



Formation Angular

Initiation

Votre formateur : Sonia BAIBOU

Nous contacter :

Christophe Guérault

c.gueroult@coderbase.io

+33 6 56 85 84 33



CODERBASE IT

WE THINK HUMAN FIRST

0.1

Présentations





Participants

- Les bases du HTML, CSS, JS
- Les concepts de la POO
- Avez vous déjà utilisé un framework js type angular, angular js, react ou vueJs ?
- Droits admin obligatoires sur les postes
- Quelles sont vos attentes par rapport à cette formation ?
- Quelles sont les technos et langages sur lesquels vous travaillez habituellement ?

0.2

Go to Slack





Accès à Slack

- Vérifiez dans vos email que vous avez bien reçu le lien d'invitation au Slack de la formation.
- Merci de bien vouloir me rejoindre sur Slack.

0.3

GitHub





Le dépôt GitHub de la formation

- Donner le lien du dépôt de la formation sur Slack
- A chaque notion abordée : un commit
- Vous pourrez fork mon dépôt après la formation
- Si vous souhaitez refaire l'application après la formation, vous aurez accès à tous les commit pour chaque étape de l'application.

0.4

Supports de cours et livres





Support et **livres**

- Adresse gmail des participants dans slack
- Partage du support de cours en lecture seule
- Dossier Livres avec les droits éditeur

0.5

Déroulement de la formation





Horaires, temps de pause et dossier pédagogique

- De 9h à 17h ou de 9h30 à 17h30 (voir convocations)
- Une pause de 15mn le matin et une pause de 15mn l'après midi
- Midi, une pause déjeuner de 1h
- Dernier jour de la formation se termine à 15h (voir convocations)
- Obligation du dernier jour à faire avant 15h donc, au retour de la pause déjeuner : vérifier les feuilles d'émarginement et faire les évaluations formateur
- Si certifications à passer, elles se feront en ligne le dernier jour l'après midi, après les évaluations formateur.
- Le dernier jour nous passerons en mode Atelier.

0.6

Déroulé pédagogique





L'application CRM

- Une application de A à Z
- Prestations ensemble étape par étape. TP clients pour validation des acquis de milieu de formation.
- <https://crm-paris-sept-2019.web.app/prestations>



Le processus pédagogique

- Pour chaque notion abordée :
 - Ce que nous voulons faire, quelle fonctionnalité développer
 - L'Objectif attendu suite à la notion que nous allons aborder
 - La partie théorique et où trouver l'information dans la doc officielle
 - La pratique en codant sur l'application
 - Le code sur Slack pour ceux qui veulent faire un copier-coller
 - Un petit exercice de 5mn pour valider l'acquisition de la notion abordée



L'Atelier du dernier jour

- Un ensemble de TP qui vous obligeront à écrire votre propre code, à chercher dans la doc, sur google et à revoir le code que nous avons écrit durant la semaine
- Le but étant que vous soyez le plus à l'aise possible avec le framework Angular et que vous sachiez développer vos propres fonctionnalités Angular à la fin de la formation.

0.7

Installations outils et IDE





Install NodeJS

- NodeJs et NPM (Node Package Manager) pour installer les librairies utilisées durant la formation et l'outil Angular CLI
<https://nodejs.org/en/>
- Version LTS (tout le monde la même version)
- Vérifiez dans cmd la version installée

Ou NVM (Node Version Manager)

<https://github.com/coreybutler/nvm-windows/releases>

Nvm list

Nvm install 14.17.0

Nvm use 14.17.0



Install VsCode et Plugins

- <https://code.visualstudio.com/>
- Angular Essentials (John Papa)
- Auto Rename Tag
- Sass
- Préférences de VsCode avec l'autosave et settings.json
File> Préférences > Settings puis clic sur Icône en haut à droite

The screenshot shows the Visual Studio Code interface with the settings.json file open. The file path is C:\Users\chris\AppData\Roaming\Code\User\settings.json. The code in the file is as follows:

```
C:\> Users > chris > AppData > Roaming > Code > User > settings.json > ...
1 "window.zoomLevel": 2,
2 "angular.enableExperimentalIvyPrompt": false,
3 "git.enableSmartCommit": true,
4 "[typescript]": {
5   "editor.defaultFormatter": "esbenp.prettier-vscode",
6   "editor.codeActionsOnSave": {
7     "source.organizeImports": true
8   }
9 },
10 "editor.formatOnSave": true,
11 "files.autoSave": "onFocusChange",
12 "[html)": {
13   "editor.defaultFormatter": "vscode.html-language-features"
14 },
15 "javascript.updateImportsOnFileMove.enabled": "always",
16 "terminal.integrated.shellArgs.windows": []
```



Install Angular CLI en global

- Doc officielle => Getting started => Setup
<https://www.npmjs.com/package/@angular/cli/v/13.3.9>

```
npm i -g @angular/cli@13.3.9
```

- Angular CLI est un outil développé par Angular Teams
- Gestionnaire de Bundle
- Permet de concevoir des applications modulaires
- Permet en ligne de commande de créer des workspaces Angular, applications, librairies, composants (Modules, services, pipes, directives, components,)
- Install par défaut un ensemble d'outils et dépendances (TypeScript, compilateur JS, Karma pour les Test Unitaires et Protractor pour les tests End To End)

0.8

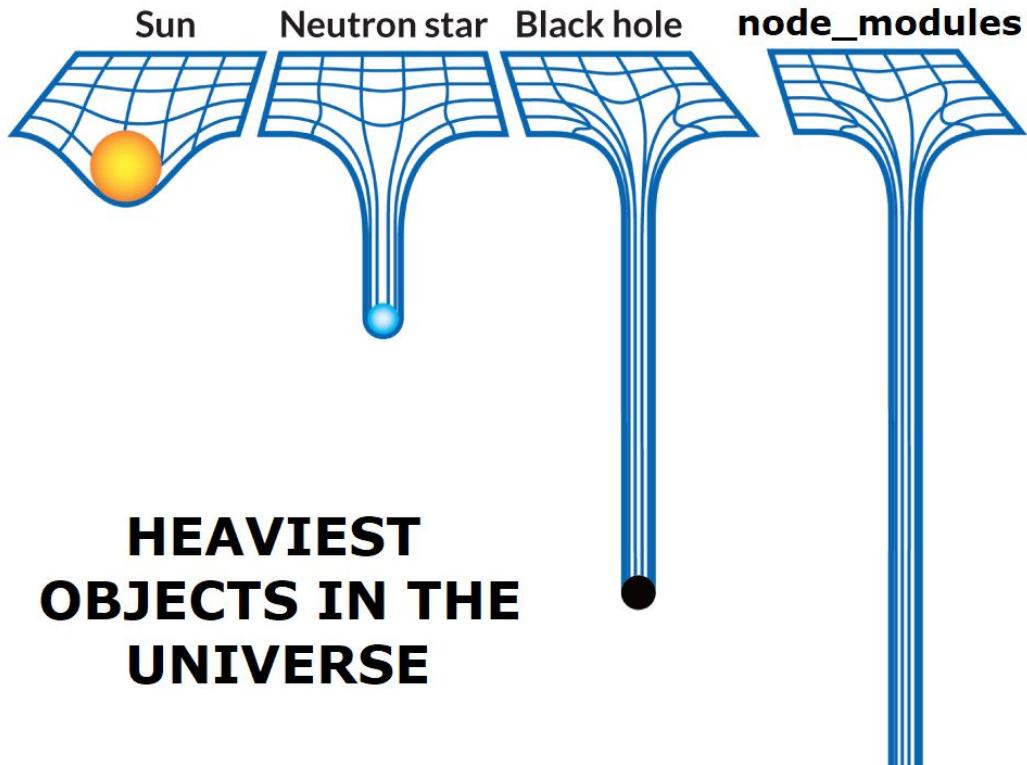
**Workspace et application
Angular avec Angular CLI**





Le workspace et application Angular

- `ng new`
 - Tslint config
 - Config de TypeScript
 - Config de Karma pour les tests unitaires
 - Git ignore et readme.md (avec dépôt git local si git installé sur vos machines)
 - Nodes modules (package.json et package-lock.json)
 - .editorConfig
 - Angular.json avec config de l'application
 - Main.ts qui démarre l'application
 - Les polyfills
 - Les styles globaux
 - L'application avec app module, app routing module et app component
 - Le dossier environment
 - Le dossier assets
 - La page d'index



**HEAVIEST
OBJECTS IN THE
UNIVERSE**

**Pas besoin d'importer
des node_modules**

npm install

0.9

TypeScript





Surlangage **TypeScript**

- Un surlangage développé par Microsoft.
- <https://www.typescriptlang.org/>
- Syntaxe plus proche de l'objet (java, C#)
- Apporte quelques éléments supplémentaires (enums, typage fort, interfaces...)
- Aucun navigateur ne comprend le typeScript puisque ce n'est pas un langage
- Un compilateur js est nécessaire pour compiler une appli écrite en typeScript
- Votre application doit être compilée dans une version de EcmaScript implémentée sur les navigateurs.

0.10

EcmaScript





Le standard **EcmaScript**

- Standardise le JavaScript
- ES5, ES6, ES7, ES8, ES9, ES10, ES11
- ES, ES2015, ES2016, ES2017, ES2018, ES2019, ES2020
- Chaque version apporte son lot de nouveautés
- Tous les navigateurs implémentent ES5
- Les navigateurs les plus récents implémentent ES6
- Utiliser différentes versions de ES dans une application implique qu'il va falloir compiler le Js en ES5 ou ES6



Back vs Front

Code Back

Maintenabilité
Réutilisabilité
Objets
Encapsulation
Héritage

Code Front

Maintenabilité
Réutilisabilité
Objets
Encapsulation
Héritage

- Les web components permettent de mettre en application côté Front les concepts généralement utilisés côté Back (L'utilisation des Objets, l'héritage, l'encapsulation, la maintenabilité, la réutilisabilité)
- https://developer.mozilla.org/fr/docs/Web/Web_Components



Web Components

Index.html :

- Fichier html
- Fichier css
- Fichier js

- Fichier html
- Fichier css
- Fichier js

- Fichier html
- Fichier css
- Fichier js

- Un web component est un package constitué d'un fichier html, d'un fichier JS et d'un fichier css
- Quand on instancie une Class (fichier JS) un template (fichier html) est instancié
- Une instance de la Class est associée à une instance du template
- L'instance de la Class permet de contrôler l'instance du template associée
- Le JavaScript et le css sont encapsulés.



Instancier un Web Component

app-component.html :

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

```
<app-login></app-login>
```

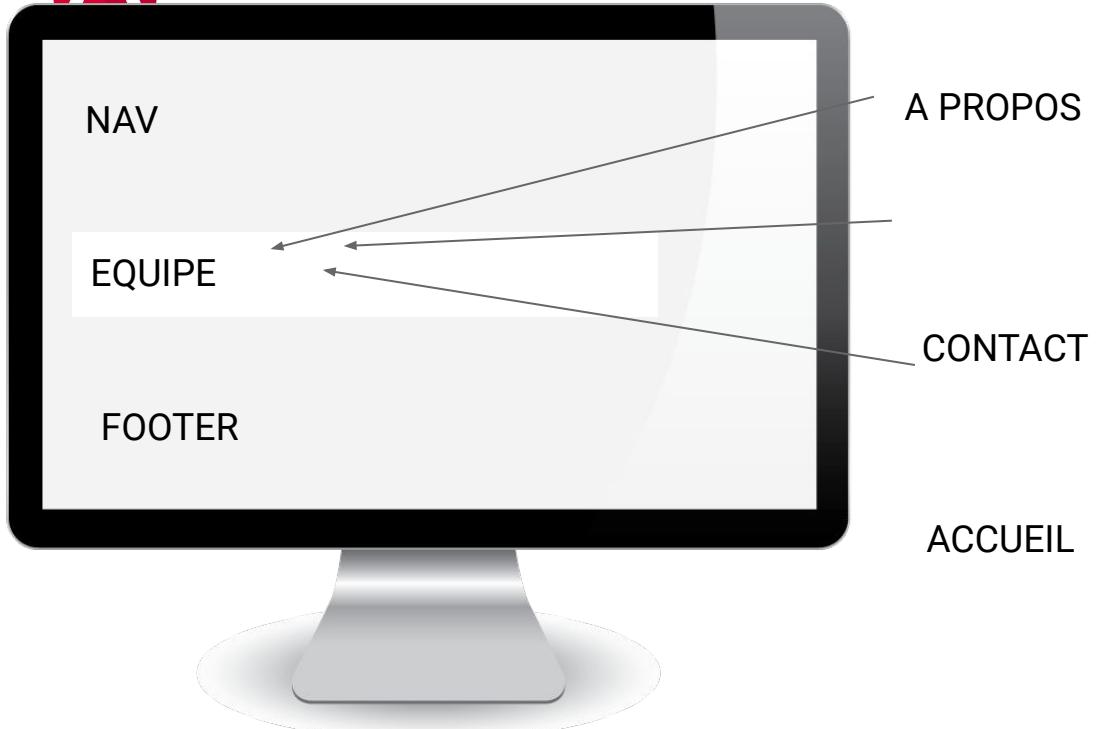
```
<app-form></app-form>
```

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

- On peut créer autant d'instances d'un web component que l'on veut dans un template html
- On peut imbriquer des web components dans des templates de web components
- On crée une instance d'un web component avec Angular dans un template html en utilisant son selector
- Pour l'encapsulation Angular : Shadow Dom, Emulated (par défaut) ou None

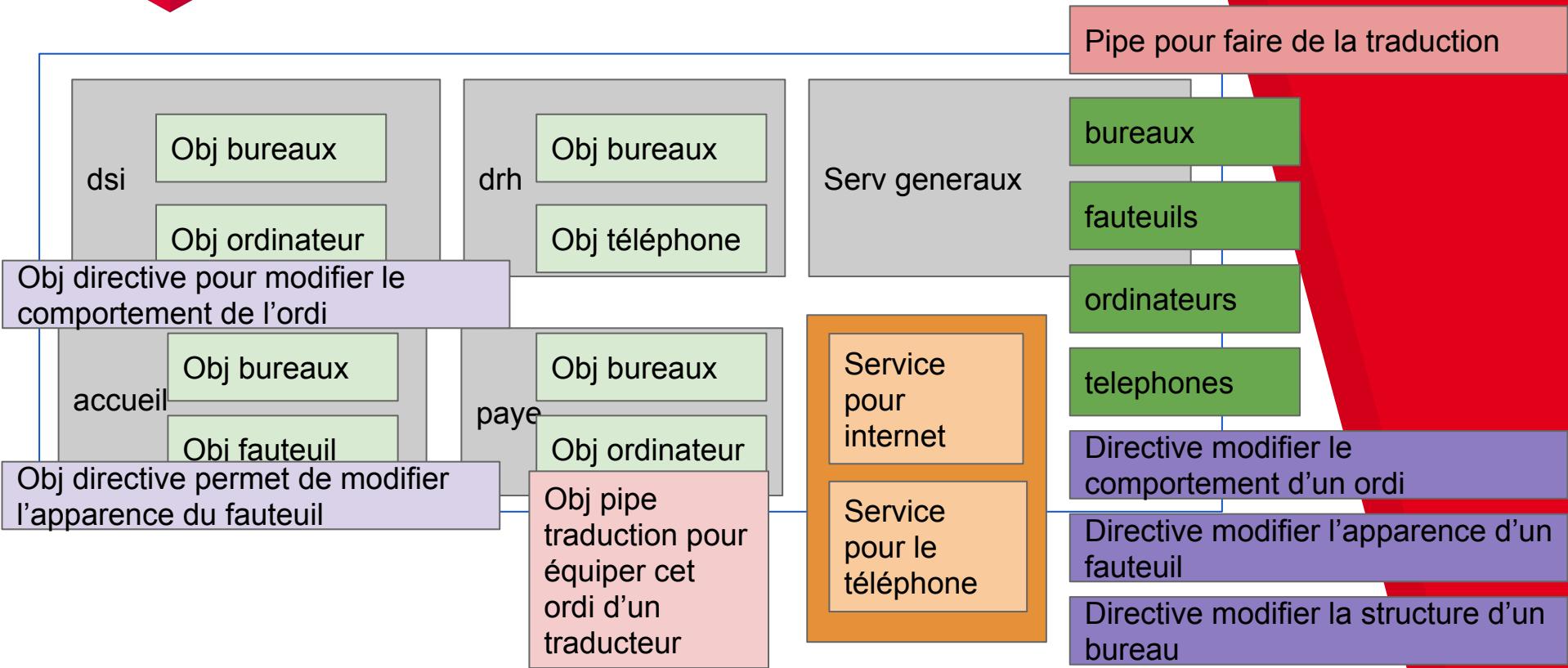


Fonctionnement





Modules Angular et composants



0.12

Les tests unitaires





Tests unitaires

- `ng test`
 - Par défaut Angular a choisi Karma
 - On peut aussi utiliser Jest (couramment utilisé par les devs)
 - On va voir TestBed et RouterTestingModule d'angular
 - Les test sont écrits avec Jasmine, librairie js pour les tests unitaires
- A la fin de la formation :
 - Un schematics pour install Jest à la place de Karma (plus utilisé et plus de ressources sur le web)
 - Un CRM complet avec un exemple de test unitaire pour chaque composant d'une appli angular

0.13

Lancer l'appli Angular en
Local

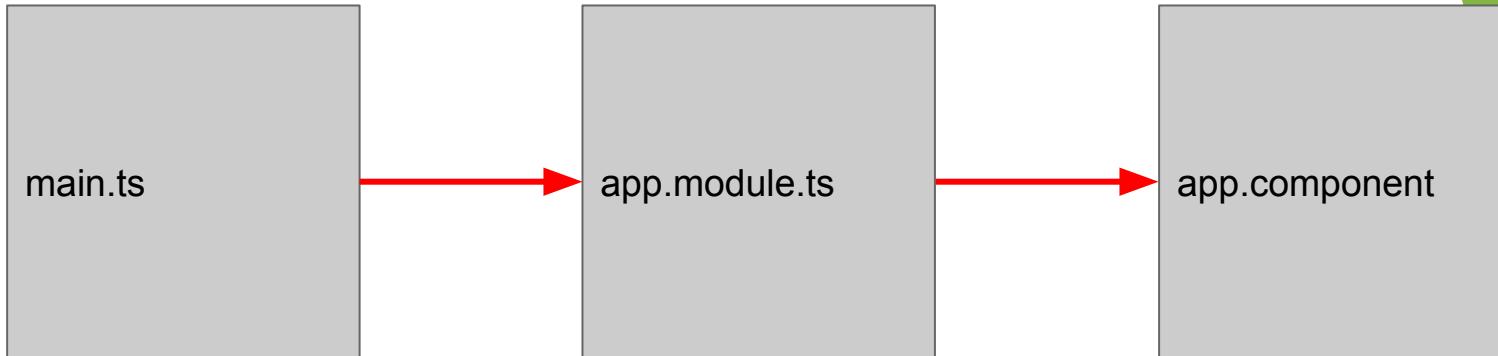




Ng serve

- `ng serve` / `npm run start` permet de lancer l'appli en local
 - L'appli s'ouvre par défaut dans un localhost (voir l'url dans le terminal, cela change selon la version d'angular)
 - On constate qu'`<app-root>` reste présent dans le DOM et qu'entre la balise `<app-root></app-root>` tout le html présent sur `app-component.html` est rendu dans de DOM

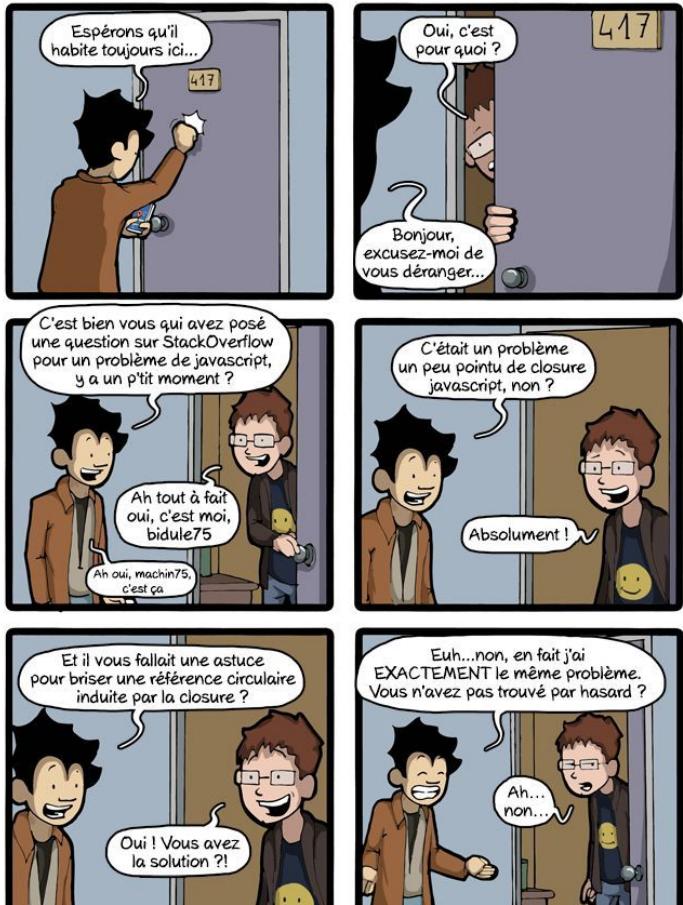
En résumé



1

La création des modules







Le découpage en modules

AppModule

SharedModule

AppRoutingModule

CoreModule

- **ng g module shared**
- **ng g module core**



Les Modules Agiles

UiModule

IconsModule

TemplatesModule

- **ng g module ui**
- **ng g module icons**
- **ng g module templates**



Les Modules métiers & leur module routing associé

OrdersModule

OrdersRoutingModule

ClientsModule

ClientsRoutingModule

LoginModule

LoginRoutingModule

- **ng g module orders --routing**
- **ng g module clients --routing**
- **ng g module login --routing**

Ne pas oublier la commande `-routing`, sinon, vous devrez créer le fichier de routing manuellement.



Exercice

- Créez le module PageNotFound associé à son module de routing PageNotFoundRouting



Correction

PageNotFoundModule

PageNotFoundRoutingModule

- Ng g module page-not-found --routing



Ce qu'il faut retenir

- Le découpage de l'application en module est le premier niveau de granularité de l'architecture de votre application
- Chaque module a un rôle et une utilité spécifique
- Les modules orientés Agile favorisent les changements en cours de projet et limitent les risques de régression au niveau du css
- Les modules qui déclarent des composants qui constituent les vues de l'application doivent être accompagnés de leur propre module de routing.



