

ANGULAR

Développement Web



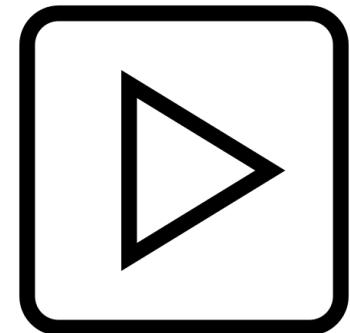
Objectifs pédagogiques

Comprendre et mettre en place une application web à l'aide du framework Angular



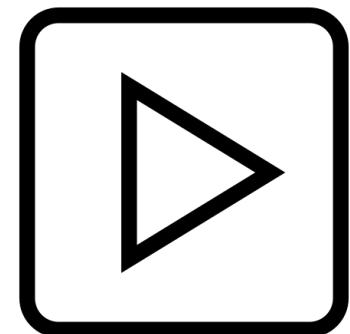
Sommaire (1/2)

- Historique et Installation
- Types de données, Boucles et conditions, Fonctions
- JavaScript orienté objet et TypeScript
- Angular et ses composants
- Les Gabarits (Template)
- La hiérarchie des composants
- Les Services



Sommaire (2/2)

- Les Formulaires
- Le routeur de navigation
- Programmation réactive et observables
- Tests unitaires et Tests de bout en bout
- Aller plus loin



Historique et installation

Développement Web



Angular

AngularJS

- En 2010, création du Framework AngularJS
 - Utilise JQuery ou sa JQLite afin de modifier le DOM
- Très mauvaise position au niveau des performances par rapport à ses concurrents du moment
- Son architecture est basée sur le modèle MVC
- Aucun support mobile



Angular 1/3

- En 2014, création du Framework Angular et mise en production en 2016
- Grand mécontentement pour les développeurs utilisant AngularJS
 - Angular 2 n'est pas une mise à jour
- Gain de productivité remarquable car composant réutilisé à l'infini
- Abandon l'architecture MVC pour une architecture modulaire
- Basée sur une programmation orientée composant



Angular 2/3

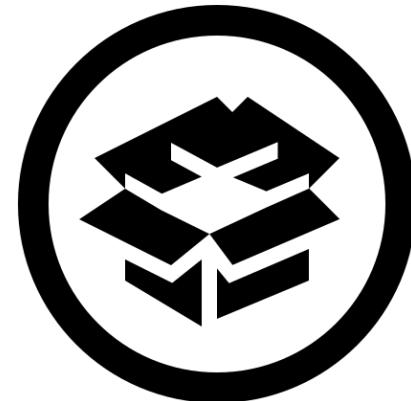
- Angular 2 s'appuie sur TypeScript, JavaScript et ECMAScript ajoutés à l'optimisation de code grâce au typage et à l'orienté objet
- Performances ont considérablement augmenté grâce à l'injection de dépendances et à l'inversion de contrôle
- Support mobile largement présent

Angular 3/3

- Angular suit le principe de versions sémantiques 2.0.0 décrit dans un document disponible sur Internet à l'adresse : <https://semver.org>
- La documentation de la prochaine version d'Angular est disponible :
 - <http://next.angular.io>
- Google a promis une version majeure tous les six mois
 - plusieurs versions mineures par semaine
- En cas d'abandon d'une fonctionnalité dans une nouvelle version :
 - une nouvelle façon d'effectuer la même chose est proposée
- Les mises à jour d'Angular et de son utilitaire en ligne de commande sont alignées

Installation

Développement Web



Angular

Outils

- **Visual Studio Code**
 - environnement de développement écrit par Microsoft au cours de l'année 2015 et entièrement gratuit
 - **Open source, visible :** <https://github.com/Microsoft/vscode>
 - <https://code.visualstudio.com/download>
- **Google Chrome**
 - Sorti en 2008, il est le navigateur le plus utilisé dans le monde en 2012
- **Node.js (version >= 10.13.0 pour Angular CLI)**
 - Permet l'exécution de code JavaScript sans être à l'intérieur d'un navigateur web
 - <https://nodejs.org/en/download/>
- **NPM**
 - Partie intégrante de Node.js
 - recense, installe et désinstalle les paquets et dépendances nécessaires

Les commandes NPM

- **npm –version**
 - Retourne la version
- **npm config edit**
- **npm set init.author.name** SAUVAGE Boris
- **npm set init.author.email** sauvageboris.pro@gmail.com
- **npm set init.author.url** <http://www.mywebsite.fr>
 - Configure NPM pour la création des projets
- **npm init**
 - démarrer un nouveau projet JavaScript, créer un fichier **package.json** indispensable
- **Angular CLI génère automatiquement un fichier package.json lors de l'initialisation d'un nouveau projet**

Angular CLI

- Outil de référence pour l'initialisation et la gestion d'un nouveau projet Angular
 - Simple et complet
 - Permet d'initialiser un projet Angular
- Permet la génération automatique de composants, routes et services
- Permet le lancement d'un serveur web pour le rendu
- Paquet JavaScript
 - installation possible avec **npm**
 - **npm install -g @angular/cli**
- L'argument **-g** permet l'installation de façon globale sur l'ordinateur
 - **npm list -g --depth=0** => Permet de voir les dépendances globales

TYPESCRIPT

Développement Web



Historique

- TypeScript est une surcouche du langage JavaScript
- Contributeur principal : Anders Hejlsberg (créateur de Delphi et de Turbo Pascal)
 - Souhait de conserver ce qui fait la force de JavaScript en supprimant les imperfections dépeintes par certains réfractaires au langage
- Il propose :
 - Annotations
 - Enumérations
 - un typage des variables et fonctions
- Se rapproche de ses concurrents célèbres et populaires tels que Java et C++
- Le TypeScript est transcompilé
 - code TypeScript transformé en JavaScript et ensuite être intégré à une page HTML

Intégrer TypeScript à un projet Web

- À la racine du projet, installez **TypeScript** avec : `npm install -g typescript`
- Sur **VS Code**, il est possible de créer des fichiers avec **l'extension .ts**

- Afin de transcompiler les fichiers Typescript en fichiers JavaScript :

```
tsc -v
```

```
tsc fichier.ts
```



- Il est ensuite possible d'intégrer ces fichiers JavaScript via **le <script></script>**
- Il est aussi possible de lancer le fichier directement avec node :

```
node fichier.js
```

Typage sur TypeScript

- **TypeScript** permet d'indiquer au compilateur et aux développeurs quel est le type attendu dans une variable et même dans le retour d'une fonction
- Pour typer une variable, il suffit d'indiquer son type : `let uneVariable: string;`
- Pour une fonction, c'est le même principe :
`function uneFonction(): string {
 return 'une chaîne de caractères';
}`
- Il existe 7 types :
 - Une chaîne de caractères (**string**)
 - Un nombre (**number**)
 - Un booléen (**boolean**)
 - Un tableau (**string[]**)
 - Une énumération (**myEnum**)
 - Aucun type (**void**)
 - Un type, mais non défini pour le moment (**any**)

Typage sur TypeScript

- Exemple avec les différents types :

```
let uneChaineDeCaractere: string;
// Nombre
let unNombre: number;
// Booléen
let unBoolean: boolean;
// Tableaux
let unTableauDeChaineDeCaracteres: string[];
let unTableauDeNombres: number[];
// Enumération
enum uneEnumeration {}
let uneVariableDeTypeEnum: uneEnumeration;
// Sans aucun type
let uneVariableSansType: void;
// Sans type défini pour l'instant
let uneVariableSansTypeDefiniPourLinstant: any;
```

Les énumérations

- Une **énumération** est une liste de constantes
- Ressemble énormément au type tableau, mais diffère beaucoup dans sa déclaration

```
enum PointsCardinaux {  
    N,  
    S,  
    E,  
    O  
}
```

```
// Déclaration  
let uneDirection: PointsCardinaux;  
// Utilisation  
uneDirection = PointsCardinaux.N; // uneDirection = 0  
uneDirection = PointsCardinaux.S; // uneDirection = 1  
uneDirection = PointsCardinaux.E; // uneDirection = 2  
uneDirection = PointsCardinaux.O; // uneDirection = 3
```

Les fonctions

- Déclarer une fonction en utilisant **TypeScript** ajoute quelques éléments à sa signature

```
function traducteurMajuscule(mot: string): string {  
    return mot.toUpperCase();  
}  
  
let phrase: string = 'Il était une fois';  
let tabMots: string[] = phrase.split(' ');  
for (let i = 0; i < tabMots.length; i++) {  
    console.log(traducteurMajuscule(tabMots[i]));  
}
```

tsc fichier.ts

node fichier.js

IL
ÉTAIT
UNE
FOIS



- Cette fonction prend un mot de type chaîne de caractères en argument et le traduit en majuscules

Les fonctions particulières

- **Fonction anonyme** : Une fonction peut-être directement affectée à une variable

```
let motTraduit = function traducteurMajuscule(mot: string): string {  
    return mot.toUpperCase();  
};
```

- **Fonction lambda ou fléchées** : grâce à ES6, il est possible de raccourcir avec =>

```
let motTraduit = (mot: string): string => {  
    return mot.toUpperCase();  
};
```

```
let value = motTraduit("example")
```

- Inutile de préciser que c'est une fonction
- Fonction anonyme donc pas besoin de nom
- Le type de retour de la fonction est implicite

Les classes

- **Classe minimale** : Ressemble à la syntaxe d'une classe Java
 - contient des attributs, un constructeur, des méthodes et des accesseurs

```
class Voiture {  
}  
// Déclaration d'une nouvelle voiture  
let maVoiture = new Voiture();
```

- **Attributs** : précédés du mot-clé **private** afin de respecter l'encapsulation

```
class Voiture {  
    private _kilometrage: number;  
    private _proprietaire: string;  
    private _couleur: string;  
}
```

Les getters/setters

- Permettent d'accéder aux attributs privé d'une classe et de les modifier

```
// Getters
get kilometrage() { return this._kilometrage; }
get proprietaire() { return this._proprietaire; }
get couleur() { return this._couleur; }
```

```
// Setters
set kilometrage(km: number) { this._kilometrage = km; }
set proprietaire(nom: string) { this._proprietaire = nom; }
set couleur(couleur: string) { this._couleur = couleur; }
```

```
// Déclaration d'une nouvelle voiture
let maVoiture = new Voiture();
```

```
// Utilisation des setters
maVoiture.proprietaire = 'Actarus';
maVoiture.kilometrage = 120000;
maVoiture.couleur = 'grise';
```

```
// Affichage de la voiture grâce aux getters
console.log('Ma voiture : ' + maVoiture.proprietaire + ' - ' + maVoiture.kilometrage + ' km - ' + maVoiture.couleur);
```

Le Constructeur

- Permet de construire une l'instance grâce à des paramètres précis

```
class Voiture {  
    // Attributs  
    private _kilometrage: number;  
    private _proprietaire: string;  
    private _couleur: string;  
    // Le constructeur  
    constructor(kilometrage: number, proprietaire: string, couleur: string) {  
        this._kilometrage = kilometrage;  
        this._proprietaire = proprietaire;  
        this._couleur = couleur;  
    }  
}
```

```
let maVoiture = new Voiture(180000, 'CapitaineFlam', 'Rouge');
```

Les Méthodes

- Une méthode est une fonction qui représente une action possible de l'objet
- **Exemple :** La voiture peut **avancer** ou **reculer**

```
avance(nbreDeMetres: number): void { this._kilometrage += nbreDeMetres; }
recule(nbreDeMetres: number): void { this._kilometrage -= nbreDeMetres; }
```

- **Utilisation :**

```
let maVoiture = new Voiture(180000, 'CapitaineFlam', 'Rouge');
console.log('kilométrage au départ : ' + maVoiture.kilometrage);
console.log('J'avance ...');
maVoiture.avance(100);
console.log('Je recule ...');
maVoiture.recule(50);
console.log('kilométrage à l'arrivée : ' + maVoiture.kilometrage);
console.log('Si quand j'avance, tu recules, ...');
```

L'héritage

- **Une classe** peut hériter d'une autre classe

- La classe qui hérite de l'autre possède les mêmes attributs et méthodes que celle-ci, mais peut en posséder d'autres
- Permet de factoriser les attributs et méthodes redondants

```
class TracteurTondeuse extends Voiture {  
    private _nbHelices: number;  
  
    constructor(kilometrage: number, propriétaire: string, couleur: string) {  
        super(kilometrage, propriétaire, couleur);  
        this._nbHelices = 2;  
    }  
  
    abaisse(nbreDeCm: number) {  
        console.log('hélice abaissée de ' + nbreDeCm + 'centimètres');  
    }  
  
    remonte(nbreDeCm: number) {  
        console.log('hélice remontée de ' + nbreDeCm + 'centimètres');  
    }  
}
```

```
let tt = new TracteurTondeuse(12, 'Jean Dupont', 'Vert');  
console.log('Kilométrage : ' + tt.kilometrage);  
tt.abaisse(10);  
tt.avance(0.1)  
tt.remonte(10);  
tt.recule(0.1);  
console.log('Kilométrage : ' + tt.kilometrage);
```

Les interfaces

- Les interfaces Typescript ont pour vocation la vérification de type

```
export interface User {  
    id: number;  
    firstname: string;  
    lastname: string;  
    weight?: number;  
}
```

```
users: User[] = [{  
    id: 1,  
    firstname: "Boris",  
    lastname: "S",  
    weight: 80  
}, ...]
```

- Utilisable pour transmettre des données sans aucun comportement
- Elles sont souvent utilisées en Angular avec **HttpClient** et **HttpClientModule**

Le symbole ? et le symbole !

- Avec TypeScript, il est possible de déclarer une propriété optionnelle avec ?

```
export interface User {  
    id: number;  
    firstname: string;  
    lastname: string;  
    weight?: number;  
}
```

```
users: User[] = [  
    { id: 1, firstname: "John", lastname: "Doe", weight: 87 },  
    { id: 1, firstname: "Boris", lastname: "S" }  
];
```

- Le symbole ! indique qu'une valeur détectée comme non initialisée, l'est pourtant :

```
class Example {  
  
    citation!: string;  
  
    init() {  
        this.citation = "Un problème sans solution est un problème mal posé";  
    }  
}
```

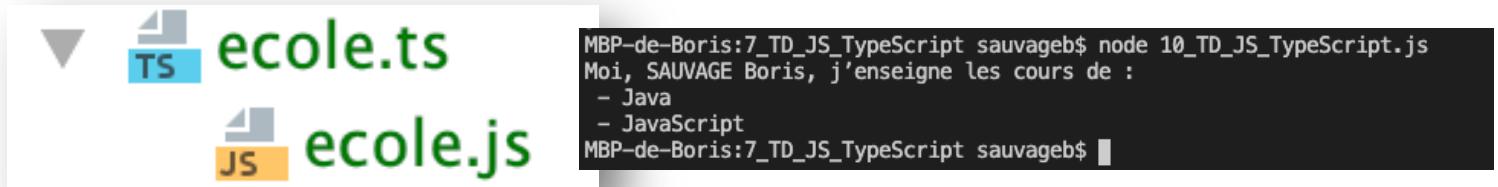
Exercice : Typescript 1/2

- Créez une classe Personne possédant les attributs nom, prénom, âge, taille et poids et constructeurs que vous jugerez utile.
- Créez ensuite une classe Etudiant qui hérite de la classe Personne et qui possède, en plus, un attribut indiquant dans quelle promotion un étudiant se trouve.
- Finalement, créez une classe Formateur qui hérite de la classe Personne et qui possède un attribut supplémentaire qui est la liste des cours qu'un formateur est en mesure de délivrer. Créez une énumération Cours au préalable.
- Et pour finir, créez un objet, instance de Formateur, qui donne les cours de Java et de JavaScript. Puis affichez-le.

TypeScript

Exercice : TypeScript 2/2

- **Résultat attendu :**



- **tsc --init** : initialise le projet avec Typescript
- **tsc -p *.ts -outDir dist** : compile le projet avec le target du fichier **tsconfig.json**

```
{
  "compilerOptions": {
    /* Basic Options */
    // "incremental": true, /* Enable incremental compilation */
    "target": "es5", /* Specify ECMAScript target version: 'ES3' (default), 'ES5', 'ES2015', 'ES2016', 'ES2017', 'ES2018', 'ES2019',
    'ES2020', or 'ESNEXT'. */
    "module": "commonjs",
  }
}
```

ANGULAR CLI

Développement Web



Introduction

- **Angular CLI** est un utilitaire en ligne de commande
 - permet de créer, de gérer, de construire et de tester une application Angular
- Disponible dans les paquets du gestionnaire npm
 - <https://www.npmjs.com/package/@angular/cli>
- Installation du CLI Angular (Command Line Interface)
 - **npm install -g @angular/cli**
 - Installation vérifiable avec la commande **ng v**
- Utilisation du CLI
 - Une fois installé, de nombreuses commandes sont disponibles préfixées de **ng**

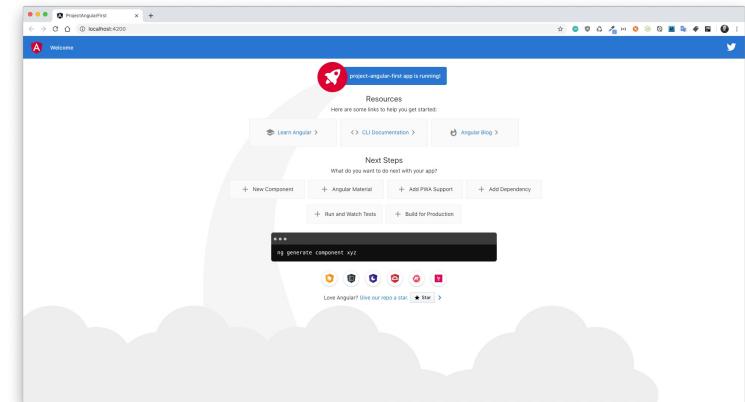
Introduction

- Quelques commandes utilisable depuis n'importe quel dossier de votre machine
 - [ng new](#)
 - [ng serve](#)
 - [ng generate](#)
 - [ng lint](#)
 - [ng test](#)
 - [ng e2e](#)
- Lien vers les commandes détaillées <https://angular.dev/cli>

Angular CLI

Création d'une application Angular

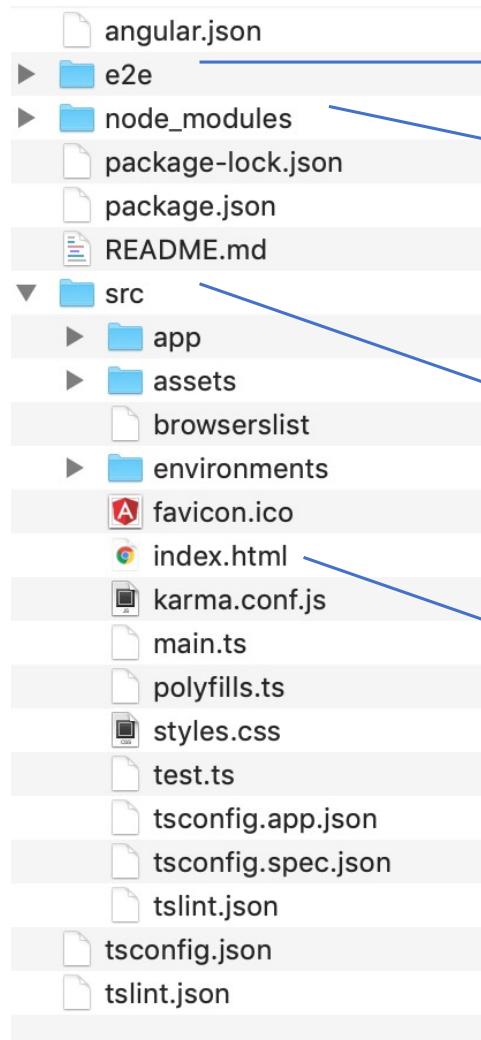
- Créer le projet avec le CLI :
- `ng new project-angular-first`
- Lancer ensuite le projet :
 - `ng serve --open --port 4401`



<http://localhost:4200/>

```
chunk {main} main.js, main.js.map (main) 60.7 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 141 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.15 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 12.4 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 3 MB [initial] [rendered]
Date: 2020-05-03T21:31:04.015Z - Hash: f0396e2f59dcf2939ca1 - Time: 6453ms
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
: Compiled successfully.
```

Architecture d'un projet généré



e2e : tests bout en bout de Protractor

node_modules : dépendances de l'application

src : contient les sources de l'application

fichier principal : point de lancement

Point de lancement de l'application

- index.html est le point d'entrée :

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

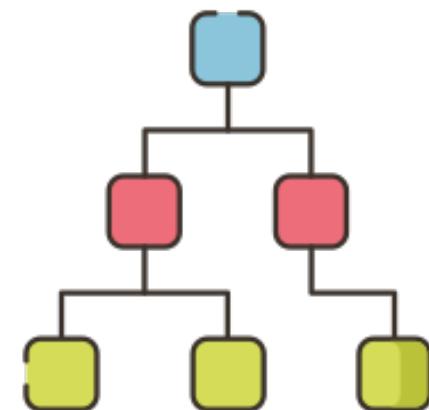


Mais où est le contenu de la page ?

- Il y a une balise Angular : `<app-root></app-root>`
- Cette balise est un *Component* (un bout d'interface de l'appli)
- Quand Angular rencontre la balise `<app-root>` dans le document HTML, il sait qu'il doit en remplacer le contenu par celui du template défini dans le dossier app

Architecture

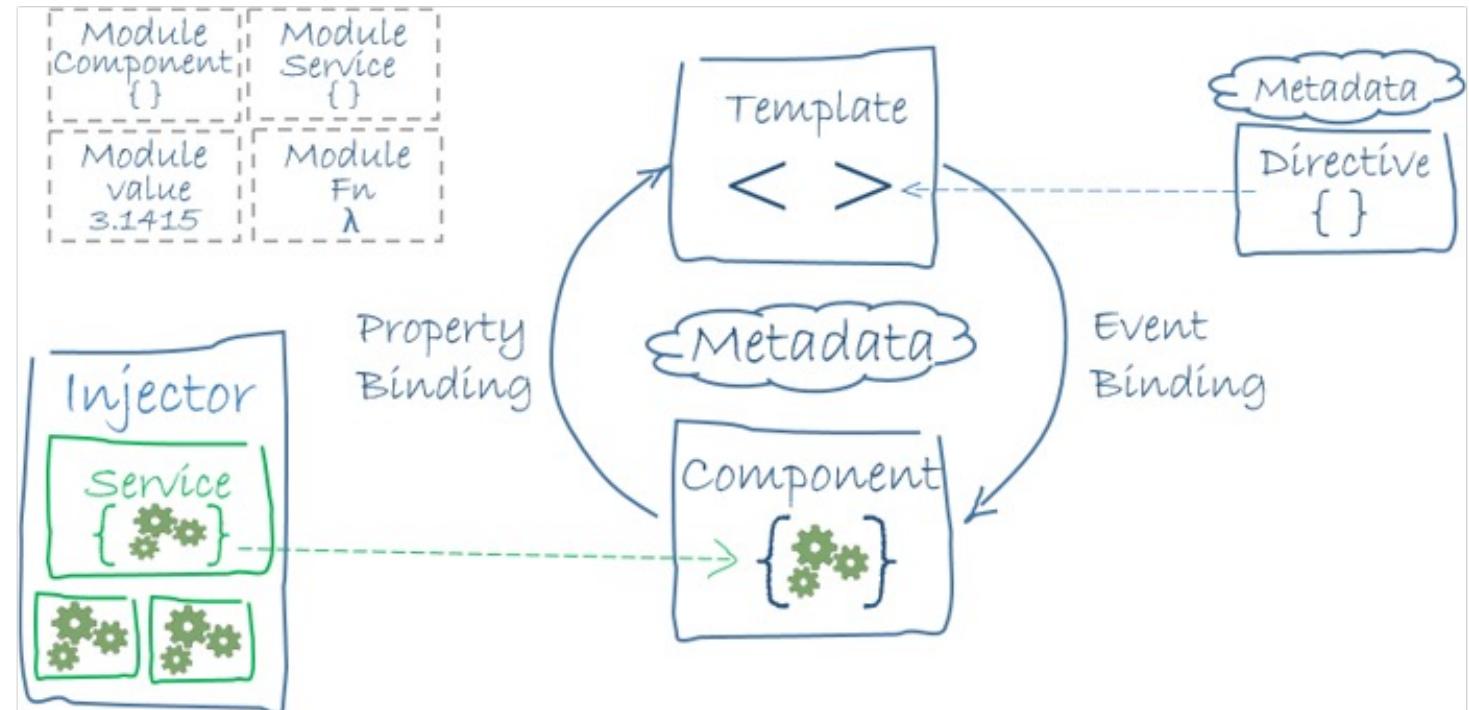
Développement Web



Angular Architecture

Architecture globale

- Composants
- Directives
- Services
- Modules

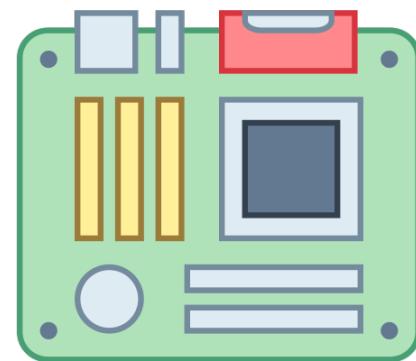


Evolution depuis Angular 15

- Initialement, l'architecture d'un projet Angular est construite sur des modules
- Depuis la version 15, nous sommes passés aux composants standalone
 - Tous les composants peuvent fonctionner sans module
- Les architectures des projets évoluent pour faire disparaître la complexité
 - Les composants gèrent eux même les imports nécessaires
 - Certains services deviennent de simples fonctions
 - La centralisation par module est revisitée

Le composant

Développement Web



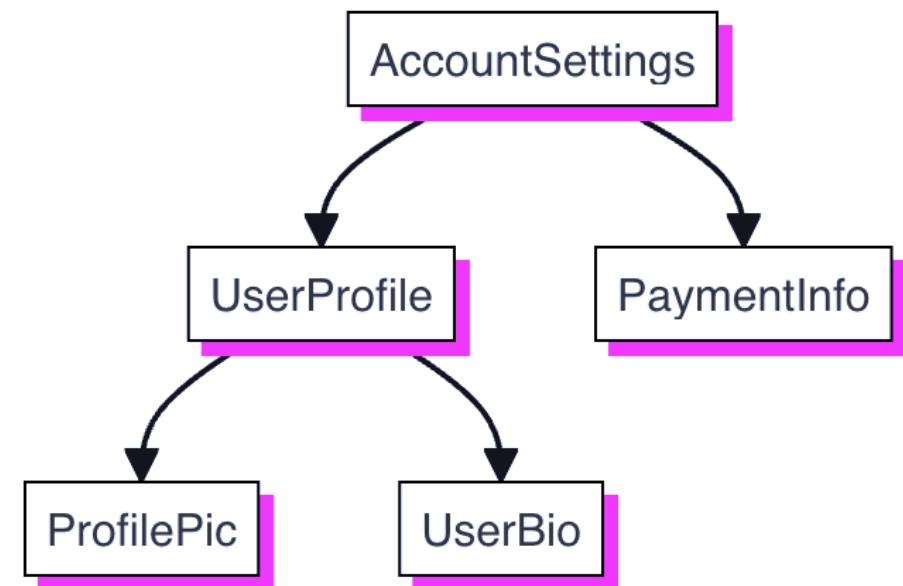
Le composant

Introduction

Angular est basé sur une architecture modulaire :

- Découpage en parties faciles à comprendre avec des responsabilités claires
- Au sein de cette architecture, de nombreux composants seront à créer

```
✓  src
  ✓  app
    >   profile-photo
    ✓  user-profile
      user-profile.component.css
      user-profile.component.html
      user-profile.component.spec.ts
      user-profile.component.ts
      app.component.css
      app.component.html
      app.component.spec.ts
      app.component.ts
```



Le composant

Le composant 1/3

Chaque composant doit avoir :

- Une **Classe Typescript** avec des comportements (gestion d'un état, des entrées utilisateurs, etc)
- Un **modèle HTML** qui contrôle ce qui s'affiche dans le DOM
- Un sélecteur CSS qui définit la manière dont le composant s'utilise dans HTML

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

@Component({
  selector: 'profile-photo',
  standalone: true,
  template: '',
})
export class ProfilePhoto {
  /* Comportements du composant ici */
}
```

ng generate component profilePhoto

Le composant

Le composant 2/3

D'autres métadonnées courantes sont utilisées dans les composants :

- **standalone: true** : Permet de créer des composants sans avoir besoin de modules complexes
- **styles** : Permet d'appliquer du CSS au composant

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

@Component({
  selector: 'profile-photo',
  standalone: true,
  template: '',
  styles: `img { border-radius: 50%; }`,
})
export class ProfilePhoto {
  /* Comportements du composant ici */
}
```

Le composant

Le composant 3/3

Pour séparer le HTML et le CSS dans des fichiers séparés, il est possible d'utiliser :

- templateUrl
- styleUrls

```
// profil-photo.components.ts
import {Component} from '@angular/core';

@Component({
  selector: 'profile-photo',
  standalone: true,
  templateUrl: 'profile-photo.component.html',
  styleUrls: 'profile-photo.component.css',
})
export class ProfilePhoto {
  /* Comportements du composant ici */
}
```

```
<!-- profile-photo.component.html -->

```

```
/* profile-photo.component.css */
img {
  border-radius: 50%;
}
```

Le composant

Les sélecteurs

Chaque composant définit un sélecteur CSS

- [```
@Component\({
 selector: 'profile-photo',
 ...
}\)
export class ProfilePhoto {}
```](https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks>Selectors</a></li><li>• Détermine la manière dont le composant pourra être utilisé</li></ul></div><div data-bbox=)

Un élément peut être rattaché à un sélecteur CSS

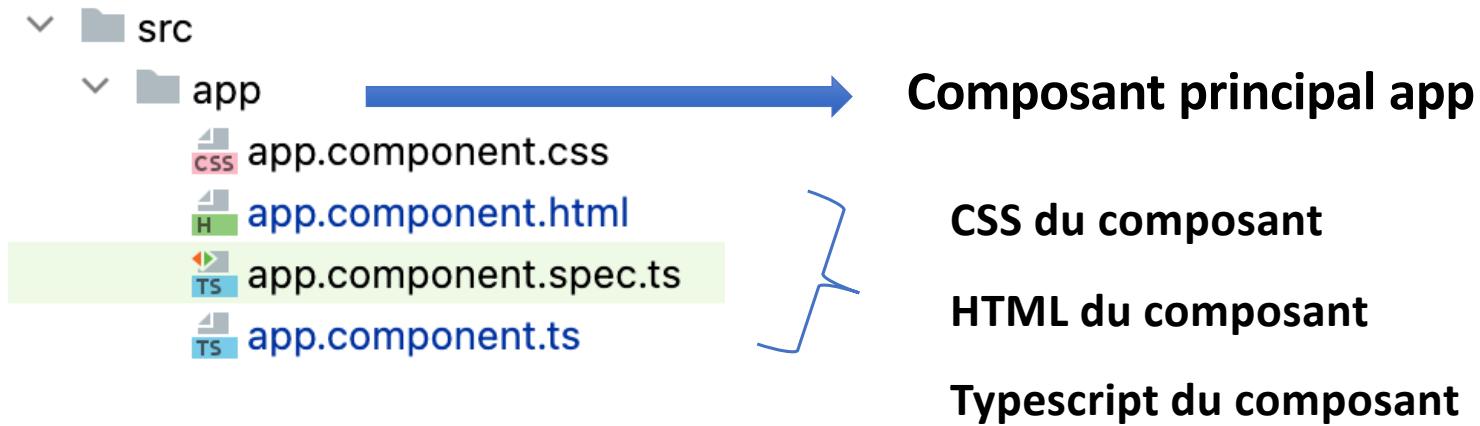
- Si plusieurs éléments correspondent au même sélecteur, alors **Angular gère une erreur**

Les sélecteurs sont sensibles à la casse

## Le composant

# Composant principal 1/2

Par défaut, votre application possède composants principal :



Le sélecteur est **app-root**

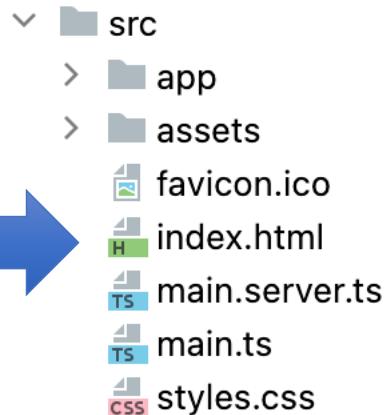
```
@Component({
 selector: 'app-root',
 templateUrl: './app.component.html',
 styleUrls: ['./app.component.css']
})
export class AppComponent {}
```

## Le composant

# Composant principal 2/2

L'application démarre et affiche ce composant principal grâce à index.html

- Lorsqu'un sélecteur est détecté, il est remplacé par le composant



Le composant principal est utilisé  
grâce au sélecteur **app-root**

```
<!doctype html>
<html lang="en">
<head>
 <meta charset="utf-8">
 <title>Angular Demo</title>
 <base href="/">
</head>
<body>
 <app-root></app-root>
</body>
</html>
```

# Importation et Utilisation

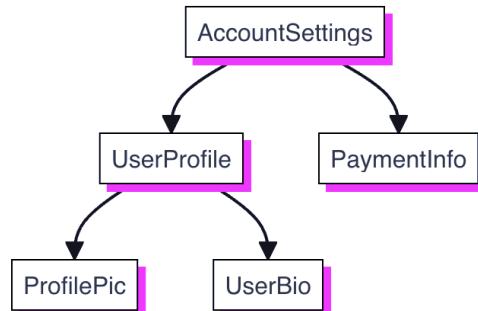
Développement Web

## Importation et utilisation

# Utiliser un composant 1/3

- Vous pouvez utiliser un composant dans un autre

