Математичка гимназија

МАТУРСКИ РАД
- из програмирања -

Програм за тестирање знања

Ученик:
Филип Стевановић IVц

Ментор:
Петар Радовановић

Београд, јун 2021.

Садржај

1

Увод

Циљ овог рада био је да се направи апликација за прављење тестова, њихово задавање и дистрибуцију, као и само рађење, аутоматско прегледање и мануално оцењивање. Апликација је направљена у формату Web апликације управо због највеће доступности и најлакшег приступа. Жеља и план за будућност је да се ова апликација објави и буде доступна свима на интернету.

Идеја за овај пројекат је настала у складу са тренутном ситуацијом широм света и пандемијом коронавируса. Како је огроман број школских и образовних установа био приморан на прилагођавање и прелазак на online наставу, тестирања и провере знања су добили другачији облик него раније. Ова апликација могла би да пружи још један начин да се тестирање и оцењивање ђака спроведе на даљину, и да се омогуће слични услови као и у школи. Такође, сем професора и наставника, апликација може пружити услуге и другима, за креирање анкета или квизова.

У овом раду најпре ћемо се упознати са Web програмирањем, основним технологијама и програмским језицима које су коришћене при прављењу апликације, и на крају са самом апликацијом.

2

Web апликације

Web апликације су апликације којима корисници приступају преко мреже, као што је интернет, и које корисницима пружају услуге преко Web претраживача. За разлику од обичних апликација које се покрећу и раде помоћу оперативног система, Web апликације раде на Web серверима који од клијената, преко претраживача добијају информације односно HTTP (Hypertext Transfer Protocol) захтеве, на које сервер одговара у складу са добијеним подацима.

Можемо и сами закључити да је основни циљ Web апликација омогућавање што бољег односа између клијента (програма који корисник користи за слање захтева) и сервера. Свака Web апликација има два основна дела. Део који је задужен за оно што клијент види и задужен за интеракције клијената са апликацијом - Frontend и део који је задужен за све процесе који се дешавају на серверу, за примање и слање информација клијентима - Backend.

2.1 Frontend

Frontend је део Web апликације који корисник види на Web browser-у. Како је сваки корисник у директном контакту са овим делом апликације, основни задатак Frontend-а је да омогући клијентима што разумљивији приказ апликације (независно од клијента, величине екрана), и да им обезбеди могућност за све предвиђене функционалности саме апликације.

Frontend апликације се прави коришћењем 3 основна језика:

1 HTML - Hypertext Markup Language је стандарни описни програмски језик за представљање докумената, који се приказују на Web страницама. Он описује основну структуру и дефинише основне делове интернет странице.

2 CSS - Cascading Style Sheets је програмски језик који се користи за описивање елемената документа написаног у неком описном језику. Он служи за организовање елемената, њиховог приказа на страници, тј. за стилизовање и дизајнирање интернет странице

3 JavaScript је динамичан, скриптни програмски језик високог нивоа који се користи за интеракцију корисника са Web страницом. Може се користити и као Backend језик али је основни задатак динамичка измена садржаја документа, брисање и додавање елемената HTML-а.

Ова три програмска језика заједно са DOM-ом чине DHTML или динамички HTML. DOM или Document Object Model је модел који дефинише HTML елементе као објекте и представља спону између HTML-а и JavaScript-а. DOM представља документ као бинарно стабло у чијем се сваком чвору налази објекат. Методе које користи омогућавају промену структуре, садржаја и стила документа што је и задатак JavaScript-а.

Поред њих могу се користити и други програмски језици, или неки други Framework. Framework је већ постојећи и написан стандардизовани код, најчешће надоградња JavaScript-а или CSS-а који служи за одређену врсту апликације и могу убрзати и олакшати прављење апликације. Најпознатији су: Angular JS, React.js, vue.js, Bootstrap итд.

2.2 Backend

Backend је део Web апликације који корисник не види и са којим нема никакву директну интеракцију, већ који утиче на све процесе у позадини, на Web серверу. Ипак корисници индиректно комуницирају са Backend-ом преко Frontend-а и захтева који се шаљу серверу. Главни циљ Backend-а апликације је у томе да чува и организује све потребне податке о апликацији, о сваком кориснику, и да контролише сва дешавања на Frontend-у. Поред тога Backend има и улогу у неким активностима које се одвијају у потпуности без корисника, као што су креирање библиотека, рад са системским компонентама и писање API-а или ти Application Programming Interface који служи да омогући и дефинише комуникацију између две или више различитих апликација.

Backend се углавном састоји од три основна дела:

1 Сервер - Рачунар на коме се налази апликација и који прима све захтеве клијената

2 Апликација - Програм који слуша захтеве, одговара на њих обављајући задате задатке, комуницира са базом података и шаље одговоре клијентима

3 База података - организован систем свих података о аплкацији и о свим њеним корисницима

Велики је број програмских језика који се користе за прављење Backend-а Web апликације а неки од оних који су најчешћи су: PHP, Java, C++, Ruby, као и Node.js о коме ћемо нешто више рећи јер је он коришћен као Backend за ову апликацију. Што се тиче базе података, оне могу бити SQL базе које се користе када је јасно дефинисана структура података, а неке од таквих база су MySQL, PostgreSQL, Oracle; а постоје и NoSQL базе које обезбеђују много већу флексибилност и могућности мењања структуре саме базе. Један пример је MongoDB која је коришћена у овој апликацији.

2.3 Коришћене технологије

За потребе Frontend-а ове апликације коришћена су основна три језика: HTML, CSS, JavaScript као и EJS или Embedded JavaScript. EJS је језик за шаблонирање који дозвољава динамичко приказивање HTML докумената уз помоћ JavaScript-а. Он је повезан и са Backend-ом јер омогућава приказивање потребних података са сервера, тј. из базе података, на Frontend-у апликације. Често се користи у вези са Node.js и Express апликацијама што је и случај у овој апликацији.

Node.js је вишеплатформско JavaScript радно окружење дизајнирано за извршавање JavaScript кода и ван Web browser-а, односно на серверу. У овој апликацији је коришћен као основни Backend програмски језик. Node.js апликације раде на само једном процесу и не креирају нове thread-ове за сваки нови захтев. Због асинхроног приступа У/И (улазно-излазним) операцијама не долази до блокирања процеса јер се не чека на завршетак претходног захтева већ се одмах прелази на следећи. Node.js је open-source окружење за које постоји велики број библиотека, модула који се користе за олакшавање писања кода одређених делова апликације. NPM или Node package manager је софтвер који чува све познате Node.js пакете, односно модуле и са којим се преко конзоле могу инсталирати потребни модули за апликацију.

