POLYTECH MARSEILLE Aix*Marseille Université **Informatique – 4A**

Applications web et mobiles

TP n°6 : Les topics

Étape 30 – Liens vers les topics

Quand l'utilisateur clique sur une ligne de votre table des cours, cela le mène à la page des *topics* correspondante, cf. le forum de démonstration. Pour cela, ajoutez une ancre `<a></a>` dans la colonne des noms de cours. Pour l'instant, mettez dans l'ancre une url *bidon*. Pour éviter d'avoir à rajouter des ancres sur les autres colonnes, exploitez la classe `stretched-link` de bootstrap : (https://getbootstrap.com/docs/5.3/helpers/stretched-link). Dans cette documentation, regardez bien où la classe `position-relative` doit être ajoutée.

⚠ N'oubliez pas que, dans les ancres, il faut utiliser `routerLink` et non `href` [1].

Étape 31 – Les vrais liens vers les topics

Maintenant, remplacez l'URL « bidon » par la vraie URL dont vous avez besoin (par exemple, `/topics/id`, où `id` permet d'identifier le *cours* dans votre base de données). Évidemment, pour que cela fonctionne, il faudra créer un composant `topics`.

Étape 32 – Backend : récupération de la liste des topics par PHP

Dans votre *backend*, écrivez un script PHP qui vous renvoie la liste des topics associés à un cours ainsi que le nom du cours en question (ce sera utile pour afficher le breadcrumb en haut de la page des topics). Bien évidemment, vous testerez que l'utilisateur suit bien le cours en question avant de renvoyer la liste des topics demandée. Testez votre script avec Postman.

Étape 33 – Récupération des topics dans la classe TypeScript

À l'instar de ce que vous avez fait pour la classe `CoursComponent`, vous devez faire en sorte que votre classe TypeScript `TopicsComponent` récupère la liste des topics du cours sélectionné en appelant votre script PHP via votre classe `MessageService`. Ici, toutefois, il faut que vous transmettiez au script PHP l'identifiant du cours en question, en le passant comme deuxième argument à votre méthode `sendMessage`. Encore faut-il que votre classe `TopicsComponent` connaisse cette information qui, on

1. Si vous utilisez `href` au lieu de `routerLink`, cela obligera *Angular* à retélécharger toute votre application. L'attribut `routerLink` permet de juste modifier la vue, sans nécessiter de nouveau téléchargement.

peut le rappeler, est accessible via le paramètre `id` que vous avez déclaré dans vos routes (fichier `app-routes.ts`). Fort heureusement, *Angular* vous permet d'accéder à ce paramètre facilement : dans le constructeur de `TopicsComponent`, rajoutez par *dependency injection* une instance, appelons-la `route`, de la classe `ActivatedRoute`. Dans ce cas, un appel à :

```
this.route.snapshot.paramMap.get('id');
```

vous renverra une chaîne de caractères contenant la valeur du paramètre `id`. Pour être plus précis, le type de la valeur retournée par cet appel est `string|null`. Cela risque de vous poser des problèmes par la suite. En effet, il serait judicieux que ce soit juste une `string`. Pour pallier cela, vous pouvez utiliser l'astuce suivante, qui vous permettra de produire un `this.course_id` de type `string` :

```
const id = this.route.snapshot.paramMap.get('id');
this.course_id = id === null ? '' : id;
```

Si `sendMessage` vous renvoie un message d'erreur, changez la vue vers la page de login (via la méthode `navigateByUrl` que vous aviez utilisée pour la page de login). Sinon, affichez dans la console le contenu du message renvoyé par `sendMessage`.

Comme pour les cours, n'oubliez pas que tout ce qui concerne la récupération d'informations distantes doit être réalisé dans la méthode `ngOnInit` et non dans le constructeur.

Étape 34 – La table des données

Écrivez le template HTML ainsi que les instructions TypeScript affichant la table avec les *topics*.