

Framework Angular



Pr Adil ANWAR

anwar@emi.ac.ma

Prérequis

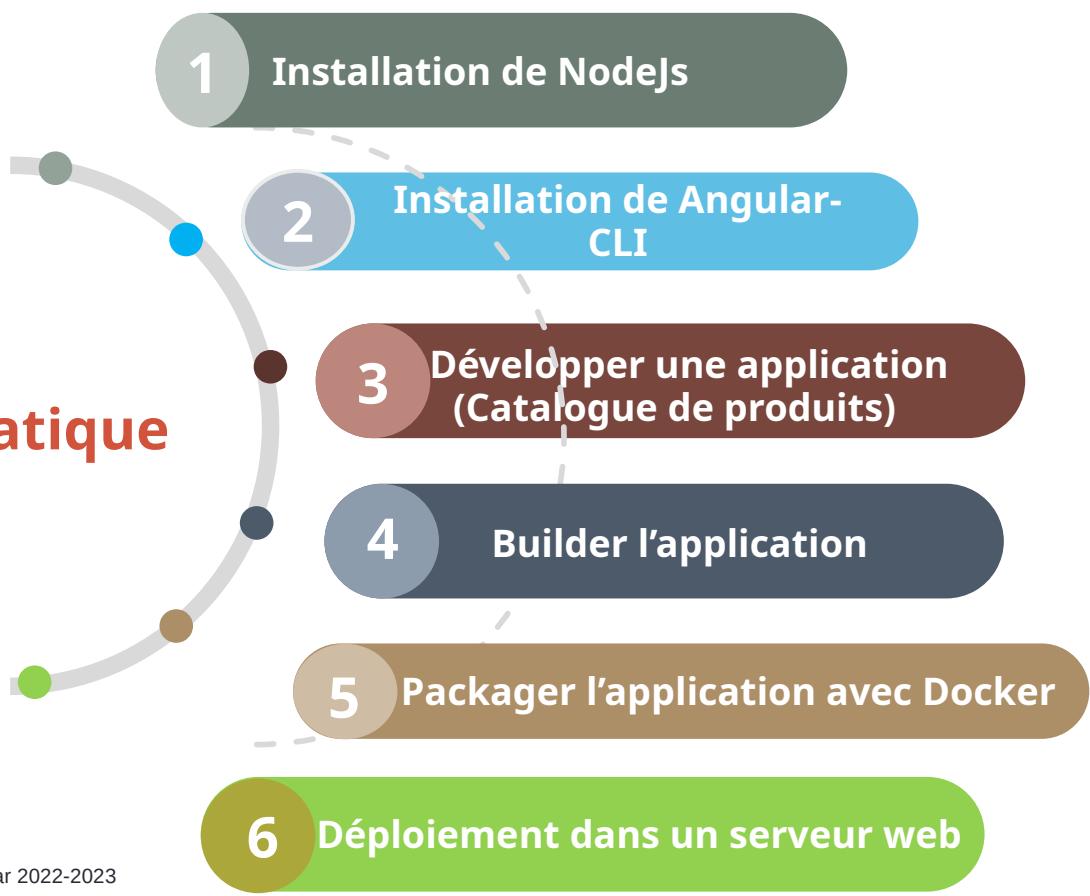
- Connaître les fondamentaux du web (protocole http, Architecture client/server, etc.)
- Être à l'aise avec les technologies client side (HTML, CSS, JS,)

Objectifs de formation

- A la fin de ce cette formation, vous devez savoir :
- Fondamentaux de **Angular**
- Être capable de développer une application web avec Angular
- Etre capable de builder, déployer une application Angular



Pratique



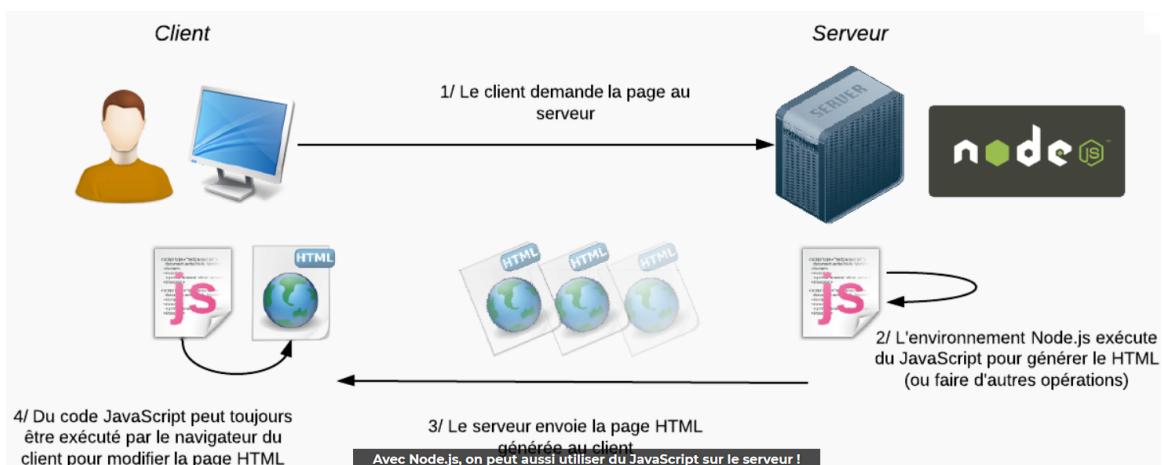
Chapitre 1 : Généralités sur le web et le Framework Angular

Introduction

- Le développement Web moderne consiste à créer des applications dites **front-end** qui se connectent à des serveurs pour soumettre ou obtenir des données. Les applications **front-end** utilisent l'approche de la page unique ou client lourd.
- Dans cette application, une application client communique avec un serveur **backend** via JSON, HTML, XML, texte, ou d'autres formats qui sont transmis via le protocole HTTP.
- Le serveur communique avec des bases de données et d'autres services. En d'autres termes, le serveur agit en tant qu'intermédiaire entre l'application **front-end** et la base de données ou d'autres services.
- Le serveur exécute des tâches que l'application **front-end** ne peut pas effectuer: l'authentification, validation des données, utilisation d'un système de fichiers, cryptage, etc.
- Le serveur communique avec le client de l'application front-end, et inversement, en distinguant différents points de terminaison par des URL conformément au protocole HTTP.
- Chaque requête HTTP de données a une **URL**, des **en-têtes** et un **corps**.

Le web comment ça marche ?

- Le world wide Web utilise le protocole **Hyper Transfer Protocol** (http) pour transférer des documents hypermédia (texte, images, vidéos ,etc.)
- Qu'est le rôle du serveur ?
 - écouter et accepter les requêtes http entrantes
 - analyser la requête http pour déterminer ce qui est demandé
 - localiser (et / ou créer) la ressource demandée
 - construire et renvoyer la réponse http



C'est quoi Node.js ?

- Node est utilisé pour implémenter les serveurs **backend**, qui sont souvent considéré comme des API JSON RESTful (mais ils pourraient également être d'autres formats).
- Node ressemble à d'autres serveurs comme le serveur Web Apache + PHP ou Tomcat + Java.
- Node n'est pas un framework en soi car il offre des modules de bas niveau (url, http, fs, etc.).
- Le framework **Express** est souvent utilisée pour implémenter des applications Web et des serveurs RESTful dans Node.
- L'adhésion aux architectures web services REST pour vos applications Web vous donnera la possibilité d'exploiter vos serveurs pour d'autres clients (mobile, IoT, public) et vous permettra de changer facilement de clients front-end en raison du couplage faible fourni par l'API.

Node.js

- environnement d'exécution JavaScript asynchrone basé sur des événements pour la création d'applications Web.
- traite les requêtes http comme des événements invoquant des callback fonctions / gestionnaires d'événements qui construisent la réponse http
- Node.js comprend également un gestionnaire de paquetages pour simplifier le déploiement d'applications JavaScript.
- Node.js interprète et exécute votre JavaScript en utilisant la machine virtuelle V8 de Google, qui est le même environnement d'exécution JavaScript utilisé par Google Chrome.

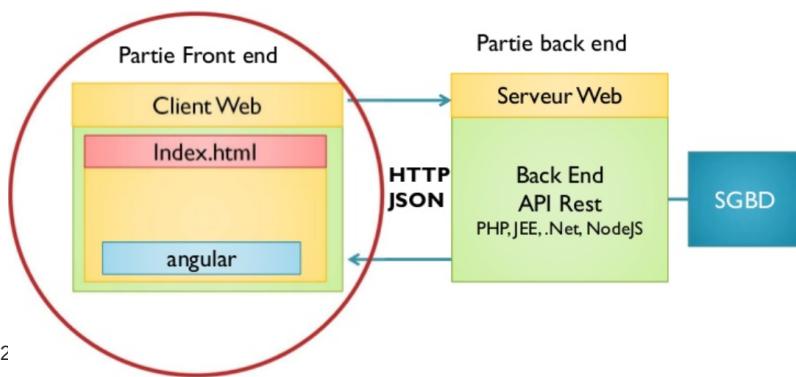


Quand utiliser Node.js ?

- JavaScript est un langage basé sur les **événements**, donc Node.js est lui-même basé sur les évènements. Du coup, c'est toute la façon d'écrire des applications web qui change ! Et c'est de là que Node.js tire toute sa puissance et sa rapidité.
- Node est idéal pour la diffusion en continu ou les applications en temps réel basées sur des événements, telles que:
 - Applications de chat
 - Applications temps réel et environnements collaboratifs
 - Serveurs de jeux
 - Serveurs de streaming
 - Un serveur de Chat
 - Un système d'upload très rapide
 - ... et de façon générale n'importe quelle application qui doit répondre à de nombreuses requêtes rapidement et efficacement, en temps réel
- Idéal pour écrire du code JavaScript partout!
- Node est utilisé par plusieurs organismes : Microsoft, Yahoo, LinkedIn, eBay, Dow Jones, Cloud9, The New York Times, etc,

Angular, c'est quoi ?

- Angular est un framework de développement web qui permet de créer des applications utilisant HTML et JavaScript / TypeScript.
- un framework JavaScript qui permet de créer des SPAs réactives.
 - Application dont la navigation se fait sans rechargement de la page
- Il permet de créer des applications en composant des **templates** HTML avec un balisage personnalisé (propre à Angular)
- Avec Angular on crée des classes de **composants** pour gérer ces templates à l'aide de la **liaison de données**, en ajoutant une logique applicative dans les **services** et en mettant en boîte des composants et des services dans des **modules**.



