UBALIA

AJC
FORMATION

Formation : Angular

Sommaire :

Introduction

Définition

- **Package Manager :** Fournit une méthode d'installation de nouvelles dépendances, appelées paquets, gère l'emplacement des paquets et offre la possibilité de publier des paquets.

Les gestionnaires de paquets les plus connus sont npm, pnpm et yarn.

Package management basics

Installation de npm

- **NodeJS :** Environnement d'exécution asynchrone piloté par événements.

Node.js est un environnement d'exécution JavaScript open-source et multi-plateformes.

Node.js

Installation d'Angular CLI

- **Angular CLI :** Interface en ligne de commande utilisé pour initialiser, développer, maintenir des applications Angular directement via la ligne de commande

Angular CLI est utilisé pour créer des projets, générer du code d'application et de bibliothèque et effectuer diverses tâches de développement.

CLI Overview and Command Reference - Setting up the local environment and workspace

Environnements de développement

TypeScript

Définition

- **TypeScript :** Langage de programmation fortement typé s'appuyant sur JavaScript.

TypeScript c'est JavaScript avec une syntaxe pour les types.

TypeScript

Types en TypeScript

- Boolean
- Number
- String
- Array
- Any
- Void
- Enum

Types primitifs

```
var isNoon: boolean;
```

```
var simple: string = 'string simple quote';
var double: string = "string avec double quote";
```

```
let anInt: number = 6;
let decimal: number = 2.9;
let hex: number = 0xa5d2;
```

Autres types

```
var names: string[] = ["Pierre", "Paul", "Jacques"];
var otherNames: Array<string> = [" Pierre", "Paul", "Jacques"];
```

```
var variableAny: any = "Sylvain";
variableAny = 2018;
```

```
enum WaterState{
   solid,
   liquid,
   gaz
}

var waterState: WaterState = WaterState.liquid;
```

Fonctions

```
function greeter(name: string): string {
   return "Bonjour, " + name;
}
```

Paramètres optionnels

```
function greeter(name: string, lastname?: string): string {
   var fullName: string = name;
   if(lastname)
       fullName += " " + lastname;

   return "Bonjour, " + fullName;
}
```

Paramètres par défaut

```
function greeter(name: string, lastname: string = ""): string {
   var fullName: string = name;
   if(lastname !== "")
       fullName += " " + lastname;

   return "Bonjour, " + fullName;
}
```

Interfaces

```
interface IPerson{
   firstName: string;
   lastName: string;
}

var myVar: IPerson =  { firstName:"Sylvain", lastName: "Sidambarom"}
```

Classes

```
class Personnage {
   public fullname: string;

   constructor(firstname: string, lastname: string) {
       this.fullname = `${firstname} ${lastname}`;
   }

   public greet(name?: string): string {
       if(name)
           return "Bonjour " + name + "! Je m'appelle " + this.fullname;

       return "Bonjour! Je m'appelle " + this.fullname;
    }
}

let vendeur = new Personnage("Pierre","Paul");
```

Modules - export

```
export module Module1{
    export class Person{
        constructor(){
        }

        greeting(): void{
            greeting("Module1.print()");
        }
    }
    export const pi: number = 3.14;

    export interface IGreeter{
        greeting():void;
    }

    export function farwell(){
        alert("Bye!! You're just called a method");
    }
}
```

Modules - import

```
import {Module1} from './app/module1'
let md = new Module1.Person();
md.greeting();
let person : Module1.IGreeter = md;
```

Arrow functions

```
class MyClass {
    name = "MyClass";
    getName = () => {
      return this.name;
    };
}
```

Decorators

- **Decorator :** Fonction pouvant être attachée aux classes et à leurs membres, tels que les méthodes et les propriétés.

```
const addFuelToRocket = (target: Function) => {
  return class extends target {
    fuel = 100
  }
}

@addFuelToRocket
class Rocket {}
```

```
const rocket = new Rocket()
console.log((rocket).fuel) // 100
```

