|  |  |  |
| --- | --- | --- |
| https://lh3.googleusercontent.com/bp-GG9Gbeny_wpnPNjcSZybeULDTOgEKwJrlqheMuW4tbAaTIwGCY6CjnTKosyFYXLCpZjCaGoj8tVR6QqF1yhFJ44s3o_D616TZbgGuHQ63I6T1TEl90t7jV-piMsxYGvhZOcM | УНИВЕРЗИТЕТ У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ** | https://lh6.googleusercontent.com/wuhTsaU9YeguPWKYBddtOUmq5ZRI0kRmjmGOx04yuEtwe4exqyOnlQz8mAaWY8Pu337-rfYSaTvlaae9sAV7W44jtyoMUOlmuuIN0dOMuJZNCEz_bC198lRPst3eUoRQvrORVlc |

Миле Праштало

**ВЕБ АПЛИКАЦИЈА ЗА ИНТЕРНЕТ ОБРАЗОВАЊЕ**

ДИПЛОМСКИ РАД

- Основне академске студије -

Нови Сад, 2020

|  |  |  |
| --- | --- | --- |
| https://lh6.googleusercontent.com/wuhTsaU9YeguPWKYBddtOUmq5ZRI0kRmjmGOx04yuEtwe4exqyOnlQz8mAaWY8Pu337-rfYSaTvlaae9sAV7W44jtyoMUOlmuuIN0dOMuJZNCEz_bC198lRPst3eUoRQvrORVlc | УНИВЕРЗИТЕТ У НОВОМ САДУ ●**ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**  21000 НОВИ САД, Трг Доситеја Обрадовића 6 | Датум: |
|  |
| **ЗАДАТАК ЗА ИЗРАДУ ДИПЛОМСКОГ (BACHELOR) РАДА** | Лист/Листова: |
|  |

*(Податке уноси предметни наставник - ментор)*

|  |  |
| --- | --- |
| Врста студија: | ☐☒ Основне академске студије  ☐☐ Основне струковне студије |
| Студијски програм: | **Рачунарство и аутоматика** |
| Руководилац студијског програма: | **Проф. др Милан Видаковић** |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент: | **Миле Праштало** | Број индекса: | **SW44/16** |
| Област: | **Веб програмирање** | | |
| Ментор: | **Проф. др Милан Видаковић** | | |
| НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ  И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА  ДИПЛОМСКИ (Bachelor) РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:   * проблем – тема рада; * начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна; * литература | | | |

**НАСЛОВ  ДИПЛОМСКОГ  (BACHELOR)  РАДА:**

|  |
| --- |
| **Веб апликација за интернет образовање** |

**ТЕКСТ ЗАДАТКА:**

|  |
| --- |
| Задатак рада представља развој веб апликације за интернет образовање. Серверски део апликације ће бити реализован у програмском језику Котлин, коришћењем Spring Boot окружења. Клијентски део задатка ће бити реализовати коришћењем Angular 9 окружења. |

|  |  |
| --- | --- |
| Руководилац студијског програма: | Ментор рада: |
|  |  |

|  |
| --- |
| Примерак за:  ☐☐ Студента;   ☐☐ Ментора |

Образац **Q2.НА.15-04** - Издање 1

Sadržaj

[1 Uvod 5](#_Toc52820643)

[1.1 Postojeća rešenja: 5](#_Toc52820644)

[1.2 Inspiracija 5](#_Toc52820645)

[2 Opis korišćenih tehnologija 6](#_Toc52820646)

[2.1 Kotlin 6](#_Toc52820647)

[2.1.1 Prednosti Kotlina 6](#_Toc52820648)

[2.2 Spring boot 8](#_Toc52820649)

[2.2.1 Spring Data 8](#_Toc52820650)

[2.2.2 Spring Security 10](#_Toc52820651)

[2.3 Angular 10](#_Toc52820652)

[2.3.1 Angular CLI 10](#_Toc52820653)

[2.3.2 Komponente 11](#_Toc52820654)

[2.3.3 Servisi 11](#_Toc52820655)

[2.3.4 Rutiranje 11](#_Toc52820656)

[2.3.5 Quill biblioteka 11](#_Toc52820657)

[3 Specifikacija 13](#_Toc52820658)

[3.1 Class Diagram 13](#_Toc52820659)

[3.2 Use case diagram 14](#_Toc52820660)

[3.3 Database diagram 15](#_Toc52820661)

[4 Implementacija 16](#_Toc52820662)

[4.1 Login i registracija 16](#_Toc52820663)

[4.1.1 Registracija 16](#_Toc52820664)

[4.1.2 Login 18](#_Toc52820665)

[4.2 Početna strana 20](#_Toc52820666)

[4.2.1 Početna strana za profesora 20](#_Toc52820667)

[4.2.2 Početna strana za školu 21](#_Toc52820668)

[4.2.3 Početna strana za studenta 22](#_Toc52820669)

[4.3 Kreiranje kursa 22](#_Toc52820670)

[4.3.1 Tipovi kursa 23](#_Toc52820671)

[4.4 Prikaz kursa 25](#_Toc52820672)

[4.5 Kreiranje lekcije 27](#_Toc52820673)

[4.5.1 Nova lekcija 28](#_Toc52820674)

[4.5.2 Linkovana lekcija 29](#_Toc52820675)

[4.6 Prikaz lekcije 31](#_Toc52820676)

[4.6.1 Komentari 34](#_Toc52820677)

[4.7 Rad sa ocenama 38](#_Toc52820678)

[4.8 Izostanci 40](#_Toc52820679)

[4.9 Testovi 42](#_Toc52820680)

[4.9.1 Kreiranje testa 42](#_Toc52820681)

[4.9.2 Rađenje testa 46](#_Toc52820682)

[4.9.3 Ocenjivanje testa 49](#_Toc52820683)

[4.9.4 Ručno ocenjivanje testa 51](#_Toc52820684)

[5 Zaključak 53](#_Toc52820685)

[6 Literatura 54](#_Toc52820686)

[7 KLJUČNA DOKUMENTACIJSKA INFORMACIJA 55](#_Toc52820687)

[8 KEY WORDS DOCUMENTATION 58](#_Toc52820688)

# Uvod

Zadatak rada predstavlja razvoj softvera za online obrazovanje. Softver bi trebao da ima ulogu i u učenju na daljinu i kao alat prilikom regularnog izvođenja nastave.

Kako se razvija veb aplikacija, potrebno je razviti klijentski i serverski deo.

Prvo poglavlje je uvodno, u drugom se nalazi opis korišćenih tehnologija. Treće poglavlje sadrži detaljnu specifikaciju i UML dijagrame. Četvrto poglavlje predstavlja opis implementacije i delove programskog koda. U petom poglavlju se nalazi zaključak.

## Postojeća rešenja:

Softveri za online obrazovanje postoje već godinama. Na tržištu su najpopularniji

1. Google Classroom
2. TalentLMS
3. iTunes U

### Google Classroom:

Google Classroom je trenutno vodeći softver na polju online obrazovanja. U martu 2020-e godine je dostigao 100 miliona korisnika. Besplatan je za korišćenje za škole. Takođe postoje i brojne aplikacije koje se mogu integrisati sa Google Classroom-om.

### TalentLMS

TalentLMS je softver čija je uloga između softvera za online obrazovanje i online obučavanja. Softver je dostupan u besplatnoj i verziji koja se plaća. Besplatna verzija ima ograničenje od maksimum 5 korisnika po kursu i 10 kurseva po korisniku. Cena verzije koja se plaća iznosi od 59 dolara mesečno do 429 dolara mesečno.

### iTunes U

iTunes U je aplikacija koju razvija kompanija Apple. Postoji ograničenje od 12 kurseva i 50 učenika po kursu za besplatnu verziju. Takođe aplikacija je dostupna samo na uređajima koji imaju MacOS i iOS operativne sisteme.

## Inspiracija

Inspiracija za ovaj softver su prvenstveno web sajtovi edX i Coursera. To su web sajtovi na kojima najprestižniji svetski univerziteti postavljaju svoje kurseve besplatno. Takođe je kao inspiracija poslužio i Canvas kompanije Instructure koji se koristi na FTN-u.

