

TD Front-End

Introduction

Comme dans la vraie vie, nous allons avoir affaire à un client (appelons le Fabien) qui sait à peu près ce qu'il veut. L'avantage pour nous c'est que cela va nous permettre de pouvoir faire de nombreux changements et ainsi apprendre les bases pour survivre dans Angular 😬

Il vous a transmis un répertoire Github contenant le code de son précédent développeur qui est malheureusement tombé malade 😞

Oui, le TD fait 8 pages mais ne vous inquiétez pas tout va bien se passer 🚀

Starter projet

La première étape est d'installer le starter. Toutes les instructions se trouvent dans le fichier README.md de ce répertoire :

<https://github.com/NablaT/starter-quizz-2022>

Exercice 0 : Prenez vos marques !

Petit exercice d'introduction des familles pour vous permettre de faire copain-copain avec le code existant. Vous avez de la chance, le développeur précédent semble avoir bien organisé son code 😊

1) Allez dans le fichier **src/mocks/quiz-list.mock.ts** et trouvez la constante **QUIZZ_LIST**. Cette dernière contient une liste de Quiz (ici elle a 2 quizz). Modifiez le nom d'un des quizz et observez maintenant les changements dans votre navigateur. N'hésitez pas à répéter cette action si besoin.


2) Comprenons ce qui vient de se passer :

- un mock est un fichier qui contient des (fausses) données.
- les objets sont typés avec les interfaces déclarées dans **src/models/quiz.model.ts**.
- le mock des quiz est importé dans le service **src/services/quiz.service.ts** et sert ensuite d'initialisation au flux d'un observable (BehaviorSubject).
- le composant QuizList (**src/app/quizzes/quiz-list/quiz-list.component.ts**) s'abonne au flux de l'observable et stocke les données transmises par le flux dans une variable **quizList**
- dans l'html associé au composant QuizList (**src/app/quizzes/quiz-list/quiz-list.component.html**), nous trouvons un ngFor qui permet d'itérer sur la liste des quizz (la fameuse variable **quizList**) et ainsi d'afficher chaque quizz avec le composant **app-quiz**
- Le composant app-quiz (**src/app/quizzes/quiz/quiz.component.ts**) affiche alors le nom du quiz qui lui est donné en paramètre.

- 3) Modifiez du HTML dans le fichier **src/app/quizzes/quiz/quiz.component.html** : mettez un "Nom :." devant le **{{quiz.name}}** et constatez le changement
- 4) Modifiez du CSS dans le fichier **src/app/quizzes/quiz/quiz.component.scss** en ajoutant "background: red;" dans le block du "h2"
- 5) Modifiez maintenant l'html pour ajouter le thème du quizz. Vous pouvez voir que l'objet Quiz (**src/models/quiz.model.ts**) contient un attribut **theme**. Revenez à l'html du composant Quiz (**src/app/quizzes/quiz/quiz.component.html**) et ajouter le thème grâce au property binding d'Angular (syntaxe avec double accolade **{{ }}**): ** Theme: {{quiz.theme}} **. Constatez le changement.
- 6) Vous pouvez voir dans votre application qu'un de vos quiz n'a pas de thème. Si vous retournez dans le model du Quiz (**src/models/quiz.model.ts**), vous noterez que l'attribut **theme** a un point d'interrogation lors de sa définition. Cela signifie que l'attribut est optionnel, il peut être non défini. Vous pouvez d'ailleurs le constater dans le fichier (**src/mocks/quiz-list.mock.ts**), l'objet **QUIZ_LIST** contient un thème uniquement pour le premier Quiz et le compilateur de Typescript ne s'en plaint pas. Enlevez, le point d'interrogation dans l'objet Quiz (**src/models/quiz.model.ts**) et vous allez voir que votre application ne compile plus!

```
ERROR in src/mocks/quiz-list.mock.ts(44,5): error TS2741: Property '
theme' is missing in type '{ name: string; questions: Question[]; }' but
required in type 'Quiz'.
```

Retournez dans le fichier de mock (**src/mocks/quiz-list.mock.ts**) et ajoutez un thème pour le quiz "Les Sports". Vous verrez alors l'erreur disparaître de votre console et le thème apparaître sur votre application.

