Nr. Indexu: 80791
Karolina Siwiak
I Informatyka
magisterskie

Przedmiot: Problemy bezpieczeństwa komputerowego w systemach informatycznych

Projekt - NodeJS i JSON Web Token

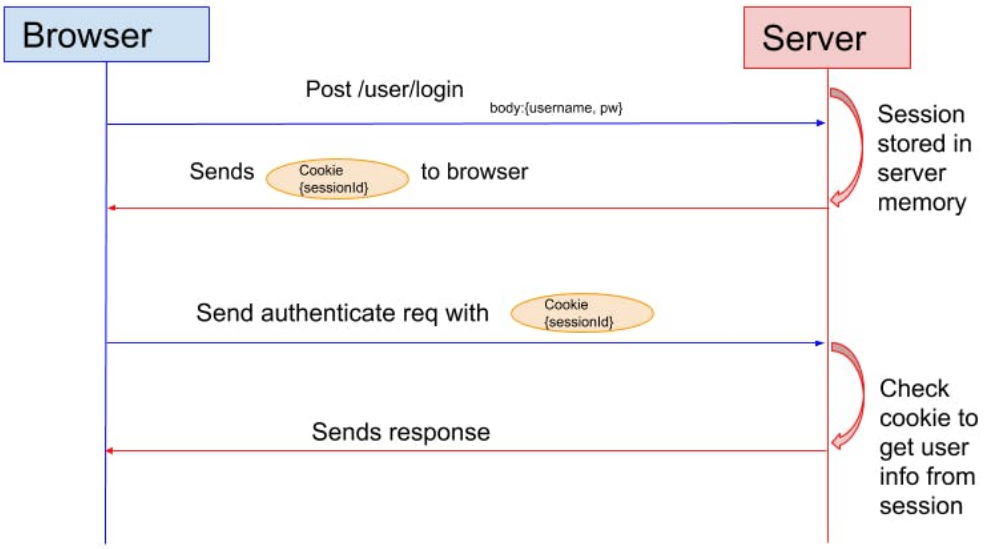
Rok akademicki 2021 - 2022
Semestr zimowy

Spis treści

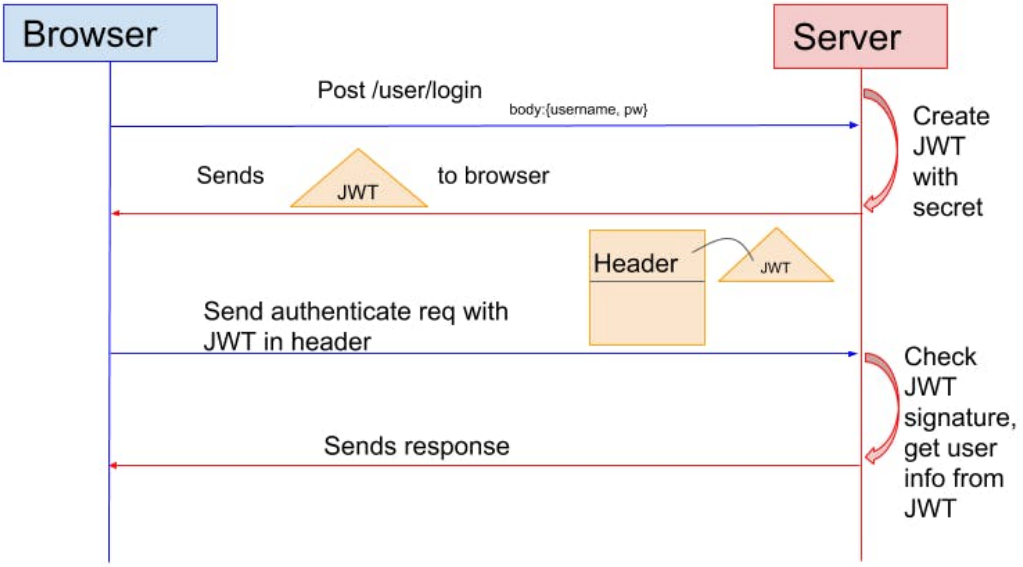
1. JWT

JWT czyli JSON Web Tokens to otwarta metoda RFC 7519, służąca do bezpiecznego reprezentowania roszczeń między dwiema stronami - klientem oraz serwerem. JWT jest wykorzystywany głównie do autoryzacji użytkowników oraz do ich uwierzytelniania.

