

Durée : 2h30

Angular Test

Partie 1 : Questions cours :

Question 1: En: What are Single Page Applications?

Une single page application est une application web possédant une unique page qui est réactive, donc qui se met à jour.

Question 2: What are some advantages of using Angular framework for building web applications?

Angular est un framework javascript permettant la création rapide de fonction utilisant des variables réactive afin de modifier la page en temps réel, d'avoir une structure html plus intuitive.

Cela permet à l'utilisateur un développement plus rapide .

Question 3: In Angular, how can you interact between Parent and Child components?

Il est possible d'interagir entre l'enfant et le parent à travers des @Input et des @Output

Question 4. Write an example usage of ngFor for displaying all items from an array '*Items*' in a list with .

```
<li *ngFor="let item of Items">
  {{ item }}
</li>
```

Question 5. Write an example usage of ngIf for displaying an item from an array 'Items' in a list with a condition .

```
<li *ngFor="let item of Items">
  <p *ngIf="item.id == 3">{{ item }}</p>
</li>
```

Question 6: What are the basic parts of an Angular application?

```
* src/
* node_modules/
* angular.json
* package.json
* package-lock.json
```

Question 7 : What is OnInit() ?

OnInit() est une méthode appelée lorsque le composant vient de finir de s'instancier

Question 7 : Why we use async/await with promess?

On utilise `async` afin de spécifier qu'une fonction est asynchrone.

Une fonction asynchrone est une fonction qui possède un appel de fonction qui va utiliser le mot clé `await`.

Le mot clé `await` indique que l'on souhaite attendre la réponse d'une fonction avant de passer à la suite.

Par exemple, lorsque l'on fait appelle à une API, la fonction nous retourne une promesse.

Si « `await` » n'est pas spécifié, le programme va passer aux instructions suivantes sans connaître le résultat de l'appel à l'API.

Si « `await` » est spécifié, le programme attendra la réponse de l'API : que la promesse soit résolue.

Question 8 : comment crée-t-on une *promise* ?

Aide : il y a une nouvelle classe *Promise*, dont le constructeur attend une fonction avec deux paramètres, *resolve* et *reject*.

```
const getUser = function (login) {  
  return new Promise (function() {  
    // async stuff, like fetching users from server, returning a response  
    if (response.status === 200) {  
      return Promise.resolve()  
    } else {  
      return Promise.reject()  
    }  
  })  
};
```

Question 8 : Prenons un cas d'utilisation simple, où on doit récupérer un utilisateur, puis ses droits, puis mettre à jour un menu quand on a récupéré tout ça .

```
//Avec les callBacks :
getUser(login, function (user) {
  getRights(user, function (rights) {
    updateMenu(rights);
  });
});
```

```
//Avec les Promises ???
getUser().then((user) => {
  getRight().then((right) => {
    updateMenu(right)
  }, (error) => {
    console.log(error)
  })
}, (error) => {
  console.log(error)
})
```

Question 9 : Faire Une gestion d'erreur globale pour toute la chaîne .

```
try {
  let user = await getUser(login)
  let rights = await getRights(user)
} catch (err) {
  console.log(err)
}
```

Question 10 : Faire une gestion d'erreur par *promise*.

```
try {
  let user = await getUser(login)
} catch (err) {
  console.log(err)
}

try {
  let rights = await getRights(user)
} catch (err) {
  console.log(err)
}
```

Partie 3 : Tuto code

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 **QUIZZ_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** défini 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 quizz. Décommentez le code de la fonction et ajoutez l'appel au service de quizz 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 quizz dans la liste ([doc array](#))
 - Mise à jour de l'observable ([Update observable with new value](#)).

- Et ça y est !! Vous ajoutez de nouveaux quizz, 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 quizz lors de sa création avant de l'ajouter...

Exercice 4 : Trop de quizz, tue le quizz... (Bonus)

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 !

