

Cours Inf3522 - Développement d'Applications N tiers

Lab 14 : Sécurisation de l'application

Ce lab expliquera comment mettre en place l'authentification sur notre interface utilisateur (frontend) lorsque nous utilisons l'authentification JSON Web Token (JWT) dans le backend. Au début, nous activerons la sécurité dans notre backend pour permettre l'authentification JWT. Ensuite, nous créerons un composant pour la fonctionnalité de connexion. Enfin, nous modifierons nos fonctionnalités CRUD pour envoyer le jeton dans l'en-tête d'autorisation de la requête vers le backend.

Sécuriser le backend

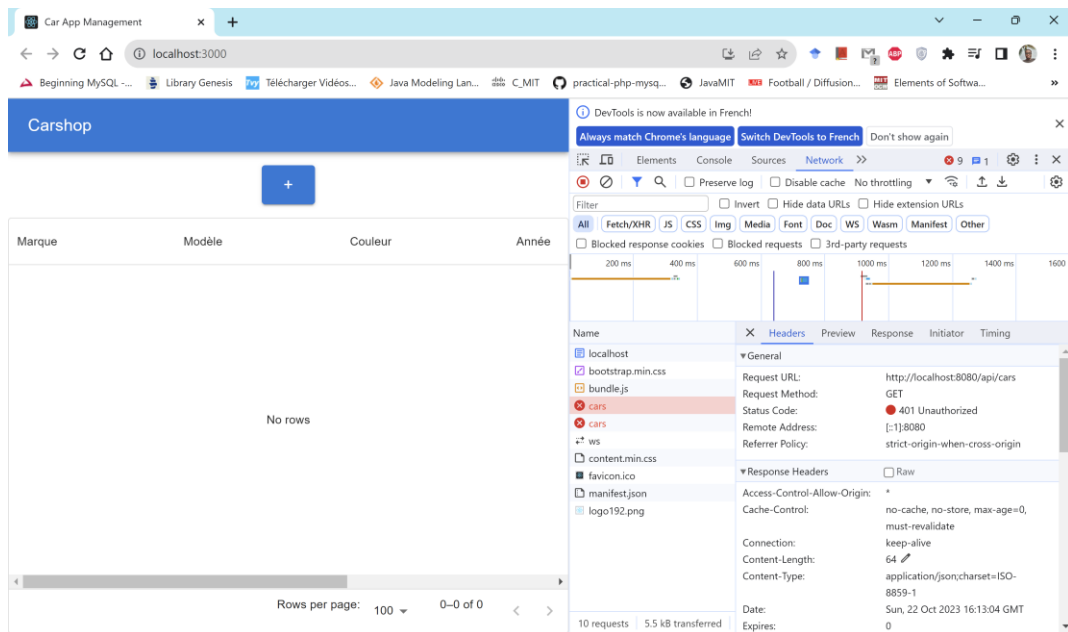
Ouvrez votre projet backend et ouvrez le fichier SecurityConfig.java dans la vue de l'éditeur. Nous avons commenté la sécurité et accordé l'accès à tous les points de terminaison. Maintenant, nous pouvons supprimer cette ligne et les commentaires de la version d'origine. À présent, la méthode configure de votre fichier SecurityConfig.java devrait ressembler à ce qui suit :

@Bean

```
SecurityFilterChain configureSecurity(HttpSecurity http) throws Exception {  
    return http  
        .csrf(csrf -> csrf.disable())  
        .cors(withDefaults())  
        .sessionManagement(management -> management  
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS))  
        .authorizeRequests(authorizeRequests ->  
            authorizeRequests  
                .requestMatchers(HttpMethod.POST, "/login").permitAll()  
                .anyRequest().authenticated())  
        .exceptionHandling().authenticationEntryPoint(exceptionHandler).and()  
        .addFilterBefore(authenticationFilter,  
            UsernamePasswordAuthenticationFilter.class)  
        .httpBasic(withDefaults())  
        .build();  
}
```

Exécutez le backend et vérifiez dans la vue de la console que l'application a démarré correctement. Exécutez le frontend en tapant la commande "npm start" dans votre terminal, et le navigateur devrait s'ouvrir à l'adresse localhost:3000.