Conseils pratiques

- Ne créez que des modules à la racine de votre projet
- 1er niveau de granularité = modules
- Modules = 1 niveau dans l'arborescence des dossiers à la racine
- Architecture plus simple à comprendre

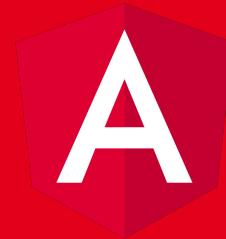


Commit 1 : Modules

- Retrouvez sur le dépôt Github de la formation le commit 1 concernant la création des modules

2

Imports des modules au démarrage





6 mois plus tard...



CommitStrip.com

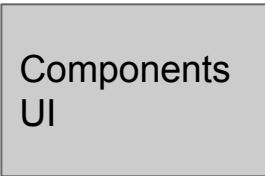




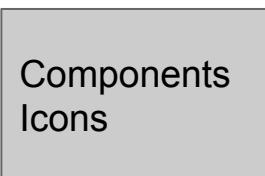
CoreModule le chef d'orchestre

- AppModule a pour unique responsabilité de démarrer l'appli
- Il importe le CoreModule
- C'est le CoreModule qui a la responsabilité d'exporter tous les modules qui doivent être chargés au démarrage de l'appli.

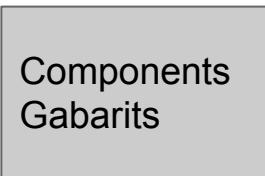
UI Module



IconsModule



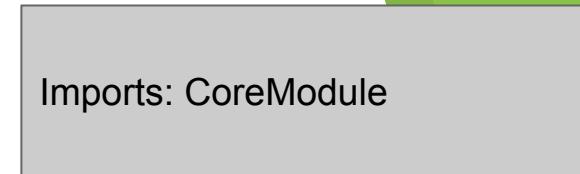
TemplatesModule



CoreModule



AppModule





Import de CoreModule sur AppModule

- Le style guide Angular recommande de n'importer le CoreModule qu'une seule fois dans toute l'appli au démarrage et jamais dans aucun autre module

```
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { CoreModule } from './core/core.module';

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



Export des modules sur CoreModule

- Tous les modules qui doivent être chargé au démarrage de l'appli sont exportés sur CoreModule
- AppModule, en important CoreModule, importera en même temps tous les modules exportés sur CoreModule
- C'est la responsabilité du CoreModule de décider quels sont les modules qui doivent être chargés au démarrage de l'appli.

```
@NgModule({  
  declarations: [],  
  imports: [CommonModule],  
  exports: [UiModule, IconsModule, TemplatesModule, LoginModule]  
})  
export class CoreModule {}
```



Ce qu'il faut retenir

- Le rôle de AppModule est de démarrer l'appli
- C'est le seul à importer CoreModule
- Le rôle de CoreModule est d'exporter tous les modules qui doivent être chargés au démarrage
- Les modules nécessaires au démarrage sont ceux qui contiendront des components nécessaires au démarrage de l'appli sur la page d'accueil



Conseils pratiques

- Rester dans une logique : une class = un rôle
- Si le rôle de AppModule est de démarrer l'appli
- Le rôle de CoreModule est celui d'orchestrer votre appli



Commit 2 : import CoreModule

- Retrouvez sur le dépôt Github de la formation le commit 2 avec l'import de CoreModule sur AppModule et les exports de UiModule, TemplatesModule et IconsModule sur CoreModule

3

Les vues de l'application





Une vue = une page

- Les vues sont des webcomponents qui correspondent aux pages de l'application
- Ce sont des webcomponents intelligents. Qui traitent la logique dans le ts (datas à récupérer par exemple, quelques fonctions appelées dans le html)
- Créer un webcomponent par vue permet d'avoir une conception Agile
- Le dynamisme se fait au niveau des webcomponents qui sont instanciés dans chacune de ses vues (components réutilisables)
- Nous pourrons ainsi accepter en cours de projet des changements sur une vue sans avoir à refaire l'existant.



Création des **components pages** dans un dossier **pages**

- **ClientsModule :**
 - Ng g component page-list-clients
 - Ng g component page-add-client
 - Ng g component page-edit-client
- **LoginModule :**
 - Ng g component page-sign-in
 - Ng g component page-sign-up
 - Ng g component page-reset-password
 - Ng g component page-forgot-password



Exercice

- Créez les components pour les pages suivantes et rattachez les aux bons modules : Ng generate component name
 - PageListOrdersComponent
 - PageAddOrderComponent
 - PageEditOrderComponent
 - PageNotFoundComponent



Correction

- **OrdersModule :**
 - **ng g component page-list-orders**
 - **ng g component page-add-order**
 - **ng g component page-edit-order**
- **PageNotFoundModule :**
 - **ng g component page-not-found**



Ce qu'il faut retenir

- Un webcomponent par vue vous permettra d'avoir une conception plus agile et d'accepter les modifications en cours de projet
- Dans chaque vue nous utiliserons un ensemble de webcomponent réutilisables
- Appliquer des règles de nomenclature et utiliser des préfixes pour vos components
- Le découpage en vues est le deuxième niveau de granularité de l'architecture de votre appli, donc 2e niveau dans l'arborescence des dossiers
- Les vues sont des components intelligents (logique et prises de décisions dans le ts)



Conseils pratiques

- Mettez-vous d'accord sur les règles de nommage de vos components
- Pré-fixez toujours vos noms de component
- Pour les vues de l'application, utilisez le préfixe view ou page et idéalement, créez vos components dans un répertoire nommé “views” ou “pages”
- Pour éviter de vous tromper d'emplacement lorsque vous créez des webcomponents, faites un clic droit sur le dossier dans lequel vous créez le webcomponent et choisissez “ouvrir dans le terminal intégré”
- Si vous devez renommer un webcomponent, supprimez-le et recréez-le ça ira plus vite. Pensez à supprimer la déclaration du component supprimé dans le module.



Commit 3 : **pages**

- Retrouvez sur le dépôt Github de la formation le commit 3 concernant les pages de l'application

compodoc

<https://compodoc.app/>

<https://compodoc.app/guides/installation.html>

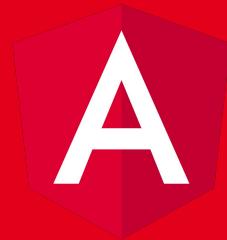
“compodoc”: “compodoc -p tsconfig.json -o -s”

Details des commandes :

<https://compodoc.app/guides/options.html>

4

Config scss





Création des fichiers de variables

- Les variables css et scss sont utilisées dans toute l'appli dans les styles de component
- Une appli plus simple à maintenir
- src > stylings > un fichier de variables pour le thème, le layout et les fonts

Variables css

```
:root {  
  --app-h1-fs: 1.5rem;  
  --app-h2-fs: 1.2rem;  
  --app-h3-fs: 1.1rem;  
  --app-regular-fs: 1rem;  
}
```

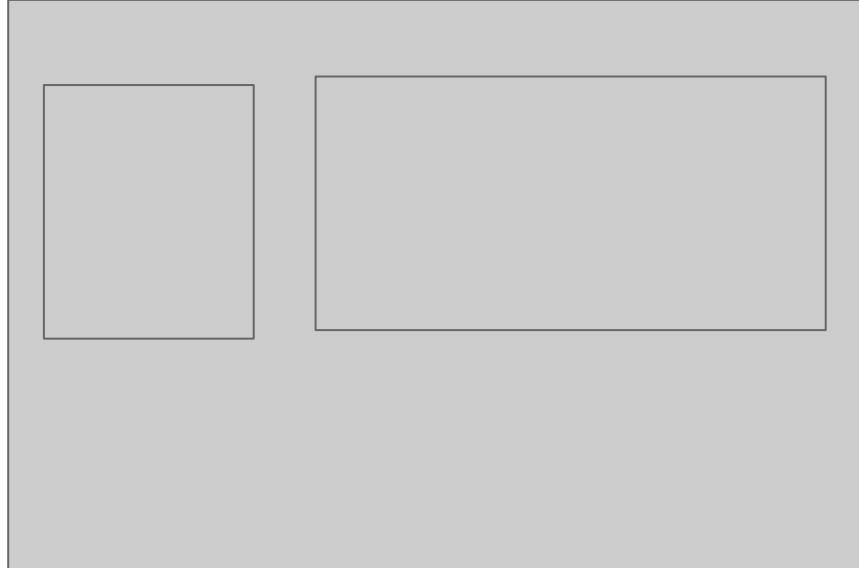
Variables scss

```
$sm-breakpoint: 576px;  
$md-breakpoint: 768px;  
$lg-breakpoint: 992px;  
$xl-breakpoint:  
  1200px;  
$gutter: 15px;
```

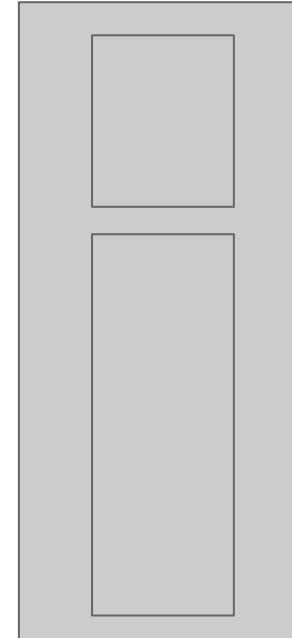
Restent intact dans le fichier css compilé et donc on peut y accéder après la compilation

sont remplacées par des valeurs. On n'y accède plus après la compilation

Vue desktop



Moins de 768px





Fichier de config `_fonts.scss`

```
:root {  
    --app-h1-fs: 1.5rem;  
    --app-h2-fs: 1.2rem;  
    --app-h3-fs: 1.1rem;  
    --app-regular-fs: 1rem;  
}
```



Fichier de config `_layout.scss`

- Ici on garde les variables scss car on va les utiliser dans nos media queries et le standard W3C n'autorise pas encore les variables css dans les media queries

```
// breakpoints
$sm-breakpoint: 576px;
$md-breakpoint: 768px;
$lg-breakpoint: 992px;
$xl-breakpoint: 1200px;
// gutters
$gutter: 15px;
```



Fichier de config `_theme.scss`

```
:root {  
  --app-primary: #651a97;  
  --app-primary-dark: #390958;  
  --app-secondary: #910835;  
  --app-dark: #2a2a2a;  
  --app-light: #ffffff;  
  --app-darklight: #ebedf2;  
  --app-darklight2: #bebdbd;  
  --app-error: #e2445c;  
  --app-warning: #f25c05;  
  --app-success: #9ebf6b;  
}
```



Ce qu'il faut retenir

- Les variables css restent intactes dans le fichier css compilé et sont accessibles après la compilation
- Les variables scss sont remplacées par des valeurs dans le fichier css compilé et ne sont plus accessibles après la compilation
- Les media queries ne permettent pas l'utilisation des variables css
- Le _ (trait de soulignement) est un partielle pour scss. Cela signifie que la feuille de style va être importée (@import) dans une feuille de style principale, à savoir styles.scss. L'avantage d'utiliser les partiels est que vous pouvez utiliser de nombreux fichiers pour organiser votre code et tout sera compilé sur un seul fichier
- <https://stackblitz.com/edit/angular-css-part>



Conseils pratiques

- Privilégiez les variables css (sauf incompatibilité avec les navigateurs anciens)
- Créez des fichiers distincts pour le thème, le layout, et les fonts. Cela permet de s'y retrouver plus facilement



Commit 4 : config scss

- Retrouvez sur le dépôt Github de la formation le commit 4 concernant la configuration scss de l'application

5

**Install framework css
Bootstrap & librairie de
components ng-bootstrap**





About bootstrap & ng-bootstrap

- Bootstrap est un framework css mais il est dépendant de jQuery et PopperJs (accordéons ou les datePickers par exemple)
- En installant Bootstrap avec npm, nous allons avoir un warning dans la console du terminal nous indiquant que bootstrap est dépendant de PopperJs
- Nous souhaitons utiliser uniquement la partie css de bootstrap et les composants bootstrap indépendants de PopperJs pour rester sur du typeScript.
- En installant ng-bootstrap en plus de bootstrap, nous retrouverons tous les composants de bootstrap animés (accordéons ou datePickers) mais en typescript
- Ng-bootstrap est une librairie écrite spécialement pour Angular



Install bootstrap & ng-bootstrap

- Le schematics `ng add @ng-bootstrap/ng-bootstrap` s'occupe de dire à angular cli tout ce qu'il faut faire pour installer bootstrap et ng-bootstrap

```
The package @ng-bootstrap/ng-bootstrap@11.0.0 will be installed and executed.  
Would you like to proceed? Yes  
✓ Package successfully installed.  
UPDATE package.json (1269 bytes)  
✓ Packages installed successfully.  
UPDATE src/app/app.module.ts (526 bytes)  
UPDATE src/styles.scss (354 bytes)  
UPDATE src/polyfills.ts (2567 bytes)  
macbookpro@iMacdeMcBookPro crm %
```

Doc : <https://ng-bootstrap.github.io/#/home>



Package.json modifié

```
"dependencies": {  
    "@angular/animations": "~12.0.0",  
    "@angular/common": "~12.0.0",  
    "@angular/compiler": "~12.0.0",  
    "@angular/core": "~12.0.0",  
    "@angular/forms": "~12.0.0",  
    "@angular/localize": "~12.0.0",  
    "@angular/platform-browser": "~12.0.0",  
    "@angular/platform-browser-dynamic": "~12.0.0",  
    "@angular/router": "~12.0.0",  
    "@ng-bootstrap/ng-bootstrap": "^10.0.0",  
    "bootstrap": "^4.5.0",  
    "rxjs": "~6.6.0",  
    "tslib": "^2.1.0",  
    "zone.js": "~0.11.4"
```



App.module.ts modifié

- NgbModule n'est pas utilisé dans app.component

```
import { NgbModule } from '@ng-bootstrap/ng-bootstrap';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { CoreModule } from './core/core.module';

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



styles.scss modifié

- On en profite pour importer thème et fonts, nos 2 fichiers qui contiennent du css brut (variables) afin qu'elles soient ajoutées au début de notre fichier css compilé
- On importera layout uniquement dans les fichiers scss où l'on voudra utiliser les variables scss qu'il contient

```
@import "~bootstrap/scss/bootstrap";  
@import "./stylings/fonts";  
@import "./stylings/theme";
```



polyfills.ts modifié

- Cet import est nécessaire au bon fonctionnement de ng-bootstrap

```
import '@angular/localize/init';
```



Ce qu'il faut retenir

- `ng add` existe depuis Angular 6 et permet à `angular cli` de faire appel à des `schematics` externes
- Pensez à importer vos variables css sur `styles.scss` après l'import de `bootstrap` pour le cas où vous utiliseriez des noms de variables identiques à celles de `bootstrap` (auquel cas vous surchargez les variables `bootstrap`)



Conseils pratiques

- Privilégier ng add au lieu de faire des npm install pour bootstrap puis npm install pour ng-bootstrap
- Faites attention à la version de bootstrap utilisée par ng-bootstrap, vous devez utiliser la même version de bootstrap quand vous récupérez des composants css sur le site de bootstrap



Commit 5 : **ng-bootstrap**

- Retrouvez sur le dépôt Github de la formation le commit 5 concernant l'installation de bootstrap & ng bootstrap

6

**UiComponent
(html et scss uniquement)**





Le **html et scss**

- Dans le dossier UI, on crée un dossier components et on génère notre component ui
 - **Ng g component ui --export**
- On récupère la partie html de UiComponent
- On récupère la partie scss de UiComponent
- Pensez à ajouter UiComponent dans les exports de UiModule pour autoriser son utilisation à l'extérieur de UiModule. Ici nous voulons l'instancier dans AppModule.

HTML



SCSS



HTML

```
<header class="fixed-top">
    Header
</header>
<main>
    <div class="shadow position-fixed">
        </div>
        <aside>
            <div class="scroller">
                Nav
            </div>
        </aside>
        <div class="page">
            <div class="contents">
                Content
            </div>
            <footer>
                Footer
            </footer>
        </div>
    </main>
```

```
@NgModule({  
  declarations: [  
    UiComponent  
,  
  imports: [  
    CommonModule  
,  
  exports: [  
    UiComponent  
,  
  ]  
})
```

**Besoin
d'exporter le
composant
depuis
UiModule**

```
@NgModule({  
  declarations: [],  
  imports: [  
    CommonModule  
,  
    exports : [  
      UiModule,  
      TemplatesModule,  
      IconsModule,  
    ]  
  ]  
})  
export class CoreModule {}
```

```
  ,  
  imports: [  
    BrowserModule,  
    AppRoutingModule,  
    CoreModule  
  ],
```

CoreModule
 exporte
 UiModule

AppModule
 importe
 CoreModule



Ce qu'il faut retenir

- Dans les déclarations d'un @NgModule, on met des components, des pipes, des directives
- Dans les imports d'un @NgModule, on ne peut ajouter que des modules
- Dans les exports d'un @NgModule, on ajoute des components, pipes, directives ou modules



Commit 6 : **UiComponent html / scss**

- Retrouvez sur le dépôt Github de la formation le commit 6 concernant la création de UiComponent partie html / css

Challenge Fin jour 2

Créer un nouveau projet challenge2

Installer Bootstrap avec la commande ng add

Créer

- module core;
- module home;
- module shop;
- module shared

Core instancie home, shop et shared

Dans home, créer pages > page-homepage avec un composant accueil (avec uniquement le texte par défaut)

Dans shop, créer pages > page-add-product

Dans shared, créer composant nav avec menu de navigation: accueil/ contact/ panier.

Afficher nav dans app-root

7

Header, Nav et Footer





Créer 3 components header, nav & footer

- On crée un dossier components dans notre **coreModule** :
 - Ng g component header --export
 - Ng g component nav --export
 - Ng g component footer --export

```
@NgModule({
  declarations: [
    HeaderComponent,
    NavComponent,
    FooterComponent
  ],
  imports: [
    CommonModule
  ],
  exports : [
    HeaderComponent,
    NavComponent,
    FooterComponent,
    UiModule,
    TemplatesModule,
    IconsModule,
  ]
})
export class CoreModule { }
```



impossibilité d'utiliser ces 3 components dans uiComponent ?

- Il est impossible d'utiliser ces 3 components dans UiComponent mais pourquoi ?
 - Premièrement car l'import de CoreModule dans UiModule créé un maximum call stack avec ces deux class qui s'auto instancient en boucle
 - Secondement car on veut un component Ui réutilisable dans une autre appli donc ils n'ont rien à faire dans UiComponent
 - Et enfin parce que le style guide angular préconise d'importer le CoreModule qu'une seule et unique fois sur AppModule

```
<app-ui>
    <!-- pas visible -->
    <!-- ici du texte -->
</app-ui>
```



Ajouter header, nav et footer dans app-component.html

- Dans le DOM, header, nav et footer component n'apparaissent pas car seul le code html qui se trouve sur ui.component.html est rendu à l'intérieur du host element

```
<app-ui></app-ui>
```

```
<app-ui>
  <app-header></app-header>
  <app-nav></app-nav>
  <app-footer></app-footer>
</app-ui>
```

The screenshot shows a code editor with two tabs: 'app.component.html' and 'ui.component.html'. The 'app.component.html' tab shows the overall structure:

```
src > app > ui > pages > ui > ui.component.html > ⚒
  1   <header class="fixed-top">
  2     header
  3     <ng-content></ng-content>
  4
  5   </header>
```

The 'ui.component.html' tab shows the content of the 'ui' page:

```
<header class="fixed-top">
  header
  <ng-content></ng-content>
</header>
```



Transclusion simple avec ng-content

- Dans **ui.component.html** on insère plusieurs directives `<ng-content>` pour projeter plusieurs web Components à différents endroits
- Ng-content n'est jamais présente dans le DOM, elle est remplacée par l'élément que vous projetez
- Si vous mettez plusieurs directives `<ng-content>`, seule la dernière est utilisée pour projeter tout ce qui se trouve (html ou components) entre `<app-ui></app-ui>`



Transclusion multiple avec ng-content select="css selector"

- Les projections sont faites correctement et vous pouvez le constater sur la page web de votre navigateur
- Pas très réutilisable car n'autorise que la projection d'éléments qui s'appellent app-header, app-nav et app-footer
- Si dans une autre application je souhaite réutiliser UiComponent et projeter autre chose, cela ne fonctionnera pas

```
<!-- projection (transclusion) de HeaderComponent ici -->
<ng-content select="app-header"></ng-content>

<!-- projection (transclusion) de navComponent ici -->
<ng-content select="app-nav"></ng-content>

<!-- projection (transclusion) de footerComponent ici -->
<ng-content select="app-footer"></ng-content>
```



Transclusion multiple avec ng-content select="css selector"

- Ainsi, vous pourrez réutiliser ui component dans toutes vos appli et projeter tout ce que vous voulez en utilisant les bons noms de class

Ui.component.html

```
<!-- projection (transclusion) de headerComponent ici -->  
<ng-content select=".tr-header"></ng-content>
```

App.component.html

```
<!-- projeter le template header.component.html -->  
<app-header class="tr-header"></app-header>
```



Ce qu'il faut retenir

- Ng content n'est jamais rendue dans le DOM, elle est remplacée par le ou les éléments que vous projetez
- Pour la transclusion multiple, la propriété select sur ng-content est obligatoire car si vous utilisez plusieurs directives ng-content sans select, seule la dernière va être utilisée pour la projection
- Si vous utilisez plusieurs éléments ng-content avec un select identique, seule la première sera utilisée pour la projection



Conseils pratiques

- Privilégiez les sélecteurs de class pour vos directives ng-content avec select (transclusions multiples)
- Définissez une règle de nommage pour ces noms de class (ex préfixe tr pour transclusion)



Commit 7 : la transclusion

- Retrouvez sur le dépôt Github de la formation le commit 7 concernant l'ajout des components header, nav et footer ainsi que la simple / multiple transclusion



Travaux pratiques

TP :

- Créer 1 web component ui2Component avec 4 blocs qui s'affichent les uns en dessous des autres dans lesquels vous projetez header, nav, un div `class="tr-content"` et footer



Correction TP (ui2)

- Dans **uiModule/components** :
 - **ng g component Ui2 --export**
- On crée 4 blocs contenant nos 4 ng-content pour la transclusion
 - **ui2Component.html** :

```
<header>
  <ng-content select=".tr-header"></ng-content>
</header>
<main>
  <div> <ng-content select=".tr-nav"></ng-content> </div>
  <div> <ng-content select=".tr-content"></ng-content> </div>
  <footer> <ng-content select=".tr-footer"></ng-content> </footer>
</main>
```

```
<div class="myflex">
  <header>
    <!-- *ngIf="!open" -->
    <span (click)='toggle()'>
      <!-- Ici on déplace le ng-content dans les conditions
          pour éviter la projection
      -->
      <div *ngIf="open; else elseBlock" select=".tr-btn-close"></div>

      <ng-template #elseBlock>
        <ng-content select=".tr-btn-open"></ng-content>
      </ng-template>
    </span>

    <!-- header -->
    <!-- contenu projeté -->
    <!-- transclusion -->
    <ng-content select=".tr-header"></ng-content>
  </header>
```

8

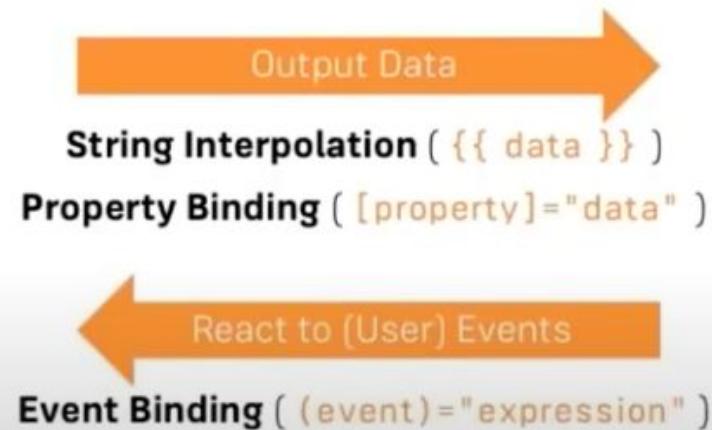
Directive d'attribut NgClass + Event Binding



Communication à l'intérieur d'un même composant

Databinding = Communication

TypeScript Code
(Business Logic)



Template (HTML)



Btn nav méthode toggle()

- On veut créer un bouton qui va ouvrir et fermer notre menu de navigation : voir démo
- On va donc devoir écrire une méthode `public toggle(): void` dans laquelle on passe une variable de type `boolean` de true à false
 - **ui.component.ts :**

```
public open : boolean = true;

public toggle():void {
    this.open = !this.open;
```

- On veut ensuite pouvoir utiliser cette méthode lorsque l'on clique sur un bouton. Il existe une solution et celle-ci est un event `(click)=""` que l'on va bind à notre méthode



Btn nav event binding

- On veut ensuite pouvoir utiliser cette méthode lorsque l'on clique sur un bouton. Il existe une solution et celle-ci est un event `(click)=""` que l'on va bind(relier) à notre méthode
 - **Ui.component.html (au-dessus de ng-content de header) :**

```
<button (click)="toggle()">  
  Open/close  
</button>  
<ng-content select=".tr-header">...
```

- On se rend compte qu'il nous faut une solution pour pouvoir donner/retirer dynamiquement une class css à notre nav pour pouvoir fermer celui-ci. La solution est la directive `[ngClass]=""` qui nous permet justement de faire cela.

