe(
  (timeInfo)=>{ console.log(timeInfo + " " + new Date(timeInfo).toLocaleTimeString());
    this.store.dispatch(increment()); }
)
}

// dispatch actions onEvents :
onIncrement() { this.store.dispatch(increment()); }
onDecrement() { this.store.dispatch(decrement()); }
onReset() { this.store.dispatch(reset()); }
onDisablePub() { this.store.dispatch(disablePub()); }
onEnablePub() { this.store.dispatch(enablePub()); }
onTogglePub() { this.store.dispatch(togglePub()); }

ngOnInit(): void { }
setClasses(){ return {
  "positif" : this.count_value >= 0 ,
  "negatif" : this.count_value < 0    }
}
}

```

xyz/my-counter/my-counter.component.html

```

<p>my-counter with @ngrx/store</p>
<button (click)="onReset()">Reset Counter to 0</button> &nbsp;
<button (click)="onDecrement()">Decrement (--)</button> &nbsp;
Current Count:<span [ngClass]="setClasses()"> {{ count_value }}</span> &nbsp;
<button (click)="onIncrement()">Increment (++)</button>
<hr/>
<button (click)="onTogglePub()">TogglePub(On/Off)</button> &nbsp;
<button (click)="onDisablePub()">Disable (Off)</button> &nbsp;
Current Pub: <span>{{ pub$ | async }}</span> &nbsp;
<button (click)="onEnablePub()">Disable (On)</button>
<p *ngIf="(pub$ | async) == true">pub (advertisement)</p>

```

my-counter with @ngrx/store

Reset Counter to 0 Decrement (--) Current Count: **3** Increment (++)

TogglePub(On/Off) Disable (Off) Current Pub: true Disable (On)

pub (advertisement)

1.3. Redux ou équivalent utile avec angular ?

Avis "surtout pas de redux avec angular" :

- <https://www.developpez.net/forums/d2013063/javascript/bibliotheques-frameworks/surtout-redux-angular/>
- <https://www.journaldunet.com/solutions/dsi/1487396-surtout-pas-de-redux-avec-angular/>
- possibilité de se programmer assez facilement un mini store simplifié (mais opérationnel) en s'appuyant sur "service angular partagé" + "Observable" + " get /set sur le service" .

Avis "bonnes idées de redux avec angular" :

- l'existence même de @ngrx/store (et les arguments présents dans la documentation)
- possibilité de coder des "mini règles métiers" au sein des "reducers" de manière à garantir une cohérence de l'état global de l'application même en cas de "demandes de mise à jour concurrentes"
- pour les développeurs qui connaissent à la fois "react/redux" et "angular" @ngrx.store peut apporter un style de codage homogène .

ANNEXES

X - 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) 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" .

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

Pour les relativement anciennes versions d'angular (v5, ...) , il est nécessaire de :

1. télécharger "ngx-bootstrap" via "**npm install -s**"
2. ajuster le fichier angular.json (anciennement angular-cli.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/datepicker/bs-datepicker.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/datepicker';
import { CarouselModule } from 'ngx-bootstrap/carousel';
...
imports: [
  BrowserModule, FormsModule, HttpClientModule, AppRoutingModule, ChartsModule,
  BrowserAnimationsModule,
  TabsModule.forRoot(), BsDatepickerModule.forRoot(), CarouselModule.forRoot()
]
...
```

Avec une version récente d'angular (ex : v7 , v8) , la commande "ng add ..." convient parfaitement pour automatiser tout cela:

ng add ngx-bootstrap

puis (selon les besoins) :

ng add ngx-bootstrap --component *componentName*

exemple-of-componentName : **accordion** , **alerts** , **buttons** , ... , **tabs** , ...

1.3. Site de référence pour ngx-bootstrap

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

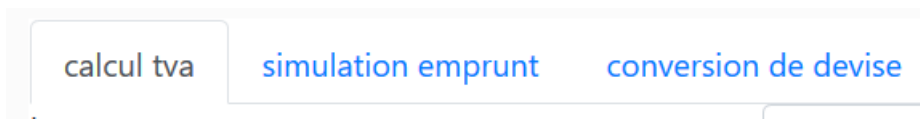
<https://valor-software.com/ngx-bootstrap/#/documentation>

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

1.4. Quelques composants de ngx-bootstrap

Onglets (tabs) :

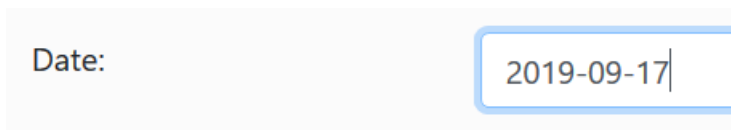
```
<tabset>
  <tab heading="calcul tva">
    <app-tva></app-tva>
  </tab>
  <tab heading="simulation emprunt">
    <app-simu-emprunt></app-simu-emprunt>
  </tab>
  <tab heading="conversion de devise">
    <app-conversion></app-conversion>
  </tab>
</tabset>
```

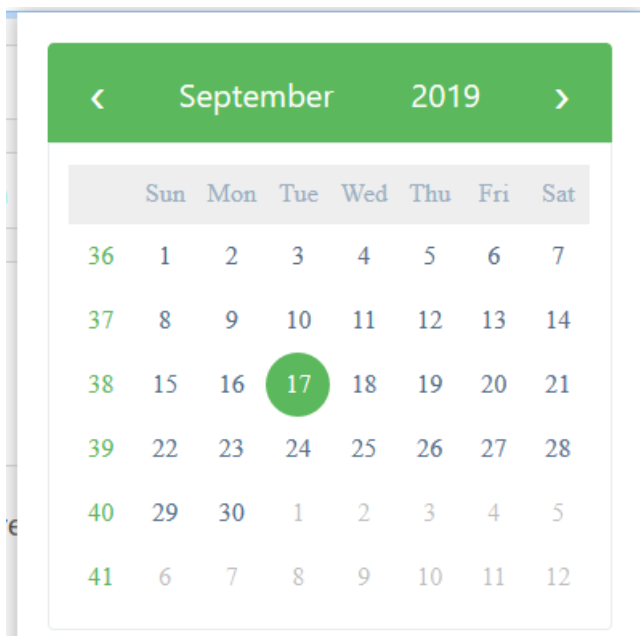


Calendrier javascript (datePicker) :

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

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





Carousel (slide-show , défilement d'images)

```
<carousel>
  <slide *ngFor="let cd of myCarrouselDefs">
    <!-- <a [routerLink]="[cd.path]" > -->
    <img [src]="cd.image" [alt]="cd.text"
      style="display: block; width: 100%;">
    <div class="carousel-caption d-none d-md-block">
      <h4>{{cd.text}}</h4>
    </div>
    <!-- </a> -->
  </slide>
</carousel>
```

```
export class WelcomeComponent implements OnInit {
  myCarrouselDefs=[
    { image : "assets/images/emprunt.jpg" , text : "tva, emprunts" , path:"/ngr/basic" },
    { image : "assets/images/achats.jpg", text : "achats" , path:"/ngr/browse-products" }
  ];
}
```

1.5. Exemples de composants personnalisés s'appuyant sur bootstrap-css

En plus des composants prédéfinis de "ngx-bootstrap", il est assez facile de mettre en oeuvre un paquet de composants personnalisés réutilisables basés sur bootstrap-css (et un peu sur ngx-bootstrap).