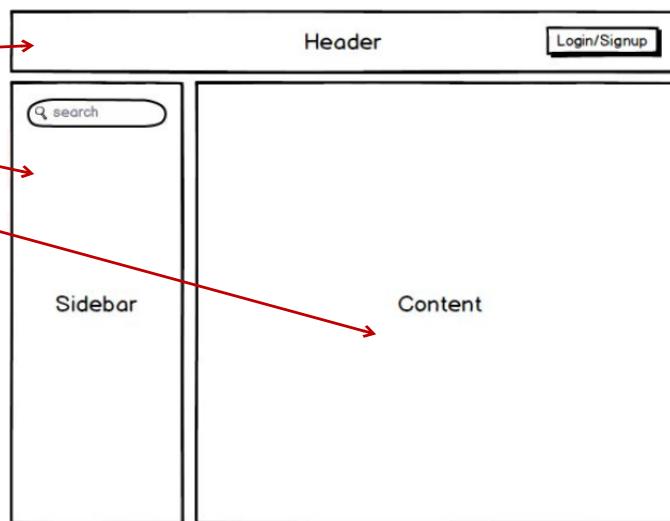
Philosophie d'Angular

- Angular est un framework orienté **composant**. Tu vas écrire de petits composants, et assemblés, ils vont constituer une application complète.
- Un composant est un groupe d'éléments HTML, dans un template, dédiés à une tâche particulière.
- Pour cela, tu auras probablement besoin d'un peu de logique métier derrière ce template, pour peupler les données, et réagir aux événements par exemple.
- Web components c'est quoi ?
 - Un standard a été défini autour de ces composants : le standard Web Component ("composant web"). Même s'il n'est pas encore complètement supporté dans les navigateurs, on peut déjà construire des petits composants isolés, réutilisables dans différentes applications.
 - Cette orientation composant est largement partagée par de nombreux frameworks front-end : c'est le cas depuis le début de ReactJS. EmberJS et AngularJS ont leur propre façon de faire quelque chose de similaire ; et les petits nouveaux Aurelia ou Vue.js parient aussi sur la construction de petits composants.

Adil Anwar 2022-2023

Web components : Exemple

- Avant de commencer le développement d'une application Angular, il faut:
 - Découper l'application en plusieurs composants
 - Décrire la responsabilité de chaque composant
 - Spécifier les inputs/outputs de chaque composant
- Ici nous avons créé trois composants
 - Header Component
 - SidebarComponent
 - ContentComponent



Adil Anwar 2022-2023

AngularJS, Angular 2, Angular

- AngularJS est la première version qui a fait la popularité du framework.
- Angular 2 est une réécriture complète du Framework AngularJS. Sa base représente le socle futur du framework
- Angular 4 est un update d'Angular 2. La version 3 a été sautée pour des “problématiques” de versioning,
- Angular sort une nouvelle version majeure tous les 6 mois :
 - Angular 2 est sorti en septembre 2016,
 - Angular 4 en mars 2017,
 - Angular 5 en novembre 2017,
 - Angular 6 en mai 2018,
 - Angular 7 en octobre 2018,
 - Angular 8 en mai 2019.
- Dernière version, 17.1.2 (31 janvier 2024)
- les mises à jour sont désormais complètement automatisées. Pour exemple, cela prend 5 minutes de passer d'Angular 6 à 8 avec la commande *ng update*.

✍Adil Anwar 2022-2023

Découvrir TypeScript

✍Adil Anwar 2022-2023

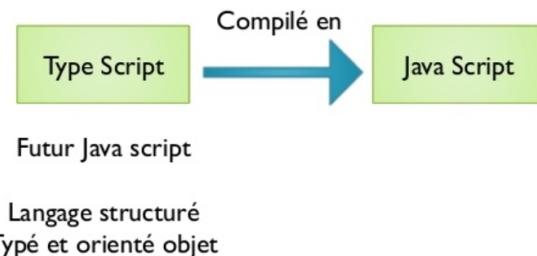
Développement web avec JavaScript

- JavaScript est un excellent langage pour développer des applications de petites tailles
=> inadapté pour les applications complexes,
 - un code JavaScript complexe
 - nombreuses lacunes dans le langage (typage dynamique, etc.).
- Évolution du langage, il est devenu le moyen idéal pour écrire :
 - des applications multiplates-formes (bureau, mobile, web, etc.).
 - Des applications indépendantes aux navigateurs
- Mais, lors de sa création, JavaScript n'était pas destiné à cela, il était principalement conçu pour des utilisations simples dans de petites applications.
- **Constatations :**
 - besoin de suivre la croissance spectaculaire de l'utilisation de JavaScript,
 - besoin de simplification et d'amélioration du langage
 - besoin de détecter le plus tôt les erreurs au cours du développement, plutôt qu'au moment de l'exécution
 - évoluer le langage afin de créer facilement et efficacement des applications de grande taille.

Adil Anwar 2022-2023

Découvrir TypeScript

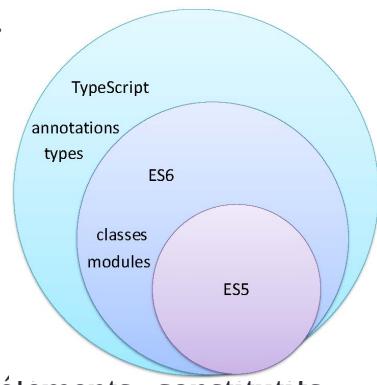
- TypeScript est un langage de programmation open source développé et maintenu par Microsoft. C'est un **sur-ensemble** typé de JavaScript qui se compile en JavaScript. Par conséquent, un programme JavaScript est également un programme TypeScript valide
- TypeScript ajoute le **typage statique** ou strict au langage JavaScript, en plus de la programmation orientée objet basée sur les classes.
- Les navigateurs ne comprenant que JavaScript, lorsque vous écrivez votre application en TypeScript, vous avez besoin d'un compilateur (**Transpileur**) qui puisse compiler votre code et le convertir en JavaScript.
- **Transpileur**, tout comme la compilation, transforme du code dans un langage vers un autre langage du même niveau : ex Exemple transformer la version de JavascriptES6 vers la version ES5



Adil Anwar 2022-2023

Propriétés du langage TypeScript

- TypeScript peut être utilisé pour écrire à la fois le code côté client (exécuté sur un navigateur) et le code côté serveur (exécuté sur le serveur d'applications) à l'aide de la bibliothèque Node.js.
- Cela signifie qu'avec TypeScript, vous n'apprenez qu'un seul langage pour devenir un développeur Web à pile complète (fullstack).
- TypeScript implémenté les fonctionnalités de ES6
 - Variables fortement typées.
 - Paramètres de fonction fortement typés.
 - Des classes.
 - Interfaces
 - Héritage.
- Modules permettant de diviser les éléments en éléments constitutifs permettant à plusieurs personnes de développer une application.



Adil Anwar 2022-2023

TypeScript et Angular

- A partir de la version 2.0 d'Angular, le framework entend tirer parti des nouveautés de EcmaScript 6 et même plus,
- avec TypeScript, qui rassemble à ES6 et un système de typage et d'annotations (décorateurs).
- Les développeurs pourront choisir d'écrire en TypeScript (préconisé) ou en ES6 (très rare) avec une phase de transpilation vers ES5 ou directement en ES5 (extrêmement rare).

Adil Anwar 2022-2023

Les types de TypeScript

- Le fait de différencier le type d'une variable
- Dans un langage dit typé, on ne peut enregistrer qu'un type de valeur dans une variable.

```
let variable : Type ;  
const unNombre : number = 12;
```

- **boolean:**

- Valeurs : true, false
- let isDone : boolean = false;

- **number:**

- Comme en JS, tous les nombres sont des flottants.
- Accepte aussi les valeurs : Binaires, Octales, Hexadecimale

```
let decimal: number = 6;  
let hex: number = 0xf00d;  
let binary: number = 0b1010;  
let octal: number = 0o744;
```

Adil Anwar :

Les types de TypeScript

- **string:** Chaîne de caractères

```
let prenom: string = 'Adrien';  
let nom: string = "Vossough";
```

- Nous pouvons taper une chaîne sur plusieurs lignes avec le templating.

```
let texte: string = `mon prénom est ${ prenom }  
et mon nom est ${ nom }`;
```

- Équivalent à :

```
let texte2: string = "mon prénom est " + prenom + "\n"  
+ "et mon nom est " + nom;
```

Adil Anwar 2022-2023

Les types de TypeScript

- **Array:** Tableau typé
- Le tableau suivant ne peut contenir que des nombres :

```
let tab0: number[] = [ -5, 2, 8 ];

tab0.push(12.56); // OK

tab0.push("Salut !"); // Erreur
// Argument of type 'string' is not assignable to parameter of type 'number'.
```

- Il existe une seconde écriture utilisant des tableaux génériques :

```
let tab1: Array<number> = [ -8, 15.9, 0 ];
```

Adil Anwar 2022-2023

Les types de TypeScript

- **Tuple:** Tableau avec différents types

```
// déclaration
let user: [string, number];
// initialisation correcte
user = ["Adrien", 35]; // ok
// initialisation incorrecte
user = [35, "Adrien"]; // Erreur
```

- Utilisation

```
console.log( user[0] );
// affiche : Adrien

console.log( user[1] );
// affiche : 35
```

Adil Anwar 2022-2023

Les types de TypeScript

- **Enum:** Ensemble de nombres constants ayant un nom.

```
enum Fruit {Banane, Kiwi, Orange};  
let k: Fruit = Fruit.Kiwi;  
  
console.log(k);  
// Affiche : 1
```

- Par défaut, la première valeur vaut 0.
- Il est possible de définir des valeurs manuellement :

```
enum Couleur {Rouge, Vert=-0.2, Jaune=58};  
let n: Couleur = Couleur.Jaune;  
  
console.log(n);  
// Affiche : 58
```

- Il est possible de récupérer la chaîne de caractères ainsi.

```
const recupererLaChaineDesCaracteres = () => {  
    let s: string = Couleur[-0.2];  
  
    console.log(s);  
    // Affiche : Vert
```

Adil An

Les types de TypeScript

- **any:** Accepte n'importe quel type de variable.

```
let v: any = "Salut";  
v = 1;  
v = true;  
  
let liste: any[] = ["Adrien", false, 123];  
liste[0] = 35;
```

- **void:** Accepte que les valeurs null et undefined
 - `let n: void = null;`
- Généralement utilisé pour indiquer qu'une fonction ne renvoie pas de valeurs.

```
function direCoucou(): void {  
    console.log("Coucou !");  
}
```

- Il existe d'autres types mais très peu utilisés directement comme `null` et `undefined`, car très limités.