 **Tip:** Pensez à ouvrir votre developper console sur votre browser (F12 sur chrome) pour voir vos erreurs de compilation.

Vous devriez maintenant un peu mieux comprendre comment le tout s'articule et comment les concepts vus en cours font pour fonctionner ensemble. En cas de besoin n'hésitez pas à refaire ce bref tutoriel pour bien assimiler toutes les notions présentées ici (mock, service, observable, composant (typescript, html, css), model).

Exercice 1 : Un quizz sans question ..?

Notre cher Fabien souhaiterait que l'on puisse afficher le nombre de questions qu'il y a pour chaque quizz afin d'estimer le temps qu'il faut pour le faire.

Étapes :

- Dans **src/app/quizzes/quiz/quiz.component.html**, ajoutez une balise HTML (celle de votre choix) et affichez le nombre de questions grâce à la longueur du tableau de questions: **quiz.questions.length**. Utilisez la même syntaxe que celle utilisée pour afficher la valeur du thème (property binding **{{ }}**).

Fabien voudrait maintenant avoir des quizz avec des questions.

Étapes:

- Dans le fichier **src/mocks/quiz-list.mock.ts**, vous pouvez voir qu'un mock de question est commenté. Décommentez le et ajoutez ce mock dans le quiz "**Les Acteurs**" dans l'objet **QUIZ_LIST**. Avec Typescript (et plus généralement avec JavaScript), vous pouvez directement mettre votre variable à l'intérieur du tableau vide: **[]** se transforme en **[QUESTION_ACTOR]**. Constatez le changement sur votre navigateur.
- Le quiz de sport reste toujours vide. Créez une nouvelle constante **QUESTION_SPORT** et utilisez la pour le quiz sur les sports.

Exercice 2 : Les icônes, c'est bien 🧐💧🌵

Afin de reconnaître d'un coup d'œil les thèmes, Fabien nous demande d'ajouter des icônes propres à chaque thème. Il vous fait confiance sur le choix des icônes, ne le décevez pas 😊

Étapes :

- Dans **src/app/quizzes/quiz/quiz.component.html**, ajoutez une icône spécifique en fonction de la thème : [liste des icônes](#). Vous avez déjà un exemple d'utilisation des icons dans ce fichier.
- Utiliser la directive ***ngIf** pour afficher l'icône en fonction du thème. À l'intérieur de la directive ***ngIf**, vous pouvez mettre les mêmes expressions que dans un if classique.. Pour comparer la valeur du thème et une string, on souhaite utiliser [une comparaison stricte](#).


Exercice 3 : Un formulaire qui marche, c'est mieux !

Fabien nous a envoyé un mail dont voici le sujet : "À part regarder des quizz, je peux faire quoi ?". Vous l'aurez compris, il commence à s'impatienter et le formulaire qui ne marche pas l'a particulièrement énervé 😡

Étapes :

- dans le fichier **src/app/quizzes/quiz-form/quiz-form.component.html**, ajoutez un nouveau label et input text pour pouvoir donner le thème. Vous pouvez également voir dans ce fichier la fonction qui est appelée lors du clic sur le bouton "Create". Cette fonction devrait se trouver dans la classe du composant...
- **src/app/quizzes/quiz-form/quiz-form.component.ts**, ajoutez le nouvel attribut **theme** dans l'objet **quizForm** définit dans le constructeur comme pour l'attribut **name** ([exemple de création d'un form group avec plusieurs attributs](#)). Vous devriez maintenant pouvoir remplir le thème de votre quizz.

Note: Vous trouverez également [des exemples utilisant directement la classe FormControl\(\)](#) pour construire votre quizForm. Ces expressions sont équivalentes, dans notre cas nous utilisons un FormBuilder pour simplifier le code.

 **Tip:** La dernière ligne dans **quiz-form.component.html** est commentée. **Décommentez là** pour voir votre objet quizForm changer dynamiquement lorsque vous remplissez le formulaire !