A practical guide to TypeScript decorators

Templates

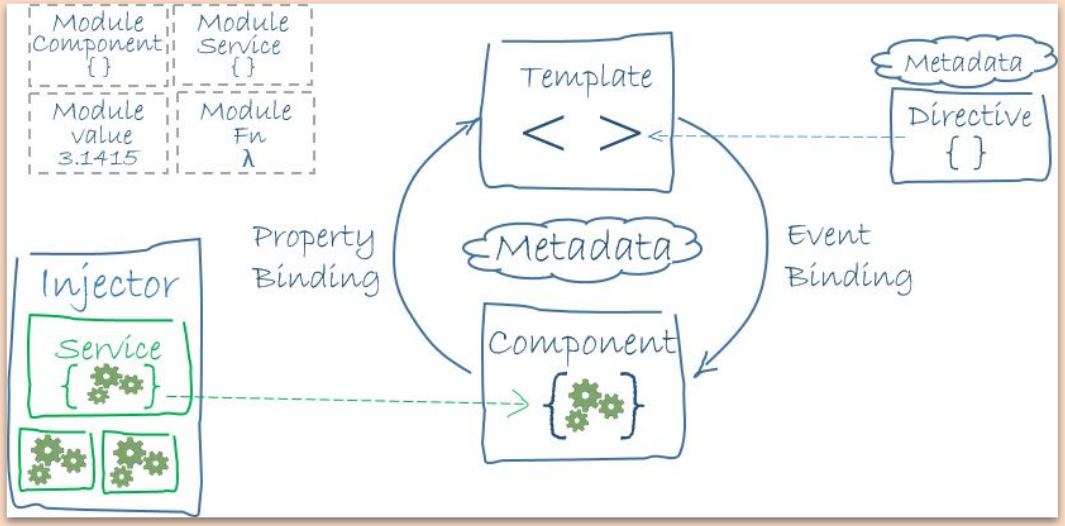
Définition

- **Angular :** Plateforme de développement, construite sur TypeScript, comprenant un cadre basé sur des composants pour la création d'applications web évolutives.

Angular est un framework permettant de créer des applications client en HTML et en JavaScript ou TypeScript.

What is Angular? - ARCHITECTURE OVERVIEW

Éléments principaux d'Angular

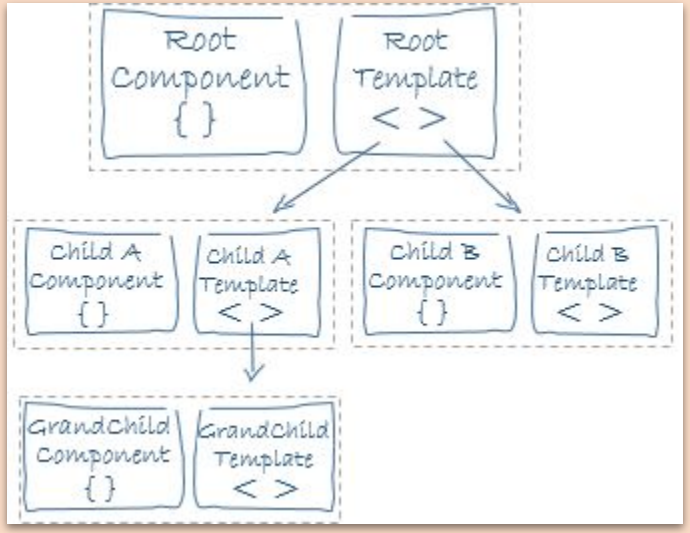


ARCHITECTURE OVERVIEW

Types en TypeScript

• **Modules**	Classe avec un décorateur @NgModule
• **Components**	Contrôle une partie de l'écran, appelée vue
• **Templates**	Forme de HTML indiquant comment rendre le composant
• **Metadata**	Indique comment traiter une classe
• **Data binding**	Coordonne les parties d'un template avec celles d'un composant
• **Directives**	Instructions de rendu des templates
• **Services**	Englobe toute valeur, fonction ou caractéristique nécessaire
• **Dependency injection**	Fournit les dépendances dont la classe a besoin

Architecture d'un template



Templates and views

Exemple de template

```
<h2>Hero List</h2>

<p><em>Select a hero from the list to see details.</em></p>
<ul>
  <li *ngFor="let hero of heroes">
    <button type="button" (click)="selectHero(hero)">
      {{hero.name}}
    </button>
  </li>
</ul>

<app-hero-detail *ngIf="selectedHero" [hero]="selectedHero"></app-hero-detail>
```

Interpolation

L'interpolation consiste à intégrer des expressions dans du texte balisé.

