Formation Angular

Plan du cours

- Angular, c'est quoi ?
- Pourquoi Angular ?
- Angular, historique
- AngularJS
- Angular vs AngularJS
- Utiliser Angular, Prérequis ?
- Angular, Principales fonctionnalités ?
- L'architecture d'Angular (Module, Component, Template, Metadata, Data Binding, Directive et Pipe)
- Les tests avec Angular
- Les formulaires avec Angular

Angular, c'est quoi?

- Framework open source ;
- Ecrit en Typescript ;
- Permet la création d'applications Web de type SPA « Single Page Applications » :des applications web accessibles via une page web unique qui permet de fluidifier l'expérience utilisateur et d'éviter les chargements de pages à chaque nouvelle action ;
- Orienté composant (Component) ;
- Développé par une équipe (Google + une communauté de particuliers + de sociétés) ;
- Refonte complète du Framework AngularJS (développé par la même équipe).

Angular, c'est quoi?

Une plateforme de développement, réalisée avec TypeScript. Cette plateforme contient :

- ✓ Un Framework orienté composant pour le développement des applications web scalable ;
- ✓ Une collection de librairies offrant plusieurs fonctionnalités (routing, forms, management, client-server communication, …) ;
- ✓ Une suite d'outils de développement pour vous aider à développer, construire, tester et mettre à jour votre code.

✓ Voir le site officiel pour plus de précisions : https://angular.io/guide/what-is-angular

Pourquoi Angular?

- Angular est géré par Google : il y a donc peu de chances qu'il disparaisse, et l'équipe de développement du framework est excellente.
- Le TypeScript : ce langage permet un développement beaucoup plus stable, rapide et facile.
- Le framework *Ionic* : le framework permettant le développement d'applications mobiles multi-plateformes à partir d'une seule base de code — utilise Angular.

Angular, historique

Angular	Date de sortie
AngularJS 1.x	Octobre 2010
Angular 2.0	Septembre 2016
Angular 4.0	Mars 2017
Angular 5.0	Novembre 2017
Angular 6.0	Mai 2018
Angular 7.0	Octobre 2018
Angular 8.0	Mai 2019
Angular 9.0	Février 2020
Angular 10.0	Juin 2020
Angular 11.0	Novembre 2020

AngularJS 1.7.x

- Dernière version : AngularJS 1.7.x ;
- AngularJS est en mode LTS (Long Term Support) ;
- Basé sur MVC ;
- Utilise JavaScript ;
- Site officiel : *https://angularjs.org*

AngularJS vs Angular :

Architecture :

AngularJS	Angular
suit l'architecture MVC	✓ *component-based framework* ✓ a remplacé les contrôleurs et les $scope par des composants ce qui permet de diviser une application en plusieurs composants pouvant être appelés chacun à part et réutilisés si besoin. Ceci facilite le développement des applications, améliore la flexibilité, la modularité et la testabilité du code de l'application. ✓ Chaque composant (Component) est constitué d'un template HTML, d'un fichier CSS et un fichier TypeScript (qui définit la logique du composant).

Language :

AngularJS	Angular
JavaScript	✓ TypeScript qui est un sur-ensemble de JavaScript et permet d'améliorer et sécuriser la production du code JavaScript.

AngularJS vs Angular :

Syntaxe :

Angular	AngularJS
Utilise directement les éléments du DOM HTML (href, src et hidden)	ng-href, ng-src, ng-show et ng-hide
(click)=”doSomething()	ng-click=”fonction()
Pas besoin d’utiliser la directive ng-app	
L’annotation **@Injectable** est utilisée pour simplifier le mécanisme d’injection de dépendances.	
*ngFor	ng-repeat
ngClass	ng-class
ngModel	ng-model
La syntaxe du routage a été modifiée	
Utilise Angular CLI	
Support des applications mobiles	Non

Plan du cours

- Angular, c'est quoi ?
- Pourquoi Angular ?
- Angular, historique
- AngularJS
- Angular vs AngularJS
- **Utiliser Angular, Prérequis ?**
- Angular, Principales fonctionnalités ?
- L'architecture d'Angular (Module, Component, Template, Metadata, Data Binding, Directive et Pipe)

Prérequis : NodeJS et NPM

- https://nodejs.org/fr

- Node JS est une plateforme logicielle libre en JavaScript orientée vers les applications réseau événementielles. Parmi les modules natifs de Node JS, on retrouve http qui permet le développement de serveur HTTP. Il est donc possible de se passer de serveurs web tels que *Nginx* ou *Apache* lors du déploiement de sites d'applications web développés avec Node JS.