Express.js је основни серверски Node.js framework који га надограђује тиме што омогућава лакше рутирање апликације, омогућава коришћење језика за шаблонирање као што је EJS, и користи се са великим бројем других модула који

побољшавају серверски део апликације. У овој апликацији коришћени су NPM модули: bcrypt који служи за енкриптовање података о корисницима (њиховим шифрама), body-parser који служи за издвајање потребних делова из захтева који се шаљу на сервер, ejs, express, express-session, који служи за одржавање улогованог корисника кроз различите делове апликације express-flash који служи за обавештења од сервера кориснику, method-override који служи за слање захтева за ажурирање и брисање, mongoose који служи за рад са MongoDB базом података и passport који служи за систем за логовање корисника.

Као база података у овој апликацији коришћен је MongoDB. MongoDB је нерелациона база отвореног кода која чува податке у виду JSON (JavaScript Object Notation) докумената са динамичким шемама (шеме би у релационим SQL базама биле табеле). JSON је стандард за приказивање података дизајниран да буде размуљив људима.

3

Апликација

3.1 Креирање апликације

Да би апликација радила локално потребно је имати инсталиран Node.js и MongoDB. Апликација се креира куцањем npm init у конзоли а затим попуњавањем потребних података. Инсталирањем неопходних модула, који су већ набројани у претходном поглављу, помоћу команде npm install [име модула] Node.js ствара package.json фајл који садржи све најосновније податке о апликацији.

```
{
  "name": "projekat2",
  "version": "1.0.0",
  "description": "Program za testiranje znanja",
  "main": "app.js",
  "scripts": {
    "start": "node app.js",
    "dev": "nodemon app.js"
  },
  "author": "Filip Stevanovic",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.0.1",
    "body-parser": "^1.19.0",
    "ejs": "^3.1.5",
    "express": "^4.17.1",
    "express-flash": "0.0.2",
    "express-session": "^1.17.1",
    "method-override": "^3.0.0",
    "mongoose": "^5.11.13",
```

```
    "passport": "^0.4.1",
    "passport-local": "^1.0.0"
  },
  "devDependencies": {
    "nodemon": "^2.0.7"
  }
}
```

Од до сада непоменутих модула, односно dependency-а, остао је nodemon који нема неку улогу у апликацију на самом крају, већ служи да се сервер аутоматски ресетује чим се деси било каква промена у коду на Backend-у што нам значи док је апликација у dev фази ондосно у фази развића. Сервер покрећемо покретањем скрипте node app.js куцањем npm run start. Фајл app.js је основни део апликације у коме се иницјализују сви коришћени модули, повеже са базом података, одреде се руте и покрене сервер.

```
const express = require('express');
const bodyParser = require('body-parser'); //submit form body
const methodOverride = require('method-override');//PUT, DELETE requests
    iz forma
const mongoose = require('mongoose'); //database
const passport = require('passport'); //za user authetication
const flash = require('express-flash'); //za poruke kroz redirect
const session = require('express-session');//za odrzavanje ulogovanog
    usera kroz ceo website
const port = 3000;

const initializePassport = require('./passport-config');
initializePassport(passport);

const app = express();

app.set('view engine', 'ejs');
app.set('views',__dirname + '/views');

app.use('/public',express.static('public'));

app.use(bodyParser.urlencoded({extended: false}));
app.use(bodyParser.json());

app.use(methodOverride('_method'));
```

```
app.use(flash());
app.use(session({
    secret: 'secret',
    resave: false,
    saveUninitialized: false
}))
app.use(passport.initialize());
app.use(passport.session());

//connect to database
mongoose.connect('mongodb://localhost:27017/testapp',{useNewUrlParser:
    true, useUnifiedTopology: true});
const db = mongoose.connection;
db.on('error', function(error){
    console.error(error);
});
db.once('open', function(){
    console.log('Connected to Mongoose');
});


//Routes
const indexRouter = require('./routes/index');
const testRouter = require('./routes/tests');
const questionRouter = require('./routes/questions');
const groupRouter = require('./routes/groups');

app.use('/', indexRouter);
app.use('/tests', testRouter);
app.use('/questions',questionRouter);
app.use('/groups',groupRouter);


//PORT
app.listen(port, function(error){
    if(error){
        console.log('greska', error);
    }
    else{
        console.log('Server is listening on port ' + port);
    }
});
```

3.2 Почетна страница

Слика 1: Почетна страница

Почетна страница садржи навигациони мени са линковима ка осталим деловима апликације, тј. ка осталим рутама (нпр. рута Tests са наставком /tests може водити ка линку који почиње са http://localhost:3000/tests/). Навигациони мени је део и сваке друге странице, тј. он представља header апликације. Ниједној од рута сем саме почетне странице се не може приступити уколико корисник није улогован. Зато се и на почетној страни налази део за логовање и линк за регистровање корисника.

3.3 Аутентикација корисника

Аутентикација корисника је битан део велике већине апликација. Неопходно је корисницима обезбедити увид у сопствене податке, и све податке чувати у бази података у складу са тим са којим корисником су они у вези. Исто тако треба ограничити корисника да може да види само његове податке (у случају ове апликације његове тестове, питања, резултате или групе) а забранити приступ

подацима неког другог корисника. Ту на сцену ступа аутентикација корисника која омогућава серверу да контролише који је корисник улогован са ког уређаја и да зна које податке из базе може да приказује кориснику и којим подацима корисник може да приступа. За потребе аутентикације корисника у овој апликацији коришћен је passport модул.

Функција за проверу при логовању корисника:

```
const authenticateUser = async function (email, password, done){
    try{
        const user = await User.findOne({email: email});
        if(!user){
            return done(null,false,{message: 'No user with that email'});
        }
        if(await bcrypt.compare(password, user.password)){
            return done(null, user);
        }
        else{
            return done(null, false, {message: 'Incorrect password'})
        }
    }
    catch(e){
        return done(e);
    }
}
```