Vous devriez maintenant constater que la page de liste et la liste de voitures sont vides. Si vous ouvrez les outils de développement et l'onglet Réseau, vous remarquerez que le statut de la réponse est "401 Unauthorized". C'est en fait ce que nous voulions, car nous n'avons pas encore effectué d'authentification par rapport à notre frontend :



Maintenant, nous sommes prêts à travailler avec l'interface utilisateur (frontend).

Sécuriser le frontend

L'authentification a été mise en œuvre dans le backend à l'aide de JWT. Dans le lab_5, nous avons créé l'authentification JWT, et tout le monde est autorisé à accéder au point de terminaison **/login** sans authentification. Sur la page de connexion de l'interface utilisateur, nous devons d'abord appeler le point de terminaison **/login** en utilisant les informations d'identification de l'utilisateur pour obtenir le jeton. Ensuite, le jeton sera inclus dans toutes les requêtes que nous envoyons au backend.

Commençons par créer un composant de connexion qui demande les informations d'identification de l'utilisateur pour obtenir un jeton depuis le backend :

Créez un nouveau fichier appelé Login.js dans le dossier des composants.

Ajoutez le code de base suivant au composant Login. Nous importons également SERVER_URL, car il est requis dans une requête de connexion :

```
JS Login.js
src > components > JS Login.js > [0] default
1 import React, { useState } from "react";
2 import { SERVER_URL } from "../constants.js";
3
4 function Login() {
5   return <div></div>;
6 }
7
8 export default Login;
```

Nous avons besoin de trois valeurs d'état pour l'authentification : deux pour les informations d'identification (nom d'utilisateur et mot de passe), et une valeur booléenne pour indiquer le statut de l'authentification. La valeur initiale de l'état de statut de l'authentification est "false" (Ligne 5-9).

```

JS Login.js X
src > components > JS Login.js > ...
1 import React, { useState } from "react";
2 import { SERVER_URL } from "../constants.js";
3
4 function Login() {
5   const [user, setUser] = useState({
6     username: "",
7     password: "",
8   });
9   const [isAuthenticated, setAuth] = useState(false);
10  return <div></div>;
11 }
12
13 export default Login;
14 |

```

Dans l'interface utilisateur, nous allons utiliser la bibliothèque de composants Material UI (MUI), comme nous l'avons fait pour le reste de l'interface utilisateur. Nous avons besoin de quelques composants `TextField` pour les informations d'identification, du composant `Button` pour appeler une fonction de connexion, et du composant `Stack` pour la mise en page. Ajoutez les imports des composants au fichier `login.js` (Ligne 4-6) :

```

JS Login.js X
src > components > JS Login.js > ...
1 import React, { useState } from "react";
2 import { SERVER_URL } from "../constants.js";
3
4 import Button from "@mui/material/Button";
5 import TextField from "@mui/material/TextField";
6 import Stack from "@mui/material/Stack";
7
8 function Login() {
9   const [user, setUser] = useState({
10     username: "",
11     password: "",
12   });
13   const [isAuthenticated, setAuth] = useState(false);
14   return <div></div>;
15 }
16
17 export default Login;
18 |

```

Ajoutez les composants importés à une interface utilisateur en les incluant dans l'instruction `return`. Nous avons besoin de deux composants `TextField` : un pour le nom d'utilisateur et un pour le mot de passe. Un composant `Button` est nécessaire pour appeler la fonction de connexion que nous allons mettre en œuvre plus tard dans cette section (Ligne 14-29) :

```

JS Login.js X
src > components > JS Login.js > @ default
5 import TextField from "@mui/material/TextField";
6 import Stack from "@mui/material/Stack";
7
8 function Login() {
9   const [user, setUser] = useState({
10     username: "",
11     password: "",
12   });
13   const [isAuthenticated, setAuth] = useState(false);
14   return (
15     <div>
16       <Stack spacing={2} alignItems="center" mt={2}>
17         <TextField name="username" label="Username" onChange={handleChange} />
18         <TextField
19           type="password"
20           name="password"
21           label="Password"
22           onChange={handleChange}
23         />
24         <Button variant="outlined" color="primary" onClick={login}>
25           Login
26         </Button>
27       </Stack>
28     </div>
29   );
30 }
31
32 export default Login;
33 |