# Opis korišćenih tehnologija

Serverski deo aplikacije je napisan u programskom jeziku Kotlin [1], verzija 1.3 korišćenjem Spring boot radnog okvira [2]. Za lakše upravljanje podacima je korišćen Spring Data [3], a za kontrolu pristupa aplikaciji je korišćen Spring Security [4].

Za implementaciju klijentskog dela aplijacije je korišćen Angular 9 [5]. Kako bi se pojednostavi rad sa Angularom, korišćen je Angular CLI (*Command Line Interface*) [6]. Za stilizovanje aplikacije je korišćen Angular Material [7]. Takođe je korišćena i biblioteka Quill, odnosno njena adaptacija za Angular ngx-quill [8].

## Kotlin

Kotlin je višeplatformski statički tipiziran programski jezik koji je razvila kompanija JetBrains. Prvi put je prikazan 2011. godine, a verzija 1 je izašla 2015. godine. Kotlin može da se koristi za programiranje:

* Mobilnih uređaja(Mobile cross-platform),
* Native – Kada ne želimo da koristimo virtuelne mašine,
* Data science,
* Server-side – Kotlin je interoperabilan sa Java bibliotekama pa je moguće koristiti web framework-e koji su razvijeni za Javu. Takođe se razvija i Ktor kao server-side framework,
* Web frontend – Kotlin može da se transpajlira u javascript i da se koristi za izradu frontend aplikacija i
* Android – Kotlin je postao dominantan jezik za izradu android aplikacija

### Prednosti Kotlina

Kotlin je moderniji programski jezik i pojednostavljuje i uvodi neke nove stvari u odnosu na Javu. Detaljnije poređenje se može pronaći na [9].

#### Type inference za promenljive

Kotlin kompajler može da prepozna tip promenljive kojoj je dodeljena vrednost prilikom inicijalizacije.

val pi = 3.14 // Double  
val e = 2.7182818284 // Double  
val eFloat = 2.7182818284f // Float, actual value is 2.7182817

#### Funkcije

Funkcije u kotlinu su first-class, što znači da se mogu smeštati u promenljive, strukture podataka, prosleđivati kao argumenti i biti povratne vrednosti funkcija višeg reda.

#### Smart casts

Ako želimo da proverimo da li je objekat instanca neke pod klase možemo da koristimo operator is. Ako koristimo operator is u if ili when uslovu objekat je automatski kastovan u pod klasu.

Primer:

Imamo klasu ChildClass koja nasleđuje klasu ParentClass i koja ima metodu „testMethod“.

Java:

ParentClass a = new ChildClass();  
if(a instanceof ChildClass){  
 ChildClass child = (ChildClass) a;  
 child.testMethod();  
}

Kotlin:

val a:ParentClass = ChildClass() //Može se i izbeći navođenje tipa  
if(a is ChildClass){  
 a.testMethod()  
}

#### Stream funkcije.

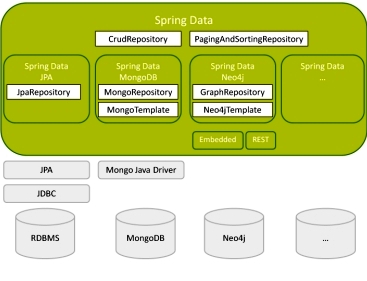
Stream funkcije poput filter, map i drugih su u Kotlinu urađene čitljivije nego u Javi. Nad kolekcijama je moguće pozivati maf, filter, reduce i druge funkcije. Rezultat poziva tih funkcija je nova lista.

## Spring boot

Spring boot je radni okvir koji služi da se pojednostavi razvoj spring aplikacija. Prilikom inicijalizacije projekta na https://start.spring.io/ Mogu da se izaberu build alati (Maven ili Gradle), programski jezik (Java, Kotlin ili Groovy) kao i verzija Spring boot-a.

### Spring Data

Spring Data je poseban projekat čiji je cilj da pojednostavi rad sa bazama podataka u Spring projektu. Omogućuje lakši rad sa SQL i NoSQL bazama podataka koristeći repozitorijume.

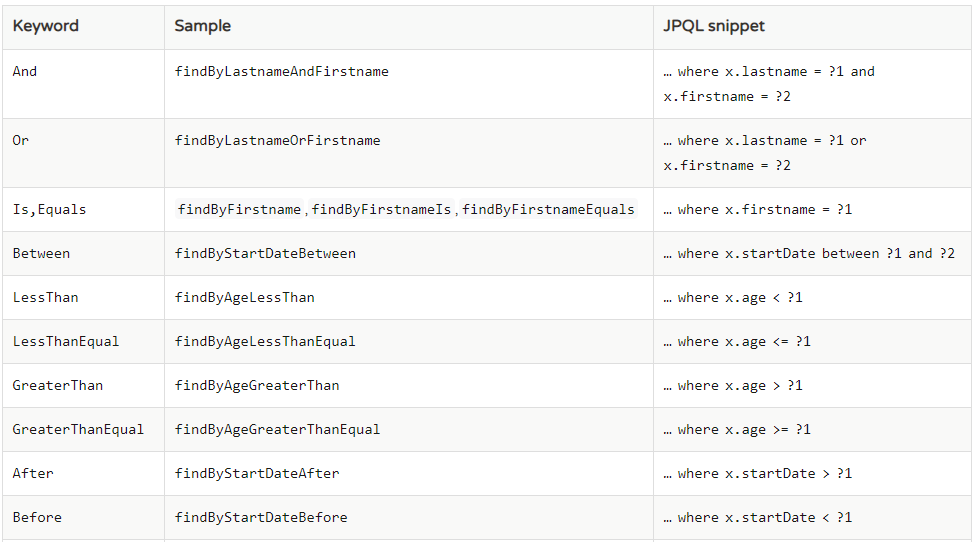


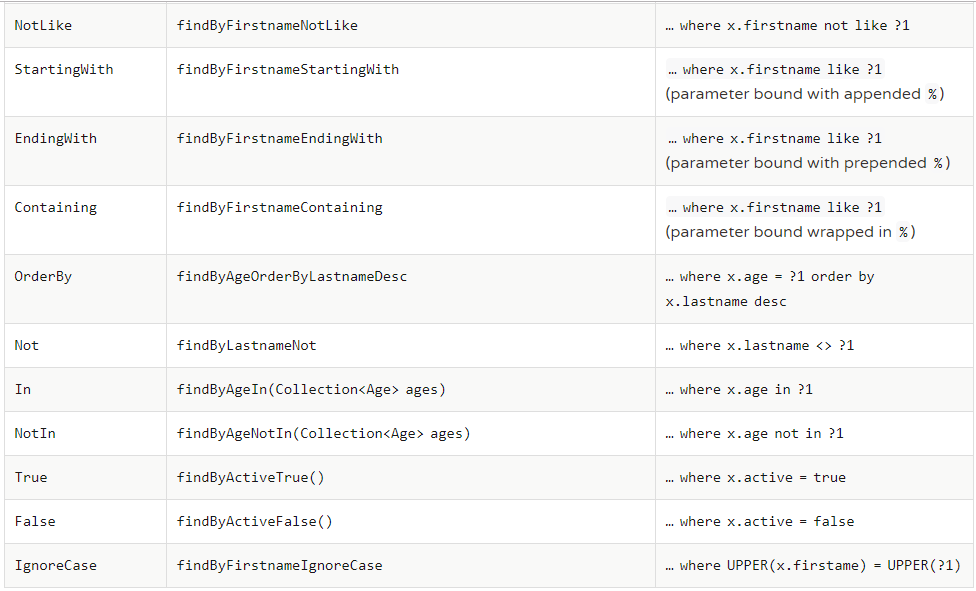
Slika 1.1 Spring Data

Spring Data JPA omogućava kompatibilnost sa JPA (Java Persistance API). Uvodi repozitorijume za rad sa entitetima. Repotitorijum je interfejs koji nasleđuje klasu Repository. Da bi definisali novi repozitorijum za entitet potrebo je kreirati novi interfejs koji će nasleđivati neku od specijalizacija interfejsa Repository i proslediti klasu entiteta i klasu id-a.  
Specijalizacije Repository interfejsa su:

* *CrudRepository* – Specijalizacija Repository interfejsa. Sadrzi CRUD  (*create, update, delete*) operacije nad entitetom,
* *PagingAndSortingRepository* – specijalizacija CRUD repozitorijuma,  omogućuje paginaciju i sortiranje i
* *JPARepository* – specijalizacija *PagingAndSorting* interfejsa. Sadrži dodatnu podršku za JPA.