<https://angular.io/api/common/NgClass>

- **ui.component.html :**

- `Object` - keys are CSS classes that get added when the expression given in the value evaluates to a truthy value, otherwise they are removed.

```
<main [ngClass]="{ 'close-aside' : open }">
```

Utiliser l'opérateur ternaire

```
<div [ngClass]="varA === varB ? 'css-class-1' : 'css-class-2'">
```



Btn nav UI ready to (re?)use

- Tout fonctionne, seulement notre ui n'est pas totalement réutilisable. En effet on se rend compte que nous obligeons l'utilisation d'une balise `<button>` et c'est dommage
- il serait préférable de laisser l'utilisateur choisir ce qu'il souhaite projeter (comme par exemple une icône) à la place de notre bouton et que tout cela fonctionne
- Utilisons donc la transclusion vu que nous avons appris à l'utiliser précédemment :

- **ui.component.html :**

```
<ng-content (click)="toggle()" select=".tr-btn"></ng-content>
```

- **app.component.html :**

```
<button class="tr-btn">open/close</button>
```



Btn nav UI ready to (re)use

- On se rend compte que notre méthode `public toggle():void` n'est plus appelée et que notre `(click)=""` ne fonctionne plus. C'est dû au fait que notre balise `<ng-content>` n'est pas rendue dans le DOM. Trouvons une solution
- Utilisons une balise `` qui englobe notre `<ng-content>` pour y insérer notre eventBinding.
 - **ui.component.html :**

```
<span (click)="toggle()">
  <ng-content select=".tr-btn"></ng-content>
</span>
```

- En effet la balise `` reste présente dans le DOM et cela fonctionne.



Ce qu'il faut retenir

- Une directive est une class avec le décorateur @Directive
- Les directives d'attribut permettent de manipuler les attributs d'éléments html
- La directive NgClass permet d'ajouter ou de supprimer une class sur un élément du DOM
- Ne mettez pas d'event sur un ng-content car il n'apparaît pas dans le DOM donc pensez à mettre votre event sur un élément qui reste présent dans le DOM
- La syntaxe pour l'eventBinding est (event)="fx()"



Commit 8 : **ngclass/event binding**

- Retrouvez sur le dépôt Github de la formation le commit 8 concernant la correction du tp ui2, la directive d'attribut ngClass ainsi que l'event binding (event)="fx()"

Différence entre class binding et ngClass

<https://jscurious.com/ngclass-vs-class-binding-in-angular/>

9

**IconsModule, icons
components & FontAwesome**





Avantages d'un module icons

Les avantages à utiliser un module pour nos icons sont nombreux :

- réunis au même endroit (+ maintenance)
- préfixer le nom de nos components icon (+ lisibilité)
- limiter les imports de FontAwesome (+ pratique)
- On limite la duplication de code (+ lisibilité)
- seulement un import à modifier sur IconsModule, le ts et le html pour changer de police en cours de projet (+ Agilité)
- modifier les icons components impacte toutes les instances de nos components icons et on ne risque pas d'en oublier (- regression)
- réutiliser nos icons pour autre app ⇒ c/c le dossier Icons (+ Réutilisabilité)



Générer nos **components icons**

- On Crée un dossier components dans IconsModule
 - `ng g component icon-nav --export`
 - `ng g component icon-close --export`
 - `ng g component icon-edit --export`
 - `ng g component icon-delete --export`



Font Awesome pour Angular

- Font Awesome propose une librairie pour Angular et même un schematics pour l'installation
- <https://github.com/FortAwesome/angular-fontawesome>
 - **ng add @fortawesome/angular-fontawesome@0.10.0**
 - On sélectionne free solid icons et free regular icons
- Les icons que nous allons utiliser pour nos components :
 - Icon-nav : faBars | Icon-close : faTimes
 - Icon-edit : faEdit
 - Icon-delete : faTrash

Compatibility table

@fortawesome/angular-fontawesome	Angular	Font Awesome	ng-add
0.1.x	5.x	5.x	not supported
0.2.x	6.x	5.x	not supported
0.3.x	6.x && 7.x	5.x	not supported
0.4.x, 0.5.x	8.x	5.x	not supported
0.6.x	9.x	5.x	supported
0.7.x	10.x	5.x	supported
0.8.x	11.x	5.x	supported
0.9.x	12.x	5.x	supported
0.10.x	13.x	5.x && 6.x	supported

Erreur

Pensez à redémarrer l'appli
car package.json a été
modifié.



Package.json & appModule modifiés

- **package.json :**

```
"@fortawesome/angular-fontawesome": "^0.9.0",
"@fortawesome/fontawesome-svg-core": "^1.2.35",
"@fortawesome/free-regular-svg-icons": "^5.15.3",
"@fortawesome/free-solid-svg-icons": "^5.15.3",
```

- **appModule :**

```
imports: [
  BrowserModule,
  AppRoutingModule,
  CoreModule,
  FontAwesomeModule
]
```



Imports de FontAwesomeModule

- **App Module : on retire l'import de FontAwesomeModule**

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

- **IconsModule : on ajoute l'import de FontAwesomeModule**

```
import { FontAwesomeModule } from '@fortawesome/angular-fontawesome';  
@NgModule({  
  declarations: [IconNavComponent, IconCloseComponent, IconEditComponent],  
  imports: [CommonModule, FontAwesomeModule],  
  exports: [IconNavComponent, IconCloseComponent, IconEditComponent],})  
export class IconsModule {}
```



TypeScript de nos icons components

- Dans la documentation fontAwesome il est écrit que l'on doit initialiser une variable et l'affecter à une icône puis de l'import depuis leur librairie

```
import { Component, OnInit } from '@angular/core';
import { faTimes } from '@fortawesome/free-solid-svg-icons';

@Component({
  selector: 'app-icon-close',
  templateUrl: './icon-close.component.html',
  styleUrls: ['./icon-close.component.scss'])
export class IconCloseComponent implements OnInit {
  public myIcon = faTimes;
  constructor() {}

  ngOnInit(): void {}
}
```

- On c/c sur le ts de icon-nav, icon-delete et icon-edit et on change l'icône



icones

- close : import {faTimes} from '@fortawesome/free-solid-svg-icons';
- delete : import { faTrash } from '@fortawesome/free-solid-svg-icons';
- edit : import { faEdit} from '@fortawesome/free-solid-svg-icons';
- nav : import { faBars} from '@fortawesome/free-solid-svg-icons';



Html de nos icons components

- Dans la documentation fontAwesome il est écrit que l'on doit, dans notre balise `<fa-icon>`, faire un property binding `[icon]` avec notre variable `myIcon`

```
<fa-icon [icon]="myIcon"></fa-icon>
```

- On c/c sur le html de icon-nav, icon-delete et icon-edit sans modifications vu qu'on donne le même nom à notre variable



Scss de nos icons components

- On ajoute juste un peu de css sur nos icons et nos components sont prêts à l'emploi (dans chaque fichier scss de chaque composant)

```
fa-icon {  
  cursor: pointer;  
  margin-right: 5px;  
}
```

- On c/c sur le css de icon-nav, icon-delete et icon-edit



Utilisons nos icons

- Dans app.component.html : on remplace notre bouton par une instance de notre IconCloseComponent et avec la transclusion il va s'intégrer parfaitement

```
<app-icon-close class="tr-btn"></app-icon-close>
```



Ce qu'il faut retenir

- Créer un module et des components pour nos icônes limite la régression, aide à la maintenabilité et la réutilisabilité de nos icônes tout en permettant une méthode de travail orientée “Agile”
- Pensez à bien exporter vos components icon (--export) pour pouvoir les utiliser à l'extérieur d'IconsModule



Conseils pratiques

- Pré-fixez le nom des icônes
- Ng add pour installer fontAwesome est un schematics externe qui s'occupe pour nous d'installer la librairie, les polices d'icons et d'ajouter du code dans notre application
- Evitez de créer un template et un fichier scss générique mutualisé par toutes les class IconNavComponent, IconDeleteComponent, IconEditComponent et IconCloseComponent



Commit 9 : icons components et FontAwesome

- Retrouvez sur le dépôt Github de la formation le commit 9 concernant iconComponent et fontAwesome

10

NglIf & ng-template





ngIf else

- On veut que notre icône change si notre menu nav est ouvert ou fermé, pour ce faire on va utiliser la directive `*ngIf=""` qui nous permet d'avoir des conditions dans notre html

- ui.component.html :**

```
<span (click)="toggle()">
  <ng-content *ngIf="open; else elseBlock"
    select=".tr-icon-open"></ng-content>
  </span>
  <ng-template #elseBlock>
    <ng-content select=".tr-icon-close"></ng-content>
  </ng-template>
```

- app.component.html :**

```
<app-icon-close class="tr-icon-close"></app-icon-close>
<app-icon-nav class="tr-icon-open"></app-icon-nav>
```

```
<!-- besoin d'un élément dans le template pour pouvoir utiliser l'directive -->
<span (click)="toggle()">
  <ng-content
    *ngIf="open"
    select=".tr-icon-close"></ng-content>

  <ng-content
    *ngIf="!open"
    select=".tr-icon-nav"></ng-content>
</span>
```

```
<!-- Besoin d'un élément dans le DOM qui soit

  <ng-content
    *ngIf="open; else elseBlock"
    select=".tr-icon-close"
  ></ng-content>

  <ng-template #elseBlock>
    <ng-content select=".tr-icon-nav">
      </ng-content>
    </ng-template>
</span>
```

*ngIf then else block

```
<!-- structure avec ngIf, then, else -->
<!-- ici on a deux ng-template -->
<!-- on peut utiliser une DIV ou ng-container -->
<div *ngIf="open; then thenBlock; else elseBlock" select=".tr-icon-close">
</div>
```



Ce qu'il faut retenir

- Ni ng-content, ni ng-template ne sont rendus dans le DOM
- La référence que vous mettez sur ng-template se fait à l'aide du symbole #
- Vous utilisez dans le else le nom de la référence que vous avez mis sur le ng-template



Conseils pratiques

- Le nom que vous choisissez pour donner une référence à ng-template ne doit pas être le même qu'une variable existante dans le ts
- Le nom de référence sur un ng-template doit être unique



Commit 10 : NgIf et ng-template

- Retrouvez sur le dépôt Github de la formation le commit 10 concernant les structures if...else avec ng-template

11

NglIf, ng-template & ng-container





Exercice

- Utilisez la structure *ngIf; then ... else avec 2 ng-template pour afficher app-icon-nav si le menu est fermé ou app-icon-close quand le menu est ouvert
- Bonus si vous utilisez ng-container



Correction

```
<span (click)="toggle()">
  <ng-container *ngIf="open; then thenBlock else elseBlock">
    </ng-container>
</span>
<ng-template #thenBlock>
  <ng-content select=".tr-icon-open"></ng-content>
</ng-template>
<ng-template #elseBlock>
  <ng-content select=".tr-icon-close"></ng-content>
</ng-template>
```



Conseils pratiques

- Quand vous avez juste besoin d'ajouter une directive de structure sur un élément html qui n'a pas d'intérêt à être dans le DOM, privilégiez ng-container



Commit 11 : NgIf, ng-template & ng-container

- Retrouvez sur le dépôt Github de la formation le commit 11 concernant les structures if...then...else avec ng-template & ng-container

12

Ui2 agilité





Ui2 pour l'agilité

- On va créer une 2ème Ui pour cause de changements en cours de projet. Vu que nos components sont indépendants et que nous travaillons avec une méthode vraiment agile, on va pouvoir répondre facilement à la demande



Travaux pratiques

- TP :
 - Créer 1 component ui2Component avec 4 blocs qui s'affichent les uns en dessous des autres dans lesquels vous projetez header, nav, un div `class="tr-content"` et footer



Correction TP

- Dans **uiModule/components :**
 - **ng g component Ui2 --export**
- On crée 4 blocs contenant nos 4 ng-content pour la transclusion
 - **ui2Component.html :**

```
<header>
  <ng-content select=".tr-header"></ng-content>
</header>
<main>
  <div> <ng-content select=".tr-nav"></ng-content> </div>
  <div> <ng-content select=".tr-content"></ng-content> </div>
  <footer> <ng-content select=".tr-footer"></ng-content> </footer>
</main>
```



Une ui avancée

- On va rajouter un peu de code se trouvant dans notre ui de base pour étoffer un peu notre ui

html

Ts

scss



- On s'aperçoit qu'en changeant simplement sur notre app.component.html le selector app-ui par app-ui2 on change complètement notre ui en conservant le reste



Ce qu'il faut retenir

- Les transclusions avec ng-content facilitent la conception d'applications agile



Conseils pratiques

- Mettez vous d'accord sur les conventions de nommage pour les noms de class que vous créez pour les transclusions
- Vous pourrez réutiliser les même noms pour les transclusions si vous changez de user interface
- Gardez le module ui le plus indépendant possible de tout autre module



Commit 12 : ui2 & agilité

- Retrouvez sur le dépôt Github de la formation le commit 12 concernant notre ui2 et l'agilité au sein de notre projet

13

Router Angular



Le Router



Suivant l'url demandée par l'utilisateur, Angular va remplacer Accueil par d'autres composants.
La nav reste en place si elle ne doit pas être modifiée.
Uniquement la partie à modifier est modifiée.



Le router angular

- Le router est une class de l'api angular contenant des variables, des méthodes etc.. comme toute autre class du framework. Initialisée au démarrage de l'application.
- Comment fonctionne-t-il : Suivant les paths dans l'url on va demander au router d'instancier ou de détruire un ensemble de webcomponents qui constituent bout à bout les vues de notre application
- On met tous ces paths dans une config du router qui est en fait un tableau d'objets routes et tous ces objets constituent la config de l'application
- Pour que ça fonctionne on a besoin **d'une seule instance de routerModule** dans toute notre appli et c'est pour ça qu'on va utiliser `forRoot` dans notre premier fichier de routing et `forChild` pour les suivants. Ca va nous permettre de garder une seule et unique instance de routerModule associée à un config de routes qui viendront s'ajouter dans l'ordre des imports des fichiers de routing



Le router angular

- Nous pouvons avoir plusieurs routers dans l'application
- Par défaut, quand nous ne fournissons pas de nom pour le router, celui-ci devient principal.
- Il n'y a qu'un seul router principal.
- Nous pouvons définir plus d'une router avec un **nom** : (named router outlet)

```
<router-outlet></router-outlet>
<router-outlet name='left'></router-outlet>
<router-outlet name='right'></router-outlet>
```

Access :

<https://angular-ivy-5qkx9k.stackblitz.io/> (<primary-route>(<routerOutletName>:<secondaryPath>))

Doc : <https://angular.io/api/router/RouterOutlet>

En général, d'autres solution existent, autre que les routers multiples

forRoot / forChild

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

On instancie une seule fois RouterModule.

.forChild permet de travailler sur la même instance de RouterModule.

Cela permet d'ajouter des routes avec .forChild dans chaque module.



Basic routes

- **app-routing.module.ts :**

```
import { PageSignInComponent } from './...'
const routes: Routes = [
  { path: '', redirectTo: '/sign-in', pathMatch: 'full' },
  { path: 'sign-in', component: PageSignInComponent }
];
```

Le slash permet de supprimer les autres éléments de la route et de les remplacer par ce string

Cette route est uniquement la route vide sans rien après. (par exemple /sign-in infos)

- **App-component.html :**

```
<router-outlet class="tr-content"></router-outlet>
```

- **<router-outlet>** est une directive utilisée par le router comme point de repère, elle reste présente dans le DOM

- **Ui-component.html :**

```
<div class="contents">
  <!-- contenu -->
  <ng-content select=".tr-content"></ng-content>
</div>
```

Le router ajoute dynamiquement des web components correspondant à notre route actuelle sous notre directive



Exercice

- En suivant l'exemple fait pour la route sign-in , faire les routes de :
 - **Sign-up**
 - **Resetpassword**
 - **Forgotpassword**



Correction

- Il nous suffit simplement d'écrire les routes sign-up, reset-password & forgot-password en calquant l'exemple de sign-in plus haut

```
{ path: 'sign-up', component: PageSignUpComponent },  
{ path: 'reset', component: PageResetPasswordComponent },  
{ path: 'forgot', component: PageForgotPasswordComponent }
```



Ce qu'il faut retenir

- Les méthodes `forRoot()` et `forChild()` servent à ne pas dupliquer les instances de Router et de toujours garder une seule configuration & (ordre des routes se fait dans l'ordre des imports)
- En fonction des path présents dans l'url on va pouvoir demander au routeur d'instancier ou de détruire un ensemble de components qui constituent la vue de notre application
- Ne créez pas de routes au chargement de l'application pour des components déclarés dans des modules qui ne sont pas importés au démarrage de l'appli (modules chargés ultérieurement) sinon il importera vos components au démarrage alors que vous souhaitez qu'ils soient chargés plus tard (lorsque le router aura chargé le module qui les déclare)



Conseils pratiques

- Testez toujours que vos routes fonctionnent dans le navigateur avant d'écrire trop de code, vous gagnerez du temps sur le déboggage



Commit 13 : Router angular

- Retrouvez sur le dépôt Github de la formation le commit 13 concernant le router angular & les routes basiques de notre application

14

lazy loading





Basic routes & lazy loading

- On va charger des modules en lazy loading, mais qu'est ce que c'est ?
Documentation : <https://angular.io/guide/lazy-loading-ngmodules>
- Le lazy loading consiste à demander au Router Angular de charger **des modules (et non pas des composants)** après le démarrage de l'appli uniquement lorsque l'utilisateur navigue dans l'application
- C'est très important pour les performances de l'application car seuls les modules indispensables au démarrage sont chargés et ensuite les modules non essentiels au démarrage sont chargés
- On va créer la route pour le lazy loading d'ordersModule et vérifier dans la partie Network (JS) de l'inspecteur d'éléments Chrome

```
{  
  path: 'orders',  
  loadChildren: () => import('./orders/orders.module').then(m =>  
    m.OrdersModule)  
},
```



Basic routes & lazy loading

- Vous pouvez vérifier en demandant l'url /orders.
- Dans Network > choisissez les fichiers JS
- Le fichier js du module apparaît uniquement quand vous visitez la page /orders

The screenshot shows the Network tab in the Chrome DevTools developer tools. The tab is labeled 'Network' and has several filter options: 'Preserve log', 'Disable cache', 'No throttling', and a speed dial icon. Below the tabs are filters for 'Invert', 'Hide data URLs', and categories: All, Fetch/XHR, JS, CSS, Img, Media, and Font. The main table lists network requests with columns for Name and Status. A red box highlights the last row, which corresponds to the question in the slide.