L'exemple suivant correspond à des extraits d'un module (nommé "*bs-util*") de composants réutilisables :

Dans **src/bs-util** (à côté de *src/app*) :

bs-util.module.ts

```
...
import { BsDropdownModule } from 'ngx-bootstrap/dropdown';
import { CollapseModule } from 'ngx-bootstrap/collapse';
import { ModalModule } from 'ngx-bootstrap/modal';
...
@NgModule({
  imports: [
    CommonModule, FormsModule, RouterModule, BrowserAnimationsModule,
    CollapseModule.forRoot(), BsDropdownModule.forRoot(), ModalModule.forRoot()
  ],
  exports: [
    BsuTogglePanelComponent,
    BsuMyFormGroupWithLabelComponent, BsuNavBarComponent,
    BsuOverviewCardComponent, BsuModalComponent
  ],
  declarations: [ BsuTogglePanelComponent,
    BsuMyFormGroupWithLabelComponent, BsuNavBarComponent, BsuNavItemComponent,
    BsuDropdownMenuComponent,
    BsuOverviewCardComponent, BsuModalComponent
  ]
})
export class BsUtilModule { }
```

Ce module pourra ensuite être globalement importé par un module applicatif :

```
import { BsUtilModule } from 'src/bs-util/bs-util.module';
...
@NgModule({
  ...
  imports: [
    BrowserModule, FormsModule, BsUtilModule
  ],
  ...})
export class AppModule { }
```

code complet au bout de l'URL suivante : <https://github.com/didier-mycontrib/angular8plus> (partie *ng-bs4-app/src/bs-util*).

2. 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 "....." .

2.1. 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 { }

```

3. 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

4. Quelques composants "angular-material"

Card (panneau d'encadrement) :



Basic

```
<mat-card class="my-card">
  <mat-card-header class="my-card-header">
    <mat-card-title>Basic</mat-card-title>
    <!-- <mat-card-subtitle>angular material</mat-card-subtitle> -->
  </mat-card-header>

  <mat-card-content class="my-card-content">
    ...
  </mat-card-content>
</mat-card>
```

```
.basic { background: white; }
.my-card { border: 1px; 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>

  <div>
    <label>kind: </label>
    <mat-radio-group placeholder="kind" name="kind" [(ngModel)]= "person.kind" >
      <mat-radio-button matInput *ngFor="let k of listeKind" [value]="k">{{k}}</mat-radio-button>
    </mat-radio-group>
    <!-- mat-radio-group cannot be put inside mat-form-field , ??? -->
  </div>

  <mat-form-field [appearance]=" 'standard' ">
    <mat-label>email:</mat-label>
    <input matInput name="email" type="email" placeholder="email" [(ngModel)]= "person.email" />
  </mat-form-field>

  <mat-checkbox name="crazy" [(ngModel)]= "person.crazy" >crazy (as checkbox)</mat-checkbox>
  <mat-slide-toggle name="crazy" [(ngModel)]= "person.crazy" >crazy (as slide-toggle)</mat-slide-toggle>
  <!-- mat-checkbox cannot be put inside mat-form-field , already has a label -->

  <mat-form-field [appearance]=" 'standard' ">
    <mat-label>birthday:</mat-label>
    <input matInput name="birthday" [matDatepicker]= "myDatePicker"
      placeholder="birthday" [(ngModel)]= "person.birthday" /> <!-- type="date" if no picker -->
    <mat-datepicker-toggle matSuffix [for]= "myDatePicker"></mat-datepicker-toggle>
    <mat-datepicker #myDatePicker></mat-datepicker>
  </mat-form-field>

  <mat-form-field [appearance]=" 'standard' ">
    <mat-label>childNumber:</mat-label>
    <input matInput name="childNumber" type="number" placeholder="childNumber" [(ngModel)]= "person.childNumber" />
  </mat-form-field>

  <mat-form-field [appearance]=" 'standard' ">
    <mat-label>preferedColor:</mat-label>
    <input matInput name="preferedColor" type="color" placeholder="preferedColor" [(ngModel)]= "person.preferedColor" />
  </mat-form-field>

  <mat-form-field [appearance]=" 'standard' ">
    <mat-label>password:</mat-label>
    <input matInput name="password" type="password" placeholder="password" [(ngModel)]= "person.password" />
  </mat-form-field>

  <mat-form-field [appearance]=" 'standard' ">
    <mat-label>country:</mat-label>
    <input matInput name="country" placeholder="country"
      (ngModelChange)= "adjustFilteredCountries()"
      [(ngModel)]= "person.country" [matAutocomplete]= "autoCountry" />
    <!-- for reactiveForm , [formControl]= "myControl" and no ngModel -->
    <mat-autocomplete #autoCountry= "matAutocomplete">
      <mat-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

Hi didier

What's your favorite animal?

Favorite Animal

Dog

No Thanks

Ok

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 *

didier

Next

☒ isLinear (step by step, cannot skip step)

1 Fill out your name — 2 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}}

XI - Annexe – socket.io

1. Socket.io

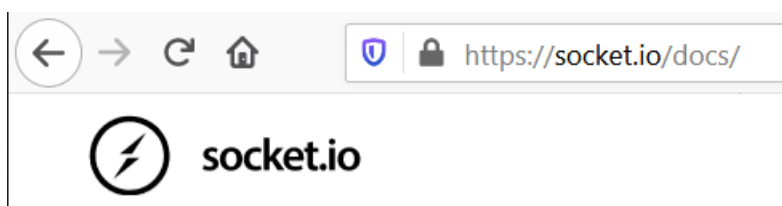
1.1. Présentation de Socket.io

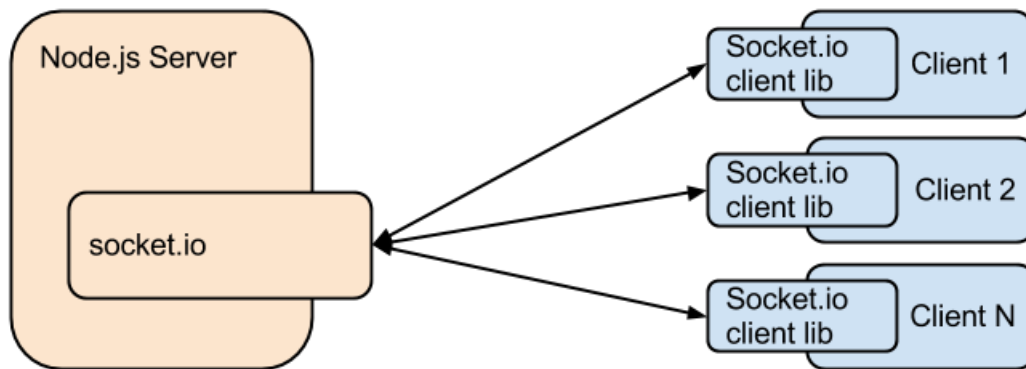
Socket.io est une **api javascript** qui encapsule et améliore la prise en charge des "**websockets**".