```
@Component({
 selector: 'profile-photo',
 ...
})
export class ProfilePhoto {}
```



```
@Component({
 selector: 'app-user-profile',
 standalone: true,
 imports: [ProfilePhoto],
 template: `
 <profile-photo/>
 <button>Upload a new profile photo</button>
 `,
 ...
})
export class UserProfileComponent {}
```

- Angular fait la correspondance des sélecteurs à la compilation

## Importation et utilisation

# Utiliser un composant 2/3

Un composant doit être rendu disponible pour être utilisé par d'autres composants

- Deux manières sont utilisables : **composant autonome** ou **NgModule**

- Soit en étant un **composant autonome** (Depuis Angular 15)

```
@Component({
 standalone: true,
 selector: 'profile-photo',
})
export class ProfilePhoto {}
```

```
@Component({
 standalone: true,
 imports: [ProfilePhoto],
 template: `<profile-photo />`
})
export class ProfilePhoto {}
```

## Importation et utilisation

# Utiliser un composant 3/3

Un composant doit être rendu disponible pour être utilisé par d'autres composants

- Deux manières sont utilisables : **composant autonome** ou **NgModule**

- Soit en étant dans un fichier **NgModule** (Avant Angular 15)

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

```
@Component({
 ...
 selector: 'profile-photo',
})
export class ProfilePhoto { }
```

```
@Component({
 ...
 template: `<profile-photo />`
})
export class ProfilePhoto { }
```

# CSS et Portée

Développement Web

# Le CSS par composant

Chaque composant peut inclure du CSS s'appliquant sur le DOM de ce dernier :

- Dans un fichier séparé avec `styleUrl`
- Directement avec `styles`

```
@Component({
 ...
 styleUrl: 'profile-photo.component.css',
})
export class ProfilePhoto {}
```

```
@Component({
 ...
 styles: `img { border-radius: 50%; }`,
})
export class ProfilePhoto {}
```

Ces styles font partie du système de modularisation Javascript

- Ils sont inclus automatiquement par Angular lors du chargement du composant
- Angular supporte l'utilisation de Sass, Less, Stylus

CSS et portée

## Le CSS global à l'application

Si vous souhaitez appliquer un CSS global à l'application, vous pouvez utiliser **styles.css**

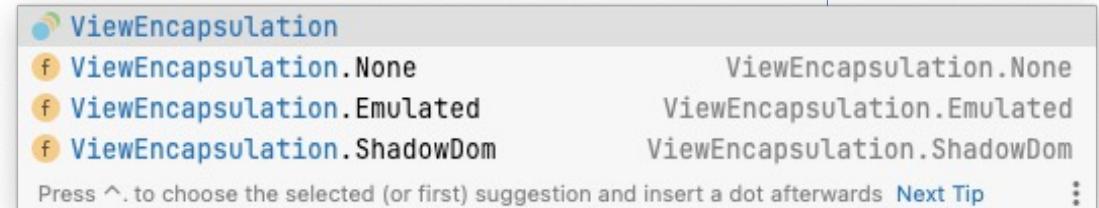
# Portée du style CSS 1/4

Angular contrôle la portée du CSS grâce à 3 modes d'encapsulation :

- ViewEncapsulation.Emulated (par défaut)
- ViewEncapsulation.None
- ViewEncapsulation.ShadowDom

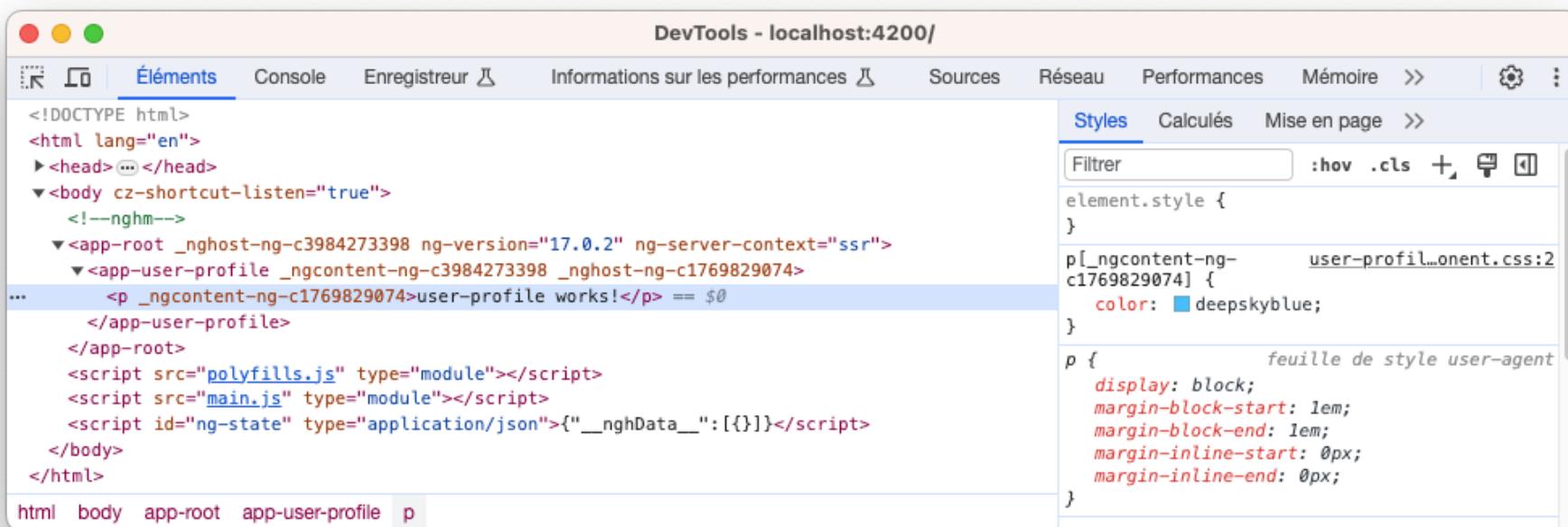
```
import {Component, ViewEncapsulation} from '@angular/core';
import {CommonModule} from '@angular/common';

@Component({
 selector: 'app-user-profile',
 encapsulation: ViewEncapsulation.
 ...
})
export class UserProfileComponent {}
```



# Portée du style CSS 2/4

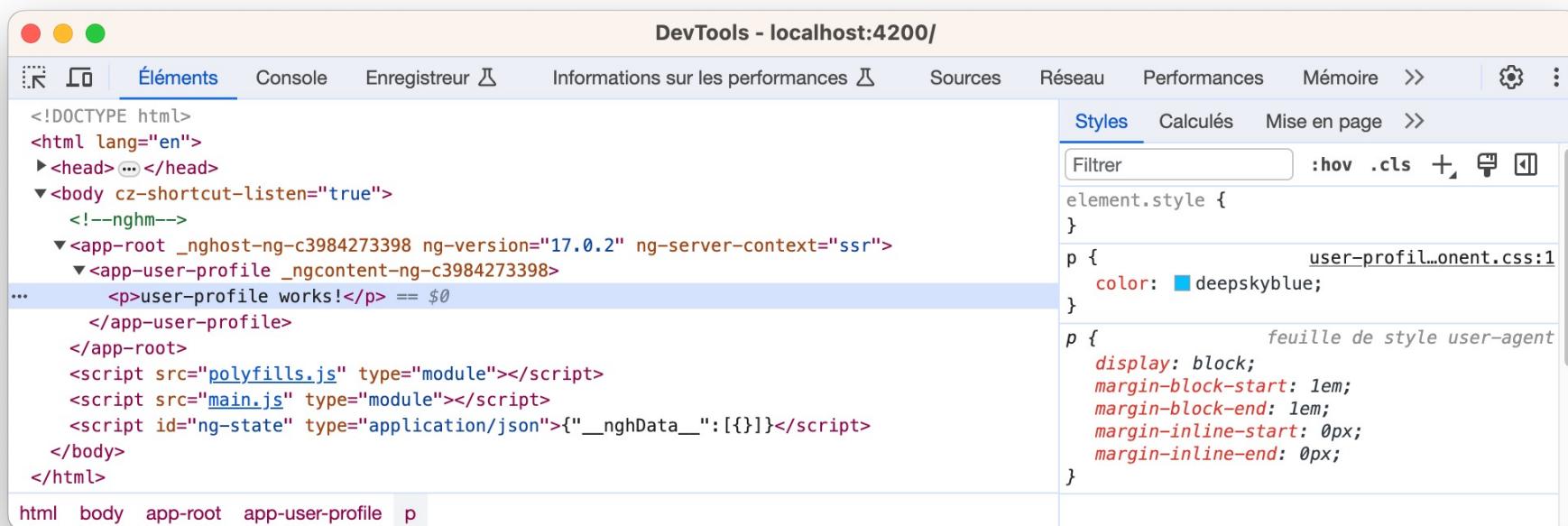
- ViewEncapsulation.Emulated (**par défaut**)
  - Le CSS défini dans le composant peut affecter les éléments en dehors du composant
- Angular limite la portée des règles CSS au composant uniquement
  - Ajoute un préfixe dans le DOM du paragraphe P



## CSS et portée

# Portée du style CSS 3/4

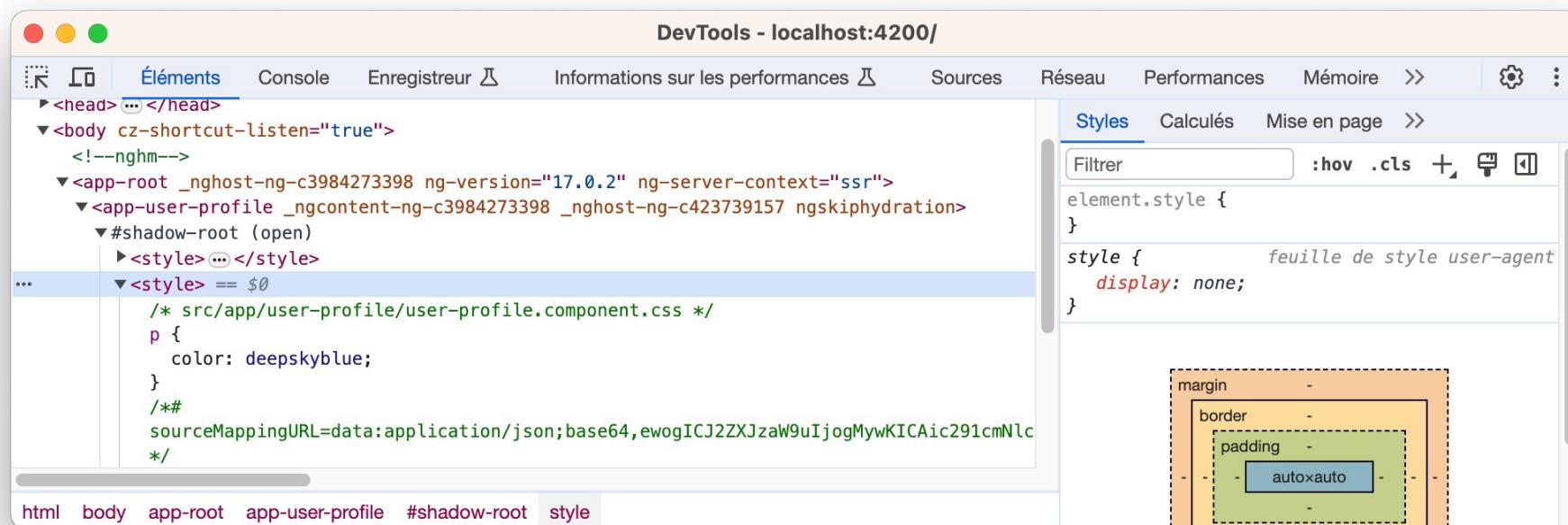
- ViewEncapsulation.None
  - Le shadow DOM ne sera pas utilisé. Le CSS ne sera pas encapsulé
- Il est possible que des règles CSS d'autres composants écrasent les règles du composant
  - Aucun mécanisme d'isolation est proposé



## CSS et portée

# Portée du style CSS 4/4

- ViewEncapsulation.ShadowDom
  - Le shadow DOM est utilisé. Le CSS est encapsulé dans le shadow DOM du composant
- **Angular limite la portée des règles CSS au composant uniquement (Shadow DOM)**
  - [https://developer.mozilla.org/fr/docs/Web/API/Web\\_components/Using\\_shadow\\_DOM](https://developer.mozilla.org/fr/docs/Web/API/Web_components/Using_shadow_DOM)



## Framework CSS

- Angular supporte l'utilisation de nombreux Framework CSS :
  - Material Angular (<https://material.angular.io>)
  - NG Bootstrap (<https://ng-bootstrap.github.io>)
  - PrimeNG (<https://www.primefaces.org>)
  - Tailwind (<https://tailwindcss.com/docs/guides/angular>)
  - ...

# CYCLE DE VIE

Développement Web

## Cycle de vie

# Cycle de vie des composants 1/3

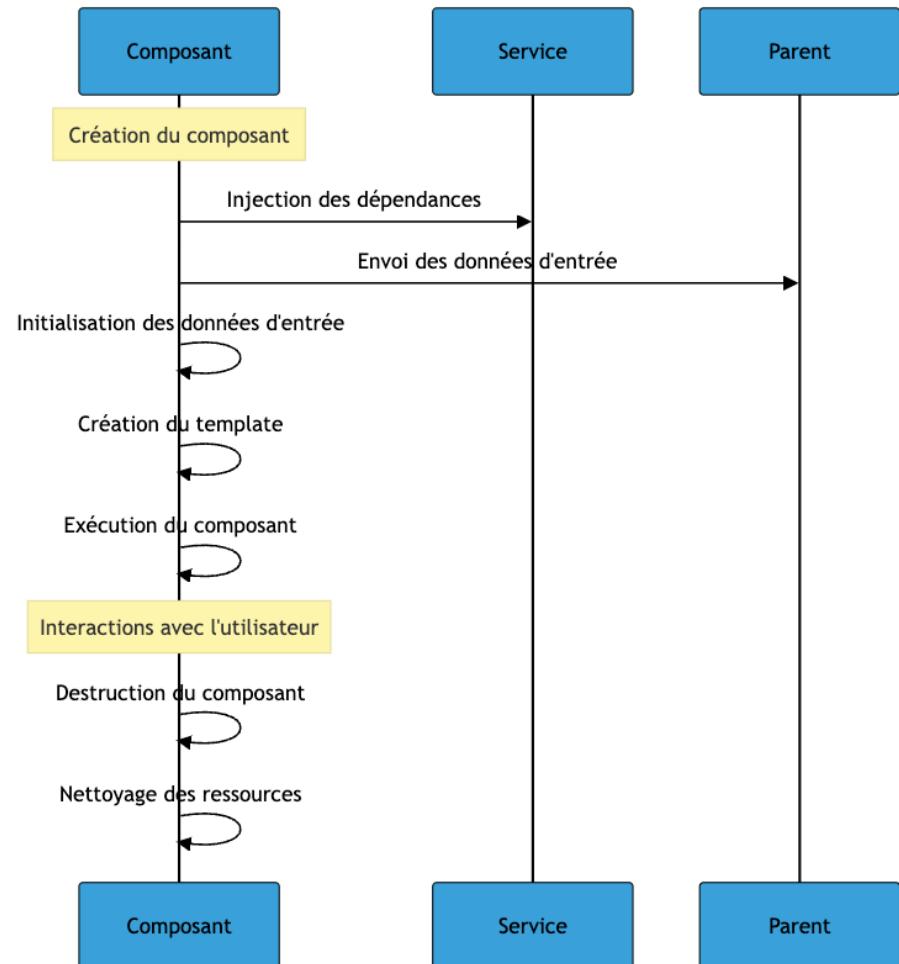
- Ensemble des étapes du composant depuis sa création jusqu'à sa destruction
- Chaque étape est disponible pour exécuter du code à des moments précis du cycle

```
@Component({
 ...
})
export class AppComponent implements OnChanges, OnInit, DoCheck, AfterContentInit,
 AfterContentChecked, AfterViewInit, AfterViewChecked, OnDestroy{
 ngDoCheck(): void {}
 ngOnDestroy(): void {}
 ngOnInit(): void {}
 ngAfterContentChecked(): void {}
 ngAfterContentInit(): void {}
 ngAfterViewChecked(): void {}
 ngAfterViewInit(): void {}
 ngOnChanges(changes: SimpleChanges): void {}
}
```

## Cycle de vie

# Cycle de vie des composants 2/3

- Création du composant
- Injection des dépendances
- Envoi des données d'entrée
- Initialisation des données d'entrée
- Création du template
- Exécution du composant
- Interaction avec l'utilisation
- Destruction du composant
- Nettoyage des ressources



## Cycle de vie

# Cycle de vie des composants 3/3

1. **Création du composant** : le composant est créé
2. **Injection des dépendances** : les dépendances du composant sont injectées
3. **Envoi des données d'entrée** : le composant parent envoie les données au composant enfant
4. **Initialisation des données d'entrée** : le composant enfant initialise ses données d'entrée avec les valeurs fournies par le composant parent
5. **Création du template** : le template du composant est compilé et le rendu est effectué
6. **Exécution du composant** : le composant est en cours d'exécution et interagit avec l'utilisateur
7. **Interactions avec l'utilisateur** : le composant réagit aux événements de l'utilisateur (clics..)
8. **Destruction du composant** : le composant est détruit et ses ressources sont libérées
9. **Nettoyage des ressources** : le composant effectue le nettoyage des ressources

## Cycle de vie

# Méthodes du cycle de vie 1/2

- **Le constructeur** ne fait pas parti du cycle de vie (utilisé pour l'injection)
- **ngOnInit** : appelée une seule fois après l'initialisation du composant
- **ngOnChanges** : appelée chaque fois que des modifications sont apportées aux propriétés de l'objet d'entrée du composant
- **ngDoCheck** : appelée à chaque itération de la boucle de vérification de l'état du composant
- **ngAfterContentInit** : appelée une seule fois après que le contenu soit initialisé

## Cycle de vie

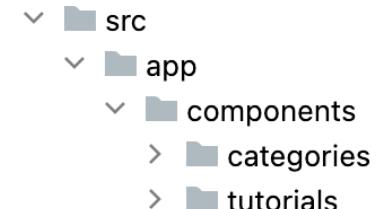
# Méthodes du cycle de vie 2/2

- **ngAfterContentChecked** : appelée à chaque itération de la boucle de vérification du contenu du composant
- **ngAfterViewInit** : appelée une seule fois après que la vue du composant est initialisée
- **ngAfterViewChecked** : appelée à chaque itération de la boucle de vérification de la vue du composant
- **ngOnDestroy** : appelée avant la destruction du composant

## Cycle de vie

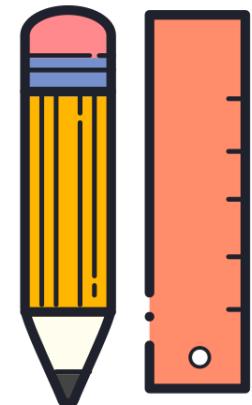
# Exercice : Angular

- Créez une nouvelle application Angular nommée **tutorials**
- Cette application est une application de gestion de tutoriels
- **Tutorials** sera le fil rouge de la formation. D'abord élémentaire, elle permettra l'assimilation de l'ensemble des concepts que doit maîtriser un développeur Angular junior.
- => Créez 2 composants nommés tutorials et categories dans un répertoire composants dans le répertoire source de l'application
- `ng generate component components/tutorials`
- `ng generate component components/categories`



# Les Templates

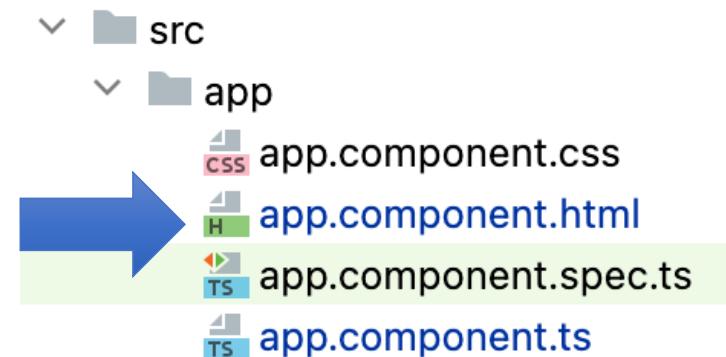
Développement Web



## Les Templates

# Introduction

- Dans chacun de vos composant, il y a une partie HTML
- Vous n'avez pas besoin de redéfinir `<html>`, `<body>`
  - Ces balises sont déjà incluses dans `index.html`
- Les `<script>` ne sont pas pris en charge dans les templates
  - Pour éviter le risque d'attaques par injection de script



# Les Templates

## Interpolation

- Utilisation dans le HTML d'une variable déclarée dans un **component**
  - Autrement dit, l'affichage de données TypeScript dans un gabarit

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

 person = {
 lastname: "Boris",
 firstname: "SAUVAGE »
 };
}
```

```
<p>Firstname : {{person.firstname}}</p>

<p>Lastname : {{person.lastname}}</p>
```

Firstname : Boris  
Lastname : SAUVAGE

# Liaison de propriété 1/2

- Mécanisme à sens unique permettant de définir la propriété d'un élément d'affichage
- Défini par les crochets [ **propertyName** ]
- Exemples :

```
<img
 [alt] = "bookName"
 [src] = "bookPictureUrl">
```

```
...
export class ExampleComponent {

 bookName = "Angular by Google";
 bookPictureUrl = "https://angular.io/assets/images/logos/angular/logo-nav@2x.png";
}
```

## Les Templates

### Liaison de propriété 2/2

- L'omission des [ ] permet à Angular de traiter le côté droit comme une string

```

```

- Si la valeur est une variable, alors vous devrez utiliser :

```
<img
 [alt]="bookName"
 [src]="bookPictureUrl">
```

- **bookName** et **bookPictureUrl** sont des variables présentes dans le Typescript

# Property Binding VS Interpolation

- Angular propose d'interpoler des données grâce aux `{} {}`
- Les expressions utilisées doivent rester simple
- Les méthodes doivent être **performantes**, sans **effet de bord et prédictibles**
- **Utiliser l'interpolation uniquement pour définir le contenu d'un élément HTML**

Exemple :



```

 Action
</button>
```

Action

# Les Directives

Développement Web



## Les directives

# Les directives

Permettent d'ajouter du comportement à des éléments HTML

- Les directives structurelles : modifient le DOM en ajoutant/modifiant un élément
  - `ngIf`, `ngFor`, `ngSwitch`, `ngTemplate`
- Les directives d'attributs : modifient l'apparence
  - `ngClass`, `ngStyle`, `ngModel`
- Les directives de contenu : incluent des éléments
  - `ngContent`, `ngContainer`
- La directive de chargements paresseux :
  - `@defer`, `@placeholder`, `@error`, `@loading()`
- Les directives personnalisées

## Les directives

# Données pour les directives

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

@Component({
 selector: 'app-root',
 templateUrl: './app.component.html',
 styleUrls: ['./app.component.css']
})
export class AppComponent {

 fruits: any[] = [
 {
 "name": "Mandarin",
 "price": 0.85,
 "origin": "SPAIN"
 },
 {
 "name": "Pear",
 "price": 1.75,
 "origin": "BELGIUM"
 },
 {
 "name": "Apple",
 "price": 2.56,
 "origin": "FRANCE"
 }
];
}
```

## Les directives

### \*ngFor

```
<!-- ngFor -->
<h2>All Fruits :</h2>
<ul *ngFor="let fruit of fruits">
 {{fruit.name}} - {{fruit.price}}€

```

#### All Fruits :

- Mandarin - 0.85€
- Pear - 1.75€
- Apple - 2.56€

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

```
fruits: any[] = [
 {
 "name": "Mandarin",
 "price": 0.85,
 "origin": "SPAIN"
 },
 {
 "name": "Pear",
 "price": 1.75,
 "origin": "BELGIUM"
 },
 {
 "name": "Apple",
 "price": 2.56,
 "origin": "FRANCE"
 }
];
```

## Les directives

### @for (> d'Angular 14)

```
<!-- ngFor -->
<h2>All Fruits :</h2>
<ul @for="let fruit of fruits">
 {{fruit.name}} - {{fruit.price}}€

```

<https://angular.dev/api/core/@for>

```

 @for (fruit of fruits; track fruit.id) {
 {{ fruit.name }}
 }

```

## Les directives

### \*ngIf

```
<!-- nglf -->
<h2>Cheapest fruits :</h2>
<ul *ngFor="let fruit of fruits">
 <li *ngIf="fruit.price < 1">{{fruit.name}} - {{fruit.price}}€

```

#### Cheapest fruits :

- Mandarin - 0.85€

<https://angular.dev/api/core/@if>

```
fruits: any[] = [
 {
 "name": "Mandarin",
 "price": 0.85,
 "origin": "SPAIN"
 },
 {
 "name": "Pear",
 "price": 1.75,
 "origin": "BELGIUM"
 },
 {
 "name": "Apple",
 "price": 2.56,
 "origin": "FRANCE"
 }
];
```

# Les directives

## [ngSwitch]

```
<!-- ngSwitch -->
<h2>Fruits (origin):</h2>
<table>
 <thead>
 <tr>
 <th>name</th>
 <th>price (€)</th>
 <th>origin</th>
 <th>flag</th>
 </tr>
 </thead>
 <tbody *ngFor="let fruit of fruits" [ngSwitch]="fruit.origin">
 <tr>
 <td>{{fruit.name}}</td>
 <td>{{fruit.price}}</td>
 <td>{{fruit.origin}}</td>
 <td *ngSwitchCase="''FRANCE''"></td>
 <td *ngSwitchCase="''SPAIN''"></td>
 <td *ngSwitchCase="''BELGIUM''"></td>
 <td *ngSwitchDefault>N/A</td>
 </tr>
 </tbody>
</table>
```

### Fruits (origin):

name	price (€)	origin	flag
Mandarin	0.85	SPAIN	
Pear	1.75	BELGIUM	
Apple	2.56	FRANCE	

<https://angular.dev/api/core/@switch>

## Les directives

### [ngClass]

```
<style>
 .blue { color: #3f51b5; }
 .underline { text-decoration: underline; }
</style>
```

```
<!-- ngClass -->
<h2>Fruits (expensive but local):</h2>
<ul *ngFor="let fruit of fruits">
 <li [ngClass]="{'blue': fruit.price > 2, 'underline': fruit.origin === 'FRANCE'}">{{fruit.name}}

```

#### Fruits (expensive but local):

- Mandarin
- Pear
- [Apple](#)

## Les directives

### [ngStyle]

```
<!-- ngStyle -->
<h2>Fruits with color :</h2>
<ul *ngFor="let fruit of fruits">
 <li [ngStyle]="{color:fruit.color}">{{fruit.name}}

```

#### Fruits with color :

- Mandarin
- Pear
- Apple

```
fruits: any[] = [
 {
 "name": "Mandarin",
 "price": 0.85,
 "origin": "SPAIN",
 "color": "orange"
 },
 {
 "name": "Pear",
 "price": 1.75,
 "origin": "BELGIUM",
 "color": "GreenYellow"
 },
 {
 "name": "Apple",
 "price": 2.56,
 "origin": "FRANCE",
 "color": "green"
 }
];
```

## Les directives

# Éléments utilisable avec les directive

Il existe des éléments souvent utilisables avec les directives

- **ng-template**
  - Permet de définir un modèle qui s'affichera à travers l'utilisable d'une condition
- **ng-container**
  - Permet de regrouper des éléments sans ajouter des balises inutiles dans le DOM
- **ngTemplateOutlet**
- **ng-content**
  - Permet de définir un contenu réutilisables s'adaptant à différents contextes

## Les directives

# ng-template

```
<div *ngIf="fruits else noFruits">
 <ul *ngFor="let fruit of fruits">
 {{fruit.name}}

</div>

<ng-template #noFruits>
 <div>Aucun fruit disponible.</div>
</ng-template>
```

```
fruits: any[] = [];
```

Fruits :  
Aucun fruit disponible.

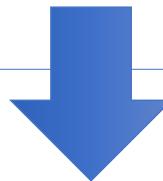
## Les directives

# ng-container

```
<!-- Exemple sans ngContainer -->

 <div *ngFor="let fruit of fruits">
 <li *ngIf="fruit.price < 1">{{fruit.name}} - {{fruit.price}}€
 </div>

```



```
<!-- Exemple avec ngContainer -->

 <ng-container *ngFor="let fruit of fruits">
 <li *ngIf="fruit.price < 1">{{fruit.name}} - {{fruit.price}}€
 </ng-container>

```

# Les directives ng-content

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

@Component({
 selector: 'app-my-component',
 template: `
 <h3>Projection à plusieurs emplacements</h3>

 <h4>Première projection:</h4>
 <ng-content></ng-content>

 <h4>Seconde projection:</h4>
 <ng-content select="[second]"></ng-content>
 `

})
export class MyComponent {}
```

```
<app-my-component>
 <p second>
 Je suis projeté dans la 2ème section !
 </p>
 <p>La première section est pour moi !</p>
</app-my-component>
```

## Projection à plusieurs emplacements

Première projection:

La première section est pour moi !

Seconde projection:

Je suis projeté dans la 2ème section !