```

Mettez en place la fonction de gestion des modifications pour les composants `TextField` afin de sauvegarder les valeurs saisies dans les états (Ligne 16-18) :

```

JS Login.js x
src > components > JS Login.js > Login
14   const [isAuthenticated, setAuth] = useState(false);
15
16   const handleChange = event => {
17     setUser({ ...user, [event.target.name]: event.target.value });
18   };
19
20   return (
21     <div>

```

Comme indiqué dans le lab_5, la connexion s'effectue en appelant le point de terminaison /login en utilisant la méthode POST et en envoyant l'objet utilisateur dans le corps de la requête. Si l'authentification réussit, nous recevons un jeton dans l'en-tête de réponse "Authorization". Nous sauvegarderons ensuite le jeton dans le stockage de session (session storage) et définirons la valeur de l'état "isAuthenticated" sur "true". Le stockage de session est similaire au stockage local (local storage), mais il est effacé à la fin d'une session de page. Lorsque la valeur de l'état "isAuthenticated" est modifiée, l'interface utilisateur est à nouveau rendue (Ligne 20-34).

```

JS Login.js x
src > components > JS Login.js > Login
16   const handleChange = event => {
17     setUser({ ...user, [event.target.name]: event.target.value });
18   };
19
20   const login = () => {
21     fetch(SERVER_URL + "login", {
22       method: "POST",
23       headers: { "Content-Type": "application/json" },
24       body: JSON.stringify(user),
25     })
26     .then(res => {
27       const jwtToken = res.headers.get("Authorization");
28       if (jwtToken !== null) {
29         sessionStorage.setItem("jwt", jwtToken);
30         setAuth(true);
31       }
32     })
33     .catch(err => console.error(err));
34   };
35
36   return (
37     <div>

```

Nous pouvons mettre en place un rendu conditionnel qui affiche le composant de connexion (Login) si l'état "isAuthenticated" est faux, ou le composant Carlist si l'état "isAuthenticated" est vrai. Tout d'abord, nous devons importer le composant Carlist dans le fichier Login.js :

import Carlist from './Carlist';

Ensuite, nous devons mettre en œuvre les modifications suivantes dans l'instruction return. Cela signifie que si "isAuthenticated" est vrai, le composant Carlist sera rendu, sinon le composant de connexion (Login) sera rendu (Lignes 38-40 et Ligne 57).

```

JS Login.js x
src > components > JS Login.js > Login
37
38   if (isAuthenticated) {
39     return <Carlist />;
40   } else {
41     return (
42       <div>
43         <Stack spacing={2} alignItems="center" mt={2}>
44           <TextField name="username" label="Username" onChange={handleChange} />
45           <TextField
46             type="password"
47             name="password"
48             label="Password"
49             onChange={handleChange}
50           />
51           <Button variant="outlined" color="primary" onClick={login}>
52             Login
53           </Button>
54         </Stack>
55       </div>
56     );
57   }
58 }
59
60 export default Login;
61

