JSON Web Token

Dan je primjer autentikacije koristeći jwt token. Više o tome što je to točno možete pročitati na sljedećem linku https://jwt.io/.

Najčešća upotreba JWT-a

JWT se najčešće koristi za **autentikaciju**. Jednom kada je user ulogiran, JSON web token će se kroz response vratiti klijentu. Zaštićene rute će zahtjevati JWT. Kada se napravi zahtjev na zaštićene rute, idealno bi bilo da user preglednik pošalje JWT na server. To se obično radi tako da se u zahtjev umetne '**Authorization'** header koristeći **Bearer** schemu definiranu u Oauth 2.0: https://swagger.io/docs/specification/authentication/oauth2/ . Sadržaj headera bi trebao izgledati: Authorization: Bearer <token>.

Struktura JSON Web token-a

Sastoji se od tri dijela koja su odvojena točkom:

Header (zaglavlje) -prvi dio tokena se obično sastoji od dva dijela. U njemu se definira tip tokena (koji je najčešće JWT) i algoritam koji će se primjeniti za izradu potpisa.

Payload - drugi dio jwt-a koji sadrži podatke o korisniku.

Signature (potpis) - za kreiranje potpisanog dijela potrebno je uzeti enkodirani header i enkodirani payload i SECRET te na to primjeniti algoritam koji je specificiran u zaglavlju. Potpis se koristi kako bi se provjerilo je li poruka prilikom prijenosa promijenjena. U slučaju tokena koji su potpisani sa privatnim ključem, može se napraviti provjera je li korisnik onaj za koga se izdaje. Rezultat su tri stringa koja su kodirana i razdvojena točkom.

Opis primjera sa predavanja:

Klijentska aplikacija

Izmijenjen je primjer klijentske aplikacije sa prethodnih predavanja. Mockani podaci su zamijenjeni sa pravim podacima koji dolaze sa poslužiteljske express.js aplikacije. Budući da je ruta „api/genres" na poslužiteljskoj strani zaštićena (pročitati niže), samo se prijavljenim korisnicima select kontrola popuni žanrovima. Prijava se odvija na putanji /login, a registracija na putanji /register. Prijavom se šalje zahtjev na poslužiteljsku (express.js) aplikaciju. Poslužiteljska aplikacija provjerava nalazi li se korisnik u bazi. Ukoliko se nalazi, i lozinka je točna, u tijelo odgovora se šalje accessToken (jwt token). Klijentska aplikacija dobije odgovor te u localStorage https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage ubacuje token pod ključem „token". Klijentska aplikacija nadalje vodi računa o tome da umetne token u header prilikom zahtjevanja žanrova.

Opis primjera po datotekama:

App.js
U App.js komponentu dodane su još tri komponente: Register, Login i Logout.

Register.js
Dodana je forma za registraciju. Cijeli JSX je jasan sam po sebi. Najbitniji dio je realizacija funkcije handleRegister(). Ukoliko se password i repeat password inputi poklapaju (front end validacija), napravit će se post zahtjev (post request) na express.js aplikaciju (i to na putanju za registraciju). Slanje zahtjeva se radi pozivom fetch funkcije (https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API), kojoj je prvi argument url na koji se request šalje, a drugi argument je objekt u kojem se definira tip request-a, njegovo tijelo i headeri. Ukoliko se drugi argument ne navede, defaultno ponašanje je slanje get zahtjeva. Fetch funkcija vraća promise. Ako se promise uspješno resolve-a, korisnika se preusmjeri na /login. Ukoliko se promise reject-a, na konzolu se ispiše poruka o greški.

Login.js
Najbitniji dio je realizacija funkcije handleLogin() koja će se pozvati kada se forma submit-a. Napravi se zahtjev na server (slično kao u komponenti Register.js). Ukoliko se response reject-a, na konzolu se ispiše poruka o greški. Ako je promise resolve-an, provjerava se sadrži li u sebi ključ accessToken (vidi niže **ruta /login**). Ukoliko postoji, dodat će ga se u localstorage pod ključem token. Nadalje, pozivom funkcije navigate (iz paketa @reach/router) korisnika se preusmjeri na početnu stranicu. Local storage je došao sa HTML5, te je mehanizam sličan cookie-u. Sve stranice koje dolaze sa iste domene dijele isti local storage. Podaci ostaju pohranjeni u local storage-u kao parovi ključ/vrijednost. Osnovna razlika između local storage-a i cookie-a je u tome što se podaci iz local storage-a ne šalju automatski na server sa svakim request-om. Još jedna razlika je ta, što podaci iz local storage-a ne expire-aju, nego se brišu ručno putem JavaScript koda. Više pročitati na https://developer.mozilla.org/enUS/docs/Web/API/Window/localStorage.

Logout.js
Ukoliko korisnik dodje na ovu rutu, iz local storage-a se izbriše vrijednost pod ključem “token”. Kako aplikacija više ne posjeduje token sa kojim se verificira prema serverskoj aplikaciji, korisnik je odjavljen.

