

# **Napredni razvoj programske potpore za web**

**- predavanja -  
2021./2022.**

---

## **4. Načini provjere autentičnosti korisnika**

# Creative Commons



- **slobodno smijete:**
  - **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo
  - **prerađivati** djelo
- **pod sljedećim uvjetima:**
  - **imenovanje:** morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
  - **nekomercijalno:** ovo djelo ne smijete koristiti u komercijalne svrhe.
  - **dijeli pod istim uvjetima:** ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.

*U slučaju daljnog korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.*

*Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.*

*Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava.*

*Tekst licence preuzet je s <http://creativecommons.org/>*

# Provjera autentičnosti

- engl. *authentication* (fra. *authentification*)
- provjera autentičnosti (izvornosti, valjanosti) korisnika, utvrđivanje (dokazivanje) identiteta korisnika
  - „autentičan = koji je izvoran, koji zaista potječe od onoga kome se pripisuje; istinit, nepatvoren, vjerodostojan”
    - [https://hjp.znanje.hr/index.php?show=search\\_by\\_id&id=e11hXA%253D%253D](https://hjp.znanje.hr/index.php?show=search_by_id&id=e11hXA%253D%253D)
  - „Autentifikacija je utvrđivanje autentičnosti, tj. identiteta osobe koja pristupa podatcima”
    - Leksikografski zavod Miroslav Krleža  
<http://www.enciklopedija.hr/Natuknica.aspx?ID=68380>
- Različiti mehanizmi
  - Lozinka, pin, kartica, token, otisak prsta
  - MFA (*Multi-factor authentication*) – kombinacija dvaju ili više različitih mehanizama prijave

# Autorizacija korisnika

- engl. *authorization*
- „Postupak provjere ima li korisnik, računalo ili program pravo pristupa nekomu računalnomu sustavu”
  - Leksikografski zavod Miroslav Krleža
- „Stjecanje prava korištenja sustava i podataka pohranjenih na njemu”
  - Hrvatski jezični portal
- Provjera autentičnosti i autorizacija su često isprepleteni

# Kako identificirati i/ili autorizirati korisnika?

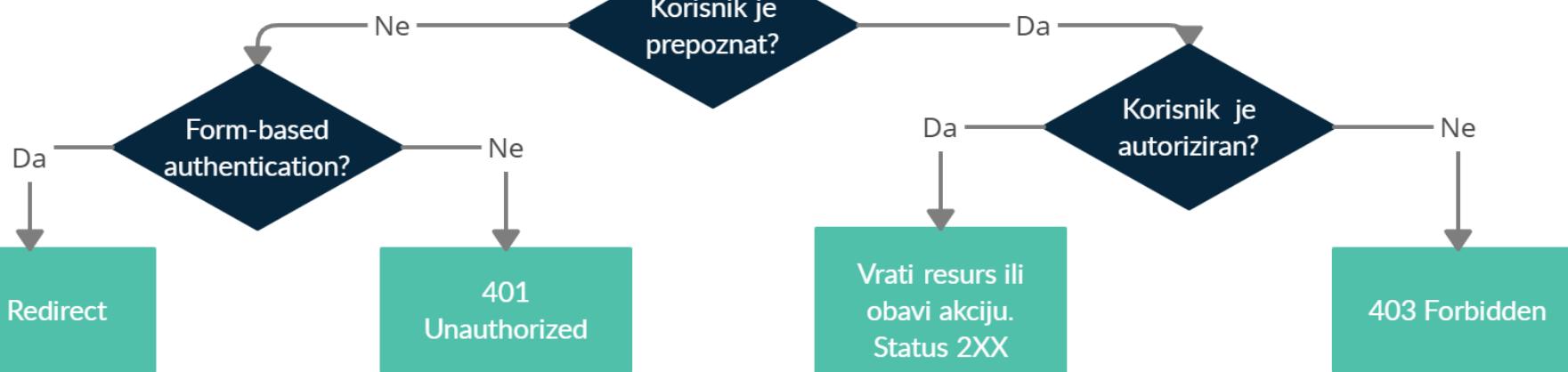
- Podsjetnik:
  - HTTP je aplikacijski protokol bez stanja (engl. *stateless*), tj. transakcije dohvata resursa su međusobno neovisne.
  - U svakom zahtjevu prema zaštićenom resursu ili stranici treba iznova identificirati i/ili autorizirati korisnika
    - Što se može slati?
      - korisničko ime i lozinka prilikom svakog zahtjeva
        - Podržano na razini HTTP protokola (vidi sljedeći slajd)
      - cookie ili token
        - Jednom identificiranom korisniku se pridruži cookie ili token čija se autentičnost može naknadno provjeriti i utvrditi o kome se radi i/ili kojom vrstom prava vlasnik (tokena) raspolaže
        - Tko i kako će inicijalno identificirati korisnika je zasebno pitanje
  - Kako klijentu ukazati da je potrebna autentifikacija?

# Odgovori ne[autentificiranim | neautoriziranim] korisnicima

- Prilikom zahtjeva prema zaštićenoj stranici ili resursu prvo se pokušava identificirati (autentificirati) korisnika
- Neprijavljeni korisnici će dobiti *challenge* koji se svodi na preusmjeravanje na formu za prijavu ili na odbijanje zahtjeva s informacijom o vrsti provjere autentičnosti koju treba koristiti
  - npr. WWW-Authenticate: <type> realm=<realm>
  - realm je proizvoljni atribut koji opisuje područje zaštite

Zahtjev prema  
štićenoj  
stranici/resursu

Načelno, više aplikacija može koristiti isti realm, što sugerira  
pregledniku da koristi iste podatke



# Provjere autentičnosti na razini HTTP protokola

- HTTP protokol podržava dva mehanizma: *Basic* i *Digest*
  - Ako preglednik na neki zahtjev kao odgovor dobije status 401 (*Unauthorized*) i zaglavlje WWW-Authenticate postavljeno na *Basic* ili *Digest*, prikazat će korisniku dijalog za upis korisničkog imena i zaporce

The screenshot shows a web browser window with the URL `127.0.0.1:4010/private`. The browser's address bar also displays `127.0.0.1:4010/private`. To the left of the browser, there is a summary of the request and response headers.

**Request URL:** `http://127.0.0.1:4010/private`

**Request Method:** GET

**Status Code:** 401 Unauthorized

**Remote Address:** 127.0.0.1:4010

**Referrer Policy:** strict-origin-when-cross-origin

---

**Response Headers** [View source](#)

**Connection:** keep-alive

**Date:** Sat, 04 Sep 2021 18:35:05 GMT

**Keep-Alive:** timeout=5

**Transfer-Encoding:** chunked

**WWW-Authenticate:** Basic realm="FER-Web2 Examples"

The browser's main content area contains a login form with two fields: "Korisničko ime" (Username) and "Zaporka" (Password). Below the fields are two buttons: "Prijava" (Login) and "Odustani" (Cancel).

# Provjera autentičnosti kroz formu

- Obično se provjera korisnika obavlja kroz posebnu formu (engl. *Form Based Authentication*)
  - POST zahtjev u kojem se šalje korisničko ime i lozinka
    - Umjesto korisničkog imena i lozinke može se slati i jednokratno generirani kod s posebnog uređaja
      - posebni USB ključevi
      - generatori tokena (fizički ili kao aplikacije) ...
- Ako se koriste dva ili više mehanizama za provjeru autentičnosti (npr. korisničko ime i lozinka + kod iz aplikacije), tada se radi o višefaktorskoj autentifikaciji (MFA engl. *Multi Factor Authentication*)
- Identificiranom korisniku se izdaje cookie ili token koji se onda šalje u sljedećim zahtjevima

# Sadržaj *cookiea* ili tokena

- Sadržaj identifikacijskog *cookiea* treba biti takav da aplikacija može jednoznačno identificirati korisnika te da može provjeriti autentičnost *cookiea* (je li možda korisnik sam izmijenio cookie).
  - Naziv ključa i vrijednost proizvoljni i/ili ovisni o korištenim tehnologijama
- Slično vrijedi i za token koji je *string*
  - Kodiranje tog stringa može biti standardizirano, npr. za tzv. JWT tokene (više o tome naknadno)

# Tko će identificirati i/ili autorizirati korisnika?

- Provjera na razini aplikacije (engl. *Application Based Authentication*)
  - Aplikacija pohranjuje podatke o korisnicima i njihovim lozinkama
- Provjera korištenjem vanjske usluge (eng. *third party authentication*)
  - Prijava/provjera korisnika se obavlja putem vanjske usluge koja određenim mehanizmima (standardima) aplikaciji proslijedi podatke o prijavljenom korisniku
  - AAI, Microsoft (Azure), Facebook, Google, Auth0, ...
  - Aplikacija može imati svoju bazu korisnika, ali ne i njihove lozinke

# Sadržaj primjera (1)

- 01-basic-auth
  - Demonstrira se *Basic Authentication* na jednostavnoj web-aplikaciji
- 02-digest-auth
  - Demonstrira se *Digest Authentication* na jednostavnoj web-aplikaciji
- 03-token-auth
  - Demonstrira se autorizacija pomoću tokena. Token izdaje jednostavna web-aplikacija nakon prijave korisnika, a demonstrira se pozivom iz konzole (cURL)
- 04-token-auth-cors
  - Nadgradnja prethodnog primjera u kojem se token koristi iz JavaScripta neke druge web-aplikacije od one koja je izdala token i koja ima zaštićeni resurs dostupan nosiocu valjanu tokena.
    - Napomena: Neprecizno rečeno, u primjerima ćemo taj JavaScript kod proglašiti za Single-Page aplikaciju (SPA), jer se demonstracijski primjeri sastoje samo od jedne stranice s uključenim JavaScript kodom, ali princip može vrijediti i općenito

# Sadržaj primjera (2)

- 05-cookie-auth
  - Demonstrira se autorizacija temeljem kolačića (*cookie*) kojeg je web-aplikacija postavila nakon prijave korisnika
- 06-cookie-auth-js
  - Nadgradnja prethodnog primjera u kojoj JavaScript kod neke stranice, poslužene na istom serveru kao web-aplikacija, koristi *cookie* za dohvatanje zaštićenih resursa
- 07-cookie-auth-js-cors
  - Varijacija prethodnog primjera u kojem se JavaScript kod nalazi u nekoj drugoj web-aplikaciji od one koja izdaje *cookie*.
- 08-oauth2
  - Primjer web-aplikacije na kojoj se prijava vrši preko vanjskog servisa za prijavu korisnika po protokolima OAuth2 i OpenIdConnect (OIDC)
- 09-auth2-spa-and-web-api
  - Primjer više aplikacija koje koriste sustav jednostrukе prijave korisnika (eng. single-sign-on) po protokolima OAuth2 i OIDC

# Struktura pojedinog primjera (1)

- Serverski kod pisan u Node.js-u
- Pojedina putanja vraća json, http status o pogrešci ili vraća stranicu definiranu nekim predloškom.
- Za predloške korišten pug
- Minimalistički primjeri kako bi se očuvao fokus na načinima prijave

```
html
  head
    title Basic Auth demo
  body
    if user.isAuthenticated
      h1 Welcome #{user.username}
    ul
      li: a(href='/', title='Home') Home
      li: a(href='/private') Private content (only for Alice and Bob)
      li: <span>Note: Use <i>some password</i> as password</span>
```

```
const express = require('express');
const app = express();
app.use(express.static('public'));
app.set('view engine', 'pug');
...
app.get('/', function (req, res) {
  res.render('index', {user : req.user});
});
01-basic-auth.js
app.get('/private', ...)
```

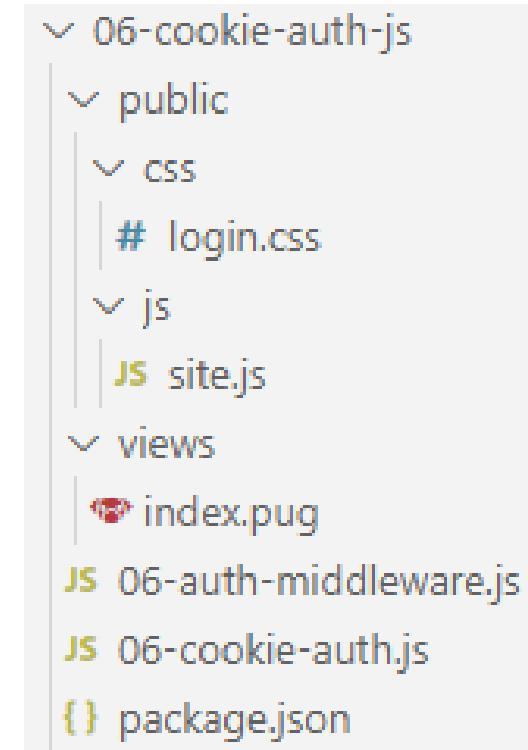
# Struktura pojedinog primjera (2)

- Pojedini primjeri koriste osjetljive podatke o lozinkama ili postavkama aplikacije te su smještene u datoteku .env
  - U konkretnim primjerima prikazan konkretni sadržaj
  - Dohvat vrijednosti se vrši preko process.env uz prethodnu instalaciju paketa *dotenv*

```
const dotenv = require('dotenv');
dotenv.config();
... process.env.COOKIE_KEY ...
```

- Korišteni paketi navedeni u *package.json*
- Mapa *public* sadrži stilove i skripte
- Mapa *views* sadrži predloške pisane u *pugu*
- js datoteke sadrže
  - kod web-aplikacije
  - kod koji pokreće server za SPA s kodom u mapi public/js
  - kod za *middleware*

```
const express = require('express');
const app = express();
app.use(express.static('public'));
```



# Postavljanje informacije o korisniku

- Uloga *middlewarea* je izvršiti određeni kod prije koda za obradu pojedinog zahtjeva
- Omogućava npr. odbijanje zahtjeva, preusmjeravanje na neku drugu adresu i/ili rekonstrukciju informacije u korisniku
  - Ovisno od primjera do primjera rekonstrukcija će biti drugačija
    - različiti header, cookie, OAuth2, ...
    - nakon „prolaza“ kroz middleware `req.user` sadrži `isAuthenticated` i (opcionalno) `username`
- Osim u primjeru s OAuth2 lozinka za pojedinog korisnika je `some_password`
  - U stvarnim primjerima smjestiti lozinku (tj. njen hash) u bazu podataka