Par défaut, l'interpolation utilise les accolades {{ et }} comme délimiteurs.

```
currentCustomer = 'Maria';
```

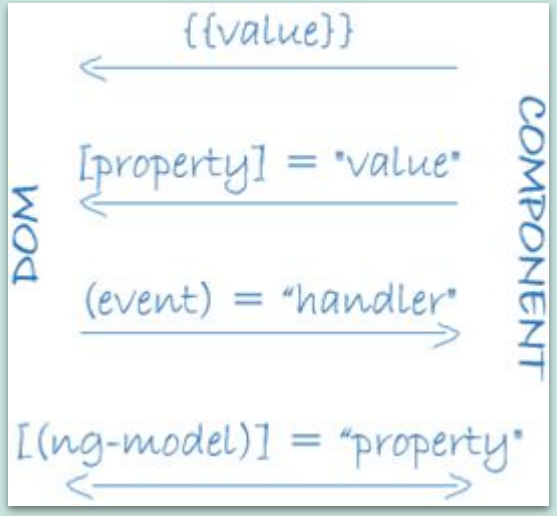
```
<h3>Current customer: {{ currentCustomer }}</h3>
```

Data binding

Angular prend en charge la liaison des données bidirectionnelle, permettant de connecter les parties d'un template avec les parties d'un composant.

```
<app-hero-detail [hero]="selectedHero"></app-hero-detail>
<button type="button" (click)="selectHero(hero)">
  {{hero.name}}
</button>
```

```
<input type="text" id="hero-name" [(ngModel)]="hero.name">
```



Directives structurelles

Les directives structurelles modifient la présentation en ajoutant, supprimant et remplaçant des éléments dans le DOM.

```
<li *ngFor="let hero of heroes"></li>

<app-hero-detail *ngIf="selectedHero"></app-hero-detail>
```

Structural directives

Pipes

Les pipes permettent de déclarer des transformations de valeur d'affichage dans le template HTML.

La transformation se fait par l'utilisation de l'opérateur pipe (|).

```
<h3>Current customer: {{ currentCustomer }}</h3>
```

```
<!-- Default format: output 'Jun 15, 2015'-->
<p>Today is {{today | date}}</p>

<!-- fullDate format: output 'Monday, June 15,
2015'-->
<p>The date is {{today | date:'fullDate'}}</p>
```

Composants et services

Après création du composant

- Dossier au nom du composant
- Un fichier composant `<component-name>.component.ts`
- Un fichier template `<component-name>.component.html`
- Un fichier CSS `<component-name>.component.css`
- Un fichier de spécifications de tests `<component-name>.component.spec.ts`

Angular components overview

Décorateur @Component

- **selector** Balise HTML
- **templateUrl** Chemin du template
- **styleUrls** Chemin des styles

```
@Component({
  selector: 'app-component-overview',
  templateUrl: './component-overview.component.html',
  styleUrls: ['./component-overview.component.css']
})
```

Angular components overview

Service

Un service est généralement une classe dont l'objectif est étroit et bien défini. Il doit faire quelque chose de spécifique et le faire bien.

En distinguant les services des composants, Angular accroît la modularité et la réutilisabilité.

```
export class Logger {
  log(msg: any)   { console.log(msg); }
  error(msg: any) { console.error(msg); }
  warn(msg: any)  { console.warn(msg); }
}
```

Introduction to services and dependency injection

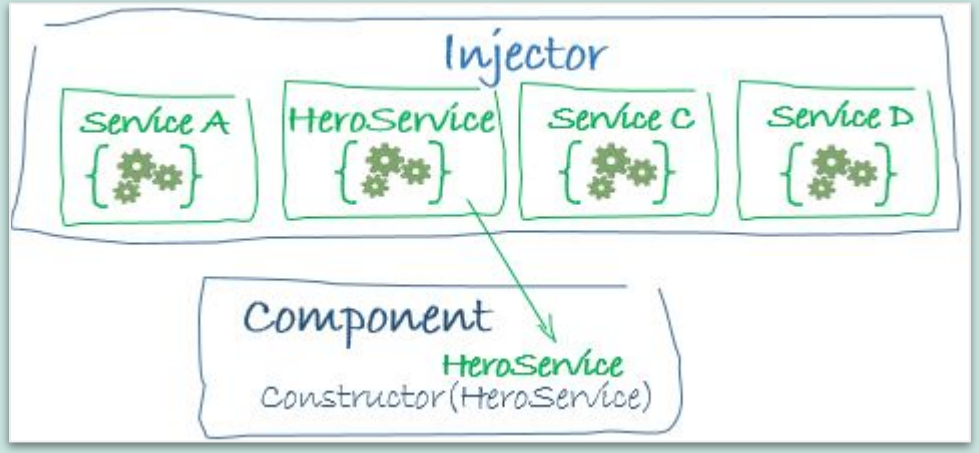
Service

```
export class HeroService {
  private heroes: Hero[] = [];

  constructor(
    private backend: BackendService,
    private logger: Logger) { }

  getHeroes() {
    this.backend.getAll(Hero).then( (heroes: Hero[]) => {
      this.logger.log(`Fetched ${heroes.length} heroes.`);
      this.heroes.push(...heroes); // fill cache
    });
    return this.heroes;
  }
}
```