- Nous avons vu précédemment que la fonction **addQuizz()** était appelée lorsque l'utilisateur cliquait sur le bouton Create. Nous devons mettre à jour cette fonction afin de pouvoir envoyer notre nouveau quiz. Décommentez le code de la fonction et ajoutez l'appel au service de quiz et de sa fonction **addQuiz(quiz: Quiz)**. ([exemple d'utilisation service](#))
- Dans **src/services/quiz.service.ts** remplissez la méthode addQuiz en 2 lignes :
 - Ajout du nouveau quiz dans la liste ([doc array](#))
 - Mise à jour de l'observable ([Update observable with new value](#)).
- Et ça y est !! Vous ajoutez de nouveaux quiz, ça marche ! 🚀
- Attendez, pourquoi est-ce qu'on ne voit pas le nombre de questions pour le nouveau quizz ? 😞 Ouvrez votre console de votre navigateur (F12 et choisissez l'onglet Console).

```
✖ ERROR TypeError: Cannot read property 'length' of undefined
    at Object.eval [as updateRenderer] (QuizComponent.html:10)
    at Object.debugUpdateRenderer [as updateRenderer] (core.js:45294)
    at checkAndUpdateView (core.js:44277)
    at callViewAction (core.js:44637)
    at execComponentViewsAction (core.js:44565)
    at checkAndUpdateView (core.js:44278)
    at callViewAction (core.js:44637)
    at execEmbeddedViewsAction (core.js:44594)
    at checkAndUpdateView (core.js:44272)
    at callViewAction (core.js:44637)
```

- "length of undefined" ? 😞 Où avons nous utilisé length ? Et si on cliquait sur "QuizComponent.html:10" ? Il se peut qu'il faille mettre à jour notre objet quiz lors de sa création avant de l'ajouter...

Exercice 4 : Trop de quizz, tue le quizz...

Voilà qui est nettement mieux, mais à force d'ajouter des quizz, la page de Fabien est complètement remplie ! Il vous demande donc d'ajouter en urgence une fonctionnalité lui permettant de supprimer des quizz car il n'arrive plus à vivre dans ces conditions.

Étapes :

- Dans QuizComponent:
 - **src/app/quizzes/quiz/quiz.component.html** : Ajoutez un bouton de suppression

- **src/app/quizzes/quiz/quiz.component.ts** :
 - Ajoutez un `@Output` pour notifier le composant parent de la suppression. L'output a comme type **`EventEmitter<Quiz>`**.

Note :

Vous avez un exemple d'output appelé **quizSelected** dans QuizComponent sur lequel QuizListComponent réagit (regardez dans la console de votre navigateur) + [documentation pour les @Output](#)

- **src/app/quizzes/quiz/quiz.component.ts**: Créez une fonction **`deleteQuiz()`** qui va émettre le quiz courant.
- **src/app/quizzes/quiz/quiz.component.html**: Ajoutez la fonction **`deleteQuiz()`** à votre bouton de suppression pour qu'elle soit appelée lors du (**`click`**). Vous avez un exemple dans ce composant avec la fonction **`selectQuiz()`**.
- Dans QuizListComponent :
 - **src/app/quizzes/quiz-list/quiz-list.component.ts** : Créez une fonction **`deleteQuiz(quiz: Quiz)`** dans la classe.
 - **src/app/quizzes/quiz-list/quiz-list.component.html** : Modifiez l'appel du composant **`<app-quiz>`** afin de rajouter votre nouvel Output (comme ce qui existe déjà pour **`quizSelected`**) et votre nouvelle fonction (**`deleteQuiz($event)`**) qui sera appelée lorsqu'un événement est émis.
 - **src/app/quizzes/quiz-list/quiz-list.component.ts** : Ajoutez un **`console.log(quiz)`** dans votre fonction **`deleteQuiz()`** et vérifiez que vous voyez bien un log dans la console de votre navigateur lorsque vous cliquez sur votre bouton **`delete`**.
- Dans QuizService : On souhaite maintenant pouvoir supprimer le quiz de la liste
 - **src/services/quiz.service.ts** : Créez une fonction **`deleteQuiz(quiz: Quiz)`** pour supprimer le quiz de la liste et mettre à jour l'observable. Pour mettre à jour l'observable, il vous suffit d'utiliser la fonction **`.next()`** sur votre observable.

```
this.quizzes$.next(quizList);
```

- Dans QuizListComponent:
 - **src/app/quizzes/quiz-list/quiz-list.component.ts** : Dans la fonction **`deleteQuiz(quiz)`**, Appelez votre fonction **`deleteQuiz(quiz)`** de votre service **`QuizService`** et testez la suppression !