Klasycznym sposobem autoryzacji są pliki cookie - server przechowuje w swojej pamięci dane, dotyczące sesji m.in. sessionID. Każde żądanie, wysłane do serwera zaopatrzone jest odpowiednimi plikami cookie (obraz 1), które następnie jest porównywane z zapisanymi danymi na serwerze - jeżeli cookie pokrywają się, serwer wykonuje odpowiednie żądanie i wysyła odpowiedź o powodzeniu operacji.



Obraz 1. Autoryzacja klasyczna - za pomocą plików cookie.



Obraz 2. Autoryzacja za pomocą metody JSON Web Token.

Natomiast powyższa metoda JWT używa do tego celu tokenów - obiektów JSON. Użytkownik, wysyłając żądanie do serwera w nagłówku umieszcza, przydzielony mu przez serwer token (obraz 2). Po dokonaniu pomyślnej weryfikacji tokenu, odpowiednie operacje zostaną wykonane i serwer wysyła odpowiedź do zlecającego użytkownika.

W przeciwieństwie do cookies, tokeny JSON Web zazwyczaj nie są przechowywane po stronie serwera, ale po stronie klienta (np:. przechowuje je przeglądarka) - oznacza to, iż tokeny są bezstanowe.
Pliki cookie sesji działają tylko w jednej domenie lub w jej subdomenach. W przypadku tokenów JWT nie ma takich ograniczeń, ponieważ tokeny są przechowywane w nagłówku żądania - żądanie z wymaganym tokenem może zostać wysłane nawet do innego serwera, który odczyta token i go zweryfikuje.

1.1 Tokeny JWT

Tokeny generuje się na podstawie sekretu. Poniżej znajduje się sekret dla access tokenu, który użyłam w projekcie.

d334f5841b7327cfe7d646855f7cd55179e3b151523db277de1cc04df23a766f30eca9e43690
1d5889b7fd812180bddadb372741f50ed94950d8361db06dc55c

Na podstawie powyższego sekretu został wygenerowany poniższy przykładowy access token.

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
.eyJpZCI6IjYxZTQ1MWRjN2NjZTdmODgyMzYyYmI5MiIsImVtYWlsIjoiYWRtaW5AbWFpbC5jb20iLCJwc3dkIjoiJDJiJDEwJGY2Nm83R25vc0E4czEucnJhcEZEanVUUi9zcDFUUWxjWnRtWmxzS0FuenNmSktPVHByUWplIiwiaWF0IjoxNjQyNDQ0NTI5LCJleHAiOjE2NDI0NTUzMjI9
.dbPQtOhMNLgadnZXLYgwRkP6jyMthmLzAY19iL9MVCM

Niebieski fragment tokenu to nagłówek - zawarte są w nim informacje dotyczące rodzaju tokenu - w naszym przypadku rodzaj tokenu = JWT, oraz informacja o użytym algorytmie do utworzenia ciągu znaków.
Czerwony fragment reprezentuje zawartość - w moim projekcie zawartością są dane o użytkowniku - jego id w bazie danych, adresie email oraz zahaszowane hasło.
Ostatni fragment odpowiada za potwierdzenie autentyczności tokenu. Zielony fragment jest inaczej określany jako podpis cyfrowy - metoda ocenia autentyczność tokenu - nie możemy go wykorzystać po upłynięciu czasu ważności - co zabezpiecza użytkowników przed przechwyceniem tokenu. Im krótsza data ważności, tym mniejszy zysk na ukradzionym tokenie.

1.2 Kiedy zaleca się stosować cookies, a kiedy tokeny JWT?

Tabela 1. Zalecenia dotyczące zastosowania plików cookies i tokenów JWT.