Name	Status
runtime.js	200
polyfills.js	304
vendor.js	304
main.js	304
styles.js	304
src_app_orders_orders_module_ts.js	304



Exercice

- En prenant exemple sur la route que nous venons d'écrire pour le lazy loading d'OrdersModule, écrivez les routes pour le lazy loading de **ClientsModule** et **PageNotFoundModule**

```
{  
  path: 'orders',  
  loadChildren: () => import('./orders/orders.module').then(m =>  
    m.OrdersModule)  
},
```



Correction

- Path clients :

```
{  
  path: 'clients',  
  loadChildren: () =>  
    import('./clients/clients.module').then(m => m.ClientsModule)  
,
```

- Path pageNotFound :

```
{  
  path: '**',  
  loadChildren: () =>  
    import('./page-not-found/page-not-found.module')  
.then(m => m.PageNotFoundModule)  
,
```



Ce qu'il faut retenir

- Le lazy loading est très important pour les performances de l'appli
- Chaque module chargé en lazy loading est représenté par un fichier js
- Vérifiez dans le Network (js) que les modules sont bien chargés quand vous allez sur les routes correspondantes
- Le path: “**” de votre pageNotFound doit toujours se trouver en dernier dans la config de votre tableau de routes puisqu'il englobe tous les paths existants hormis ceux précisés au dessus de celui-ci (wildcard)



Conseils pratiques

- Testez toujours que vos routes fonctionnent dans le navigateur avant d'écrire trop de code, vous gagnerez du temps sur le déboggage

Au démarrage

AppRoutingModule.ts est
instancié

```
$ ng generate module orders --routing
```

Quand l'utilisateur
visite /orders

OrderRoutingModule est
instancié (lazyloading)
Les composants qui sont
déclarés dans les routes
sont aussi instanciés.



Commit 14 : router angular & lazy loading

- Retrouvez sur le dépôt Github de la formation le commit 14 concernant le lazy loading

15

Preloading strategy





Le Preloading de modules

- Vous pouvez modifier la stratégie de preloading des modules en passant un deuxième paramètre au RouterModule.forRoot()
Documentation : <https://angular.io/guide/lazy-loading-ngmodules>

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

```
imports: [
  RouterModule.forRoot(routes,
  { preloadingStrategy: PreloadAllModules }
)],
```

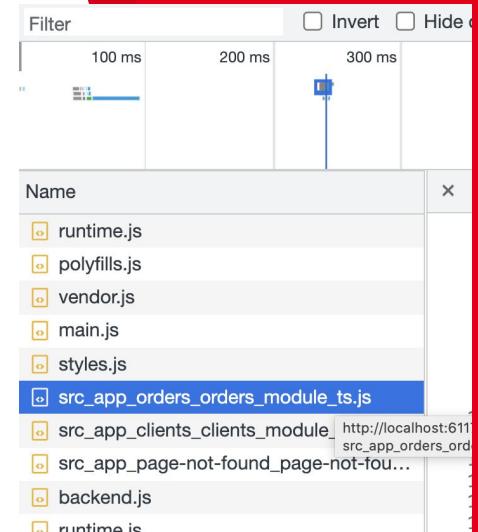
- Vous pouvez demander à Angular de précharger tous les modules qui doivent être chargés en lazy loading en tâche de fond immédiatement après le démarrage de l'appli et l'affichage de la home
- Vous pouvez si vous le souhaitez (nous ne le ferons pas ici) choisir de précharger certains modules chargés en lazy loading uniquement



Ce qu'il faut retenir

- Le preloadingStrategy:
PreloadAllModules n'impacte pas les performances au démarrage de l'appli
- Bénéfique pour la réactivité de vos pages quand vous naviguez sur votre appli car le Router a préchargé vos modules, il ne lui reste qu'à instancier des webcomponents pour les ajouter dans le DOM

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





Conseils pratiques

- Pensez toujours à bien vérifier le bon fonctionnement du preloadAllModules dans le Network (js)



Commit 15 : **preloading strategy**

- Retrouvez sur le dépôt Github de la formation le commit 15 concernant le preloading strategy

16

**Routes orders / clients
page not found**



```
@NgModule({
  imports: [RouterModule.forRoot(routes, {
    preloadingStrategy: PreloadAllModules
})],
  exports: [RouterModule]
})

export class AppRoutingModule {
  constructor(
    private router : Router
  ){
    console.log(this.router.config)
  }
}
```

La class Router est le service permettant la navigation.
Elle contient des propriétés et des méthodes.
On peut y avoir accès dans toute l'application.
La propriété config indique les routes déclarées.

```
▼ (7) [{} , {} , {} , {} , {} , {} , {}] ⓘ
▶ 0: {path: '' , redirectTo: '/sign-in' , path
▶ 1: {path: 'sign-in' , component: f}
▶ 2: {path: 'reset-password' , component: f}
▶ 3: {path: 'sign-in' , component: f}
▶ 4: {path: 'orders' , _loader$: Observable,
▶ 5: {path: 'clients' , _loader$: Observable,
▶ 6: {path: '**' , _loader$: Observable, _lo
length: 7
```

Nommage des fichiers :
orders.module.ts
orders-routing.module.ts

```
@NgModule({  
    declarations: [  
        PageListOrdersComponent,  
        PageAddOrderComponent,  
        PageEditOrderComponent  
    ],  
    imports: [  
        CommonModule,  
        OrdersRoutingModule  
    ]  
})  
export class OrdersModule { }
```

```
import { NgModule } from '@angular/core';  
import { RouterModule, Routes } from '@angular/ro  
import { PageListOrdersComponent } from './pages/  
  
const routes: Routes = [  
    {path: 'liste', component: PageListOrdersCompo  
];  
  
@NgModule({  
    imports: [RouterModule.forChild(routes)],  
    exports: [RouterModule]  
})  
export class OrdersRoutingModule { }
```

Ici, on pense à déclarer ce module dans les imports du module concerné

Ajouter une route vide après le premier path

TS orders-routing.module.ts ×

TS orders.module.ts

TS app-routing

src > app > orders > TS orders-routing.module.ts > ...

```
1 import { NgModule } from '@angular/core';
2 import { RouterModule, Routes } from '@angular/router';
3 import { PageListOrdersComponent } from './pages/page-li
4
5 const routes: Routes = [
6   || {path: '', component: PageListOrdersComponent}
7 ];
```



Routes de notre module OrdersRoutingModule

- Ecrire les routes du module Orders :
 - List
 - Add
 - edit

```
const routes: Routes = [
  {path: '', component: PageListOrdersComponent},
  {path: 'add', component: PageAddOrderComponent},
  {path: 'edit', component: PageEditOrderComponent},
];
```



Exercices

- En prenant exemple sur orders-routing.module.ts, écrire les routes pour :
 - **clients-routing.module.ts :**
 - PageListClientsComponent
 - PageAddClientComponent
 - PageEditClientComponent
 - **page-not-found-routing.module.ts :**
 - PageNotFoundComponent
 - Déplacer les routes sur **login-routing.module.ts** :
 - PageSignInComponent
 - PageSignUpComponent
 - PageresetPasswordComponent
 - PageforgotPasswordComponent

faire en sorte que cela fonctionne sans tomber sur pageNotFound



Correction 1/2

- **Clients-routing.module.ts :**

```
const routes: Routes = [
  {path: '', component: PageListClientsComponent},
  {path: 'add', component: PageAddClientComponent},
  {path: 'edit', component: PageEditClientComponent},
];
```

- **Page-not-found-routing.module.ts :**

```
const routes: Routes = [
  {path: '', component: PageNotFoundComponent},
];
```



Correction 2/2

- **Login-routing.module.ts :**

```
const routes: Routes = [
  {path: 'sign-in', component: PageSignInComponent},
  {path: 'sign-up', component: PageSignUpComponent},
  {path: 'reset', component: PageResetPasswordComponent},
  {path: 'forgot', component: PageForgotPasswordComponent},
];
```

- Il faut importer coreModule **avant** AppRoutingModule dans app.module.ts car comme expliqué un peu plus tôt les routes s'ajoutent à notre config dans l'ordre des imports et pour que tous nos paths soient situés avant pageNotFound, il faut procéder ainsi.
- **Attention à l'ordre dans lequel on déclare les routes !**
- **Attention également à bien exporter LoginModule depuis CoreModule**



Ce qu'il faut retenir

- Séparez vos routes sur des fichiers de routing à part pour chaque module qui contient des vues (pages) de l'application
- Vous aurez dans un seul dossier : vos components, le module qui les déclare et le module pour les routes concernées par cette partie de l'application (plus facile pour la maintenabilité et la réutilisabilité)



Conseils pratiques

- Sur chaque module de routing pensez à exporter RouterModule de telle sorte que le module associé qui importe votre module de routing importe en même temps RouterModule



Commit 16 : Routes orders clients page not found

- Retrouvez sur le dépôt Github de la formation le commit 16 concernant les routes orders / clients & pageNotFound

17

Navigation





Ajouter un menu de navigation

Dans `core>components>nav.component.html` je crée mon template :

```
<ul class="nav flex-column">
  <li class="nav-item">
    <a class="nav-link">Orders</a>
  </li>
  <li class="nav-item">
    <a class="nav-link">Clients</a>
  </li>
</ul>
```



Qu'est ce que **routerLink** & **routerLinkActive** ?

- RouterLink et RouterLinkActive sont deux directives angular
 - ***RouterLink** sert à faire des redirections vers des routes dans le html
 - ***RouterLinkActive** sert à ajouter ou retirer une classe en fonction de la route sur laquelle nous sommes
 - Attention de ne pas utiliser href qui rafraîchit la page
 - Besoin d'importer RouterModule



Navigation avec routerLink et routerLinkActive

- On va en premier lieu importer **RouterModule** dans core.module.ts pour pouvoir utiliser ces deux directives
- On va ensuite utiliser nos deux directives dans notre **nav.component.html** :

```
<a class="nav-link" routerLink="/orders" routerLinkActive="active">Orders</a>
<a class="nav-link" routerLink="/clients" routerLinkActive="active">Clients</a>
```

- Et pour finir on va écrire du css pour personnaliser notre class active dans **nav.component.scss** :

```
.active{
  background: var(--app-light);
}
```



Ce qu'il faut retenir

- Il est nécessaire d'importer RouterModule dans le module où vous souhaitez utiliser RouterLink & RouterLinkActive
- **RouterLink** sert à faire des redirections vers des routes dans le html
- **RouterLinkActive** sert à ajouter ou retirer une classe en fonction de la route sur laquelle nous sommes

```
nav.component.html          TS core.module.ts ×

src > app > core > TS core.module.ts > CoreModule
  ...
  9 |   import { RouterModule } from '@angular/router';
 10 |
 11 |
 12 | @NgModule({
 13 |   declarations: [
 14 |     HeaderComponent,
 15 |     NavComponent,
 16 |     FooterComponent
 17 |   ],
 18 |   imports: [
 19 |     CommonModule,
 20 |     RouterModule
 21 |   ],
 22 |   exports : [
 23 |     HeaderComponent,
 24 |     NavComponent,
 25 |     FooterComponent,
 26 |     UiModule,
 27 |     TemplatesModule,
 28 |     IconsModule
 29 |   ]

```



Conseils pratiques

- Utilisez le nom “active” pour la class de votre RouterLinkActive pour gagner en lisibilité et compréhension du code

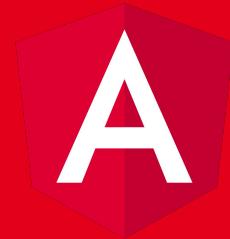


Commit 17 : Navigation

- Retrouvez sur le dépôt Github de la formation le commit 17 concernant la navigation et les directives RouterLink / RouterLinkActive

18

json server





Json server ?

- Json server est une fausse API côté front qui lorsqu'on va faire des appels Http va mettre à jour un fichier json
- On va donc l'installer:
 - **sudo npm install -g json-server**
- On crée ensuite un dossier api à la **racine de l'application** (en dehors de src)
- À l'intérieur on créer le fichier db.json
- Et enfin pour lancer json server:
 - **json-server --watch api/db.json**

```
"watch": "ng build --watch --configuration development",
"compodoc": "npx compodoc -p tsconfig.doc.json -o -s",
"test": "ng test",
"jsonServer" : "json-server --watch mock_db/db.json"
},  
Uninstalling
```

Autre outil mockoon
A ouvrir en tant que Administrateur

Create mock APIs in seconds

Mockoon is the easiest and quickest way to design and run mock REST APIs.

No remote deployment, no account required, **free** and **open-source**.

[Download !\[\]\(bfa5b70f8092543bcba8501dda59bb82_img.jpg\)](#)

[Documentation](#)



api/db.Json & script

- On ajoute un script sur **Packages.json** :

```
"api": "json-server --watch api/db.json"
```



- On copie le code ci-joint dans notre fichier db.json => **db.json** :



Ce qu'il faut retenir

- Json server est un émulateur de base de données et n'est pas une vraie base de données



Conseils pratiques

- Pensez à ajouter une ligne dans les scripts de votre package.json pour pouvoir lancer votre (fake) api plus rapidement (“npm run api”)



Commit 18 : json server

- Retrouvez sur le dépôt Github de la formation le commit 18 concernant l'installation de json server

19

Enums, interfaces, models





Enums, interfaces & models

- Les enums, interfaces et models ne possèdent pas de decorator et on n'a pas besoin de les déclarer dans `@NgModule` ni de les exporter (donc peu importe le répertoire, nos modules ne seront pas interdépendants)
- On va donc créer 3 dossiers dans **core module** :
 - Enums
 - Interfaces
 - models



Enum

- On va créer notre enum. Pour ce faire il suffit de taper la commande :
 - **ng g enum state-order**

State-order.ts :

```
export enum StateOrder {  
    CANCELLED = "CANCELLED",  
    OPTION = "OPTION",  
    CONFIRMED = "CONFIRMED"  
}
```



Interface

- On va créer notre interface. Pour ce faire il suffit de taper la commande :
 - **ng g interface order-i**

Order-i.ts :

```
import { StateOrder } from "../enums/state-order";

export interface OrderI {
  "tjmHt": number;
  "nbJours": number;
  "tva": number;
  "state": StateOrder;
  "typePresta": string;
  "client": string;
  "comment": string;
  "id": number;
}
```



Model

- On va créer notre model. Pour ce faire il suffit de taper la commande :
 - **ng g class order**

Order.ts :

```
export class Order implements OrderI {  
    tjmHt = 1200;  
    nbJours = 1;  
    tva = 20;  
    state = StateOrder.OPTION;  
    typePresta!: string;  
    client!: string;  
    comment!: string;  
    id!: number;  
  
    constructor(obj?: Partial<Order>) {  
        if(obj) Object.assign(this, obj);  
    } }
```

l'objectif est de passer uniquement certaines propriétés mais d'obtenir tout un objet conforme au model Order
new Order({tjmHt:1300, client:'Atos'})
utilisation de Object.assign(objCible, objSource)



Ce qu'il faut retenir

- En typescript on peut juste créer des enums qui retournent des types number ou string
- L'interface est comme un “contrat” qui est passé avec les développeurs quand ils travaillent sur les objets pour respecter l'orthographe, le type... De cette manière le code reste maintenable et cohérent
- En mode JS strict une propriété doit être initialisé et typée sinon JS considère que cette variable peut être de type string ou null
- Le point d'interrogation en TS permet de déclarer un paramètre optionnel



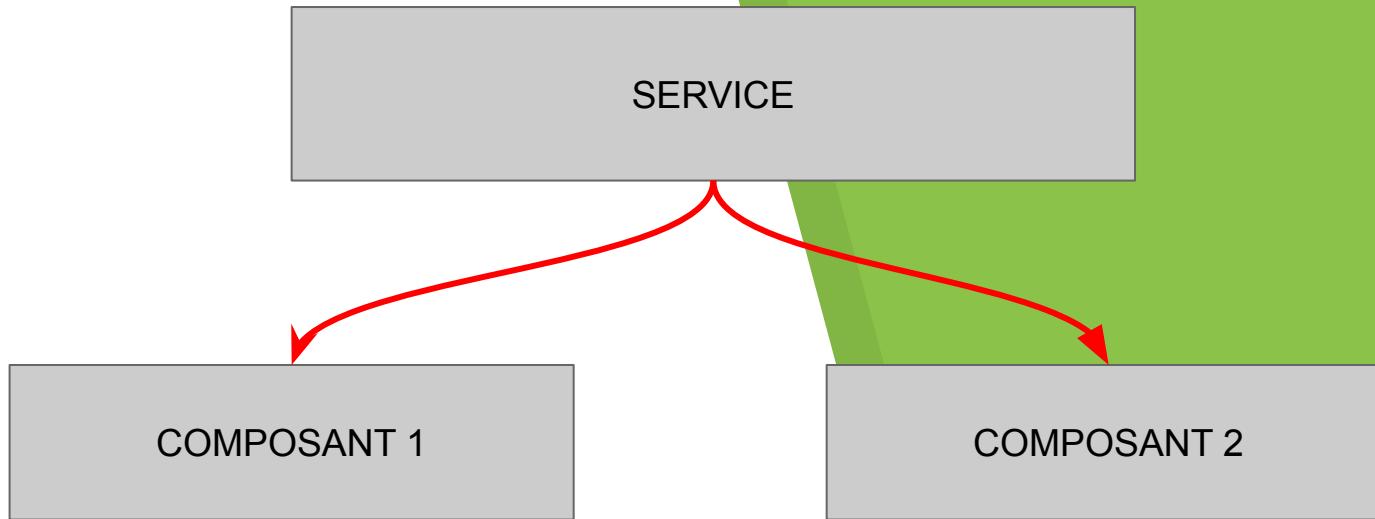
Commit 19 : Enums, interfaces & models

- Retrouvez sur le dépôt Github de la formation le commit 19 concernant les enums les interfaces et les models pour la partie orders

20

Services angular & appels http (Observable d'rxjs)

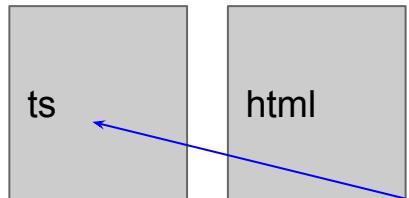






Appels http

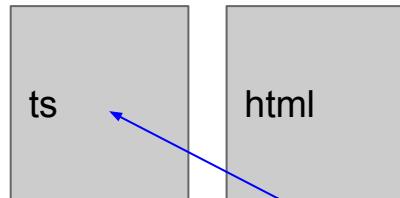
Page list orders



http.get(...)

http.delete(...)

Page edit orders



http.get(..., item.id)
http.put(...)

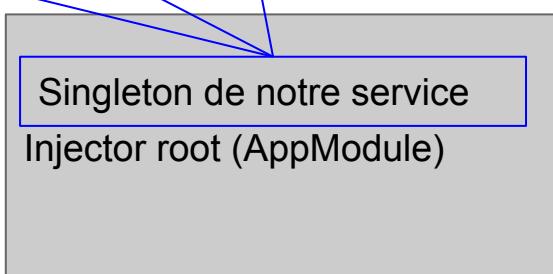
Page add orders



http.post(..., item)

Une Class @Injectable()
pour manager notre
collection (un service)

Get collection()
Set collection(...)
AddItem(item)
DeleteItem(id)
UpdateItem(item)
GetItemById(id)



Définition Singleton

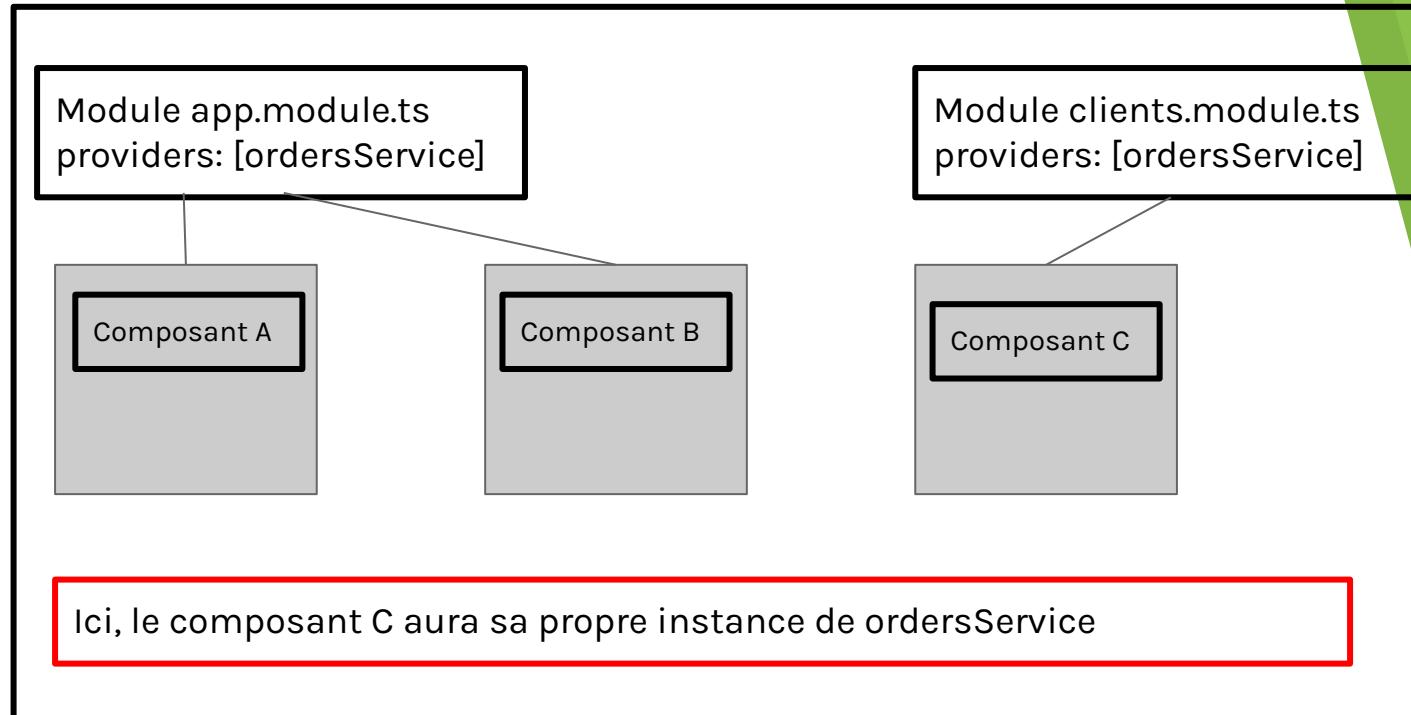
The singleton design pattern restricts the instantiation of a class to a single instance. This is done in order to provide coordinated access to a certain resource, throughout an entire software system. Through this design pattern, the singleton class ensures that it's only instantiated once, and can provide easy access to the single instance.



