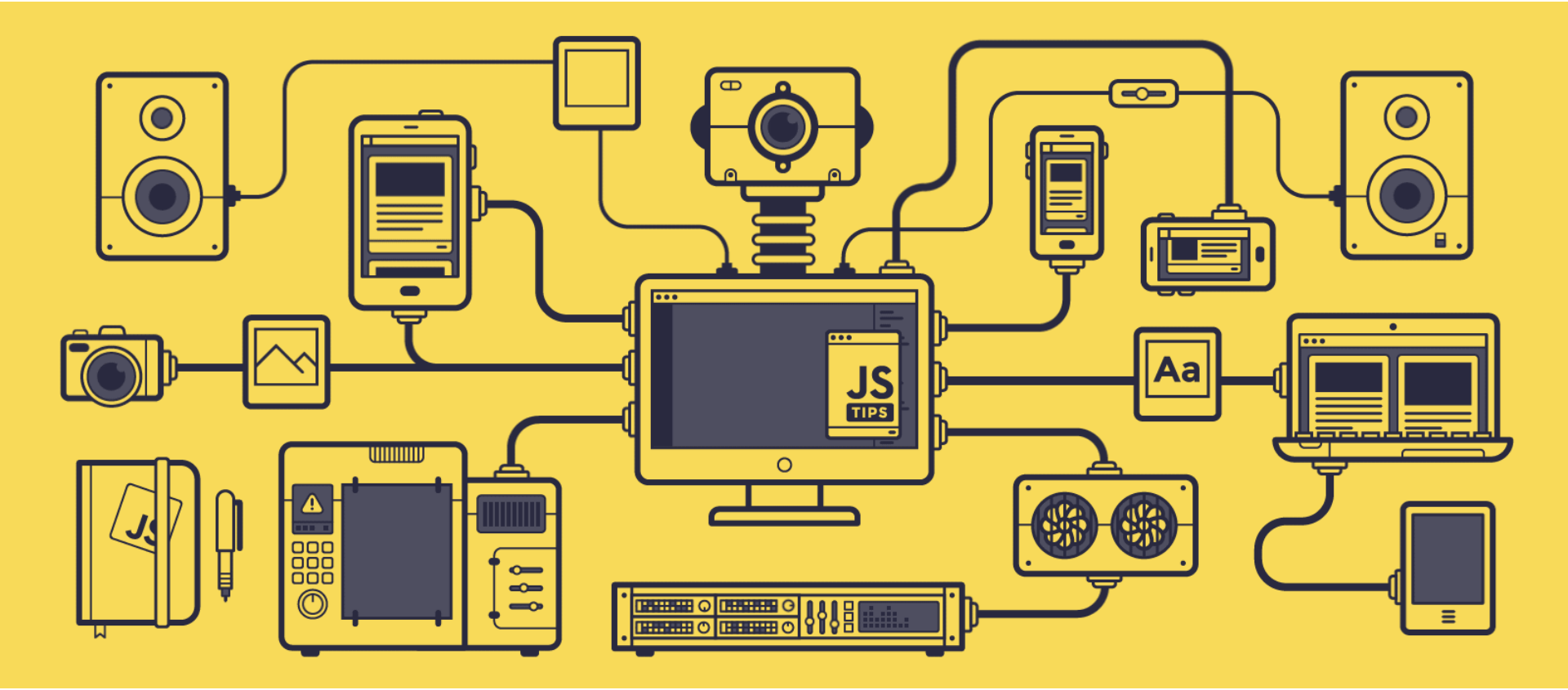
FORMATION ANGULAR 8

Octobre 2019

Mustapha BARGACH

retengr
knowledge transfer

PRÉSENTATION



1. Présentation des outils
2. Javascript et Typescript
3. Angular CLI
4. Angular
5. Ecosystème
6. Architecture logicielle

POURQUOI ANGULAR ?

- Pour l'utilisateur
  - Responsive Web Design
  - Performance
  - RIA

- Pour le développeur
  - Réduction des coûts de développement
  - Responsive Web Design
  - Lisibilité : Sémantique et simplicité
  - Modularité
  - Portabilité
  - Documentation et support de la communauté
  - Testabilité

ET POURQUOI PAS LES AUTRES ?

OUTILS

NODE ET NPM

Plateforme JavaScript

Intègre un serveur HTTP

Implémentation de CommonJS

Nativement scalable

Possibilité d'applications full stack JS

Gestionnaire des dépendances et de la configuration du projet

Suit un formalisme : package.json

Méta-données du projet

Dépendances de développement et de production

Gestion des scripts

OUTILS

GIT

Logiciel de gestion de version décentralisée

Forte capacité de débrancher et de fusionner

Fonctionnalités dédiées au développement communautaire

Performance

OUTILS

BUILD

OUTILS

IDE

Produit JetBrains

IDE à part entière, supporte des fonctionnalités avancées de développement

Payant

Produit Microsoft

Entre l'éditeur et l'IDE

Gratuit

Customisable par plugins

1. Présentation des outils
2. **Javascript et Typescript**
3. Angular CLI
4. Angular
5. Ecosystème
6. Architecture logicielle

JAVASCRIPT / TYPESCRIPT

UN PEU D'HISTOIRE

JAVASCRIPT / TYPESCRIPT

ECMASCRIPT

- Standard de spécifications mise en œuvre notamment pour Javascript
- Première version en juin 1997
- Dernière version ES 10 (ES 2019)
- Transcompilateur, tel que Babel.js, pour définir une cible de développement et une cible de production

JAVASCRIPT / TYPESCRIPT

PROTOTYPAGE

- Sensiblement différent de la programmation objet
- Les prototypes sont muables à l'inverse des objets
- Les déclarations de fonction sont des prototypes
- Possibilité d'hériter d'un prototype au travers de chaines de prototypages et d'ouvrir au polymorphisme

JAVASCRIPT / TYPESCRIPT

TYPAGE

- Javascript est un langage non typé avec inférence de type
- Typescript est un sucre syntaxique permettant d'apporter une programmation orientée objet, typée fortement et proposant des types génériques
- Typescript appartient à la vision du développeur et nécessite d'être transpilé en Javascript pour le runtime
- Ouverture des navigateurs pour interpréter directement du typescript

JAVASCRIPT / TYPESCRIPT

DÉCLARATIONS DE VARIABLES

- var : ancienne définition d'une variable en Javascript. Est soumis au hoisting et à l'inférence de type.

- let : définition d'une variable en Typescript et en Javascript (ES 7). N'est pas soumis au hoisting.

- const : déclaration d'une constante immuable. Respecte les mêmes paradigmes que le final d'une variable en Java

JAVASCRIPT / TYPESCRIPT

COLLECTIONS

- Array : le prototype propose un ensemble de fonctions orientés streams, comme par exemple map, reduce, filter ou encore sort.

- Map : collection indexée par clé / valeur

- Set : collection indexée par valeur unique

JAVASCRIPT / TYPESCRIPT

FALSY ET TRUTHY VALUES

- Toujours faux :
  - false
  - 0 (zero numérique)
  - '' ou "" (chaîne de caractères vide)
  - null
  - undefined
  - NaN

- Toujours vrai :
  - '0' (en tant que chaîne de caractères)
  - 'false' (en tant que chaîne de caractères)
  - [] (un tableau vide)
  - {} (un objet vide)
  - function(){} (une fonction “vide”)

JAVASCRIPT / TYPESCRIPT – TP

MANIPULATION DU DOM

- Aller sur GitHub : https://github.com/MoosArga/formation_angular et cliquer sur le lien du TP
- Dans la balise div#hobbies, afficher une liste non ordonnée du tableau hobs
- Dans la fonction changeCivilite, modifier la balise div#presentation pour afficher alternativement Bonjour Madame et Bonjour Monsieur
- Modifier la fonction précédente pour changer la couleur de fond de la balise div#presentation en fonction de Bonjour Madame et Bonjour
- Supprimer l'attribut onClick du bouton et associer l'évènement du click sur le bouton directement dans le code JavaScript

JAVASCRIPT / TYPESCRIPT

SYNTAXE TYPESCRIPT

```typescript
// Déclaration d'une variable (hors classe)
let message: string = 'Bonjour...'

// Déclaration d'une fonction
function concatener(a: string, b: string): string {
  return a + b;
}

// Déclaration d'une classe
class Employe {
  name: string;
  private matricule: number;
  constructor(name: string) {
    this.name = name;
  }
}

// Déclaration d'une interface
interface Matricule {
    a: number;
}
```

JAVASCRIPT / TYPESCRIPT

SYNTAXE TYPESCRIPT

```
// Valeur optionnelle
function add(a: number, b?: number): number

interface Matricule {
  a: number;
  version?: number;
}

// Abscence de retour
function fn(): void

// Typage abstrait
let nom: any = 'Nom'

// Typage implicite
let nom = 'Nom'
let chien = new Animal()

// Type union
function faisQQChose(a :string|number): string|boolean
```