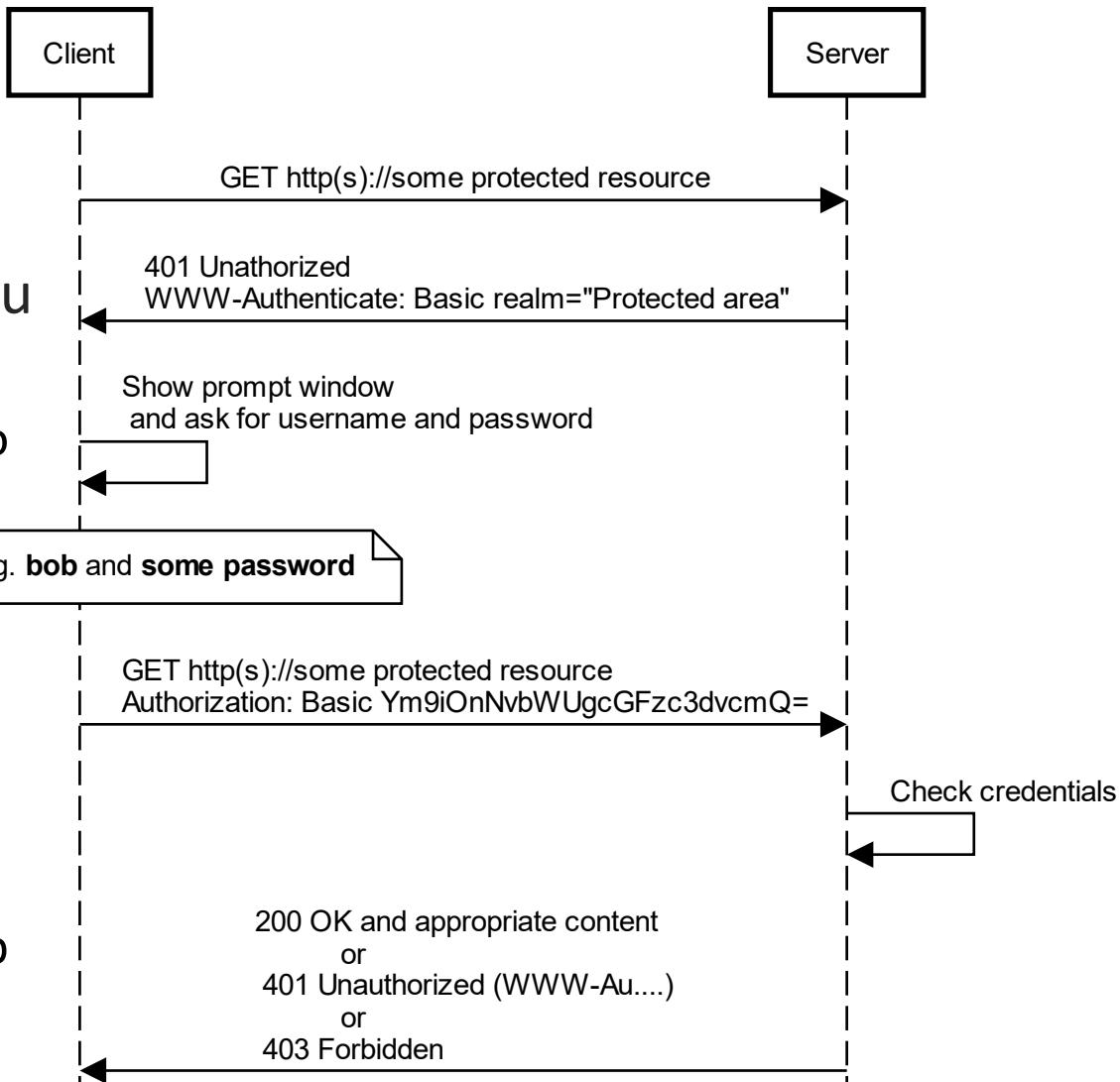
# 01 - Basic Authentication

- <https://datatracker.ietf.org/doc/html/rfc7617>
- Korisničko ime i lozinka se šalju prilikom svakog zahtjeva
  - Korisničko ime i lozinka spojeni u jedan string kodiran u Base64 formatu (kodiranje ≠ kriptiranje)
    - npr. za korisničko ime *boris* i lozinku *neka lozinka*, dobiveni string je Qm9yaXM6bmVrYSBsb3ppbmth
  - Dio zaglavlja u obliku Authorization: Basic Base64EncodedString
- Prednost:
  - Jednostavno za implementaciju, podržano u svim preglednicima
- Nedostatci:
  - kodiranje je reverzibilno što čini koncept vrlo nesigurnim i smije se koristiti samo preko https-a
  - **Nemoguće odjaviti korisnika iz aplikacije! Zatvoriti preglednik?**

# Tipični slijed za *Basic Authentication*

- Rezultat zadnjeg koraka na prikazanom dijagramu slijeda može biti:

- 401 ako korisnik nije unio ispravno korisničko ime i lozinku
- 200 ako je korisnik unio ispravno ime i lozinku i ima pravo pristupa
- 403 ako je korisnik unio ispravno ime i lozinku, ali nije autoriziran za pristup konkretnom resursu



# Primjer za Basic authentication - middleware

- korisničkoime:lozinka kodirani s Base64

```
function getUserInfo(req, res, next) {  
    req.user = {isAuthenticated : false };  
    if (req.headers.authorization) {  
        let data = req.headers.authorization.replace(/^Basic /, '');  
        data = Buffer.from(data, 'base64').toString('utf8');  
        const loginInfo = data.split(':');  
        const username = loginInfo[0];  
        const password = loginInfo[1];  
        if (password === 'some password') {  
            req.user.isAuthenticated = true;  
            req.user.username = username;  
        }  
        else { //invalid username or password  
            authenticationNeeded(req, res, next);  
            return;  
        }  
    }  
    next();  
}
```

01-auth-middleware.js

Middleware kojim se potencijalni korisnik prepoznaje iz zaglavlja zapis oblika  
Authorization: Basic  
Base64EncodedString

Prilikom pristupa zaštićenim stranicama neprijavljenom korisniku šaljemo odgovor sa status 401 i načinom prijave

```
const realm = "FER-Web2 Examples";  
function authenticationNeeded(req, res, next) {  
    if (!req.user.isAuthenticated) {  
        res.writeHead(401, {  
            'WWW-Authenticate'  
            : 'Basic realm=' + realm + ''});  
        res.end('Authentication is needed');  
    }  
    else next();  
}
```

# Primjer za Basic authentication

```
...
const auth = require('./01-auth-middleware');
app.use(auth.getUserInfo);
...
app.get('/', function (req, res) {
    res.render('index', {user : req.user});
});
app.get('/private', auth.authenticationNeeded, function (req, res) {
    const username = req.user.username;

    if (username.toLowerCase() === 'alice' || username.toLowerCase() === 'bob'){
        res.render('private', {username : username});
    }
    else {
        res.status(403);
        res.end('Forbidden for ' + username);
    }
});
});
```

Za sve putanje pokušavamo utvrditi tko je prijavljen korisnik

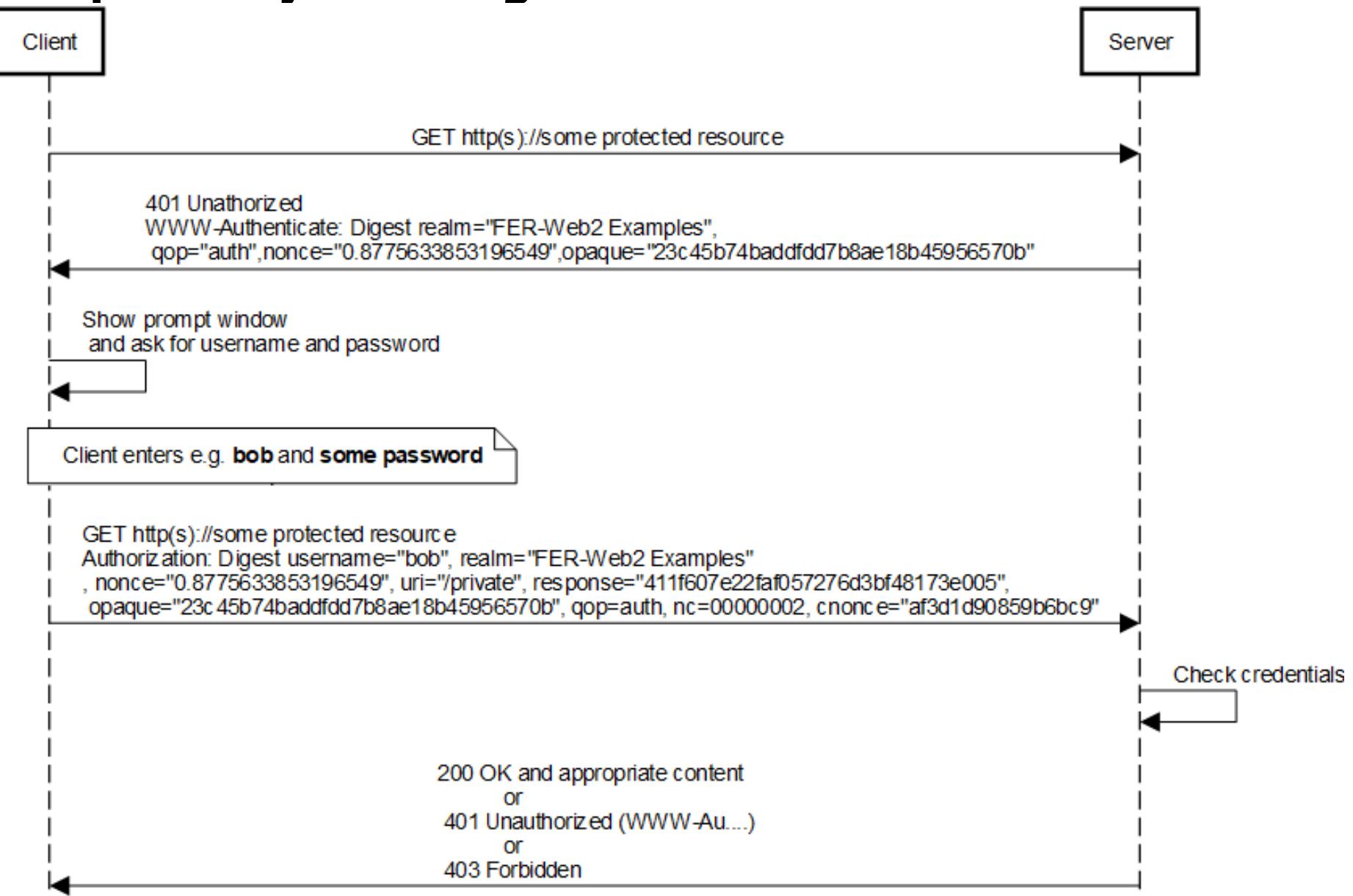
Pristup pojedinoj stranici mogu imati samo prijavljeni korisnici, a onda kroz kod vršimo autorizaciju

01-basic-auth.js

## 02 - Digest authentication

- Kao i kod *Basic Authentication* odgovor preglednika na *401 Unauthorized WWW-Authenticate:Digest* se svodi na otvaranje dijloga za unos korisničkog imena i lozinke
- Umjesto lozinke šalje se kombinacija sažetaka (digest)
- U primjeru koji slijedi opisana varijanta s *qop = auth*
  - *qop = quality of protection*
  - detaljnije na <https://www.ietf.org/rfc/rfc2617.txt>
- Napomena: Sadržaj glavne stranice je isti, mijenja se samo middleware dio

# Tipični slijed za *Digest Authentication*



# Digest authentication – opis parametara

- Neprijavljenom korisniku server šalje tzv. *nonce*
  - npr. pseudo-slučajni broj
- Klijent izračuna dvije hash vrijednosti koristeći MD5
  - HA1 = MD5(username:realm:password)
  - HA2 = MD5(request method:uri)te upotrijebi *nonce*, *cnonce (client nonce)* i *nc (nonce/request counter)* za novi sažetak
  - response = MD5(HA1:nonce:nc:cnonce:qop:HA2)
- Klijent šalje izračunate vrijednosti, a server računa svoj *response* koji se mora poklapati s onim kojeg mu je klijent poslao.
  - Parametar *opaque* treba biti nepromijenjen (može poslužiti kao identifikator sjednice)

# Middleware za Digest authentication (1)

```
function getUserInfo(req, res, next) {  
    req.user = {isAuthenticated : false};  
    if (req.headers.authorization) {  
        const authString = req.headers.authorization.replace(/^Digest /, '');  
        const authData = parseAuthenticationString(authString);  
        const username = authData.username;  
            //in our example all users has 'some password' as password  
        const ha1 = md5Digest(` ${username}: ${realm}: some password`);  
        const ha2 = md5Digest(` ${req.method}: ${authData.uri}`);  
        const calculatedResponse = md5Digest(` ${ha1}: ${authData.nonce}: ${authData.nc}: ${  
authData.cnonce}: ${authData.qop}: ${ha2}`);  
        if (calculatedResponse !== authData.response) { //invalid digest  
            authenticationNeeded(req, res, next); return;  
        }  
        else{  
            req.user.isAuthenticated = true; req.user.username = username;  
        }  
    }  
    next();  
}
```

Middleware kojim se potencijalni korisnik prepoznaže iz zaglavlja zapis oblika Authorization: Digest i parova ključ=vrijednost

02-auth-middleware.js

# Middleware za Digest authentication (2)

```
function parseAuthenticationString(authString) {  
    var authData = {};  
    authString.split(", ").forEach(function(pair) {  
        const arr = pair.split('=');  
        authData[arr[0]] = arr[1].replace(/\//g, ''));  
    });  
    return authData;  
}
```

Parovi ključ=vrijednost bili odvojeni zarezom. Vrijednostima treba ukloniti navodnike.