Rappel : les **websockets** sont une technologie permettant d'établir des **communications bidirectionnelles** via un **canal construit au dessus du protocole HTTP** et cela permet au coté serveur d'envoyer spontanément des informations (ou événements) au client (fonctionnant dans un navigateur). C'est une des technologies de "**push**".

Valeurs ajoutées 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 être établies même en présence de proxy http et de load-balancer (en mettant en oeuvre des mécanismes supplémentaires pour compenser les limitations des "websockets" généralement utilisées en interne)
ne gère pas le broadcasting	gère (si besoin) le broadcasting
pas d'option pour le "fallback"	comporte des options pour le "fallback"





documentation sur site officiel de Socket.io (présentation, concepts):

- <https://socket.io/docs/>

bibliothèque socket.io:

- <https://www.npmjs.com/package/socket.io-client>

- <https://www.npmjs.com/package/@types/socket.io-client>

1.2. chat-socket.io coté serveur (en version "typescript")

package.json

```
{
  "name": "chat-socket-io",
  "version": "0.1.0",
  "dependencies": {
    "ent": "^2.2.0",
    "express": "^4.17.1",
    "socket.io": "^2.2.0"
  },
  "description": "Chat temps réel avec socket.io",
  "devDependencies": {
    "@types/ent": "^2.2.1",
    "@types/express": "^4.17.0",
    "@types/socket.io": "^2.1.2"
  }
}
```

chat-socket-io/src/app.ts

```
import express, { Request, Response } from 'express';
import * as http from 'http';
import * as ent from 'ent'; // Permet de bloquer les caractères HTML
import * as sio from 'socket.io';

const app :express.Application = express();
const server = http.createServer(app);
const io = sio.listen(server);

//les routes en /html/... seront gérées par express
//par de simples renvois des fichiers statiques du répertoire "/html"
app.use('/html', express.static(__dirname+"/html"));

app.get('/', function(req :Request, res : Response ) {
  res.redirect('/html/index.html');
});

//events: connection, message , disconnect and custom_event like nouveau_client

io.sockets.on('connection', function (socket:any) {
  // Dès qu'on nous donne un pseudo,
  // on le stocke en variable de session/socket et on informe les autres personnes :
  socket.on('nouveau_client', function(pseudo:string) {
    pseudo = ent.encode(pseudo);
    socket.pseudo = pseudo;
    socket.broadcast.emit('nouveau_client', pseudo);
  });

  // Dès qu'on reçoit un message, on récupère le pseudo de son auteur
  //et on le transmet aux autres personnes
  socket.on('message', function (message:string) {
    message = ent.encode(message);
```

```

    socket.broadcast.emit('message', {pseudo: socket.pseudo, message: message});
  });
});

server.listen(8383,function () {
  console.log("http://localhost:8383");
});

```

1.3. chat-socket-io coté client (html/js)

chat-socket-io/dist/lib/socket.io.js (à télécharger)

chat-socket-io/dist/html/index.html

```

<html>
  <head>
    <meta charset="utf-8" />
    <title>Chat temps réel avec socket.io</title>
    <style>
      #zone_chat strong { color: white; background-color: black; padding: 2px; }
    </style>
  </head>

  <body>
    <h1>Chat temps réel avec socket.io (websocket)</h1>
    <p><i>(version adaptée de https://openclassrooms.com/fr/courses/1056721-des-applications-
    ultra-rapides-avec-node-js/1057959-tp-le-super-chat)</i></p>

    message:<input type="text" name="message" id="message" size="50" autofocus />
    <input type="button" id="envoi_message" value="Envoyer" />

    <div id="zone_chat"></div>

    <script src="lib/socket.io.js"></script>
    <script>
      var zoneChat = document.querySelector('#zone_chat');
      var zoneMessage = document.querySelector('#message');
      // Connexion au serveur socket.io :
      var socket = io.connect('http://localhost:8383');

      // On demande le pseudo, on l'envoie au serveur et on l'affiche dans le titre
      var pseudo = prompt('Quel est votre pseudo ?');
      socket.emit('nouveau_client', pseudo);
      document.title = pseudo + ' - ' + document.title;

      // Quand on reçoit un message, on l'insère dans la page
      socket.on('message', function(data) {
        insereMessage(data.pseudo, data.message)
      })

      // Quand un nouveau client se connecte, on affiche l'information :
      socket.on('nouveau_client', function(pseudo) {

```

```

zoneChat.innerHTML+=<p><em>' + pseudo + ' a rejoint le Chat !</em></p>
+zoneChat.innerHTML;

})

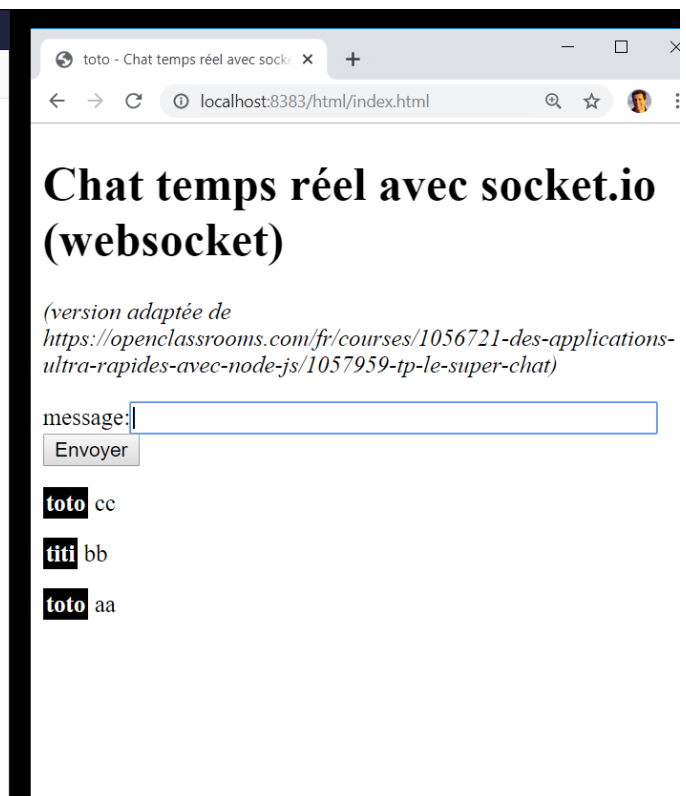
// Lorsqu'on click sur le bouton, on transmet le message et on l'affiche sur la page
document.querySelector('#envoi_message').addEventListener('click',function () {
    var message = zoneMessage.value;
    socket.emit('message', message); // Transmet le message aux autres
    insereMessage(pseudo, message); // Affiche le message aussi sur notre page
    zoneMessage.value=""; zoneMessage.focus(); // Vide la zone de Chat et remet le focus dessus
    return false; // Permet de bloquer l'envoi "classique" du formulaire
});

// fonction utilitaire pour ajouter un message dans la page :
function insereMessage(pseudo, message) {
    zoneChat.innerHTML+=<p><strong>' + pseudo + '</strong>' + message
    + '</p>'+zoneChat.innerHTML;
}
</script>
</body>
</html>

```

localhost:8383 indique

Quel est votre pseudo ?