## Les directives

# @defer : chargement paresseux 1/2

- Permet de charger de contenu de manière différée
  - Attente d'une interaction utilisateur pour charger les ressources nécessaires
  - Réduit le temps de chargement initial en optimisant les performances
- **@defer** : Le bloc charge son contenu seulement si certaines conditions sont remplies
- **@placeholder** : Contenu affiché en attendant le chargement
- **@error** : Contenu affiché en cas d'erreur de chargement
- **@loading** : Contenu affiché pendant une durée précise

## Les directives

# @defer : chargement paresseux 2/2

```
<h3>Interaction</h3>
```

```
@defer (on interaction) {
 <div>Cliqué</div>
}
@placeholder {
 <div>Placeholder (cliquez dessus !)</div>
}
```

```
<h3>Timer(5s)</h3>
```

```
@defer (on timer(5s)) {
 <div>Visible après 5s</div>
}
@placeholder {
 <div>Placeholder</div>
}
```

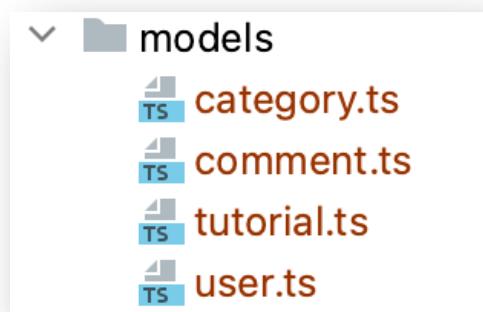
- Il existe de nombreux déclencheurs comme :
  - **on hover** : se déclenche au survol
  - **on idle** : se déclenche lorsque le navigateur est inactif
  - **prefetch on hover** : précharge le contenu au survol pour une interaction future
  - <https://angular.dev/guide/templates/defer#>

# Exercice 1/3

- À partir de l'application **Tutorials** et des 2 composants créés au début :
  - affichez le composant **categories** en page d'accueil
  - Utilisez son sélecteur pour le placer judicieusement dans le fichier **app-component.html**
- Créez un tableau nommé **categories** dans le fichier **categories.component.ts**
  - Il doit contenir une dizaine de catégories
  - (Java, Javascript, Typescript, PHP, C#, Angular, Ruby, Python, C++, Rust, VBA...)
- Dans le template **categories.component.html**, affichez la liste des catégories Utilisez une directive de structure pour afficher l'ensemble sous forme de liste

## Exercice 2/3

- Pour rendre l'ensemble plus agréable à regarder, utilisez Material Design que vous intégrerez à l'application grâce à son paquet npm :
  - <https://material.angular.io/guide/getting-started>
- Aussi, définissez des interfaces pour définir tous vos éléments



# Exercice 3/3 : Aperçu

Tutorials App

## Liste des Catégories

Java

Javascript

TypeScript

PHP

C#

Angular

Ruby

Python

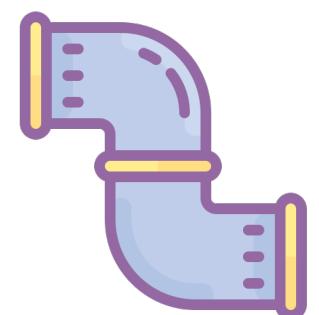
C++

Rust

VBA

# Les pipes

Développement Web



## Les pipes

# Introduction

- **Pipe** : petit bout de code ajouté à la suite d'une variable interpolée
  - Modifie l'affichage de la variable
- **CommonModule** : contient les pipes les plus utilisés
- Un pipe peut-être paramétré pour s'adapter parfaitement aux besoins du développeur

```
 {{ yourVariable | yourPipe }}
```

## Les pipes

# Liste des pipes 1/2

- Il existe de nombreux fournies par Angular :

DatePipe : Utilisé pour formater les dates

UpperCasePipe : Transforme le texte en majuscules

LowerCasePipe : Transforme le texte en minuscules

CurrencyPipe : Formate les nombres en devises

DecimalPipe : Formate les nombres en décimales

PercentPipe : Formate les nombres en pourcentages

SlicePipe : Extrait une portion d'un tableau ou d'une chaîne

## Les pipes

# Liste des pipes 2/2

AsyncPipe : Gère les observables et les promesses

JsonPipe : Affiche des données JSON formatées

KeyValuePipe : Utilisé pour itérer sur les propriétés d'un objet

I18nSelectPipe : Utilisé pour effectuer une sélection basée sur des clés de traduction

I18nPluralPipe : Utilisé pour effectuer une sélection de texte basée sur le genre, le nombre, etc

TitleCasePipe : Transforme le texte en une forme de titre

AsyncTimePipe : Transforme les timestamps en temps écoulé

Les pipes personnalisés

## Les pipes

# Pipe des chaînes de caractères

- **3 Pipes** : permettent de modifier l'affichage d'une variable String
  - **LowerCasePipe**, pour transformer toutes les lettres en minuscules
  - **UpperCasePipe**, pour transformer toutes les lettres en majuscules
  - **TitleCasePipe**, pour adopter le style « **Titre** » qui consiste à mettre en majuscule la première lettre de chaque mot d'une phrase

```
<div class="container text-center">
 <h2>Les pipes</h2>
 <p>{{citation}}</p>

 <p>{{citation | lowercase}}</p>
 <p>{{citation | uppercase}}</p>
 <p>{{citation | titlecase}}</p>
</div>
```

## Les pipes

Un problème sans solution est un problème mal posé

un problème sans solution est un problème mal posé

UN PROBLÈME SANS SOLUTION EST UN PROBLÈME MAL POSÉ

Un Problème Sans Solution Est Un Problème Mal Posé

# Pipe des nombres

- **3 Pipes** : permettent de mettre en forme des nombres
  - **DecimalPipe**
  - **PercentPipe**
  - **CurrencyPipe**
- **Par défaut**, Angular transforme les chiffres en se calant sur le modèle américain
  - à savoir un point entre la partie entière et la partie décimale d'un nombre
  - une date au format année-mois-jour
- Il utilise une variable **LOCALE\_ID** qu'il va falloir modifier

## Les pipes

# Internationalisation des Pipes (< Angular 15)

```
import {BrowserModule} from '@angular/platform-browser';
import {LOCALE_ID, NgModule} from '@angular/core';
import {AppRoutingModule} from './app-routing.module';
import {AppComponent} from './app.component';
import {registerLocaleData} from '@angular/common';
import localeFR from '@angular/common/locales/fr';

registerLocaleData(localeFR);

@NgModule({
 declarations: [
 AppComponent
],
 imports: [
 BrowserModule,
 AppRoutingModule
],
 providers: [{provide: LOCALE_ID, useValue: 'fr'}],
 bootstrap: [AppComponent]
})
export class AppModule {
```



## Les pipes

# Internationalisation des Pipes (> Angular 15)

- Dans **main.ts**

```
import {registerLocaleData} from '@angular/common';
import localeFr from '@angular/common/locales/fr';
import {bootstrapApplication} from '@angular/platform-browser';
import {AppComponent} from './app/app.component';
import {appConfig} from "./app/app.config";

registerLocaleData(localeFr);

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



- Dans **app.config** :

```
import {ApplicationConfig, LOCALE_ID, provideZoneChangeDetection} from '@angular/core';
import {provideRouter} from '@angular/router';

import {routes} from './app.routes';
import {provideAnimationsAsync} from '@angular/platform-browser/animations/async';

export const appConfig: ApplicationConfig = {
 providers: [
 provideZoneChangeDetection({eventCoalescing: true}), provideRouter(routes), provideAnimationsAsync(),
 {provide: LOCALE_ID, useValue: 'fr-FR'}
]
};
```



## Les pipes

### Pipe : DecimalPipe

- Ce tube permet d'adapter le nombre aux préférences de l'utilisateur pour effectuer un arrondi par exemple  

{{variable | number : miseEnForme }}
- La mise en forme contient 3 critères :
  - le nombre minimum de chiffres avant la virgule
  - le nombre minimum de chiffres après la virgule
  - le nombre maximum de chiffres après la virgule
- **1.2-4** signifie, par exemple, que le nombre possède au **minimum 1 chiffre avant la virgule, 2 au minimum après la virgule et 4 au maximum**
- **Si le nombre ne contient qu'un seul chiffre avant la virgule** et que les paramètres indiquent un chiffre supérieur, Angular rajoute des 0
- **Même chose pour les chiffres après la virgule.** La seule condition vérifiée à la compilation est, qu'après la virgule, le chiffre minimum est bien inférieur au chiffre maximum

## Les pipes

### Pipe : DecimalPipe

```
myNumber = 3.14159265358979323846264338379;
```

```
<div class="container text-center">
 <h2>DecimalPipe</h2>
 <p>pi : {{myNumber}}</p>
 <p>{{myNumber | number: '2.3-4' }}</p>
</div>
```

DecimalPipe  
pi : 3.141592653589793  
03,1416

## Les pipes

### Pipe : PercentPipe

- Ce pipe permet la mise en forme d'un chiffre en pourcentage

```
{variable | percent : miseEnForme }}}
```

- Encore une fois, la mise en forme contient 3 critères :
  - le nombre minimum de chiffres avant la virgule
  - le nombre minimum de chiffres après la virgule
  - le nombre maximum de chiffres après la virgule

```
<div class="container text-center">
 <h2>PercentPipe</h2>
 <p>Part de marché : {{myNumber}}</p>
 <p>{{myNumber | percent: '2.3-4'}}</p>
</div>
```

```
myNumber = 0.123456789;
```

### PercentPipe

Part de marché : 0.123456789

12,3457 %

- Le pipe se charge de l'affichage du pourcentage en respectant les spécifications locales grâce à la variable LOCALE\_ID. Il ajoute ensuite, en suffixe, le caractère %.

## Les pipes

# Pipe : CurrencyPipe 1/2

- C'est ce pipe qui permet l'affichage monétaire de chiffres. Il ne fait cependant aucune conversion de devises. Il affiche juste le nombre suivi ou précédé du symbole monétaire en respectant un format concernant le nombre de chiffres avant et après la virgule.

```
{variable | currency : codeMonetaire : affichage : miseEnForme}
```

- Le code monétaire, défini par la norme ISO4217 représente la monnaie utilisée dans un pays. Par exemple, EUR pour l'euro, XPF pour le franc pacifique et l'USD pour le dollar américain.
- L'affichage permet de modifier la façon dont le symbole va être rendu à l'écran. code montre le code monétaire précédemment paramétré, symbol affiche le symbole. On peut inscrire une chaîne de caractères afin d'afficher des mots à la place du symbole monétaire.
- La mise en forme contient les mêmes trois paramètres que précédemment, à savoir le nombre de chiffres minimum avant et après la virgule et le nombre de chiffres maximum après la virgule.

## Les pipes

# Pipe : CurrencyPipe 2/2

```
<div class="container text-center">
 <h2>CurrencyPipe</h2>
 <p>Tesla M3 : {{teslaM3}}</p>
 <p>48600,00€ => {{teslaM3 | currency: 'EUR' : 'symbol'}}</p>
 <p>48600,00 EUR => {{teslaM3 | currency: 'EUR' : 'code'}}</p>
 </p>
 <p>48600,00 euros, et pas très chère. => {{teslaM3 | currency:'EUR' : 'euros, et pas très chère.'}}</p>
 <p>48600 € => {{teslaM3 | currency: 'EUR' : 'symbol': '1.0-0'}}</p>
</div>
```

## CurrencyPipe

Tesla M3 : 48600

48600,00€ => 48 600,00 €

48600,00 EUR => 48 600,00 EUR

48600,00 euros, et pas très chère. => 48 600,00 euros, et pas très chère.

48600 € => 48 600 €

## Pipe : Date

- Le formatage des dates est possible avec Angular
- JavaScript utilise la même méthode que les systèmes UNIX pour définir le temps, mais en étant beaucoup plus précis
- UNIX définit une date et une heure comme le nombre de secondes écoulées depuis le 01 janvier 1970 à minuit, JavaScript compte le nombre de millisecondes

```
{variable | date : format : timezone }
```

- Le Pipe Date utilise le LOCALE\_ID
- Les formats sont disponible sur <https://angular.io/api/common/DatePipe>

## Les pipes

# Pipe : Date

```
<div class="container text-center">
 <h2>Le pipe Date</h2>
 <p>Nous sommes le {{today}}</p>
 <p>Nous sommes le {{today | date}}</p>
 <p>Nous sommes le {{today | date : 'shortDate'}}. Il est {{today | date : 'shortTime'}}.</p>
</div>
```

## Le pipe Date

Nous sommes le Sat Jan 30 2055 13:30:00 GMT+0100 (heure normale d'Europe centrale)

Nous sommes le 30 janv. 2055

Nous sommes le 30/01/2055. Il est 13:30 .

## Les pipes

# Pipe : objet via les API

- Une API permet de récupérer des données d'autres environnements
  - Le site api.gouv.fr récense l'ensemble des API de l'État. Tout ceci fait partie d'un énorme projet appelé « big data » permettant une meilleure transparence des données publiques
- **JsonPipe** : permet d'afficher un objet en utilisant le format JSON
  - Angular n'est pas capable d'afficher un objet complexe
- **AsyncPipe** : permet d'utiliser les promesses/observables (Programmation réactive)
  - Il affiche la dernière valeur reçue de la part de ces deux éléments
- une promesse est un objet dont la valeur n'est pas encore disponible
  - Angular n'affichera pas [object Promise] lorsque la promesse sera initialement vide, mais une chaîne de caractères vide

## Les pipes

### Pipe : Json

- L'affichage d'un objet dans l'html

```
<h2>Pipe Json</h2>

<p>{{fruits}}</p>
```



**Pipe json**  
[object Object],[object Object],[object Object]

```
fruits: any[] = [
 {
 "name": "Mandarin",
 "price": 0.85,
 "origin": "SPAIN",
 "color": "orange"
 }];
```

- Avec le pipe json, il est possible de voir l'objet complet :
  - Équivalent à JSON.stringify()

```
<h2>Pipe Json</h2>

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



**Pipe json**  
[ { "name": "Mandarin", "price": 0.85, "origin": "SPAIN", "color": "orange" } ]

## Les pipes

# Pipe : AsyncPipe (Promesse)

## Le AsyncPipe

After(5000)

## Le AsyncPipe

C'est bon !!!

```
<div class="container text-center">
 <h2>Le AsyncPipe</h2>
 <div>{{unePromesse | async}}</div>
</div>
```

```
// La promesse résout après 5000 millisecondes
// En attendant, {{unePromesse}} affiche une chaîne
// de caractères vide
unePromesse = new Promise((resolve) => {
 setTimeout(() => {
 resolve('C\'est bon !!!');
 }, 5000);
});
```

## Les pipes

# Pipe : AsyncPipe (Observable)

## Le AsyncPipe

After(5000)

## Le AsyncPipe

C'est remarquable !!!

```
<div class="container text-center">
 <h2>Le AsyncPipe</h2>
 <div>{{unObservable | async}}</div>
</div>
```

```
// L'observable résout après 5000 millisecondes
// En attendant, {{unObservable}} affiche une chaîne
// de caractères vide
unObservable = new Observable(observer => {
 setTimeout(() => {
 observer.next('C\'est remarquable !!!');
 }, 5000);
});
```

# Exercice 1/5 : Sujet

- À partir de l'application Tutorials créée précédemment, ajoutez un tableau contenant des tutoriels au composant tutorials
- Un tutoriel est composé d'un id, un titre, une description, un contenu, un auteur, une catégorie, une date de création, une liste de commentaires :
  - Un Auteur est composé d'un id, d'un nom, d'un prénom, d'un email
  - Une Catégorie représente une des catégories déclarée précédemment
- Les Commentaires peuvent être afficher grâce à un bouton
- Avec l'aide du framework CSS, affichez une liste des tutoriels dans le template du composant tutoriels :
  - Affichez la date sous un format lisible
  - Afficher le nom de l'auteur en majuscule

# Exercice 2/5 : Aperçu

Tutorials App

**Initiation au langage Java**  
Lorem ipsum dolor sit amet, consectetur adipisicing elit  
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Alias animi aperiam aspernatur cupiditate deserunt digniss...  
[Jane WAAAA](#) [j.waaaa@tuto.fr](#)

En ligne depuis le mercredi 2 décembre 2026 [Details](#) [Partager](#)

**Commentaires** 2

Super tuto ! Bravo Jane  
Très intéressant, merci pour le partage

**Initiation au macro VBA**  
Lorem ipsum dolor sit amet, consectetur adipisicing elit  
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Alias animi aperiam aspernatur cupiditate deserunt digniss...  
[Boris SAU](#) [b.sau@tuto.fr](#)

En ligne depuis le mardi 16 septembre 2025 [Details](#) [Partager](#)

**Commentaires** 1

Excel est vraiment génial avec VBA

**Afficher une liste avec Material Angular**  
Lorem ipsum dolor sit amet, consectetur adipisicing elit  
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Alias animi aperiam aspernatur cupiditate deserunt digniss...  
[John DOE](#) [j.doe@tuto.fr](#)

En ligne depuis le mardi 30 janvier 2024 [Details](#) [Partager](#)

**Commentaires**

# Exercice 3/5 : Pipe personnalisé

- Afin de typer les Tutoriels utilisés dans le tableau, utilisez une interface
  - Faire un pipe personnalisé pour afficher les tutoriels du plus récent au moins récent

```
@Pipe({
 name: 'sortTutorialByDate', standalone: true
})
export class SortTutorialByDatePipe implements PipeTransform {

 transform(tutorials: Tutorial[] | null, order: string): Tutorial[] {

 if (!tutorials || tutorials.length == 0) return [];

 if (order === 'DESC') {
 return tutorials.sort((a, b) => return new Date(b.createdAt).getTime() - new Date(a.createdAt).getTime());
 }
 return tutorials.sort((a, b) => return new Date(a.createdAt).getTime() - new Date(b.createdAt).getTime());
 }
}
```

ng generate pipe pipes/sortTutorialByDate

\*ngFor="let tuto of tutorials | sortTutorialByDate:'DESC'"

# Exercice 4/5 : Pipe personnalisé

- Afin d'utiliser ce pipe, vous pouvez mettre en place une sélection de tri :

The screenshot shows a web page with a blue header bar. Below it, there's a search bar with placeholder text and a button labeled "Rechercher". On the left, a sidebar titled "Tutoriels :" lists categories like "Tutoriels", "Cours", "Vidéos", and "Livres". To the right, there's a "Tri :" dropdown menu set to "Plus récents". The main content area displays a card for a tutorial titled "Initiation au langage Java". The card includes a snippet of text, author information ("Jane WAAAAA" and email "j.waaaa@tuto.fr"), a timestamp ("En ligne depuis le mercredi 2 décembre 2026"), and two buttons: "Details" and "Partager". At the bottom of the card, there's a "Commentaires" button with a red badge indicating 2 comments.

<https://material.angular.io/components/select/examples>

# Exercice 5/5 : Directive

- Mettre en place une directive qui applique un style au survole d'un tutoriel

**Initiation au langage Java**  
Lorem ipsum dolor sit amet, consectetur adipisicing elit  
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Alias animi aperiam aspernatur cupiditate deserunt digniss...  
Jane WAAAAA j.waaaa@tuto.fr

En ligne depuis le mercredi 2 décembre 2026

**Details** **Partager**

**Commentaires** 2

**Initiation au macro VBA**  
Lorem ipsum dolor sit amet, consectetur adipisicing elit  
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Alias animi aperiam aspernatur cupiditate deserunt digniss...  
Boris SAU b.sau@tuto.fr

En ligne depuis le mardi 16 septembre 2025

**Details** **Partager**

**Commentaires** 1

<https://angular.io/guide/attribute-directives>

# Les décorateurs

Développement Web



# Les décorateurs

## Introduction

- Angular fournit de nombreux décorateurs pour annoter des classes

**@Component**

**@Directive**

**@Injectable**

**@Module**

**@Pipe**

**@Input**

**@Output**

**@HostListener**

**@ViewChild**

**@ViewChildren**

**@ContentChild**

**@ContentChildren**

**@HostBinding**

**@Attribute**

**@NgModule**

**@Inject**

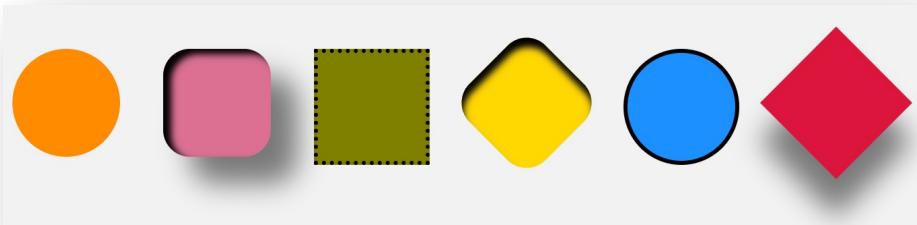
...

## Les décorateurs

### @Input

- Utilisé pour déclarer une propriété d'entrée dans un composant
  - permet de passer des données du composant parent au composant enfant

```
<app-example-input
[circleShape]="true"
[withBorderSolid]="true">
</app-example-input>
```



```
@Component({
 selector: 'app-example-input',
 templateUrl: './example-input.component.html',
 styleUrls: ['./example-input.component.css']
})
export class ExampleInputComponent {

 @Input() circleShape: boolean = false;

 @Input() withBorderSolid: boolean = false;
}
```

<https://stackblitz.com/edit/angular-exemple-input-output>

## Les décorateurs

### @Output 1/4

- Utilisé pour déclarer un événement personnalisé dans un composant
  - Cet événement peut être émis vers le composant parent pour notifier des changements

```
import {Component, EventEmitter, OnInit, Output} from '@angular/core';
import {Observable} from "rxjs";

@Component({
 selector: 'app-example-output',
 template: '<button (click)="onButtonClick()">Cliquez-moi !</button>',
 styleUrls: ['./example-output.component.css']
})
export class ExampleOutputComponent {
 @Output() buttonClicked: EventEmitter<void> = new EventEmitter<void>();

 onButtonClick() {
 this.buttonClicked.emit();
 }
}
```

## Les décorateurs

# @Output 2/4

- Le composant parent :
  - Définit un évènement (**buttonClicked**) correspondant à l'évènement du composant enfant
  - L'évènement **@Output** permet de notifier le composant parent

```
@Component({
 selector: 'app-parent-component',
 template: `
 <app-example-output (buttonClicked)="handleButtonClick()"></app-example-output>
 <p *ngIf="buttonClicked">Le bouton a été cliqué !</p>
 `,
})
export class ParentComponent {
 buttonClicked = false;

 handleButtonClick() {
 this.buttonClicked = true;
 }
}
```

## Les décorateurs

# @Output 3/4

- Il est également possible d'envoyer une information dans l'événement

```
export class ChildComponent {
 @Output() buttonClicked: EventEmitter<Tutorial> = new EventEmitter<Tutorial>();

 onButtonClick() {
 this.buttonClicked.emit(selectedTutorial);
 }
}
```

```
export class ParentComponent {

 handleButtonClick(tutorialEmitted : Tutorial) {
 }
}
```

## Les décorateurs

# @Output 4/4

- Il est également possible d'envoyer une information dans l'événement
- Composant enfant :**

```
@Output()
unEvenement = new EventEmitter();

unClicSurLeBouton() {
 this.unEvenement.emit('Bonjour à tous');
}
```

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

```
<button (click)="unClicSurLeBouton()">Envoyer au parent</button>
```

- Component parent :**

```
messageDuFils: string;

receptionMessage(msgRecu: string) {
 this.messageDuFils = msgRecu;
}
```

```
<p>Message du fils : {{messageDuFils}}</p>

<app-component-child (unEvenement)="receptionMessage($event)">
</app-component-child>
```

# Exercice 1/2

Mettre en place une architecture hiérarchique dans **le composant tutorials** :

- Créez un nouveau composant **comments**
  - possède un paramètre en entrée : un objet tutorial
- Le composant **tutorials** est le **composant parent** des composants **comments**
  - fournit à chaque enfant un objet commentaire

# Exercice 2/2

Composant comments

Composant tutorials

The image displays three cards, each representing a different tutorial or article. The first card is titled 'Initiation au langage Java'. It shows a comment from 'Jane WAAAAA' at 'j.waaaa@tuto.fr' posted 'En ligne depuis le mercredi 2 décembre 2026'. The second card is titled 'Initiation au macro VBA'. It shows a comment from 'Boris SAU' at 'b.sau@tuto.fr' posted 'En ligne depuis le mardi 16 septembre 2025'. The third card is titled 'Afficher une liste avec Material Angular'.

**Initiation au langage Java**  
Lorem ipsum dolor sit amet, consectetur adipisicing elit  
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Alias animi aperiam aspernatur cupiditate deserunt digniss...  
Jane WAAAAA j.waaaa@tuto.fr  
En ligne depuis le mercredi 2 décembre 2026 [Details](#) [Partager](#)

**Commentaires**  
2  
Super tuto ! Bravo Jane  
Très intéressant, merci pour le partage

**Initiation au macro VBA**  
Lorem ipsum dolor sit amet, consectetur adipisicing elit  
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Alias animi aperiam aspernatur cupiditate deserunt digniss...  
Boris SAU b.sau@tuto.fr  
En ligne depuis le mardi 16 septembre 2025 [Details](#) [Partager](#)

**Commentaires**  
1  
Afficher une liste avec Material Angular  
Lorem ipsum dolor sit amet, consectetur adipisicing elit

# Autres solutions

- Il existe d'autres méthodes pour communiquer entre les composants Angular :
  - **@ViewChild et @ViewChildren**
    - <https://stackblitz.com/edit/exemple-manipulation-avec-viewchild>
    - <https://stackblitz.com/edit/exemple-manipulation-avec-viewchildren>
  - **@ContentChild et @ContentChildren**
  - **Subject et Services**
    - Les sélecteurs du premier composant et celui du deuxième se trouvent au même niveau
      - <https://stackblitz.com/edit/angular-10-communication-avec-subject-et-service>

## Les décorateurs

### @ViewChild

- <https://angular.fr/components/view-child.html>

```
@Component({
 selector: 'app-root',
 standalone: true,
 imports: [CommonModule],
 template: `
 <p>Bonjour </p>
 <button (click)="sayHello()">Dire Bonjour</button>
 `,
})
export class DemoComponent {
 @ViewChild('firstNameSpan') firstNameEl!: ElementRef;

 sayHello() {
 this.firstNameEl.nativeElement.innerText = 'Boris';
 }
}
```

## Les décorateurs

# @ViewChildren 1/2

- Permet de récupérer la référence d'un ou plusieurs éléments enfants de votre vue

```
@Component({
 selector: 'app-root',
 standalone: true,
 imports: [CommonModule],
 template: `
 <div #child1></div>
 <div #child2></div>
 `,
})
export class AppComponent implements AfterViewInit {
 @ViewChildren('child1, child2') children!: QueryList<ElementRef>;
 ngAfterViewInit() {
 console.log(this.children);
 }
}
```



## Les décorateurs

# @ViewChildren 2/2

- **QueryList** représente une liste d'objets pouvant être observés
  - utilisée avec les décorateurs de requête tels que `@ViewChildren` et `@ContentChildren`
- possède plusieurs méthodes :
- `.forEach()` pour itérer sur chaque élément de la liste
- `.changes` pour s'abonner aux changements de la liste
- `.length` pour obtenir le nombre d'éléments dans la liste

```
export class AppComponent implements AfterViewInit {
 @ViewChildren('child1, child2') children!: QueryList<ElementRef>

 ngAfterViewInit() {
 this.children.forEach((child) => {
 console.log(child.nativeElement.textContent);
 });
 }
}
```

## Les décorateurs

# @ContentChild & @ContentChildren

- Permet de récupérer la référence d'un ou de plusieurs enfants dans contenu d'un composant
- Vous pouvez l'utiliser pour accéder à un élément HTML ou à une directive présente dans le contenu (inclus) d'un composant

# Les liaisons (binding)

Développement Web



## Les liaisons

# Introduction 1/2

- Une liaison crée une connexion en direct :
  - entre une partie de l'interface utilisateur créée à partir d'un modèle (un élément, une directive ou un composant DOM)
  - le modèle (l'instance de composant à laquelle appartient le modèle)
- Pratique pour :
  - synchroniser la vue avec le modèle
  - notifier le modèle lorsqu'un événement a lieu dans la vue
  - notifier le modèle lorsqu'une action utilisateur a lieu dans la vue

L'algorithme de détection des changements d'Angular est chargé de maintenir la synchronisation de la vue et du modèle

## Les liaisons

# Introduction 2/2

Voici des exemples de liaison :

- interpolations de texte `{{value}}` (Vu précédemment)
- liaison de propriété `[value]` (Vu précédemment)
- liaison d'événement `(event)` (Vu précédemment)
- liaison bidirectionnelle `[(value)]` (Vu dans le mat-select)

Les liaisons comportent toujours 2 parties :

- une cible qui recevra la valeur liée
- une expression de modèle qui produit une valeur à partir du modèle

## Les liaisons

# Liaison bidirectionnelle

- Donne aux composants de votre application un moyen de partager des données
- Combine la liaison de propriété [] avec la liaison d'événement ()
- La syntaxe est une combinaison de crochets et de parenthèses [ () ]

## Les liaisons

# Liaison bidirectionnelle

- Exemple :

```
<div class="sort-header">
 <h2>Tutoriels :</h2>

 <mat-form-field>
 <mat-label>Tri :</mat-label>
 <mat-select [(value)]="sortByDate">
 <mat-option value="DESC">Plus récents</mat-option>
 <mat-option value="ASC">Plus anciens</mat-option>
 </mat-select>
 </mat-form-field>

</div>
```

```
@Component({
 selector: 'app-tutorials',
 standalone: true,
 templateUrl: './tutorials.component.html',
 styleUrls: ['./tutorials.component.css'
})
export class TutorialsComponent implements OnInit {

 sortByDate: string = "DESC";

}
```

## Les liaisons

# Liaison bidirectionnelle et @Output

- Exemple :

```
@Component({
 selector: 'app-count',
 template: `
 <p>Composant Enfant : {{n}}</p>
 <button (click)="up()">Incrémenter (depuis Enfant)</button>
 `
})
export class CountComponent {

 @Input() n: number = 0;
 @Output() nChange: EventEmitter<number> = new EventEmitter();

 up() {
 this.n++;
 this.nChange.emit(this.n);
 }
}
```

```
@Component({
 selector: 'app-root',
 template: `
 <div>
 <p>Composant parent : {{val}}</p>
 <button (click)="up()">Incrémenter (depuis Parent)</button>
 <app-count [(nChange)]="val"></app-count>
 </div>
 `
})
export class AppComponent {
 val: number = 0;

 up() {
 this.val++;
 }
}
```

Composant parent : 3

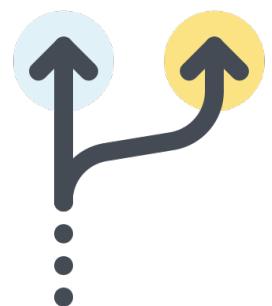
Incrémenter (depuis Parent)

Composant Enfant : 3

Incrémenter (depuis Enfant)

# Routeur : les routes

Développement Web



# Introduction

- Depuis 1993, les applications web ont bien changé
  - Nombre important de données à afficher
  - Nombre croissant d'utilisateurs habituées à une réaction instantanée au clic
  - Chaque page était affichée entièrement à chaque rechargement du navigateur
    - Surcharge réseau et mauvaises performances
- La **Single Page Application (SPA)** : concept d'application web monopage
  - Permet d'augmenter l'expérience utilisateur
  - Application composée d'une seule page qui charge les éléments à la demande
  - Accessible sur tablette, smartphone, ordinateurs, avec une consommation réseau raisonnable
- Le routage Angular va permettre la navigation au sein de la SPA

# Angular CLI et le routage

- Lors de création d'un projet, Angular implémente un système de routeur

- Présence d'une balise HTML dans le fichier index.html

```
<base href="/">
```

- La création d'un fichier **app-routes.ts** qui contiendra nos routes

```
import { Routes } from '@angular/router';
export const routes: Routes = [];
```

## Les routerLink

- Avec une SPA, l'ensemble des liens hypertextes est à modifier
- Utilisation de la directive Angular **routerLink**
  - Remplace le comportement HTML classique d'une balise `<a href="#">Lien</a>`
- Visuellement, le résultat est le même
  - la directive modifie le DOM
  - Avec href, celà recharge la page`<a routerLink="/url">Lien SPA</a>`
- Au clic sur le lien, Angular parcourt le tableau à la recherche d'une occurrence de routes et effectue la navigation vers le composant chargé du rendu

## Route active

- Il est possible de déterminer la route active avec la balise

```
Lien A
Lien B
```

- Et avec un style par d'exemple ici...

```
.active {
 border: solid;
}
```



Routeur : les routes

## Le fichier app-routes.ts

- Classe TypeScript exportée contenant un tableau de routes nommé routes

```
export const routes: Routes = [];
```

- Une route est un objet JavaScript qui contient une URL et un composant

```
{path: 'url', component: MyComponent}
```

```
const routes: Routes = [
 {path: 'linkA', component: MyAComponent},
 {path: 'linkB', component: MyBComponent}
];
```



localhost:4200/linkA

localhost:4200/linkB

## Fonctionnement

- Angular traite les routes de la première à la dernière jusqu'à ce qu'il en rencontre une qui corresponde. Dans ce cas, il n'évalue pas les suivantes
- Avec une SPA, il faut indiquer à Angular à quel endroit faire le rendu du composant

```
<router-outlet></router-outlet>
```

- Cette balise indique l'endroit où Angular va mettre le composant routé

```
Lien HTML
Lien A
Lien B
<router-outlet></router-outlet>
```

## Routes et route Joker

- La navigation fonctionne lorsque l'utilisateur navigue sur les routes `/`, `/linkA` et `/linkB`
- **Toutes les autres urls non gérées** provoquent une erreur (**dans la console**)
- Il est possible de mettre en place une route Joker (Qui accepte toutes les routes non définies)

```
const routes: Routes = [
 {path : 'linkA', component : MyAComponent},
 {path : 'linkB', component : MyBComponent},
 {path: '**', component: NotFoundComponent}
];
```

## Routes paramétrées

- Il est possible de passer des paramètres dans l'URL pour paramétrer la page affiché
- Par exemple, on aimerais définir **/technos** (affichera les technos), et :
  - **/technos/1** : affichera JavaScript
  - **/technos/2** : affichera Python
  - **/technos/3** : affichera Go
- Nous devons donc créer un seul composant et une route paramétrée
- Il est possible de passer des paramètres dans l'URL pour paramétrer la page affiché

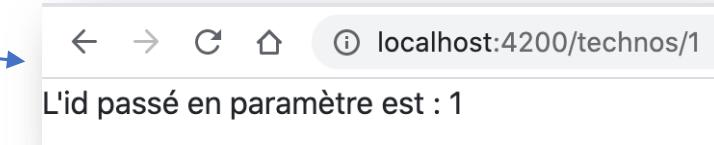
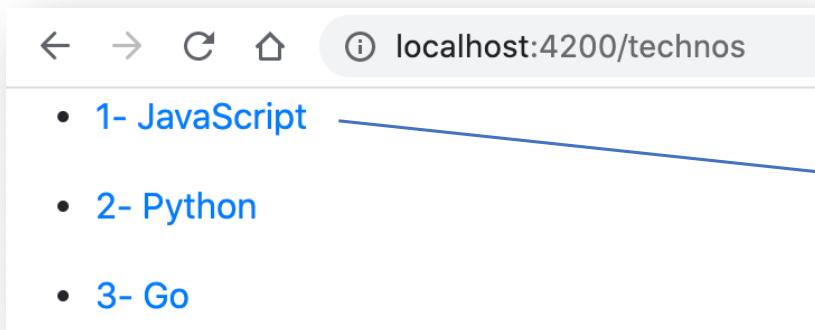
```
const routes: Routes = [
 {path: 'technos', component: TechnosComponent},
 {path: 'technos/:idParam', component: TechnoDetailsComponent},
 {path: '**', component: JokerComponent},
];
```

## Routeur : les routes

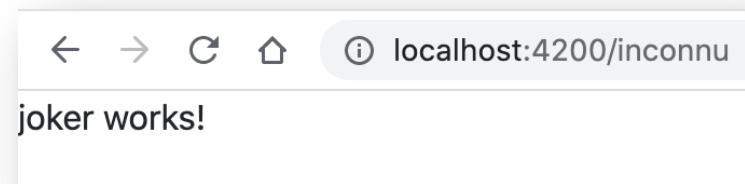
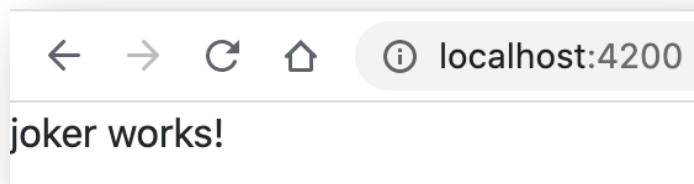
# Objectifs souhaités

- Avoir trois composants
- Avoir un routage fonctionnel

```
ng g c composants/joker
ng g c composants/technos
ng g c composants/technoDetails
```



- Gérer les routes inconnues



## Routeur : les routes

# TechnosComponent

- Activer le routage sur la SPA avec la balise suivante :

 app.component.html

```
<router-outlet></router-outlet>
```

- TechnosComponent :

```
<div>
 <ul *ngFor="let techno of technologies">
 {{techno.id}} - {{techno.nom}}

</div>
```

- 1 - JavaScript
- 2 - Python
- 3 - Go

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

@Component({
 selector: 'app-example-component',
 templateUrl: './techno.component.html',
 styleUrls: ['./techno.component.css']
})
export class TechnosComponent {

 technologies = [
 {
 id: 1, nom: 'JavaScript', frameworks: ['Angular', 'React', 'Vue']
 },
 {
 id: 2, nom: 'Python', frameworks: ['Django', 'Flask']
 },
 {
 id: 3, nom: 'Go', frameworks: ['Go']
 }
];
}
```

## Routage vers TechnoDetailsComponent

- Configuration de la liste avec des liens paramétrés vers **TechnoDetailsComponent** :

Avant