- Node.js est ainsi le point central qui va permettre d'exécuter des programmes écrits en javascript côté serveur. Node.js utilise un outil npm (Node Pakage Manager). npm simplifie la vie du développeur en permettant de publier et de partager des librairies Node.js. npm permet notamment de simplifier l'installation, la mise à jour ou la désinstallation de ces librairies. On pourra parler de librairies, de paquets ou de dépendances (en anglais packages ou dependencies).

Pré-requis : Angular CLI (CLI)

- Angular CLI (Command Line Interface) **https://angular.io/cli**

permet de générer, compiler et déployer des projets Angular.

- Les dernières versions de ces outils sont disponibles ci-dessous :
  - https://github.com/angular/angular-cli/releases
  - https://github.com/angular/angular/releases

- La procédure d'installation est détaillée sur le site officiel d'Angular.

Pré-requis : Visual Studio Code (VSD)

- Visual Studio Code est l'éditeur utilisé dans la plupart des conférences sur Angular.
- VS code est un éditeur de code développé par Microsoft pour Windows, Linux et OS X.
- Le site officiel est là : https://code.visualstudio.com

Plan du cours

- Angular, c'est quoi ?
- Pourquoi Angular ?
- Angular, historique
- AngularJS
- Angular vs AngularJS
- Utiliser Angular, Prérequis ?
- **Angular, Principales fonctionnalités ?**
- L'architecture d'Angular (Module, Component, Template, Metadata, Data Binding, Directive et Pipe)

Angular 2.0

- Refonte complète d'AngularJS ;
- Orientée Mobile ;
- *https://angular.io/*

Angular 3.0 a été sauté !

Pour un problème de versions des librairies d'Angular 2 :

- *@angular/core* v2.3.0
- *@angular/compiler* v2.3.0
- *@angular/compiler-cli* v2.3.0
- *@angular/http* v2.3.0
- *@angular/router* v3.3.0

➔ Dans Angular 2, la librairie *router* était déjà dans la version 3 !.

Angular 4

- Optimisation du code généré par rapport aux composants ;
- Compilation rapide ;
- Animations ajoutées facilement par l'importation du module {*BrowserAnimationsModule*} de @*angular/platform-browser/animations* dans *NgModule* ;
- ***ngIf/else*** a été introduite ;
- Pas besoin d'écrire un modèle pour la validation des e-mails dans Angular 4.
- Possibilité d'utiliser TypeScript 2.1 ou 2.2 (dans Angular 2, la seule version supportée est 1.8)

Angular 5

- Build Optimizer. Le produit généré par Angular CLI utilise «Build Optimizer » par défaut ;
- Angular Universal State Transfer API et DOM Support ;
- Amélioration du compilateur ;
- "Internationalization" par rapport aux *Number*, *Date* et *Currency* ;
- @*angular/http* est obsolète (remplacée par @*angular/common/http*) ;
- *Angular Forms* a ajouté *updateOn, Blur, Submit ;*
- Le support de *RxJS 5.5* (programmation réactive) ;
- Nouveaux événements du cycle de vie du routeur : *GuardsCheckStart, ChildActivationStart, ActivationStart, GuardsCheckEnd, ResolveStart, ResolveEnd, ActivationEnd, ChildActivationEnd ;*

Angular 6

- Introduction des deux commandes *ng update* et *ng add* au niveau de *Angular CLI* ;
- *Angular Elements* (@*angular/elements*) : Composants web personnalisés ;
- Component Dev Kit (CDK) : Ensemble de composants web primitives (https://material.angular.io/cdk) ;
- Angular Material Starter Components (@*angular/*material) ;
- CLI Workspaces ;
- Schematics ;
- Library Support. Possibilité de créer nos propres librairies ;
- Tree Shakable Providers ;
- Amélioration des performances pour les Animations ;
- RxJS v6.

Angular 7

- L’invite de commande fourni par Angular CLI. L'interface de ligne de commande invite les utilisateurs à exécuter des commandes courantes telles que *ng new* ou *ng add @angular/material* pour vous aider à découvrir des fonctionnalités intégrées telles que le routage ou la prise en charge SCSS ;
- Ajout de la notion du « budgets » dans Angular CLI ;
- *Angular Material & CDK*
  - *Virtual Scrolling*
  - *Drag and Drop*
- Le support de la projection de contenu dans *Angular Element* ;
- *TypeScript 3.1*
- *RxJS 6.3*
- Le support de *Node 10*

Angular 8

- *Differential Loading by Default* : il s'agit d'un processus par lequel le navigateur choisit entre JavaScript moderne ou hérité en fonction de ses propres capacités ;
- Import dynamique de la configuration du module *Router* ;
- Builder APIs au niveau de *Angular CLI. E*xemples de commandes : *ng build, ng test, ng run ;*
- Le support de *web worker ;*
- Les commandes *ng new* et *ng deploy* au niveau d'Angular CLI.