JAVASCRIPT / TYPESCRIPT

HORROR SHOW

```
f=(a,b)=>a+b;

g=(a)=>(b)=>a+b;

k=(a,b)=>{a+b}

[1,2,3].reduce((a,b)=>a+b,0)
```

```
civilizeMoi:(string)=>string=(nom)=>{return "M."+nom};

civilizeMoi(nom:string):string {return "M."+nom};

constructor(private http:HttpClient) { }

http:HttpClient;
constructor(http:HttpClient) {
    this.http=http;
}
```

1. Présentation des outils
2. Javascript et Typescript
3. **Angular CLI**
4. Angular
5. Ecosystème
6. Architecture logicielle

ANGULAR CLI

FONCTIONNEMENT

- Outil en ligne de commande
- Création de projet et gestion des méta données associées
- Génération de composants, modules, classes, directives, …
- Gestion du packaging
- Déploiement
- Gestion d’un linter TypeScript
- Gestion des tests automatisés (unitaires et E2E)
- Intégration de bibliothèques

ANGULAR CLI

COMMANDES

- Toutes les commandes commencent par *ng*
- Création de projet : *ng new <nomduprojet>*
- Génération de composants : *ng generate …*
- Gestion du packaging : *ng build*
- Déploiement local : *ng serve*

ANGULAR CLI - TP

GÉNÉRATION

- Installer la bibliothèque NPM @angular/cli en global :

SCSS

```scss
section {
    height: 100px;
    width: 100px;

    .class-one {
        height: 50px;
        width: 50px;

        .button {
            color: #074e68;
        }
    }
}
```

CSS

```css
section {
    height: 100px;
    width: 100px;
}

section .class-one {
    height: 50px;
    width: 50px;
}

section .class-one .button {
    color: #074e68;
}
```

1. Présentation des outils
2. Javascript et Typescript
3. Angular CLI
4. **Angular**
5. Ecosystème
6. Architecture logicielle

ANGULAR

INTRODUCTION

- Framework Javascript libre et open source de Google. Première version en 2009.

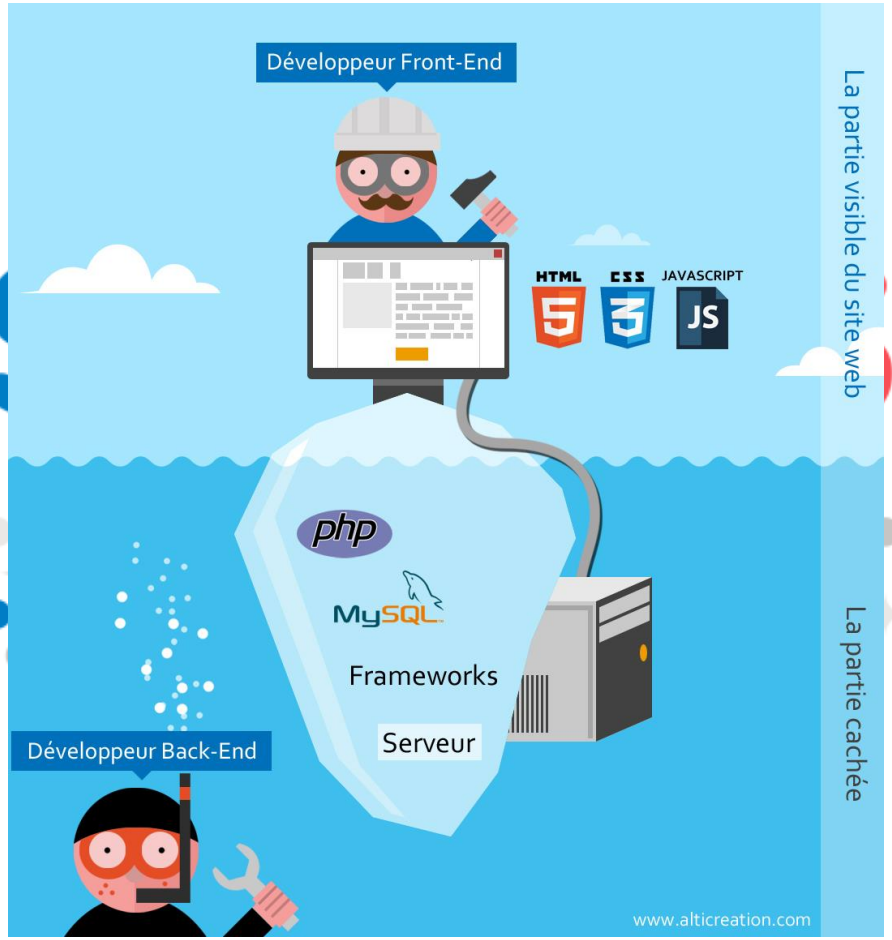
- Dans sa première version, orientée MVVM. Aujourd'hui, il est dans la trajectoire des architectures orientées composants.

- Le framework est repensé entre la version 1 et 2 pour gagner en simplicité, en performance, en lisibilité et en structure.

- Depuis la version 6, le framework intègre Angular CLI.

