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

Applications web et mobiles

TP n°4 : Login et messagerie

Étape 16 – Navbar

Dans votre frontend, rajoutez un nouveau composant `navbar`. Celui-ci contiendra la barre de navigation en haut des pages. Pour la créer, exploitez la classe « navbar » de la balise `<nav>` fournie par bootstrap (cf. https://getbootstrap.com/docs/5.3/components/navbar).

Comme vous pouvez le voir sur le forum de démonstration, la *navbar* est identique, quelle que soit la page du *frontend*. Pour l'insérer de manière automatique dans toutes les pages, il suffit d'inclure ce composant dans `app.component.html` au dessus des balises du *router-outlet*. Faites-le et vérifiez que vous voyez bien votre barre de navigation.

Étape 17 – Création de la page de login

Rajoutez un nouveau composant `login`, qui contiendra à terme votre page de login. Comme ce sera une page, il faut qu'elle ait sa propre URL. Faites en sorte, en éditant les routes du fichier `app-routes.ts` que ce composant soit accessible via le chemin `'login'`. Vérifiez que vous voyez bien le composant (message « login works ! »).

Étape 18 – Template HTML de la page de login

Écrivez le contenu du template HTML du composant `login` (`login.component.html`). Pour l'instant, focalisez-vous uniquement sur l'affichage produit (par les balises `<input type="text">`, `<button>`, *etc.*). Ici, il est inutile d'utiliser des formulaires (`<form>`).

Quand l'affichage vous paraît correct, faites en sorte que, si on clique sur le bouton « se connecter », cela affiche dans la console le *login* et le *password* saisis par l'utilisateur. On rappelle que si l'on clique droit dans une page, en sélectionnant l'item « Inspect element », on ouvre les outils de développement du *browser*, dont un des *tabs* correspond à la console.

Étape 19 – Création d'un service de messagerie

Créez un service `message` via la commande `ng generate service message/message` (cela vous créera un répertoire `message` contenant le service). Ce service sera utilisé pour abstraire la transmission des requêtes `http` transmises au *backend*.

Étape 20 – Méthode d'envoi de messages

Rajoutez à votre classe `MessageService` une méthode `sendMessage` qui prend en argument une chaîne de caractères `url` représentant une URL ainsi qu'un deuxième argument `data` (de type `any`), qui représente les données du POST à envoyer au *backend*. Pour l'instant, indiquez que votre fonction renverra une valeur de n'importe quel type (`any`).

Pour rendre votre programme assez générique, la chaîne `url` ne va pas contenir l'URL complète à laquelle on essaye d'accéder, mais seulement la partie après le dernier « / », et sans l'extension « .php ». Par exemple, si l'URL complète est :

```
https://christophe-gonzales.pedaweb.univ-amu.fr/forum/fr-FR/backend/checkLogin.php
```

le paramètre `url` ne contiendra que `checkLogin`. Vous allez sauvegarder toute la partie de l'URL complète avant le dernier « / », autrement dit le préfixe de l'URL complète, dans un membre de votre classe `MessageService`. La première opération que doit donc réaliser votre méthode `sendMessage` consiste à recréer l'URL complète en concaténant le préfixe, l'argument `url` et l'extension « .php ».

Pour tester votre méthode `sendMessage`, faites en sorte qu'elle retourne l'URL que vous avez complétée. Dans votre composant `LoginComponent`, passez une instance de `MessageService` par *dependency injection* et faites en sorte que la méthode `sendMessage` soit exécutée quand vous cliquez sur le bouton « se connecter » du `LoginComponent` et que l'url qu'elle retourne soit affichée dans la console.

⚠ À ce stade, dans la console de votre navigateur, vous devriez voir une exception `NullInjectorError`. Pour pallier cela, il faut rajouter `provideHttpClient()` dans le tableau des `providers` du fichier `app.config.ts`, comme vu en cours.

Étape 21 – Type de retour de sendMessage

Il faut définir le format des messages qui vont transiter de votre *backend* vers votre *frontend Angular*. Pour cela, regardez ce que renvoient les fonctions `sendMessage` et `sendError` du fichier `helper.php` de votre *backend*. Cela vous indique les champs des objets JSON qui vous seront transmis. Créez dans le fichier `message.service.ts` une interface `PhpData` qui représente ce type d'objets JSON.

Étape 22 – sendMessage et les données du POST

Le 2ème argument de la méthode `sendMessage` de votre classe `MessageService`, appelons-le `data` est un objet Javascript/TypeScript (dont le type sera `any`) contenant tous les paramètres permettant de spécifier à quel cours/topic/post, on souhaite accéder. Par exemple, pour authentifier votre utilisateur, `data` sera égal à un objet :

```
{ login: 'mon_login', password: 'mon_password' }
```

Pour transmettre cet objet Javascript à votre *backend*, il faut le transformer en `FormData`, qui correspond précisément au type d'objets que l'on transmet via des POSTs avec encodage « multipart/form-data » (l'encodage attendu par votre *backend* PHP). Opérez cette transformation, comme vu en cours (cf. https://developer.mozilla.org/fr/docs/Web/API/FormData). Vous devez faire en sorte que cette transformation fonctionne quels que soient les champs de l'objet, pas seulement le login/password.