Angular 9

- Le moteur de compilation *Ivy* pour améliorer les performances d'Angular ;
- *@angular/localize* : le système natif d'internationalisation (i18n) d'Angular devient plus souple ;
- Mode strict : Une nouvelle option lors de la création de votre application Angular vous permet de configurer automatiquement *TypeScript* en mode strict, afin d'assurer une meilleure qualité de développement ;
- Une nouvelle option d'Angular CLI permet de distinguer les différents types de composants, et la gestion des *modals*, *dialogs* et autres popups est simplifiée.
- *TypeScript 3.7* et *ES2020* : Angular supporte désormais *TypeScript 3.6* et *TypeScript 3.7*, ce qui permet d'utiliser deux nouveautés très attendues de l'ES2020 : *l'optional chaining* et le *nullish coalescing*.

Angular 10

- Configuration stricte. Lors de la création d'un projet, Angular CLI vous propose désormais d'activer directement des options de compilation stricte, pour une meilleure fiabilité du code ;
- Bazel. Un nouvel outil de build est disponible en option dans Angular 10 ;
- Support de TypeScript 3.9 ;
- Support de TypeScript 4.0 ;

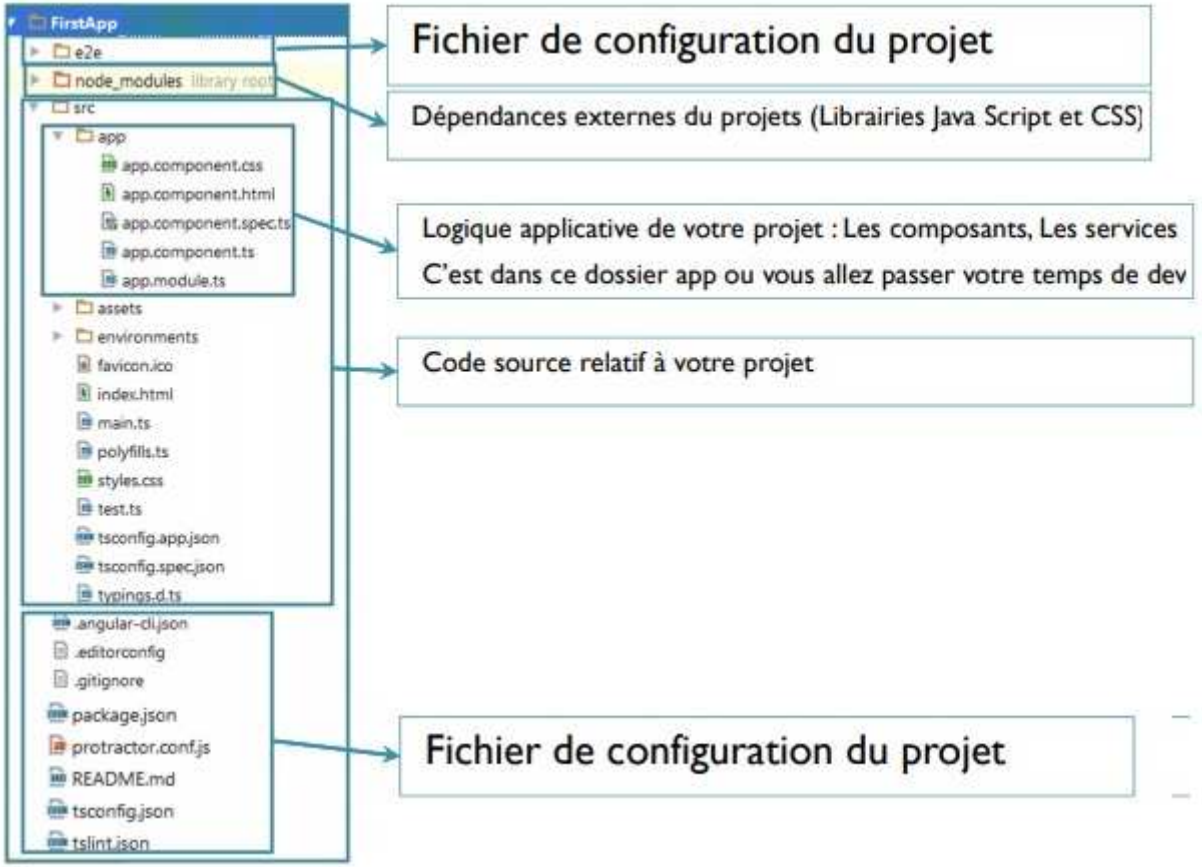
Angular 11

- *Trusted Types* : un nouveau standard JavaScript pour améliorer la sécurité et limiter le risque d'attaques XSS ;
- *HMR* : Le support du *Hot Module Replacement* est simplifié ;
- Plus de performance (compilation plus rapide, …) ;
- Gestion des erreurs. Les erreurs sont normalisées via des codes détaillés dans la documentation (https://angular.io/errors) ;
- Fin du support de IE 9 & 10. Angular 11 ne supporte plus Internet Explorer 9 et 10, mais supporte toujours IE 11 ;
- Support de TypeScript 4.1.

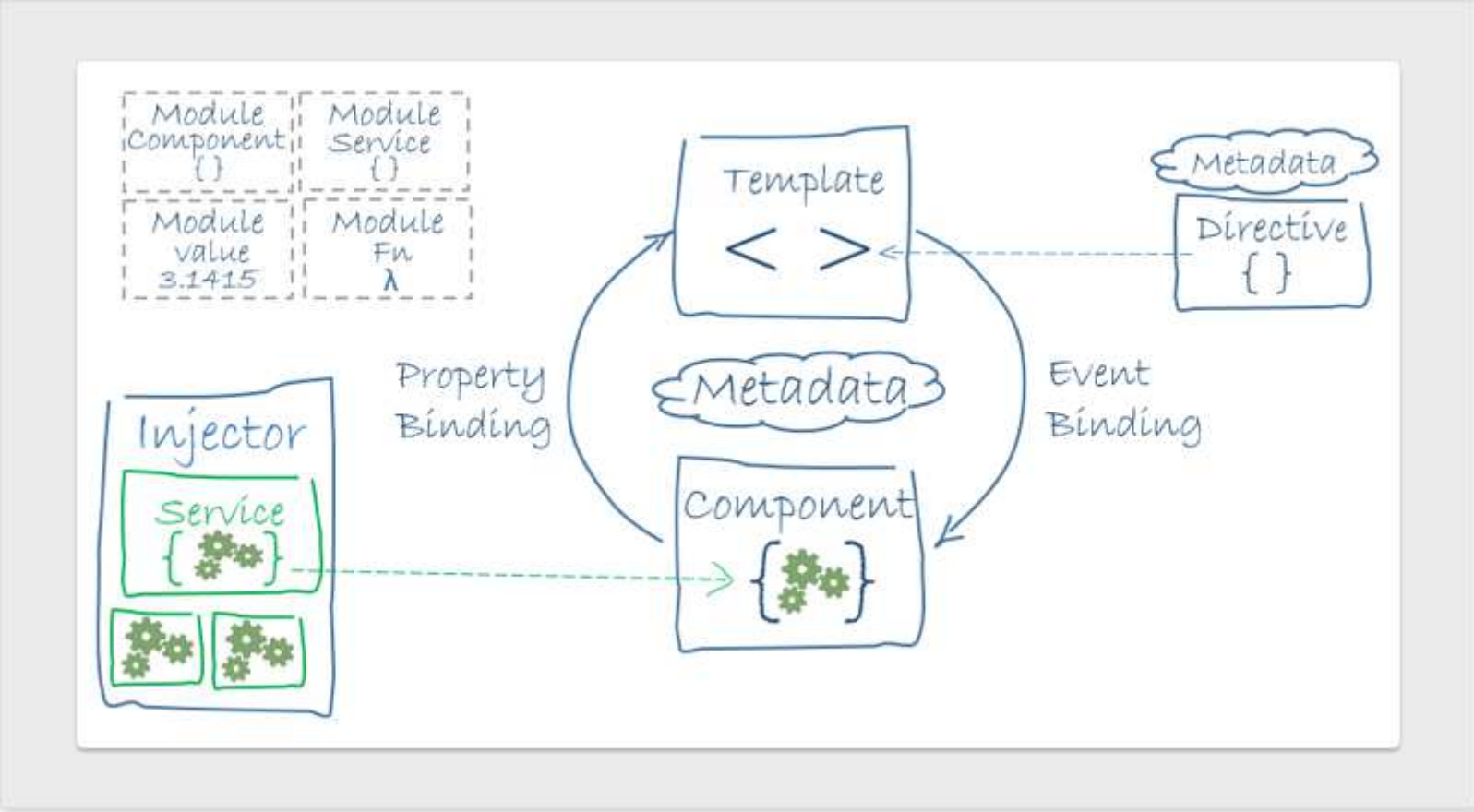
Plan du cours

- Angular, c'est quoi ?
- Pourquoi Angular ?
- Angular, historique
- AngularJS
- Angular vs AngularJS
- Utiliser Angular, Prérequis ?
- Angular, Principales fonctionnalités ?
- **L'architecture d'Angular (Module, Component, Template, Metadata, Data Binding, Directive et Pipe)**