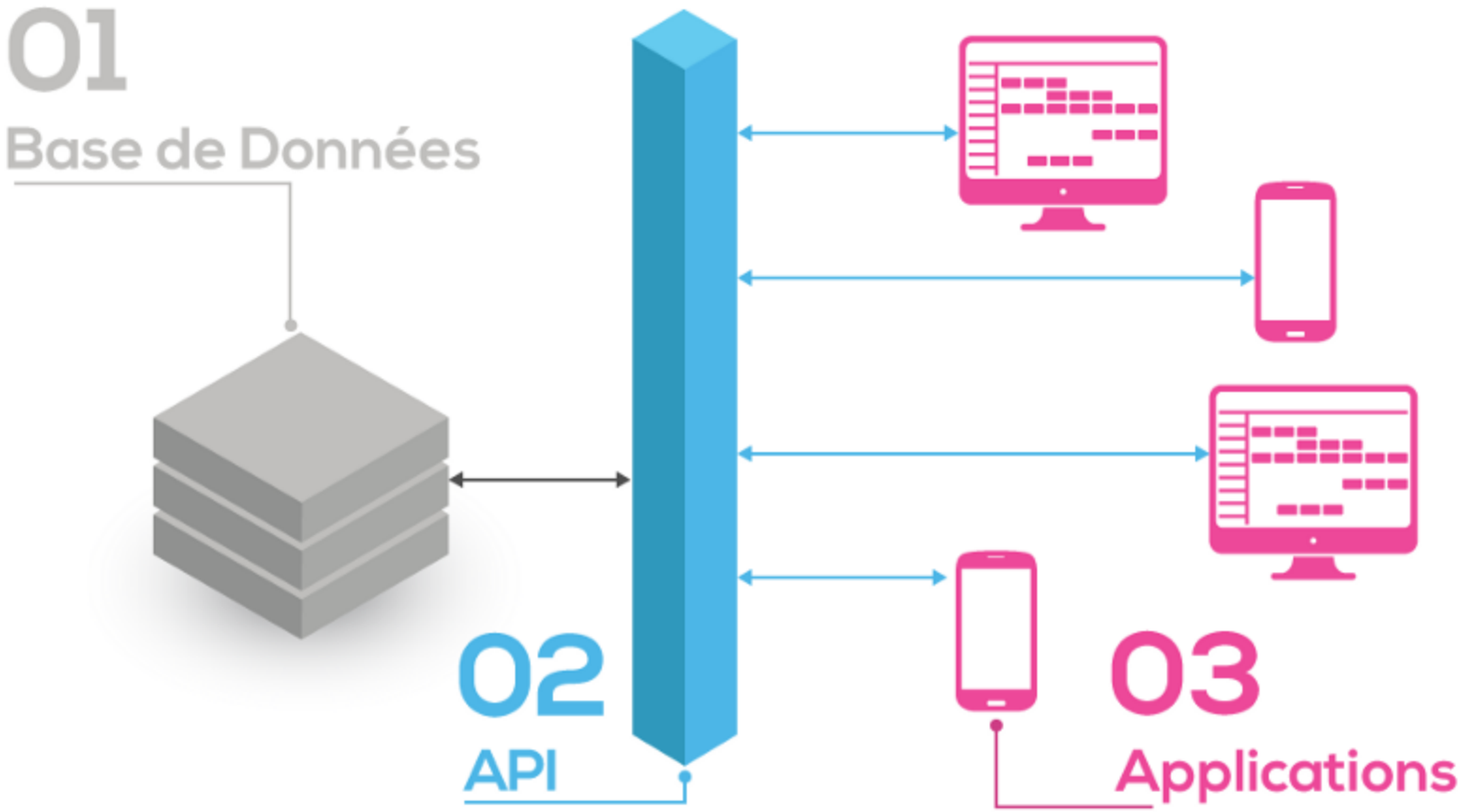
ANGULAR

LES ESSENTIELS



ANGULAR

LES APIS



ANGULAR

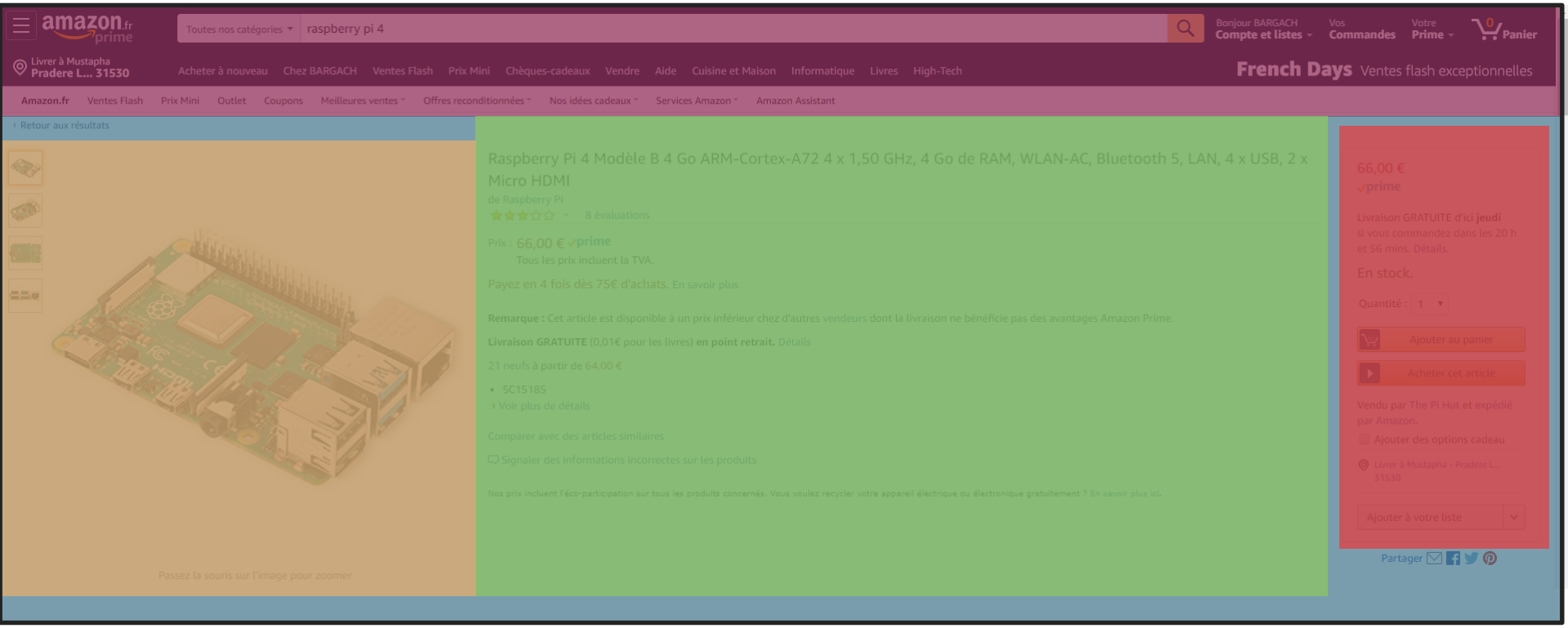
LES APIS

- Installer la librairie json-server : *npm install –g json-server*
- Au sein du repository, lancer la commande :
  json-server --watch formations.json
- Depuis un navigateur, essayer les urls suivantes :
  - *http://localhost:3000/formations*
  - *http://localhost:3000/formations?nom_like=scr*

Type	Décorateur
Module : bloc cohésif de fonctionnalités. On parle de *root module* et de *feature module*	@NgModule
Composant : élément dédié au contrôle d'une vue. Il est rattaché à un template (vue) et gère la communication entre le composant et la vue	@Component
Service : élément dédié à la gestion d'un service fonctionnel, d'une valeur. Il est injecté dans les modules ou dans les composants	@Injectable
Directive : élément dédié à restructurer et à modifier d'autres éléments. Le composant est une spécification de la Directive.	@Directive
Filtre : élément dédié à la transformation de données pour correspondre à une syntaxe lisible	@Pipe

ANGULAR

LES ESSENTIELS



Module App	Module Product	Module Shared
Component Layout	Component Product	Component Cart-Bar
Component Header	Component Product-Viz	
	Component Product-Details	

ANGULAR

LES ESSENTIELS : MODULE

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

@NgModule({
  declarations: [], // Déclarations de composants, de directives et de filtres associés à ce module
  imports: [], // Imports d'autres modules
  providers: [], // Déclaration des services injectés par ce module
  exports: [], // Exports des composants, filtres et directives
  bootstrap: [] // Composant sur lequel le module doit démarrer (uniquement pour le root module)
})
export class AppModule { }
```

ng **g**enerate **m**odule « nom »

ANGULAR - TP

LES ESSENTIELS : MODULE

- Créer un module features/formation
- Importer le module dans le module app

ng **g**enerate **m**odule « nom »

ANGULAR

LES ESSENTIELS : COMPOSANT

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

@Component({
  selector: 'app-home', // Sélecteur HTML pour intégrer ce composant
  templateUrl: './home.component.html', // Template de la vue associée à ce composant
  providers: [], // Déclarations des services injectés par ce composant
  styleUrls: ['./home.component.scss'] // Déclaration des styles associés à ce composant et ce template
})
export class HomeComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

ng **g**enerate **c**omponent « nom »

ANGULAR - TP

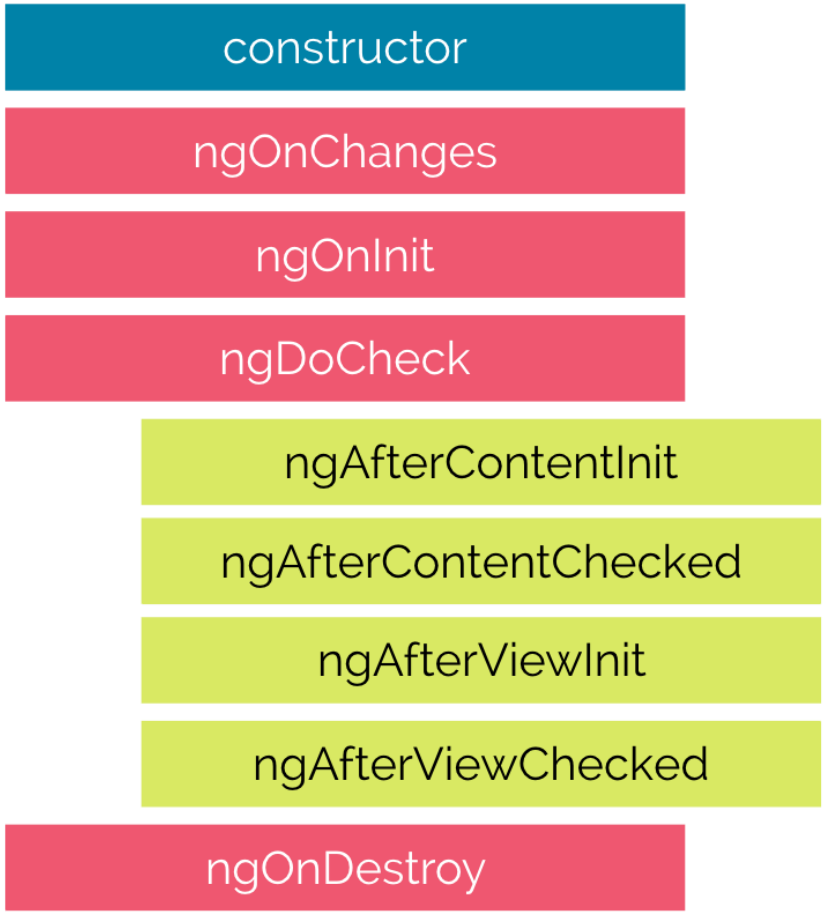
LES ESSENTIELS : COMPOSANT

- Créer un composant formation au sein du module formation
- Dans le template HTML, ajouter :
  - Un titre h1 avec comme texte : « Mes Formations »
  - Une div vide
  - Une div, contenant la date du jour (saisir arbitrairement la date du jour dans un format de votre choix)
- Exporter ce composant dans le module formation
- Ajouter le sélecteur de ce composant dans le template du composant app (app.component.html)

ng **g**enerate **c**omponent « nom »

ANGULAR

LES ESSENTIELS : COMPOSANT LIFE CYCLE HOOKS



ANGULAR

LES ESSENTIELS : SERVICE

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

  constructor() { }

}
```

ng **g**enerate **s**ervice « nom »

ANGULAR - TP

LES ESSENTIELS : SERVICE

- Créer un service dans le dossier shared/service/formation-dao

ng **g**enerate **s**ervice « nom »

ANGULAR

LES ESSENTIELS : DIRECTIVE

```
import { Directive } from '@angular/core'
;
@Directive({
  selector: '[highlight]'
})
export class HighlightDirective {

  constructor() { }

}
```

ng **g**enerate **d**irective « nom »

ANGULAR

LES ESSENTIELS : FILTRE

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

@Pipe({
  name: 'civilite'
})
export class CivilitePipe implements PipeTransform {

  transform(value: any, args?: any): any {
    return (args === 0 ? "Monsieur" : "Madame") + " " + value;
  }

}
```

ng **g**enerate **p**ipe « nom »

ANGULAR - TP

LES ESSENTIELS : FILTRE

- Créer un module, appelé shared
- Créer un filtre, appelé notation au sein du dossier shared/filtre
- Dans la fonction de transformation, retourner la valeur en paramètre suffixée de «/10 »
- Exporter le filtre dans le module shared
- Importer le module shared dans le module formation

ng **g**enerate **p**ipe « nom »

ANGULAR

BINDING DE PROPRIÉTÉ

- Le property Binding est un mécanisme de Data Binding "one way". Il permet de répercuter dans le DOM les valeurs des propriétés du composant. L'annotation associée est […]