Spring Data eleminiše potrebu ručnog pisanja upita za bazu podataka time što se upiti automatski generišu iz naziva metoda koje su deklarisane u interfejsu koristeći nazive atributa i ključne reči.





Nazivi funkcija u Spring Data

### Spring Security

Spring Security je radni okvir koji omogućuje autentifikaciju i autorizaciju u spring aplikacijama. Omogućava konfigurisanje zaštite pristupa REST ulaznim tačkama aplikacije specificiranjem korisničkih uloga koje imaju pristup.

Pored zaštite ulaznih tačaka, Spring Security sadrži i password encoder.

## Angular

Angular je radni okvir za razvoj klijentskih veb aplikacija. Razvijen je od strane Google-a, nastao je temeljima Angular JS, sa kojim ima malo sličnosti. Inicijalna verzija 2.0 je predstavljena 2016. godine. Trenutnoje najnovija verzija verzija 10.1. U ovoj projektu je korišćena verzija 9, prvenstveno zbog kompatibilnosti sa bibliotekama.

### Angular CLI

Angular CLI je alat koji se koristi iz komandne linije. Ubrzava razvijanje Angular aplikacija tako što pojednostavljuje kreiranje novog projekta, novih komponenti i servisa. Takođe olakšava održavanje sa build i lint funkcijama.

### Komponente

Komponente su osnovni gradivni blokovi Angular aplikacije. Komponente se sastoje od TypeScript klase, Html fajla i CSS fajla. Komponenta u TS fajlu je obeležena sa @Component dekoratorom. Takođe se u TS fajlu nalaze i:

* *selector* – naziv HTML elementa koji koristimo prilikom kreiranja komponente,
* *templateUrl* – Putanja do HTML template fajla
* *styleUrls* – Putanja do CSS fajla koji se primenjuju za tu komponentu

### Servisi

Angular servisi su singleton objekti koji se instanciraju samo jednom tokom života aplikacije. Osnovna uloga servisa je da organizuju poslovnu logiku, podatke i funkcionalnosti između različitih komponenti.

### Rutiranje

Rutiranje služi za navigaciju između komponenti. Na osnovu URL adrese se određuje koja komponenta će se učitati. Putanje se specificiraju u app-routing.module.ts fajlu. Prikaz dela putanja u projektu se nalazi ispod.

const routes: Routes = [

  {path: '', component: IndexComponent, canActivate: [NotLoggedInGuard]},

  {path: 'login', component: LoginComponent, canActivate: [NotLoggedInGuard]},

  {path: 'register', component: RegisterComponent, canActivate: [NotLoggedInGuard]},

  {path: 'home', component: HomeComponent, canActivate: [LoggedInGuard]},

  {path: 'create-course', component: CreateCourseComponent, canActivate: [LoggedInGuard]},

  {path: 'edit-course/:id', component: EditCourseComponent, canActivate: [LoggedInGuard]},

  {path: 'course/:id', component: CourseDetailsComponent, canActivate: [LoggedInGuard]},

  {path: 'school/:id', component: SchoolDetailsComponent, canActivate: [LoggedInGuard]}

];

### Quill biblioteka

Quill editor je besplatan open source text editor. Prilikom izrade aplikacije je korišćen ngx-quill, wrapper oko Quill biblioteke za Angular. Koristi se tako što se u html fajlu doda element „quil-editor“. To je dovoljno da se kreira polje za unos teksta sa toolbar-om. Ovo će kreirati editor sa svim elementima toolbar-a. Moguće je definisati toolbar samo sa određenim elementima. U tom slučaju je elementu potrebno proslediti elemente toolbara. Svaka promena prilikom pisanja ili menjanja teksta pokreće događaj „onContentChanged“. U objektu koji se prosledi funkciji se nalazi i sadržaj editora u html formatu.

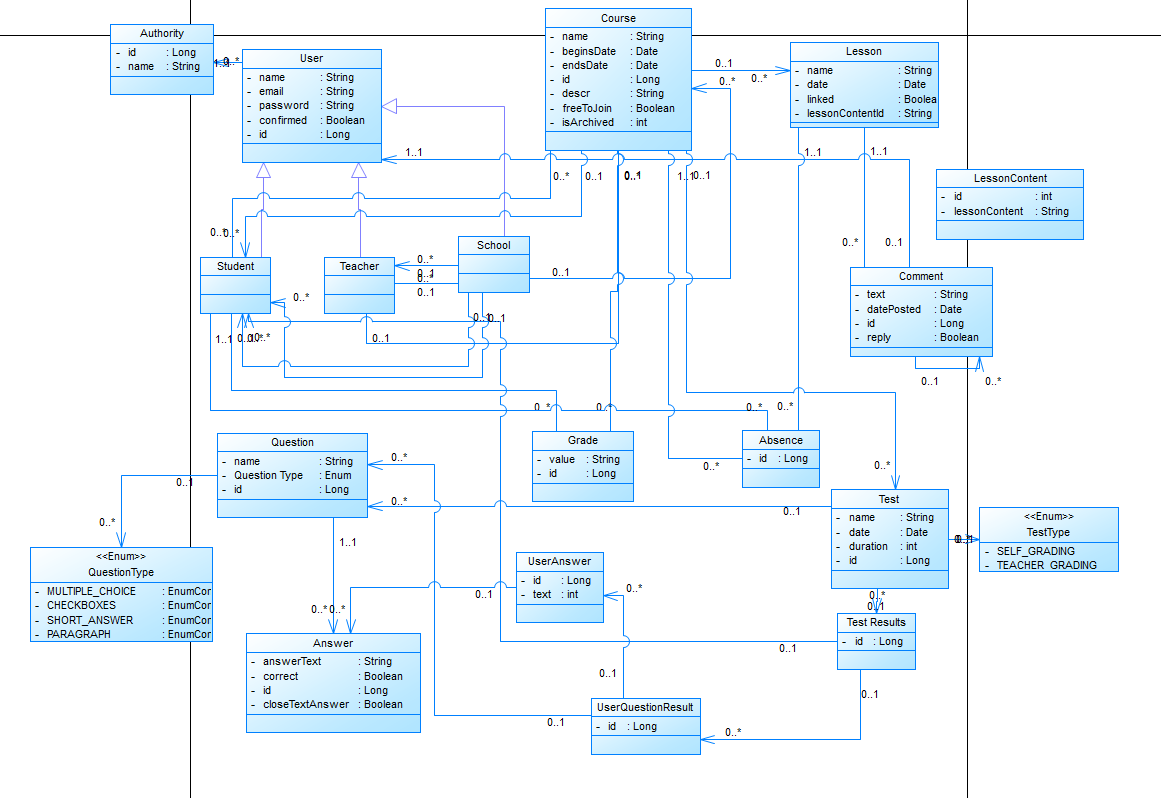
<quill-editor (onContentChanged)="quillEditorChange($event)" ()"></quill-editor>

# Specifikacija

Zadatak obuhvata izradu sistema za online obrazovanje. Osnovne funkcionalnosti su:

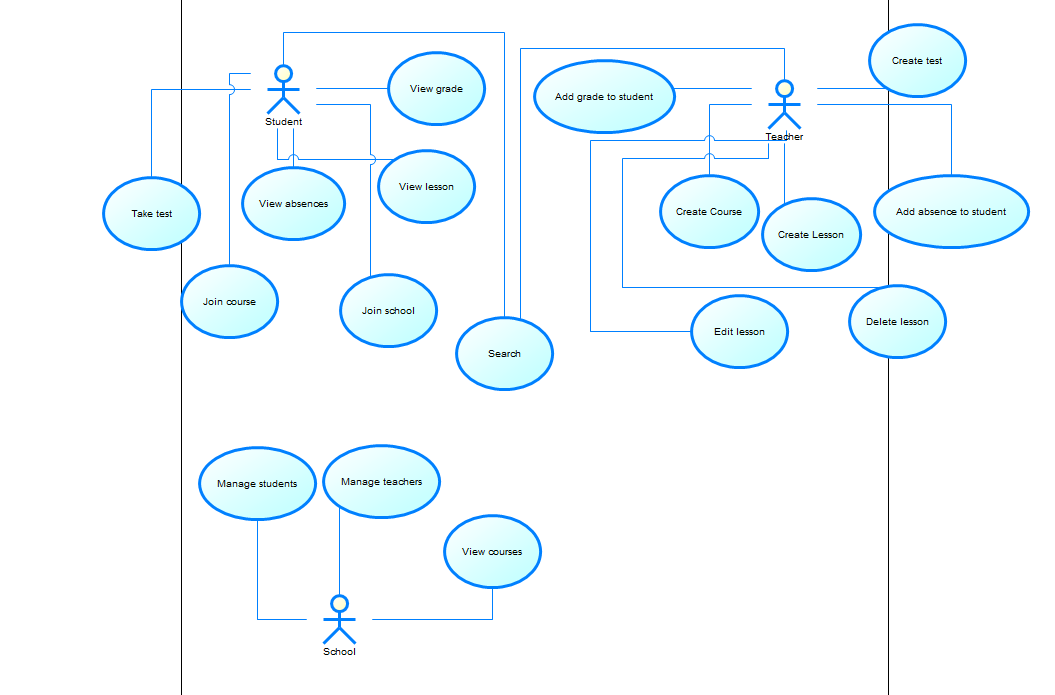
* Kreiranje i pregled kursa,
* Kreiranje i pregled lekcije
* Unos ocena,
* Unos izostanaka i
* Rad sa testovima

## Class Diagram



Class diagram modela podataka

## Use case diagram



Use case diagram

Prikazane su osnovne funkcionalnosti.

Student može da:

* Pristupi kursu,
* Uradi test,
* Vidi lekciju,
* Vidi ocenu i
* Vidi odsustvo

Nastavnik može da

* Kreira kurs,
* Kreira test,
* Upravlja lekciju,
* Unosi ocene i
* Unosi odsustva

Škola može da:

* Upravlja studentima,
* Upravlja nastavnicima i
* Vidi kurseve

## Database diagram

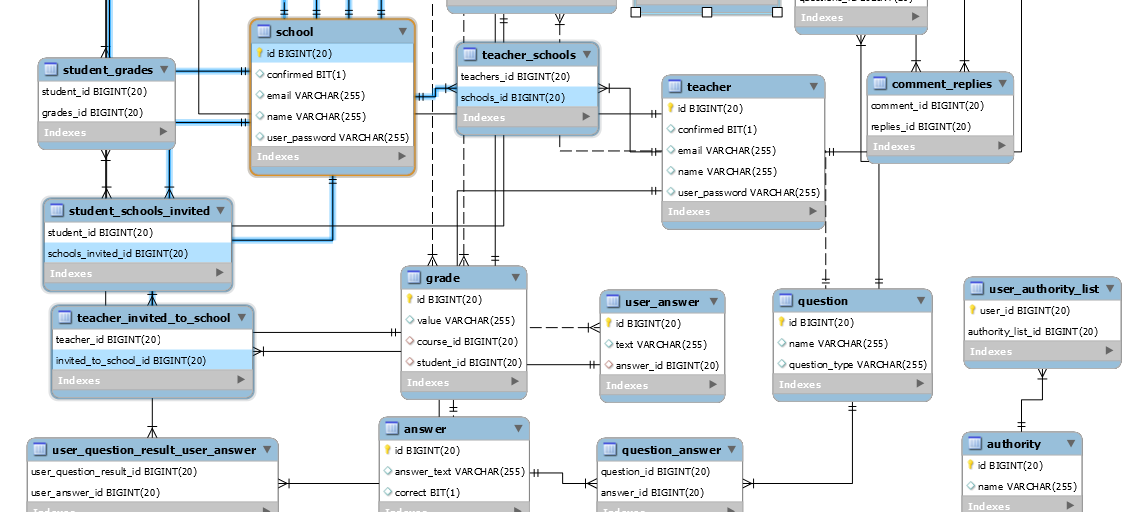
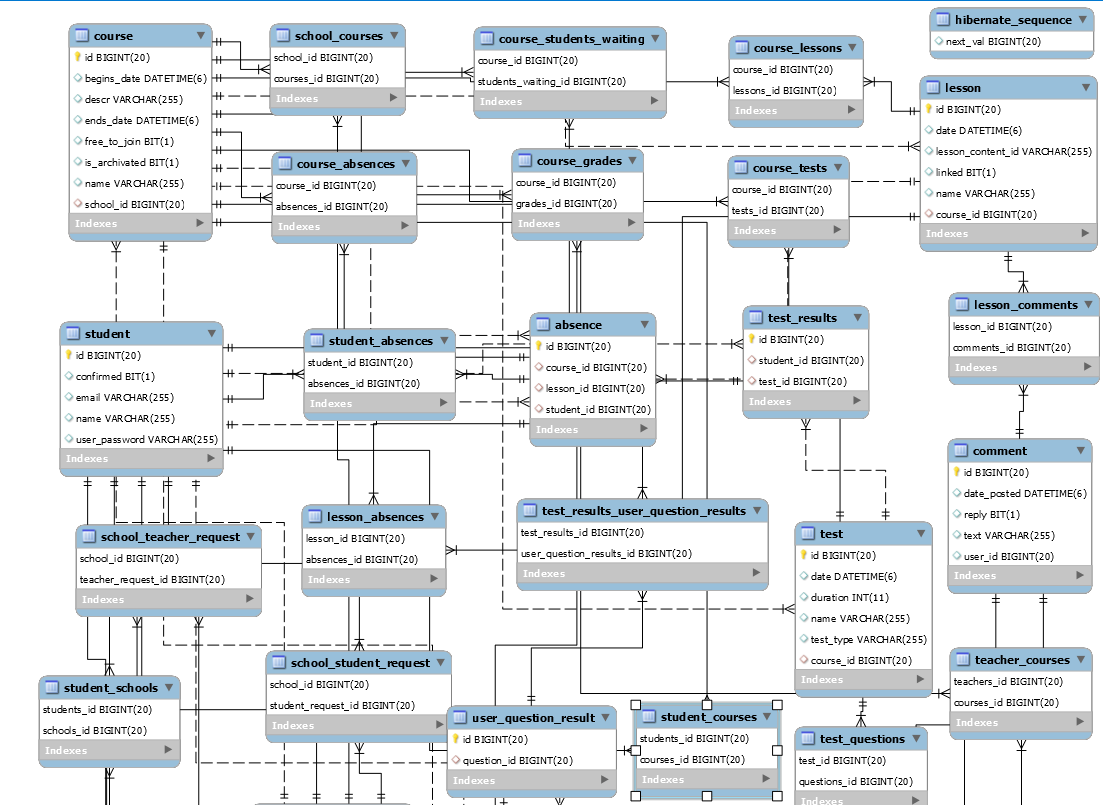
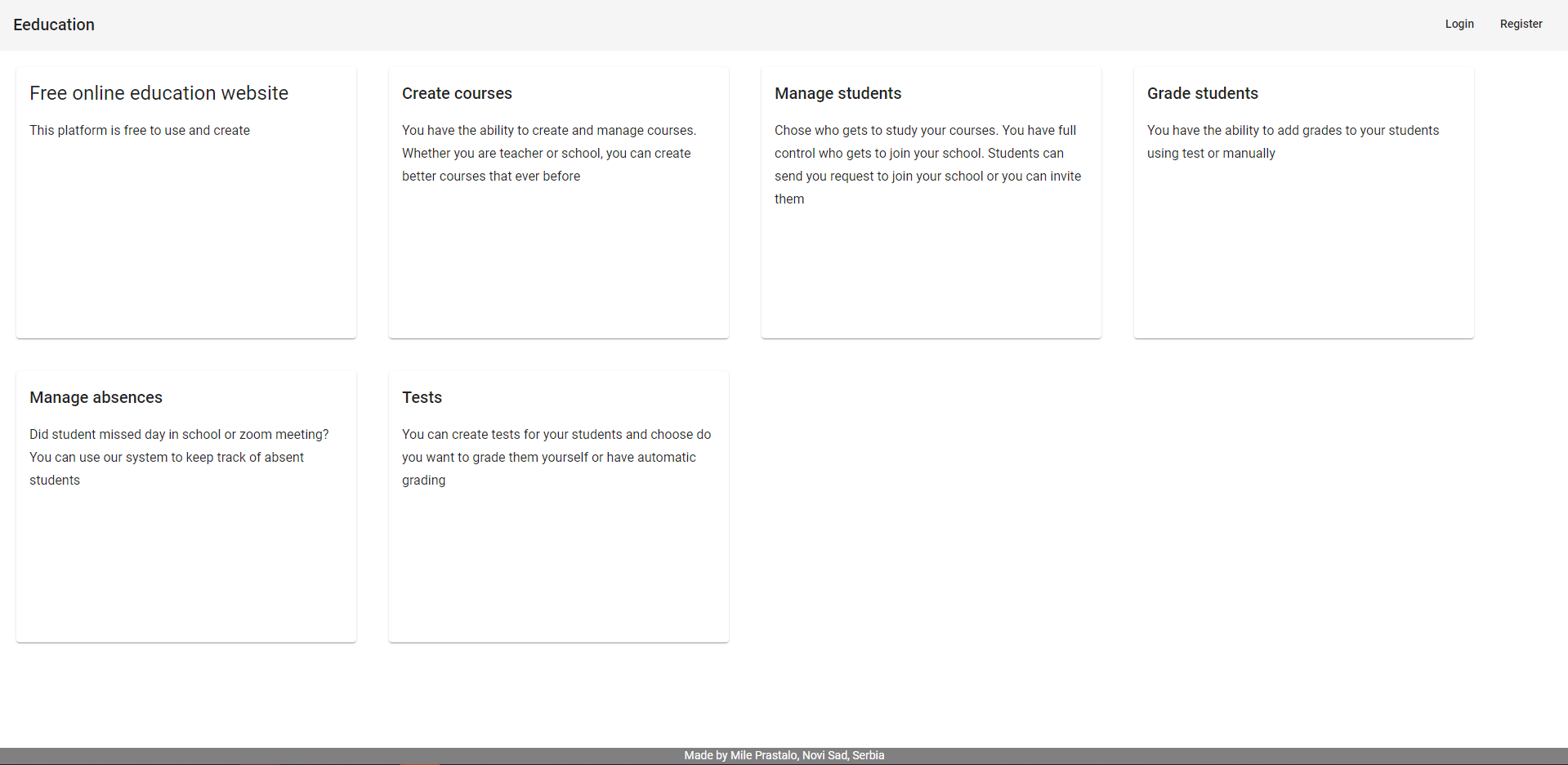