Arborescence d'un projet Angular



Comprendre l'architecture Angular



Comprendre l'architecture Angular

- Une application Angular se compose de :
  - Un ou plusieurs modules dont un est principal ;
  - Chaque module peut inclure :
    - Des composants web (les IHM) ;
    - Des services pour la logique applicative. Les composants peuvent utiliser les services via le principe de l'IOC ;
    - Les Directives ;
    - Les pipes : utilisés pour formater l'affichage des données dans les composants.

- Voir le site officiel d'Angular : https://angular.io/guide/architecture.

Comprendre l'architecture Angular

- Les notions fondamentales d'Angular :
  - Module ;
  - Component ;
  - Template ;
  - Metadata ;
  - Data Binding ;
  - Directive ;
  - Pipe.

Les Modules ?

- Le Module est le bloc de code conçu pour effectuer une seule tâche. Nous pouvons exporter le module sous forme de classe ;
- Chaque application Angular possède un module racine ;
- Un Module peut représenter le tout ou une partie de l'application ;
- Un Module peut être partagé à d'autres modules ;
- Angular possède son propre système de modularité appelé *NgModules* ;
- Chaque application Angular possède au moins une classe de module appelé classiquement *AppModule* ;
- Un Module est une classe avec un décorateur @*NgModule* (Decorator) ;
- Les décorateurs sont des fonctions qui modifient les classes JavaScript ;
- Angular possède de nombreux décorateurs qui attachent des métadonnées aux classes pour configurer et donner le sens à ces classes (exemple : *src/app/app.module.ts*)

Les Modules ?

```
import {
BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
@NgModule({
  declarations: [AppComponent],
    imports: [
    BrowserModule,AppRoutingModule],
    providers: [],
    bootstrap: [AppComponent]})
export class AppModule { }
```

Components ?

- Dans Angular, tout est composant. Un composant est une balise HTML personnalisée (ex : app-root, app-shop, …) ;
- Définit par : 1 *selector* (le nom de la balise), 1 *template*, 1 ou plusieurs fichiers de style CSS (facultatif) ;
- Représenté par : 1 fichier Typescript (.ts), 1 fichier HTML (.html), 1 ou plusieurs fichiers de style CSS (facultatif) ;
- Un composant peut utiliser d'autres composants ;
- Toute application Angular doit avoir au moins un composant (*root Component*) ;
- Le composant racine (root) connecte la hiérarchie du composant avec la page Document Object Model (DOM);
- Chaque composant définit une classe qui contient les données de l'application (*data*) et une logique, et est associée à un *template* HTML qui définit une vue à afficher dans un environnement cible ;
- Le décorateur @*Component ()* identifie la classe immédiatement comme un composant et fournit le modèle et les métadonnées spécifiques au composant associé.

Components ?

```
import { Component, OnInit } from '@angular/core';
import {RecordsService} from "../records.service";

@Component({
  selector: 'employee',
  templateUrl: './employee.component.html',
  styleUrls: ['./employee.component.css'],
  providers : [RecordsService]
})
export class EmployeeComponent implements OnInit {
  employee1 : string[]=[]
  employee2 : string[]=[]
  employee3 : string[]=[]
  constructor(private service:RecordsService) { }

  ngOnInit(): void {
  }

}
```

Template ?

- C’est la page html :

```
<button type="button" name="buttom" (click)='getEmployee1()'>Employee1</button>
<ul class="list-group">
<li *ngFor="let info of employee1" class="list-group-item">{{info}}</li>
</ul>

<pre></pre>

<button type="button" name="buttom" (click)='getEmployee2()'>Employee2</button>
<ul class="list-group">
<li *ngFor="let info of employee2" class="list-group-item">{{info}}</li>
</ul>

<pre></pre>

<button type="button" name="buttom" (click)='getEmployee3()'>Employee3</button>
<ul class="list-group">
<li *ngFor="let info of employee3" class="list-group-item">{{info}}</li>
</ul>
```

Metadata

```
@Component({
  selector: 'employee',
  templateUrl: './employee.component.html',
  styleUrls: ['./employee.component.css'],
  providers : [RecordsService]
})
```

Data binding : L'interpolation

L'interpolation est un mécanisme qui va, dans un premier temps, analyser les templates pour retrouver les {{interpolations}}, afin de placer, dans un second temps, des écouteurs de changement sur les propriétés de component qui sont liées par leurs noms.