```
<div>
 <ul *ngFor="let techno of technologies">
 {{techno.id}} - {{techno.nom}}

</div>
```

- 1 - JavaScript
- 2 - Python
- 3 - Go

Après

```
<div>
 <ul *ngFor="let techno of technologies">
 <a [routerLink]="'/technos', techno.id">{{techno.id}}- {{techno.nom}}

</div>
```

- 1- JavaScript
- 2- Python
- 3- Go

# TechnoDetailsComponent

- Récupération de l'ID au sein du composant **TechnoDetailsComponent**

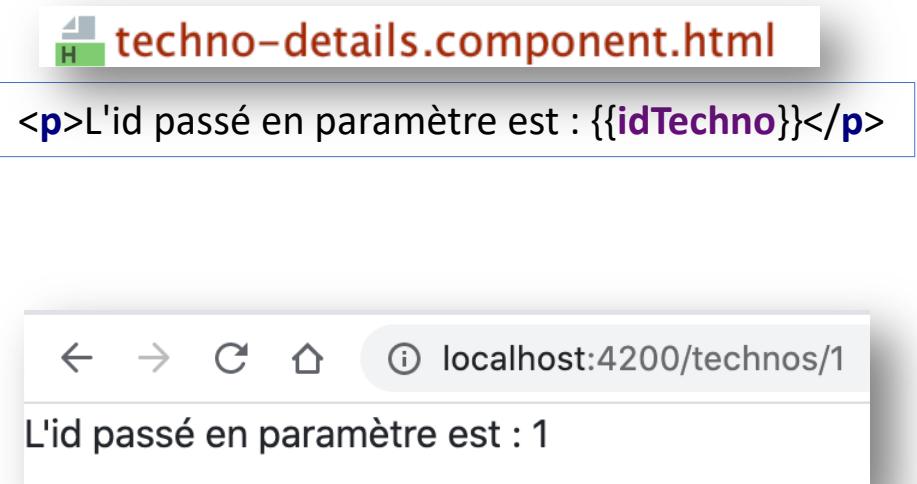
```
@Component({
 selector: 'app-technodetails',
 templateUrl: './techno-details.component.html',
 styleUrls: ['./techno-details.component.css']
})
export class TechnoDetailsComponent implements OnInit {

 // Attribut de la classe
 idTechno: string;

 // Injection de la dépendance dans le constructeur
 constructor(private routeActive: ActivatedRoute) {}

 ngOnInit() {
 // Récupération du paramètre
 const id = this.routeActive.snapshot.paramMap.get('id');
 // Affectation à l'attribut de la classe
 this.idTechno = id;
 }
}
```

 **techno-details.component.ts**



## Routeur : les routes

# Séparation vue/contrôleur

- Utiliser le routeur pour faire une redirection programmatique

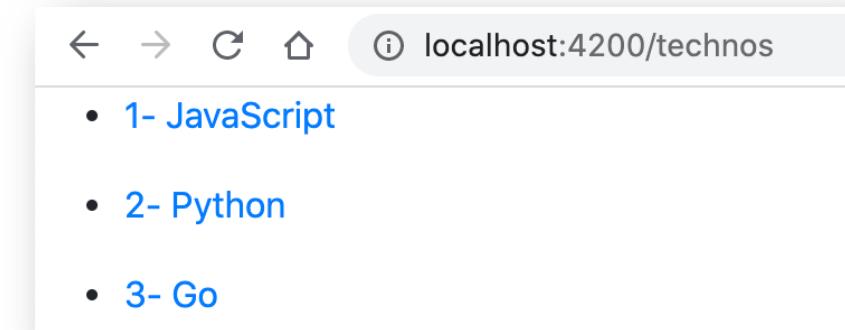
```
@Component({
 selector: 'app-techno-component',
 templateUrl: './techno.component.html',
 styleUrls: ['./techno.component.css']
})
export class TechnoComponent {

 constructor(private routeur: Router) {}

 clicSurUneTechno(uneTechno) {
 this.routeur.navigate(['/technos', uneTechno.id]);
 }
}
```

```
<div>
 <ul *ngFor="let techno of technologies">
 <a (click)="clicSurUneTechno(techno)">{{techno.id}} - {{techno.nom}}

</div>
```



- Avant :

```
<div>
 <ul *ngFor="let techno of technologies">
 <a [routerLink]="['/technos', techno.id]">{{techno.id}}- {{techno.nom}}

</div>
```

# Hiérarchie dans les routes

- Il existe de nombreuses applications pour faire du CRUD, avec de nombreuses routes
  - **/collection** : pour voir l'ensemble de la collection
  - **/collection/id** : pour lire l'élément avec l'identifiant id
  - **/collection/add/id** : pour ajouter l'élément avec l'identifiant id dans la collection
  - **/collection/delete/id** : pour supprimer l'élément avec l'identifiant id
  - **/collection/update/id** : pour mettre à jour l'élément avec l'identifiant id
- Angular permet de hiérarchiser les composants et les routes
  - l'ajout, la suppression et la modification d'un élément de la collection ne pourra se faire qu'à l'intérieur du composant responsable du rendu de l'URL **/collection**
- Au sein du composant parent **/collection**, les composants enfants sont insérés grâce à une deuxième directive **<router-outlet>**
- L'enfant est inséré dans le parent, lui-même inséré dans le composant principal de l'application

## Routeur : les routes

# Hiérarchie dans les routes

```
const routes: Routes = [
{
 path: 'technos', component: TechnoComponent,
 children: [
 {path: 'add', component: TechnoCreateComponent},
 {path: ':id', component: TechnoDetailsComponent},
 {path: 'delete/:id', component: TechnoDeleteComponent},
 {path: 'update/:id', component: TechnoUpdateComponent}
]
}, { path: '**', component: NotFoundComponent }];

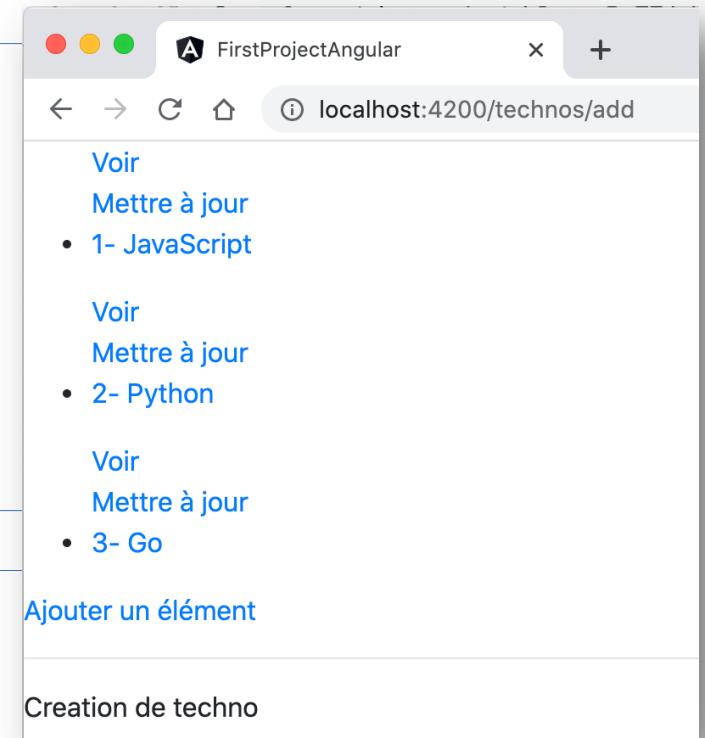
```

```
<div>
<ul *ngFor="let techno of technologies">
 <a [routerLink]=[techno.id]">Voir

 <a [routerLink]=[['update', techno.id]]>Mettre à jour

 <a [routerLink]=[['/technos', techno.id]]>{{techno.id}}- {{techno.nom}}

</div>
Ajouter un élément
<hr/>
<router-outlet></router-outlet>
```



## Routeur : les routes

# Les Redirections

- Il est nécessaire d'utiliser le module **Router** :

```
import {Router} from '@angular/router';
constructor(private router: Router) {}
```

- Rediriger vers une url : `this.router.navigate('myUrl');`



<http://localhost:4200/myUrl>

- Rediriger vers une url, avec des paramètres (**Param**) :

```
const myParam = 'example';
this.router.navigate(['myUrl', myParam]);
```



<http://localhost:4200/myUrl/example>

- Rediriger vers une url avec des paramètres recherche (**QueryParam**) :

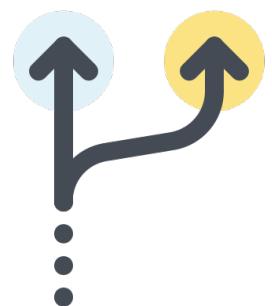
```
const myParam = 'example';
this.router.navigate(['myUrl'], { queryParams: { search: myParam }});
```



<http://localhost:4200/myUrl?search=example>

# Routeur : Les guards

Développement Web



## Les Guards

- Certaines routes sont plus sensibles que d'autres (suppression, ajout)
- Angular propose plusieurs interfaces pour limiter les accès :
  - CanActivate (<https://angular.io/api/router/CanActivate>)
  - CanActivateChild (<https://angular.io/api/router/CanActivateFn>)
  - CanDeactivate (<https://angular.io/api/router/CanDeactivateFn>)
  - CanMatch (<https://angular.io/api/router/CanMatchFn>)

```
ng generate guard AuthenticationGuard
```

```
? Which type of guard would you like to create?
❯● canActivate
○ canActivateChild
○ deactivate
○ match
```

## Routeur : les guards

### CanActivate

- Détermine si une route peut être activée
- Contrôle l'accès à une route spécifique en fonction de certaines conditions

**Exemple :** Nous souhaitons vérifier si l'utilisateur est authentifié avant de lui permettre d'accéder à une page protégée.

```
export const AuthGuard: CanActivateFn = (route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree => {
 return inject(AuthenticationService).isAuthenticated() || inject(Router).createUrlTree(['/login']);
};
```

```
export const routes: Routes = [
{
 path: '/profil',
 component: ProfilComponent,
 canActivate: [AuthGuard]
},
];
```

```
@Injectable({ providedIn: 'root' })
export class AuthenticationService {

 isAuthenticated(): boolean {
 return Math.random() >= 0.5;
}
```

Routeur : les guards

## CanActivateChild

- Similaire à CanActivate
- Contrôle l'accès aux routes enfants d'une route principale

**Exemple :** Nous souhaitons vérifier si l'utilisateur a les autorisations pour accéder aux sous-route d'une route cible

```
const routes: Routes = [
 {
 path: 'parent', component: ParentComponent, canActivate: [AuthGuard],
 canActivateChild: [CanActivateChildGuard],
 children: [
 {
 path: 'child',
 component: ChildComponent
 }
 ...
 }
```

Routeur : les guards

## CanDeactivateFn

- Permet de déterminer si une route peut être désactivée
- Si le garde renvoie **true**, la navigation continue

**Exemple** : Nous souhaitons vérifier si l'utilisateur a sauvégarde ses modifications avant de quitter la page.

```
const routes: Routes = [
 {
 path: 'user/:id',
 component: UserComponent,
 canDeactivate: [(component: UserComponent) =>
 !component.hasUnsavedChanges],
 },
];
```

## Routeur : les guards

### CanMatch

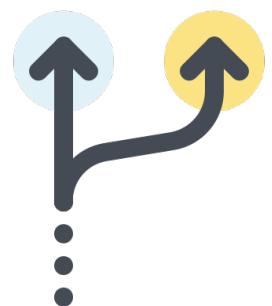
- Permet de déterminer une route en fonction d'une correspondance de modèle

**Exemple :** Nous souhaitons afficher un composant ou un autre en fonction d'une condition

```
const routes: Routes = [
 {
 path: 'customers',
 canMatch: [() => inject(FeatureFlagsService).hasPermission('beta')],
 loadComponent: () => import('./page/orders-beta.component').then(o => o.OrdersComponent)
 },
 {
 path: 'customers',
 loadComponent: () => import('./page/orders.component').then(o => o.OrdersComponent)
 }];
]
```

# Routeur : les paramètres

Développement Web



# Récupération des paramètres 1/3

- Il est nécessaire d'utiliser le module **ActivatedRoute** :

```
import {ActivatedRoute} from '@angular/router';
constructor(private activatedRoute: ActivatedRoute) {}
```



<http://localhost:4200/myUrl/example>

- Avec ActivatedRoute, il est possible de récupérer les **Param** et les **QueryParam**
- Avec les Observables :
  - Fonctionne s'il y a un changement de paramètre sur le même composant
- Sans les Observables :
  - Fonctionne une seule fois, à la construction du composant
  - Changer d'url component/param1 vers component/param2 ne recréera pas votre composant

# Récupération des paramètres 2/3

Récupération des **Param** de l'url dans un composant :



<http://localhost:4200/myUrl/example>

- **Sans Subscription :**

- La valeur ne changera pas si vous changez de paramètre, en restant sur la même route

```
ngOnInit() {
 const snapshotParam = this.activatedRoute.snapshot.paramMap.get('example');
}
```

- **Avec Subscription :**

- Fonctionne si le paramètre change avec le même composant : `myUrl/example1 => myUrl/example2`

```
this.activatedRoute.paramMap.subscribe(params => {
 const subscribedParam = params.get('example');
});
```

Routeur : les paramètres

# Récupération des paramètres 3/3

Récupération des **QueryParam** de l'url dans un composant :

http://localhost:4200/myUrl?search=example

- **Sans Subscription :**

- La valeur ne changera pas si vous changez de paramètre, en restant sur la même route

```
ngOnInit() {
 const snapshotQueryParam = this.activatedRoute.snapshot.queryParamMap.get('search');
}
```

- **Avec Subscription :**

- Fonctionne si le paramètre change avec le même composant : myUrl?search=ex1 => myUrl?search=ex2

```
this.activatedRoute.queryParamMap.subscribe(queryParams => {
 const subscribedParam = queryParams.search;
});
```

# Exercice 1/2

- Mettez en place une barre de navigation sur la page d'accueil, avec les liens suivants :
  - Page d'accueil
  - Voir les catégories
  - Voir les tutoriels
  - Créer une catégorie
  - Créer un tutoriel
- Créez les composants supplémentaires que vous jugerez nécessaires et utilisez les directives **routerLink** et **<router-outlet>** dans le fichier **app.component.html**

# Exercice 2/2

Tutorials App   Catégories   Tutoriels   Ajouter Tutoriel

**Tutoriels :**      Tri : [Plus récents](#)

**Initiation au langage Java**  
Lorem ipsum dolor sit amet, consectetur adipisicing elit  
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Alias animi aperiam aspernatur cupiditate deserunt digniss...  
Jane WAAA    j.waaa@tuto.fr

En ligne depuis le mercredi 2 décembre 2026      [Details](#)    [Partager](#)

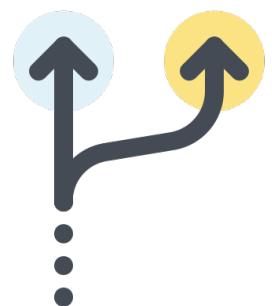
2      [Commentaires](#)

**Initiation au macro VBA**  
Lorem ipsum dolor sit amet, consectetur adipisicing elit  
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Alias animi aperiam aspernatur cupiditate deserunt digniss...  
Boris SAU    b.sau@tuto.fr

En ligne depuis le mardi 16 septembre 2025      [Details](#)    [Partager](#)

# Routeur : Les resolvers

Développement Web



## Les resolvers

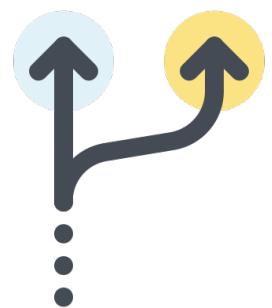
- Avec le routing, il est possible de charger les données avant le chargement de la route

```
export const tutorialResolver: ResolveFn<Tutorial> = (
 route: ActivatedRouteSnapshot,
 state: RouterStateSnapshot,
) => {
 return inject(TutorialsService).getTutorial(route.paramMap.get('id')!);
};
```

```
export const routes: Routes = [
 ...
 {
 path: 'detail/:id',
 component: TutorialDetailComponent,
 resolve: {tuto: tutorialResolver},
 }
];
```

# Routeur avancé

Développement Web



Routeur avancé

## Routeur avancé

- Avec le routing, Il est possible de chargé des modules en Lazy :
  - <https://stackblitz.com/edit/angular-example-routing-lazy-loading>

# Création d'un module avec routing

- Pour créer un nouveau module avec un routing intégré :

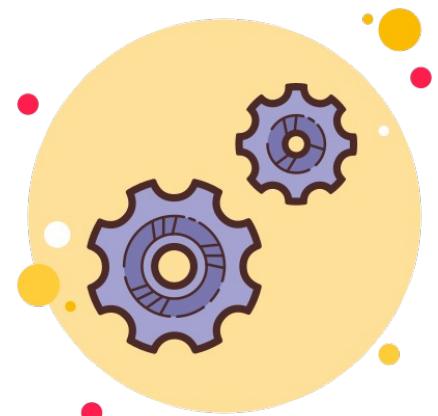
```
ng generate module myModule --routing
```

- Pour créer un composant dans un module précis :

```
ng generate component myModule/myComponent
```

# Les services

Développement Web



# Injection de dépendances et IOC

- L'**Inversion Of Control** est le patron d'architecture communs aux Frameworks
  - Elle est utilisable sous plusieurs formes
- La forme la plus commune est l'**injection de dépendances**
- L'injection de dépendances se fait via le constructeur

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

 constructor(private myDependancy: Dependancy) {}
}
```

## Les services

# Le Services 1/2

- Classe TypeScript **injectable** dans un ou plusieurs composants
  - Il existe des services fournis avec Angular
  - Nous pouvons en créer pour répondre à des besoins métier
- Un service permet de réutiliser des bouts de code partout dans l'application
  - En utiliser un, c'est créer une instance de celui-ci dans le constructeur du composant qui en a besoin
  - Le composant a ainsi accès à ses attributs et méthodes
- Pour créer un service avec le Angular CLI :  

```
ng generate service services/todoList
```

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

@Injectable({
 providedIn: 'root'
})
export class TodoListService {

 constructor() { }
}
```

## Les services

# Le Services 2/2

- Un service est souvent utilisé pour définir les opérations CRUD d'une application
  - Create Read Update Delete : 4 fonctions élémentaires de création, de lecture, de mise à jour et de suppression d'objets ou de collections dans une base de données

```
import { Injectable } from '@angular/core';
@Injectable({
 providedIn: 'root'
})
export class TodoListService {

 private todoList: any[] = [
 {name: 'Tondre la pelouse'},
 {name: 'Faire les courses'},
 {name: 'Faire le ménage'},
];

 getTodoList(): any[] {
 return this.todoList;
 }
}
```

```
import {Component, OnInit} from '@angular/core';
import {ExampleService} from './../../services/example.service';

@Component({
 selector: 'app-service',
 templateUrl: './example.component.html',
 styleUrls: ['./example.component.css']
})
export class TodolistComponent implements OnInit {

 todoList!: any[];

 constructor(private todoListService: TodoListService) {}

 ngOnInit(): void {
 this.todoList = this.todoListService.getTodoList();
 }
}
```

## Les services

### Exemple 1/2

```
export class CitationService {

 citations = [
 'Tout ce que je sais, c\'est que je ne sais rien', 'Un seul être vous manque, et tout est dépeuplé',
 'Un peuple qui oublie son passé se condamne à le revivre', 'Le hasard, c\'est Dieu qui se promène incognito',
 'Il ne faut jamais dire jamais', 'Un tableau ne vit que par celui qui le regarde'
];
 getIterations() {
 return this.citations;
 }
 deleteIteration(indexDeLaIteration) {
 this.citations.splice(indexDeLaIteration, 1);
 return this.citations;
 }
 addIteration(iteration) {
 this.citations.push(iteration);
 return this.citations;
 }
 updateIteration(indexDeLaIteration, iteration) {
 this.citations[indexDeLaIteration] = iteration;
 return this.citations;
 }
}
```

## Les services

# Exemple 2/2

```
@Component({
 selector: 'app-service',
 templateUrl: './service.component.html',
 styleUrls: ['./service.component.css']
})
export class ServiceComponent implements OnInit {

 citations = [];

 constructor(private citationService: CitationService) {}

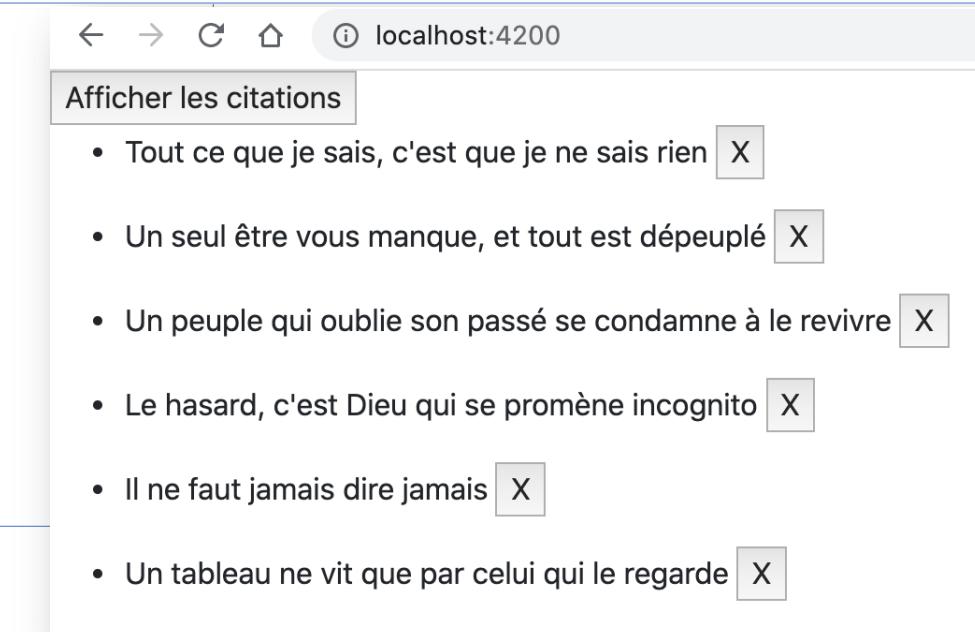
 ngOnInit() {}

 affiche() {
 this.citations = this.citationService.getCitations();
 }

 supprime(index) {
 this.citations = this.citationService.deleteCitation(index);
 }
}
```

```
<div>
 <button (click)="affiche()">Afficher les citations</button>
 <ul *ngFor="let citation of citations; let index = index">
 {{citation}} <button (click)="supprime(index)">X</button>

</div>
```

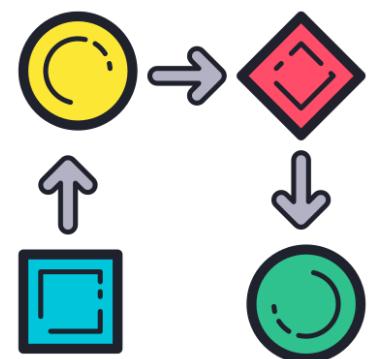


# Exercice 1/1

- Créez deux **services pour gérer les catégories et les tutoriels**
- **Le composant categories** ne doit plus contenir de tableau JSON
  - S'il souhaite utiliser la liste des categories, alors il doit faire appel au service dédié
- Déplacez le **tableau des categories** du fichier **categories.component.ts** dans le fichier TypeScript du service créé précédemment
- Procédez de la même manière pour la partie des tutoriels

# Programmation réactive

Développement Web



# Programmation asynchrone 1/2

- Paradigme de programmation qui traite de l'exécution de tâches de manière non bloquante et asynchrone
- Couramment utilisée pour des opérations longues ou des événements asynchrones
  - tels que les appels réseau, les opérations de fichiers, les interactions utilisateur, etc
- Permet à un programme de continuer à fonctionner sans attendre la fin de chaque tâche
- Améliore la réactivité et l'efficacité globale de l'application
- La notion de **callback** et de **promesse** font référence à ce **paradigme asynchrone**

## Programmation réactive

# Programmation asynchrone 2/2

```
function faireQqcALAncienne(successCallback, failureCallback) {
 console.log("C'est fait");
 // réussir une fois sur deux
 if (Math.random() > 0.5) {
 successCallback("Réussite");
 } else {
 failureCallback("Échec");
 }
}

function successCallback(résultat) {
 console.log("L'opération a réussi avec le message : " + résultat);
}

function failureCallback(erreur) {
 console.error("L'opération a échoué avec le message : " + erreur);
}

faireQqcALAncienne(successCallback, failureCallback);
```

```
function faireQqc() {
 return new Promise((successCallback, failureCallback) => {
 console.log("C'est fait");
 // réussir une fois sur deux
 if (Math.random() > 0.5) {
 successCallback("Réussite");
 } else {
 failureCallback("Échec");
 }
 });
}

const promise = faireQqc();
promise.then(successCallback, failureCallback);
```

Enchainement de Callback  
(Callback hell, Pyramide de callbacks)

Utilisable des promesses

## Programmation réactive

# Promesse 1/2

- Valeur que le développeur ne connaît pas encore
  - Trois états : **en attente, reçue ou en erreur**
  - Elle ne peut pas être annulée
  - [https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global\\_Objects/Promise](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Promise)
- Une promesse possède deux méthodes importantes : **resolve()** et **reject()**
  - **resolve()** intervient lorsque l'action entreprise par la promesse s'est bien déroulée

```
const message = new Promise((resolve, reject) => {
 resolve('Promesse reçue');
});
```

- **reject()** intervient lorsqu'une erreur s'est produite dans le traitement de la promesse

```
const message = new Promise((resolve, reject) => {
 reject("Une erreur s'est produite");
});
```

## Programmation réactive

# Promesse 2/2

- Utilisation du pipe **async** afin d'afficher la données à caractère asynchrone

```
@Component({
 selector: 'app-promise',
 templateUrl: './promise.component.html',
 styleUrls: ['./promise.component.css']
})
export class PromiseComponent implements OnInit {

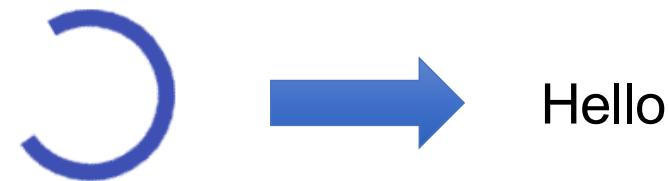
 message!: Promise<string>;

 ngOnInit(): void {

 this.message = new Promise((resolve, reject) => {
 setTimeout(() => {
 resolve("Hello");
 }, 5000);
 });
 }
}
```

```
<p *ngIf="message | async as data; else loading">{{data}}</p>

<ng-template #loading>
 <mat-spinner></mat-spinner>
</ng-template>
```



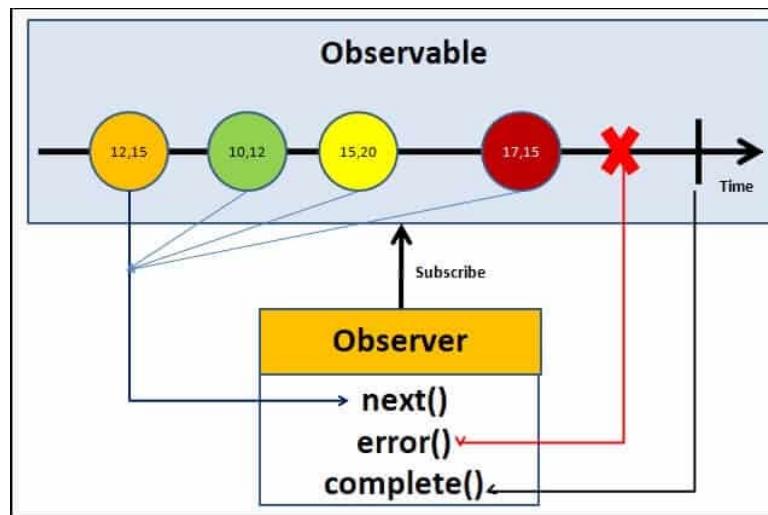
# Programmation réactive

- Repose sur l'utilisation d'observables, qui sont des flux de données asynchrones
  - Émettent des valeurs au fil du temps
  - Les observables représentent des flux continus de données (événements, mises à jour en temps réel, etc)
- Angular est basée sur la programmation réactive
  - Emission de données depuis une ou plusieurs sources
- Les données sont traitées à un flux
  - Le flux représente de la donnée qui arrive de manière ordonnée
- Le flux émet 3 types de signaux :
  - une valeur
  - une erreur
  - Un signal indiquant qu'il n'y a plus de données

## Programmation réactive

# Observable 1/2

- Flux de données auquel on peut souscrire
  - Processus divisé en deux parties distinctes : **l'observable** et **l'observateur** qui **souscrit à l'observable**
  - L'objectif est de voir l'état du flux au fil du temps et de recueillir toutes les modifications qu'il subirait



- Différents des promesses, les observables peuvent changer un nombre infini de fois de valeur
  - peut être **observé** puis **libéré**, puis **observé**, alors qu'une promesse ne peut être annulée
  - On peut observer mille fois un observable, mais une fois mille promesses

## Programmation réactive

# Observable 2/2

- **Observable** est une **classe TypeScript** située dans un paquet de la librairie **rxjs**
  - Permet d'utiliser l'asynchrone et les observables plus facilement
- La fonction est une série d'évènements que l'observable émettra
- Chaque émission s'effectue grâce à la méthode **next()**

```
const maVariable = new Observable(fonction () {
});
```

- L'observable ci-dessous émet la valeur 1, puis 2, puis 3

```
monObservable = new Observable(obs => {
 obs.next(1);
 obs.next(2) ;
 obs.next(3) ;
});
```

## Programmation réactive

# La librairie rxjs

- Simplifie grandement le travail des observables des observateurs et de l'asynchrone en général
  - <https://www.learnrxjs.io/learn-rxjs/>
- Exemple d'un Observable avec RxJs : 

`const maVariable = of(1, 2, 3);`
- la librairie apporte énormément de méthodes et d'opérateurs
  - **interval()** : permet de spécifier l'intervalle de temps avant d'émettre un nouvel évènement
  - **pipe(autreMethodeRxjs)** : permet de combiner des méthodes rxjs
  - **map(fonction)** : permet d'effectuer des opérations (définies dans la fonction) sur les données récupérées
  - **filter(critères)** : filtre les données récupérées pour n'en garder qu'une partie selon certains critères
  - **concat()** : concatène les données
  - **subscribe(fonction)** : permet d'appliquer la fonction à chaque détection d'évènement qui va modifier les données
- Nous allons faire un exemple avec les Observables

# Programmation réactive

## Observable : exemple

### ng generate component composants/observable

```
import {Component, OnDestroy, OnInit} from '@angular/core';
import {interval, Subscription} from 'rxjs';
import {takeWhile} from 'rxjs/operators';

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

```
<div class="m-4">
 <button (click)="appelDelObservable()" class="btn btn-primary">
 <span *ngIf="clique" class="spinner-grow spinner-grow-sm mr-1"
 role="status" aria-hidden="true">{{message}}
 </button>
</div>
```

```
export class ObservableComponent implements OnDestroy {

 subscription: Subscription; // L'observateur

 message = 'En attente du clic';

 appelDelObservable() {
 const observableCompteur = interval(1000); // L'observable
 this.subscription =
 observableCompteur
 .pipe(takeWhile(val => val < 4))
 .subscribe(x => {
 console.log(x);
 this.message = x;
 }, (error) => {
 this.message = 'Une erreur est survenue ' + error;
 }, () => {
 this.message = 'Chargement terminé';
 });
 }

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

En attente du clic

1

Chargement terminé

## Programmation réactive

# La librairie rxjs

- On peut voir un observable comme un tableau un peu particulier qui évolue dans le temps sur lequel des fonctions sont appliquées
- De plus, chacune de ces fonctions retourne un observable, donc elles peuvent s'enchaîner
- Pour appliquer un filtre afin de ne souscrire à l'évènement que lorsque le nombre émis par l'observable est multiple de 2 :

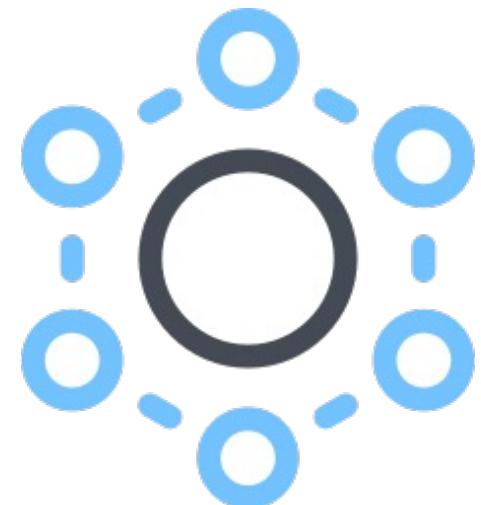
```
this.sub = this.observableCompteur.pipe(filter(num => num % 2 === 0)).subscribe(x => console.log(x));
```

- Pour y souscrire que lorsque le nombre est multiple de 2 et lui appliquer une taxe de 15 % :

```
this.sub = this.observableCompteur
 .pipe(
 filter(num => num % 2 === 0),
 map(x => x * 0.15)
)
 .subscribe(x => console.log(x));
```

# Appels réseaux

Développement Web



# Le client HTTP : Module

1/4

- Pour effectuer des appels HTTP, vous pouvez utiliser **HttpClient**
  - Présent depuis la version 4 d'Angular
  - <https://angular.io/api/common/http/HttpClient>
- Vous devez importer **HttpClientModule** :

```
export const appConfig: ApplicationConfig = {
 providers: [
 provideRouter(routes),
 // ...
 provideHttpClient(),
]
};
```



**Angular 14+**

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

@NgModule({
 imports: [
 // ...
 HttpClientModule,
],
})
export class AppModule {}
```