Créer notre service

- Créer un dossier services dans orders.module
 - **ng g service orders**
- C'est le décorator `@Injectable()` qui fait de cette classe un service (singleton)
Documentation : <https://angular.io/guide/dependency-injection>
- Ce qui permet à Angular dès le démarrage de l'application de mettre à disposition ce service dans le système d'injection au niveau root de l'application c'est `providedIn: 'root'`
- Placer tous les services au même endroit (cohérence).
Service qui gère la collection clients dans dossier client
Service qui gère la collection orders dans dossier orders
- Attention de ne pas utiliser le décorateur + providers en même temps

Organisation avec plusieurs instances d'un service





HttpClient

- On doit importer **HttpClientModule** sur **appModule**
Documentation : <https://angular.io/guide/http>

- **App.module.ts :**

```
import { HttpClientModule } from '@angular/common/http';

@NgModule({
  imports: [ HttpClientModule ]
```

- Ensuite on doit injecter **httpClient** pour faire des appels http dans notre service

- **orders.service.ts :**

```
import { HttpClient } from '@angular/common/http';
@Injectable({
  providedIn: 'root'
})
export class OrdersService {
  constructor(private http: HttpClient) {}
```



Premiers appels http

- On va créer une variable collection de type observable de tableau order et une variable urlApi qui récupère notre urlApi dans environment.ts

```
public collection$!: Observable<Order[]>;  
private urlApi = environment.urlApi;
```

- Dans le constructor on va initialiser notre collection en récupérant les orders de notre fake api grâce à la méthode `get` de httpService

```
constructor(private httpClient: HttpClient) {  
  this.collection = this.httpClient.get<Order[]>(`${this.urlApi}/orders`);  
}
```

Environnement - localhost

```
export const environment = {  
  production: false,  
  urlApi : 'http://www.localhost:3000'  
};
```

environments > **TS** environment.prod.ts > ...

```
export const environment = {  
  production: true,  
  urlApi : 'http://www.localhost:3001'  
};
```

Lister les actions à mener sur la collection orders

```
// get collection  
  
// set collection  
  
// edit state item  
  
// update item in collection (put ou patch)  
  
// add item in collection (post)  
  
// delete item in collection (delete)  
  
// get by id in collection
```

Utiliser getter et setter

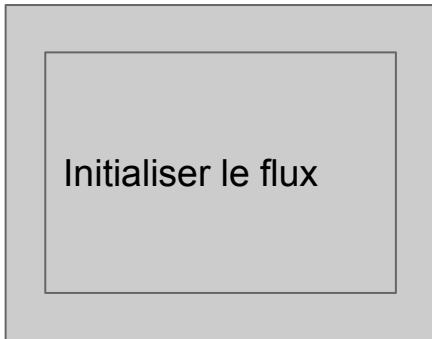
```
public getCollection(){}
public setCollection(){}
```

Au lieu de créer deux méthodes, on crée une seule propriété avec le préfixe get ou set. En fonction de son usage, JavaScript sait ce qu'il doit appeler



Observable

Observable



observable.subscribe()

observable.subscribe()

observable.subscribe()



Exemple Observable Rxjs

```
[  
 {  
   Name: 'chris',  
   Id: 1  
 },  
 {  
   Name: 'chris',  
   Id: 1  
 },  
 ]
```

```
('http://localhost:3000/orders')
```

On crée une variable Obj et on lui passe une fonction callback.

En paramètre, on passe les observers (listX).

La méthode next permet de passer le tableau json aux subscribers.

Service Angular

```
Obj = new Observable((listX)=> {  
   listX.next(tab.json),  
   listX.error(js),  
   listX.complete()  
)
```

```
this.http.get<Order []>
```

Component ts

```
Obj.subscribe(  
  (flux)=> {flux etant tab.json},  
  (e) => {js},  
  () => { executer mon propre js }  
)
```

```
// reste à faire sur  
page-list-orders
```



Injection de Orders Service dans page-list-orders

- On injecte notre service orders et on souscrit à notre collection
Ici on invoque le get collection()
 - **Page-list-orders.ts :**

```
constructor(private ordersService: OrdersService) {  
    this.ordersService.collection.subscribe();  
}
```

- On écrit un console.log() dans notre service pour vérifier qu'il est bien instancié
 - **Orders.service.ts :**

```
constructor(private http: HttpClient ) {  
    console.log( 'service orders instancied');  
    this.collection = this.http.get<Order[]>(`${this.urlApi}/orders`);  
}
```

Attention : tant qu'on n'injecte pas le service dans un composant, celui-ci n'est pas instancié (mais disponible dans l'application).

On vérifie que cela fonctionne en allant à l'adresse : <http://localhost:4200/orders>

La méthode httpClient.get récupère le flux de données de l'API et crée un nouvel observable pour nous.

Elle appelle systématiquement les trois méthodes (next, error, complete).

Elle passe la réponse json à la méthode .next

C'est au moment du subscribe
que l'observable est instancié et
donc l'appel HTTP.



Commit 20 : Services & appels http

- Retrouvez sur le dépôt Github de la formation le commit 20 concernant les Services angular et appels http (Observable d'rxjs)

21

Observable, Subject & BehaviorSubject



Lien vers démo

Observable froid :

<https://angular Ivy-h3kcfv.stackblitz.io>
<https://angular Ivy-k5zknz.stackblitz.io>

Observable chaud avec new Subject():

<https://angular Ivy-vjorcm.stackblitz.io>

Observable chaud avec BehaviorSubject(valeur):



Observable, Subject & BehaviorSubject

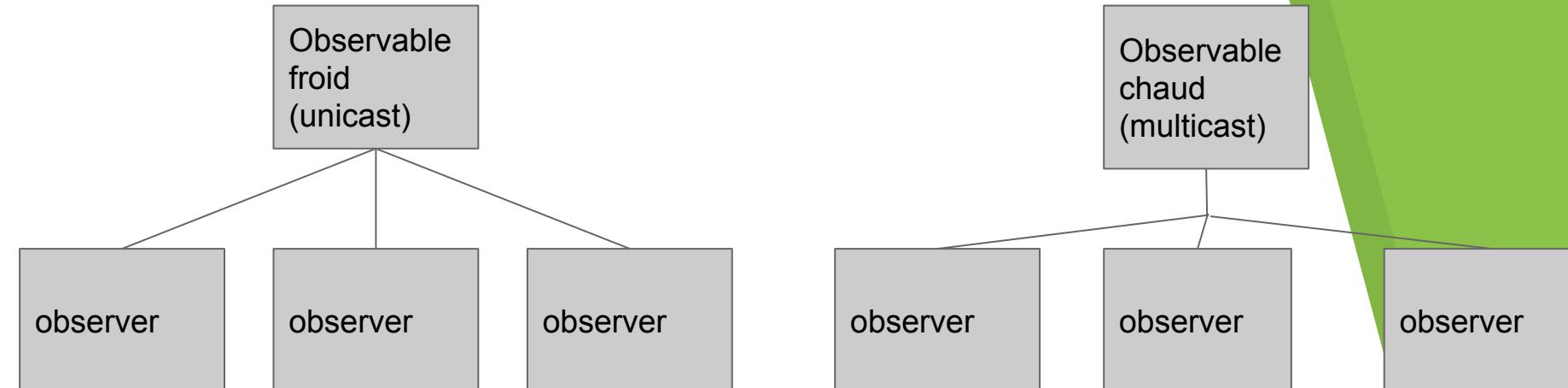
- RxJS fait partie intégrante d'Angular
- Tous les objets de la librairie RxJS sont des observables
- Le design pattern est le même (observer), c'est juste les comportements qui sont différents
- Les appels http retournent un observable
- Il y a deux types d'observables :
 - **Les observables froids** (unicast) plusieurs subscribe plusieurs instances du même flux de données
 - **Les observables chauds** (multicast) plusieurs subscribe toujours une seule et unique instance du même flux de données
- Doc. ReactiveX : <http://reactivex.io/> / Doc RxJS : <https://rxjs.dev/>



Observable chaud vs Observable froid

- **Observable Froid** : observable qui lorsqu'on y souscrit, nous retourne un flux de données. Si il y a une modification du flux de données on devra unsubscribe() puis subscribe() à nouveau pour récupérer le flux modifié
- **Observable Chaud** : Initialisation ou pas d'un flux de données et possibilité avec la méthode next de modifier le flux encore et encore tout en pouvant récupérer le flux modifié dans le subscribe()

Observables chauds et froids



Observable froid : plusieurs flux de données (instances) pour chacun des observers

Observable chaud : un seul flux de données (instance) pour tous les observers



Observable chaud vs Observable froid

Flux	Observable Froid	Obs .subscribe()	Flux de données retournées par itérations une seule fois	Modifie le flux	unsubscribe()	Obs .subscribe()
Flux	Observable Chaud	Obs .subscribe()	Flux de données retournées par itérations (si n'a pas été init pour récupérer un flux vide)	Modifie le flux	Modifie le flux	Modifie le flux



Exemples d'observables

- On initialise une variable `obs` de type Observable et on passe à notre méthode `.next()` un paramètre. Notre paramètre est un objet `Math` que l'on link à la méthode `.random()` qui nous retourne aléatoirement un chiffre entre 0 & 1

```
private obs = new Observable((listXsubscribe) => {  
    listXsubscribe.next(Math.random());  
});
```

- Ensuite on effectue quelques petits tests dans le constructor pour comprendre le fonctionnement des observables :

```
constructor() {  
    this.obs.subscribe((data) => console.log(data));  
    this.obs.subscribe((data) => console.log(data));  
}
```



Exemples de subjects

- On initialise une variable `subj` de type Subject à laquelle on affecte un `new Subject()`

```
private subj = new Subject();
```

- Ensuite on effectue quelques petits tests dans le constructor pour comprendre le fonctionnement des subjects :

```
constructor() {
  this.subj.subscribe((data) => console.log(data));
  this.subj.next('test');
  this.subj.next('test 2');
  this.subj.subscribe((data) => console.log(data));
  this.subj.next(Math.random());
  this.subj.subscribe((data) => console.log(data));
}
```



Exemples de behaviorSubjects

- On initialise une variable `behave` de type BehaviorSubject à laquelle on affecte un `new BehaviorSubject(0)`

```
private behave = new BehaviorSubject(0);
```

- Ensuite on effectue quelques petits tests dans le constructor pour comprendre le fonctionnement des behaviorSubjects :

```
constructor() {
  this.behave.subscribe((data) => console.log(data));
  this.behave.next(1);
  this.behave.subscribe((data) => console.log(data));
  this.behave.next(2);
  this.behave.next(Math.random());
  this.behave.subscribe((data) => console.log(data));
  this.behave.subscribe((data) => console.log(data));
}
```



Commit 21 : Observable, Subject & BehaviorSubject

- Retrouvez sur le dépôt Github de la formation le commit 21 concernant les observables, subjects et behaviorSubjects

22

Templates de page avec un title statique



titre

contenu page-list-orders

→ Modèle pleine largeur

```
<app-template-full-width>
  <!-- ici contenu de cette page est projeté
       à la place de ng-content -->
  <h1>contenu page-list-orders</h1>
</app-template-full-width>
```

titre

contenu page-list-orders

→ Modèle container

```
<app-template-container>
  <!-- ici contenu de cette page est projeté
       à la place de ng-content -->
  <h1>contenu page-list-orders</h1>
</app-template-container>
```



Components de templatesModule

- Dans templates module créer un dossier components
 - `ng g component template-full-width --export`
 - `ng g component template-container --export`
- On ajoute un peu de code à nos components :
 - **Template-full-width.component :**
 - **Template-container.component :**

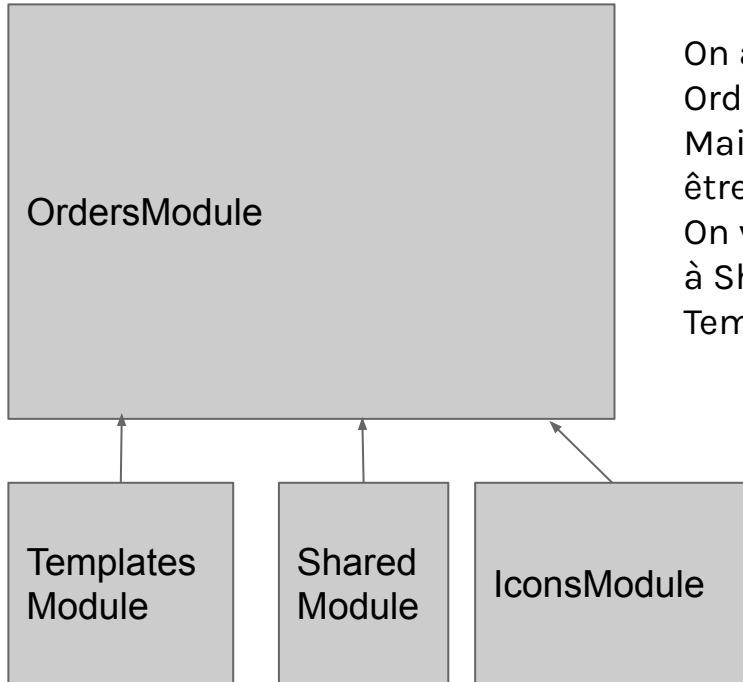
Html	Scss	Ts

Html	Scss	Ts

Initialiser les propriétés en mode strict de JavaScript

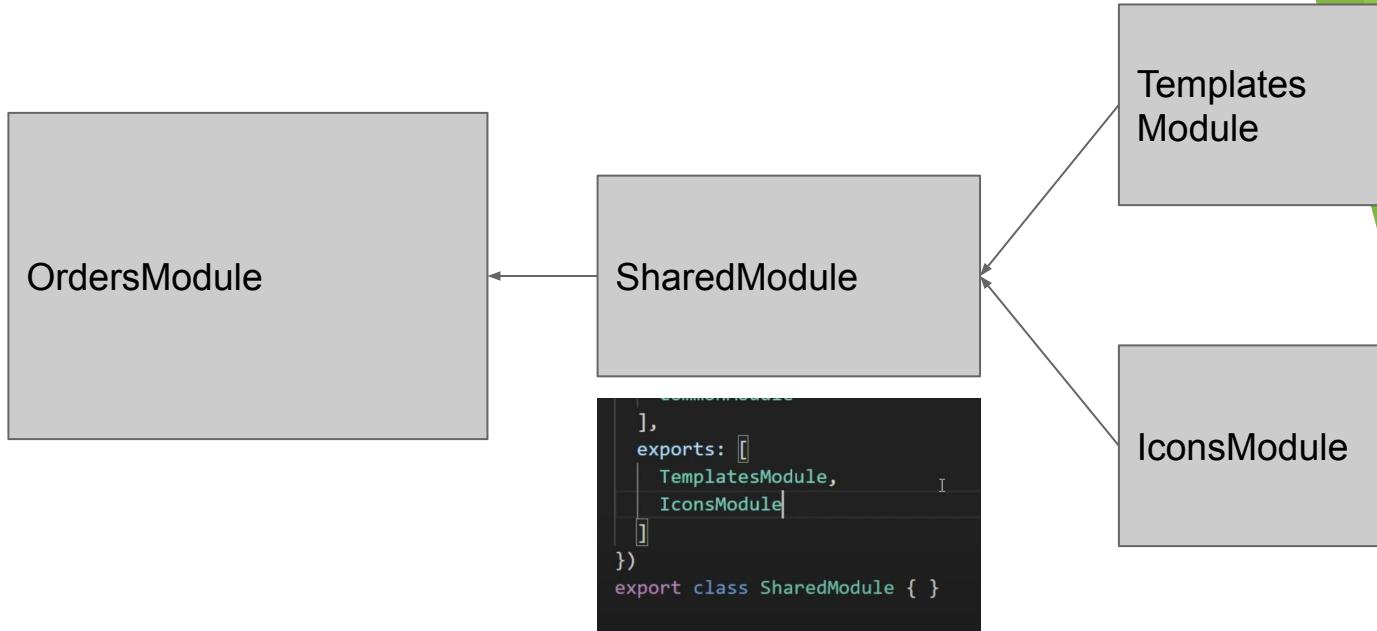
```
export class TemplateContainerComponent implements OnInit {  
  
    // besoin d'ajouter (!) ou undefined  
    // si propriété pas initialisée dans le constructor  
    // public title!: string;  
    // public title: string | undefined;  
    public title: string;  
  
    constructor() {  
        this.title = "titre"  
    }  
  
    ngOnInit(): void {  
    }  
}
```

Importer TemplatesModule dans OrdersModule, une bonne idée ?



On a besoin de TemplatesModule dans OrdersModule.
Mais ce n'est pas le seul module à devoir être importés (Shared, Icons...).
On va optimiser cette écriture en laissant à SharedModule la mission d'exporter TemplatesModule et IconsModule

SharedModule exporte les modules Templates et Icons





Les imports & exports nécessaires

- Sur SharedModule on exporte TemplatesModule et IconsModule
 - **shared.module.ts :**

```
exports: [
  TemplatesModule,
  IconsModule
]
```

- Sur OrdersModule on import SharedModule
 - **orders.module.ts :**

```
imports: [
  CommonModule,
  OrdersRoutingModule,
  SharedModule
]
```

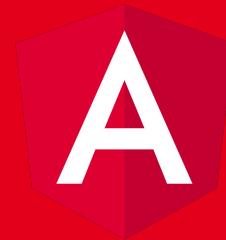


Commit 22 : Templates de page avec un title statique

- Retrouvez sur le dépôt Github de la formation le commit 22 concernant les templates de page & les imports / exports de TemplatesModule / SharedModule

23

@Input() data binding & lifecycle hooks





Réaffecter une variable public de deux components différents ?

- On va effectuer des tests afin de comprendre le fonctionnement d'Angular :
 - On va créer une instance de notre component **template-full-width** sur **app-list-orders.component.html**. On y initialise une variable `title = 'List Orders'` et on commente la valeur de title dans le constructor du template. Spoiler: cela ne fonctionne pas
 - En effet app-template est un web component. Il est donc encapsulé et on ne peut pas y accéder même si la propriété title est en public
 - Il existe cependant une solution ; nous pouvons accéder depuis le component parent à la propriété title du web component grâce au decorator `@Input()` et ainsi bind (lier) notre `@input()` à une variable située dans notre **page-list-orders.component.ts**
 - Ajoutons donc notre `@input()` : `@Input() public title!: string;`

Parent - Enfant

HTML

```
<app-enfant title="List Orders">
```

```
    TS du composant enfant  
    @Input() title
```

On passe une valeur directement.

TS app-enfant est lu
HTML app-enfant
Puis valeur title modifie le template HTML

Sans @Input, on ne peut pas agir sur les
props du composant enfant

```
composant parent  
public propParent = "Titre depuis le parent"
```

```
composant enfant  
@Input() title
```

```
HTML  
<app-enfant  
[title]='propParent'>
```

Une propriété existe dans le composant parent, ici propParent.

On veut faire passer cette information au composant enfant de façon dynamique.

On relie une propriété du composant enfant, title, avec une propriété du parent.

Besoin d'utiliser @Input() pour récupérer cette valeur.

```
<app-template-container [propEnfant]="propParent">
  <!-- ici contenu de cette page est projeté
  à la place de ng-content -->
  <h1>contenu page-list-orders</h1>
</app-template-container>
```

```
export class TemplateContainerComponent implements OnInit {

  // import de la valeur depuis Parent
  @Input() propEnfant!: string;

  constructor() {}

  ngOnInit(): void {
  }

}
```

Maintenant, on a accès à la valeur dans le composant enfant



@Input() Property binding (html)

- On va bind la variable `@Input() public title` de notre `app-template-full-width.component` à une variable `public myTitle` que l'on va initialiser & affecter à une string `'List Orders'` dans `page-list-orders.component.ts`
- On va ensuite créer un bouton avec un eventBinding `(click)` lié à une méthode `changeTitle()` qui nous permettra de changer notre titre lorsque l'on clique dessus
 - `page-list-orders.component.html` :

```
<app-template-full-width [title]="myTitle">
  <button (click)="changeTitle()">Change title</button>
  <p style="background-color: red;">
    Lorem ipsum dolor, sit amet consectetur adipisicing elit.
  </p>
</app-template-full-width>
```



@Input() Property binding (ts)

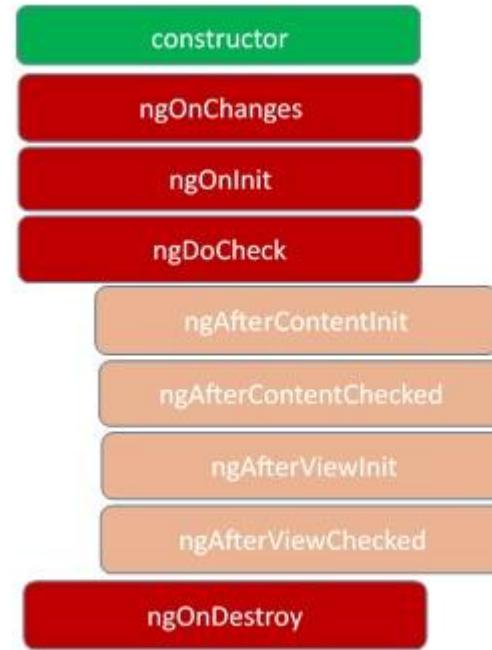
- On initialise notre variable `public myTitle` que l'on affecte à une string `'List Orders'` puis on crée notre méthode `public changeTitle(): void` dans laquelle on veut ré-affecter notre variable `this.myTitle` à une string `"My List Orders"`
 - o `page-list-orders.component.ts` :

```
export class PageListOrdersComponent implements OnInit {  
  public myTitle = 'List Orders';  
  constructor(private ordersService: OrdersService) {  
    this.ordersService.collection.subscribe();  
  }  
  ngOnInit(): void {  
  }  
  public changeTitle(): void{  
    this.myTitle= "My List Orders"  
  }  
}
```



Les lifecycle hooks

- Les **lifecycle hooks** sont des fonctions appelées de manière séquentielle dans un ordre bien précis. On peut choisir de les utiliser ou non suivant nos besoins
- Les plus souvent utilisés sont
 - **ngOnInit()**
 - **ngOnChanges()** --> *ChangeDetection
 - **ngOnDestroy()**



**ChangeDetection* : détecte les changements de valeur et permet à Angular d'actualiser la partie html sans avoir besoin de détruire le ts et le html du web component



Les lifecycle hooks les plus courants

- **ngOnInit()** :
 - On l'utilise quand on veut initialiser des propriétés au moment où un web component (ou directive) est initialisé. Par exemple AddOrdersComponent n'aura à initialiser qu'une seule fois notre formulaire
- **ngOnChanges()** :
 - On l'utilise quand on veut à chaque fois qu'un @Input change de valeur on détecte ce changement. Il n'y a pas besoin de le créer sauf si on a besoin de code au moment où ce changement est effectué
- **ngOnDestroy()** :
 - On l'utilise quand on veut que du code soit exécuté au moment où le web component est détruit. Par exemple un formulaire devra retenir les champs déjà renseignés si l'utilisateur quitte la page pour y revenir plus tard



Tests sur les lifecycle hooks

- Appliquons quelques petits tests pour comprendre le fonctionnement des lifecycle hooks :

```
export class TemplateFullWidthComponent implements OnInit {
  @Input() public title!: string;
  constructor() {
    // this.title = 'Le titre est ici'
    console.log(this.title);
  }
  ngOnChanges(arg: SimpleChanges) {
    console.log(arg, 'ngOnChanges');
  }
  ngOnInit(): void {
    console.log(this.title);
  }
}
```



Exercice

- Faire en sorte de modifier notre code pour tester avec un objet à la place du string dans la propriété myTitle.