02-auth-middleware.js

```
const realm = "FER-Web2 Examples";  
function authenticationNeeded(req, res, next) {  
    if (!req.user.isAuthenticated) {  
        res.writeHead(401, { 'WWW-Authenticate' :  
`Digest realm="${realm}",qop="auth",nonce="${Math.random()}",opaque="${hash}"`});  
        res.end('Authentication is needed');  
    }  
    else {  
        next();  
    }  
}
```

Prilikom pristupa zaštićenim stranicama neprijavljenom korisniku šaljemo odgovor sa status 401 i načinom prijave

# Potencijalni problem s Digest Authentication

- Isti kao i kod *Basic Authentication*, nije moguće odjaviti korisnika
- Hipotetski moguće koristiti i bez https-a, ali nije preporučeno (ne toliko jak kriptografski ključ)
- Dodatno, problem potrebe za čuvanjem izvorne lozinke na serveru
  - Čuvanje „čistih“ lozinki svih korisnika na serveru je potencijalno veći problem od krađe lozinke jednog korisnika

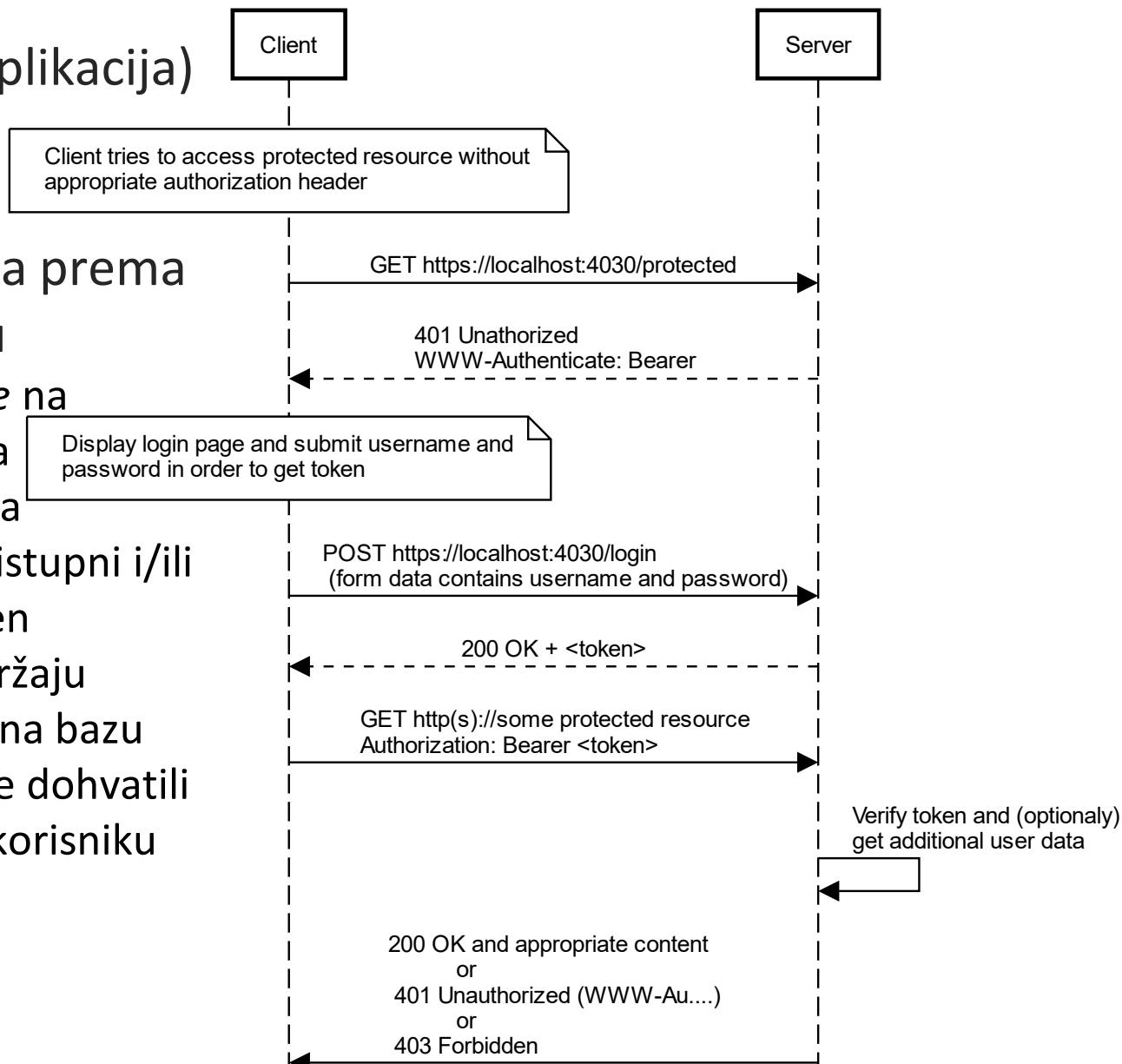
# 03 - Bearer (Token) authentication

- <https://datatracker.ietf.org/doc/html/rfc6750>
- Korisnik se identificira (pristupnim) tokenom (engl. *access token*) - „*Omogući pristup nositelju ovog tokena*”
  - Zaglavje zahtjeva sadrži zapis oblika Authorization: Bearer <token>
    - koristiti samo preko https-a!
  - Što je token i kako nastaje?
    - Izdaje ga onaj kojem se korisnik prijavio na neki način
    - Može biti bilo kakav tekst kojim se na serveru jednoznačno odredili podaci o korisniku (npr. upitom u bazu podataka) i/ili kako bi se odredilo ima li korisnik pravo pristupa
      - Često je to JWT (detaljnije uskoro) čime se omogućava da token sadrži informacije o korisniku (tko je, koja prava posjeduje, ...)
- Server neprijavljenom korisniku šalje status 401 *Unauthorized*, a u zaglavju *WWW-Authenticate: Bearer*
  - **Preglednik ne nudi nikakvu reakciju na ovaj odgovor!**

# Tipični slijed za *Token Authentication*

- Klijent (klijentska aplikacija) mora na neki način dobiti token kojeg će slati u zahtjevima prema zaštićenom resursu

- Server (*middleware* na serveru) provjerava autentičnost tokena
- Token može biti pristupni i/ili identifikacijski token
- Ovisno o vrsti i sadržaju tokena vrši se upit na bazu podataka kako bi se dohvatili potrebni podaci o korisniku



# JWT (JSON Web Token)

- JWT predstavlja standard kodiranja tokena koji u sebi nosi informacije o korisniku (engl. *claims*)
- JWT se sastoji od tri dijela kodirana u Base64 formatu:
  - Zaglavje s tipom tokena i hash algoritmom
  - Sadržaj (korisni podaci, engl. *payload*)
    - Parovi oblika ključ vrijednost
    - Trebao bi barem sadržavati ključeve *iat* (*issued at*), *exp* (*expiration time*) i nešto što identificira korisnika
      - U našem primjeru *username*, inače se za to koristi ključ *sub*
  - Potpisni dio (temeljem sadržaja)
- Primjer tokena:  
`eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmtSI6ImFsaWN1Iiwicm9sZSI6ImFkbWluIiwiaWF0IjoxNjMxNDU2NAzLCJleHAiOjE2MzE0NTY1MjN9.ekPa6i2rdMx4-y1FGgjXxVQcUXS1rP_T98DndEj1GIg`
  - Pogledati sadržaj na <https://jwt.io>

# Primjer JWT tokena i opisa na jwt.io

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmtzSI6ImFkbWluIiwiaWF0IjoxNjMxNDU2NDAzLCJleHAiOjE2MzE0NTY1MjN9.ekPa6i2rdMx4-y1FGgjXxVQcUXS1rP\_T98DndEj1GIg

Sun Sep 12 2021 16:22:03 GMT+0200 (srednjoeuropsko ljetno vrijeme)

HEADER:

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

PAYOUT:

```
{  
  "username": "alice",  
  "role": "admin",  
  "iat": 1631456403,  
  "exp": 1631456523  
}
```

VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  your-256-bit-secret  
) □ secret base64 encoded
```

# Opis primjera kreiranja i korištenja tokena

- Web-aplikacija se sastoji od naslovnice, login forme koja nakon uspješne prijave vraća korisnikov token te putanje koja autentificiranim korisnicima vraća određeni sadržaj
  - Klijent u ovom primjeru može biti *curl*, *Postman* ili slično.
    - Primjer poziva zaštićene stranice iz naredbenog retka

```
curl --  
header "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJc  
2VybmFtZSI6ImJvYiIsImhdCI6MTYyNjgwMDkxNywiZXhwIjoxNjI2ODA4MTE3fQ.AZiuxW  
iHMIzxJx1Rzeo37D4IRzEi1bgCej7Q2WAQLqY" http://127.0.0.1:4030/protected
```
    - Preglednik ne šalje token sam od sebe (kao što će biti slučaj s *cookiem*)
- Za izradu i provjeru tokena koristi se paket *jsonwebtoken*
  - Ključ zapisan u datoteci *.env* sa sadržajem *TOKEN\_KEY=unijeti proizvoljni ključ*

# Primjer za Token authentication (1)

03-token-auth.js

```
const auth = require('./03-auth-middleware');
app.use(auth.verifyToken);
...
app.get('/', function (req, res) {
  res.render('index', {user : req.user});
});
app.post('/login', function (req, res) {
  const username = req.body.username;
  const password = req.body.password;
  if (password !== 'some password')
    res.render('login');
  else {
    const payload = { username };
    if (username.toLowerCase() === 'alice') {
      payload['role'] = 'admin';
    }
    const token = auth.createToken(payload);
    res.json(token);
  }
});
```

Middleware koji provjerava token te metode koje kreiraju token odvojene u posebnu datoteku

Korisničko ime i lozinka zapisani u *FormData* dijelu zahtjeva

```
const app = express();
...
app.use(express.urlencoded({ extended: true }));
```

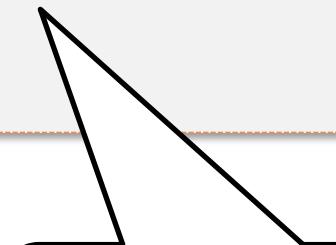
Za autentificiranog korisnika kreiramo token u kojem je zapisano korisničko ime te informacija pripada li administratorskoj grupi.

U primjeru token vraćen kao json

# Primjer za Token authentication (2)

03-auth-middleware.js

```
const jwt = require("jsonwebtoken");
const dotenv = require('dotenv');
dotenv.config();
...
function createToken(payload) {
  return jwt.sign(payload, process.env.TOKEN_KEY, { expiresIn : "2m" });
  //2 minutes
}
```



Ključ kojim se potpisuje token  
zapisan u datoteci .env sa sadržajem  
TOKEN\_KEY=unijeti proizvoljni ključ

# Primjer za Token authentication (3)

03-auth-middleware.js

```
function verifyToken(req, res, next) {  
    req.user = {isAuthenticated: false, isAdmin: false};  
    let token = req.headers.authorization?.replace(/^Bearer /, '');  
  
    if (token) {  
        try {  
            token = jwt.verify(token, process.env.TOKEN_KEY);  
            req.user.isAuthenticated = true;  
            req.user.username = token.username;  
            req.user.isAdmin = token.role === 'admin';  
        } catch (err) {  
            return res.status(401).send("Invalid Token");  
        }  
    }  
    return next();  
}
```

Koristi se za dijelove  
u kojima korisnik  
mora biti prijavljen

Provjerava se postoji li token u zaglavju zahtjeva te je li ispravan.  
Nakon izvršavanja req.user sadrži informaciju o korisniku (je li prijavljen, tko je, je li administrator, ...)

```
function authenticationNeeded(req, res, next) {  
    if (!req.user.isAuthenticated) {  
        res.writeHead(401, { 'WWW-Authenticate': 'Bearer' });  
        res.end('Authentication is needed');  
    }  
    else next();  
}
```

# Primjer za Token authentication (4)

03-token-auth.js

Samo za prijavljene korisnike

```
app.get('/protected', auth.authenticationNeeded, function (req, res) {
  const username = req.user.username;

  if (username.toLowerCase() === 'alice' || username.toLowerCase() === 'bob') {
    const data = {
      'CurrentTime' : Date.now(),
      'Message' : `Welcome ${username}`
    };
    res.json(data);
  }
  else {
    res.status(403);
    res.end('Forbidden for ' + username);
  }
});
```

# Primjer korištenja tokena iz Javascript aplikacije

- Web-aplikacija/servis (koja stvara token i ima zaštićeni resurs) se nalazi na portu 4041, a prijava i dohvata zaštićenog resursa se odvija iz Javascript koda u sklopu stranice izložene na portu 4042
  - Forma za prijavu prebačena na klijentsku (single-page) aplikaciju (vidi site.js i client.pug)
  - Javascript kod prikupi podatke i pošalje ih web-aplikaciji na provjeru kako bi dobio token koji će kasnije koristiti

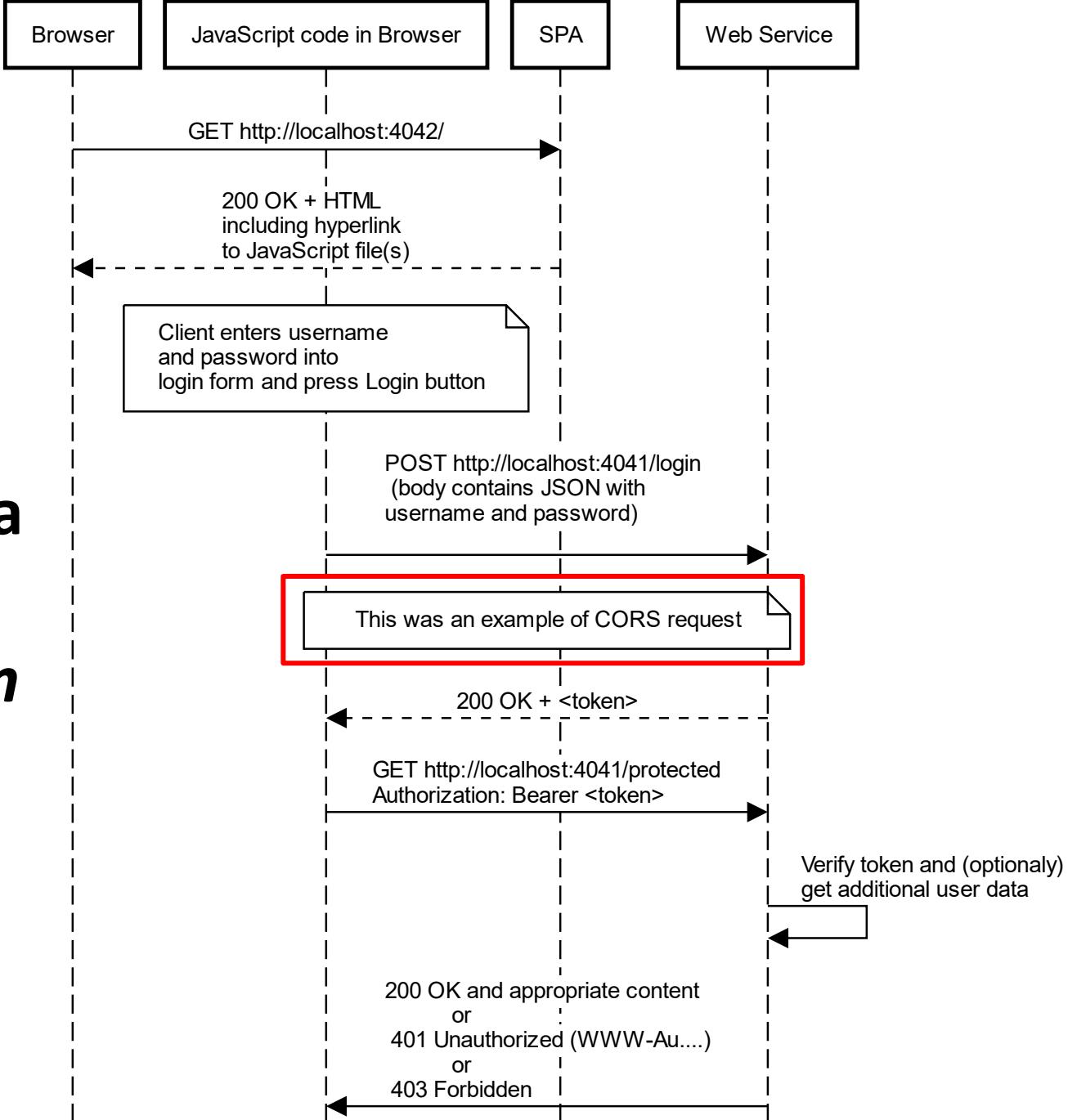
```
const username = document.getElementById('username').value;
const password = document.getElementById('password').value;
axios.post(` ${serverUri}/login`, {username, password})
  .then(res => {
    const jwt = res.data;
    storeToken(jwt);
    showLoginForm(false);
  })
  .catch(function (error) {
    document.getElementById('content').innerHTML = error;
});
});
```

Gdje spremiti token?

Kod koji se izvrši  
klikom na login

04-token-auth-cors/public/js/site.js

# Tipični slijed za *Token Authentication* iz SPA



# CORS

- CORS = Cross-Origin Resource Sharing
  - Pozivatelj i web-aplikacija nisu na istom serveru (portu)
  - Javascript kod s porta 4042 poziva resurs iz web-aplikacije na portu 4041
  - Nije dozvoljeno, osim ako web-aplikacija eksplicitno ne dozvoli

Access to XMLHttpRequest at 'http://localhost:4041/protected' from origin 'http://127.0.0.1:4042' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource.

- Preglednik dozvolu provjerava metodom OPTIONS

Access-Control-Allow-Methods: GET,HEAD,PUT,PATCH,  
POST,DELETE

Access-Control-Allow-Origin: \*

Dio zaglavlja u rezultatu poziva  
OPTIONS za URL  
`http://localhost:4041/protected`

U aplikaciji dopustimo CORS  
(u ovom primjeru bez ograničenja)