**Avant Angular 14**

## Le client HTTP : Service

2/4

- L'utilisation de **HttpClient** est recommandée dans les services
  - Vous devez injecter le client dans le constructeur du service

```
import { HttpClient, Observable } from '@angular/common/http';

@Injectable({
 providedIn: 'root',
})
export class ExampleService {

 constructor(private http: HttpClient) {}

 getUsers(): Observable<any[]> {
 return this.http.get<any[]>('https://localhost:8080/api/users');
 }
}
```

## Le client HTTP : Typage

3/4

- L'utilisation des interfaces est importante pour représenter les données reçues
  - Vous pourrez ainsi utiliser votre type dans la méthode

```
getUsers(): Observable<User[]> {
 return this.http.get<User[]>('https://localhost:8080/api/users');
}
```

```
[
 {
 "id": 1,
 "firstName": "Bob",
 "lastName": "Smith",
 "email": "bob.smith@hotmail.com"
 },
 ...
]
```



```
export interface User {
 id: number;
 firstName: string;
 lastName: string;
 email: string;
}
```

## Le client HTTP : Utilisation

4/4

- Pour terminer, vous pouvez utiliser votre service dans un composant
  - Vous pourrez ainsi récupérez la liste des utilisateurs et les afficher

```
@Component({
 ...
 selector: 'app-user-list'
})
export class UserListComponent implements OnInit {
 users: User[] = [];

 constructor(private userService: UserService) {}

 ngOnInit() {
 this.userService.getUsers().subscribe(users => {
 this.users = users;
 });
 }
}
```

# Abonnement aux observables 1/3

- Il existe deux méthodes pour s'abonner à un observable :
  - En utilisant `.subscribe()` comme dans l'exemple précédent
  - En utilisant le **pipe async** dans le HTML du composant

```
@Component({
 selector: 'app-user-list'
})
export class UserListComponent implements OnInit {
 users: User[] = [];

 constructor(private userService: UserService) { }

 ngOnInit() {
 this.userService.getUsers().subscribe(users => {
 this.users = users;
 });
 }
}
```

```
@Component({
 selector: 'app-user-list'
})
export class UserListComponent implements OnInit {
 private userService = inject(UserService);
 users$: Observable<User[]> = this.userService.getUsers();

}

<ng-container *ngFor="let u of users$ | async">
 <p>{{u.firstName}}</p>
</ng-container>
```

## Abonnement aux observables 2/3

- Pourquoi dois-je faire la requête dans **ngOnInit()** ?
  - S'exécute une seule fois au cours du cycle de vie du composant
  - Permet d'obtenir les données avant l'affichage du composant
  - **Améliore les performances**
- **S'inscrire** à l'observable nécessite de se **Désinscrire**. Pourquoi ?
  - Les observables se complètent automatiquement une fois la requête terminée
  - Lors d'un changement de route, des fuites de mémoires peuvent survenir
- Utiliser **le pipe async** ou bien l'abonnement avec **.subscribe()** ?
  - L'approche avec le pipe **async** est plus propre, plus efficace et **recommandée**
  - L'approche manuelle **.subscribe()** nécessite de gérer l'abonnement/désabonnement

## Abonnement aux observables 3/3

- Si vous choisissez la méthode manuelle **subscribe()**, vous pouvez désabonner :
  - Il est nécessaire de sauvegarder votre abonnement dans **Subscription**

```
@Component({...})
export class UserListComponent implements OnInit, OnDestroy {

 private subscription!: Subscription;
 users: User[] = [];

 constructor(private userService: UserService) {}

 ngOnInit() {
 this.subscription = this.userService.getUsers().subscribe(users => this.users = users);

 ngOnDestroy() {
 if (this.subscription) {
 this.subscription.unsubscribe();
 }
 }
}
```

## Appels réseaux

### Méthode subscribe()

- Avec l'abonnement manuel avec **subscribe()**, Il est possible de récupérer les erreurs :

```
ngOnInit() {
 this.subscription = this.userService
 .getUsers()
 .subscribe({
 next: (users: User[]) => {
 this.users = users;
 },
 error: (err: any) => console.error(err);
 });
}
```

## Exemple complet

- Retrouvez un exemple avec l'appel de l'API [randomuser](#) :
- <https://stackblitz.com/edit/angular-httpclient-abonnement-avec-le-pipe-async>
- Le composant **async** fournit un exemple d'appel réseau avec le **pipe async**
- Le composant **subscribe** fournit un exemple d'appel réseau avec **subscribe()**

# Les environnements

Développement Web

# Configurer des environnements 1/3

- Un environnement dans le contexte d'Angular se réfère à une configuration spécifique
- La configuration peut varier selon le contexte de déploiement

Exemple :

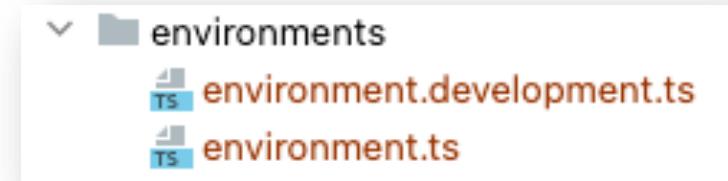
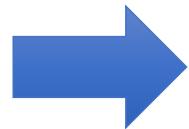
en développement, vous pourriez avoir besoin d'une URL d'API différente de celle de la production

<https://angular.io/guide/build>

## Configurer des environnements 2/3

- Pour générer les fichiers d'environnement, vous pouvez utiliser la commande :

```
ng generate environments
```



- Par défaut, le dossier créé contient deux fichiers d'environnements :
  - environnement.ts (**pour la production**)
  - environment.development.ts (**pour le développement**)
- Pour ajouter un autre environnement, il suffit de créer un fichier

```
// environment.staging.ts
export const environment = {
};
```

## Les environnements

# Configurer des environnements 3/3

- Vous pouvez utiliser la configuration de l'environnement dans votre application

```
@Injectable({
 providedIn: 'root'
})
export class ExampleService {
 baseUrl: string = environment.BASE_API_URL;
}
```

```
// environment.development.ts
export const environment = {
 production: false,
 // Careful, must NOT end with /
 BASE_API_URL: 'http://localhost:8080/api'
};
```

```
// environment.ts
export const environment = {
 production: true,
 // Careful, must NOT end with /
 BASE_API_URL: 'http://api.mycompany.fr'
};
```

# Construire avec l'environnement ciblé

- Si vous avez plusieurs environnements, vous pouvez utiliser la configuration de l'environnement dans votre application :

```
ng serve --configuration=development
```

- Pour la construction de l'application, le même paramètre est utilisé :

```
ng build --configuration=staging
```



## Les environnements

# Injection environnements 1/2

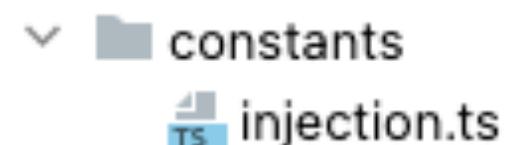
- Vous pouvez aussi rendre injectable directement la BASE\_URL de votre environnement

```
import {BASE_API_URL} from "./constants/injection";
import {environment} from "../environments/environment";

export const appConfig: ApplicationConfig = {
 providers: [
 ...
 { provide: BASE_API_URL, useValue: environment.BASE_API_URL }
];
};
```

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

```
export const BASE_API_URL = new InjectionToken<string>('BASE_API_URL');
```



## Les environnements

# Injection environnements 2/2

```
import {Inject, Injectable} from '@angular/core';
import {BASE_API_URL} from "../constants/injection";

@Injectable({
 providedIn: 'root'
})
export class CategoriesService {

 constructor(private http: HttpClient, @Inject(BASE_API_URL) private baseUrl: string) {
 }

 getAll(): Observable<Category[]> {
 return this.http.get<Category[]>(`${this.baseUrl}/categories`);
 }
}
```

# Exercice 1/1

- Mettre en place les appels réseaux vers le webservice pour afficher les tutoriels et les categories
- Vous utiliserez le client HttpClient
  - Avant Angular 14, n'oubliez pas d'importer le module **HttpClientModule**
  - Avec Angular 14+, n'oubliez pas de définir **provideHttpClient()** dans **app.config.ts**
- Mettre en place un environnement de développement
- Utilisez les interfaces pour typer vos résultats

# Les formulaires

Développement Web



## Les formulaires

# Introduction aux formulaires

Les formulaires doivent être intelligents et s'adapter aux réactions des utilisateurs

Angular propose 2 façons d'écrire un formulaire :

- **Template Forms :**
  - La première consiste à mettre en place l'ensemble des champs dans le template
  - Méthode suffisante pour un formulaire simple, nécessitant peu de contrôle
  - <https://stackblitz.com/edit/angular-exemple-formulaire-template-avec-validation>
- **Reactive Forms :**
  - Consiste à décrire le formulaire dans le composant TypeScript
  - Permet de mettre en place des tests unitaires plus facilement
  - <https://stackblitz.com/edit/angular-exemple-formulaire-reactive-avec-validation>

## Exemple :

le bouton permettant de valider le formulaire reste grisé tant que l'utilisateur n'a pas coché la case acceptant les conditions générales d'utilisation d'un site internet

# Les Template Forms

Développement Web



## Les formulaires template

# FormsModule

- module Angular qui contient une série de directives et de services
  - Permet de créer et de gérer des formulaires dans votre application Angular
- Contient les directives suivantes :
  - **ngForm** : initialise un formulaire Angular à partir d'un formulaire HTML
  - **ngModel** : fait la liaison entre la valeur d'un champ et l'attribut d'un `@Component`
  - **ngModelGroup** : regroupe des champs afin de partager une validation/gestion commune
  - **formControlName** : fait la liaison d'un `FormControl` dans un `FormGroup`
- Vous devez importer **FormsModule** pour faire des formulaires

## Les formulaires template

### ngForm 1/2

- module d'Angular qui fournit des directives/classes pour faire des formulaires HTML
  - Permet de créer des formulaires dynamiques
  - Permet de valider les données saisies par l'utilisateur
- Utilisé pour accéder aux données saisies par l'utilisateur
  - Permet de vérifier si le formulaire est valide avant de soumettre les données

```
<form #myForm="ngForm">

</form>
```

## Les formulaires template

# ngForm 2/2

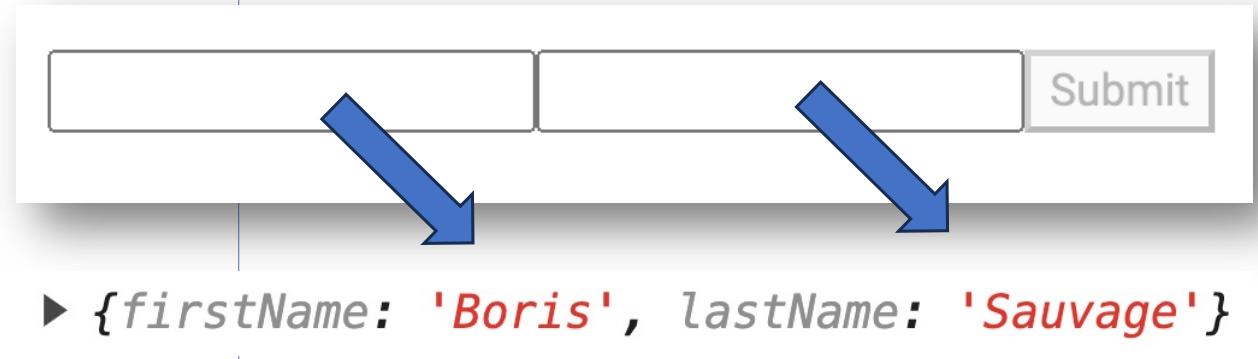
- Exemple d'utilisation de ngForm avec ngModel :

```
import {Component} from '@angular/core';
import {FormsModule, NgForm} from "@angular/forms";

@Component({
 selector: 'app-form',
 standalone: true,
 imports: [FormsModule],
 templateUrl: './form.component.html',
 styleUrls: ['./form.component.css'
})
export class FormComponent {

 onSubmit(form: NgForm) {
 console.log(form.value);
 // ..instructions
 }
}
```

```
<form #myForm="ngForm" (ngSubmit)="onSubmit(myForm)">
<input type="text" name="firstName" ngModel required>
<input type="text" name="lastName" ngModel required>
<button type="submit" [disabled]="!myForm.valid">Login</button>
</form>
```



## Les formulaires template

# ngModel 1/2

- Directive qui fait la liaison entre un attribut de @Component et un champs
  - synchronise les données saisies du formulaire vers @Component et inversement

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

@Component({ ... })
export class FormComponent {

 username: string = "";
 email: string = "";
 feedback: string = "";

 onSubmit() {
 console.log(`Pseudo : ${this.username}`);
 console.log(`Email: ${this.email}`);
 console.log(`Avis: ${this.feedback}`);
 // instruction
 }
}
```

```
<form (ngSubmit)="onSubmit()">
 <label>Nom :</label>
 <input type="text" [(ngModel)]="username" name="username">
 <label>Email :</label>
 <input type="email" [(ngModel)]="email" name="email">
 <label>Avis :</label>
 <textarea [(ngModel)]="feedback" name="feedback"></textarea>
 <button type="submit" (click)="onSubmit()">Soumettre</button>
</form>
```

The screenshot shows a simple form with three text inputs and one textarea, each preceded by a label. Below the form is a single button labeled "Soumettre".

Nom :

Email :

Avis :

**Soumettre**

## Les formulaires template

### ngModel 2/2

- ngModel est utilisé pour lier :
  - l'attribut "username" du @Component à l'input de type text
  - l'attribut "email" du @Component à l'input de type email
  - l'attribut "feedback" du @Component à la textarea
- À chaque saisies dans les champs, les attributs sont mis à jour avec ces informations
  - La syntaxe `[]()` permet la liaison bidirectionnelle
- `[]` est utilisée pour la **liaison unidirectionnelle** de données
  - Les données sont reliées d'un objet à un autre sans la mise à jour dans l'autre sens
- `()` est utilisée pour la **liaison de données dans l'autre sens**
  - Les données sont reliées d'un objet à un autre et sont mises à jour dans les deux sens

## Les formulaires template

# ngModelOptions 1/2

- Permet de configurer le comportement de liaison des données bidirectionnelles lorsque vous utilisez ngModel dans vos modèles HTML
- Voici quelques-unes des options disponibles avec ngModelOptions :

### updateOn

- Paramètre le moment où la valeur de ngModel doit être mise à jour
- Les options possibles sont "blur" (par défaut), "submit", "change" ou une combinaison de ces événements séparés par une virgule

### debounce

- Spécifie un délai avant que la valeur du modèle ne soit mise à jour
- Evite les mises à jour trop fréquentes du modèle lors de la saisie

## Les formulaires template

# ngModelOptions 2/2

Exemple :

```
<form>
 <label>Nom :</label>
 <input type="text" [(ngModel)]="username"
 name="username" [ngModelOptions]="{updateOn: 'blur'}">
</form>
```

# Les Reactive Forms

Développement Web



# Les ReactiveForms 1/2

- Les formulaires réactifs sont manipulés en tant qu'objet dans le **@Composant**
- Nécessite l'import du module **ReactiveModule**
  - Utilisation de **FormGroup** pour définir un formulaire
  - Utilisation de **FormControl** pour définir un champ

```
monFormulaire = new FormGroup({
 unChamp: new FormControl(''),
 unAutre: new FormControl("")
});
```

La chaîne de caractères vide est la valeur initiale du champ

```
@NgModule({
 declarations: [
 AppComponent,
 ...
],
 imports: [
 ...,
 ReactiveFormsModule
],
 providers: [
 { provide: FormBuilder }
]
})
```



## Les reactiveForms

# Les ReactiveForms 2/2

Liaison du HTML avec le formulaire défini dans le composant :

- **[formGroup]** lie la balise HTML **<form>** à l'instance de **FormGroup**
- **formControlName** lie une balise HTML d'un élément à une instance de **FormControl**

```
<form [formGroup]="myForm" (ngSubmit)="onSubmit(myForm)">
 <label>Nom :</label>
 <input type="text" formControlName="username" name="username">

 <label>Email :</label>
 <input type="email" formControlName="email" name="email">

 <label>Avis :</label>
 <textarea formControlName="feedback" name="feedback"></textarea>

 <button type="submit">Soumettre</button>
</form>
```

```
myForm!: FormGroup;

ngOnInit() {
 this.myForm = new FormGroup({
 username: new FormControl(""),
 email: new FormControl(""),
 feedback: new FormControl("")
 });
}
```

# Les reactiveForms

## FormBuilder

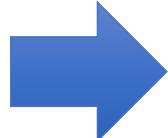
- Il est possible de simplifier le FormGroup avec un FormBuilder

```
export class FormComponent implements OnInit {

 form!: FormGroup;

 ngOnInit() {
 this.form = new FormGroup({
 username: new FormControl(""),
 email: new FormControl(""),
 feedback: new FormControl("")
 });

 }
}
```



```
export class FormComponent implements OnInit {

 form!: FormGroup;

 constructor(private fb: FormBuilder) {}

 ngOnInit() {
 this.form = this.fb.group({
 username: "",
 email: "",
 feedback: ""
 });
 }
}
```

## Les reactiveForms

# PatchValue : remplir le formulaire

- Il est possible de remplir un formulaire avec des données en utilisant **patchValue**
  - Dans le cas d'une édition par exemple

```
ngOnInit() {
 this.form = this.fb.group({
 username: "",
 email: "",
 feedback: ""
 });

 this.form.patchValue({
 username: 'sauvageb',
 email: 'sauvageboris.pro@gmail.com',
 feedback: 'Angular is cool'
 })
}
```

# Soumettre le formulaire

- Pour envoyer les informations du formulaire, il suffit d'utiliser **ngSubmit**

```
<form [formGroup]="form" (ngSubmit)="onSubmit(form)">
 ...
 <button type="submit">Soumettre</button>
</form>
```

- Et dans le @Component, utiliser le **formGroup.value** :

```
onSubmit(form: FormGroup) {
 if (form.valid) {
 console.log(`Form : ${JSON.stringify(this.form.value)}`);
 }
}
```

Form : {"username":"sauvageb","email":"sauvageboris.pro@gmail.com","feedback":"Angular is cool"}

# Exercice 1/3

- Mettre en place un formulaire pour ajouter un tutoriel
- Le formulaire devra être réactifs (**ReactiveFormsModule**)
  - Utilisez **formGroup**, **formControlName** et **(ngSubmit)**
- Commencez par définir un formulaire avec les champs :
  - Titre, Description, Contenu (<https://material.angular.io/components/form-field/overview>)
  - Date de création (<https://material.angular.io/components/datepicker/overview>)
- Vous pouvez créer une interface **CreateTutorial**
  - contenant les informations nécessaires à envoyer

```
export interface CreateTutorial {
 title: string;
 description: string;
 content: string;
 category: Category;
 author: User;
}
```

## Exercice 2/3

Dans votre méthode exécutée par (**ngSubmit**), vous pourrez :

- Envoyez les données grâce à votre **tutorialService**
  - Vous pouvez créer une interface CreateTutorial contenant uniquement les informations nécessaires
  - Une méthode devra utiliser **httpClient.post()** <https://angular.io/api/common/http/HttpClient#post>
- Les méthodes de vos services renvoient des Observables
  - Vous devez systématiquement souscrire à ces méthodes
  - Si vous utilisez **subscribe()**, pensez à **unsubscribe()** dans le **ngOnDestroy**
- Redirigez l'utilisateur vers la page affichant tous les tutoriels
  - <https://angular.io/api/router/Router#navigate>

# Exercice 3/3

- Ensuite, envisagez de compléter le formulaire avec la sélection d'une catégorie

## Ajouter un tutoriel

Titre

---

Description

---

Choisir une catégorie

---

Tutoriel

---

Choisir une date

---

MM/DD/YYYY

Ajouter

## Ajouter un tutoriel

Titre  
Créer un formulaire en Javascript

---

Description  
Mise en place d'un formulaire réactif avec ReactiveFormsModule

---

Choisir une catégorie  
Angular

---

Tutoriel  
Mettre en place un formulaire pour ajouter un tutoriel  
Le formulaire devra être réactifs (ReactiveFormsModule)  
Utilisez formGroup, formControlName et (ngSubmit)  
Commencez par définir un formulaire avec les champs :  
Titre, Description, Contenu (<https://material.angular.io/components/form-field/overview>)  
Date de création (<https://material.angular.io/components/datepicker/overview>)  
Vous pouvez créer une interface CreateTutorial contenant les informations nécessaires à envoyer

---

Choisir une date  
30/01/2030

---

MM/DD/YYYY

Ajouter

# La validation

Développement Web



# La validation des formulaires

## Introduction

Les formulaires doivent pouvoir être validés

Angular propose de valider les formulaires dans les deux méthodes :

- **Template Forms :**
  - Utilisation de propriété natives comme **required**, **minLength**, **maxLength**
- **Reactive Forms :**
  - Utilisation de validateurs Typescript synchrone et asynchrone

# Validation Template

Développement Web



# Validation des champs de saisie

- Angular propose des validateurs de champs de saisie
  - répondent à la plupart des problématiques des applications web modernes
  - Chacun de ces validateurs est une méthode provenant de la classe **Validators**
- Pour un champ de **n'importe quel type** :
  - **required()** permet d'indiquer que le champ est requis
  - **pattern(regEx)** permet de valider l'entrée saisie si elle correspond à l'expression régulière
- Pour un champ de type **texte** :
  - **minLength(nombre)** permet de s'assurer que la chaîne a le nombre minimum de caractères
  - **maxLength(nombre)** permet de s'assurer que la chaîne n'a pas plus qu'un certain nombre
- Pour une **adresse email** :
  - **email()** permet de s'assurer que la chaîne entrée est bien une adresse e-mail
- Pour un **nombre** :
  - **min(nombre)** permet de définir une valeur minimum
  - **max(nombre)** permet de définir une valeur maximum

## La validation des template forms

# Validateurs 1/4

- La conception d'un *template form* se fait dans l'HTML
  - Il est possible d'ajouter des validateurs aux champs (required, minLength, etc)

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

<input type="text" [(ngModel)]="firstName" name="firstName" required minLength="2" maxLength="25">
```

### Attribut HTML disponibles pour les validateurs

required

min / max

pattern

minLength / maxLength

step

## La validation des template forms

# Validateurs 2/4

- Vous pouvez connaître son état validé ou non dans le @Component

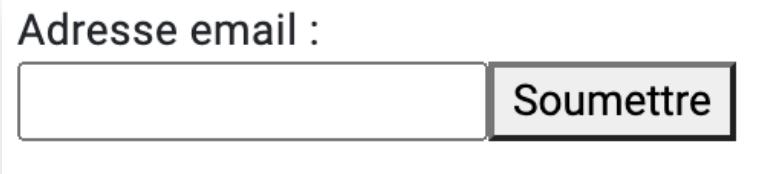
```
@Component({ ... })
export class FormComponent {

 email: string = "";

 onSubmit(form: NgForm) {
 if (form.invalid){

 }
 }
}
```

```
<form #form="ngForm" (submit)="onSubmit(form)">
 <label for="email">Adresse email :</label>
 <input type="email" id="email"
 [(ngModel)]="email" name="email" required>
 <button type="submit">Soumettre</button>
</form>
```



## La validation des template forms

# Validateurs 3/4

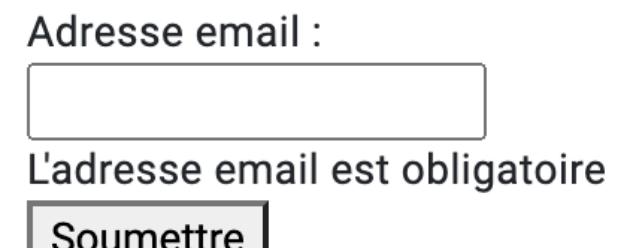
- Vous pouvez connaître son état validé ou non dans le HTML
  - Utilisation de #emailRef="ngModel" pour référencer le champ

```
<form #form="ngForm" (submit)="onSubmit(form)">
 <label for="email">Adresse email :</label>

 <input type="email" id="email"
 [(ngModel)]="email" name="email" #emailRef="ngModel" required>

 <div *ngIf="emailRef.invalid && emailRef.touched">
 <div *ngIf="emailRef.errors?.['required']">L'adresse email est obligatoire</div>
 </div>

 <button type="submit">Soumettre</button>
</form>
```



## La validation des template forms

# Validateurs 4/4

- Il est intéressant de référencer le formulaire entier avec `ngForm`
  - Vous pourrez ainsi afficher les erreurs de tous vos champs facilement

```
<form #formRef="ngForm" (submit)="onSubmit(formRef)">
 <label for="email">Adresse email :</label>
 <input type="email" id="email" ngModel name="email" required>

 <div *ngIf="formRef.controls['email']?.invalid && formRef.controls['email']?.touched">
 L'adresse email est obligatoire
 </div>

 <button type="submit">Soumettre</button>
</form>
```

- L'accès aux contrôles du formulaire est simplifié :

`formRef.controls['email']?.invalid`

`formRef.controls['email']?.touched`

# Validation des formulaires 1/3

Angular propose plusieurs états à utiliser sur les champs d'un formulaire :

- Ces états sont toutes des propriétés de la classe **AbstractControl**
  
- **Valid** : Les validations du champ sont OK
- **Invalid** : Les validations du champ ne passent pas
  
- **Dirty** : La valeur du champ a changé
- **Pristine** : La valeur du champ n'a pas changé
  
- **Touched** : Le champ a été visité **et** a perdu le focus
- **Untouched** : Le champ n'a pas été visité

## La validation des template forms

# Validation des formulaires 2/3

- <https://stackblitz.com/edit/angular-exemple-etats-des-champs-formulaire>

Angular - Exemple états des champs d'un formulaire

Prénom\*

**Valider** **Annuler**

Le champ prénom est invalide

Le champ prénom est pristine (la valeur n'a pas changé)

Le champ prénom est untouched (Champ non visité)

Angular - Exemple états des champs d'un formulaire

Prénom\*

**Valider** **Annuler**

Le champ prénom est valide

Le champ prénom est dirty (la valeur a changé)

Le champ prénom est touched (Champ a été visité et a perdu le focus)

## La validation des template forms

# Validation des formulaires 3/3

- <https://stackblitz.com/edit/angular-exemple-etats-des-champs-formulaire>

```
<div class="header">
 <h2>Exemple états des champs d'un formulaire</h2>
</div>
<form
 (ngSubmit)="f.form.valid && onSubmit()"
 #f="ngForm">
 <mat-card class="main-form">
 <mat-form-field appearance="fill">
 <mat-label>Prénom</mat-label>
 <input matInput name="firstName"
 [(ngModel)]="model.firstName"
 #firstName="ngModel" required />
 </mat-form-field>
 <mat-card-actions>

 <button mat-flat-button color="primary"
 type="submit">Valider</button>

 <button mat-flat-button color="warn"
 type="reset">Annuler</button>

 </mat-card-actions>
 </mat-card>
```

```
<mat-card>
 <!-- VALID / INVALID -->
 @if (firstName.valid) {
 <p class="text-success">Le champ prénom est valide</p>
 } @if (firstName.invalid) {
 <p class="text-danger">Le champ prénom est invalide</p>
 }
</mat-card>
<mat-card>
 <!-- DIRTY / PRISTINE -->
 @if (firstName.dirty) {
 <p class="text-primary">Le champ prénom est dirty (la valeur a changé)</p>
 } @if (firstName.pristine) {
 <p class="text-secondary">Le champ prénom est pristine (la valeur n'a pas changé)</p>
 }
</mat-card>
<mat-card>
 <!-- TOUCHED / UNTOUCHED -->
 @if (firstName.touched) {
 <p class="text-info">Le champ prénom est touched (Champ a été visité et a perdu le focus)</p>
 } @if (firstName.unouched) {
 <p class="text-secondary">Le champ prénom est untouched (Champ non visité)</p>
 }
</mat-card>
</form>
```

## La validation des template forms

# ngModelGroup 1/2

- Permet de regrouper des champs dans le formulaire
  - Il est également possible de nommer le modelGroup pour la validation

```
@Component({
 selector: 'app-form',
 standalone: true,
 imports: [
 FormsModule
],
 templateUrl: './form.component.html',
 styleUrls: ['./form.component.css']
})
export class FormComponent {

 user: any= {firstName: "", lastName: ""};

}
```

```
<form #form="ngForm" (submit)="onSubmit()">
 <div ngModelGroup #nameCtrl="ngModelGroup">
 <input name="first" [(ngModel)]="user.firstName">
 <input name="last" [(ngModel)]="user.lastName">
 </div>
 <button type="submit">Soumettre</button>
</form>
```

## La validation des template forms

# ngModelGroup 2/2

- Démonstration de la validation du groupe de modèles

```
<form #form="ngForm" (submit)="onSubmit()">
 <div ngModelGroup #nameCtrl="ngModelGroup">
 <input name="first" [(ngModel)]="user.firstName">
 <input name="last" [(ngModel)]="user.lastName">
 </div>
 <button type="submit">Soumettre</button>
 <p>Formulaire Valide {{nameCtrl.valid}}</p>
</form>
```

# Validation Reactive

Développement Web



## Observable de FormControl

- Il est possible d'écouter les changements d'un **FormControl**

**Exemple :** Sur un champs de recherche, nous pouvons récupérer les saisies :

```
ngOnInit() {
 this.searchField = new FormControl();

 this.searchField.valueChanges
 .pipe(debounceTime(400), distinctUntilChanged())
 .subscribe((terms: any) => console.log(terms));
}
```

- **debounceTime** permet d'attendre 400ms avant de poursuivre le flux
- **distrincUntilChanged** émet la valeur saisie seulement si elle a changé dans la saisie
- Enfin, pourrions appeler une API Rest pour rechercher en base de données

## La validation des reactive forms

# Validateurs synchrones 1/2

- Sur chaque FormControl, il est possible d'ajouter des validateurs :

```
export class FormComponent implements OnInit {

 usernameControl = new FormControl("", [Validators.required, Validators.minLength(5)]);
 emailControl = new FormControl("", [Validators.required, Validators.email]);
 feedbackControl = new FormControl("", [Validators.required]);
 form!: FormGroup;

 constructor(private fb: FormBuilder) {}

 ngOnInit() {
 this.form = this.fb.group({
 username: this.usernameControl,
 email: this.emailControl,
 feedback: this.feedbackControl
 });
 }
}
```

## La validation des reactive forms

# Validateurs synchrones 2/2

- Il existe de nombreux validateurs existants :

Validateur	Description
Validators.required	Le champ de formulaire doit être renseigné
Validators.minLength(n)	Le champ doit avoir au moins n caractères
Validators.maxLength(n)	Le champ doit avoir au plus n caractères
Validators.email	Le champ doit être un format de courriel valide
Validators.pattern(regex)	Le champ doit respecter un motif spécifique défini par la expression régulière regex
Validators.nullValidator	Ne valide aucune valeur. Utilisé pour désactiver la validation d'un contrôle de formulaire.
Validators.compose([])	Prend en entrée un tableau de validateurs et renvoie un nouveau validateur qui exécute chaque validateur dans le tableau et renvoie une erreur si l'un d'eux échoue.
Validators.min(n)	Exige que la valeur du champ de formulaire soit supérieure ou égale à n.
Validators.max(n)	Exige que la valeur du champ de formulaire soit inférieure ou égale à n.

## La validation des reactive forms

# Affichage des erreurs

- Dans l'HTML, il est possible d'afficher les erreurs des **Validators** :

```
<form [FormGroup]="form" (ngSubmit)="onSubmit(form)">
 <label>Nom :</label>
 <input type="text" formControlName="username" name="username">

 <div *ngIf="usernameControl.invalid && (usernameControl.dirty || usernameControl.touched)">
 <div *ngIf="usernameControl.errors?.['required']">Le champ est obligatoire.</div>
 <div *ngIf="usernameControl.errors?.['minlength']">Le champ doit avoir au moins {{usernameControl.errors?.['minlength'].requiredLength }} caractères (en cours : {{ usernameControl.errors?.['minlength'].actualLength }}).</div>
 </div>
</div>
...
...
```

Nom :

Le champ doit avoir au moins 5 caractères (en cours : 1).