1.4. Socket.io coté client (intégré dans Angular)

npm install socket.io-client --save

src/app/common/service/chat.service.ts

```
import { Injectable } from '@angular/core';
import * as sio from 'socket.io-client';
import { Observable } from 'rxjs';

@Injectable({ providedIn: 'root' })
export class ChatService {
  private url : string = 'http://localhost:8383';
  //https://github.com/didier-mycontrib/tp_node_js (chat-socket-io)

  private socket;

  constructor() {
    this.socket = sio(this.url);
  }

  public sendMessage(message:string, eventName:string="message") {
    this.socket.emit(eventName, message);
  }

  public getMessagesObservable(eventName:string="message") : Observable<any>{
    return Observable.create((observer) => {
      this.socket.on(eventName, (message) => {
        observer.next(message);
      });
    });
  }
}
```

chat.component.ts

```
import { Component, OnInit } from '@angular/core';
import { ChatService } from 'src/app/common/service/chat.service';

interface EventMessagewithPseudo{
  pseudo : string;
  message : string;
}

@Component({
  selector: 'app-chat',
  templateUrl: './chat.component.html',
  styleUrls: ['./chat.component.scss']
})
export class ChatComponent implements OnInit {
  pseudo:string;//undefined by default
  pseudoSent : boolean = false;
  newMessage: string;
  messages: EventMessagewithPseudo[] = [];
}
```

```

constructor(private chatService : ChatService) { }

onSetPseudo() {
  this.chatService.sendMessage(this.pseudo,"nouveau_client");
  this.pseudoSent = true;
}

onSendMessage() {
  this.chatService.sendMessage(this.newMessage); //envoi du message (pour diffusion)
  //push() ajoute à la fin, unshift() ajoute au début :
  this.messages.unshift({pseudo:this.pseudo ,
                           message:this.newMessage}); // pour affichage local du message envoyé
  this.newMessage = "";
}

ngOnInit() {
  this.chatService
    .getMessagesObservable("nouveau_client")
    .subscribe((username: string) => {
      this.messages.unshift( {pseudo:username , message:' a rejoint le Chat !'});
    });

  this.chatService
    .getMessagesObservable("message")
    .subscribe((evtMsgWithPseudo: any) => {
      this.messages.unshift(evtMsgWithPseudo);
    });
}

```

chat.component.html

```

<p>chat 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)="Sevent.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;}

```

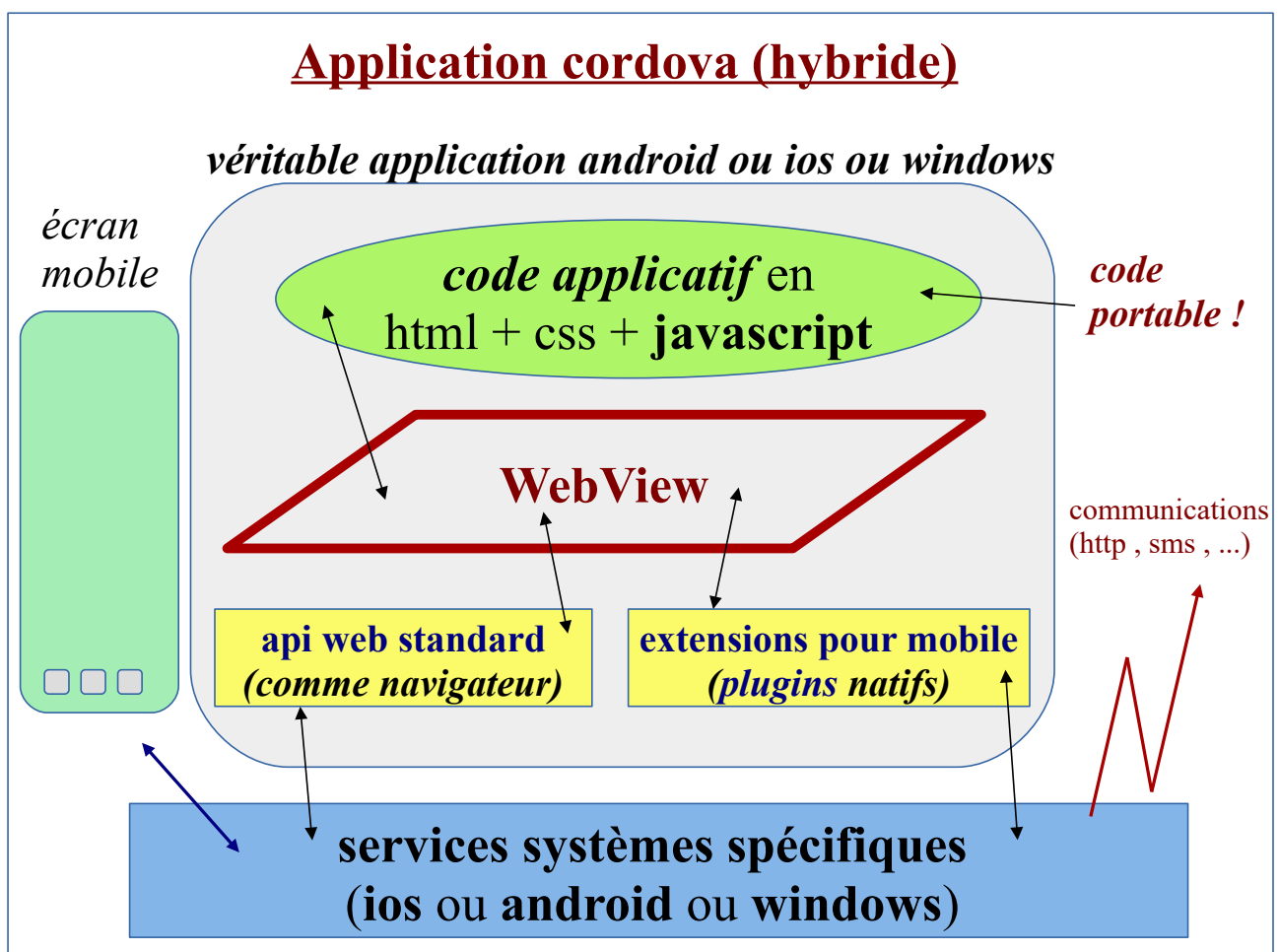

XII - Annexe – Ionic/cordova et NativeScript

1. Cordova (utilisé en interne par ionic)

Apache Cordova est une **technologie javascript** (s'appuyant sur npm) qui permet de **développer des applications mobiles supportées par différentes plateformes** : Android , Apple/Ios ,

Une **application cordova** est quelquefois qualifiée d'**hybride** (pas complètement native , pas complètement interprétée telle qu'une application web) .

