Laboratorul 05: Autentificare si Autorizare cu JWT

La laboratorul trecut am reusit sa dezvoltam un REST Api care se conecteaza la o baza de date si functioneaza ca o biblioteca online. Din acel Api lipsesc, insa, autentificarea si autorizarea.

Autentificare

Autentificarea se refera la procesul prin care o entitate isi dovedeste identitatea in contextul unui sistem. Autentificarea se poate realiza prin multe moduri: introducere de pin, desenare de model, username si parola, date biometrice, etc...

In cazul nostru, vom defini identitatea utilizatorului prin doua campuri **username** si **password**.

In baza de date, parola este stocata criptat. Asadar, atunci cand inregistrati un utilizator, parola acestuia trebuie criptata. Atunci cand se autentifica, parola trebuie verificata cu cea din baza de date. In scheletul de laborator aveti cele doua functii pentru lucru cu parole

Autorizare

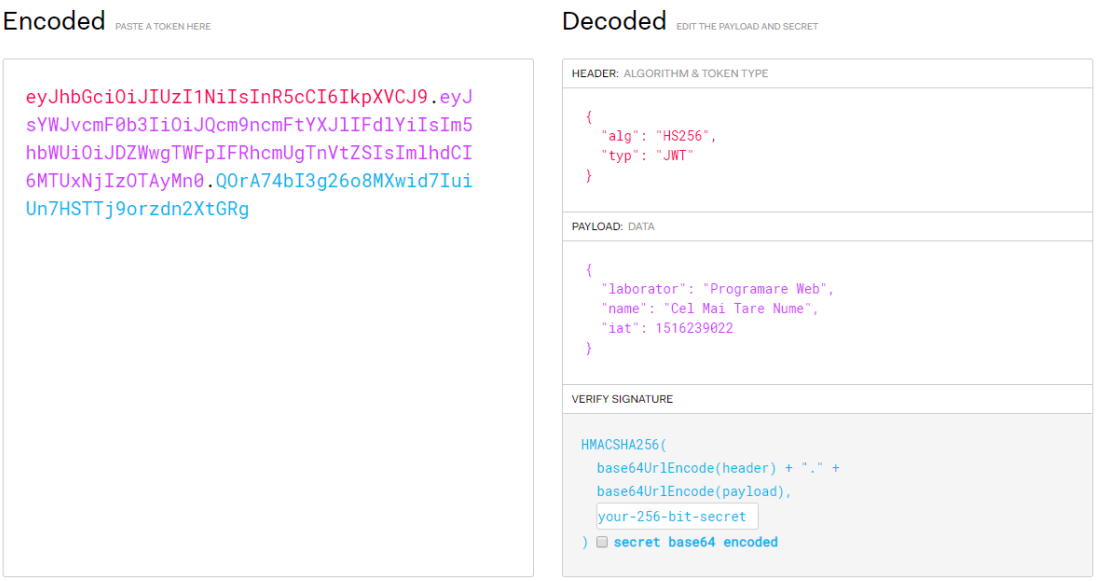
Autorizarea se refera la procesul prin care o entitate poate accesa o resursa pe baza rolului sau alocat in sistem. Autorizarea are sens atunci cand anumite resurse din sistem sunt accesibile in functie de permisiuni.

In cazul nostru, vom defini trei roluri:

- **administratorul** (ADMIN), care va putea face orice operatie de tip CRUD
- **managerul** (MANAGER), care va putea face operatii CRUD doar pe resursele bibliotecii, cele implementate in laboratorul trecut
- **utilizatorul** (USER), care va putea face doar R (read)pe resursele bibliotecii