```
<h1 [title]="titre.name"></h1>
```

- Il est aussi possible d'utiliser des fonctions

```
<h1 [title]="getName()"></h1>
```

ANGULAR

INTERPOLATION

- L'interpolation est du sucre syntaxique sur du binding de propriété
- Sensible à la casse
- Par exemple

```
<h1>{{ titre.value }}</h1>
```

équivaut à

```
<h1 [textContent]="titre.value"></h1>
```

ANGULAR - TP

INTERPOLATION

- Variabiliser le titre Mes formations dans le composant formation
- Interpoler cette nouvelle variable dans le titre h1
- Variabiliser la date du jour dans le composant formation
- Interpoler cette nouvelle variable dans la dernière div
- Ajouter un filtre sur l'interpolation de la date l'afficher sous la forme jour/mois/année

ANGULAR

BINDING D'ÉVÈNEMENT

- L' event Binding est également un mécanisme de binding "one way". il permet de répercuter dans le composant les évènements du DOM (clic, focus, survol, change, …). L'annotation associée est (…)

```
<button (click)="hideDetails()">Réduire</button>
<input type="text" (keydown.space)="refreshSearch()"/>
```

ANGULAR

NGMODEL

- Le ng model est le binding « two-way ». En effet, il permet de répercuter dans le DOM les valeurs des propriétés du composant, et inversement. L'annotation associée est [(ngModel)], appelée banana-box

```
<input type="text" [(ngModel)]="currentUser" name="user-field"/>
```

Attention : pour utiliser ngModel il faut importer FormsModule

ANGULAR - TP

ÉVÈNEMENT ET NGMODEL

- Dans le template du composant formation, ajouter en bas de page une div contenant :
  - Un formulaire contenant un input et 2 boutons
- Dans le composant formation, ajouter une variable de type number nbFormation et 2 méthodes : ajouterFormation et retirerFormation. Ces 2 méthodes incrémenteront et décrémenteront respectivement la variable nbFormation
- Associer la variable nbFormation à l'input par ngModel
- Associer les méthodes au clic de chacun des 2 boutons. Renseigner un libellé pour chaque bouton
- Modifier le ng model par un binding de propriété sur la valeur de l'input

ANGULAR

VARIABLES LOCALES

- Les variables locales permettent d'assigner une variable à un élément du DOM afin de s'y référer directement depuis le DOM.

```
<input type="text" #name/>
<label>{{ name.value }}</label>

<input type="text" #searchField/>
<button (click)="searchField.focus()">Vas chercher !</button>
```

ANGULAR

LES ESSENTIELS : HTTP

- Angular offre nativement un client HTTP pour consommer les API Rest du backend, au travers du module HttpClientModule
- Consommation d'une requête GET :

```
import { Injectable } from "@angular/core";
import { HttpClient } from '@angular/common/http'
import { Observable } from "rxjs";

@Injectable()
export class RechercheService {

  constructor(private http: HttpClient) { }

  vasChercher(): Observable<any> {
    return this.http.get('api/search');
  }

}
```

ANGULAR

DIRECTIVES DE STRUCTURES

- ***ngIf :** gestion de l'affichage contextuelle du template

```
<div *ngIf="choix && choix.length; else noChoix">

    [...]

</div>
<ng-template #noChoix>
  Pas de choix de saisi !!
</ng-template>
```

- ***ngFor :** répéter une vue selon une liste d'entrants

```
<div *ngIf="choix && choix.length; else noChoix">

    <ul>
      <li *ngFor="let c of choix">{{ c }}</li>
    </ul>

</div>
<ng-template #noChoix>
  Pas de choix de saisi !!
</ng-template>
```

ANGULAR

DIRECTIVES DE STRUCTURES

- **ngSwitch :** gestion de l'affichage contextuelle du template

```
<div [ngSwitch]="nbMessage">
  <div *ngSwitchCase=0>Vous n'avez pas de nouveaux messages</div>
  <div *ngSwitchCase=1>Vous avez 1 nouveau message</div>
  <div *ngSwitchDefault>Vous avez {{ nbMessage}} nouveaux messages</div
</div>
```

ANGULAR - TP

HTTP ET DIRECTIVES DE STRUCTURES

- Créer une classe formation dans shared/model, contenant les propriétés suivantes :
  - nom de type string
  - chargeH de type number
  - typeF de type string
  - note de type number
- Dans le service formation-dao, intégrer le client HTTP, ajouter une fonction renvoyant un Observable d'un tableau de formation.
- Dans cette fonction, retourner l'appel GET vers l'API /api/formations.
- Dans le composant formation, importer le service formation-dao et créer une variable formation$ de type Observable d'un tableau de formation
- Afficher dans la div intermédiaire du composant, une liste non ordonnée des formations : Nom (en minuscule) : nombres d'heures restants (chargeH)

ANGULAR

DIRECTIVES DE TEMPLATE

- **ngStyle :** appliquer un style à un template

```
<div class="progress-bar">
  <div class="progression" [style.width.%]="avancement"></div>
</div>

<div class="progress-bar">
    <div class="progression" [ngStyle]="{ width: avancement, 'background-color': avancementColor }"></div>
</div>
```

- **ngClass :** appliquer une classe conditionnellement à un template

```
<div [class.success]="isSuccess()"></div>
<div [ngClass]="{ 'success': isSuccess(), 'error': !isSuccess() }"></div>
<div [ngClass]="isSuccess() ? 'success' : 'error'"></div>
```

ANGULAR

QUELQUES DIRECTIVES UTILES

- **ng-content :** pouvoir projeter un contenu au sein d'un composant
- **ng-template :** définition d'un template de référence principalement pour les cas non passants des ***ngIf** et les validations de formulaires
- **ng-container :** création d'un élément non généré dans le DOM. Permet d'englober plusieurs éléments et de gérer la combinatoire ***ngIf** et ***ngFor**

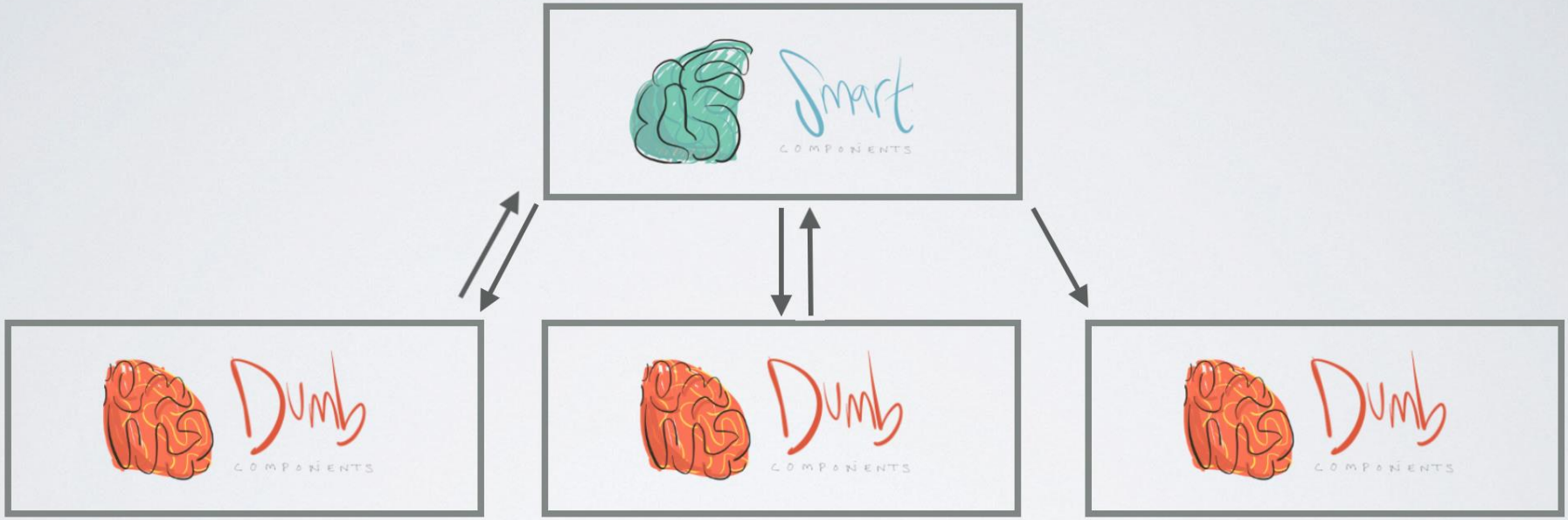
ANGULAR - TP

DIRECTIVES DE TEMPLATES

- Modifier la liste des formations pour afficher en rouge les formations dont la charge en heure restante est supérieure à 20 heures.
- Ajouter un texte pour donner la note par formation en utilisant le filtre créé précédemment
- Pour les formations dont la charge en heure est égale à 0, afficher un texte de remplacement indiquant : « Formation <nom de la formation> est finie »

ANGULAR

INTERACTION INTER-COMPOSANTS



ANGULAR

INTERACTION INTER-COMPOSANTS

- **@Input :** Communication d'un model à un composant fils