Diagram šeme baze podataka

# Implementacija

## Login i registracija

Kako bi korisnik mogao da pristupi funkcionalnostima aplikacije potrebno je da ima nalog u sistemu. Neregistrovani korisnik ima pristup početnoj stranici na kojoj se nalazi promotivni sadržaj, login stranici i stranici za registraciju.



Početna stranica

Klikom na dugme login korisnik je prebačen na formu za logovanje, a klik na dugme register ga prebacuje na formu za registraciju.

### Registracija

U formi za registraciju korisnik unosi svoje podatke. Potrebno je da unese svoje

* ime,
* email adresu,
* šifru,
* zatim da ponovi šifru,
* izabere državu i
* izabere tip korisnika

Tipovi korisnika mogu biti:

* Student,
* Nastavnik (Teacher) i
* Škola (School)

Nakon što korisnik unese sve podatke i pritisne dugme za registraciju, poziva se metoda register iz Authentication Controller-a. Ispod je prikazan deo AuthenticationController-a sa metodom register.

@RestController  
@RequestMapping("auth")  
@CrossOrigin  
class AuthenticationController(private val authenticationService: AuthenticationService) {  
  
 @PostMapping(value = ["register"], consumes = [MediaType.*APPLICATION\_JSON\_VALUE*], produces = [MediaType.*APPLICATION\_JSON\_VALUE*])  
 fun register(@RequestBody registrationRequest: RegistrationRequest): ResponseEntity<RegistrationResponse> {  
 val response = authenticationService.register(registrationRequest)  
 return ResponseEntity(RegistrationResponse(response), HttpStatus.OK)  
 }  
}

Metoda za registraciju u kontroleru

Metoda u kontroleru poziva metodu u Authentication Service-u i od nje kao povratnu vrednost dobija string sa porukom u slučaju uspešne registracije.

@Service  
class AuthenticationService(private var authenticationManager: AuthenticationManager,  
 private var jwtUserDetailsService: JwtUserDetailsService,  
 private val jwtTokenUtil: JwtTokenUtil,  
 private val authorityRepository: AuthorityRepository,  
 private val schoolService: SchoolService,  
 private val teacherService: TeacherService,  
 private val studentService: StudentService,  
 private val userService: UserService,  
 private val emailService: EmailService) {  
  
  
 fun register(registrationRequest: RegistrationRequest): String {  
 val authorities = HashSet<Authority>()  
 val bc = BCryptPasswordEncoder()  
 return when (registrationRequest.userType) {  
 UserType.SCHOOL -> {  
 authorities.add(authorityRepository.findByName("SCHOOL")!!)  
 var school = School(null, registrationRequest.email, bc.encode(registrationRequest.password), registrationRequest.name, authorities, false, HashSet(), HashSet(), HashSet(), HashSet(), HashSet())  
 school = schoolService.save(school)  
 emailService.sendRegistrationEmail(school)  
 "School has been successfully registered"  
 }  
 UserType.TEACHER -> {  
 authorities.add(authorityRepository.findByName("TEACHER")!!)  
 var teacher = Teacher(null, registrationRequest.email, bc.encode(registrationRequest.password), registrationRequest.name, authorities, false, HashSet(), HashSet(), HashSet())  
 teacher = teacherService.save(teacher)  
 emailService.sendRegistrationEmail(teacher)  
 "Teacher has been successfully registered"  
 }  
 UserType.STUDENT -> {  
 authorities.add(authorityRepository.findByName("STUDENT")!!)  
 var student = Student(null, registrationRequest.email, registrationRequest.name, bc.encode(registrationRequest.password), authorities, false, HashSet(), HashSet(), HashSet(), HashSet(), HashSet())  
 student = studentService.save(student)  
 emailService.sendRegistrationEmail(student)  
 "Student has been successfully registered"  
 }  
 }  
 }  
 }  
 }

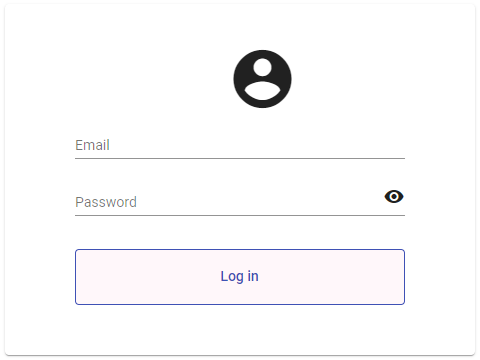
}

Metoda register u authentication service-u

Nakon što je korisnik uspešno registrovan, na email adresu mu stiže link na koji treba da klikne kako bi potvrdio svoj nalog. Nakon čega može da se uloguje na sistem.

