Vežba 5

U vežbi 5 pravimo *auth servis* i dodajemo login i register funkcionalnosti u vue.js aplikaciju. Buildujemo vue.js aplikaciju i postavljamo je da se servisa sa *app servisa*.

Podsetnik: u vežbi 3 napravili smo dva servisa.
API servis je postavljen na port 9000
APP servis je postavljen na port 8000

Ovde ćemo dodati AUTH servis, na port 9001, i potrebnu tabelu i model za korisnike u bazu
Dodaćemo i potrebne login i register stranice u administratorsku aplikaciju (app servis/static/administrator/) i u korisničku aplikaciju (vue.js).
Dodaćemo potrebnu logiku za autentifikaciju korisnika, čuvanje tokena i slanje tokena servisu.
Dodaćemo potrebnu logiku na strani servisa da na osnovu tokena proveri i spreči akcije korisnika ako nije ulogovan.
Dodaćemo potrebnu logiku u vue.js aplikaciju da na osnovu tokena prikaže/sakrije delove aplikacije koji su nedostupni "gostu".

Auth web servis

Napravićemo novi (treći) node.js express web servis, koji služi za authentifikaciju korisnika i ima rute /register i /login

1. Napravite folder auth_server
2. Sa npm init inicijalizujte projekat
3. Napravite fajl app.js
4. Instalirajte express
5. Instalirajte sequelize - biće nam potrebno da se nakačimo na bazu zbog tabele *users*
    a. Pogledajte vežbu 3 za detalje postavljanja sequelize u projekat
    b. Podesite config/config.json, ovaj servis koristiće istu bazu kao onaj iz vežbe 3
6. Napravite novi model *Users*
    a. Treba da ima atribute
        i. username - string, unique
        ii. password - string
        iii. admin - boolean, true ako jeste, false ako je običan korisnik
        iv. email - string, takođe unique
    b. Evo primera celog modela:

```
Users.init({
    username: {
      type: DataTypes.STRING,
      allowNull: false,
      unique: true
    },
    password: {
      type: DataTypes.STRING,
      allowNull: false
```

```
    },
    admin: {
      type: DataTypes.BOOLEAN,
      allowNull: false,
      defaultValue: false
    },
    email: {
      type: DataTypes.STRING,
      allowNull: false,
      unique: true,
      validate: {
        isEmail: {
          msg: "Nije email"
        }
      }
    }
  }
```

7. Napravite migracioni fajl *users*

```
'use strict';
module.exports = {
  up: async (queryInterface, DataTypes) => {
    await queryInterface.createTable('Users', {
      id: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: DataTypes.INTEGER
      },
      username: {
        type: DataTypes.STRING,
        allowNull: false,
        unique: true
      },
      password: {
        type: DataTypes.STRING,
        allowNull: false
      },
      admin: {
        type: DataTypes.BOOLEAN,
        allowNull: false,
        defaultValue: false
      },
      email: {
```

```
            type: DataTypes.STRING,
            allowNull: false,
            unique: true,
            validate: {
              isEmail: {
                msg: "Nije email"
              }
            }
          },
          createdAt: {
            allowNull: false,
            type: DataTypes.DATE
          },
          updatedAt: {
            allowNull: false,
            type: DataTypes.DATE
          }
        });
      },
      down: async (queryInterface, DataTypes) => {
        await queryInterface.dropTable('Users');
      }
    };
```

   a. Vodite računa o stranim ključevima i relacijama - unutar baze imaćete podatke koji su vezani za korisnika (npr. narudržbina), pa ako to već nista napravili, postavite relacije kako treba