Une fois cette résolution de liaisons faite, Angular va valoriser les interpolations avec ces propriétés lors d'un changement de ces dernières.

```
<button type="button" name="buttom" (click)='getEmployee1()'>Employee1</button>
<ul class="list-group">
<li *ngFor="let info of employee1" class="list-group-item">{{info}}</li>
</ul>
```

Property binding

Le *Property Binding* sur un élément du DOM :

typescript

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

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

```html
<div class="container">
  <div class="row">
    <div class="col-xs-12">
      <h2>Mes appareils</h2>
      <ul class="list-group">
        <app-appareil></app-appareil>
        <app-appareil></app-appareil>
        <app-appareil></app-appareil>
      </ul>
      <button class="btn btn-success" disabled>Tout allumer</button>
    </div>
  </div>
</div>
```

html

```html
<button class="btn btn-success" [disabled]="!isAuth">Tout allume</button>
```

Le *property binding*

Le *Property Binding* sur un *attribute directive* :

Un *attribute directive* est une directive modifiant le comportement ou l'apparence d'un élément.
Angular propose une série *d'attribute directives* tels que *ngClass* ou *ngStyle*.

app.component.ts

```
1.  import { Component } from '@angular/core';
2.
3.  @Component({
4.    selector: 'app-root',
5.    template: ` <div [align]="alignement"
6.    [ngStyle]="{color:couleur}">Personne : {{person}} | Age : {{age}} |
7.    Adresse : {{address.street}}</div> `
8.  })
9.  export class AppComponent {
10.
11.   person:string= 'John Doe';
12.   age:number= 30;
13.   address:any= {street:'rue du Paradis', city:'75010 Paris'};
14.   alignement:string = 'right'; couleur:string = 'red';
15. }
```

Le *property binding*

Le *Property binding* sur une propriété d'un *component* :

app.component.ts

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

@Component({
  selector: 'app-root',
  template: ` <div [align]="alignement"
  [ngStyle]="{color:couleur}">Personne : {{person}} | Age : {{age}}
  Adresse : {{address.street}}</div>
  <app-comp1 [monAdresse]="address"></app-comp1> `
})
export class AppComponent {

  person: string= 'John Doe';
  age: number= 30;
  address: any= {street: 'rue du Paradis', city: '75010 Paris'};
  alignement: string = 'right';
  couleur: string = 'red';
}
```

Comp1.component.ts

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

@Component({
  selector: 'app-comp1',
  template: ` {{monAdresse.street}} `
})
export class Comp1Component {

  @Input() monAdresse: any;

}
```

Data binding : *L'event binding*

```
<h1>Inscription</h1>
<div class="container">
    <form #login="ngForm" (ngSubmit)="submit(login)">
```

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

@Component({
  selector: 'inscription',
  templateUrl: './inscription.component.html',
  styleUrls: ['./inscription.component.css']
})
export class InscriptionComponent  {

  constructor() { }

  submit(login:any){
    console.log("Form submitted",login);
  }

}
```

Le two-way Data Binding

La liaison à double sens (ou two-way binding) utilise la liaison par propriété et la liaison par événement en même temps ; on l'utilise, par exemple, pour les formulaires, afin de pouvoir déclarer et de récupérer le contenu des champs, entre autres.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';


import { AppComponent } from './app.component';
import { MonPremierComponent } from './mon-premier/mon-premier.component';
import { AppareilComponent } from './appareil/appareil.component';
import { FormsModule } from '@angular/forms';


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

```
<li class="list-group-item">
  <h4>Appareil : {{ appareilName }} -- Statut : {{ getStatus() }}</h4>
  <input type="text" class="form-control" [(ngModel)]="appareilName">
</li>
```

Les Directives

- Les directives sont des instructions intégrées dans le DOM que vous utiliserez presque systématiquement quand vous créerez des applications Angular. Quand Angular lit votre template et rencontre une directive qu'il reconnait, il suit les instructions correspondantes.

- Angular offre la possibilité de créer vos propres directives.

- Il existe deux types principaux de directive : les directives structurelles et les directives par attribut.

- ***Les directives structurelles*** modifient la structure du document : ****ngIf*** et ****ngFor***.

- À la différence des directives structurelles, ***les directives par attribut*** modifient le comportement d'un objet déjà existant ***: ngModel, ngClass, ngStyle.***

La Directive structurelle : *ngIf