Procédure d'injection d'un service



Injection de dépendance

Le décorateur `@Injectable()` définit une classe comme un service et permet à Angular de l'injecter dans un composant en tant que dépendance.

```
@Injectable({
 providedIn: 'root',
})
```

```
constructor(private service: HeroService) { }
```

Formulaires

Deux types de formulaires

- **Reactive :** Fournit un accès direct et explicite au modèle objet du formulaire sous-jacent.
- **Template-Driven :** S'appuie sur les directives du template pour créer et manipuler le modèle d'objet sous-jacent.

➔ Avec les formulaires réactifs, le modèle du formulaire est défini directement dans la classe du composant.

➔ Dans les formulaires pilotés par un template, le modèle du formulaire est implicite, plutôt qu'explicite.

Introduction to forms in Angular

Validation Template-Driven

```
<input type="text" id="name" name="name" class="form-control"
      required minlength="4" appForbiddenName="bob"
      [(ngModel)]="hero.name" #name="ngModel">

<div *ngIf="name.invalid && (name.dirty || name.touched)" class="alert">
  <div *ngIf="name.errors?.['required']">
    Name is required.
  </div>
  <div *ngIf="name.errors?.['minlength']">
    Name must be at least 4 characters long.
  </div>
  <div *ngIf="name.errors?.['forbiddenName']">
    Name cannot be Bob.
  </div>
</div>
```

Validating form input - Track control states

Validation Reactive (composant)

```
ngOnInit(): void {
  this.heroForm = new FormGroup({
    name: new FormControl(this.hero.name, [
      Validators.required,
      Validators.minLength(4),
      forbiddenNameValidator(/bob/i) // <-- Here's how you pass in the custom validator.
    ]),
    alterEgo: new FormControl(this.hero.alterEgo),
    power: new FormControl(this.hero.power, Validators.required)
  });
}

get name() { return this.heroForm.get('name'); }
get power() { return this.heroForm.get('power'); }
```

Validation Reactive (template)

```
<input type="text" id="name" class="form-control"
      formControlName="name" required>

<div *ngIf="name.invalid && (name.dirty || name.touched)"
    class="alert alert-danger">

  <div *ngIf="name.errors?.['required']">
    Name is required.
  </div>
  <div *ngIf="name.errors?.['minlength']">
    Name must be at least 4 characters long.
  </div>
  <div *ngIf="name.errors?.['forbiddenName']">
    Name cannot be Bob.
  </div>
</div>
```

Observables et RxJs

Principe

Les observables permettent de faire passer des messages entre les différentes parties de l'application. Ils constituent une technique de traitement des événements, de programmation asynchrone et de traitement des valeurs multiples.

➔ Le modèle de l'observateur est un modèle de conception logicielle dans lequel un objet, appelé le sujet, maintient une liste de ses dépendants, appelés observateurs, et les notifie automatiquement des changements d'état.

Using observables to pass values

Exemple

```
// Create simple observable that emits three values
const myObservable = of(1, 2, 3);

// Create observer object
const myObserver = {
  next: (x: number) => console.log('Observer got a next value: ' + x),
  error: (err: Error) => console.error('Observer got an error: ' + err),
  complete: () => console.log('Observer got a complete notification'),
};

// Execute with the observer object
myObservable.subscribe(myObserver);

// Logs:
// Observer got a next value: 1
// Observer got a next value: 2
// Observer got a next value: 3
// Observer got a complete notification
```

RxJS

RxJS (Reactive Extensions for JavaScript) est une bibliothèque de programmation réactive utilisant des observables qui facilite la composition de code asynchrone ou basé sur des callbacks.

```
import { from, Observable } from 'rxjs';

// Create an Observable out of a promise
const data = from(fetch('/api/endpoint'));
// Subscribe to begin listening for async result
data.subscribe({
  next(response) { console.log(response); },
  error(err) { console.error('Error: ' + err); },
  complete() { console.log('Completed'); }
});
```

