Tutoriel sur une introduction au framework web Angular

Par iner dukoid

Date de publication : 1 janvier 2021

Dernière mise à jour : 1 février 2021

Pour réagir au contenu de ce tutoriel, un espace de dialogue vous est proposé sur le forum.
Commentez

I - Mémo Angular....7
II - Objectif de ce tutoriel....7
III - Angular....7
IV - Les types d'applications que l'on peut créer....7
V - Les technologies qui composent le framework....7
VI - L'ensemble des technologies qui gravitent autour d'Angular....9
VII - Fonctionnement du framework....10
VIII - Angular : ce qu'il faut pour démarrer....10
IX - Quelques outils pour développer....11
X - Description des fichiers d'un projet Angular....12
X-A - Description du dossier angular-skeleton1....12
X-B - Plus qu'à lancer :....13
X-C - Le point de départ....13
X-D - Principe....14
X-E - Conclusion....14
X-E-1 - Schéma....14
X-F - Remarques....14
XI - Théorie sur les modules....15
XI-A - À savoir....15
XI-A-1 - Description d'un module Angular....15
XI-B - Exemple....15
XI-B-1 - Schéma....16
XI-B-2 - Remarques....16
XI-B-3 - Conclusion....17
XI-C - Remarques générales....17
XII - Les composants web....17
XII-A - Pratique....17
XII-A-1 - Schéma....18
XII-A-2 - Remarques....18
XII-B - Description....18
XII-B-1 - Allons voir :....18
XII-B-2 - Remarques....19
XII-C - Allons un peu plus loin avec ce composant....19
XII-C-1 - À savoir....19
XII-D - Le modèle de données....19
XII-D-1 - À savoir....19
XII-E - Les services....19
XII-E-1 - Pratique....19
XII-E-2 - Remarques....22
XII-E-3 - Résultat....22
XII-F - Conclusion....22
XIII - Création du 1er module et de ses composants....22
XIII-A - Exemple : créer un nouveau module et lui attacher deux composants....23
XIII-B - Pratique....23
XIII-B-1 - Création du nouveau module....23
XIII-B-2 - Allons voir :....24
XIII-B-3 - Ensuite, créons les deux composants....24
XIII-B-4 - Allons voir....24
XIII-B-5 - Résultat....24
XIII-B-6 - Conseils....25
XIII-B-7 - Remarques....26
XIII-B-8 - Voyons le résultat à l'écran :....26
XIII-C - Conclusion....26
XIV - Communication entre les composants....26
XIV-A - Les différentes techniques....26
XIV-B - Two-way Data Binding....26
XIV-B-1 - Pratique....26
XIV-B-2 - On obtient ceci à l'écran....29

XIV-B-3 - Conclusion 29
XIV-C - Communication par service (technique par observable ou par variable) 29
XIV-C-1 - Schéma 29
XIV-C-1-a - Principes 30
XIV-C-2 - Pratique 30
XIV-C-2-a - Remarques générales 31
XIV-C-2-b - comp1.component 32
XIV-C-2-c - comp2.component 33
XIV-C-2-d - comp3.component 34
XIV-C-3 - Résultat 35
XIV-C-4 - À savoir 35
XIV-C-5 - Conclusion 36
XV - Accès au composant enfant 36
XV-A - Pratique 36
XV-B - Remarques 38
XV-C - Conclusion 38
XVI - API WEB (récupération de données) et les fichiers d'environnement (DEV et PROD) 38
XVI-A - Environnements DEV et PROD 39
XVI-A-1 - À savoir 39
XVI-A-2 - Pratique 39
XVI-A-3 - Remarques 40
XVI-B - Le modèle de données 40
XVI-C - Remarque 40
XVI-D - Le service 40
XVI-E - Composant 41
XVI-E-1 - Remarques 42
XVI-E-2 - Remarques 44
XVI-E-3 - Remarques 45
XVI-F - Résultat 45
XVII - SCSS 45
XVII-A - À savoir sur le SCSS : 45
XVII-B - Le css et un projet Angular 45
XVII-C - Pratique 45
XVII-C-1 - Remarques pour comp1.component 46
XVII-C-2 - Remarques pour comp2.component 47
XVII-C-3 - Remarques pour comp3.component 47
XVII-D - Résultat 47
XVII-E - Conclusion 47
XVIII - Routing 47
XVIII-A - À savoir 47
XVIII-B - Pratique 47
XVIII-C - Schéma 48
XVIII-D - Conseils 51
XIX - Formulaires 51
XIX-A - Les formulaires + la gestion des erreurs + Angular Material 51
XIX-A-1 - Principes de base 51
XIX-A-2 - Pratique 52
XIX-B - Les formulaires dynamiques 58
XIX-B-1 - Principes de base 58
XIX-B-2 - Remarques 58
XIX-B-3 - Pratique 58
XX - Les filtres (PIPES) 60
XX-A - Pratique 60
XX-B - Résultat 62
XX-C - Conclusion 62
XXI - Les directives 63
XXI-A - Description 63
XXI-B - Théorie 63

XXI-B-1 - La directive d'attribut....63
XXI-B-2 - La directive structurelle....64
XXI-C - Pratique....65
XXI-C-1 - La directive d'attribut....65
XXI-C-1-a - Pratique....65
XXI-C-1-b - À savoir....68
XXI-C-1-c - Conclusion....68
XXI-C-2 - La directive structurelle....68
XXI-C-2-a - Pratique....69
XXI-C-2-a-i - Remarques....70
XXI-C-2-a-ii - Énumération....70
XXI-C-2-a-iii - Le service....71
XXI-C-2-a-iv - La directive....71
XXI-C-2-a-v - À savoir....72
XXI-C-2-a-vi - Conclusion....73
XXII - en cours....73
XXIII - Cycle de vie Angular....73
XXIII-A - Pratique....73
XXIII-B - Résultat....75
XXIII-C - Conclusion....75
XXIV - Architecture avancée : modules, composants web....75
XXIV-A - Qu'est-ce que sont les services ?....76
XXIV-B - Qu'est-ce que le routing ?....76
XXIV-C - Exemple avec un site e-commerce....76
XXIV-C-1 - Organisation d'un projet....77
XXIV-C-1-a - Principes....77
XXIV-C-1-b - Schéma A....77
XXIV-C-1-c - Remarques....78
XXIV-C-1-d - Schéma B....78
XXIV-D - Pratique....80
XXIV-D-1 - Résultat....83
XXIV-E - Bonus....83
XXIV-E-1 - Pratique....83
XXIV-E-1-a - (A1) Importer le package en ligne de commande....83
XXIV-E-1-b - (A2) Créer un composant qui a pour but d'afficher un pdf....83
XXIV-E-1-c - (A3) Importer dans le fichier module du produit le nouveau package....84
XXIV-E-1-d - (A4) Ajouter ce composant dans la page : produit....84
XXIV-E-1-e - Conclusion....84
XXIV-E-1-f - À savoir....85
XXV - Les composants web réutilisables....85
XXV-A - À savoir....85
XXV-B - Description....85
XXV-B-1 - Remarques....85
XXV-B-2 - Inventaires....86
XXV-C - Pratique....86
XXV-D - Résultat....92
XXV-E - Conclusion....92
XXVI - Mise en production : Firebase hosting....92
XXVI-A - Pratique....93
XXVI-B - À savoir....93
XXVI-C - Résultat....94
XXVII - Angular elements....94
XXVII-A - Pratique....95
XXVII-B - Schéma....95
XXVII-C - Comment utiliser ce composant web classique dans un projet Angular....100
XXVIII - Docker....100
XXVIII-A - Installation....100
XXVIII-B - Remarques....100

XXVIII-C - Pratique....101
XXVIII-C-1 - Cas 1 : simplement tester la version en prod (/dist)....101
XXVIII-C-2 - Cas 2 : en dev avec le hot reload (ou live reload)....102
XXVIII-C-2-a - À savoir....102
XXVIII-C-3 - Remarques....102
XXVIII-C-4 - Quelques commandes utiles....103
XXVIII-D - ngx-deploy-docker....103
XXVIII-D-1 - prod....103
XXIX - Étude de cas n°1 : authentification + accès sécurisé à une API....104
XXIX-A - Limitation....104
XXIX-B - Schéma....105
XXIX-C - L'application Angular : /angular-auth-jwt1....106
XXIX-C-1 - À savoir....106
XXIX-C-1-a - Interceptors....106
XXIX-C-1-b - Guards....106
XXIX-C-1-c - JWT....106
XXIX-C-1-d - Le service http et les generics....107
XXIX-C-1-e - Le modèle de l'utilisateur courant ICurrentUser et le service auth.service.ts....107
XXIX-C-1-f - Bonnes pratiques....108
XXIX-C-2 - Pratique....109
XXIX-C-2-a - /core....109
XXIX-C-2-b - /features....116
XXIX-C-2-c - /pages....122
XXIX-C-2-d - /shared....126
XXIX-C-2-e - app....126
XXIX-C-2-f - Configurer Docker dans l'application Angular....128
XXIX-D - Le serveur : node.js du dossier : /node-api....128
XXIX-D-1 - Remarques....128
XXIX-D-2 - Pratique....128
XXIX-D-3 - Remarques....131
XXIX-E - Docker : gestion de l'application et du serveur : node.js....131
XXIX-F - Lancement avec Docker....132
XXIX-F-1 - Remarques....132
XXIX-G - Lancement sans Docker....132
XXX - Étude de cas n°2 : Angular + NestJS : authentification + accès sécurisé à une API....132
XXX-A - Qu'est ce que NestJS ?....132
XXX-B - Pratique....132
XXX-C - Description rapide du back avec NestJS....133
XXX-C-1 - Fonctionnalités....133
XXX-C-2 - Les différents décorateurs dans les controlleurs....133
XXX-C-2-a - Exemple....133
XXX-C-2-b - auth.controller.ts, user.controller.ts, product.controller.ts :....134
XXX-C-2-b-i - Au niveau de la classe :....134
XXX-C-2-b-ii - Au niveau des fonctions de la classe :....134
XXX-C-3 - URLs....134
XXX-C-3-a - /auth/auth.controller.ts....134
XXX-C-3-b - /user/user.controller.ts....134
XXX-C-3-c - /api/product.controller.ts....134
XXX-C-4 - Exemples d'accès aux urls....134
XXX-C-4-a - login....135
XXX-C-4-b - register....135
XXX-C-4-c - refresh token....135
XXX-C-4-d - Obtenir la liste de tous les utilisateurs....135
XXX-C-5 - Validation....135
XXX-C-6 - Configuration....136
XXX-C-6-a - jwt....136
XXX-C-6-b - La base de donnée....136
XXX-C-7 - swagger....137

XXX-C-8 - CORS 137
XXX-C-9 - 137
XXX-C-10 - Les modules 137
XXX-D - Résultat 138
XXXI - Angular Universal (SSR) 138
XXXI-A - À savoir 138
XXXI-B - Pratique 138
XXXI-C - Résultat 139
XXXI-D - Pratique : installation d'Angular Universal 139
XXXI-D-1 - À savoir 140
XXXI-E - Pratique : compilation et exécution 140
XXXI-E-1 - Résultat 140
XXXI-F - En production 140
XXXI-G - Performance à l'affichage de la première page 140
XXXI-H - Transfert d'état de rendu côté serveur pour les requêtes HTTP 140
XXXI-H-1 - Pratique 141
XXXI-H-1-a - Résultat 144
XXXI-I - Conclusion 144
XXXII - Gestion dynamique des metas tags pour le SEO 144
XXXII-A - Pratique 144
XXXII-B - À savoir, ... 145
XXXII-C - Pratique 145
XXXII-C-1 - Exemple 1 : des tags dans le composant de démarrage app.component 145
XXXII-C-1-a - Résultat 146
XXXII-C-2 - Exemple 2 : modifier le tag description sur la page 2 146
XXXII-C-2-a - Résultat 147
XXXII-D - Conclusion 147
XXXII-E - Remarques 147
XXXIII - Les Progressive Web App (PWA) 147
XXXIII-A - Pratique 147
XXXIII-A-1 - Résultat 150
XXXIII-B - Conclusion 151
XXXIV - Gestion de l'état 151
XXXIV-A - Un petit mot sur : ngrx, ngxs 151
XXXIV-B - Un petit mot sur : Akita 151
XXXIV-C - Un petit mot sur la gestion d'état en général 151
XXXIV-D - Un système customisé pour Angular (pour comprendre le fonctionnement) 151
XXXIV-E - Pratique 152
XXXIV-F - Résultat 157
XXXIV-G - Récapitulatif 158
XXXIV-H - stackblitz 158
XXXV - Un petit mot pour la fin 158
XXXVI - Remerciements 158

I - Mémo Angular

Bienvenue sur le mémorandum du framework Angular de Google.

Version Angular : 10+
Version tutoriel : 1.46

II - Objectif de ce tutoriel

- Ce tutoriel est sous la forme d'un mémo présentant les chapitres les plus importants du framework Angular.
- Chaque chapitre est composé de codes sources agrémentés de diverses explications et points-clés à connaitre.
- Ce mémo est destiné aux développeurs Angular débutants et aussi confirmés.
- Connaissances prérequises :
    - TypeScript : débutant ;
    - Angular : débutant.

III - Angular

- Angular est un framework JavaScript complet, il dispose de tous les outils nécessaires pour développer rapidement une application web dynamique de n'importe quelle taille.

IV - Les types d'applications que l'on peut créer

- Single Page App (SPA) :
    - développer des sites web dynamiques, en front. On appelle cela le SPA ;
    - le navigateur du client exécute le code JavaScript et fait tourner l'application de façon dynamique, et ce, sur une seule page web.
- Applications hybrides pour smartphone :
    - développer des applications hybrides pour smartphone avec Ionic 4 ou encore Angular for NativeScript.
- Progressive Web Apps (PWA) :
    - on peut concevoir des PWA (Progressive web App), des applications de bureau ou smartphone qui ont l'avantage de se baser sur une URL pour être installée (avec sa petite icône de lancement) ;
    - et donc, pas besoin d'un système de store pour distribuer ces applications PWA.
- Composants web :
    - des composants web qui peuvent être utilisés dans n'importe quels autres projets ou technologies web.

V - Les technologies qui composent le framework

- Angular Universal et le Server-side rendering (SSR) :
    - **https://angular.io/guide/universal** ;
    - **https://www.ganatan.com/tutorials/server-side-rendering-avec-angular-universal** ;
    - Angular Universal est une solution de prérendu pour Angular ;
    - rendre un projet Angular côté serveur afin qu'il soit pris en compte pour le référencement SEO et améliorer les performances de démarrage d'une application.

- Angular elements - Composant web réutilisable :
    - **https://angular.io/guide/elements** ;
    - il est possible de développer des composants graphiques (comme des widgets) afin de les intégrer dans un site web classique HTML, et ce, avec une simple balise ;
    - et donc, chaque composant contenu dans un seul fichier JS de quelques Ko contiendra tout le nécessaire (html, css, JS) pour un fonctionnement autonome dans une page web HTML quelconque.
- Lazy loading :
    - le chargement différé : une partie d'un projet se charge quand l'utilisateur veut y accéder ;
    - une fois chargé, c'est mis en cache automatiquement ;
    - utile pour réduire le temps de chargement des gros projets ;
    - chargement rapide de la 1re page.
- Packages Angular :
    - Angular propose divers packages aux développeurs (boîte à outils) pour aider à développer plus rapidement ;
    - ainsi, les principaux packages (librairies) disponibles sont sur les domaines suivants : Formulaires, Routing, HTTP, Tests…
- angular-cli :
    - toute une série de commandes afin d'accélérer et faciliter le développement ;
    - **https://cli.angular.io/** ;
    - dans une console, des commandes pour créer un projet, compiler, exécuter… ;
    - mais aussi d'autres commandes pour créer des fichiers de code (créer le squelette d'un composant, d'un service, d'un module, d'un fichier routing…) ;
    - live reload :
        - dans un projet, à chaque enregistrement de n'importe quel fichier (suite à une modification du code), le live-reload recompile automatiquement la partie modifiée et rafraichit automatiquement le navigateur.
- Compilation AOT (Ahead of Time)
    - optimise le code (supprime le code inutilisé)
- Environnements
    - plusieurs environnements : dev, prod…
- La documentation officielle :
    - **https://angular.io/** ;
    - très bonne documentation, très complète.
- Côté développement, c'est :
    - la possibilité d'intégrer n'importe quelle librairie externe JavaScript pour pouvoir l'utiliser ;
    - utilise le Modèle Vue Contrôleur (MVC) pour la séparation des responsabilités. Plus précisément, c'est du MVVM :
        - MVC (Model-View-Controller) : le contrôleur manipule le modèle, la vue affiche le modèle,
        - MVVM (Model-View-ViewModel) : le modèle MVVM prend en charge la liaison de données bidirectionnelle entre View et ViewModel. Cela permet la propagation automatique de changement de ViewModel vers la vue ;
    - avec son architecture orientée MVC, on a l'avantage de l'homogénéité entre les projets ce qui apporte une grande maintenabilité ;

- utilise l'injection de dépendances (DI) pour faciliter l'accès aux diverses librairies pour les composants ou pour les services d'un projet.

VI - L'ensemble des technologies qui gravitent autour d'Angular

- Scully :
  - **https://scully.io/** ;
  - Angular Universal permet du SSR ;
  - avec Scully, vous pouvez faire du SSG, c'est-à-dire du prérendu des pages pour plus de performances et la prise en compte du SEO ;
  - l'inconvénient du SSG c'est que seules les pages statiques seront référencées contrairement au SSR où en plus les pages dynamiques seront référencées ;
  - pour un site web statique, utilisez plutôt le SSG (avec Scully) sinon le SSR (avec Angular universal) :
- TypeScript :
  - Angular utilise par défaut le langage TypeScript de Microsoft (JavaScript ES6, légèrement remanié) ;
  - ce qu'apporte TypeScript :
    - ajoute du typage fort,
    - ajoute la détection d'erreurs à la compilation (et non plus seulement à l'exécution comme avec JavaScript),
    - fournit les features qui sont dans ES6 et dans ES5 et a donc bien souvent une avance,
    - de façon générale, syntaxiquement, il y a peu de différences entre ES6 et TypeScript ,
    - le compilateur de TypeScript n'ajoute aucune dépendance à JavaScript,
    - le code JavaScript généré par TypeScript est d'une grande qualité,
    - en plein développement d'un projet, vous pouvez passer de TypeScript à ES6 sans problème.
- Redux pour Angular (NgRx):
  - **https://ngrx.io/** ;
  - NgRx se présente comme un système de centralisation des données et des actions ;
  - à savoir qu'on peut se passer de redux sur Angular parce que le framework Angular propose un mécanisme de communication le « two way data binding » que l'on associe à un service qui peut être utilisé comme store de données.
  - une alternative : ngxs est une version de redux pour angular basé sur le modèle CQRS **https://www.ngxs.io/**;
  - une autre alternative : Akita (que je recommande) est un système très simple de gestion de l'état **https://datorama.github.io/akita/**;
- RXJS - La programmation réactive :
  - **https://angular.io/guide/rx-library** ;
  - le concept de RXJS repose sur l'émission de données depuis une ou plusieurs sources (producteurs) à destination d'autres éléments appelés consommateurs ;
  - elle repose sur le design pattern : Observable / Observer.
- Webpack :
  - c'est le gestionnaire de ressources (CSS, images…) qui est intégré dans Angular.
- Les tests unitaires :
  - Jasmine & Karma.
- Bibliothèque CSS et de mise en page :
  - Angular Material :

        - une librairie UI entièrement gratuite, développée aussi par Google,
        - **https://material.angular.io/** ;
    - Ngx-Bootstrap :
        - **https://valor-software.com/ngx-bootstrap/#/** ,
        - c'est bootstrap (bibliothèques de composants JavaScript pour le design), une version facile à installer dans un projet Angular ;
    - Angular Flex-Layout :
        - **https://github.com/angular/flex-layout**,
        - de la mise en page avec du Flexbox, une version facile à installer dans un projet Angular.
- ngx ROCKET :
    - **https://ngx-rocket.com/home** ,
    - ngx ROCKET est une surcouche supplémentaire à angular-cli ;
    - il permet de faire plus qu'angular-cli dans la génération de projet ;
    - description : **https://itnext.io/creating-enterprise-angular-apps-with-the-ngx-rocket-generator-871ac76ace87** .

VII - Fonctionnement du framework

- Le cœur d'Angular : les composants web :
    - Angular est un framework basé sur les composants web ;
    - principe : on imbrique les composants web les uns avec les autres pour construire un widget, une fonctionnalité, une page, un projet…
- Les versions :
    - Google met à jour le framework tous les 6 mois,
    - mais n'ayez crainte, de version en version c'est compatible ;
- Angular et l'apprentissage :
    - Angular est un framework complet donc il faut un certain temps pour le maitriser, mais c'est le cas pour n'importe quel autre framework quand on veut aborder tous les sujets.
- Comment fonctionne techniquement la mise à jour dynamique sur Angular :
    - Angular 10+ fonctionne avec le nouveau moteur ivy qui utilise la technique de «l'Incremental DOM» ;
    - Incremental DOM : chaque composant est compilé dans une série d'instructions. Ces instructions créent des arborescences DOM et les mettent à jour sur place lorsque les données changent.

VIII - Angular : ce qu'il faut pour démarrer

- node.js :
    - **https://nodejs.org/en/** ;
    - installer node.js (de préférence, la version LTS) ;
    - npm est la commande de node.js dans une console ;
    - pour connaitre la version de npm installé sur votre système : `npm -v`
    - pourquoi node.js ?

        - quand vous allez installer node.js sur votre système d'exploitation celui-ci va installer un dossier : «node_modules»,
        - ce dossier est en quelque sorte le dossier global qui contiendra tous les modules nécessaires (du code, des outils…) pour faire fonctionner diverses applications,
        - par exemple, angular-cli est un module qui se retrouve dans ce dossier node_modules, car il ne doit pas faire partie d'un quelconque projet Angular mais être disponible pour tous les projets Angular ;
    - Remarques

        - Vous remarquerez qu'un projet Angular dispose aussi d'un dossier *node_modules*, qui n'a rien à voir avec celui en global. Ce dossier est réservé uniquement aux packages utiles au bon fonctionnement du projet en question ;
        - par exemple dans un projet, on peut avoir besoin un package de conversion HTML en PDF et donc nous allons mettre ce package dans *./node_modules* du projet ,
        - il y a donc le dossier *node_modules* global installé sur le système et le dossier *node_modules* d'un projet.
- Installation d'angular-cli :

    - **https://cli.angular.io/** ;
    - angular CLI permet de lancer des commandes en ligne (via la commande ng) pour effectuer diverses tâches comme :

        - créer un projet Angular,
        - compiler et lancer un projet,
        - ajouter au projet des squelettes de fichiers de types : composants, directives, services, modules, pipes, interfaces…
    - nous voulons utiliser angular-cli pour tous nos projets Angular, donc nous allons l'installer en global ;
    - pratique :
      `npm install -g @angular/cli`
    - remarque :

        - « -g » pour indiquer de mettre le package en global (le dossier *node_modules* du système d'exploitation) ;
    - pour connaitre la version d'angular CLI installée sur son système :
      `ng version`

IX - Quelques outils pour développer

- Éditeur de code :

    - **https://code.visualstudio.com/** ;
    - je recommande Visual Studio Code de Microsoft qui est gratuit ;
    - installer une extension à VS pour mieux prendre en compte Angular et typeScript

```
1. fichier -> preferences -> extensions
2.
3. en haut à gauche, dans la barre de recherche: EXTENSIONS
4. tapez : angular
5. choisissez´: Angular Essentials (Version 9) et cliquez sur le bouton  installer
```

- Éditeur de code online :

    - **https://stackblitz.com** ;
    - un éditeur basé sur un navigateur ;

- cela permet de tester rapidement une idée, partager des démos, une application complète, des extraits de code, ou écrire du code lorsque vous êtes loin de votre propre machine ;
- utile aussi pour partager le code sur un forum d'entraide.

- Developer Tool pour navigateur (Google Chrome ou Mozilla Firefox) :
  - Angular DevTools ;
    - **https://chrome.google.com/webstore/detail/angular-devtools/ienfalfjdbdpebioblfackkekamfmbnh** ;
    - Angular DevTools étend Chrome DevTools en ajoutant des capacités de débogage et de profilage spécifiques à Angular. ;
    - une fois installé sur Chrome, allez dans : "plus d'outils" -> "outils de développement" -> dans les onglets, choisir le nouvel onglet "Angular" ;
  - Augury ;
    - **https://augury.rangle.io/** ;
    - est l'extension Developer Tool la plus utilisée pour le débogage et le profilage des applications Angular dans les navigateurs Google Chrome et Mozilla Firefox ;
  - Angular State Inspector :
    - **https://chrome.google.com/webstore/detail/angular-state-inspector/nelkodgfpddgpdbcjinaaalphkfffbem?hl=en**,
    - permet d'inspecter l'état sur la portée de chaque élément DOM,
    - moins complet qu'augury, mais ce que fait Angular State Inspector, il le fait bien.

X - Description des fichiers d'un projet Angular

- Créer un projet que l'on nommera : `angular-skeleton1`

```
1. ng new angular-skeleton1
2. strict ? NO
3. routing ? YES
4. SCSS
```

`cd angular-skeleton1`

X-A - Description du dossier angular-skeleton1

/angular-skeleton1

```
1. /dist                          // dossier contenant les sources pour le déploiement (qui a
 été généré avec la commande ng build --prod)
2. /e2e                           // stock des scripts pour effectuer des tests unitaires
3. /node_modules                  // tous les plugins node.js qui ont été installés via npm
 (packages Angular et des packages utiles pour notre projet que l'on installe)
4. /src
5.     /app                       // app est le premier composant, Angular démarre sur celui-ci
6.         app.component.scss     // le CSS appliqué uniquement à ce composant :
 app.component.html
7.         app.component.html     // Le V de MVC (la vue)
8.         app.component.ts       // Le C de MVC  (similaire à un contrôleur)
9.         app.component.spec.ts  // fichier pour les tests (peut être supprimé si on ne fait
 pas de test)
10.        app.module.ts           // le module (import de librairies et configuration pour le
 bon fonctionnement du composant app)
11.        app-routing.module.ts  // le module pour la gestion du routing (correspondance URL /
 composant)
```

```
12.     /assets             // les ressources : images...
13.     /environments       // environnements d'exécution : prod, dev ou test...
14.     browserslist
15.     favicon.ico
16.     index.html          // le fichier de démarrage qui sera chargé par le navigateur du
 client
17.     karma.conf.js       // fichier de paramétrage du Test runner Karma (les tests unitaires)
18.     main.ts             // contient l'ensemble du projet en typescript
19.     polyfills.ts        // normalisation entre les différents navigateurs
20.     styles.scss         // le CSS global qui sera accessible à tous les composants
21.     test.ts
22.     tsconfig.app.json   // configuration typescript
23.     tsconfig.spec.json
24.     tslint.json         // règles d'écriture du code typescript
25. .gitignore              // les fichiers et dossiers à ignorer pour GIT
26. angular.json            // fichier de configuration utilisé par Angular CLI
27. package.json            // en lien avec le dossier /node_modules - liste les dépendances npm
28. package-lock.json       // en lien avec le dossier /node_modules - liste les versions exactes
 des dépendances - 'npm install' se base sur ce fichier
29. README.md               // présentation du projet en markdown pour github
30. tsconfig.json           // fichier de configuration pour le compilateur de TypeScript
31. tslint.json             // les règles de codage TypeScript
32.                         // permet de vérifier les fichiers TypeScript
```

X-B - Plus qu'à lancer :

ng serve

ou en lançant automatiquement le projet sur le navigateur par défaut

ng serve -o

http://localhost:4200/

Welcome to angular-skeleton1!

X-C - Le point de départ

/src/index.html

```
1. ...
2. <app-root></app-root>
3. ...
```

- le fichier *index.html* est l'unique page que le navigateur chargera ;
- ce fichier contient une balise : <app-root></app-root> ;
- c'est dans cette balise qu'est projetée toute l'application Angular

/src/app
Dans ce dossier, il y a le composant racine : app.component… (css, html, ts)
Et son module : app.module.ts

```
1. index.html        <app-root></app-root>                  // est la balise qui représente le
 composant racine  /app/app.component.html
2.                                                          // ne jamais modifier le fichier
 index.html
```

/src/app/app.component.ts

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

```
2.
3. @Component({                                          // un décorateur '@Component', qui permet
 de configurer la classe AppComponent.ts avec selector, templateUrl et styleUrls
4.     selector: 'app-root',                             // le nom du sélecteur: 'app-root'      --
> le même nom qu'on retrouve comme balise dans le fichier:  /src/index.html -> <app-root></app-
root>
5.     templateUrl: './app.component.html',              // indication de la vue associée
6.     styleUrls: ['./app.component.scss']               // indication du fichier CSS (qui sera
 appliqué uniquement à app.component.html)
7. })
8. export class AppComponent {
9.     title = 'angular-skeleton1';                      // déclarer une variable title
10.                                                      // qui sera uniquement disponible dans la
 vue app.component.html
11. }
```

/src/app/app.component.html

```
1. ...
2. <h1>Welcome to {{ title }}!</h1>
3.
4. <router-outlet></router-outlet>   // facultatif, mais doit être présent pour du routing
5.                                   // toutes pages du routing seront projetées ici
```

// {{ title }} sera remplacé par le texte de la variable : title = 'angular-tuto1';

X-D - Principe

- on déclare une variable dans le : …component.ts et on l'affiche dans sa vue : …component.html

X-E - Conclusion

- app.component.ts est le composant racine et il lui est associé son module racine app.module.ts. L'ensemble représente le point de départ d'un projet.
- index.html est l'unique page qui sera chargée. Ensuite, le projet fonctionnera de façon dynamique.
- index.html contient la balise `<app-root></app-root>`, le sélecteur où est projeté le composant racine app.component.

X-E-1 - Schéma

```
1.   index.html                     // la seule page qui sera chargée par le navigateur
2.     -------------------------
3.       <app-root></app-root>          // dans cette balise sera projeté le projet Angular
 compilé en JavaScript (1)
4.       ...                            // en quelque sorte, c'est : app.module qui est le
 module de démarrage,
5.                                      // celui qui va contenir tous les autres composants web
6.       ...
7.       ...
8.       ...
9.       <script src="main.js">...      (1) contient tout le projet Angular compilé en
 JavaScript
10.     -------------------------
```

X-F - Remarques

- Depuis la version 10 d'Angular, pour économiser des Ko, la spécification ES5 n'est plus prise en compte donc ça ne tournera plus sur IE11 (mais il est possible de le prendre en compte).
- Nous verrons plus loin, comment organiser un projet avec ses modules, composants…

XI - Théorie sur les modules

Un peu de théorie sur les modules pour vous mettre un peu le concept dans la tête avant d'aller plus loin.

- IMPORTANT : il faut diviser un projet en plusieurs modules de fonctionnalités.
- En principe, un module = une fonctionnalité.

XI-A - À savoir

- app.module a pour seule fonctionnalité : le « point de démarrage ».

XI-A-1 - Description d'un module Angular

- Le module racine : app.module.ts représente le contexte de démarrage de l'application et ne dispose que d'un seul composant : app.component

/src/app/app.module.ts

```
1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { AppComponent } from './app.component';
4.
5. @NgModule({
6.   declarations: [               // *1
7.     AppComponent,
8.   ],
9.   imports: [                    // *2
10.     BrowserModule,                              // BrowserModule :    le composant racine
 s'exécutera dans un navigateur
11.   ],
12.   exports: [                    // *3
13.
14.   ],
15.   providers: [                  // *4
16.
17.   ],
18.   bootstrap: [AppComponent]     // **5          // à indiquer uniquement dans le module
 racine : app.module.ts
19. })
20. export class AppModule { }
```

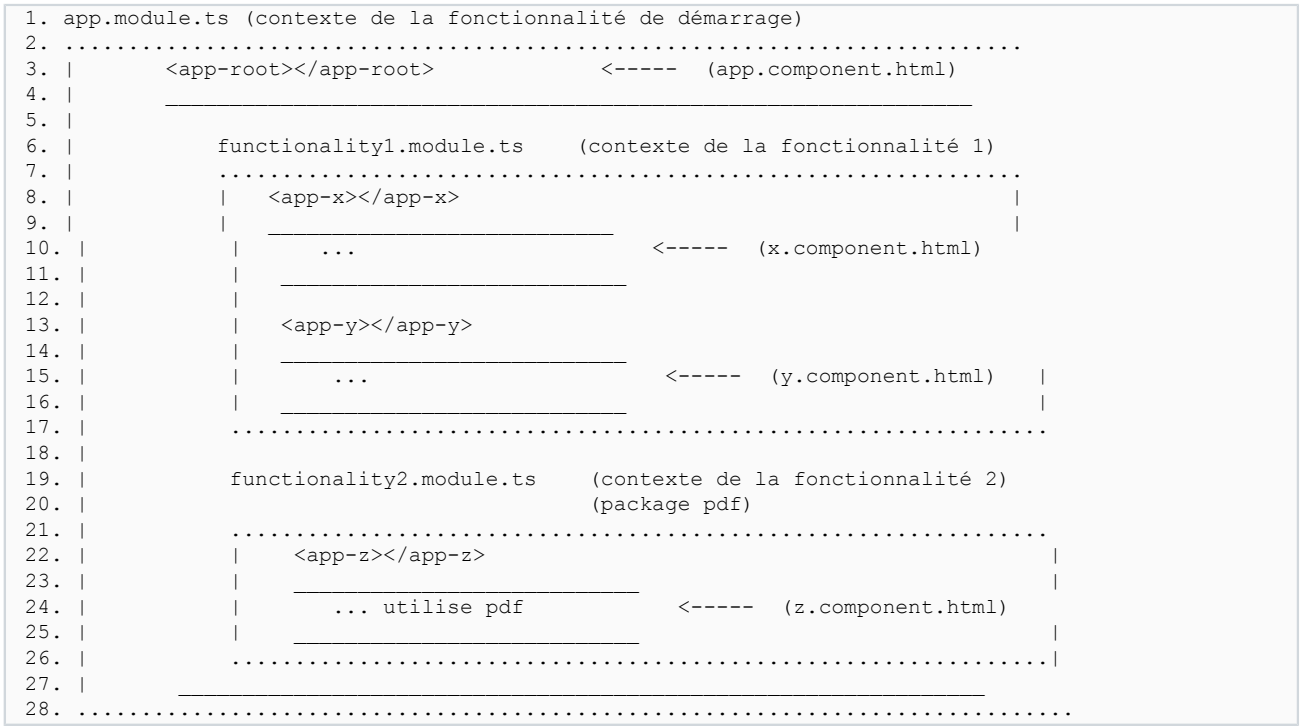
```
1. en général, voici les tâches que l'on peut faire dans un module:
2.
3. *1      déclarer tous les composants que l'on a besoin d'utiliser
4. *2      importer d'autres modules provenant de son propre projet ou provenant de sources
 externes comme celui d'un package npm (dossier /node_modules du projet)
5.         ici on importe le package BrowserModule, car Angular va s'exécuter dans un navigateur

6. *3      indiquer ici les composants web qui doivent être accessibles lors d'un import de ce
 module
7. *4      c'est ici qu'on déclare nos services pour qu'ils soient disponibles dans nos
 composants
8. **5     uniquement présent dans le module racine.
9.           - Dans 'bootstrap', on doit indiquer quel composant est le point d'entrée
10.           - rappel:  app.component.ts est le 1er composant projeté dans la balise : <app-
root></app-root> du fichier: index.html
```

XI-B - Exemple

Le module de démarrage + deux autres modules.

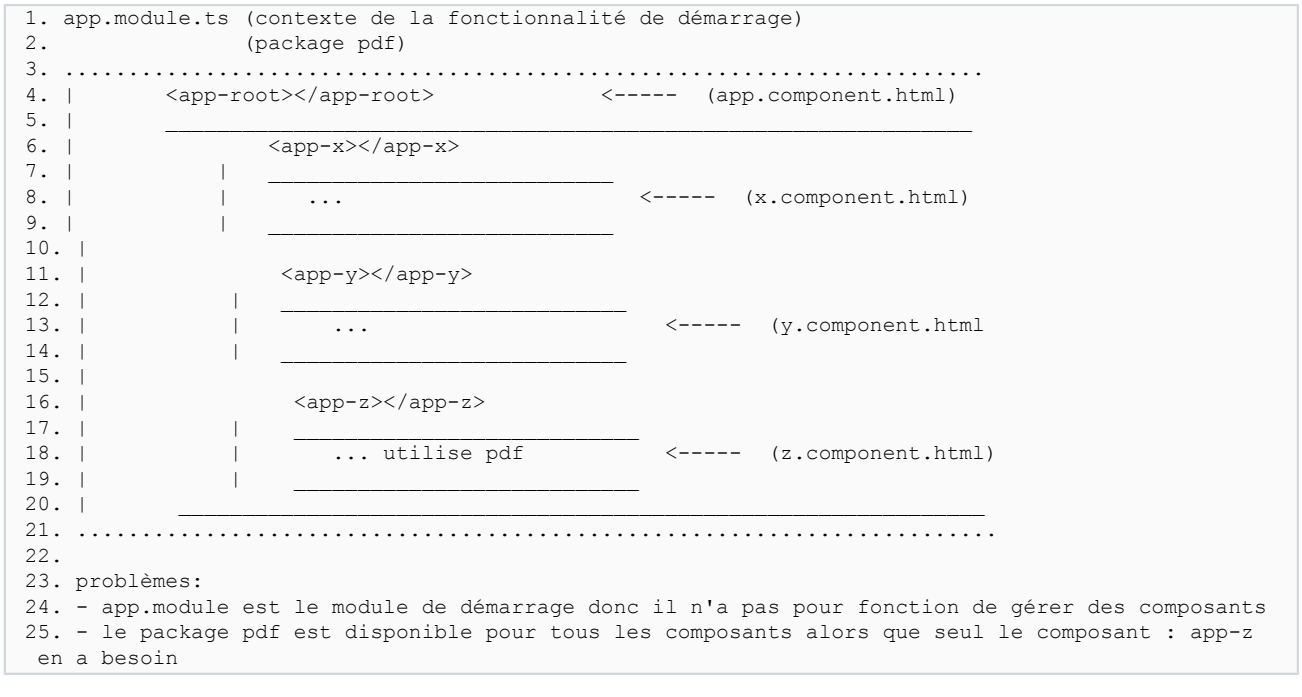
XI-B-1 - Schéma



- Le contexte d'exécution de démarrage app.module.ts est composé de son composant app.component.ts et de deux modules : functionality1.module.ts et functionality2.module.ts.
- functionality1.module.ts déclare deux composants : x et y.
- functionality2.module.ts déclare un composant z et importe un package externe : pdf.

XI-B-2 - Remarques

- Nous aurions très bien pu mettre tous les composants dans app.module.
- Dans ce cas, nous aurions eu ça :



XI-B-3 - Conclusion

- Un module = une fonctionnalité.
- Un module est aussi en quelque sorte un contexte d'exécution que l'on paramètre (le contexte d'une fonctionnalité).
- Un module dispose de certains composants qui s'exécutent dans un contexte.

Mais aussi :

- dans un projet, on peut avoir plusieurs modules (ou chaque module dispose de ses propres composants et de ses propres packages externes) ;
- on emboîte les composants entre eux pour former un projet ;
- les deux grands avantages des modules :
    - rendre les modules (avec ses composants) indépendants les uns des autres,
    - cette indépendance permet de pouvoir réutiliser l'ensemble (module + composants) ailleurs dans un autre projet,
    - réduire en plusieurs modules permet une meilleure maintenance.

Nous verrons plus loin sur les modules dans un autre chapitre.

XI-C - Remarques générales

- Dans le tutoriel, pour un souci de clarté et pour ne pas complexifier, parfois, je ne respecterai pas la bonne pratique évoquée en haut (de ne pas tout mettre dans app.module).

XII - Les composants web

Les composants web sont un ensemble de normes qui permettent à JavaScript de s'exécuter dans un nœud DOM isolé. De cette façon, vous pouvez créer par programme un widget ou même une application entière. Comme pour tout autre nœud DOM, vous utilisez des événements simples et des attributs / propriétés pour communiquer avec le monde extérieur. Pour le reste de la page HTML, le composant web n'est qu'une simple balise :

- les composants web sont les briques pour construire une application Angular ;
- chaque composant est composé de quatre fichiers : .ts, .html, .css. et .spec.ts (test) ;
- le composant web fonctionne de façon indépendante, il a son propre CSS, son propre contrôleur, son propre template ;
- on construit l'ensemble d'un projet en emboîtant les composants web.

XII-A - Pratique

Créer un nouveau projet : `angular-first-component1`

```
1. ng new angular-first-component1
2. strict ? NO
3. routing ? NO
4. SCSS
```

Pour débuter, on va créer un composant : app-first.component et on va l'ajouter directement dans le module racine app.module.ts.

XII-A-1 - Schéma

```
1.      app-module (contexte de démarrage)
2.      ..................................................
3.          app-root   (app.component.html)
4.          ____________________________________
5.
6.                 app-first   (first.component.html)
7.                 ____________________________
8.                   ...
9.                 ____________________________
10.           ____________________________________
11.      ..................................................
```

XII-A-2 - Remarques

Le composant : app-first cherche son contexte vers le haut. Il cherche d'abord un module dans le dossier sur lequel il se trouve, n'en trouvant pas, il remonte et arrive sur app-module.module.ts

```
1. app.module.ts
2.      app.component.ts
3.        first.component.ts                // le nouveau composant (qui est inclus dans
 app.component)
```

Utilisons angular-cli et sa commande : ng

```
1. ng generate component components/first --module=app
2. // ou la version raccourcie :
3. ng g c components/first --module=app
```

// '--module=app' permet d'ajouter automatiquement la déclaration du composant : 'first.component.ts' dans le module 'app.module.ts'.
(Pas besoin de faire cette tâche nous-mêmes à la main, merci angular-cli.)