Les types de TypeScript

- Cast : Changer une valeur dans un **typeA** vers une valeur de **typeB**

```
let uneValeur: any = "Peut-être une chaîne ?";  
  
let longueur: number = (<string>uneValeur).length;
```

- Le type any pouvant être de n'importe quel type, nous indiquons que uneValeur est de type String.
- Dans ce cas, TypeScript nous fait confiance et autorise l'utilisation des méthodes propre aux objets String.
- Seconde syntaxe :

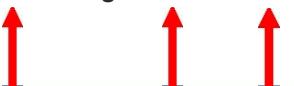
```
let uneValeur: any = "Peut-être une chaîne ?";  
  
let longueur: number = (uneValeur as string).length;
```

Adil Anwar 2022-2023

Les fonctions : Typage

- Type des paramètres et des valeurs de retour.

Type du 1^{er} argument Type du 2^{ème} argument Type de la valeur renvoyée



```
function addition(x: number, y: number): number {  
    return x + y;  
}  
  
let addition2 = function(x: number, y: number): number {  
    return x + y;  
};
```

Utilisation :

```
let resultat0 = addition("b", 2); // erreur de type  
let resultat1 = addition(2); // erreur, il manque un paramètre  
let resultat2 = addition(2, 3, 8); // erreur, paramètre en trop  
  
let resultat2 = addition(55, 20); // ok  
let resultat2 = addition2(10, 40); // ok
```

Fonctions : paramètre optionnel

- Paramètre pouvant être omis lors de l'appel de la fonction
- En javascript, tu ne passes pas les paramètres à l'appel de la fonction, leur valeur sera undefined. Mais en TypeScript, si tu déclares une fonction avec des paramètres typés, le compilateur déclare une erreur si les oublies :

```
function addition(x: number, y?: number): number {
    if(y) {          // nous testons si y est défini
        return x + y;
    }
    return x;
}
```

```
let resultat1 = addition(2);           // ok
let resultat2 = addition(55, 20);      // ok
```

Adil Anwar 2022-2023

Fonctions : Paramètre par défaut

- Si le paramètre est omis, il recevra une valeur par défaut.
- Le type d'un paramètre ayant une valeur par défaut peut être omis
- La valeur par défaut définit le type du paramètre

```
function addition(x: number, y = 12): number {
    return x + y;
}

function addition2(x = 8, y: number): number {
    return x + y;
}
```

```
let resultat1 = addition(2);           // ok
let resultat2 = addition(55, 20);      // ok

console.log(resultat1, resultat2);

addition2(5, 2);                     // ok
addition2(3);                      // erreur, paramètre manquant
addition2(undefined, 5);            // ok
```

Rappel de la POO : Classes

- Classe : Schéma à suivre pour la création d'une classe

```
class Bonjour {  
    message: string;  
  
    constructor(message: string) {  
        this.message = message;  
    }  
    affiche() {  
        console.log( "Bonjour " + this.message );  
    }  
}  
  
let bjr = new Bonjour("Adrien");  
  
bjr.affiche()
```

➤ Adil Anwar 2022-2023

Rappel de la POO : l'héritage

- Héritage : Permet de créer une classe "parent" qui donnera toutes ses caractéristiques à ses enfants
- L'héritage multiple n'existe pas en TypeScript

Classe mère : Animale

```
class Animale {  
    nom: string;  
    constructor(leNom: string) {  
        this.nom = leNom;  
    }  
    deplacer(distance: number = 0) {  
        console.log(` ${this.nom} bouge de ${distance}m. `);  
    }  
}
```

Classe enfant : Chien

```
class Chien extends Animale {  
    constructor(name: string) {  
        super(name);  
    }  
}
```

Chien de par son parent :

- A un nom
- Peut se déplacer

Utilisation:

```
let monChien = new  
Chien("Milou");  
monChien.deplacer(5);
```

➤ Adil Anwar 2022-2023

Rappel de la POO : L'héritage

- Si nous considérons que l'héritage est reprendre l'ADN du parent, nous pouvons considérer que l'enfant est une évolution.
- Nous pouvons donc lui ajouter de nouveaux attributs et fonctions que le parent n'a pas.

Classe enfant : Chien

```
class Chien extends Animale {
    couleurPoil: string;

    constructor(name: string) {
        super(name);
    }

    manger() {
        console.log("Je mange");
    }
}
```

Utilisation:

```
let monChien = new Chien("milou");
monChien.deplacer(5);
monChien.manger();
```

Adil Anwar 2022-2023

Rappel de la POO : L'héritage

- **super** : Indique que nous utilisons une fonction créée dans le parent

Classe enfant : Chien

```
class Chien extends Animale {
    constructor(name: string) {
        super(name);
    }
}
```

- Dans ce cas, super(), indique que nous appelons la fonction **constructor** du parent.

Adil Anwar 2022-2023

Les interfaces

- TypeScript permet l'utilisation des interfaces pour améliorer le typage dynamique des objets.
 - Permet d'utiliser dans une fonction des objets de types différents mais qui ont un comportement commun
 - Se base sur une série d'attributs et de méthodes attendus.
 - L'objet est considéré valide quel que soit sa classe s'il respecte les attributs et les méthodes attendus.

Sans interface :

```
function test(obj) {  
    console.log( obj.x + 3 ); // additionne l'attribut x avec 3  
}  
  
let o1 = {};  
let o2 = { x: "coucou" };  
let o3 = { x: 10 };  
  
test(o1); // affiche: NaN car undefined + 3  
test(o2); // affiche: coucou3  
test(o3); // affiche: 13
```

Adil Anwar 2022-2023

Les interfaces

Avec interface: 1ère écriture

```
function test( obj: {x: number} ) {  
    console.log( obj.x + 3 ); // additionne l'attribut x avec 3  
}
```

Avec interface: 2ème écriture

```
interface xNumber {  
    x: number;  
}  
  
function test( obj: xNumber ) {  
    console.log( obj.x + 3 ); // additionne l'attribut x avec 3  
}
```

```
let o1 = {};  
let o2 = { x: "coucou" };  
let o3 = { x: 10 };  
  
test(o1); // erreur car pas d'attribut x  
test(o2); // erreur car x n'est pas de type number  
test(o3); // affiche: 13
```



Nous avons bien filtré les objets, seuls ceux ayant un comportement attendu sont acceptés

Les modules

- Un module est une façon de découper du code pour le réutiliser facilement
- S'apparente à un import de package en Java ou un include en PHP

module1.js

```
export function test(message) {
    console.log(message);
}
export function truc() {
    console.log("Je ne fais rien");
}
```

module2.js

```
import {test, truc} from './module1.js';
test("coucou");
truc();
```

module3.js

```
import {*} from './module1.js';
test("coucou");
truc();
```

Adil Anwar 2022-2023

Les décorateurs

- C'est une façon de faire de la métaprogrammation. Une sorte d'annotation, ne servant pas réellement au langage lui-même mais plutôt aux frameworks et aux librairies.
- Les décorateurs attachent dynamiquement des responsabilités supplémentaires à un objet. Il fournit une alternative souple à l'héritage, pour étendre des fonctionnalités..
- En Angular, on utilisera les **annotations** fournies par le framework. Leur rôle est assez simple: ils ajoutent des **métadonnées** à nos classes, propriétés ou paramètres pour par exemple indiquer "cette classe est un composant", "cette propriété est un champ input d'un composant"
- En TypeScript, les annotations sont préfixées par **@**, et peuvent être appliquées sur une classe, une propriété de classe, une fonction, ou un paramètre de fonction. Pas sur un constructeur en revanche, mais sur ses paramètres oui.

Adil Anwar 2022-2023

Les décorateurs

- Comment reconnaître un décorateur ?
- on trouve des décorateurs sur différents objets comme les classes, propriétés ou méthodes.
- Par exemple, sur une classe :
 - `@Component`
 - `@Directives`
 - `@Injectable()`
- Sur un attribut
 - `@Input()`
 - `@Output()`

❖ Adil Anwar 2022-2023

Atelier

- Définir une interface User ayant comme propriétés
 - `firstname`, `lastname`, `email`, `age` et liste de rôles.
- L'âge doit être marqué comme propriété optionnelle.
- Dans la classe du composant, créer un user et l'afficher sur la page (sans ses rôles).

❖ Adil Anwar 2022-2023

Angular: Notions de base



Adil Anwar 2022-2023

Angular CLI

- Angular CLI est une interface de ligne de commande que vous pouvez utiliser lors de la création d'applications avec Angular.
- À partir de cet outil ligne de commande :
 - vous pouvez créer des projets
 - ajouter des fichiers à des projets existants
 - tester, déboguer et déployer vos applications Angular.
- le processus de génération d'un modèle générique d'application basé sur les normes de construction d'une application Angular.
- Une hiérarchie de dossiers de base est créée selon les meilleures pratiques de mise en page et de nommage.
- le CLI peut être installer en utilisant l'outil npm

npm install -g @angular/cli

Adil Anwar 2022-2023

Commandes du CLI

- Pour générer la structure d'un projet Angular, on utilise le CLI via sa commande **ng** suivie de l'option **new** et le nom du projet
ng new my-app
- Cette commande génère les différents fichiers requis par une application basique Angular et installe aussi toutes les dépendances requises par le projet.
- La commande ng new peut être utilisée avec des options :
 - Annuler la création des fichiers de tests
ng new my-app --skip-tests=true
 - crée des fichiers .scss pour les styles plutôt que des fichiers .css
ng new --style=scss :

Adil Anwar 2022-2023

Commandes du CLI

- Pour tester un projet Angular, on exécute la commande **ng serve** à partir de la racine du projet.
- Cette commande compile le code source du projet pour transpiler le code TypeScript en JavaScript et en même temps démarre un serveur web local basé sur Node.js pour déployer l'application localement.
- Pour tester le projet, il suffit de lancer le navigateur et accéder à l'URL:
http://localhost:4200 Welcome to App!
- modifier le port par défaut avec l'option --port
ng serve --port 5000



