AU PROGRAMME:

- le client angular ng
- les composants
- le databinding
- le cycle de vie
- le change-detection

MA 1ÈRE APP!

SANS PROXY :

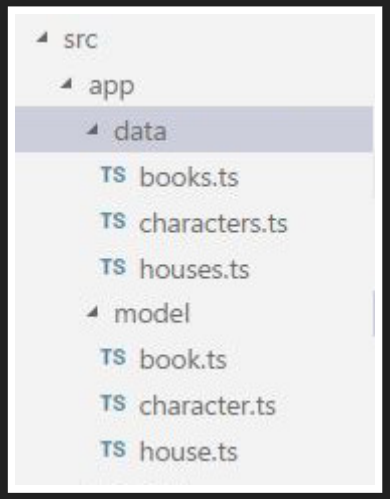
```
ng new hold-the-dom --skip-tests --style=scss
cd hold-the-dom
ng serve --open
```

RÉSEAU CGI :

```
ng new hold-the-dom --skip-install --skip-tests --style=scss
cd hold-the-dom
echo https-proxy=http://fr-proxy.groupinfra.com:3128 > .npmrc
echo proxy=http://fr-proxy.groupinfra.com:3128 >> .npmrc
npm install
ng serve --open
```

IMPORTER LES DONNÉES GOT

- copier les répertoires data et model



INSTALLER ANGULAR MATERIAL

```
npm install --save @angular/material
npm install --save @angular/cdk @angular/animations
```

IMPORTER UN THEME EXISTANT

- dans
  style.scss

```
@import "~@angular/material/prebuilt-themes/indigo-pink.css"
```

- deeppurple-
  amber
- purple-green
- pink-bluegrey

RAJOUTER LE MODULE GRID

- dans app.module.ts:

```
import { MatGridListModule } from '@angular/material/grid-li
@NgModule({
...
imports: [MatGridListModule],
...
})
```

TEST

- copier dans app.component.html

```
<mat-grid-list cols="4" rowHeight="2:1" gutterSize="20px">
  <mat-grid-tile [style.background]="'lightblue'">toto</mat-gr
  <mat-grid-tile>titi</mat-grid-tile>
  <mat-grid-tile [style.background]="'lightblue'">tata</mat-gr
  <mat-grid-tile>tonton</mat-grid-tile>
  <mat-grid-tile>tutu</mat-grid-tile>
  <mat-grid-tile [style.background]="'lightblue'">tati</mat-gr
  <mat-grid-tile>prout</mat-grid-tile>
</mat-grid-list>
```

- ng serve --open ===> vérifier le rendu

GIT COMMIT!

Faire des commit intermédiaires pour voir facilement les fichiers générés

TEMPLATE EXPRESSIONS {{

ou interpolation ou la moustache

- pas de changement avec AngularJS
- pas de modif dans la moustache {{ toto = 'titi'}}
- concat. et arithmétique {{ 'toto' + 'titi' }}
- opérateur ternaire {{ grosseCom ? 'caca' : 'pipi' }}

STRUCTURAL DIRECTIVES

```
*ngIf="lines"
*ngFor="let l of lines"
[ngSwitch]
```

1 seule directive par élément

ATTRIBUTE DIRECTIVES

```
ngStyle="{ 'style1': i==5, 'style2': true }"
ngClass="{ 'classe1': i==5, 'classe2': true }"
```

Exercice 1

NG-CONTAINER

Elément transparent pour le DOM

Le code suivant plante :

```
<tbody>
    <tr *ngIf="gridData1" *ngFor="let row of gridData1.rows">
    <tr *ngIf="gridData2" *ngFor="let row of gridData2.rows">
```

Une solution :

```
<tbody>
  <ng-container *ngIf="gridData1">
    <tr *ngFor="let row of gridData1.rows">
```

NGTEMPLATE

- code HTML à afficher que dans certains cas
- par exemple dans le "else" du ngIf :

```
<div *ngIf="!inProgress; else progressBar">
...
<ng-template #progressBar>
```

code plus lisible

Exercice 2

LE STANDARD HTML5

WEB COMPONENT

CUSTOM ELEMENT

```
class Character extends HTMLElement {
  ...
}
customElements.define('character', Character);
```