```
<li class="list-group-item">
  <div style="width:20px;height:20px;background-color:red;"
      *ngIf="appareilStatus === 'éteint'"></div>
  <h4>Appareil : {{ appareilName }} -- Statut : {{ getStatus() }}</h4>
  <input type="text" class="form-control" [(ngModel)]="appareilName">
</li>
```

La Directive structurelle : *ngFor

```ts
export class AppareilComponent implements OnInit {

  @Input() appareilName: string;
  @Input() appareilStatus: string;

  constructor() {
```

```ts
export class AppComponent {
  isAuth = false;

  appareils = [
    {
      name: 'Machine à laver',
      status: 'éteint'
    },
    {
      name: 'Frigo',
      status: 'allumé'
    },
    {
      name: 'Ordinateur',
      status: 'éteint'
    }
  ];

  constructor() {
```

```html
<div class="container">
  <div class="row">
    <div class="col-xs-12">
      <h2>Mes appareils</h2>
      <ul class="list-group">
        <app-appareil  *ngFor="let appareil of appareils"
                       [appareilName]="appareil.name"
                       [appareilStatus]="appareil.status"></app-appareil>
      </ul>
      <button class="btn btn-success"
              [disabled]="!isAuth"
              (click)="onAllumer()">Tout allumer</button>
    </div>
  </div>
</div>
```

La Directive par attribut : ngStyle

html

```html
<h4 [ngStyle]="{color: getColor()}">Appareil : {{ appareilName }} -- Statut : {{ getStatus() }}</h4>
```

```
getColor() {
    if(this.appareilStatus === 'allumé') {
      return 'green';
    } else if(this.appareilStatus === 'éteint') {
      return 'red';
    }
}
```

La Directive par attribut : ngClass

html

```html
<li [ngClass]="{'list-group-item': true,
                'list-group-item-success': appareilStatus === 'allumé',
                'list-group-item-danger': appareilStatus === 'éteint'}">
  <div style="width:20px;height:20px;background-color:red;"
       *ngIf="appareilStatus === 'éteint'"></div>
  <h4 [ngStyle]="{color: getColor()}">Appareil : {{ appareilName }} -- Statut : {{ getStatus() }}
</h4>
  <input type="text" class="form-control" [(ngModel)]="appareilName">
</li>
```

Les Pipes

- Les pipes (/pʌɪp/) prennent des données en input, les transforment, et puis affichent les données modifiées dans le DOM.
- Angular fournit plusieurs Pipe.
- Angular offre la possibilité de créer nos propres Pipes.

```ts
export class AppComponent {
  isAuth = false;
  lastUpdate = new Date();
```

```html
<h2>Mes appareils</h2>
<p>Mis à jour : {{ lastUpdate }}</p>
```

```html
<p>Mis à jour : {{ lastUpdate | date }}</p>
```

```html
<p>Mis à jour : {{ lastUpdate | date: 'short' }}</p>
```

```html
<p>Mis à jour : {{ lastUpdate | date: 'yMMMMEEEEd' }}</p>
```

```html
<p>Mis à jour : {{ lastUpdate | date: 'yMMMMEEEEd' | uppercase }}</p>
```

Les Pipes

Le Pipe ***asyn*** est un cas particulier mais extrêmement utile dans les applications Web, car il permet de gérer des données asynchrones, par exemple des données que l'application doit récupérer sur un serveur.

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