Here are some links to help you start:

- [Tour of Heroes](#)
- [CLI Documentation](#)
- [Angular blog](#)

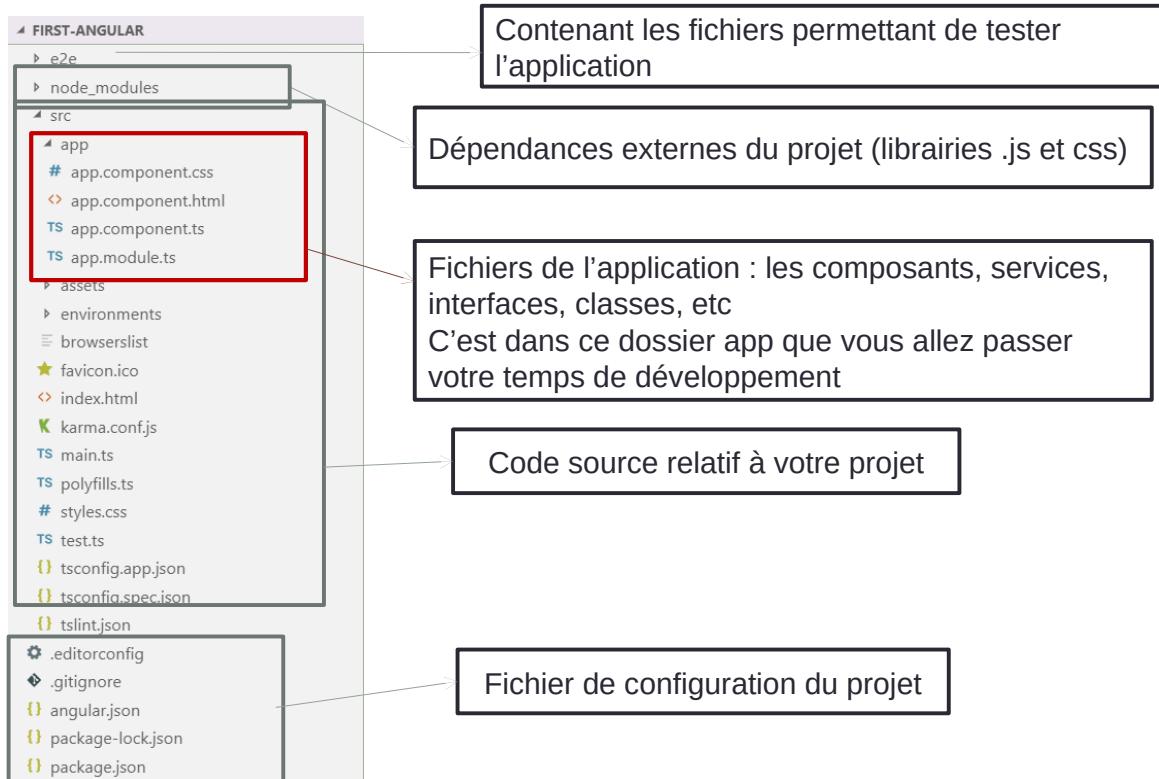
Adil Anwar 2022-2023

Commandes de génération

- le CLI permet aussi de créer facilement des composants de votre application sans avoir à vous soucier d'inclure correctement le code dans votre application.
- Le CLI génère les fichiers au format approprié et les modifie pour inclure correctement le composant nouvellement généré.
- Parmi les différents générateurs pouvant être exécutés par la CLI, citons:
 - Modules : **ng generate module my-module**
 - Composants : **ng generate component my-component**
 - Interfaces : **ng generate interface my-interface**
 - Services : **ng generate service my-service**
 - Classes : **ng generate class my-class**
 - Guards : **ng generate guard my-guard**
- Le CLI propose aussi d'autres commandes utiles :
 - Pour déployer l'application : **ng build --target=production --base-href /**
 - Pour exécuter les tests unitaires avec le framework jasmine: **ng test**
 - Pour exécuter des tests bout en bout avec Protractor : **ng e2e**

Adil Anwar 2022-2023

Structure d'un projet Angular



Structure d'un projet Angular



Index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>FirstAngular</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

Structure d'un projet Angular



main.ts

```
import { bootstrapApplication } from '@angular/platform-browser';
import { appConfig } from './app/app.config';
import { App } from './app/app';

bootstrapApplication(App, appConfig)
  .catch((err) => console.error(err));
```

Structure d'un projet Angular



app.component.ts

```
import { Component, signal } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { ListMovies } from './list-movies/list-movies';

@Component({
  selector: 'app-root',
  imports: [ListMovies],
  templateUrl: './app.html',
  styleUrls: ['./app.scss']
})
export class App {
  protected readonly title = signal('movies-front');
}
```

app.component.html

```
<!--The content below is only a placeholder and can be replaced-->
<div style="text-align:center">
  <h1>
    | Welcome to {{ title }}!
  </h1>
  
  <li> <a target="_blank" rel="noopener" href="https://angular.io/cli"</li>
  <li> <a target="_blank" rel="noopener" href="https://blog.angular.i</li>
</ul>
```

Welcome to App!



Here are some links to help you start:

- [Tour of Heroes](#)
- [CLI Documentation](#)
- [Angular blog](#)

Structure d'un projet Angular



app.component.css

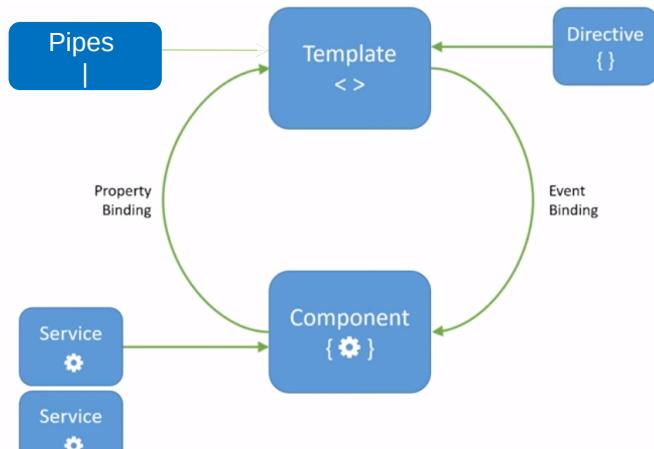
Ce fichier est généré vide, il contient la logique de présentation des données (règles css)

app.component.spec.ts

Les fichiers .spec sont des tests unitaires pour vos fichiers sources. La convention pour les applications Angular 2 est d'avoir un fichier .spec pour chaque fichier .ts. Ils sont exécutés à l'aide du Framework JavaScript de tests unitaires Jasmine via le programme de tâches Karma lorsque vous utiliser la commande **ng test**

Architecture d'Angular

- Une application Angular se compose de :
 - Un à plusieurs modules dont un est principal
 - Chaque module peut inclure :
 - Des composants web : la partie visible de l'application
 - Des services pour la logique applicative. Les composants peuvent utiliser les services via le principe de l'injection de dépendances.
 - Les directives : un composant peut utiliser des directives
 - Les pipes : utilisées pour formater l'affichage des données



Adil Anwar 2022-2023

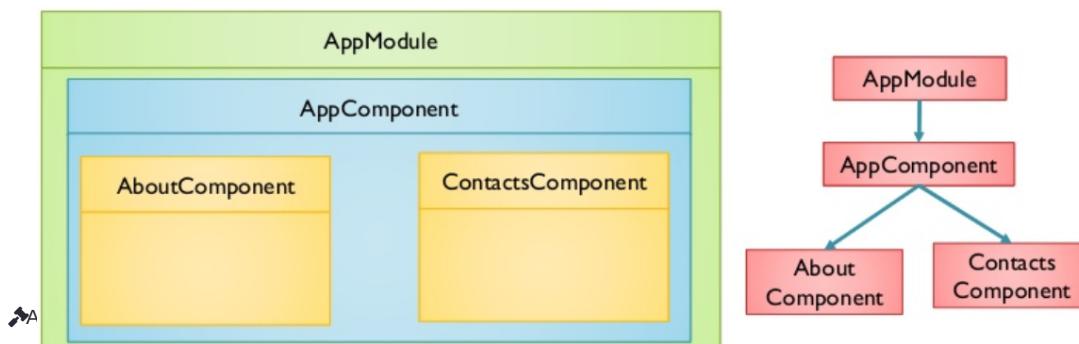
Composants



Adil Anwar 2022-2023

Composants

- Un composant est utilisé par contrôler une vue ou une partie de l'écran.
- L'une des caractéristiques la plus utile de l'architecture d'Angular est l'utilisation de composants **réutilisables** pour l'affichage des données.
- En utilisant un composant, vous pouvez coupler la logique d'affichage des données, les styles et le HTML dans un conteneur facilement portable dans toute votre application.
- Le code que vous allez ajouter à ce code HTML est responsable de la définition des liaisons aux données et des directives au sein du code HTML fournissant les aspects dynamiques d'un composant Angular.



Génération d'un composant avec le CLI

- Pour créer un composant il faut taper la commande :
- > **ng generate component component-name** : où component-name est le nom du composant à créer
- Exemple : `ng generate component git-search`
 - le générateur va créer un dossier `/src/app/git-search`, ainsi qu'un fichier `.css`, `.ts`, `.html` et `.spec.ts` pour le composant `git-search`. Ces fichiers ont chacun un objectif.
 - Le fichier `.ts` : contient la logique TypeScript du composant et est l'endroit où vous placeriez les méthodes, l'intégration de service et toute autre logique.
 - Le fichier `.css` : est automatiquement injecté dans votre composant. On peut écrire automatiquement des feuilles de code CSS spécifiquement adaptées au composant, sans avoir à vous soucier de la réutilisation des classes.
 - Le fichier `.html` : est l'endroit où il faut écrire le code de la vue (template).
 - Le fichier `.spec.ts` : est l'endroit où on peut écrire des tests Karma pour le composant.
 - Les tests seront traités dans la partie Angular avancé.

