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

Applications web et mobiles

TP n°9 : Sécurité

Étape 47 – Service d'authentification

Dans la classe `LoginComponent`, on peut utiliser directement la classe `MessageService` pour réaliser l'authentification de l'utilisateur mais ce n'est pas souhaitable car, si l'utilisateur accède directement à une autre page de votre application, cette page (ce composant) ne saura pas si l'utilisateur s'est bien authentifié ou non. Pour pallier ce problème, créez un nouveau service `auth`, qui conservera cette information dans un de ses attributs. Créez donc cet attribut. Par *dependency injection*, une instance de `MessageService` sera passée au constructeur de la classe `AuthService`. Dans le constructeur, affichez dans la console un message indiquant qu'une instance de `AuthService` vient d'être construite.

Rajoutez à votre classe `AuthService` une méthode `sendAuthentication` qui, étant donné une chaîne de caractères `login` et une chaîne `password` passées en paramètres, transmet ces informations au *backend* et renvoie un Observable sur la réponse du serveur. Modifiez votre composant `LoginComponent` afin qu'il n'utilise plus `MessageService` mais `AuthService`. Testez bien que votre nouveau composant fonctionne correctement.

Rajoutez une deuxième méthode `finalizeAuthentication` qui, étant donné un message au format `PhpData`, donc une réponse reçue du serveur, vérifie si ce message indique une authentification réussie ou non. La fonction met alors à jour l'attribut de sa classe indiquant si on est bien connecté ou non.

L'idée derrière ces 2 méthodes est la suivante : le composant `LoginComponent` va appeler la première méthode afin d'envoyer les identifiants de l'utilisateur vers le *backend*. Le `LoginComponent` récupérera tout de suite l'Observable et, en souscrivant à ce dernier, il récupérera le message de retour du *backend* une fois que celui-ci l'aura envoyé. `LoginComponent` transmettra alors ce message à la méthode `finalizeAuthentication` qui mettra à jour l'attribut indiquant si la connexion a été réussie ou non. Par la suite, il suffira de lire la valeur de cet attribut pour déterminer si, oui ou non, l'utilisateur est bien connecté.

Modifiez `LoginComponent` pour utiliser, maintenant, `AuthService` plutôt que `MessageService`.

Étape 48 – Garantir l'authentification pour accéder aux cours

Actuellement, vous ne pouvez empêcher un utilisateur d'accéder à la page de cours sans s'authentifier. Pour forcer cette authentification, créez une garde :

```
ng generate guard auth
```

et indiquez que vous souhaitez que cette garde implémente « canActivateChild ».

Dans votre fichier `app-routes.ts`, vous ferez en sorte que les routes `cours` et `topics/:id` soient

enfants d'un path vide, cf. l'URL :
https://angular.io/guide/router-tutorial-toh#canactivatechild-guarding-child-routes.
Ce path ne sera toutefois associé à aucun composant.

Avant la propriété `children`, rajoutez :

```
canActivateChild: [authGuard],
```

Éditez le fichier `auth.guard.ts` et faites en sorte que la fonction `authGuard` renvoie `true`, comme le fait le premier exemple de la section « Guard the admin feature » de l'URL https://angular.io/guide/router-tutorial-toh#canactivate-requiring-authentication. Pour l'instant, on peut encore accéder aux cours sans authentification, mais, chaque fois que l'on essaye d'y accéder, on passe d'abord par votre fonction `authGuard` qui indique si, oui ou non, elle vous permet ces accès. Pour s'en convaincre, il suffit que votre fonction `authGuard` affiche un message dans la console.

Maintenant, dans le fichier `auth.guard.ts`, faites en sorte que la fonction `authGuard` renvoie `false`. Vous ne pouvez plus accéder aux cours (en tous cas, vous ne voyez plus la liste des cours).

Étape 49 – Finalisation de la garde

Modifiez votre fonction `authGuard` de manière à utiliser votre service `AuthService`. L'exemple juste avant la section « Add the LoginComponent » de l'URL https://angular.io/guide/router-tutorial-toh#canactivate-requiring-authentication montre comment faire :

Il faut que votre fonction `authGuard` récupère votre instance de `AuthService` et regarde si l'attribut qu'elle contient indique si on est loggué ou non. Pour obtenir cette instance, utilisez l'instruction `inject(AuthService);`, comme indiqué dans l'exemple.

⚠ Il y a une petite subtilité concernant les injections : si vous avez inclus `AuthService` dans les `providers` du fichier `login.component.ts`, ce provider aura construit une instance de `AuthService` et votre `authGuard` en construira une autre. Vous pourrez l'observer dans les messages de la console. Cela vous empêchera d'accéder à la page des cours. Pour pallier cela, supprimez `AuthService` de la liste des `providers` du fichier `login.component.ts`.

Normalement, si vous essayez d'accéder à la page des cours sans authentification, vous n'y aurez pas accès. En revanche, une fois authentifié, vous aurez accès à votre page de cours.

Optionnel : pour ceux/celles qui ont fini en avance

Étape 50 – Déploiement de votre site

Actuellement, votre site est en développement (sur le port 4200 de Localhost ou 127.0.0.1). Si vous souhaitez le déployer sur votre serveur Apache, vous devrez réaliser les opérations suivantes :

Tout d'abord, en production, l'URL de votre *backend* est en général différente de celle utilisée en développement : typiquement, l'adresse IP est différente de 127.0.0.1. En conséquence, vous devez spécifier à votre classe `MessageService` la bonne URL.

Ensuite, dans votre répertoire `frontend`, tapez la commande :

```
ng build --base-href /forum-angular-apache/
```

Cela vous compilera votre application dans un sous-répertoire de `frontend` appelé `dist/frontend/browser`.

 le dernier « / » est important, ne l'oubliez pas.

Déplacez ce répertoire via la commande :

```
mv dist/frontend/browser XXX/forum-angular-apache
```

où `XXX` est le répertoire racine de votre serveur Apache. Maintenant, dans votre navigateur, l'URL `http://127.0.0.1/forum-angular-apache` contient votre application et elle est servie par votre serveur Apache.

Le `build` permet de produire une version optimisée de votre application. Vous pouvez d'ailleurs utiliser l'application `lighthouse` si vous souhaitez vérifier l'efficacité de votre application (cf. `https://github.com/GoogleChrome/lighthouse`).