

Security - Poradniczek

Jak to działa

Nasz backend jest "zabezpieczony" najmniej bezpiecznym sposobem jaki jest - czyli **Basic Authentication**.

Polega ona na dodaniu nagłówka `Authorization` do zapytania.

Nagłówek ma następującą wartość:

```
Basic <token>
```

gdzie:

```
<token> ::= Base64URL(<username>:<password>)
```

`Base64URL` to popularna funkcja kodująca (dwukierunkowa) - znajdźcie se w necie.

Jak widać, nazwa użytkownika i hasło leci w plain text (base64 nie szyfruje tylko koduje!) po http - czyli super bezpiecznie xd.

Backend

Security domyślnie jest włączone.

Dostęp do API jest bezstanowy, tzn. backend nie trzyma sesji. W każdym zapytaniu musi być zatem umieszczony poprawny header `Authorization` zgodnie ze specyfikacją **Basic Authentication**.

Informacje o zalogowanym użytkowniku możecie sprawdzić pod endpointem: `/api/userinfo`.

Klasa `AccountController` realizująca powyższy endpoint zawiera przykładowy kod, jak otrzymać informacje o koncie zalogowanego usera i jak na ich podstawie wyciągnąć właściwą encję użytkownika z bazy za pomocą `UserService`.

W przypadku, gdy header `Authorization` jest nieporadny albo wystąpi jakikolwiek błąd w autentykacji, backend zwraca pustą odpowiedź z error-kodem 401 (UNAUTHORIZED)

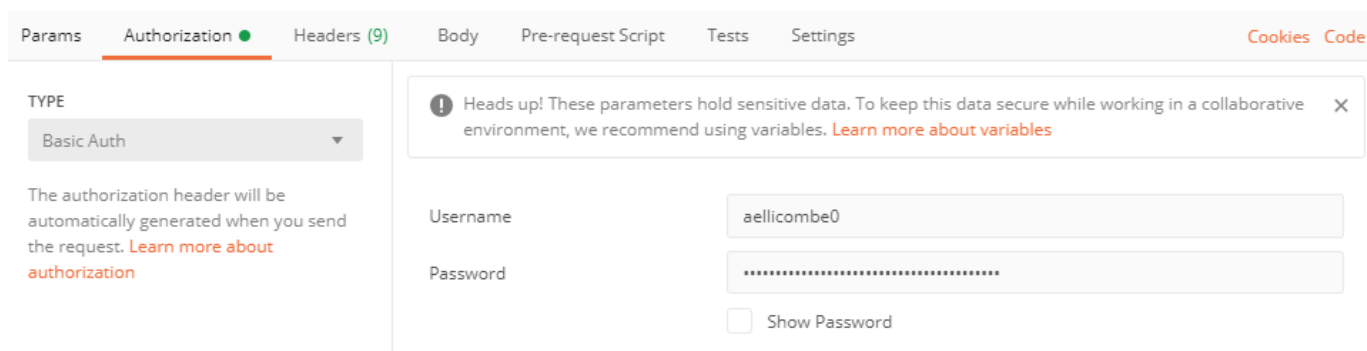
W celu wyłączenia security należy wykomentować te adnotacje w klasie `SecurityConfiguration`:

```
@Configuration
@EnableWebSecurity
```

Testowanie API w Postmanie

Postman wspiera domyślnie **Basic Authentication**, także nie musicie się bawić w żadne dziwne enkodowanie. Wystarczy ustawić typ `BasicAuth` i wkleić username i password w zakładce

`Authorization` zapytania, jak na obrazku.



Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies Code

TYPE

Basic Auth

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)

Username aellicombe0

Password

☐ Show Password

Użytkownicy znajdują się w tabeli `account` w pliku `data.sql`.

Bazodanowy hash traktujemy jako hasło - nie robimy właściwego hashowania z powodu lenistwa (trzeba by to implementować na backendie) oraz dlatego, że pan, który generował mocki do bazy wygenerował hashe, ale nie zapisał haseł, z których zostały one wygenerowane. Jawi się tutaj piękno hashowania - tych haseł nie możemy już odzyskać. Musielibyśmy wygenerować nowe hasła, a potem hashe. Pytanie po co...? Mamy już wystarczająco bezpieczny backend xd.

Frontend

Frontend musi do każdego zapytania dodawać header `Authorization`, żeby dostać odpowiedź a API. Jest już zaimplementowany interceptor, który to robi.

Jest również zaimplementowany Guard, który chroni ścieżki routera i przekierowuje na login page, jeżeli user nie jest zalogowany.

Powyższe elementy działają już globalnie i automatycznie. Jak chcecie teraz programować wasze komponenty, to **musicie mieć na uwadze 3 rzeczy**.

1. Dodawanie ścieżek do guarda

Żeby ścieżka routera była zabezpieczona, należy zarejestrować ją w pliku `app-routing.module.ts` jako dziecko ścieżki z guardem. Poniższy kod prezentuje konfigurację routera, w której wszystkie ścieżki oprócz `login-page` są zabezpieczone. Guard domyślnie przekierowuje do strony logowania, gdy nie ma zalogowanego użytkownika.

```
const routes: Routes = [
  {path: 'login-page', component: LoginPageComponent},
  {
    path: '', canActivate: [AuthGuardService], children: [
      {path: 'home', component: HomePageComponent},
      {path: 'user-details', component: UserDetailsComponent},
      {path: 'exam-list', component: LabExamListComponent},
      {path: 'exam-list/:id', component: LabExamDetailsComponent}
    ]
  }
]
```

2. Pobieranie informacji o zalogowanym użytkowniku

W tym celu używamy serwisu `UserService`. Udostępnia on w sposób **asynchroniczny** informację o użytkowniku zwracając instancję klasy `User` przez metodę `getAuthenticationEvent(): Observable<User>`.

Wynik tej metody należy subskrybować przez `subscribe()`. Kiedy zajdzie jakieś zdarzenie związane z logowaniem / wylogowaniem, serwis rozgłasza subskrybentom instancję `User` albo `null` (jeżeli nastąpiło wylogowanie albo logowanie się nie powiodło).

3. Automatyczne logowanie

Żeby łatwiej się programowało i nie trzeba było przy każdym odświeżeniu się logować, stworzona została dodatkowa konfiguracja, która na starcie loguje usera. Wystarczy w tym celu uruchamiać aplikację z konfiguracją `developer`:

```
ng serve -c developer
```

W pliku `/src/environments/auto.login.credentials.ts` znajdują się passy usera, którego chcecie logować. Możecie wziąć z bazy passy dowolnego usera (plik `data.sql` w projekcie backendowym lub podejrzeć bazkę przez pgAdmin)/