Composants

- le fichier de classe créé lors de la génération d'un composant contiendra les propriétés et les méthodes qui seront utilisées par l'application.
- On trouve dans ce fichier également des métadonnées utilisées pour fournir des informations supplémentaires sur le composant .
- Les métadonnées sont définies à l'aide de **décorateurs**.

```
import { Component, signal } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { ListMovies } from './list-movies/list-movies';
@Component({
  selector: 'app-root',
  imports: [ListMovies],
  templateUrl: './app.html',
  styleUrls: ['./app.scss']
})
export class App {
  protected readonly title = signal('movies-front');
}
```

Adil Anwar 2022-2023

Anatomie d'un composant Angular

- Les instructions d'importation sont utilisées pour accéder aux fonctionnalités d'autres codes tels que les bibliothèques et autres classes.

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

- Cette ligne est responsable de l'importation de l'objet composant à partir de la bibliothèque @ angular / core. C'est l'une des principales bibliothèques utilisées dans Angular.

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

- C'est le segment où vous définissez votre composant. Le @ est un symbole TypeScript qui nous aide à décrire un décorateur Angular.
- Dans ce cas, @Component est un décorateur qui indique à Angular que nous souhaitons créer un nouveau composant.
- Un décorateur en Angular est précédée du symbole @ et est considérée comme un mécanisme permettant d'ajouter des métadonnées à une classe, aux membres d'une classe ou aux arguments de méthode d'une classe.

Adil Anwar 2022-2023

Anatomie d'un composant Angular

- le décorateur @Component contient un objet avec des métadonnées et des informations pour ce composant.
 - **selector** : spécifie le nom d'une balise HTML qui contiendra ce composant lorsque l'application est en cours d'exécution. Dans le code HTML, vous trouverez les balises `<app-home-page> </ app-home-page>` pour ce composant.
 - **templateUrl** : l'utilisation de cette option vous permet de spécifier un modèle qui contiendra le code HTML pour ce composant. Vous pouvez également choisir d'utiliser l'option template: et spécifier le droit HTML "en ligne" dans le corps du composant.
 - **imports** : permet de référencer d'autres composants, modules, ou directives utilisés à l'intérieur du composant.
 - **styleUrls** : renvoie à une feuille de style CSS externe qui sera appliquée au rendu HTML du composant.

Adil Anwar 2022-2023

Anatomie d'un composant Angular

- La déclaration d'exportation
- ```
export class HomePageComponent implements OnInit
```
- Le mot-clé **export** permet à votre composant d'être disponible pour être importé dans d'autres fichiers du projet.
  - Le mot-clé **class** identifie ce fichier en tant que fichier de classe.
  - L'identifiant `HomePageComponent` est le nom de la classe. Ce sera le nom que vous utiliserez dans une instruction import dans les autres fichiers de votre application et vous fournira un identificateur clair pour cette classe.
  - Le dernier mot-clé, **implements**, ne sera pas toujours présent sur tous les composants. Dans ce cas, cela indique que notre classe `HomePageComponent` implémentera également des fonctionnalités spécifiées dans un autre interface, appelé `OnInit`.

Adil Anwar 2022-2023

# Atelier : mon premier composant

- Créer une classe Type script “ProductItem” définit par les attributs suivants :
  - nom
  - l’url de l’image du produit,
  - le prix,
  - la catégorie,
  - la description
- Créer un constructeur pour initialiser les attributs
- Créer un composant “list-product” qui affiche plusieurs produits dans une page.

Adil Anwar 2022-2023

## Templates

- La vue est la partie visuelle du composant. En utilisant l’attribut template du décorateur @Component, nous déclarons le modèle HTML que le composant utilisera:
- Deux solutions sont possibles:
  - Template inline :

```
@Component({
 selector: 'inventory-app-root',
 template: `
 <div class="inventory-app">
 </div> `
 }
})
```

- Dans cette exemple ci-dessus, notez que nous utilisons la syntaxe de la chaîne multiligne de TypeScript.

- Nous pouvons également déplacer notre template vers un fichier séparé et utiliser l’attribut templateUrl pour le référencer:

```
@Component({
 selector: 'inventory-app-root',
 templateUrl: './app.component.html'
})
```

Adil Anwar 2022-2023

# Templates : bootstrap

- On peut utiliser les classes de bootstrap dans les templates en utilisant le CLI
- **Installation** : npm install bootstrap --save
- Vérifier qu'un répertoire bootstrap a bien été créé dans nodes\_modules.
- **Intégration dans le projet**
  - Ouvrir le fichier .angular.json (pour Angular 6 ou supérieur) situé à la racine de votre projet
- Dans la rubrique styles, ajouter le chemin relatif vers le fichier bootstrap.css (le chemin est "./node\_modules/bootstrap/dist/css/bootstrap.css" ,
- **Utilisation** : on peut utiliser maintenant bootstrap dans les component.html

```
<button class="btn btn-success"
| (click)="onAllumer()">Tout allumer</button>
<button class="btn btn-danger"
| (click)="onEteint()">Tout éteindre</button>
<ul class="list-group">
| <li class="list-group-item" *ngFor="let element of ensemble">
| | {{ element }}
|

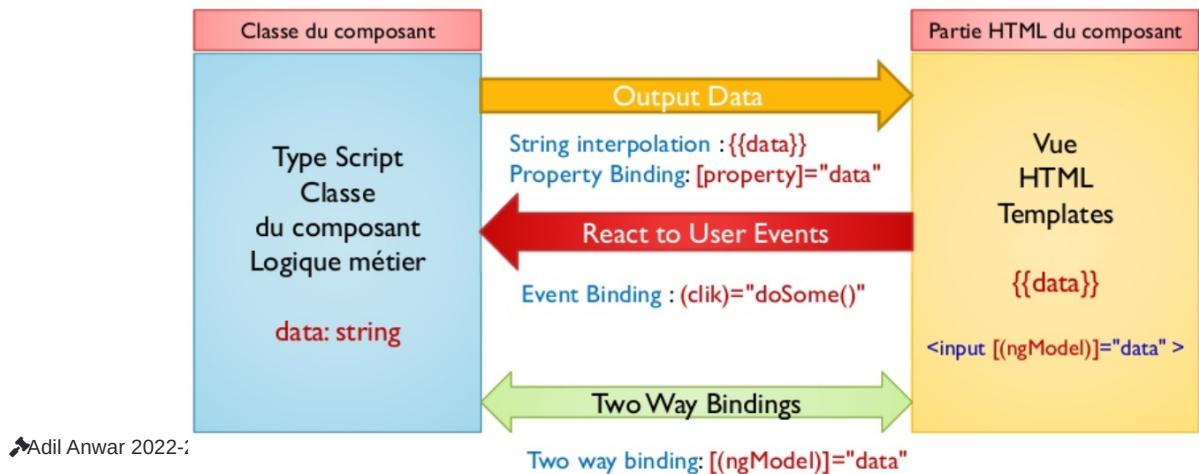
```

## Databinding



# Liaison de données (Databinding)

- La liaison est le terme utilisé pour décrire comment les données d'une application sont «liées» à l'interface utilisateur.
- La liaison est donc responsable de la communication entre nos données et notre interface utilisateur.
- La liaison nous permet de transmettre les données à l'interface utilisateur via l'utilisation de propriétés, dans la classe qui définit notre modèle, et l'utilisation de directives, qui utilisent un concept appelé interpolation.



## Databinding

- Il existe quatre types de liaison de données que nous pouvons effectuer dans Angular.
  - Liaison d'interpolation :
  - liaison de propriété
  - liaison d'événement
  - liaison bidirectionnelle.
- **L'interpolation** est réalisée à l'aide de la syntaxe d'interpolation, également appelée moustache, où nous plaçons la propriété de données dans le code HTML à l'aide de la double accolade, `{{films[0].titre}}`.

Exemple : `<h1> {{titre}} </h1>`

# Databinding - Interpolation

- La classe UserComponent a une propriété, *firstName*. Et le template a été enrichi avec une balise <h1>, utilisant la fameuse notation avec double-accolades(les "moustaches") pour indiquer que cette expression doit être évaluée.
- Ce type de databinding est **l'interpolation**.

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

@Component({
 selector: '[app-user]',
 template: `<h1> User : {{firstName}} </h1>`,
})
export class UserComponent {
 firstName = 'Adil'
}
```

Adil Anwar 2022-2023

# Databinding - Interpolation

Il faut cependant se rappeler une chose importante : si on essaye d'afficher une variable qui n'existe pas, au lieu d'afficher undefined, Angular affichera une chaîne vide. Et de même pour une variable null.

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

@Component({
 selector: '[app-user]',
 template: `<h1> User : {{firstName}} </h1>`,
})
export class UserComponent {
}
```

Adil Anwar 2022-2023

# Databinding - Interpolation

- Que se passe-t-il si mon objet *user* est en fait récupéré depuis le serveur, et donc indéfini dans mon composant au début ? Que pouvons-nous faire pour éviter les erreurs quand le template est compilé ?.

```
@Component({
 selector: 'user-app',
 // user is undefined
 // but the ?. will avoid the error
 template: `
 <h1> User </h1>
 <h2> Welcome {{user?.name}} </h2>
 `})
export class UserComponent {
 user: any;
}
```

Adil Anwar 2022-2023

# Databinding – binding de propriété

- En Angular, on peut écrire dans toutes les propriétés du DOM via des attributs spéciaux sur les éléments HTML, entourés de crochets [ ].

```
<p> {{user.name}} </p>
<p [textContent]="user.name"></p>
```

- La syntaxe à base de crochets permet de modifier la propriété `textContent` du DOM, et nous lui donnons la valeur `user.name` qui sera évaluée dans le contexte du composant courant, comme pour l'interpolation.
- Note que l'analyseur est sensible à la casse, ce qui veut dire que la propriété doit être écrite avec la bonne casse.

