Examen de première session

Durée : 1 heure 30

Documents autorisés : 1 feuille A4 recto-verso

Exercice 1 (3 points) — *Login – backend PHP*

Une base de données contient une table `users` représentée dans la figure 1, dont les mots de passe sont non chiffrés. On suppose que l'on possède un fichier `mysqlConnect.php` qui crée, en tant que variable **globale**, une instance `$pdo` de PDO et qui connecte celle-ci à la base de données.

id	login	password
1	admin	123
...	...	...
42	grumph	burp

FIGURE 1 – La table `users`.

On possède également le fichier `helper.php` que vous avez utilisé en TP (excepté qu'il n'inclut pas de fichier `auth.php`). Écrivez en `PHP` un script `login.php` qui teste qu'il a bien reçu en POST un login et un password. Si ce n'est pas le cas ou si la table `users` ne contient pas ce couple login/password, le script renvoie un message d'erreur. Sinon, il stocke dans une variable de session l'id de l'utilisateur et renvoie le message « Connexion ok ». Pour l'envoi des messages, le script utilisera uniquement les fonctions `sendMessage` et `sendError` du fichier `helper.php`.

Exercice 2 (4 points) — *Login – frontend Angular*

On souhaite avoir un composant Angular permettant de réaliser la connexion à notre application, comme indiqué dans la figure 2. Pour cela, on suppose qu'on a déjà programmé un service de messagerie via une classe `MessageService`, qui contient une méthode `sendMessage(url: string, data: any): Observable<any>` qui transmet à l'URL spécifiée dans son premier argument les données contenues dans l'objet `data` du 2ème argument, encodées sous forme de `FormData`.

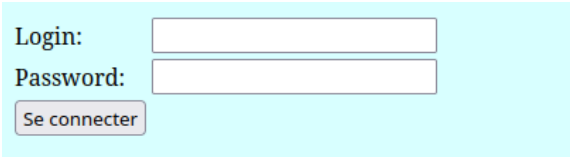

FIGURE 2 – Le composant de connexion Angular.

Écrivez le contenu du template HTML ainsi que la classe TypeScript du composant Angular. Quand l'utilisateur clique sur le bouton « Se connecter », le composant transmet ces données à l'URL `http://127.0.0.1/login.php` de l'exercice 1. Si ce script indique que tout s'est bien passé, l'utilisateur est redirigé vers la vue/page `/dashboard` de l'application, sinon le message d'erreur renvoyé par le script `login.php` apparaît, comme le montre la figure 3. En ce qui concerne le .ts, vous n'indiquerez que le contenu de la classe. Il est donc inutile d'écrire les `import` ou le décorateur `@component`.

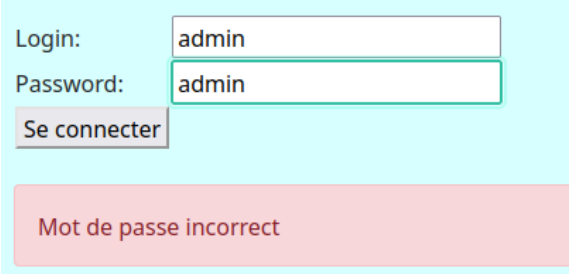

FIGURE 3 – Le composant de connexion Angular.

Exercice 3 (2 points) — *Annonces – backend Node, partie 1*

Notre base de données contient une table `annonces` représentée dans la figure 4.

auteur	annonce	date
toto	annonce1	11/02/2011
...	...	...
titi	grumph	22/02/2022

FIGURE 4 – La table `annonces`.

Écrivez en javascript une fonction `getAnnonces` qui prend en argument le nom d'un auteur et qui renvoie une `Promise` qui contiendra la liste de toutes les annonces de cet auteur. Vous supposerez ici que votre programme contient une variable « globale » `db` qui est déjà connectée avec la base de données (via l'instruction `const db = mysql2.createConnection(...)`).

Exercice 4 (2 points) — *Annonces – backend Node, partie 2*

On suppose que la fonction de l'exercice précédent est écrite et exportée dans un fichier `getAnnonces.js`. On suppose également que les fonctions `sendMessage` et `sendError` des TP sont écrites et exportées dans le fichier `message.js`, comme en TP. On s'intéresse ici à réaliser une partie d'un backend Node/Express. Pour cela, on supposera que le frontend appelle votre backend via un `post` en utilisant la route `/annonces`. Il lui transmet le nom d'un auteur sous la forme d'un objet javascript `{auteur: valeur}` contenu dans le `body` de sa requête (comme vous l'avez vu en TP). Le backend lui renvoie alors la liste des annonces de cet auteur ou un message d'erreur (via les fonctions `sendMessage` et `sendError`).

Écrivez une fonction Javascript `annonces` qui s'occupe de traiter toute la requête du frontend (test de l'existence du nom d'auteur, envoi d'un message d'erreur s'il n'existe pas ou récupération et envoi de la liste des annonces de l'auteur sinon). On ne vérifiera pas, ici, que l'utilisateur s'est identifié.

Exercice 5 (1 point) — *Annonces – backend Node, partie 3*

Indiquez l'instruction Express qui permet d'appeler la fonction de l'exercice 4 quand le frontend envoie sa requête via un `post` sur la route `/annonces`.

Exercice 6 (2 points) — *Annonces – frontend en Angular, partie 1*

Écrivez la méthode `sendMessage` d'un service de messagerie permettant de requêter votre backend node. Son prototype est : `sendMessage(url: string, data: any): Observable<any>`. Elle permet de transmettre dans son `body` les données `data` à l'`url` du backend Node/Express.

Exercice 7 (2 points) — *Annonces – frontend en Angular, partie 2*

Écrivez un composant `AfficheUneAnnonceComponent` en Angular (partie TypeScript et template HTML, comme dans l'exercice 2) qui récupère par *property binding* le texte d'une annonce et l'affiche dans une balise `<div class="alert alert-info">`.

Exercice 8 (4 points) — *Annonces – frontend en Angular, partie 3*

On souhaite maintenant développer un composant Angular `AnnoncesComponent` permettant d'afficher la liste des annonces d'un auteur. Dans le fichier `app-routing.module.ts`, la route de ce composant est la suivante, qui permet d'identifier de quel auteur il s'agit :

```
{path: 'annonces/:auteur', component: AnnoncesComponent}
```

Écrivez un composant Angular (partie TypeScript et template HTML) qui récupère les annonces de l'auteur en utilisant le service de messagerie de l'exercice 6 et qui les affiche en utilisant le composant `AfficheUneAnnonceComponent` de l'exercice 7, cf. la figure 5. On supposera ici que l'url du backend à requêter est : `http://127.0.0.1:3000/annonces`

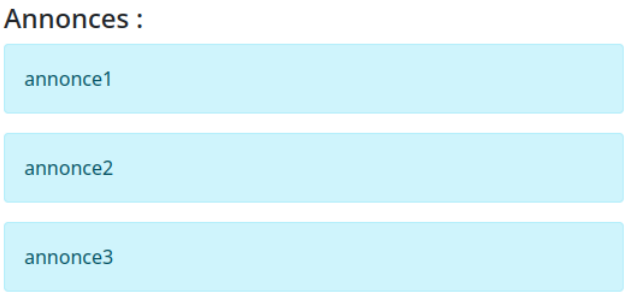

FIGURE 5 – Exemple d'affichage d'annonces.