Maxime Norval
Expert Angular

Projet Fil Rouge

Pendant cette formation, vous allez créer
une superbe boutique d'animaux de compagnie en ligne !
On pourra y consulter les animaux et en créer de nouveaux.

Sommaire

Partie 1

1- Utilisez Angular CLI pour créer une application que vous pourrez nommer « pet-shop » (ou tout titre qui vous semblera approprié pour une telle application) :
   - N'ajoutez pas de routing pour le moment,
   - Choisissez le format SCSS,
   - Videz le contenu du template du composant « app».

2- Sans utiliser le CLI, créez un répertoire « pet » (à la racine de « app ») avec le composant « pet » : le modèle complet (classe TypeScript « PetComponent » avec son décorateur), son template (HTML) et sa feuille de style (SCSS). Puis :
   - Créez un paragraphe contenant « Hello World! » dans le template du composant « pet »,
   - Faites les manipulations nécessaires pour que ce texte puisse s'afficher à l'écran, puis lancez l'application en mode développement (« ng serve »), ouvrez votre navigateur et naviguez vers l'URL http://localhost:4200,
   - Voyez-vous « Hello World » ?
     - Si oui → Félicitations !
     - Si non → Qu'avez-vous oublié ?

[CORRECTION]

3- Avec le CLI :
   - À l'intérieur du répertoire « pet », créez 2 composants « pet-list » et « pet-detail ». Ce seront les enfants de « pet » : affichez-les, dans cet ordre-là, dans le template de « pet » (vous pouvez supprimer le paragraphe « Hello World »),
   - À l'intérieur du répertoire « pet-list », créez un composant « pet-list-item » (mais ne l'instanciez dans aucun template pour l'instant),
   - À la racine du composant « app », créez un composant « navbar ». Affichez-le au-dessus du composant « pet ».

[CORRECTION]

4- Un peu de mise en forme (faite directement en groupe) :
   - Dans « styles.scss » :
     - Donnez une hauteur de 100% aux tags « html » et « body »,
     - Donnez une marge de 0 et la police « Helvetica Neue » au tag « body ».
   - Dans « app.component.html » :
     - Imbriquez « app-navbar » et « app-pet » chacune dans une div avec les classes respectives « navbar » et « content »,
     - Imbriquez le tout dans une div avec la classe « app ».
   - Dans « app.component.scss » :
     - La classe « app » a une hauteur de 100%,
     - La classe « navbar » a une hauteur de 60px,
     - La classe « content » a une marge de 20px et une hauteur de 100% - 60px (hauteur de la navbar) – 40px (marges du haut et du bas).

- Dans « navbar.component.html » : supprimez le contenu et les remplacer avec une div de classe « navbar » (de manière générale, on utilisera toujours la bonne pratique d'imbriquer tout le template d'un composant dans une div qui a le même nom que ce composant).
- Dans « navbar.component.scss » : donnez une hauteur de 100% à la classe « navbar » et donnez-lui une couleur d'arrière-plan (par exemple, « blueviolet »).
- Dans « pet.component.html » : imbriquez tout dans une div de classe « pet ».
- Dans « pet.component.scss » :
  - La classe pet a une hauteur de 100%, et c'est une Flexbox,
  - Le tag « app-pet-list » a une propriété « flex » de 1, et le tag « app-pet-detail » a une propriété « flex » de 2 (il prend 2 fois plus de place que le précédent,
  - Ces 2 tags ont un padding de 10px.

5- Nous allons maintenant nous occuper de nos petits animaux !
   - Manuellement (sans le CLI) : Créez un répertoire « model » dans le répertoire « pet ». Puis, à l'intérieur, créez un fichier « pet.ts ». Depuis ce fichier, exportez une interface IPet qui impose les propriétés suivantes :
     - Un « id » qui est un nombre,
     - Un « name » qui est un string,
     - Une « species » (espèce) qui est un string mais ne peut avoir comme valeur que « chien », « chat », « poisson » ou « lapin »,
       [astuce : utilisez un « type », que vous créerez dans le même fichier et que vous exporterez aussi],
     - Un « price » (en euros), qui est un nombre,
     - « isAvailable », qui est un booléen.
   - Importez cette interface pour que le fichier TypeScript du composant « pet » la connaisse. Créez une propriété pets qui est un tableau de IPet déclaré comme un tableau vide. Le composant « pet » implémentera l'interface « OnInit ». Dans la méthode « ngOnInit », faites ensuite appel à une méthode privée « createPets » qui ne renvoie rien mais peuple le tableau avec 4 animaux avec des ID de 1 à 4 (vous pouvez utiliser une bouche). Par exemple :
     - 1 : milou, un chien disponible qui coûte 500 euros,
     - 2 : garfield, un chat non disponible qui coûte 400 euros,
     - 3 : nemo, un poisson disponible qui coûte 10 euros,
     - 4 : bugs bunny, un lapin non disponible qui coûte 50 euros.
   - Le composant « pet-list » a également une propriété « pets » qui est un tableau de IPet déclarée comme un tableau vide. Passez-lui donc les petits animaux, que vous afficherez en console depuis son ngOnInit()…