Adil Anwar 2022-2023

# Databinding - binding de propriété

- Bien sûr, une expression peut aussi contenir un appel de fonction

```

```

```

```

Adil Anwar 2022-2023

# Databinding - Événements

- Les **événements** sont un mécanisme permettant de réagir aux interactions de l'utilisateur.
- La liaison d'événement : est réalisée lorsque nous associons un événement à une méthode de notre classe.
- Dans le code ci-dessous, nous entourons l'attribut d'événement click entre parenthèses de l'élément button et fournissons le nom de la méthode dans notre classe, qui générera l'événement.

```
@Component({
 selector: 'app-users',
 template: `<h2>Users</h2>
 <button (click)="refreshUsers()">
 Refresh the users list
 </button>
 <p>{users.length} users</p>
 `
})
export class UsersComponent {
 users: any = [];
 refreshUsers() {
 this.users = [{ name: Rabat }, { name: 'Fès' }];
 }
}
```

Adil Anwar 2022-2023

## Databinding - two-way databinding

- La directive **[`(ngModel)`]** met à jour la valeur de l'input à chaque changement de la variable `searchedText`, d'où la partie `[ngModel]="searchText"`. Elle met donc à jour également l'attribut `searchedText` avec la valeur saisie dans l'input.
- La directive génère un événement depuis un output nommé **(`ngModelChange`)** à chaque fois que l'input est modifié par l'utilisateur, ceci déclenche l'exécution automatique de la méthode `searchText()` (`ngModelChange)="searchText()"`)
- Pour utiliser ces directives il faut ajouter le module **`FormsModule`** aux tableau imports[] défini dans le dans le fichier .ts du component

```
import { FormsModule } from '@angular/forms'
imports: [FormsModule]

<input type="text" [(NgModel)]="searchedText"
class="form-control" (ngModelChange)="searchProduct()" />
</div>
```

## Databinding

- En conclusion
  - Le système de template d'Angular nous propose une syntaxe puissante pour exprimer les parties dynamiques de notre HTML. Elle nous permet d'exprimer du binding de données, de propriétés, d'événements, et des considérations de templating, d'une façon claire, avec des symboles propres :
    - `{()}` pour l'interpolation,
    - `[]` pour le binding de propriété,
    - `()` pour le binding d'événement,
    - `#` pour la déclaration de variable locale

# Propriétés et événements



Adil Anwar 2022-2023

## Interaction entre composants

- Une application Angular est composée de plusieurs composants.
- En utilisant des formulaires et des liens, on peut envoyer des données d'un composant à un autre,
- On peut ajouter le sélecteur d'un premier composant dans la template d'un deuxième composant
  - on appelle le premier composant : composant fils
  - on appelle le deuxième composant : composant parent
- En utilisant les décorateurs `@Input()` et `@Output()` les deux composants peuvent échanger de données.
- `@Input()` : permet à un composant fils de récupérer des données de son composant parent,
- `@Output()` : permet à un composant parent de récupérer des données de son composant enfant,

Adil Anwar 2022-2023

# Composants - Inputs

- Les entrées sont parfaites pour passer des données d'un élément supérieur à un élément inférieur.
- Par exemple, si on veut avoir un composant affichant une liste d'utilisateurs, il est probable qu'on ait un composant supérieur contenant la liste, et un autre composant inférieur affichant un utilisateur
- Le décorateur `@Input()` permet de déclarer une **propriété** comme une **entrée** d'un composant.
- Les entrées spécifient les paramètres que nous attendons de notre composant.
- Lorsque nous spécifions qu'un composant prend une entrée, il est attendu que la classe de définition aura une variable d'instance.

Adil Anwar 2022-2023

## Interaction entre composants

```
@Component({
 selector: 'app-users',
 template: `<app-user [user]="selectedUser">
 </app-user>`
})
export class UsersComponent {
 selectedUser: User = { id: 1, name: 'Salman' };
}

@Component({
 selector: 'app-user',
 template: `<div>{{user.name}}</div>`
})
export class UserComponent {
 @Input() user: User;
}
```

Adil Anwar 2022-2023

# Les entrées d'un composant - Alias

- Si on souhaite exposer la propriété par un terme différent de celui de la propriété, il faudrait ajouter l'alias en paramètre de l'appel du décorateur.

```
@Input('userInput') user: User;
```

```
@Component({
 selector: 'app-user',
 template: `<div>{{user.name}}</div>`
})
export class UserComponent {
 @Input('user') internalUser: User;
}

@Component({
 selector: 'app-users',
 template: `<app-user [user]="selectedUser">
 </app-user>`
})
export class UsersComponent {
 selectedUser: User = { id: 1, name: 'Salman' };
}
```

Adil Anwar 2022-2023

# Composants – Événements

- En Angular, les données entrent dans un composant via des propriétés, et en sortent via des événements.
- Les événements sont un mécanisme permettant à un composant de notifier son parent et au parent de gérer la notification
- Les événements spécifiques sont émis grâce à un *EventEmitter*, et doivent être déclarés dans le décorateur, via l'attribut outputs. Comme l'attribut inputs, il accepte un tableau contenant la liste des événements que le composant peut déclencher.
- Et, comme pour les inputs, on préfère généralement le décorateur `@Output()`

Adil Anwar 2022-2023

# Composants – Événements

- Si on veut émettre un événement appelé userSelected. Il y aura trois étapes à réaliser :
  1. déclarer l'événement en l'annotant du décorateur @Output()
  2. créer un *EventEmitter*
  3. émettre un événement quand le user est sélectionné avec la méthode emit

```
import { Component, Output } from '@angular/core';
@Component({
 selector: 'app-users',
 template: `
 <app-user (userSelected)="modifyUser($event)">
 </app-user>
 `
})
export class UsersComponent {
 modifyUser(event) {
 console.log('Selected user\'s email'+ event.email);
 }
}

Adil Anwar 2022-2023
```

```
import { Component, Output } from '@angular/core';
@Component({
 selector: 'app-user',
 template: `
 <button (click)="clickedUser()">
 {{user.name}}
 </button>
 `
})
export class SelectableUserComponent {
 @Output() userSelected = new EventEmitter<User>();
 clickedUser() {
 this.userSelected.emit(this.user);
 }
}
```

# Atelier : Binding de propriétés et Événements

- Refactorer l'application de gestion des films dans une approche one-way data flow:
- Les formulaires et listings doivent être mis dans des composants spécifiques
- Les composants d'alerte doivent être paramétrables par le message à afficher.

# Directives



Adil Anwar 2022-2023

## Directives

- Les directives sont chargées de fournir des fonctionnalités dans une application Angular et aident à transformer le DOM.
- Les directives sont essentiellement des commandes envoyées à Angular pour exécuter des traitements spécifiques sur des affichages de données ou de templates.
- Il existe deux types de directives
  - directives structurelles
  - directives d'attribut.
- Les directives structurelles vous permettent de modifier la mise en page de la page en manipulant le DOM.
- Une directive d'attribut modifiera simplement le comportement ou l'apparence d'un élément DOM existant.

Adil Anwar 2022-2023

## Directives -- ngIf

- les directives structurelles manipulent la présentation HTML via le DOM. La manipulation consiste à ajouter, supprimer ou modifier un élément.
- Les deux directives structurelles les plus couramment utilisées sont ngIf et ngFor.
- La directive \* ngIf est une directive intégrée fournie avec Angular
- Cette directive évaluera une expression booléenne et, en fonction du résultat, affichera ou masquera un élément HTML.

```
import { Component } from '@angular/core';
@Component({
 selector: 'app-users',
 template: `
 <div *ngIf="users.length">
 <h2>Users</h2>
 </div>`})
export class UsersComponent {
 users: Array<any> = [];
```

Adil Anwar 2022-2023

## Directives - \*ngIf else

- Au lieu de mettre une condition sur un template et son contraire pour afficher deux blocs dont un uniquement doit être affiché, la syntaxe est dorénavant plus simplifiée,

### Angular 2

```
<div *ngIf="condition">Mon Bloc OK</div>
<div *ngIf="!condition">Mon Autre Bloc</div>
```

### Angular 4

```
<div *ngIf="condition; else elseBlock">
 Mon Bloc OK
</div>
<ng-template #elseBlock>
 Mon Autre Bloc
<ng-/template>
```

Adil Anwar 2022-2023

## Directives -- \*ngFor

- **ngFor** : utiliser pour afficher des données d'une façon itérative
- Exemple : afficher les éléments d'un tableau

```
@Component({
 selector: 'app-users',
 template: `

 <li *ngFor="let user of users">{ user.name }
 `
})

export class UsersComponent {
 users = [
 {name: 'ali'}, {name: adil}
];
}
```

Adil Anwar 2022-2023

## Directives -- \*ngFor

- Et pour avoir l'indice de l'itération courante

```
<app-appareil *ngFor="let appareil of appareils ; let i = index"
 [appareilName]="appareil.name"
 [appareilStatus]="appareil.status"
 [indexOfAppareil]= "i"
 [id]="appareil.id"></app-appareil>
```

Adil Anwar 2022-2023

## Directives d'attributs -- ngStyle

- Si on veut changer plusieurs styles en même temps, nous pouvons utiliser la directive ngStyle :
- Note que la directive attend un objet dont les clés sont les styles à définir.

```
<h4 [ngStyle]="{color: getColor()}"> {{ product.name }}</h4>
```

Adil Anwar 2022-2023

## Directives d'attributs -- ngClass

- La directive ngClass est conçue pour vous permettre d'appliquer dynamiquement CSS à un élément. La directive utilise une expression pour évaluer le CSS à appliquer.
- la directive ngClass permet d'ajouter ou d'enlever dynamiquement des classes sur un élément. Comme pour le style, on peut soit définir une seule classe avec le binding de propriété :

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

Adil Anwar 2022-2023

# Atelier Binding et Directives

- Créer une mini application affichant une liste de produits initialement stockés dans un tableau dans le composant list-produit.
- Chaque produit doit être passé comme paramètre au composant fils produit-item,
- Afficher le message « produit disponible » en vert s'il le produit est disponible en stock ou « en rupture de stock » en rouge dans le cas contraire.
- En termes de structure :
  - votre ProduitListComponent contiendra l'array des produits
  - votre ProduitListComponent affichera un ProduitItemComponent pour chaque produit dans l'array
  - Chaque ProduitItemComponent affichera le nom, le prix, la description, et l'image du produit dans le template, un lien pour la description et la catégorie.
  - Chaque ProduitItemComponent affichera un bouton (commander) qui permet d'ajouter le produit dans le panier.

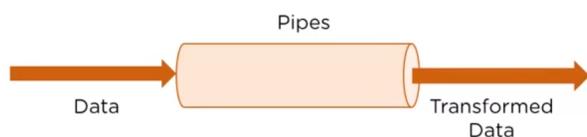
Adil Anwar 2022-2023

## Pipes



# Pipes

- Un pipe vous permet d'utiliser des données en tant qu'entrée et de les **transformer** en une sortie plus souhaitable.