Mettez des commentaires à la fin de vos lignes ajoutées, nous ne garderons pas ce code, il vous sera plus facile de le supprimer ainsi



Correction 1/3

- On va donc affecter notre propriété `public myTitle` à un objet plutôt qu'à un string, on update notre méthode `public changeTitle(): void` et on change également le type de notre propriété `@Input() public title!: string` pour qu'elle contienne notre objet :

- **page-list-orders.component.ts :**

```
public myTitle = { name: 'List Orders' };
public changeTitle(): void {
    this.myTitle.name = "My Order's list";
}
```

- **template-full-width.component.ts :**

```
@Input() public title!: any;
```



Correction 2/3

- Il ne nous reste plus qu'à modifier notre component templateFullWidth pour qu'il affiche la propriété `.name` de notre objet

- template-full-width.component.html :

```
<h1>{ { title.name } }</h1>
```

- Au clic sur le bouton, on constate que le **changeDetection** ne se fait pas dans le lifecycle hook `ngOnChanges()`. Cela implique qu'il se fait dans Le lifecycle hook `ngDoCheck()`
- On se rend vite compte qu'il y a un souci et que `ngDoCheck()` nous fait un **changeDetection** à tous les lifecycle hooks et qu'il faudrait vraiment passer par `ngOnChanges()`. Trouvons une solution
- <https://angular.io/api/core/DoCheck>



Correction 3/3

- Il suffit de réaffecter notre propriété par un nouvel objet plutôt que de réaffecter la propriété name de cet objet dans notre méthode

`public changeTitle(): void` . De ce fait le changeDetection se fera dans `ngOnChanges()`

- o **page-list-orders.component.ts :**

```
public changeTitle(): void {  
    this.myTitle = { name: "My Order's list" };  
}
```

/!\ Si vous avez besoin, il vous suffit de pull mon code sur Github juste après le commit de ce chapitre /!



Commit 23 : @Input() data binding & lifecycle hooks

- Retrouvez sur le dépôt Github de la formation le commit 23 concernant les @input() data binding et les lifecycle hooks

24

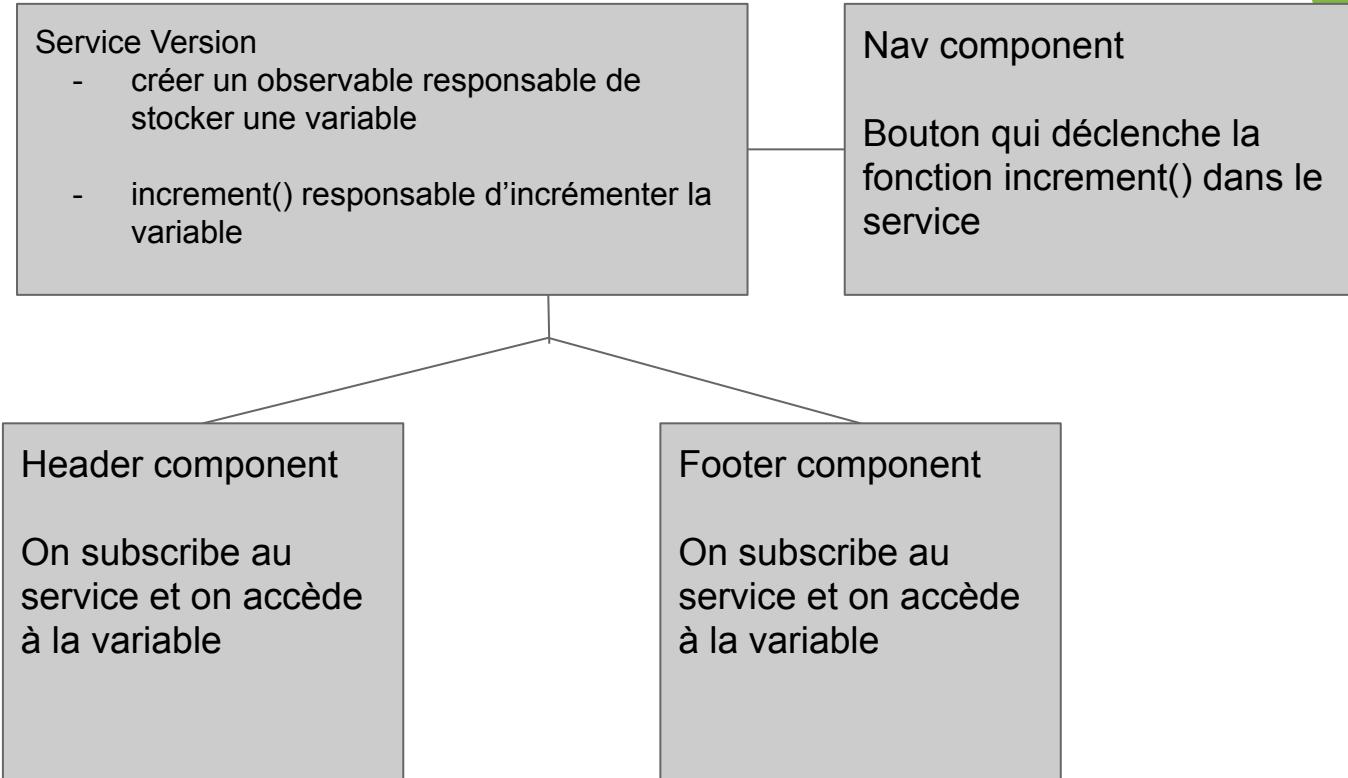
**Exercice “version” sur les
services & observables**





Exercice “version” sur les services & observables

- Exercice de validation des acquis :
 - On veut afficher la version de notre application dans le header et le footer puis ajouter un bouton dans le nav pour incrémenter cette version dynamiquement en cliquant
 - On va créer un service car les éléments concernés sont au même niveau de l'architecture et non pas parent -> enfant
 - Nous allons devoir injecter ce service dans header nav et footer pour récupérer la propriété version dans chaque component afin de l'afficher





Correction 1/3

- On commence donc par créer notre **versionService**. Pour ce faire nous allons créer un dossier “services” dans notre core module et générer un service
 - **ng g service version**
- On va initialiser une variable `public numVersion` et l'affecter à un `new BehaviorSubject(1)` puis on va créer notre fonction `public incrementVersion(): void` qui va nous permettre d'incrémenter notre variable en cliquant sur un bouton

```
export class VersionService {  
  public numVersion = new BehaviorSubject(1);  
  constructor() {}  
  
  public incrementVersion() {  
    this.numVersion.next(this.numVersion.value + 1);  
    console.log(this.numVersion.value); } }
```



Correction 2/3

- On injecte notre service dans nos 3 components header, nav & footer pour pouvoir l'utiliser par la suite. On crée une propriété `public v` et une méthode `public increment(): void` qui nous seront elles aussi utiles par la suite

- **Core > components > HeaderComponent.ts & footerComponent.ts :**

```
export class HeaderComponent implements OnInit {
  public v!: number;
  constructor(private versionService: VersionService) {
    this.versionService.numVersion.subscribe((number) => this.v = number);
  }
}
```

- **navComponent.ts :**

```
export class NavComponent implements OnInit {
  constructor(private versionService: VersionService) { }
  ngOnInit(): void {}
  public increment(): void{
    this.versionService.incrementVersion();
  }
}
```



Correction 3/3

- On va s'occuper de la partie html de nos 3 components :

- **header.component.html :**

```
<p>Version n°: {{ v }}</p>
```

- **nav.component.html :**

```
<button (click)="increment()">Increment Version</button>
```

- **footer.component.html :**

```
<p>Version n°: {{ v }}</p>
```

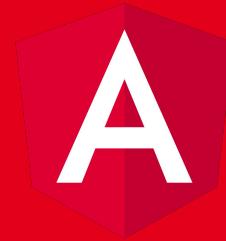


Commit 24 : Exercice “version” sur les services & observables

- Objectif : un bouton dans la nav incrémente une variable. Le résultat est affiché dans deux components
- Retrouvez sur le dépôt Github de la formation le commit 24 concernant l'exercice “version” sur les services & observables

25

Cpnts simples, property
binding & lifecycle hooks





**Objectif : afficher la liste dans
le composant page-list-orders**



Retirer le bouton changeTitle()

- Dans page-list-orders, on peut retirer le bouton changeTitle()



Création des components simples dans shared

- Dans notre **sharedModule** on va commencer par créer un dossier components dans lequel on va générer 2 components ; **table-light** et **btn** :
 - **ng g component table-light --export**
 - **ng g component btn --export**
- Puis on va placer les selectors de nos 2 components dans le template de notre **page-list-orders.component.html** :

```
<app-table-light></app-table-light>
```

```
<app-btn></app-btn>
```



Récupération des datas à afficher dans nos 2 components

- On crée une propriété `public collection` de type `Order[]` ainsi qu'une propriété `public headers` qu'on initialise à un tableau de strings (titres des colonnes) contenant les propriétés de notre OrderModel.
- Dans le 1er paramètre (next) de notre subscribe on va récupérer notre `data` et l'affecter à `this.collection`
 - **page-list-orders.components.ts :**

```
export class PageListOrdersComponent implements OnInit {  
  public myTitle = 'List Orders';  
  public collection!: Order[];  
  public headers = [ ]  
  constructor(private ordersService: OrdersService) {  
    this.ordersService.collection.subscribe((data) => this.collection = data);  
  }  
}
```



Récupération des datas à afficher dans nos 2 components

- On va créer 2 input `@Input() public collection` & `@Input() public headers`
Que l'on va binder avec nos deux variables `public collection` & `public headers`
dans le template de notre component pageListOrders

- table-light.components.ts :

```
export class TableLightComponent implements OnInit {  
  @Input() public collection!: Order[];  
  @Input() public headers!: string[];
```

- page-list-orders.components.html :

```
<app-table-light [collection]='collection' [headers]='headers'>  
</app-table-light>
```



Property binding & lifecycles

tests de comportement

- Ensuite on va effectuer quelques petits tests pour comprendre un peu mieux ces fameux life cycles
 - **table-light.components.ts :**

```
constructor() {  
    console.log(this.collection);  
    console.log(this.headers);  
}  
  
 ngOnChanges(): void{  
    console.log(this.collection);  
    console.log(this.headers);  
}  
  
 ngOnInit(): void {  
    console.log(this.collection);  
    console.log(this.headers); } }
```

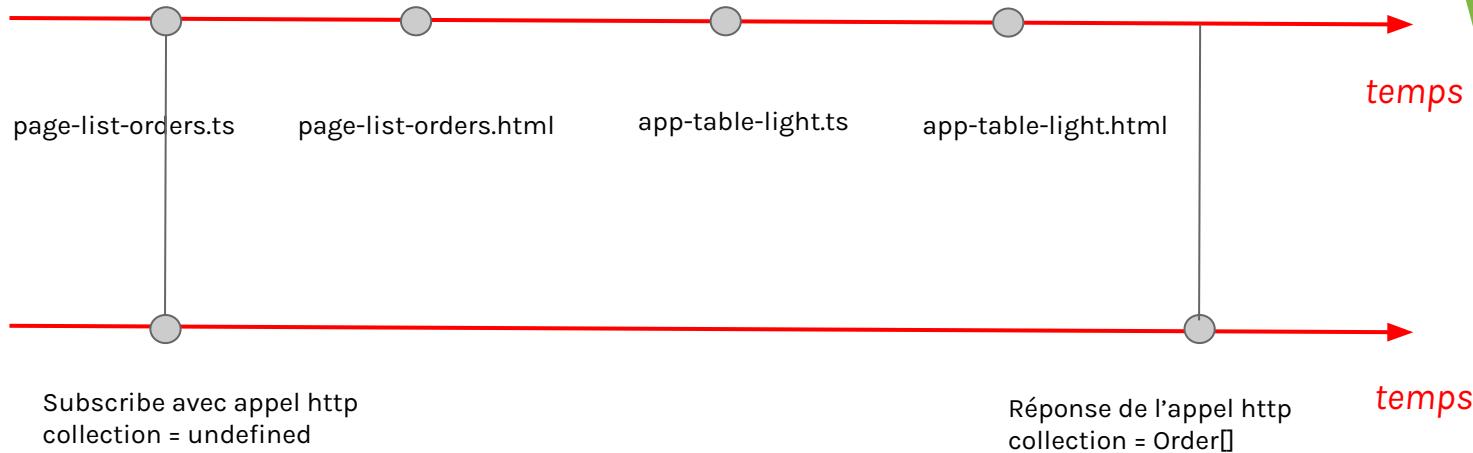
```
undefined  
▶ (4) [{} , {} , {} , {} ]
```

```
table-light.component.ts:20  
table-light.component.ts:20
```

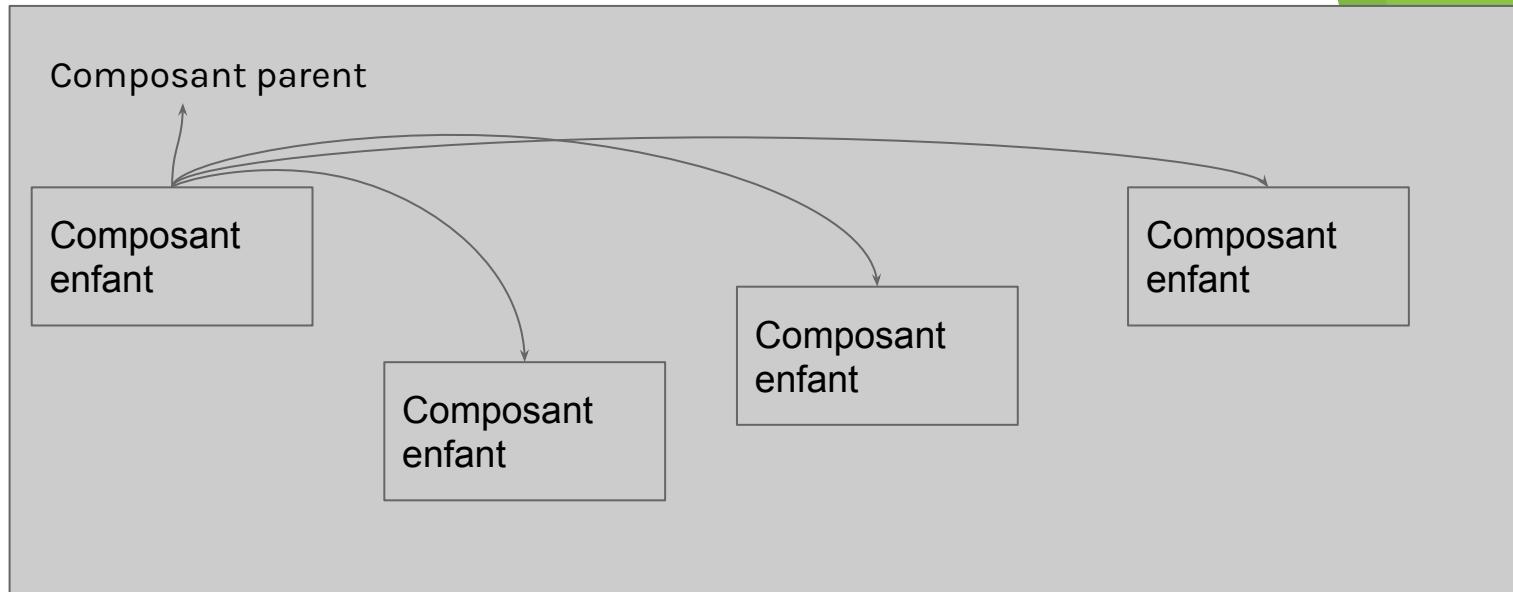
Au démarrage de l'application, la variable collection est vide (undefined), puis prend une valeur.
=> nouvel appel ngOnChanges

/orders

ngOnChanges() est déclenché.
Un ChangeDetection est déclenché et réévalue les composants concernés (sans produire une nouvelle instance TS/HTML)



Stratégie ChangeDetection à l'intérieur d'une même vue



Un seul `ngOnChanges()` va déclencher des `ngOnChanges` dans tous les autres composants de la vue (16ms par `ngOnChanges`)



Optimisation du code performances de l'application

- Pour éviter de lancer inutilement des ChangeDetection et de binder des @input() avec undefined avant de récupérer notre collection on peut demander à Angular d'instancier ce web component uniquement quand la collection aura une valeur
- Cela marche aussi si l'on sélectionne l'ensemble de ce qui se trouve dans app-template-full-width avec un `*ngIf=""`
 - **page-list-orders.component.html :**

```
<app-template-full-width [title]="myTitle" *ngIf="collection">
```



Commit 25 : Components simples avec property binding & lifecycle hooks

- Retrouvez sur le dépôt Github de la formation le commit 25 concernant les components stupides avec property binding et lifecycles

26

Projection d'un tableau dynamique



```
<table class="table">
    <!-- thead utile pour le référencement -->
    <thead>
        <!-- table row (ligne) -->
        <tr>
            <!-- table header cells (th) -->
            <th scope="col">#</th>
            <th scope="col">titre 2 </th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <th scope="row">1</th>
            <!-- table data cells (td) -->
            <td>Mark</td>
            <td>Otto</td>
            <td>@mdo</td>
        </tr>
    </tbody>
</table>
```



Tableau dans le template de tableLightComponent

- Dans shared > components>table-light, on récupère un exemple de tableau sur bootstrap et on épure le code pour nos besoins

- Table-light.component.html :



!/\ page-list-orders.component.html :

=>

```
<app-table-light  
[collection]='collection'  
[headers]='headers'>  
</app-table-light>
```

Ce code rend notre component
table-light difficilement réutilisable !\

- On va devoir trouver une solution pour que notre component soit facilement réutilisable



Projection du tableau avec ng-content

- On retire le contenu de la balise `<tbody>` pour y mettre notre `<ng-content>` ce qui va nous permettre de pouvoir projeter ce que l'on veut dans notre tableau et ainsi gagner en réutilisabilité
 - **table-light.component.html :**

```
<table class="table">
  <thead>
    <tr>
      <th *ngFor="let item of headers" scope="col">{{ item }}</th>
    </tr>
  </thead>
  <tbody>
    <ng-content></ng-content>
  </tbody>
</table>
```



Projection du tableau avec ng-content

- On va pouvoir projeter ici nos td, ce qui nous donne un avantage conséquent ; nous pouvons utiliser notre variable `let item` située dans notre `*ngFor=""` pour lister les propriétés de notre tableau sans avoir à faire de property binding et supprimer notre `@Input() public collection` et notre bind `[collection]="collection"`

- page-list-orders.component.html :

```
<app-table-light [headers]="headers">
  <tr *ngFor="let item of collection">
    <td>{{ item.typePresta }}</td>
    <td>{{ item.client }}</td>
    <td>{{ item.nbJours }}</td>
    <td>{{ item.tjmHt }}</td>
    <td>{{ item | total }}</td>
    <td>{{ item | total : true }}</td>
    <td>{{ item.state }}</td>
  </tr>
</app-table-light>
```

```
</thead>

<tbody>
  <ng-content></ng-content>
</tbody>

</table>
```



Commit 26 : Projection d'un tableau dynamique

- Retrouvez sur le dépôt Github de la formation le commit 26 concernant la projection d'un tableau dynamique dans pageListOrdersComponent & TableLightComponent

27

**Pipe perso & rxjs filter
Unsubscribe exemples
& pipe async**





Comment procéder ?

- **Question ?**
 - Comment calculer total HT et total TTC et les afficher dans notre tableau de la manière la plus correcte possible ?



Appel de fonction dans le html

- On va essayer d'appeler une méthode dans la page html dans un premier temps

- **page-list-orders.ts :**

```
public total(val: number, coef: number, tva?: number): number {  
  console.log('total called');  
  if (tva) return val * coef * (1 + tva / 100);  
  return val * coef;  
}
```

- **page-list-orders.html :**

```
<td>{{ total(item.tjmHt, item.nbJours) }}</td>  
<td>{{ total(item.tjmHt, item.nbJours, item.tva) }}</td>
```



Trouver une autre solution ?

