

* Présentation:

* Single Page Application

naviguer et feel sans rechargement de la page

* TypeScript

" " + variable + " " \Leftrightarrow " \${variable} "

let tabo: number[] let tabo: Array<number> =

enum Fruit {Banane, Kaki}

let R: Fruit = Fruit.Kaki // R = 1

let s: string = Fruit[1] // s = Kaki

Cas

let uneValeur: any = " "

let longueur: number = (uneValeur as string).length

Fonctions

paramètre optionnel : param?

par défaut fum (x=10, y)

let uneValeur: any = " "

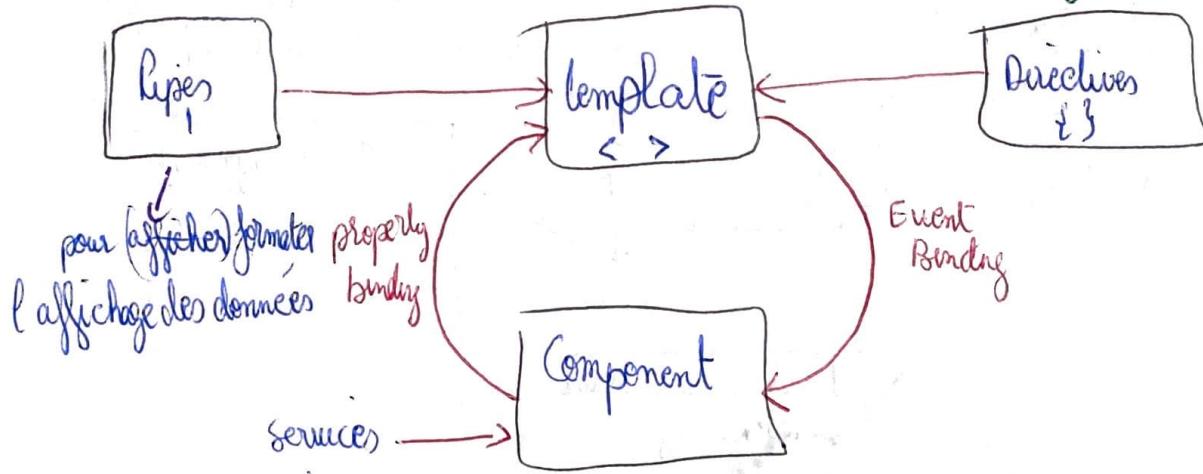
let longueur: number = (uneValeur as string).length

Angular

Structure d'un projet Angular

app.component.ts

Les fichiers .spec sont des tests unitaires pour les fichiers sources. Ils sont exécutés à l'aide du framework Jasmine via Karma avec la commande ng test



Modules:

Grâce aux modules Angular, on peut subdiviser l'app en plusieurs composants réutilisables

app Angular comporte au moins un module racine AppModule

@NgModule est un décorateur qui prend en param un objet JS qui contient les métadonnées dont les propriétés décrivent le module

Syntaxe :

* declarations : lit des composants constituant le module. tout ce qui se trouve dans une déclaration est visible pour les applications utilisant ce module.

* imports : rendre autres modules visibles dans ce module.

* exports : liste ou de toutes les déclarations que vous souhaitez rendre disponibles (visibles) au autres modules qui importe ce module

* providers : enregistrer les services à disposition pour id. ces services seront accessibles dans tout le module

* bootstrap : principalement utilisé pour le chargement dynamique de composants

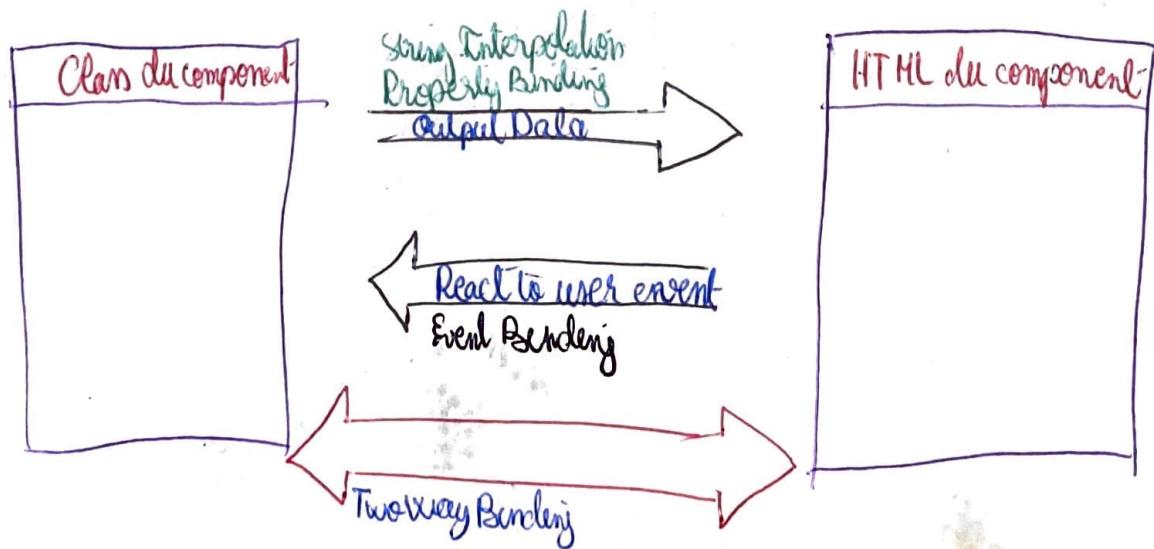
Composants :