## La validation des reactive forms

# Validateurs synchrones personnalisés

- Il est possible de créer des fonctions de validation pour les FormControl :

```
emailValidator(domain: string): ValidatorFn {
 return (control: AbstractControl<string>): { [key: string]: any } | null => {
 const forbidden = !control.value.endsWith(domain);
 return forbidden ? {'forbiddenEmail': {value: control.value}} : null;
 };
};
```

```
<label>Email :</label>
<input type="email" formControlName="email"
name="email">
<div *ngIf="emailControl.hasError('forbiddenEmail')">
 <p>L'adresse email est invalide</p>
</div>
```

```
emailControl = new FormControl('', [
 Validators.required,
 this.emailValidator('gmail.com')
]);
```

Email :   
L'adresse email est invalide

# Exercice 1/2

- Sur le formulaire d'ajout de tutoriel, ajouter des contraintes sur les valeurs
  - Affichez des messages d'erreurs si les champs ne sont pas valides
  - Désactivez le bouton d'ajout si le formulaire est invalide
- Si vous le souhaitez, vous pouvez créer un validateur personnalisé pour la date :

```
dateInPast(): (control: AbstractControl) => Observable<{ [key: string]: boolean } | null> {
 return (control: AbstractControl): Observable<{ [key: string]: boolean } | null> => {
 return new Observable((observer) => {
 const currentDate = new Date();
 const selectedDate = new Date(control.value);
 currentDate.setHours(0, 0, 0, 0);
 selectedDate.setHours(0, 0, 0, 0);
 if (selectedDate < currentDate) {
 observer.next({dateInPast: true});
 } else {
 observer.next(null);
 }
 observer.complete();
 });
 };
}
```

```
createdAt: [new Date(), [Validators.required],
 [this.dateInPast()]],
```

# La validation des formulaires réactifs

## Exercice 2/2

### Ajouter un tutoriel

Titre \*

Le titre ne doit pas être vide

Description \*

La description ne doit pas être vide

Choisir une catégorie \*

La categorie ne doit pas être vide

Tutoriel \*

Le tutoriel ne doit pas être vide

Choisir une date \*

17/10/2023

La date ne peut pas être passée

### Ajouter un tutoriel

Titre  
Créer un formulaire en Javascript

Description  
Mise en place d'un formulaire reactif avec ReactiveFormsModule

Choisir une catégorie  
Angular

Tutoriel  
Mettre en place un formulaire pour ajouter un tutoriel  
Le formulaire devra être réactifs (ReactiveFormsModule)  
Utilisez formGroup, formControlName et (ngSubmit)

Commencez par définir un formulaire avec les champs :  
Titre, Description, Contenu (<https://material.angular.io/components/form-field/overview>)  
Date de création (<https://material.angular.io/components/datepicker/overview>)

Vous pouvez créer une interface CreateTutorial contenant les informations nécessaires à envoyer

Choisir une date  
30/01/2030

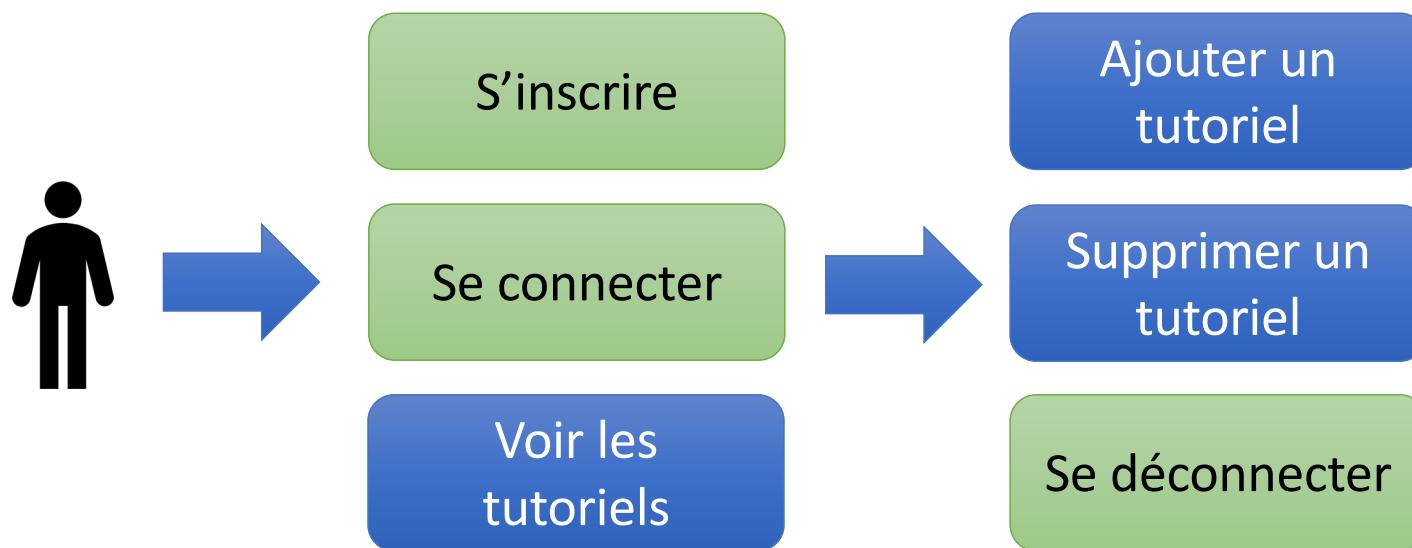
MM/DD/YYYY

# Authentification

Développement Web

# Introduction 1/2

- **Angular** est un Framework Javascript pour construire des applications web
  - Les applications Angular s'exécutent dans le navigateur
- Il est souvent nécessaire de mettre en place une session utilisateur

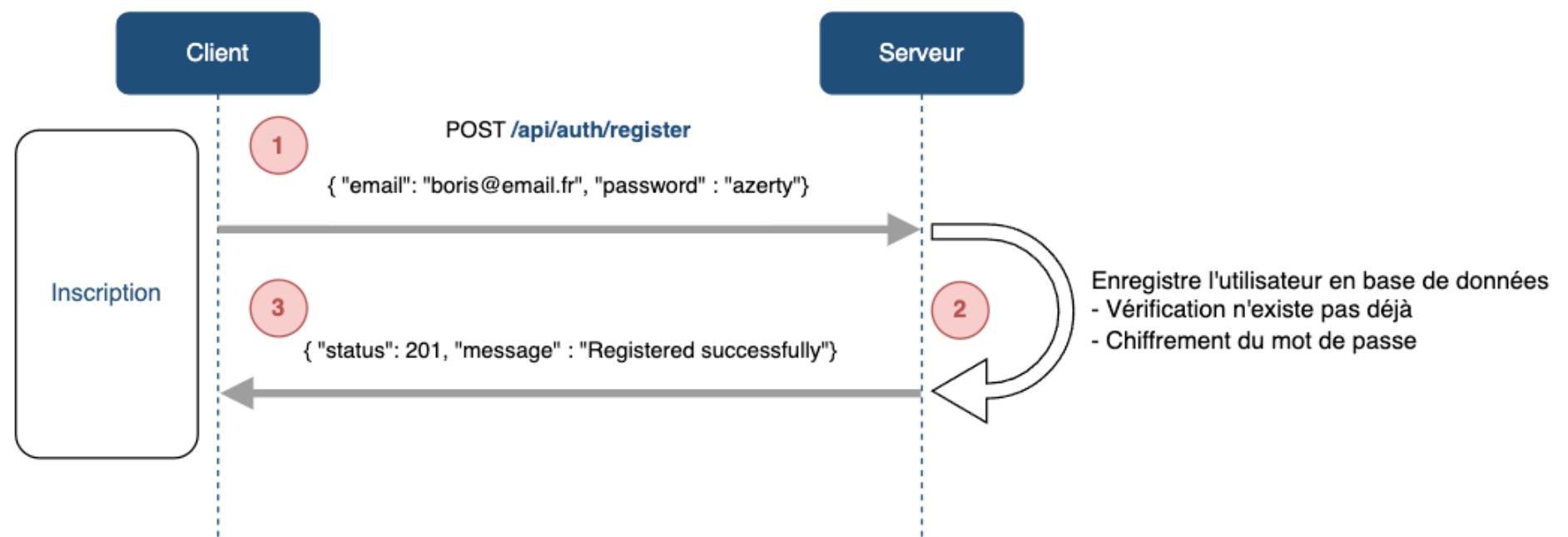


# Introduction 2/2

- Il existe de nombreuses techniques pour gérer l'authentification et l'autorisation
  - Protocole Oauth
  - Protocole Oauth OIDC (OpenID Connect)
  - Le standard ouvert JSON Web Token
- Nous utiliserons le standard JWT : <https://jwt.io/>
  - Définit dans la RFC7519 (<https://datatracker.ietf.org/doc/html/rfc7519>)
  - Jeton d'accès qui permet un échange sécurisé de données entre deux parties

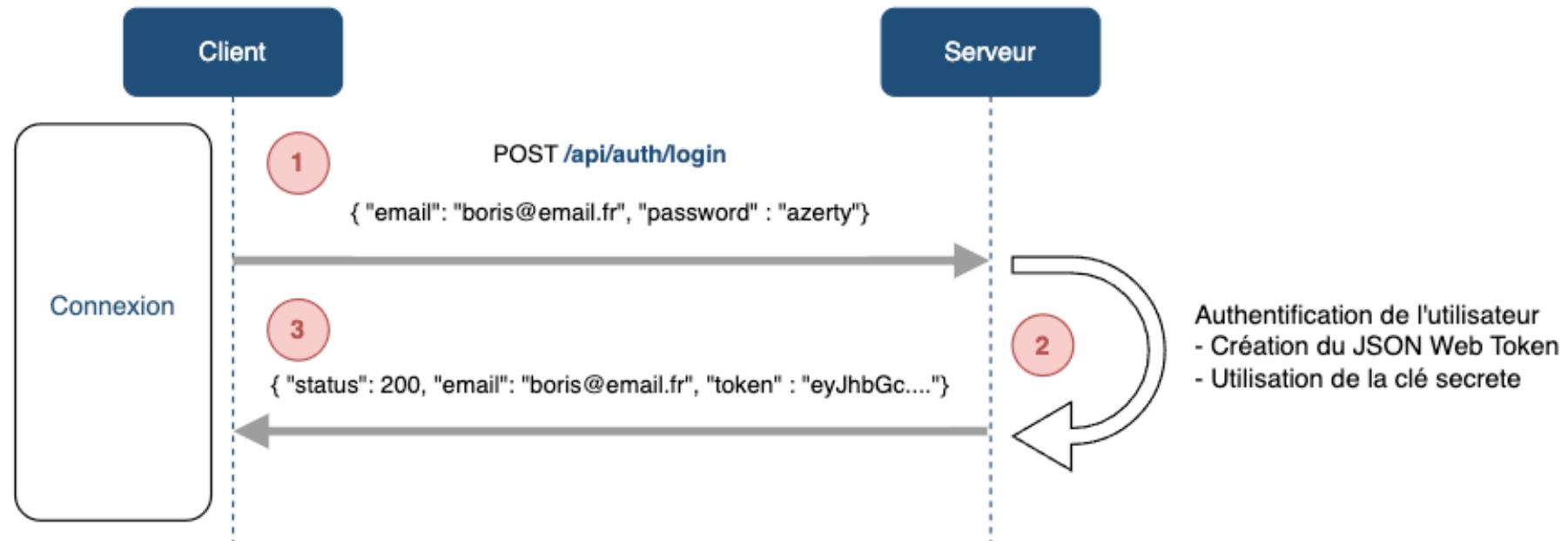
# JWT : Les échanges client / serveur 1/3

- À gauche, le client Angular doit permettre à l'utilisateur de s'inscrire
  - L'envoi d'un email et d'un mot de passe sont effectué via un formulaire
  - Le formulaire envoie les informations en HTTP POST



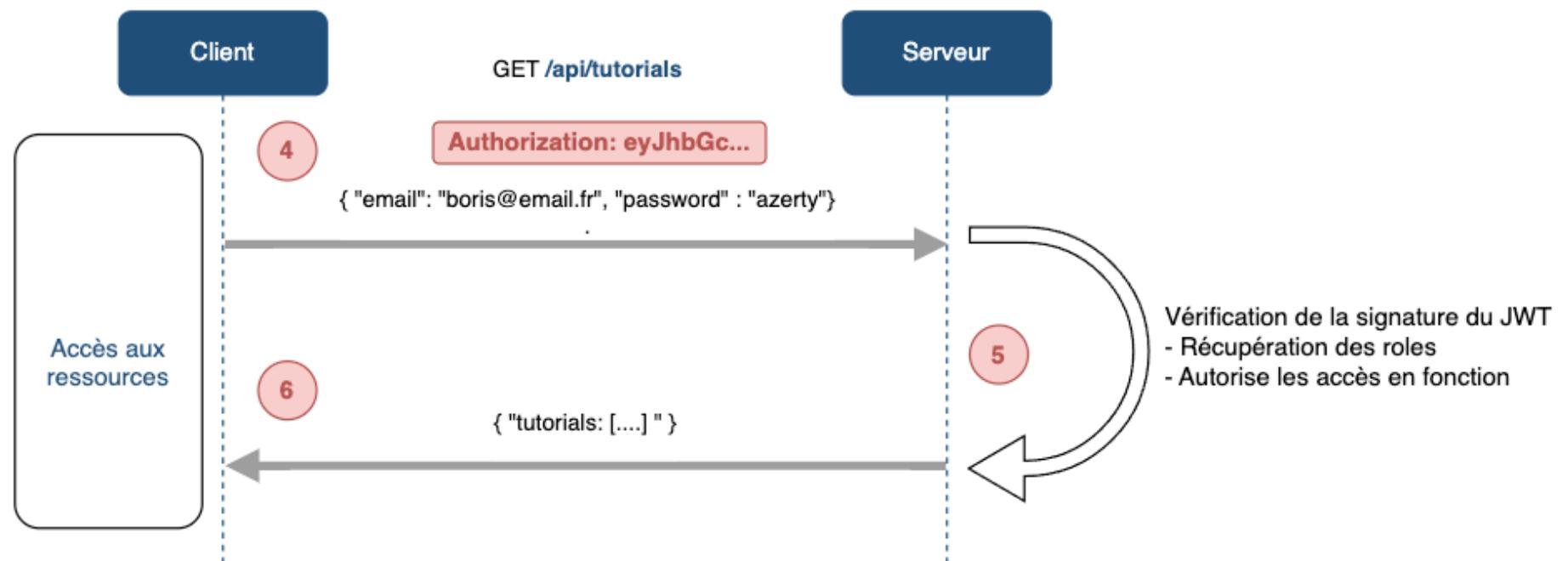
# JWT : Les échanges client / serveur 2/3

- Une fois inscrit, l'utilisateur peut se connecter sur l'application Angular
  - L'envoi d'un email et d'un mot de passe sont effectués via un formulaire
  - Le formulaire envoie les informations en HTTP POST
  - Une fois authentifié, le serveur renvoie un jeton JWT



# JWT : Les échanges client / serveur 3/3

- Le jeton a été stocké dans l'application Angular
  - Ce stockage permet d'identifier notre utilisateur comme Authentifié
  - Il peut désormais accéder aux ressources protégés en envoyant le jeton



# Le JWT 1/2

Authorization: eyJhbGc...

- Composé de 3 parties codées en base64, la structure du JWT est la suivante :
  - HEADER.PAYLOAD.SIGNATURE
- **Header** contient le type de jeton et l'algorithme de chiffrement utilisé
- **Payload** contient l'émetteur du jeton, le sujet, l'application concernée, l'expiration, l'identifiant utilisateur, les roles
- **Signature** est créée en fonction du **Header** et du **Payload**, chiffrée avec une clé secrète

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SfIKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV\_adQssw5c

# Le JWT 2/2

- Il est possible d'utiliser le site internet jwt.io pour encoder ou décoder un jeton
  - Permet de comprendre le JWT

**Encoded** PASTE A TOKEN HERE

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJzYXV2YWdlYm9yaXMucHJvQHR1dG9yaWFsLmZyIiwiaWF0IjoxNzA3ODI5ODA5LCJleHAiOjE3MDg20TM4MDl9.dVfflY-vlnjNJRQW1fKdpGidvP1oF7BHGhpnnq0rYRS-AmmHpWCjsYqHcIq700wEizeXvrgW8xy2mX7GcRTmqQ
```

**Decoded** EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
 "alg": "HS512"
}
```

PAYLOAD: DATA

```
{
 "sub": "sauvageboris.pro@tutorial.fr",
 "iat": 1707829809,
 "exp": 1708693809
}
```

VERIFY SIGNATURE

```
HMACSHA512(
 base64UrlEncode(header) + "." +
 base64UrlEncode(payload),
 mypasswordmypassword
) secret base64 encoded
```

SHARE JWT

Signature Verified

# Angular et JWT

- **Angular** est un Framework Javascript pour construire des applications web
  - Les applications Angular s'exécutent dans le navigateur
  - Une session peut être mise en place sur notre application
- L'authentification et l'autorisation des utilisateurs doivent être implémentés
- Pour cela, Angular offre de nombreux mécanismes :
  - Les **Services** pour implémenter le stockage de la session côté client
  - Les **Guards** pour protéger des routes dans la navigation au sein de l'application
  - Les **Intercepteurs** HTTP pour ajouter des en-têtes à chaque requête sortante
  - Les **Resolvers** pour orienter et conditionner l'accès à une route
- Nous allons présenter ces mécanismes dans la suite de la présentation

# TokenStorageService

- Un **Service** peut-être utilisé pour le stockage de la session côté client

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

const TOKEN_KEY = 'auth-token';
const USER_KEY = 'auth-user';

@Injectable({ providedIn: 'root' })
export class TokenStorageService {

 clearSession(): void {
 window.sessionStorage.clear();
 }

 public saveToken(token: string): void {
 window.sessionStorage.removeItem(TOKEN_KEY);
 window.sessionStorage.setItem(TOKEN_KEY, token);
 }

 public getToken(): string | null {
 return window.sessionStorage.getItem(TOKEN_KEY);
 }
}
```

```
public saveUser(user: any): void {
 window.sessionStorage.removeItem(USER_KEY);
 window.sessionStorage.setItem(USER_KEY, JSON.stringify(user));
}

public getUser(): any {
 const user = window.sessionStorage.getItem(USER_KEY);
 if (user) {
 return JSON.parse(user);
 }
 return null;
}
```

# AuthenticationService

- Un **Service** peut-être utilisé pour les fonctionnalités utilisateur

```
@Injectable({ providedIn: 'root' })
export class AuthenticationService {

 login(email: string, password: string): Observable<string> {
 return this.http.post<string>(`${this.baseUrl}/auth/signin`,
 {"email": email, "password": password}, httpOptions)
 .pipe(map((data: any) => {
 this.tokenStorage.saveToken(data.token);
 this.tokenStorage.saveUser(data);
 return this.tokenStorage.getUser();
 })
);
 }

 register(username: string, email: string, password: string): Observable<any> {
 return this.http.post(` ${this.baseUrl}/signup`, {
 username, email, password }, httpOptions);
 }
}
```

```
logout(): void {
 this.tokenStorage.clearSession();
}

isConnected(): boolean {
 return !!this.tokenStorage.getToken();
}

getConnectedUser() {
 return this.tokenStorage.getUser();
}
```

# AuthGuard

- Un **Guard** peut-être utilisé pour sécuriser une route si l'utilisateur n'est pas connecté

```
import {CanActivateFn, Router} from '@angular/router';
import {AuthenticationService} from "../services/authentication.service";
import {inject} from "@angular/core";

export const AuthGuard: CanActivateFn = (route, state) => {
 let authentication = inject(AuthenticationService);
 return authentication.isConnected() || inject(Router).createUrlTree(['/auth/login']);
};
```

```
export const routes: Routes = [
 {path: "", pathMatch: 'full', redirectTo: '/home'},
 {path: 'home', component: CategoriesComponent},
 ...
 {path: 'tutorials/add', component: AddTutorialComponent, canActivate: [AuthGuard]},
 ...
 {path: 'auth/register', component: RegisterComponent},
 {path: '**', redirectTo: '/'}
];
```



# LoginResolver

- Un **Resolver** peut-être utilisé pour sécuriser une route si l'utilisateur n'est pas connecté

```
export const LoginResolver: ResolveFn<boolean> = (route, state) => {
 const authenticationService = inject(AuthenticationService);
 if (authenticationService.isConnected()) {
 inject(Router).navigateByUrl('/home');
 return false;
 }
 return true;
};
```

```
export const routes: Routes = [
 ...
 {
 path: 'auth/login',
 component: LoginComponent, resolve: {
 ready: LoginResolver,
 }
 },
 ...
];
```



# Interceptor

- Un **Interceptor** est une fonction middleware
  - Permet de traiter les requêtes et les réponses HTTP avant qu'elles ne soient envoyées/reçues

```
ng generate interceptor auth
```

```
export const AuthInterceptor: HttpInterceptorFn = (req, next) => {

 let tokenStorageService = inject(TokenStorageService);
 let userToken = tokenStorageService.getToken();

 const modifiedReq = req.clone({
 headers: req.headers.set('Authorization', `Bearer ${userToken}`),
 });

 return next(modifiedReq);
};
```



```
export const appConfig: ApplicationConfig = {
 providers: [
 ...
 provideHttpClient(withInterceptors([AuthInterceptor])),
 ...
]
};
```

# Tests Unitaires

Développement Web



# Introduction 1/2

- **Angular** permet l'écriture de tests rapidement et simplement
  - **Jasmine** et **Karma** sont les librairies en charge des tests unitaires
  - **Protractor** est en charge des tests de bout en bout
- **Tester une application** c'est vérifier qu'à partir de données connues, le comportement correspond parfaitement à ce qui était attendu
- **Un test unitaire** : permet de vérifier une portion d'un élément isolé
  - méthodes d'un composant, d'un service, d'un pipe, d'une balise HTML
- **Un test unitaire** ne vérifie pas les dépendances
  - Il faut simuler le résultat d'appel des dépendances afin de ne pas biaiser le test
- **Principe d'isolation** : La modification d'un service ne doit jamais faire échouer les tests unitaires d'un composant qui en dépend. Ce seront les tests unitaires du service lui-même qui seront impactés

# Introduction 2/2

- **FIRST** désigne les grands principes d'un tests unitaire
  - Fast (rapide)
  - Isolated (isolé)
  - Repeatable (répétable)
  - Self validating (auto-validation)
  - Thorough (couvrant tous les scénarios de l'élément testé)
- **Jasmine** permet d'écrire les tests unitaires
- **Karma** permet de lancer un navigateur pour exécuter tous les tests
- Une interface web permet de vérifier en temps réel les tests qui ont échoué et chaque modification du code dans l'IDE relance les tests dans le navigateur instantanément

# Jasmine

- Angular recherche les tests unitaires dans les fichiers se terminant par spec.ts
- Le fichier de tests unitaires de la classe TypeScript **maClasse.ts** est **maClasse.spec.ts**
- Jasmine possède plusieurs méthodes pour une bonne écriture de tests
  - **describe()** : description de l'objet à tester
  - **beforeEach / afterEach** : exécution de code avant / après chaque test
  - **it()** : bloc de description de la fonctionnalité à tester
  - **expect()** : évaluation du cas à tester
- Jasmine permet également de mocker des objets ou des méthodes
  - **spyOn** : mock de la méthode d'un objet
  - **createSpyObj** : mock d'un objet dans son intégralité

```
describe(description, () => {
 it(nomDuTest, () => {
 expect().methodeJasmine();
 });
});
```

# Jasmine

- Jasmine propose de nombreux paramètre de comparaison :
  - **toBe** : s'attend à trouver une égalité stricte (égalité de valeurs et de types)
  - **toEqual** : s'attend à trouver une égalité non stricte (égalité de valeurs)
  - **toContain** : s'attend à trouver un élément à l'intérieur d'un tableau ou d'une chaîne de caractères
  - **toBeDefined** : s'attend à trouver un objet défini
  - **toBeNull** : s'attend à trouver une valeur de l'objet nulle
  - **toBeTruthy** : s'attend à trouver une valeur de l'objet vraie
  - **toBeFalsy** : s'attend à trouver une valeur de l'objet fausse
  - **toHaveBeenCalled** : s'attend à ce que la méthode ait bien été appelée
  - **toHaveBeenCalledWith** : s'attend à ce que la méthode ait bien été appelée avec les paramètres requis
  - **not** : permet de s'attendre au contraire. **not.toBeTruthy** est donc le même test que **toBeFalsy**

```
describe(description, () => {
 it(nomDuTest, () => {
 expect().toBeTruthy();
 });
});
```

# Tests unitaire basique

```
import {Calculatrice} from './calculatrice';

describe('Calculatrice', () => {
 it('doit retourner l\'addition des deux nombres passés en paramètres', () => {
 const calculatrice: Calculatrice = new Calculatrice();
 expect(calculatrice.addition(7, 1)).toEqual(8);
 });
 it('doit retourner la soustraction des deux nombres passés en paramètres', () => {
 const calculatrice: Calculatrice = new Calculatrice();
 expect(calculatrice.soustraction(7, 1)).toEqual(6);
 });
 it('doit retourner la multiplication des deux nombres passés en paramètres', () => {
 const calculatrice: Calculatrice = new Calculatrice();
 expect(calculatrice.multiplication(7, 1)).toEqual(7);
 });
 it('doit retourner la division des deux nombres passés en paramètres', () => {
 const calculatrice: Calculatrice = new Calculatrice();
 expect(calculatrice.division(7, 1)).toEqual(7);
 });
});
```

ng generate class classes/calculatrice

```
export class Calculatrice {

 addition(var1: number, var2: number) {
 return var1 + var2;
 }

 soustraction(var1: number, var2: number) {
 return var1 - var2;
 }

 multiplication(var1: number, var2: number) {
 return var1 * var2;
 }

 division(var1: number, var2: number) {
 return var1 / var2;
 }
}
```

# Tests unitaire basique

- Jasmine propose plusieurs astuces utiles lors de la rédaction de tests
  - Écrire **fit** au lieu de **it** permet d'indiquer à Jasmine de lancer uniquement ce test précis
  - Écrire **xit** au lieu de **it** indique, au contraire, de ne pas le lancer
  - C'est utilisable avec **xdescribe()** ou **fdescribe()**

```
xdescribe('Calculatrice', () => {
 xit('doit retourner l\'addition des deux nombres passés en paramètres', () => {
 expect(calculatrice.addition(7, 1)).toEqual(8);
 });
});
```

- Jasmine introduit le concept de **fixture** grâce à la méthode **beforeEach()**
  - L'ensemble des instructions de cette méthode sera exécuté avant chaque test

```
// Définition d'une calculatrice
let calculatrice: Calculatrice;

// Avant chaque test
beforeEach(() => {
 calculatrice = new Calculatrice();
 console.log('Chargement de la calculatrice avant chaque test');
});
```

# Lancement des tests

- Angular CLI permet de lancer :
  - les tests unitaires
  - Karma afin de visualiser les résultats dans un navigateur web



- Après la compilation, **Jasmine** affiche le résultat des tests dans la console

```
LOG: 'Chargement de la calculatrice avant chaque test'
Chrome 81.0.4044 (Mac OS X 10.15.1): Executed 0 of 25 SUCCESS (0 secs / 0 secs)
LOG: 'Chargement de la calculatrice avant chaque test'
Chrome 81.0.4044 (Mac OS X 10.15.1): Executed 1 of 25 SUCCESS (0 secs / 0.006 secs)
LOG: 'Chargement de la calculatrice avant chaque test'
Chrome 81.0.4044 (Mac OS X 10.15.1): Executed 2 of 25 SUCCESS (0 secs / 0.007 secs)
LOG: 'Chargement de la calculatrice avant chaque test'
Chrome 81.0.4044 (Mac OS X 10.15.1): Executed 3 of 25 SUCCESS (0 secs / 0.008 secs)
Chrome 81.0.4044 (Mac OS X 10.15.1): Executed 4 of 25 (skipped 21) SUCCESS (0.053 secs / 0.009 secs)
TOTAL: 4 SUCCESS
TOTAL: 4 SUCCESS
```

- **Karma** montre une version web de la même chose



# Lancement des tests

- Par défaut, l'option **watch** est à vrai
- `ng test`
- Karma garde le navigateur web ouvert pour pouvoir effectuer des modifications dans les tests et visualiser les résultats instantanément
  - Pour lancer les tests, refermer le navigateur et voir le résultat uniquement sur la console, il faut spécifier à Jasmine de ne pas surveiller en continu les modifications

`ng test --watch=false`

# Tests unitaires d'un composant

- Pour tester un composant Angular, une librairie appelée **@angular/core/testing** est utilisée
- La méthode **TestBed.configureTestingModule( { } )** est comparable au décorateur **@NgModule()** du fichier **app.module.ts**
  - Elle prend en paramètre un objet composé des attributs **imports, declarations, providers et schemas**
- La première chose à tester est de savoir si le composant a réussi à s'instancier
  - **TestBed** permet d'instancier un composant grâce à la méthode **createComponent(unComposant)**

```
ComponentFixture<ExampleComponent> fixture = TestBed.createComponent(ExampleComponent);
```

- Les principales propriétés et méthodes de cette classe sont les suivantes :
  - **componentInstance** : l'instance de la classe ExampleComponent
  - **debugElement** : objet permettant d'inspecter et de manipuler le DOM
  - **detectChanges()** : déclenche la Change Detection

# Tests unitaires d'un composant

```
import {ComponentFixture, TestBed} from '@angular/core/testing';

describe('Exemple de composant à tester', () => {
 let composant: ExampleComponent;
 let fixture: ComponentFixture<ExampleComponent>;

 beforeEach(() => {
 // Le composant est configuré et compilé
 TestBed.configureTestingModule({
 imports: [HttpClientModule],
 declarations: [ExampleComponent],
 providers: [],
 schemas: []
 }).compileComponents();
 // Le composant est créé
 fixture = TestBed.createComponent(ExampleComponent);
 // L'instance du composant est dans une variable
 composant = fixture.componentInstance;
 });
);

fit('doit être créé', () => {
 // La variable est à vrai si le composant a bien été créé
 expect(composant).toBeTruthy();
});
```

```
MBP-de-Boris:first-project-angular sauvage$ ng test --watch=false
10% building 2/2 modules 0 active15 05 2020 08:18:56.876:INFO [karma-server]: Karma v4.4.1 server started at http://0.0.0.0:9876/
15 05 2020 08:18:56.878:INFO [launcher]: Launching browsers Chrome with concurrency unlimited
15 05 2020 08:18:56.881:INFO [launcher]: Starting browser Chrome
15 05 2020 08:19:00.941:INFO [Chrome 81.0.4044 (Mac OS X 10.15.1)]: Connected on socket hXj5gS2DGkWYP50fAAAA with id 39978414
LOG: 'Chargement de la calculatrice avant chaque test'
Chrome 81.0.4044 (Mac OS X 10.15.1): Executed 0 of 25 SUCCESS (0 secs / 0 secs)
LOG: 'Chargement de la calculatrice avant chaque test'
Chrome 81.0.4044 (Mac OS X 10.15.1): Executed 1 of 25 SUCCESS (0 secs / 0.003 secs)
LOG: 'Chargement de la calculatrice avant chaque test'
Chrome 81.0.4044 (Mac OS X 10.15.1): Executed 2 of 25 SUCCESS (0 secs / 0.004 secs)
LOG: 'Chargement de la calculatrice avant chaque test'
Chrome 81.0.4044 (Mac OS X 10.15.1): Executed 3 of 25 SUCCESS (0 secs / 0.005 secs)
LOG: 'Chargement de la calculatrice avant chaque test'
Chrome 81.0.4044 (Mac OS X 10.15.1): Executed 4 of 25 SUCCESS (0 secs / 0.005 secs)
Chrome 81.0.4044 (Mac OS X 10.15.1): Executed 5 of 25 (skipped 20) SUCCESS (0.079 secs / 0.038 secs)
TOTAL: 5 SUCCESS
TOTAL: 5 SUCCESS
```

# Tests unitaires d'un template 1/4

- **@angular/core/testing** possède une interface **debugElement** qui permet d'interagir avec l'ensemble des éléments HTML du template
  - Celle-ci peut récupérer un élément par son id, sa classe CSS ou sa balise
- Récupération d'un élément HTML :

```
unElementHTML = fixture.nativeElement.querySelector('balise');
unElementHTML = fixture.nativeElement.querySelector('.css');
unElementHTML = fixture.nativeElement.querySelector('#id');
```

- Pour interagir avec le template, il faut utiliser l'objet **DebugElement** :

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

# Tests unitaires d'un template 2/4

- Test avec un composant et sa vue html :

```
import {Component, OnInit} from '@angular/core';
import {ExampleService} from '../services/example.service';

@Component({
 selector: 'app-example',
 templateUrl: './example.component.html',
 styleUrls: ['./example.component.css']
})
export class ExampleComponent implements OnInit {

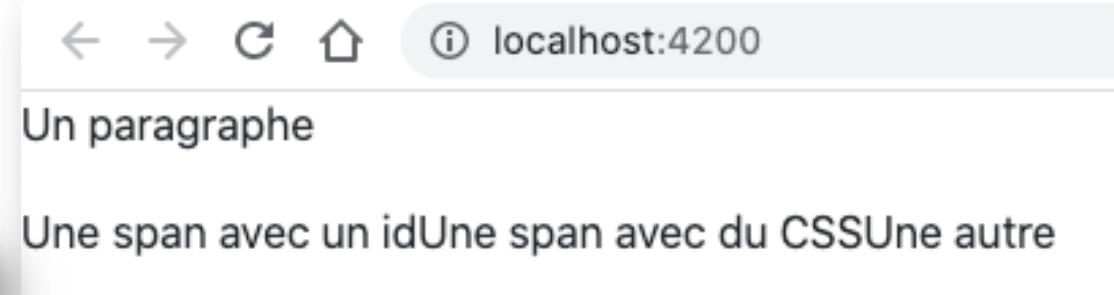
 constructor(private exampleService: ExampleService) { }

 ngOnInit(): void {
 this.exampleService.exampleGET().subscribe(data => {
 console.log(data.age);
 });
 }
}
```

 example.component.ts

```
<div>
 <p>Un paragraphe</p>
 Une span avec un id
 Une span avec du CSS
 Une autre
</div>
```

 example.component.html



# Tests unitaires d'un template 3/4

```
import {ExampleComponent} from './example.component';
import { ComponentFixture, TestBed} from '@angular/core/testing';
import {DebugElement} from '@angular/core';
import {HttpClientModule} from '@angular/common/http';

describe('Example Component', () => {
 let component: ExampleComponent;
 let fixture: ComponentFixture<ExampleComponent>;
 let debugElement: DebugElement;

 beforeEach(() => {
 TestBed.configureTestingModule({
 imports: [HttpClientModule],
 declarations: [ExampleComponent],
 providers: [],
 schemas: []
 }).compileComponents();
 fixture = TestBed.createComponent(ExampleComponent);
 component = fixture.componentInstance;
 // Activation des changements d'une instance de DebugElement
 fixture.detectChanges();
 debugElement = fixture.debugElement;
 console.log(debugElement);
 });
});
```