[CORRECTION]

6- Bien ! Maintenant, débarrassons-nous de ce disgracieux « console.log() » et affichons plutôt notre liste d'animaux pour de vrai : Pour chaque animal, le composant « pet-list » devra afficher un enfant « pet-list-item », qui lui-même affichera le nom de l'animal.

[NB : N'assignez pas de valeur à la propriété « pet » du composant « pet-list-item ». Vous aurez probablement une erreur de compilation mais c'est à cause du mode strict de TypeScript

qui est activé par défaut. Ajoutez simplement un point d'exclamation à la fin du nom de la propriété pour indiquer à TypeScript qu'il a une valeur (qui lui est donnée par son parent).]

[CORRECTION]

7- Dans le composant « pet-list», créez maintenant une méthode « onClickPet » qui prend un « petId » en paramètre et affiche l'id de l'animal cliqué en console. Une fois que ça fonctionne, retirez le « console.log ».

[CORRECTION]

8- Facile ? OK, passons aux choses sérieuses !

Notre objectif final, pour boucler cette première partie, sera d'afficher les informations de l'animal sélectionné dans le composant « pet-detail ». Ce dernier est l'enfant de « pet », qui est aussi le parent de « pet-list » dans lequel on sélectionne l'animal. Il faut donc :

- d'abord, que « pet-list » informe son parent de l'animal sélectionné,
- puis, que ce parent transmette cet animal à son autre enfant, « pet-detail », qui l'affichera enfin.

Commençons par la première étape :

- Déclarez une propriété selectedPet du composant « pet ». Mais ne lui assignez aucune valeur (pas d'animal sélectionné au départ). Cette fois-ci, cette propriété peut être nulle ou undefined (n'utilisez pas de point d'exclamation (!)). Elle a donc un type alternatif : « IPet » ou « undefined » ou « null ».
- Le composant « pet-list » doit ***émettre un évènement*** quand on clique sur un « pet-list-item ». Cet évènement doit ***informer de l'ID*** du « pet » cliqué. Il doit être ***disponible en sortie*** du composant, pour pouvoir être écouté par son parent. Vous suivez ? Créez la propriété « petSelected » du composant « pet-list ». Typez-la, assignez-la et décorez-là comme il faut, puis, à partir de « onClickPet » émettez l'ID du « pet » sélectionné grâce à cette propriété.
- Enfin, dans le composant « pet », créez une méthode « onSelectPet » qui prend un « petId » en paramètre et définit le « selectedPet » à partir de celui-ci *[astuce : utilisez la méthode JavaScript « find » qui s'applique sur les tableaux]*. Depuis le template de ce composant, écoutez ensuite l'évènement que vous venez de créer dans l'enfant, et déclenchez la méthode « onSelectPet » en lui passant l'id du « pet » sélectionné grâce au mot-clé « $event ». Dans cette dernière méthode, affichez le « selectedPet » en console pour vérifier que tout marche. Si c'est le cas, chapeau ! Le plus dur est fait (enlevez-moi quand-même ce « console.log ») …

[Déjà envie d'arrêter Angular ? Pas de panique, il existe une méthode beaucoup plus simple pour communiquer entre les composants qui n'ont pas de lien de parenté directe. Mais nous y viendrons bientôt …]

[CORRECTION]

9- Il ne reste plus qu'à passer le « selectedPet » de « pet » à « pet-detail ». À cette étape, vous devriez trouver ça facile. Créez une propriété « pet » dans « pet-detail » et faites le nécessaire …

[CORRECTION]

10- Dans le template de pet-detail, créez une « div » qui ne s'affiche que si la propriété « pet » est définie et non nulle. Dedans, affichez :

- Le nom de l'animal dans un titre « h1 »,
- Son espèce dans un sous-titre « h2 »,
- Son prix (en euros) et sa disponibilité (« Disponible » ou « Vendu » dans des paragraphes (« p »).