XII-B - Description

Quatre fichiers ont été créés (first.component…: html, ts, css, spec) et un fichier: 'app.module.ts' a été modifié

```
1. le contrôleur       first.component.ts          // code métier, récupération de données... et
 fournit des données à la vue
2. la vue              first.component.html        // la vue doit se contenter d'afficher les
 données reçues du contrôleur
```

XII-B-1 - Allons voir :

/app/app.module.ts // le fichier qui a été modifié

```
1. ...
2.   declarations: [
3.       AppComponent,
4.       FirstComponent,              // le composant 'first' a bien été ajouté, il peut donc
 être utilisé
5. ...
```

/app/app.component.html // app.component étant le composant de démarrage

```
1. <app-first></app-first>             <!-- on indique l'utilisation du composant:
 first.component -->
```

XII-B-2 - Remarques

- Pourquoi la balise avec ce nommage : <app-first> ?
- Allez voir dans le composant : first.component.ts et regardez la ligne : selector: 'app-first',
- la balise : <app-first></app-first> correspond au selector : 'app-first'.

XII-C - Allons un peu plus loin avec ce composant

- Nous allons enrichir le composant en décrivant différentes manières de passer des variables à la vue.
- Pour cela, on va utiliser un modèle et une classe service dans lesquels on va stocker des valeurs.

XII-C-1 - À savoir

- Un modèle de données est une sorte de contenant permettant de définir certaines valeurs. Dans un projet, on manipule des modèles de données.
- Un service (provider en anglais), est une classe où l'on met son code métier, pour stocker des valeurs et pour communiquer avec d'autres services ou composants.

XII-D - Le modèle de données

ng g i models/model-x

/models/model-x.ts

```
1. export interface ModelX {
2.   name: string;             // obligatoire
3.   firstname: string;        // obligatoire
4.   job?: string;             // facultatif.   avec ?, 'job' est rendu optionnel
5. }
```

XII-D-1 - À savoir

- En programmation orientée objet, une interface permet de donner un comportement à une classe.

XII-E - Les services

- Un service contient du code métier (propriétés, fonctions).
- Les composants web utilisent le code métier des services.

XII-E-1 - Pratique

Notre composant web : app-first.component va utiliser ce service afin d'y stocker des valeurs

ng g s services/stored1

/services/stored1-service.ts

```
1. import { Injectable } from '@angular/core';
2. import { ModelX } from '../models/model-x';
3.
4. @Injectable({
5.   providedIn: 'root'            // le service sera une instance singleton de niveau : root
 c'est-à-dire depuis la racine du projet
```

```
6. })                              // et donc la même instance sera fournie aux composants qui la
 demandent
7. export class Stored1Service {
8.
9.   public storedValue1 = 'texte1 from service';                    // on précise : public donc
 il sera accessible depuis une autre classe
10.   storedValue2 = 'texte2 from service';                          // les propriétés sont par
 défaut en : private
11.   storedModel1: ModelX = {name: 'shrader', firstname: 'Hank'};   // on précise que
 storedModel1 est du type: ModelX
12.                                                                  // remarquez qu'on ne
 renseigne pas : job, car il est optionnel
13.   constructor() { }
14.
15.   getStoredValue2(): string {                                    // les fonctions sont
 par défaut en : public
16.     return this.storedValue2;
17.   }
18.
19.   getStoredModel1(): ModelX {                                    // : ModelX, pour
 préciser le type de retour : ModelX
20.     return this.storedModel1;
21.   }
22. }
```

/app/first.component.html

```
1. <p>first works!</p>
2. <hr>
3. <p>(1) variable1={{variable1}}</p>
4. <p>(2) array1={{array1}}</p>
5. <p>(3) objet1={{objet1|json}} | objet1.val1={{objet1.val1}} | objet1.val2={{objet1.val2}}</p>
6. <p>(4) dataObs={{dataObs$|async}}</p>
7. <hr>
8. <p>(5) model1={{model1|json}} avec job: model1.job={{model1.job}}</p>
9. <p>(6) model2={{model2|json}}</p>
10. <hr>
11. <p>(7) value1Service={{value1Service}}</p>
12. <p>(8) value2Service={{value2Service}}</p>
13. <p>(9) storedModel1={{storedModel1|json}}</p>
14. <hr>
15. <p>(10) dataFn1()={{dataFn1()}}</p>
16. <hr>
17. <p>(11) privValue= </p>
18. <hr>
19. <p>(12) stored2Service.storedValue1={{stored2Service.storedValue1}}</p>
20. <p>(12) stored2Service.getStoredModel1()={{stored2Service.getStoredModel1()|json}}</p>
21. <hr>
22. <p>(13) data2={{data2|json}}</p>
```

/app/first.component.ts

```
1. import { Component, OnDestroy, OnInit } from '@angular/core';
2. import { Observable, of, Subscription } from 'rxjs';                    //  ne pas
 oublier d'importer les classes
3. import { ModelX } from '../../models/model-x';                          //  que l'on va
 utiliser dans le composant
4. import { Stored1Service } from '../../services/stored1.service';        //
5.
6. @Component({
7.   selector: 'app-first',
8.   templateUrl: './first.component.html',
9.   styleUrls: ['./first.component.scss']
10. })
11. export class FirstComponent implements OnInit, OnDestroy {
12.   // par défaut, les variables et les fonctions du composant sont déclarées en: public
13.   //
14.   // public variable1 = 'hello from variable';
15.   variable1 = 'hello from variable';                                   // (1) une
 variable
```

```
16.   array1 = [10, 'Breaking bad', 30];                                    // (2) un tableau
17.   objet1 = {val1: 'Ehrmantraut', val2: 'Mike'};                         // (3) un objet
 quelconque
18.   dataObs$: Observable<string> = of('text from observable A');  // (4) avec un observable, la
 vue souscrit automatiquement à l'observable pour obtenir les données
19.                                                                   //     par convention on met
 un $ à la fin de la variable pour indiquer que c'est un observable (facultatif)
20.                                                                   //     sachez également que
 la vue se désabonne automatiquement quand le composant auquel il appartient n'est plus utilisé
21.                                                                   //     et donc pas besoin de
 se désabonner dans ngOnDestroy()
22.   model1: ModelX = {name: 'White', firstname: 'Walter', job: 'chimiste'};   // (5) un objet
 qui implémente un modèle - (voir /models/model-x.ts)
23.   model2: ModelX = {name: 'Pinkman', firstname: 'Jesse'};                 // (6) idem -
 avec une propriété en moins
24.   value1Service: string = this.stored1Service.storedValue1;               // (7) depuis la
 propriété d'un service (qui doit être public)
25.   value2Service: string = this.stored1Service.getStoredValue2();          // (8) depuis une
 fonction d'un service
26.   storedModel1: ModelX;                                                   // (9) initialisé
 dans : ngOnInit()
27.   private privValue = 100;                                                 // (11) n'est pas
 accessible depuis la vue
28.   subData2: Subscription;                                                   // (13) pour
 pouvoir se désabonner
29.   dataObs2$: Observable<ModelX> = of({name: 'Fring', firstname: 'Gustavo'});  // (13)
 l'observable qui va transmettre un objet de type : Modelx aux souscripteurs
30.   data2: ModelX;                                                          // (13) en lien
 avec la vue
31.
32.                                                           // constructor(.......)
33.                                                           // automatiquement, l'injection
 de dépendance (DI) a injecté l'instance de Stored1Service
34.                                                           // stored1Service contient
 l'instance qui va être utilisée n'importe où dans le composant avec : this.stored1Service
35.   constructor(
36.     private stored1Service: Stored1Service,                               // on le met en
 private pour le protéger afin qu'il ne soit accessible qu'ici (dans le contrôleur)
37.     public stored2Service: Stored1Service                                 // (12) mauvaise
 pratique : en public, le service: stored2Service est accessible depuis la vue
38.   ) { }
39.
40.   ngOnInit(): void {
41.     this.storedModel1 = this.stored1Service.getStoredModel1();   // (9) à l'initialisation du
 composant, on appelle la fonction du service pour récupérer la variable : storedModel1
42.
43.     this.subData2 = this.dataObs2$.subscribe((data: ModelX) => {    // (13) on souscrit à
 l'observable pour récupérer la valeur. subData2 contient cette souscription
44.       // ici, on effectue un éventuel traitement...                  //
45.       this.data2 = data;                                            //      on transmet la
 valeur reçue à data2 pour la vue
46.     });
47.   }
48.
49.   dataFn1(): string {                                                 // (10) une fonction peut
 aussi être appelée dans la vue (par défaut une fonction est public)
50.     return 'texte de la fonction';                                    //      il faut donc
 retourner une valeur
51.   }
52.
53.   ngOnDestroy(): void {
54.     this.subData2.unsubscribe();                                      // (13) il faut toujours
 se désabonner à un observable que l'on a souscrit manuellement
55.                                                                       //      sinon il y a un
 risque de fuite de mémoire
56.   }
57. }
```

XII-E-2 - Remarques

- (9) Voyez le typage de la variable storedModel1 `storedModel1: ModelX;` et de la fonction du service `getStoredModel1(): ModelX {...}`
    - le typage permet de sécuriser la variable et la fonction contre les erreurs ;
    - la donnée que l'on manipule ne peut être que du type ModelX sinon une erreur arrive à la compilation.
- (4) et (13) permet de faire la même chose sur un observable. La méthode (4) est plus réduite en code. On privilégie la méthode (13) quand on a un traitement à effectuer sur les données reçues de l'observable.
- À propos de cette écriture :

```
1. constructor(private monService: MonService) {  }
```

en réalité, sachez que c'est un raccourci syntaxique pour :

```
1. private monService: MonService;
2.
3. constructor(monService: MonService) {
4.   this.monService = monService;
5. }
```

XII-E-3 - Résultat

```
1. ng serve
```

http://localhost:4200/

à l'écran, vous obtenez :

```
1. (1) variable1=hello from variable
2. (2) array1=10,Breaking bad,30
3. (3) objet1={ "val1": "Ehrmantraut", "val2": "Mike" } | objet1.val1=Ehrmantraut |
 objet1.val2=Mike
4. (4) dataObs=text from observable A
5. (5) model1={ "name": "White", "firstname": "Walter", "job": "chimiste" } avec job:
 Model1.job=chimiste
6. (6) model2={ "name": "Pinkman", "firstname": "Jesse" }
7. (7) value1Service=texte1 from service
8. (8) value2Service=texte2 from service
9. (9) storedModel1={ "name": "shrader", "firstname": "Hank" }
10. (10) dataFn1()=texte de la fonction
11. (11) privValue=
12. (12) stored2Service.storedValue1=texte1 from service
13. (12) stored2Service.getStoredModel1()={ "name": "shrader", "firstname": "Hank" }
14. (13) data2={ "name": "Fring", "firstname": "Gustavo" }
```

XII-F - Conclusion

- Vous avez vu ce que sont : un composant, un modèle de données et un service.
- De plus, depuis un composant, vous avez vu différentes façons de passer des données à son template.

XIII - Création du 1er module et de ses composants

- Dans un projet, nous avons un module racine : app.module.ts et son composant: app.component.ts.

XIII-A - Exemple : créer un nouveau module et lui attacher deux composants

voici le schéma :

```
1.      app-module
2.      ..........................................................
3.          <app-root   (app.component.html)
4.          ________________________________________
5.
6.              partial-module
7.              ................................................
8.
9.                  <app-header   (header.component.html)
10.                  ____________________________
11.                      ...
12.                  ____________________________
13.                  ...
14.                  ...
15.                  <app-footer   (footer.component.html)
16.                  ____________________________
17.                      ...
18.                  ____________________________
19.              ................................................
20.          ________________________________________
21.      ..........................................................
22.
23. dossiers:
24.
25. /app
26.    app.module.ts                          // le module racine, est toujours présent
27.                                           // on importe : partial.module.ts (pour utiliser les
 composants de ce module)
28.                                           // sinon on ne pourrait pas utiliser : app-header et
 app-footer
29.    app.component...(ts,html...)           // app-root : le composant racine
30. /partials
31.    partial.module.ts                                  // header et footer doivent être déclarés
 dans son module : partial.module.ts (pour être utilisés dans son module)
32.                                                       // header et footer doivent être rendus
 exportables dans son module : partial.module.ts (pour être utilisés dans un autre module)
33.    /header
34.      header.component....(ts,html...)       // app-header
35.    /footer
36.      footer.component....(ts,html...)       // app-footer
```

XIII-B - Pratique

```
1. ng new angular-module1
2. strict ? NO
3. routing ? NO
4. SCSS
```

XIII-B-1 - Création du nouveau module

Donc nous mettons le module et ses composants dans un dossier : /partials :

- on commence par créer le module : partials-module où l'on va mettre nos composants

```
1. ng g m partials --module=app
```

- avec : `--module=app` on indique à angular-cli de rajouter l'import dans le module app.module.ts (comme ça, on n'a pas à le faire) ;
- quand on crée un module, un dossier du même nom est créé ;

- on obtient bien : /partials/partials.module.ts.

XIII-B-2 - Allons voir :

/app/app.module.ts

```
1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { AppComponent } from './app.component';
4. import { PartialsModule } from './partials/partials.module';
5.
6. @NgModule({
7.   declarations: [AppComponent],
8.   imports: [
9.     BrowserModule,
10.     PartialsModule,                          // c'est OK ! pour utiliser les composants :
 header.component et footer.component dans la page: app.component.html
11.   ],
12.   providers: [],
13.   bootstrap: [AppComponent],
14.   schemas: [  ],
15. })
16. export class AppModule { }
```

XIII-B-3 - Ensuite, créons les deux composants

Ajoutons deux composants au module : partials-module

```
1. ng g c partials/header --module=partials
2. ng g c partials/footer --module=partials
```

XIII-B-4 - Allons voir

/partials/partials.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3. import { HeaderComponent } from './header/header.component';
4. import { FooterComponent } from './footer/footer.component';
5.
6. @NgModule({
7.   declarations: [HeaderComponent, FooterComponent],            // c'est ok !
8.   imports: [
9.     CommonModule
10.   ]
11. })
12. export class PartialsModule { }
```

XIII-B-5 - Résultat

```
1. ng serve
```

Allons voir : **http://localhost:4200/**

- Rien ne se passe à l'écran, c'est normal.
- On a créé et paramétré les composants, mais il faut indiquer où ils doivent être intégrés dans les vues.
- Nous utilisons ces composants dans le composant racine : app.component.html, car le header et le footer sont toujours présents, quelle que soit la page où l'on se trouve.

/app/app.component.html

```
<app-header></app-header>                <!-- on est dans : app.component du module : app.module -->
<hr>                                     <!-- donc : app.module doit connaitre l'existence de ces composants : app-header et app-footer -->
<h1>Welcome to {{ title }}!</h1>         <!-- pour cela dans : app.module on importe le module : partials.module (qui contient ces composants) -->
<hr>
<app-footer></app-footer>
```

Vous constaterez des erreurs dans la console des outils dev de votre navigateur et des erreurs à l'écran

```
'app-header' is not a known element
'app-footer' is not a known element
```

- C'est normal. Il y a une chose en plus que vous devez connaitre :

partials-module

```
  declarations: [HeaderComponent, FooterComponent],       // OK, on a déclaré les deux composants !
  imports: [
    CommonModule
  ]
  ...
```

app-module

```
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule,
    PartialsModule              // OK, on a importé : PartialsModule !
  ],                            //, mais sachez que quand on importe d'un côté, il faut exporter de l'autre
  ...                           // chose que nous n'avons pas faite
```

- Pour importer, il faut exporter.
- Réécrivons le module : partials-module.

partials-module.module.ts

```
...
@NgModule({
  declarations: [HeaderComponent, FooterComponent],
  imports: [
    CommonModule
  ],
  exports: [HeaderComponent, FooterComponent]             // ici, on rend exportables les composants que l'on souhaite
                                                          // afin qu'ils soient importables dans un autre module
...
```

XIII-B-6 - Conseils

- Quand vous manipulez les modules, il faut toujours relancer : `ng serve` (car les modifications des modules ne sont pas toujours prises en compte dans le live reload).

XIII-B-7 - Remarques

- Pourquoi doit-on rendre exportable un composant, ne peut-il pas se faire automatiquement ?
- Pour des raisons de sécurité, on veut avoir le choix de ne pas rendre exportable un composant afin qu'il soit restreint de n'être utilisable que dans son module.

XIII-B-8 - Voyons le résultat à l'écran :

```
header works!
_____________________________
Welcome to angular-tuto1!
_____________________________
footer works!
```

XIII-C - Conclusion

- Création d'un module et de ses composants.
- Depuis un module « supérieur » : on importe ce nouveau module.
- Depuis le nouveau module : on déclare les composants (pour qu'ils soient rendus utilisables).
- Depuis le nouveau module : on exporte les composants (pour qu'ils soient pris en compte dans l'import d'un autre module).

XIV - Communication entre les composants

On peut avoir besoin de communiquer entre deux composants pour passer des données de l'un vers l'autre.

XIV-A - Les différentes techniques

```
- le Two-way Data Binding :                une technique pour communiquer du parent vers
 l'enfant et de l'enfant vers le parent
                                          Les deux composants doivent être imbriqués, un est
 considéré comme le parent et l'autre, l'enfant
- par service avec une variable :          un ou plusieurs composants peuvent communiquer avec
 un ou plusieurs autres composants à travers une variable
- par service avec un observable :         un ou plusieurs composants peuvent communiquer avec
 un ou plusieurs autres composants à travers un observable
                                          avantages avec l'observable :
                                              - on peut envoyer une donnée ou un flux de
 données (une valeur puis une autre)
                                              - les clients (composants, services...) qui ont
 souscrit à cet observable reçoivent la ou les valeurs et peuvent ainsi agir
```

XIV-B - Two-way Data Binding

Communication : parent vers enfant ou enfant vers parent.

XIV-B-1 - Pratique

```
ng new angular-two-way1
strict ? NO
routing ? NO
SCSS
```

```
ng g c parent --module=app
ng g c child --module=app
```

Mettons en place :

- le parent contenant l'enfant ;
- la communication parent vers enfant : avec différents types de données (number, string, observable…) ;
- la communication enfant vers parent : avec un EventEmitter.

app.component.html

```
<app-parent></app-parent>
```

parent.component.html

```
<div style="background: lavender;">
  <p>parent works!</p>

  <app-child
    [myMessage1]="'hello toto!'"
    [myVariable1]="variable1"
    [myData1]="data1"
    [myData2]="data2"
    [myDataObs$]="dataObs$"
     [myDataFn]="dataFn()"

     (sendMessage1)="receptionFromChild($event)"
  ></app-child>
  <p>message provenant de l'enfant ---> (7)
 messageReceptionFromChild=<b>{{ messageReceptionFromChild }}</b></p>

  <hr>
  <h2>test de ngOnChanges</h2>
    <button (click)="clickUpdateMyMessage()">(8) cliquez ici ! modification de la variable:
 variable1(2) et voir si l'enfant a détecté la modification dans ngOnChanges()  (voir console)</
button>
</div>
```

parent.component.ts

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

@Component({
  selector: 'app-parent',
  templateUrl: './parent.component.html',
  styleUrls: ['./parent.component.scss']
})
export class ParentComponent implements OnInit {
  // envoi vers enfant
  variable1 = 'hello from child variable';                              // (2) (8)
  data1 = [10, 'coucou', 30];                                           // (3)
  data2 = {name: 'joe', firstanme: 'black'};                            // (4)
  dataObs$: Observable<string> = of('text from observable A');          // (5)

  // réception de l'enfant
  messageReceptionFromChild: string;                                    // (7)

  dataFn(): string {                                                    // (6)
    return 'text from function';
  }

  constructor() { }

  ngOnInit(): void { }

  receptionFromChild(event: string): void {                             // (7)
    this.messageReceptionFromChild = event;
  }

```

```
31.   clickUpdateMyMessage(): void {                                    // (8)
32.     this.variable1 = '***** texte de variable1 modifié ******';     // on modifie :
 variable1 pour déclencher la détection dans le composant enfant
33.   }
34. }
```

child.component.html

```
1. <div style="background: gainsboro; margin-left: 24px;">
2.   <p>child works!</p>
3.
4.   <h2>réception enfant:</h2>
5.   <p>(1) myMessage1={{myMessage1}}</p>
6.   <p>(2) myVariable1={{myVariable1}}</p>
7.   <p>(3) myData1={{myData1|json}}</p>
8.   <p>(4) myData2={{myData2|json}}</p>
9.   <p>(5) myDataObs$={{myDataObs$|async}}</p>
10.   <p>(6) myDataFn={{myDataFn}}</p>
11.
12.   <hr>
13.   <h2>vers le parent:</h2>
14.   <button (click)="clickSendMessage()">(7) cliquez ici ! envoi d'un message du composant
 enfant vers le parent</button>
15. </div>
```

child.component.ts

```
1. import { Component, OnInit, Input, Output, EventEmitter, OnChanges,
 SimpleChanges } from '@angular/core';
2. import { Observable } from 'rxjs';
3.
4. @Component({
5.   selector: 'app-child',
6.   templateUrl: './child.component.html',
7.   styleUrls: ['./child.component.scss']
8. })
9. export class ChildComponent implements OnInit, OnChanges  {
10.   // @Input()       ->   réception d'une donnée en entrée
11.   @Input() myMessage1: string;                    // (1)
12.   @Input() myVariable1: string;                   // (2)
13.   @Input() myData1: any;                          // (3)
14.   @Input() myData2: any;                          // (4)
15.   @Input() myDataFn: string;                      // (5)
16.   @Input() myDataObs$: Observable<string>;        // (6)
17.
18.   // @Output()      ->    envoi vers le parent
19.   @Output() sendMessage1 = new EventEmitter<string>();            // (7) une sortie vers le
 parent qui écoute
20.
21.   constructor() { }
22.
23.   ngOnInit(): void {
24.   }
25.
26.   clickSendMessage(): void {                                      // (7)
27.     this.sendMessage1.emit('text from child');                    // émission de données sur
 la sortie : @Output() sendMessage1
28.   }
29.
30.   ngOnChanges(changes: SimpleChanges): void {                     // (8) ngOnChanges détecte
 le changement de valeur uniquement avec les variables d'entrées : @Input() .......
31.     console.log('--------------------------');                    //
32.     //
33.     // afficher toutes les variables qui ont été modifiées
34.     for (const propName in changes) {                             // Angular met dans
 "changes" toutes les variables qui ont été modifiées
35.       const change = changes[propName];
36.       console.log('propriété qui a été modifiée=' + propName);
37.       console.log('ancienne valeur=' + change.previousValue);
38.       console.log('nouvelle valeur=' + change.currentValue);
```

```
    }
    //
    // disons que je veux savoir si la variable: myVariable1 est modifiée  (les autres variables ne m'intéressent pas)
    for (const propName in changes) {
      if (propName === 'myVariable1') {                          // la variable qui m'intéresse
        console.log('============================');
        console.log('myVariable1 a été modifié=' + propName);     // forcement propName = 'myVariable1'
        const change = changes[propName];
        console.log('ancienne valeur=' + change.previousValue);
        console.log('nouvelle valeur=' + change.currentValue);
      }
    }
  }
}
```

```
ng serve
```

XIV-B-2 - On obtient ceci à l'écran

```
parent works!

        child works!
        réception enfant:
        (1) myMessage=hello toto!
        (2) myVariable=hello from child variable
        (3) myData1=[ 10, "coucou", 30 ]
        (4) myData2={ "name": "joe", "firstanme": "black" }
        (5) myDataObs$=text from observable A
         (6) myDataFn=text from function
        vers le parent:
        bouton [(7) envoi d'un message du composant enfant vers le parent]

message provenant de l'enfant ---> (7) messageReceptionFromChild=

test de ngOnChanges
bouton [(8) modification de la variable: variable1(2) et voir si l'enfant a détecté la modification dans ngOnChanges()  (voir console)]
```

XIV-B-3 - Conclusion

- La plupart des communications dont on a besoin dans un projet sont une communication parent / enfant.
- Two-way Data Binding est uniquement valable pour des composants imbriqués parent / enfant.
- On peut détecter le changement de valeur des variables en entrée (`@Input()…`) dans le composant enfant via : `ngOnChanges()`.

XIV-C - Communication par service (technique par observable ou par variable)

- Plusieurs composants peuvent communiquer entre eux, quel que soit leur emplacement (imbriqués ou pas).

XIV-C-1 - Schéma

```
      app.module.ts (contexte d'exécution) - avec une instance du service : stored1.service.ts
      ..........................................................
      |     app-root
      |     ____________________________________________________
      |         app-comp1       <--- accès au service : stored1.service.ts
      |         ___________________________________
      |             ...
      |         ___________________________________
      |
```

```
|          app-comp2       <--- accès au service : stored1.service.ts
|          ____________________________
|               app-comp3  <--- accès au service : stored1.service.ts
|               ____________________________
|                     ...
|               ____________________________
|          ____________________________
|     ____________________________________________________
|
.....................................................
```

XIV-C-1-a - Principes

- Les composants : app-comp1, app-comp2, app-comp3 font partie du module : app.module.ts.
- On indique que le service est en 'root' donc une seule et même instance (singleton) de : stored1.service.ts sera injectée à tous ceux qui le demandent.
- Ces trois composants peuvent modifier ou lire la variable qui se trouve dans le service : stored1.service.ts.
- Il y a une imbrication entre : app-comp2 et app-comp3, mais ça importe peu vu que nous utilisons la communication par service (contrairement au two way data binding).

XIV-C-2 - Pratique

```
ng new angular-service-com1
strict ? NO
routing ? NO
SCSS
```

```
cd angular-service-com1
ng g c comp1 --module=app
ng g c comp2 --module=app
ng g c comp3 --module=app
ng g s services/stored1
ng g i models/i-user
```

/app/app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { Comp1Component } from './comp1/comp1.component';
import { Comp2Component } from './comp2/comp2.component';
import { Comp3Component } from './comp3/comp3.component';

@NgModule({
  declarations: [
    AppComponent,
    Comp1Component,           // c'est OK !
    Comp2Component,           // les composants sont bien déclarés ici
    Comp3Component            //
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

/models/i-user.ts

```
export interface IUser {
  name: string;
  firstname: string;
  genre: 'madame' | 'monsieur' | 'mademoiselle';
```

```
5. }
```

/app/services/store1.service.module.ts

```
1. import { Injectable } from '@angular/core';
2. import { BehaviorSubject, Subject } from 'rxjs';
3. import { IUser } from '../models/i-user';
4.
5. @Injectable({
6.   providedIn: 'root'        // l'instance du service sera du niveau 'root' c.-à-d. la même
 instance dans tout le projet
7. })                          // tous les composants du projet auront accès à la même instance
 du service
8. export class Stored1Service {
9.   private message1 = 'avec une variable du service : texte initial';
                                        // (1) par service
10.   private messageSubject: BehaviorSubject<string> = new BehaviorSubject<string>('avec un
 observable du service: texte initial');   // (2) par observable
11.
                                       // BehaviorSubject = initialiser avec une valeur
12.   private dataSubject: Subject<IUser> = new Subject<IUser>();
              // (4) par observable
13.
              // Subject = pas d'initialisation de valeur
14.
15.   constructor() { }
16.
17.   getMessage1(): string {                                    // (1)
18.     return this.message1;
19.   }
20.
21.   setMessage1(message1: string): void {                      // (1)
22.     this.message1 = message1;
23.   }
24.
25.   getMessageSubject(): BehaviorSubject<string> {             // (2)
26.     return this.messageSubject;
27.   }
28.
29.   getDataSubject(): Subject<IUser> {                         // (4)
30.     return this.dataSubject;
31.   }
32.
33.   emitDataSubject(user: IUser): void {                       // (4)
34.     this.dataSubject.next(user);
35.   }
36. }
```

XIV-C-2-a - Remarques générales

- Bonne pratique : on met les propriétés d'un service en private.
- Pour récupérer une propriété depuis l'extérieur de la classe, private oblige à faire appel à une fonction comme getData1().
- Idem si on veut affecter une valeur à une propriété, private oblige de passer par une fonction comme setData1(….).
- BehaviorSubject, est un observable/observer (les deux à la fois). :
    - BehaviorSubject : on peut à la fois l'écouter avec .subscribe() ou émettre une valeur avec .next(..).
- BehaviorSubject doit être obligatoirement initialisé à la création :

```
1. private dataBSubject: BehaviorSubject<string> = new BehaviorSubject<string>('par observable:
 texte initiale');
2.                                                                                  ^  valeur
 initialisée   ^
```

```
3. <string>    ---> pour indiquer que la valeur des données que l'on manipule dans l'observable
 sera du type string.
```

- L'instance du service sera du niveau 'root' c.-à-d. la même instance dans tout le projet :
    - donc tous les composants du projet auront accès à la même instance du service. C'est pour cela que si un composant X modifie une propriété d'un service, le composant Y peut accéder à la valeur qui a été modifiée (puisque c'est la même instance) ;
    - il est possible de configurer un service pour n'avoir une instance qu'au niveau composant (ainsi un composant X peut avoir une instance d'un service A et un autre composant Y peut avoir une autre instance d'un même service A).

app.component.html

```
1. <app-comp1></app-comp1>
2. <hr>
3. <app-comp2></app-comp2>
```

comp1.component.html

```
1. <div style="background: pink;">
2.   <p>comp1 works!</p>
3.   <p>(1) réception par variable     -------------> getMessage1()={{getMessage1()}}</p>
4.   <p>(2) réception par observable -------------> messageSubject={{messageSubject|async}}</p>
5.   <p>(3) message={{message}}</p>
6.   <button (click)="choice(1)">monsieur dexter holland</button>
7.   <button (click)="choice(2)">madame agnes obel</button>
8. </div>
```

XIV-C-2-b - comp1.component

```
1. {{dataBSubject|async}}      le pipe async permet à angular de souscrire de façon automatique à
 dataBSubject et ainsi récupérer la valeur de l'observable
2.                             le pipe async écoute l'observable
```

comp1.component.ts

```
1. import { Component, OnDestroy, OnInit } from '@angular/core';
2. import { Stored1Service } from '../services/stored1.service';
3. import { BehaviorSubject, Subscription } from 'rxjs';
4. import { IUser } from '../models/i-user';
5.
6. @Component({
7.   selector: 'app-comp1',
8.   templateUrl: './comp1.component.html',
9.   styleUrls: ['./comp1.component.scss']
10. })
11. export class Comp1Component implements OnInit, OnDestroy {
12.   messageSubject: BehaviorSubject<string>;                          // (2) un BehaviorSubject
 est un observable un peu "spécial"
13.   messageSubscription: Subscription;                                // (3) pour le
 désabonnement
14.   message: string;                                                  // (3) une variable
 string pour la vue
15.
16.   constructor(private stored1Service: Stored1Service) { }
17.
18.   ngOnInit(): void {
19.     this.messageSubject = this.stored1Service.getMessageSubject();    // (2) on récupère
 l'observable du service et on l'affecte à une variable local : dataBSubject du composant
20.                                                                       //     afin qu'il soit
 accessible par la vue (c'est le pipe |async de la vue qui va souscrire à l'observable)
21.
```

```
22.
  this.messageSubscription = this.stored1Service.getMessageSubject().subscribe((txt: string) => {
   // (3) ou on souscrit soi-même à l'observable (on écoute l'observable)
23.       //
24.       // ici, on peut appliquer divers traitements
25.       //
26.       this.message = txt;                                                    // (3) et on
 affecte à la variable du composant : dataBSubject3 la valeur reçue de l'observable data
27.     });
28.
29.     //
30.     //  avec (2) et (3) on obtient le même résultat, sachez juste qu'il est possible de faire
 de ces deux manières
31.     //
32.   }
33.
34.   getMessage1(): string {                                          // (1) on retourne une
 variable dans un service
35.     return this.stored1Service.getMessage1();                      //     il sera accessible
 par la vue
36.   }
37.
38.   ngOnDestroy(): void {
39.     this.messageSubscription.unsubscribe();                        // (3)  toujours se
 désabonner quand on souscrit manuellement dans un composant
40.   }
41.
42.   choice(nb: number): void {
43.     const data1: IUser = { name: 'Holland', firstname: 'Dexter', genre: 'monsieur'};
44.     const data2: IUser = { name: 'Obel', firstname: 'Agnes', genre: 'madame'};
45.
46.     if (nb === 1) {
47.       this.stored1Service.emitDataSubject(data1);
48.     } else if (nb === 2) {
49.       this.stored1Service.emitDataSubject(data2);
50.     }
51.   }
52. }
```

XIV-C-2-c - comp2.component

- On optera pour la méthode (3) pour des données complexes ou il faudra les traiter avant de les transmettre à la vue.

comp2.component.html

```
1. <div style="background: yellow;">
2.   <p>comp2 works!</p>
3.   <p>(1) réception par variable      -------------> getMessage1()={{getMessage1()}}</p>
4.   <button (click)="cliqueUpdateService()">changer le texte par variable</button>
5.   <br>
6.   <br>
7.   <p>(2) réception par observable -------------> messageSubject={{messageSubject|async}}</p>
8.   <button (click)="cliqueUpdateObservable()">changer le texte par l'observable</button>
9.   <hr>
10.   <button (click)="resetAll()">reset</button>
11.   <hr>
12.
13.   <app-comp3></app-comp3>
14. </div>
```

comp2.component.ts

```
1. import { Component, OnInit } from '@angular/core';
2. import { BehaviorSubject } from 'rxjs';
3. import { Stored1Service } from '../services/stored1.service';
4. import { IUser } from '../models/i-user';
```

```
5.
6. @Component({
7.   selector: 'app-comp2',
8.   templateUrl: './comp2.component.html',
9.   styleUrls: ['./comp2.component.scss']
10. })
11. export class Comp2Component implements OnInit {
12.   messageSubject: BehaviorSubject<string>;                                  // (2)
13.
14.   constructor(private stored1Service: Stored1Service) { }
15.
16.   ngOnInit(): void {
17.     this.messageSubject = this.stored1Service.getMessageSubject();          // (2)
18.   }
19.
20.   getMessage1(): string {                                                   // (1)
21.     return this.stored1Service.getMessage1();
22.   }
23.
24.   cliqueUpdateService(): void {
25.     this.stored1Service.setMessage1('par variable : *** texte changé par le composant: comp2
 ***');                     // (1)
26.   }
27.
28.   cliqueUpdateObservable(): void {
29.     this.stored1Service.getMessageSubject().next('par observable : <<< texte changé par le
 composant: comp2 >>>');     // (2)
30.   }
31.
32.   resetAll(): void {
33.     this.stored1Service.setMessage1('');                                    // (1)
34.     this.stored1Service.getMessageSubject().next('');                       // (2) mauvaise pratique on
 fait le next... dans le composant
35.     this.stored1Service.emitDataSubject(null);                              // (4) bonne pratique : on
 demande au service d'effectuer le next...
36.   }                                                                         //                     car
 c'est le service qui est chargé d'effectuer des actions sur les observables
37. }
```

XIV-C-2-d - comp3.component

comp3.component.html

```
1. <div style="background: azure; margin-left: 24px;">
2.   <p>comp3 works!</p>
3.   <p>(1) réception par variable      -------------> getMessage1()={{getMessage1()}}</p>
4.   <p>(2) réception par observable ------------> messageSubject={{messageSubject|async}}</p>
5.   <p>(4) réception par observable ------------> dataSubject={{dataSubject|async|json}}</p>
6. </div>
```

comp3.component.ts

```
1. import { Component, OnInit } from '@angular/core';
2. import { Stored1Service } from '../services/stored1.service';
3. import { BehaviorSubject, Subject } from 'rxjs';
4. import { IUser } from '../models/i-user';
5.
6. @Component({
7.   selector: 'app-comp3',
8.   templateUrl: './comp3.component.html',
9.   styleUrls: ['./comp3.component.scss']
10. })
11. export class Comp3Component implements OnInit {
12.   messageSubject: BehaviorSubject<string>;                                  // (2)
13.   dataSubject: Subject<IUser>;                                              // (4)
14.
15.   constructor(private stored1Service: Stored1Service) { }
16.
```

```
17.   ngOnInit(): void {
18.     this.messageSubject = this.stored1Service.getMessageSubject();       // (2)
19.     this.dataSubject = this.stored1Service.getDataSubject();             // (4)
20.   }
21.
22.   getMessage1(): string {                                                 // (1)
23.     return this.stored1Service.getMessage1();
24.   }
25. }
```

XIV-C-3 - Résultat

- Vous avez constaté que lorsque l'on modifie les variables dans le service, tous les composants se mettent à jour automatiquement.

XIV-C-4 - À savoir

- Ce qui se passe c'est qu'à chaque modification, Angular déclenche la détection de changement des données pour chaque composant.
- La différence entre la communication service par variable et par observable :

    - **Par variable**

        - Avantages ;
        - très simple : quand on modifie la valeur de la variable du service, tous les composants qui l'affichent mettent à jour la nouvelle valeur dans sa vue.
          Inconvénients :
        - si on utilise des centaines de variables, Angular doit surveiller la modification du moindre changement sur une de ces variables et déclencher la mise à jour dans tous les composants qui l'utilisent. Ça peut donc être coûteux en performances. (Cette technique ne doit donc être utilisée que pour quelques variables.)

    Par observable

        - Avantages :

            - pas besoin d'utiliser la détection de changement de valeur d'Angular. Avec les observables, on émet une valeur et tous les clients observeurs qui ont souscrit reçoivent la nouvelle donnée et la mettent à jour. Pour les performances, c'est beaucoup mieux ;
            - de plus, on peut effectuer une action quand une valeur est reçue par un client qui a souscrit à l'observable (voir point (3)).

        Inconvénients :

            - il faut écrire autant d'observables qu'il y a de variables, mais il y a une astuce. On regroupe les données dans des modèles de données, de toute façon, la plupart du temps, on gère des modèles de données (voir point (4)). On émet le groupe dont une ou des valeurs ont été modifiées. Cela permet de gérer par paquets et donc, si on a des centaines de variables, c'est plus pratique et performant.

                - Remarque : utiliser la technique par regroupement pour le système par variable ne sert à rien parce que Angular doit quand même surveiller la modification de l'ensemble des valeurs des paquets.

XIV-C-5 - Conclusion

- Les composants communiquent en s'échangeant des données à travers un service via la technique de la variable ou de l'observateur.
- Il n'y a qu'une seule instance du service qui est partagée par tous les composants.
- Il faut privilégier la technique par observable (Subject ou BehaviorSubject).
- Subject : à la souscription, attend la prochaine valeur qui sera émise.
- BehaviorSubject : à la souscription, récupère la dernière valeur et attend la prochaine valeur qui sera émise.
- Il faut privilégier : BehaviorSubject pour fournir une valeur à la souscription ou lorsque vous avez du routing.
- En effet, pour le routing, lors d'une émission de donnée sur une page quand vous changez de page la souscription au BehaviorSubject vous enverra la dernière valeur émise (celle de la page précédente).

XV - Accès au composant enfant

- Depuis le composant parent, il est possible d'accéder aux propriétés, aux fonctions et au DOM du composant enfant.

XV-A - Pratique

```
app.component                    composant parent
  child.component                composant enfant
```

```
ng new angular-viewchild1
strict ? NO
routing ? NO
SCSS
```

ng g c child

app.component.html

```
<p><b>parent: app.component.ts</b></p>

<h2>partie 1 : accès à n'importe quel composant enfant</h2>
<app-child #child1></app-child>

<hr>

<h2>partie 2 : accès à n'importe quel élément HTML</h2>
<input #someInput placeholder="Votre langage de dév préféré">
```

app.component.ts

```
import { Component, ViewChild, AfterViewInit, OnInit, ChangeDetectorRef, ElementRef, Renderer2 } from '@angular/core';
import { ChildComponent } from './child/child.component';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent implements AfterViewInit,  OnInit   {
  // partie 1
  @ViewChild(ChildComponent) childComponent: ChildComponent;      // : ChildComponent     --> accès au composant enfant (le contrôleur .ts)
                                                                   // dans la vue il correspond à : <app-child #child1></app-child>
                                                                   //, car il y a qu'une seule balise : <app-child
```

```
14.   // s'il y a plusieurs balises <app-child dans la vue, pour cibler la bonne, il faut
 utiliser cette syntaxe:
15.   // @ViewChild('child1') childComponent: ChildComponent;    // 'child1' == #child1
16.   // et un autre composant: @ViewChild('child2') childComponent: ChildComponent;   // avec
 dans la vue :  <app-child #child2></app-child>
17.
18.
19.   @ViewChild('child1', { read: ElementRef, static: false }) childElementRef: ElementRef;
  // : ElementRef    --> accès au DOM
20.                                                                                    //
 donc on peut faire : this.childElementRef.nativeElement.__(élément DOM)___
21.   // partie 2
22.   @ViewChild('someInput') someInput: ElementRef;                  // 'someInput'    -->
 correspond à : #someInput dans la vue
23.                                                                   // : ElementRef    --> de
 type elementRef (accès au DOM)
24.
25.   constructor(private cd: ChangeDetectorRef) {  }
26.
27.   // composant enfant :  depuis le composant parent, ViewChild permet d'accéder aux
 propriétés, aux fonctions et au DOM du composant enfant
28.   // ngAfterViewInit : pour accéder et modifier un composant enfant
29.
30.   ngOnInit(): void {  }
31.
32.
33.   ngAfterViewInit(): void {                              // ngAfterViewInit :  tous les
 composants enfants ont été initialisés et vérifiés
34.     // partie 1 : accès au composant enfant
35.     console.log("*** cycle ngAfterViewInit");
36.
37.     // accès à une variable
38.     console.log(this.childComponent.data);               // c'est le bon endroit pour accéder
 à un composant, car il a été initialisé
39.                                                          // on accède bien au composant enfant
 et à sa propriété : data
40.     // accès à une fonction
41.     this.childComponent.myFunctionChild();               // on accède à une fonction du
 composant enfant
42.
43.     // setter une variable
44.     this.childComponent.data = 'modifié par le parent';
45.     this.cd.detectChanges();                             // IMPORTANT : après modification,
 toujours lancer la détection
46.                                                          // sinon on a une erreur de type :
 Expression has changed after it was last checked
47.                                                          // cela est dû au fait que quand on
 modifie une variable directement, le système de détection est dans la confusion
48.                                                          // quelle est la bonne valeur ?
 Celle-ci ou celle du composant enfant
49.                                                          // donc on lance la détection pour
 dire que c'est celle-ci
50.
51.     // allez voir dans la console le DOM que l'on peut modifier
52.     console.log(this.childElementRef.nativeElement);
53.
54.     // on modifie la couleur du composant enfant
55.     this.childElementRef.nativeElement.style.color = 'white';     // à éviter, utiliser
 plutôt la technique des directives pour modifier du DOM
56.
57.     // partie 2: accès au DOM depuis n'importe quel élément
58.     console.log(this.someInput);                                          // aller voir le
 contenu dans la console
59.     this.someInput.nativeElement.value = 'du parent';                     // on modifie
 directement la value de la balise: input
60.   }
61. }
```

XV-B - Remarques

- ViewChild
  - permet de récupérer n'importe quelle balise enfant du composant.
- ElementRef
  - est la classe qui représente le DOM de n'importe quel composant ou élément HTML ;
  - c'est donc cet ElementRef que l'on manipule pour accéder ou modifier le DOM de l'élément.
- ChildComponent
  - est la classe du composant, on peut donc accéder aux propriétés et aux fonctions de ce composant.