SearchParams.js
U ovoj komponenti su napravljene izmjene. Pozivi mock-anog API-a su zamjenjeni pozivima na pravi API naše poslužiteljske express.js aplikacije. Nadalje zahtjev za žanrovima je proširen umetanjem Authorization headera sa jwt tokenom.

Serverska aplikacija:
Proširena je serverska aplikacija sa prethodnog predavanja. Prvo je dodan još jedan model (UserModel). U dokumentaciji od paketa “mongoose” se može proširiti znanje o definiranju Schema-e za modele. Zatim su dodane još dvije rute (login i register). Cilj predavanja je dobiti

klijentsku aplikaciju koja konzumira podatke sa serverske aplikacije. Autentikacija je realizirana preko JSON Web token-a. Kako bi se osigurao uspješan povrat podataka sa servera, također je potrebno pobrinuti se za CORS (Cross-Origin Resource Sharing). Serverska aplikacija po default-u šalje odgovore klijentskoj aplikaciji koja ima isti "origin" (odnosno koja dolazi sa iste domene). Budući da se u našem primjeru poslužiteljska aplikacija poslužuje na portu 4000, a klijentska na portu 1234, to su dvije odvojene aplikacije, koje se nalaze na dvije odvojene domene. Zbog toga je na serverskoj strani potrebno uključiti paket "cors". Koristeći paket "cors" u serverskoj aplikaciji (app.use(cors())) se "white list-aju" domene koje će server poslužiti. U našem primjeru su "white-listane" sve domene, što nije dobra praksa (zbog toga što je fokus predavanja stavljen na JWT token). Detaljnije o CORS-u pročitati na https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS.

Opis primjera po datotekama:

app.js
Zbog modularnosti, u app.js su dodana tri odvojena router-a.

```
const movieRouter = express.Router();
const genreRouter = express.Router();
const userRouter = express.Router();
```

ruta /register
Na rutu register je stavljen middleware koji provjerava je li user već postoji u bazi, i ako ne postoji dodaje ga se. U protivnom se vraća error response.

ruta /login
Na rutu login je dodan middleware, kojim se provjerava postoji li user u bazi. Baza vraća kolekciju User modela koji imaju traženi email. Kolekcija može biti prazna ako niti jedan User ne postoji sa tim emailom. Ukoliko kolekcija nije prazna, korisnik postoji i uzima se prvi (ujedno i jedini) iz kolekcije. Ukoliko user postoji, provjerava se točnost lozinke, te se generira token koji se umetne u odgovor pod ključem "accessToken". Token se generira i potpisuje koristeći pomoćnu funkciju signJwt() koja je izdvojena u odvojenu datoteku ./jwt.js (opisana je u nastavku). Kasnije će se u klijentskoj aplikaciji implementirati spremanje JWT-a. Na klijentskoj strani aplikacije će se voditi računa i o tome da se token umetne u header requesta, kako bi se uspješno mogla obaviti autentikacija.

Serversko generiranje jwt token-a.
U express aplikaciju dodana je datoteka .env i instaliran paket *dotenv*. U njoj se nalazi generirani "SECRET" koji će se iskoristiti za potpisivanje jwt tokena. SECRET se generira jednom, koristeći paket 'crypto'. SECRET se moze generirati u node konzoli te potom kopirati u .env. Unutar app.js datoeke je require-an dotenv, te pozvana funkcija config(). Rezultat toga je učitavanje svih varijabli iz datoteke .env u process.env varijablu. Potreba za generiranjem novog SECRET-a nije česta. Nekad se koristi u svrhu odjave svih usera – jer se na taj način svi postojeći jwt tokeni više ne mogu dekodirati (stari secret više nije valjan).

jwt.js
Implementirana je funkcija signJwt() koja kao argument prima user_id. Koristeći funkciju sign() koja dolazi iz paketa ‘jsonwebtoken’ generira se token. Token se dobije tako što se u funkciju sa SECRET-om pošalje i objekt sa ključem sub. U objekt se osim ključa “sub” mogu poslati i ostali podaci o korisniku.
Implementirana je i funkcija verifyJwt() koja u request-u traži zaglavlje ‘authorization’. Ukoliko ga ne pronađe ili ne uspije pronaći token nakon ljučne riječi Bearer, vraća odgovor “401”. Ukoliko je traženi token tu, iskoristi se funkcija verify() koja dolazi iz paketa ‘jsonwebtoken’. U funkciju se šalju token, SECRET, i callback funkcija. U callback funkiju će se prosljediti eventualni error i otpakirani token, tj. objekt s podacima. Ukoliko dođe do greške, ili u objektu (payload) nedostaje ključ sub, vraća se 401. Budući da je funkcija verifyJwt() middleware, ukoliko se token uspješno verificira, poziva se next() funkcija kako bi se prešlo na sljedeći middleware koji je vezan za tu rutu.

ruta /genres
Na rutu /genres dodan je middleware verifyJwt koji je opisan iznad. Ukoliko je server pronašao JWT token u zaglavlju zahtjeva, sljedeći middleware u nizu preuzima obradu request-a. U našem primjeru dozvojeno je dohvaćanje svih žanrova iz baze podataka ukoliko je token valjan.