```
var cors = require('cors');  
app.use(cors());
```

04-token-auth-cors.js

# Dodavanje tokena u zahtjev

- Token se stavlja u zaglavlje zahtjeva prema zaštićenom resursu

04-token-auth-cors/public/js/site.js

```
function getdata() {  
    ...  
    axios.get(`${serverUri}/protected`, {  
        headers : {  
            Authorization : `Bearer ${getToken()}`  
        }  
    })  
    .then(function (response) {  
        ...  
    })  
    ...  
}  
  
function getToken() {  
    ... //vrati spremljeni token  
}
```

# Gdje spremiti token?

- Token (načelno) ne bi trebalo spremati u pregledniku (npr. u *local storage*), jer je dostupan svim skriptama koje su uključene u konkretnu aplikaciju
  - Pospremiti u varijablu?
    - funkcioniра do prvog osvježavanja stranice
  - Worker? Nešto treće?
  - Većina izvora navodi što ne bi trebalo, ali ne i što bi trebalo i kako to napraviti
    - Koliko npr. vjerujemo kodu od axiosa, bootstrapa, ... ?

Token u primjeru  
pohranjen  
u *localStorage*

04-token-auth-cors/public/js/site.js

```
const JWT_TOKEN = 'jwt';

function getToken() { return localStorage.getItem(JWT_TOKEN); }

function storeToken(jwt) { localStorage.setItem(JWT_TOKEN, jwt); }

function removeToken() { localStorage.removeItem(JWT_TOKEN); }
```

# Opaska oko zaglavlja za token

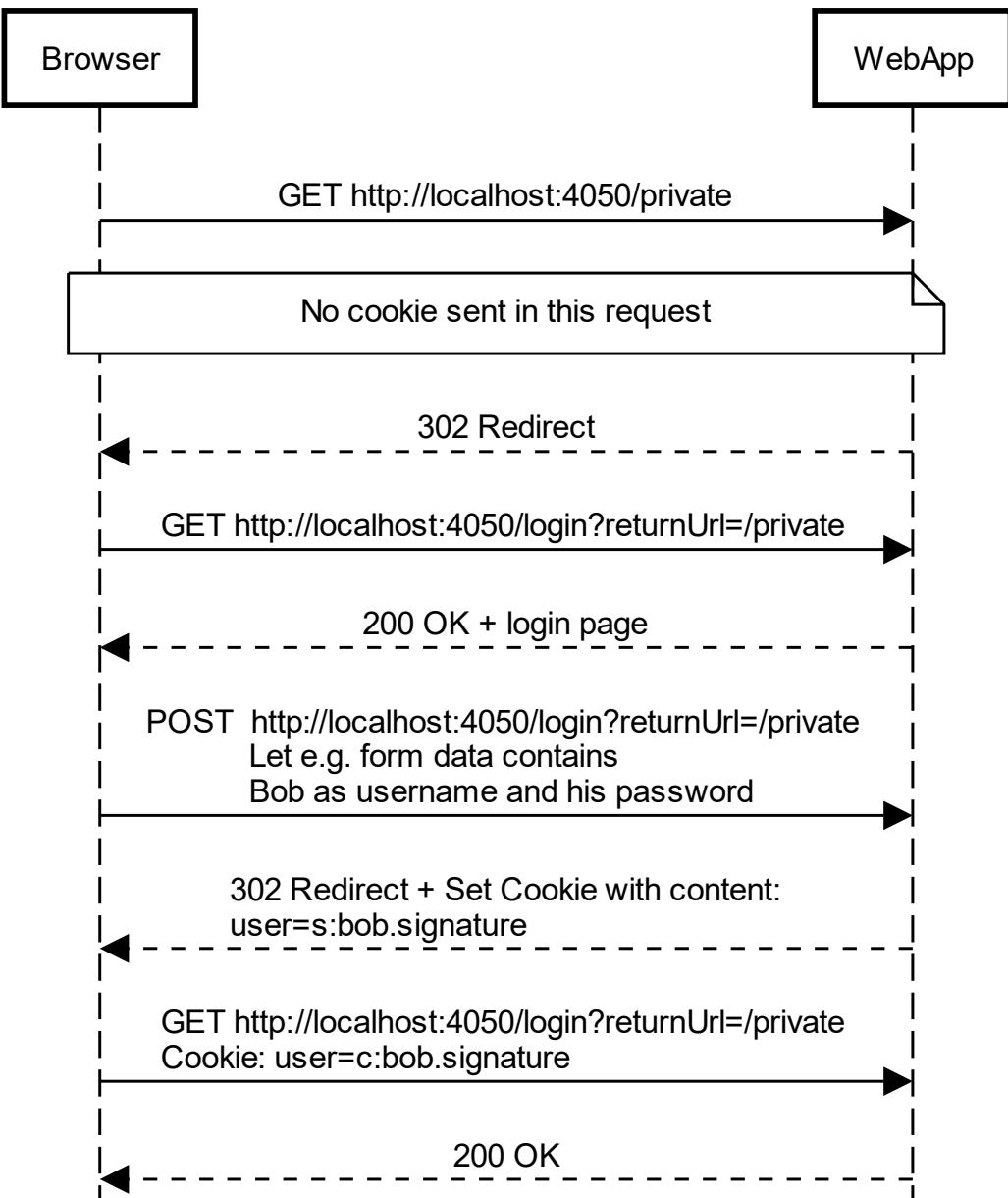
- Standardno zaglavljje za token je *Authorization: Bearer token*, ali nije neuobičajeno da je token dio URL-a ili naveden u zaglavljju s X-Auth-Token ili slično
  - Sličnost s API ključevima ili drugim jedinstvenim identifikatorima korisnika ili aplikacije

# Cookies

- U prethodnim primjeri u zaglavlju zahtjeva se nalazio zapis oblika *Authorization tip sadržaj*
  - u slučaju Basic i Digest to je automatski dodavao preglednik
  - tokene je trebalo eksplicitno dodati
- Korisnika možemo identificirati i temeljem *cookiea*
- Cookie je par *ključ = vrijednost* kojeg server (web-aplikacija) vrati u nekom od odgovora, korisnikov preglednik pohrani, a zatim prilikom **svakog** sljedećeg zahtjeva šalje natrag
  - osim kod prijave korisnika koristi se i za pamćenje raznih korisničkih postavki, ali i za praćenje korisnika i analizu ponašanja
- Da bi preglednik pohranio *cookie*, od servera mora primiti odgovor koji u zaglavlju sadrži *Set-Cookie*
  - Više vrsta *cookiea* i mogućih postavki – do dalnjeg nam je bitno da je označen sa **secure** (sadržaj je potpisani) i **http-only** (ne može mu se pristupiti iz JavaScripta)
    - <https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies>

# Tipični slijed za *Cookie Authentication*

- Nakon prijave klijent dobije *cookie* kojeg šalje u svakom sljedećem zahtjevu
- Cookie je par *ključ=vrijednost*
- Što je ključ, a što vrijednost?
  - Ovisi o aplikaciji
    - u primjeru koji slijedi to je naziv korisnika
- Može li se sadržaj *cookiea* ručno promijeniti
  - Da! Zato se koristi *signed cookie* koji osim vrijednosti zadrži i potpis određen šifrom unutar aplikacije



# Primjer za Cookie (1)

05-cookie-auth.js

```
const auth = require('./05-auth-middleware');
auth.initCookieAuth(app);
app.use(auth.getUserFromCookie);

app.get('/login', function (req, res) {
  res.render('login');
});

app.post('/login', function (req, res) {
  var username = req.body.username;
  var password = req.body.password;
  if (password !== 'some password') {
    res.render('login');
  }
  else {
    auth.signInUser(res, username);
    res.redirect ... o ovome nešto kasnije
  }
});
```

Aktiviramo middleware koji provjera cookie te sadrži metode koje kreiraju token (zasebna datoteka)

Za autentificiranog korisnika kreiramo cookie (vidi kod u 05-auth-middleware.js)

Korisnika odjavimo uklanjanjem cookiea (vidi kod u 05-auth-middleware.js).

Iako postoji puno slučajeva gdje to nije izvedeno ili nije moguće izvesti, metoda za odjavu bi trebala biti POST, a ne GET

```
app.post('/logout', function (req, res) {
  auth.signOutUser(res);
  res.redirect("/");
});
```

# Primjer za Cookie (2)

05-cookie-auth.js

```
app.get('/private', function (req, res) {  
  if (req.user.isAuthenticated) {  
    if (req.user.username.toLowerCase() === 'alice' || ...  
      res.render('private', {username : req.user.username});  
    }  
    else {  
      res.status(403);  
      res.end('Forbidden for ' + req.user.username);  
    }  
  }  
  else  
    res.redirect(302, 'login?returnUrl=/private');  
});
```

Korisnika koji nije prijavljen (a trebao bi biti) šaljemo na stranicu za prijavu šaljući povratnu informaciju gdje se vratiti nakon uspješne prijave

- Nakon prijave korisnika vraćamo na adresu zapisanu u *returnUrl*.
  - Iz sigurnosnih razloga trebalo bi preusmjeravati samo na lokalne adrese kako bi se izbjeglo npr. login?returnUrl=http://some phishing site koji bi korisnika namamio da ponovo upiše svoje podatke misleći da mu prethodna prijava nije bila uspješna.

# Primjer za Cookie (3)

05-auth-middleware.js

```
const dotenv = require('dotenv');
dotenv.config();
const cookieParser = require('cookie-parser');

function initCookieAuth(app) {
  app.use(cookieParser(process.env.COOKIE_KEY));
}

function signInUser(res, username) {
  res.cookie('user', username, {
    signed : true,
    httpOnly: true
  });
}

function signOutUser(res, username) {
  res.clearCookie('user');
}
```

Middleware za rekonstrukciju korisnika iz *cookiea* i metode za stvaranje i brisanje *cookiea* ostvarene su korištenjem paketa *cookie-parser*

Potpisani kolačić prefiksira vrijednost sa *s*: te kao sufiks stavlja potpis (HMAC + Base64). Korisnik može promijeniti sadržaj *cookiea*, ali potpis više neće biti dobar, jer nema ključ za potpis *cookiea*

*HttpOnly* onemogućava pristup kolačiću iz JavaScripta

# Primjer za Cookie (4)

05-auth-middleware.js

```
function initCookieAuth(app) {  
    app.use(cookieParser(process.env.COOKIE_KEY));  
}  
  
function getUserFromCookie(req, res, next) {  
    const username = req.signedCookies?.user;  
    if (username) {  
        req.user = {  
            isAuthenticated : true,  
            username  
        };  
    }  
    else {  
        req.user = {  
            isAuthenticated : false  
        };  
    }  
    next();  
}
```

Middleware za  
rekonstrukciju korisnika  
iz cookiea. Oslanjamo  
se na *cookie-parser*.

Ključ pod kojim smo  
stavili neku vrijednost  
prilikom prijave  
korisnika.

# Cookie i JavaScript (1)

06-cookie-auth.js

- Što ako se štićenom resursu pristupa iz JavaScripta?