### Login

Ukoliko korisnik ima nalog, potrebno je da se uloguje. Forma za loin se sastoji od polja za email adresu i polja za lozinku.



Forma za login

Nakon što korisnik pritisne dugme za login poziva se metoda onLoginSubmit na klijentskoj strani u fajlu login.component.ts

onLogInSubmit() {  
 const loginData = new AuthenticationRequest(this.email, this.password);  
 this.authenticationService.login(loginData).subscribe(  
 (response => {  
 if (response != null) {  
 ***localStorage***.setItem('token', response.token);  
 this.snackBar.open('Logged In successfully.');  
 const jwt: JwtHelperService = new JwtHelperService();  
 const info = jwt.decodeToken(response.token);  
 ***console***.log(info);  
 const role = info.role[0].authority;  
 ***localStorage***.setItem('role', info.role[0].authority);  
 ***localStorage***.setItem('id', ***String***(response.user.id));  
 this.snackBar.open('Logged In successfully.');  
  
 if (role === 'SCHOOL') {  
 this.router.navigateByUrl('/school-home');  
 } else if (role === 'STUDENT') {  
 this.router.navigateByUrl('/student-home');  
 } else if (role === 'TEACHER') {  
 this.router.navigateByUrl('/teacher-home');  
 }  
 }  
 }),  
 (error => {  
 this.snackBar.open(error.error.message);  
 }));  
}

Login frontend

Metoda login u Authentication Service-u na klijentskoj strani poziva metodu login u AuthenticationController-u

@RestController  
@RequestMapping("auth")  
@CrossOrigin  
class AuthenticationController(private val authenticationService: AuthenticationService) {  
  
 @PostMapping(value = ["login"], consumes = [MediaType.*APPLICATION\_JSON\_VALUE*], produces = [MediaType.*APPLICATION\_JSON\_VALUE*])  
 fun login(@RequestBody authenticationRequest: AuthenticationRequest): ResponseEntity<AuthenticationResponse> {  
 val token = authenticationService.login(authenticationRequest)  
 val user = authenticationService.getUserByEmail(authenticationRequest.email)  
 val userDto = UserBasicDto(user.id, user.email, user.name)  
 val authenticationResponse = AuthenticationResponse(token, userDto)  
 return ResponseEntity(authenticationResponse, HttpStatus.OK)  
 }

}

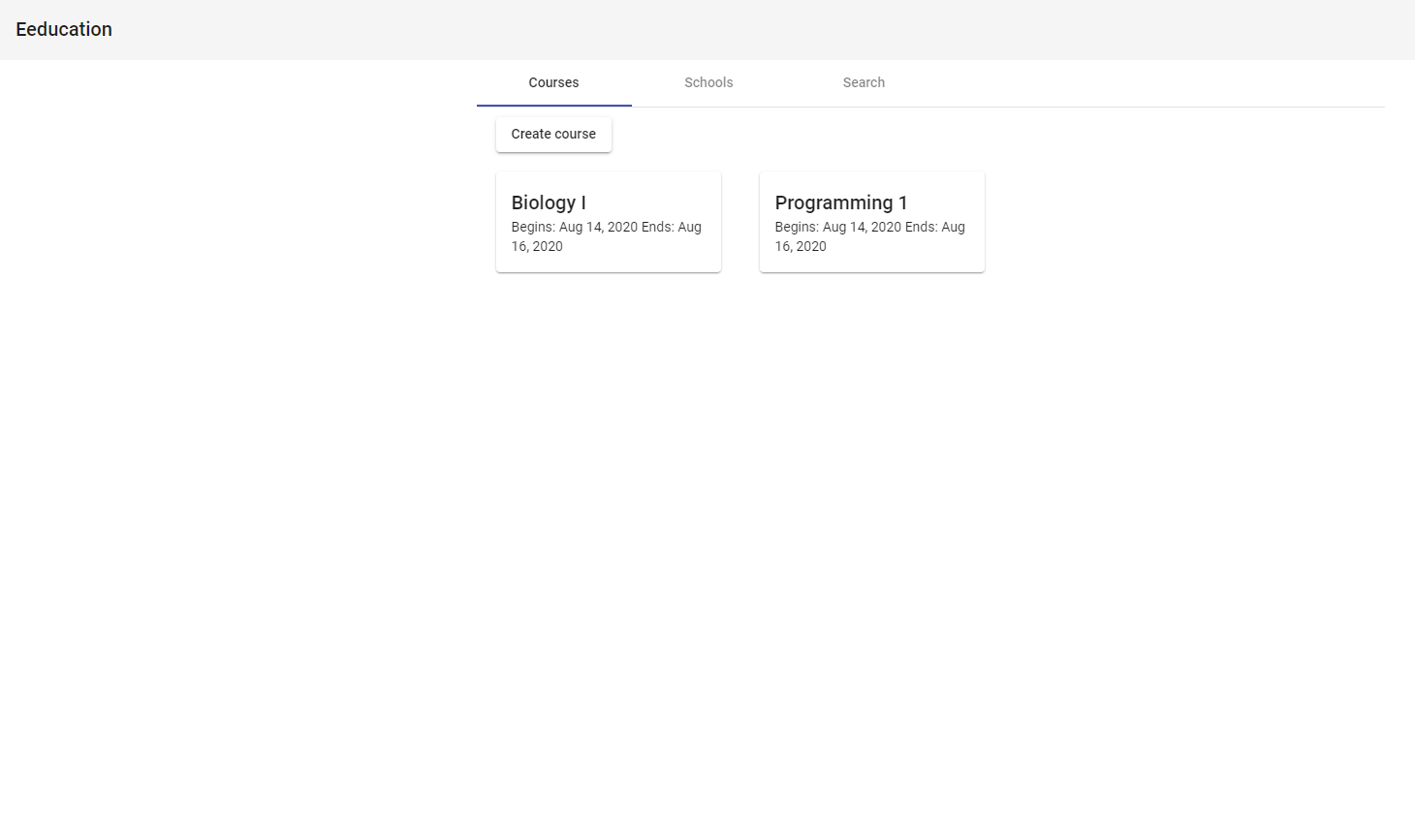
Login backend

Metoda login u Authentication Service-u na serverskoj strani proverava da li korisnik postoji, yatim generiše JSON Web Token koji pored osnovih podataka sadrži i ulogu korisnika. Kao odgovor se šalje objekat tipa AuthenticationResponse koji sadrži token i osnovne podatke o korisniku (id, email i ime). Kada odgovor stigne klijentu, klijent sačuva token, ulogu i id korisnika u local storage i na osnovu uloge korisnik je preusmeren na početnu stranu.