```
import { Component, OnInit, Input } from '@angular/core'
;
@Component({
  selector: 'app-progress-bar',
  templateUrl: './progress-bar.component.html',
  styleUrls: ['./progress-bar.component.scss']
})
export class ProgressBarComponent implements OnInit {

  @Input() avancement;

  constructor() { }

  ngOnInit() {
  }

}
```

ANGULAR

INTERACTION INTER-COMPOSANTS

- **@Input :** Communication d'un model à un composant fils
  - Template

```
<div class="progress-bar">
  <div class="progression" [style.width.%]="avancement"></div>
</div>
```

  - Utilisation depuis un autre composant

```
<app-progress-bar [avancement]="avancement"></app-progress-bar>
```

ANGULAR

INTERACTION INTER-COMPOSANTS

- **@Output :** Communication d'un évènement à un composant parent

```
import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core'
;
@Component({
  selector: 'app-progress-bar',
  templateUrl: './progress-bar.component.html',
  styleUrls: ['./progress-bar.component.scss']
})
export class ProgressBarComponent {

  @Input() avancement;
  @Output() catchInteraction: EventEmitter<any> = new EventEmitter<any>();

  constructor() { }

  userWantsDetails(): void {
    this.catchInteraction.emit();
  }

}
```

ANGULAR

INTERACTION INTER-COMPOSANTS

- **@Output :** Communication d'un évènement à un composant parent
  - Template

```
<div class="progress-bar">
    <div class="progression" (click)="userWantsDetails()" [style.width.%]="avancement"></div>
</div>
```

  - Utilisation depuis un autre composant

```
<app-progress-bar [avancement]="avancement" (catchInteraction)="catchInteractionFromProgressBar()"></app-progress-bar>
```

INTERACTION INTER-COMPOSANTS - TP

- Créer un composant ProgressionBar dans le dossier shared/widget, rattaché au module shared et exporter le
- Ajouter une variable en input (@Input()), pour donner au composant la taille de la barre de chargement à présenter
- Intégrer ce composant, par son sélecteur HTML, dans le composant formation pour chaque formation. L'objectif est de présenter l'avancement de la note dans une barre de progression. (Il faudra multiplier la note par 10)
- Ajouter un évènement au survol de la barre de progression, dans le composant ProgressionBar, et renvoyer le sur un output (@Output()) avec un EventEmitter avec l'avancement courant
- Intercepter l'évènement renvoyé par le composant ProgressionBar dans le composant Formation et logger l'information : « L'utilisateur cherche quelque chose sur la formation avec l'état d'avancement <avancement>% »

ANGULAR

ROUTER

- Permet de gérer la navigation dans l'application au travers d'un module dédiée
- Une route = un composant
- Possibilité de passer des données, *query param* ou *route param*
- Possibilité d'ajouter des traitements en amont ou en aval de la route : Guards
- Le composant de la route est placé dans le router-outlet.

  <router-outlet></router-outlet>
- Il est possible d'avoir plusieurs router-outlet :
  - inclus les uns dans les autres (filiation de routes) ;
  - de même niveau en les nommant (routage parallèle) : named outlet

ANGULAR

ROUTER

- Il est possible d'intégrer des liens de navigations avec router link

```
<li><a class="l-menu-item" routerLinkActive="l-is-active-menu" routerLink="pointeuse">Badgeuse</a></li>
  <li><a class="l-menu-item" routerLinkActive="l-is-active-menu" [routerLink]="['decompte']">Décomptes</a></li>
```

ANGULAR
ROUTER

- Exemple de déclaration d'une route à la racine d'un projet :

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { InfosComponent } from './infos/infos.component';

const routes: Routes = [
  {
    path: 'lazy',
    loadChildren: './lazy/lazy.module#LazyModule', // Avant Angular 8
    loadChildren : () => import('./lazy/lazy.module').then(m => m.LazyModule), // Angular 8
  }
];

  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class ApplicationRoutingModule { }
```

ANGULAR

ROUTER : LAZY LOADING

- Chargement d'une route à la demande de l'utilisateur (lorsqu'il veut accéder à cette URL) :

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { InfosComponent } from './infos/infos.component';
import { NotfoundComponent } from './notfound/notfound.component';

const routes: Routes = [
  { path: 'users', loadChildren: 'user/user#UserModule' }
];
```

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { UserProfilComponent } from './user-profil/user-profil.component';
import { UserHabilitationsComponent } from './user-habilitations/user-habilitations.component'
;
const routes: Routes = [
  { path: '', component: UserProfilComponent },
  { path: 'auth', component: UserHabilitationsComponent },
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class UserRoutingModule { }
```

ANGULAR

ROUTER

- Exemple de déclaration de routes enfants

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { InfosComponent } from './infos/infos.component';
import { NotfoundComponent } from './notfound/notfound.component';
import { AppComponent } from '../app.component';

const routes: Routes = [
  {
    path: 'myapp',
    component: AppComponent,
    children: [
      { path: '', pathMatch: 'full', redirectTo: 'infos' },
      { path: 'infos', component: InfosComponent },
      { path: 'users', loadChildren: 'user/user#UserModule' }
    ]
  }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class ApplicationRoutingModule { }
```

ANGULAR

ROUTER : GESTION DES PARAMÈTRES (PATH PARAM)

```
const routes: Routes = [
  { path: 'infos', component: InfosComponent },
  { path: 'infos/:id', component: InfoDetailComponent }
];
```

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { Subscription } from 'rxjs';

@Component({
  selector: 'app-info-detail',
  templateUrl: './info-detail.component.html',
  styleUrls: ['./info-detail.component.scss']
})
export class InfoDetailComponent implements OnInit {

  id: number;
  paramSub: Subscription;

  constructor(private route: ActivatedRoute) { }

  ngOnInit() {
    this.paramSub = this.route.params.subscribe(params => {
      this.id = params['id'];
    });
  }
```

ANGULAR

ROUTER : GESTION DES PARAMÈTRES (QUERY PARAM)

```
const routes: Routes = [
  { path: 'infos', component: InfoDetailComponent }
];
```

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { Subscription } from 'rxjs';

@Component({
  selector: 'app-info-detail',
  templateUrl: './info-detail.component.html',
  styleUrls: ['./info-detail.component.scss']
})
export class InfoDetailComponent implements OnInit {

  id: number;
  paramSub: Subscription;

  constructor(private route: ActivatedRoute) { }

  ngOnInit() {
    this.paramSub = this.route.queryParams.subscribe(params => {
      this.id = params['id'];
    });
  }

}
```

ANGULAR

ROUTER : RÉCUPÉRATION D'UNE DONNÉE STATIQUE

```
const routes: Routes = [
  { path: 'infos', component: InfoDetailComponent, data: { pageTitle: 'Actus Angular'} }
];
```

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { Subscription } from 'rxjs';

@Component({
  selector: 'app-info-detail',
  templateUrl: './info-detail.component.html',
  styleUrls: ['./info-detail.component.scss']
})
export class InfoDetailComponent implements OnInit {

  title: string;
  dataSub: Subscription;

  constructor(private route: ActivatedRoute) { }

  ngOnInit() {
    this.dataSub = this.route.data.subscribe(data => {
      this.title = data.pageTitle;
    });
  }

}
```

ANGULAR

ROUTER : RÉSOLUTION

- Déclaration d'une route avec une résolution :

```
const routes: Routes = [
  { path: 'infos', component: InfoDetailComponent, resolve: { info: ResolverInfoService} }
];
```

- Définition du service de résolution:

```
import { Injectable } from '@angular/core';
import { Resolve, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';
import { HttpClient } from '@angular/common/http';

@Injectable()
export class ResolverInfoService implements Resolve<any> {

  constructor(private http: HttpClient) { }

  resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
    return this.http.get('api/infos');
  }

}
```

ANGULAR

ROUTER : RÉSOLUTION

- Utilisation de l'information résolue :

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { Subscription } from 'rxjs';

@Component({
  selector: 'app-info-detail',
  templateUrl: './info-detail.component.html',
  styleUrls: ['./info-detail.component.scss']
})
export class InfoDetailComponent implements OnInit {

  infos;
  dataSub: Subscription;

  constructor(private route: ActivatedRoute) { }

  ngOnInit() {
    this.dataSub = this.route.data.subscribe(data => {
      this.infos = data.infos;
    });
  }

}
```

ANGULAR

ROUTER : GUARDS

- Les guards sont des contrôles à la navigation qui s'appliquent sur une route.

- 4 types de guards :
  - CanActivate : valide l'accès à la route
  - CanActivateChild : valide l'accès aux enfants de la route
  - CanDeactivate : valide que l'on a le droit de quitter la route
  - CanLoad : valide que l'on peut charger la route en lazy loading

ANGULAR

ROUTER : GUARDS

```
const routes: Routes = [
  { path: 'infos', component: InfoDetailComponent, canActivate: [HabilitationInfoService] }
];
```

```
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';
import { Observable } from 'rxjs';
import { UserService } from '../user/user.service';