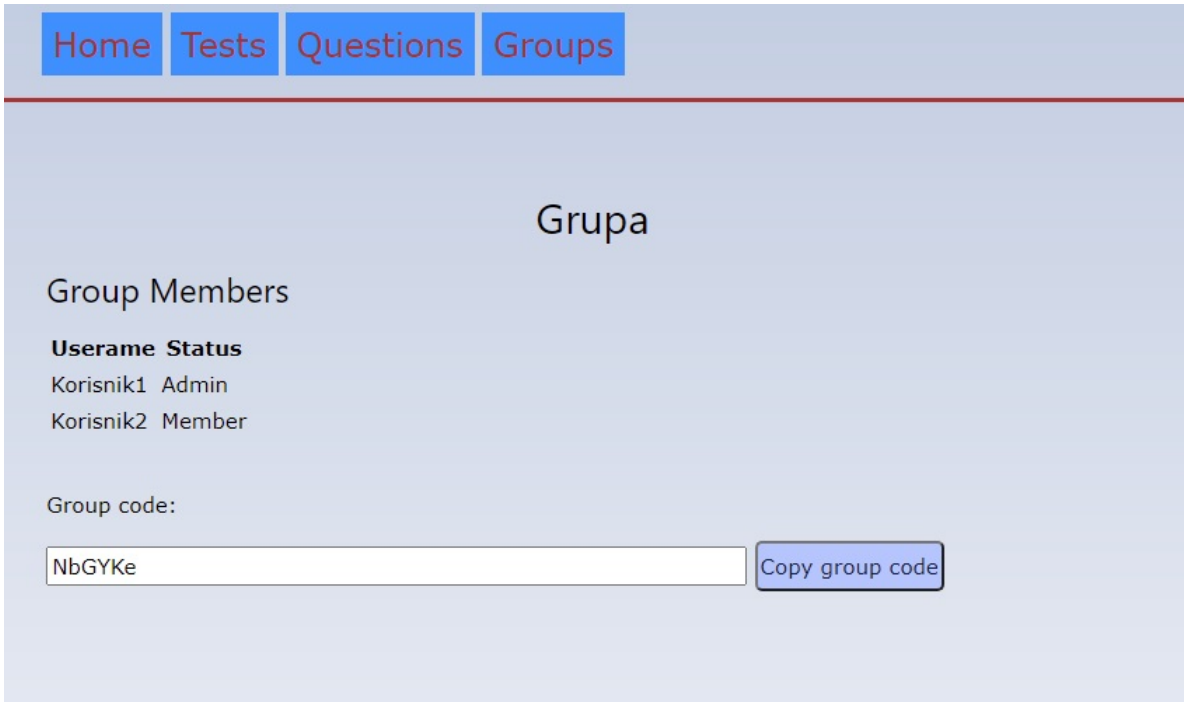
Регистровање корисника уколико су сви услови испуњени (нпр. не постоји корисник са тим истим именом):

```
const hashedPassword =await bcrypt.hash(req.body.password, 10);
const user = new User({
    username: req.body.username,
    email: req.body.email,
    password: hashedPassword
})
console.log(user);
await user.save();
console.log('Sacuvan user');
res.redirect('/');
```

где је User модел односно шема у бази података која чува основне податке о кориснику и чијим се референцирањем корисник повезује са својим тестовима, питањима...

3.4 Групе корисника

Корисници могу да праве групе ради лакшег задавања тестова одређеној групи људи. При прављењу групе потребно је само одредити име групе а апликација сама групи додељује јединствени код од 6 слова којим се други људи могу прикључити групи укуцавши потребан код када у header-у под делом Groups кликну на Join group.



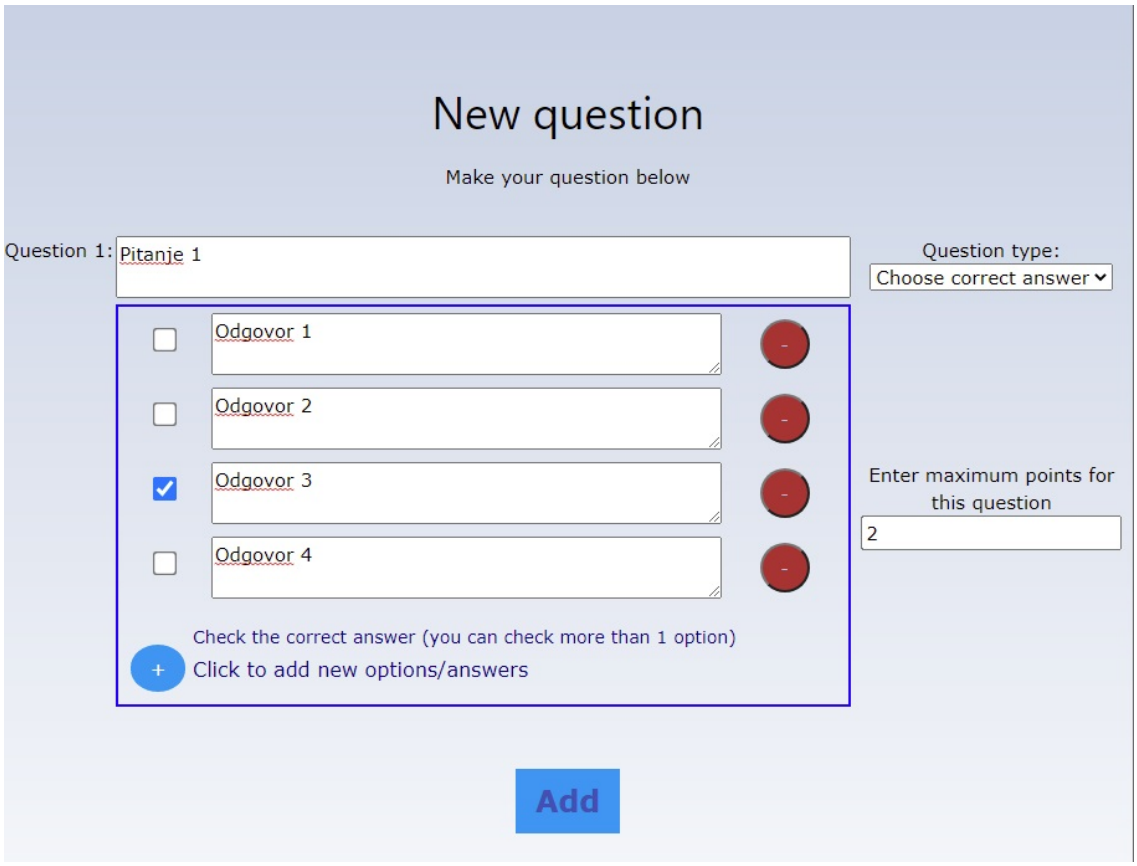
Слика 2: Страница приказивања групе

Admin групе, односно онај корисник који је направио групу је једини који може да задаје тестове групи, тј. осталим члановима групе.