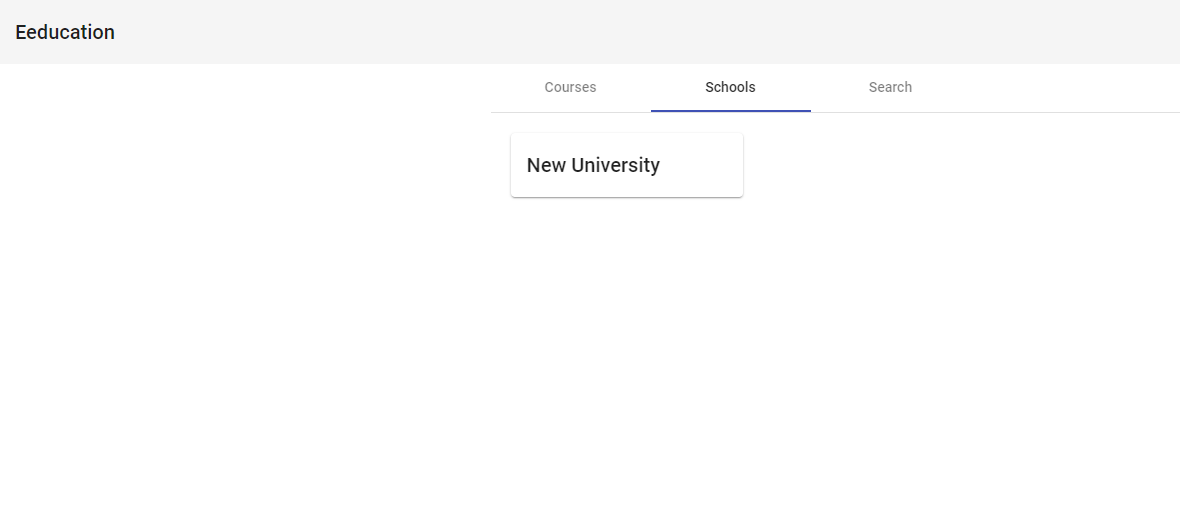
## Početna strana

### Početna strana za profesora

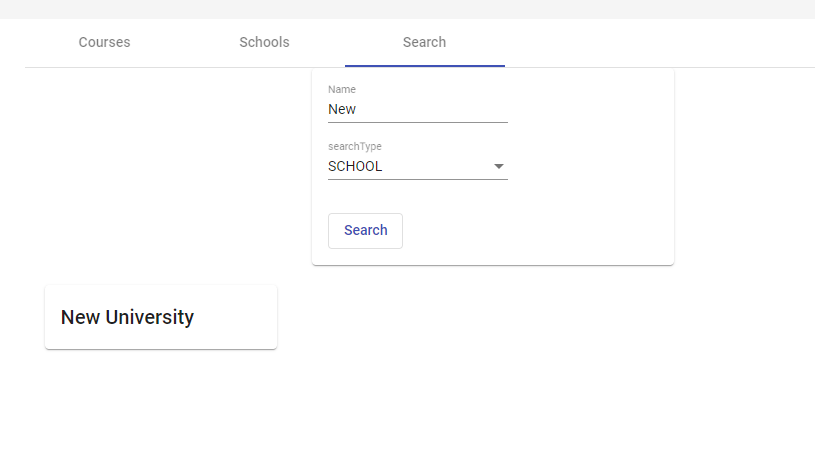
Nastavnik nakon uspešno logovanja vidi tab sa kursevima gde može da vidi sve kurseve na kojima je prijavljen. Pored taba sa kursevima se nalazi tab sa Školama gde nastavnik može da vidi škole u kojima predaje. Pored škola se nalazi tab sa pretragom.



Pregled Kurseva.



Pregled škola.

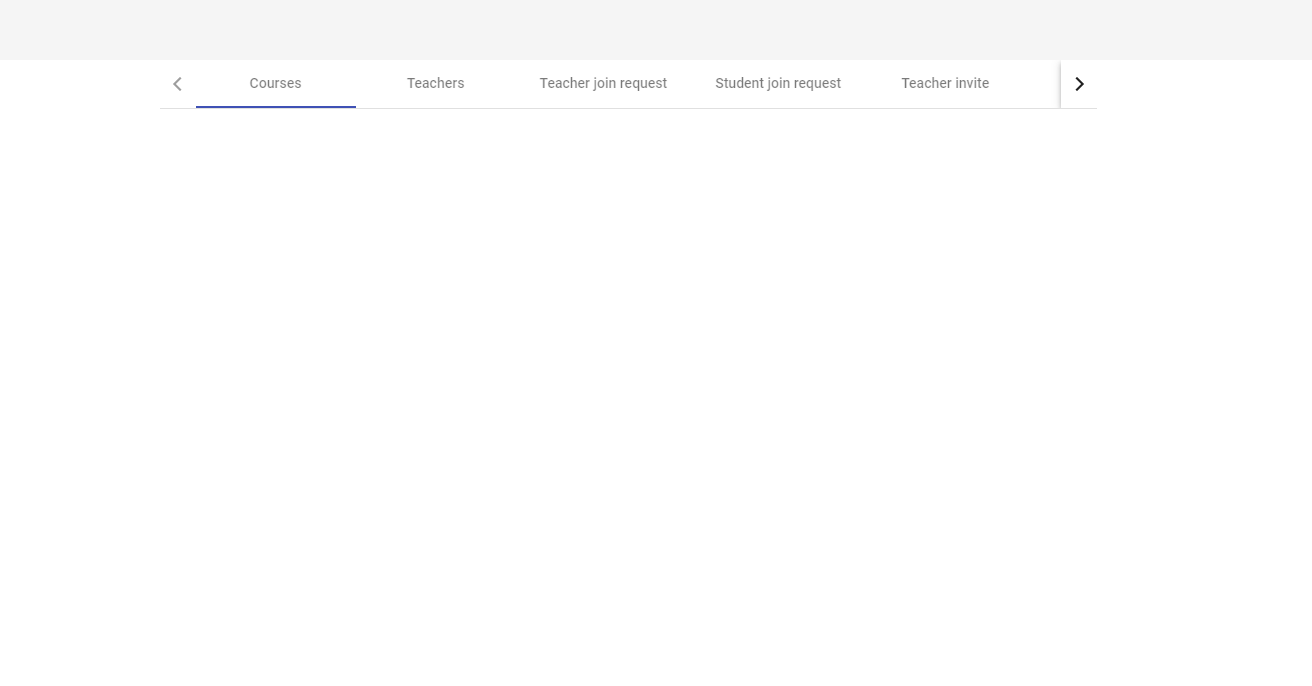


Pretraga:

### Početna strana za školu

Škola nakon uspešnog logovanja vidi tabove

* Courses – Škola može da vidi listu svojih kurseva,
* Teachers – Lista nastavnika koji su deo škole,
* Teacher join request – Lista nastavnika koji su poslali zahtev za učlanjenje u školu. Škola može da prihvati ili da odbije zahtev,
* Student join request - Lista studenata koji su poslali zahtev za učlanjenje u školu. Škola može da prihvati ili da odbije zahtev,
* Teacher invite – Škola može da pošalje pozivnicu nastavniku i
* Student invite – Škola može da pošalje pozivnicu studentu



Početna stranica za školu

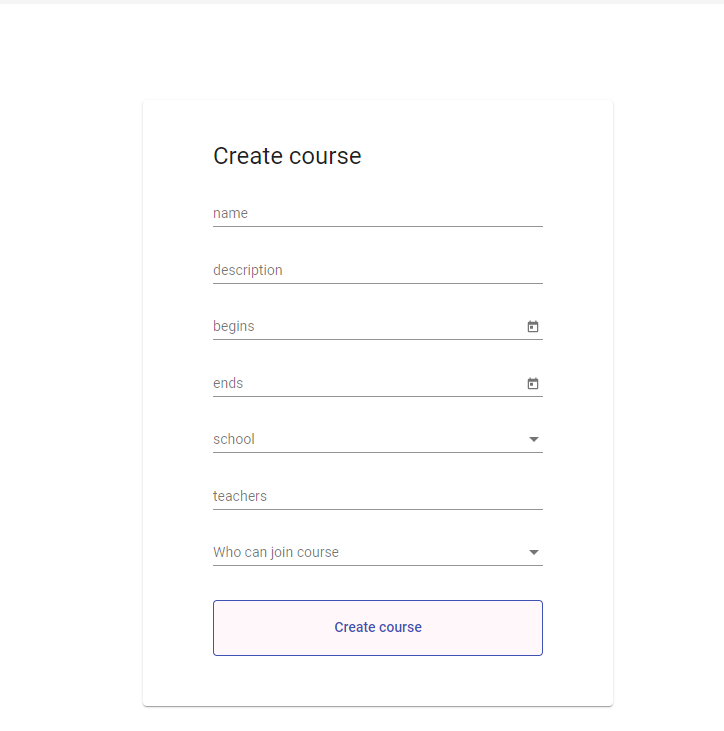
### Početna strana za studenta

Student nakon uspešno logovanja vidi tab sa kursevima gde može da vidi sve kurseve na kojima je prijavljen. Pored taba sa kursevima se nalazi tab sa Školama gde student može da vidi škole koje pohađa. Pored škola se nalazi tab sa ocenama gde student može da vidi svoje ocene grupisane po kursu. Pored Ocena se nalazi pretraga gde student može da pretraži škole i kurseve.