```
...  
app.post('/login', function (req, res) {  
    const username = req.body.username;  
    const password = req.body.password;  
    if (password !== 'some password') {  
        res.status(401).send("Invalid username or password");  
    }  
    else {  
        const payload = {  
            username  
        };  
        auth.signInUser(res, username);  
        res.sendStatus(204);  
    }  
});
```

Forma za prijavu je prebačena u JavaScript, a web-aplikacija (u ovom slučaju) ima ulogu servisa koji provjerava korisnika i postavlja *cookie*

Odjava se svodi na uklanjanje *cookiea* uz status 204 (No Content)

```
app.post('/logout', function (req, res) {  
    auth.signOutUser(res);  
    res.sendStatus(204);  
});
```

# Cookie i JavaScript (2)

06-cookie-auth-js/public/js/site.js

```
function login() {  
    ...  
    const username = document.getElementById('username')  
        .value;  
    const password = document.getElementById('password')  
        .value;  
    return axios.post('login', {username, password})  
        .then(res => ...  
  
function getdata() {  
    ...  
    axios.get('protected')  
        .then(res => ...
```

Cookie se automatski šalje. Primjetiti da u SPA dijelu nema eksplicitnog rukovanja *cookie'm* niti bi to bilo moguće, jer je *http-only*

Obratiti pažnju da je JavaScript unutar iste aplikacije (poslužen iscrtavanjem index.pug)

```
e-auth-js > views > index.pug  
html  
head  
style  
    include ../public/css/login.css  
title Simple js for get data  
body  
- var loginFormStyle = authenticated ? 'none' :  
- var logoutButtonStyle = authenticated ? 'block' : 'none'  
div(class="login-form", style="display:" + loginFormStyle)  
    h1 Login and get data  
    form(action="", method="post")  
        label(for="username") Username:  
        input(type="text", id="username", name="username")  
        label(for="password") Password:  
        input(type="password", id="password", name="password")  
        input(type="button", id="btnLogin", value="Login")  
div(class="content-form")  
    input(type="button", id="btnLogout", value="Logout")  
    input(type="button", id="btnGetData", value="Get Data")  
div(id="content")  
script(src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js")  
script  
    include ../public/js/site.js
```

```
06-cookie-auth-js  
|__ public  
|   |__ css  
|   |__ js  
|       |__ site.js  
|__ views  
|   |__ index.pug  
|       |__ 06-auth-middleware.js  
|       |__ 06-cookie-auth.js
```

```
9  auth.initCookieAuth(app);  
10 app.use(auth.getUserFromCookie);  
11  
12 app.get('/', function (req, res) {  
13     res.render('index', {  
14         user : req.user  
15     });  
16 });  
17  
18 app.post('/login', function (req, res) {  
19     const username = req.body.username;
```

# Cookie i JavaScript (3)

07-cookie-auth-js-cors/public/js/site.js

```
function login() {  
    ...  
    const username = ...  
    return axios.post(`/${serverUri}/login`,  
        {username, password},  
        {withCredentials : true})  
        .then(res => ...  
  
function getdata() {  
    axios.get(`/${serverUri}/protected`, {withCredentials : true})  
    .then(...  
}  
}
```

U ovom primjeru je JavaScript kod u zasebnoj aplikaciji (**ali i dalje na istom serveru/domeni**). Da bi axios poslao *cookie* (i mogao spremiti *cookie* koji dobije u odgovoru) potrebno je postaviti `withCredentials` na `true`  
<https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/withCredentials>

```
var cors = require('cors');  
app.use(cors({credentials: true, origin: 'http://127.0.0.1:4072'}));
```

Web-aplikacija u kojoj nastaje cookie mora omogućiti CORS, ali to više ne može biti bez ograničenja

07-cookie-auth-js-cors.js

# Podsjetnik: Koliko traje cookie?

- *Cookie* kreiran u prethodnom primjeru traje do isteka sjednice (obično završava zatvaranjem preglednika, ali ovisi o pregledniku) ili do isteka eksplicitno postavljenog vremena
- Moguće napraviti trajni *cookie* koji nije vezan za sjednicu
  - Određen atributima *Expires* (apsolutno vrijeme) i *Max-Age* (izražen u sekundama)
    - *Max-age ima prednost u odnosu na Expires*
- Za detaljnije pogledati slajdove iz predmeta *Razvoj programske potpore za web*
  - [https://www.fer.unizg.hr/\\_download/repository/12-DYN-WEB-4-Sjednice.pdf](https://www.fer.unizg.hr/_download/repository/12-DYN-WEB-4-Sjednice.pdf)
  - U nastavku je dan kratki podsjetnik na važnije elemente bitne za kontekst ovih primjera

# Podsjetnik: Kome preglednik šalje *cookie*? (1)

- Atributi *Path* i *Domain*
  - Preglednik serveru šalje *cookie* samo ako URL zahtjeva sadrži *Path* naveden u *cookieu*
    - U prethodnom primjerima *path* je bio postavljan na /
  - Slično vrijedi i za svojstvo *Domain*
    - Ako se izostavi odnosi se točno na server na kojem je kreiran, (ne uključuje poddomene)
    - Ako se navede domena, onda su uključene i pod-domene
      - Npr. ako je *cookie* postavljen na domenu unizg.hr, on će biti poslan prilikom zahtjeva na npr. www.fer.unizg.hr
  - Primjer: Ako je domena postavljena na unizg.hr, a putanja na /projekt, onda će *cookie* biti poslan na npr. www.fer.unizg.hr/projekti, ali ne i na www.unizg.hr/promjene
- Svojstvo *Secure* osigurava da *cookie* neće biti poslan ako veza nije uspostavljena preko https-a

# Podsjetnik: Kome preglednik šalje cookie? (2)

- Zamislimo sljedeću situaciju:
  - U aplikaciji 1 prijavljeni korisnici će na adresi app1.com/photo npr. dobiti svoj avatar
    - Adresa je ista za sve korisnike koji se prepoznaju po *cookieu*
  - Aplikacija 2 na adresi app2.com u html-u ima  
``
  - Što ako se korisnik nije odjavio iz aplikacije 1 prije posjeta aplikaciji 2? Hoće li preglednik prilikom dohvata slike slati korisnikov *cookie* za app1.com?
- Navedeno je primjer *cross-site* zahtjeva jer je izvor zahtjeva app2.com, a resurs na app1.com
  - Podložno CSRF (*Cross-Site Request Forgery*) napadima
  - Što ako se navedena poveznica nalazi negdje unutar app2.com? - Tada je to same-site zahtjev
- Detaljnije u predavanjima o sigurnosti web-aplikacija

# Podsjetnik: Kako se definira Same-site?

- Obično definirano domenom, ali što je ista domena?
- Obično zadnja dva zapisa u adresi
  - Primjerice www.fer.unizg.hr i www.unizg.hr bi spadali u istu domenu unizg.hr
- Pravilo ne može biti globalno primijenjeno
  - Npr. app1.azurewebsites.net i app2.azurewebsites.net ne spadaju u kategoriju *same-site*
  - company1.co.uk i company2.co.uk
- Lista javnih sufiksa: <https://publicsuffix.org>
- Napomena:
  - Domena ≠ firma
    - Firma može imati više domena i u tom slučaju radi se o cross-site zahtjevima

# Podsjetnik: Moguće vrijednosti za Same-site

- Strict
  - Preglednik šalje *cookie* ako su zadovoljeni uvjeti domene i putanje te zahtjev ne dolazi sa stranice iz neke druge domene
    - npr. preglednik neće poslati *cookie* ako je na stranici app2.com korisnik kliknuo na link koji ga vodi na app1.com
    - neće biti podržani ni scenariji poveznice na sliku, javascript koda itd...
- Lax
  - Za razliku od Strict slat će cookie ako se radi o GET zahtjevu koji mijenja stranicu u pregledniku
    - Npr. klik na app1.com u pregledniku vodi korisnika na app1 i tada se šalje cookie
    - ali ako je poveznica na app1.com unutar *iframe* ili se radilo o *ajax* pozivu ili npr. dohvatu slike, cookie neće biti poslan
- None
  - Nema limita i preglednik uvijek šalje cookie prilikom zahtjeva
- Što ako nije ništa navedeno?

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite>

# Prijave korištenjem vanjskih servisa

- Što ako želimo prijavu korisnika odraditi putem nekog od vanjskih servisa?
  - AAI, Google, Microsoft, Twitter, Facebook, Github, Okta, Auth0, Identity Server, ...
  - Umjesto vlastite stranice za prijavu, aplikacija nas preusmjerava na vanjski servis gdje obavljamo prijavu, a aplikacija dobije ili dohvati „povratnu informaciju“ o tome
    - tehnički detalji ovise o protokolu i vrsti aplikacije
- Omogućava koncept jedinstvene prijave (engl. Single Sign On - SSO)
  - Korisnik koristi više aplikacija u koje se treba prijaviti vanjskim računom, ali korisničke podatke unosi samo jednom
    - Svakim sljedećim preusmjeravanjem na vanjski servis korisnik će biti prepoznat po *cookieu*

# Edgar kao primjer korištenja vanjske usluge za autentifikaciju



**FER**

Please, login here using AAI.

**AAI@EduHr**

Autentikacijska i autorizacijska infrastruktura znanosti i visokog obrazovanja u Republici Hrvatskoj

KORISNIČKA OZNAKA

ZAPORKA

PRIJAVA

- Edgar ima vlastitu bazu korisnika, ali ne i lozinki
  - Korisnik se odredi temeljem jedinstvenog identifikatora s AAI-a ili Googlea
  - Ako je korisnik već ranije u istom pregledniku bio prijavljen na AAI, odnosno Google, srednji korak se preskače
    - (Single sign on)

**!FER**

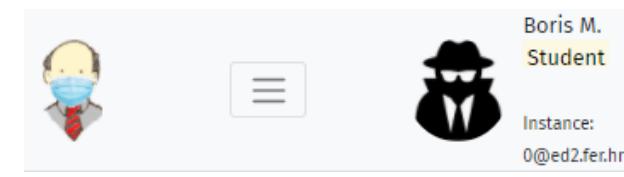
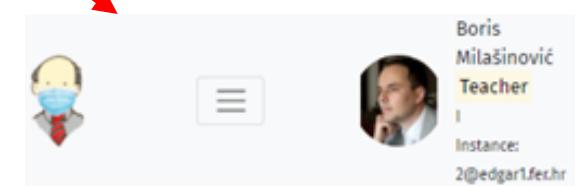
**G Login with Google**

**G Prijavite se putem Googlea**

**Prijava**

Nastavi do aplikacije [fer.hr](#)

E-pošta ili telefon \_\_\_\_\_

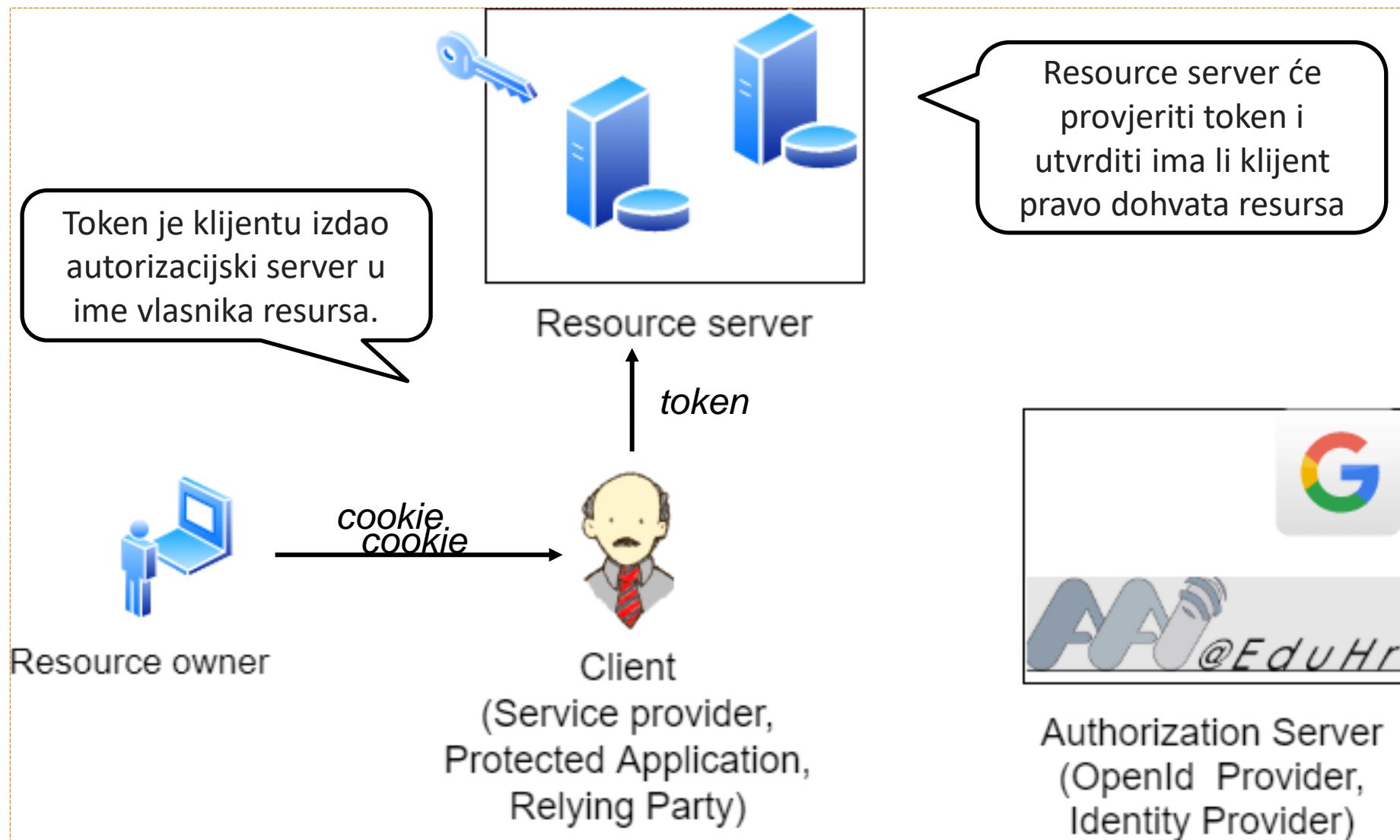


# Tko/što je tko/što na primjeru Edgara

# Šira slika

- U slučaju Edgara, AAI ili Google se koriste za prijavu kako bi se identificirao korisnik i povezao s ostalim podacima
  - **Edgar postavi svoj *cookie***
    - Što bi se dogodilo ako obrišemo Edgarov cookie?
      - Korisniku je ostao *cookie* za AAI-a ili Google (Single Sign On)
      - Što se događa kad se korisnik odjavljuje s Edgara? (Single Sign Out)
- U nekom širem kontekstu Edgar bi mogao dohvaćati podatke koje korisnik ima pohranjene negdje drugdje
  - Hipotetski to bi npr. mogla biti prijava ispita na ISVU ili generiranje karte za brucošijadu
    - Primijetiti da je u jednom slučaju bitan identitet, a u drugom samo pravo pristupa
  - *Resource Server* – usluga (API) koja klijentu (aplikaciji) pruža korisnikove podatke ili podatke na koje korisnik ima pravo
    - **Za provjeru autentičnosti se šalje token**

# Tko je tko u (hipotetskom) širem primjeru



# Protokoli OAuth2 i OpenID Connect

- Komplementarni protokoli
  - OAuth2 je autorizacijski protokol koji omogućava aplikacijama pristup drugim aplikacijama bez dijeljenja vjerodajnica
  - **OpenID Connect (OIDC) je nadgradnja na OAuth2** koja omogućava provjeru i dobivanje informacija o autoriziranom korisniku
  - Načelno, OAuth2 = „evo popis što smijem”, OIDC = „ja sam ...”
- Aplikacija mora biti evidentirana na (OAuth2/OIDC) autorizacijskom serveru
  - Obično ClientId + ClientSecret + popis dopuštenih povratnih adresa nakon prijave
  - tokovi razmjene podataka ovisno o vrsti klijenta i mogućnosti čuvanje tajnog ključa i (ne)postojanja interaktivnog korisnika
    - klasična web-aplikacija ili SPA, mobilna, servis...
      - Više o tokovima naknadno

# Vrste tokena u OAuth2/OIDC

- *id\_token*
  - identifikacijski token (engl. *identity token*) u JWT formatu
  - sadrži jedinstveni identifikator korisnika (*sub*) i informaciju kad i kako je korisnik autentificiran te do kad token vrijedi
  - Podaci unutar tokena nazivaju se tvrdnje (engl. *claims*)  
[https://openid.net/specs/openid-connect-core-1\\_0.html#StandardClaims](https://openid.net/specs/openid-connect-core-1_0.html#StandardClaims)
    - Ako pojedine tvrdnje nisu dio tokena, mogu se dobiti preko odgovarajuće adrese na IdP-u (obično /userinfo )
- pristupni token (*access token*)  
<https://datatracker.ietf.org/doc/html/rfc6749>
  - može biti JWT, ali i proizvoljnog formata
  - obično kratkog trajanja (npr. za AAI 1h)
- token za osvježavanje pristupnog tokena (*refresh token*)
  - omogućava novi pristupni token bez ponovne autentifikacije
    - Npr. za AAI traje 8h

# Opseg (Scope)

- Opseg kod protokola OAuth2 omogućava granulaciju dozvola pojedinoj aplikaciji
  - koristi se u kombinaciji s dozvolama na *Resource serveru*
  - odobreni opsezi zapisani u pristupnom tokenu
- U slučaju OIDC-a i identifikacijskog tokena, opseg služi za određivanje grupa podataka o korisniku koje će se upisati u identifikacijski token ili biti dostupne preko odgovarajućeg servisa
  - Podaci korisnika (engl. *claims*) se grupiraju po opsezima  
[https://openid.net/specs/openid-connect-basic-1\\_0.html#Scopes](https://openid.net/specs/openid-connect-basic-1_0.html#Scopes)
- [https://wiki.srce.hr/pages/viewpage.action?pageId=59867172#Autentikacijapomo%C4%87uprotokolaOpenIDConnect\(OIDC\)-Opseziitvrđnje\(eng.ScopesandClaims\)](https://wiki.srce.hr/pages/viewpage.action?pageId=59867172#Autentikacijapomo%C4%87uprotokolaOpenIDConnect(OIDC)-Opseziitvrđnje(eng.ScopesandClaims))
- Napomena: Aplikacija prilikom kontakta s autorizacijskim serverom navodi koje opsege treba te oni moraju biti u skupu opsega dopuštenih toj aplikaciji

# Auth0 kao primjer vanjske usluge za prijavu korisnika

- <https://auth0.com/>
  - OAuth2 + OpenID Connect
- Prijava korisnika
  - temeljem podataka iz baze podataka
  - vanjskim servisima (npr. Google)
  - različitim varijantama koje ne uključuju lozinke (npr. SMS, e-mail) ili koriste autentifikaciju s više načina (engl. *multi-factor authentication*)
- Besplatni plan uključuje 7000 aktivnih korisnika, neograničeni broj prijava i mogućnost prijave putem dvije društvene mreže
- Potrebni koraci
  - Prijaviti se na Auth0 i kreirati odgovarajuće okruženje
  - Evidencirati klijentske aplikacije

# Primjer s Auth0 iz perspektive korisnika

- Pristup stranici za prijavljene korisnike okida proces prijave preko servisa Auth0

The screenshot shows two tabs in a browser:

- The top tab is titled "OAuth2/OpenID login demo" and displays a menu with links: "Home", "Private content", and "Sign up".
- The bottom tab is also titled "OAuth2/OpenID login demo" and shows the URL "localhost:4080/private".

A red arrow points from the "Private content" link in the top tab to the URL in the bottom tab, indicating that the user has access to private content without logging in.

This page is only for logged users

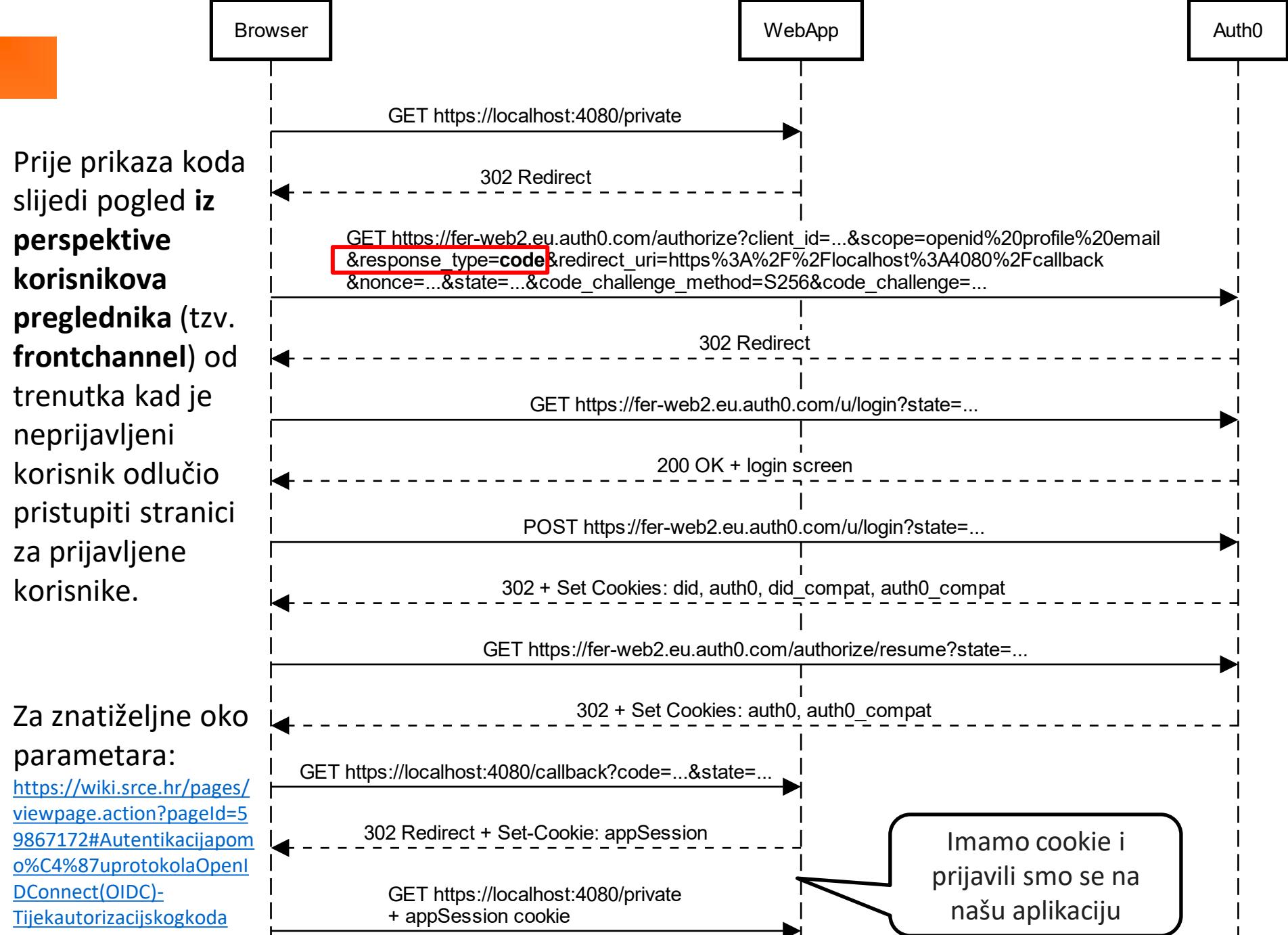
Your data is: {"nickname": "boris.milasinovic", "name": "Boris Milašinović", "picture": "https://s.gravatar.com/avatar/2a9e3b993a61s=480&r=pg&d=https%3A%2F%2Fcdn.auth0.com%2Favatars%2F18T09:27:02.318Z", "email": "boris.milasinovic@fer.hr", "email\_ver

The screenshot shows a browser window with the title "Log in to FER-Web2 WebApp". The URL in the address bar is "fer-web2.eu.auth0.com/...".

The page content includes:

- A red star logo.
- The word "Welcome".
- A message: "Log in to fer-web2 to continue to FER-Web2 WebApp."
- An "Email address" input field containing "boris.milasinovic@fer.hr".
- An "Password" input field with masked text and a visibility icon.
- A "Forgot password?" link.
- A large blue "Continue" button.
- A link "Don't have an account? Sign up".

A red arrow points from the URL in the top-left tab of the first screenshot to the "Email address" field in this screenshot, illustrating the single sign-on process.



# Evidentiranje aplikacije na usluzi Auth0 (1)

- U izborniku s aplikacija kreirati novu aplikaciju tipa *Regular Web Applications*

Create application

Name \*

FER-Web2 WebApp

You can change the application name later in the application settings.

Choose an application type



Native

Mobile, desktop,  
CLI and smart  
device apps  
running natively.

e.g.: iOS,  
Electron, Apple  
TV apps



Single Page Web  
Applications

A JavaScript  
front-end app  
that uses an API.

e.g.: Angular,  
React, Vue



Regular Web  
Applications

Traditional web  
app using  
redirects.

e.g.: Node.js  
Express,  
ASP.NET, Java,  
PHP



Machine to  
Machine  
Applications

CLIs, daemons or  
services running  
on your backend.

e.g.: Shell script

# Evidentiranje aplikacije na usluzi Auth0 (2)

- U našoj aplikaciji moramo evidentirati *Domain* i *ClientId*
  - *ClientSecret* nam treba ovisno o tipu autorizacijskog toka
- Dodatno, potrebno je postaviti sljedeće:
  - Allowed Callback URLs: `https://localhost:port/callback`
  - Allowed Logout URLs: `https://localhost:port`
  - *Napomena: localhost u slučaju lokalnog razvoja, inače adresa servera*
- Korisnike možemo dodati u izborniku User Management → Users
  - Može se omogućiti i samostalna registracija korisnika (sign-up) iz aplikacije

The screenshot shows the 'Settings' tab of an application configuration page. The application is named 'FER-Web2 WebApp', which is identified as a 'Regular Web Application'. The 'Client ID' is listed but redacted. Below the main title, there are tabs for 'Quick Start', 'Settings' (which is active), 'Addons', 'Connections', and 'Organizations'. A large central panel is titled 'Basic Information'. It contains fields for 'Name \*' (set to 'FER-Web2 WebApp') and 'Domain' (set to 'fer-web2.eu.auth0.com').

# Povezivanje web-aplikacije s Auth0 (1)

- Iako bi teoretski trebalo raditi i ako koristimo http, primjer nije mogao raditi bez https veze, za što je potreban certifikat
  - Za lokalni razvoj napravimo vlastiti (inače npr. <https://letsencrypt.org>)
    - preglednik će nas upozoravati na neispravni certifikat