Vérifiez que tous d'affiche bien quand vous naviguez entre les animaux.

[CORRECTION]

11- Dernières retouches de mise en forme (en groupe)

- « pet-list »
  - HTML : Div « pet-list » qui englobe tout
  - SCSS :
    - Height : 100%
    - Overflow : auto (scrollbar automatiquement ajoutée si son contenu dépasse sa hauteur)
- « pet-list-item »
  - HTML : Donnez la classe « pet-list-item » à la div qui englobe tout,
  - SCSS :
    - border: 1px solid blueviolet;
    - border-radius: 10px;
    - margin: 5px 0;
    - padding: 20px;
    - text-align: center;
    - cursor: pointer;
    - &:hover
      - background-color: blueviolet;
      - color: white;

- « pet-detail »
  - HTML :
    - Div « pet-detail » qui englobe tout
    - Div « title » qui englobe les 2 titres (« h1 » et « h2 »)
    - Classe « price » sur le paragaphe correspondant
    - Sur l'autre paragraphe : [class]="pet.isAvailable ? 'available' : 'sold'"
    - Le *ngIf devient : *ngIf="!!pet; else noPetSelected". Rajoutez class="pet-info" à cette div
    - On ajoute un ng-template avant la dernière balise « div » fermante : <ng-template #noPetSelected> Merci de sélectionner un animal. </ng-template>
  - SCSS :

- .pet-detail
    - height: 100%;
    - text-align: center;
    - border: 1px solid grey;
    - border-radius: 50px;
    - .pet-info
        - height: 100%;
        - display: flex;
        - flex-direction: column;
        - justify-content: space-around;
        - .price { font-size: 200%; }
        - .available { color: green; }
        - .sold { color: red; } }
    - .no-pet-selected
        - height: 100%;
        - display: flex;
        - justify-content: center;
        - align-items: center;
        - font-size: 200%;

Partie 2

12- Service « pet »
- Sans utiliser le CLI :
  - Créez un service « pet » dans le répertoire du même nom.
  - Rendez-le singleton sans toucher au fichier « app.module.ts ».
- Utilisez ce service pour la communication entre les composants, qui ne se fait plus par les entrées et sorties (les décorateurs « Input » et « Output » doivent tous être supprimés, à l'exception d'un seul « Input » dans « pet-list-item »). De plus, toute la logique métier relatif aux « pets » doit être entièrement déléguée au service.

[CORRECTION]

13- Désélection d'un « pet »
- Dans le modèle de « pet-list-item », arrangez-vous pour savoir si le « pet » est celui sélectionné.
- Dans le CSS de « pet-list-item », donnez le même style à un item quand il a la classe « selected » que quand la souris passe dessus.
- Dans le HTML de ce composant, donnez la classe « selected » au « pet » quand ce dernier est sélectionné.
  - Indice : « [ngClass]={nom-de-la-classe : condition} »
- Arrangez-vous pour que, lorsqu'on clique sur un « pet » sélectionné, il soit désélectionné.

[CORRECTION]

14- Dans la liste et dans le détail des « pets », affichez les noms d'animaux avec la première lettre de chaque mot en majuscule, sans toucher au modèle.

[CORRECTION]

15- Affichage d'une image avec le property binding
- Dans l'interface « IPet », ajoutez une propriété « imageUrl » qui est un string optionnel (utilisez un « ? » à la fin du nom de la propriété).
- Dans le service, attribuez une URL à chacun des animaux

Pour Milou, Garfield, Nemo et Bugs Bunny, vous pouvez prendre :
['https://cdn001.tintin.com/public/tintin/img/static/milou/milou_v3.png',
'https://upload.wikimedia.org/wikipedia/en/thumb/b/bc/Garfield_the_Cat.svg/1200px-Garfield_the_Cat.svg.png',
'https://easydrawingguides.com/wp-content/uploads/2017/05/How-to-Draw-Nemo-20.png',
'https://img.src.ca/2015/07/27/1250x703/150727_co2q2_aetd_bugs-bunny_sn1250.jpg']