3.5 Креирање тестова и питања

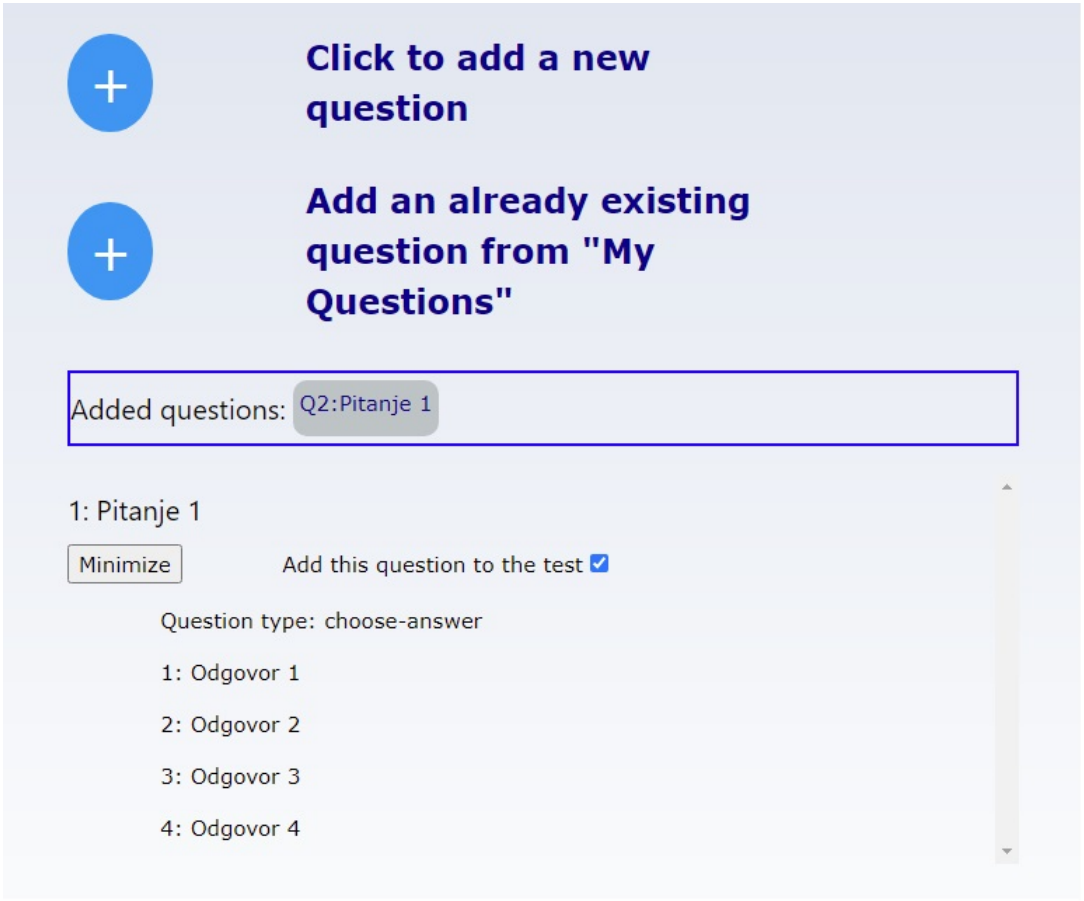
Прављењу новог теста или питања корисник приступа кликтањем на дугме New Test односно New Question. Прављење новог питања даје кориснику могућност да направи једно питање које касније може додати у било који тест. У апликацији постоје три различита типа питања: Питање са одабиром тачног или

тачиних одговора; питање са опцијама тачно или нетачно; као и питање са попуњавањем тачног одговора.



Слика 3: Додавање питања са одабиром тачних одговора

Прављење теста је доста слично прављењу питања са тим што се у тест може додати већи број питања, укључујући и нова и већ постојећа питања.



Слика 4: Додавање постојећих питања у нови тест

Када корисник заврши са прављењем теста или питања Frontend апликације шаље POST захтев Backend-у са подацима које је корисник унео. POST захтев је један од 4 основна захтева који се сальу серверу уз GET, PUT, DELETE захтеве и POST захтев означава да се неки нови подаци шаљу серверу и треба их сачувати у базу података. Backend издваја унесене податке корисника уз помоћ модула body-parser и прерађује их тако да одговарају формату модела, тј. шеме из базе података (шеме Question и Test). Уколико је све успешно тест и сва нова питања, или ново питање се чува у базу података и повезује са тренутним корисником.

Пример кода за моделовање шеме Test:

```
const mongoose = require('mongoose');
const Exam = require('./exam');
```

```
const testSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    questions: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Question',
        required: true
    }],
    createdAt: {
        type: Date,
        required:true,
        default: Date.now
    },
    createdBy: {
        type: mongoose.Schema.Types.ObjectId,
        required: true,
        ref: 'User',
    }
});

testSchema.pre('remove', function(next){//ne smemo brisati ako se test
    koristi u nekom examu
    Exam.find({test: this.id}, function(err,exams){
        if(err){
            next(err); //samo ako se desi greska u DB
        }
        else if(exams.length>0){
            next(new Error('This test is used in at least one exam so it
                cannot be deleted'));
        }
        else{
            next();
        }
    })
})

const Test = mongoose.model('Test',testSchema);

module.exports = Test;
```

Чување нових питања из теста у базу и њихово додавање на нови тест, након претходног форматирања да све одговара моделу Question:

```
        if(req.user){
            questions[i].createdBy=req.user.id;
        }

        console.log(questions[i]);

        //cuvanje u bazu
        try {
            await questions[i].save();
            console.log('Sacuvano pitanje');
        } catch {
            console.log('greska za pitanje');
            ok=false;
        }

        test.questions.push(questions[i].id);

        i++;//prelazak na sledece pitanje
```

Додавање већ постојећих питања у нови тест и чување новог теста у базу

```
//dodavanje vec postojecih pitanja u test
let niz = req.body.existing_questions;//string ID
let nizID = [];
if(niz != null){
    if(Array.isArray(niz)){
        max= niz.length;
        let j=0;
        while(j<max){
            niz[j].trim();
            nizID[j] = mongoose.Types.ObjectId(niz[j]);
            console.log(nizID[j]);
            test.questions.push(nizID[j]);
            j++;
        }
    }
    else{
        niz.trim();
```

```
            nizID[0] = mongoose.Types.ObjectId(niz);
            console.log(nizID[0]);
            test.questions.push(nizID[0]);
        }

    }
    else if(max==0){//ovo je stari max iz proslog while ako je 0 nema
        novih pitanja, sto onda sad znaci da nema pitanja uopste sto je
        greska
        ok=false;
    }

    //da se zna koji user je napravio
    if(req.user){
        test.createdBy=req.user.id;
    }

    //cuvanje celog testa u DB
    if(!ok){
        res.render('newtest',{
            test: test,
            questions: questions,
            errorMessage: 'Error creating a test'
        });
        console.log('Greska za neko pitanje pa se nije sacuvao ni test');
    }
    else{
       try{
            await test.save();
            console.log('Sacuvan Test');
            res.redirect('/tests');
        }
        catch{
            res.render('newtest',{
                test: test,
                errorMessage: 'Error creating a test'
            });
            console.log('Greska za test');
        }
    }
```