Concrètement , une **application cordova** est **principalement codée comme une application web (HTML + CSS + javascript)** en s'appuyant sur des **api complémentaires javascript spécifiques pour indirectement contrôler certaines fonctionnalités d'un smartphone** .



Lors de la construction (build) , il est possible de générer des variantes de l'application pour différentes plateformes (android , Ios, ...).

L'**application construite** est une **réelle application (android ou Ios ou ...)** qui **incorpore un composant spécifique "WebView"** permettant d'exécuter le code "**javascript**" de l'application ainsi que le rendu **HTML+CSS** .

Grâce à une collection de **plugins** abstraits , la **"WebView"** cordova permet de indirectement **contrôler les fonctionnalités du smartphone** (camera , vibreur , ...) .

Apache cordova permet finalement de **développer avec les technologies web standard (HTML + CSS + javascript) des applications mobiles (assez portables) qui s'installent et s'utilisent comme des applications natives android ou ios ou windows** .

Sur certains points (vidéos , ...) , une **application cordova sera moins rapide (moins performante) qu'une application native** (spécialement conçue pour une plateforme spécifique) mais pour tous un tas d'applications standards (applications de gestion, ...) , les performances d'une application cordova seront très convenables .

L'ancien nom de **Cordova** était **PhoneGap** (et quelquefois "phone Callback")

2. Présentation de ionic

ionic est un **framework de haut niveau** permettant de **développer des applications mobiles en mode hybride** (en partie HTML/CSS/JS) .

ionic a fait le choix de s'appuyer sur les frameworks suivants :

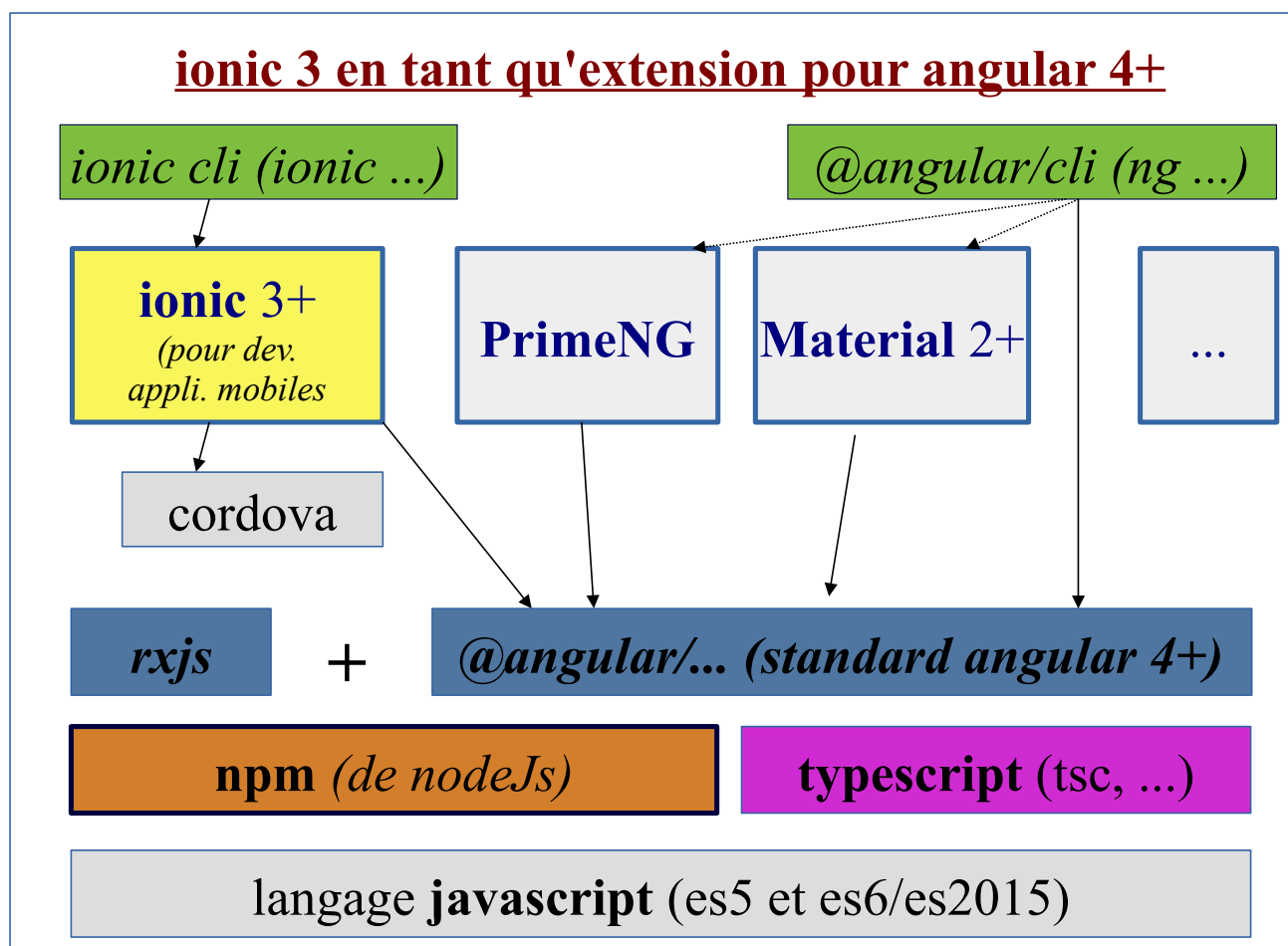
- **Angular** (v1, v2, v>=4)
- **Cordova**
- ...

ionic 1.x s'appuyait sur **AngularJs (1.x)**

ionic 2 (qui n'a seulement existé en version bêta) s'est appuyé sur l'éphémère **angular 2**

ionic 3 (ou +) s'appuie maintenant sur **Angular >=4** (enfin mature)

Concrètement , d'un point de vue développement , ionic comporte un utilitaire en ligne de commande "**ionic CLI**" (de la même manière qu' **angular-CLI** (ng ...) et **cordova CLI** (cordova ...) .



incontournable ionic CLI

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

```
ionic start my-ionic-app tabs -- création d'une nouvelle appli ionic
                                selon un template (tabs ou blank ou autre)
ionic g component cxy -- génération d'un nouveau (sous-)composant
ionic g page pl -- génération d'un composant de type "page ionic"
ionic g service sa -- génération d'un nouveau service
ionic g provider pxy -- génération d'un service d'accès à des données
                        distantes via HTTP (rest Api)

ionic g ...
ionic serve -l -- build + démarrage serveur de test
                (http://localhost:8100/ionic-lab)

ionic cordova build android -prod -- construction de l'application
                                   mobile (via cordova)

ionic ...
```

Avertissement :

La suite de ce support de cours n'exposera que les aspects spécifiques à ionic 3+ .