- Dans le template de « pet-detail », au-dessus du paragraphe indiquant le prix, placez une balise image auto-fermante (« <img /> »). Servez-vous du property-binding pour lier sa source (« src ») à l'URL du « pet ».
- Mise en forme de l'image dans le CSS :
  - Le tag « img » enfant d'un élément de classe « pet-info » doit avoir :

- Une hauteur et une largeur de 50% maximum
- Une marge auto

[CORRECTION]

Partie 3

16- Création d'un composant « add-pet » et de la navigation provisoire vers celui-ci :
- Générer un composant « add-pet » sous le répertoire « pet ».
- Dans le pet-service, créez une propriété booléenne « isCreatingPet », initialisée à « false ».
- Créez un bouton « Create/Consult » dans la navbar qui inverse la valeur de cette propriété du service grâce à une méthode « togglePetCreation() » dans ce dernier et à l'injection de dépendance.
- Dans le template du composant « pet », imbriquez les 2 enfants dans une div à laquelle vous donnerez une classe « pet-read-container » qui a une hauteur de 100%. C'est elle qui est désormais une flexbox (et non plus la div de classe « pet »).
- Cette div n'est affichée que si « isCreatingPet » du service est false. Sinon, affichez le composant « add-pet »

17- Création d'un formulaire réactif
- Rendez les formulaires réactifs disponibles dans votre application.
- Dans « add-pet », créez un formulaire réactif qui comprend :
  - Le nom
    - Un input de type « text »
    - Obligatoire, minimum 3 caractères, et « chien » par défaut
  - L'espèce
    - Un select avec les 3 options correspondantes
    - Obligatoire
  - Le prix
    - Un input de type « number »
    - Obligatoire et au moins égal à zéro
  - La disponibilité
    - Un input de type « checkbox »
    - Obligatoire et « true » par défaut
  - L'URL de l'image
    - Un input de type « text »
    - Facultatif
- À la soumission du formulaire, affichez un objet avec ses valeurs en console, et affichez sa validité.

[CORRECTION]

18- Création d'un « pet » grâce au formulaire
- Dans le service « pet », créez une méthode « createPet() » :
  - Elle prend en paramètres : nom, espèce, prix, disponibilité, et URL.
  - Elle génère un id qui incrémente le plus grand de la liste déjà existante *[indice : Math.max() ou reduce()]*.
  - Elle crée un pet et l'ajoute à sa liste.
- Depuis le composant, appelez cette méthode à la soumission du formulaire, seulement si ce dernier est valide.

- Après vous être assuré de cette première « sécurité », ajoutez-en une en désactivant le bouton si le formulaire est invalide.
- Enfin, arrangez-vous pour réaffichez la liste des pets après la création.

[CORRECTION]

19- Mise en forme (en groupe)

Partie 4

20- Passage aux API

- Supprimez la méthode createPets, ainsi que son contenu et ses références, du service. Vous allez maintenant travailler avec de vraies données récupérées depuis un serveur.
- Rendez le service « HttpClient » disponible dans votre application, puis injectez-le dans le service.
- Créez une méthode getPet() qui renvoie « void » au départ.
  - Elle effectuera un « get » sur une URL préalablement attribuée en propriété privée « _petsUrl » :

https://formation-6e588-default-rtdb.europe-west1.firebasedatabase.app/pets.json

  - Dans cette méthode, souscrivez au « get » et affichez le résultat de ce qu'il émet en console.
  - Appelez « getPets() » dans le constructeur du service.
  - En laissant la souscription, pipez l'Observable renvoyé par « get() » pour appliquer dessus l'opérateur « map() » de RxJS. Votre but est d'afficher en console un tableau d'objets formatés comme des « IPet » à une exception près : leur id sera un string, qui correspond aux clés de l'objet initialement affiché en console.

[NB : Utilisez « for…in » pour itérer dans les propriétés d'un objet.]

[NB2 : Si TypeScript vous pose problème, utilisez le mot-clé « any » pour l'instant.]

  - Changez maintenant l'interface IPet en conséquence : l'id est de type string.
  - Dans la callback du « subscribe() », assigner le résultat à la propriété « pets » du service.
  - Commentez la méthode addPets() et ses références.
  - Réparez l'application en typant l'id correctement à chaque endroit où c'est nécessaire.
  - L'application doit maintenant fonctionner correctement, sauf pour l'ajout de « pet ».

