

Spring Security

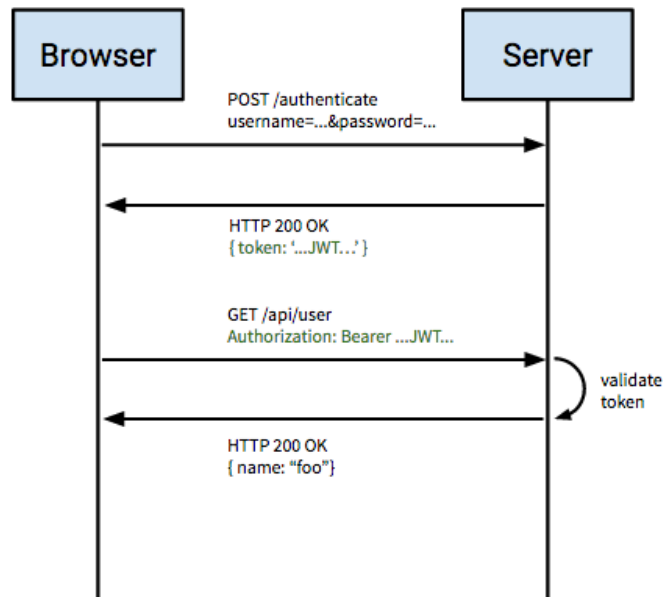
Tema ovih vežbi je obezbeđivanje *Spring Boot* aplikacije, to jest autentifikacija korisnika i kontrola pristupa različitim servisima/funkcionalnostima koje aplikacija nudi. Spring nudi već gotov *framework* (**Spring Security**) koji olakšava autentifikaciju i autorizaciju korisnika, ali takođe i nudi funkcionalnosti za zaštitu od *cross origin resource sharing*-a, *cross site request forgery*-a i mnogih drugih bezbednosnih problema koji se javljaju pri razvijanju veb aplikacija. *Spring Security* se lako integriše sa *Spring Boot* veb aplikacijama i jednostavno ga uključujemo u projekat pomoću *Maven*-a.

Proći ćemo dva različita načina za autentifikaciju korisnika: Korišćenjem *JSON web token*-a (**JWT**) i upotrebom otvorenog protokola **OAuth 2.0**.

Autentifikacija i Autorizacija uz JWT

JWT je niz karaktera pomoću kog mogu da se razmenjuju informacije o pravima (*claims*) koje određen korisnik poseduje. Na slici 1 je prikazan tok informacije između veb pretraživača i *back-end* aplikacije koja koristi JWT za autentifikaciju i autorizaciju. Korisnik šalje zahtev sa svojim kredencijalima, i ako su isti ispravni, server generiše JWT (dugačak niz karaktera) gde se nalaze informacije o korisniku. Generisan token se vraća klijentu kao HTTP odgovor, i čuva se lokalno, najčešće unutar *Local Storage*-a veb pretraživača. Uz svaki naredni zahtev koji korisnik šalje ka serveru nije neophodno slati ponovo kredencijale (korisničko ime i šifru), već se u *header* zahteva ubaci dobijen JWT na osnovu kog server može da, nakon validacije samog tokena, identifikuje korisnika i proveriti da li on ima pravo da pristupi *endpoint*-u. Po *default*-u server očekuje token u *header*-u sa nazivom "*Authorization*", a pre samog tokena treba da se nalazi reč "*Bearer*". Da bi omogućili rad sa JWT-om, potrebno je isti uključiti preko *Maven*-a, zajedno sa *Spring Security*-em. Dekodiranje JWT-a možete pogledati na sajtu jwt.io

Modern Token-Based Auth



Slika 1:

<https://medium.com/@Raulgzm/securing-golang-api-using-json-web-token-jwt-2dc363792a48>