Les principaux aspects généraux de angular 4+ sont supposés être déjà connus.

Les principaux éléments de cordova sont également supposés avoir été déjà abordés.

Ce support de cours correspond à une initiation à ionic 3+.

A futur approfondissement via le site de référence de ionic est implicitement nécessaire pour acquérir plein de détails sur ce framework.

3. Installation de Ionic

ionic s'appuie sur nodeJs (et sa sous partie npm) qui doivent si besoin être préalablement installés.

3.1. installation de ionic CLI

```
(sudo) npm install -g ionic@latest
```

verification de l'installation :

```
ionic info
```

```
cli packages: (C:\Users\didier\AppData\Roaming\npm\node_modules)
  @ionic/cli-utils : 1.19.0
  ionic (Ionic CLI) : 3.19.0
```

System:

```
Node : v6.11.2
npm   : 3.10.10
OS    : Windows 10
```

...

3.2. installation des technologies complémentaires

installation (si nécessaire) d'une version récente de cordova

```
(sudo) npm install -g cordova
```

ainsi que **toutes les technologies de build** spécifique à une (ou plusieurs) plateforme(s) :
android , ios , ...

--> se référer aux dépendances techniques des constructions cordova .

4. Utilisation pratique de Ionic

4.1. Création d'une nouvelle application ionic

```
ionic start ionic-app-name template-name
```

Liste des **templates** disponibles :

- **tabs** : a simple 3 tab layout
- **sidemenu**: a layout with a swipable menu on the side
- **blank**: a bare starter with a single page
- **super**: starter project with over 14 ready to use page designs
- **tutorial**: a guided starter project

exemples:

```
ionic start my-ionic3-angular4-app blank
```

```
ionic start my-ionic-app tabs
```

Cette commande est interactive .

-répondre **yes** à "utiliser cordova"

-répondre **no** (or yes) à ionic PRO (sachant qu'un compte/login est nécessaire pour ionic PRO)

La structure de l'application construite est très proche d'une application angular >4 classique (construite via ng new) .Le fichier **package.json** ressemble à celui-ci :

```
{
  "name": "my-ionic-app", "version": "0.0.1",
  "author": "Ionic Framework",
  "homepage": "http://ionicframework.com/", "private": true,
  "scripts": {
    "clean": "ionic-app-scripts clean",
    "build": "ionic-app-scripts build",
    "lint": "ionic-app-scripts lint",
    "ionic:build": "ionic-app-scripts build",
    "ionic:serve": "ionic-app-scripts serve"
  },
  "dependencies": {
    "@angular/common": "5.0.3",
    "@angular/compiler": "5.0.3",
    "@angular/compiler-cli": "5.0.3",
    "@angular/core": "5.0.3",
    "@angular/forms": "5.0.3",
    "@angular/http": "5.0.3",
    "@angular/platform-browser": "5.0.3",
    "@angular/platform-browser-dynamic": "5.0.3",
    "@ionic-native/core": "4.4.0",
    "@ionic-native/splash-screen": "4.4.0",
    "@ionic-native/status-bar": "4.4.0",
    "@ionic/storage": "2.1.3",
    "ionic-angular": "3.9.2",
    "ionicons": "3.0.0",
    "rxjs": "5.5.2",
```

```

"sw-toolbox": "3.6.0",
"zone.js": "0.8.18"
},
"devDependencies": {
  "@ionic/app-scripts": "3.1.6",
  "typescript": "2.4.2"
},
"description": "An Ionic project"
}

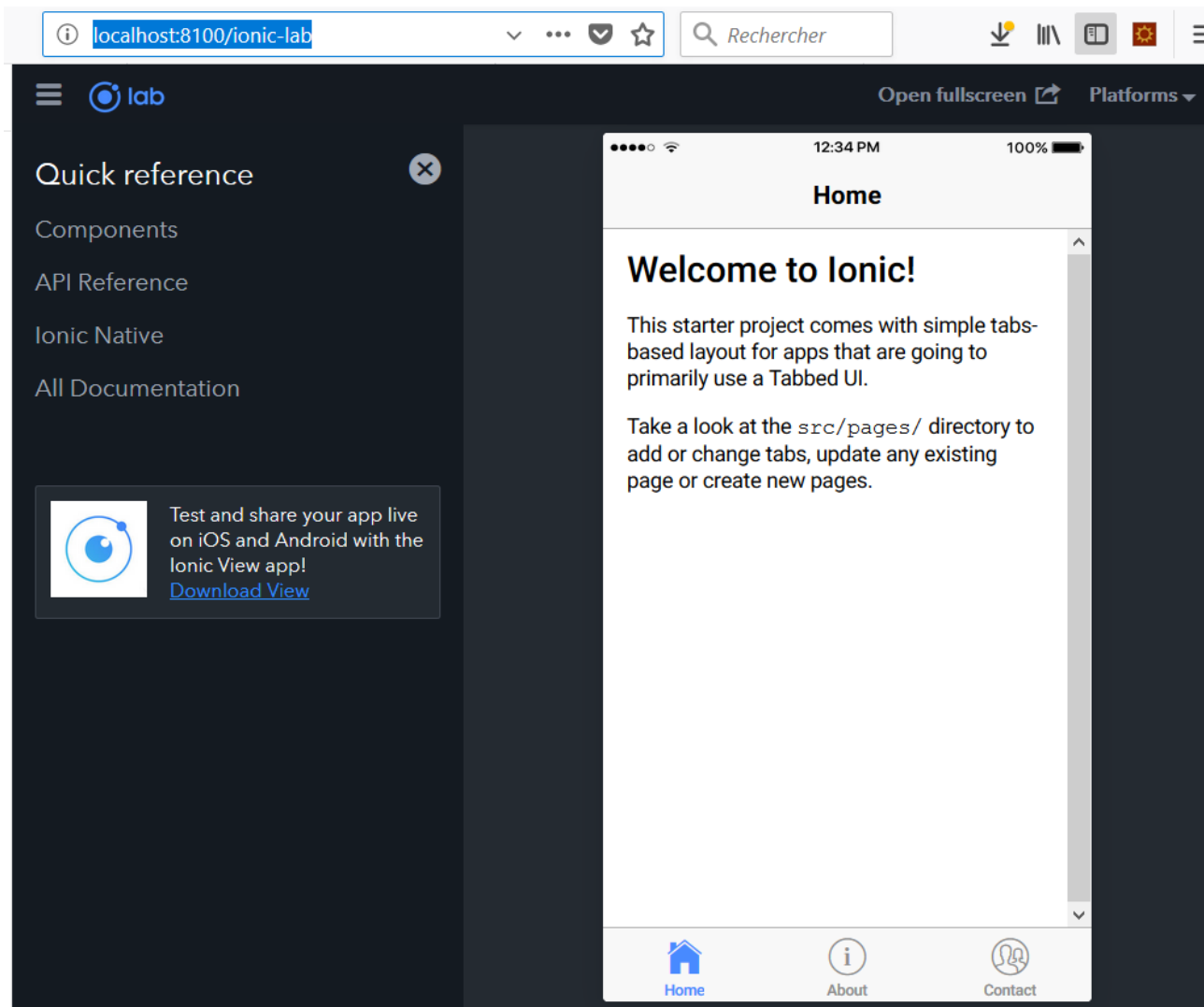
```