SHADOW DOM

isolation DOM/CSS

```
character-container = element.attachShadow({mode: open})
.character-inner {
  background-color: #dedede;
}
```

TEMPLATE

html caché, réutilisable

```
<template id="character-template">
  <div class="character-inner">
    <h2 class="character-name"></h2>
    <span class="character-gender"></span>
  </div>
</template>
```

HTML IMPORT

importer un template HTML à partir d'un fichier séparé

```
<link rel="import" href="./character/character.html">
```

ANGULAR COMPONENTS

STRUCTURE

- 1 fichier html
- 1 fichier scss,
- 1 fichier ts
  - selector: nom de la balise HTML
  - template: référence vers le fichier HTML
  - style: référence vers le fichier SCSS
  - classe controller, annotée @Component
- importé/exporté dans un module

DUMB COMPONENTS

- Composant de *présentation*
- Indépendant du contexte
- Pas de logique métier
- Réutilisables

Exemple:

tous les composants angular-material

SMART COMPONENTS

- *Containers*
- Connaissent le contexte
- Exécutent de la logique métier
- Peu réutilisables

Exemple:

tous les composants qui modifient vos objets métiers directement

Exercice 3.1

DATABINDING

NGMODEL

- AngularJS databinding

```
<input ng-model="name">
```

- Equivalent Angular 2+

```
<input [(ngModel)]="name">
```

2 BANANES DANS UNE BOÎTE

[()]


ngModel

[] === PASSER DES DONNÉES

```
<input [value]="name">
```

() === ECOUTER UN ÉVÈNEMENT

```
<input (change)="name = $event">
```

COMPOSANTS PERSONNALISÉS AVEC @INPUT

```
<character-grid [characters]="characters">

@Input() characters: Character[];
```

exercice 3.2

exercice 4

EVÉNEMENTS UTILISATEUR @OUTPUT

```
@Output() select = new EventEmitter<Character>();

onSelect(it: Character) {
  this.select.emit(it);
}
```

exercice 5

DIRECTIVES PERSONALISÉES

```
<p appHighlight>Highlight me!</p>


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

NGCONTENT

Transclusion:

- quand on veut un composant "conteneur" assez générique
- on place le contenu HTML dans un placeholder

EXEMPLE AVEC ANGULAR-MATERIAL