Da bi aplikacija, uz pomoć *Spring Security framework*-a, “automatski” radila autentifikaciju korisnika, generisanje jwt-a i kasniju autorizaciju na osnovu istog, potrebno je uraditi određenu konfiguraciju. Unutar projekta “Vezbe06JWT” kofigurisanje *security*-a odrađeno je u kalsi “SecurityConfiguration”. Pri konfiguraciji možemo navesti za koje zahteve hoćemo da radimo autentifikaciju, a koje delove naše aplikacije hoćemo da ostavimo otvorene i dostupne svim korisnicima, bez obzira da li poseduju token ili ne (slika 2). Pored toga, uz pomoć anotacije `@PreAuthorize()` možemo za svaki *endpoint* pojedinačno da navedemo kojoj grupi korisnika je dostupan, a kojoj ne (slika 3).

```
httpSecurity.headers().frameOptions().disable();
httpSecurity.csrf().disable() HttpSecurity
    .sessionManagement() SessionManagementConfigurer<HttpSecurity>
    .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
    .and() HttpSecurity
    .authorizeRequests() ExpressionUrlAuthorizationConfigurer<H>.ExpressionInterceptUrlRegistry
    .antMatchers( ...antPatterns: "/h2-console/**").permitAll()
    .antMatchers(HttpMethod.POST, ...antPatterns: "/users/login").permitAll()
    .antMatchers(HttpMethod.POST, ...antPatterns: "/users").permitAll()
    .anyRequest().authenticated();
```

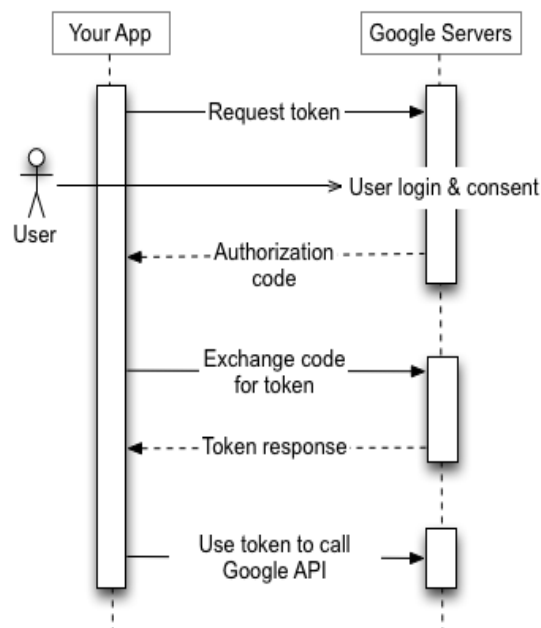
Slika 2: Konfiguracija Security-a

```
@PreAuthorize("hasRole('ADMIN')")
@PostMapping
public ResponseEntity<Club> create(@RequestBody Club club){
    Club createdClub= clubService.save(club);
    if(createdClub == null){
        return new ResponseEntity<>(null,HttpStatus.NOT_ACCEPTABLE);
    }
    return new ResponseEntity<>(createdClub, HttpStatus.OK);
}
```