```
openssl req -nodes -new -x509 -keyout server.key -out server.cert
```

```
const express = require('express');
var fs = require('fs')
var https = require('https')
const app = express();
...
const port = 4080;
https.createServer({
    key: fs.readFileSync('server.key'),
    cert: fs.readFileSync('server.cert')
}, app)
.listen(port, function () {
    console.log(`Server running at https://localhost:${port}`);
});
```

08-oidc/08-webapp.js



Vaša veza nije privatna

Napadači možda pokušavaju ukrasti vaše zaporke, poruke ili brojeve kreditnih kartica

NET::ERR\_CERT\_AUTHORITY\_INVALID

Sakrij napredno

Poslužitelj nije mogao dokazati da je **lokalni** njegov sigurnosni certifikat nije pouzdan. Konfiguracijom ili napadom na vašu vezu.

[Idi na web-lokaciju localhost \(nije sigurno\)](#)

# Povezivanje web-aplikacije s Auth0 (2)

```
const { auth, requiresAuth } = require('express-openid-connect');  
  
const config = {  
    authRequired : false,  
    idpLogout : true, //login not only from the app, but also from identity provider  
    secret: process.env.SECRET,  
    baseURL: `https://localhost:${port}`,  
    clientID: process.env.CLIENT_ID,  
    issuerBaseURL: 'https://fer-web2.eu.auth0.com'  
    clientSecret: process.env.CLIENT_SECRET,  
    authorizationParams: {  
        response_type: 'code' ,  
        //scope: "openid profile email"  
    },  
};  
  
app.use(auth(config));
```

08-oidc/08-webapp.js

Paket za povezivanje  
s OpenID uslugom

Proizvoljni ključ s kojim  
će biti potpisano cookie.

Identifikator naše aplikacije na  
Auth0 i tajni ključ aplikacije

Okruženje koje smo kreirali na Auth0  
– izdavatelja tokena, tj. identity  
provider

Tri prepostavljena. Mogu  
se dodati i drugi, ali  
openid je obvezan

Vrsta autorizacijskog koda,  
u ovom slučaju  
*Authorization Code Flow*

Uključivanje ovog *middleware*  
automatski se definira ponašanje  
za putanje login, logout i callback

CLIENT\_ID=prepisati s Auth0  
SECRET=proizvoljno  
CLIENT\_SECRET=prepisati s Auth0

08-oauth2 / .env

# Samostalna registracija novih korisnika

08-oidc / 08-webapp.js

```
app.get("/sign-up", (req, res) => {
  res.oidc.login({
    returnTo: '/',
    authorizationParams: { screen_hint: "signup" },
  });
});

app.get('/', function (req, res) {
  req.user = {
    isAuthenticated : req.oidc.isAuthenticated()
  };
  if (req.user.isAuthenticated)
    req.user.name = req.oidc.user.name;
  res.render('index', {user : req.user});
});