@Injectable()
export class HabilitationInfoService implements CanActivate {

  constructor(private userService: UserService) { }

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<any
> { return this.userService.isAuthorized();
  }
}
```

ANGULAR

SYNTHÈSE

- Les solutions de routage :

Type de solution	Avantages	Inconvénients
Routage centralisé avec références des components de chaque module	Simple	Pas en accord avec la spécification de *features modules* Peut devenir illisible et très verbeux
Routages déportés dans chaque *feature module*, gestion des routes filles par Lazy Loading	Respect de la spécification *feature module* Lisibilité Extensibilité / Maintenabilité	Gestion du lazy loading à surcharger Déclaration par référence du path du module et non par component

ANGULAR - TP

ROUTER

- Créer une lien dans le composant App avec un RouterLink vers /formation
- Déporter l'affichage du composant Formation sur la route /formation
  - Une première fois par référence du component formation
  - Une seconde fois par lazy loading du module formation
- Déporter le titre « Mes formations » en data de la route précédemment crée et récupérer cette data dans le component formation
- Créer une nouvelle route /formation/:nomFormation avec un nouveau composant FormationDetail. A l'initialisation du composant, récupérer le paramètre dans la route et rechercher la formation associée.

ANGULAR - TP

ROUTER

- Créer une nouvelle route /formation/formation-detail avec le composant FormationDetail. A l'initialisation du composant, récupérer le query param (nomFormation) et rechercher la formation associée.
- Créer un composant layout/error et lui associer la route *eager* error dans l'app routing.
- Créer un nouveau service : UserService (shared/service/User), qui permet de rechercher un utilisateur par son id.
- Créer un service qui aura pour responsabilité de rechercher l'utilisateur courant (bouchonné sur l'id 1)
- Créer une guard UserGuard (shared/guard/User) sur la route /formation qui ne permet d'accéder à la route que si l'utilisateur courant (appel du service) a le rôle administrateur. Si l'utilisateur n'est pas habilité, l'utilisateur est redirigé sur /error.

ANGULAR

GESTION DE LA VALIDATION

- Deux types de validation pour les formulaires :
- **Template-Driven :** Pilotée par les templates HTML
  - Simple pour des formulaires simples
  - Parasite la lecture des templates
  - Très difficilement testable
- **Model-Driven :** Appelée aussi Reactive Form Validation. Elle est pilotée par le component
  - Verbeux
  - Extensible et Maintenable
  - Testable

ANGULAR

VALIDATION - TERMINOLOGIE ANGULAR

- **valid** : si le champ ou le formulaire est valide avec les validations et contraintes qui lui sont appliquées
- **errors** : objet contenant les erreurs du champ ou du formulaire
- **dirty** : faux jusqu'à ce que l'utilisateur modifie le champ ou du formulaire
- **pristine** : opposé de dirty
- **touched** : faux jusqu'à ce que l'utilisateur soit entré et sorti du champ ou d’un des champs du formulaire
- **untouched** : opposé de touched

VALIDATION – LES VALIDATEURS

- **min/max :** valeurs minimale et maximale admises pour un champ de type numérique
- **required :** champ requis
- **requiredTrue :** valeur à vrai pour un champ de type booléen
- **email :** respect le pattern d'une adresse mail
- **minLength/maxLength :** tailles minimale et maximale d'un champ
- **pattern :** respect d'un pattern formulé en expression régulière
- **Et pour le reste ?** A vous de les créer au travers d'une fonction qui renvoie un objet ValidatorFn en Model-Driven ou une directive implémentant l'interface Validator en Template-Driven

ANGULAR

VALIDATION – TEMPLATE DRIVEN

```
<input type="text" [(ngModel)]="candidat.name" name="field-name" required minlength="4" pattern="/[a-zA-z]*/g" #name="ngModel">

<div *ngIf="name.invalid && (name.dirty || name.touched)">
  <div *ngIf="name.errors.required">
    Le nom est requis
  </div>
  <div *ngIf="name.errors.minlength">
    Le nom doit faire au moins 4 caractères
  </div>
  <div *ngIf="name.errors.pattern">
    Le nom ne doit comporter que des lettres
  </div>
</div>
```

ANGULAR

VALIDATION – MODEL DRIVEN

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { Subscription } from 'rxjs';
import { FormControl, Validators } from '@angular/forms';

@Component({
  selector: 'app-info-detail',
  templateUrl: './info-detail.component.html',
  styleUrls: ['./info-detail.component.scss']
})
export class InfoDetailComponent implements OnInit {

  name;

  constructor(private route: ActivatedRoute) { }

  ngOnInit() {
    this.name = new FormControl('',
    [ Validators.required, Validators.minLength(4), Validators.pattern('/[a-zA-Z]*/')]);
  }

}
```

ANGULAR

VALIDATION – MODEL DRIVEN

```
<input type="text" formControlName="name">

<div *ngIf="name.invalid && (name.dirty || name.touched)">
  <div *ngIf="name.errors.required">
    Le nom est requis
  </div>
  <div *ngIf="name.errors.minlength">
    Le nom doit faire au moins 4 caractères
  </div>
  <div *ngIf="name.errors.pattern">
    Le nom ne doit comporter que des lettres
  </div>
</div>
```

ANGULAR

VALIDATION – GESTION DES STYLES

- Les classes apposées par Angular sont :
  - .ng-valid
  - .ng-invalid
  - .ng-pending
  - .ng-pristine
  - .ng-dirty
  - .ng-untouched
  - .ng-touched

```
input.ng-valid {
  border-left: 1px solid green;
}

input.ng-invalid {
  border-left: 1px solid red;
}
```

ANGULAR

VALIDATION – TP

- Créer un formulaire avec un champ de saisie en dernière position dans le template de formation. Positionner un label pour le champ : « Titre formation »
- Créer une variable tFormation de type string et associez-là au template de cet input.
- Ajouter les validateurs suivants : minlength(2) et un pattern pour n'avoir que des lettres, dans le template html
- Définir une variable local dans cet input #tc et associer là à la directive ngModel
- En dessous de cette input, créer une balise pre et afficher le tableau d'erreurs de la variable tc (utiliser le filtre json pour un affichage brut)

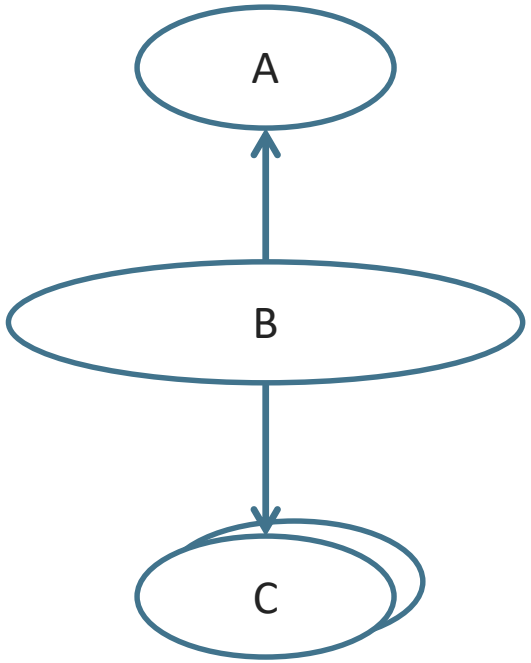
ANGULAR

VALIDATION – TP

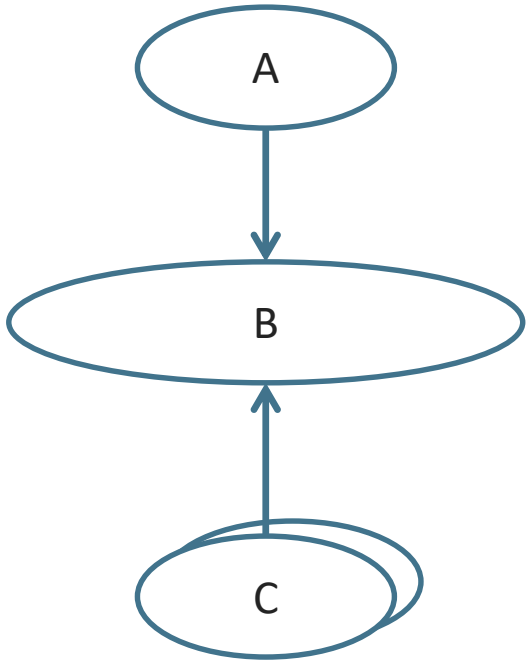
- Ajouter les styles suivants sur cet input :
  - Lorsque l'utilisateur saisit, une bordure bleue à gauche de l'input
  - Lorsque le champ est valide, une bordure verte à gauche de l'input
  - Lorsque le champ est invalide, une bordure rouge à gauche de l'input

- La programmation réactive est la programmation réagissant aux évènements. Le système détecte et prend en compte les changements à l'inverse d'une programmation impérative.