[The RxJS library]

Observable d'évènement

```
import { fromEvent } from 'rxjs';

const el = document.getElementById('my-element')!;
// Create an Observable that will publish mouse movements
const mouseMoves = fromEvent<MouseEvent>(el, 'mousemove');

// Subscribe to start listening for mouse-move events
const subscription = mouseMoves.subscribe(evt => {
  // Log coords of mouse movements
  console.log(`Coords: ${evt.clientX} X ${evt.clientY}`);
  // When the mouse is over the upper-left of the screen,
  // unsubscribe to stop listening for mouse movements
  if (evt.clientX < 40 && evt.clientY < 40) {
    subscription.unsubscribe();
  }
});
```

Emission d'évènement

```
/* Composant enfant */
export class ZippyComponent {
  visible = true;
  @Output() open = new EventEmitter<any>();
  @Output() close = new EventEmitter<any>();

  toggle() {
    this.visible = !this.visible;
    if (this.visible) {
      this.open.emit(null);
    } else {
      this.close.emit(null);
    }
  }
}
```

```
<!-- Template parent -->
<app-zippy (open)="onOpen($event)"
(close)="onClose($event)"></app-zippy>
```

Observables in Angular

Routing

Principe

Dans une application à page unique, ce que l'utilisateur voit est modifié en affichant ou en masquant les parties de l'écran qui correspondent à des composants particuliers.

➔ Le routeur Angular est utilisé pour gérer la navigation d'une vue à l'autre. La navigation se fait en interprétant une URL de navigateur comme une instruction de changement de vue.

Angular Routing

Mise en place

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router'; // CLI imports router

const routes: Routes = [
  { path: 'first-component', component: FirstComponent },
  { path: 'second-component', component: SecondComponent },
];

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

Common Routing Tasks

Redirection & wildcard

```
const routes: Routes = [
  { path: 'first-component', component: FirstComponent },
  { path: 'second-component', component: SecondComponent },
  { path: '',   redirectTo: '/first-component', pathMatch: 'full' }, // redirect to `first-component`
  { path: '**', component: PageNotFoundComponent },  // Wildcard route for a 404 page
];
```

Configuration

```
const appRoutes: Routes = [
  { path: 'crisis-center', component: CrisisListComponent },
  { path: 'hero/:id',      component: HeroDetailComponent },
  {
    path: 'heroes',
    component: HeroListComponent,
    data: { title: 'Heroes List' }
  },
  { path: '',
    redirectTo: '/heroes',
    pathMatch: 'full'
  },
  { path: '**', component: PageNotFoundComponent }
];
```

Router reference

HTTP

Concept

La plupart des applications front-end doivent communiquer avec un serveur par le biais du protocole HTTP, afin de télécharger ou d'envoyer des données et d'accéder à d'autres services back-end.

➔ Angular fournit une API HTTP client pour les applications Angular, la classe de service HttpClient dans @angular/common/http.

Communicating with backend services using HTTP

Options de requête

```
options: {
  headers?: HttpHeaders | {[header: string]: string | string[]},
  observe?: 'body' | 'events' | 'response',
  params?: HttpParams|{[param: string]: string | number | boolean |
ReadonlyArray<string | number | boolean>},
  reportProgress?: boolean,
  responseType?: 'arraybuffer'|'blob'|'json'|'text',
  withCredentials?: boolean,
}
```

Envoi d'une requête

```
{
  "heroesUrl": "api/heroes",
  "textfile": "assets/textfile.txt",
  "date": "2020-01-29"
}
```

```
showConfig() {
  this.configService.getConfig()
    .subscribe((data: Config) => this.config = {
        heroesUrl: data.heroesUrl,
        textfile:  data.textfile,
        date: data.date,
    });
}
```

```
configUrl = 'assets/config.json';

getConfig() {
  return this.http.get<Config>(this.configUrl, options);
}
```

Gestion d'erreur

```
/** POST: add a new hero to the database */
addHero(hero: Hero): Observable<Hero> {
  return this.http.post<Hero>(this.heroesUrl, hero, httpOptions)
    .pipe(
      catchError(this.handleError('addHero', hero))
    );
}
```

UBALIA

AJC FORMATION

Contact : jacques.bach@ubalia.fr

@UbaliaFR Ubalia Ubalia