[CORRECTION]

21- Post à l'API

- Décommentez la méthode « addPets() » du service
- Ne changez pas sa signature mais remplacez son contenu : elle doit maintenant faire un post à l'API.
  - La création d'id n'est maintenant plus nécessaire : il est récupéré grâce à l'API et à notre méthode « getPets().
  - Souscrivez au retour du « post() » pour rappeler « getPets() » une fois le résultat reçu.
- N'oubliez pas de rappeler cette méthode d'ajout depuis le composant « add-pet ».
- Naviguez sur votre application et effectuez 1 ou 2 posts depuis le formulaire (pas plus s'il vous plait, vous travaillez tous sur la même API et les requêtes ne sont pas illimitées avec ma version gratuite de Firebase :D)

[CORRECTION]

Partie 5

22- Mise en place du routage

- Créez un composant « home » (à la racine) qui affiche « Hello World! »
- Créez manuellement un fichier « app-routing.module.ts » puis, dedans :
  - Créez un module « AppRoutingModule »
  - Créez les routes suivantes :
    - « home », qui correspond au composant « home »
    - « pet », qui correspond au composant « pet »
  - Faites en sorte qu'une route vide ou inconnue renvoie vers « home »
  - Enregistrez ces routes grâce à un router que vous exporterez vers le AppModule
  - Mettez un lien fonctionnel vers chacune de ces routes dans le template de la navbar.
  - Mettez un « router-outlet » au bon endroit.
  - Vérifiez que le routing fonctionne.

[CORRECTION]

23- Sous-routes

- Créez un composant « pet-index » dans le répertoire « pet »
  - Copiez le contenu du template de « pet » dedans, en remplaçant la classe « pet » par « pet-index », et en supprimant toute référence à « addPet() » et « isCreatingPet » (c'est désormais inutile).
  - Faites de même pour le CSS.
  - Dans la div de classe « pet » du composant « pet », remplacez tout le contenu par un router-outlet .
  - Dans le CSS de « pet », supprimez tout le style de la classe « pet-read-container » et de ses enfants ;
- Enregistrez deux routes enfants de « pet » :
  - « index », qui mène à « pet-list »
  - « add », qui mène à « add-pet »
- Par défaut, les routes « pet » et « pet/[url-inconnue] » mènent à « index »
- Déplacez les répertoires « pet-list » et « pet-detail » dans « pet-index » comme ce sont désormais ses enfants. Mettez à jour les imports partout où c'est nécessaire..

Indice : this.router.navigate(['..', 'add'], { relativeTo: this.route });

- Ajoutez un bouton « Nouveau » en haut de la « pet-list », qui redirige vers « add-pet » depuis le modèle et un bouton « Retour à la liste » en haut de « add-pet » qui redirige vers « pet-index » depuis le modèle
- Supprimez complètement l'ancienne logique de navigation liée au booléen « isCreatingPet » du service « pet ». Mettez en place une vraie redirection depuis « add-pet » à la validation du formulaire.

[CORRECTION]

24- Sous-sous-route et son paramètre

- Débarrassez-vous de la propriété « selectedPet » du service et de toute la logique liée.

- Enregistrez une route enfant de « pet/index » sous forme d'un paramètre « id »
- Redirigez vers cette route quand on clique sur un « pet-list-item » (depuis le template), en remplaçant le paramètre par l'id du « pet » cliqué.
- Mettez le « router-outlet » au bon endroit pour afficher le pet-detail avec la route adéquate.
- Créer une méthode « petWithId() » dans le service qui sélectionne le « pet » souhaité dans sa liste.
- Depuis le « pet-detail », récupérez le paramètre « id » de la route et servez-vous-en pour appeler le service et récupérer le « pet » à afficher.
- Enfin, dans « pet-list-item », utilisez la directive « routerLinkActive » pour donner la classe « selected » à un item si sa route est active *[routerLinkActive=«nom-de-la-classe»]*.

[CORRECTION]

25- Guard d'authentification

- Créer un service « auth » avec une propriété booléenne « isAuth » initialisée à false et une méthode « toggleAuth() » qui connecte et déconnecte une fois sur deux.
- Injectez ce service dans la navbar. Créez-y un bouton « Connexion/Déconnexion », vert quand on est connecté, rouge sinon. Ce bouton déclenche la méthode « toggleAuth() » du service.
- Réservez l'accès à la route « pet » au visiteur authentifié.

[CORRECTION]

26- Mise en forme (en groupe)