



Applications web et mobiles

TP n°5 : Du login à la page de cours

Étape 23 – Envoi réel de messages

Modifiez le type de retour de votre méthode `sendMessage` : c'est maintenant un `Observable<PhpData>` (cf. l'étape 21 pour la définition de `PhpData`). Rajoutez à votre méthode les instructions qui envoient les informations du `FormData` à votre *backend* via son instance `http` de `HttpClient`. N'oubliez pas l'option `{withCredentials: true}` afin de capturer les cookies de session.

Utilisez votre méthode `sendMessage` dans votre composant `LoginComponent` : quand vous cliquez sur le bouton de connexion, `sendMessage` envoie les informations de connexion à votre script `checkLogin.php`, et ce dernier vous retourne un message vous indiquant si tout s'est bien passé. Ce message est récupéré via un `subscribe`, comme vu en cours. Affichez le dans la console afin de vérifier qu'il est correct.

Étape 24 – Gestion des erreurs

Dans la page de login, faites en sorte que la classe `LoginComponent` possède un attribut `errorMessage`, qui contiendra les messages d'erreur éventuels du *backend*. Par exemple, si, dans le forum de démonstration, vous ne remplissez pas le login ou le password et que vous cliquez sur le bouton de connexion, vous verrez apparaître un message d'erreur. Dans votre forum, faites en sorte que, si `errorMessage` est non vide, on voit le message en bas de la page sur un fond rose (cf. les *alerts* de *bootstrap*), et que l'on ne voit rien sinon.

Étape 25 – L'utilisateur est loggué

Si le message de retour de `checkLogin.php` indique que l'authentification n'est pas réussie, affichez dans votre page web le message d'erreur renvoyé par votre *backend* (cf. la variable `errorMessage`). Si, en revanche, le serveur vous indique que l'authentification a été réussie, exécutez l'instruction :

```
this.router.navigateByUrl('/cours');
```

où `router` est une instance de la classe `Router` passée par *dependency injection* à `LoginComponent`, et où `'/cours'` est l'url dans votre application de la page des cours. La méthode `navigateByUrl` permet de dire à *Angular* de changer de vue (de page) et d'afficher dans le browser, non plus l'URL `127.0.0.1:4200/login`, mais l'URL `127.0.0.1:4200/cours`.

Pour tester votre application, il vous faudra créer un nouveau composant `cours` correspondant à la liste des cours et indiquer dans le `app-routes.ts` que la route de ce composant est `'cours'`.

Étape 26 – Récupération de la liste des cours

Dans la méthode `ngOnInit` du composant `CoursComponent`, faites en sorte de récupérer du *backend* la liste des cours suivis par l'utilisateur, et mettez à jour votre *backend* afin que celui-ci vous envoie ces informations (n'oubliez pas de vous servir de la variable `$_SESSION` de PHP).

Dans le fichier `cours.component.ts`, créez une interface `Cours` indiquant le type des données contenues dans les enregistrements retournés par votre *backend* (autrement dit celles contenues dans les éléments du champ `data` du `PhpData`).

Affichez dans une console la liste des informations transmises par votre *backend* afin de déboguer.

Étape 27 – La table des cours

Pour afficher la liste des cours, le plus simple est d'utiliser une table d'*Angular Material*, cf. l'URL <https://material.angular.io/components/table/overview>. En haut de la page, cliquez sur « API ». Cela vous indique le module à rajouter dans votre composant `CoursComponent`.

Revenez à l'onglet « OVERVIEW ». Lisez le premier exemple de cette page. Notez qu'en cliquant en haut à droite de l'exemple sur l'icône <>, vous pourrez visualiser le template HTML et le code TypeScript de l'exemple. Adaptez l'exemple pour afficher vos cours.



ce qu'il faut retenir de cet exemple :

- Dans le HTML :
 - `dataSource` est l'attribut de `CoursComponent` contenant les données de la base de données ;
 - les `matColumnDef` correspondent aux noms des colonnes de votre base de données ;
 - dans les balises `<th>`, on indique les titres des colonnes ;
 - dans les `{{element.champ}}` des `<td>`, les `champs` correspondent aux noms des colonnes de votre base de données ;
 - les deux `<tr>` à la fin de l'exemple correspondent respectivement à la 1ère ligne de la table (avec les noms des colonnes), et aux lignes contenant les cours.
- Dans le TypeScript :
 - `PeriodicElement` correspond à votre interface `Cours` ;
 - `ELEMENT_DATA` est le tableau de `data` retourné par votre *backend* ;
 - `TableBasicExample` correspond à `CoursComponent` ;
 - dans `displayedColumns`, on indique les noms des colonnes de la base de données que l'on souhaite afficher.

Optionnel : pour ceux/celles qui ont fini en avance

Étape 28 – Pagination


Dans le forum de démonstration, vous pouvez observer qu'en bas des tables, il y a un composant permettant de découper la table en plusieurs pages (un *paginateur*). Pour en rajouter un, observez l'exemple de l'URL <https://material.angular.io/components/table/overview#pagination>.

 ce qu'il faut retenir de cet exemple :

- Dans le HTML :
 - Seule la balise `mat-paginator` a besoin d'être ajoutée après la table.
- Dans le TypeScript :
 - il faut ajouter `MatPaginatorModule` dans la liste des modules ;
 - dans la classe `CoursComponent`, il faut redéfinir le `dataSource` :

```
dataSource = new MatTableDataSource<Cours>([]);
```
 - il faut ajouter aussi `@ViewChild(MatPaginator) paginator!: MatPaginator;`
 - dans la méthode `ngOnInit()`, après avoir récupéré les données du *backend*, on initialise le `datasource` et le `paginator` :

```
this.dataSource.data = res.data; où res est l'objet retourné par votre MessageService.  
Ici, notez que c'est bien le champ data du dataSource qu'il faut affecter. Ensuite :  
this.dataSource.paginator = this.paginator;
```

 Pour que cela fonctionne correctement, il faut ajouter `provideAnimations()` dans la liste des `providers` du fichier `app.config.ts`

Étape 29 – Tri des colonnes

L'URL <https://material.angular.io/components/table/overview#sorting> vous montre comment trier les colonnes de vos tables *Angular Material*.

 ce qu'il faut retenir de cet exemple :

- Dans le HTML :
 - il faut ajouter `matSort` dans la balise `<table>` ;
 - il faut ajouter `mat-sort-header` à chaque balise `<th>` que l'on souhaite pouvoir trier.
- Dans le TypeScript :
 - il faut ajouter `MatSortModule` à la liste des imports ;
 - il faut ajouter `@ViewChild(MatSort) sort!: MatSort;` dans la classe `CoursComponent` ;
 - Dans le `ngAfterViewInit`, il faut ajouter `this.dataSource.sort = this.sort;`