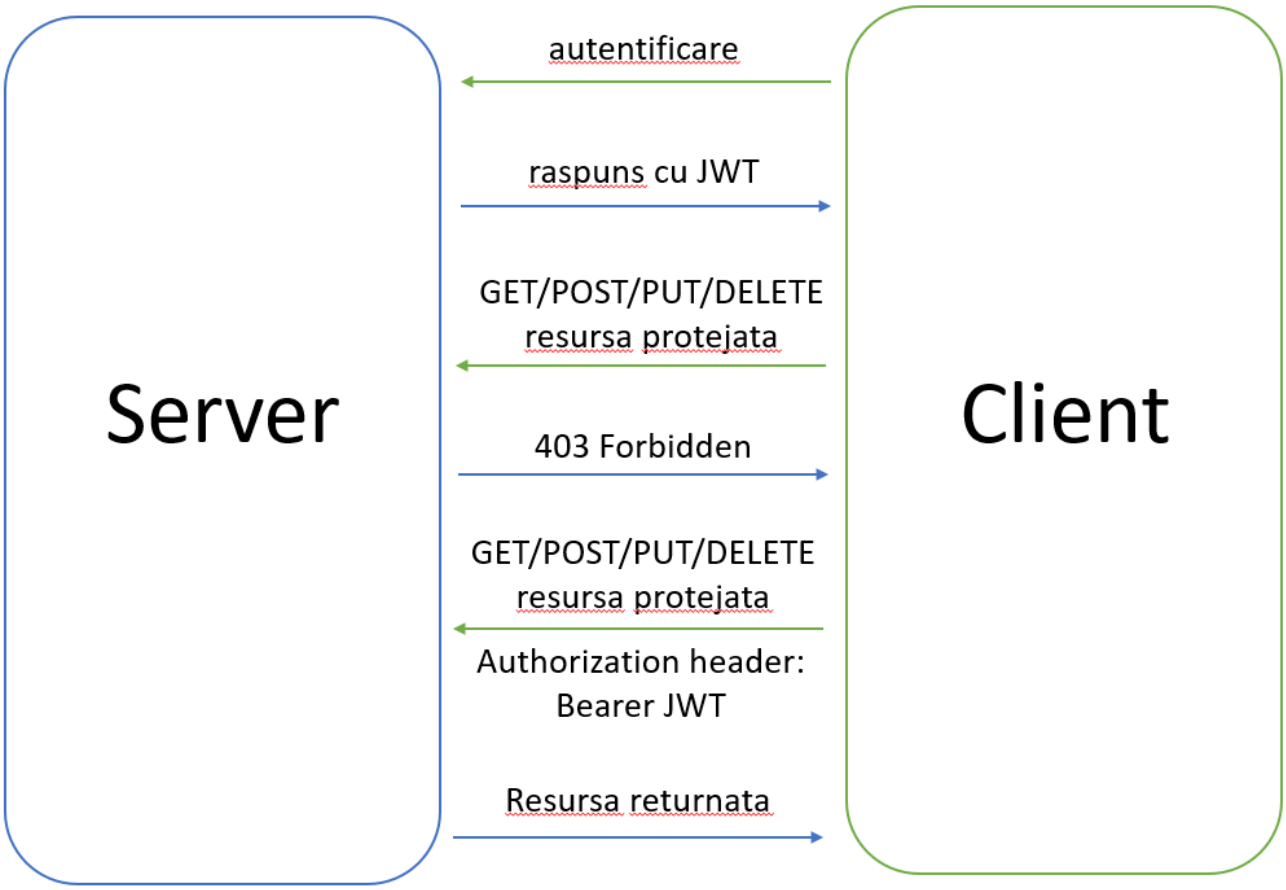
JWT Token

Tokenurile Web JWT [https://jwt.io/] sunt un standard deschis, descris in RFC 7519. Acestea reprezinta o incapsulare a unui set de informatii, care sunt codificate si semnate de catre server si trimise la client pentru utilizari ulterioare.



Avantajul major al tokenului JWT consta in verificarea integritatii: Informatiile codificate sunt semnate (hashuite), iar semnatura este atasata la finalul tokenului. Atunci cand serverul primeste tokenul, acesta re-semneaza informatiile codificate si compara rezultatul cu hashul atasat. Daca cele doua semnaturi difera, tokenul este considerat invalid. In plus, **spre deosebire** de cookies, care se trimit implicit la fiecare request, clientul **trebuie** sa stie cand sa trimita token-ul JWT catre server, in **header**. Token-ul se trimite in header-ul cererilor care acceseaza **resurse protejate**.

De obicei, token-urile JWT se folosesc in contextul unui framework de securitate web complex, precum OAuth2 [https://oauth.net/2/] si OpenID Connect [https://en.wikipedia.org/wiki/OpenID_Connect], insa, pentru simplitate, ne vom rezuma la urmatorul flow:



Informatiile din token **nu** sunt criptate!

Intotdeauna, cand lucrati cu JWT si resurse protejate, tokenul trebuie inclus in **Authorization header** in cererea catre backend

In token vom include urmatoarele informatii:

- **userId** - pentru a sti despre ce utilizator e vorba
- **userRole** - pentru a sti ce rol are utilizatorul care a efectuat cererea

Daca nu includeti tokenul in **Authorization header**, serverul trebuie sa intoarca **403 Forbidden**

Daca includeti tokenul in **Authorization header**, dar rolul din token nu are permisiuni pentru cererea respectiva, serverul trebuie sa intoarca **401 Unauthorized**

JWT Token in NodeJS si Express

Dupa cum v-ati obisnuit la ultimele 3 laboratoare, Express se bazeaza pe **middlewares**. Pentru a lucra cu JWT, este nevoie de 3 lucruri:

- Logica de generare token JWT la autentificare
- **Middleware** de **extragere** token din **headerul** cererii, de **validare** token si de **returnare** informatie decriptata
- **Middleware** de **verificare** si **validare** rol

Middlewares vor fi atasate rutelor care trebuie protejate. Exemplu:

```
router.get('/rutaProtejataPentruAdmin', authorizeAndExtractTokenAsync, authorizeRoles('ADMIN'), async (req, res, next) => {
  // do smth
});

router.get('/rutaProtejataPentruAdminSiUser', authorizeAndExtractTokenAsync, authorizeRoles('ADMIN', 'USER'), async (req, res, next) => {
  // do smth
});
```

Ramane la latitudinea voastra daca vreti sa separati logica de extragere, validare token si de verificare rol in 2 middlewares sau nu. Ele, bineinteles, pot fi comprimate si in cadrul unui singur middleware.

