ANGULAR 2

Passons maintenant à la version actuelle d'Angular qui utilise Typescript et non Javascript.

TYPESCRIPT

Quelques opérateurs :

- & : ET logique
- | : OU logique
- && : ET ALORS
- || : OU ALORS
- = : Affectation
- == : égalité avec projection de type (5 == 5 && 5 == "5")
- != : inégalité avec projection de type
- === : égalité dîte stricte (de valeur et de type)
- !== : inégalité dîte stricte (5 === 5 && 5 !== "5")
- Opérateur ternaire : condition?valeurSiVrai:valeurSiFaux
  5===x?"banco":"manqué"
- Destructuration :
  - [a, b] = [b, a]
  - [a, b, ...rest] = [0, 1, 2, 3, 4, 5, 6]
  - {nom:N, prénom:P} = {nom:"Bob", prénom:"Kelso", âge:56}

Quelques instructions de contrôles:

- if (CONDITION) {INSTRUCTIONS} else {INSTRUCTIONS}
- while (CONDITION) {INSTRUCTIONS}
- do {INSTRUCTIONS} while(CONDITION)
- witch (EXPRESSION) { case VALUE_1: INSTRUCTIONS; break; ... case VALUE_N:
  INSTRUCTIONS; break; default: INSTRUCTIONS
- for(INIT; CONDITION; INCR) {INSTRUCTIONS}
- for(let i in ITERABLE) {INSTRUCTIONS}
- for(let v of COLLECTION) {INSTRUCTIONS}
- try {INSTRUCTIONS} catch(ERROR) {INSTRUCTIONS}

Types de bases : string (voir sur MDN)

- Comparaison avec === et !==
- Copie : slice([début, [fin]])
  - let copieTexte = monTexte1.slice();
  - let troisPremiers = monTexte1.slice(0, 3)
  - let troisDerniers = monTexte1.slice(-3)
  - let deQuatreàHuit = monTexte1.slice(4,8);
- Tronçonnage : split
  - let TableauChar = monTexte1.split();
  - let TableauMots = monTexte.split(" ");
  - let TableauMots2 = monTexte1.split(/\s*/)

Interface et classes ressemblant à un language Java ou C#

INSTALLATION

Npm est le gestionnaire de paquets de Javascript : https://www.npmjs.com/
Télécharger et installer le grâce à NodeJs : https://nodejs.org/en/download/

Pour ce cours, je travaillerais avec l'IDE Visual Code, vous avez le choix pour votre IDE.

PROJET

- Créer un nouveau dossier "angular"
- Aller dans ce dossier (cd/dir) et exécuter :
    - npm init
    - npm install node
        - Le développement angular se fait avec NodeJs, la mise en production peut se faire avec n'importe quel serveur en back.
    - npm install @types/node
    - npm install typescript
    - npm install --save @angular/core @angular/compiler @angular/common @angular/platform-browser @angular/platform-browser-dynamic rxjs reflect-metadata zone.js
    - npm install @angular/cli
        - On installe enfin angular