4.2. Test de l'application via ionic-lab

cd my-ionic-app

ionic serve -l

Cette commande lance une **construction de l'application**, un **déploiement de celle-ci sur un serveur de test** et un navigateur connecté sur l'URL suivante : <http://localhost:8100/ionic-lab>



4.3. Construction de l'application mobile

```
ionic cordova build ios -prod
```

```
ionic cordova build android -prod
```

NB :

l'application android construite (*app-debug.apk*) est générée dans le répertoire

my-ionic-app\platforms\android\app\build\outputs\apk\debug

--> ce fichier peut éventuellement être directement recopié sur un smartphone (ou tablette) android via un câble USB (ou autre) de façon à être testé sans émulateur .

4.4. Test de l'application sur un émulateur de plateforme mobile

si nécessaire (pour s'adapter à une version récente de cordova) :

```
ionic cordova rm android  
ionic cordova add android  
ionic cordova rm ios  
ionic cordova add ios
```

```
ionic cordova run ios -prod
```

```
ionic cordova run android -prod
```

cette commande :

- installe et configure si nécessaire cordova-android
- construit (via gradle) l'application android
- lance un émulateur android pour tester l'application

Autre solution : utiliser directement les commandes **emulator** et **adb** de android pour tester le fichier ".apk" généré .

5. Structure d'une application ionic

Une application ionic est assez souvent structurée comme un **ensemble de composants "page"** (onglets ou autres) .

5.1. Module principal "ionic"

app.module.ts

```
import { NgModule, ErrorHandler } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { IonicApp, IonicModule, IonicErrorHandler } from 'ionic-angular';
import { MyApp } from './app.component';

import { AboutPage } from '../pages/about/about';
import { ContactPage } from '../pages/contact/contact';
import { HomePage } from '../pages/home/home';
import { TabsPage } from '../pages/tabs/tabs';

import { StatusBar } from '@ionic-native/status-bar';
import { SplashScreen } from '@ionic-native/splash-screen';
import { ConversionPage } from '../pages/conversion/conversion';
import { ChangeProvider } from '../providers/change/change';
import '../providers/rxjs-extensions';
import { HttpModule } from '@angular/http';
import { FormsModule } from '@angular/forms';
import { Geolocation } from '@ionic-native/geolocation';
import { ChartPage } from '../pages/chart/chart';
import { ChartsModule } from 'ng2-charts';

@NgModule({
  declarations: [
    MyApp, AboutPage, ContactPage, ConversionPage, ChartPage, HomePage, TabsPage
  ],
  imports: [
    BrowserModule, HttpModule, FormsModule, ChartsModule,
    IonicModule.forRoot(MyApp)
  ],
  bootstrap: [IonicApp],
  entryComponents: [
    MyApp, AboutPage, ContactPage, ConversionPage, ChartPage, HomePage, TabsPage
  ],
  providers: [
    StatusBar, SplashScreen,
    Geolocation,
    {provide: ErrorHandler, useClass: IonicErrorHandler},
    ChangeProvider
  ]
})
export class AppModule {}
```


5.2. Composant principal "app" d'une appli "ionic / tabs"

app.html

```
<ion-nav [root]="rootPage"></ion-nav>
```

app.component.ts

```
import { Component } from '@angular/core';
import { Platform } from 'ionic-angular';
import { StatusBar } from '@ionic-native/status-bar';
import { SplashScreen } from '@ionic-native/splash-screen';

import { TabsPage } from '../pages/tabs/tabs';

@Component({
  templateUrl: 'app.html'
})
export class MyApp {
  rootPage:any = TabsPage;

  constructor(platform: Platform, statusBar: StatusBar, splashScreen: SplashScreen) {
    platform.ready().then(() => {
      // Okay, so the platform is ready and our plugins are available.
      // Here you can do any higher level native things you might need.
      statusBar.styleDefault();
      splashScreen.hide();
    });
  }
}
```

5.3. Série d'onglets "tabs"

pages/tabs/tabs.html

```
<ion-tabs>
  <ion-tab [root]="tab1Root" tabTitle="Home" tabIcon="home"></ion-tab>
  <ion-tab [root]="tab2Root" tabTitle="About/GPS" tabIcon="information-circle"></ion-tab>
  <ion-tab [root]="tab3Root" tabTitle="Contact" tabIcon="contacts"></ion-tab>
  <ion-tab [root]="tab4Root" tabTitle="Conversion" tabIcon="information-circle"></ion-tab>
  <ion-tab [root]="tab5Root" tabTitle="Chart" tabIcon="information-circle"></ion-tab>
</ion-tabs>
```

pages/tabs/tabs.ts

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

import { AboutPage } from '../about/about';
import { ContactPage } from '../contact/contact';
import { HomePage } from '../home/home';
import { ConversionPage } from '../conversion/conversion';
import { ChartPage } from '../chart/chart';

@Component({
```

```

    templateUrl: 'tabs.html'
  })
  export class TabsPage {

    tab1Root = HomePage;
    tab2Root = AboutPage;
    tab3Root = ContactPage;
    tab4Root = ConversionPage;
    tab5Root = ChartPage;

    constructor() {

    }
  }
}

```

5.4. Exemple de navigation vers une autre page ou onglet :

```

import { Component } from '@angular/core';
import { NavController } from 'ionic-angular';

@Component({
  selector: 'page-home',
  templateUrl: 'home.html'
})
export class HomePage {

  constructor(public navCtrl: NavController) {

  }

  onVersXyPage(){
    //this.navCtrl.setRoot(XyPage); //navig ok but not active
    //this.navCtrl.push(XyPage); //navig ok but should be pop/back
    this.navCtrl.parent.select(3); //position 3 + 1
  }