8. instalirajte bcrypt, jsonwebtoken, dotenv sa npm install
   a. Ovo su pomoćne biblioteke koje nam trebaju za hash algoritam za password i za rad za JWT, i za upotrebu .env fajla koji će nam služiti kao konfiguracioni fajl za ovaj servis
9. U glavnom folderu auth servisa napravite fajl .env u koji ćete upisati token secret
   a. Ovaj fajl treba da bude u gitignore
   b. Sa dotenv bibliotekom ovaj fajl biće učitan direktno u nodejs proces, u *process.env*
   c. Možete generisati izvršavanjem sledećeg u konzoli:
      `node -e "console.log(require('crypto').randomBytes(32).toString('hex'));"`
   d. U fajl upišite generisanu random vrednost
      `ACCESS_TOKEN_SECRET=generisani string`
10. Otvorite app.js i dodajte potrebne module, postavite express instancu, podesite CORS i pokrenite express na portu 9001. Napravite i dve post rute: login i register

```
const express = require('express');
const { sequelize, Users } = require('./models');
const bcrypt = require('bcrypt');
```

```
const jwt = require('jsonwebtoken');
const cors = require('cors');
require('dotenv').config();

const app = express();

var corsOptions = {
    origin: 'http://127.0.0.1:8000',
    optionsSuccessStatus: 200
}

app.use(express.json());
app.use(cors(corsOptions));

app.post('/register', (req, res) => {
});

app.post('/login', (req, res) => {
});

app.listen({ port: 9001 }, async () => {
    await sequelize.authenticate();
});
```

11. Ruta /register treba da na osnovu prosleđenog username, password, email, napravi novi model user-a i sačuva ga u bazu. Password ne čuvamo kao plaintext, već ga šifrujemo hash funkcijom (one-way encryption) tako da je uskladišten u nečitljivom obliku. Nakon kreiranja korisnika, generišemo token i vraćamo token kao rezultat, čime efektivno omogućavamo login na klijentskoj aplikaciji (registruje se i odmah je i ulogovan) - da ne komplikujemo sa verifikacijom slanjem mejla itd.

```
app.post('/register', (req, res) => {
    const obj = {
        username: req.body.username,
        email: req.body.email,
        admin: false,
        password: bcrypt.hashSync(req.body.password, 10)
    };
    Users.create(obj).then( rows => {
        const usr = {
            userId: rows.id,
            user: rows.username
        };
        const token = jwt.sign(usr,
process.env.ACCESS_TOKEN_SECRET);
```

```
            //console.log(token);
            res.json({ token: token });
        }).catch( err => res.status(500).json(err) );
});
```

12. Ruta /login treba da na osnovu poslatog username i password pošalje upit u bazu, u tabelu *users* i pronađe red-model koji ima takav username i password. Prvo tražimo po username (koji je unique, pa ako ima biće tačno jedan takav zapis), a zatim proveravamo da li je prosleđeni password jednak onom u bazi. Pošto je kriptovan prilikom zapisa, sada i prilikom poređenja treba da prosleđeni password kriptujemo na isti način, pa da poredimo da li su šifrati jednaki - kao jesu, onda su i originali jednaki, tj. prosleđeni password jeste tačan. Isto kao i u register, nakon uspešne provere generišemo token i šaljemo klijentu.

```
app.post('/login', (req, res) => {
    Users.findOne({ where: { username: req.body.username } })
      .then( usr => {
        if (bcrypt.compareSync(req.body.password, usr.password))
{
            const obj = {
                userId: usr.id,
                user: usr.username
            };
            const token = jwt.sign(obj,

process.env.ACCESS_TOKEN_SECRET);
            res.json({ token: token });
        } else {
            res.status(400).json({ msg: "Invalid credentials"});
        }
      })
      .catch( err => res.status(500).json(err) );
});
```

13. Sada je auth servis završen.
14. Napravite jednog admin korisnika tako što ćete iz thunderclient gađati rutu /register. U request body stavite ovakav sadržaj

```
{
    "username": "admin",
    "email": "admin@test.raf",
    "password": "adminpass"
}
```

15. U bazi, sa phpmyadmin, u tabeli users promenite u koloni admin vrednost sa false na true.
16. Možete i da kopirate ovaj šifrovani password i napravite seeder sa ovim podacima.