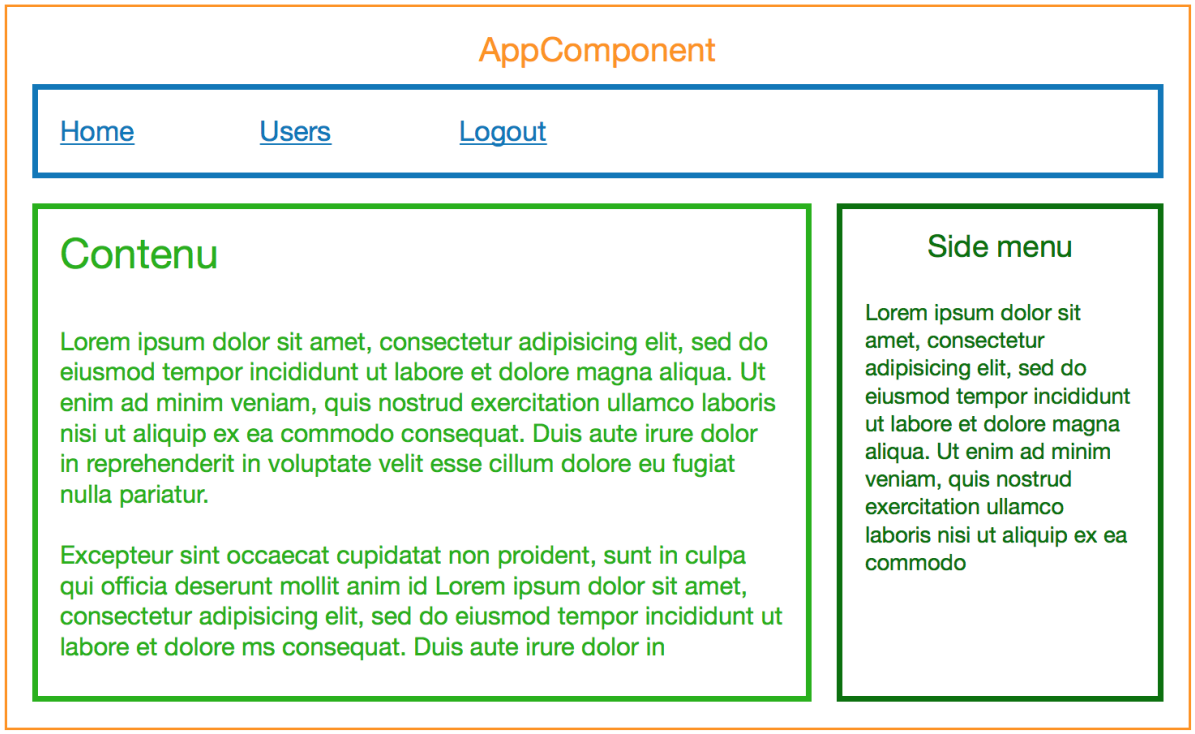
Remarque : pour laisser tourner le compilateur ts en fond et qu'il compile à chaque sauvegarde : tsc --watch

- Créer un répertoire "projects"
- Dans "projects", ng new lilltepony pour créer votre application littlepony
- Se mettre dans le répertoire projet.
- Installation de bootstrap : npm install bootstrap@3.3.7 --save (ou @last ou @numero-version)
- Pour intégrer les styles bootstrap, ajouter dans angular.json :
    - "styles": [
    - "../node_modules/bootstrap/dist/css/bootstrap.css",
    - "styles.scss"

Attention : les libraires s'ajoutent par installation par par lien, les styles s'ajoutent par la configuration, pas par les liens.

- Tester votre application en la laçant : ng serve
- Aller sur localhost:4200 avec un navigateur
- Chercher dans quel fichier se trouve la page html affichée.

APP COMPONENT



- Dans votre AppComponent (le point d'entrée métier en angular), initialiser un attribut hello à "coucou" et afficher le et rien que lui dans le html : {{hello}}

CLASSE

- Créer une classe Pony (un nom, une couleur et un âge) : ng g class Pony

Attention : Angular utilise Typescript et non Javascript

COMPONENT

- Ajouter un component Ponies : ng g component Ponies
- Dans Ponies, déclarer un attribut tableau de Pony et initialiser le dans le constructeur.
- Dans la vue de Ponies, afficher votre liste de Pony grâce à *ngFor="let e of tab".

La directive NgFor permet de faire une boucle sur un tableau du component (l'index est facultatif) :

```
<div *ngFor="let item of items; let i=index">{{i + 1}} - {{item.name}}</div>
```

L'option trackBy permet de boucler sur de grandes listes pour éviter des latences si modifications des éléments :

```
<div *ngFor="let item of items; trackBy: trackByItems"> ({{item.id}}) {{item.name}} </div>
```

MOCK

- Créer un fichier mock-ponies.ts qui représente un tableau de pony en constante.
- Intégrer ce mock dans Ponies.