/child/child.component.html

```
<div style="background: brown; margin-left: 24px; z-index: 10; padding: 12px;">
  <p>child works!</p>
  <p><b>enfant: child.component.ts</b></p>

  <button (click)="data = 'modifié par l\'enfant'">modifier data</button>
  <p>data = {{data}}</p>
</div>
```

/child/child.component.ts

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

@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrls: ['./child.component.scss']
})
export class ChildComponent implements OnInit {
  data = 'initialisation from child';          // une variable, ici,  dans le composant enfant
                                                // le composant parent pourra accéder à cette
 variable
   constructor() { }

   ngOnInit(): void {  }

   myFunctionChild(): void {                     // une fonction
     console.log('myFunctionChild()');           // le composant parent va accéder à cette
 fonction
   }
}
```

XV-C - Conclusion

- Depuis le composant parent, avec @ViewChild :
  - on peut accéder à n'importe quel composant enfant ;
  - on peut accéder aux variables, aux fonctions et aux DOM de n'importe quel composant enfant.
- Les accès et modifications doivent se faire dans le cycle ngAfterViewInit().

XVI - API WEB (récupération de données) et les fichiers d'environnement (DEV et PROD)

- Nous allons voir comment récupérer des données en interrogeant une API externe.
- Nous aurons donc besoin d'un service API qui retourne du JSON, pour cela il existe une API en ligne où l'on peut faire des tests de récupération.

- **https://jsonplaceholder.typicode.com**.
- Sur votre navigateur : **https://jsonplaceholder.typicode.com/todos/1** vous verrez que le JSON du todo ayant id=1 est affiché en JSON.

Avant de voir l'API, faisons un petit tour sur les environnements DEV et PROD, car on va en avoir besoin.

XVI-A - Environnements DEV et PROD

XVI-A-1 - À savoir

- Angular propose deux environnements par défaut : environment.ts (environnement de dév) et environment.prod.ts (de production donc).
- Quand on lance :

```
1. ng serve          // c'est l'environnement de dev qui est lancé donc c'est le fichier
 environment.ts  qui est pris en compte
2. ng serve --prod   // c'est l'environnement de prod qui est lancé donc c'est le fichier
 environment.prod.ts  qui est pris en compte
3. ng build --prod   // compile tout le projet et envoie le résultat dans le dossier: /dist pour
 le déploiement, donc c'est le fichier environment.prod.ts  qui est pris en compte
```

XVI-A-2 - Pratique

- Nous allons créer un composant qui sera chargé de récupérer une liste de todos ou un todo précis.
- Nous allons voir différentes manières et astuces pour y arriver.
- J'ai décidé de faire de cette fonctionnalité un module afin qu'il soit exporté et qu'il puisse être utilisé dans un autre module/fonctionnalité

```
1. ng new angular-api1
2. strict ? NO
3. routing ? NO
4. SCSS
```

```
1. ng g m todo
2. ng g c /todo/todo-display
3. ng g i /todo/models/todo
4. ng g s /todo/services/todo-http
```

- Dans le fichier environnement, on indique divers paramètres du projet. C'est en quelque sorte là où on met les variables globales d'un projet.
- Nous allons mettre dans ce fichier l'URL de l'API externe

/environments/environment.prod.ts

```
1.   urlApi: 'https://jsonplaceholder.typicode.com',
```

/environments/environment.ts

```
1.   urlApi: 'https://jsonplaceholder.typicode.com',           // normalement en dev, on met une
 URL d'un serveur de dev ou de test
2.                                                            //, mais comme je n'en ai pas sous
 la main et que de toute façon c'est un faux serveur de prod, on utilise le même que celui en
 prod
```

/todo/todo.module.ts

```
1. import { CommonModule } from '@angular/common';
```

```
2. import { HttpClientModule } from '@angular/common/http';
3. import { TodoDisplayComponent } from './todo-display/todo-display.component';
4.
5. @NgModule({
6.   declarations: [TodoDisplayComponent],
7.   imports: [
8.     CommonModule,
9.     HttpClientModule,                          // on a besoin du package Http pour effectuer
 nos requêtes à l'API
10.   ],
11.   exports: [TodoDisplayComponent],            // on exporte le composant afin que depuis un
 autre module on puisse l'importer et l'utiliser
12.   providers: [  ],
13. })
14. export class TodoModule { }
```

XVI-A-3 - Remarques

- Comprenez bien le fait d'avoir créé un module pour le composant :
    - ça le rend exportable ;
    - dans ce module : todo.module, on importe le module : HttpClientModule, car seuls les composants de ce module ont besoin du package HttpClientModule (en considérant que les autres fonctionnalités (ou modules) du projet n'en ont pas besoin).

XVI-B - Le modèle de données

/todo/models/todo.ts

```
1. export interface Todo {
2.   userId: number;
3.   id: number;
4.   title: string;
5.   completed: boolean;
6. }
```

XVI-C - Remarque

- Pour connaitre les propriétés à écrire, nous nous basons sur ce que nous envoie l'API pour un todo, voir : **https://jsonplaceholder.typicode.com/todos/1**.

XVI-D - Le service

- Un service qui ne va contenir uniquement que des requêtes HTTP pour interroger l'API et donc nous le nommons : todo-http.service.ts.
- Si besoin, on pourrait écrire un autre service pour y mettre du code métier autre.
- Bonne pratique : un service = code métier du même thème

/todo/services/todo-http.service.ts

```
1. import { Injectable } from '@angular/core';
2. import { environment } from '../../../environments/environment';  // importer le fichier
 d'environnement, car on en a besoin
3.                                                                   // automatiquement, si on
 est en développement ou en production
4.                                                                   // est chargé soit le
 fichier environment.ts soit le fichier : environment.prod.ts
5. import { HttpClient } from '@angular/common/http';
6. import { Observable } from 'rxjs';
7. import { Todo } from '../models/todo';
```

```
8.
9. @Injectable({
10.   providedIn: 'root'
11. })
12. export class TodoHttpService {
13.
14.   private urlApi: string = environment.urlApi;            // on récupère l'URL du fichier
 d'environnement (dev ou prod)
15.
16.   constructor(private http: HttpClient) {                 // utiliser le package HTTP
 d'Angular
17.   }
18.
19.   getTodos(): Observable<Todo[]> {                        // la fonction: getTodos() accepte
 en retour un observable ayant pour type de données : Todo[]
20.
21.     return this.http.get<Todo[]>(`${this.urlApi}/todos`); // l'observable que l'on récupère
 du get doit contenir des données du type : Todo[]
22.   }
23.
24.   getTodo(id: number): Observable<Todo> {                 // ici, c'est un simple : Todo
25.
26.     return this.http.get<Todo>(`${this.urlApi}/todos/${id}`);
27.   }
28.
29.   getTodosWithAny(): Observable<any> {                    // any est à éviter
30.                                                           // il faut toujours utiliser un
 typage
31.     return this.http.get(`${this.urlApi}/todos`);
32.   }
33. }
```

XVI-E - Composant

/todo/todo-display.component.html

```
1. <p>comp-a1 works!</p>
2.
3. <h3>par observable</h3>
4. <p>A1: todo1 = {{todo1$ | async | json}}</p>          <!-- | async: souscrit à l'observable et
 récupère les données -->
5. <p>A1: todo1.title = {{(todo1$ | async)?.title}}</p>  <!-- (todo1$ | async)    d'abord on
 souscrit -->
6.                                                       <!-- ?.title             ensuite on
 accède à la propriété -->
7.                                                       <!-- ? (facultatif: permet d'éviter une
 erreur si todo1$ est null) -->
8.
9.
10.
11. <h3>par variable</h3>
12. <p>A2: <ng-container *ngIf="todo2">todo2 = {{todo2 | json}}</ng-container></p>   <!-- |json :
 convertit un objet en texte -->
13. <p>A2: todo2.title = {{todo2?.title}}</p>
14.                                     <!-- pour un objet on utilise un *ngIf (*ngIf="todo2")
 -->
15.                                     <!-- et pour une propriété d'un objet, on utilise
 '?' (todo2?.title) -->
16.                                     <!-- todo2 est asynchrone, il met quelques ms pour
 avoir une réponse de l'API -->
17.                                     <!-- durant ces ms, la vue pense que todo2 est
 undefined et affiche une erreur -->
18.
19. <p>en utilisant *ngIf (sans utiliser ?)</p>    <!-- à la place d'utiliser le '?' :
 {{todo2?.title}}, une autre façon est d'utiliser *ngIf="..."" -->
20. <ng-container *ngIf="todo2">A2: todo2.title = {{todo2.title}}</ng-container>   <!--
 *ngIf="todo2"  si todo2 existe -->
21.
22. <hr>
23.
```

```
<h2>les todos de l'utilisateur : 6</h2>
<h3>B1 : par observable</h3>
<ul>
  <li *ngFor="let todo of todosByUser$ | async">todo.title = {{todo.title}}</li>
</ul>

<h3>B2 : par variable</h3>
<ul>
  <li *ngFor="let todo of todosByUser">todo.title = {{todo.title}}</li>
</ul>

<hr>

<h2>Cas particulier : liste de todos qui est filtrée dans la vue</h2>                <!-- on filtre par la vue (on aurait pu filtrer directement à la source) -->
<h2>B1 : par observable les todos de l'utilisateur: 6 ayant été complétés</h2>       <!-- mais c'est pour l'exemple -->
<ul>
  <ng-container *ngFor="let todo of todosByUser$ | async">        <!-- ng-container est une balise "fantôme", juste un "container" (n'est pas inséré dans le DOM) -->
                                                                  <!-- il sert à ne pas casser la structure <ul> / <li> -->
    <li *ngIf="todo.completed">todo.title = {{todo.title}}</li>   <!-- on filtre avec le *ngIf -->
  </ng-container>                                                 <!-- pas besoin du '?' puisqu'on utilise un *ngIf -->
</ul>

<hr>

<h2>C1 : par observable: les todos de l'utilisateur (Avec: Any)</h2>       <!-- avec any, ça fonctionne, mais il faut éviter et utiliser le typage -->
<ng-container *ngFor="let todo of todosWithAny$ | async">
  <p *ngIf="todo.completed">todo.title = {{todo.title}}</p>
</ng-container>
```

XVI-E-1 - Remarques

- On utilise *ngIf=…« … » pour tester la variable avant l'affichage.
- Ou on utilise : '?' avec monObjet?.maPropriete. '?' : indique d'effectuer un test d'existence pour les données asynchrones (requêtes HTTP).
- La balise : `<ng-container ...>` est une balise neutre, on aurait pu utiliser un `<div ....>` avec une balise : `<div>`

```
<div *ngFor="let todo of todosWithAny$ | async">
  <p *ngIf="todo.completed">todo.title = {{todo.title}}</p>
</div>
```

- On aurait dans le DOM :

```
<div><p>titre 1</p></div>
<div><p>titre 2</p></div>
<div><p>titre 3</p></div>
```

- avec une balise : `<ng-container>`

```
<ng-container *ngFor="let todo of todosWithAny$ | async">
  <p *ngIf="todo.completed">todo.title = {{todo.title}}</p>
</ng-container>
```

- on aurait dans le DOM :

```
<p>titre 1</p>
<p>titre 2</p>
<p>titre 3</p>
```

ng-container = aucune balise = juste un container qui n'a pas d'impact dans le DOM

/todo/todo-display.component.ts

```
1. import { Component, OnDestroy, OnInit } from '@angular/core';
2. import { Observable, Subscription,  } from 'rxjs';
3. import { map } from 'rxjs/operators';
4. import { TodoHttpService } from '../services/todo-http.service';
5. import { Todo } from '../models/todo';
6.
7. @Component({
8.   selector: 'app-todo-display',
9.   templateUrl: './todo-display.component.html',
10.   styleUrls: ['./todo-display.component.scss']
11. })
12. export class TodoDisplayComponent implements OnInit, OnDestroy {
13.
14.   //  un todo par son id
15.   todo1$: Observable<Todo>;              // A1 : la donnée de l'observable doit être de
 type: Todo
16.   todo2: Todo;                           // A2 : la variable doit être du type: Todo
17.   subTodo2: Subscription;                // A2 : pour le désabonnement
18.
19.   // les todos appartenant à un utilisateur
20.   todosByUser$: Observable<Todo[]>;      // B1 : doit être un observable dont les données
 doivent être un tableau de type Todo[]
21.   todosByUser: Todo[];                   // B2 : la variable doit être un tableau de type
 Todo[]
22.   subTodosByUser: Subscription;          // B2 : pour le désabonnement
23.
24.   // tous les todos (de tous les utilisateurs)
25.   todosWithAny$: Observable<any>;        // C1 : à éviter avec 'any'. Il faut utiliser le
 typage avec des modèles
26.   todos$: Observable<Todo[]>;            // D1
27.   todos: Todo[];                         // D2
28.   subTodos: Subscription;                // D2 : pour le désabonnement
29.
30.   constructor(private todoHttpService: TodoHttpService) { }
31.
32.   ngOnInit(): void {                     // c'est dans ngOnInit qu'on initialise les données
33.
34.     // ------------------------------------------------------
35.     // un todo par son id
36.     const userId1 = 2;
37.
38.     // A1 : retourne un observable
39.     this.todo1$ = this.todoHttpService.getTodo(userId1);    // A1
40.                                                             // todo1$ est passé à la vue sous
 forme d'observable
41.                                                             // c'est la vue qui va souscrire
 à l'observable via le pipe async
42.     // A2 : retourne les données brut
43.     this.subTodo2 = this.todoHttpService.getTodo(userId1).subscribe((todo: Todo) => {   // A2
44.       this.todo2 = todo;                                                                //
 todo2 est passé à la vue sous forme de données brutes
45.     });
46.
47.     // ------------------------------------------------------------------------
48.     // les todos appartenant à un utilisateur
49.     // cas d'utilisation : si l'API envoi qu'une liste de todos et qu'on ne puisse pas
 récupérer un todo par l'id
50.     const userId2 = 6;
51.
52.     // B1 : retourne un observable
53.     this.todosByUser$ = this.todoHttpService.getTodos().pipe(     // B1
54.       map((todos: Todo[]) =>                                      // récupère les todos[]
55.         todos.filter((todo: Todo) => todo.userId === userId2)     // pour chaque élément :
 todo on filtre avec la condition: todo.userId === 6
56.       ));                                                         // retourne un observable
 (contenant les données qui ont été filtrées)
57.
```

```
58.     // B2 : retourne les données brutes
59.     this.subTodosByUser = this.todoHttpService.getTodos().pipe(    // B2
60.       map((todos: Todo[]) =>
61.         todos.filter((todo: Todo) => todo.userId === userId2)
62.       )
63.     ).subscribe((todos: Todo[]) => this.todosByUser = todos );     // retourne des données
 brutes
64.
65.     // ---------------------------------------------------------------------------
66.     // tous les todos (de tous les utilisateurs)
67.
68.     // C1 : retourne un observable
69.     this.todosWithAny$ = this.todoHttpService.getTodosWithAny();  // C1
70.
71.     // D1 : retourne un observable
72.     this.todos$ = this.todoHttpService.getTodos();                // D1
73.
74.     // D2 : retourne les données brutes
75.     this.subTodos = this.todoHttpService.getTodos().subscribe((todos: Todo[]) => {  // D2
76.       this.todos = todos;
77.     });
78.   }
79.
80.   ngOnDestroy(): void {                          // il faut toujours se désabonner aux
 observables que l'ont a souscrits manuellement
81.     this.subTodo2.unsubscribe();                 // A2
82.     this.subTodosByUser.unsubscribe();           // B2
83.     this.subTodos.unsubscribe();                 // D2
84.   }
85. }
```

XVI-E-2 - Remarques

- Les méthodes d'interrogation de l'API via des requêtes HTTP sont des observables, car on obtient les données quelques millisecondes plus tard : c'est asynchrone.
- On peut soit envoyer un observable à la vue pour qu'il se charge automatiquement de souscrire.
- Ou on peut souscrire soi-même à l'observable et envoyer les données brutes à la vue.
- À savoir : syntaxe raccourcie du constructeur :

```
1. constructor(private todoService: TodoService) {
2. }
3. // on accède dans les fonctions avec : this.todoService.
```

équivaut à :

```
1. private todoService: TodoService
2.
3. constructor(todoService: TodoService) {
4.   this.todoService = todoService;
5. }
6. // dans les fonctions on accède avec this.todoService.
```

app.module.ts

```
1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { AppComponent } from './app.component';
4. import { TodoModule } from './todo/todo.module';
5.
6. @NgModule({
7.   declarations: [
8.     AppComponent
9.   ],
10.   imports: [
11.     BrowserModule,
12.     TodoModule,                    // importer le module : todo.module.ts
```

```
13.   ],                                // pour pouvoir utiliser le composant (qui a été rendu
 exportable)
14.   providers: [],
15.   bootstrap: [AppComponent]
16. })
17. export class AppModule { }
```

app.component.html

```
1. <app-todo-display></app-todo-display>       <!-- le composant : app-todo-display appartient au
 module: todo.module.ts -->
2.                                             <!-- on peut l'utiliser dans app parce qu'on a
 importé le module en question -->
```

XVI-E-3 - Remarques

- Nous importons le module : todo.module.ts là où on a besoin de sa fonctionnalité.

XVI-F - Résultat

Voyez les résultats des différents points…

XVII - SCSS

Maintenant, voyons comment est géré le CSS dans les composants.

XVII-A - À savoir sur le SCSS :

- Le SCSS est une amélioration du css.
- Le SCSS est converti en CSS (car seul le CSS est compris par le navigateur).
- Le SCSS apporte une écriture plus puissante, utilisation de variables…

XVII-B - Le css et un projet Angular

- Chaque composant dispose de son propre css.
- Il existe une classe CSS en global : styles.scss. Le CSS dans ce fichier est accessible depuis tous les composants du projet.
- Il faut éviter d'utiliser le fichier : styles.css. C'est une question d'organisation, ce fichier ne doit pas être un fourre-tout pour tous les composants.

XVII-C - Pratique

- Dans le projet, nous mettons une même classe css : 'bg-1' dans le fichier global styles.scss, 'bg-1' dans comp1.component.scss et 'bg-1' dans comp2.component.scss.
- Nous allons voir comment se comporte l'ensemble du projet quand une classe est nommée de façon identique partout.

```
1. ng new angular-scss1
2. strict ? NO
3. routing ? YES
4. SCSS
```

```
1. ng g c comp1
2. ng g c comp2
3. ng g c comp3
```

app.component.html

```
<app-comp1></app-comp1>
<hr>
<app-comp2></app-comp2>
<hr>
<app-comp3></app-comp3>
```

/src/styles.scss

```
.bg-global-almond {
  padding: 12px;
  background: blanchedalmond;
}

.bg-1 {
  padding: 12px;
  background: magenta;
  color: black;
   border: 1px solid black;
}
```

/comp1/comp1.component.html

```
<p>comp1 works!</p>
<div class="bg-global-almond">classe css : bg-global-almond   du fichier global : /src/
style.scss</div>
<div class="bg-1">prise en compte de la classe css : bg-1 du composant :
 comp1.component.scss</div>
```

/comp1/comp1.component.scss

```
.bg-1 {
  padding: 12px;
  background: blue;
  color: white;
}
```

XVII-C-1 - Remarques pour comp1.component

- Une classe css : 'bg-1' est définie dans comp1.component.scss.
- Il a le même nom : 'bg-1' que celui du composant comp2 et également du fichier global styles.scss.
- Sachez que la classe : 'bg-1' du composant comp1 a la priorité sur celui du global styles.scss

/comp2/comp2.component.html

```
<p>comp2 works!</p>

<div class="bg-1">prise en compte de la classe css : 'bg-1' du composant :
 comp2.component.scss</div>
```

/comp2/comp2.component.scss

```
.bg-1 {
  padding: 12px;
  background: green;
  color: white;
}
```

XVII-C-2 - Remarques pour comp2.component

- Une classe CSS : 'bg-1' est définie dans : comp2.component.scss.
- Il a le même nom : 'bg-1' que celui du composant comp1 et du fichier global : styles.scss.
- Sachez que la classe : 'bg-1' du composant comp2 a la priorité sur celui du global : styles.scss

/comp3/comp3.component.html

```
1. <p>comp3 works!</p>
2.
3. <div class="bg-1">prise en compte de la classe css : 'bg-1' du fichier global : styles.scss
 (car 'bg-1' est absent du composant : comp3.component.scss)</div>
```

/comp3/comp3.component.scss

XVII-C-3 - Remarques pour comp3.component

- Ici, aucune classe CSS : 'bg-1' est définie dans le composant : comp3.component.scss.
- Résultat : c'est la classe CSS : 'bg-1' du fichier global : styles.scss qui est prise en compte.

XVII-D - Résultat

Plus qu'à constater les résultats à l'écran.

XVII-E - Conclusion

- La classe CSS du composant a la priorité absolue.
- Si un composant ne trouve pas la classe CSS dans son fichier… component.scss, alors il va la chercher dans le fichier global styles.scss.

XVIII - Routing

Comment écrire le routing sur un projet Angular ?

XVIII-A - À savoir

- Quand on parle de routing, on parle de page, on associe une page à une URL.
- Sachez qu'une page n'est rien de plus qu'un composant (un composant qui est conçu comme une page : son header, son contenu, son footer…).
- Le routing consiste à associer une URL à un composant (ou page).
- Quand on demande de changer de page via une URL, c'est sa page (composant) correspondante qui est projetée en avant (et remplace l'ancienne page).
- Les pages (composants) sont projetées dans la balise : `<router-outlet></router-outlet>` du composant racine app.component.html.

XVIII-B - Pratique

- On va voir les principales manières de gérer le routing dans un projet.

```
1. ng new angular-routing1
2. strict ? NO
3. routing ? YES
```

```
4. SCSS
```

```
1. ng g c comp1
2. ng g c comp1/comp11m
3. ng g c comp1/comp12m
4. ng g c comp1u
5. ng g c comp2
6. ng g c comp3
7. ng g c page-not-found
```

XVIII-C - Schéma

```
1.     app.component.html
2.       <router-outlet></router-outlet>
3.
 ----------------------------------------------------------------------------------------------------------------------------------------

4.     page comp1                          /comp1                le composant: comp1 sera projeté
 dans: <router-outlet></router-outlet> de : app.component.html
5.       <router-outlet></router-outlet>
6.                                         /comp1/comp11m        le composant: comp11m sera
 projeté dans: <router-outlet></router-outlet> de : comp1.component.html
7.                                         /comp1/comp12m        le composant: comp12m sera
 projeté dans: <router-outlet></router-outlet> de : comp1.component.html
8.     page comp1u                         /comp1/comp1u         le composant: comp1u sera
 projeté dans: <router-outlet></router-outlet> de : app.component.html
9.     page comp2                          /comp2                le composant: comp2 sera projeté
 dans: <router-outlet></router-outlet> de : app.component.html
10.     page comp3                          /comp3/5              le composant: comp3 sera
 projeté dans: <router-outlet></router-outlet> de : app.component.html
11.
12.     remarques:
13.     dans le composant: comp1 on projette soit comp11m avec l'URL : /comp1/comp11, soit :
 comp12m avec l'url : /comp1/comp12m (pas les 2 en même temps)
14.     donc ça peut être utile quand on veut obligatoirement associer une zone d'un composant à
 une URL
15.     par exemple, on veut qu'un système d'onglets fonctionne ainsi : l'onglet 1 est associé à
 l'URL : /comp1/comp11m, l'onglet 2 est associé à l'URL : /comp1/comp12m
```

app-routing.module.ts

```
1. import { Routes, RouterModule } from '@angular/router';
2. import { Comp1Component } from './comp1/comp1.component';
3. import { Comp2Component } from './comp2/comp2.component';
4. import { Comp3Component } from './comp3/comp3.component';
5. import { Comp11mComponent } from './comp1/comp11m/comp11m.component';
6. import { Comp12mComponent } from './comp1/comp12m/comp12m.component';
7. import { Comp1uComponent } from './comp1u/comp1u.component';
8. import { PageNotFoundComponent } from './page-not-found/page-not-found.component';
9.
10. const routes: Routes = [
11.   { path: 'comp1', component: Comp1Component,       // comp1 possède des enfants, donc il
 faudra mettre <router-outlet></router-outlet> dans comp1.component.html
12.     children: [
13.       { path: 'comp11m', component: Comp11mComponent },           // sera projeté dans
 <router-outlet> de : comp1.component.html
14.       { path: 'comp12m', component: Comp12mComponent },           // sera projeté dans
 <router-outlet> de : comp1.component.html
15.     ]
16.   },
17.   { path: 'comp1/comp1u', component: Comp1uComponent },         // une URL à 2 niveaux : /
comp1/comp1u associée à un composant
18.   { path: 'comp2', component: Comp2Component },                 // une URL à 1 niveau : /
comp2 associée à un composant
19.   { path: 'comp3/:comp3Id', component: Comp3Component },        // ':' pour indiquer que
 c'est une variable qui peut prendre n'importe quelle valeur
20.   { path: '',   redirectTo: '/comp1', pathMatch: 'full' },      // en l'absence d'URL sur le
 navigateur, on redirige vers : /comp1
```

```
21.   { path: '**', component: PageNotFoundComponent },            // une route inconnu -->
 route for a 404 page
22. ];
23. @NgModule({
24.   imports: [RouterModule.forRoot(routes)],
25.   exports: [RouterModule]
26. })
27. export class AppRoutingModule { }
```

app.component.html

```
1. <!-- dans le fichier: app-routing.module.ts il y a l'association: url/composant -->
2. <!-- donc si on a l'url, on a son composant -->
3.
4. <nav>
5.   <ul>
6.     <li><a [routerLink]="['/comp1']">aller à la page: /comp1</a></li>
  <!-- /comp1, son composant(ou page) sera projeté dans le <router-outlet> du composant racine :
 app.component.html, donc ici même (voir plus bas) -->
7.
8.     <li><a [routerLink]="['/comp1/comp11m']">aller à la page mixte : /comp1/comp11m</a></li>
  <!-- /comp1, son composant(ou page) sera projeté dans le <router-outlet> du composant racine:
 app.component.html, donc ici même (voir plus bas) -->
9.                                                                                    <!-- /
comp11m, son composant sera projeté dans le <router-outlet> qui se trouve dans le composant
 précédent : comp1.component.html (voir fichier) -->
10.
11.     <li><a [routerLink]="['/comp1/comp12m']">aller à la page mixte : /comp1/comp12m</a></li>
  <!-- /comp1, son composant(ou page) sera projeté dans le <router-outlet> du composant racine:
 app.component.html, donc ici même (voir plus bas) -->
12.                                                                                     <!-- /
comp12m, son composant sera projeté dans le <router-outlet> qui se trouve dans le composant
 précédent : comp1.component.html (voir fichier) -->
13.
14.     <li><a [routerLink]="['/comp1/comp1u']">aller à la page : /comp1/comp1u</a></li>
  <!-- son composant: comp1u.component sera projeté dans le <router-outlet> du composant racine:
 app.component.html, donc ici même (voir plus bas) -->
15.
16.     <li><a [routerLink]="['/comp2']">aller à la page : /comp2</a></li>
   <!-- /comp2, son composant(ou page) sera projeté dans le <router-outlet> du composant racine:
 app.component.html, donc ici même (voir plus bas) -->
17.
18.     <li><a [routerLink]="['/comp3/1999']">aller à la page : /comp3/1999 (on passe la valeur
 1999)</a></li>  <!-- cette fois, nous passons une valeur dans l'url -->
19.
      <!-- la valeur sera récupérée dans le composant : comp3.component.ts -->
20.
21.     <li><a [routerLink]="['/comp3', comp3Id]">aller à la page : /comp3/...via variable
 comp3Id du composant...</a></li>  <!-- comp3Id est une variable du composant actuel:
 app.component.ts (puisqu'on est dans la page: app.component.html) -->
22.
                          <!-- la valeur de comp3Id sera passée au composant : comp3component.ts
 -->
23.   </ul>
24. </nav>
25. <button (click)="goToPageComp3()">aller à la page /comp3/...(génère valeur au hasard)...</
button>
26.
27. <hr>
28.
29. <router-outlet></router-outlet>    <!-- c'est ici que sont injectées les pages -->
```

app.module.ts

```
1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { AppRoutingModule } from './app-routing.module';
4. import { AppComponent } from './app.component';
5. import { Comp1Component } from './comp1/comp1.component';
6. import { Comp11mComponent } from './comp1/comp11m/comp11m.component';
```

```
import { Comp12mComponent } from './comp1/comp12m/comp12m.component';
import { Comp2Component } from './comp2/comp2.component';
import { Comp3Component } from './comp3/comp3.component';
import { PageNotFoundComponent } from './page-not-found/page-not-found.component';
import { Comp1uComponent } from './comp1u/comp1u.component';

@NgModule({
  declarations: [
    AppComponent,
    Comp1Component,
    Comp11mComponent,
    Comp12mComponent,
    Comp2Component,
    Comp3Component,
    PageNotFoundComponent,
    Comp1uComponent,
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,            // le fichier du routing
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  comp3Id = 2024;

  constructor(private router: Router) {}

  goToPageComp3(): void {
    const itemId = Math.floor(Math.random() * Math.floor(2500));

    this.router.navigate(['/comp3', itemId]);
  }
}
```

/comp1/comp1.component.html

```
<h2 style="background: lightsalmon;">comp1 works!</h2>

<router-outlet></router-outlet>          <!-- c'est ici que seront projetés les composants :
 comp11m.component et comp12m.component -->
```

/comp1/comp11m/comp11m.component.html

```
<h2 style="background: lightcyan;">comp11m works!</h2>
```

/comp1/comp12m/comp12m.component.html

```
<h2 style="background: lightgreen;">comp12m works!</h2>
```

/comp3/comp3.component.html

```
<h2>comp3 works!</h2>
```

```
2. <p>id de l'URL={{id}}</p>
```

/comp3/comp3.component.ts

```
1. import { Component, OnInit } from '@angular/core';
2. import { ActivatedRoute } from '@angular/router';
3.
4. @Component({
5.   selector: 'app-comp3',
6.   templateUrl: './comp3.component.html',
7.   styleUrls: ['./comp3.component.scss']
8. })
9. export class Comp3Component implements OnInit {
10.   id: string;
11.
12.   constructor(private route: ActivatedRoute) { }        // le package pour gérer la route
 actuelle, celle qui est activée en ce moment
13.                                                        // forcément dans ce composant, on
 est sur l'URL : /comp3/....
14.   ngOnInit(): void {
15.     this.route.paramMap.subscribe(params => {          // on souscrit au paramMap de
 l'objet : route pour accéder à toutes les informations de celui-ci
16.       this.id = params.get('comp3Id');                 // on récupère : comp3Id que l'on
 affecte à id pour qu'il soit visible dans la vue
17.     });
18.   }
19. }
```

XVIII-D - Conseils

- Tout comme avec les modules, quand on touche au routing il vaut mieux relancer la commande : ng serve

XIX - Formulaires

Pour une gestion des formulaires, on utilise le formBuilder pour créer des formulaires et lui associer des champs. Ensuite, dans la vue, les champs input sont liés aux champs définis dans le contrôleur ci-dessus.

- formBuilder est le créateur de formulaire.
- FormGroup est l'objet qui représente un formulaire.
- FomControl est l'objet qui représente un champ.
- validator est une contrainte de validation.

XIX-A - Les formulaires + la gestion des erreurs + Angular Material

Le projet consiste à créer un formulaire d'enregistrement classique avec l'affichage des messages d'erreurs.

XIX-A-1 - Principes de base

Côté contrôleur :

- un objet : FormGroup peut contenir des objets FormControl et aussi d'autres objets : FormGroup
- un objet : FormControl est un champ sur lequel on peut lui appliquer un validator

```
1. myForm: FormGroup;
2. ...
3. ...
4. this.myForm = this.formBuilder.group({
5.     mycontrol1: ['', Validators.required],
6.     mycontrol2: [''],
```

```
7.
8.      // pour l'exemple : avec mynested1 on imbrique un formulaire dans un autre (nested1 dans
 myForm)
9.      mynested1: this.formBuilder.group({                //  pour cela, on mentionne un noeud
 'mynested1' de type FormGroup
10.         mycontrol11: ['', Validators.required],
11.         mycontrol12: [''],
12.     })
13. });
14. ...
15. onSubmit() {
16. ...
```

Côté vue :

- la balise <form> est le formGroup ;
- chaque champ input de la vue est lié à un champ FormControl du FormGroup par la directive : formControlName.

```
1. <form [formGroup]="myForm" (ngSubmit)="onSubmit()">
2.      <input formControlName = "mycontrol1">
3.      <input formControlName = "mycontrol2">
4.
5.      <div formGroupName="mynested1">                 <!-- c'est important de mentionner le
 noeud d'imbrication : 'mynested1' -->
6.          <input formControlName = "mycontrol11">
7.          <input formControlName = "mycontrol12">
8.      </div>
9.
10.      <button type="submit">Valider</button>
11. </form>
```

XIX-A-2 - Pratique

```
1. ng new angular-form1
2. strict ? NO
3. routing ? NO
4. SCSS
```

Histoire d'avoir un joli formulaire, utilisons le package angular material.

```
1. ng add @angular/material
2. - styles ? NO
3. - animations ? NO
```

```
1. ng g m register --module=app
2. ng g c register/form-register --module=register
3. ng g m shared/material-design --module=register
```

/shared/material-design.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3. // Material
4. import { MatGridListModule } from '@angular/material/grid-list';
5. import { MatInputModule } from '@angular/material/input';
6. import { MatButtonModule } from '@angular/material/button';
7. import { NoopAnimationsModule } from '@angular/platform-browser/animations';
8. import { MatFormFieldModule } from '@angular/material/form-field';
9. import { MatCheckboxModule } from '@angular/material/checkbox';
10.
11. @NgModule({
12.   declarations: [],
13.   imports: [
14.     CommonModule,
```

```
15.
16.     MatFormFieldModule,               // on importe uniquement les composants dont on a
 besoin
17.     MatGridListModule,                // inutile de tout importer
18.     NoopAnimationsModule,
19.     MatInputModule,
20.     MatButtonModule,
21.     MatCheckboxModule,
22.
23.   ],
24.   exports: [
25.     MatFormFieldModule,               // ne pas oublier d'exporter pour qu'il puisse être
 importé dans le module qui le demande
26.     MatGridListModule,
27.     NoopAnimationsModule,
28.     MatInputModule,
29.     MatButtonModule,
30.     MatCheckboxModule,
31.
32.   ]
33. })
34. export class MaterialDesignModule { }
```

app.module.ts

```
1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { AppComponent } from './app.component';
4. import { RegisterModule } from './register/register.module';
5.
6. @NgModule({
7.   declarations: [
8.     AppComponent,
9.   ],
10.   imports: [
11.     BrowserModule,
12.     RegisterModule,                          // on importe le module register pour pouvoir
 utiliser son composant : form-register.component
13.   ],
14.   providers: [],
15.   bootstrap: [AppComponent]
16. })
17. export class AppModule { }
```

app.component.html

```
1. <app-form-register></app-form-register>
```

/register/register.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3. import { FormRegisterComponent } from './form-register/form-register.component';
4. import { ReactiveFormsModule } from '@angular/forms';
5. import { MaterialDesignModule } from '../shared/material-design/material-design.module';
6. import { NoopAnimationsModule } from '@angular/platform-browser/animations';
7.
8. @NgModule({
9.   declarations: [FormRegisterComponent],
10.   imports: [
11.     CommonModule,
12.     ReactiveFormsModule,          // IMPORTANT : le module Angular pour les formulaires
13.                                   // si vous utilisez les formulaires dans plusieurs
 composants
14.                                   // vous pouvez importer ce module à un niveau plus haut,
 par exemple dans : app.module.ts
15.                                   // car il faut éviter d'importer un module dans plusieurs
 endroits
```

```
16.
17.     MaterialDesignModule,          // le module material de : /shared/material-design/
material-design.module.ts
18.     NoopAnimationsModule,          // pas d'animation
19.   ],
20.   exports: [
21.     FormRegisterComponent,         // on exporte le composant pour qu'il soit disponible à la
 racine (qui va l'importer)
22.   ]
23. })
24. export class RegisterModule { }
```

/register/validators/must-match.validator.ts

```
1. import { FormGroup } from '@angular/forms';
2.
3. export function MustMatchValidator(controlName: string, matchingControlName: string) {
  // correspond aux champs: password et confirmPassword
4.     return (formGroup: FormGroup) => {
5.         const control = formGroup.controls[controlName];                              //
 password
6.         const matchingControl = formGroup.controls[matchingControlName];              //
 confirmPassword
7.
8.         if (matchingControl.errors && !matchingControl.errors.mustMatch) {     // si déjà
 trouvé une erreur ailleurs dans un autre champ
9.             return;                                                                    // alors pas
 besoin d'analyser le contrôle des mots de passe
10.         }
11.
12.         if (control.value !== matchingControl.value) {                          // si les 2
 mots de passe ne correspondent pas
13.             matchingControl.setErrors({ mustMatch: true });                         // il y a une
 erreur
14.         } else {
15.             matchingControl.setErrors(null);                                        // sinon il n'y
 a pas d'erreur
16.         }
17.     }
18. }
```

/register/form-register/form-register.component.html

```
1. <p>register-form works!</p>
2.
3. <!--    version simplifiée (sans le design du material)
4.
5. <form [formGroup]="registerForm" (ngSubmit)="onSubmit()">
6.   <div formGroupName="identity">
7.     <select formControlName = "title">
8.       <option value="mr">Mr</option>
9.       <option value="mme">Mme</option>
10.     </select>
11.     <input formControlName = "lastName">
12.     <input formControlName = "firstName">
13.   </div>
14.
15.   <input formControlName = "email">
16.   <input formControlName = "password">
17.   <input formControlName = "confirmPassword">
18.
19.   <button type="reset" (click)="onReset()">Annuler</button>
20.   <button type="submit">S'enregistrer</button>
21. </form>
22.
23. -->
24.
25. <form [formGroup]="registerForm" (ngSubmit)="onSubmit()">           <!-- IMPORTANT: indiquer
 le formulaire de base qui englobe tous les champs -->
26.
```

```
27.   <div id="container">                    <!-- FLEXBOX (voir le fichier .scss) -->
28.
29.     <div class="bloc">                    <!-- 1er élément de FLEXBOX (bloc à gauche) -->
30.       <div formGroupName="identity">      <!-- IMPORTANT : indiquer le noeud d'une
 imbrication du formulaire -->
31.               <div>
32.                 <mat-form-field appearance="fill">
33.                   <mat-label>Civilité</mat-label>
34.                   <select matNativeControl formControlName = "title" [ngClass]="{ 'is-
invalid': submitted && f_identity.title.errors }">
35.                     <option value="mr">Mr</option>
36.                     <option value="mme">Mme</option>
37.                   </select>
38.                   <mat-error *ngIf="submitted && f_identity.title.errors" class="invalid-
feedback">
39.                     <div *ngIf="f_identity.title.errors.required">La civilité
 est <strong>obligatoire</strong></div>
40.                   </mat-error>
41.                 </mat-form-field>
42.               </div>
43.
44.               <div>
45.                 <mat-form-field appearance="fill" class="example-full-width">
46.                   <mat-label>Nom</mat-label>
47.                   <input matInput placeholder = "Entrez votre
 nom" formControlName = "lastName" class="form-control" [ngClass]="{ 'is-invalid': submitted &&
 f_identity.lastName.errors }">
48.                   <mat-error *ngIf="submitted && f_identity.lastName.errors" class="invalid-
feedback">
49.                     <div *ngIf="f_identity.lastName.errors.required">Le nom
 est <strong>obligatoire</strong></div>
50.                   </mat-error>
51.                 </mat-form-field>
52.               </div>
53.
54.               <div>
55.                 <mat-form-field appearance="fill" class="example-full-width">
56.                   <mat-label>Prénom</mat-label>
57.                   <input matInput placeholder = "Entrez votre
 prénom" formControlName = "firstName" class="form-control" [ngClass]="{ 'is-invalid': submitted
 && f_identity.firstName.errors }">
58.                   <mat-error *ngIf="submitted && f_identity.firstName.errors" class="invalid-
feedback">
59.                     <div *ngIf="f_identity.firstName.errors.required">Le prénnom
 est <strong>obligatoire</strong></div>
60.                   </mat-error>
61.                 </mat-form-field>
62.               </div>
63.       </div>
64.     </div>
65.
66.     <div class="bloc">                    <!-- 2ème élément de FLEXBOX (bloc à droite) -->
67.             <div>
68.               <mat-form-field appearance="fill" class="example-full-width">
69.                 <mat-label>email</mat-label>
70.                 <input matInput placeholder = "Entrez votre
 email" formControlName = "email" class="form-control" [ngClass]="{ 'is-invalid': submitted &&
 f.email.errors }">
71.                 <mat-error *ngIf="submitted && f.email.errors" class="invalid-feedback">
72.                   <div *ngIf="f.email.errors.required">L'email est <strong>obligatoire</
strong></div>
73.                   <div *ngIf="f.email.errors.email">L'email doit être dans un format valide</
div>
74.                 </mat-error>
75.               </mat-form-field>
76.             </div>
77.
78.             <div>
79.               <mat-form-field appearance="fill" class="example-full-width">
80.                 <mat-label>Password</mat-label>
```