Configuratia Laboratorului

Laboratorul acesta vine in completarea laboratorului trecut. Peste ce am facut data trecuta va trebui sa adaugam sistemul de autentificare si autorizare.

Bineinteles, baza de date este usor modificata, cu 2 tabele noi adaugate:

Biblioteci utilizate

In NodeJS, cele mai folosite 2 biblioteci pentru lucrul cu JWT sunt jsonwebtoken [https://www.npmjs.com/package/jsonwebtoken] si Passport JWT [http://www.passportjs.org/packages/passport-jwt/].

Noi va recomandam sa utilizati biblioteca de baza, **jsonwebtoken**, intrucat este simplu de utilizat si nu ofera nicio functionalitate out-of-the-box. Passport este considerat framework, deci implicit vine cu un bagaj de documentatie considerabil.

Pentru partea de criptare de parole, am folosit biblioteca Bcryptjs [https://www.npmjs.com/package/bcryptjs], o implementare scrisa in JS pur a bibliotecii consacrate Bcrypt

Scheletul de Laborator

Scheletul de laborator [https://gitlab.com/tehnologiiweb/lab5] vine in completarea scheletului laboratorului trecut.

Am adaugat o sectiune noua de proiect, **WebCore** care inglobeaza functionalitatile ce tin de de securitate si user management. In plus, am adaugat controllere, repository-uri, filtre si modele noi pentru partea de utilizatori si roluri.

Mediul de Lucru

Mediul de lucru ramane identic, pe baza de **Docker** si **Docker Compose**.

Datele folosite la laboratorul trecut nu vor mai exista, intrucat am modificat volumul de date folosit in Docker Compose pentru a nu fi nevoie sa stergeti voi de mana datele.

Ca sa fiti siguri ca nu aveti probleme cu datele din laboratorul trecut, executati comanda **docker system prune –volumes**

Exercitii

Acomodare

Parcurgeti scheletul de laborator si puneti intrebari asistentului.

Timp: 20 minute

Intrebarile ce tin de infrastructura prezentata la laborator care vor fi puse dupa acest termen, nu vor fi luate in calcul

Rezolvare efectiva

Trebuie sa implementati logica de **Inregistrare**, **Autentificare** si **Autorizare** pentru endpointurile create in laboratorul trecut. Pentru scheletul curent, veti avea TODO-uri puse in cod. Pentru ce ati rezolvat data trecuta, trebuie sa aplicati autorizarea in functie de roluri, conform enuntului:

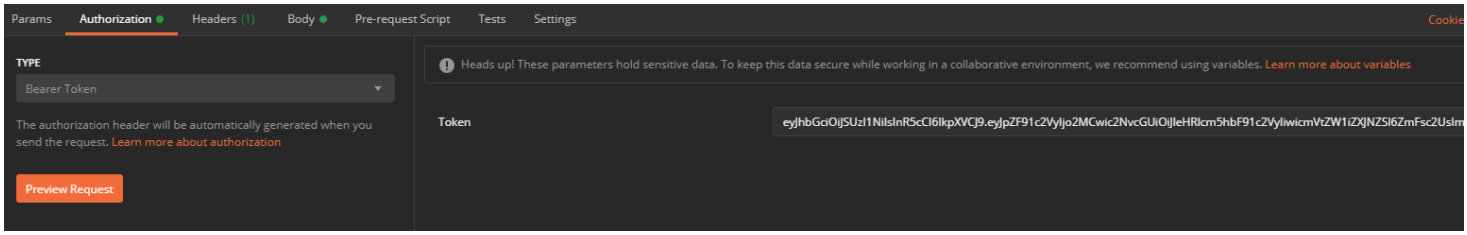
1. administrator - are voie sa faca orice
2. manager - are voie sa faca orice **doar** pe resursele din laboratorul trecut (cele care tin de carti, autori si edituri)
3. user - are voie doar sa execute **read** pe resursele din laboratorul trecut

Aveti deja un administrator creat, odata cu pornirea bazei de date din docker, cu credentialele admin admin

Timp: 2 zile

Testare cu Postman

Pentru a testa, este nevoie ca, pentru rutele protejate, sa adaugati token-ul JWT in cerere, in sectiunea Authorization → Bearer Token



pw/laboratoare/05.txt · Last modified: 2021/04/03 21:07 by alexandru.hogea