Fichier code :

Lien téléchargement du code : <https://we.tl/t-aeWk8Njnuz>



Figure 1 : quiz.service.ts

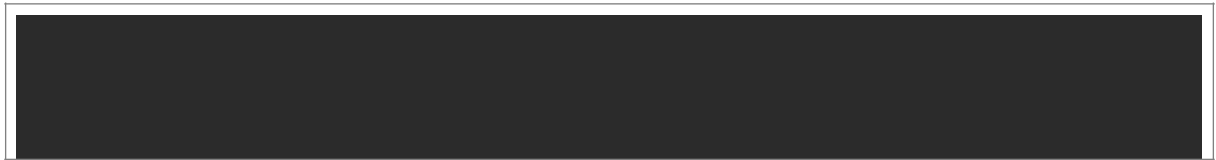


Figure 2. quiz.component.html

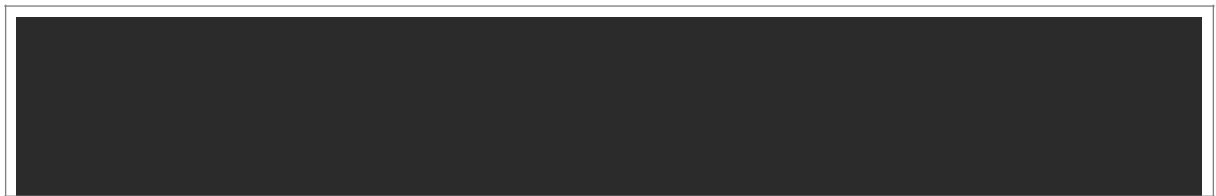


Figure 3. quiz.component.ts.

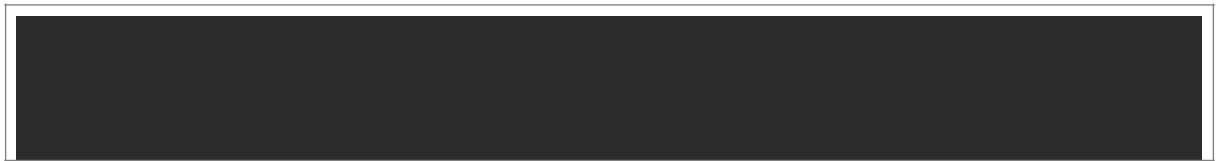


Figure 4. : quiz-list.component.ts.

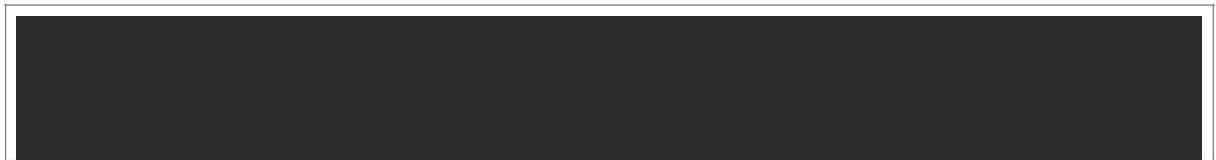


Figure 5. quiz-list.component.html

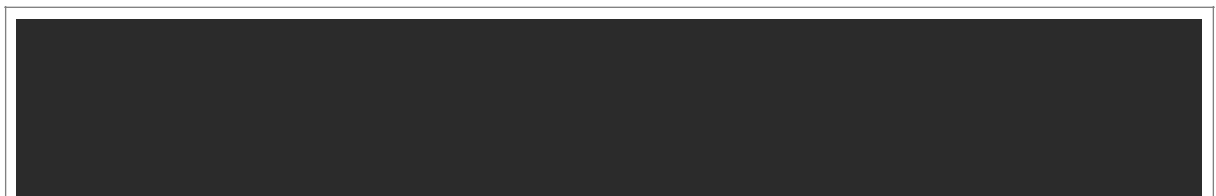


Figure 6. quiz-form.component.ts.



Figure 7. quiz-form.component.html.