- Les pipes sont définies par le symbole « | » inspiré de pipes Unix
- En effet, les données peuvent être quelque peu cryptiques ou ne pas nécessairement être "conviviales" comme dans un format de données.

Exemple : une date peut prendre la forme suivante: lastUpdate = new Date()

- <p>mise à jour : {{lastUpdate}}</p>
- Ce paragraphe affiche : Sam. Juin 20 2024 04:00:00 GMT +0100 (CET)
- Bien qu'il soit lisible dans une certaine mesure, tous les utilisateurs ne se soucient pas de l'heure ni du décalage.
- Angular contient en fait une collection de pipes intégrés pour les tâches courantes telles que **DatePipe**, **UpperCasePipe**, **LowerCasePipe** et **CurrencyPipe**.
- Pour utiliser les pipes prédéfinies il faut importer le **CommonModule** du package :  
import { CommonModule } from '@angular/common'
- 

# Pipes

- Une pipe a le rôle de modifier une donnée dans le .component.html.

• Exemple : {{ "bonjour" | uppercase }}  
<!-- BONJOUR -->

- Certaines pipes peuvent prendre des paramètres

```
{{ maDate | date:'d MMM y' }}
<!-- affiche 19 oct 2018 -->
```

- Le pipe date est très utile pour afficher les dates sous un format compréhensible par les utilisateurs
- Exemple d'une date : Sat. Jun 20 2017 04:00:00 GMT -0800 (Heure standard du Pacifique)
- Il est possible de chaîner les pipes

```
|{{ maDate | date:'d MMM y' | uppercase }}
<!-- affiche 19 OCT 2018 -->
```

## Le pipe async

- Le pipe `async` est un cas particulier mais extrêmement utile dans les applications Web, car il permet de gérer des données **asynchrones**, par exemple des données que l'application doit récupérer sur un serveur (une promise ou observable).
- pour simuler ce comportement asynchrone on va créer une Promise qui va se résoudre au bout de quelques secondes. Modifiez `lastUpdate` comme suit :

```
lastUpdate = new Promise((resolve, reject) => {
 const date = new Date();
 setTimeout(
 () => {
 resolve(date);
 }, 2000
);
});
```

- Si vous enregistrez le fichier, l'application vous créera une erreur :
  - Error: InvalidPipeArgument: '[object Promise]' for pipe 'DatePipe'.
- **Explication** : au moment de générer le DOM, `lastUpdate` est encore une Promise et n'a pas de valeur modifiable par les pipes.
- **Solution** : ajouter le pipe `AsyncPipe` en début de chaîne pour dire à Angular d'attendre l'arrivée des données  
`<p>Mis à jour : {{ lastUpdate | async | date: 'y MMMM d' }}</p>`

## Pipes : création

- Une pipe est une classe décorée par le décorateur `@pipe`
- Elle implémente la méthode `transform(value: any, args?:any)` de l'interface ***PipeTransform***
- Elle peut prendre des paramètres
- Pour créer une pipe
  - ng generate pipe nomPipe
- Exemple : définissons une pipe (getchar qui retourne la première lettre d'une chaîne de caractères)
  - ng generate pipe getChar
- Cette commande va générer deux fichiers :
  - `get-char.pipe.ts`
  - `get-char.pipe.spec.ts`

## Pipes : Création

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
 name: 'getChar'
})
export class GetCharPipe implements PipeTransform {

 transform(value: any, args?: any): any {
 return null;
 }
}
```

- value correspond à la valeur de la chaîne qu'on va modifier.

## Pipes : Création

- Modifions la méthode transform dans le fichier get-char.pipe.ts pour retourner la première lettre

```
transform(value: any, args?: any): any {
 return value[0];
}
```

- Pour tester, écrire dans un .component.html

```
 {{ "bonjour" | getchar }}
<!-- affiche b -->
 {{ "wick" | getchar }}
<!-- affiche w -->
 {{ "john" | getchar }}
<!-- affiche j -->
```

## Pipes : Création

- Modifions get-char.pipe.ts pour retourner un caractère à une position donnée (pos)

```
transform(value: any, pos?: any): any {
 if(pos && pos < value.length)
 return value[pos];
 return value[0];
}
```

- Pour tester, écrire dans un .component.html

```
{{ "bonjour" | getchar:2 }}
<!-- affiche n -->
{{ "wick" | getchar:6 }}
<!-- affiche w -->
{{ "john" | getchar }}
<!-- affiche j -->
```

## L'injection de dépendances



# L'injection de dépendance

- La documentation sur Angular définit l'injection de dépendance (DI) comme «un moyen pour fournir à une nouvelle instance d'une classe, les dépendances complètes qu'elle nécessite ».
- L'injection de dépendances est un **design pattern** bien connu. Un composant peut avoir besoin de faire appel à des fonctionnalités qui sont définies dans d'autres parties de l'application (un service, par exemple).
- C'est ce que l'on appelle une dépendance : **le composant dépend du service**. Au lieu de laisser au composant la charge de créer une instance du service, **l'idée est que le framework crée l'instance du service lui-même, et la fournit au composant** qui en a besoin.
- C'est un concept largement utilisé côté serveur, mais AngularJS 1.x était un des premiers à l'apporter côté client.

Adil Anwar 2022-2023

## L'injection de dépendance : Principe de fonctionnement

- Lorsque Angular crée un composant, il demande d'abord à un injecteur les services requis.
- Un injecteur maintient un conteneur d'instances de service qu'il a créé précédemment.
- Si une instance de service demandée n'est pas dans le conteneur,
- l'injecteur en fait une et l'ajoute au conteneur avant de renvoyer le service à Angular.
- Lorsque tous les services demandés ont été résolus et retournés, Angular peut appeler le constructeur du composant avec ces services comme arguments.
- Il s'agit d'une injection de dépendance

Adil Anwar 2022-2023

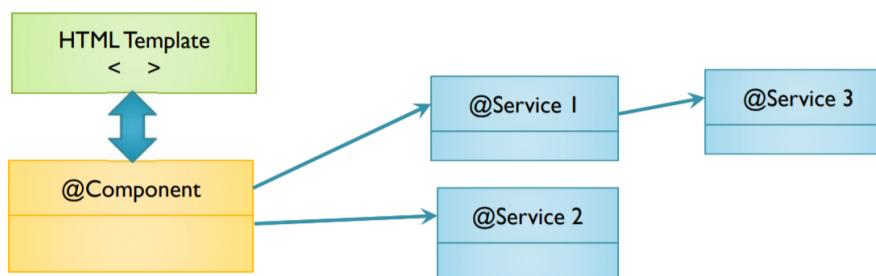
# Injection de dépendances - Angular

- Pour faire de l'injection de dépendances, on a besoin :
  - d'une façon d'enregistrer une dépendance, pour la rendre disponible à l'injection dans d'autres composants/services.
  - d'une façon de déclarer quelles dépendances sont requises dans chaque composant/service.
- Le framework se chargera ensuite du reste. Quand on déclarera une dépendance dans un composant, il regardera dans le registre s'il la trouve, récupérera l'instance existante ou en créera une, et réalisera enfin l'injection dans notre composant.

Adil Anwar 2022-2023

## Services

- Les services sont des classes spécifiques dans Angular qui sont conçus pour créer une logique réutilisable pouvant être injectée dans plusieurs composants.
- Un service est généralement une classe avec un but étroit et bien défini.
- Généralement, les composants se limitent à l'affichage et à la gestion des événements utilisateurs dans la vue du composant.
- L'exécution des traitements en local ou en back end sont attribués aux services.
- Quand un événement survient dans la vue, le composant fait appel à des fonctions dans les services pour effectuer des traitements et fournir des résultats



Adil Anwar 2022-2023

# Services – Création

- Le pattern DI peut être utilisé dans une application Angular pour créer une classe (appelé un service) et l'injecter dans d'autres composants de votre application.
- le service importe le décorateur **@Injectable**, ce qui permet d'injecter ce service dans d'autres composants.
- Par exemple, dans cette application on créer une classe ProductService qui va faire un traitement et doit partager le résultat avec plusieurs composants,
- Au lieu d'écrire ce code dans votre application plusieurs fois, vous avez décidé de créer une classe simple qui implémente cette logique:
- Pour créer un service il faut utiliser le CLI avec une simple commande:

```
ng generate service product
```

- Exemple

```
import { Injectable } from '@angular/core';
@Injectable()
export class ProductService {
 constructor() {
 }
}
```

Adil Anwar 2022-2023

# Services -- Enregistrement

- Pour utiliser un service, il faut préalablement enregistrer un fournisseur de ce service avec l'injecteur.
- Un fournisseur de service est une fabrique qui permet de gérer l'instanciation des services.
- Vous pouvez enregistrer les fournisseurs dans les modules ou dans les composants.
- En général, ajoutez des fournisseurs dans le module racine afin que la même instance d'un service soit disponible partout.
- Pour cela il faut mettre à jour le module **appModule** pour ajouter la classe *GitSearchService*

**providers:** [  **ProductService, AuthService, AuthGuardService** ],

Adil Anwar 2022-2023

## Services -- Utilisation

- Enfin, vous pouvez utiliser la classe ProductService dans les composants
  - Ajouter une instruction d'importation pour la classe
  - Injecter une instance de classe via le constructeur en ajoutant un paramètre de constructeur correspondant au type de classe

```
export class ProductListComponent implements OnInit {
 products : Array<Product>;
 selectedProducts : Array<Product>;
 constructor(private productService : ProductService) {
 }

 ngOnInit() {
 this.products = this.productService.products;
 }
}
```

Adil Anwar 2022-2023

## Atelier Services

- Créer un service ProductService exposant les méthodes suivantes:
  - add(p: Produit): void
  - get(id: number): Product
  - getAll(): Array<Produit>
- Refactorer l'application en centralisant la gestion du tableau des produits dans le service
- Ajouter un composant ProduitDetailComponent pour afficher les informations détaillées d'un Produit à partir de la liste des produits.

Adil Anwar 2022-2023



Adil Anwar 2022-2023

## Communication avec HttpClient

- Dans les communications entre les clients front-end et les services back-end en utilisant le protocole HTTP, nous constatons que les navigateurs modernes utiliseront le protocole XMLHttpRequest ou l'API fetch().
- **HttpClient** est une bibliothèque HTTP asynchrone intégrée à Angular, remplaçant d'autres méthodes telles que XMLHttpRequest, \$.ajax() de jQuery ou l'API fetch.
- Angular 4.3 et versions ultérieures utilisent la bibliothèque **HttpClient** comme mécanisme permettant de gérer les requêtes http dans une application Angular.

Adil Anwar 2022-2023

# Communication avec HttpClient

- Avant d'utiliser HttpClient, il faut d'abord importer le module **HttpClientModule** dans le module racine de l'application.