```

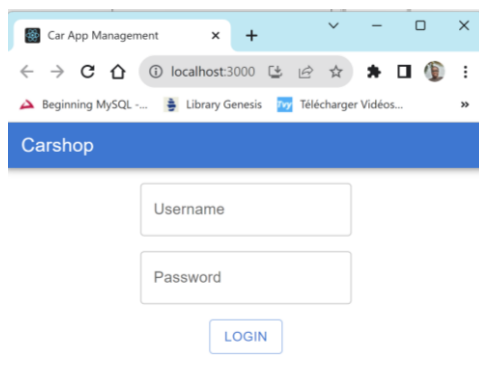
Pour afficher le formulaire de connexion, nous devons rendre le composant Login à la place du composant Carlist dans le fichier App.js (Lignes 5 et 15) :

```

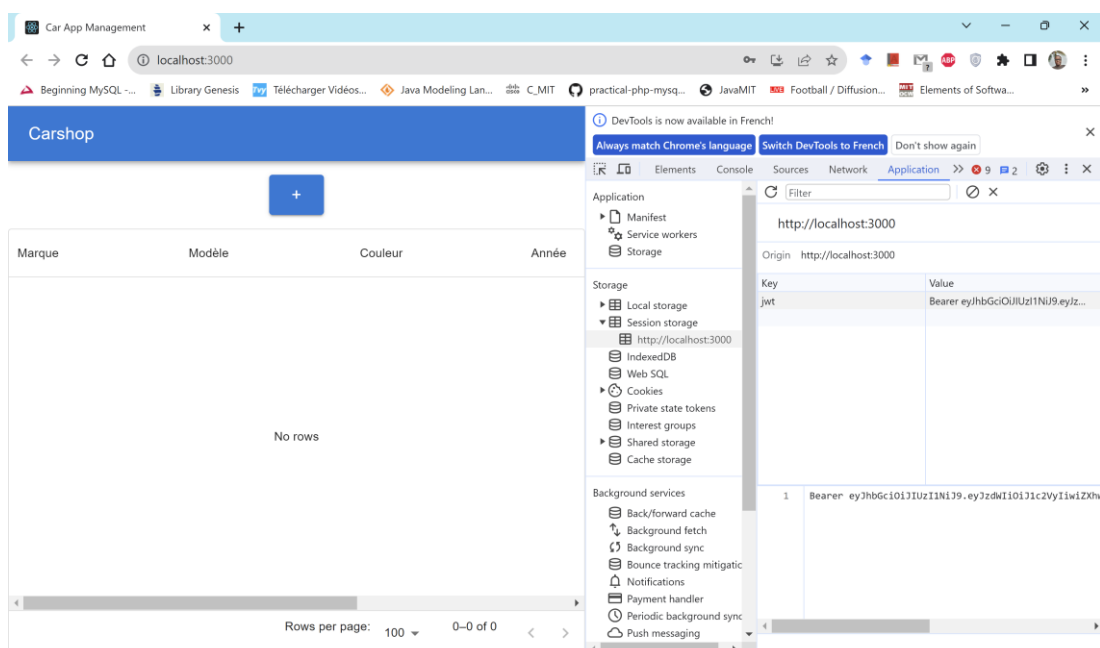
1 import "../App.css";
2 import AppBar from "@mui/material/AppBar";
3 import Toolbar from "@mui/material/Toolbar";
4 import Typography from "@mui/material/Typography";
5 import Login from "../components/Login";
6
7 function App() {
8   return (
9     <div className="App">
10       <AppBar position="static">
11         <Toolbar>
12           <Typography variant="h6">Carshop</Typography>
13         </Toolbar>
14       </AppBar>
15       <Login />
16     </div>
17   );
18 }
19 export default App;
20

```

Maintenant, lorsque votre interface utilisateur (frontend) et votre backend sont en cours d'exécution, votre interface utilisateur devrait ressembler à la capture d'écran suivante :



Si vous vous connectez en utilisant les informations d'identification utilisateur (user/user) ou administrateur (admin/admin), vous devriez voir la page de liste de voitures. Si vous ouvrez l'onglet "Application" des outils de développement, vous pourrez constater que le jeton est maintenant enregistré dans le stockage de session.