Exercice 5 : `input[type=text]` → `select`

Fabien nous fait savoir qu'il ne trouve pas cela très adapté d'entrer du texte pour renseigner le thème. Il aimerait passer par une sélection parmi une liste.

Étapes :

- Dans **src/app/quizzes/quiz-form/quiz-form.component.html**, modifiez le champ d'input text par un [select](#).

Le développeur malade vous envoie un de ses **select** d'une ancienne application pour vous aider :

Dans sa classe **RecipeFormComponent** :

```
public INGREDIENT_LIST: string[] = ['Pasta', 'Nutella'];
```

Dans l'html du composant **RecipeFormComponent**:

```
Ingredient:
<select name="ingredient-field" id="ingredient" [formControlName]="ingredient">
  <!-- We set the formControlName to the string 'ingredient'-->
  <option *ngFor="let currentIngredient of INGREDIENT_LIST" [value]="currentIngredient">
    <!-- We set the value of the option to the object currentIngredient created from the ngFor-->
    {{ currentIngredient }}
  </option>
</select>
```

Exercice 6 : Date de création du quiz

Fabien souhaiterait maintenant pouvoir connaître la date de création d'un quizz. Ajoutons ça dans la liste des quizz.

Étapes:

- Mettez à jour le modèle Quiz en ajoutant un attribut **creationDate?: Date** (la classe Date est native en Javascript, pas besoin de l'importer).
- Dans **src/app/quizzes/quiz-form/quiz-form.component.ts**, ajoutez la date lors de la création de l'objet ([documentation Date\(\)](#) - heure courante)
- Dans **src/app/quizzes/quiz/quiz.component.html**, affichez la date de création seulement si elle est définie. Testez le résultat.
- ... Fabien n'est pas très content de ce format de date ! 😞 C'est vrai que c'est pas très user friendly. Pour un meilleur résultat, nous pouvons utiliser le pipe **date** d'Angular. Vous trouverez [ici un exemple simple](#) d'utilisation du pipe date. Vous pouvez définir le format de la date que vous voulez, voici [la documentation](#) expliquant les formats acceptés.

Exercice 7 : Je veux la vraie liste de quiz

L'ancien développeur est de retour dans la course et a pu coder une partie du back-end. Fabien veut désormais récupérer les quizz depuis un serveur.

URL contenant la liste des quizz:

<https://raw.githubusercontent.com/NablaT/starter-quiz-two/master/mock-quiz.json>

Étapes :

- Ajoutez HttpClientModule dans la liste des **imports:[]** dans **@NgModule** dans **src/app/app.module.ts** ([exemple ici](#))

Note: On découvre un nouveau fichier ici: le module de votre application. Il va contenir l'ensemble de vos dépendances et vos composants. Il a la structure suivante:

```
@NgModule({
  declarations: [
    AppComponent,
    QuizListComponent,
    QuizComponent,
    HeaderComponent,
    QuizFormComponent
  ],
  imports: [
    BrowserModule,
    ReactiveFormsModule,
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Declare your components

Import the modules of your dependencies

Il possède plusieurs tableaux:

- declarations: contient l'ensemble de vos composants. Tous les composants doivent être déclarés ici pour pouvoir fonctionner.
- imports: contient l'ensemble de vos dépendances. On importera ici uniquement des modules. C'est ici que vous ajoutez votre **HttpClientModule**.
- providers: Contiendra l'ensemble des classes qui ont besoin de passer par le dependency injection d'angular. On en parlera plus tard.
- bootstrap: Le composant **root** de notre application.

Étapes (suite):

- Dans QuizService:
 - Créez un attribut pour stocker l'url à appeler
 - Injectez HttpClient dans le constructeur du service ([exemple ici](#))
 - Créez une fonction **getQuizzes()** qui fait l'appel à l'url et récupère la liste des quizz en utilisant **this.http.get<Quiz[]>(url)** ([exemple ici avec /** GET heroes from the server */](#). **Attention** dans l'exemple ils retournent l'observable, mais nous ne voulons pas return).
 - On utilise le **subscribe** sur le **this.http.get<Quiz[]>(url)** pour récupérer la réponse reçue et ainsi mettre à jour la liste des quizz ainsi que le behavior subject. (vous avez un exemple de subscribe dans QuizListComponent). **Attention**, l'objet récupéré est de type **{ quizzes: Quiz[] }** et votre attribut **quizzes** est de type **Quiz[]**.

Note : Pour mettre plusieurs lignes dans votre subscribe, vous devez rajouter des accolades. Le développeur précédent vous a trouvé un exemple avec un ancien projet:

```

this.ticketService.tickets$.subscribe( next: (tickets) => this.ticketList = tickets);
this.ticketService.tickets$.subscribe( next: (tickets) => {
  this.ticketList = tickets;
  console.log(tickets);
});

```

Vous pouvez aussi créer une fonction et l'appeler pour éviter d'avoir des pavés illisibles dans vos subscribes 😊

Fin du TD

Wahou ! Enfin arrivé au bout du TD.. 1... 😊. Fabien va pouvoir enfin faire mumuse avec votre super projet et vous faire de nouveaux retours d'ici peu !

Suite TD1

Exercice 8 : Je veux naviguer dans mon application

Maintenant que nous avons notre liste de quizz, Fabien aimerait pouvoir naviguer au sein de son application pour pouvoir éditer un quiz et rajouter des questions. Pour cela nous allons déjà commencer par afficher une nouvelle page dédiée à l'édition du quizz. Vous allez donc devoir setup un système de routing.

Étapes pour le routing :

- Création du module AppRoutingModuleModule. Ce module se trouve dans un fichier appelé **app.routing.module.ts** et il se trouve au même niveau que les autres fichiers **app.*** (app.module.ts etc...). Vous avez de la chance, le développeur d'avant à trouver un exemple d'**AppRoutingModule** 😊.

```

import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { TicketListComponent } from '../tickets/ticket-list/ticket-list.component';
import { UserListComponent } from '../users/user-list/user-list.component';

const routes: Routes = [
  {path: 'tickets', component: TicketListComponent},
  {path: 'users', component: UserListComponent}
];

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

```


Divisons ce fichier en deux parties:

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

1

- 1 - On crée la classe **AppRoutingModule**. Cette classe étant un module, on ajoute **@NgModule** pour définir ses dépendances (comme dans **app.module.ts**). De quelles dépendances avons-nous besoin ? Seulement du **RouterModule** (toutes les fonctionnalités liées à la navigation). On appelle la fonction **forRoot()** qui prend en paramètre nos nouvelles routes.

Si on importe pas le **RouterModule**, on va avoir l'erreur suivante:

```
✖ ERROR NullInjectorError: StaticInjectorError(AppModule)[QuizListComponent -> Router]:
  StaticInjectorError(Platform: core)[QuizListComponent -> Router]:
    NullInjectorError: No provider for Router!
    at NullInjector.get (http://localhost:4200/vendor.js:36417:27)
    at resolveToken (http://localhost:4200/vendor.js:51335:24)
    at tryResolveToken (http://localhost:4200/vendor.js:51261:16)
    at StaticInjector.get (http://localhost:4200/vendor.js:51111:20)
    at resolveToken (http://localhost:4200/vendor.js:51335:24)
    at tryResolveToken (http://localhost:4200/vendor.js:51261:16)
    at StaticInjector.get (http://localhost:4200/vendor.js:51111:20)
    at resolveNgModuleDep (http://localhost:4200/vendor.js:62298:29)
    at NgModuleRef_.get (http://localhost:4200/vendor.js:63364:16)
    at resolveDep (http://localhost:4200/vendor.js:63895:45)
```

AppComponent.html:4