```
                <input matInput #password placeholder = "mot de
 passe" formControlName = "password" class="form-control" [ngClass]="{ 'is-invalid': submitted &&
 f.password.errors }">
                <mat-hint>{{password.value?.length || 0}} caractère(s) (6 minimum)</mat-hint>
                <mat-error *ngIf="submitted && f.password.errors" class="invalid-feedback">
                  <div *ngIf="f.password.errors.required">Le mot de passe
 est <strong>obligatoire</strong></div>
                  <div *ngIf="f.password.errors.minlength">Le mot de passe doit contenir au
 moins 6 caractères</div>
                </mat-error>
              </mat-form-field>
            </div>

            <div>
              <mat-form-field appearance="fill" class="example-full-width">
                <mat-label>Confirme Password</mat-label>
                <input
 matInput placeholder = "confirmation" formControlName = "confirmPassword" class="form-control"
 [ngClass]="{ 'is-invalid': submitted && f.confirmPassword.errors }">
                <mat-error *ngIf="submitted && f.confirmPassword.errors" class="invalid-
feedback">
                  <div *ngIf="f.confirmPassword.errors.required">La confirmation
 est <strong>obligatoire</strong></div>
                  <div *ngIf="f.confirmPassword.errors.mustMatch">Les mots de passe sont
 différents</div>
                </mat-error>
              </mat-form-field>
            </div>

             <div style="height: 24px;"></div>

             <div>
               <mat-checkbox id="acceptTerms" formControlName = "acceptTerms">Check me!</mat-
checkbox>
               <mat-error *ngIf="submitted && f.acceptTerms.errors" class="invalid-feedback">
                 <div *ngIf="f.acceptTerms.errors.required">Vous devez <strong>accepter</
strong> les termes</div>
               </mat-error>
             </div>

             <div style="height: 24px;"></div>

             <div>
               <button mat-raised-button color="accent" type="reset"
 (click)="onReset()">Annuler</button>
                
               <button mat-raised-button color="primary" type="submit">S'enregistrer</button>
             </div>
     </div>

   </div>

</form>
```

/register/form-register/form-register.component.scss

```
#container {
  display: flex;
  justify-content: center;
}

.bloc {
  margin: 24px;
}
```

/register/form-register/form-register.component.ts

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

```
3. import { MustMatchValidator } from '../validators/must-match.validator';   // import un
 validator personnalisé
4.
5. @Component({
6.   selector: 'app-form-register',
7.   templateUrl: './form-register.component.html',
8.   styleUrls: ['./form-register.component.scss']
9. })
10. export class FormRegisterComponent implements OnInit {
11.
12.   registerForm: FormGroup;          // le groupe de champs pour la vue
13.   submitted = false;                // un indicateur pour savoir si le formulaire à été
 soumis ou pas
14.
15.   constructor(private formBuilder: FormBuilder) { }
16.
17.   ngOnInit(): void {        // c'est dans ngOnInit, qu'on initialise les données nécessaires
 au bon fonctionnement du composant
18.                             // ngOnInit est appelé qu'une seule fois et avant l'affichage de
 la vue
19.
20.       this.registerForm = this.formBuilder.group({                              //
 registerForm = contient des FormControl et FormGroup
21.                                                                                 // et on
 indique les validations sur les champs si besoin
22.         identity: this.formBuilder.group({                                      // une
 imbrication de formulaire avec un noeud de type : FormGroup nommé : 'identity'
23.             title: ['', Validators.required],
24.             firstName: ['', Validators.required],
25.             lastName: ['', Validators.required],
26.         }),
27.
28.         email: ['', [Validators.required, Validators.email]],
29.         password: ['', [Validators.required, Validators.minLength(6)]],
30.         confirmPassword: ['', Validators.required],
31.         acceptTerms: ['', Validators.requiredTrue]
32.       }, {                                                  // on met ici les validators
 personnalisés pour analyser la validité de plusieurs champs
33.           validator: MustMatchValidator('password', 'confirmPassword')   // un validator
 personnalisé:   (voir /register/validators/must-match.validator.ts)
34.                                                             // qui permet de comparer les
 2 champs: 'password' et 'confirmPassword' si identique ou pas ?
35.       });
36.
37.       this.registerForm.get('identity.lastName').setValue('heisenberg');  // on initialise le
 champs: lastName avec la valeur: 'heisenberg'
38.
39.       this.registerForm.valueChanges.subscribe((formValues: any) => { // si le formulaire est
 modifié (quel que soit le champ)
40.         console.log(formValues);                                      // on affiche dans la
 console le formulaire entier
41.       });
42.
43.
  this.registerForm.get('identity.lastName').valueChanges.subscribe((lastName: string) => {   //
 si le champ: lastName est modifié
44.         console.log(lastName);                                                          //
 on affiche dans la console la valeur du champ
45.       });
46.   }
47.
48.   get f_identity() {                                            // astuce: un raccourci
 pour la vue
49.     return this.registerForm['controls']['identity']['controls'];   // au lieu d'écrire à
 chaque fois: registerForm['controls']['identity']['controls']
50.                                                                 // on écrit à la place :
 f_identity
51.   }
52.
53.   get f() {                                     // astuce: un raccourci pour la vue
54.     return this.registerForm.controls;          // au lieu d'écrire à chaque fois:
 registerForm.controls
```

```
55.                                             // on écrit à la place : f
56.     }
57.
58.     onSubmit(): void {                      // à la soumission du formulaire, clique sur le
 bouton: "s'enregistrer"
59.         this.submitted = true;
60.
61.         // on arrête ici si le formulaire est invalide
62.         if (this.registerForm.invalid) {      // s'il y a une erreur, le formulaire est
 invalide
63.             return;
64.         }
65.
66.         // affiche une alerte avec le contenu du formulaire en json
67.         alert('SUCCESS!! :-)\n\n' + JSON.stringify(this.registerForm.value, null, 4));
68.     }
69.
70.     onReset(): void {                 // clique sur le bouton: "annuler"
71.         this.submitted = false;
72.         this.registerForm.reset();     // on efface tout le formulaire
73.     }
74. }
```

XIX-B - Les formulaires dynamiques

Avec les formulaires dynamiques, on peut leur ajouter des champs et en supprimer.

XIX-B-1 - Principes de base

Avec FormArray on peut lui ajouter des champs (FormControl) ou des groupes de champs (FormGroup).

XIX-B-2 - Remarques

Dans cet exemple, nous ne nous occupons pas du design ni des erreurs.

XIX-B-3 - Pratique

- on crée à la volée un certain nombre de champs.
- cliquer sur le bouton '+' pour ajouter un nouveau champ.
- cliquer sur le bouton '-' pour supprimer un champ.

Dans le même projet que le précédent, on va créer un nouveau composant :

```
1. cd angular-form1
2.
3. ng g c dynamic-form1 --module=app
```

On ajoute le module ReactiveFormsModule dans app.module.ts
En effet, le composant dynamic-form1 n'a pas de module, donc il remonte au niveau supérieur pour en trouver un, et dans la hiérarchie c'est app.module

app.module.ts

```
1. import { AppComponent } from './app.component';
2. import { RegisterModule } from './register/register.module';
3. import { ReactiveFormsModule } from '@angular/forms';
4. import { MaterialDesignModule } from './shared/material-design/material-design.module';
5. import { DynamicForm1Component } from './dynamic-form1/dynamic-form1.component';
6.
```

```
@NgModule({
  declarations: [
    AppComponent,
     DynamicForm1Component,      // le formulaire dynamique
  ],
  imports: [
    BrowserModule,
    RegisterModule,             // on importe le module register pour pouvoir utiliser son composant: form-register.component
    ReactiveFormsModule,        // le module Angular pour les formulaires
    MaterialDesignModule,
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

app.component.html

```
<app-form-register></app-form-register>
<hr>
<app-dynamic-form1></app-dynamic-form1>
```

/dynamic-form1/dynamic-form1.component.html

```
<p>dynamic-form1 works!</p>

<form [formGroup]="myForm" (ngSubmit)="onSubmit()">

  <input type="text" formControlName = "name" placeholder = "name">

  <div formArrayName="items">
    <div *ngFor="let item of items.controls; let i=index" class="bg-item">  <!-- comme c'est un tableau on peut définir un index -->
      <input type="text" [formControlName] = "i">                           <!-- et nommer le name de l'input avec le numéro d'index -->

      <button (click)="removeItem(i)"><b>-</b></button>
    </div>
  </div>

  <button (click)="addItem()"><b>+</b></button>
  <button type="submit">Valider</button>
</form>
```

/dynamic-form1/dynamic-form1.component.scss

```
.bg-item {
  margin: 6px;
}
```

/dynamic-form1/dynamic-form1.component.ts

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

@Component({
  selector: 'app-dynamic-form1',
  templateUrl: './dynamic-form1.component.html',
  styleUrls: ['./dynamic-form1.component.scss']
})
export class DynamicForm1Component implements OnInit {

  myForm: FormGroup;

  constructor(private formBuilder: FormBuilder) { }

```

```
15.   ngOnInit(): void {
16.
17.     this.myForm = this.formBuilder.group({
18.       name: new FormControl('', [Validators.required]),
19.       items: new FormArray([])                              // contenir les nouveaux champs
20.     });
21.
22.     for (let i=0; i<2; i++) {                               // on ajoute pour l'exemple 2
 champs
23.       this.addItem();
24.     }
25.   }
26.
27.   get items() {
28.     return this.myForm.get("items") as FormArray;
29.   }
30.
31.   onSubmit() {
32.     alert(JSON.stringify(this.myForm.value));
33.   }
34.
35.   addItem() {
36.     const arrForm = this.myForm['controls'].items as FormArray;          // le contenu de :
 this.myForm['controls'].items est un simple tableau d'objets
37.                                                                          // on veut pouvoir
 accéder aux méthodes : push(), removeAt() qui se trouvent dans FormArray
38.                                                                          // donc on caste le
 contenu avec 'as FormArray'
39.
40.     arrForm.push(                                                       // arrForm est du
 type FormArray donc on peut faire appel à sa méthode push()
41.       new FormControl('', [Validators.required])
42.     );
43.
44.     return false;
45.   }
46.
47.   removeItem(index: number) {
48.     const arrForm = this.myForm['controls'].items as FormArray;
49.     arrForm.removeAt(index);
50.   }
51. }
```

XX - Les filtres (PIPES)

- Dans la vue, on veut pouvoir formater ou appliquer un petit traitement sur les données que l'on affiche.
- Pour cela, Angular propose certains filtres (pipes).
- Mais il est possible de créer ses propres filtres personnalisés.

XX-A - Pratique

```
1. ng new angular-pipe1
2. strict ? NO
3. routing ? NO
4. SCSS
```

```
1. ng g p pipes/only-valid
2. ng g p pipes/low-price
3. ng g p pipes/new-collection
```

app.module.ts

```
1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { AppComponent } from './app.component';
4. import { OnlyValidPipe } from './pipes/only-valid.pipe';
```

```
import { LowPricePipe } from './pipes/low-price.pipe';
import { NewCollectionPipe } from './pipes/new-collection.pipe';

@NgModule({
  declarations: [
    AppComponent,
    OnlyValidPipe,          // les filtres doivent être déclarés
    LowPricePipe ,          //
    NewCollectionPipe,
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

app.component.html

```
<h1>filtres Angular</h1>
<p>(1) now={{now}}</p>
<p>(1) now={{now | date:"dd/MM/yyyy"}}</p>
<p>(2) text1={{text1 | uppercase}}</p>
<p>(3) value1={{value1 | currency:'EUR'}}</p>
<p>(4) object1={{object1 | json | uppercase}}</p>

<hr>

<h1>filtres personnalisés</h1>
<h3>(5) listObject1 avec le filtre personnalisé : onlyValid (est valide)</h3>
<ul>
  <li *ngFor="let obj of listObject1 | onlyValid">{{obj|json}}</li>
 <!-- filtre perso sur : est valide -->
</ul>

<h3>(5) listObject1 avec le filtre personnalisé : lowPrice  (est valide et prix <= 20)</h3>
<ul>
  <li *ngFor="let obj of listObject1 | lowPrice:20:true">{{obj|json}}</li>
 <!-- filtre perso sur le prix est <= 20 et qui est valide -->
</ul>

<h3>(5) listObject1 avec 2 filtres personnalisés : lowPrice et newCollection</h3>
<ul>
  <li *ngFor="let obj of listObject1 | lowPrice:20:true | newCollection: true">{{obj|json}}</
li>  <!-- utilisation de 2 filtres -->
</ul>
```

app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  now = Date();                              // (1)
   text1 = 'hello world';                     // (2)
   value1 = 105.90;                           // (3)
   object1 = { name: 'toto', job: 'dev'};     // (4)
   listObject1 = [                            // (5)
     {name: 'objet1', valid: false, price: 49.90, newCollection: false, },
     {name: 'objet2', valid: true,  price: 19.90, newCollection: true, },
     {name: 'objet3', valid: true,  price: 79.20, newCollection: false, },
   ];
}
```

/pipes/only-valid.pipe.ts

```
1. import { Pipe, PipeTransform } from '@angular/core';
2. @Pipe({
3.   name: 'onlyValid'
4. })
5. export class OnlyValidPipe implements PipeTransform {
6.
7.   transform(values: any, args?: any): any {
8.
9.     return values.filter((value: any) => value.valid);        // filtre sur la propriété valid
 qui doit être à true
10.   }
11. }
```

/pipes/low-price.pipe.ts

```
1. import { Pipe, PipeTransform } from '@angular/core';
2.
3. @Pipe({
4.   name: 'lowPrice'
5. })
6. export class LowPricePipe implements PipeTransform {
7.
8.   transform(values: any, args1: number, args2: boolean): any {  // utilisation dans la vue : |
 lowPrice:20:true
9.     const price = args1;                                        // attention à l'ordre : 20
 est args1 et true est args2
10.     const valid = args2;                                       //
11.
12.     if (price && valid) {
13.       return values.filter((value: any) => value.valid === valid && value.price <= price);
     // filtre sur la propriété valid qui doit être à true
14.     }
15.     return;
16.   }
17. }
```

/pipes/new-collection.pipe.ts

```
1. import { Pipe, PipeTransform } from '@angular/core';
2.
3. @Pipe({
4.   name: 'newCollection'
5. })
6. export class NewCollectionPipe implements PipeTransform {
7.
8.   transform(values: any, args1: number): any {
9.     const newCollection = args1;
10.     return values.filter((value: any) => value.newCollection === newCollection);        //
 filtre sur la propriété newCollection
11.   }
12.
13. }
```

XX-B - Résultat

```
1. ng serve
```

XX-C - Conclusion

- Pour un maximum de performances, il faut privilégier les filtres Angular et des filtres personnalisés.

XXI - Les directives

XXI-A - Description

- les directives sont un moyen très puissant pour manipuler des balises, il faut toujours privilégier cette technique dans votre développement ;
- ils permettent de développer des fonctionnalités dynamique via du code (au lieu de mettre le code directement dans la vue);
- par exemple, on voudrait mettre en vert un article si l'utilisateur connecté est l'auteur, en gris s'il est admin et normal pour le reste. En plus de cela, on voudrait ajouter un bouton d'édition s'il est l'auteur. On pourrait faire ça directement dans la vue avec un système de IF, et si on avait besoin de l'utiliser dans une autre page ? On se retrouverait vite avec du code html complexe. Bien sur, on pourrait résoudre ça via des composants, c'est parfaitement valable mais les directives sont vraiment spécialisés dans cette tâche et donc, profitons en. De plus sachez, qu'une directive est en réalité rien d'autre qu'un composant sans la vue (puisqu'on manipule la vue via le code).

mais encore :

- avec une directive, on peut agir grace a du code sur une balise, on pourra ainsi modifier son style, ajouter une classe, ajouter une balise, du texte... tout est possible ;
- dans une directive, on peut aussi réagir à des évenements (clics...) ;
- il existe deux types de directives, chacune spécialisé dans une tâche :
    - la directive d'attribut : pour gérer le style, les classes ;
    - la directive structurelle : pour gérer la structure du DOM, ajouter des balises, la faire apparaitre ou pas...
- mais sachez que, si besoin, rien n'empeche de gérer également le style ou les classes dans une directive structurelle ;
- la différence est que la directive d'attribut et structurelle aborde la balise sur laquelle elle est appliqué d'une manière différente :
    - avec une directive d'attribut : on accède directement à la balise (pour la modifier) sur laquelle la directive est appliqué ;
    - avec une directive structurelle : on obtient une representation virtuelle de la balise (pour la manipuler) sur laquelle la directive est appliqué. Elle n'apparait donc pas à l'écran, c'est le code qui va décider ce que l'on va en faire ;

XXI-B - Théorie

XXI-B-1 - La directive d'attribut

remarques :

```
1. <p appDirective1="toto"></p>        // sans les crochets   toto est considéré comme du texte.
2.                                     // attention : on ne peut pas faire réference à une
 variable JavaScript
3. <p [appDirective1]="'toto'"></p>    // avec les crochets, c'est interpreté en JavaScript donc
 il faut mettre des accolades
4. <p [appDirective1]="message"></p>   // avec les crochets, c'est interpreté en JavaScript donc
 on peut y mettre une variable (venant de son composant)
5.     ...components.ts
6.       message = 'tutu';
```

utilisation d'une directive sur la balise p

```
1. <p [appArticleBg]="'toto'" otherParam="'tutu'">
```

```
  <span>un article</span>
  <span>le contenu de la balise</span>
</p>
```

app-article-bg.directive.ts

```
...
@Directive({
  selector: '[appArticleBg]'                    // appArticleBg : le nom de la directive à
 utiliser dans la vue
})
export class AppArticleBgDirective {

  @Input() appArticleBg: string;    // "toto"
  @Input() otherParam: string;      // "tutu"

  constructor(
    private el: ElementRef,     // l'élément du DOM sur lequel la directive est appliqué, ici
 c'est l'élément de la balise p

    ...,                        // utiliser l'injection de dépendance
    ...,                        // pour accèder à des services si besoin
  ) { }

  @HostListener('click') onClick() {   // un écouteur sur le clic est positionné sur
 l'élément
    //
    //  on fait un traitement ici si un clic se produit sur l'élément
    //
  }

  ngOnInit() {
    // ici on fait le traitement initial que l'on souhaite
    // si besoin, on utilise les paramètres d'entrées  appArticleBg et otherParam
    // si besoin, on utilise les services du constructeur
    // on agit sur el (qui représente la balise p et son contenu)
    // el est un arbre de noeud, il contient :
    //     le noeud d'origine : p
    //        qui contient : le noeud span (le 1er span)
    //        qui contient : le noeud span (le 2eme span)
    // on peut donc agir sur n'importe quel noeud :
    //    - en modifiant son apparence (style, classe)
    //    - en lui ajoutant du texte, des données ou d'autres noeuds (balises)

    console.log(this.el.nativeElement);  // pour voir l'arbre du noeud p
  }
}
```

XXI-B-2 - La directive structurelle

- on met une * pour une directive structurelle
- la balise de la directive structurelle et son contenu n'est pas affiché, c'est une représentation virtuelle que l'on va se servir dans le code de la directive.
- la balise de la directive structurelle est aussi un contenant ou un emplacement, avec cette representation virtuelle si besoin on va modifier son apparence, son contenu et ceci une fois fait, on va l'inserer ici à cet emplacement et ce, une fois ou même plusieurs fois si besoin. Tout est possible.

utilisation :

```
<div *appUnless="condition">
  Show this sentence unless the condition is true.
</div>
```

```
...
@Directive({ selector: '[appUnless]'})
export class UnlessDirective {
```

```
4.   private hasView = false;
5.
6.   constructor(
7.     private templateRef: TemplateRef<any>,      // la representation virtuelle
8.     private viewContainer: ViewContainerRef,    // le contenant (ou emplacement)
9.
10.     ...,                        // utiliser l'injection de dépendance
11.     ...,                        // pour accèder à des services si besoin
12.   ) { }
13.
14.   @Input() set appUnless(condition: boolean) {
15.     //  accès aux données input
16.     //  accès aux services du constructeur
17.     //
18.     //  traitement de la representation virtuelle : templateRef
19.     //    - en modifiant son apparence (style, classe)
20.     //    - en lui ajoutant du texte, des données ou d'autres noeuds (balises)
21.     //
22.     //  ajout de templateRef dans le contenant : viewContainer
23.     //  c'est à ce moment que le templateRef apparait à l'écran
24.     //
25.   }
26. }
```

XXI-C - Pratique

XXI-C-1 - La directive d'attribut

- les directives d'attributs permettent de manipuler le CSS.
- ngClass, ngStyle sont des directives d'attributs Angular.
- on peut créer ses propres directives d'attributs.
- il faut utiliser le plus possible cette fonctionnalité dans un projet, car c'est très puissant et ça permet d'avoir un impact visuel réduit dans le code de la vue.

XXI-C-1-a - Pratique

- dans la 1re partie, nous allons voir plusieurs manières d'utiliser les directives Angular : ngStyle, style, ngClass, class.
- dans la 2e partie, nous allons créer une directive d'attribut personnalisée.

```
1. ng new angular-directives-attributes1
2. strict ? NO
3. routing ? NO
4. SCSS
5.
6. ng g d directives/highlight
```

app.module.ts

```
1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { AppComponent } from './app.component';
4. import { HighlightDirective } from './directives/highlight.directive';
5.
6. @NgModule({
7.   declarations: [
8.     AppComponent,
9.     HighlightDirective,       // comme les composants, les directives doivent être déclarées
 pour être utilisées
10.   ],
11.   imports: [
12.     BrowserModule
13.   ],
```

```
14.   providers: [],
15.   bootstrap: [AppComponent]
16. })
17. export class AppModule { }
```

app.component.html

```
1. <h1>PARTIE 1: ATTRIBUTES DIRECTIVES ANGULAR</h1>
2.
3. <h2>ngStyle</h2>
4.
5. <p>(1) appliquer plusieurs styles</p>
6. <div [ngStyle]="{'background-color': 'yellow', 'color': 'blue'}">"message in a bottle" (The
 police)</div>
7. <hr>
8.
9. <p>(2) appliquer un style précis</p>
10. <div [style.color]="'red'">"C'est pas ma guerre !" (Rambo)</div>
11. <hr>
12.
13. <p>(3) changer un style précis suivant une condition</p>
14. <button (click)="changeStyle3()">changer le style - flag3={{flag3}}</button>
15. <div [style.color]="flag3 ? colorA : colorB">"C'est des malades !" (Les visiteurs)</div>
  <!-- flag3=true ou false. si true -> la classe : colorA sinon la classe : colorB -->
16.
17. <hr>
18. <h2>NgClass</h2>
19.
20. <p>(4) appliquer simplement une classe</p>
21. <div [className]="'maClasse4'">"Luke, je suis ton père…" (Dark vador)</div>       <!--  voir
 la classe: maClasse4 dans : app.component.css -->
22. <div [className]="dataClasse4">"Luke, je suis ton père…" (Dark vador)</div>       <!--  voir
 la variable dataClasse4 dans : app.component.ts qui contient la classe -->
23. <hr>
24.
25. <p>(5) (une condition)? 'classeX' : 'classeY"</p>
26. <button (click)="this.flag5 = !this.flag5">changer la classe - flag5={{flag5}}</button>
27. <div [className]="(flag5) ? 'maClasse51' : 'maClasse52'">"J'ai les mains faites pour l'or, et
 elles sont dans la merde !" (Scarface)" </div>
28. <hr>
29.
30. <p>(6) une classe précise, (une condition)?</p>
31. <button (click)="changeClasse6()">changer la classe - flag6={{flag6}}</button>
32. <div [class.maClasse6]="flag6">"On se serait shooté à la vitamine C si cela avait été
 illégal" (Trainspotting )</div>
33. <hr>
34.
35. <p>(7) plusieurs classes, plusieurs (conditions)?</p>
36. <button (click)="flag71 = !flag71">changer la 1re condition - flag71={{flag71}}</button>
37. <button (click)="flag72 = !flag72">changer la 2e condition - flag72={{flag72}}</button>
38. <div [ngClass]="{'maClasse71': flag71 == flag72, 'maClasse72': !flag71 && !flag72}">"C'est à
 moi que tu parles ? C'est à moi que tu parles ??..." (Taxi driver)</div>
39.
40. <hr>
41. <hr>
42.
43. <h1>PARTIE 2 : ATTRIBUTES DIRECTIVES PERSONNALISES</h1>
44.
45. <p>(8) <span appHighlight>"survolez-moi!" - par défaut</span></p>
46. <p>(9) <span [appHighlight]="'pink'">"survolez moi!" - en précisant directement une couleur</
span></p>
47. <p>(10) <span [appHighlight]="color">"survolez-moi!" - en précisant une couleur défini dans
 le contrôleur</span></p>
48. <br><br><br>
```

app.component.ts

```
1. import { Component } from '@angular/core';
2.
3. @Component({
```

```
4.   selector: 'app-root',
5.   templateUrl: './app.component.html',
6.   styleUrls: ['./app.component.scss']
7. })
8. export class AppComponent {
9.   flag3 = false;                    // (3)
10.   colorA = 'darkblue';              // (3)
11.   colorB = 'darkslategrey';         // (3)
12.   //
13.   dataClasse4 = 'maClasse4';        // (4)
14.   //
15.   flag5 = false;                    // (5)
16.   //
17.   flag6 = false;                    // (6)
18.   //
19.   flag71 = false;                   // (7)
20.   flag72 = false;                   // (7)
21.   //
22.   color = 'yellow';                 // (10)
23.
24.   changeStyle3() {                  // (3)
25.     this.flag3 = !this.flag3;
26.   }
27.
28.   changeClasse6() {                 // (6)
29.     this.flag6 = !this.flag6;
30.   }
31. }
```

app.component.css

```
1. .maClasse4 {
2.   border: 1px solid red;
3. }
4. .maClasse51 {
5.   background: dodgerblue;
6. }
7. .maClasse52 {
8.   background: gold;
9. }
10. .maClasse6 {
11.   color: darkred;
12. }
13. .maClasse71 {
14.   background: red;
15.   color: white;
16. }
17. .maClasse72 {
18.   background: green;
19.   color: white;
20. }
```

/directives/highlight.directive.ts

```
1. import { Directive, ElementRef, HostListener, Input } from '@angular/core';
2.
3. @Directive({
4.   selector: '[appHighlight]'
5. })
6. export class HighlightDirective {
7.
8.   constructor(private el: ElementRef) {
9.     console.log(el);                                    // allez voir dans la console
 l'élément
10.   }
11.
12.   @Input('appHighlight') highlightColor: string;            // @Input -> récupère les données
 passées à la directive de la vue: [appHighlight]="...données..."
13.
```

```
14.   @HostListener('mouseenter') onMouseEnter() {              // l'événement : 'mouseenter' qui
 correspond au survol de la balise sur laquelle est appliquée la directive
15.     this.highlight(this.highlightColor || 'red');           // prend d'abord le 1er
 paramètre : this.highlightColor
16.   }                                                         // si celui-ci est vide:
 'undefined' alors par défaut il prend le 2e : 'red'
17.
18.   @HostListener('mouseleave') onMouseLeave() {              // l'événement : 'mouseleave'
19.     this.highlight(null);
20.   }
21.
22.   private highlight(color: string) {
23.     this.el.nativeElement.style.backgroundColor = color;    //  on met à jour le background
 de l'élément sur lequel est appliquée la directive
24.   }
25. }
```

XXI-C-1-b - À savoir

```
1. constructor(private el: ElementRef) {
```

Dans le constructeur, on peut utiliser l'injection de dépendance pour récupérer n'importe quels services ou packages dont on peut avoir besoin dans la directive.

el: représente le DOM de l'élément (HTML ou composant) sur lequel la directive est appliquée

```
1. this.el.nativeElement.
```

pour accéder à la lecture ou à la modification de l'élément (HTML ou composant)

```
1. @Input()
```

pour récupérer des données passées en paramètre de la directive

```
1. @HostListener
```

pour gérer les événements de l'élément (HTML ou composant).

XXI-C-1-c - Conclusion

- les directives d'attributs sont un moyen puissant pour gérer le style CSS d'un élément (HTML ou composant)/.
- en mettant une directive [appHighlight]="color" sur un élément : span, on peut faire beaucoup de choses sans surcharger la vue <span [appHighlight]="color">survolez-moi!.
- il faut toujours privilégier l'utilisation des directives.

XXI-C-2 - La directive structurelle

- une directive structurelle permet de modifier la structure du DOM sous certaines conditions et/ou en fonction de certaines valeurs.
- *ngIf est une directive structurelle Angular qui modifie le DOM (ajoute un bout de DOM ou pas en fonction d'une condition).
- *ngFor est une directive structurelle Angular qui modifie le DOM en lui ajoutant des bouts de DOM l'un à la suite de l'autre.
- on peut créer ses propres directives structurelles.
- il faut utiliser le plus possible cette fonctionnalité dans un projet, car c'est très puissant et ça permet d'avoir un impact visuel réduit dans le code de la vue.

XXI-C-2-a - Pratique

- on va créer deux directives :
- une directive pour indiquer si on est authentifié ou pas ;
- une autre directive pour indiquer si un stock est limité ou pas (on considère que le stock est limité quand le nombre est inférieur à 10)

```
ng new angular-directives-structurelles1
strict ? NO
routing ? NO
SCSS
```

```
ng g e enums/shop-params
ng g s services/user
ng g d directives/is-authenticated
ng g d directives/is-stock-limited
```

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { IsAuthenticatedDirective } from './directives/is-authenticated.directive';
import { IsStockLimitedDirective } from './directives/is-stock-limited.directive';

@NgModule({
  declarations: [
    AppComponent,
     IsAuthenticatedDirective,     // ne pas oublier de déclarer les directives
     IsStockLimitedDirective,      //
  ],
  imports: [
    BrowserModule
  ],
  providers: [ , ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

app.component.html

```
<h1>PARTIE 1: DIRECTIVES STRUCTURELLES ANGULAR</h1>

<p>*** (1)</p>
<div *ngIf="isok">from div, it's okey !</div>                              <!-- la balise : div est pris en compte-->

<p>*** (2)</p>
<ng-container *ngIf="isok">from ng-container, it's okey !</ng-container>  <!-- la balise : ng-container n'est pas prise en compte-->

<p>*** (3)</p>
<ng-container *ngIf="isok;then content_ok else content_not_ok"></ng-container>    <!-- ng-container va contenir soit le ng-template #content_ok soit #content_not_ok -->
<ng-template #content_ok>isok={{isok}} - "it's OK"</ng-template>                   <!-- et ce, suivant une condition -->
<ng-template #content_not_ok>isok={{isok}} - "is <b>not</b> OK"</ng-template>

<p>*** (4)</p>
<button (click)="inverseisok()">inverser it's Okey (true/false)</button>

<p>*** (5)</p>
<ul>
   <li *ngFor="let elem of elems;">{{elem}}</li>                                <!--  la balise <li> est répétée avec le ngFor -->
</ul>
```

```
21.
22. <p>*** (6)</p>
23. <p *ngIf="numbs">
24.   numbs=<ng-container *ngFor="let n of numbs">{{n}} </ng-container>           <!--  ng-
container est répété avec le ngFor -->
25. </p>                                                                          <!--  mais ng-
container représente aucune balise -->
26.
27. <hr>
28. <hr>
29.
30. <h1>PARTIE 2: DIRECTIVES STRUCTURELLES PERSONNALISÉES</h1>
31. <span *isAuthenticated>vous êtes authentifié !</span>           <!--  sans paramètre, c'est
 en appelant un service à l'intérieur de la directive      -->
32.                                                                 <!--  que l'on déterminera si
 l'utilisateur est authentifié ou pas -->
33. <br>
34.
35. (7) <span *isStockLimited = "nb1">stock limité 1 !</span>     <!--  "stock limité 1 !"
 s'affiche parce que nb1 est inférieur à 10 -->
36. <br>
37. (8) <span *isStockLimited = "nb2">stock limité 2 !</span>     <!--  "stock limité 2 !" ne
 s'affiche pas parce que nb2 est supérieur à 10 -->
```

XXI-C-2-a-i - Remarques

```
1. *isAuthenticated    modifie la structure du DOM dans le sens ajoute le div et son contenu dans
 le DOM ou pas
2.                       vérifie que l'utilisateur est authentifié, si c'est le cas on affiche un
 message
3.   *isStockLimited     idem
4.                       vérifie la quantité en stock et affiche un message ou pas en fonction du
 nombre
```

app.component.ts

```
1. import { Component } from '@angular/core';
2.
3. @Component({
4.   selector: 'app-root',
5.   templateUrl: './app.component.html',
6.   styleUrls: ['./app.component.scss']
7. })
8. export class AppComponent {
9.   // PARTIE 2
10.   nb1 = 4;          // (7) nous mettons des valeurs en dur
11.   nb2 = 12;         // (8) dans la réalité, ces valeurs sont bien souvent récupérées en
 interrogeant une API REST
12.
13.   // PARTIE 1
14.   isok = true;                                        // (2) (3)
15.
16.   elems = [4, 9, 'toto', 12];                         // (5)
17.   numbs: Array<number> = [10, 50, 6, 18, 32, ];       // (6)
18.
19.   inverseisok() {                                     // (4)
20.     this.isok = !this.isok;
21.   }
22. }
```

XXI-C-2-a-ii - Énumération

- j'ai décidé de mettre des paramètres dans une énumération.
- on aurait pu le mettre dans un fichier d'environnement (voir le chapitre en question).
- on va utiliser ce paramètre : StockLimited dans la directive.

/enums/shop-params.ts

```
1. export enum ShopParams {                          // énumération
2.   StockLimited = 10,                              // des variables qui ne sont censées jamais
 changer ou très rarement
3.                                                   // comme des paramètres de configuration
4. }
```

XXI-C-2-a-iii - Le service

- histoire de bien faire les choses, on crée un service : user.service.ts qui va contenir le code métier correspondant à l'utilisateur et à sa connexion.
- la directive va utiliser ce service.

/services/user.service.ts

```
1. import { Injectable } from '@angular/core';
2.
3. @Injectable({
4.   providedIn: 'root'
5. })
6. export class UserService {
7.
8.   constructor() { }
9.
10.   isAuthorized() {
11.     //
12.     //  le code pour déterminer si l'utilisateur est authentifié ou pas
13.     //
14.     return true;    // ou false
15.   }
16. }
```

XXI-C-2-a-iv - La directive

/directives/isAuthenticated.directive.ts

```
1. import { Directive, TemplateRef, ViewContainerRef, OnInit } from '@angular/core';

2. import { UserService } from '../services/user.service';          // on a besoin d'accéder à ce
 service
3.
4. @Directive({
5.   selector: '[isAuthenticated]'
6. })
7. export class IsAuthenticatedDirective implements OnInit {
8.
9.   private hasView = false;             // un flag pour connaitre l'état, s'il est affiché ou
 pas actuellement
10.
11.   constructor(private templateRef: TemplateRef<any>,            // contient le DOM de la
 balise où est écrite la directive
12.               private viewContainer: ViewContainerRef,          // l'emplacement (où est
 écrite la directive) où doit être inséré le contenu (TemplateRef)
13.               private userService: UserService)                 // toujours ->  le code
 métier dans les services
14.   { }
15.
16.   ngOnInit(): void {                                            // quand il n'y a pas de
 paramètres, on utilise ngOnInit()
17.                                                                 // utilisation dans la vue :
 <div *isAuthenticated>
18.                                                                 // il n'y a pas de paramètre,
 car on n'a pas mis par exemple : <div *isAuthenticated="value">
19.
```

```
20.     const isAuthorized = this.userService.isAuthorized();        // on récupère l'information
 du service : true ou false
21.
22.     if (isAuthorized && !this.hasView) {                          // si autorisé et pas déjà
 affiché
23.       this.viewContainer.createEmbeddedView(this.templateRef);   // on met dans le container
 le contenu de la balise (du template)
24.       this.hasView = true;                                        // est affiché
25.
26.     } else if (!isAuthorized && this.hasView) {                   // si pas autorisé et qu'il
 est affiché actuellement
27.       this.viewContainer.clear();                                 // on efface l'emplacement
28.       this.hasView = false;                                       // n'est pas affiché
29.     }
30.   }
31. }
```

/directives/isStockLimited.directive.ts

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

2. import { ShopParams } from '../enums/shop-params.enum';
3.
4. @Directive({
5.   selector: '[isStockLimited]'
6. })
7. export class IsStockLimitedDirective {
8.
9.   private hasView = false;
10.
11.   constructor(private templateRef: TemplateRef<any>, private viewContainer: ViewContainerRef,
 ) { }
12.
13.   @Input() set isStockLimited(nb: number) {                       // quand il y a passage de
 paramètres, du binding, on utilise : @Input()
14.                                                                   // @Input()    -->  la
 réception du bind
15.                                                                   // de plus, on sait que nb
 est du type : number alors on le précise (le typage de typeScript)
16.                                                                   // nb = value = 4      ->
 réception de la valeur transmise à la balise : *isStockLimit = "4"
17.
18.     if (nb < ShopParams.StockLimited && !this.hasView) {          // 4 < 10 et pas affiché
19.       this.viewContainer.createEmbeddedView(this.templateRef);   // c'est OK !  le contenu de
 templateRef : "stock limité 1!" est envoyé à l'emplacement (container)
20.       this.hasView = true;
21.     } else if (!nb && this.hasView) {                             // sinon
22.       this.viewContainer.clear();
23.       this.hasView = false;
24.     }
25.
26.     console.log(this.templateRef);          // aller voir le contenu de templateRef dans la
 console et voyez ce que l'on peut modifier
27.   }
28. }
```

XXI-C-2-a-v - À savoir

- constructor(private templateRef: TemplateRef<any>, private viewContainer: ViewContainerRef,)
- templateRef c'est le contenu. Contient le DOM de l'élément (HTML ou composant) sur lequel est appliquée la directive.
- viewContainer c'est le contenant de l'élément (HTML ou composant) où on insère le contenu (templateRef)
- this.viewContainer.createEmbeddedView(this.templateRef);
- le principe d'une directive structurelle est donc de modifier le contenu (templateRef) et de l'insérer dans le contenant (viewContainer).

XXI-C-2-a-vi - Conclusion

- les directives structurelles sont un moyen puissant pour modifier la structure du DOM d'un élément (HTML ou composant).
- en mettant une directive *isAuthenticated sur un élément span, on peut faire beaucoup de choses sans surcharger la vue <span *isAuthenticated>vous êtes authentifié !.
- il faut toujours privilégier l'utilisation des directives.

XXII - en cours...