```
<p>Mis à jour : {{ lastUpdate | async | date: 'yMMMMEEEEd' | uppercase }}</p>
```

Les tests avec Angular

Les tests unitaires utilisent :

- ✓ Karma
- ✓ Jasmine

Les tests end-to-end utilisent :

- ✓ Protractor

Pour les lancer on utilise les scripts correspondants contenus dans le fichier ***package.json.***

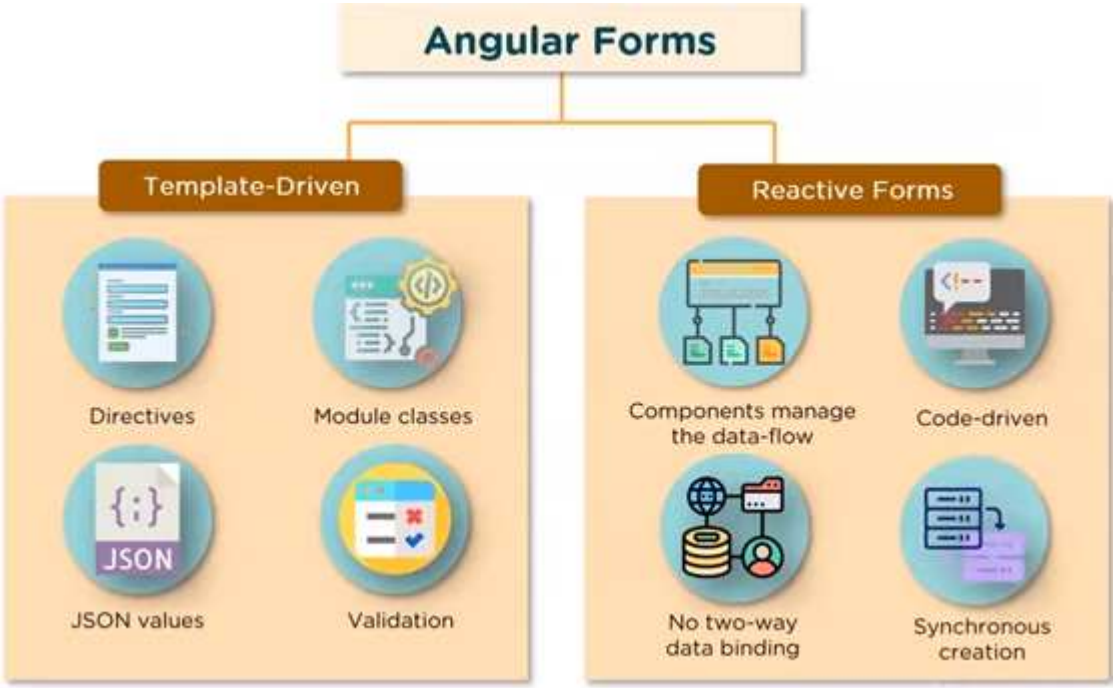
- Protractor est un Framework de test end-to-end utilisé par Angular et AngularJS
- Selenium est une librairie d'automatisation pour navigateur .
- Protractor s'appuie sur Selenium
- Selenium utilise ChromeDriver pour piloter Chrome
- Enfin "webdriver" Selenium communique avec les "navigateurs".

Les tests avec Angular

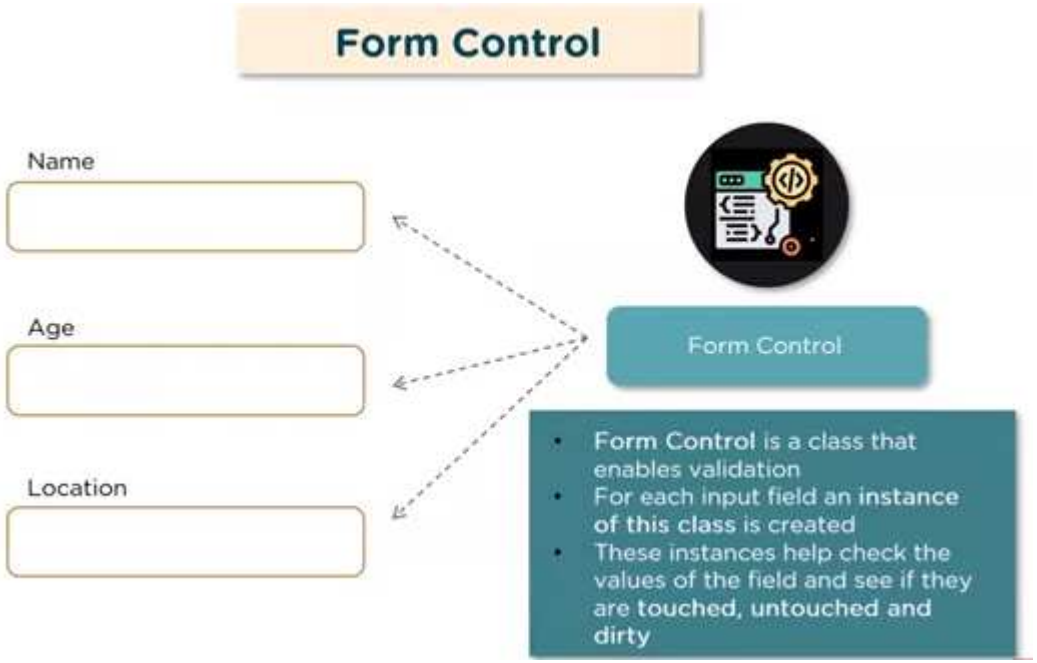
- Il faut utiliser chrome dans sa version **85 minimum**. Des erreurs peuvent survenir dans les tests end to end suivant la version de Chrome utilisée et de chromeDriver.

```
# Executer
npm run start

# Tester
http://localhost:4200/

# Effectuer des modifications
```

Les formulaires avec Angular



Les formulaires avec Angular



Les formulaires avec Angular