```
const routes: Routes = [
  {path: 'tickets', component: TicketListComponent},
  {path: 'users', component: UserListComponent}
];
```

2

- 2 - Ensuite on déclare nos routes. Dans notre cas, on a une route **quiz-list** qui affiche le composant **QuizListComponent** et une autre **edit-quiz** qui va afficher un nouveau composant **EditQuizComponent**. Ce composant n'existe pas, vous devez donc le créer dans le dossier **quizzes** en respectant la même structure que pour les autres composants.
- Une fois que votre composant est créé, il faut le **déclarer** dans **app.module.ts** comme ce qui est fait pour les autres composants.

```
declarations: [
  AppComponent,
  QuizListComponent,
  QuizComponent,
  HeaderComponent,
  QuizFormComponent
],
```

Profitez-en aussi pour importer votre AppRoutingModuleModule dans la partie **imports**.

```
imports: [  
  BrowserModule,  
  ReactiveFormsModule,  
],
```

- Remplacez l'appel aux composants dans **app.component.html** par le router-outlet. Cette balise **router-outlet** affichera le composant à afficher pour chaque route (QuizListComponent ou EditQuizComponent)

```
<app-header></app-header>  
<div class="content">  
  <router-outlet></router-outlet>  
</div>
```

- Vous pouvez maintenant naviguez à la page <http://localhost:4200/edit-quiz> et <http://localhost:4200/quiz-list>. **QuizFormComponent** a disparu, ajoutez le dans **quiz-list.component.html** pour pouvoir toujours créer des quizz.
- Fabien n'aime pas trop devoir manuellement entrer le nom de la page, il aimerait arriver par défaut sur la page qui affiche la liste des quiz. Définissez [une route par défaut](#).
- Ajoutez un lien "Edit" dans le composant QuizComponent pour pouvoir naviguer vers la page d'édition. [Ici un exemple](#).
- On veut maintenant afficher les informations du quiz qu'on a sélectionné. Pour cela, il faut rajouter un paramètre à la route pour avoir la structure suivante: **/edit-quiz/:id** avec :id une variable contenant l'id du quiz. Comme vos quiz ne contiennent pas d'id aujourd'hui, mettez à jour model et mocks pour qu'un quiz ait un id (string). Ensuite, vous trouverez [un exemple ici](#) de configuration à faire dans **app.routing.module.ts** pour donner l'id.

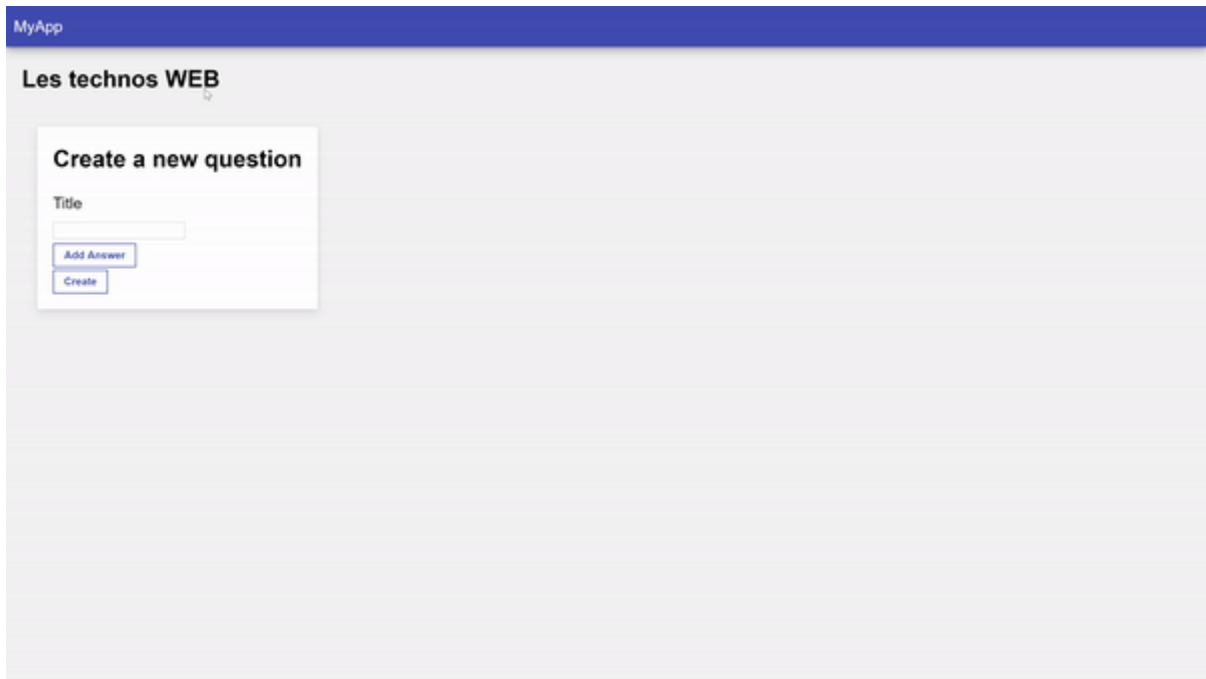
Useful link :

- Toutes les étapes du routing: [Tutorial routing](#)

Exercice 9 : Je veux ajouter des questions

Nous allons maintenant remplir notre page qui est maintenant tristement vide... Fabien souhaite voir le titre du quiz sélectionné en haut de la page et ensuite conserver la même structure que la page précédente: un widget pour créer une question et en dessous la liste des questions.

Comme Fabien est sympa, il vous a préparé une petite démo de ce qu'il voudrait pour la première version de la page d'édition d'un quiz.



- Créez 3 nouveaux composants: QuestionsComponent, QuestionFormComponent et QuestionListComponent.
- Dans EditQuiz component, récupérez l'id du quizz qui se trouve dans l'url et trouvez le quiz correspondant à cet id. Ce quiz sera nécessaire pour les composants **QuestionListComponent** et **QuestionFormComponent**.

 **Tip:** [Comment extraire l'id de la route? Section "Routable HeroDetailComponent" & "Extract the id route parameter"](#)

- Dans QuestionFormComponent, vous devez pouvoir afficher plusieurs réponses possibles. Il faut utiliser un nouveau type de form appelé FormArray. On va stocker les différentes réponses sous forme d'un tableau.