```
it('doit être créé', () => {
 expect(component).toBeTruthy();
});

fit('doit renvoyer la valeur du paragraphe', () => {
 const p = debugElement.nativeElement.querySelector('p');
 expect(p.textContent).toEqual('Un paragraphe');
});

fit('doit renvoyer la valeur de l\'élément par son id', () => {
 const parId = debugElement.nativeElement.querySelector('#uneSpan');
 expect(parId.textContent).toEqual('Une span avec un id');
});

fit('doit renvoyer la valeur de l\'élément par sa classe CSS', () => {
 const parCss = debugElement.nativeElement.querySelector('.uneClasseCSS');
 expect(parCss.textContent).toEqual('Une span avec du CSS');
});
```



# Tests unitaires d'un template 4/4

- Il est également possible de récupérer les attributs :

```
fit('doit renvoyer la valeur de l\'élément par sa balise img', () => {
 const img = debugElement.nativeElement.querySelector('img');
 expect(img).not.toBeNull();
 expect(img.getAttribute('src')).toEqual('https://img.icons8.com/officel/50/000000/test-passed.png');
});
```

```
<div>
 <p>Un paragraphe</p>
 Une span avec un id
 Une span avec du CSS
 Une autre

</div>
```



# Dépendances

- Pour être testé, les composants ont parfois besoin des données provenant d'autres services
- **Jasmine permet de mocker facilement des services :**
  - `spyOn` : créer un espion sur un objet existant
  - `jasmine.createSpy` : créer une fonction testable
  - `jasmine.createSpyObj` : créer un objet avec des fonctions d'espions interne
- À présent, peu importe le retour du service, il est possible de tester les méthodes de nos composants, qui appellent des méthodes de nos services
  - Une bonne pratique consiste à vérifier que la méthode du service a bien été appelée
- Nous allons faire un exemple avec un composant et un service

# Exemple : dépendances

```
@Component({
 selector: 'app-example',
 templateUrl: './example.component.html',
 styleUrls: ['./example.component.css']
})
export class ExampleComponent implements OnInit {

 constructor(private exampleService: ExampleService) {}

 ngOnInit(): void {
 this.exampleService.exampleGET().subscribe(data => {
 console.log(data.age);
 });
 }

 clique() {
 return this.exampleService.methodeExample();
 }
}
```

```
@Injectable({
 providedIn: 'root'
})
export class ExampleService {

 private url = 'https://api.agify.io/';

 constructor(private http: HttpClient) {}

 exampleGET(): Observable<any> {
 const params = new HttpParams().set('name', 'android').set('country_id', 'FR');
 return this.http.get(this.url, {params});
 }

 methodeExample() {
 return 'Retour d\'exemple';
 }
}
```

- Nous allons **mocké** la **methodeExample** de notre service **ExampleService**

# Exemple : dépendances

```
fdescribe('Example Component', () => {

 let service: ExampleService;

 beforeEach(() => {
 TestBed.configureTestingModule({
 imports: [HttpClientModule],
 declarations: [ExampleComponent],
 providers: [ExampleService],
 schemas: []
 }).compileComponents();

 // Récupération du service dans une variable
 service = TestBed.inject(ExampleService);
 });

 fit('doit appeler un service', () => {
 spyOn(service, 'methodeExample').and.returnValue('exemple de retour');

 expect(component клик()).toBe('exemple de retour');

 expect(service.methodeExample).toHaveBeenCalled();
 expect(service.methodeExample).toHaveBeenCalledTimes(1);
 });
});
```

# Karma

- **Configuré par défaut** pour fonctionner avec Jasmine lors de la création de l'application par Angular CLI
  - Karma laisse la possibilité au développeur de changer la librairie de test si Jasmine ne convient pas
  - Karma s'inscrit parfaitement dans un cycle de développement d'applications en intégration continue
- Fichier de configuration de Karma : **karma.conf.js**
  - contient les plug-ins utilisés, les paramètres pour la couverture de code et l'ensemble des paramètres nécessaires au lancement d'un serveur et navigateur web

## Quand utiliser Karma ?

- Lorsqu'on veut exécuter notre application dans de vrais navigateurs
- Lorsqu'on veut tester notre application dans plusieurs navigateurs (bureau, mobile et tablette)
- Lorsque vous voulez exécuter vos tests localement, lors du développement
- Lorsque vous désirez exécuter vos tests sur un serveur d'intégration continue
- Lorsque vous voulez exécuter vos tests automatiquement quand vous sauvegardez un fichier

# Karma : Lancer sur x navigateurs

- Karma peut lancer un ou plusieurs navigateurs web en même temps
- Les valeurs possibles sont **Chrome, Firefox, Opera, IE, Safari...** Mais des plug-ins supplémentaires sont à ajouter :

```
npm install karma-firefox-launcher --save
npm install karma-opera-launcher --save
npm install karma-chrome-launcher --save
npm install karma-ie-launcher --save
npm install karma-safari-launcher --save
```

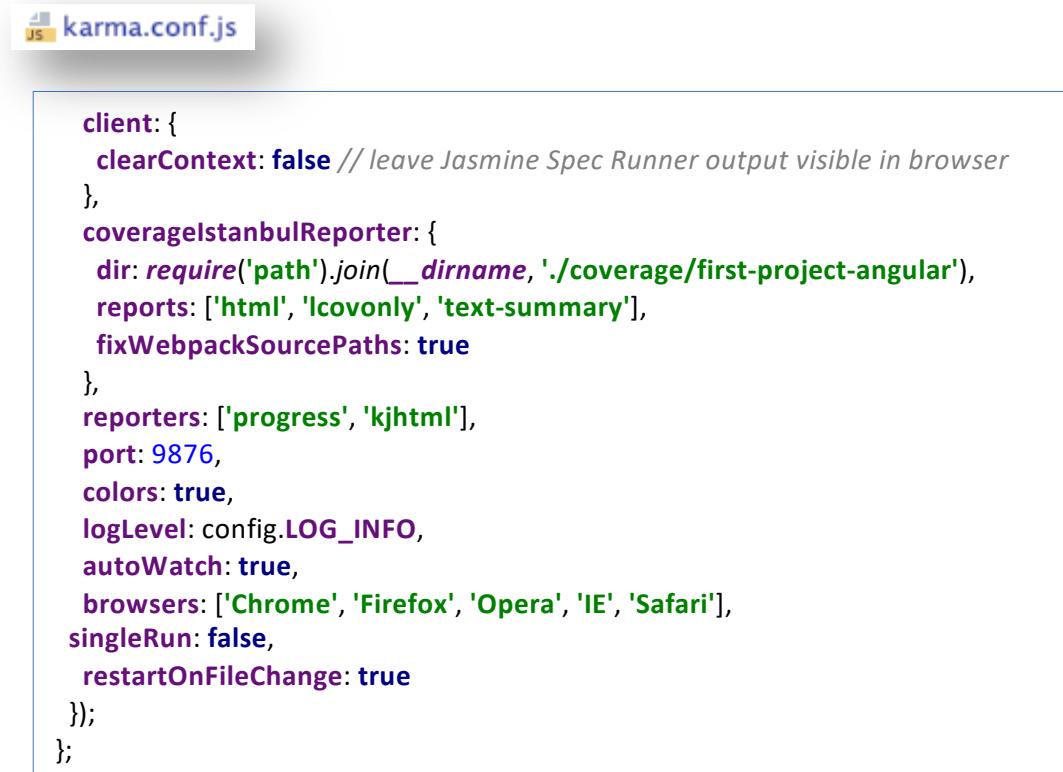
- Une fois les dépendances ajoutées, il est nécessaire d'ajouter les plugins dans la configuration de Karma :

```
module.exports = function (config) {
 config.set({
 basePath: '',
 frameworks: ['jasmine', '@angular-devkit/build-angular'],
 plugins: [
 require('karma-jasmine'),
 require('karma-chrome-launcher'),
 require('karma-firefox-launcher'),
 require('karma-opera-launcher'),
 require('karma-ie-launcher'),
 require('karma-safari-launcher'),
],
 browsers: ['Chrome', 'Firefox', 'Opera', 'IE', 'Safari'],
 })
}
```

# Karma : Lancer sur x navigateurs

- Le fichier Karma ressemble désormais à cela :

```
// Karma configuration file, see link for more information
// https://karma-runner.github.io/1.0/config/configuration-file.html
module.exports = function (config) {
 config.set({
 basePath: '',
 frameworks: ['jasmine', '@angular-devkit/build-angular'],
 plugins: [
 require('karma-jasmine'),
 require('karma-chrome-launcher'),
 require('karma-firefox-launcher'),
 require('karma-opera-launcher'),
 require('karma-ie-launcher'),
 require('karma-safari-launcher'),
 require('karma-jasmine'),
 require('karma-chrome-launcher'),
 require('karma-jasmine-html-reporter'),
 require('karma-coverage-istanbul-reporter'),
 require('@angular-devkit/build-angular/plugins/karma')
],
 });
}
```



A screenshot of a code editor showing a file named "karma.conf.js". The file contains configuration code for Karma. The code includes settings for the client (such as clearContext: false, coverageIstanbulReporter, reporters, port, colors, logLevel, autoWatch, browsers, singleRun, and restartOnFileChange), and a section for frameworks and plugins.

```
client: {
 clearContext: false // leave Jasmine Spec Runner output visible in browser
},
coverageIstanbulReporter: {
 dir: require('path').join(__dirname, './coverage/first-project-angular'),
 reports: ['html', 'lcovonly', 'text-summary'],
 fixWebpackSourcePaths: true
},
reporters: ['progress', 'kjhtml'],
port: 9876,
colors: true,
logLevel: config.LOG_INFO,
autoWatch: true,
browsers: ['Chrome', 'Firefox', 'Opera', 'IE', 'Safari'],
singleRun: false,
restartOnFileChange: true
});
```

- Les navigateurs doivent être présent sur la machine qui exécutent les tests

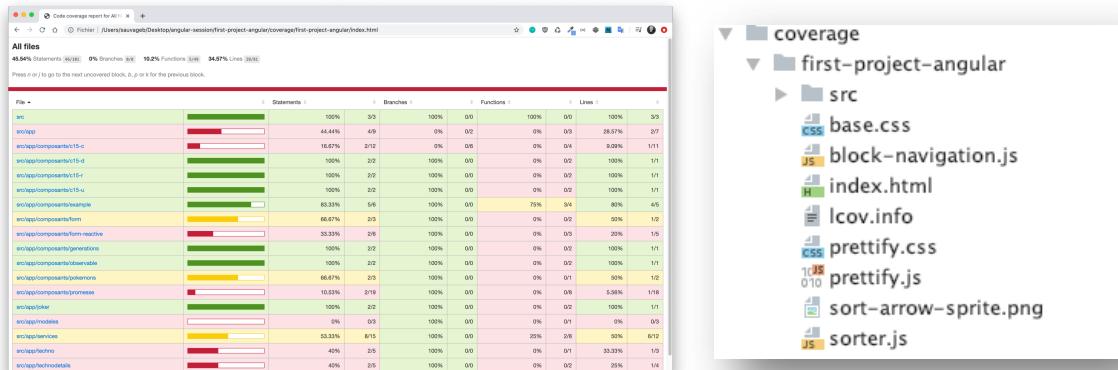
# Couverture de code

- Déterminer le pourcentage de code couvert par les tests unitaires :

```
ng test --code-coverage --watch=false
```

- Cette commande produit deux éléments :
  - un tableau dans la console qui a lancé la commande
  - un site web généré dans un dossier **coverage** de votre projet

```
===== Coverage summary =====
Statements : 45.54% (46/101)
Branches : 0% (0/8)
Functions : 10.2% (5/49)
Lines : 34.57% (28/81)
```



# Configuration de la couverture de code

- Il est possible de naviguer sur le site internet pour visualiser quelles fonctions, quels paramètres ou bouts de code ne sont pas correctement testés
  - Tests unitaires
- Définir un minimum de couverture de code est possible, au sein du fichier  karma.conf.js

```
coverageIstanbulReporter: {
 dir: require('path').join(__dirname, './coverage/first-project-angular'),
 reports: ['html', 'lcovonly', 'text-summary'],
 fixWebpackSourcePaths: true,
 thresholds: {
 statements: 80,
 lines: 80,
 branches: 80,
 functions: 80
 }
},
```

- Ici, **ng test** produira une erreur si la couverture de code n'est pas d'au moins **80 %** sur les quatre éléments du rapport

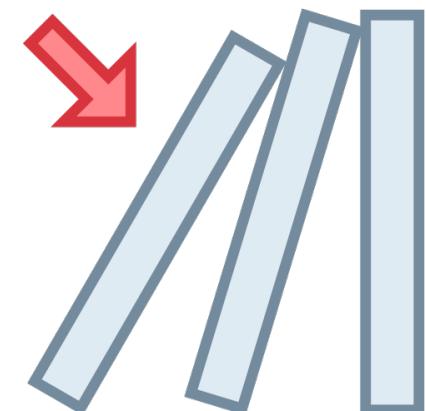
# Exercice

À partir de l'application développée dans les exercices précédents :

- Faites en sorte d'atteindre une couverture de code de 80 %

# Tests de bout en bout

Développement Web



# Introduction 1/2

- Un test de bout en bout est l'inverse d'un test unitaire
  - Le but est de vérifier l'application du début jusqu'à la fin, en passant par toutes les étapes possibles
  - L'ensemble des cas d'utilisation devant être testé
- Historiquement, un projet Angular intégrait le Framework Protractor
  - Depuis la version Angular 12 (mai 2021), Angular annonce l'abandon de celui-ci
  - Les développements de Protractor seront arrêtés fin 2022 (~Angular 15)
  - <https://github.com/angular/protractor/issues/5502>
- Désormais, nous devrons choisir un Framework :
  - Cypress
  - Nightwatch
  - WebdriverIO

# Introduction 2/2

- Afin de lancer les tests, la commande est `ng e2e`
- Une fois exécutée, la commande affiche les informations ci-dessous :

```
Cannot find "e2e" target for the specified project.
You should add a package that implements end-to-end testing capabilities.
```

For example:

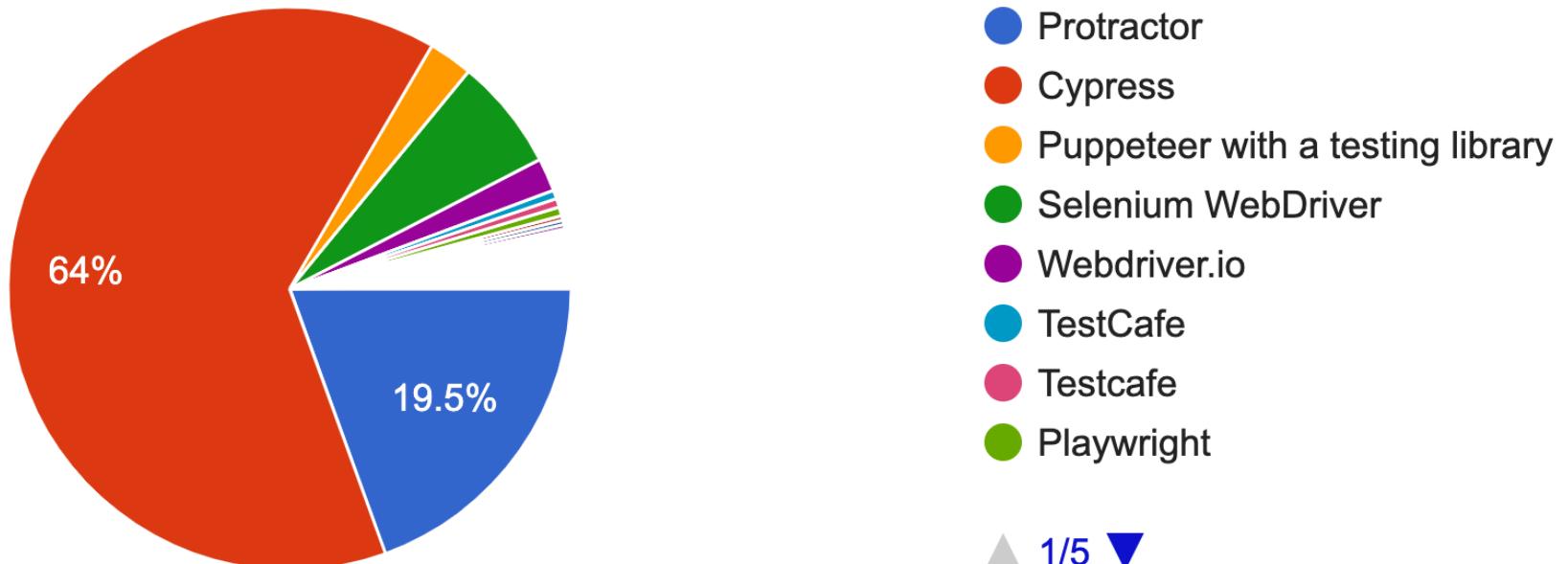
```
Cypress: ng add @cypress/schematic
Nightwatch: ng add @nightwatch/schematics
WebdriverIO: ng add @wdio/schematics
```

More options will be added to the list as they become available.

- Nous présenterons Cypress

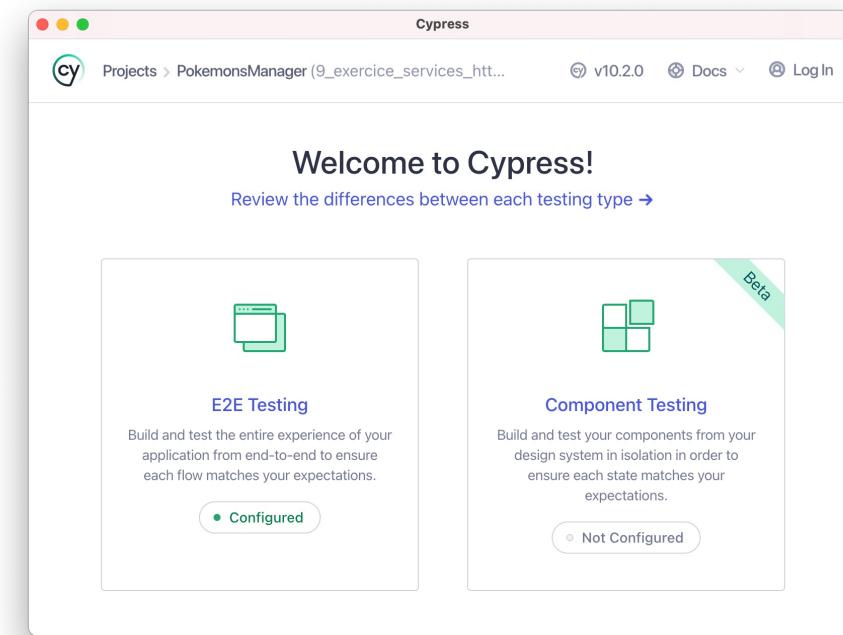
# Enquête Angular janvier 2021

What e2e testing tools do you use?



# Cypress

- Cypress est un outil opensource, d'automatisation de test pour le frontend
- Il permet d'écrire tout type de tests :
  - Tests de bout en bout
  - Test d'intégration
  - Tests unitaires
- Installation avec Angular CLI :  
`ng add @cypress/schematic`



# Installation 1/2

- Une fois installé, Cypress est automatiquement intégré à Angular

```
{
 "name": "angular-project",
 "scripts": {
 ...
 "test": "ng test",
 "e2e": "ng e2e",
 "cypress:open": "cypress open",
 "cypress:run": "cypress run"
 },
}
```

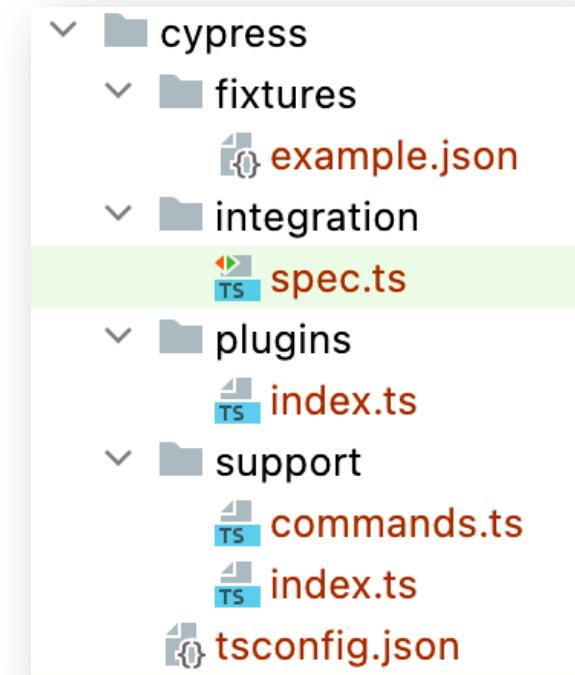


package.json

```
{
 "integrationFolder": "cypress/integration",
 "supportFile": "cypress/support/index.ts",
 "videosFolder": "cypress/videos",
 "screenshotsFolder": "cypress/screenshots",
 "pluginsFile": "cypress/plugins/index.ts",
 "fixturesFolder": "cypress/fixtures",
 "baseUrl": "http://localhost:4200"
}
```



cypress.json



# Installation 2/2

- Le fichier cypress.json définit la configuration de Cypress

```
{
 "integrationFolder": "cypress/integration",
 "supportFile": "cypress/support/index.ts",
 "videosFolder": "cypress/videos",
 "screenshotsFolder": "cypress/screenshots",
 "pluginsFile": "cypress/plugins/index.ts",
 "fixturesFolder": "cypress/fixtures",
 "baseUrl": "http://localhost:4200"
}
```



## Exemple :

Il est possible de définir un port fixe pour le lancement des tests

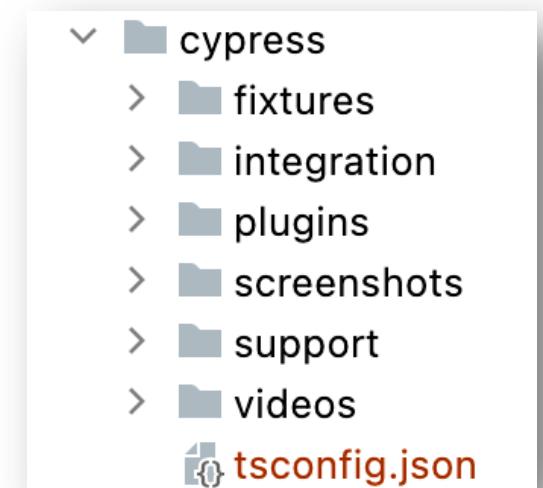
- Il est aléatoire par défaut

- <https://docs.cypress.io/guides/references/configuration#cypress-json>

# Les dossiers 1/2

Une fois intégré, Cypress est composé de plusieurs dossiers :

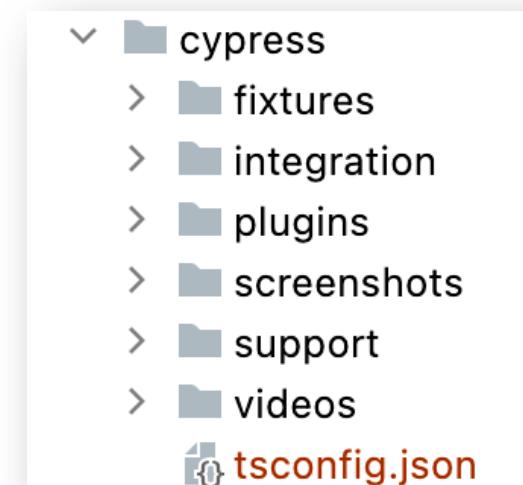
- **fixtures**
  - contient des collections de données pour mocker, accessible avec `cy.intercept`
- **integration**
  - contient les tests d'intégration
- **plugins**
  - Permet d'étendre des fonctionnalités avec des listeners/fonctions
  - <https://docs.cypress.io/api/plugins/configuration-api#Usage>
  - <https://docs.cypress.io/api/plugins/writing-a-plugin#Plugins-API>



# Les dossiers 2/2

Une fois intégré, Cypress est composé de plusieurs dossiers :

- **screenshots**
  - contient les impressions d'écrans des tests exécutés (avec cypress run)
- **support**
  - Permet d'ajouter des fonctions personnalisées comme `cy.login()`
- **videos**
  - contient les vidéos des tests exécutés (avec cypress run)



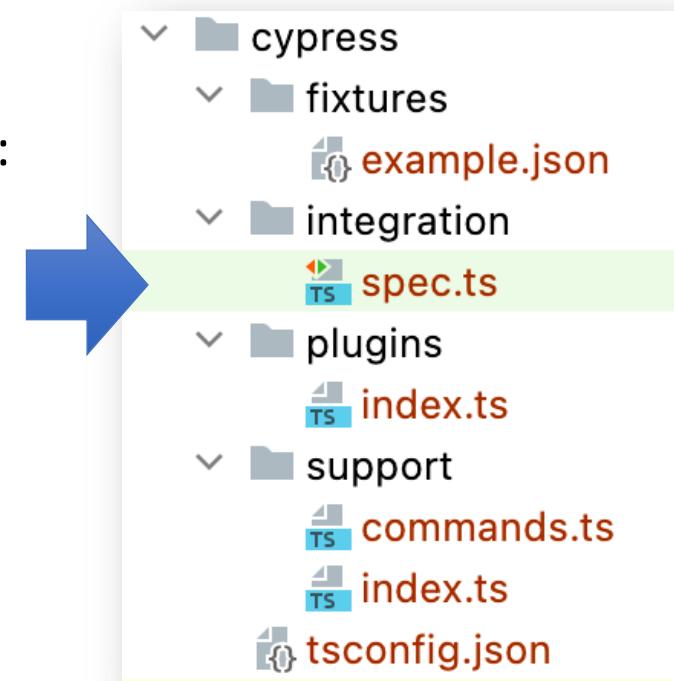
# Focus sur les tests

- Un test par défaut est créé dans le dossier integration :

```
• describe('My First Test', () => {
 it('Visits the initial project page', () => {
 cy.visit('/')
 cy.contains('Welcome')
 cy.contains('sandbox app is running!')
 })
})
```

- La variable **cy** est Cypress
- La méthode **visit** permet d'ouvrir la racine du site
- La méthode **contains** permet de vérifier que la page contient un élément

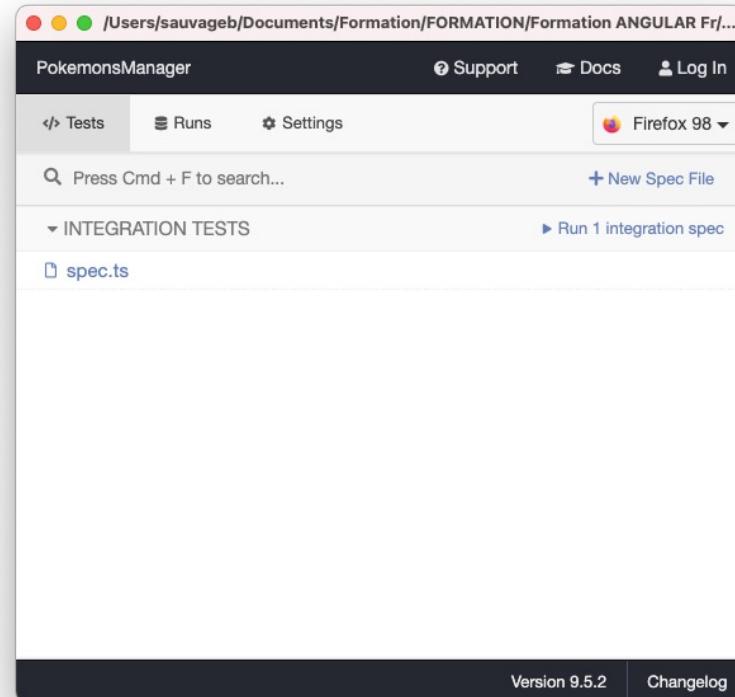
- Il est possible de créer plusieurs test dans un seul fichier, avec **it()**



# Lancement graphique 1/2

- Les tests sont exécutables grâce à la commande **ng e2e**

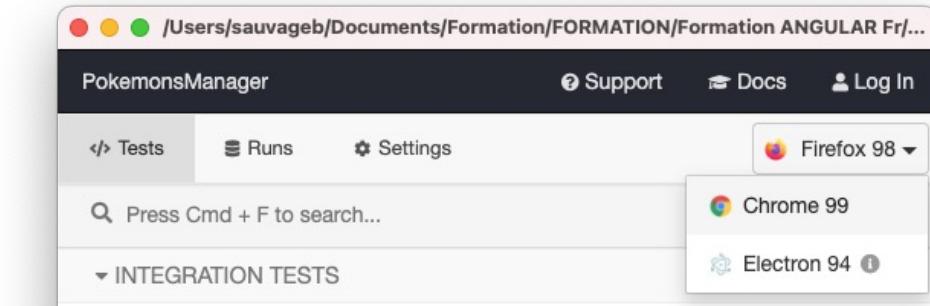
ng e2e



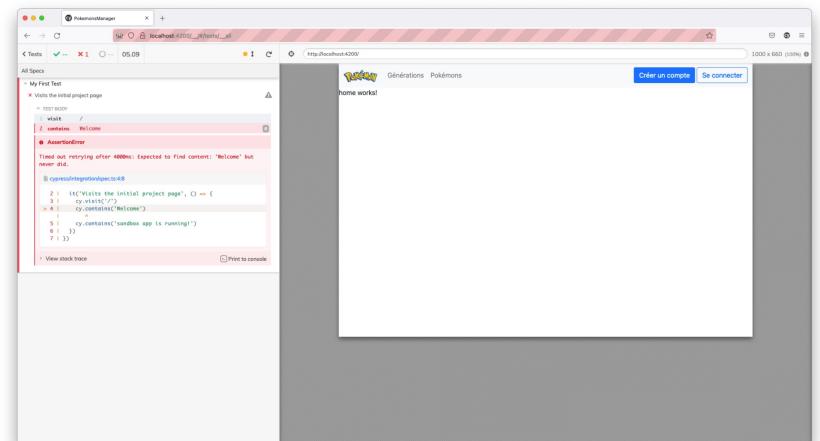
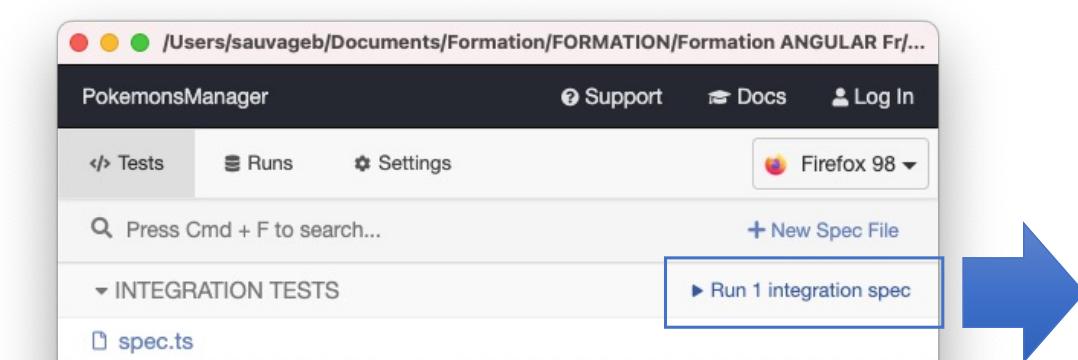
- Cypress s'ouvre et affiche la liste de vos tests

# Lancement graphique 2/2

- Les tests sont exécutables sur différents navigateurs, via une liste déroulante



- Cypress lance ensuite les tests dans le navigateur choisi



# Lancement en console 1/2

- Les tests peuvent aussi être exécuté dans la console avec **cypress run**

npx cypress run

→

```
Terminal: Local (2) +
MacBook-Pro-de-Boris:PokemonsManager sauvageb$ npx cypress run

=====
(Run Starting)

Cypress: 9.5.2
Browser: Electron 94 (headless)
Node Version: v16.13.1 (/usr/local/bin/node)
Specs: 1 found (spec.ts)

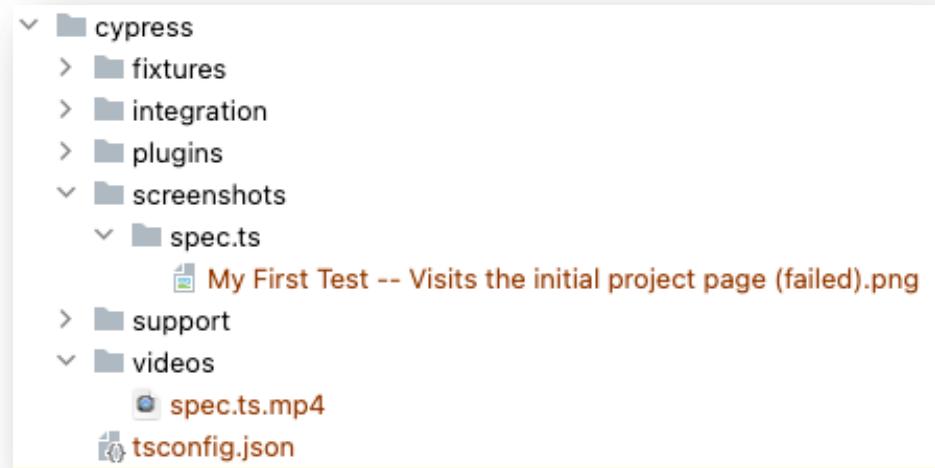
=====
Running: spec.ts (1 of 1)