Cookies	JSON Web Token
Jeżeli potrzebujemy informacji o personalizacji użytkownika, np:. motyw.	Jeżeli serwer aplikacji musi przetworzyć wiele żądań.
Przy tworzeniu prostej aplikacji, gdzie użytkownik otrzymuje dostęp do kilku informacji z bazy danych.	Jeżeli aplikacja korzysta z wielu źródeł danych API.
	Jeżeli aplikacja posiada oddzielne serwery na przykład do autoryzacji, wykonywania odpowiednich akcji lub dostępu do danych.

2. Projekt

2.1 Koncepcja projektu

Omawianym projektem jest prosta aplikacja MENU restauracyjnego.
Z poziomu aplikacji mamy dostęp do zbioru potraw, oferowanych przez pewną restaurację.
Każda potrawa posiada poniższe informacje:

- nazwa,
- cena,
- informacje o potrawie,
- kategoria,
- stan (dostępne, niedostępne, na zamówienie).
-

Jako niezalogowany użytkownik możemy:

- przeglądać listę potraw,
- sprawdzać pozostałe szczegóły potrawy,
- zalogować lub zarejestrować się.

Jako użytkownik zalogowany mamy dostęp do takich akcji, jak:

- dodanie nowej potrawy,
- edycja istniejącej potrawy,
- usunięcie istniejącej potrawy.
-

Wszystkie informacje o potrawach są przechowywane w bazie danych MongoDB.
Aplikacja jest napisana w NodeJS w języku JavaScript.

Aplikacja posiada kontrolery RESTowe :

- plik routesApi.js - do pobierania odpowiednich list potraw lub pojedynczej potrawy,
- plik routesAuthApi.js - do obsługi logowania, rejestrowania użytkownika,
- plik routesJWTApi.js - do obsługi metod JSON Web Token.

Do metod w powyższych kontrolerach wysyłamy odpowiednie żądania HTTP (poprzez axios lub fetch) GET lub POST.

2.2 Wymagania funkcjonalne

Nazwa wymagania	Czy użytkownik zalogowany?	Opis wymagania
Wyświetl wszystkie potrawy	Nie	Zostaną wyświetlone wszystkie potrawy z bazy danych.
Wyświetl szczegóły potrawy	Nie	Zostaną wyświetlone wszystkie szczegóły pojedynczej potrawy w bazie danych.
Wyświetl wszystkie potrawy z danej kategorii	Nie	Zostaną wyświetlone wszystkie potrawy z poszczególnych kategorii.
Wyświetl wszystkie potrawy o danym statusie	Nie	Zostaną wyświetlone wszystkie potrawy o określonym statusie.
Dodaj nową potrawę	Tak	Zostanie dodana do bazy danych nowa potrawa.
Edytuj istniejącą potrawę	Tak	Istniejąca potrawa zostanie zmodyfikowana w bazie danych.
Usuń istniejącą potrawę	Tak	Istniejąca potrawa zostanie usunięta z baziy danych.
Zarejestruj nowego użytkownika	Nie	Do bazy danych zostanie dodany nowy użytkownik.
Uwierzytelnij użytkownika (logowanie)	Nie	Po odpowiednim wypełnieniu formularza logowania, dane z tego formularza zostaną sprawdzone i jeżeli istnieje taki użytkownik, zostanie on uwierzytelniony i uzyska prawa do wykonania akcji, zarezerwowanych dla użytkowników uwierzytelnionych.

2.3 Wymagania niefunkcjonalne

Użytkownik rejestruje się do systemu za pomocą unikalnego adresu email oraz hasła (długość hasła nie mniejsza niż 5 znaków).
Logowanie do systemu odbywa się za pomocą adresu email i hasła.
Interfejsem systemu jest strona WWW.
System jest dostępny w języku polskim.