3.6 Едитовање и брисање

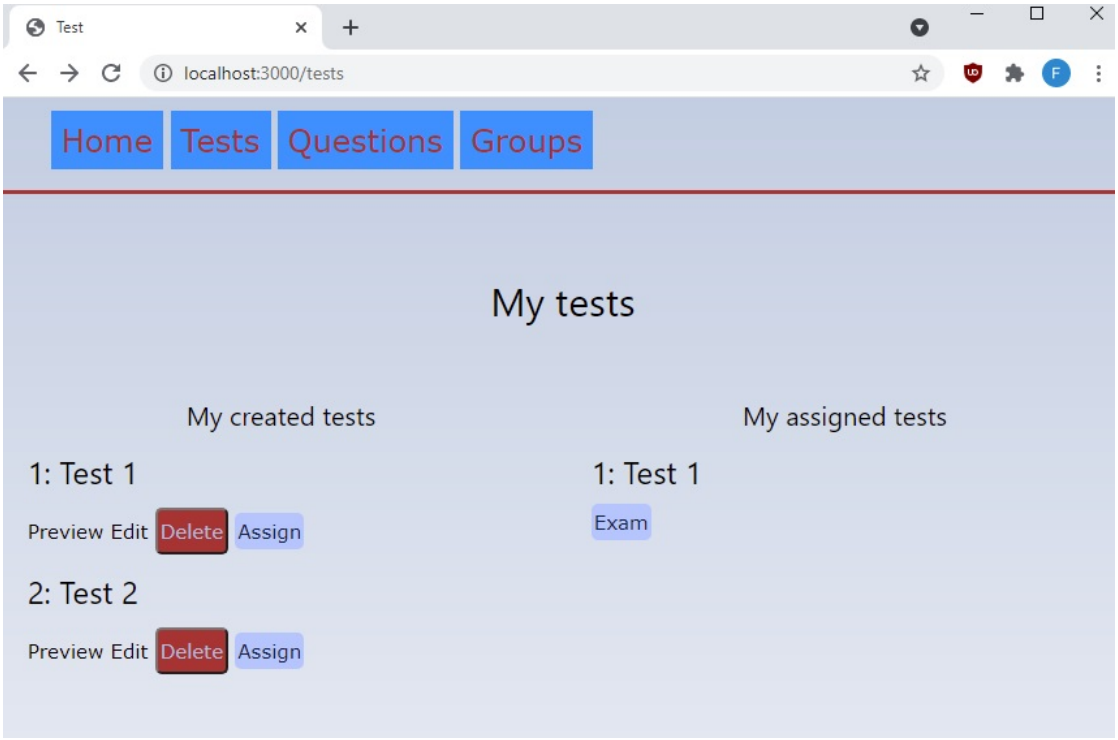
Сам процес на Backend-у едитовања теста или питања је доста сличан као и креирање нових. При едитовању теста води се рачуна да уколико се едитује неко питање теста, да се то питање промени само у том тесту а уколико је коришћено и у још неком тесту, то питање остаје непромењено у тим тестовима. Тај проблем се решио тако што када се едитује питање кроз едит теста направи се ново питање за тај тест а старо, које је едитовано се брише само из тог теста. Још једна разлика едитовања и креирања новог теста или питања је у начину на који се шаље захтев Backend-у. Како се из Web browser-а може послати само POST или GET захтев (GET захтев тражи податке од сервера, и углавном назад добија EJS документе који приказују изглед странице) а за едитовање је потребан PUT захтев који служи за ажурирање, апликација користи модул method-override који помаже да се POST метода претвори у PUT. На сличан начин се може добити и DELETE захтев, а у овој апликацији примењен је и други начин слања DELETE захтева који преко Frontend JavaScript-а шаље захтев Backend-у:

```
const deleteButton = document.querySelectorAll('.delete-button');

let string; //tests ili questions
if(document.location.href.includes('/tests')){
    string='tests';
}
else if(document.location.href.includes('/questions')){
    string='questions';
}

for(let i=0 ; i<deleteButton.length ; i++){
    deleteButton[i].addEventListener('click', async function(){
        const endpoint = `/${string}/${deleteButton[i].dataset.id}`;
        string = string.slice(0, string.length-1);
        r=confirm('Are you sure? This ' + string + ' will permanently be
            deleted from the database');
        if(r){
            try{
                const response = await fetch(endpoint,{
                    method: 'DELETE'
                });
                const data = await response.json();
                //console.log(data);
                if(data.error != null)alert(data.error);
```

```
                document.location=data.redirect;
            }
            catch(err){
                console.log(err);
            }
        }
    });
}
```

3.7 Приказивање тестова и питања

Рутери који су задужени за тестове, ондосно питања на својим index рутама (оним рутама које немају никакав наставак; те руте су http://localhost:3000/tests или http://localhost:3000/questions) апликација приказају све корисникове тестове, или питања.

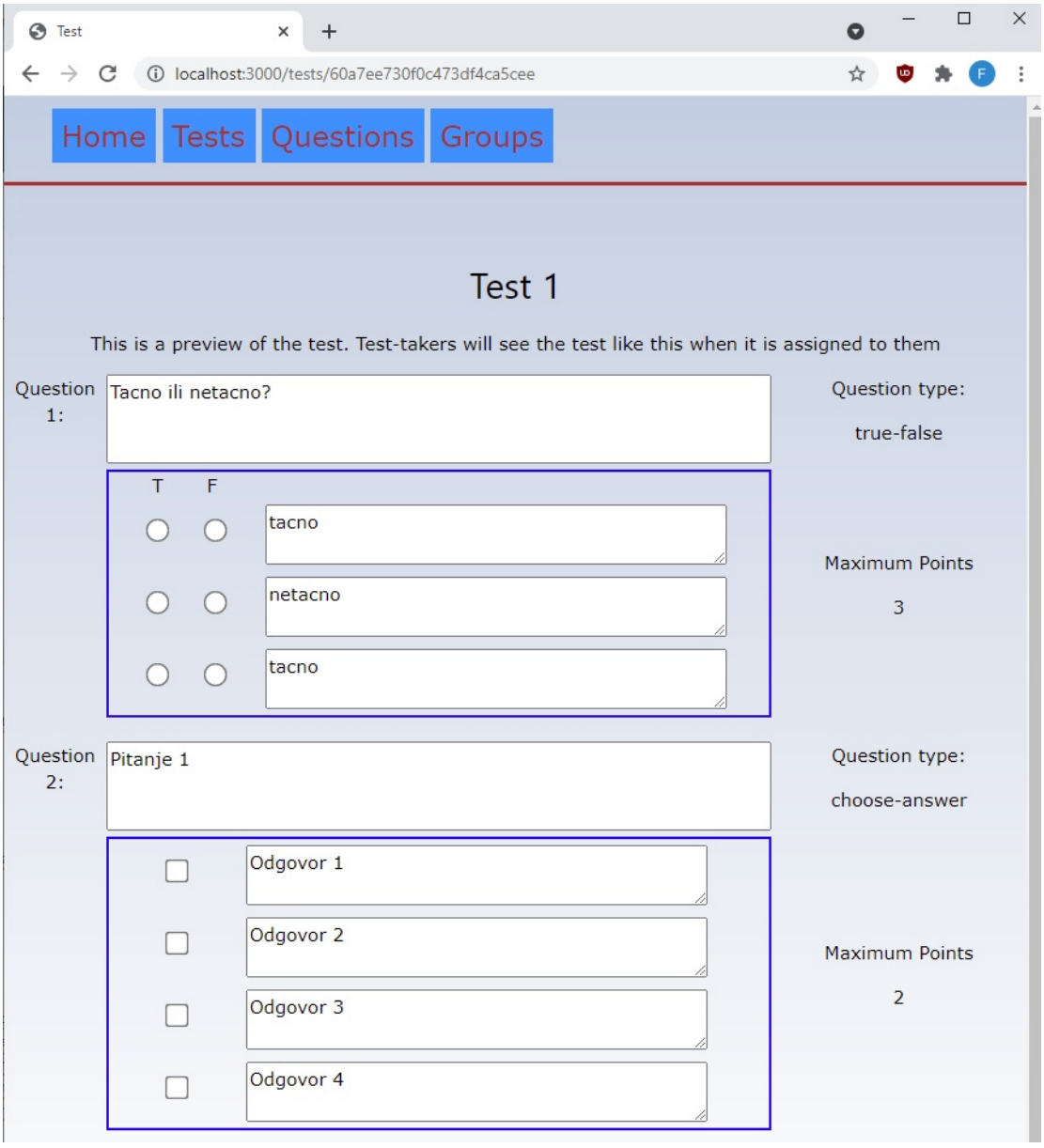


Слика 5: Страница за приказивање свих тестова

Са ове стране може се приступати појединачном прегледу сваког теста или питања, као и њиховом едитовању или брисању. Брисање питања је недозвољено

уколико постоји неки тест који садржи то питање. Исто тако брисање теста је недозвољено уколико је тај тест некада или сада задан, тј. постоји Exam тог теста.