XXIII - Cycle de vie Angular

- Angular exécute une série d'interventions diverses sur un composant web afin de l'initialiser, l'instancier, l'exécuter…
- Angular permet si on le souhaite d'intervenir au moment de chaque phase.
- La phase la plus utilisée est : ngOnInit() { …code personnalisé… }

XXIII-A - Pratique

Nous allons tester toutes les phases et l'ordre auquel elles se lancent

```
ng new angular-cycle1
strict ? NO
routing ? NO
SCSS
```

```
ng g c comp1
```

app.component.html

```
<p>app works!</p>

<button (click)="messageParent = 'un message provenant du parent'">(1) cliquez !
 messageParent = 'un message provenant du parent'</button>
<button (click)="messageParent = ''">(1) reset</button>
<p>(1) messageParent = {{messageParent}}</p>

<hr>
<app-comp1 [inputMessage]="messageParent"></app-comp1>
```

app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  messageParent = '';
}
```

comp1.component.html

```
<div style="padding: 12px; margin-left: 24px; background: lavender;">
  <p>comp1 works!</p>

```

```
4.    <p>(1) @Input() inputMessage = {{inputMessage}}</p>
5.    <br><br>
6.
7.    <button (click)="messageComp1 = 'un message de comp1'">(2) cliquez !  messageComp1 = 'un
 message de comp1'</button>
8.    <button (click)="messageComp1 = ''">(2) reset</button>
9.    <p>(2) variable de comp1: messageComp1 = {{messageComp1}}</p>
10. </div>
```

comp1.component.ts

```
1. import { Component, AfterViewInit, AfterContentInit, OnInit, SimpleChanges, DoCheck, Input,
 AfterContentChecked, AfterViewChecked, OnChanges, OnDestroy } from '@angular/core';

2.                                                          // ici, on récupère les
 implémentations de : ngOnInit et des autres fonctions de cycle
3.                                                        // '@angular/core'    le '@' indique
 que c'est un package Angular
4.                                                        // sur les packages externes (non
 Angular), on ne met pas: '@'
5. @Component({
6.    selector: 'app-comp1',
7.    templateUrl: './comp1.component.html',
8.    styleUrls: ['./comp1.component.css']
9. })
10. export class Comp1Component implements AfterViewInit, AfterContentInit, OnInit, DoCheck,
 AfterContentChecked, AfterViewChecked, OnChanges, OnDestroy {
11.    @Input() inputMessage: string;     // (1)
12.    messageComp1 = '';                 // (2)
13.
14.    constructor() {                                  // demander l'injection d'objets que l'on a
 besoin d'utiliser dans le composant
15.      console.log("cycle n°1 - constructor");
16.      //
17.      //  on ne doit rien faire ici !
18.      //
19.    }
20.
21.    ngOnInit(): void {                               //  appelé après le constructeur,
 initialise les propriétés d'entrée et le premier appel à ngOnChanges
22.      console.log("cycle n°2 - ngOnInit");           //  attention :  cette fonction n'est
 appelée qu'une seule fois
23.      // initialiser des valeurs (qui ne doit être fait qu'une seule fois)
24.      // par exemple:
25.      //      - récupérer des données d'une API
26.      //      - souscrire à des observables
27.      //    ...
28.    }
29.
30.    ngDoCheck(): void {                              // appelé chaque fois qu'une propriété
 change d'un @Input ou d'une variable
31.                                                     // ou lorsqu'une directive est vérifiée.
32.                                                     // utilisez-le pour étendre la détection
 des modifications en effectuant une vérification personnalisée
33.      console.log('------');
34.      console.log("cycle n°3 - ngDoCheck - (un changement de valeur a eu lieu)");
35.    }
36.
37.    ngAfterContentInit(): void {                     // appelé après ngOnInit lorsque le contenu
 du composant ou de la directive a été initialisé
38.      console.log("cycle n°4 - ngAfterContentInit");
39.    }
40.
41.    ngAfterContentChecked(): void {                  // appelé après chaque vérification du
 contenu du composant ou de la directive
42.                                                     // appelé après chaque fois qu'une
 vérification du contenu externe (transclusion) est faite
43.      console.log("cycle n°5 - ngAfterContentChecked");
44.    }
45.
```

```
46.   ngAfterViewInit(): void {     // appelé après ngAfterContentInit lorsque la vue du
 composant a été initialisée. S'applique uniquement aux composants.
47.                                 // il est appelé dès lors que la vue du composant ainsi que
 celles de ses enfants sont initialisées
48.     console.log("cycle n°6 - ngAfterViewInit");
49.   }
50.
51.   ngAfterViewChecked(): void {                  // appelé après chaque vérification de la
 vue du composant.
52.                                                 // s'applique uniquement aux composants.
53.     console.log("cycle n°7 - ngAfterViewChecked");
54.   }
55.
56.   ngOnDestroy(): void {                         // appelé quand le composant est détruit
 par Angular
57.                                                 // lors du routing, quand on change de page
 cette fonction est appelée
58.     console.log('cycle n°x - ngOnDestroy');
59.     // si on a des observables qui tournent dans le composant
60.     // il faut absolument se désabonner des observables ici
61.     // obsxxxxxxx.unsubscribe();
62.   }
63.
64.   ngOnChanges(changes: SimpleChanges): void {     // détection de changement de valeur
 uniquement sur les : @Input() ......
65.     console.log('-------------------------');     // donc concerne le binding du composant
 parent
66.     console.log('cycle n°x - ngOnChanges');      // cette fonction est appelée quand une
 valeur change sur n'importe quelle entrée (@Input)
67.     console.log(changes);
68.     console.log('Le @input qui a changé est:' + JSON.stringify(changes));
69.   }
70. }
```

XXIII-B - Résultat

- Cliquez alternativement sur : '(1) cliquez …' et '(1) reset' et voyez le résultat dans la console.
- Cliquez alternativement sur: '(2) cliquez …' et '(2) reset' et voyez le résultat dans la console.
- il y a une différence entre les deux ? oui -> 'ngOnChanges' en + pour (1).

XXIII-C - Conclusion

En général, les cycles les plus utilisés :

- constructor() pour demander les instances de services que l'on a besoin (DI : injection de dépendances) ;
- ngOnInit() pour initialiser des données ;
- ngOnDestroy() pour se désinscrire à des observables ;
- ngOnChanges() pour pouvoir agir lors d'une détection de changement de valeur des données en entrée (@Input).

En rapport avec l'affichage :

- avant l'affichage : constructor(), ngOnInit(), ngDoCheck() et ngOnChanges() ;
- après l'affichage : ngAfterViewChecked() et ngAfterViewInit() ;

XXIV - Architecture avancée : modules, composants web…

Voyons comment organiser un projet complexe en modules.

Mais avant d'aller plus loin, sachez que cette partie est d'un niveau avancé, revenez plus tard sur cette partie si vraiment vous débutez.

En effet, si vous débutez, vous pouvez déclarer et importer tout ce que vous voulez dans le module : app.module.ts

Voici la liste des éléments que l'on peut trouver dans un projet :

- des modules (que l'on importe dans un module) ;
- des composants (que l'on déclare et que l'on exporte dans un module) ;
- des services ;
- du routing (que l'on rattache à un module) ;
- des modèles de données.

XXIV-A - Qu'est-ce que sont les services ?

Bonnes pratiques :

- pour alléger le code et pour le rendre réutilisable, on met le code métier dans un service ;
- ainsi, si besoin, plusieurs composants peuvent faire appel à une même fonction d'un service ;
- nous verrons plus loin les services.

XXIV-B - Qu'est-ce que le routing ?

Bonnes pratiques :

- le routing permet d'associer un chemin : /mon-url à un composant web (une page) ;
- rappel : un module comprend un ou plusieurs composants web (ou page) ;
- ainsi tel chemin correspond à tel composant web du module ;
- d'un point de vue du nommage, un composant web qui sert pour le routing on le nomme simplement un « composant page » ou « page », mais techniquement rien ne change, ça reste un composant web ;
- généralement, on crée un fichier de routing à part que l'on importe dans un module.

XXIV-C - Exemple avec un site e-commerce

Voici les grandes fonctionnalités d'un site d'e-commerce : l'affichage des produits, le panier, la gestion des commandes…

Bonnes pratiques :

- une fonctionnalité = un module.

Rappel : un module est composé d'un ou plusieurs composants.

Remarque : histoire d'être complet, on a ajouté des services (providers) et du routing

Rappel sur le rôle d'un module :

- si nécessaire, rendre réutilisable le module (et donc ses composants) ;
- déclarer les composants web pour qu'ils puissent être utilisés ;
- importer des modules internes au projet ou des modules externes (dossier /node_modules) ;
- exporter les composants web afin qu'ils soient disponibles dans un autre module ;
- indiquer les services qui doivent être injectés dans les composants pour son utilisation (DI - Injection de dépendances) ;
- routing : associer un chemin URL à un composant web.

XXIV-C-1 - Organisation d'un projet

- Il n'y a pas de meilleure façon d'organiser un projet.
- Chaque type de projet peut avoir son type d'organisation.
- L'organisation que je montre ici est subjective.

XXIV-C-1-a - Principes

```
1. - un dossier /core          -->  on regroupe tout ce qui est en commun avec tout le projet en
 lui-même (les services, des modèles de données, des composants un peu particuliers...)

2. - un dossier /features      -->  disons que ce sont des bouts, des parties qui vont servir
 pour construire les pages
3.                             -->  (on utilise: /core si besoin)
4. - un dossier /pages         -->  les pages de l'application (on les construit à partir des
 composants de: /features)
5.                             -->  chaque page doit avoir son module (si on veut utiliser le
 lazy loading)
6. - un dossier /shared        -->  on regroupe tout ce qui est en commun avec plusieurs projets
7.                                 en effet, le composant: loading.component peut servir pour
 d'autres projets
```

XXIV-C-1-b - Schéma A

```
22.                 _____________________________________________________________
          |
23.
 ____________________________________________________________________________________
```

- (1) et (2) utilisent les composants de product.module qui utilisent les composants de core.module et shared.module.
- (3) utilise les composants de cart.module qui utilisent les composants de core.module et shared.module.
- (4) utilisent les composants de core.module et shared.module.
- (5) utilise les composants de partials.module (4) qui utilisent les composants de core.module et shared.module.
- (6) pages.module regroupe les modules des pages : (1), (2) et (3) pour le routing.

XXIV-C-1-c - Remarques

- Nous avons créé un module par page pour prévoir la mise en place du lazy-loading (chargement des pages « ou modules » différé).

XXIV-C-1-d - Schéma B

```
1. app.module.ts
2.   ** imports: [BrowserModule, AppRoutingModule, PartialsModule,],
3. app-routing.module.ts                    // URL : .../product  .../products  .../cart
4. /core                                    // tout ce qui est en commun entre /components et /
pages
5.     core.module.ts
6.       ** declarations: [UserProfilComponent],
7.       ** imports: [CommonModule,],
8.       ** exports: [UserProfilComponent,],
9.     /services                            // les services de portée racine qui seront
 accessibles de partout (pages, composants, autres services...)
10.         product-store.service.ts
11.         user.service.ts                  // service qui a pour code métier
 l'authentification
12.         ...
13.     /models                              // des modèles de données globales au projet
14.         product.ts                       // on aura besoin du modèle de données : product
 dans de nombreux composants et pages différentes
15.         ...                              // donc on le met ici
16.         ...
17.     /enums
18.         ...                              // des énumérations globales au projet
19.         ...
20.     /components
21.         /user-profil                     // le profil peut être utilisé par : /components et
 par /pages
22.             user-profil.component. (html, ts, css)
23.         ...
24. /features                                                    // les principales
 fonctionnalités du projet
25.     /product
26.         product.module.ts
27.           ** declarations: [ProductLineComponent, ProductAddRemoveComponent],
28.           ** imports: [CommonModule, CoreModule,],
29.           ** exports: [ProductLineComponent, ProductAddRemoveComponent],

30.         /services
31.             product.service.ts                                  // code métier uniquement
 pour : /features  (requêtes HTTP , communication avec cart...)
32.                                                                 // par exemple, dans ce
 service on utilisera le ou les services du dossier : /core/services/****
33.         /models
34.           ...                                                   // modèle de données qui vont
 servir uniquement aux composants : /components (et pas ailleurs)
35.         /components
36.             /product-line
```

```
                product-line.component. (html, ts, css)
            /product-add-remove
                product-add-remove.component. (html, ts, css)   // communique avec "cart" pour ajouter ou supprimer un produit du panier
                                                                // en utilisant le service : product.service.ts
    /cart
        cart.module.ts
          ** declarations: [CartLineComponent, CartTotalComponent, CartIconComponent,],
          ** imports: [CommonModule, CoreModule,],
          ** exports: [CartLineComponent, CartTotalComponent, CartIconComponent,],

        /services
            cart.service.ts                                     // code métier pour cart : en plus, contient la liste du panier et le total des montants
        /models
            cart.ts
        /components
            /cart-line
                cart-line.component. (html, ts, css)
            /cart-total
                cart-total.component. (html, ts, css)
            /cart-icon
                cart-icon.component. (html, ts, css)
/pages
    pages.module.ts
      ** declarations: [ ],
      ** imports: [PageProductModule, PageProductsModule, PageCartModule, ]
    /partials
        partials.module.ts
          ** declarations: [HeaderComponent, FooterComponent, ],
          ** imports: [CommonModule, CoreModule, CartModule, ],
          ** exports: [HeaderComponent, FooterComponent, ],
        /header
            header.component. (html, ts, css)
        /footer
            footer.component. (html, ts, css)
    /page-product
        page-product.module.ts
          ** declarations: [PageProductComponent, ],
          ** imports: [CommonModule, ProductModule, ],
        page-product.component. (html, ts, css)
    /page-products
        page-products.module.ts
          ** declarations: [PageProductsComponent, ],
          ** imports: [CommonModule, ProductModule, ],
        page-products.component. (html, ts, css)
        /components
            ...                                 // si besoin, on peut mettre ici des composants qui vont servir uniquement pour : page-products.component
            ...
    /page-cart
        page-cart.module.ts
          ** declarations: [PageCartComponent, ],
          ** imports: [CommonModule, CartModule, ],
        page-cart.component. (html, ts, css)    // pour construire la page, on utilise les composants qui se trouvent dans : /features/cart/components/*
/shared                                                         // on importe ici des composants "génériques" que l'on utilise pour plusieurs projets
                                                                // on importe les modules que l'on souhaite : loading.module.ts, http.module.ts, bootstrap.module.ts
                                                                // ces modules sont donc disponibles là ou on les importe
    loading.module.ts
        loading.component.ts
    http.module.ts
        http.service.ts
    bootstrap.module.ts
        intégration package externe bootstrap
```

XXIV-D - Pratique

On va utiliser Angular-cli pour faire le travail rapidement (vite fait, bien fait)

```
1. ng new angular-shop-skeleton1
2. strict ? NO
3. routing ? YES
4. SCSS
```

// en répondant : routing ? YES :
- cela va créer un fichier de configuration de routes : /app/app-routing.module.ts ;
- ce nouveau fichier sera importé dans 'imports' du module racine : app.module.ts.

```
1. ng g m core
2. ng g s /core/services/product-store
3. ng g s /core/services/user
4. ng g i /core/models/product
5. ng g c /core/components/user-profil --module=core
6.
7. ng g m /features/product
8. ng g s /features/product/services/product
9. ng g c /features/product/components/product-line --module=product
10. ng g c /features/product/components/product-add-remove --module=product
11.
12. ng g m /features/cart
13. ng g s /features/cart/services/cart
14. ng g i /features/cart/models/cart
15. ng g c /features/cart/components/cart-line --module=cart
16. ng g c /features/cart/components/cart-total --module=cart
17. ng g c /features/cart/components/cart-icon --module=cart
18.
19. ng g m /pages --module=app
20. ng g m /pages/page-product --module=pages
21. ng g c /pages/page-product --module=page-product
22. ng g m /pages/page-products --module=pages
23. ng g c /pages/page-products --module=page-products
24.
25. ng g m /pages/page-cart --module=pages
26. ng g c /pages/page-cart --module=page-cart
27.
28. ng g m /pages/partials --module=app
29. ng g c /pages/partials/header --module=partials
30. ng g c /pages/partials/footer --module=partials
```

app.module.ts

```
1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { AppRoutingModule } from './app-routing.module';
4. import { AppComponent } from './app.component';
5. import { PartialsModule } from './pages/partials/partials.module';
6. //
7.
8. import { PagesModule } from './pages/pages.module';
9.
10. @NgModule({
11.   declarations: [AppComponent],
12.   imports: [BrowserModule, AppRoutingModule, PartialsModule, ],
13.   providers: [],
14.   bootstrap: [AppComponent],
15.   schemas: [  ],
16. })
17. export class AppModule { }
```

app.component.html

```
<app-header></app-header>

<div style="background: lightyellow; padding: 128px 12px;">
  <router-outlet></router-outlet>                              <!-- ici que sont projetées les
 pages du routing -->
</div>

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

app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { PageProductComponent } from './pages/page-product/page-product.component';
import { PageProductsComponent } from './pages/page-products/page-products.component';
import { PageCartComponent } from './pages/page-cart/page-cart.component';

const routes: Routes = [
  { path: 'product', component: PageProductComponent },
  { path: 'products', component: PageProductsComponent },
   { path: 'cart', component: PageCartComponent },
];

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

dans le header on va mettre des liens, le panier et le profil.

/pages/partials/header.component.html

```
<div style="background: lightsteelblue; padding: 24px;">
  <p>header works!</p>

  <p><button (click)="navigateTo('/product')">product</button></p>
  <p><button (click)="navigateTo('/products')">products</button></p>
  <p><button (click)="navigateTo('/cart')">cart</button></p>
  <app-cart-icon></app-cart-icon>
  <app-user-profil></app-user-profil>
</div>
```

/pages/partials/header.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router'; // CLI imports router

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

   constructor(private router: Router) { }

   ngOnInit(): void {
   }

   navigateTo(value: string): void {
     this.router.navigate([value]);
   }
}
```

/pages/partials/footer.component.html

```
<div style="background: lightsteelblue; padding: 48px;">
  <p>footer works!</p>
</div>
```

/core/models/product.ts

```
export interface Product {
  id: number;
  name: string;
  description?: string;
}
```

/core/services/product-store.service.ts

```
import { Injectable } from '@angular/core';
import { Product } from '../models/product';

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

  private productSelected: Product = { id: 1, name: 'XBOX'};

  constructor() { }

  getProductSelected(): Product {
    return this.productSelected;
  }

  setProductSelected(product: Product): void {
    this.productSelected = product;
  }
}
```

/pages/page-product/page-product.component.html

```
<p>page-product works!</p>
ProductSelected={{ProductSelected|json}}

<app-product-line></app-product-line>                <!-- on utilise des composants de : /
components pour construire la page -->
<app-product-add-remove></app-product-add-remove>
```

/pages/page-product/page-product.component.ts

```
import { Component, OnInit } from '@angular/core';
import { ProductStoreService } from '../../core/services/product-store.service';
import { Product } from '../../core/models/product';

@Component({
  selector: 'app-page-product',
  templateUrl: './page-product.component.html',
  styleUrls: ['./page-product.component.scss']
})
export class PageProductComponent implements OnInit {
  ProductSelected: Product;

  constructor(private productStoreService: ProductStoreService) { }

  ngOnInit(): void {
    this.ProductSelected = this.productStoreService.getProductSelected();       // un exemple
 pour montrer qu'on utilise : /core
  }
}
```

/pages/page-products/page-products.component.html

```
<p>page-products works!</p>

<app-product-line></app-product-line>     <!-- on utilise des composants de:  /components pour construire la page -->
<app-product-line></app-product-line>
<app-product-line></app-product-line>
<app-product-line></app-product-line>
```

/pages/page-cart/page-cart.component.html

```
<p>page-cart works!</p>

<app-cart-line></app-cart-line>           <!-- on utilise des composants de : /components pour construire la page -->
<app-cart-line></app-cart-line>
<app-cart-total></app-cart-total>
```

- en vous basant sur le schéma B, complétez le code des différents modules suivants :
    - /core/core.module.ts ;
    - /features/product/product.module.ts ;
    - /features/cart/cart.module.ts ;
    - /pages/pages.module.ts ;
    - /pages/partials/partials.module.ts.

```
ng serve
```

XXIV-D-1 - Résultat

- Cliquez sur les boutons pour changer de page, constatez que tout fonctionne normalement.
- Vérifiez qu'il n'y a pas d'erreurs dans la console.

XXIV-E - Bonus

- Nous souhaitons afficher un pdf dans la page du produit.
- Pour cela on va utiliser un package externe: **https://www.npmjs.com/package/ng2-pdf-viewer**.
- Ce package sera inclus uniquement au niveau du produit, car les autres modules n'en ont pas besoin.

XXIV-E-1 - Pratique

- (A1) importer le package en ligne de commande.
- (A2) créer un composant qui a pour but d'afficher un pdf.
- (A3) importer dans le fichier module du produit le nouveau package.
- (A4) ajouter ce composant dans la page produit.

XXIV-E-1-a - (A1) Importer le package en ligne de commande

```
npm install ng2-pdf-viewer --save
```

XXIV-E-1-b - (A2) Créer un composant qui a pour but d'afficher un pdf

On met ce composant dans /features/product, car cela ne concerne que les produits

```
ng g c /features/product/product-pdf-viewer --module=product
```

/features/product/product-pdf-viewer.component.html

```
<p>product-pdf-viewer works!</p>

<pdf-viewer
  [src]="'https://vadimdez.github.io/ng2-pdf-viewer/assets/pdf-test.pdf'"
  [render-text]="true"
  style="display: block; height: 40vh;"
></pdf-viewer>
```

XXIV-E-1-c - (A3) Importer dans le fichier module du produit le nouveau package

/features/product/product.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { ProductLineComponent } from './components/product-line/product-line.component';
import { ProductAddRemoveComponent } from './components/product-add-remove/product-add-remove.component';
import { CoreModule } from '../../core/core.module';

import { ProductPdfViewerComponent } from './product-pdf-viewer/product-pdf-viewer.component';
import { PdfViewerModule } from 'ng2-pdf-viewer';                    // ici, on importe le package: PdfViewerModule

@NgModule({
  declarations: [ProductLineComponent, ProductAddRemoveComponent, ProductPdfViewerComponent],
  // le nouveau composant: ProductPdfViewerComponent
  imports: [ CommonModule, CoreModule, PdfViewerModule, ],
  // ici, on importe le package: PdfViewerModule
  exports: [ProductLineComponent, ProductAddRemoveComponent, ProductPdfViewerComponent],
  // on exporte le nouveau composant: ProductPdfViewerComponent
})
export class ProductModule { }
```

XXIV-E-1-d - (A4) Ajouter ce composant dans la page : produit

/pages/page-product/page-product.component.html

```
<p>page-product works!</p>
ProductSelected={{ProductSelected|json}}

<app-product-line></app-product-line>                <!-- on utilise des composants de : / components pour construire la page -->
<app-product-add-remove></app-product-add-remove>

<app-product-pdf-viewer></app-product-pdf-viewer>    <!-- ici,  le nouveau composant à inclure -->
```

quand vous touchez aux modules, toujours relancer avec ng serve (ne pas faire confiance au live reload) :

```
ng serve
```

XXIV-E-1-e - Conclusion

- Nous avons importé un package externe dans le module : /features/product/product.module.ts, ce package ne sera donc accessible que par les composants de ce module (pas besoin qu'il soit disponible ailleurs).

XXIV-E-1-f - À savoir

- Nous aurions même pu être plus précis en créant un module directement dans le composant comme ici : / features/product/components/product-pdf-viewer/product-pdf-viewer.module.ts et importer le package externe PdfViewerModule dans ce module (au lieu, comme actuellement dans le module : product.module.ts).
- Ensuite pour que ça fonctionne, importer product-pdf-viewer.module.ts dans le module /features/product/ product.module.ts.

XXV - Les composants web réutilisables

Nous avons vu que réutiliser des composants web était une bonne pratique, car cela améliore la maintenabilité et augmente la productivité :

- nous allons donc dans ce chapitre écrire un exemple de composant web réutilisable ;
- ce composant affichera une liste d'éléments et renverra le choix que l'utilisateur a sélectionné, un peu comme un groupe de radio boutons.

XXV-A - À savoir

Pour une capacité de réutilisation optimale, le composant doit se contenter de ne faire que sa propre fonctionnalité, celle d'afficher une liste et de retourner le choix de l'utilisateur, en somme, ce composant :

- ne devra pas contenir la liste des éléments à afficher ;
- ne devra pas aller chercher la liste des éléments à afficher (mais par contre, il la recevra)

XXV-B - Description

- le composant réutilisable sera contenu dans le dossier : /shared
- pour le design, le composant utilisera un composant UI d'Angular Material ;
- Voici le détail du composant web réutilisable :
    - réceptionne la liste des éléments (d'un certain type) à afficher ;
    - réceptionne l'élément par défaut qui doit être sélectionné (il est possible qu'il n'y ait aucun élément) ;
    - réceptionne le nom du groupe auquel appartient la liste ;
    - sélectionne l'item par défaut ou celui enregistré dans un service (lors d'une précédente sélection utilisateur) ;
    - renvoie le choix de l'utilisateur au composant qui y a fait appel ;
    - contient le type de donnée que le composant manipule. Ainsi de l'extérieur, on sait de quel type doivent être les données que l'on doit fournir au composant ;
    - enregistre le choix de l'utilisateur dans un service ;
    - le design du composant sera géré par Angular Material.

XXV-B-1 - Remarques

- pour le design, on va utiliser le toggle button d'Angular material ;
- **https://material.angular.io/components/button-toggle/overview**
- lors du routing à chaque accès à une page : page1 ou page2, les composants pages s'initialisent et donc ne conservent pas l'état de la sélection. C'est pour cela que l'on utilise un service pour stocker le choix de l'utilisateur et ainsi l'employer pour initialiser le composant avec la bonne sélection.

XXV-B-2 - Inventaires

```
- /shared/button-toggle-mat
   - /components/button-toggle-mat.component.ts              // le composant
   - /services/stored.service.ts                             // pour enregistrer le choix de
 l'utilisateur
   - /models/i-item-btm.ts                                   // le type de donnée que le
 composant manipule
   - button-toggle-mat.module.ts                             // déclare le composant et le
 service. De plus, exporte le composant.

- /pages
   - /page1                                        // composant avec une liste d'items
 'quelconque' ayant comme nom de groupe : 'choice'
   - /page2                                        // composant avec une liste d'items 'couleur'
 ayant comme nom de groupe : 'color'
                                                    // composant avec une liste d'items
 'quelconques' ayant comme nom de groupe : 'choice'
```

XXV-C - Pratique

Créer un nouveau projet : angular-re-use1

```
ng new angular-re-use1
strict ? NO
routing ? YES
SCSS

ng add @angular/material

ng g m pages --module=app
ng g m pages/page1 --module=pages
ng g m pages/page2 --module=pages
ng g c pages/page1 --module=page1
ng g c pages/page2 --module=page2

ng g m shared/button-toggle-mat --module=app
ng g c shared/button-toggle-mat/components/button-toggle-mat --module=button-toggle-mat
ng g i shared/button-toggle-mat/models/i-item-btm
ng g s shared/button-toggle-mat/services/stored
```

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { NoopAnimationsModule } from '@angular/platform-browser/animations';
//
import { PagesModule } from './pages/pages.module';

@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    NoopAnimationsModule,
    //
    PagesModule,                // pour le routing
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

app-routing.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. //
4. import { Page1Component } from 'src/app/pages/page1/page1.component';
5. import { Page2Component } from 'src/app/pages/page2/page2.component';
6.
7. const routes: Routes = [
8.   { path: 'page1', component: Page1Component },
9.   { path: 'page2', component: Page2Component },
10. ];
11.
12. @NgModule({
13.   imports: [RouterModule.forRoot(routes)],
14.   exports: [RouterModule]
15. })
16. export class AppRoutingModule { }
```

app.component.html

```
1. <ul>
2.   <li><a [routerLink]="['/page1']">aller à page1</a></li>
3.   <li><a [routerLink]="['/page2']">aller à page2</a></li>
4. </ul>
5.
6. <router-outlet></router-outlet>
```

\data\param.ts

```
1. export const ItemsChoice = [
2.   {key: 'K0', value: 'Aucun'},
3.   {key: 'K1', value: 'Choix 1'},
4.   {key: 'K2', value: 'Choix 2'},
5.   {key: 'K3', value: 'Choix 3'},
6. ];
7.
8. export const ItemsColor = [
9.   {key: 'NONE', value: 'Aucune'},
10.   {key: 'V', value: 'Vert'},
11.   {key: 'R', value: 'Rouge'},
12. ];
```

\pages\page1\page1.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3. //
4. import { ButtonToggleMatModule } from '../../shared/button-toggle-mat/button-toggle-
mat.module';
5. import { Page1Component } from './page1.component';
6.
7.
8. @NgModule({
9.   declarations: [Page1Component],
10.   imports: [
11.     CommonModule,
12.     //
13.     ButtonToggleMatModule,            // utilise le composant réutilisable du module

14.   ]
15. })
16. export class Page1Module { }
```

\pages\page2\page2.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
//
import { ButtonToggleMatModule } from '../../shared/button-toggle-mat/button-toggle-
mat.module';
import { Page2Component } from './page2.component';


@NgModule({
  declarations: [Page2Component],
   imports: [
     CommonModule,
     //
     ButtonToggleMatModule,            // utilise le composant réutilisable du module

   ]
})
export class Page2Module { }
```

\pages\pages.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
//
import { Page1Module } from './page1/page1.module';
import { Page2Module } from './page2/page2.module';

@NgModule({
  declarations: [],
  imports: [
     CommonModule,
     //
     Page1Module,
     Page2Module,

   ]
})
export class PagesModule { }
```

\pages\page1\page1.component.html

```
<p>page1 works!</p>

<h3>quelconque</h3>

<app-button-toggle-mat
  [items]="choices"
  [selectedItem]="selectedChoice"
  [group]="'choice'"
  (selectedItemEvent)="onSelectedChoice($event)"
></app-button-toggle-mat>

(1) choice={{selectedChoice|json}}
```

\pages\page1\page1.component.ts

```
import { Component, OnInit } from '@angular/core';
import { IItemBtm } from '../../shared/button-toggle-mat/models/i-item-btm';
import { ItemsChoice } from '../../data/param';

@Component({
  selector: 'app-page1',
  templateUrl: './page1.component.html',
  styleUrls: ['./page1.component.scss']
})
export class Page1Component implements OnInit {
   choices: IItemBtm[];               // on exige que ces éléments du tableau doivent être du
 type : IItemBtm
```

```
12.                                     // ainsi, il ne peut y avoir d'erreur, car notre
 composant ne gère que ce type de donnée : IItemBtm
13.   selectedChoice: IItemBtm;          // on fourni l'élément par défaut au composant
 réutilisable (non obligatoire)  (2)
14.                                     // et en même temps va contenir le choix de
 l'utilisateur
15.
16.   constructor() {
17.   }
18.
19.   ngOnInit(): void {
20.     this.choices = ItemsChoice;               // on fournit une liste d'éléments au
 composant réutilisable
21.     this.selectedChoice = this.choices[0];    // (2) par défaut, l'élément qui sera
 sélectionné sera le premier élément (non obligatoire)
22.   }
23.
24.   onSelectedChoice(item: IItemBtm) {          // lien avec le composant réutilisable enfant
25.     this.selectedChoice = item;               // (1) on réceptionne le choix de
 l'utilisateur pour l'afficher dans la vue
26.     //
27.     //  traitement
28.     //
29.   }
30. }
```

\pages\page2\page2.component.html

```
1. <p>page2 works!</p>
2.
3. <h3>couleur</h3>
4.
5. <app-button-toggle-mat
6.   [items]="colors"
7.   [group]="'color'"
8.   (selectedItemEvent)="onSelectedColor($event)"
9. ></app-button-toggle-mat>
10.
11. (1) Couleur={{selectedColor|json}}
12.
13. <hr>
14.
15. <h3>quelconque</h3>
16.
17. <app-button-toggle-mat
18.   [items]="choices"
19.   [selectedItem]="selectedChoice"
20.   [group]="'choice'"
21.   (selectedItemEvent)="onSelectedChoice($event)"
22. ></app-button-toggle-mat>
23.
24. (1) choice={{selectedChoice|json}}
```

\pages\page2\page2.component.ts

```
1. import { Component, OnInit } from '@angular/core';
2. import { IItemBtm } from '../../shared/button-toggle-mat/models/i-item-btm';
3. import { ItemsColor } from '../../data/param';
4. import { ItemsChoice } from '../../data/param';
5.
6. @Component({
7.   selector: 'app-page2',
8.   templateUrl: './page2.component.html',
9.   styleUrls: ['./page2.component.scss']
10. })
11. export class Page2Component implements OnInit {
12.
13.   //  Choix d'une couleur
14.   colors: IItemBtm[];                       // on exige que les éléments du tableau doivent
 être du type : IItemBtm
```

```
15.   selectedColor: IItemBtm;                    // pour contenir le choix de l'utilisateur
16.
17.   //  Choix d'un item quelconque
18.   choices: IItemBtm[];                        // on exige que les éléments du tableau doivent
 être du type : IItemBtm
19.   selectedChoice: IItemBtm;                   // pour contenir le choix de l'utilisateur
20.
21.   constructor() { }
22.
23.   ngOnInit(): void {                          // initialisation des données
24.     //  Choix d'une couleur
25.     this.colors = ItemsColor;                 // on fournit une liste d'éléments au composant
 réutilisable
26.                                               // sans valeur par défaut
27.     //  Choix d'un item quelconque
28.     this.choices = ItemsChoice;               // on fournit une liste d'éléments au composant
 réutilisable
29.     this.selectedChoice = this.choices[0];    // avec une valeur par défaut
30.   }
31.
32.   //  Choix d'une couleur
33.   onSelectedColor(item: IItemBtm) {           // lien avec le composant réutilisable enfant
34.     this.selectedColor = item;                // (1) on réceptionne le choix de l'utilisateur
 pour l'afficher dans la vue
35.     //
36.     //  traitement
37.     //
38.   }
39.
40.   //  Choix d'un item quelconque
41.   onSelectedChoice(item: IItemBtm) {           // lien avec le composant réutilisable enfant
42.     this.selectedChoice = item;               // (1) on réceptionne le choix de
 l'utilisateur pour l'afficher dans la vue
43.     //
44.     //  traitement
45.     //
46.   }
47. }
```

\shared\button-toggle-mat\button-toggle-mat.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3. //
4. import { ButtonToggleMatComponent } from './components/button-toggle-mat/button-toggle-
mat.component';
5. import { MatButtonToggleModule } from '@angular/material/button-toggle';
6. import { StoredService } from './services/stored.service';
7.
8.
9. @NgModule({
10.   declarations: [ButtonToggleMatComponent],
11.   imports: [
12.     CommonModule,
13.     MatButtonToggleModule,              // on importe uniquement le module : MatButton
 d'Angular Material pour pouvoir utiliser son composant
14.
15.   ],
16.   exports: [
17.     ButtonToggleMatComponent,           // on exporte le composant pour qu'il soit utilisable
 lors d'un import
18.   ],
19.   providers: [StoredService, ]          // les composants de ce module auront accès à cette
 instance du service
20. })                                      // donc : ButtonToggleMatComponent des pages : page1
 et page2 aura accès à cette instance
21. export class ButtonToggleMatModule { }
```

\shared\button-toggle-mat\models\i-item-btm.ts

```
export interface IItemBtm {
  key: string;
  value: string;
}
```

\shared\button-toggle-mat\services\stored.service.ts

```
import { IItemBtm } from '../models/i-item-btm';

export class StoredService {

  selectedItemGroup: IItemBtm[] = [];       //  un tableau contenant les couples :  'nom de groupe': sélection utilisateur

  constructor() { }

  getSelectedItem(group: string): IItemBtm {          // on récupère le choix utilisateur par son groupe
    return (undefined != this.selectedItemGroup[group]) ? this.selectedItemGroup[group] : undefined;
  }

  setSelectedItem(item: IItemBtm, group: string) {    // on enregistre le choix utilisateur par son groupe
    this.selectedItemGroup[group] = item;
  }

  getInitializedItem(defaultItem: IItemBtm, group: string): IItemBtm {      // on calcul en fonction de la valeur par défaut et de la valeur enregistrée du tableau
    if (defaultItem && this.getSelectedItem(group) === undefined) {         // s'il y a une valeur par défaut et pas de sélection utilisateur enregistrée alors...
      this.setSelectedItem(defaultItem, group);                             // c'est le choix par défaut qui est pris en compte
      return defaultItem;
    }

    return this.getSelectedItem(group);                                     // sinon c'est le choix enregistré dans le tableau qui est pris en compte
  }
}
```

\shared\button-toggle-mat\components\button-toggle-mat\button-toggle-mat.component.html

```
<p>
  Votre choix ?  
  <mat-button-toggle-group name="fontStyle" aria-label="Font Style" #group="matButtonToggleGroup" [value]="selectedItem?.key" (change)="onChange($event)">
    <ng-container *ngFor="let item of items">
      <mat-button-toggle value="{{item.key}}" >{{item.value}}</mat-button-toggle>
    </ng-container>
  </mat-button-toggle-group>
</p>
```

\shared\button-toggle-mat\components\button-toggle-mat\button-toggle-mat.component.ts

```
import { Component, Input, OnInit, Output, EventEmitter } from '@angular/core';
import { IItemBtm } from '../../models/i-item-btm';
import { StoredService } from '../../services/stored.service';

@Component({
  selector: 'app-button-toggle-mat',
  templateUrl: './button-toggle-mat.component.html',
  styleUrls: ['./button-toggle-mat.component.scss'],
})
export class ButtonToggleMatComponent implements OnInit {
  @Input() items: IItemBtm[];                                   // réceptionne la liste des éléments
```

```
12.   @Input() selectedItem: IItemBtm;                                    // l'élément qui doit être
 sélectionné par défaut
13.   @Input() group: string;                                             // le groupe auquel
 appartiennent les éléments
14.   @Output() selectedItemEvent = new EventEmitter<IItemBtm>();         // envoi le choix de
 l'utilisateur au parent : page1 ou page2
15.
16.   constructor(private storedService: StoredService) { }
17.
18.   ngOnInit(): void {                                                  // à l'initialisation du
 composant
19.     this.selectedItem = this.storedService.getInitializedItem(this.selectedItem, this.group);
   // on calcul quel item est sélectionné au 1er affichage
20.     this.selectedItemEvent.emit(this.selectedItem);
   // on retourne cet item au parent : page1 ou page2
21.   }
22.
23.   onChange(event: any) {                                              // quand un choix utilisateur
 est fait
24.     // event.value ne contient que : key
25.     // on veut pouvoir retourner l'objet entier (key + value)
26.     // donc on va le chercher dans la liste des items à partir de sa clé : key
27.     const item: IItemBtm = this.items.filter((item: IItemBtm) => item.key == event.value)[0];
   // on parcourt la liste des items et si on trouve la correspondance avec: key
28.
   // alors on retourne l'objet trouvé
29.     this.storedService.setSelectedItem(item, this.group);           // on enregistre le choix
30.     this.selectedItemEvent.emit(item);                              // on retourne le choix
 de l'utilisateur au parent : page1 ou page2
31.   }
32. }
```

XXV-D - Résultat

- Quand on sélectionne un choix celui-ci est envoyé à son parent : page1.component ou page2.component
- Remarquez que de page1 à page2 et vice versa, le choix « quelconque » garde la sélection que l'on a faite grâce au même nom de groupe.

XXV-E - Conclusion

- Le composant web peut être utilisé plusieurs fois, il suffit de copier-coller le dossier : /button-toggle-mat dans un autre projet et l'utiliser tel quel.
- Comme le composant ne dépend pas d'une liste définie, on peut lui transmettre n'importe quelle liste à condition qu'elle respecte le modèle.
- Il suffit de lui transmettre une liste à afficher, un nom de groupe et si besoin un élément par défaut.
- À savoir que lors du routing, l'accès à une page engendre l'initialisation de son composant page et de ses données.
- Pour pouvoir enregistrer des données afin de les récupérer lors de l'initialisation d'un composant page on se sert d'un service pour stocker les données (car son instance est un singleton).

XXVI - Mise en production : Firebase hosting

Utilisons le service Hosting de firebase pour mettre en production une application Angular.
Ce service est gratuit et limité, mais cela suffit largement pour tester.

(1)
On va utiliser le projet : `angular-re-use1` pour le mettre en production.

Copier / coller le projet : `angular-re-use1` et renommer le dossier en : `angular-hosting1`

Sur le nouveau dossier : angular-hosting1 faire une recherche globale et remplacer tous les mots : 'angular-re-use1' par 'angular-hosting1'
(sur visual studio code -> clic droit sur le dossier -> find in folder -> angular-re-use1 en : angular-hosting1)

(2) Ou utiliser n'importe quel projet qui tourne en local.

XXVI-A - Pratique

Créer un compte et se connecter à : **https://firebase.google.com/**

- Une fois connecté, il faut créer un projet firebase ;
- Ce projet proposera divers services : base de données firebase, google analytics, hosting, functions…
- Nous allons juste utiliser le service Hosting pour déployer notre application.