2.4 Użyte technologie w projekcie

Baza danych	MongoDB
Pobieranie danych z bazy danych	NodeJS API
Interfejs	HTML z elementami EJS
Protokół komunikacji	HTTP
Hashowanie haseł	Biblioteka bcrypt
Uwierzytelnianie użytkowników	JSON Web Token
Żądania	
GET	● / ● /przystawki ● /zupy ● /salaty ● /makarony ● /miesa ● /owoceMorza ● /desery ● /dlaDzieci ● /dostepne ● /niedostepne ● /naZamowienie ● /login ● /logOut ● /register ● /show/:id ● /delete/:id ● /edit/:id
POST	● /potrawy ● /loginTo ● /registerTo ● /edit/:id

2.5 Struktura bazy danych

Nazwa tabeli	Atrybuty tabeli
potrawy	• nazwa • cena • info • kategoria • status
users	• email • pswd • refreshToken

2.6 Zastosowanie JSON Web Token w projekcie

W projekcie zostały wykorzystane 2 rodzaje tokenów:

- refresh token - token o ważności 3 godzin (ważny dłużej niż access token),
- access token - token o ważności 15 sekund.

```
1. app.post("/generateAccessTokenApi", (req, resp) => {
2.         var id = req.body._id;
3.         var mail = req.body.email;
4.         var password = req.body.pswd;
5.         console.log('/generateAccessTokenApi');
6.
7.         var user = {
8.             id: id,
9.             email: mail,
10.             pswd: password
11.         };
12.
13.         console.log('USER: ' + user.id);
14.
```

```
15.         const accessToken = generateAccessToken(user, '15s');
16.         const refreshToken = generateRefreshToken(user, '3h');
17.
18.         saveRefreshTokenApi(user.id,refreshToken);
19.
20.         resp.json({ accessToken: accessToken });
21.         resp.end();
22.     });
```

Kod 1. Fragment kodu z pliku /web-site/routeJWTApi.js

Generowanie access tokenu i refresh tokenu odbywa się za pomocą ACCESS_TOKEN_SECRET i REFRESH_TOKEN_SECRET (kod 2, linie 2 i 6) - są to specjalnie wygenerowane ciągi znaków, które są stałe dla metody JWT - refresh token nie może być modyfikowany podczas, gdy użytkownik jest zalogowany i posiada ważny refresh token - w takim wypadku weryfikacja aktualnego refresh tokenu zakończy się niepowodzeniem, a w najgorszym przypadku użytkownik będzie musiał uwierzytelnić się ponownie. ACCESS_TOKEN_SECRET może zostać zmodyfikowany podczas działania aplikacji, ale tylko dlatego, że przypisałam ważność tokenu na bardzo krótki czas i może zostać wygenerowany nowy access token dzięki weryfikacji refresh tokenu. Wartości ACCESS_TOKEN_SECRET i REFRESH_TOKEN_SECRET zostały zapisane w pliku .env.

```
1. function generateAccessToken(user, time) {
2.     return jwt.sign(user, process.env.ACCESS_TOKEN_SECRET, { expiresIn: time });
3. }
4.
5. function generateRefreshToken(user, time) {
6.     return jwt.sign(user, process.env.REFRESH_TOKEN_SECRET, { expiresIn: time });
7. }
8.
9. function saveRefreshTokenApi(id, refreshToken) {
10.     MongoClient.connect(url, function (err, db) {
```

```
11.     if (err) throw err;
12.
13.     var ObjectId = require('mongodb').ObjectID;
14.
15.     var dbo = db.db("mydb");
16.
17.     dbo.collection("users").updateMany({_id: ObjectId(id)},
18.     {
19.         $set: { refreshToken: refreshToken }
20.     }, (err, result) => {
21.         if (err) console.log(err);
22.
23.         console.log(result);
24.     });
25.
```