}

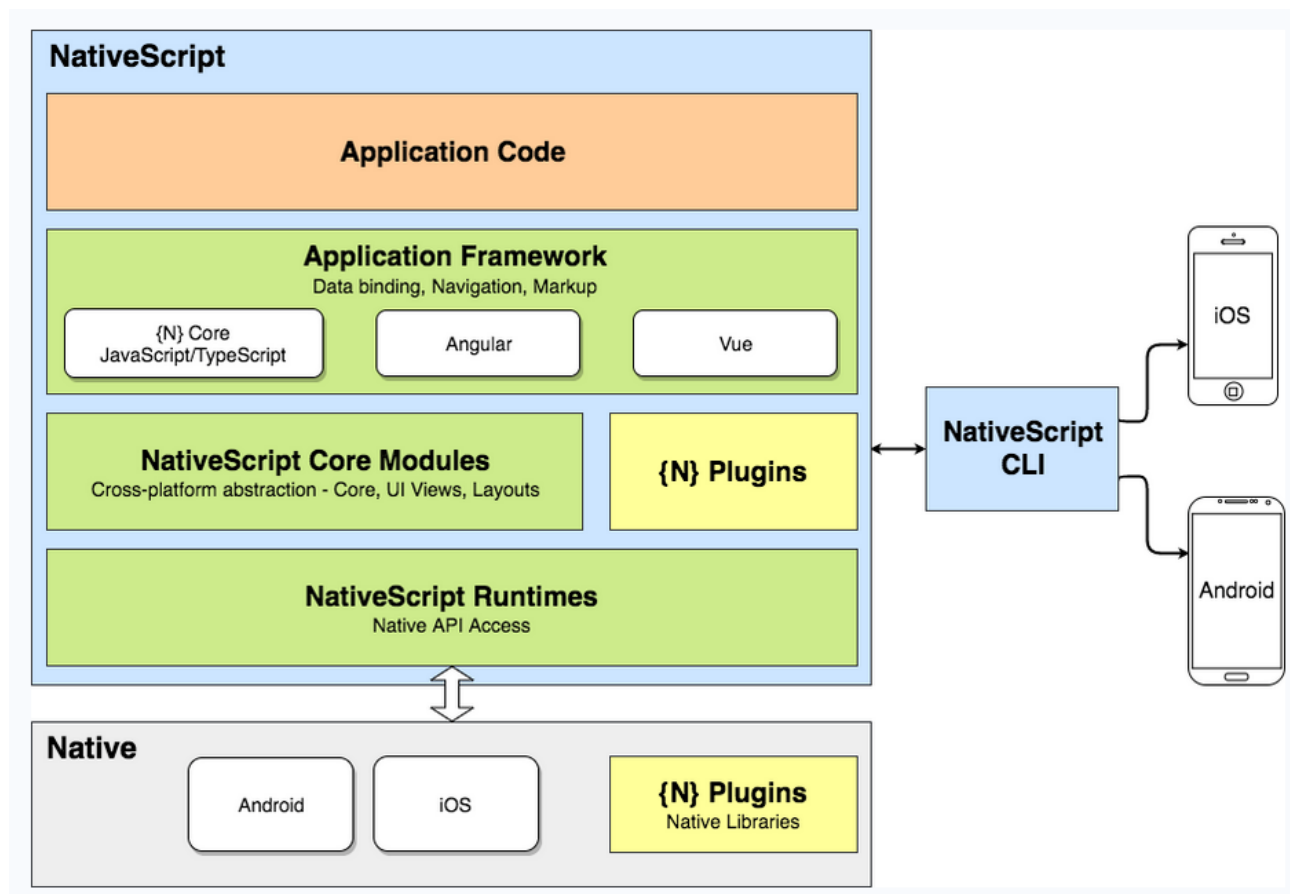
```

6. NativeScript angular (pour applications mobiles)

6.1. Présentation de NativeScript

L'extension **NativeScript** (en mode "*intégration angular*") est un *concurrent direct de ionic* et permet de développer des applications mobiles .

Contrairement à Ionic/Cordova , **NativeScript** n'utilise pas WebView (navigateur interne / caché) mais *utilise simplement une machine virtuelle javascript pour déclencher des instructions quasi natives du smartphone* (android ou iphone).



NB : NativeScript (utilisable en pure js , typescript , avec vue-js ou avec angular) est un concurrent direct de "react-native" et fonctionne à peu près sur les mêmes principes . Les performances sont à peu près comparables (et en général meilleures que celles de ionic/cordova).

A court terme, l'extension NativeScript est néanmoins beaucoup moins utilisée (communauté de développeurs beaucoup plus petite que celles des concurrent "ionic" et "react-native").

Finalement, l'entreprise "Google" a récemment proposé un framework concurrent "*flutter*" permettant de développer des applications mobiles quasi natives à base de composants basés sur le langage de programmation "DART" .

6.2. Type de code avec NativeScript Angular

- utilisation de styles css (transformés en paramétrages natifs "android" ou "ios/iphone")
- pas de HTML/DOM mais utilisation de composants abstraits spécifiques (ex : Label , TextView ,) transposés en interne en manipulation de composants natifs du smartphone (Similitude de fonctionnement avec java/awt , design-pattern "bridge/pont").
- utilisation heureusement possible des principales fonctionnalités d'angular ([(ngModel)] , routing , ...)

Exemple de template (pas vraiment .html) au sein d'une application NativeScript :

```
<ActionBar title="Details" class="action-bar"></ActionBar>
<FlexboxLayout flexDirection="column" class="page">
  <FlexboxLayout class="m-15">
    <Label class="h2" [text]="item.id + '. '"></Label>
    <Label class="h2" [text]="item.name"></Label>
  </FlexboxLayout>
  <Label class="h4" [text]="item.role"></Label>
</FlexboxLayout>
```

6.3. Premiers pas avec nativescript angular

`npm install -g nativescript@latest` : pas encore bien prêt pour angular 10 (quelques bugs) en septembre 2020 .

`npm install -g nativescript@6.1` (stable avec angular 8)
pour installer nativescript-cli (ns ou tns)

`tns create my-nsapp --ng`
pour créer une nouvelle application "my-nsapp" basé sur angular (--ng)

`cd my-nsapp`
`tns preview`

NB : pour faire fonctionner en mode "preview" l'application sur un mobile android il faut :

- installer (via le playStore) les applications "NativeScript Preview" et "NativeScript Playground"
- lancer l'application *NativeScript Playground* et scanner le *QR Code* affiché sur le PC dans le terminal où `tns preview` a été lancé .

Ceci devrait permettre (si tout va bien) , via un serveur de "NativeScript" en arrière plan, de

- récupérer le code de l'application
- le transférer sur le smartphone
- lancer l'application "NativeScript Preview" avec le code téléchargé

NB : Le code par défaut de l'application exemple affiche une liste de joueurs de foot .

Pour approfondir le sujet :

- <https://nativescript.org/nativescript-is-how-you-build-native-mobile-apps-with-angular/>
- <https://www.tektutorialshub.com/nativescript/nativescript-helloworld-example-app-with-angular/>