```
1. -> Accéder à la console -> Ajouter un projet -> nom du projet : 'hosting1' -> continuer
2.
3. Configurer Google Analytics -> Créer un compte : 'hosting1-google-analytics' -> enregistrer
4.
5. -> créer un projet
6.
7. menu de gauche -> hosting -> commencer
```

On installe en global les outils pour angular-cli afin de pouvoir lancer les commandes firebase :

```
1. npm install -g firebase-tools@latest
```

Il faut se connecter afin qu'angular-cli soit lié avec le compte firebase que vous avez créé :

```
1. firebase login                          // le navigateur chrome va s'ouvrir pour vous demandez
 de vous connecter
```

La commande suivante va effectuer quelques modifications des fichiers de votre projet pour initialiser le déploiement :

```
1. firebase init
2. ? Are you ready to proceed? (Y/n)    Y
3. -> Use an existing project           Y
4.
5. (*) Hosting             <barre espace>      pour sélectionner
6.                         <touche entrée>     pour valider
7.
8. -> Select a default Firebase project for this directory:                      hosting1-......
9.
10. -> ? What do you want to use as your public directory?                        dist/angular-
hosting1
11.
12. -> ? Configure as a single-page app (rewrite all urls to /index.html)? (y/N)   Y
13.
14. -> ? Set up automatic builds and deploys with GitHub? (y/N)                    N
```

La première fois et à chaque fois que vous mettez en production, il faut lancer les deux commandes suivantes :

```
1. ng build --prod                              // toujours builder en : --prod avant le
 deploiement
2. firebase deploy
```

XXVI-B - À savoir

C'est le contenu du dossier : /dist/angular-hosting1 qui est déployé dans le cloud Hosting.

XXVI-C - Résultat

Et voilà, il ne reste plus qu'à accéder à l'application en ligne : **https://hosting1-…….web.app/#/**
(voir le lien affiché à la fin du : firebase deploy).

Pour une version en production, vous devez lier ce lien avec un nom de domaine.

XXVII - Angular elements

Vous avez vu comment créer des composants web avec le framework Angular.
Sachez qu'il existe aussi dans le HTML 5 des composants web qui respectent des normes définies afin qu'ils puissent être pris en charge par les navigateurs (ils font donc partie des navigateurs).
Les composants web Angular et HTML 5 ont le même objectif principal, celui d'être réutilisables.

Voici à quoi pourrait ressembler un composant web HTML classique :

```
1. class HelloWorldClass extends HTMLElement {
2.
3.     constructor() {
4.         // Always call super first in constructor
5.         super();
6.     }
7.     …
8.     …
9. }
10.
11. customElements.define('hello-world', HelloWorldClass);
```

Utilisation dans une page HTML :

```
1. <hello-world></hello-world>
```

Un composant web Angular ou html peut être utilisé dans n'importe quel type de projet : Angular, HTML, Svelte, php…

Toutefois, nous devons adapter le composant web Angular afin qu'il puisse être utilisé hors contexte Angular pour le rendre compatible avec les spécifications html des navigateurs.

Pour cette conversion nous utiliserons donc Angular elements.

Pourquoi cela ?

Un composant web Angular s'exécute dans le « confort » que lui apporte le framework Angular, tout est fait pour rendre le code le plus propre possible.
Malheureusement les spécifications HTML des navigateurs limitent ou compliquent certains points.
En effet, quand on utilise ce composant dans une page web classique, il n'est plus dans le «confort» Angular.
Nous devons donc le convertir et l'adapter pour qu'il réponde à certaines contraintes que lui impose l'environnement web classique.
Mais rien de bien méchant, les principales restrictions que l'on doit prendre en compte sont les suivantes :
- les données d'entrées : @Input ne doivent pas avoir de type ;
- l'écriture des variables en entrée : @Input doit être en minuscules ; par exemple : selectedItem devient : selecteditem
- il faut utiliser l'auto properties pour la détection de changement de valeur des variables d'entrée : @Input (et non pas utiliser ngOnChange).

Pourquoi utiliser un composant web Angular sur une autre plateforme ?

Par exemple, on peut avoir un site web qui tourne en php, java, html… et on veut intégrer une nouvelle fonctionnalité. Celle-ci doit être dynamique et la programmer avec les langages habituels par exemple avec jQuery serait un peu trop compliqué, la rendrait moins performante et moins facile à maintenir.
Donc, un composant web Angular répondra à tous les inconvénients cités précédemment.

XXVII-A - Pratique

On va reprendre le projet sur le composant web réutilisable : une liste de choix avec sélection utilisateur.

XXVII-B - Schéma

```
1.                    index.html
 tbm.component
2.
3.          |    <button-toggle-mat
4.          |        items="..."                                    -------------------->
 @Input() items
5.  HTML    |        (1) selecteditem="..."                         -------------------->  (1)
 @Input() selecteditem
6.          |    >
7.          |
8.          |   (2) <button>envoie un choix</button>
9. ____________________________________________________________________________________________________
10.         |    écoute la balise <button-toggle-mat                 <--------------------
 @Output() item (le choix de l'utilisateur)
11.         |        si      une donnée est reçue du composant
12. JS      |        alors,  traite la donnée
13.         |
14.         |    (2) envoie une donnée au composant                   ---------------------->
 (1) et (2) @Input() selecteditem
15.         |    (un choix de l'utilisateur que
 met à jour la vue avec la nouvelle donnée
16.         |    l'on impose au composant)
```

(2) Ce n'est pas vraiment utile, mais j'ai mis en place cette possibilité pour montrer une communication de : index.html vers un composant

```
1. ng new web-comp-tbm
2. NO
3. NO
```

```
1. ng g c button-toggle-mat/components/button-toggle-mat --module=app
2. ng g i models/i-item-btm
3. ng g s services/stored.service.ts
```

```
1. npm i @angular/elements --save
2. ng add @angular/material
3. npm install fs-extra concat --save-dev                  // --save-dev       package qui sera
 utilisé uniquement en dev
4.                                                         //                  (ne sera pas
 intégré pour la version en prod)
```

package.json

```
1. ...
2. {
3. "scripts": {
4.     ...
5.     "build:elements": "ng build --prod --output-hashing none && node concatenate.js"
6. },
```

concatenate.js ----------> le fichier à exécuter via node

node concatenate.js ----------> lance la concaténation des fichiers .js en un seul fichier

concatenate.js

```
const _dir = './dist/web-comp-tbm';                    // les fichiers à concaténer
const _output_js_name = 'web-comp-tbm.js';             // le fichier final

const fs = require('fs-extra');
const concat = require('concat');
(async function build() {
const files = [                                        // liste des fichiers
  _dir + '/runtime.js',
  _dir + '/polyfills.js',
   // dir + '/scripts.js',
  _dir + '/main.js',
]
await fs.ensureDir('elements')
await concat(files, 'elements/' + _output_js_name);
await fs.copyFile(_dir + '/styles.css', 'elements/styles.css')      // copie le fichier .css
 dans le dossier : /elements
//await fs.copy(_dir + '/assets/', 'elements/assets/' )
})()
```

/src/app/app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule, CUSTOM_ELEMENTS_SCHEMA } from '@angular/core';
//
import  { Injector } from '@angular/core';
import  { createCustomElement } from '@angular/elements';
import { ButtonToggleMatComponent } from './button-toggle-mat/components/button-toggle-mat/
button-toggle-mat.component';
import { MatButtonToggleModule } from '@angular/material/button-toggle';
import { StoredService } from './services/stored.service';

@NgModule({
  declarations: [ButtonToggleMatComponent, ],
  imports: [ BrowserModule, MatButtonToggleModule, ],
  entryComponents : [ ButtonToggleMatComponent, ],              // le composant d'entrée
 puisque bootstrap n'est pas défini pour un composant web
  providers: [ StoredService, ],                                // un service à la porté :
 root (puisque qu'il est déclaré dans : app.module)
  schemas: [ CUSTOM_ELEMENTS_SCHEMA, ],                         // indique que le composant
 n'est pas du type : Angular
})
export class AppModule {
  constructor(private injector : Injector) {}                   // injector : intégrer dans
 le composant le service d'injection de dépendances

  ngDoBootstrap(){
    const el = createCustomElement(ButtonToggleMatComponent, {injector : this.injector});
    customElements.define('button-toggle-mat', el);             // 'button-toggle-mat' : on
 peut renommer le nom de la balise ici
  }
}
```

/src/app/models/i-item-btm.ts

```
export interface IItemBtm {
  key: string;
  value: string;
}
```

/src/app/services/stored.service.ts

```
import { IItemBtm } from '../models/i-item-btm';

```

```
3. export class StoredService {
4.
5.   selectedItemGroup: IItemBtm[] = <IItemBtm[]>[];     //  un tableau contenant les couples :
 «nom de groupe» : sélection utilisateur
6.
7.   constructor() { }
8.
9.   getSelectedItem(group: string): IItemBtm {          // on récupère le choix utilisateur par
 son groupe
10.     if (this.selectedItemGroup[group] !== undefined) {
11.       return this.selectedItemGroup[group];
12.     }
13.     return undefined;
14.   }
15.
16.   setSelectedItem(item: IItemBtm, group: string) {    // on enregistre le choix utilisateur
 par son groupe
17.     if (item !== undefined) {
18.       this.selectedItemGroup[group] = item;
19.     }
20.   }
21.
22.   getInitializedItem(defaultItem: IItemBtm, group: string): IItemBtm {      // on calcul en
 fonction de la valeur par défaut et de la valeur enregistrée du tableau
23.     if (defaultItem && this.getSelectedItem(group) === undefined) {         // si il y a une
 valeur par défaut et pas de sélection utilisateur enregistré alors...
24.       this.setSelectedItem(defaultItem, group);                             // c'est le choix
 par défaut qui est pris en compte
25.       return defaultItem;
26.     }
27.     return this.getSelectedItem(group);                                     // sinon c'est le
 choix enregistré dans le tableau qui est pris en compte
28.   }
29. }
```

/src/app/button-toggle-mat/components/button-toggle-mat/button-toggle-mat.component.html

```
1. <p>
2.   Votre choix ?  
3.   <mat-button-toggle-group name="fontStyle" aria-label="Font
 Style" value="selectedItemObj?.key" (change)="onChange($event)">
4.     <ng-container *ngFor="let item of itemsObj">
5.       <mat-button-toggle value="{{item.key}}" [checked]="item.key ==
 selectedItemObj?.key">{{item.value}}</mat-button-toggle>
6.     </ng-container>
7.   </mat-button-toggle-group>
8. </p>
9.
10. selectedItemObj={{selectedItemObj|json}}
```

/src/app/button-toggle-mat/components/button-toggle-mat/button-toggle-mat.component.ts

```
1. import { Component, Input, OnInit, Output, EventEmitter, SimpleChanges,
 OnChanges } from '@angular/core';
2. import { IItemBtm } from '../../models/i-item-btm';
3. import { StoredService } from '../../../services/stored.service';
4.
5. @Component({
6.   selector: 'app-button-toggle-mat',
7.   templateUrl: './button-toggle-mat.component.html',
8.   styleUrls: ['./button-toggle-mat.component.scss']
9. })
10. export class ButtonToggleMatComponent implements OnInit {
11.
12.   // réception des données provenant des balises du contexte web (hors Angular)
13.   @Input() items;                          // ne pas mettre de type
14.   @Input() set selecteditem(strItem) {                                    //
 selecteditem provenant du contexte web
```

```
15.                                                                                 // set
 selecteditem(strItem) est la technique de l'auto properties
16.         this.selectedItemObj = JSON.parse(strItem) as IItemBtm;                  //
 conversion : chaine -> Objet IItemBtm
17.                                                                                 // as
 IItemBtm -> cast avec IItemBtm
18.                                                                                 //
     (si pas du type IItemBtm alors cela génère une erreur)
19.         this.storedService.setSelectedItem(this.selectedItemObj, this.group);
20.    };
21.    @Input() group;                          // ne pas mettre de type
22.
23.    //  système d'envoi d'une donnée au contexte web
24.    @Output() selectedItemEvent = new EventEmitter<IItemBtm>();
25.
26.    // pour la vue
27.    itemsObj: IItemBtm[];                // les données convertis au format Objet
28.    selectedItemObj: IItemBtm;           //
29.
30.    constructor(private storedService: StoredService) { }
31.
32.    ngOnInit(): void {                                                  // Initialisation
33.      console.log('depuis le composant, action dans ngOnInit() : (initialisation du
 composant)');
34.
35.      if (!this.items) {          // les items sont obligatoires sinon on émet une erreur
36.        throw 'Vous devez transmettre une liste d\'éléments dans la balise. items="..."';
37.      }
38.
39.      this.itemsObj = JSON.parse(this.items);                                   //
 conversion : chaine -> Objet
40.
41.      if (this.selecteditem !== undefined) {
42.        this.selectedItemObj = JSON.parse(this.selecteditem) as IItemBtm;       //
 conversion : chaine -> Objet IItemBtm
43.      }
44.
   this.selectedItemObj = this.storedService.getInitializedItem(this.selectedItemObj, this.group);
45.    }
46.
47.    onChange(event: any) {                      // l'utilisateur clic sur un des choix du
 composant
48.      console.log('depuis le composant, action dans onChange() : (clic sur un choix)');
49.
50.      const item: IItemBtm = this.itemsObj.filter((item: IItemBtm) => item.key ==
 event.value)[0];  // récupère l'objet entier par la clé : KEY
51.      this.selectedItemObj = item;
52.      this.storedService.setSelectedItem(item, this.group);
53.      this.selectedItemEvent.emit(item);        // envoi au contexte web (hors Angular) l'item
 qui a été sélectionné
54.    }
55. }
```

/src/app/button-toggle-mat/components/button-toggle-mat/button-toggle-mat.component.css

```
1. @import '@angular/material/prebuilt-themes/deeppurple-amber.css';
```

/elements/index.html

```
1. <!doctype html>
2. <html lang="fr">
3. <head>
4.   <title>Angular elements</title>
5. </head>
6. <body>
7.
8.   <div style="background: #faffee; padding: 32px;">
9.     <h3>origine : index.html</h3>
10.
```

```
    <button id="choice-2">mise à jour avec le choix 2</button>
    <p id="choice-value"></p>

    <hr>

    <div style="background: #ffcfff; padding: 32px; margin-left: 32px;">
      <h3>origine : dans le composant : ButtonToggleMat</h3>

      <button-toggle-mat
        id="btm1"
        items='[{"key":"KEY1", "value":"choix 1"}, {"key":"KEY2", "value":"choix 2"}]'
        selecteditem='{"key":"KEY1", "value":"choix 1"}'
        group="'choice'"
      ></button-toggle-mat>

    </div>
  </div>

  <!-- le fichier : web-comp-tbm.js (le composant version JavaScript) -->
  <script src="web-comp-tbm.js"></script>

  <script>
    const btm = document.getElementById('btm1');              // 'btm1' est l'ID de l'élément
    const text = document.getElementById('choice-value');     // l'élément HTML pour afficher le choix en cours

    btm.addEventListener('selectedItemEvent', event => {      // on se branche sur la sortie Event du composant. «selectedItemEvent» -> voir Output() du composant
      text.innerHTML = JSON.stringify(event.detail);          // met à jour l'élément HTML : text avec la valeur reçue du composant
    });

    document.getElementById('choice-2').addEventListener('click', event => {  // on écoute l'élément HTML, si un clic est effectué
      const obj = {"key":"KEY2", "value":"choix 2"};

      btm.selecteditem = JSON.stringify(obj);                                 // on met à jour le @Input() du composant avec la nouvelle sélection
      // ou : btm.setAttribute('selecteditem', JSON.stringify(obj));          // dans un composant, quand un @Input est modifié cela déclenche : ngOnChanges() du composant
      text.innerHTML = JSON.stringify(obj);                                   // met à jour l'élément HTML : text
    });
  </script>
</body>
</html>
```

Compile, concatène les fichiers et met le résultat dans un dossier : /elements (avec le fichier : index.html déjà présent).

```
npm run build:elements
```

Pour pouvoir nous servir du fichier : index.html nous allons utiliser un serveur : node.js.

Installation du package en global :

```
npm install http-server -g
```

Et lancement du serveur dans le dossier en question :

```
cd /elements
http-server
```

http://192.168.1.39:8080/

XXVII-C - Comment utiliser ce composant web classique dans un projet Angular

(1) Copier / coller le dossier : /elements dans un nouveau projet : app/web-components/elements

(2) Importer ensuite le fichier : web-comp-tbm.js dans le module : app.module.ts

app.module.ts

```
1. ...
2. import './web-components/elements/web-comp-tbm.js';          // juste cette ligne et rien
 d'autre
3. ...
```

(3) Plus qu'à l'utiliser dans un composant Angular :

xxxx.component.html

```
1. <button-toggle-mat
2. items='[{"key":"KEY1", "value":"choix 1"}, {"key":"KEY2", "value":"choix 2"}]'
3. selecteditem='{"key":"KEY2", "value":"choix 2"}'
4. group="'choice'"
5. ></button-toggle-mat>
```

XXVIII - Docker

Docker fournit un environnement de déploiement pour chaque projet.

On travaille souvent sur plusieurs projets en même temps ou alors on veut fournir simplement le même environnement de travail aux collègues pour qu'ils puissent intervenir sur un projet.

Sans cela, il faudrait installer un environnement de travail sur son propre système pour chaque projet.

Il y a quelques années on utilisait les machines virtuelles avec vmware ou virtualbox, mais ces solutions sont très coûteuses en place et en performance.
Docker apporte simplicité, performance et taille réduite.

XXVIII-A - Installation

Télécharger et installer Docker Desktop sur votre système.
Docker Desktop est une application native qui fournit tous les outils Docker à votre ordinateur.

https://www.docker.com/

Sur le Docker Desktop, il faut créer un compte et se connecter.

XXVIII-B - Remarques

Sur windows 10, il faut activer Hyper-v, voir : **https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/quick-start/enable-hyper-v**

XXVIII-C - Pratique

XXVIII-C-1 - Cas 1 : simplement tester la version en prod (/dist)

```
1. ng new angular-docker1
2. NO
3. SCSS
4.
5. cd angular-docker1
6.
7. ng build --prod                          // générer l'application dans /dist
```

Créer le fichier Dockerfile (sans extension) dans le projet Angular :

/angular-docker1/Dockerfile-prod

```
1. FROM nginx:1.17.1-alpine
2. WORKDIR /usr/share/nginx/html
3. COPY nginx.conf /etc/nginx/nginx.conf
4. COPY ./dist/angular-docker1 /usr/share/nginx/html
```

Créer le fichier : nginx.conf pour configurer le serveur NGINX afin de fournir l'application Angular :

/angular-docker1/nginx.conf

```
1. events {
2.         worker_connections 768;
3.         # multi_accept on;
4. }
5.
6. http {
7.     include /etc/nginx/mime.types;
8.     server {
9.         listen 80;
10.         server_name localhost;
11.         root /usr/share/nginx/html;
12.         index index.html;
13.         location / {
14.             try_files $uri $uri/ /index.html;
15.         }
16.     }
17. }
```

Créer une image Docker :

```
1. docker build -f Dockerfile-prod -t angular-docker1-prod-image .
2.
3. // -f   préciser le fichier Docker qui correspond à la version en production
4. // -t  indiquer le nom de l'image que l'on veut obtenir
```

Créer le container (à partir de l'image) :

```
1. docker run --name angular-docker1-prod-container -d -p 3000:80 angular-docker1-prod-image
2.
3. --name angular-docker1-prod-container              nom du container utilisé
4. -d                                                 en arrière plan
5. -p 3000:80                                         le port du container à votre local (le
 port : 3000 en local correspond au port : 80 dans le container)
6. angular-docker1-prod-image                          nommage de l'image docker que l'on
 souhaite
```

Accéder à l'application Angular qui tourne dans un container Docker via le port : 3000 à cette adresse : **http://localhost:3000**

XXVIII-C-2 - Cas 2 : en dev avec le hot reload (ou live reload)

```
1. cd angular-docker1
```

/angular-docker1/Dockerfile-dev

```
1. FROM node:12-alpine
2. WORKDIR /app
3. COPY package.json .
4. RUN npm install
5. COPY . .
6. EXPOSE 5600 49153
7. CMD npm run start
```

/angular-docker1/docker-compose.yml

```
1. version: "3.7"
2. services:
3.   dashboard:
4.     build:
5.       context: .
6.       dockerfile: Dockerfile-dev
7.     ports:
8.       - "5600:4200"
9.       - "49153:49153"
10.     volumes:
11.       - "/app/node_modules"
12.       - ".:/app"
```

/angular-docker1/package.json

```
1. ...
2.     "start": "ng serve --host 0.0.0.0 --poll 500",                    // si votre système
 est un Windows
3.                                                                       // --poll 500
 vérifie toutes les 500ms une éventuelle modification du code
4. ...
5.     "start": "ng serve --host 0.0.0.0",                               // si votre système
 est un linux
6. ...
```

```
1. docker-compose up
2.
3. // ignorer les WARN
```

http://localhost:5600/

Faites des modifications de code et constatez que le live reload fonctionne.

XXVIII-C-2-a - À savoir

Dans le Docker Desktop, on peut visionner les containers en cours. S'ils sont marqués dans l'état « running » c'est parfait, sinon, cela indique une éventuelle erreur.

XXVIII-C-3 - Remarques

Vous avez sans doute remarqué que pour la version en production c'est le serveur : nginx qui doit founir les fichiers.

Pour la version en développement, c'est la commande : ng qui s'occupe de cette tâche afin d'avoir le live-reload.

XXVIII-C-4 - Quelques commandes utiles

On peut faire un certain nombre d'actions sur les images et containers avec le Docker Desktop (voir, supprimer, relancer).

En ligne de commande, on peut aussi effectuer ces actions :

```
1. docker image ls                              // lister toutes les images qui tournent sur votre
 machine
2. docker container ls                          // lister tous les containers qui tournent sur
 votre machine
3.
4. docker-compose up                            // lancer la procédure de création des images et
 containers (des fichiers : Dockerfile et docker-compose.yml)
5. docker-compose build                         // si on modifie :  Dockerfile ou docker-
compose.yml
6. docker ps -a                                 // lister tous les containers qui tournent sur
 votre machine
7. docker rm $(docker ps -a -q)                 // supprimer tous les containers
8. docker rmi $(docker images -q)                 // supprimer toutes les images
```

Accéder en ligne de commande dans un container :

```
1. docker ps -a                          // notez les 2 ou 3 premières lettres du CONTAINER_ID sur
 lequel vous voulez aller (pas besoin de noter l'ID entier)
2. docker exec -it ??? /bin/bash         // remplacez ??? par le CONTAINER_ID que vous avez noté
```

XXVIII-D - ngx-deploy-docker

XXVIII-D-1 - prod

Précédemment nous avons écrit à la main les fichiers Dockerfile et nginx.

cette fois le package ngx-deploy-docker va le faire à notre place.

https://www.npmjs.com/package/ngx-deploy-docker

```
1. ng new angular-ngx-docker1
2. cd angular-ngx-docker1
```

Installer le package avec ng add (ng add permet entre autres de créer des fichiers à notre place).

(cela va vous demander l'id utilisateur de votre compte Docker).

```
1. ng add ngx-deploy-docker
```

Cela va créer automatiquement un fichier : Dockerfile et un fichier nginx.conf.

Facultatif : Une petite vérification pour voir si vous être bien connecté à Docker avec votre compte.

```
1. docker login
```

Déployez votre nouvelle image. Votre projet sera automatiquement construit en mode production :

```
1. ng deploy
```

Chercher le nom de l'image qui a été créée précédemment :

```
1. docker image ls
```

Éxecuter le container à partir de l'image, nous choisissons le port 3000 :

```
1. docker run --name angular-ng-docker1-container -d -p 3000:80 ??????????/angular-ng-docker1
2.
3.     -p 3000:80                                  // le port 3000 pour accéder à l'application
4.     --name angular-ng-docker1-container        // on donne un nom pour le container
5.     docker91019/angular-ng-docker1               // le nom de l'image du : 'docker image ls'
```

localhost:3000

XXIX - Étude de cas n°1 : authentification + accès sécurisé à une API

Dans cet exemple de projet, nous allons mettre en place une application qui va se connecter à un serveur d'authentification JWT et récupérer des produits. Dans le dossier : /pack_auth1, le projet est donc divisé en deux parties :

- /pack_auth1
    - /angular-auth-jwt1 l'application Angular.
    - /node-api le serveur node.js.

Pour l'application Angular, voici les fonctionnalités :

- se connecter ;
- s'inscrire ;
- accéder à une API sécurisée ;
- sur la page 1, n'afficher les produits que si un utilisateur est connecté ;
- n'accéder à la page 2 que si un utilisateur est connecté.

Pour le serveur node.js :

- /login : reçoit l'email et le mot de passe, vérifie que l'utilisateur existe et renvoit un token + les rôles.
- /register : reçoit l'email et le mot de passe, enregistre l'utilisateur et renvoit un token + les rôles.
- /api/products : renvoie une liste de produits en json, si dans la requête est présent un token qui correspond à un utilisateur existant.

XXIX-A - Limitation

Pour ne pas alourdir le tutoriel, j'ai volontairement limité certains points :

- l'API serveur envoie une liste de produits et rien d'autre ;
- le refresh token n'est pas pris en compte ;
- à l'enregistrement d'un compte, pour l'exemple, on met le rôle "admin";
- le design et l'ergonomie ne sont pas au point.
- c'est une authentification par token et ce dernier on l'enregistre coté client (JavaScript), c'est moyen niveau sécurité. J'ai fait ce choix pour ne pas alourdir le tutoriel.
- pour un niveau de sécurité maximum, on aurait pu utiliser une authentification par cookie :
    - pour cela, il faut prévoir la gestion du cookie coté serveur ;

- à la connexion (login), le serveur sauvegarde le token dans un cookie tout en l'associant à l'identité (id...) de celui qui s'est connecté ;
- ainsi quand vous demandez d'accèder à une ressources api, le serveur vérifie dans le cookie le token et autorise ou pas l'accès ;
- quand on envoi une requête à un serveur, celui çi sait qui vous êtes, si vous vous êtes connecté auparavant... grâce au cookie coté serveur ;

XXIX-B - Schéma

```
1. /pack_auth1
2.         /angular-auth-jwt1
3.                 /core
4.                     /auth
5.                      /directives
6.                       hasRole    // une directive pour limiter certaines parties aux
 utilisateurs avec les rôles appropriés
7.                      /enums
8.                       role    // liste des rôles
9.                         /interceptors
10.                            jwt                // intercepte les requêtes vers : /API pour
 lui ajouter le token de l'utilisateur connecté
11.                        /models
12.                            i-current-user     // modèle de l'utilisateur courant,
 contient : l'état, le token...
13.                        /services
14.                            auth               // code métier pour l'authentification d'un
 utilisateur
15.                        /guards
16.                            logged             // donne l'accès à une page si un utilisateur
 est authentifié
17.                    /http
18.                        /models
19.                            base               // doit posséder un id (comme toute entité
 d'une API digne de ce nom)
20.                        /services
21.                            http               // code métier pour effectuer les requêtes
 http : get, post...
22.
23.                /features
24.                    /auth
25.                        /components
26.                            /login
27.                                ... composant
28.                            /register
29.                                /validators
30.                                    MustMatch
31.                                ... composant
32.                    /product
33.                        /models
34.                            i-product
35.                        /services
36.                            product-api        // utlise le service de requêtes :
 http.service situé dans : /core/auth
37.
38.                /pages
39.                    /page-home
40.                        ... routing + module + composant
41.                    /page-register
42.                        ... routing + module + composant
43.                    /page-login
44.                        ... routing + module + composant
45.                    /page1                                          // affiche les produits
 si un utilisateur est authentifié
46.                        ... routing + module + composant
47.                    /page2                                          // page2 est accessible
 si un utilisateur est authentifié
48.                        ... routing + module + composant
49.                    /partials
50.                        /header
```

```
51.                             ... module + composant
52.
53.                 /shared
54.                     /material-design
55.
56.                 app
57.                     ... routing + module + composant
58.
59.         \node-api
60.                 server.js           // serveur d'authentification JWT (login + inscription) +
 api/products
61.                 package.json
62.                 ...
63.
64.         docker-compose.yml      // lance deux containers basés sur les images ci dessous.
 (appli. Angular: http://localhost:5600) (serveur node: http://localhost:8000)
65.         Dockerfile.ng-app       // image de l'application Angular en mode DEV (live-reload)
66.         Dockerfile.node-api     // image du serveur node.js : serveur d'authentification JWT
 + API de produits (liste en json)
```

Les pages utilisent les features et l'ensemble utilise core.
Tout est bien classé, organisé. Vous pouvez reprendre le dossier /core pour un autre projet.

XXIX-C - L'application Angular : /angular-auth-jwt1

XXIX-C-1 - À savoir

XXIX-C-1-a - Interceptors

Les intercepteurs nous permettent d'intercepter les requêtes HTTP entrantes ou sortantes à l'aide du HttpClient. En interceptant la requête HTTP, nous pouvons modifier ou changer la valeur de la requête.

Pour l'application ?

Nous l'utilisons pour les requêtes http vers l'api.
En effet, plutôt que de rajouter le token d'accès directement aux requêtes http de base : get, post… nous interceptons les requêtes vers l'APIet lui ajoutons le token.

XXIX-C-1-b - Guards

Les gardes de route d'Angular sont des interfaces qui peuvent dire au routeur si oui ou non il doit permettre la navigation selon un itinéraire demandé. On peut l'autoriser ou pas en fonction de divers critères que l'on détermine : l'utilisateur est authentifié ? Il a un rôle précis ?
Les différents types de guard : CanActivate, CanActivateChild, CanDeactivate, CanLoad et Resolve.

Pour l'application ?

Nous devons autoriser la navigation vers la page 2 si l'utilisateur est connecté.
Pour cela, nous utiliserons : CanActivate

XXIX-C-1-c - JWT

Qu'est-ce que JWT ?

JSON web Token est un standard utilisé pour créer des jetons d'accès pour une application.
Le serveur génère un jeton qui certifie l'identité de l'utilisateur et l'envoie au client.

Le client renverra le jeton au serveur pour chaque demande suivante, afin que le serveur sache que la demande provient d'une identité particulière.

Pour l'application ?

Autoriser l'accès aux produits de l'API uniquement aux utilisateurs authentifiés (ayant un token valide).

XXIX-C-1-d - Le service http et les generics

Nous voulons écrire un service http qui s'adapte à tous les types de données : IProduct, ICategory, IOrder… plutôt que d'écrire un service http par type de données.

Par exemple, pour accéder à l'API product, on pourrait avoir :

product-api.service.ts

```
this.httpClient.get<IProduct[]>(...
this.httpClient.delete<IProduct>(...
this.httpClient.post<IProduct>(...
...
```

Les requêtes sont dépendantes du type IProduct
Il faut donc écrire un service avec le même code pour chaque type différent.

La solution est d'utiliser les generics de TypeScript :

http.service.ts

```
export abstract class HttpService<T> {
    ...
    this.httpClient.get<T[]>(...
    this.httpClient.delete<T>(...
    this.httpClient.post<T>(...
}
```

Ainsi T peut valoir : IProduct, ICategory…

Son utilisation est la suivante, par exemple :

category-api.service.ts

```
export class CategoryApiService extends HttpService<ICategory> {    // <ICategory>  on précise le type que l'on veut pour : T
}
```

product-api.service.ts

```
export class ProductApiService extends HttpService<IProduct> {      // <IProduct>   on précise le type que l'on veut pour : T
}
```

XXIX-C-1-e - Le modèle de l'utilisateur courant ICurrentUser et le service auth.service.ts

Le modèle ICurrentUser permet de sauvegarder toutes les informations concernant une authentification de l'utilisateur courant.
Il est inscrit dans le service auth.service dans un Observable.

/core/auth/models/i-current-user.ts

```
export interface ICurrentUser {
  email?: string;
  password?: string;
  token?: string;
  isLogged?: boolean;
  ...
```

Le service AuthService

/core/auth/services/auth.service.ts

```
...
  private currentUserSubject = new BehaviorSubject<ICurrentUser>({} as ICurrentUser);
  public currentUser$: Observable<ICurrentUser>;

  constructor(private http: HttpClient) {
    this.currentUser$ = this.currentUserSubject.asObservable();
  }
...
```

Tous les composants qui souscrivent à currentUserSubject seront informés de la connexion ou de la déconnexion d'un utilisateur et recevront les informations de l'utilisateur courant ICurrentUser

L'application devra donc émettre l'état de : ICurrentUser sur cet observable à ces moments clés :

- au lancement de l'application ;
- à la connexion ;
- à l'inscription ;
- à la déconnexion.

Nous avons choisi le sujet de type BehaviorSubject afin que si un composant (ou page) est initialisé plus tard, à la souscription, il reçoive automatiquement le dernier état courant de l'utilisateur.

Par exemple le composant responsable du message de bienvenue s'abonne à cet observable afin de connaitre en temps réel si un utilisateur c'est connecté ou déconnecté.

XXIX-C-1-f - Bonnes pratiques

Un Subject est à la fois un Observable où l'on peut souscrire et un Observer où l'on peut émettre.
Avec un Observable on peut uniquement souscrire.

Avec :

```
...
  private currentUserSubject = new BehaviorSubject<ICurrentUser>({} as ICurrentUser);   // Observable et Observer
  public currentUser$: Observable<ICurrentUser>;                                        // Observable
...
  constructor(private http: HttpClient) {
    this.currentUser$ = this.currentUserSubject.asObservable();        // Observable : un lien vers l'observable du sujet : currentUserSubject
...
```

```
currentUser$     --------->     est la partie observable de currentUserSubject
```

- pour un composant qui ne veut que souscrire on utilise : currentUser$.subscribe(...

- pour un composant qui doit également émettre un nouvel utilisateur, on utilise : currentUserSubject.

Pourquoi est-ce une bonne pratique ?

Un composant qui a pour seule responsabilité de souscrire ne doit pas pouvoir émettre afin d'éviter toute erreur.

XXIX-C-2 - Pratique

```
cd pack_auth1

ng new angular-auth-jwt1
 strict ?  NO
    routing ? yes
    SCSS

cd angular-auth-jwt1

ng g m pages --module=app
ng g m features --module=pages
ng g m core --module=features
ng g m core --module=pages --force

ng g m pages/page-home --module=pages --routing
ng g c pages/page-home --module=page-home
ng g m pages/page1 --module=pages --routing
ng g c pages/page1 --module=page1
ng g m pages/page2 --module=pages --routing
ng g c pages/page2 --module=page2
ng g m pages/page-register --module=pages --routing
ng g c pages/page-register --module=page-register
ng g m pages/page-login --module=pages --routing
ng g c pages/page-login --module=page-login
ng g m pages/partials --module=pages
ng g c pages/partials/header --module=partials

ng g m features/auth --module=features
ng g c features/auth/components/login --module=auth
ng g c features/auth/components/register --module=auth
ng g m features/product --module=features
ng g s features/product/services/product-api
ng g i features/product/models/i-product

ng g m shared/material-design --module=app

ng g s core/auth/services/auth
ng g i core/auth/models/i-current-user
ng g interceptor core/auth/interceptors/jwt
ng g guard core/auth/guards/logged
    (*)  CanActivate
ng g s core/http/services/http
ng g i core/http/models/base
ng g d core/auth/directives/has-role --module=core
```

```
ng add @angular/material
```

XXIX-C-2-a - /core

\core\auth\directives\has-role.directive.ts

```
import { Directive, Input, ViewContainerRef, TemplateRef, OnInit, OnDestroy,
 } from '@angular/core';
import { Subscription } from 'rxjs';
import { ICurrentUser } from '../models/i-current-user';
import { AuthService } from '../services/auth.service';

@Directive({
```

```
7.   selector: '[appHasRole]'
8. })
9. export class HasRoleDirective implements OnInit, OnDestroy {
10.   @Input() appHasRole: Array<string>;   // réception de la valeur(du rôle souhaité) défini
 dans le template
11.   subCurrentUserObs: Subscription;      // pour contenir l'observable (afin de pouvoir se
 désabonner dans le ngOnDestroy)
12.
13.   constructor(
14.     private viewContainerRef: ViewContainerRef,
15.     private templateRef: TemplateRef<any>,
16.     private authService: AuthService
17.   ) {}
18.
19.   ngOnInit(): void {
20.
21.     // on souscrit à CurrentUserObs et donc à chaque changement d'utilisateur, on reçoit le
 nouveau : user
22.     this.subCurrentUserObs = this.authService.getCurrentUserObs().subscribe((user:
 ICurrentUser) => {
23.
24.       // on oblige à ce qu'il y est au moins un rôle qui est défini dans l'utilisation de la
 directive du template
25.       if (!this.appHasRole || !this.appHasRole.length) {
26.         throw new Error('attention, il n\'y a pas de rôle défini');
27.       }
28.
29.       let hasAccess = false;
30.       if (user.roles) {
31.           hasAccess = user.roles.some(role => this.appHasRole.includes(role));     // some -->
 pour tous les roles contenu dans user
32.       }
33.       if (hasAccess) {
34.           this.viewContainerRef.createEmbeddedView(this.templateRef);
35.       } else {
36.           this.viewContainerRef.clear();
37.       }
38.     });
39.   }
40.
41.   ngOnDestroy(): void {
42.     this.subCurrentUserObs.unsubscribe();   // important : toujours, toujours se désabonner !
43.   }
44. }
```

\core\auth\enums\role-enum.ts

```
1. export enum RoleEnum {
2.   ADMIN = "admin",
3.   USER = "user",
4. }
```

/core/auth/guards/logged.guard.ts

```
1. import { Injectable } from '@angular/core';
2. import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, UrlTree } from '@angular/
router';
3. import { Observable } from 'rxjs';
4. import { AuthService } from '../../auth/services/auth.service';
5.
6. @Injectable({
7.   providedIn: 'root'
8. })
9. export class LoggedGuard implements CanActivate {       // on implémente l'interface :
 CanActivate
10.                                                          // liste des autres interfaces :
 CanActivate, CanActivateChild, CanDeactivate, CanLoad et Resolve
11.                                                          // vous pouvez allez voir son
 utilisation pour la page 2 dans : app-routing.module.ts
```

```
12.   constructor(private authService: AuthService){}
13.
14.   canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot):
 Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
15.       return this.authService.isLogged();               // isLogged est un observable qui
 retourne true ou false
16.   }
17.
18.   // remarquez les différents types que l'on peut retourner avec la méthode : canActivate
19.   // -------->  : Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean |
 UrlTree
20. }
```

/core/auth/interceptors/jwt.interceptor.ts

```
1. import { Injectable } from '@angular/core';
2. import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor } from '@angular/common/http';
3. import { AuthService } from '../services/auth.service';
4. import { Observable, throwError } from 'rxjs';
5. import { catchError } from 'rxjs/operators';
6. import { ICurrentUser } from '../models/i-current-user';
7. import { environment } from 'src/environments/environment';
8.
9. @Injectable()
10. export class JwtInterceptor implements HttpInterceptor {
11.
12.   constructor(public authService: AuthService) { }
13.
14.   intercept(request: HttpRequest<unknown>, next: HttpHandler):
 Observable<HttpEvent<unknown>> {
15.
16.     const isApiUrl = request.url.startsWith(environment.urlApi + '/' + environment.pathApi);

17.
18.     if (isApiUrl) {                                          // si c'est une requête vers
 l'api
19.       const currentUser: ICurrentUser = this.authService.currentUserValue;
20.
21.       if (currentUser && currentUser.token) {
22.         request = this.addToken(request, currentUser.token);    // on ajoute le token à la
 requête
23.       }
24.
25.       return next.handle(request).pipe(                        // on envoie la requête
26.
27.         catchError((error) => {                                // gestion d'une éventuelle
 erreur
28.
29.           if (error.error.status == 401) {                     // si l'erreur est : 401
 Unauthorized
30.
31.             this.authService.logout();                         // si pas de refresh token,
 on se déconnecte (car le token est non valide)
32.                                                                // en effet le token peut
 devenir non valide lorsqu'il a expiré
33.                                                                // le temps d'expiration est
 réglable dans le fichier /node-api/server.js
34.             //
35.             //  avec un refresh token, mettez en place ici la demande d'un nouveau token
36.             //
37.           }
38.           return throwError(error);                            // déclenche une erreur
39.         })
40.       );
41.     }
42.
43.     return next.handle(request);        // si ce n'est pas une requête vers l'api, alors on
 la renvoie tel quelle
44.   }
45.
46.   private addToken(request: HttpRequest<any>, token: string) {
```

```
47.     return request.clone({
48.       setHeaders: {
49.         'Authorization': `Bearer ${token}`
50.       }
51.     });
52.   }
53. }
```

/core/auth/models/i-current-user.ts

```
1. export interface ICurrentUser {
2.   userId?: string;
3.   email?: string;
4.   password?: string;
5.   name?: string;
6.   username?: string;
7.   roles?: Array<string>;
8.   token?: string;
9.   refresh_token?: string;
10.   isLogged?: boolean;
11. }
```

/core/auth/services/auth.service.ts

```
1. import { Injectable } from '@angular/core';
2. import { HttpClient, HttpHeaders, HttpErrorResponse  } from '@angular/common/http';
3. import { BehaviorSubject, Observable, throwError } from 'rxjs';
4. import { catchError, map, take } from 'rxjs/operators';
5. import { environment } from '../../../../environments/environment';
6. import { ICurrentUser } from '../models/i-current-user';
7.
8. const httpOptions = {
9.   headers: new HttpHeaders({ 'Content-Type': 'application/json' }),
10. };
11.
12. export interface Data {          // à la connexion et à l'enregistrement, l'api retourne ces 2
 champs :
13.   access_token: string;
14.   roles: Array<string>;
15. }
16.
17. @Injectable({
18.   providedIn: 'root',
19. })
20. export class AuthService {
21.   private currentUserSubject = new BehaviorSubject<ICurrentUser>({} as ICurrentUser);
22.   private currentUser$: Observable<ICurrentUser>;
23.
24.   constructor(private http: HttpClient) {
25.     this.currentUser$ = this.currentUserSubject.asObservable();
26.   }
27.
28.   public get currentUserValue(): ICurrentUser {                 // récupère directement la
 valeur contenu dans le sujet
29.     return this.currentUserSubject.getValue();
30.   }
31.
32.   getCurrentUserSubject(): BehaviorSubject<ICurrentUser> {      // currentUserSubject est
 private donc il faut une fonction pour le retourner à qui le demande
33.     return this.currentUserSubject;
34.   }
35.
36.   getCurrentUserObs(): Observable<ICurrentUser> {               // la partie Observable de
 currentUserSubject
37.     return this.currentUser$;
38.   }
39.
40.   login(email: string, password: string): Observable<ICurrentUser> {
41.     return this.http
```

```
42.       .post<Data>(
43.         `${environment.urlApi}/${environment.pathAuth}/login`,  // environnement est soit
 celui en PROD ou en DEV, voir : /src/environnements
44.         { email, password },
45.         httpOptions
46.       )
47.       .pipe(                              // pipe : pour indiquer que l'on va utiliser une
 série de traitement
48.         map((data: Data) => {             // map : on traite les données avant de l'envoyer
49.                                           // data { access_token: string; roles: Array<string>}
 et ICurrentUser { ..., roles?: Array<string>; token?: string; ...}
50.                                           // on remarque que des 2 cotés la syntaxe "roles" et
 le type sont égaux
51.                                           // par contre, access_token d'un coté et token de
 l'autre, je l'ai fait exprés pour vous donner un exemple
52.                                           // disons qu'on ne peut pas modifier celui envoyé par
 le serveur et on ne veut pas modifier ici dans le front
53.                                           // alors dans ce cas, on ré-ecrit le json de la façon
 suivante :
54.           return {
55.             roles: data.roles,            // on met dans la propriété roles le contenu de :
 data.roles
56.             token: data.access_token      // on met dans la propriété token le contenu de :
 data.access_token
57.           } as ICurrentUser;              // de plus on cast l'objet en : ICurrentUser, pour
 indiquer qu'on veut absolument que l'objet soit du type : ICurrentUser
58.         }),
59.         catchError(this.handleError)      // intercepte une éventuelle erreur et la renvoit
 dans la méthode :  handleError afin qu'elle y soit géré
60.                                           // (pour déporter et factoriser la gestion d'erreur)
61.       );
62.   }
63.
64.   register(email: string, password: string): Observable<ICurrentUser> {
65.     return this.http
66.       .post<Data>(
67.         `${environment.urlApi}/${environment.pathAuth}/register`,
68.         { email, password },
69.         httpOptions
70.       ).pipe(
71.         map((data: Data) => {             // à propos de Data, on type le retour car on
 s'attends à recevoir des données sous la forme de Data : { access_token: string; roles:
 Array<string>}
72.                                           // si un jour, une erreur arrive sur le back, que
 l'on ne reçoit pas exactement le type Data alors une erreur survient ici sur le front.
73.                                           // (si un changement a lieu sur le back alors ils
 doivent avertir les devs front qu'une modification à eu lieu pour faire un correctif)
74.                                           // de plus , grace au typage, l'erreur est detecté
 très tôt dans le code et le jour d'un problème on sait exactement ou cela se situe
75.                                           // si on avait mis "data: any", il n'y aurait pas eu
 d'erreur, le code aurait continué jusqu'à faire une autre erreur ou bizarrerie de fonctionnement
76.                                           // et cela aurait été plus difficile à débugguer ou
 engendrer des données éronnées en base de données
77.           return {
78.             roles: data.roles,
79.             token: data.access_token
80.           } as ICurrentUser;
81.         }),
82.         catchError(this.handleError)
83.       );
84.   }
85.
86.   logout() {
87.     const user = {} as ICurrentUser;        // un currentUser vide
88.     this.updateAndEmitCurrentUser(user);             // on enregistre et informe qu'une
 déconnexion à eu lieu
89.   }
90.
91.   isLogged(): Observable<boolean> {
92.     return this.currentUser$.pipe(
93.       map((user: ICurrentUser) => user.isLogged),   // la valeur qui doit être retourné est :
 isLogged, les autres ne nous intéresse pas
```

```
94.       take(1)
95.     );
96.   }
97.
98.   updateCurrentUser(user: ICurrentUser) {
99.     this.updateAndEmitCurrentUser(user);
100.   }
101.
102.   updateAndEmitCurrentUser(user: ICurrentUser) {
103.     localStorage.setItem("currentUser", JSON.stringify(user));      // on enregistre dans
 une petite base de donnée du navigateur.
104.                                                                    // on ne l'utilise pas
 mais je le laisse pour l'exemple au cas ou
105.     this.currentUserSubject.next(user);     // on informe tous les souscripteurs d'un nouvel
 état de : ICurrentUser
106.   }
107.
108.   // Error
109.   handleError(error: HttpErrorResponse) {
110.     if (error.error instanceof ErrorEvent) {
111.       // client-side error
112.       console.log('client-side error')
113.       return throwError(error.error.message);
114.     }
115.     // server-side error
116.     console.log('server-side error')
117.     return throwError(error);
118.   }
119. }
```

/core/http/models/base.ts

```
1. export interface Base {
2.   id: number
3. }
```

/core/http/services/http.service.ts

```
1. import { Injectable } from '@angular/core';
2. import { Observable, throwError } from 'rxjs';
3. import { catchError } from 'rxjs/operators';
4. import { HttpClient, HttpErrorResponse, HttpHeaders } from '@angular/common/http';
5. import { Base } from '../models/base';
6.
7. @Injectable({
8.   providedIn: 'root'
9. })
10. export class HttpService<T extends Base> {
11.
12.   constructor(private httpClient:
 HttpClient, private url: string, private path: string, private endpoint: string) {
13.   }
14.
15.   httpOptions = {
16.     headers: new HttpHeaders({ 'Content-Type': 'application/json' })
17.   }
18.
19.   get(): Observable<T[]> {
20.     return this.httpClient
21.       .get<T[]>(`${this.url}/${this.path}/${this.endpoint}`)
22.       .pipe(
23.         catchError(this.handleError)
24.       )
25.   }
26.
27.   getById(id: number): Observable<T> {
28.     return this.httpClient
29.       .get<T>(`${this.url}/${this.path}/${this.endpoint}/${id}`)
30.       .pipe(
```

```
31.         catchError(this.handleError)
32.       )
33.   }
34.
35.   create(item: T): Observable<T> {
36.
  return this.httpClient.post<T>(`${this.url}/${this.path}/${this.endpoint}`, JSON.stringify(item), this.httpOpti
37.       .pipe(
38.         catchError(this.handleError)
39.       )
40.   }
41.
42.   update(item: T): Observable<T> {
43.
  return this.httpClient.put<T>(`${this.url}/${this.path}/${this.endpoint}/${item.id}`, JSON.stringify(item), thi
44.       .pipe(
45.         catchError(this.handleError)
46.       )
47.   }
48.
49.   delete(item: T) {
50.
  return this.httpClient.delete<T>(`${this.url}/${this.path}/${this.endpoint}/${item.id}`, this.httpOptions)
51.       .pipe(
52.         catchError(this.handleError)
53.       )
54.   }
55.
56.   private handleError(error: HttpErrorResponse){
57.     let errorMessage = '';
58.
59.     if(error.error instanceof ErrorEvent){
60.       // error client
61.       errorMessage = error.error.message;
62.     } else {
63.       // error server
64.       errorMessage = `error status: ${error.status}, ` + `error message: ${error.message}`;
65.     }
66.     return throwError(errorMessage);
67.   }
68. }
```

abstract :

Vous avez remarquez le mot abstract dans la définition de la classe.