```
imports: [
 BrowserModule,
 RouterModule.forRoot(appRoutes),
 FormsModule,
 HttpClientModule
],
```

- Une fois que vous avez importé le module, vous pouvez l'utiliser dans votre propre composant en injectant HttpClient au service que nous créons.

```
export class ProductService {
 constructor(private http : HttpClient) { }

 getProducts() {
 return this.http.get('http://localhost:3000/api/products')
```

Adil Anwar 2022-2023

# Consommation des données avec les promises

- Il existe une deuxième alternative à la bibliothèque RxJs qui consiste à convertir une réponse HTTP de type Observable en un objet JavaScript de type **promise**.
- Exemple : `http.get(requestUrl).toPromise();`
- Avec l'objet promise on peut ensuite ajouter des gestionnaires d'événements en cas d'erreur ou si l'événement asynchrone est exécuté avec succès:

```
http.get('http://localhost:3000/api/products').toPromise()
```

```
.then((response) => {
 jsonResult = response.json();
}, (error) => { console.log('Error Occurred:', error)
});
```

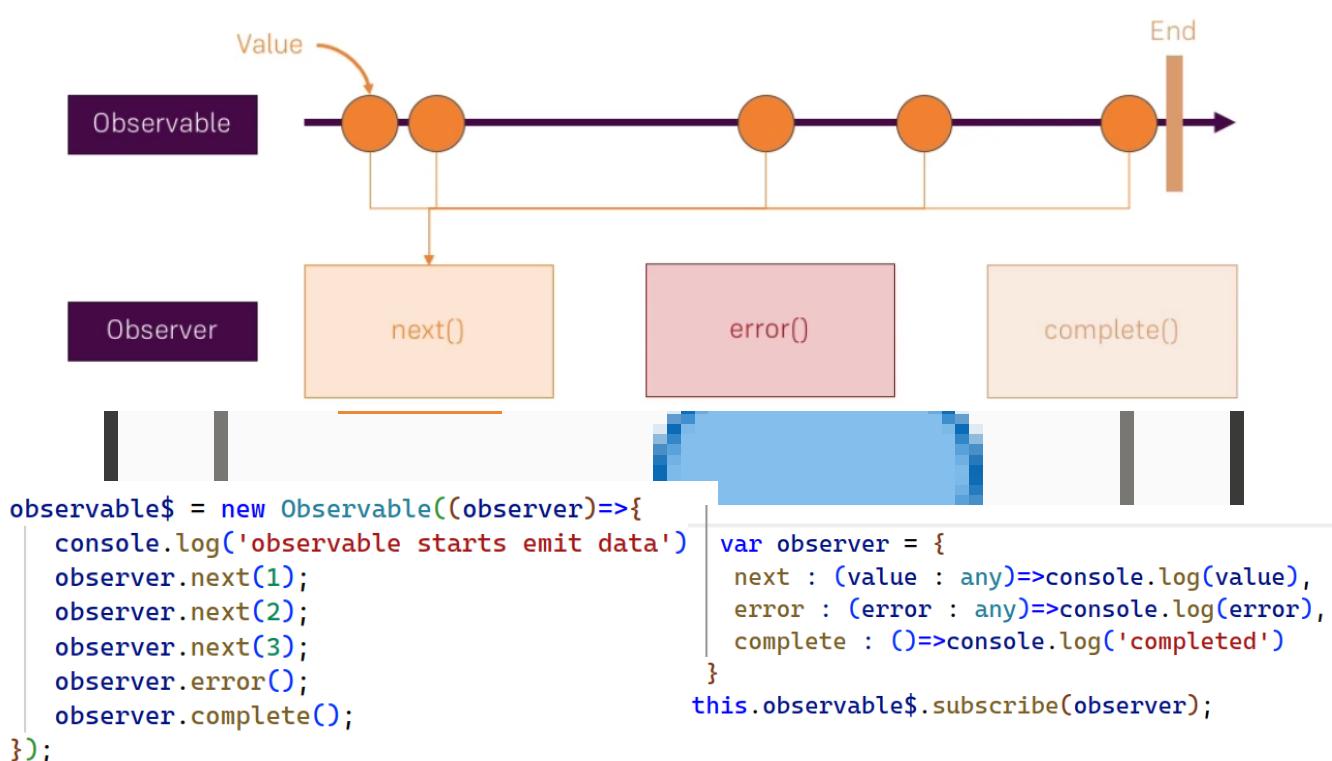
Adil Anwar 2022-2023

# Programmation réactive avec RxJs

- Pour réagir à des événements ou à des données de manière asynchrone, Angular utilise la bibliothèque **RxJS**,
- La bibliothèque la plus populaire de programmation réactive dans l'écosystème JavaScript est **RxJS**. Et c'est celle choisie par Angular.
- un **Observable** est un objet qui émet des informations auxquelles on souhaite réagir. un observable est comme une collection (tableau). Mais c'est une collection asynchrone, dont les éléments arrivent au cours du temps.
- Ces informations peuvent venir de la communication avec un serveur : le service Angular **httpClient** emploie les Observables.
- À cet **Observable**, on associe un **Observer** — un bloc de code qui sera exécuté à chaque fois que l'Observable émet une information.
- L'Observable émet trois types d'information :
  - des données (une nouvelle donnée !)
  - des erreurs (quand il y en a)
  - des terminaisons (fin du flux)

Adil Anwar 2022-2023

# Programmation réactive avec RxJs



# Communication avec HttpClient

- Par défaut, le service HttpClient réalise des requêtes AJAX avec XMLHttpRequest.
- Il propose plusieurs méthodes, correspondant au verbes HTTP communs :
- Chaque méthode renvoie un objet spécial de type **Observable** permettant à votre code d'application d'attendre de manière **asynchrone** la fin de la requête distante.
  - **get** : http.get(url: string) : Observable<Response>
  - **delete** : http.delete(url: string) : Observable<Response>
  - **post** : http.post(url: string, body: any) : Observable<Response>
  - **put** : http.put(url: string, body: any) : Observable<Response>

Adil Anwar 2022-2023

# Communication avec HttpClient

- L'appel à httpClient.get retourne un **Observable**, on peut **s'abonner** à cet observable pour obtenir la réponse. Pour cela, vous utiliserez la fonction **subscribe()**
- La réponse est un objet Response, avec quelques champs et méthodes bien pratiques. On peut ainsi facilement accéder au code status, au headers, contenu en format json etc.
- Le corps de la réponse est la partie la plus intéressante. Mais pour y accéder, il faudra utiliser une méthode :
  - **text()** si on s'attend à du texte;
  - **json()** si on s'attend à un objet JSON, le parsing étant fait automatiquement

```
getProducts() {
 return this.http.get(url: 'http://localhost:3000/api/products')
 .subscribe(next: (response : Response)=>{
 console.log(response.status);
 console.log(response.headers);
 console.log(response.json());
 }
);
}
```

Adil

# Communication avec HttpClient

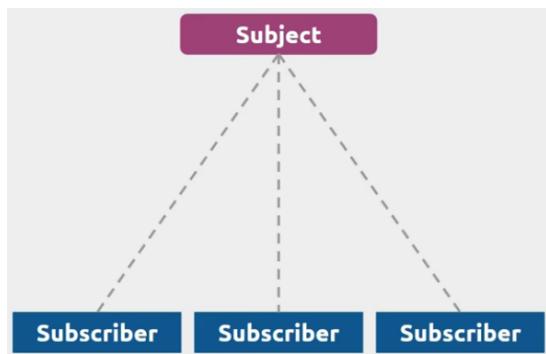
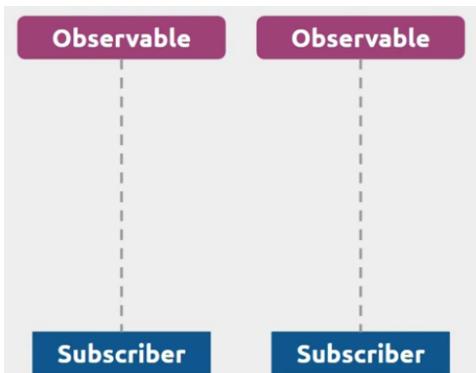
- Envoyer des données est aussi trivial.
- Il suffit d'appeler la méthode **post()**, avec l'URL et l'objet à poster :

```
ajouterProduct(product : Product){
 this.http.post(
 url: 'http://localhost:3000/products',
 product);
}
```

Adil Anwar 2022-2023

# Programmation réactive avec RxJs

- il existe un type d'observable qui permet non seulement de réagir à de nouvelles informations, mais également d'en émettre.
- Imaginez une variable dans un service, par exemple, qui peut être modifié depuis plusieurs components ET qui fera réagir tous les components qui y sont liés en même temps. Voici l'intérêt des **Subjects**.
- Un observable ne peut avoir qu'un **seul observateur**(subscriber)
- Un **subject** pour avoir **plusieurs** observateurs : Il autorise la souscription de plusieurs observateurs



# Programmation réactive avec RxJs

```
const subject = new Subject<number>();

subject.subscribe({
 next: (value) => console.log('A ', value)
})

subject.subscribe({
 next: (value) => console.log('B ', value)
})

subject.next(1);
subject.next(2);
subject.next(3);
```

Adil Anwar 2022-2023

# Programmation réactive avec RxJs

- Exemple

1. Déclaration d'un objet de type subject dans le service

```
activites: Array<any>= [];
activitesSubject = new Subject<any>();
```

2. créer une méthode qui, quand le service reçoit de nouvelles données, fait émettre ces données par le **Subject** et appeler cette méthode dans toutes les méthodes qui en ont besoin ;

```
getActivitesFromServer(){
 this.http.get(url: BACKEND_URL+'activites').subscribe(next: (activites : any[])=>{
 this.activites = activites;
 this.emitActivitesSubject();
 });
}

emitActivitesSubject() {
 this.activitesSubject.next(this.activites.slice());
}
```

Adil Anwar 2022-2023