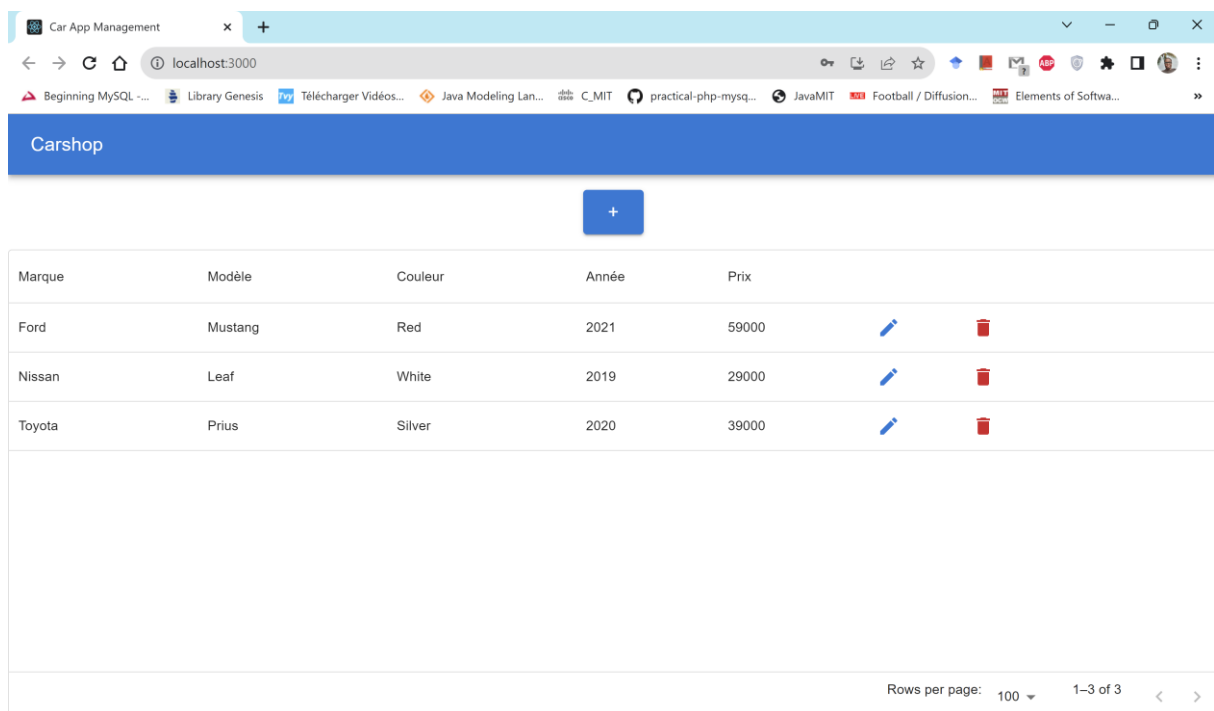
La liste de voitures est toujours vide, ce qui est correct car nous n'avons pas encore inclus le jeton dans la requête GET. Cela est nécessaire pour l'authentification JWT, que nous allons implémenter dans la prochaine étape :







Ouvrez le fichier Carlist.js dans l'éditeur de VS Code. Pour récupérer les voitures, nous devons d'abord lire le jeton depuis le stockage de session, puis ajouter l'en-tête d'autorisation avec la valeur du jeton à la requête GET. Vous pouvez voir le code source de la fonction fetchCars ici (Ligne 26-29) :

```
24
25
26 const fetchCars = () => {
27   const token = sessionStorage.getItem("jwt");
28   fetch(SERVER_URL + "api/cars", {
29     headers: { Authorization: token },
30   })
31   .then(response => response.json())
32   .then(data => setCars(data._embedded.cars))
33   .catch(err => console.error(err));
34 }
```

Vérifiez le contenu de la requête depuis les outils de développement ; vous pouvez constater qu'elle contient l'en-tête d'autorisation avec la valeur du jeton.

Toutes les autres fonctionnalités CRUD nécessitent la même modification pour fonctionner correctement.



Marque	Modèle	Couleur	Année	Prix		
Ford	Mustang	Red	2021	59000		
Nissan	Leaf	White	2019	29000		
Toyota	Prius	Silver	2020	39000		

Maintenant, toutes les fonctionnalités CRUD fonctionneront après vous être connecté à l'application.