App servis - administrator login stranica

Dodaćemo login stranicu u app servis, u okviru aplikacije za administratore

1. Otvorite app_server (iz vežbe 3)
2. U static/administrator/ napravite fajl login.html
3. Otvorite login.html i stavite kod login stranice
    a. Postavite osnovnu bootstrap stranicu i login formu, npr. nešto ovako:

```
<!DOCTYPE html>
<html lang=en>
  <head>
    <title>SJ Vezbe</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css
" rel="stylesheet">
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.mi
n.js"></script>
    <script src="login.js"></script>
  </head>
  <body onload="init()">
     <div class="container" style="width:350px; margin-top:100px">
        <form method="post" action="/login">
          <div class="mb-3">
            <label for="name" class="form-label">Username</label>
            <input type="text" class="form-control" id="username">
           </div>
           <div class="mb-3">
            <label for="password" class="form-label">Password</label>
            <input type="password" class="form-control" id="password">
           </div>
           <button class="btn btn-primary" id="login">Login</button>
        </form>
     </div>
  </body>
</html>
```

    b. Dodajte javascript kod koji će pridružiti click event na button#login. Treba poslati POST request na AUTH servis, na rutu /login

```
window.addEventListener('load', () => {
    document.getElementById('btn').addEventListener('click', (evt) => {
        evt.preventDefault();
```

```
        const data = {
            username: document.getElementById('username').value,
            password: document.getElementById('password').value
        };

        fetch('http://127.0.0.1:9001/login', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(data)
        })
            .then( res => res.json() )
            .then( data => {
                if (data.msg) {
                    //prikazi gresku
                    alert(data.msg);
                } else {
                    //zapisi token u cookie
                    document.cookie = `token=${data.token};SameSite=Lax`;
                    //redirektuj na index.html stranicu
                    window.location.href = 'index.html';
                }
            });
    });
});
```

4. Pošto je token smešten u cookie, svaki budući request ka 127.0.0.1:8000 će sadržati i token, po automatizmu (browser vodi računa o tome, šalje se u request header-u, u atributu Cookie)
5. Kada pokrenete sva tri servisa trebalo bi da 127.0.0.1:8000/administrator/login prikaže login stranicu, i nakon uspešnog logina redirektuje na index stranicu.
    a. Vodite računa o tome da **127.0.0.1** i **localhost** nisu isti *origin*, iako gađaju istu mašinu.

Logout

Logout se na klijentskoj strani svodi na prosto brisanje tokena iz cookie-a. To možemo da uradimo ovako nekako:

```
document.cookie = `token=;SameSite=Lax`;
window.location.href = 'login.html';
```

Ovo ćete pozvati na click za neki logout link ili već kako budete napravili svoju aplikaciju.

Zaštita ruta na APP servisu

Svi zahtevi ka APP servisu, koji prolaze kroz static middleware, tj. interakcija korisnika sa html stranicama, sadržaće (nakon login-a) token u cookie. Treba nam funkcija koja će da iz requesta izvuče cookie i token, a zatim i proveri da li je sadržaj tokena validan (validno potpisan kriptoalgoritmima) i na osnovu toga prihvati odnosno odbije request.

1. Otvorite app servis/app.js
2. Nakon const app=express(); napravite pomoćnu funkciju getCookies(request)

```
function getCookies(req) {
    if (req.headers.cookie == null) return {};

    const rawCookies = req.headers.cookie.split('; ');
    const parsedCookies = {};

    rawCookies.forEach( rawCookie => {
        const parsedCookie = rawCookie.split('=');
        parsedCookies[parsedCookie[0]] = parsedCookie[1];
    });

    return parsedCookies;
};
```

3. A nakon ove funkcije napravite i funkciju authToken(req, res, next) koja je zapravo middleware. Ova funkcija će proveriti token i zavisno od tokena pušta korisnika dalje odnosno redirektuje na /login

```
function authToken(req, res, next) {
    const cookies = getCookies(req);
    const token = cookies['token'];
    if (token == null) return res.redirect(301, '/login');
    jwt.verify(token, process.env.ACCESS_TOKEN_SECRET, (err,
user) => {
        if (err) return res.redirect(301, '/login');
        req.user = user;
        next();
    });
}
```

4. Logika administratorske aplikacije treba da bude sledeća: korisnik poseti 127.0.0.1:8000/administrator, web servis primeti da nije ulogovan, redirektuje na ga na /login. Korisnik se uloguje, dobije token, i redirektuje se na /administrator. Pošto je sada ulogovan, prikazuje mu se index.html. Rutu za index.html treba da zaštitimo sa authToken middleware-om, ali rute login i register treba da budu javno dostupne. Napravićemo ove rute eksplicitno, da ne bismo komplikovali sa static middleware.

```
app.get('/administrator/register', (req, res) => {
    res.sendFile('register.html', { root: './static/admin' });
});
```

```
app.get('/administrator/login', (req, res) => {
    res.sendFile('login.html', { root: './static/admin' });
});

app.get('/administrator', authToken, (req, res) => {
    res.sendFile('index.html', { root: './static/admin' });
});
```