My First Test
 1) Visits the initial project page

 0 passing (5s)
 1 failing
```

# Lancement en console 2/2

- Les résultats sont disponibles sous 3 formats différents :
  - **Console**
  - **Photo**
  - **Vidéo**

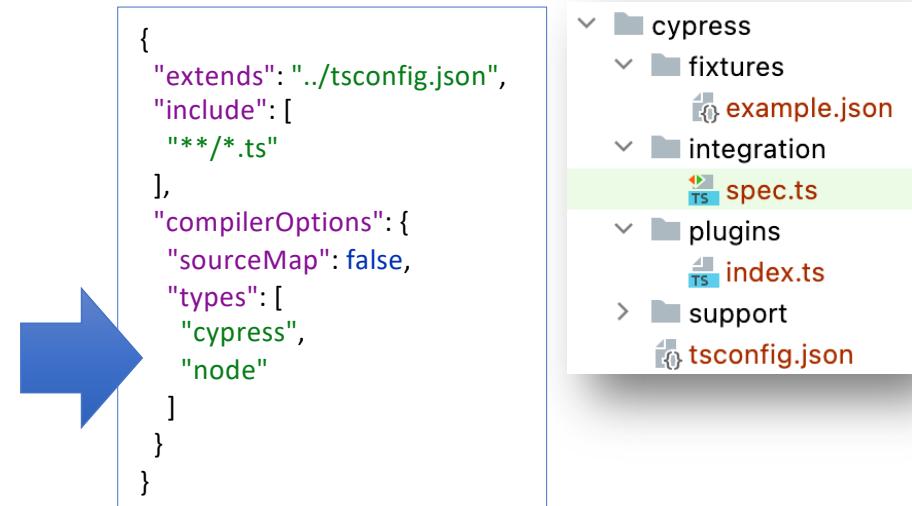


# Plugins 1/2

Initialement, le dossier **plugins/index.ts** contient des erreurs

Correction du fichier **plugins/index.ts** :

1. Ajoutez **node** dans les types du tsconfig.json



2. Installez les `@types/node` :

```
npm i -save-dev @types/node
```

3. Rajouter le typage any

```
module.exports = (on: any, config: any) => {}
```

# Plugins 2/2

- Le fichier plugin/index.ts permet d'étendre l'exécution Cypress
  - <https://docs.cypress.io/api/plugins/writing-a-plugin#Plugins-API>

Exemple :

```
// Plugins enable you to tap into, modify, or extend the internal behavior of Cypress
// For more info, visit https://on.cypress.io/plugins-api
module.exports = (on: any, config: any) => {

 on('before:run', (arg1: any, arg2: any) => {
 console.log('=====');
 console.log('== PLUGIN : BEFORE RUN TEST ==');
 console.log('=====\\n');
 })

 on('after:run', (arg1: any, arg2: any) => {
 console.log('=====');
 console.log('== PLUGIN : AFTER RUN TEST ==');
 console.log('=====\\n');
 })
}
```

MacBook-Pro-de-Boris:PokemonsManager sauvageb\$ npx cypress run

```
=====
(Run Starting)
=====
Cypress: 9.5.2
Browser: Electron 94 (headless)
Node Version: v16.13.1 (/usr/local/bin/node)
Specs: 1 found (spec.ts)

=====
== PLUGIN : BEFORE RUN TEST ==
=====
```

# Integration 1/2

- le dossier **integration** contient les fichiers de tests de votre application
- Par défaut, il y a un fichier **spec.ts**

```
describe('My First Test', () => {
 it('Visits the initial project page', () => {
 cy.visit('/')
 cy.contains('Welcome')
 cy.contains('sandbox app is running!')
 })
})
```

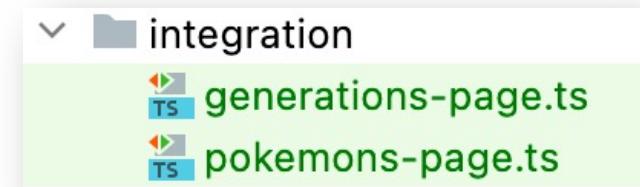


# Integration 2/2

- Afin de tester nos pages, il est possible de créer de nouveaux fichiers :

```
describe('Generations Page', () => {
 it('Visits the generations page', () => {
 cy.visit('/generations')
 cy.contains('Les différentes générations :')
 })
})
```

```
describe('Pokemons Page', () => {
 it('Visits the Pokemons page', () => {
 cy.visit('/pokemons')
 cy.title().should('eq', 'PokemonsManager')
 cy.contains('Les 20 premiers Pokémon :')
 cy.contains('Dans l\'ordre du Pokédex :')
 cy.contains('Aucune sélection de pokémon')
 })
})
```



# It.skip & It.only 1/2

- Il est possible d'ignorer un test avec **it.skip()**

```
describe('Pokemons Page', () => {

 it('Visits the Pokemons page', () => {
 cy.visit('/pokemons')
 cy.title().should('eq', 'PokemonsManager')
 cy.contains('Les 20 premiers Pokémon :)')
 cy.contains('Dans l\'ordre du Pokédex :)')
 cy.contains('Aucune sélection de pokémon')
 })

 it.skip('Search specific pokemon return result', () => {
 console.log('test in progress');
 })
})
```



# It.skip & It.only 2/2

- Il est possible d'exécuter un seul test dans le fichier avec **it.only()**

```
describe('Pokemons Page', () => {
 it('Visits the Pokemons page', () => {
 cy.visit('/pokemons')
 cy.title().should('eq', 'PokemonsManager')
 cy.contains('Les 20 premiers Pokémon :')
 cy.contains('Dans l\'ordre du Pokédex :')
 cy.contains('Aucune sélection de pokémon')
 })

 it('Search specific pokemon return result', () => {
 console.log('test 1 in progress');
 })

 it.only('Search unknown pokemon no return result', () => {
 console.log('test 2 in progress');
 })
})
```



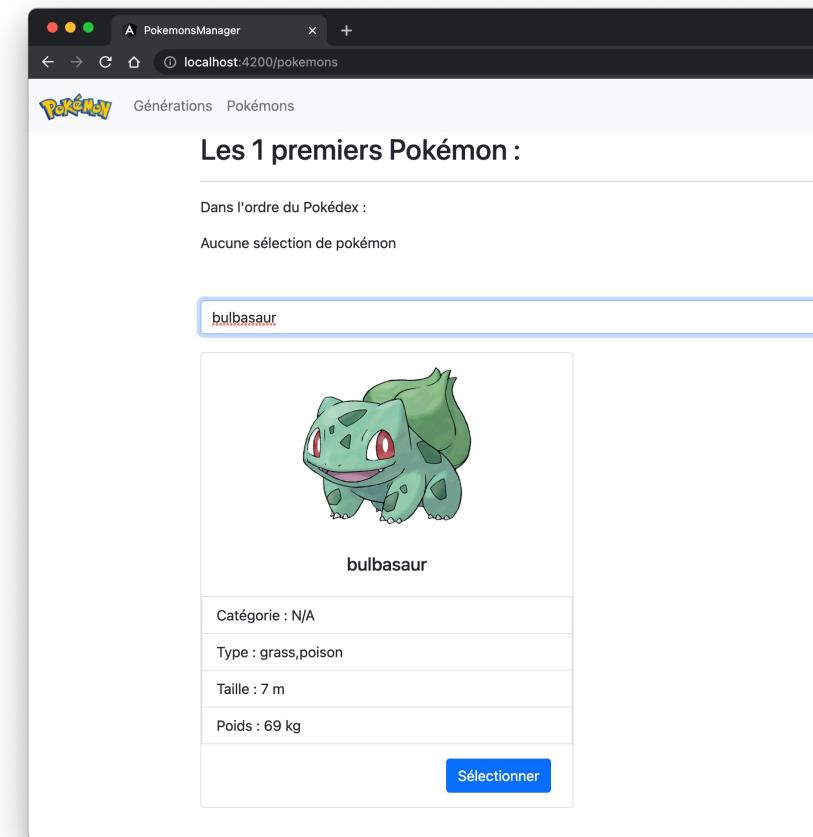
# Les sélecteurs 1/3

- Il est possible de sélectionner des éléments pour effectuer des actions

```
describe('Pokemons Page', () => {

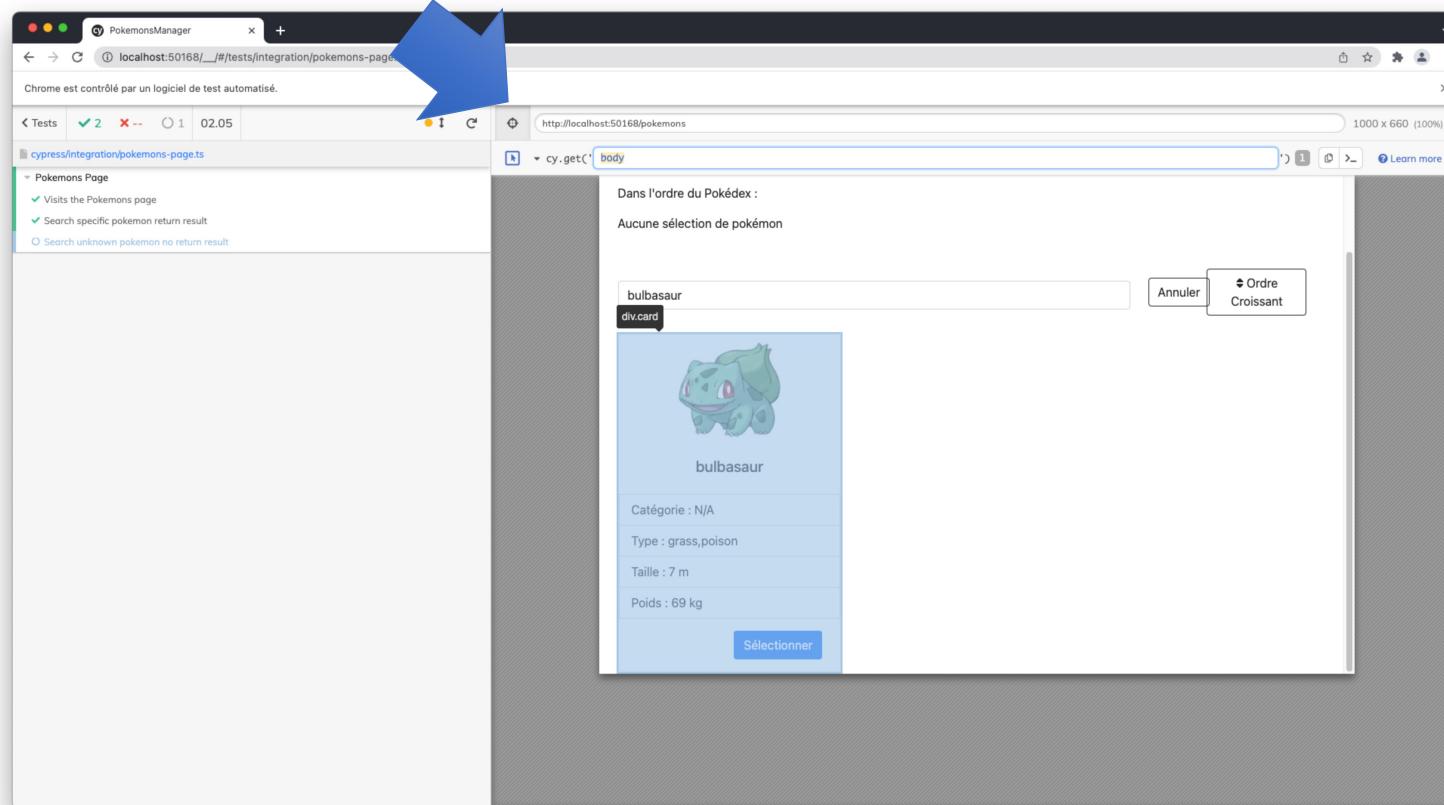
 it('Visits the Pokemons page', () => {
 cy.visit('/pokemons')
 cy.title().should('eq', 'PokemonsManager')
 cy.contains('Les 20 premiers Pokémon :)')
 cy.contains('Dans l\'ordre du Pokédex :)')
 cy.contains('Aucune sélection de pokémon')
 })

 it('Search specific pokemon return result', () => {
 const pokemonSearched = 'bulbasaur';
 cy.get('#pokemonSearch').type(pokemonSearched);
 cy.get('div.card').should('have.length', 1)
 })
})
```



# Les sélecteurs 2/3

- Il est possible d'identifier les sélecteurs des éléments avec l'outil graphique



# Les sélecteurs 3/3

- Il existe une formule pour faciliter la sélection des éléments : **cy-data**
  - Lors d'un changement dans le code, cela réduit l'impact sur les tests existants

```
describe('Pokemons Page', () => {

 // Version with cy-data
 it('Search specific pokemon return result', () => {
 const pokemonSearched = 'bulbasaur';
 cy.get('[cy-data=inputSearchPokemon]').type(pokemonSearched);
 cy.get('[cy-data=cardPokemon]').should('have.length', 1)
 })
})
```

```
<div cy-data="cardPokemon" class="card">
..
</div>
```

 [pokemon.component.html](#)

```
<input cy-data="inputSearchPokemon"
id="pokemonSearch">
```

 [pokemons.component.html](#)

# Les commandes Cypress 1/2

Il existe de nombreuses fonctions utilisables avec Cypress :

```
cy.visit('/uri')
```

- demande à Cypress de se rendre sur une page spécifique de votre application

```
cy.url().should('include', '/myUrl');
```

- Permet de vérifier l'url actuelle de votre page

```
cy.title().should('include', 'Page title');
```

- Permet de vérifier le contenu de la balise HTML title

```
cy.get("#button_validate");
```

- Permet de cibler un élément de la page HTML afin d'interagir avec

# Les commandes Cypress 2/2

`cy.pause()`

- Permet de mettre pause un test avec Cypress (debug)

`cy.debug()`

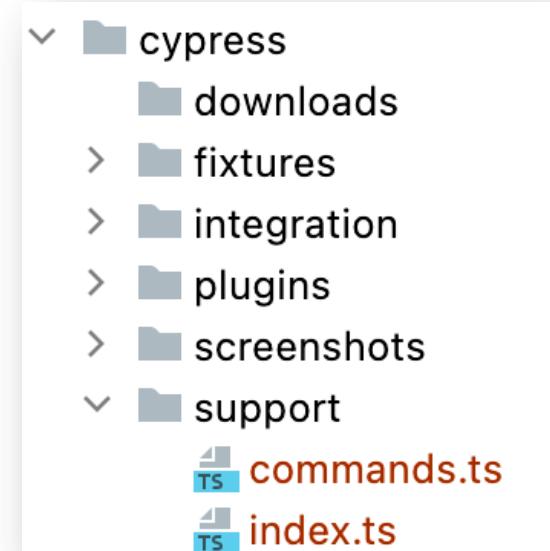
- Permet de lancer un point d'arrêt avec Cypress (nécessite d'ouvrir la console Développeur)
- <https://docs.cypress.io/api/commands/>

# Les commandes personnalisées 1/3

Il est possible d'ajouter des commandes avec Cypress en utilisant le code proposé :

```
// ****
// This example namespace declaration will help
// with Intellisense and code completion in your
// IDE or Text Editor.
// ****
// declare namespace Cypress {
// interface Chainable<Subject = any> {
// customCommand(param: any): typeof customCommand;
// }
// //
// function customCommand(param: any): void {
// console.warn(param);
// }
// }
```

 commands.ts



# Les commandes personnalisées 2/3

Exemple pour une fonction de recherche d'un Pokémon :

```
declare namespace Cypress {
 interface Chainable<Subject = any> {
 searchPokemon(pokemonName: string): typeof searchPokemon;
 }
}

function searchPokemon(pokemonName: string): void {
 cy.get('[cy-data=inputSearchPokemon]').type(pokemonName);
}

Cypress.Commands.add('searchPokemon', searchPokemon);
```

 commands.ts

```
import './commands';
```

 index.ts

# Les commandes personnalisées 3/3

Exemple pour une fonction de recherche d'un Pokémon :

**Avant**

```
describe('Pokemons Page', () => {
 it('Search specific pokemon return result', () => {
 const pokemonSearched = 'bulbasaur';
 cy.get('[cy-data=inputSearchPokemon]').type(pokemonSearched);
 cy.get('[cy-data=cardPokemon]').should('have.length', 1)
 })
})
```

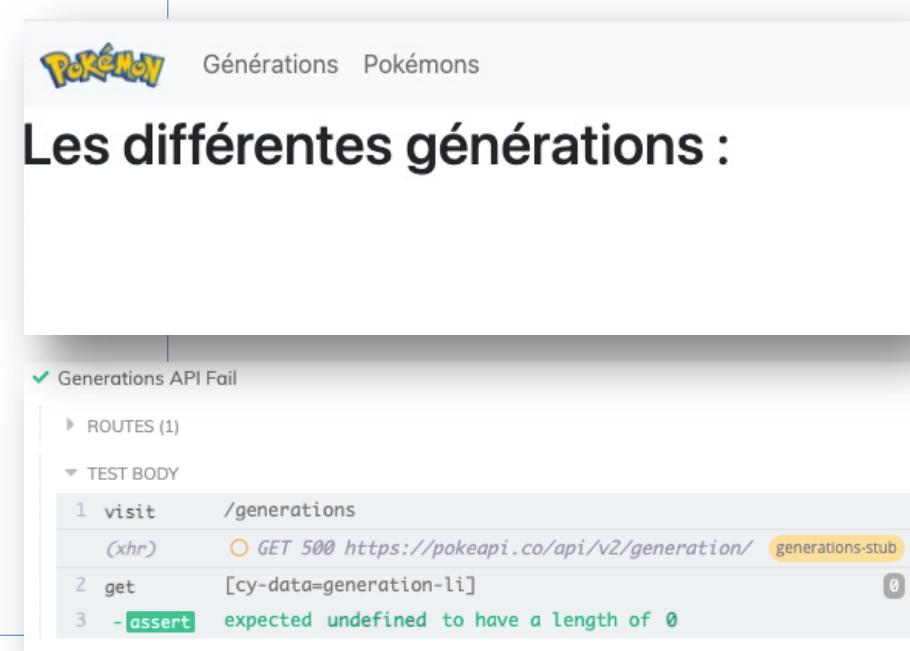
**Après**

```
describe('Pokemons Page', () => {
 it('Search specific pokemon return result', () => {
 cy.searchPokemon('bulbasaur');
 cy.get('[cy-data=cardPokemon]').should('have.length', 1)
 })
})
```

# API Stubbing

- Avec **cy.intercept()**, il est possible d'intercepter une requête réseau
  - Permet ainsi de simuler un comportement fictif

```
it('Generations API Fail', () => {
 cy.intercept(
 {
 method: 'GET',
 url: 'https://pokeapi.co/api/v2/generation'
 },
 {
 statusCode: 500,
 body: null
 }
).as('generations-stub');
 cy.visit('/generations')
 cy.get('[cy-data=generation-li]').should('have.length', 0)
})
```



# API Mock

- Avec **cy.intercept()**, il est également possible de mocker une réponse avec un JSON
  - Permet ainsi de simuler un retour précis de données

```
it('Generations API Mock with 2 element', () => {
 cy.intercept(
 {
 method: 'GET',
 url: 'https://pokeapi.co/api/v2/generation'
 },
 {
 fixture: 'generations.json'
 }
).as('getGenerations');
 cy.visit('/generations')
 cy.get('[cy-data=generation-li]').should('have.length', 2)
})
```

```
{
 "count": 2,
 "next": null,
 "previous": null,
 "results": [
 {
 "name": "generation-i",
 "url": "https://pokeapi.co/api/v2/generation/1"
 },
 {
 "name": "generation-ii",
 "url": "https://pokeapi.co/api/v2/generation/2"
 }
]
```



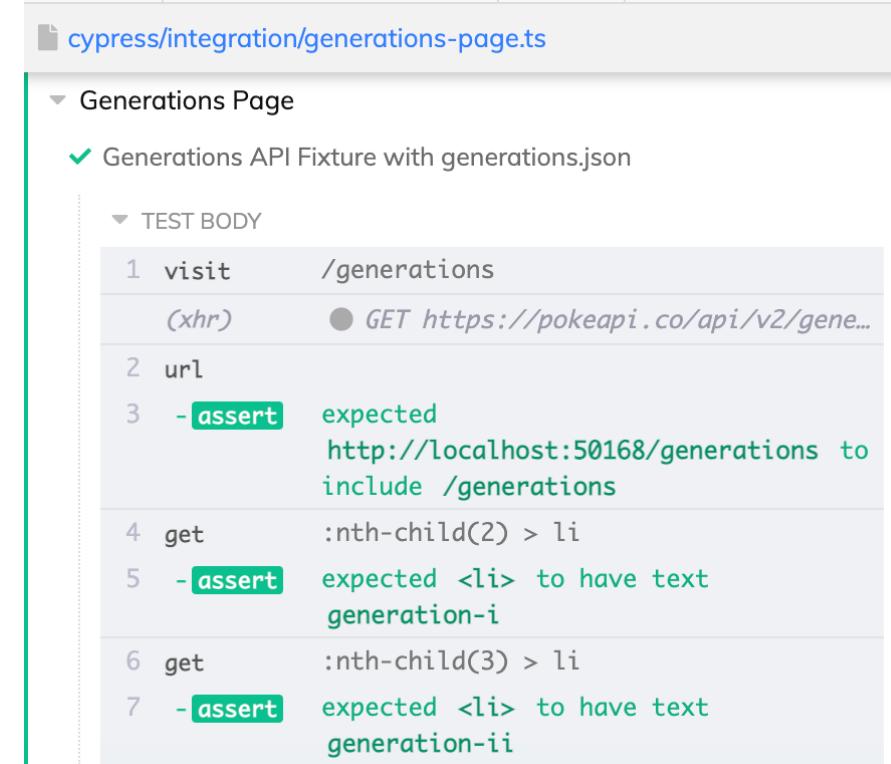
# Fixture et données

- Avec **cy.fixture()**, il est possible de manipuler et tester une données
  - <https://docs.cypress.io/api/commands/fixture>

```
it('Generations API Fixture with generations.json', () => {
 cy.visit('/generations');
 cy.url().should('include', '/generations')

 cy.fixture('generations.json').then((generation) => {

 for (var i = 0; i < generation.results.length; i++) {
 let g: { name: string, url: string } = generation.results[i];
 cy.get(`:nth-child(${i + 2}) > li`).should('have.text', g.name)
 }
 })
});
```



The screenshot shows the Cypress Test Runner interface. The left pane displays the test code in a text editor. The right pane shows the test runner's UI with the file path 'cypress/integration/generations-page.ts'. The 'Generations Page' section is expanded, revealing the 'TEST BODY' section. The test body contains the following steps:

- 1 visit /generations (xhr) GET https://pokeapi.co/api/v2/gene...
- 2 url
- 3 - assert expected http://localhost:50168/generations to include /generations
- 4 get :nth-child(2) > li
- 5 - assert expected <li> to have text generation-i
- 6 get :nth-child(3) > li
- 7 - assert expected <li> to have text generation-ii

# Migration

Si vous utilisez un projet sur Protractor, vous pouvez suivre les étapes de migrations :

- <https://docs.cypress.io/guides/migrating-to-cypress/protractor>

## Migrating from Protractor to Cypress

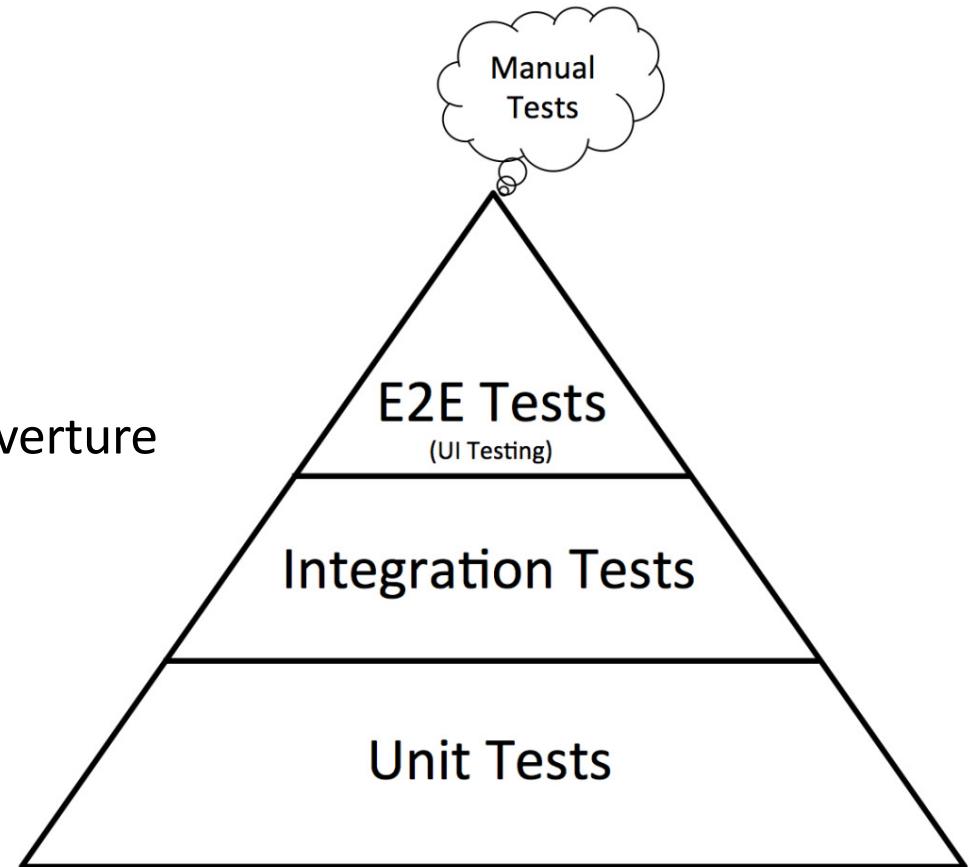
### 🎓 What you'll learn

- Benefits of using Cypress in Angular apps
- How Cypress can create reliable e2e tests for Angular apps
- How to migrate Protractor tests to Cypress

## Introduction

# Les tests

- Les tests unitaires sont les fondements
- Les tests E2E permettent une meilleure couverture
- Les tests manuels sont le sumum



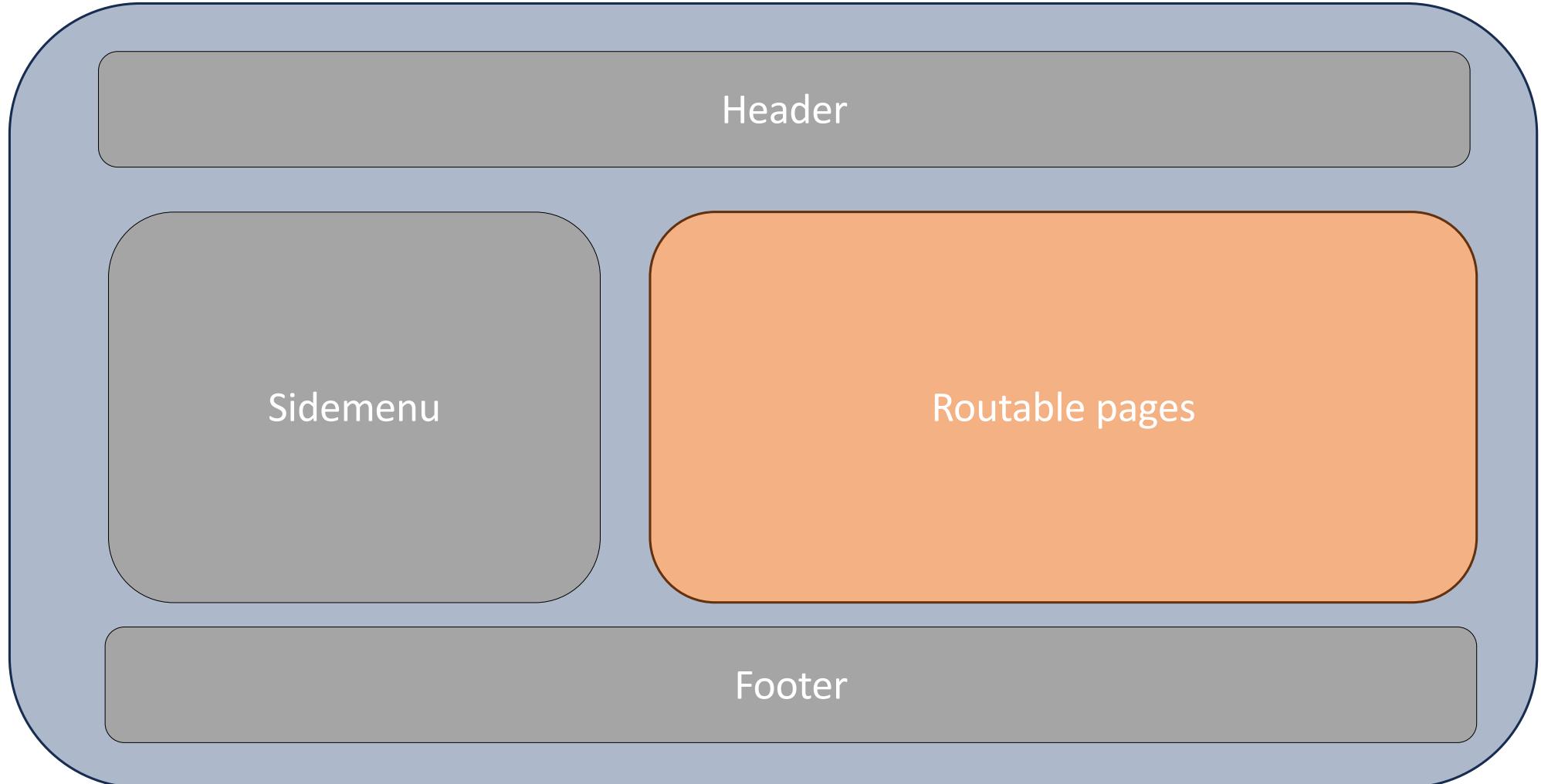
# Architecture Angular

Développement Web

# Introduction 1/2

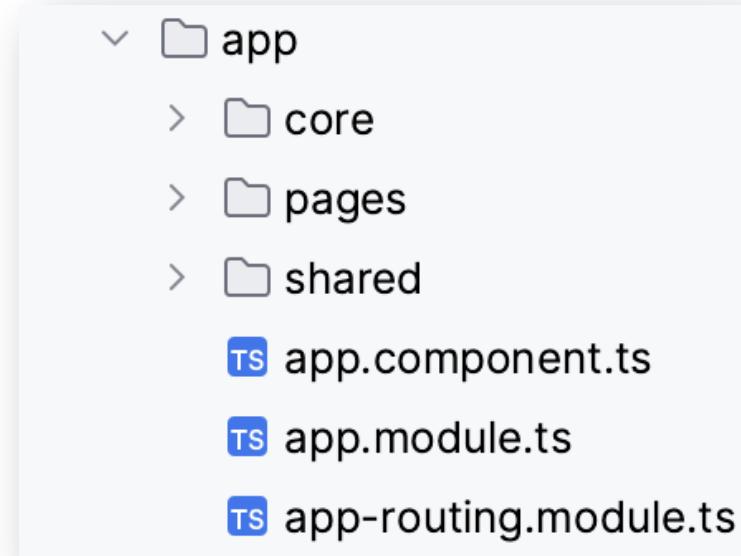
- La majorité des applications se basent sur une mise en page standard
- Les éléments communs sont le **header**, le **footer** et un **sidemenu**
- Ces composants sont souvent appelés les **shell**
  - Conçu pour être rendus une seule fois par chargement de page
  - Ils sont chargés à la racine dans le composant d'application parent

# Introduction 2/2



# Organisation d'un projet modulaire 1/2

Une organisation connue consiste à découper le projet en 3 dossiers :

- **Core**
  - **Pages**
  - **Shared**
- 
- Il est préférable de garder le module principal le plus petit possible

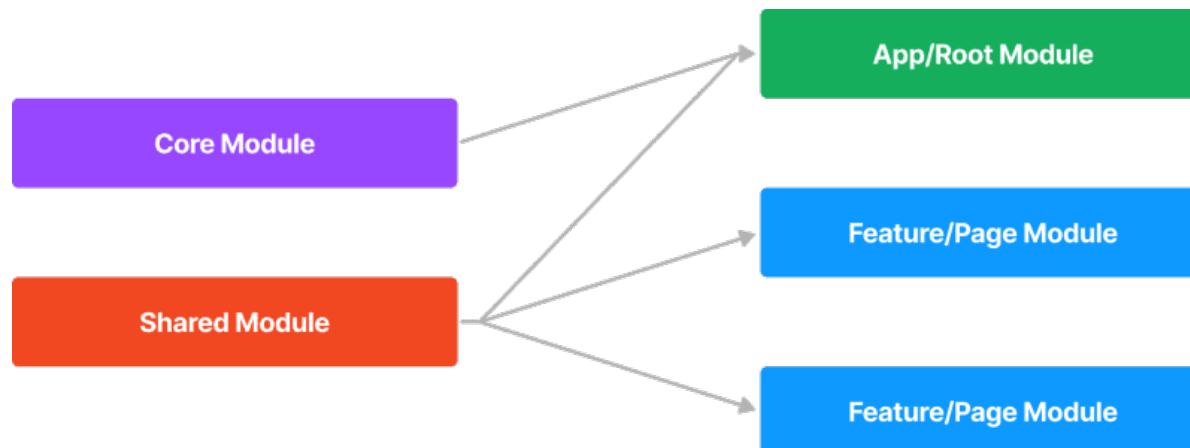
# Organisation d'un projet modulaire 2/2

Le module **Core** contient tout ce que l'on importe une seule fois dans l'application

- Les services, les modèles, les intercepteurs, les composants shell

Le module **Shared** regroupe les éléments utilisés à plusieurs endroits

Le module **Page** regroupe tous les **Features**



# Exemple d'architecture modulaire

# Les standalone

Angular 15 a introduit une nouvelle notion, les composants **standalone**

L'organisation se simplifie en faisant disparaître les modules

# Exemple d'architecture standalone