app.get('/private', requiresAuth(), function (req, res) {
  const user = JSON.stringify(req.oidc.user);
  res.render('private', {user});
});
```

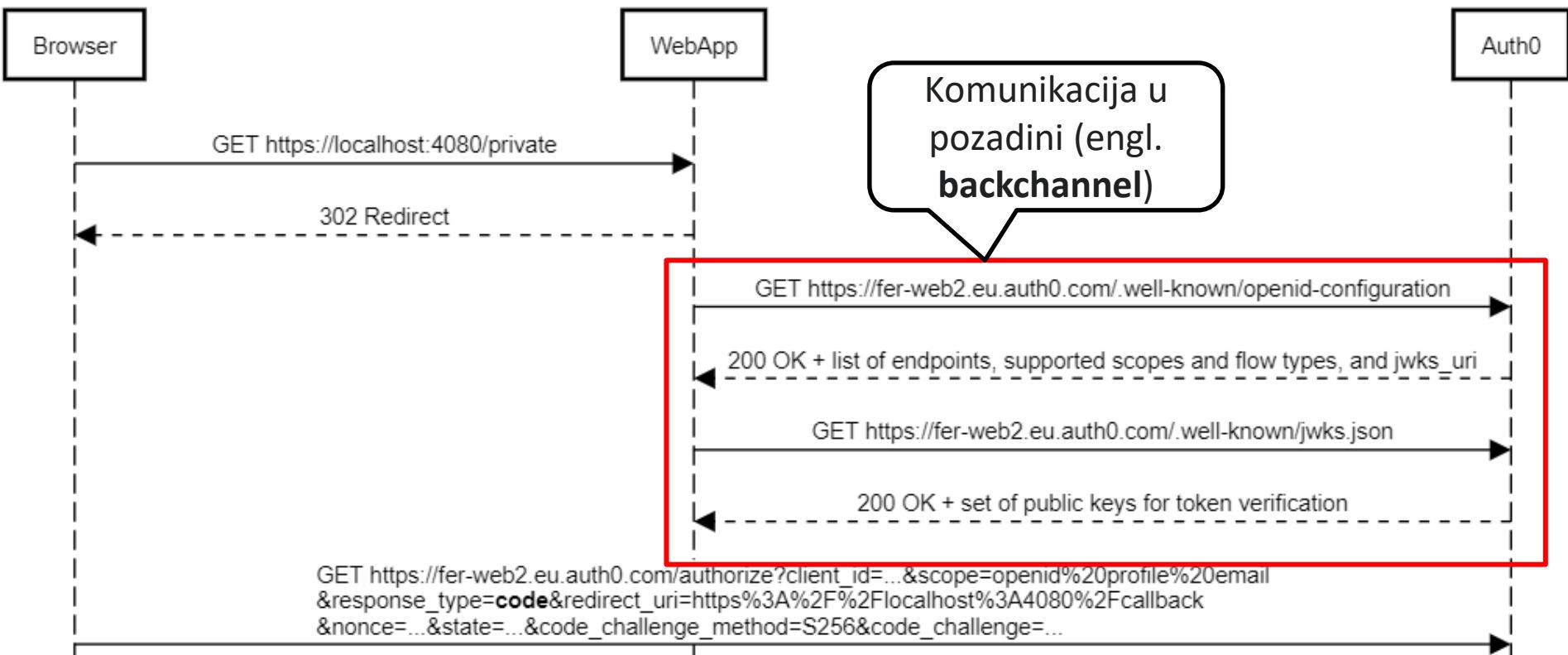
Omogućimo samostalnu registraciju korisnika

Je li korisnik prijavljen?

Tko je prijavljeni korisnik?

# Dio koji korisnik nije vido (1)

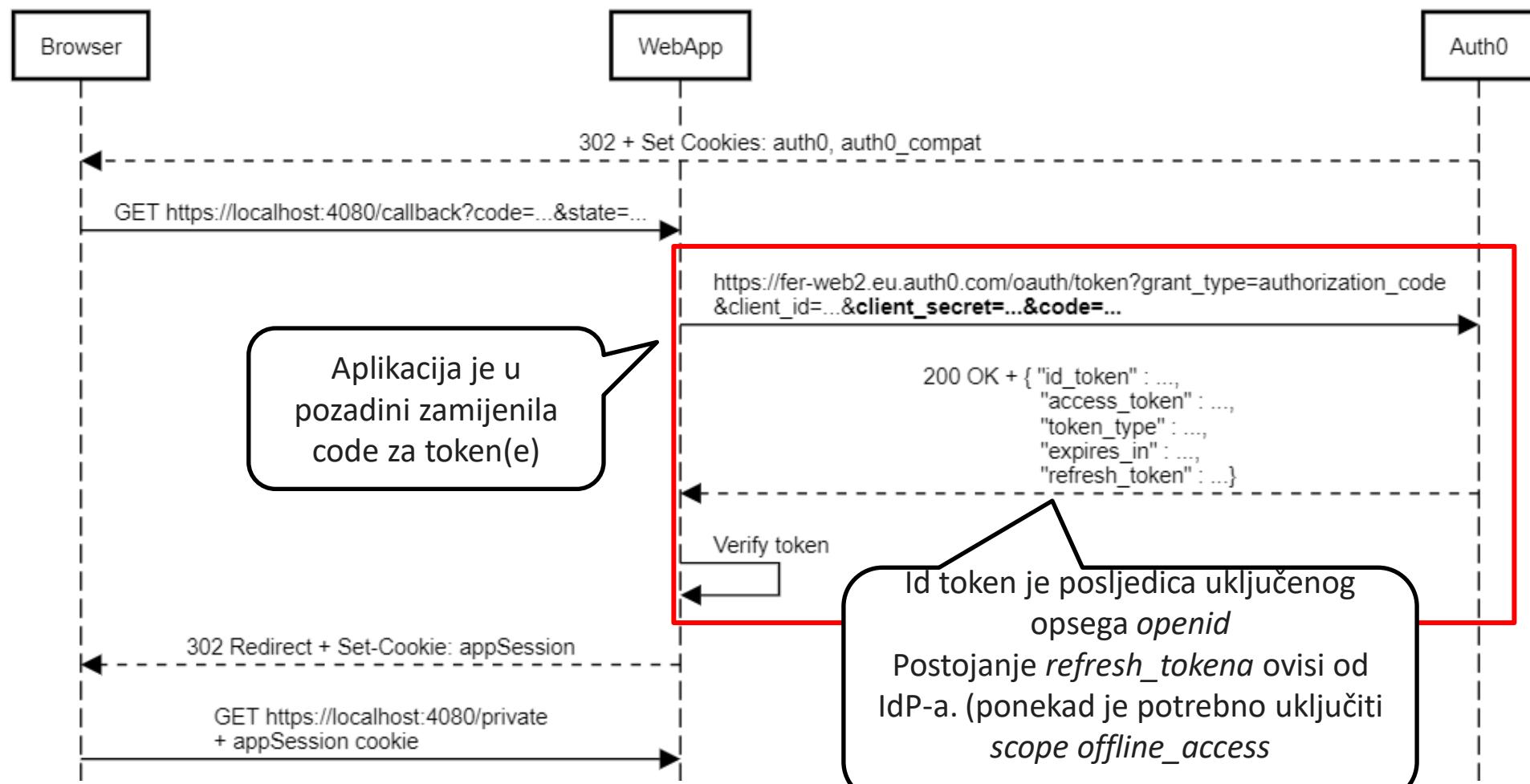
- Prije preusmjeravanja na stranicu za prijavu korisnika, aplikacija je dohvatala javne podatke za provjeru tokena



- `/.well-known/openid-configuration` je tzv. *discovery* adresa kojom neki OIDC server pruža informacije o dopuštenim opsezima i vrstama provjere te adresama servisa za dohvat javnih ključeva i o korisnicima
  - <https://fer-web2.eu.auth0.com/.well-known/openid-configuration>

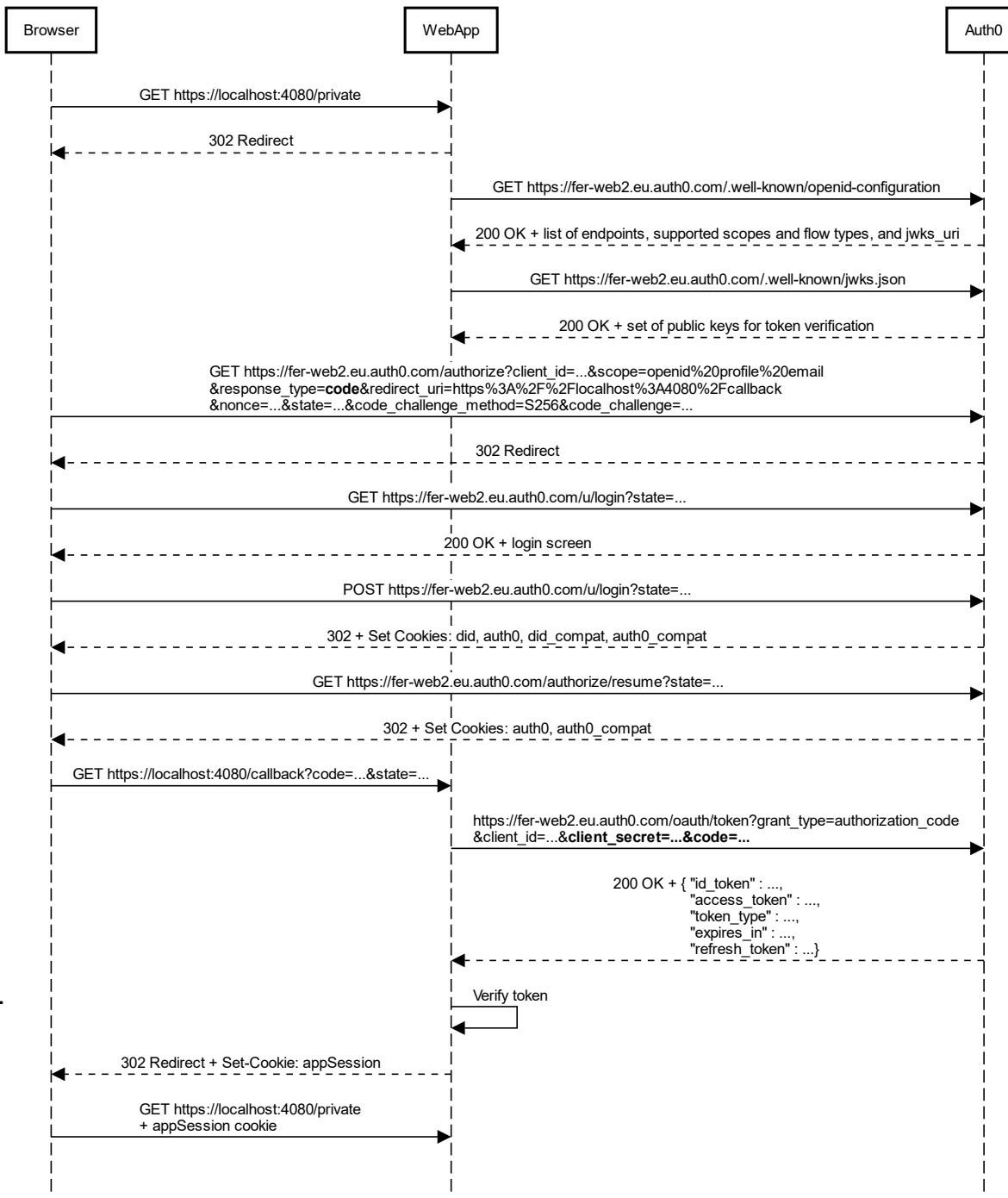
# Dio koji korisnik nije vido (2)

- Nakon što je dobila autorizacijski kod web-aplikacija je u pozadini zamijenila autorizacijski kod za identifikacijski i pristupni token



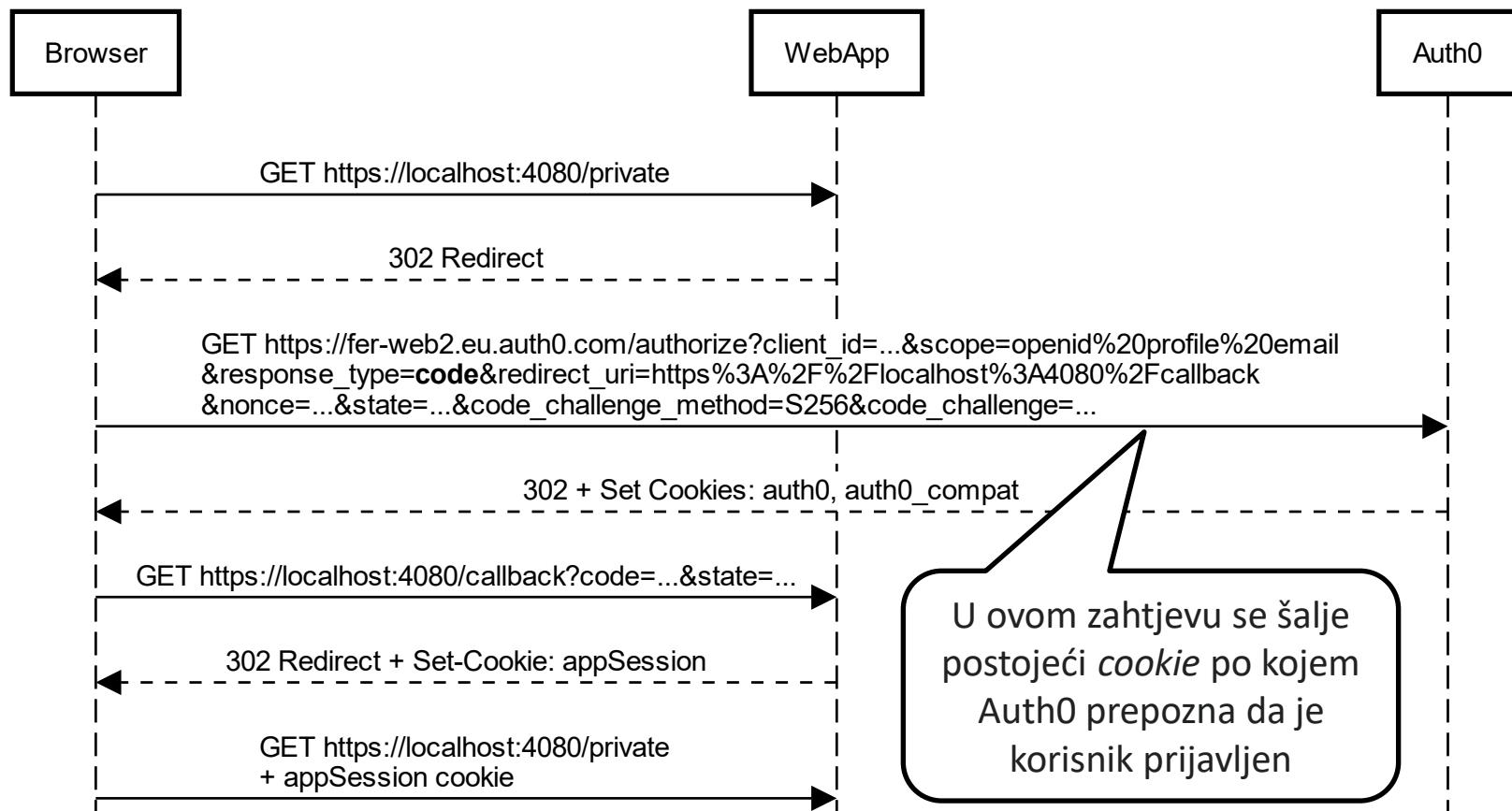
# Kompletni tok

- Navedeni tok se naziva *Authorization Code Flow*
  - prepoznajemo po `response_type=code`
  - frontchannel + backchannel
- Klijent ima id i secret i može pohraniti secret na sigurno mjesto
- Token nikad ne ide kroz preglednik
- Što ako aplikacija ne može osigurati tajnost ključa?
  - Koristiti neki drugi autorizacijski način ili Authorization Code Flow + PKCE
    - (vidi slajdove na kraju predavanja)



# Single Sign On

- Što ako je korisnik već prijavljen na Auth0, npr. zbog neke druge aplikacije?
  - Možemo simulirati prijavom na našu aplikaciju i brisanje cookiea
    - Logout nije opcija, jer napravi logout s Auth0



# OAuth2/OIDC na primjeru SPA i web-servisa

- Nalik jednom od prethodnih primjera (04-token-auth-cors)
  - SPA, tj. JavaScript kod smješten u nekoj drugoj aplikaciji treba obaviti prijavu kako bi dobio pristupni token s kojim može pozvati zaštićeni resurs na serveru (web-servisu)
- Ključna razlika je što token (za razliku od prethodnog primjera s tokenom) ne izdaje web-servis, nego vanjska usluga (Auth0 ili neki drugi OAuth2/OIDC pružatelj usluge)
  - Na SPA moramo omogućiti prijavu
  - Na web-servisu moramo uspostaviti mehanizam provjere tokena

# Auth0 i WebApi (1)

- Na Auth0 definiramo API, pri čemu su bitna 3 parametra
  - API Audience
    - Klijenti koji žele pristupiti našem API-u navode audience i web-api provjerava je li on prisutan u tokenu
    - Allow Skipping User Consent
      - potrebno zbog specifične klijentske biblioteke
      - <https://auth0.github.io/auth0-spa-js/classes/auth0client.html#gettokensilently>
    - Allow Offline Access
      - Definira može li se koristit refresh token. U kontekstu ovih predavanja je nebitno, ali može biti praktično ako se koristi neka druga biblioteka

The screenshot shows the Auth0 API Management interface. On the left, there's a sidebar with navigation links: 'Getting Started', 'Activity', 'Applications' (selected), 'APIs' (selected), and 'SSO Integrations'. The main content area has a title 'APIs' and a sub-instruction 'Define APIs that you can consume from your authorized applications.' Below this, there's a card for 'FER-Web2 WebAPI' with a 'Custom API' button. At the bottom right, it says 'API Audience: FER-Web2 WebAPI'.

# Auth0 i WebApi (2)

09-ouath2-spa-and-web-api/09-webapi.js