```
<mat-grid-tile *ngFor="let character of charactersAsync">
  <!-- <ng-content> -->
  <div>{{ character.name }}</div>
  <!-- </ng-content> -->
</mat-grid-tile>
```

Exercice 6

LIFECYCLE

ONINIT

- appelé après le constructeur
- privilégier cette méthode pour
  - s'abonner à des observables
  - manipuler les attributs @Input
  - appeler des services
  - appeler des requêtes http, etc.

ONDESTROY

- peut être utilisé pour désallouer des ressources
- unsubscribe sur les observables qui auraient été souscrits dans le code
- ex: autocomplete

DOCHECK

- appelé énormément de fois, à chaque évènement du DOM en fait
- à utiliser très prudemment seulement si Angular ne détecte pas un changement particulier
- utilisé pour le déboggage

ONCHANGES

- à chaque évènement qui va effectuer une màj du composant
- utilisé pour le déboggage... pas d'autres idées

AFTERCONTENTINIT, AFTERCONTENTCHECKED, AFTERVIEWINIT, AFTERVIEWCHECKED

voir la doc!

Exercice 6

CHANGE DETECTION

CHANGEMENTS

le state (les objets JS) changent

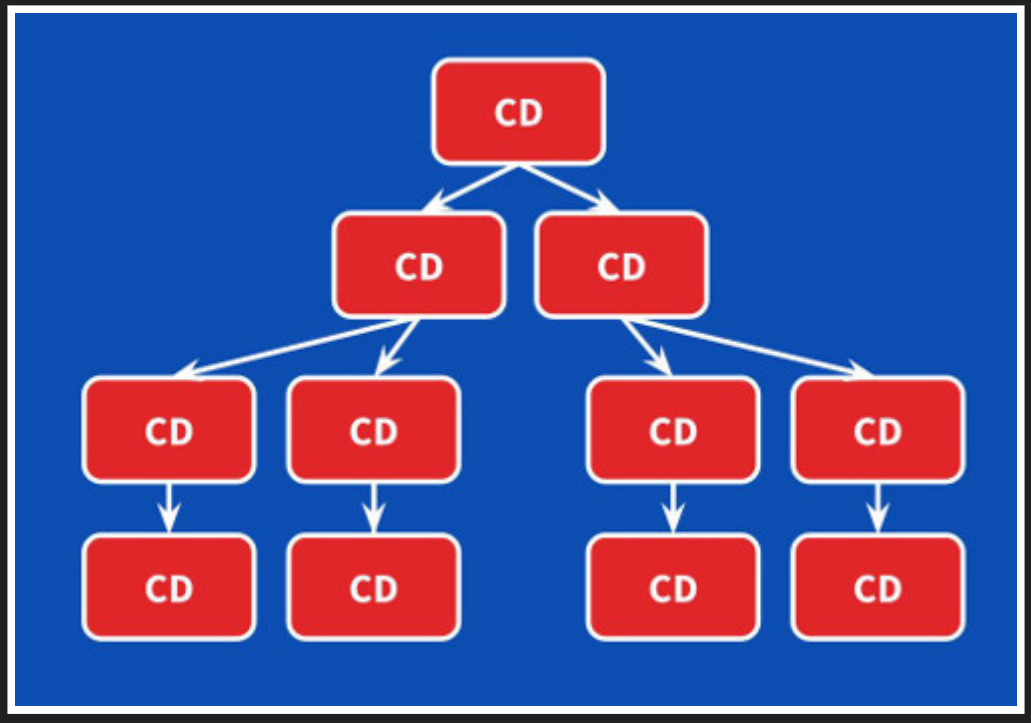
=> on veut mettre à jour la vue

- Events
- Requête HTTP
- setTimeout(),
  setInterval()

CHANGE DETECTORS

- observe les changements du state
  - *les attributs des controlleurs*
- Angular fait le tour des composants en 1 passe
  - en partant du root component jusqu'au feuilles

Unidirectional dataflow



CD STRATEGY

Angular observe par défaut les modifs

- référence des objets : `myObj = new Obj()`
- des propriétés des objets : `myObj.name = 'toto'`

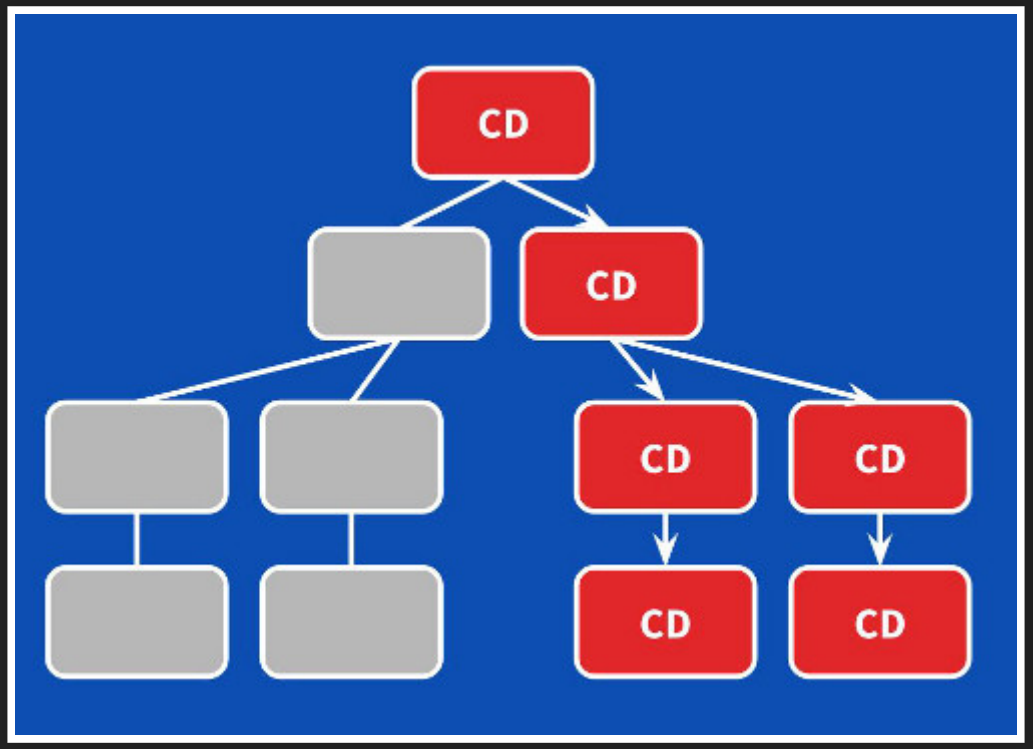
c'est ça qui coûte cher

ChangeDetectionStrategy.OnPush