- Faire un appel de méthode dans le html n'est pas une bonne pratique
 - Il ne faut jamais mettre d'appel de fonctions comme cela dans le html car les lifecycle hooks multiplient le nombre d'appels à chaque changement d'état ce qui pourrait dégrader la performance de notre application
 - Nous allons plutôt nous orienter vers les **Pipes**, cela semble être une bien meilleure pratique. Mais, qu'est-ce qu'un pipe ?
 - Un pipe c'est avant tout une Class avec le décorator @pipe()
 - Son rôle ? : Récupérer des datas en entrée et de les retourner en sortie

Cette **solution** me semble intéressante, à chaque fois qu'on en aura besoin, il suffira de faire appel à une instance du pipe en question ; de plus il va nous éviter d'avoir des changements d'état à répétition car il ne sera jamais détruit / re-créé au fil des ChangeDetection



Créons un pipe total

- On va créer un dossier pipes dans notre sharedModule
 - **ng g pipe total --export**
- Ensuite on va simplement reprendre le principe de notre fonction

```
public total(val: number, coef: number, tva?: number): number
```

Dans notre pipe

- **total.pipe.ts :**

```
export class TotalPipe implements PipeTransform {
  transform(val: number, coef: number, tva?: number): number {
    if (tva) return val * coef * (1 + tva / 100);
    return val * coef; }}
```

```
<!-- valeur de return du Pipe -->
<td>{{ i.tjmHt | total : i.nbJours }}</td>
<td>{{ i.tjmHt | total : i.nbJours: i.tva }}</td>
```

```
export class TotalPipe implements PipeTransform {
  transform(val: number, coef:number, tva?:number): number {
    if (tva){
      return val * coef *(1+ tva /100)
    }
    return val * coef
  }
}
```

On met le premier argument à gauche du nom du Pipe, puis les suivants à droite



On modifie le html & on optimise le code

- On va pouvoir modifier notre html pour utiliser notre pipe et non plus notre méthode
 - **page-list-orders.component.html :**

```
<td>{{ item.tjmHt | total: item.nbJours }}</td>
<td>{{ item.tjmHt | total: item.nbJours: item.tva }}</td>
```

- Nous avons donc trouvé la meilleure solution. En revanche, notre code n'est pas optimisé et nous pouvons encore améliorer cela. **On va ajouter directement deux méthodes dans notre core > interface & model**

- **Notre interface order-i.ts :** ◦ **Notre model order.ts :**

```
totalHT(): number;
totalTTC(): number;
```

```
public totalHT(): number{
    return this.tjmHt * this.nbJours;
}
public totalTTC(): number{
    return this.tjmHt * this.nbJours * (1 + this.tva / 100);}
```

Ajout des méthodes dans l'interface et la classe

```
export interface OrderI {  
    tjmHt: number,  
    nbJours: number,  
    tva: number,  
    state: StateOrder,  
    typePresta: string ,  
    client: string,  
    comment: string,  
    id: number,  
    totalHT(): number;  
    totalTTC(): number;  
}
```

```
export class Order implements OrderI{  
    // valeurs par défaut  
    tjmHt = 1200;  
    nbJours = 1;  
    tva = 20;  
    state = StateOrder.OPTION; // enums  
    typePresta! : string ; // besoin d'initialiser dans le con  
    client!: string;  
    comment!: string;  
    id!: number ;  
    totalHT (): number{  
        return this.tjmHt * this.nbJours;  
    }  
    totalTTC(): number{  
        return this.tjmHt * this.nbJours* (1 + this.tva/100)  
    }  
}
```



Optimisation de notre pipe

- Optimisons en conséquences de nos modifications le code de notre pipe
 - **total.pipe.ts :**

```
transform(val: any, tva?: boolean): number {  
  if (tva) return val.totalTTC();  
  return val.totalHT();  
}
```

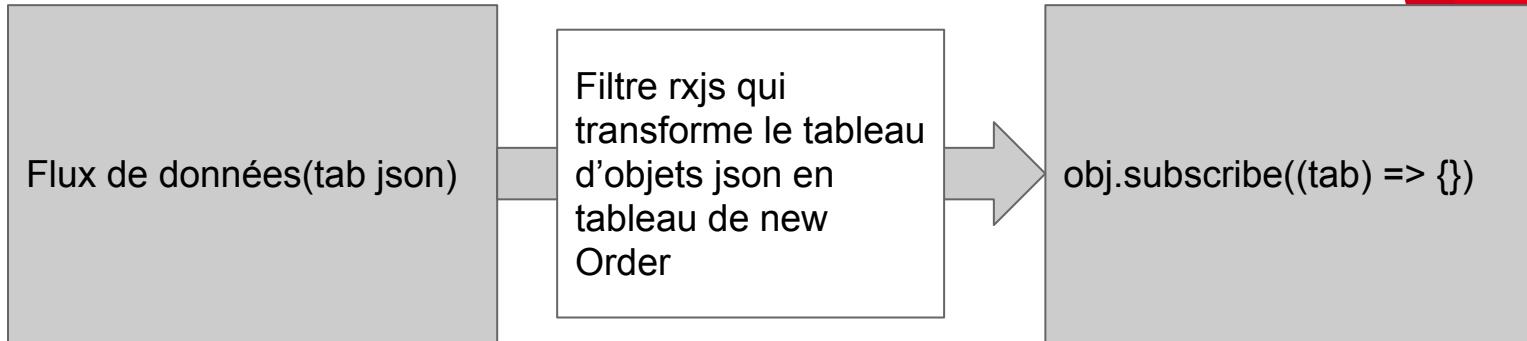
- Optimisons notre html en rapport avec nos modifications
 - **page-list-orders.component.html :**

```
<td>{{ item | total }}</td>  
<td>{{ item | total: true }}</td>
```

```
✖ ▶ ERROR TypeError: item.totalHT is not a function  
at TotalPipe.transform (total.pipe.ts:13)  
at pureFunction1Internal (core.mjs:22131)  
at Module.ɵepipeBind1 (core.mjs:22311)  
at PageListOrdersComponent_app_template_full_width_0_t  
at executeTemplate (core.mjs:9601)  
at refreshView (core.mjs:9467)  
at refreshEmbeddedViews (core.mjs:10592)  
at refreshView (core.mjs:9491)  
at refreshEmbeddedViews (core.mjs:10592)  
at refreshView (core.mjs:9491)  
  
✖ ▶ ERROR TypeError: item.totalTTC is not a function  
at TotalPipe.transform (total.pipe.ts:11)  
at pureFunction2Internal (core.mjs:22150)  
at Module.ɵepipeBind2 (core.mjs:22332)  
at PageListOrdersComponent_app_template_full_width_0_t  
at executeTemplate (core.mjs:9601)  
at refreshView (core.mjs:9467)  
at refreshEmbeddedViews (core.mjs:10592)  
at refreshView (core.mjs:9491)  
at refreshEmbeddedViews (core.mjs:10592)  
at refreshView (core.mjs:9491)
```



Filtre rxJS



Objectif : transformer le flux de données pour lui donner le format de la classe Order.

Conséquence : chaque élément du tableau aura le format de la classe Order, donc avec les méthodes totalHT() et totalTTC().

Librairie rxJS

Documentation opérateurs : <https://rxjs.dev/api>

Documentation .pipe

Ce sont les opérateurs qui permettent de transformer un flux de données dans un pipe rxJS.

Opérateur map



Filtre rxJS

- On voudrait qu'avant d'envoyer le flux de données en souscrivant à celui-ci il soit transformé en **un tableau** de new Order car dans ce cas nous aurons les méthodes `public totalHT(): number` & `public totalTTC(): number` disponibles, à la différence de l'objet json qui lui en est dépourvu.
- On va donc utiliser un filtre RxJS & les operators qu'il contient
 - **order.service.ts :**

```
this.collection = this.httpClient.get<Order[]>(`${this.urlApi}/orders`).pipe(  
  map((tab) => {  
    return tab.map((obj) => {  
      return new Order(obj);});});
```

Utilisation du pipe rxJs

map retourne le tableau stocké dans tab.

```
constructor(  
    private http : HttpClient  
) {  
    this.collection = this.http.get<Order[]>(`${this.urlApi}/orders`).pipe(  
        map((tab: any)=>{  
            return tab.map((obj: any)=>{  
                return new Order(obj)  
            })  
        })  
    )  
}
```

Ici c'est une méthode JS qui permet de prendre un tableau en input et return un nouveau tableau. Obj représente chaque item du tableau.

```
(4) [Order, Order, Order, Order] ⓘ  
▶ 0: Order {tjmHt: 1200, nbJours: 10, tva: 20, state: 'OPTION', typePresta: 'coachingDFGH', ...}  
▶ 1: Order {tjmHt: 1200, nbJours: 1, tva: 20, state: 'CONFIRMED', typePresta: 'coaching', ...}  
▶ 2: Order {tjmHt: 1200, nbJours: 1, tva: 20, state: 'CANCELED', typePresta: 'ddd', ...}  
▶ 3: Order {tjmHt: 1200, nbJours: 1, tva: 20, state: 'OPTION', typePresta: 'dsfgqdfg', ...}  
  length: 4  
[[Prototype]]: Array(0)
```



Comment unsubscribe

- Tout fonctionne, cependant... Lorsque l'on subscribe ainsi à un observable il n'y a pas de problème car nous ne l'avons pas créé, c'est httpClient.get() qui s'en est occupé et par conséquent le code nécessaire pour y dé-souscrire existe déjà. Créons alors un petit exemple pour comprendre comment unsubscribe :

```
//exemple unsubscribe
private test = new BehaviorSubject(0);
ngOnInit(): void {
    //exemple subscribe pour unsubscribe
    this.test.subscribe((data) => console.log(data));
}
```

On constate que lorsqu'on change de page, `ngOnDestroy()` détruit notre component mais nous ne dé-souscrivons pas, ce qui va créer des fuites mémoires



Unsubscribe.. C'est long

- Chaque fois que l'on crée un Observable on va donc devoir y dé-souscrire :

```
//exemple subscribe pour unsubscribe
    private sub!: Subscription;
//exemple subscribe pour unsubscribe
this.sub = this.test.subscribe((data) => console.log(data));

    //unsubscribe
ngOnDestroy(): void {
    this.sub.unsubscribe();
}
```

- On ajoute quand même beaucoup de code pour pas grand chose.. En plus, à force de faire ce genre de subscribes dans notre code on va finir par oublier d'unsubscribe aux observables qui ne sont pas créés par httpclient



Comment unsubscribe

- On va donc trouver une autre solution qui nous permettra d'avoir un code plus simple
- Dans **page-list-orders** : pour cela, on va supprimer notre propriété `public collection!: Order[]` et créer une autre propriété `public collection$: Observable<Order[]>`; dans laquelle nous allons faire une copie par référence de notre propriété
`public collection$: Observable<Order[]>;` présente dans notre OrdersService :

```
this.collection$ = this.ordersService.collection
```



Pipe async

- Nous avons maintenant un flux `collection$` auquel nous n'avons pas souscrit. On va donc y souscrire grâce à un pipe `| async` dans notre html
- Cela va créer une instance du pipe `| async` dans laquelle la méthode `transform()` récupère un observable pour y souscrire et retourner un tableau de new Order. On choisit de stocker ce tableau de new Order dans une variable locale côté html `as collection`, on boucle ensuite sur celle-ci pour lister toutes nos propriétés d'orders
 - **page-list-orders.components.html :**

```
<app-template-full-width  
[title]="myTitle" *ngIf="collection$ | async as collection">  
</app-template-full-width>
```

What Is Async Pipe?

The `async` pipe subscribes to an Observable or Promise and returns the latest value it has emitted. When a new value is emitted, the `async` pipe marks the component to be checked for changes. When the component gets destroyed, the `async` pipe unsubscribes automatically to avoid potential memory leaks. When the reference of the expression changes, the `async` pipe automatically unsubscribes from the old Observable or Promise and subscribes to the new one.

<https://angular.io/api/common/AsyncPipe>



Commit 27 : Pipe total & rxjs filter Unsubscribe & pipe async

- Retrouvez sur le dépôt Github de la formation le commit 27 concernant le pipe perso “total”, le filtre rxJS, les différents exemples d’unsubscribe & le pipe async

28

Pipes Angular & changeState





Pipes Angular & pipe Json

- Regardons le comportement du pipe `| json`. C'est un pipe utile pour nous aider à debug notre code à la manière d'un `console.log()`;
- Imaginons qu'il nous manque une data dans notre tableau et qu'on se demande pourquoi, on a un peu la flemme d'aller regarder côté api etc... et bien c'est là qu'on va pouvoir utiliser le plein potentiel de ce pipe qui nous return un objet json
- Ou encore on a tout simplement envie de comprendre ce qui se situe dans une variable html, mais l'on voit “[Object]”. Pour voir l'intérieur de ce tableau json on va pouvoir utiliser notre pipe json
 - **page-list-orders.component.html :**

```
<p>{{ collection | json }}</p>
```



Pipes Angular & pipe currency

- Quand on essaye le pipe currency on se rend vite compte qu'il utilise des constantes type en-US qui affectent le formatage et le choix de la currency. Dans la doc on remarque bien que par défaut "DEFAULT_CURRENCY_CODE" est bien 'USD'
- On a deux choix :
 - On peut appliquer ça sur un module pour surcharger la valeur par défaut d'angular.json
 - On surcharge notre useValue directement dans un pipe

Documentation : https://angular.io/api/core/DEFAULT_CURRENCY_CODE



Paramètres currency par module

- On ajoute un tableau providers à notre module en question et on y ajoute le “DEFAULT_CURRENCY_CODE” de notre choix :
 - **Orders.module.ts :**

```
providers: [
  { provide: DEFAULT_CURRENCY_CODE, useValue: 'EUR' }
],
```



Paramètres currency par pipe

- On ajoute currency: ‘choix de notre currency’ à tout les pipes de notre application auxquels on souhaite surcharger cette `useValue` :
 - `page-list-orders.html` :

```
<td>{{ item | total : true | currency: 'EUR' }}</td>
```

Le pipe currency récupère la valeur renvoyée par le premier pipe et retourne un nouveau string



Paramètres régionaux

- Pour définir les paramètres régionaux de notre application nous allons utiliser `i18n` et fournir le jeton LOCAL_ID
 - `angular.json` :

```
    "prefix": "app",
    "i18n": {
        "sourceLocale": "fr-FR"
    },
}
```

/!\ Pour voir les changements relancer ng serve dans le terminal /!\\



Notre state dans un <select>

- On va s'occuper de notre state car actuellement on affiche notre state mais on aimerait avoir un select pour pouvoir changer le state

```
<td>
    <select>
        <option value="CANCELLED" >CANCELLED</ option>
        <option value="OPTION" >OPTION</ option>
        <option value="CONFIRMED" >CONFIRMED</ option>
    </ select>
</ td>
```

- Ici, on se rapproche de ce que l'on veut, on a écrit nos states en dur, le mieux serait de les récupérer et de boucler dessus pour les afficher, pour pouvoir les changer en base de données facilement



Changer notre state ? `changeState()`

- On veut boucler sur les states existants de notre enum sauf qu'ici `stateOrder` est un objet et il attend un import qu'on ne peut pas placer dans le html. De plus `*ngFor=""` ne peut boucler que dans un tableau, or nous avons ici un objet

- `page-list-orders.component.html` :

```
<option  
  *ngFor="let state of stateOrder"  
  [value]='state'  
  [selected]='state === item.state'>  
</option>
```

- On va donc devoir trouver un moyen de récupérer les propriétés d'un objet dans un tableau pour parcourir celui-ci



Object.values changeState()

- On peut trouver `Object.values()` qui va nous permettre d'obtenir notre tableau dans une variable `public states`

- **page-list-orders.component.ts :**

```
public states = Object.values(StateOrder);
```

- **page-list-orders.components.html :**

```
<option  
*ngFor="let state of states" [value]="state" [selected]="state === item.state">  
{{ state }}  
</option>
```



Si un des états (confirmed, cancelled ou option) correspond à l'état de l'objet item, alors il est sélectionné. Selected est true.



Event (change) `changeState`

- On va créer un event (change) car si le state change côté front on veut faire un appel http pour pouvoir update les données côté back

- `page-list-order.component.html` :

```
<select (change)="changeState(item, state)"> </select>
```

- Seulement la variable `let state` est en dehors du scope et on ne peut pas l'utiliser. Il y a néanmoins une solution ; on va pouvoir envoyer l'event à la méthode grâce au mot clé `$event`

```
<select (change)="changeState(item, $event)"> </select>
```



Méthode OrdersService.changeState()

- On commence par créer une méthode

```
public changeState(item: Order, state: StateOrder): Observable<Order>
```

Et on déclare une `const obj` afin de créer un nouvel objet de type Order à partir de notre Order passé en paramètres.

On va ensuite passer à `obj.state` le `state` passé en paramètres

Et enfin on return `this.ordersService.update(obj)`

- Orders.service.ts :

```
// change state item
public changeState(item: Order, state: StateOrder): Observable<Order>
{
    const obj = new Order({...item});
    obj.state = state;
    return this.update(obj);
}
```

```
JavaScript + No-Library (pure JS) ▾   Tidy
1  const obj = { prenom:"jean"}
2
3  const obj2 = obj;
4
5  obj2.prenom = "valérie";
6  console.log(obj) // valérie
7
```

Copie par référence

```
1 ▼ let obj = {  
2   prenom: 'jean'  
3 }  
4 // ici il reste indépendant  
5 let obj2 = {...obj}  
6 obj2.prenom= 'nouveau'  
7 // ici obj est intacte  
8 console.log(obj, 'obj')  
9  
10  
11  
12 ▼ let objR = {  
13   prenom: 'jean'  
14 }  
15 // ici, ils vont pointer vers le même objet  
16 let obj2R = objR  
17  
18 obj2R.prenom = 'nouveau'  
19  
20 console.log(objR, 'objR')
```



Méthode OrdersService.update()

- On va devoir créer une méthode `public update(item: Order): Observable<Order>` dans notre service qui va nous retourner un appel `this.http.put<Order>(`${this.urlApi}/orders/${item.id}`, item)`
 - Orders.service.ts :

```
// update item in collection
public update(item: Order): Observable<Order> {
  return this.http.put<Order>(`${this.urlApi}/orders/${item.id}`, item);
}
```

Documentation PUT :
<https://angular.io/guide/http#making-a-put-request>

Retourner l'objet dans le body de la réponse

The screenshot shows a REST API interface with the following details:

- API Endpoint:** /orders/:id (PUT)
- Response 1 (200) Status & Body:** 200 - OK
- Body Content:** 1 {{body}}
- Network Timeline:** Shows a single request taking approximately 10000 ms.
- Table Headers:** Name, Headers, Payload, Preview, Response, Initiator, Timing.
- Table Data:** A single row with the value 1 {"tjmHt":1200,"nbJours":10,"tva":20,"state":"OPTION",...} under the Response column.



Méthode `changeState()`

- On va créer une méthode `public changeState(item: Order, event: any): void` dans laquelle on veut initialiser une `const state` qui récupère notre `event.target.value` pour ensuite la passer ainsi que notre paramètre `item` aux paramètres de notre méthode `ordersService.changeState(item, state)`
- Il ne nous reste plus qu'à `.subscribe()` et affecter `item` à `data` en paramètre next de notre souscription
 - **page-list-order.component.ts :**

```
public changeState(item: Order, event: any): void {  
  const state = event.target.value;  
  this.ordersService.changeState(item, state).subscribe((data) => item = data);  
}
```



Méthode **changeState()**

```
public changeState(item: Order, event : Event): void{
    const target = event.target as HTMLSelectElement;
    console.log(target);
    const state = target.value as StateOrder;
    console.log(target.value);
}
```



Commit 28 : Pipe Angular & changeState

- Retrouvez sur le dépôt Github de la formation le commit 28 concernant les pipes angular, la méthode changeState() & OrderService.changeState() + OrderService.update()

29

StateDirective & HostBinding





SCSS de nos **statesOrder**

- On veut changer la couleur en fonction du state sélectionné
 - **page-list-orders.component.scss :**

```
.state-cancelled{  
  background: var(--app-error);  
}  
.state-option{  
  background: var(--app-warning);  
}  
.state-confirmed{  
  background: var(--app-success);  
}
```



NGclass StateOrder

- On veut utiliser `[ngClass]` pour donner une class scss suivant le state sélectionné. On va juste s'assurer que `data.state` est bien affecté à `item.state`

- **page-list-orders.component.html :**

```
<td [ngClass]="{
  'state-cancelled': item.state === 'CANCELLED',
  'state-confirmed': item.state === 'CONFIRMED',
  'state-option': item.state === 'OPTION'
}>
</td>
```

- **page-list-orders.component.ts :**

```
.subscribe((data) => item.state = data.state);
```

Problème : l'item ne change pas de couleur quand on change son statut.

```
public changeState(item: any, e: any): void{
    const state = e.target.value;
    this.ordersService.changeState(item, state).subscribe((data: any)=> {
        console.log(data, "data");
        item = data;
        console.log(item, "item");
    })
}
```

Raison : ici, la variable item prend la référence de data



La solution `Object.assign`

- Cela fonctionne seulement ce n'est toujours pas la bonne solution car cela crée un décalage entre le front et le back
 - `page-list-orders.component.ts` :

```
.subscribe((data) => Object.assign(item, data));
```



Créer une **custom directive**

- La directive NgClass fonctionne mais on aimerait une directive qui réponde encore mieux à nos besoins ; être capable de générer des noms de classe à partir d'un state.
- On va créer un dossier directives dans notre sharedModule
 - **Ng g directive state --export**



Créer une **custom directive**

- Faisons quelques tests.
- Nous allons faire passer la valeur item.state à la directive state

```
<td appState [test]='item.state'>
  <select (change)='changeState(item, $event)'>
    <option
      *ngFor='let state of states'
      [value]='state'
      [selected]='state === item.state'
    >{{state}}</option>
  </select>
</td>
```

```
export class StateDirective {
  @Input() test!:string
  constructor() {
    console.log(this.test);
  }
  ngOnChanges(){
    console.log(this.test);
  }
}
```



Notre **custom directive**

- On initialise un `@Input()` `appState` de type `string` qu'on va pouvoir bind à `item.state` dans notre html et on fait un `console.log(this.appState)`
Pour voir ce que l'on obtient

- **state.directive.ts :**

```
export class StateDirective {  
  @Input() appState!: string;  
  constructor() {}  
  ngOnChanges() {  
    console.log(this.appState);  
  }  
}
```

On peut améliorer l'écriture en nommant la propriété comme la directive.

[appState] fait référence à la directive et au nom de la propriété.