Dodavanje tokena u administratorsku aplikaciju

Nakon login-a token je smešten u cookie, i šalje se kad god od APP servisa tražimo neki html fajl. Ali unutar html fajlova podaci (forme) se šalju na API servis, a cookie se tu ne šalje - API servis je drugi *origin* i cookie nema veze sa tim. Propisan način da prosledimo token je da u request dodamo Authorisation header i tu smestimo token.

1. Ovo će nam trebati inicijalno. Ne mora da bude u load event-u, može da se smesti i u script tag u head delu svake stranice, jer je cookie nešto što je dostupno bez obzira na DOM stablo
```
const cookies = document.cookie.split('=');
const token = cookies[cookies.length - 1];
```
2. Ovako izgleda GET zahtev:
```
fetch('http://127.0.0.1:8000/api/messages', {
        headers: {
            'Authorization': `Bearer ${token}`
        }
})
```
3. Ovako izgleda POST zahtev:
```
fetch('http://127.0.0.1:8000/api/messages', {
      method: 'POST',
      headers: {
          'Content-Type': 'application/json',
          'Authorization': `Bearer ${token}`
      },
      body: JSON.stringify(data)
})
```

Zaštita ruta na API servisu

1. Na API servis kopirajte .env fajl sa auth servisa
2. Instalirajte dotenv, jsonwebtoken i dodajte u aplikaciju
```
const jwt = require('jsonwebtoken');
require('dotenv').config();
```

3. Request koji stiže na API servis sadržaće token u Authorisation header-u. Potreban nam je middleware koji će da proveri da li token postoji u header-u, da proveri sadržaj tokena i da na osnovu toga prosledi dalje request, odnosno vrati odgovarajuću grešku.

```
function authToken(req, res, next) {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];
  if (token == null) return res.status(401).json({ msg: err });
  jwt.verify(token, process.env.ACCESS_TOKEN_SECRET, (err, user)
=> {
     if (err) return res.status(403).json({ msg: err });
     req.user = user;
     next();
  });
}
```

4. Mozete da napravite dve verzije authToken() middleware-a, za proveru da li je ulogovan i za proveru da li je admin, i da ih koristite zavisno od rute, gde vam koji treba. U token mozete da smestite i podatak isAdmin: bool
5. Ovu funkciju treba da primenimo na sve one rute koje treba zaštititi od neautorizovanog pristupa: deo get ruta i sve post, put, delete rute.

```
route.use(authToken);
```

6. Po potrebi možete da napravite i dve varijante ove funkcije: adminAuth() i userAuth() i da tako zaštitite rute na osnovu privilegije koju user ima, a koja se upisuje u token prilikom login, kada se token kreira.

Dodavanje tokena u vue.js

Sada ćemo u klijentsku vue.js aplikaciju koju ste napravili u vežbi 4 da dodamo mogućnost za login i registraciju. Treba da dodamo token i potrebne mutacije i akcije u state, a zatim i da to upotrebimo u App.vue

State

1. Otvorite store/index.js
2. U state dodajte atribut token
3. U mutacije dodajte dve mutacije: setToken i removeToken
4. U akcije dodajte dve akcije: login i register
5. Evo kompletnog isečka store-a, u kome su potrebni elementi

```
export default new Vuex.Store({
  state: {
    token: ''
  },
  mutations: {
    setToken(state, token) {
```

```
      state.token = token;
      localStorage.token = token;
    },
    removeToken(state) {
      state.token = '';
      localStorage.token = '';
    },
  },
  actions: {
    register({ commit }, obj) {
      fetch('http://127.0.0.1:9001/register', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(obj)
      }).then( res => res.json() )
        .then( data => commit('setToken', data.token) );
    },
    login({ commit }, obj) {
      fetch('http://127.0.0.1:9001/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(obj)
    }).then( res => res.json() )
      .then( data => {
        if (data.msg) {
          alert(data.msg);
        } else {
          commit('setToken', data.token)
        }
      });
    }
  }
})
```