```
@Component({
  template: `
    <h2>{{myObj.name}}</h2>
  `,
  changeDetection: ChangeDetectionStrategy.OnPush
})
class VCardCmp {
  @Input() myObj;
}
```

- Angular n'observe que les changement de références
- si `myObj.name` est modifié, rien ne se passe

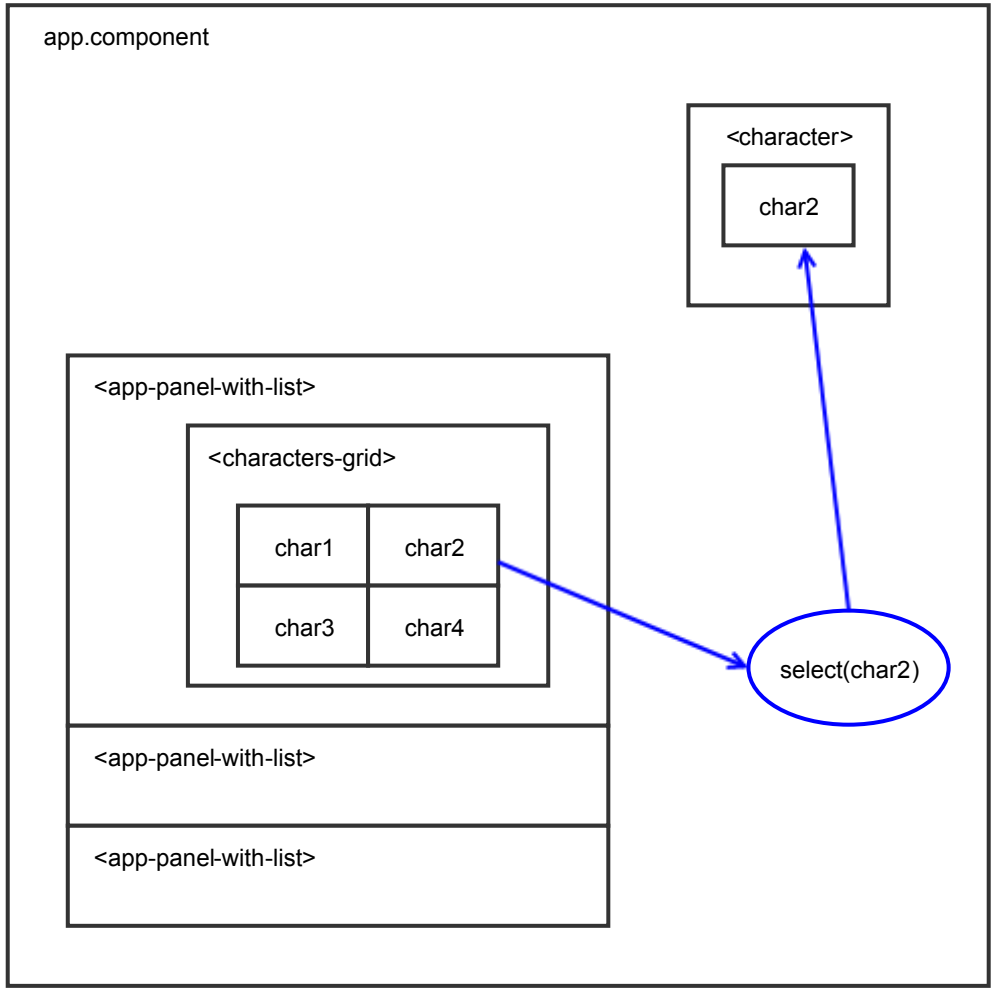
Plus performant!

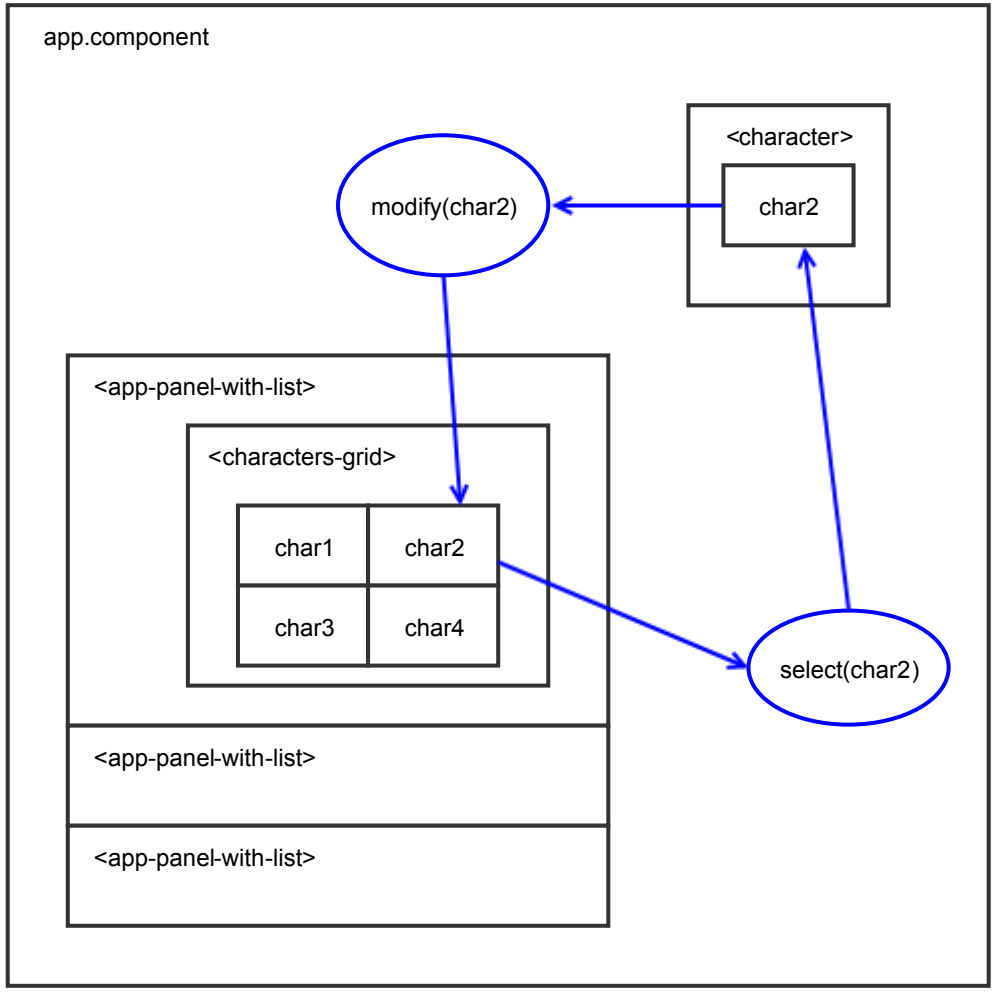


Mais ça ne marche pas tout le temps!

- il faut s'assurer que rien ne modifiera `myObj.name`
- si on veut modifier le nom, créer un nouvel obj systématiquement
- utiliser une lib comme Immutable.js

Exercice 8


app.component
<character>
char2
<app-panel-with-list>
<characters-grid>
char1
char2
char3
char4
select(char2)
<app-panel-with-list>
<app-panel-with-list>


app.component
<character>
char2
modify(char2)
<app-panel-with-list>
<characters-grid>
char1
char2
char3
char4
select(char2)
<app-panel-with-list>
<app-panel-with-list>

OBSERVABLES / CHANGEDETECTORREF

on va allez se coucher plutôt...

Référence:

https://blog.thoughtram.io/angular/2016/02/22/angular-2-change-detection-explained.html#what-causes-change