Преглед појединачног теста омогућава професорима да виде како ће њихов тест изгледати онима који га буду радили.



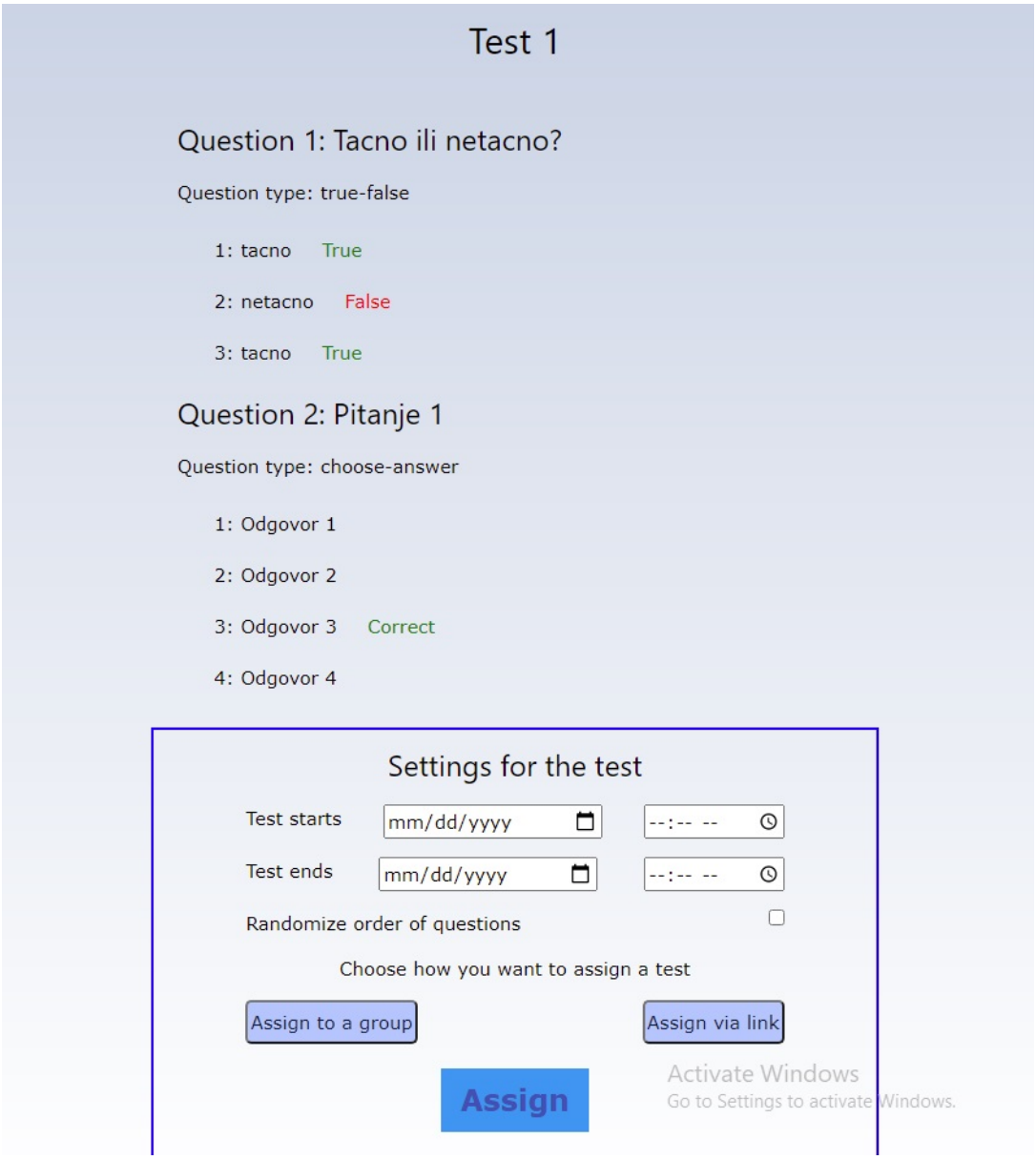
Слика 6: Страница за преглед (preview) теста

На страни за приказивање свих тестова професор види и све претходне

испите (односно Exam-ове), а такође одатле има опцију код сваког теста да га додели, тј. зада ђацима (Assign дугме).

3.8 Задавање тестова

При задавању тестова апликација нуди преглед теста са тачним одговорима и неке опције које треба попунити за задавање.



Слика 7: Страница за задавање теста

Задавање теста се може одрадити на два начина. Прво, задавање групи, је могуће само уколико је корисник који задаје тест админ неке групе (само администратори група могу да задају тестове у групи). Други начин је задавње преко линка, за које ће апликација направити линк који онда професор може проследити ученицима. Што се тиче задавање преко линка, постоји опција да се ограничи ко може приступити тесту, тј. да се обезбеди листа мејлова који смеју да раде тест; а постоји и опција да се тест остави отвореним и онда свако ко приступи линку може да уради тест уз обавезно попуњавање мејла и имена. Додатне опције при задавању су време почетка и краја које имају нека ограичења (нпр. време поцетка не може бити пре тренутног времена или време краја не може бити пре времена почетка), као и опција произвољног редоследа питања који сваком ученику, односно свакоме ко ради тест може бити другачији.