```
constructor(public FormBuilder: FormBuilder) {  
  // Form creation  
  this.initializeQuestionForm();  
}  
  
private initializeQuestionForm() {  
  this.questionForm = this.formBuilder.group({  
    label: [''],  
    answers: this.formBuilder.array([])  
  });  
}
```

- On ajoute un getter pour exposer notre form answers.

```
get answers() {
  return this.questionForm.get('answers') as FormArray;
}
```


- Dans l'html, on rajoute un bouton pour ajouter des réponses et on ajoute nos champs pour chaque réponse.

```
<button class="button-card" (click)="addAnswer()">Add Answer</button>
<div class="answer" formArrayName="answers">
  <div *ngFor="let address of answers.controls; let i=index">
    <div class="answer-form" [formGroupName]="i">
      <label>
        Answer
        <input type="text" [formControlName]='value'>
        <br>
        Correct
        <input type="checkbox" [formControlName]='isCorrect'>
      </label>
    </div>
  </div>
</div>
```

- Ensuite, on appelle la fonction addAnswer() qui va rajouter un FormGroup dans notre FormArray

```
private createAnswer() {
  return this.formBuilder.group({
    value: '',
    isCorrect: false,
  });
}

addAnswer() {
  this.answers.push(this.createAnswer());
}
```

 **Tip:** Plus d'informations sur [les FormArray ici](#)

- Le développeur qui vous aide est maintenant de nouveau malade ! Mais Fabien s'attend toujours à ce que l'application soit terminée. Il va falloir terminer seul l'implémentation des fonctionnalités de la démo...

Exercice 10 : Je veux des utilisateurs

Fabien voudrait maintenant pouvoir ajouter et supprimer des utilisateurs dans son application. On souhaite avoir la même structure de page que pour l'ajout d'un quiz: un formulaire en haut pour créer un utilisateur et la liste en dessous avec la possibilité de les supprimer.