Cela permet d'indiquer que la classe sera abstraite, qu'elle ne peut pas être instancié en faisant : new HttpService()

Est seulement autorisé, qu'une classe étend celle-ci : class ..... extends HttpService...

/core/core.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3. import { HasRoleDirective } from './auth/directives/has-role.directive';
4.
5. @NgModule({
6.   declarations: [HasRoleDirective],
7.   imports: [
8.     CommonModule
9.   ],
10.   exports: [HasRoleDirective]   // ne pas oublier d'exporter la directive pour être utilisé
 ailleurs lors d'un import de CoreModule
11. })
12. export class CoreModule { }
```

XXIX-C-2-b - /features

/features/auth/components/login/login.component.html

```
<p>login works!</p>

<div id="container">
   <form [formGroup]="loginForm" (ngSubmit)="onSubmit()">

     <div>
       <mat-form-field appearance="fill" class="example-full-width">
         <mat-label>email</mat-label>
         <input matInput placeholder = "Entrez votre email" formControlName = "email" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.email.errors }">
         <mat-error *ngIf="submitted && f.email.errors" class="invalid-feedback">
           <div *ngIf="f.email.errors.required">L'email est <strong>obligatoire</strong></div>
           <div *ngIf="f.email.errors.email">L'email doit être dans un format valide</div>
         </mat-error>
       </mat-form-field>
     </div>

     <div style="height: 12px;"></div>

     <div>
       <mat-form-field appearance="fill" class="example-full-width">
         <mat-label>Password</mat-label>
         <input matInput #password placeholder = "Entrez votre email" formControlName = "password" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.password.errors }">
         <mat-hint>{{password.value?.length || 0}} caractère(s) (6 minimum)</mat-hint>
         <mat-error *ngIf="submitted && f.password.errors" class="invalid-feedback">
           <div *ngIf="f.password.errors.required">Le mot de passe est <strong>obligatoire</strong></div>
           <div *ngIf="f.password.errors.minlength">Le mot de passe doit contenir au moins 6 caractères</div>
         </mat-error>
       </mat-form-field>
     </div>

     <div style="height: 24px;"></div>

     <div>
       <button mat-raised-button color="accent" type="reset" (click)="cancel()">Effacer</button>
        
       <button mat-raised-button color="primary" type="submit">Se connecter</button>
     </div>

     <div *ngIf="error">
       <div style="height: 24px;"></div>

       <mat-error  class="invalid-feedback">
         <div>{{error}}</div>
       </mat-error>
     </div>

   </form>
</div>
```

/features/auth/components/login/login.component.ts

```
import { Component, OnInit, Output, EventEmitter } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { AuthService } from 'src/app/core/auth/services/auth.service';
import { Router } from '@angular/router';
import { ICurrentUser } from 'src/app/core/auth/models/i-current-user';
import { Subscription } from 'rxjs';

```

```
8. @Component({
9.   selector: 'app-login',
10.   templateUrl: './login.component.html',
11.   styleUrls: ['./login.component.scss']
12. })
13. export class LoginComponent implements OnInit {
14.
15.   @Output() cancelEvent = new EventEmitter<boolean>();          // averti le parent que
 l'utilisateur souhaite fermer ou annuler la demande de connexion
16.                                                                 // ainsi le parent peut
 fermer le composant : Login
17.   subLogin : Subscription;
18.   loginForm: FormGroup;
19.   submitted = false;
20.   error: string;
21.
22.   constructor(private formBuilder: FormBuilder, private authService:
 AuthService, private router: Router) { }
23.
24.   ngOnInit(): void {
25.       this.loginForm = this.formBuilder.group({
26.           email: ['', [Validators.required, Validators.email]],
27.           password: ['', [Validators.required, Validators.minLength(6)]],
28.       }, {
29.
30.       });
31.   }
32.
33.   get f() {
34.     return this.loginForm.controls;
35.   }
36.
37.   onSubmit(): void {
38.     this.submitted = true;
39.     this.error = null;
40.
41.     if (this.loginForm.invalid) {
42.       return;
43.     }
44.
45.
  this.subLogin = this.authService.login(this.loginForm.value.email, this.loginForm.value.password).subscribe((us
 ICurrentUser) => {
46.       // initialisation
47.       user.isLogged = true;
48.       user.email = this.loginForm.value.email;
49.       // enregistre et émet le nouvel utilisateur pour les composants qui ont souscrit
50.       this.authService.updateAndEmitCurrentUser(user);
51.       // clos le formulaire de connexion
52.       this.cancelEvent.emit(true);
53.       // à la connexion, on se rends à la page : /home
54.       this.router.navigateByUrl('/home');
55.     },
56.     error => {
57.       if (error.status == 401) {
58.         this.error = 'l\'email ou le mot de passe est incorrect';
59.       }  else {
60.         this.error = error.message + ' status : ' + error.status;
61.       }
62.     });
63.   }
64.
65.   cancel() {
66.     this.cancelEvent.emit(true);
67.   }
68.
69.   ngOnDestroy(): void {
70.     if (this.subLogin) {
71.       this.subLogin.unsubscribe();          // important : toujours se désabonner !
72.     }
73.   }
74. }
```

/features/auth/components/login/login.component.scss

```
#container {
  display: flex;
  justify-content: space-around;
}

#container form {
  min-width: 296px;
}
```

/features/auth/components/register/register.component.html

```
<p>register works!</p>

<div id="container">
  <form [formGroup]="registerForm" (ngSubmit)="onSubmit()">

    <div>
      <mat-form-field appearance="fill" class="example-full-width">
        <mat-label>email</mat-label>
        <input matInput placeholder = "Entrez votre email" formControlName = "email" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.email.errors }">
        <mat-error *ngIf="submitted && f.email.errors" class="invalid-feedback">
          <div *ngIf="f.email.errors.required">L'email est <strong>obligatoire</strong></div>
          <div *ngIf="f.email.errors.email">L'email doit être dans un format valide</div>
        </mat-error>
      </mat-form-field>
    </div>

    <div style="height: 12px;"></div>

    <div>
      <mat-form-field appearance="fill" class="example-full-width">
        <mat-label>Password</mat-label>
        <input matInput #password placeholder = "Entrez votre email" formControlName = "password" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.password.errors }">
        <mat-hint>{{password.value?.length || 0}} caractère(s) (6 minimum)</mat-hint>
        <mat-error *ngIf="submitted && f.password.errors" class="invalid-feedback">
          <div *ngIf="f.password.errors.required">Le mot de passe est <strong>obligatoire</strong></div>
          <div *ngIf="f.password.errors.minlength">Le mot de passe doit contenir au moins 6 caractères</div>
        </mat-error>
      </mat-form-field>
    </div>

    <div style="height: 12px;"></div>

    <div>
      <mat-form-field appearance="fill" class="example-full-width">
        <mat-label>Confirme Password</mat-label>
        <input matInput placeholder = "Entrez votre email" formControlName = "confirmPassword" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.confirmPassword.errors }">
        <mat-error *ngIf="submitted && f.confirmPassword.errors" class="invalid-feedback">
          <div *ngIf="f.confirmPassword.errors.required">La confirmation est <strong>obligatoire</strong></div>
          <div *ngIf="f.confirmPassword.errors.mustMatch">Les mots de passe sont différents</div>
        </mat-error>
      </mat-form-field>
    </div>


    <div style="height: 24px;"></div>

    <div>
```

```
48.       <button mat-raised-button color="accent" type="reset" (clic)="onReset()">Annuler</
button>
49.        
50.       <button mat-raised-button color="primary" type="submit">Inscription</button>
51.     </div>
52.
53.     <div *ngIf="error">
54.       <div style="height: 24px;"></div>
55.
56.       <mat-error  class="invalid-feedback">
57.         <div>{{error}}</div>
58.       </mat-error>
59.     </div>
60.
61.   </form>
62. </div>
```

/features/auth/components/register/register.component.scss

```
1. #container {
2.   display: flex;
3.   justify-content: space-around;
4. }
5.
6. #container form {
7.   min-width: 296px;
8. }
```

/features/auth/components/register/register.component.ts

```
1. import { Component, OnInit } from '@angular/core';
2. import { FormBuilder, FormGroup, Validators } from '@angular/forms';
3. import { AuthService } from 'src/app/core/auth/services/auth.service';
4. import { MustMatch } from './validators/MustMatch';
5. import { Router } from '@angular/router';
6. import { ICurrentUser } from 'src/app/core/auth/models/i-current-user';
7. import { Subscription } from 'rxjs';
8.
9. @Component({
10.   selector: 'app-register',
11.   templateUrl: './register.component.html',
12.   styleUrls: ['./register.component.scss']
13. })
14. export class RegisterComponent implements OnInit {
15.
16.   registerForm: FormGroup;
17.   submitted = false;
18.   subRegister: Subscription;
19.   error: string;
20.
21.   constructor(private formBuilder: FormBuilder, private authService:
 AuthService, private router: Router) { }
22.
23.   ngOnInit(): void {
24.     this.registerForm = this.formBuilder.group({
25.         email: ['', [Validators.required, Validators.email]],
26.         password: ['', [Validators.required, Validators.minLength(6)]],
27.         confirmPassword: ['', Validators.required],
28.     }, {
29.         validator: MustMatch('password', 'confirmPassword')
30.     });
31.
32.
33.     this.registerForm.controls['email'].setValue('test1@test.fr');
34.     this.registerForm.controls['password'].setValue('222222');
35.     this.registerForm.controls['confirmPassword'].setValue('222222');
36.   }
37.
38.   get f() {
39.     return this.registerForm.controls;
```

```
  }

  onSubmit(): void {
    this.submitted = true;
    this.error = null;

    if (this.registerForm.invalid) {
      return;
    }


 this.subRegister = this.authService.register(this.registerForm.value.email, this.registerForm.value.password).s
ICurrentUser) => {
        // initialisation
        user.isLogged = true;
        user.email = this.registerForm.value.email;
        // enregistre et émet le nouvel utilisateur pour les composants qui ont souscrit
        this.authService.updateAndEmitCurrentUser(user);
        // à la connexion, on se rends à la page : /home
        this.router.navigateByUrl('/home');
      },
      error => {
        if (error.status == 401) {
          this.error = 'l\'email ou le mot de passe existe déjà';
        }  else {
          this.error = error.message + ' status : ' + error.status;
        }
      }
    )
  }

  onReset(): void {
    this.submitted = false;
    this.registerForm.reset();
  }

  ngOnDestroy(): void {
    if (this.subRegister) {
      this.subRegister.unsubscribe();          // important : toujours se désabonner !
    }
  }
}
```

/features/auth/components/register/validators/must-match.validator.ts

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

export function MustMatchValidator(controlName: string, matchingControlName: string) {
 // correspond aux champs : password et confirmPassword
    return (formGroup: FormGroup) => {
        const control = formGroup.controls[controlName];                                //
password
        const matchingControl = formGroup.controls[matchingControlName];            //
confirmPassword

        if (matchingControl.errors && !matchingControl.errors.mustMatch) {    // si déjà
trouvé une erreur ailleurs dans un autre champ
            return;                                                             // alors pas
besoin d'analyser le contrôle des mots de passe
        }

        if (control.value !== matchingControl.value) {                          // si les deux
mots de passe ne correspondent pas
            matchingControl.setErrors({ mustMatch: true });                     // il y a une
erreur
        } else {
            matchingControl.setErrors(null);                                    //
        }
    }
}
```

/features/auth/auth.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3. import { LoginComponent } from './components/login/login.component';
4. import { RegisterComponent } from './components/register/register.component';
5. import { MaterialDesignModule } from 'src/app/shared/material-design/material-design.module';
6. import { ReactiveFormsModule } from '@angular/forms';
7.
8. @NgModule({
9.   declarations: [LoginComponent, RegisterComponent],
10.   imports: [
11.     CommonModule,
12.     ReactiveFormsModule,
13.     MaterialDesignModule,
14.   ],
15.   exports: [LoginComponent, RegisterComponent],
16. })
17. export class AuthModule { }
```

/features/product/models/i-product.ts

```
1. import { Base } from "src/app/core/http/models/base";
2.
3. export interface IProduct extends Base {
4.   name: string;
5.   cost: number;
6.   quantity: number;
7. }
```

/features/product/services/product-api.services.ts

```
1. import { Injectable } from '@angular/core';
2. import { environment } from '../../../../environments/environment';
3. import { HttpClient } from '@angular/common/http';
4. import { IProduct } from '../models/i-product';
5. import { HttpService } from '../../../core/http/services/http.service';
6.
7. @Injectable({
8.   providedIn: 'root'
9. })
10. export class ProductApiService extends HttpService<IProduct> {      // on hérite de la
 classe : HttpService, donc de toutes ses méthodes et propriétés
11.                                                                     // <IProduct> : on
 précise à la classe que le type 'generic' doit être du type : IProduct
12.
13.   constructor(httpClient: HttpClient) {
14.     super(                          // super : permet de faire appel au constructeur de la
 classe que l'on hérite (HttpService)
15.
16.       httpClient,
17.       environment.urlApi,
18.       environment.pathApi,
19.       environment.endPointProducts
20.     );
21.   }
22. }
```

/features/product/product.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3.
4. @NgModule({
5.   declarations: [],
6.   imports: [ CommonModule ],
7. })
8. export class ProductModule { }
```

/features/features.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3. import { CoreModule } from '../core/core.module';
4. import { AuthModule } from './auth/auth.module';
5. import { ProductModule } from './product/product.module';
6.
7. @NgModule({
8.   declarations: [],
9.   imports: [
10.     CommonModule,
11.     CoreModule,
12.     AuthModule,
13.     ProductModule
14.   ]
15. })
16. export class FeaturesModule { }
```

XXIX-C-2-c - /pages

/pages/page-register/page-login.component.html

```
1. <p>page-login works!</p>
2.
3. <app-login></app-login>
```

/pages/page-register/page-login.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3.
4. import { PageLoginRoutingModule } from './page-login-routing.module';
5. import { PageLoginComponent } from './page-login.component';
6. import { AuthModule } from 'src/app/features/auth/auth.module';
7.
8.
9. @NgModule({
10.   declarations: [PageLoginComponent],
11.   imports: [CommonModule, PageLoginRoutingModule, AuthModule, ]
12. })
13. export class PageLoginModule { }
```

/pages/page-register/page-register.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3. import { PageRegisterRoutingModule } from './page-register-routing.module';
4. import { PageRegisterComponent } from './page-register.component';
5. import { AuthModule } from 'src/app/features/auth/auth.module';
6.
7. @NgModule({
8.   declarations: [ PageRegisterComponent ],
9.   imports: [ CommonModule, PageRegisterRoutingModule, AuthModule, ],
10. })
11. export class PageRegisterModule { }
```

/pages/page-register/page-register.component.html

```
1. <p>page-register works!</p>
2. <app-register></app-register>
```

/pages/page-register/page-register.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { PageRegisterRoutingModule } from './page-register-routing.module';
import { PageRegisterComponent } from './page-register.component';
import { AuthModule } from 'src/app/features/auth/auth.module';

@NgModule({
  declarations: [ PageRegisterComponent ],
  imports: [ CommonModule, PageRegisterRoutingModule, AuthModule, ],
})
export class PageRegisterModule { }
```

/pages/page1/page1.component.html

```
<p>page1 works!</p>

<div id="container" *ngIf="isLogged$ | async else notlogged">
  <mat-card *ngFor="let item of products$ | async">
    <p><b>{{item.name}}</b></p>
    <p>prix : {{item.cost}}</p>
    <p>quantité : {{item.quantity}}</p>
  </mat-card>
</div>

<ng-template #notlogged >
  <p>Vous n'êtes pas connecté !</p>
</ng-template>
```

/pages/page1/page1.component.scss

```
#container {
  display: flex;
  justify-content: space-around;
}
```

/pages/page1/page1.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Observable } from 'rxjs';
import { AuthService } from 'src/app/core/auth/services/auth.service';
import { IProduct } from 'src/app/features/product/models/i-product';
import { ProductApiService } from 'src/app/features/product/services/product-api.service';

@Component({
  selector: 'app-page1',
  templateUrl: './page1.component.html',
  styleUrls: ['./page1.component.scss']
})
export class Page1Component implements OnInit {

  products$: Observable<IProduct[]>;    // $ : c'est juste pour indiquer que c'est un observable (pas obligatoire)
  isLogged$: Observable<boolean>;

  constructor(private productApi: ProductApiService, private authService: AuthService) {    }

  ngOnInit(): void {
    this.isLogged$ = this.authService.isLogged();
    this.products$ = this.productApi.get();
  }
}
```

/pages/page1/page1.module.ts

```
import { CommonModule } from '@angular/common';
import { Page1RoutingModule } from './page1-routing.module';
import { Page1Component } from './page1.component';
```

```
import { MaterialDesignModule } from 'src/app/shared/material-design/material-design.module';

@NgModule({
  declarations: [Page1Component],
  imports: [
    CommonModule,
     Page1RoutingModule,
     MaterialDesignModule,
   ]
})
export class Page1Module { }
```

/pages/page2/page2.component.html

```
<p>page2 works!</p>
<p>Accès à la page autorisé !</p>
```

/pages/partials/header/header.component.html

```
<div class="content">
  <p>header works!</p>

  <div id="container">
    <div *ngIf="(currentUser$ | async) as user">
      <p *ngIf="user.isLogged">Vous êtes connecté avec l'email : {{user.email}} ! <a href="#"
 (click)="logout()"><b>déconnexion</b></a></p>
      <p *ngIf="!user.isLogged">Vous n'êtes pas connecté !</p>

      <ul>
         <li><a [routerLink]="['/home']">home</a></li>
         <li><a [routerLink]="['/page1']">page 1 - les produits</a></li>
         <li><a [routerLink]="['/page2']">page 2</a></li>
         <li *ngIf="!user.isLogged"><a [routerLink]="['/login']">se connecter</a></li>
         <li *ngIf="!user.isLogged"><a [routerLink]="['/register']">s'inscrire</a></li>
       </ul>
    </div>

    <div>
      <p *appHasRole="[RoleEnum.ADMIN]">Vous avez le rôle ADMIN</p>
      <p *appHasRole="[RoleEnum.USER]">Vous avez le rôle USER</p>
      <p *appHasRole="[RoleEnum.USER, RoleEnum.ADMIN]">Vous avez le rôle USER et/ou ADMIN</p>
    </div>

  </div>
</div>
```

/pages/partials/header/header.component.scss

```
.content {
  background: #fcfcfc;
  padding-left: 24px;
}

#container {
  display: flex;
  justify-content: space-between;
}
```

/pages/partials/header/header.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Observable } from 'rxjs';
import { ICurrentUser } from 'src/app/core/auth/models/i-current-user';
import { AuthService } from 'src/app/core/auth/services/auth.service';
import { RoleEnum } from '../../../core/auth/enums/role-enum';

@Component({
```

```
8.   selector: 'app-header',
9.   templateUrl: './header.component.html',
10.   styleUrls: ['./header.component.scss']
11. })
12. export class HeaderComponent implements OnInit {
13.
14.   currentUser$: Observable<ICurrentUser>;
15.   RoleEnum: typeof RoleEnum = RoleEnum;                  // on récupère les énumerations des
 rôles pour le template
16.
17.   constructor(private auth: AuthService) { }
18.
19.   ngOnInit(): void {
20.     this.currentUser$ = this.auth.getCurrentUserObs();  // on récupère l'Observable au lieu
 du "subjet" car on ne doit rien émettre, juste écouter !
21.   }
22.
23.   logout() {
24.     this.auth.logout();
25.     return false;
26.   }
27. }
```

/pages/partials/partials.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3. import { HeaderComponent } from './header/header.component';
4. import { AppRoutingModule } from 'src/app/app-routing.module';
5. import { AuthModule } from 'src/app/features/auth/auth.module';
6. import { CoreModule } from '../../core/core.module';
7.
8. @NgModule({
9.   declarations: [HeaderComponent],
10.   imports: [
11.     CommonModule,
12.     AppRoutingModule,
13.     AuthModule,
14.     CoreModule,
15.   ],
16.   exports: [HeaderComponent],
17. })
18. export class PartialsModule { }
```

/pages/pages.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3. import { FeaturesModule } from '../features/features.module';
4. import { CoreModule } from '../core/core.module';
5. import { Page1Module } from './page1/page1.module';
6. import { Page2Module } from './page2/page2.module';
7. import { PageRegisterModule } from './page-register/page-register.module';
8. import { PartialsModule } from './partials/partials.module';
9. import { PageHomeModule } from './page-home/page-home.module';
10. import { AuthModule } from '../features/auth/auth.module';
11. import { PageLoginModule } from './page-login/page-login.module';
12.
13. @NgModule({
14.   declarations: [],
15.   imports: [
16.     CommonModule,
17.     FeaturesModule,
18.     CoreModule,
19.     Page1Module,
20.     Page2Module,
21.     PageLoginModule,
22.     PageRegisterModule,
23.     PartialsModule,
24.     PageHomeModule,
```

```
25.     AuthModule,
26.     PageLoginModule,
27.   ],
28.   exports: [PartialsModule],            // on exporte le module car il sera utilisé par le
 composant de démarrage : app.component.html
29. })
30. export class PagesModule { }
```

XXIX-C-2-d - /shared

/shared/material-design/material-design.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3. // Material
4. import { MatInputModule } from '@angular/material/input';
5. import { MatButtonModule } from '@angular/material/button';
6. import { NoopAnimationsModule } from '@angular/platform-browser/animations';
7. import { MatFormFieldModule } from '@angular/material/form-field';
8. import { MatCheckboxModule } from '@angular/material/checkbox';
9. import { MatCardModule } from '@angular/material/card';
10.
11. @NgModule({
12.   declarations: [],
13.   imports: [
14.     CommonModule,
15.     MatFormFieldModule,               // on importe uniquement les composants dont on a
 besoin
16.     NoopAnimationsModule,
17.     MatInputModule,
18.     MatButtonModule,
19.     MatCheckboxModule,
20.     MatCardModule,
21.   ],
22.   exports: [
23.     MatFormFieldModule,               // ne pas oublier d'exporter pour qu'il puisse être
 importé dans le module qui le demande
24.     NoopAnimationsModule,
25.     MatInputModule,
26.     MatButtonModule,
27.     MatCheckboxModule,
28.     MatCardModule,
29.   ]
30. })
31. export class MaterialDesignModule { }
```

XXIX-C-2-e - app

app-routing.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { LoggedGuard } from './core/auth/guards/logged.guard';
4. import { PageHomeComponent } from './pages/page-home/page-home.component';
5. import { PageLoginComponent } from './pages/page-login/page-login.component';
6. import { PageRegisterComponent } from './pages/page-register/page-register.component';
7. import { Page1Component } from './pages/page1/page1.component';
8. import { Page2Component } from './pages/page2/page2.component';
9.
10. const routes: Routes = [
11.   { path: 'home', component: PageHomeComponent },
12.   { path: 'page1', component: Page1Component },
13.   {
14.     path: 'page2',
15.     component: Page2Component,
16.     canActivate: [LoggedGuard]                        // on utilise le 'guard' de la route
 pour : /page2
```

```
    },
    { path: 'login', component: PageLoginComponent },
    { path: 'register', component: PageRegisterComponent },
    { path: '',   redirectTo: '/home', pathMatch: 'full' },
];

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

app.component.html

```
<app-header></app-header>
<hr>
<router-outlet></router-outlet>
```

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { PagesModule } from './pages/pages.module';
import { NoopAnimationsModule } from '@angular/platform-browser/animations';
import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/http';
import { JwtInterceptor } from './core/auth/interceptors/jwt.interceptor';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    PagesModule,
    NoopAnimationsModule,
    HttpClientModule,
  ],
  providers: [
    { provide: HTTP_INTERCEPTORS, useClass: JwtInterceptor, multi: true },    // On utilise
 l'interceptor pour intercepter les requêtes et lui ajouter le token
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

/environments/environment.prod.ts

```
export const environment = {
  production: true,
  urlApi: 'http://localhost:8000',
  endPointProducts: 'products',
  pathApi: 'api',
  pathAuth: 'auth',
};
```

/environments/environment.ts

```
export const environment = {
  production: false,
  urlApi: 'http://localhost:8000',
  endPointProducts: 'products',
  pathApi: 'api',
  pathAuth: 'auth',
};
```

XXIX-C-2-f - Configurer Docker dans l'application Angular

```
cd angular-auth-jwt1
```

.dockerignore

```
node_modules
```

package.json

```
...
  "scripts": {
    "ng": "ng",
    // windows 10 :
    "start-hr": "ng serve --host 0.0.0.0 --poll 500",       // --poll 500      regarde les changements dans le code toutes les 500ms
    // ou sur linux...
    // "start-hr": "ng serve --host 0.0.0.0",
...
```

XXIX-D - Le serveur : node.js du dossier : /node-api

XXIX-D-1 - Remarques

C'est un tutoriel sur Angular donc je ne m'étendrai pas sur des explications pour node.js et ni pour Docker.

XXIX-D-2 - Pratique

```
cd pack_auth1/node-api
```

.dockerignore

```
node_modules
```

package.json

```
{
  "name": "json-server-api",
  "version": "1.0.0",
  "description": "Simple Fake API",
  "main": "main.js",
  "scripts": {
    "start": "json-server --watch ./database.json",
    "start-auth": "node server.js"
  },
  "author": "ME:)",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.19.0",
    "json-server": "^0.14.2",
    "jsonwebtoken": "^8.1.0"
  }
}
```

server.js

```
const fs = require("fs");
const bodyParser = require("body-parser");
const jsonServer = require("json-server");
```

```
4. const jwt = require("jsonwebtoken");
5.
6. const server = jsonServer.create();
7. const userdb = JSON.parse(fs.readFileSync("./users.json", "UTF-8"));
8.
9. server.use(bodyParser.urlencoded({ extended: true }));
10. server.use(bodyParser.json());
11. server.use(jsonServer.defaults());
12.
13. const SECRET_KEY = "123456789";
14. const expiresIn = "1h";
15.
16. // Create a token from a payload
17. function createToken(payload) {
18.   return jwt.sign(payload, SECRET_KEY, { expiresIn });
19. }
20.
21. // Verify the token
22. function verifyToken(token) {
23.   return jwt.verify(token, SECRET_KEY, (err, decode) => decode !== undefined ? decode : err);
24. }
25.
26. // Check if the user exists in database
27. function isAuthenticated({ email, password }) {
28.   return (userdb.users.findIndex((user) => user.email === email &&
 user.password === password) !== -1);
29. }
30.
31. // Register New User
32. server.post("/auth/register", (req, res) => {
33.   console.log("register endpoint called; request body:");
34.   console.log(req.body);
35.   const { email, password } = req.body;
36.   const roles = ["user"];                          // le rôle admin pour tous les utlisateurs
 qui s'inscrivent
37.                                                    // c'est un tableau car un utilisateur peut
 avoir plusieurs rôles
38.
39.   if (isAuthenticated({ email, password }) === true) {
40.     const status = 401;
41.     const message = "Email and Password already exist";
42.     res.status(status).json({ status, message });
43.     return;
44.   }
45.
46.   fs.readFile("./users.json", (err, data) => {
47.     if (err) {
48.       const status = 401;
49.       const message = err;
50.       res.status(status).json({ status, message });
51.       return;
52.     }
53.
54.     // Get current users data
55.     var data = JSON.parse(data.toString());
56.
57.     // Get the id of last user
58.     var last_item_id = data.users[data.users.length - 1].id;
59.     userdb.users.push({ id: last_item_id + 1, email: email, password: password, roles:
 roles, });
60.
61.     //Add new user
62.     data.users.push({ id: last_item_id + 1, email: email, password: password, roles: roles,
  }); //add some data
63.   });
64.   // Create token for new user
65.
66.   token = createToken({ email, password });
67.   user = {
68.     'access_token': token,
69.     'roles': roles
70.   }
```

```
71.
72.   res.status(200).json(user);
73. });
74.
75. // Login to one of the users from ./users.json
76. server.post("/auth/login", (req, res) => {
77.   console.log("login endpoint called; request body:");
78.   console.log(req.body);
79.   const { email, password } = req.body;
80.   if (isAuthenticated({ email, password }) === false) {
81.     const status = 401;
82.     const message = "Incorrect email or password";
83.     res.status(status).json({ status, message });
84.     return;
85.   }
86.
87.   user = getUserdb(email);
88.   user.access_token = createToken({ email, password });
89.   console.log("/auth/login", user)
90.
91.   res.status(200).json(user);
92. });
93.
94. server.use(/^(?!\/auth).*$/, (req, res, next) => {
95.   if (
96.     req.headers.authorization === undefined ||
97.     req.headers.authorization.split(" ")[0] !== "Bearer"
98.   ) {
99.     const status = 401;
100.     const message = "Error in authorization format";
101.     res.status(status).json({ status, message });
102.     return;
103.   }
104.   try {
105.     let verifyTokenResult;
106.     verifyTokenResult = verifyToken(req.headers.authorization.split(" ")[1]);
107.
108.     if (verifyTokenResult instanceof Error) {
109.       const status = 401;
110.       const message = "Access token not provided";
111.       res.status(status).json({ status, message });
112.       return;
113.     }
114.     next();
115.   } catch (err) {
116.     const status = 401;
117.     const message = "Error access_token is revoked";
118.     res.status(status).json({ status, message });
119.   }
120. });
121.
122. server.get("/api/products", (req, res) =>
123.   res.json([
124.     { id: 1, name: "Product001", cost: 10, quantity: 1000 },
125.     { id: 2, name: "Product002", cost: 20, quantity: 2000 },
126.     { id: 3, name: "Product003", cost: 30, quantity: 3000 },
127.     { id: 4, name: "Product004", cost: 40, quantity: 4000 },
128.   ])
129. );
130.
131. server.listen(8000, () => {
132.   console.log("Run Auth API Server");
133. });
134.
135. function getUserdb(email) {
136.   return (userdb.users.find((user) => user.email === email));
137. }
138.
139. function getRolesFromUserdb(email) {
140.   return getUserdb(email).roles;
141. }
```

users.json

```
{"users":[{"id":1,"email":"bruno@email.com","password":"bruno123"}]}
```

XXIX-D-3 - Remarques

Sachez que le fichier server.js est une version simple juste pour faire tourner l'application.
Sur Internet, on peut trouver des codes sources respectant le standard des bonnes pratiques.

Vous avez vu qu'on peut faire du back JavaScript via node.js. Express est juste une surcouche à node.js pour écrire moins de code et plus facilement. Sachez qu'il existe un framework basé sur node.js, Express et Angular pour écrire du back comme ici encore plus facilement et surtout avec une code mieux structuré comme l'est Angular, ce framework s'appelle NestJS.

XXIX-E - Docker : gestion de l'application et du serveur : node.js

docker-compose.yml est utilisé pour lancer plusieurs containers basés sur des images Docker.
En effet, nous avons besoin d'un container pour l'application Angular et un container pour le serveur node.js.

```
cd pack_auth1
```

docker-compose.yml

```
version: "3.7"
services:
  node-api:
    build:
      context: .
      dockerfile: Dockerfile.node-api
    image: pack_auth1/node-api:latest
    volumes:
      - ./node-api/src:/root/node-api/src
    ports:
      - 8000:8000
    restart: always
    container_name: node-api
  ng-app:
    build:
      context: .
      dockerfile: Dockerfile.ng-app
    image: pack_auth1/angular-auth-jwt1:latest
    volumes:
      - ./angular-auth-jwt1/src:/root/angular-auth-jwt1/src
    ports:
      - "5600:4200"
      - "49153:49153"
    restart: always
    container_name: angular-auth-jwt1
```

Dockerfile.ng-app

```
FROM node:12-alpine
WORKDIR /root/
COPY ./angular-auth-jwt1 /root/angular-auth-jwt1
WORKDIR /root/angular-auth-jwt1
RUN npm install
EXPOSE 5600 49153
CMD npm run start-hr
```

Dockerfile.node-api

```
1. FROM node:12-alpine
2. WORKDIR /root/
3. COPY ./node-api /root/node-api/
4. WORKDIR /root/node-api/
5. RUN npm install
6. CMD npm run start-auth
```

XXIX-F - Lancement avec Docker

```
1. cd pack_auth1
```

```
1. docker-compose build        // à faire une fois et à chaque changement d'un fichier du
 dossier : /node-api
2. docker-compose up           // lance le Docker de l'application et du serveur node.js
```

http://localhost:5600
// l'application est accessible sur le port : 5600 du container Docker
// en mode développement, car c'est un ng serve dans l'image (voir start-hr dans le fichier package.json de l'application)
// et avec live-reload

XXIX-F-1 - Remarques

Pour une version en production, il faut une image avec un serveur nginx par exemple, car seul un serveur http a la fiabilité et la performance pour fournir des fichiers.

XXIX-G - Lancement sans Docker

```
1. cd pack_auth1/angular-auth-jwt1
2. ng serve -o
3.
4. cd pack_auth1/node-api
5. node server.js
```

http://localhost:4200

XXX - Étude de cas n°2 : Angular + NestJS : authentification + accès sécurisé à une API

On va reprendre le même projet que l'étude de cas n°1 mais au lieu d'utiliser Node.js pour le back, on va utiliser le framework NestJS.

XXX-A - Qu'est ce que NestJS ?

- NestJS est un cadre pour construire des applications NodeJS côté serveur ;
- est basé sur TypeScript, NodeJS et Express ;
- avantages :
- NestJs est une abstraction de NodeJS donc nous pouvons utiliser les bibliothèques NodeJS ;
- propose une architecture modèle / controlleur, ainsi nous avons une bonne organisation du code ;
- dispose de commandes Nest CLI ;

XXX-B - Pratique

Cette fois je vais vous fournir le git afin que vous puissiez récuperer entièrement le projet.

```
1. git clone https://github.com/vaka440/pack-auth3.git
```

Il y a 4 container Docker correspondant à :

- l'application Angular : **http://localhost:5600**
- le swagger de l'api NestJS : **http://localhost:3000/swagger-api/**
- adminer, le phpmyadmin pour postgre : **http://localhost:5050** user: admin@admin.com password: root
- la base de donnée postgre qui fonctionne en interne

```
cd pack-auth3
docker-compose up -d --build
```

XXX-C - Description rapide du back avec NestJS

XXX-C-1 - Fonctionnalités

- serveur d'authentification : login, register, refreshtoken (refaire une demande d'un token lorsque celui en cours a expiré)
- serveur de données : /api/products
- sécurité sur les données : token valide ? accès aux données par le rôle ("USER", "ADMIN") ?
- validation des données sur les requêtes entrantes : lorsque le serveur reçoit par exemple une requête de connexion, le serveur vérifie que les données json de la requête dans le body respectent certains critères définis dans le DTO. Par exemple, si on indique que le mot de passe doit faire 6 caractères minimums, cela va vérifier que le password respecte bien la contrainte sinon un message d'erreur explicatif du rejet est envoyé à l'application "le password doit faire 6 caractères minimum". On effectue une validation de 1er niveau afin d'éviter que ce soit le serveur de données ici postgre qui rejette la requête SQL car le password ne respecte pas la contrainte, on gagne donc en performances.

XXX-C-2 - Les différents décorateurs dans les controlleurs

XXX-C-2-a - Exemple

- afin d'avoir une bonne organisation du code, on créé des contrôleurs par thème et en fonction des urls : UserController, MessageController, CategoryController....
- un controlleur dispose du minimum nécessaire pour fonctionner afin de ne pas surcharger en code ;
- on utilise des décorateurs afin de réduire au maximum le code.

```
@ApiTags('mes-messages')
@Controller('mes-messages')
export class MessageController {

  @Get('message1')
  getMessage1() {
      return   JSON.stringify([
        message: "voici le message 1"
      ]);
  }

  @Get('message2')
  getMessage2() {
      return   JSON.stringify([
        message: "voici le message 2"
      ]);
  }
}
```

Accès aux urls :

- /mes-messages/message1
- /mes-messages/message2

XXX-C-2-b - auth.controller.ts, user.controller.ts, product.controller.ts :

XXX-C-2-b-i - Au niveau de la classe :

```
@ApiTags('user')
@Controller('user')
export class UserController {
    ...
```

XXX-C-2-b-ii - Au niveau des fonctions de la classe :

Suivants les besoins, on utilise ou pas les décorateurs : UseGuards, Roles...