* utilisés pour contrôler une rue ou une partie de l'écran

Import {Component} from '@angular/core'; importez l'objet Component

Databinding

La liaison est responsable de la communication entre nos données et notre UI



4 types

- liaison d'interpolation
- liaison de propriété
- liaison d'événement
- liaison bidirectionnelle

{ {variable} } if the variable can be undefined {{user?.name}}

<p>{{user.name}}</p>

<p [textContent] = "user.name"></p>
propriété DOM

→ est réalisé pour associer un evt à une méthode de classe

<button (click)="refreshUsers()"> Refresh user list </button>

ts: export class UserComponent {
users: any = [];
refreshUsers() {
this.users = [{name: 'Robert'}, {name: 'Felix'}];
}}

Quelques événements:

(ngSubmit) : avec les forms pour réagir à la soumission

(mouseenter), (mouseleave) : réagir à l'entrée ou la sortie de la souris à un elt

(keyup) : détecter la pression d'une touche de clavier <input (keyup)="onKeyUp(\$event)">

(change):

local variable

var local

<input type="text" [(name)]="name" {{name.value}}>

sont de vars qui on peut déclarer dans le template et non accessible directement par

to

→ synchroniser automatiquement les données entre le modèle (, to) et la view

Souvent utilisé avec [(ngModel)]="view"

<input type="text" [(ngModel)]="user.name"/>

<input type="text" [(ngModel)]="user.name" (ngModelChange)="user.name=\$event"/>

> "user.name=\$event"

ngModel nécessite l'importation du module FormsModule

Interaction entre composants

(fils)

on peut ajouter le selecteur d'un 1^{er} composant dans le template d'un 2^{eme} composant (parent)

- @Input permet à un fils de récupérer les données de son parent

- @Output .. parent .. - fils

```
userComponent  
@Component({  
  selector: 'app-user',  
  template: <app-user [user] = "selectedUser">  
    </app-user>
```

```
})  
export class UserComponent {  
  selectedUser: User = {id: 1, name: 'salma'};
```

```
userComponent  
@Component({  
  selector: 'app-user',  
  template: <div>{{user.name}}</div>  
})
```

```
export class UserComponent {  
  @Input() user: User;
```

En Angular, les données entrent dans un composant via des propriétés, et en sortent via des éts.

Si on veut remplacer la propriété par un terme différent

@Input('user') InternalUser: User

Les éts sont un mécanisme permettant à un composant de modifier son parent et au parent de gérer modifications pour émettre un evt:

1- Déclarer l evt en l annotant @Output

2- Crée un Event Emitter

3- Émettre un evt dans une méthode

Parent

template :

```
<app-user (userSelected)="modifyUser($e)">
</app-user>
```

```
export class UserComponent {
  modifyUser($event) {
    log('---')
  }
}
```

Child

template

```
<button (click)="clickedUser()">
</button>
```

```
export class UserComponent {
  @Output userSelected = new EventEmitter();
  clickedUser() {
    this.userSelected.emit(this.user)
  }
}
```

Directives

structurelles : modifier la mise en page en manipulant DOM

2 types : d'attribut modifier le comportement ou l'apparence d'un élément DOM existant