Aplikacija

6. Otvorite App.vue
7. U mainnav, stavite linkove za /register, /login i logout.
   Prikaz zavisi od postojanja tokena iz state-a

```
<b-nav-item v-if="!token" to="/register">Register</b-nav-item>
<b-nav-item v-if="!token" to="/login">Log In</b-nav-item>
<b-nav-item v-else @click="logout()">Log Out</b-nav-item>
```

8. Na početku script bloka stavite import za potrebne delove iz vuex

```
import { mapActions, mapState, mapMutations } from 'vuex';
```

9. Token dohvatite iz state, kao computed property

```
computed: {
  ...mapState([
    'token'
  ])
},
```

10. U metode uvucite removeToken i setToken mutacije, i napravite metod logout() koji poziva mutaciju removeToken

```
methods: {
  ...mapMutations([
    'removeToken',
    'setToken'
  ]),
  logout() {
    this.removeToken();
  }
},
```

11. Kada se aplikacija pokrene (mounted event) treba da proverimo da li u localstorage postoji token, i ako postoji da ga ubacimo u state. Na taj način zapamtićemo korisnika koji se ulogovao pa refreshovao stranicu.

```
mounted() {
  if (localStorage.token) {
    this.setToken(localStorage.token);
  }
},
```

12. Napravite views/Login.vue i stavite login formu, definišite potrebne varijable za username i password, uvucite akciju login, i definišite handler za onsubmit, koji poziva akciju login i redirektuje na Home rutu. Evo primera celog fajla. Korisimo bootrap-vue komponente za formu

```
<template>
  <div id="app">
    <Header title="Log In"/>
    <b-form @submit="onSubmit">
      <b-form-group label="User Name:" label-for="name">
        <b-form-input id="name" v-model="form.name"
            placeholder="Enter name" required></b-form-input>
      </b-form-group>
      <b-form-group label="Password:" label-for="password">
        <b-form-input id="password" v-model="form.password"
            type="password" required></b-form-input>
      </b-form-group>
      <b-button type="submit" variant="primary">Login</b-button>
```

```
      </b-form>
    </div>
  </template>
  <script>
    import Header from '@/components/Header.vue';
    import { mapActions } from 'vuex';
    export default {
      name: 'Login',
      components: {
        Header
      },
      data() {
        return {
          form: {
            username: '',
            password: ''
          }
        }
      },
      methods: {
        ...mapActions([
          'login'
        ]),
        onSubmit(e) {
          e.preventDefault();
          this.login(this.form);
          this.$router.push({ name: 'Home' });
        }
      }
    }
  </script>
  <style scoped>
  </style>
```

13. Napravite views/Register.vue koji sadrži formu za registraciju. Slično kao login, samo što forma ima dodatno polje email, i poziva akciju register. Evo primera celog fajla.

```
<template>
  <div id="app">
    <Header subtitle="Create account"/>
    <b-form @submit="onSubmit">
      <b-form-group label="Email address:" label-for="email">
        <b-form-input id="email" v-model="form.email" type="email"
          placeholder="Enter email" required></b-form-input>
```

```
      </b-form-group>
      <b-form-group label="User Name:" label-for="name">
        <b-form-input id="name" v-model="form.name"
            placeholder="Enter name" required></b-form-input>
      </b-form-group>
      <b-form-group label="Password:" label-for="password">
        <b-form-input id="password" v-model="form.password"
            type="password" required></b-form-input>
      </b-form-group>
      <b-button type="submit" variant="primary">Register</b-button>
    </b-form>
  </div>
</template>
<script>
  import Header from '@/components/Header.vue';
  import { mapActions } from 'vuex';
  export default {
    name: 'Register',
    components: {
      Header
    },
    data() {
      return {
        form: {
          email: '',
          name: '',
          password: '',
        }
      }
    },
    methods: {
      ...mapActions([
        'register'
      ]),
      onSubmit(e) {
        e.preventDefault();
        this.register(this.form);
        this.$router.push({ name: 'Home' });
      }
    }
  }
</script>
<style scoped>
```

```
</style>
```

14. I za kraj, proverite da li svi fetch() pozivi sadrže potreban Authentication header sa tokenom