Kod 2. Fragment kodu z pliku /web-site/routeJWTApi.js

Podczas rejestrowania użytkownika do bazy danych zostaje wygenerowany refresh token oraz access token. Do bazy danych trafia refresh token (patrz kod 1, linia 18 oraz kod 2, linie 9-24), na którego podstawie będziemy tworzyć nowe access tokeny użytkownika (kod 1, linia 15 oraz kod 2, linie 1-3), jeżeli stracą one ważność, a użytkownik będzie nadal zalogowany. Po pomyślnej rejestracji zostaje wygenerowany i zapisany do zmiennej sesyjnej access token (kod 3, linie 7-12).

```
1. fetch('http://localhost:8080/generateAccessTokenApi', {
2.       method: 'POST',
3.       body: JSON.stringify(user),
4.       headers: { 'Content-Type': 'application/json' }
5.      })
6.       .then(res => res.json())
7.       .then(response => {
8.        console.log('ODP TOKEN: ' + response.accessToken);
9.
10.         let ac = response.accessToken;
11.        req.session.accessToken = ac;
12.        req.session.save();
13.       }).catch(error => {
14.        console.log(error);
15.       });
```

Kod 3. Fragment kodu z pliku /web-site/myServer.js

Podczas logowania użytkownika do bazy danych zostaje usunięty dotychczasowy refresh token i dodany nowy refresh token dla tego użytkownika. Po pomyślnym zalogowaniu zostaje wygenerowany i zapisany do zmiennej sesyjnej access token.

Dzięki zastosowaniu access tokenu, możliwe jest uwierzytelnienie danego użytkownika - dostęp do akcji takich, jak:

- dodanie nowej potrawy,
- edycja istniejącej potrawy,
- usunięcie istniejącej potrawy.

Mimo, iż access token jest dostępny w zmiennej sesyjnej - jest on aktualny przez bardzo krótki czas. Minimalizujemy w ten sposób ryzyko podszycia się intruza pod innego użytkownika przez przechwycony token.

```
1. app.post("/refreshAccessTokenApi", async (req, resp) => {
2.         const authHeader = req.headers['authorization'];
3.
4.         console.log("----------------------refreshAccessTokenApi");
5.
6.         const token = authHeader && authHeader.split(' ')[1];
7.         if (token == null) { console.log("WESZLO PRZEZ ERROR"); return
resp.json('ERROR'); }
8.
9.         var id0;
10.          var email0;
11.
12.          jwt.verify(token, process.env.REFRESH_TOKEN_SECRET, (err, user) => {
13.              if (err) {
14.                  resp.json({ data: "ERROR" });
15.                  resp.end();
16.              } else {
17.                  id0 = user.id;
18.                  email0 = user.email;
19.                  password0 = user.password
20.
21.                  let zmienna = {
22.                      id: id0,
23.                      email: email0,
24.                      pswd: password0
25.                  }
26.
```

```
27.             const accessToken = generateAccessToken(zmienna, '15s');
28.
29.             resp.json({ accessToken: accessToken });
30.             resp.end();
31.         }
32.
33.     });
34. });
```

Kod 4. Fragment kodu z pliku /web-site/routeJWTApi.js

Jeżeli access token stracił ważność, a użytkownik wciąż jest zalogowany w aplikacji należy odświeżyć access token za pomocą refresh tokenu, który jest zapisany w bazie danych.

Refresh token zostaje przesłany do weryfikacji w nagłówku żądania (kod 4, linia 2). Zostaje on następnie “enkapsulowany” z fragmentu nagłówka (kod 4, linia 6), by później został zweryfikowany poprzez funkcję jwt.verify (kod 4, linia 12). Po weryfikacji refresh tokenu (kod 4, linia 12) zostaje wygenerowany nowy access token i przyporządkowany do tego użytkownika (kod 4, linia 27) - access token znowu zostaje zapisany do zmiennej sesyjnej i zostaje wykonana akcja, wywołana przez użytkownika przed sprawdzeniem tokenu.

Źródła:

https://hashnode.com/post/cookie-vs-token-authentication-ckpws12e1039ia9s13slka08y
https://blog.i-systems.pl/json-web-tokens-jwt/
https://www.youtube.com/watch?v=mbsmsi7l3r4&t=546s
https://www.youtube.com/watch?v=7nafaH9SddU&t=1114s
https://www.youtube.com/watch?v=2jqok-WgelI
https://www.youtube.com/watch?v=7Q17ubqLfaM
https://www.youtube.com/watch?v=T0k-3Ze4NLo&t=2272s