```
const iss = 'https://fer-web2.eu.auth0.com';
var jwtCheck = jwt({
  secret: jwks.expressJwtSecret({
    cache: true,
    rateLimit: true,
    jwksRequestsPerMinute: 5,
    jwksUri: `${iss}/.well-known/jwks.json`
  },
  audience: `FER-Web2 WebAPI`,
  issuer: `${iss}/`,
  algorithms: ['RS256']
});

const hostname = '127.0.0.1';
const port = 4091;
app.listen(port, hostname, () => {
  console.log(`Web API running at http://${hostname}:${port}/`);
});
```

Middleware koji ćemo provjeriti ispravnost tokena tako da će u pozadini dohvatiti skup javnih ključeva s navedene adrese

Primijetiti da ovdje trebamo jedino podatke o izdavatelju (iss) i namjeni (audience)

# Auth0 i WebApi (3)

09-ouath2-spa-and-web-api/09-webapi.js

```
app.get('/protected', jwtCheck, async function (req, res) {  
    let user = req.user;  
    const bearer = req.headers.authorization;  
  
    U ovom trenutku samo znamo  
    da pozivatelj ima pravo  
    pristupa (autoriziran je preko  
    OAuth2), ali ne znamo tko je  
  
    try{  
        const res = await axios.post(`${iss}/userinfo`,  
            {}, {headers : {  
                Authorization : bearer  
            }});  
        user = res.data;  
    }  
    catch(err) {  
        console.log(error);  
    }  
    res.json(JSON.stringify(user));  
});
```

Aktiviramo middleware za provjeru tokena

Sadržaj pristupnog tokena

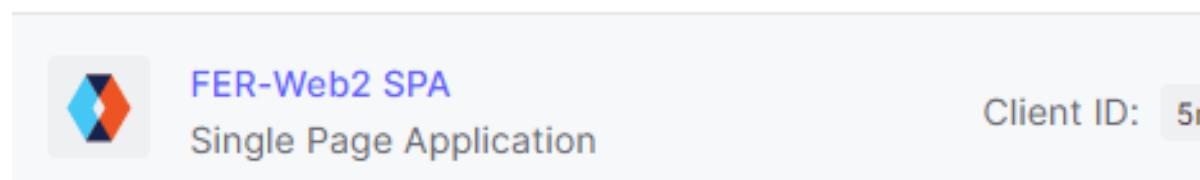
"iss": "https://fer-web2.eu.auth0.com/",  
"sub": "auth0|60f946b6da00c900682c6ec3",  
"aud": [  
 "FER-Web2 WebAPI",  
 "https://fer-web2.eu.auth0.com/userinfo"  
],  
"iat": 1627157622,  
"exp": 1627244022,  
"azp": "5nfNZH9Lf0DKU2hrmCuXe4CEfs8iI9JU",  
"scope": "openid profile email"

Informacije o korisniku

"sub": "auth0|60f946b6da00c900682c6ec3",  
"nickname": "boris.milasinovic",  
"name": "Boris Milašinović",  
"picture": "https://s.gravatar.com/avatar/auth0.com%2Favatars%2Fbo.png",  
"updated\_at": "2021-07-24T20:13:41.762Z",  
"email": "boris.milasinovic@fer.hr",  
"email\_verified": true

# Auth0 i SPA (1)

- Na Auth0 definiramo novu aplikaciju (tipa Single Page Application), pri čemu su osim *ClientId* bitna 3 parametra
  - Allowed Callback, Allowed Logout i Allowed Web Origins
  - Sva tri oblika https://server:port
    - U primjeru s predavanja https://localhost:4092
  - Napomena: **Ovaj tip aplikacije ne koristi Client Secret, jer se smatra da ga ne može pospremiti na siguran način**



# Auth0 i SPA (2)

09-ouath2-spa-and-web-api / 09-spa.js

```
app.get('/', function (req, res) {  
    res.render('index-spa');  
});  
  
app.get("/auth_config.json", (req, res) => {  
    res.json({  
        "domain": "fer-web2.eu.auth0.com",  
        "clientId": process.env.SPA_CLIENT_ID,  
        "audience" : 'FER-Web2 WebAPI'  
    });  
});  
  
const port = 4092;  
https.createServer({  
    key: fs.readFileSync('server.key'),  
    cert: fs.readFileSync('server.cert')  
}, app)  
.listen(port, function () {  
    console.log(`SPA running at https://localhost:${port}/`);  
});
```

Putanja koja vraća json s postavkama za OAuth2 (poziva se prilikom inicijalizacije SPA – vidi sljedeći slajd)

Audience ovdje predstavlja željenog krajnjeg primatelja tokena. Tipično je to adresa resursa, ali prilikom definiranja webapi-a na Auth0 smo tako nazvali

# Auth0 i SPA (3)

09-... / public / js / site.js

```
let auth0 = null;  
const fetchAuthConfig = () => fetch("/auth_config.json");  
const configureClient = async () => {  
    const response = await fetchAuthConfig();  
    const config = await response.json();  
    auth0 = await createAuth0Client({  
        domain: config.domain,  
        client_id: config.clientId,  
        audience: config.audience  
    });  
};  
const login = async () => {  
    await auth0.loginWithRedirect({  
        redirect_uri: window.location.origin  
    });  
};  
const logout = async () => {  
    await auth0.logout({ returnTo: window.location.origin });  
};
```

Koristimo auth0-spa-js  
<https://github.com/auth0/auth0-spa-js>

Kod za prijavu i odjavu

# Auth0 i SPA (4)

```
const serverUri = "http://127.0.0.1:4091";  
  
const getdata = async () => {  
    try {  
        // Get the access token from the Auth0 client  
        const token = await auth0.getTokenSilently();  
  
        // Make the call to the API, setting the token  
        // in the Authorization header  
        const response = await fetch(` ${serverUri}/protected` , {  
            headers: {  
                Authorization: `Bearer ${token}`  
            }  
        });  
    }  
};  
...  
09-... / public / js / site.js
```

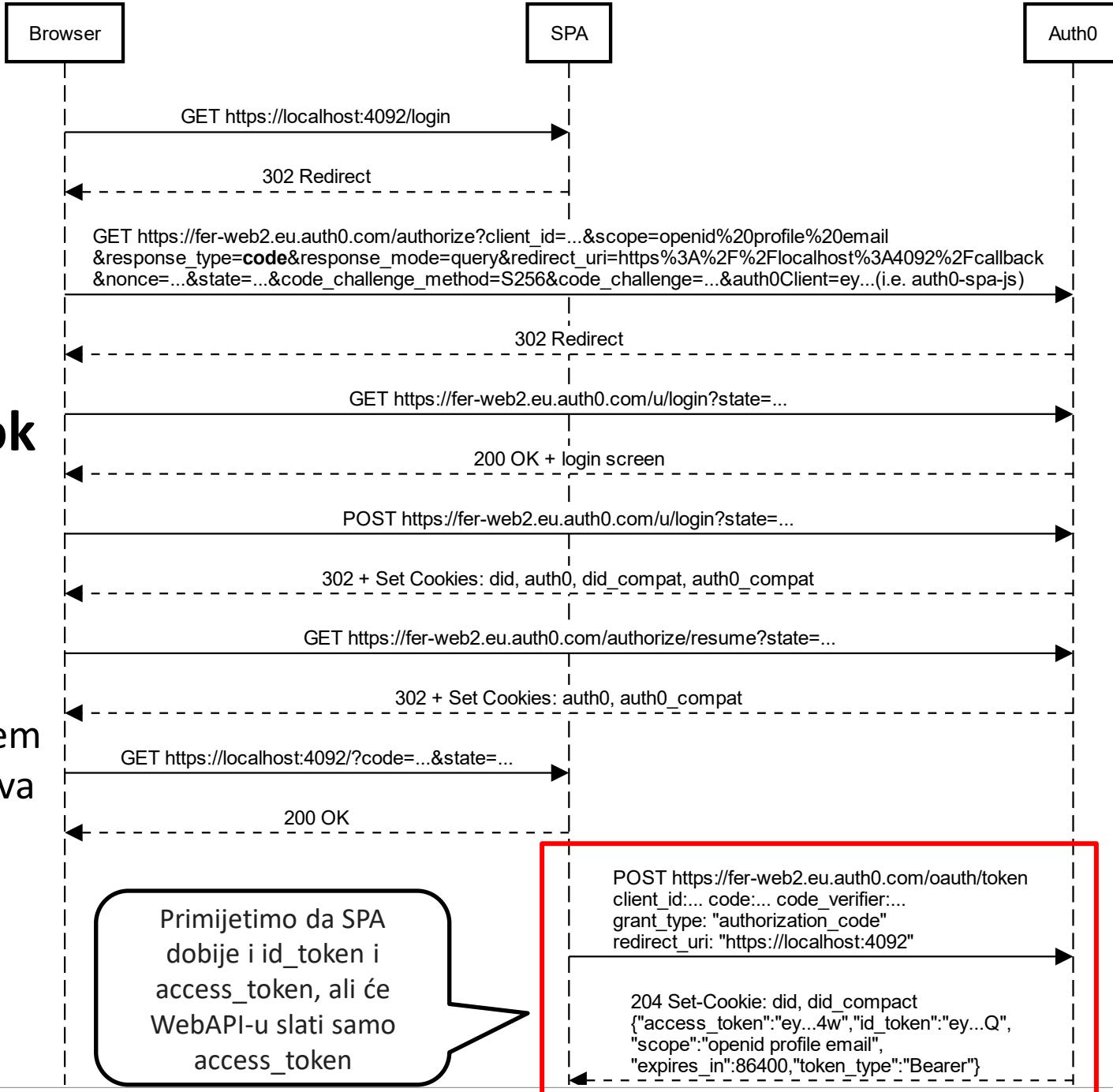
Upotrijebi postojeći token ili u pozadini zatraži novi.  
Više na <https://auth0.github.io/auth0-spa-js/classes/auth0client.html#gettokensilently>

Pristupni token

Napomena: U samoj datoteci nalazi se još dio koda koji prihvata preusmjeravanje nakon logina.  
Definiran je pod `window.onload` =  
i treba ga upotrijebiti „as-is”

# Kako je išao tok u slučaju SPA?

Napomena: S dijagrama je izostavljen dio u kojem SPA ima token i poziva webapi



# Authorization Code Flow + PKCE

- U oba primjera korišten Authorization Code Flow + PKCE
- Authorization Code Flow je koncept u kojem aplikacija preusmjeri korisnika na autorizacijski server koji nakon uspješne prijave kao povratnu informaciju kroz preglednik vrati kod koji se šalje aplikaciji
  - Aplikacija u pozadini preko POST zahtjeva šaljući kod i svoje podatke (*client id* i *client secret*) zauzvrat dobije token
  - Omogućava provjeru autentičnosti klijenta
  - Problem čuvanja šifre (mobilne aplikacije, SPA, ...)
- PKCE = Proof Key for Code Exchange
  - Novi zahtjev, novi *hash* random stringa
  - Razmjena u pozadini koristi taj hash i/umjesto *client secret*
  - Omogućava provjeru radi li se o istoj aplikaciji koja je započela zahtjev

# Ostale vrste tokova

- *Client Credentials Flow*
  - Nema interaktivnog korisnika, aplikacija dobije token samo temeljem svojih podataka (*client id + client secret*)
    - Dodatni mehanizam zaštite su klijentski certifikati
- *Resource Owner Password Flow*
  - Samo za iznimne slučajeve i potpuno povjerljive aplikacije
  - Korisnik upisuje svoje korisničko ime i lozinku u samu aplikaciju, koja onda to prenosi na autorizacijski server kako bi dobila pristupne tokene
    - Bi li u nečiju aplikaciju ukucali svoje podatke s Googlea?
- *Implicit flow*
  - Korišten za JavaScript aplikacije prije postojanja CORS-a
  - Svodi se na to da se umjesto autorizacijskog koda vraćao pristupni token što se smatra nesigurnim
- *Hybrid flow*
  - Npr. identifikacijski token vraćen kroz preglednik, pristupni u pozadini
  - Parameter *response type* je u tom slučaju *code id\_token*
    - [https://openid.net/specs/openid-connect-core-1\\_0.html#HybridFlowAuth](https://openid.net/specs/openid-connect-core-1_0.html#HybridFlowAuth)
    - <https://www.scottbrady91.com/OpenID-Connect/OpenID-Connect-Flows>

# Neke opaske oko odjave korisnika (za značajne)

- Odjavom iz pojedine aplikacije dolazi do odjave i iz OAuth2/OIDC pružatelja usluge (u ovom slučaju Auth0)
- Dobra praksa bi bila da tada dođe i do odjave iz svih aplikacija koje su izvršili prijavu preko tog pružatelja usluge
  - Po OIDC-u to može biti Front-Channell (GET) i Back-Channell (POST)
    - [https://openid.net/specs/openid-connect-backchannel-1\\_0.html](https://openid.net/specs/openid-connect-backchannel-1_0.html)
    - [https://openid.net/specs/openid-connect-frontchannel-1\\_0.html](https://openid.net/specs/openid-connect-frontchannel-1_0.html)
  - Nažalost, Auth0 ne podržava tako opciju,
    - <https://auth0.com/docs/logout/log-users-out-of-applications>
- a npr. Identity server 4 da
  - <https://docs.identityserver.io/en/latest/topics/signout.html#notifying-clients-that-the-user-has-signed-out>