Impératif



Réactif



ANGULAR

PROGRAMMATION RÉACTIVE : PROMISE

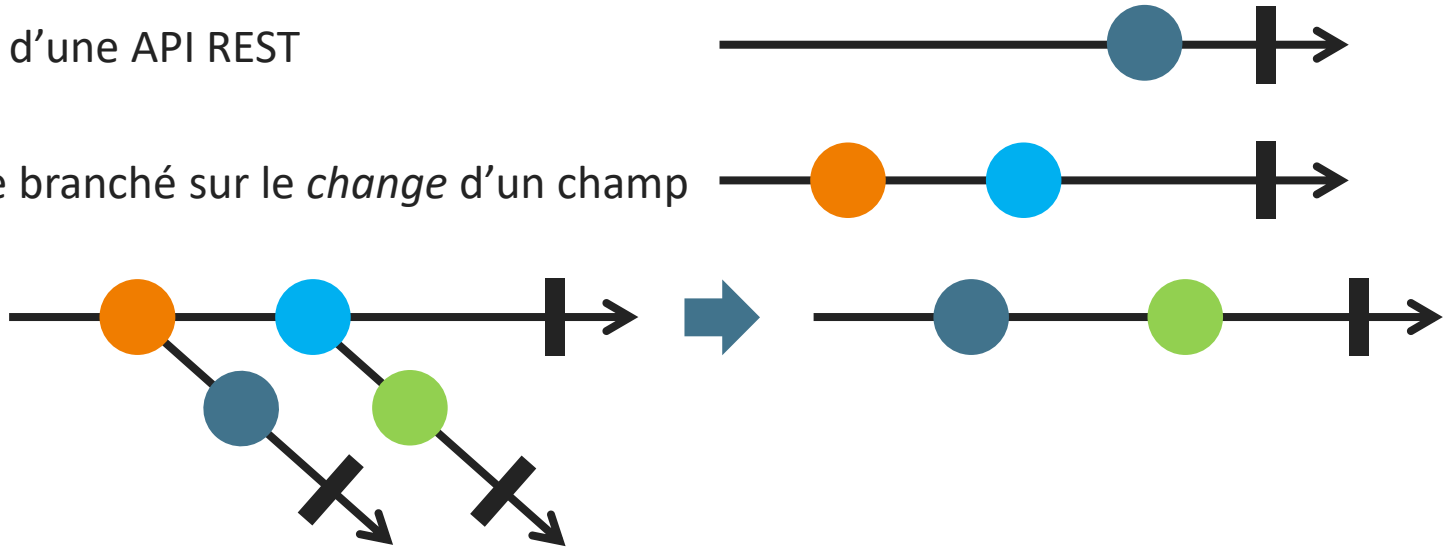
- Une Promise représente une valeur qui peut être disponible maintenant, dans le futur voire jamais.
- Une Promise est dans un de ces états :
  - *pending (en attente)* : état initial, la promesse n'est ni remplie, ni rompue
  - *fulfilled (tenue*) : l'opération a réussi
  - *rejected (rompue)* : l'opération a échoué
- Une Promise est unique et ne renvoie qu'une seule valeur (objet, tableau,…)
- Une Promise ne peut pas être annulée

PROGRAMMATION RÉACTIVE : OBSERVABLE

- Un observable représente un flux de plusieurs valeurs séquencé dans le temps.
- Un observable admet 3 types d'évènements :
  - **Next :** appelé pour chaque élément de la séquence
  - **Error :** appelé en cas d'erreur
  - **Completed :** appelé en fin de séquence
- Un observable envoie un ensemble de valeur (objet, tableau)
- Un observable peut être annulé.
- L'objet Observable propose un ensemble de méthodes pour traiter les flux, à l'image de la gestion des tableaux en Javascript (map, filter, …)

- Le flux d'un observable est représenté ainsi :

- Chaînage d'opérations:
  - Appel Http d'une API REST
  - Observable branché sur le *change* d'un champ
  - Chaîner



ANGULAR

PROGRAMMATION RÉACTIVE : OBSERVABLE - TP

- Au dessus de la liste des formations, créer un input.
  Dans le composant créer un FormControl nomFormation et associer le à l'input créé.

- Créer une nouvelle fonction dans le service formation-dao qui renvoie aussi un observable d'un tableau de formations et qui a pour paramètre un string. Renvoyer le résultat de l'API /api/formations avec comme query param "nom_like" est égal au critère. Si la chaîne est vide, renvoyer l'ensemble des formations (findAll)

- À l'initialisation du composant, récupérer les valeurs en entrée du form control et chaîner les opérations suivantes :
  - Initialiser la recherche avec la valeur vide
  - Attendre 1 sec entre chaque changement
  - Ne pas considerer les valeurs répétées (deux fois la même saisie consecutive)
  - Ne considérer que les valeurs vides ou les valeurs de plus de 2 caractères
  - Transformer la valeur saisie en caractères minuscules et enlever les espaces avant et après
  - Récupérer l'observable du nouveau service de formation-dao pour ce changement

- Associer la variable formations à ce nouvel observable

ANGULAR

LES VERSIONS PRÉCÉDENTES

- Avant Angular 6, il y a principalement 2 changements majeurs :
  - L'injection des services est toujours déléguée à un composant ou un module (pas de méta providedIn)
  - Les opérateurs d'observables se chainent les uns à la suite des autres (absence de pipe)

ANGULAR
ET LA SUITE

- Angular 7
  - Amélioration des commandes d'intégration de librairies
  - Améliorations RIA
- Angular 8
  - Amélioration des performances et simplification des livrables
    - Changement de syntaxe pour le chargement des modules *lazy-loadés*
  - Améliorations des services workers

1. Présentation des outils
2. Javascript et Typescript
3. Angular CLI
4. Angular
5. **Ecosystème**
6. Architecture logicielle

ECOSYSTÈME

BACKEND

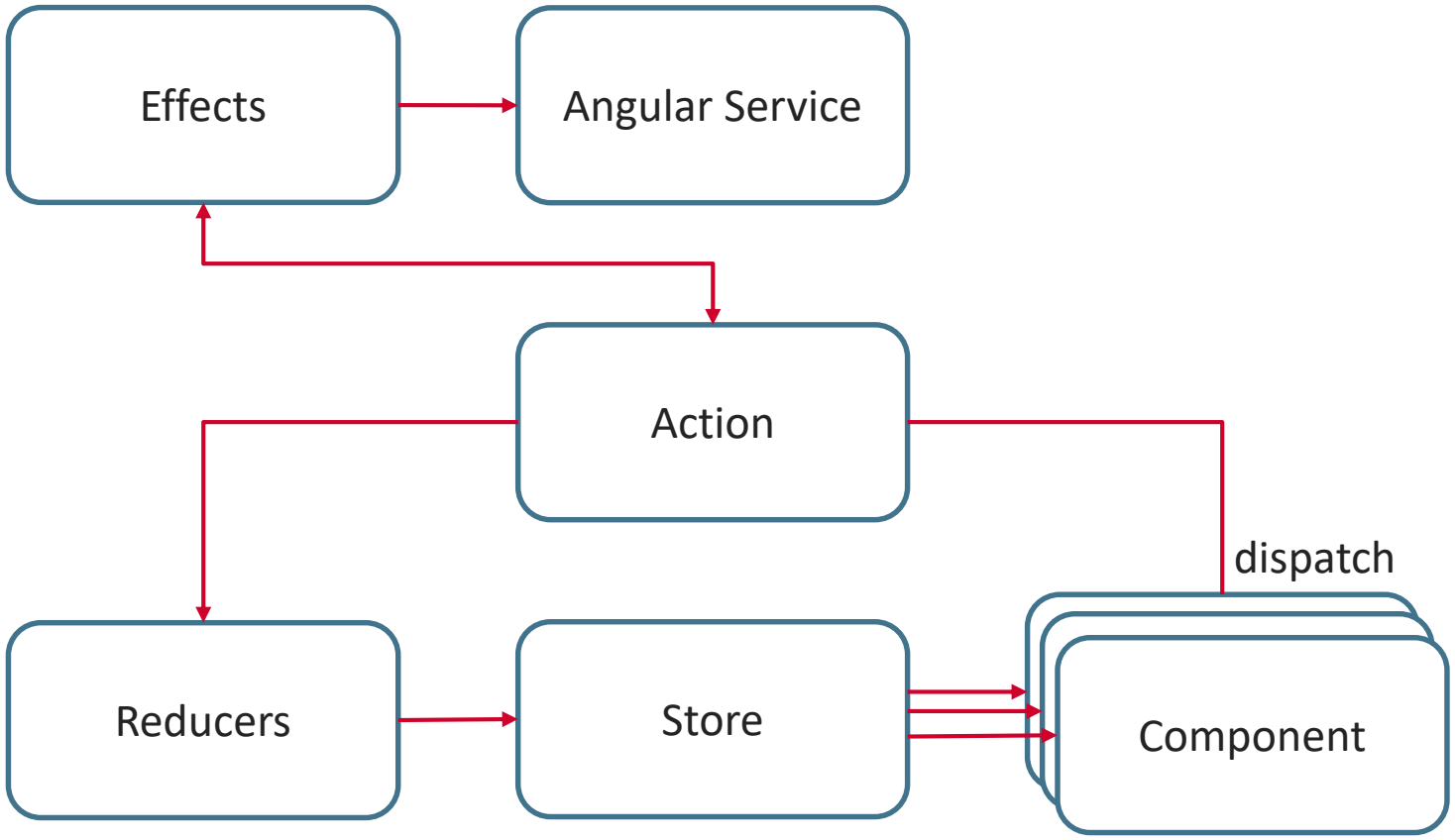
ECOSYSTÈME

NGRX

- Bibliothèque qui intègre le pattern Redux
  - Flux de données unidirectionnel
  - Séparation des responsabilités d'écritures et de lecture des données
  - L'état des données est formalisé par une suite séquentielle d'évènements
  - Intègre le pattern immutable