EXERCICE
Un Race est une course avec des poney. Elle a lieu à une date et un lieu donnés. Ajouter un classe Race, un mock et un composant pour afficher toutes les courses.

MASTER/DETAILS

- Générer un composant pony-details.
- Dans son template, ajouter une <div *ngIf="pony"> qui permet d'afficher les détails d'un Pony.
- Intégrer ce composant dans Ponies grâce à <app-pony-detail [pony]="..."></app-pony-detail>

Directives conditionnelles:

- NgIf—conditionally creates or destroys subviews from the template.
- NgSwitch—a set of directives that switch among alternative views.

EXERCICE
Faire de même (master/détails) avec les courses.

MISE EN FORME

Certaines données sont correctes mais ont un format non user friendly. Pour les modifier, on utilise les Pipe d'angular.

- Pour l'affichage de la date au format français, modifier son affichage : {{ raceDate | date : "dd/MM/yyyy" }}
- Lancer votre application et regarder la modification.

Les pipes d'Angular sont les suivants :

- DatePipe: Formats a date value according to locale rules.
- UpperCasePipe: Transforms text to all upper case.
- LowerCasePipe: Transforms text to all lower case.
- CurrencyPipe: Transforms a number to a currency string, formatted according to locale rules.
- DecimalPipe: Transforms a number into a string with a decimal point, formatted according to locale rules.

- PercentPipe: Transforms a number to a percentage string, formatted according to locale rules.

- Création d'un pipe custom : on souhaite afficher les noms des courses tout en majuscules, chaque lettre séparée par un point => custom pipe
- ng g pipe NameRacePipe
- Dans la méthode transform(race: Race, args?: any) retourner le nom formaté correctement

Pour modifier le css/html de vos éléments vous pouvez utiliser :

- NgClass—adds and removes a set of CSS classes.
- NgStyle—adds and removes a set of HTML styles.
- NgModel—adds two-way data binding to an HTML form element.

EXERCICE
Ajouter des pipes pour que les données soient affichées proprement. Faire en sorte que les poney soit affichés dans la couleur qui leur est attribuée.

ROUTES

- Créer un component Menu qui représente un menu : Accueil, Poneys, Courses.
- Intégrer votre Menu à votre AppComponent.
- Dans app.module, ajouter :
    - import { RouterModule, Routes } from '@angular/router';
    - const appRoutes: Routes = [
    - { path: 'ponies', component: PoniesComponent },
    -];
- Et dans son html
    - <router-outlet></router-outlet>
    - <nav>
        <a routerLink="/ponies">Afficher les poneys
      </nav>

EXERCICE
Faire de même avec les courses.

FORMULAIRES

- Ajouter un component Pony
- Importer FormsModule dans votre app.module
- Ajouter dans votre component Pony, un model qui représente un poney par défaut (les attributs de ce pony seront ceux affichés dans le formulaire) et une méthode

onSubmit qui représente la validation du formulaire par l'utilisateur (pour le moment, on affiche en console le pony créé par l'utilisateur).

```
<form (ngSubmit)="onSubmit()">
    <div class="form-group">
      <label for="name">Name</label>
      <input type="text" class="form-control" id="name" required
      [(ngModel)]="model.name" name="name">
    </div>
    <div class="form-group">
      <label for="weight">Poids</label>
      <input type="text" class="form-control" id="weight" required
      [(ngModel)]="model.weight" name="weight">
    </div>
    <div class="form-group">
      <label for="age">Age</label>
      <input type="text" class="form-control" id="age" required
      [(ngModel)]="model.age" name="age">
    </div>
```

Si vous voulez gérer les inputs pour l'affichage, vous pouvez utiliser :

State	Class if true	Class if false
The control has been visited.	`ng-touched`	`ng-untouched`
The control's value has changed.	`ng-dirty`	`ng-pristine`
The control's value is valid.	`ng-valid`	`ng-invalid`

Exemple :

```
<div [hidden]="name.valid || name.pristine" class="alert alert-danger">
<button type="button" class="btn btn-default" (click)="newHero(); heroForm.reset()">
New Hero
</button>
</div>
```

Vous avez aussi accès à tous les attributs d'événement avec Angular comme (click) par exemple. Attention ne pas utiliser ceux du JS natif et bien respecter la syntaxe Angular.