- **page-list-orders.component.html :**

```
<td [appState]="item.state">
```

Mettre en place le HostBinding

```
// on veut appState vaut canceled = state-canceled  
// on veut appState vaut option = state-option  
// on veut appState vaut confirmed = state-confirmed  
  
// binder ce string avec la propriété 'class' du host element td
```

```
<td _ngcontent-htb-c42>14400</td>  
►<td _ngcontent-htb-c42 ng-reflect-app-state="CANCELLED" class="state-canceled">...</td> ==  
</tr>
```



Custom directive & host binding

- On s'aperçoit avec notre `console.log()` que nous ne récupérons pas ce qu'il nous faut donc on va devoir utiliser un `@HostBinding` pour récupérer correctement nos states sous type `string` et les passer dans la méthode `.toLowerCase()` pour les récupérer en lowercase

- o **state.directive.ts :**

```
export class StateDirective {  
  @Input() appState!: string;  
  @HostBinding('class') tdClassName!: string  
  constructor() {}  
  ngOnChanges() {  
    console.log(this.appState);  
    this.tdClassName = `state-${this.appState.toLowerCase()}`;  
  }  
}
```



Commit 29 : StateDirective & hostBinding

- Retrouvez sur le dépôt Github de la formation le commit 29 concernant notre custom directive & le hostBinding

30

**Btn component
ready to (re)use**





Générer notre **component btn**

- On veut ajouter un bouton sur nos pages similaires tout en choisissant le texte qui apparaît sur ce bouton et le lien vers lequel il renvoie. On va donc générer un component btn dans notre sharedModule/components
 - **Ng g component btn**
- Comme il y a deux variables qui changent selon la page ciblée nous aurons 2 @Input à créer dans le ts de notre bouton
 - **btn.component.ts :**

```
export class BtnComponent implements OnInit {  
  @Input() public label!: string;  
  @Input() public route!: string;  
  constructor() { }  
}
```



Component btn import de RouterModule

- On va ajouter RouterModule aux imports de notre SharedModule pour pouvoir utiliser la directive `[routerLink]` dans le html de notre btnComponent
 - **shared.module.ts :**

```
imports: [
  CommonModule,
  RouterModule
]
```

- **btn.component.html :**

```
<a [routerLink]="route">{{ label }}</a>
```



Component btn ready to (re)use

- On n'a plus qu'à ajouter le selector `<app-btn>` partout où l'on souhaite ajouter un bouton & bind nos @input
 - **page-list-orders.component.html :**

```
<app-btn label="add an order" route="add"></app-btn>
```

Petit rappel :

*Si on ajoute un slash devant, cela
correspond à nomdedomaine/add et non pas
nomdedomaine/orders/add*

```
<app-btn label="Ajouter" route="/add"></app-btn>
```



Commit 30 : Btn component ready to (re)use

- Retrouvez sur le dépôt Github de la formation le commit 30 concernant notre component btn réutilisable

31

Tp sur la partie clients





Fournitures du tp

- Une interface **client-i** comportant (un model qui implémente client-i)
 - Un state : stateClient (**enum** avec un state ACTIVE ou INACTIVE)
 - Un taux tva
 - Un id
 - Un name
 - Un Total ca ht
 - Un Email (vérifier dans db.json les propriétés) si présentes
 - Un totalITTC() (méthode)



Fournitures du tp

- Une fake api pour récupérer des objets json correspondant à notre interface/model
- Un service pour faire des appels http
- Une liste des clients sur le path localhost/clients ‘ ‘
- 2 class css pour nos states

32

Formulaire & addOrder





Créer un formulaire (re)utilisable

Page add order

Ts

```
Public item = new Order()
```

Html

```
<app-form-order  
[init] = "new Order"  
(submitted) = "add($event)">
```

Page edit order

Ts

```
Public item$ = l'observable return by  
http.getItemById(id)
```

Html

```
<app-form-order  
[init] = "order existant"  
(submitted) = "update($event)">
```

Form order

Ts

```
@Input() init  
@Output() submitted
```

Html

```
Formulaire html
```

- Problématique : ce même formulaire doit être réutilisable sur plusieurs pages.
- Aussi, on verra que ce n'est pas ce composant qui fait appel au service dédié.
- On crée un dossier components dans ordersModule et on génère notre formulaire
 - **ng g component form-order**

page-add-order

Add Order

app-form-order

app-template-container

type

client

nbJours

tjmHt

tva

state

CANCELED

comment

Submit

En résumé

page-add-order
passe un new order
à app-form-order

page-edit-order
passe un order
existant à
app-edit-order



Html & ts de notre formulaire

ts

```
export class FormOrderComponent implements OnInit {
  public states = Object.values(StateOrder);
  @Input() init!: Order;
  constructor () {}

  ngOnInit (): void {}
}
```

```
<form>

  <div class="form-group">
    <label for="typePresta">type</label>
    <input type="text" class="form-control" id="typePresta">
  </div>

  <div class="form-group">
    <label for="client">client</label>
    <input type="text" class="form-control" id="client">
  </div>

  <div class="form-group">
    <label for="nbJours">nbJours</label>
    <input type="number" class="form-control" id="nbJours">
  </div>

  <div class="form-group">
    <label for="tjmHt">tjmHt</label>
    <input type="number" class="form-control" id="tjmHt">
  </div>

  <div class="form-group">
    <label for="tva">tva</label>
    <input type="number" class="form-control" id="tva">
  </div>

  <div class="form-group">
    <label for="state">state</label>
    <select class="form-control" id="state">
      <option fngFor="let state of states" [value]="state">{{ state }}</option>
    </select>
  </div>

  <div class="form-group">
    <label for="comment">comment</label>
    <textarea class="form-control" id="comment"></textarea>
  </div>

  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

- On recrée simplement notre property binding pour les states dans le ts ainsi que notre input pour initialiser un new order
- On affiche le form dans add-order component

type	
client	
nbJours	
tjmHt	
tva	
state	CANCELED



Afficher notre formulaire

- On va créer une instance de notre `<app-template-container>` en lui donnant pour titre `title="add order"` ainsi qu'une instance de notre `<app-form-order>` dans laquelle on ajoutera un input pour initialiser un `new Order()` comme on a pu le comprendre avec le slide sur la partie théorique

- **page-add-order.component.html :**

```
<app-template-container title="add order">
  <app-form-order [init]="item"></app-form-order>
</app-template-container>
```

- **page-add-order.component.ts :**

```
public item = new Order();
```

Deux façons de créer des formulaires avec Angular

Template Driven Form

Logique dans le HTML

Difficile à réutiliser

Reactive Form

HTML et logique TS séparés

Contrôle avancé

Fonctionne avec des observables
qui renvoient en temps réel les
données renseignées



Commit 32 : Formulaire & addOrder

- Retrouvez sur le dépôt Github de la formation le commit 32 concernant notre component btn réutilisable

33

ReactiveFormsModule



Demo Reactive Form

<https://stackblitz.com/edit/angular Ivy-4camqf>

Démo Form Builder

<https://stackblitz.com/edit/angular-ivy-ihgqpn>

Avantages de FormBuilder : une écriture simplifiée

```
< src/app/profile-editor/profile-editor.component.ts (instances)

profileForm = new FormGroup({
  firstName: new FormControl(''),
  lastName: new FormControl(''),
  address: new FormGroup({
    street: new FormControl(''),
    city: new FormControl(''),
    state: new FormControl(''),
    zip: new FormControl('')
  })
});
```

```
< r/profile-editor.component.ts (instances)

profileForm = this.fb.group({
  firstName: '',
  lastName: '',
  address: this.fb.group({
    street: '',
    city: '',
    state: '',
    zip: ''
  })
});
```



Doc ReactiveFormsModule

- Consultons un peu la documentation des Reactive Forms
- Il nous faut utiliser ReactiveFormsModule, nous choisissons de faire un **export** de celui-ci dans notre SharedModule
- Il nous faut ajouter la propriété `[formGroup]="form"` à notre balise form dans le HTML
- Il nous faut ajouter dans chacune des balises input de notre formulaire une propriété `formControlName=""` qu'on affecte en fonction de notre ts



Doc ReactiveFormsModule

```
<input formControlName="typePresta" type="text" class="form-control" id="typePresta">  
<input formControlName="client" type="text" class="form-control" id="client">  
<input formControlName="nbJours" type="number" class="form-control" id="nbJours">  
<input formControlName="tjmHt" type="number" class="form-control" id="tjmHt">  
<input formControlName="tva" type="number" class="form-control" id="tva">  
<select formControlName="state" class="form-control" id="state">  
    <option *ngFor="let state of states" [value]="state">{{ state }}</option>  
</select>  
<textarea formControlName="comment" class="form-control" id="comment"></textarea>
```



Initialiser notre Reactive Form

```
export class FormOrderComponent implements OnInit {  
    // on crée un tableau à partir de Enum StateOrder  
    public states = Object.values(StateOrder);  
    @Input() init!: Order;  
    // on initialise form de type FormGroup  
    public form!: FormGroup;  
  
    // On injecte FormBuilder  
    constructor(private fb : FormBuilder) { }  
  
    // initialise le form  
    ngOnInit(): void {  
        this.form = this.fb.group({  
            tjmHt: [this.init.tjmHt],  
            nbJours : [this.init.nbJours],  
            tva: [this.init.tva],  
            state: [this.init.state],  
            typePresta : [this.init.typePresta],  
            client: [this.init.client],  
            comment:[this.init.comment],  
            id: [this.init.id]  
        })  
    }  
}
```

- On utilise `private fb : FormBuilder` dans notre constructor
- On initialise une propriété `public form!: FormGroup`
- On initialise notre formulaire en utilisant `FormBuilder` (Ici on va donner le même nom que nos propriétés d'objet Order car on a besoin de `this.form.value`). Autrement il faut utiliser `map()` avant d'enregistrer en db. Or ici on pourra directement enregistrer `this.form.value` car cela correspond aux noms des propriétés de notre objet Order.
- On donne des valeurs par défaut
- Attention, si linitialisation du formulaire est dans le constructeur, la prop init vaut undefined. On a alors le choix entre `ngOnChanges` ou `ngOnInit()`. Mais comme on a besoin que d'une seule init alors on choisit `ngOnInit()`.

[ngValue] vs value

```
<div class="form-group">
  <label for="state">state</label>
  <select formControlName="state" class="form-control form-select" id="state">
    <option *ngFor="let state of states" [ngValue]="state">{{ state }}</option>
  </select>
</div>
```

[value] renvoie une string.

[ngValue] retourne tous types de datas (number, object) sans les convertir en string.
<https://tutorialsforangular.com/2021/01/24/using-value-vs-ngvalue-in-angular/>



Commit

- Commit étape 33 : FormBuilder and ReactiveFormsModule with FormControl and FormGroup

34

EventEmitter & router.navigate()





(ngSubmit)

- On veut récupérer les données du formulaire lorsque l'on clique sur submit et qu'elles viennent s'ajouter à tous nos orders.
- On va donc utiliser la directive `(ngSubmit)` qui va détecter quand le bouton sera cliqué et donc appeler une méthode `= "onSubmit () "`
- On va aussi changer la propriété `[value]` de notre `<option>` par `[ngValue]` qu'on peut utiliser dans un form. Elle nous permet cette fois-ci de bind notre value avec autre chose qu'un string uniquement
- **form-order.component.html :**

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

```
  <option *ngFor="let state of states" [ngValue]="state">{{ state }}</option>
```

```
(ngSubmit)="onSubmit()"
```

```
public onSubmit(){
  console.log(this.form.value);
}
```

```
▼ {tjmHt: 1200, nbJours: 1, tva: 20, state: 'OPTION', typePresta: null, ...} ⓘ f
  client: null
  comment: null
  id: null
  nbJours: 1
  state: "OPTION"
  tjmHt: 1200
  tva: 20
  typePresta: null
  ► [[Prototype]]: Object
```



EventEmitter

- On va créer notre méthode `onSubmit()`

```
public onSubmit(): void{
    console.log(this.form.value);
    this.submitted.emit(this.form.value);
}
```

- Ensuite on va initialiser notre `@output`:

```
@Output() submitted = new EventEmitter<Order>();
```

- `EventEmitter`, pourquoi ? Et bien parce qu'on a besoin d'un `Output` en premier lieu pour accéder à la propriété `submitted` depuis le composant parent puis d'un objet qui soit à la fois un objet et un observable chaud

Composant Parent

HTML

```
<app-enfant  
[propEnfant]='propParent'  
(notifAdd)="expression($event)">
```

TS

```
public expression(data){  
// service  
}
```

Composant Enfant

```
@Output notifAdd = new EventEmitter()
```

```
onSubmit(){  
    this.notifAdd.emit(this.form.value);  
}
```



EventEmitter

- Lorsque l'on arrive sur pageAddOrder, on doit pouvoir bind notre output avec une méthode (pour effectuer une action) mais on veut pas que la méthode soit triggered directement.
- On veut donc passer un flux avec notre output à notre méthode action() (ce flux on le récupère grâce à \$event). Etant donné que notre EventEmitter n'est pas initialisé, on ne récupère aucun flux au chargement de la page, c'est parfait pour nous car on veut que ce flux soit transmis uniquement au moment où l'on clique sur l'envoi du formulaire.
- On va utiliser notre méthode `onSubmit()` pour initialiser ce flux de notre EventEmitter lorsque l'on cliquera sur le bouton d'envoi, alors à ce moment nous pourrons récupérer le flux dans notre méthode action(\$event)



EventEmitter

```
(submitted)="action($event)">
```

Subscribe() en tâche de fond à un Observable qui ne doit pas être initialisé.

Puis, quand Observable reçoit un flux de données, le .subscribe déclenche action() et lui transmet le flux de données (\$event).

Documentation :

<https://angular.io/api/core/EventEmitter>



EventEmitter

- **page-add-order.component.html :**

```
<app-form-order [init]="item" (submitted)="action($event)"></app-form-order>
```

- On va ensuite créer notre méthode add qui fera un appel http post dans notre OrdersService :

- **orders.service.ts :**

```
public add(item: Order): Observable<any> {
    return this.httpClient.post<any>(` ${this.urlApi}/orders`, item);
}
```

- On va ensuite pouvoir créer notre méthode action() qui sera triggered par notre EventEmitter lorsqu'il aura reçu le flux (quand on appuie sur le bouton d'envoi, cela sera géré par notre méthode onsubmit())



EventEmitter

- **page-add-order.component.ts :**

```
public action(item: Order): void {
    this.ordersService.add(item).subscribe()
}
```

Pas de subscribe, pas d'appel HTTP.

Que fait la classe HttpClient ?

Appel HTTP avec .subscribe

Les méthodes GET/PUT/POST/DELETE de la classe HttpClient font un appel vers le endPoint

l'API retourne une data

HttpClient crée un New Observable

Initialise cet observable en appelant la méthode next avec le flux de données



Router.navigate()

- Tout fonctionne, en revanche on aimerait être redirigé directement vers notre liste pour voir notre liste mise à jour sur le front, il existe pour ça `router.navigate()` qui nous permet de faire des redirections dans le typescript. Il suffit de faire l'import de `Router` et de l'injecter dans notre constructor pour utiliser `this.router.navigate()`

Page-add-orders.component.ts :

```
constructor(private ordersService: OrdersService, private router: Router) {}

public action(item: Order): void {
    this.ordersService.add(item).subscribe(() => {
        this.router.navigate(['orders']);
    });
}
```



Commit

- Commit étape 34: EventEmitter and Router.navigate()

35

Validators and CSS





Validators and CSS

- On veut mettre des validators pour contrôler certains champs et les rendre obligatoires. On va donc mettre nos validators sur les form-control de notre form-order
- **form-orders.component.ts :**

```
    typePresta : [this.init.typePresta, Validators.required ],  
    client : [this.init.client, [Validators.required, Validators.minLength(2)]],
```

- On va ensuite faire le nécessaire côté html pour rendre l'interface utilisateur plus agréable

Documentation : <https://angular.io/api/forms/Validators>



Validators and CSS

- Dans un premier temps on va s'occuper de désactiver le bouton submit lorsque le formulaire n'est pas valide et de l'activer si le formulaire est correctement remplis : (true or false)

```
<button [disabled]="form.invalid" type="submit" class="btn"
```

- On va ensuite ajouter des messages d'erreur "champ invalide" avec deux class bootstrap `class="alert"` & `class="alert-danger"` et nous voulons afficher ces messages d'erreurs quand notre champ est invalide et que l'on perd le focus sur celui-ci :

```
<div class="form-group">
  <label for="typePresta">type</label>
  <input formControlName="typePresta" type="text" class="form-control "
    id="typePresta">
    <div class="alert alert-danger" *ngIf="form.controls['typePresta'].invalid &&
      form.controls['typePresta'].touched">champ invalide</div>
</div>
```



Validators and CSS

- On peut ajouter très facilement du css, il suffit de regarder les class ajoutées par angular en inspectant les éléments de la page
- **form-order.component.scss :**

```
Form{  
  .ng-valid.ng-touched{  
    border: 3px solid var(--app-success);  
  }  
  .ng-invalid.ng-touched{  
    border: 3px solid var(--app-error);  
  }  
}
```



Commit

- Commit étape 35 : Validators and CSS

36

TP Edit order





TP Edit order

C'est à vous!

Vous devez désormais développer la partie edit order et faire en sorte que vous puissiez avoir un icône dans le tableau du CRM qui vous permettra d'accéder au formulaire pour modifier un item en particulier.



TP Edit order

- Ajouter app-template-container avec un nouveau titre
- Faire en sorte d'utiliser le formulaire sur page-edit-orders
- Déclarer item et action dans la logique du component (injecter OrdersService dans le constructor)
- Préparer la redirection
- Ajouter un header Action devant les autres colonnes
- Créer une cellule <td> pour l'edit avec un événement cliquable qui renverra à la méthode goToEdit()
- Créer goToEdit()
- Injecter le router dans le constructor de page-list-orders
- Dans orders-routing.module modifier le path edit pour qu'il prenne un paramètre id dans l'url



TP Edit order

En partant de page-edit-orders:

- Récupérer id dans l'url (ActivatedRoute)
<https://angular.io/api/router/ActivatedRouteSnapshot>
- Appeler this.ordersService.getItemById(id)
- Dans le subscribe vous initialisez item
- Attention de ne pas binder init trop tôt dans le html (réponse asynchrone)
- Si possible, utilisez le pipe async dans le html sur item\$



Commit

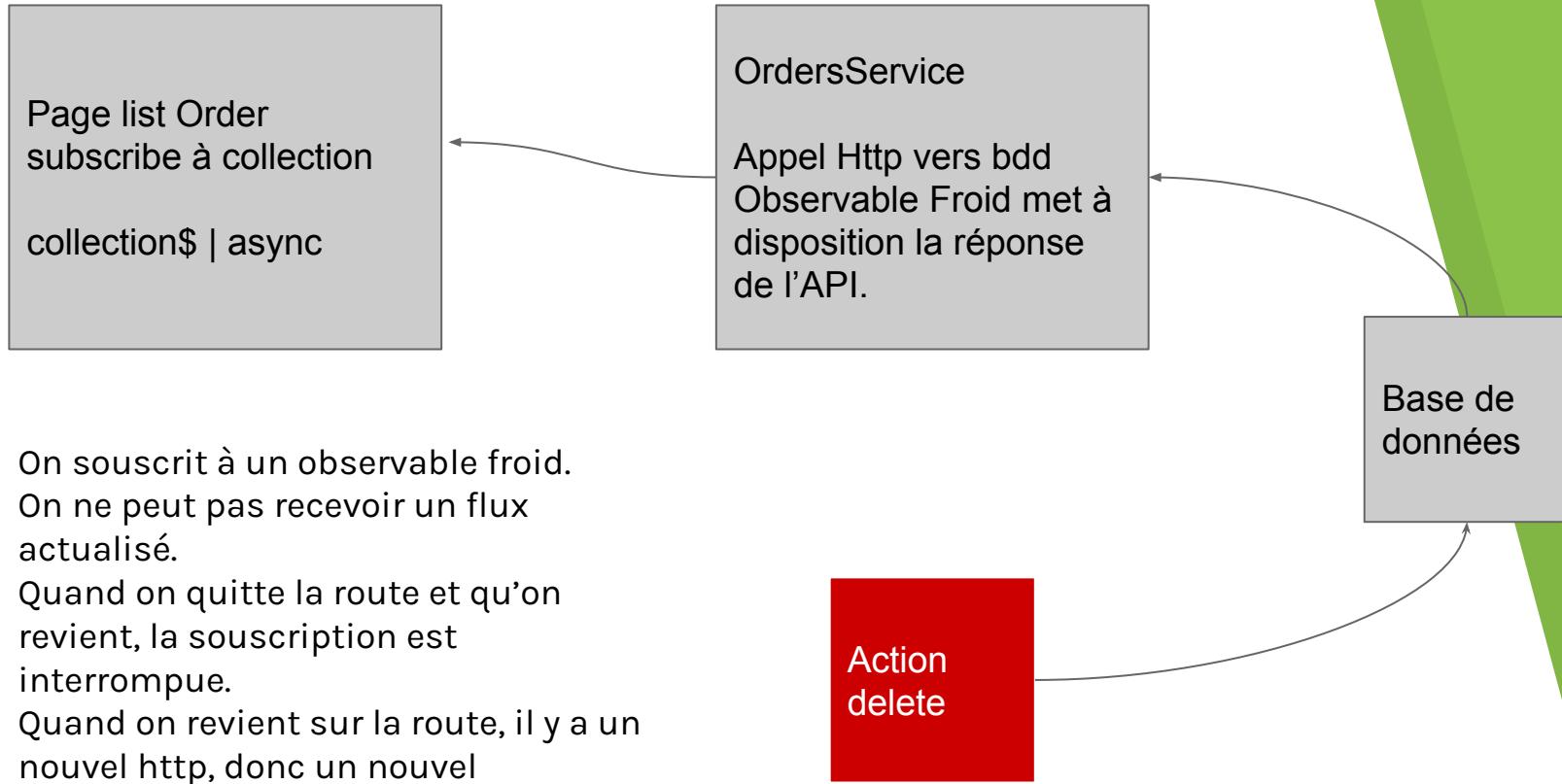
- Commit chapitre 16 : TP Edit order

37

TP delete order



Première instantiation de page-list-order



Besoin d'un observable chaud

OrdersService

New BehaviorSubject

```
public refreshCollection(){
    // appel this.http.get déplacé ici
    // subscribe
    // résultat utilisé pour initialisé
    collection$(BehaviorSubject)
}
```

```
public refreshCollection(): void{
    this.http.get<Order[]>(`${this.urlApi}/orders`).pipe(
        map((tab)=>{
            return tab.map((obj)=>{
                return new Order(obj)
            })
        })
    ).subscribe((data)=>{
        this.collection$.next(data)
    })
}
```

Dans le constructor de OrdersList

```
constructor(  
    private http: HttpClient  
) {  
    this.refreshCollection()  
}
```

Page List Orders
(private ordersService)

Service
refreshCollection()

A la première visite sur pagelist Orders, le service est instancié.
A ce moment-là la fonction refreshCollection est appelée

Appel asynchrone



TP Delete order

- Ajoutez un icône delete qui lorsqu'on va cliquer dessus va supprimer un item côté back et mettre à jour la page.

Seulement on ne va pas quitter la route donc le web component ne sera pas rechargé et comme on a un pipe async qui souscrit à un observable classique d'rxjs, même si on met à jour notre collection côté front ça ne nous permettra pas d'actualiser notre tableau en temps réel on devra quitter la page puis revenir pour voir le résultat.



TP Delete order

- Gérez cette problématique et faites en sorte que la suppression d'un élément soit visible après le click sur l'icône de suppression.
- Pensez également à vérifier si add et edit fonctionnent toujours sinon rendre ces pages de nouveau fonctionnelles



TP Delete order

- Commit chapitre 17: tp delete order



Formation Angular

Initiation

Nous contacter :

Christophe Guérout

c.gueroult@coderbase.io

+33 6 56 85 84 33



CODERBASE IT

WE THINK HUMAN FIRST