ECOSYSTÈME

NGRX



ECOSYSTÈME

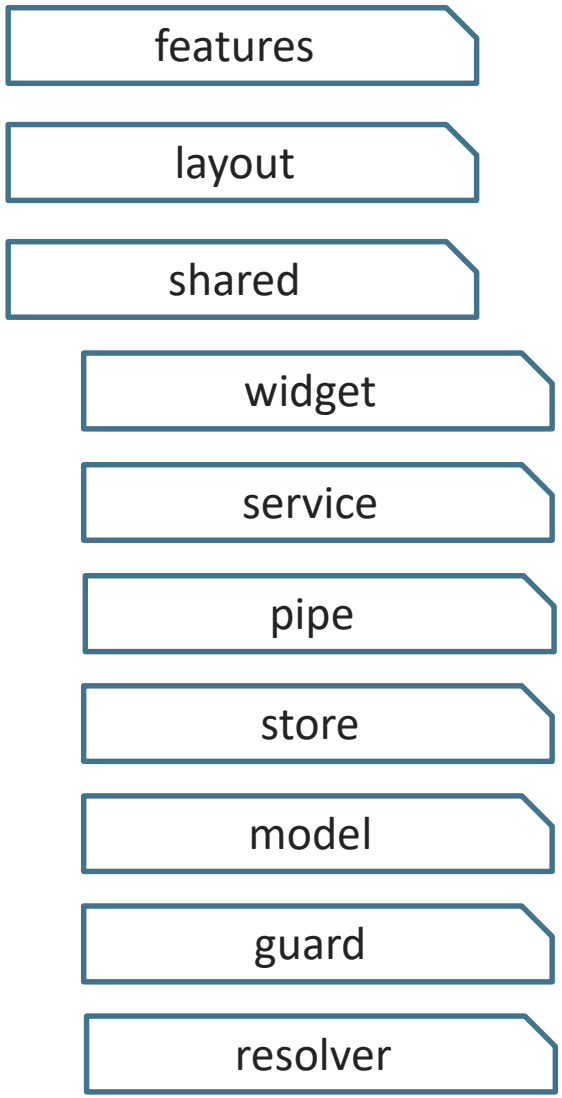
PROGRESSIVE WEB APP

- Depuis les versions 6
- ng add @angular/pwa
  - Intégration d'un service worker pour assurer le proxy HTTP
  - Intégration du manifest de l'application
- Pas de solution native pour le développement en local. Il faut déployer sur un serveur dédié tel que Node, http-server ou autre
- Vérification des critères PWA depuis LightHouse sur Google Chrome

1. Présentation des outils
2. Javascript et Typescript
3. Angular CLI
4. Angular
5. Ecosystème
6. **Architecture logicielle**

ARCHITECTURE LOGICIELLE

ARBORSECENCE



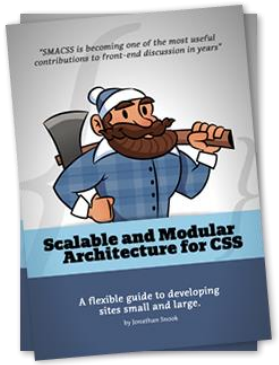
ARCHITECTURE LOGICIELLE

GESTION DES STYLES

Web Components

Component

Feuille de style



SMACSS

Base

Layout

Module

State

Theme

Variables

ARCHITECTURE LOGICIELLE

PACKAGING & DELIVERY

- ng build --prod
- Pour les projets Java : maven-frontend-plugin

```
<execution>
    <id>build frontend</id>
    <goals>
        <goal>npm</goal>
    </goals>
    <phase>generate-resources</phase>
    <configuration>
        <arguments>run-script build</arguments>
    </configuration>
</execution>
```

ARCHITECTURE LOGICIELLE

PACKAGING & DELIVERY : AOT

- Le build pour un environnement prod est obligatoirement réalisé en compilation AOT, *Ahead-of-time* (en opposition à JIT*, Just-In-Time)*
- AOT entraine un temps de compilation plus important que JIT mais permettra d'avoir un livrable plus léger et plus performant (plus de compilation dans le navigateur)
- En somme, en développement privilégier le JIT pour maintenir une rapidité de redéploiement ; en production privilégier AOT pour assurer des performances et un livrable plus léger
- Attention, en intégration continue, tester l'AOT-ready ! ça évitera les surprises avant de passer en production. Les pièges à éviter : Making Your Application AOT Ready

- **Protractor :** Angular-CLI embarque la bibliothèque Protractor pour le développement de tests *end-to-end* automatisés

- **Jasmine :** Angular-CLI embarque la bibliothèque Jasmine pour le développement de tests unitaires automatisés. Les commandes de génération de composants, de directives, de services et de filtres fournissent automatiquement un fichier de tests associés (suffixé de specs)

ARCHITECTURE LOGICIELLE

TSLINT

- Angular-CLI se base sur la libraire TS Lint pour l'analyse statique du code
- Permet de configurer les règles :
  - Formatage et indentation
  - Règles de développement associées au langage JavaScript
  - Règles de développement associées au langage TypeScript
  - Règles de développement associées à Angular

ARCHITECTURE LOGICIELLE

DEBUGGING

Routage et affichage de données

Formulaire, récupération et envoi de données

Affichage de graphiques avec une librairie

TP1 - TRY AN API

ROUTAGE ET AFFICHAGE DE DONNÉES

- Récupérer la branche tp1 : git checkout tp1

- L'application est composée d'un entête comportant un menu avec 2 items :
  - Les unités (page par défaut)
  - Les arènes

En dessous de cette entête une section est dédiée à l'affichage des unités ou des arènes

- Exercice 1 :
  - Créer un module associé à l'item Unités et afficher la liste des unités (cards dans l'API) ordonnés par nom.
  - Créer un module associé à l'item Arènes et afficher la liste des unités ordonnés par nom
  - Associer chacun des modules au menu relatif. Positionnez la route par défaut sur le menu Unités.

- Exercice 2 :
  - Pour chaque entrée (unité ou arène), créer un lien amenant vers des pages de détails de l'unité ou de l'arène.
  - Dans la page de détail, afficher l'ensemble des informations de l'entité et l'image associée

TP2 - CHAINES DE CONTROLES

FORMULAIRE ET ENVOI DE DONNÉES

- Récupérer la branche tp2 : git checkout tp2

- L'application est composée d'un entête et deux sections :
  - Un formulaire de création de commande
  - Une section affichant les 3 dernières commandes

- Exercice 1 :
  - Créer un formulaire comprenant les champs obligatoires suivants :
    - Nom (sans espace)
    - Nombre de couverts (entier positif)
    - Date et heure (supérieure à la date / heure actuelle)

  Renseigner clairement les erreurs pour aider la saisie en dessous du formulaire et ajouter un bouton pour enregistrer la commande.
  - Dans la seconde section, affichées les 3 informations d'une commande.

- Exercice 2 :
  - Pour aider à la saisie, pour toute nouvelle commande, il faut :
    - Ajuster le nombre de couverts au nombre de couverts majoritairement choisi dans les commandes existantes,
    - Ajuster la date et l'heure au premier jour de la semaine suivante, une heure et demi plus tard que l'heure actuelle

TP3 - DATAVIZ

AFFICHAGE DE GRAPHIQUES AVEC UNE LIBRAIRIE

- Récupérer la branche tp3 : git checkout tp3
- L'application est composée d'un entête et deux sections :
  - Une section de saisie de formules et de données
  - Une section d'affichage des graphiques
- Exercice 1 :
  - Ajouter 2 champs de saisie dans la première section correspondant aux variables a et b de la fonction ax + b.
  - Dans la seconde section, afficher la courbe ax +b dans un graphique de type LineChart
- Exercice 2 :
  - Ajouter 4 champs de saisie dans la première section correspondant aux coûts de votre projet : bug, analyse, report et dette logicielle
  - Dans la seconde section, afficher la répartition de ces 4 coûts dans un graphique de type PieChart

QUESTIONS / RÉPONSES

RESSOURCES

- Angular.io : référence de l'ensemble de la documentation Angular
- http://slides.com/brunobaia/everything-is-a-stream : présentation sur les streams, la différence entre Observable et Promise
- http://slides.com/brunobaia/redux-with-angular-ngrx#/ : présentation sur NgRx par Bruno Baia
- www.learnrxjs.io/operators/ : ensemble des opérateurs RxJs et documentations associées
- https://embed.plnkr.co/plunk/3e2txp : Plunker sur une recherche sur Observable avec l'API Youtube
- angular.io/api/forms/Validators : référentiel de l'ensemble des validateurs nativement intégrés dans Angular
- angular.io/api?type=pipe : référentiel des filtres nativement intégrés
- https://update.angular.io/ : service d'aide à la migration de version