Korisnički računi i OAuth2 autentikacija

Uvod
Cilj predavanja je implementirati "Login with Google" funkcionalnost, koja će našoj aplikaciji omogućiti autentikaciju korisnika koristeći Googleve korisnčke račune.

Mehanizam koji želimo implementirati opisan je na sljedećoj adresi, pa to detaljno proučite:
https://developers.google.com/+/web/signin/server-side-flow

1. Stvaranje računa za našu aplikaciju na Google Developers Console

2. Dodavanje Google Sign In gumba u aplikaciju:

   U index.html treba dodati:

   JavaScript za button:

```
<script
src="https://apis.google.com/js/client:platform.js?onload=start" async
defer>
</script>
```

   HTML za button dodamo tamo na DOM gdje zelimo da se login button nalazi

```
<span class="g-signin"
   data-scope="https://www.googleapis.com/auth/plus.login
https://www.googleapis.com/auth/userinfo.email"

data-clientid="225925658800-aldm0vn094hmo30kh1cs2t475boaq7al.apps
.googleusercontent.com"
   data-cookiepolicy="single_host_origin"
   data-redirecturi="postmessage" ,
   data-callback="googleSignInCallback">
</span>
```

   Svi atributi
   Najvažniji atributi:
   - data-redirecturi: mora biti "postmessage" za ovaj flow
   - data-scope: ovlasti prema Google korisničkom računu koje traži naša aplikacija
   - data-clientid: client_id naše aplikacije stvoren u prethodnom koraku

- data-callback: metoda koja će se pozvati nakon što korisnik obavi autorizaciju kod Google-a

3. Slanje authorization_code s klijenta na server, te spremanje dobivenog JWT tokena za daljnje korištenje:

```
$http.post('oauth2/google', null, {
   headers: { authorization_code: authResult.code }
})
.success(function (data) {
   $http.defaults.headers.common['token'] = data;
   $rootScope.loggedIn = data;
})
```

4. Primanje authorizatition_code na server, razmjena za access_token i dohvat podataka o korisniku:

```
@RequestMapping(value = "/oauth2/google", method = RequestMethod.POST)
public ResponseEntity authorizeWithGoogle(HttpServletRequest request)
throws IOException, GeneralSecurityException {


   String authorizationCode = request.getHeader("authorization_code");

   GoogleAuthorizationCodeTokenRequest tokenRequest = new
GoogleAuthorizationCodeTokenRequest(new NetHttpTransport(), new
GsonFactory(),
           SecurityConfig.CLIENT_ID, SecurityConfig.CLIENT_SECRET,
authorizationCode, "postmessage");

   GoogleTokenResponse response = tokenRequest.execute();

   Google google = new GoogleTemplate(response.getAccessToken());

   if (appUserService.getByUsername(person.getAccountEmail()) == null) {
       appUserService.create(person.getAccountEmail(),
person.getGivenName(),       person.getFamilyName());
    }
```

5. Server generira i potpisuje JWT token s podacima korisnika (email + exp time):

```
ObjectMapper mapper = new ObjectMapper();
String tokenContent = mapper.writeValueAsString(new
JWTClaims(person.getAccountEmail(), System.currentTimeMillis() +
SecurityConfig.TOKEN_EXPIRATION));

RsaSigner signer = new RsaSigner((RSAPrivateKey)
rsaKeysService.getKeys().getPrivate());
```

```
String jwt = JwtHelper.encode(tokenContent, signer).getEncoded();
```

6. Konfiguracija za Spring Security modul, koja dopušta pristup API-ju samo autoriziranim korisnicima s autoritetom USER, dok svi mogu dohvaćati ostali sadržaj i pristupati modulu za autorizaciju:

```
@Override
protected void configure(HttpSecurity http) throws Exception {
   http
            .authorizeRequests()
            .antMatchers("/oauth2/**").permitAll()
            .antMatchers("/tournaments/**").hasAuthority("USER")
            .antMatchers("/players/**").hasAuthority("USER")
            .antMatchers("/**", HttpMethods.GET).permitAll();
```

7. Filter koji ce provjeravati za sve requestove imaju li valjan JWT token i ako imaju autorizirati taj zahtjev

8. Implementirati klasu koja rukuje s neovlaštenim pristupom zaštićenom resursu (AuthenticationEntryPoint), jer po defaultu Spring Security preusmjerava zahtjev što nama ne odgovora jer imamo single-page aplikaciju i ne želimo da nas Spring preusmjerava preko browsera na drugi URL