Backend део за чување задатих тестова у шему Exam

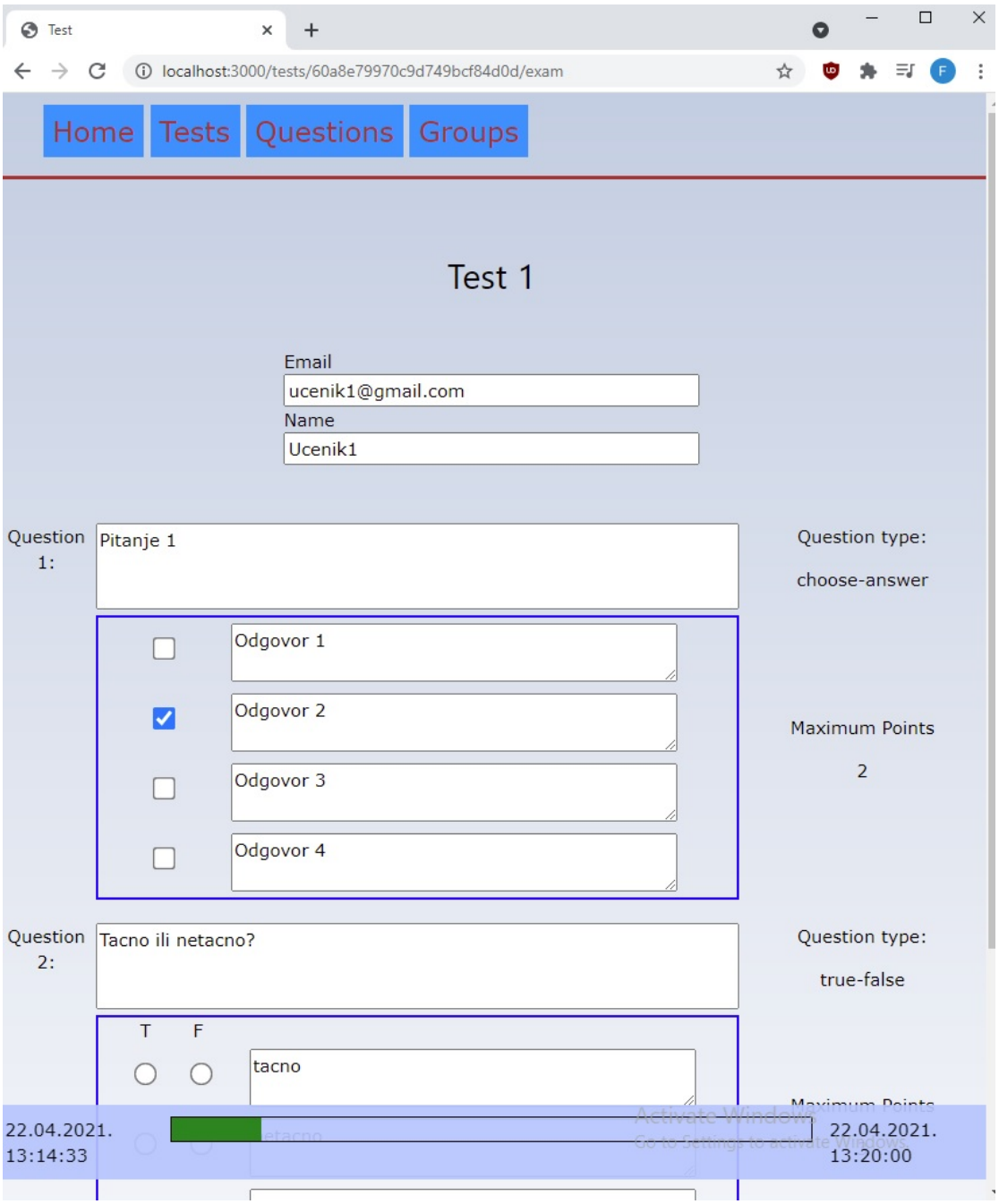
```
let today = new Date();
today=today.toISOString();
let dateNow=today.split('T')[0];
let timeNow=today.split('T')[1];

//za startsAt
if(req.body.date_start != '' && req.body.date_start != null){
    if(req.body.time_start != '' && req.body.time_start != null){
        start=req.body.date_start + 'T' + req.body.time_start;
    }
    else if(req.body.date_start != dateNow){
        start=req.body.date_start + 'T' + timeNow;
    }
    else{
        start=req.body.date_start + 'T00:00:00';
    }
}
else{
    if(req.body.time_start != '' && req.body.time_start != null){
        start=dateNow + 'T' + req.body.time_start;
    }
    else{
        start=dateNow + 'T' + timeNow;
    }
}

//za endsAt
if(req.body.date_end != '' && req.body.date_end != null){
```

```
        if(req.body.time_end != '' && req.body.time_end != null){
            end=req.body.date_end + 'T' + req.body.time_end;
        }
        else{
            end=req.body.date_end + 'T23:59:59';
        }
    }
    //else ostace samo undefined

    let startsAt = new Date(start);
    let endsAt;
    if(end){
        endsAt = new Date(end);
        //console.log(endsAt);
    }
    console.log(startsAt);
    console.log(endsAt);

    let random;

    if(req.body.random){
        random=true;
    }
    else{
        console.log('nema randoma');
        random=false;
    }

    const exam = new Exam({
        test: req.params.id,
        teacher: req.user.id,
        startsAt: startsAt,
        endsAt: endsAt,
        random: random
    })
    let group;
    let emails
    if(req.body.group){
        try{
            group = await Group.findOne({code: req.body.group});
            exam.group = group.id;
        }
        catch{
```

```
            res.redirect(`/tests/${req.params.id}/assign`);
            return;
        }
        console.log(group);
    }
    else{//samo tad moze da bude emailova iz whitelist
        console.log('nema grupe');
        if(req.body.emails && req.body.emails.length > 0){
            emails = req.body.emails;
            exam.emails=emails;
        }
    }

    console.log(exam);

    try{
        await exam.save();
        console.log('Test assigned');
        res.redirect(`/tests/${exam.id}/exam`);
    }
    catch(e){
        console.log('greska-nije assignovano');
        console.log(e);
        req.flash('info', 'Test was not assigned');
        res.redirect('/tests');
    }
```

3.9 Рађење и резултати теста

Када се тест додели, они који имају право да га раде, ће одласком на линк или преко своје групе у којој је задан тест моћи да приступе тесту. Тест се може радити само у временском интервалу који је одабран при додељивању и пре и после тога ће бити забрањен приступ свима уз одговарајућу поруку на Frontend-у. Такође уколико време истекне током рађења теста, он се неће аутоматски затворити већ ће се при следећем освеживању страници, или при покушају предаје приказати порука да је тест истекао. Зато апликација пружа јасан увид у преостало време до краја теста да би они који раде тест били свесни времена којим располажу.



Слика 8: Изглед странице током рађења теста

Када заврши тест ученик чека своје резултате. Апликација не дозвољава да се резултати виде док време за тест није истекло (сем у случају да је време теста неограничено), па је неопходно сачекати крај теста за добијање резултата. Резултати који се првобитно добију су прелиминарни (аутоматски прегледано

од стране компјутера, односно апликације) а званични резултати се добијају када професор прегледа тест.