Dans la phase finale, nous allons mettre en place un message d'erreur qui est affiché à l'utilisateur si l'authentification échoue. Nous utilisons le composant Snackbar de MUI pour afficher le message :

Ajoutez l'importation suivante dans le fichier Login.js :

```
import Snackbar from '@mui/material/Snackbar';
```

Ajoutez un nouvel état appelé "open" pour contrôler la visibilité du Snackbar :

```
const [open, setOpen] = useState(false);
```

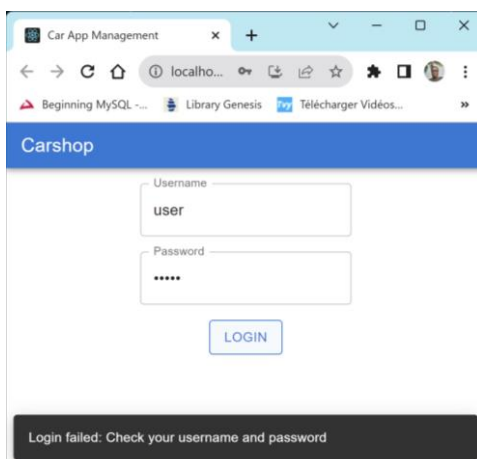
Ajoutez le composant Snackbar à l'instruction return (Ligne 56-61) :

```
JS Login.js x
src > components > JS Login.js > Login
42   return (
43     <div>
44       <Stack spacing={2} alignItems="center" mt={2}>
45         <TextField name="username" label="Username" onChange={handleChange} />
46         <TextField
47           type="password"
48           name="password"
49           label="Password"
50           onChange={handleChange}
51         />
52         <Button variant="outlined" color="primary" onClick={login}>
53           Login
54         </Button>
55       </Stack>
56       <Snackbar
57         open={open}
58         autoHideDuration={3000}
59         onClose={() => setOpen(false)}
60         message="Login failed: Check your username and password"
61       />
62     </div>
63   );
64 }
65
66
67 export default Login;
```

Ouvrez le composant Snackbar si l'authentification échoue en définissant la valeur de l'état "open" sur true (Ligne 34-36) :

```
JS Login.js x
src > components > JS Login.js > Login > login > then() callback
23   const login = () => {
24     fetch(SERVER_URL + "login", {
25       method: "POST",
26       headers: { "Content-Type": "application/json" },
27       body: JSON.stringify(user),
28     })
29     .then(res => {
30       const jwtToken = res.headers.get("Authorization");
31       if (jwtToken === null) {
32         sessionStorage.setItem("jwt", jwtToken);
33         setAuth(true);
34       } else {
35         setOpen(true);
36       }
37     })
38     .catch(err => console.error(err));
39   };
40
```

Si vous vous connectez maintenant avec de mauvaises informations d'identification, vous verrez le message suivant :



La fonction de déconnexion est beaucoup plus simple à mettre en œuvre. Fondamentalement, il vous suffit de supprimer le jeton du stockage de session et de changer la valeur de l'état "isAuthenticated" à "false", comme le montre le code source suivant (Ligne 25-28) :

A screenshot of a code editor window titled 'JS Login.js'. The breadcrumb navigation shows 'src > components > JS Login.js > Login'. The code is as follows:

```
25 const logout = () => {  
26   sessionStorage.removeItem("jwt");  
27   setAuth(false);  
28 };  
29
```

Maintenant, nous avons terminé avec notre application de gestion de voitures.

Résumé

Dans ce lab, nous avons appris comment mettre en place une fonctionnalité de connexion pour notre interface utilisateur (frontend) lorsque nous utilisons l'authentification JWT. Après une authentification réussie, nous avons utilisé le stockage de session pour sauvegarder le jeton que nous avons reçu du backend. Le jeton a ensuite été utilisé dans toutes les requêtes que nous avons envoyées au backend ; par conséquent, nous avons dû modifier nos fonctionnalités CRUD pour qu'elles fonctionnent correctement avec l'authentification.