```
  @Get('products')                    chemin url
  @Get('id/:id')
  @UseGuards(JWTGuard)                accès sécurisé sur un token valide
  @Roles(Role.Admin)                  accès sécurisé sur un rôle précis
  @UseGuards(JWTGuard, RolesGuard)    les 2

  ... fonction ...
```

Créer son propre décorateur : @User() (voir /user/user.decorator.ts)

```
...
  getMe(@User() user: RequestWithUser) {
    return user;
  }
...
```

XXX-C-3 - URLs

XXX-C-3-a - /auth/auth.controller.ts

```
/auth/login         anonyme
/auth/register      anonyme
/auth/refresh       anonyme
```

XXX-C-3-b - /user/user.controller.ts

```
/user/me            jwt
/user/all           jwt     Role.Admin
/user/create        jwt     Role.Admin
/user/delete        jwt     Role.Admin
/user/id/:id        jwt     Role.Admin
/user/all/:skip     jwt     Role.Admin
```

XXX-C-3-c - /api/product.controller.ts

```
api/products        jwt
```

XXX-C-4 - Exemples d'accès aux urls

- via l'application Angular ou pour tester avec postman ou curl ;
- sachez que lors de l'enregistrement d'un utilisateur, il obtient le rôle : "USER".

XXX-C-4-a - login

```
1. POST http://localhost:3000/auth/login
2. Content-Type application/json
3. Body raw json {"email": "toto1@toto.fr", "password": "tototo"}
```

XXX-C-4-b - register

```
1. POST http://localhost:3000/auth/register
2. Content-Type application/json
3. Body raw json {"email": "toto1@toto.fr", "password": "tototo"}
```

XXX-C-4-c - refresh token

```
1. POST http://localhost:3000/auth/register
2. Content-Type application/json
3. Body raw json { refresh_token: "................................."}
```

XXX-C-4-d - Obtenir la liste de tous les utilisateurs

- être authentifié (jeton)
- avoir le rôle "ADMIN" (attention: tous les utilisateurs enregistrés ont uniquement le rôle "USER")

```
1. GET http://localhost:3000/user/all
2. Content-Type application/json
3. Authorization  Bearer ......................
```

XXX-C-5 - Validation

Il y a une validation de 1er niveau sur les données reçus :

- /validations/validation-filter.ts
- /validations/validation-exception.ts
- les DTO : /auth/request.ts

main.ts

```
1. ...
2.   app.useGlobalFilters(
3.     new ValidationFilter()
4.   );
5. ...
```

/auth/request.ts

```
1. ...
2.   @IsNotEmpty({ message: 'A email is required' })       // contraintes obligatoire
3.   @MinLength(6, { message: 'Your email must be at least 6 characters' })   // 6 caractères
 minimum
4.   @IsEmail({}, { message: 'Your email is invalid' })         // mauvais format de l'email
5.   readonly email: string             // les 3 décorateurs précédents sont appliqués au champ
 email
6. ...
```

Par exemple pour le `login` :

```
1. POST http://localhost:3000/auth/login
```

```
2. Content-Type application/json
3. Body raw json {"email": "toto1@toto.fr", "password": "tototo"}
```

- si on met 2 caractères à password {"email": "toto1@toto.fr", "password": "to"} au lieu de 6 alors la validation bloque la requête et reponds une erreur (avec description de l'erreur) et ceci sans accèder à la base de donnée.
- utilisation de la classe validation-filter.ts qui gère les validations
- et les décorateurs comme : @IsNotEmpty ou @MinLength du fichier : requests.ts
- remarque 1 : le fichier requests.ts est en quelque sorte un DTO
- remarque 2 : le fichier : user.dto.ts ne possède pas de validation, on aurait pu en mettre.

XXX-C-6 - Configuration

XXX-C-6-a - jwt

.env

```
1. JWT_SECRET=pd5s378ee9zs4f5g
2. EXPIRES_IN=6s
```

- EXPIRES_IN ---------> le temps d'expiration du token
- EXPIRES_IN ---------> j'ai mis 6 secondes pour l'exemple, la valeur normale serait de quelques heures

XXX-C-6-b - La base de donnée

renommer : ormconfig.json en ormconfig.json.back
car on utilise le .env

.env

```
1. JWT_SECRET=pd5s378ee9zs4f5g
2. EXPIRES_IN=6s
3.
4. DB_PORT=5432
5. DB_HOST=db
6. DB_USER=root
7. DB_PASSWORD=root
8. DB_NAME=test
```

app.module.ts

```
1. ...
2. ...
3. import { ConfigModule } from '@nestjs/config';
4.
5. @Module({
6.   imports: [
7.     ConfigModule.forRoot(),
8.     TypeOrmModule.forRootAsync({
9.       useFactory: () => ({
10.         type: 'postgres',
11.         host: process.env.DB_HOST,                          // DB_HOST=db    "db" est le nom
 du container Docker
12.         port: parseInt(process.env.DB_PORT),
13.         username: process.env.DB_USER,
14.         password: process.env.DB_PASSWORD,
15.         database: process.env.DB_NAME,
16.         entities: [UserEntity, RefreshTokenEntity],         // ici, ne pas oublier d'indiquer
 les entités que la base doit gérer
17.         keepConnectionAlive: true,
```

```
18.         synchronize: true,
19.       })
20.     }),
21.     ...
22.     ...
```

- une entité comme : /models/user.model.ts est une représentation objet d'une table dans la base de donnée.
- dans le code, on manipule des entités

XXX-C-7 - swagger

main.ts

```
1. ...
2. ...
3.   SwaggerModule.setup('swagger-api', app, document);        // "swagger-api"  -> path, à
 modifier
4. ...
```

Accès à swagger :

http://localhost:3000/swagger-api/

XXX-C-8 - CORS

main.ts

```
1. ...
2.   app.enableCors({
3.     origin: ['http://localhost:5600'],                     // ne pas oublier ici de mettre
 les urls des applications qui sont autorisés à accéder au serveur
4.   });
5. ...
```

XXX-C-9 -

XXX-C-10 - Les modules

Exemple avec AuthModule:

```
1. @Global()
2. @Module({
3.   providers: [AuthService, RefreshTokensService, TokensService, ],    // on déclare ici les
 services qui seront gérer par l'injection de dépendance (DI)
4.
5.   exports: [],                                                       // on peut exporter un
 controleur ou un module afin qu'il soit importable ailleurs
6.
7.   imports: [                                                        // on importe des
 packages avec éventuellement une configuration
8.     TypeOrmModule.forFeature([UserEntity, RefreshTokenEntity]),      // on déclare ici les
 entités qui doivent être gérer par l'ORM
9.
10.     JwtModule.register({
11.       secret: process.env.JWT_SECRET,
12.       signOptions: { expiresIn: process.env.EXPIRES_IN },
13.     }),
14.
15.     UserModule,                                                   // ce module aura besoin
 du module : UserModule
```

```
16.                                                          // pour la connexion et
 l'enregistrement d'un utilisateur
17.   ],
18.   controllers: [AuthController,],                        // déclarer ici les
 contrôleurs qui doivent être prises en compte
19. })
20. export class AuthModule {}
```

XXX-D - Résultat

- j'ai mis un délais d'expiration très court (de 6 secondes) afin que vous constatez le fonctionnement du refreshToken
- pour tester, faites ceci :
- afficher sur le coté l'outil de dév du navigateur réseau
- inscrivez vous, connectez vous
- allez rapidement sur la page 1 - les produits
- les produits sont affichés
- cliquez sur la home page
- attendez 6 secondes
- reclic sur la page 1 - les produits
- les produits sont affichés mais vous avez vu dans l'onglet Réseau qu'une demande de refreshtoken a été émise, donc Angular à reçu un nouveau token valide et a ainsi pu récuperer les produits ;
- pour voir comment fait angular pour refaire une demande d'un nouveau token, allez voir le code du fichier : /angular-auth-jwt1/src/app/auth/interceptors/jwt.interceptor.ts

XXXI - Angular Universal (SSR)

Angular Universal est une solution de pré rendu pour Angular.
Il s'exécute sur le serveur, générant des pages d'application statiques qui seront ensuite amorcées sur le navigateur en tant qu'application.

Le robot de Google arrive à parser le code JavaScript pour le référencement, ce n'est pas le cas des autres moteurs de recherche comme bing…
Pour remédier à cela, on utilise Angular Universal (SSR) afin que des pages statiques soient rendues pour le référencement.

XXXI-A - À savoir

Voici les étapes du SSR :

- (1) le navigateur récupère le HTML et le CSS rendus et affiche l'application « statique » --> pour le référencement.
- (2) le navigateur affiche la page « statique » --> rapidité d'affichage sur la 1re page (car à ce niveau, l'application n'est pas encore téléchargée).
- (3) le navigateur récupère, analyse, interprète et exécute JavaScript --> pour faire tourner l'application.
- (4) l'application Angular est amorcée, remplaçant l'ensemble de l'arborescence DOM par la nouvelle application « en cours d'exécution ».
- (5) l'application est initialisée, récupérant souvent des données à partir d'un serveur distant ou d'une API.
- (6) l'utilisateur interagit avec l'application.

XXXI-B - Pratique

```
1. ng new angular-ssr1
2.  strict ? NO
3.     routing ? YES
4.     SCSS
```

```
ng g c pages/page1 --module=app
ng g c pages/page2 --module=app
```

app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
//
import { Page1Component } from 'src/app/pages/page1/page1.component';
import { Page2Component } from 'src/app/pages/page2/page2.component';

const routes: Routes = [
  { path: 'page1', component: Page1Component },
  { path: 'page2', component: Page2Component },
   { path: '', redirectTo: '/page1', pathMatch: 'full' },
];

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

app.component.html

```
<ul>
  <li><a [routerLink]="['/page1']">aller à page1</a></li>
  <li><a [routerLink]="['/page2']">aller à page2</a></li>
</ul>

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

XXXI-C - Résultat

```
ng serve
```

http://localhost:4200

Affichez le code source de la page et constatez que la balise app-root est vide comme ici <app-root></app-root>

XXXI-D - Pratique : installation d'Angular Universal

La commande suivante va effectuer des ajouts et des modifications afin de mettre en place Angular Universal.

```
ng add @nguniversal/express-engine
```

Les modifications sont les suivantes :

```
src/
  index.html                app web page
  main.ts                   bootstrapper pour le client
  main.server.ts            * bootstrapper pour le serveur
  style.css                 styles
  app/ ...
    app.server.module.ts    * le module coté serveur
server.ts                   * Node.js express
tsconfig.json               TypeScript configuration
tsconfig.app.json            TypeScript browser application configuration
tsconfig.server.json         TypeScript server application configuration
tsconfig.spec.json           TypeScript tests configuration
```

XXXI-D-1 - À savoir

Il y a donc deux parties : le client (main.ts) et le serveur (main.server.ts).
Le navigateur reçoit du serveur Node.js express la partie serveur avec les pages statiques (le balisage pour le SEO) ensuite la partie client prend le relais pour faire tourner l'ensemble en tant qu'application Angular.

XXXI-E - Pratique : compilation et exécution

Il faut toujours compiler avant exécution :

```
1. npm run build:ssr
```

Lancer l'application :

```
1. npm run serve:ssr
```

Allons voir à quoi correspond serve:ssr dans le package.json :

package.json

```
1. ...
2.   "serve:ssr": "node dist/angular-ssr4/server/main.js",
```

Le serveur Node.js est lancé pour servir les pages statiques.

XXXI-E-1 - Résultat

http://localhost:4000

Affichez le code source de la page et constatez qu'il y a du contenu dans la balise <app-root>.....</app-root>
Les moteurs de recherche autre que Google (qui n'a pas besoin, car il sait lire le JavaScript) vont pouvoir référencer les pages.

XXXI-F - En production

Sur un serveur Node.js, vous envoyez le contenu du dossier /dist et exécutez la commande node dist/angular-ssr4/server/main.js ou npm run serve:ssr
Donc avec Angular Universal, vous devez obligatoirement utiliser un serveur Node.js et donc vous ne pouvez pas servir l'application comme auparavant avec un serveur nginx ou apache.

XXXI-G - Performance à l'affichage de la première page

Comme vous l'avez compris, le navigateur reçoit la page statique du serveur Node.js express.
Ce qui a pour conséquence une performance à l'affichage de la première page.
Ensuite l'application Angular prend le contrôle.

XXXI-H - Transfert d'état de rendu côté serveur pour les requêtes HTTP

À propos du SSR, on a vu que la première étape était la construction du rendu côté serveur pour être envoyé au navigateur pour affichage, ensuite, que l'application était envoyée au navigateur pour la prise de contrôle par Angular.

Sachez qu'il y a un cas où il y a un petit souci entre le SSR et le monde JavaScript c'est celui d'effectuer des requêtes API, rien d'important, mais on perd quelques millisecondes.
En effet, si la page effectue une requête API, pour construire le rendu de la première étape il lance cette requête pour récupérer les données.
Quelques millisecondes plus tard, le navigateur reçoit l'application et s'initialise, à l'initialisation du composant page, il effectue la même requête API.
Comme on est des perfectionnistes, on trouve ça plutôt dérangeant d'exécuter 2 fois la même requête api, c'est une perte de temps quand on veut des performances.

La solution nommée State Transfer consiste à mettre en cache les requêtes API et le résultat lors du premier appel côté serveur et ensuite lors du deuxième appel côté application le State Transfer nous fournit les requêtes API mises en cache.
Ce mécanisme est transparent et fonctionne globalement pour toutes les requêtes communes entre le rendu serveur et l'application navigateur.

XXXI-H-1 - Pratique

Nous avons besoin de lancer une requête API donc nous allons le faire en page 1.

Voici les ajouts et modifications à faire :

/page1/page1.component.html

```
<p>page1 works!</p>

{{todos$ | async |json}}
```

/page1/page1.component.ts

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

@Component({
  selector: 'app-page1',
  templateUrl: './page1.component.html',
  styleUrls: ['./page1.component.scss']
})
export class Page1Component implements OnInit {
  todos$;

  constructor(private http: HttpClient) { }

  ngOnInit(): void {
    // pour simplifier la démonstration je mets l'appel à l'API ici dans le composant (au lieu de le mettre dans un service)
    this.todos$ = this.http.get(`https://jsonplaceholder.typicode.com/todos`);
  }
}
```

/transfer-state/BrowserStateInterceptor.ts

```
import { HttpEvent, HttpHandler, HttpInterceptor, HttpRequest, HttpResponse } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { makeStateKey, TransferState } from '@angular/platform-browser';
import { Observable, of } from 'rxjs';

@Injectable({
    providedIn: 'root'
})
export class BrowserStateInterceptor implements HttpInterceptor {

```

```
    constructor(
        private transferState: TransferState,
    ) { }

    intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
        if (req.method === 'GET') {
            const key = makeStateKey(req.url);
            const storedResponse: string = this.transferState.get(key, null);
            if (storedResponse) {
                const response = new HttpResponse({ body: storedResponse, status: 200 });
                return of(response);
            }
        }

        return next.handle(req);
    }
}
```

app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule, BrowserTransferStateModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { Page1Component } from './pages/page1/page1.component';
import { Page2Component } from './pages/page2/page2.component';
import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/http';
import { BrowserStateInterceptor } from './transfer-state/BrowserStateInterceptor';
import { TransferHttpCacheModule } from '@nguniversal/common';

@NgModule({
  declarations: [
    AppComponent,
    Page1Component,
    Page2Component
  ],
  imports: [
    BrowserModule.withServerTransition({ appId: 'serverApp' }),
    AppRoutingModule,
    HttpClientModule,                                   // pour pouvoir lancer des requêtes API : get, post...
    //
    TransferHttpCacheModule,                            // un module de gestion de cache
    BrowserTransferStateModule,                         // notre classe pour gérer le transfer state coté application

  ],
  providers: [
    {                                                   //
      provide: HTTP_INTERCEPTORS,                       // Intercepte les requêtes de l'application
      useClass: BrowserStateInterceptor,                // gestion des requêtes mise en cache
      multi: true                                       //
    },
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

/transfer-state/ServerStateInterceptor.ts

```
import { HttpHandler, HttpInterceptor, HttpRequest, HttpResponse } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { makeStateKey, TransferState } from '@angular/platform-browser';
import { tap } from 'rxjs/operators';

@Injectable()
export class ServerStateInterceptor implements HttpInterceptor {

```

```
    constructor(private transferState: TransferState) { }

    intercept(req: HttpRequest<any>, next: HttpHandler) {
        return next.handle(req).pipe(
            tap(event => {
                if ((event instanceof HttpResponse && (event.status === 200 ||
 event.status === 202))) {
                    this.transferState.set(makeStateKey(req.url), event.body);
                }
            }),
        );
    }
}
```

app.server.module.ts

```
import { NgModule } from '@angular/core';
import { ServerModule, ServerTransferStateModule } from '@angular/platform-server';
import { HTTP_INTERCEPTORS } from '@angular/common/http';
import { ServerStateInterceptor } from './transfer-state/ServerStateInterceptor';
import { AppModule } from './app.module';
import { AppComponent } from './app.component';

@NgModule({
  imports: [
    AppModule,
    ServerModule,
    ServerTransferStateModule,
  ],
  bootstrap: [AppComponent],
  providers: [
    {
        provide: HTTP_INTERCEPTORS,           // Intercepte les requêtes côté serveur
        useClass: ServerStateInterceptor,     // gestion des requêtes interceptées
        multi: true
    }
  ],
})
export class AppServerModule {}
```

app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
//
import { Page1Component } from 'src/app/pages/page1/page1.component';
import { Page2Component } from 'src/app/pages/page2/page2.component';

const routes: Routes = [
  { path: 'page1', component: Page1Component },
  { path: 'page2', component: Page2Component },
   { path: '',   redirectTo: '/page1', pathMatch: 'full' },
];

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

XXXI-H-1-a - Résultat

```
npm run build:ssr
npm run serve:ssr
```

Dans les outils de développement du navigateur et l'onglet réseau, vous pouvez constater que le GET de la requête API n'est pas fait parce qu'il a été intercepté et les données récupérées du cache.

XXXI-I - Conclusion

- Angular Universal fournit le code pour le référencement SEO et permet un gain de performance dû à l'affichage du rendu de la première page.
- pour une optimisation des performances, il faut mettre en place le Transfer State pour gérer les requêtes API communes côtés serveur et navigateur.

XXXII - Gestion dynamique des metas tags pour le SEO

Les balises metas décrivent des détails sur le contenu de votre page aux moteurs de recherche.
Le service meta d'Angular facilite l'obtention ou la définition de différentes balises métas en fonction de l'itinéraire actif actuel dans votre application.

Voici un exemple pour une page HTML classique :

```
<head>
  <meta charset="UTF-8">
  <meta name="description" content="Free Web tutorials">
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <meta name="author" content="John Doe">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
```

Il existe aussi des metas tags pour les médias sociaux en général (facebook, Twitter…) :

```
<meta property="og:title" content="European Travel Destinations">
<meta property="og:description" content="Offering tour packages for individuals or groups.">
<meta property="og:image" content="http://euro-travel-example.com/thumbnail.jpg">
<meta property="og:url" content="http://euro-travel-example.com/index.htm">
<meta name="twitter:card" content="summary_large_image">
```

Ou précisément pour Twitter :

```
<meta name="twitter:title" content="European Travel Destinations ">
<meta name="twitter:description" content=" Offering tour packages for individuals or groups.">
<meta name="twitter:image" content=" http://euro-travel-example.com/thumbnail.jpg">
<meta name="twitter:card" content="summary_large_image">
```

XXXII-A - Pratique

```
ng new angular-meta1
 strict ?  NO
    routing ? YES
    SCSS

ng g c pages/page1 --module=app
ng g c pages/page2 --module=app
```

app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
//
import { Page1Component } from 'src/app/pages/page1/page1.component';
import { Page2Component } from 'src/app/pages/page2/page2.component';

const routes: Routes = [
  { path: 'page1', component: Page1Component },
  { path: 'page2', component: Page2Component },
   { path: '', redirectTo: '/page1', pathMatch: 'full' },
];

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

app.component.html

```
<ul>
  <li><a [routerLink]="['/page1']">aller à page1</a></li>
  <li><a [routerLink]="['/page2']">aller à page2</a></li>
</ul>

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

Il faut installer Angular Universal pour pouvoir modifier dynamiquement les metas tags :

```
ng add @nguniversal/express-engine
```

XXXII-B - À savoir,

voici la liste des actions que l'on peut faire dynamiquement sur les tags :

```
class Meta {
  addTag(tag: MetaDefinition, forceCreation: boolean = false): HTMLMetaElement | null
  addTags(tags: MetaDefinition[], forceCreation: boolean = false): HTMLMetaElement[]
  getTag(attrSelector: string): HTMLMetaElement | null
  getTags(attrSelector: string): HTMLMetaElement[]
  updateTag(tag: MetaDefinition, selector?: string): HTMLMetaElement | null
  removeTag(attrSelector: string): void
  removeTagElement(meta: HTMLMetaElement): void
}
```

XXXII-C - Pratique

XXXII-C-1 - Exemple 1 : des tags dans le composant de démarrage app.component

On va ajouter tous les tags (title, description…) au composant app.component.
Ainsi, toutes les pages disposeront de ces tags.

app.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Title, Meta } from '@angular/platform-browser';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
```

```
  styleUrls: ['./app.component.scss']
})
export class AppComponent implements OnInit{


  constructor(private titleService: Title, private metaTagService: Meta) {}

  ngOnInit() {

    this.titleService.setTitle('Le titre de la page');

    this.metaTagService.addTags([
      { name: 'keywords', content: 'angular, meta' },
      { name: 'description', content: "La description de votre page telle qu'elle devrait apparaître dans les résultats de recherche Google" },
      { name: 'robots', content: 'index, follow' },
      { name: 'author', content: 'mc guyver' },
      { name: 'viewport', content: 'width=device-width, initial-scale=1' },
      { name: 'date', content: '2021-02-03', scheme: 'YYYY-MM-DD' },
      { charset: 'UTF-8' }
    ]);


    // médias sociaux : facebook - twitter
    this.metaTagService.addTag({ name: 'og:title', content: "Le titre de votre page tel qu'il devrait apparaître sur facebook" });
    this.metaTagService.addTag({ name: 'og:description', content: "app: description - La description de votre page telle qu'elle devrait apparaître dans les résultats de recherche facebook" });
    this.metaTagService.addTag({ name: 'og:image', content: "Une URL d'image qui doit représenter votre page"});
    this.metaTagService.addTag({ name: 'og:url', content: "L'URL canonique de votre page"});
    this.metaTagService.addTag({ name: 'og:type', content: "Le type de votre page" });
    this.metaTagService.addTag({ name: 'twitter:card', content: "résumé de l'image" });

    this.metaTagService.addTag({ name: 'og:site_name', content: "le nom qui doit être affiché pour l'ensemble du site" });
    this.metaTagService.addTag({ name: 'twitter:image:alt', content: 'altDesc' })
  }
}
```

XXXII-C-1-a - Résultat

```
npm run build:ssr
npm run serve:ssr
```

http://localhost:4000/

Allez sur la page 1 et ensuite sur la page 2.
Affichez le code source des deux pages et constatez la présence des tags (keywords, description…) dans le HEAD

XXXII-C-2 - Exemple 2 : modifier le tag description sur la page 2

L'exemple 1 a initialisé toute une série de tags, le but dans cet exemple est de modifier le tag description de la page 2.

page2.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Title, Meta } from '@angular/platform-browser';

@Component({
  selector: 'app-page2',
  templateUrl: './page2.component.html',
  styleUrls: ['./page2.component.scss']
})
```

```
9. export class Page2Component implements OnInit {
10.
11.   constructor(private titleService: Title, private metaService: Meta) {}
12.
13.   ngOnInit() {
14.     this.metaService.updateTag({ name: 'description', content: 'page 2 : description -
 updated' });
15.   }
16. }
```

```
1. npm run build:ssr
2. npm run serve:ssr
```

http://localhost:4000/

XXXII-C-2-a - Résultat

Allez sur la page 1 et regardez le tag description (qui hérite de app.component), vous devriez avoir app: description
Allez sur la page 2 et regardez le tag description, vous devriez avoir 'page 2 : description - updated'

XXXII-D - Conclusion

Vous pouvez ajouter ou modifier dynamiquement des tags avec Angular Universal (SSR) en fonction des pages.
C'est du SSR donc en production ça fonctionne avec un serveur Node.js.

XXXII-E - Remarques

- sans SSR, on peut ajouter manuellement des tags de façon fixe sur le fichier : /src/index.html
- attention : si vous ajoutez plusieurs fois un tag description par exemple, il y en aura plusieurs dans le HEAD. Donc selon votre stratégie, faites attention avec les addTag et updateTag dans vos pages.

XXXIII - Les Progressive Web App (PWA)

Une progressive web application (PWA) offre un haut niveau d'expérience utilisateur, car elle possède les mêmes fonctionnalités que les applications natives.
PWA ne nécessite pas d'être déployé via les magasins d'applications, nous les déployons à partir de serveurs web via des URL.

Les principaux avantages :

- fonctionne sur presque tous les ordinateurs de bureau, mobiles ou tablettes ;
- est toujours à jour à l'aide des services worker ;
- fonctionne en hors ligne ou sur des réseaux instables ;
- est facilement installable.

XXXIII-A - Pratique

Nous allons créer une application Angular et nous la configurerons en mode PWA.

```
1. ng new angular-pwa
2.  strict ?  NO
3.     routing ? YES
4.     SCSS
5.
6. ng g c pages/page1 --module=app
7. ng g c pages/page2 --module=app
```

app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
//
import { Page1Component } from 'src/app/pages/page1/page1.component';
import { Page2Component } from 'src/app/pages/page2/page2.component';

const routes: Routes = [
  { path: 'page1', component: Page1Component },
  { path: 'page2', component: Page2Component },
   { path: '', redirectTo: '/page1', pathMatch: 'full' },
];

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

app.component.html

```
<ul>
  <li><a [routerLink]="['/page1']">aller à page1</a></li>
  <li><a [routerLink]="['/page2']">aller à page2</a></li>
</ul>

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

page1.component.html

```
<p>page1 works!</p>

{{product$ | async |json}}
```

page1.component.ts

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

@Component({
  selector: 'app-page1',
  templateUrl: './page1.component.html',
  styleUrls: ['./page1.component.scss']
})
export class Page1Component implements OnInit {
  product$;

  constructor(private http: HttpClient) { }

  ngOnInit(): void {
    // pour simplifier la démonstration je mets l'appel à l'API ici dans le composant (au
 lieu de le mettre dans un service)
    this.product$ = this.http.get(`https://reqres.in/api/products/3`);
  }
}
```

Installation du package et ajouts des fichiers nécessaires au fonctionnement du PWA :

```
ng add @angular/pwa
```

La commande ci-dessus ajoute automatiquement les fichiers suivants :

```
/app
    /src
        manifest.webmanifest                        (qui doit être utilisé par le fichier
 index.html)
        /assets
            /icons
                ...                                 (des icônes de différentes tailles)
 (l'icône sur le bureau)
                ...
    ngsw-config.json                                (le service worker)  (qui doit être
 utilisé par le module app.module.ts)
```

Et effectue une mise à jour sur le fichier index.html :

index.html

```
...
<link rel="manifest" href="manifest.webmanifest">
<meta name="theme-color" content="#1976d2">
...
```

Et modifie le module de démarrage app.module.ts pour lui ajouter le gestionnaire de service worker :

app.module.ts

```
imports: [
    ...
    ServiceWorkerModule.register('ngsw-worker.js', { enabled: environment.production })
    ...
```

Qu'est-ce qu'un service worker ?

Il permet une intégration approfondie de la plateforme, telle que la prise en charge hors ligne, la synchronisation en arrière-plan, la mise en cache riche et les notifications push.

Il faut indiquer au service Worker quelles ressources doivent être mises en cache.
Pour cela, ajoutez la partie dataGroups dans le fichier ngsw-config.json :

ngsw-config.json

```
{
  "$schema": "./node_modules/@angular/service-worker/config/schema.json",
  "index": "/index.html",
  "assetGroups": [
    {
      "name": "app",
      "installMode": "prefetch",
      "resources": {
        "files": [
          "/favicon.ico",
          "/index.html",
          "/manifest.webmanifest",
          "/*.css",
          "/*.js"
        ]
      }
    },
    {
      "name": "assets",
      "installMode": "lazy",
      "updateMode": "prefetch",
      "resources": {
        "files": [
          "/assets/**",
```

```
          "/*.(eot|svg|cur|jpg|png|webp|gif|otf|ttf|woff|woff2|ani)"
        ]
      }
    }
  ],

  "dataGroups": [
    {
      "name": "api-performance",
      "urls": [
        "/assets/i18n/**",
        "/api/**"
      ],
      "cacheConfig": {
        "strategy": "performance",
        "maxSize": 100,
        "maxAge": "3d"
      }
    }
  ]
}
```

Dans urls on indique les assets et les URL externes :

- si nécessaire, on peut indiquer le dossier des images à mettre en cache.
- on a ajouté /api/** pour mettre en cache les résultats des requêtes API.

XXXIII-A-1 - Résultat

On peut voir le résultat avec la version production, pour cela on installe en global un petit serveur http pour lancer l'application qui se trouve dans /dist.

```
npm install -g http-server
```

```
cd angular-pwa
ng build --prod

cd dist/angular-pwa
http-server -o
```

http://192.168.1.39:8080

Une fois lancé sur le navigateur chrome, allez sur les 3 points verticaux en haut à droite et sélectionnez Installer angular-pwa
Vous constaterez que l'application se lance en mode PWA.
De plus, le fait de l'installer ajoute une icône sur le bureau et donc vous pourrez désormais lancer l'application PWA via cette icône.

- icône sur le bureau
- après installation, l'icône sur le bureau représente le logo Angular, celui qui se trouve dans : /assets/icons et donc vous pouvez personnaliser l'icône en changeant les images de ce dossier.
- vous pouvez voir que les icônes sont référencées dans le fichier manifest.webmanifest ainsi que le nom sous l'icône name (que vous pouvez modifier).
- désinstallation
- Pour désinstaller l'application PWA, dans celui-ci cliquez sur les 3 points et desinstaller angular-pwa
- offline
- pour tester le mode offline, sur chrome, dans les outils de développement, l'onglet Network et sur le select online choisissez offline.
- naviguez sur les pages 1 et 2 et constatez qu'en offline les ressources APIsont bien affichées (à la page 1).

XXXIII-B - Conclusion

- à partir d'un lien url on accède à la version web sur le navigateur et on fait l'action d'installer l'application via les options de chrome.
- dans le fichier ngsw-config.json on ajoute les ressources qui doivent être mises en cache (images, les url vers des ressources comme /api/**)
- sachez qu'il est possible d'ajouter un bouton 'installer l'application' pour éviter d'aller dans les options.
- Il est aussi possible de détecter une nouvelle version de l'application et de donner la possibilité de mettre à jour l'application.

XXXIV - Gestion de l'état

- on a vu dans le chapitre XIV la communication entre les composants ;
- on peut communiquer via le two way data binding ou par service ;
- par service, étant donné que celui-ci est un singleton, son instance est disponible pour tous les composants qui le demandent. Dans cette instance, on peut donc accéder et modifier en temps réel des données ;
- comme on l'a vu dans ce chapitre, ce qui est déjà mieux, on peut utiliser un observable afin que quand une donnée est modifié les composants qui ont souscrit à cet observable soient informés de ce changement.

XXXIV-A - Un petit mot sur : ngrx, ngxs

- ngrx est une version pour Angular de Redux ;
- ngxs est une version plus simple de ngrx (et donc de redux) ;
- ngrx est un magasin d'état qui centralise les données ;
- ngrx permet de gérer l'état avec un système composé de reducer, d'action et de selector.

XXXIV-B - Un petit mot sur : Akita

- Akita est une alternative simple de gestion de l'état à ngrx;
- je conseille d'utiliser Akita;

XXXIV-C - Un petit mot sur la gestion d'état en général

- sachez qu'avec Angular, dans la plupart des cas, ceci fera parfaitement l'affaire : un BehaviorSubject associé à un modèle de donnée, le tout dans un service;

XXXIV-D - Un système customisé pour Angular (pour comprendre le fonctionnement)

- cet exemple est uniquement destiné à comprendre le fonctionnement d'une gestion d'état. Préférez Akita qui est basé sur le même système que mon exemple ;
- de plus, cet exemple permettra de vous montrer différentes notions comme l'héritage, les classes génériques... ;
- pas de débat sur l'utilité de ngrx(redux) ; à vous de voir si cela vous est utile ;
- je considère redux (ngrx) comme quelque chose de lourd et pénible à utiliser ;
- notre exemple de système customisé proposera donc :
    - un abonnement pour souscrire et recevoir les données qui ont été modifiés ;
    - respecte l'immuabilité (à chaque modification, une nouvelle référence) ;
    - un mode DEV que l'on règle dans le fichier d'environnement afin de faciliter le débuggage.

XXXIV-E - Pratique

- on va gérer l'état de 2 types différents, une liste et une simple valeur ;
- dans le dossier : /todo, on gère une liste de todo sur laquelle on va ajouter, supprimer ou modifier des éléments ;
- dans le dossier : /counter, on gère une simple donnée numérique sur laquelle on va incrémenter son élément.

```
ng new angular-state-manager1
strict ? NO
routing ? NO
SCSS
```

```
ng g m todo --module=app

ng g s core/store/base-store

ng g m /todo --module=app
ng g c todo/components/todo --module=todo
ng g i todo/models/i-todo
ng g s todo/services/store-todo

ng g m /counter --module=app
ng g c counter/components/counter --module=counter
ng g i counter/models/i-count
ng g s counter/services/store-count
```

/core/store/base-store.service.ts

```
import { Injectable } from "@angular/core";
import { BehaviorSubject } from "rxjs";
import { uuid } from "./uuid";
import { environment } from "../../../environments/environment";

export interface Base {
  id: string;
}

export interface INotif<T extends Base> {
  action: "ADD" | "REMOVE" | "UPDATE" | "COMPLETED";
  item: T | string;
  items: Array<T>;
}

@Injectable({
  providedIn: "root",
})
export abstract class BaseStoreService<T extends Base> {
  // Gestion de l'état
  private readonly _values = new BehaviorSubject<T[]>(this.getInitial());
  readonly values$ = this._values.asObservable();

  get values(): T[] {
    return this._values.getValue();
  }

  set values(val: T[]) {
    this._values.next(val);
  }

  // Gestion d'une notification pour le mode développement
  private _notifs: Array<INotif<T>> = [];

  get notifs(): INotif<T>[] {
    return this._notifs;
  }

  addNotif(notif: INotif<T>) {
```

```
40.     this.notifs.push(notif);
41.   }
42.
43.   // on déclare les fonctions en abstract pour pouvoir les redéfnier dans : StoreTodoService
 et StoreCountService
44.   abstract getTypeName(): any;
45.
46.   abstract getInitial(): any;
47.
48.   //  Les actions de base : ADD, REMOVE, UPDATE, FINDBYID...
49.   add(value: T) {
50.     if (value) {
51.       value.id = uuid();
52.       this.values = [...this.values, value];
53.
54.       this.devMode("ADD", value);
55.     }
56.   }
57.
58.   remove(value: T) {
59.     this.values = this.values.filter((v: T) => v.id !== value.id);
60.     this.values = [...this.values];
61.
62.     this.devMode("REMOVE", value);
63.   }
64.
65.   update(value: T, devName = "UPDATE") {
66.     const index = this.values.indexOf(value);
67.
68.     this.values[index] = {
69.       ...value,
70.     };
71.
72.     this.values = [...this.values];
73.
74.     this.devMode(devName, value);
75.   }
76.
77.   findById(id: string): T | undefined {
78.     return this.values.find((v: T) => v.id === id);
79.   }
80.
81.   // en mode DEV, on affiche dans la console l'action qui a été réalisé (pour le débuggage)
82.   devMode(action: string, item: T | string) {
83.     if (environment.storeInDevMode) {
84.       const notif = {
85.         action: action,
86.         item: item,
87.         items: this.values,
88.       } as INotif<T>;
89.       this.addNotif(notif);
90.       console.log();
91.       console.log(
92.         "DEV MODE - notifications : " + this.getTypeName(),
93.         this.notifs
94.       );
95.     }
96.   }
97. }
```

/core/store/uuid.ts

```
1. export function uuid() {
2.   return "xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx".replace(/[xy]/g, function (c) {
3.     var r = (Math.random() * 16) | 0,
4.       v = c == "x" ? r : (r & 0x3) | 0x8;
5.     return v.toString(16);
6.   });
7. }
```

/counter/components/counter/counter.component.html

```
<p>counter works!</p>
<div *ngIf="count$ | async as obj">count = {{ obj[0].value }}</div>
```

/counter/components/counter/counter.component.ts

```
import { Component, OnInit } from "@angular/core";
import { Observable } from "rxjs";
import { ICount } from "../../models/i-count";
import { StoreCountService } from "../../services/store-count.service";

@Component({
  selector: "app-counter",
  templateUrl: "./counter.component.html",
  styleUrls: ["./counter.component.scss"],
})
export class CounterComponent implements OnInit {
  count$: Observable<ICount[]> = this.storeCountService.values$;

  constructor(private storeCountService: StoreCountService) {}

  ngOnInit(): void {
    this.storeCountService.inc(); // on appelle 3 fois l'incrémentation
    this.storeCountService.inc();
    this.storeCountService.inc();
  }
}
```

/counter/models/i-count.ts

```
export interface ICount {
  id: string;
  value: number;
}
```

/counter/services/store-count.service.ts

```
import { Injectable } from "@angular/core";
import { BaseStoreService } from "../../core/store/base-store.service";
import { ICount } from "../models/i-count";

@Injectable({
  providedIn: "root",
})
export class StoreCountService extends BaseStoreService<ICount> {
  getTypeName(): string {
    // pour le mode DEV, toujours renvoyer le type
    return "ICount";
  }

  getInitial(): ICount[] {
    // ici, il faut toujours retourner un tableau de quelquechose.
  // ça peut être un tableau avec un objet contenant une propriété comme ici
  // value va contenir la valeur du compteur
  //
  // ou alors on peut initialiser avec des données que l'on récupere d'une api
  //
    return [{ value: 0 } as ICount];
  }

  // incrémentation d'une valeur
  // comme c'est une action particulière qui ne fait pas partie des actions de bases comme
 (ADD, REMOVE, UPDATE...)
  // on surcharge en écrivant ici la fonction
  inc() {
```

```
28.     const obj = this.values[0] as ICount; // comme ce n'est pas une liste, c'est un tableau
 avec un seul index qui est égal à 0
29.     if (obj) {
30.       obj.value++; // on effectue l'action ici, on incrémente
31.
32.       this.update(obj, "INC"); // on fait appelle à update de la classe abstraite :
 BaseStoreService
33.       // en précisant le nom de l'action : 'INC'
34.     }
35.   }
36. }
```

/counter/counter.module.ts

```
1. import { NgModule } from "@angular/core";
2. import { CommonModule } from "@angular/common";
3. import { CounterComponent } from "./components/counter/counter.component";
4.
5. @NgModule({
6.   declarations: [CounterComponent],
7.   imports: [CommonModule],
8.   exports: [CounterComponent],
9. })
10. export class CounterModule {}
```

app.component.html

```
1. <app-todo></app-todo>
2. <hr />
3. <app-counter></app-counter>
```

/counter/counter.module.ts

```
1. import { NgModule } from "@angular/core";
2. import { CommonModule } from "@angular/common";
3. import { CounterComponent } from "./components/counter/counter.component";
4.
5. @NgModule({
6.   declarations: [CounterComponent],
7.   imports: [CommonModule],
8.   exports: [CounterComponent],
9. })
10. export class CounterModule {}
```

/todo/components/todo/todo.component.html

```
1. <p>todo works!</p>
2.
3. <div *ngFor="let v of values$ | async">{{ v | json }}</div>
```

/todo/components/todo/todo.component.ts

```
1. import { Component, OnInit } from "@angular/core";
2. import { Observable } from "rxjs";
3. import { ITodo } from "../../models/i-todo";
4. import { StoreTodoService } from "../../services/store-todo.service";
5.
6. @Component({
7.   selector: "app-todo",
8.   templateUrl: "./todo.component.html",
9.   styleUrls: ["./todo.component.scss"],
10. })
11. export class TodoComponent implements OnInit {
12.   values$: Observable<ITodo[]> = this.storeTodoService.values$;
13.
14.   constructor(private storeTodoService: StoreTodoService) {}
```

```
15.
16.   ngOnInit() {
17.     //
18.     console.log("add todo1 --------------------------");
19.     const todo1 = { title: "do1", isCompleted: false } as ITodo;
20.     this.storeTodoService.add(todo1);
21.     //
22.     console.log("remove todo1 -------------------------");
23.     this.storeTodoService.remove(todo1);
24.     //
25.     console.log("add todo1 todo2 --------------------------");
26.     const todo2 = { title: "do2", isCompleted: false } as ITodo;
27.     this.storeTodoService.add(todo1);
28.     this.storeTodoService.add(todo2);
29.     //
30.     console.log("findbyId -------------------------");
31.     const ftodo2 = this.storeTodoService.findById(todo2.id);
32.     console.log("todo trouvé : ", ftodo2);
33.     console.log();
34.     //
35.     console.log("setCompleted -------------------------");
36.     if (ftodo2) {
37.       this.storeTodoService.setCompleted(ftodo2.id, true);
38.     }
39.   }
40. }
```

/todo/models/i-todo.ts

```
1. export interface ITodo {
2.   id: string;
3.   isCompleted: boolean;
4.   title: string;
5. }
```

/todo/services/store-todo.service.ts

```
1. import { Injectable } from "@angular/core";
2. import { ITodo } from "../models/i-todo";
3. import { BaseStoreService } from "../../core/store/base-store.service";
4.
5. @Injectable({
6.   providedIn: "root",
7. })
8. export class StoreTodoService extends BaseStoreService<ITodo> {
9.   // ne pas oublier de nommer le type, cela va servir pour le mode DEV
10.   getTypeName(): string {
11.     return "ITodo";
12.   }
13.
14.   getInitial(): Array<ITodo> {
15.     // pour l'initialisation d'une liste, toujours retourner un tableau vide
16.  //
17.  // ou alors on peut initialiser avec des données que l'on récupere d'une api
18.  //
19.     return [];
20.   }
21.
22.   // si on a une action particulière à faire (qui ne fait pas partie des actions de base
 comme : ADD, REMOVE...)
23.   // elle est spécifique à Todo, on est dans le service todo
24.   // donc on l'a met ici
25.   setCompleted(id: string, isCompleted: boolean) {
26.     let todo: ITodo | undefined = this.values.find((v: ITodo) => v.id === id);
27.     if (todo) {
28.       todo.isCompleted = isCompleted;
29.       this.update(todo, "COMPLETED");
30.     }
31.   }
```

```
32. }
```

/todo/todo.module.ts

```
1. import { NgModule } from "@angular/core";
2. import { CommonModule } from "@angular/common";
3. import { TodoComponent } from "./components/todo/todo.component";
4.
5. @NgModule({
6.   declarations: [TodoComponent],
7.   imports: [CommonModule],
8.   exports: [TodoComponent],
9.   providers: [],
10. })
11. export class TodoModule {}
```

app.component.html

```
1. <app-todo></app-todo>
2. <hr />
3. <app-counter></app-counter>
```

app.module.ts

```
1. import { NgModule } from "@angular/core";
2. import { BrowserModule } from "@angular/platform-browser";
3. import { AppRoutingModule } from "./app-routing.module";
4. import { AppComponent } from "./app.component";
5. import { TodoModule } from "./todo/todo.module";
6. import { CounterModule } from "./counter/counter.module";
7.
8. @NgModule({
9.   declarations: [AppComponent],
10.   imports: [BrowserModule, AppRoutingModule, TodoModule, CounterModule],
11.   providers: [],
12.   bootstrap: [AppComponent],
13. })
14. export class AppModule {}
```

/environments/environments.ts

```
1. export const environment = {
2.   production: false,
3.   storeInDevMode: true,
4. };
```

/environments/environments.prod.ts

```
1. export const environment = {
2.   production: true,
3.   storeInDevMode: true,
4. };
```

XXXIV-F - Résultat

- dans le composant : todo.component.ts, on effectue diverses actions (ADD, REMOVE...) ;
- en activant le mode DEV (voir les fichiers d'environnements), est affiché dans la console toutes les étapes par lesquelles l'état est passé (ADD, ADD, REMOVE...)

XXXIV-G - Récapitulatif

- on active ou pas le mode DEV
- pour créer une gestion d'état, il suffit de créer son modèle de données et son service héritant de : BaseStoreService
- exemple pour gérer l'état d'une liste de produits que l'on sélectionne parmi des produits :

i-product-choice.ts

```
export interface IProductChoice {
...
...
```

store-product-choice.service.ts

```
import { Injectable } from "@angular/core";
import { IProductChoice } from "../models/i-product-choice";
import { BaseStoreService } from "./base-store.service";

@Injectable({
  providedIn: "root",
})
export class StoreProductChoiceService extends BaseStoreService<IProductChoice> {
  getTypeName(): string {
     return "IProductChoice";
   }

   getInitial(): Array<IProductChoice> {
     return [];
   }
}
```

Et voilà, via this.storeProductChoiceService., vous pouvez effectuer les actions de bases (ADD, REMOVE, UPDATE...)

XXXIV-H - stackblitz

https://stackblitz.com/edit/angular-ivy-ycnz4n

XXXV - Un petit mot pour la fin

Si vous constatez des erreurs ou des inexactitudes, n'hésitez pas à m'en informer sur le forum ou par mail, merci.

Ce tutoriel présente les bases les plus importantes à connaitre sur chaque chapitre, pour approfondir vos connaissances il faut vous servir de la documentation officielle ou d'autres tutoriels que l'on peut trouver sur Internet.

XXXVI - Remerciements

Je tiens à remercier **LittleWhite** pour sa relecture technique, **Mickael Baron** et **Malick** pour leur encadrement ainsi que **Claude Leloup**, **jacques_jean** et **escartefigue** pour la relecture orthographique.