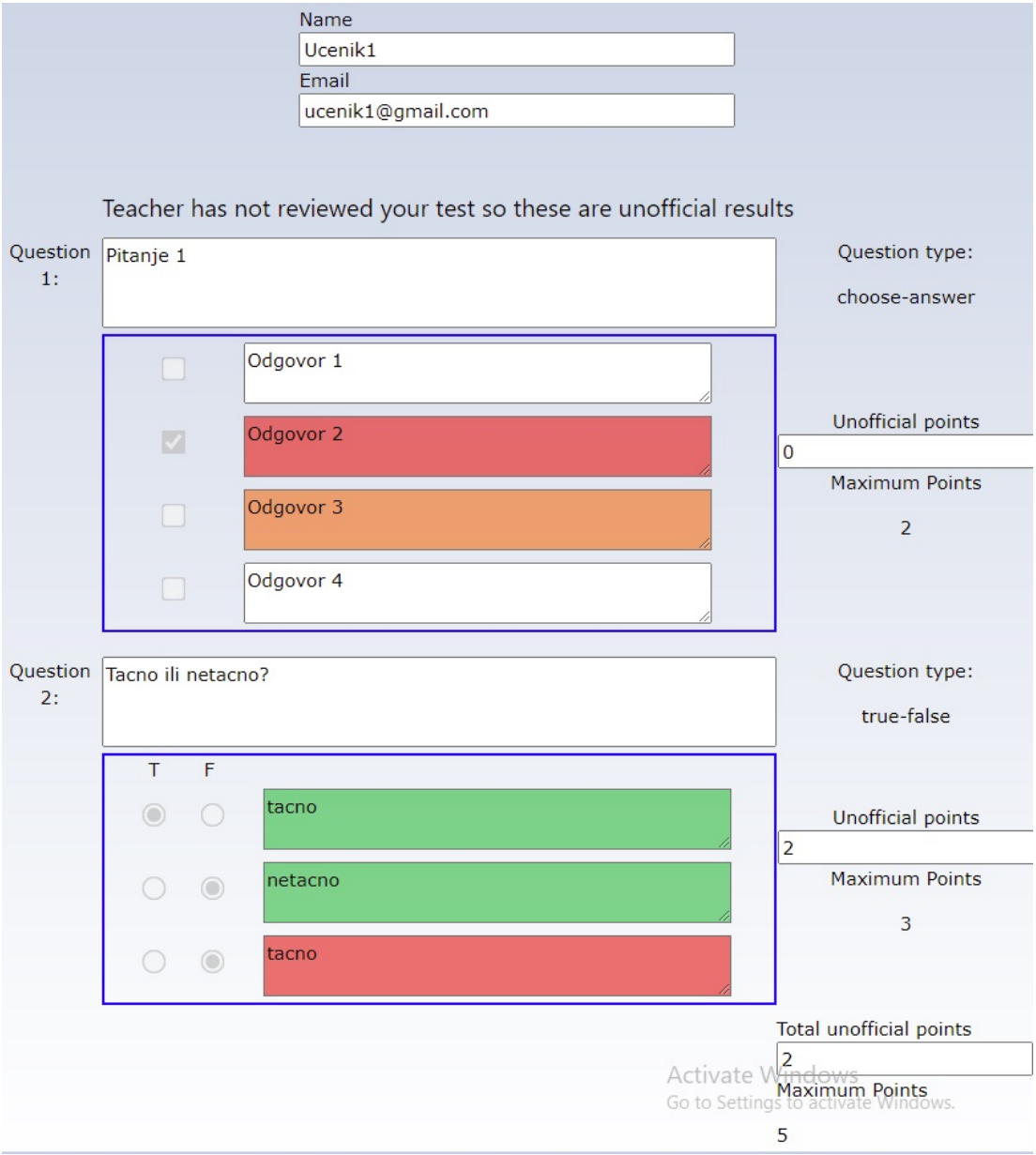
Део кода за чување одговора полагача на тесту који се касније додају у базу, у шему Result:

```
let autoPoints = [];
let searchstring;
let answers = [];
let j;
let tfarray;
for(let i=0 ; i<questions.length ; i++){
    //autoPoints[i]=0;
    if(questions[i].questionType=='choose-answer'){
        searchstring = 'answercheck'+(i+1);
        answers[i]={
            array: req.body[searchstring]
        }
    }
    else if(questions[i].questionType=='true-false'){
        j=0;
        tfarray = [];
        for(j ; j<questions[i].correct.length ; j++){
            searchstring = 'check_tf'+(i+1)+'_'+(j+1);
            if(!req.body[searchstring]){
                tfarray[j]=-1;
            }
            else if(req.body[searchstring] == 'true'){
                tfarray[j]=1;
            }
            else if(req.body[searchstring] == 'false'){
                tfarray[j]=0;
            }
        }
        answers[i]={
            array: tfarray
        }
    }
    else if(questions[i].questionType=='type-answer'){
        searchstring = 'typeanswer'+(i+1);
        answers[i]={
            string: req.body[searchstring]
        }
    }
```

```
        //console.log(answers[i]);
        result.answers.push(answers[i]);

        autoPoints[i]=automatic(result.answers[i], questions[i]);//funkcija
            koja radi automatski pregled svakog pitanja

        result.autoPoints=autoPoints;
    }
```

Функција за аутоматско прегледање:

```
function automatic(answer, question){
    let points=0;
    if(question.questionType == 'choose-answer'){
        if(JSON.stringify(answer.array) ===
            JSON.stringify(question.correct)){//poredjenje nizova
            pretvaranjem u string
            points=question.maxPoints;
        }
    }
    else if(question.questionType == 'type-answer'){
        if(question.answersText.includes(answer.string)){
            points=question.maxPoints;
        }
    }
    else if(question.questionType == 'true-false'){
        for(let i=0 ; i<question.correct.length ; i++){
            if(answer.array[i] == question.correct[i]){
                points+=(question.maxPoints/question.correct.length);
            }
        }
    }
    return points;
}
```



Слика 9: Страница са резултатима теста

Изглед стране са званичним резултатима изгледа доста слично као и изглед са прелиминарним сем што ученик има увид да је професор прегледао његов тест. Такође сам преглед од стране професора је доста сличан, само што професор, односно задавач теста има могућност да промени број освојених питања, које се онда, слањем POST захтева серверу, чува као атрибут Points у шеми Result за резултат датог корисника, тј. полагача теста (полагач уопште не мора бити корисник у случају додељивања теста преко линка).

4

Закључак

Ова апликација представља један велики пројекат који још увек може бити много дорађиван. Основни циљ апликације је испуњен тиме што су урађени делови за прављење, додељивање и рађење тестова, односно квизова. Постоје велике могућности надоградње као што су на пример: додавање већег броја типова питања, омогућивање додавања слика, фајлова, дорађивање дела са групама, чување детаљније статистике о тестовима, успешности и резултатима, итд.

Frontend апликације урађен је на енглеском језику због коначног циља да апликација буде објављења на интернету и доступна свима широм света, али се веома лако може превести на српски, или било који други језик.

На самом крају желео бих да се захвалим свом ментору, Петру Радовановићу, на помоћи коју ми је пружио како приликом овог рада, тако и током три године које ми је предавао програмирање.

Литература

[1] Web Development, https://www.w3schools.com/whatis/

[2] https://developer.mozilla.org/en-US/

[3] Full Stack Web Development Course, https://www.youtube.com/playlist?list=PLZlA0Gpn_vH8jbFkBjOuFjhxANC63OmXM