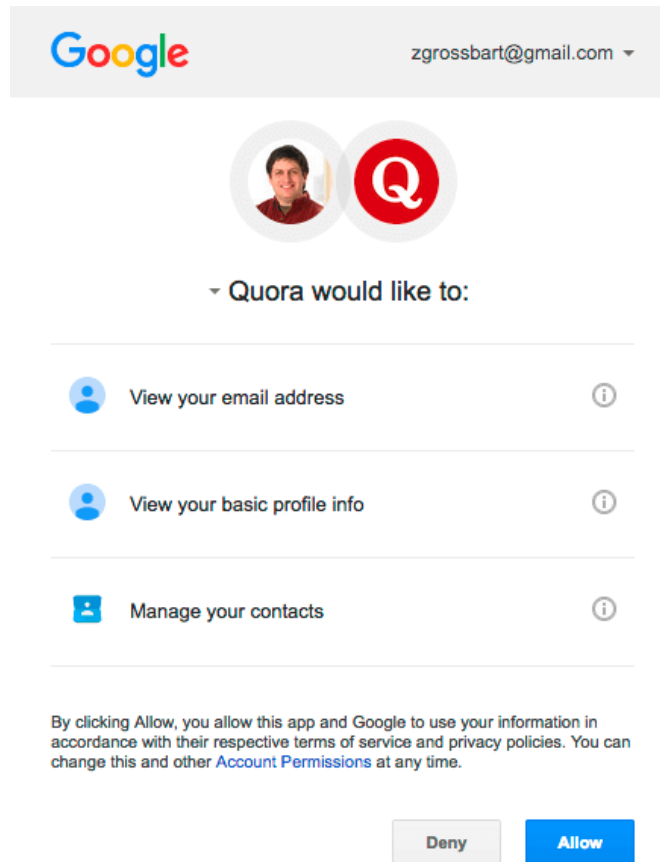
Slika 3: Autorizacija po ulogama

Autentifikacija uz OAuth 2.0

OAuth 2.0 je druga generacija otvorenog protokola za autorizaciju. Omogućuje aplikacijama da pristupe podacima korisnika koji se nalaze na nekim drugim veb aplikacijama, uz prethodnu dozvolu tog korisnika. Tok informacija pri korišćenju ovog protokola prikazan je na slici 4. U trenutku kada naša aplikacija hoće da pristupi bilo kom resursu koji se nalazi na serverima neke druge aplikacija (u primeru na slici je to *Google*), korisnik se *redirect*-uje na *OAuth* stranicu pomenute aplikacije. Deljenje resursa pomoću *OAuth* protokola omogućavaju svi veći sajtovi (Google, Facebook, Twitter, GitHub...), tako da ste se sigurno u nekom trenutku već susreli sa ovim protokolom, makar kao korisnici. Primer Google-ove *OAuth* login stranice možete videti na slici 5. Nakon što unesete ispravne kredencijale i dozvolite pristup vašim podacima, Google, ili neka druga aplikacija kojoj prisutpate, vas vraća nazad na vašu aplikaciju i šalje kod za autorizaciju uz pomoć kog se kasnije može preuzeti token za pristup željenim podacima.



Slika 4: <https://developers.google.com/identity/protocols/oauth2>




Slika 5

Ovaj protokol je pre svega namenjen za autorizaciju pristupa podacima, ali se danas često koristi za autentifikaciju/prijavljivanje korisnika. Ako ne želimo da čuvamo šifre korisnika u našoj bazi podataka, ili hoćemo da omogućimo korisniku da pristupi našoj aplikaciji bez potreba da se registruje i unosi svoje podatke, možemo ga preusmeriti na bilo koji veći servis gde on već poseduje kreiran nalog. Ako naša aplikacija dobije odgovarajuće povratne informacije od OAuth provajdera, znamo da je identitet korisnika proveren i da može da pristupi našoj aplikaciji.

Da bi naša aplikacija mogla da *redirect*-uje korisnike na OAuth stranicu nekog drugog sajta, potrebno je da na tom sajtu registrujemo aplikaciju. Svi sajtovi koji nude OAuth usluge imaju detaljnu dokumentaciju kako da se registruje aplikacija, i procedure su često veoma jednostavne. Za primer odrađen na vežbama korišćen je *GitHub* kao *OAuth* provajdera. Nakon logovanja na GitHub i odlaska na podešavanja (*Settings*) sa leve strane se nalazi meni u kom treba odabrati *Developer settings* opciju. Nakon toga prelazimo na prozor *OAuth Apps* i bira se opcija *New OAuth App*. Kada se formular popuni kao što je prikazano na slici 6, biće kreirani Client ID i Client Secret, slika 7. Ove podatke treba ubaciti u `application.properties` fajl

Vezba06OAuth aplikacije, kao bi komunikacija sa GitHub OAuthom bila uspešna.

Application name *



Something users will recognize and trust.

Homepage URL *

The full URL to your application homepage.

Application description

This is displayed to all users of your application.

Authorization callback URL *

Your application's callback URL. Read our [OAuth documentation](#) for more information.

Update application


Slika 6

Client ID

526a70c5bd1212f5453c

Client secrets

[Generate a new client secret](#)



Client secret

*****21d215d8

Added 6 days ago by VeljkoMaksimovic

Last used within the last week

Delete

Slika 7