FORMULAIRE REACTIVE

On peut utiliser les FormBuilder pour créer un formulaire:
https://angular.io/api/forms/FormBuilder
Petite démonstration.

EXERCICE
Ajouter un formulaire avec la méthode que vous voulez pour rentrer des courses.

SERVICES

- Générer un service pony : ng g service pony
- importer les observables
    - import { Observable } from 'rxjs';
- Créer une propriété tableau de pony
- Ajouter une méthode pour ajouter un pony à la propriété de type tableau
- Dans le getter de cette propriété retourner un Observable du tableau:
    - Observable.of(this.ponies)
- Dans votre gestion du formulaire
    - Injecter votre service
        - constructor(private ponyService : PonyService)
- Faire en sorte que le pony rentré par l'utilisateur soit rentré dans la liste du service
- Dans app.module :
    - Enregistrer votre service : providers: [PonyService],
- Dans app.component
    - Injecter votre service pour afficher dynamiquement la liste de pony et non plus en dur

EXERCICE
Ajouter proprement un service lié au mock pour les courses.

AVEC UN API REST

- Regarder la pseudo API sur ludivinecrepin.fr/api
    - pony-add.php, pony-get-id.php,pony-get.php,pony-update.php, idem pour race
- Modifier vos services pour qu'ils appellent cette API au lieu des mocks :
    - Ajouter en attribut l'url de l'API :
        - url: String = 'http://localhost:8080/MyLittlePonies-0.0.1-SNAPSHOT/pony/';
    - Ajouter en attribut les options de la requête HTTP :
        - httpOptions = {
            headers: new HttpHeaders({

```
            'Content-type': 'application/json'
            })
        };
```

- Modifier le constructeur :
  - constructor(private http: HttpClient) {}
- Changer les retours :
  - return this.http.get<Array<Pony>>(this.url + '/ponies', this.httpOptions);
  - return this.http.get<Pony>(this.url + '' + id, this.httpOptions);
- Changer les méthodes POST/UPDATE, souscrivez au service :
  - this.http.post(this.url + '/add', p, this.httpOptions).subscribe();

EXERCICE
Appeler maintenant l'API pour les courses.

GARD

EXERCICE
Ajouter un formulaire dans le menu qui permet à un utilisateur d'entrer son login (aucune vérification). Une fois le login entré, le formulaire devient un bouton "Déconnexion". Le login doit être stocké en session pour preuve de connexion et le bouton doit effacé le login en session au clic.

Faire en sorte que les pages d'ajout ne puissent être accessibles que si un utilisateur est connecté :

- Générer votre guard : ng generate guard connexion-guard
- Choisir CanActivate
- Ajotuer dans les routes qui demandent une connexion canActivate: [ConnexionGuard]
- Tester que la guard fonctionne correctement

TEST UNITAIRE

Les tests s'exécutent avec Karma et la commande ng test. Le pattern des fichiers de test est défini dans le fichier karma.conf.js, par défaut il s'agit de "*.spec.ts".
Chaque test est un it :
it('nom du test', () => { code du test });

EXERCICE
Créer un test unitaire sur le pipe créé.

LIBRAIRIES GRAPHIQUES

Il existe plusieurs librairies qui fournissent des éléments graphiques pour Angular (i-e-JQueryUI pour Angular) comme par exemple :

- Material : https://material.angular.io/ (attention au problème de compatibilité)

- PrineNg : https://material.angular.io/

EXERCICE
Choisir un élément graphique d'une librairie graphique et utiliser le dans votre application.

404

EXERCICE
Une page 404 est un composant qui a comme pattern " ** ". Créer votre propre page 404.

MISE EN PRODUCTION

Sur le serveur de production, vous n'avez qu'à déposer le build du projet : ng build
Documentation : https://angular.io/cli/build

Selon le temps : test e2e