*ngIf

```
<div *ngIf="condition; else #elseBlock">
  <div> text
</div>
<ng-template #elseBlock>
  else block
<ng-template #elseBlock>
  non autre b
```

i est l'indice

*ngFor

avoir l'index : <div *ngFor="let var of vars; let i=index">

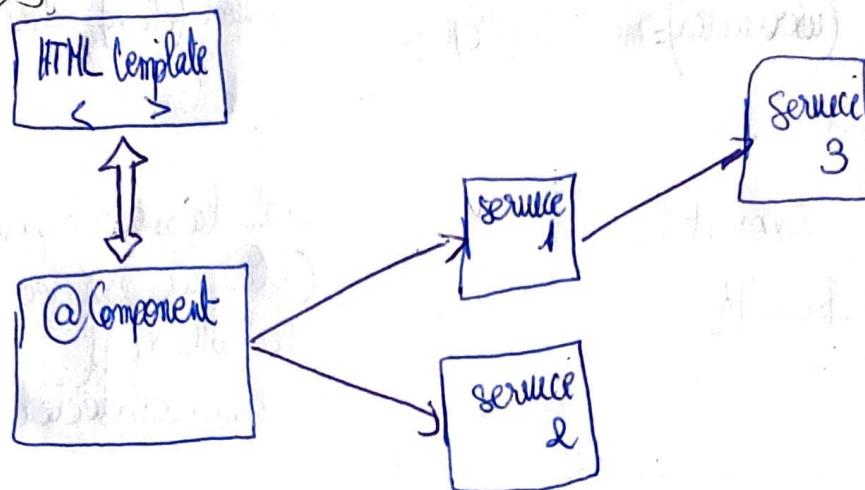
```
{ {i} }</div>
```

[ngStyle]: pour appliquer dynamiquement style CSS

pour gérer dynamiquement les classes CSS

```
<li [ngClass]="{ 'list-group-item': true,
  'list-group-item-success': complexe > 0,
  'danger': complexe < 0 }" >
```

Services



Routage :

Pour ajouter la fonctionnalité du routage :

1. import { RouterModule, Router } from '@angular/router';

2. const appRoutes: Routes = [

{ path: 'films', component: ListFilmsComponent },

{ path: ' ', component: DefaultFilmComponent }

]

3. déclarer RouterModule.forRoot(appRoutes) dans imports [,
sous AppModule]

4. <router-outlet></router-outlet>

* router.navigate(['films'])

* Si { path: 'film/:id', component: DetailsFilmComponent }

alors dans DetailsFilmComponent:

ngOnInit() {

const id: number = this.route.snapshot.params['id'];

et pour url [routerLink] = ['/film', film.id]

Guards:

gérer l'accès aux endpoints (authorizations, authentication, ... etc)

```
export class AuthGuardService implements CanActivate {  
  canActivate(route: ActivatedRouteSnapshot,  
             state: RouterStateSnapshot):
```

```
    Observable<boolean> | promise<boolean> | boolean {
```

```
      if (this.authService.auth) { return true; }
```

```
      else { this.router.navigate(['/auth']); }
```

```
}
```

```
}
```

```
{ path: 'admin', component: AdminComponent, canActivate: [AuthGuardService] }
```

HTTP

1. il faut importer le module HttpClientModule

* Observable émet des informations auxquelles on souhaite réagir.

* à cet Observable on associe un **Observer**-un bloc exécuté à chaque émission d'info -
à cet Observable on associe un **Subject**- observable avec plusieurs observers **new Subject<any>()**.

Méthodes

```
get HttpClient.get(url: string): Observable<Response>
```

```
post HttpClient.post(url: string, body: any): " " " "
```

```
put HttpClient.put(url: string, body: any): " " " "
```

```
Delete HttpClient.delete(url: string): Observable<Response>
```

`Rttp.get(url: string).subscribe(response: Response) => { ... });`

A handwritten blue arrow pointing to the right, indicating the direction of the next section.

`Http.get(url:string).toPromise().then((response) => { -- }) ;`

Formulaires

piloter par template

2 méthodes pour créer
template : piloter par code

```
<form (ngSubmit)="register(userForm)" # userForm="ngForm">
  <div>
    <input name="username" required minlength="3" [(ngModel)]="user.username">
  </div>
  <button type="submit">Register </button>
</form>
```

$\langle \text{1 form} \rangle$

at's:
registered (form:

```
const username: string = form.value['username'];
```

}

plusieurs attributs de validation:

valid = so le champ valide

Villa presline f derles

valid = vrai
presline + derlqy
derlqy = false jusqu'à ce que user modifie la valeur du champ
soit entré dans le champ

~~1. $\frac{1}{2} \times 10^3$ J = 500 J~~

length = 11 mm

touched - A is near the

$\text{Mg}^{2+} + \text{H}_2\text{O} \rightarrow \text{Mg(OH)}_2$

Valeur : valeur du champ

Yakut - River of Camp

2009-2010

eggshells ...

cross :-

Chu