## Kreiranje kursa

Prilikom kreiranja kursa, potrebno je uneti:

* Naziv kursa,
* Opis kursa,
* Datum kada kurs počinje,
* Datum kada se kurs završava,
* Iz liste škola potrebno je izabrati školu kojoj kurs pripada, ili opciju „None“, ako se kreira kurs koji ne pripada ni jednoj školi,
* Potom dodati listu nastavnika, tako što će se u polje „teachers“ upisati email adrese nastavnika, odvojene zarezom i
* Takođe je potrebno izabrati tip kursa, odnosno, ko može da se pridruži kursu



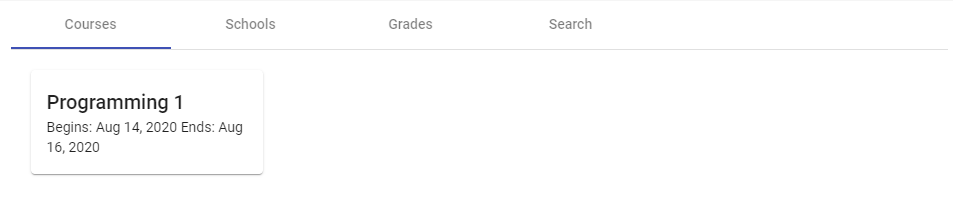
Forma za kreiranje kursa

### Tipovi kursa

Postoje dva tipa kursa. Prvi tip je kurs gde studenti mou da se pridruže bez odobrenja. Drugi tip je kurs gde nastavnik može da izabere da li će da prihvati ili da odbije studentov zahtev.

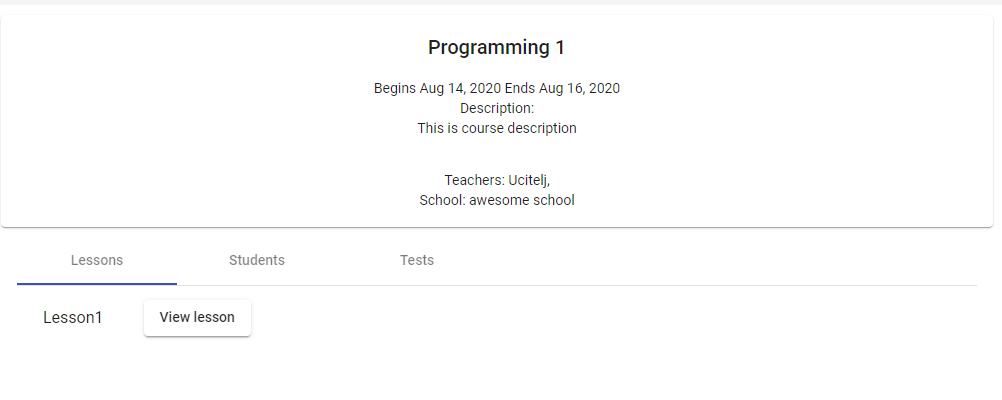
## Prikaz kursa

Da bi se došlo do deteljnog prikaza kursa sa početne stranice, potrebno je da se klikne na karticu na kojoj se nalazi naziv kursa.



Prikaz kurseva

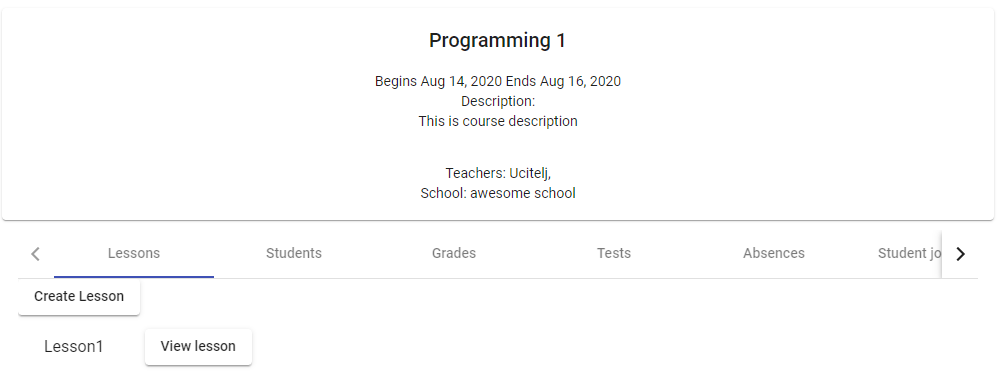
Prikaz kursa se razlikuje za studenta i učitelja. Student ima pristup lekcijama, može da vidi druge studente i polaže test.



Početna stranica kursa za studenta.

Nastavnik ima više opcija za rad sa kursom.

* Može da kreira i pregleda lekcije, vidi listu studenata,
* dodaje i vidi ocene,
* kreira i pregleda testove,
* kreira i vidi izostanke, i
* odobrava zahteve studenata da se pridruže ako je potrebno odobrenje za prisup kursu.



Početna stranica kursa za nastavnika

## Kreiranje lekcije

Nastavnik ima opciju da kreira lekciju. Prilikom kreiranja lekcije nastavnik unosi:

* naziv,
* datum i
* da li kreira novu lekciju ili sadržaj preuzima iz neke ranije lekcije

Bez obzira koji tip lekcije nastavnik izabere, poziva se metoda onLessonSubmit u create-lesson.component.ts u Angular projektu.

onLessonSubmit() {  
 let lessonPath: string = null;  
 let createLesson: CreateLesson = null;  
 if (this.lessonType === 'LINK') {  
 lessonPath = this.link;  
 createLesson = new CreateLesson(this.courseId, '', this.name, this.date, lessonPath);  
 } else {  
 createLesson = new CreateLesson(this.courseId, this.htmlContentFromQuill, this.name, this.date, lessonPath);  
 }  
 this.lessonService.createLesson(createLesson).subscribe(  
 response => {  
 this.router.navigateByUrl('course/' + this.courseId);  
 }  
 );  
}

onLessonSubmit metoda

Potom komunikacija teče to LessonControllera na serverskoj strani.

@RestController  
@RequestMapping(value = ["api/lessons"], produces = [MediaType.*APPLICATION\_JSON\_VALUE*])  
@CrossOrigin  
class LessonController(private val lessonService: LessonService) {  
  
 //Create lesson  
 @PostMapping(consumes = [MediaType.*APPLICATION\_JSON\_VALUE*])  
 @PreAuthorize("hasAuthority('TEACHER')")  
 fun createLesson(@RequestBody createLessonDto: CreateLessonDto): ResponseEntity<LessonDto> {  
 val lesson = lessonService.createLesson(createLessonDto)  
 val dto = LessonDto(lesson, createLessonDto.lessonContent,lesson.lessonContentId)  
 return ResponseEntity(dto, HttpStatus.OK)  
 }

}

Deo LessonControllera

Metoda u kontroleru poziva metodu createLesson u LessonService klasi koja čuva lekciju u bazi podataka.

fun createLesson(createLessonDto: CreateLessonDto): Lesson {  
 val course = courseService.findById(createLessonDto.courseId)  
  
 val isLinked = createLessonDto.lessonPath != null  
  
 val lessonContent: LessonContent = if (createLessonDto.lessonPath == null) {  
 val content = LessonContent(null, createLessonDto.lessonContent)  
 lessonContentRepository.save(content);  
  
 } else {  
 lessonContentRepository.findById(createLessonDto.lessonPath!!).orElseThrow **{** EntityNotFoundException("Lesson path not found") **}** }  
 userService.teacherTeacherCourse(userService.getLoggedInUser()!!, course)  
 val lesson = save(Lesson(null, course, HashSet(), HashSet(), createLessonDto.name, createLessonDto.date, lessonContent.id!!, isLinked))  
 (course.lessons as MutableSet).add(lesson)  
 courseService.save(course)  
  
 return lesson  
}

Lesson Service.

U metodi createLesson, kreira se objekat klase Lesson. Prvo se u slučaju da lekcija nije linkovana, sadržaj lekcije doda u MongoDB bazu podataka. Potom se objekat čuva u MySql bazi podataka, gde je lessonContentId id u MongoDb bazi podataka.

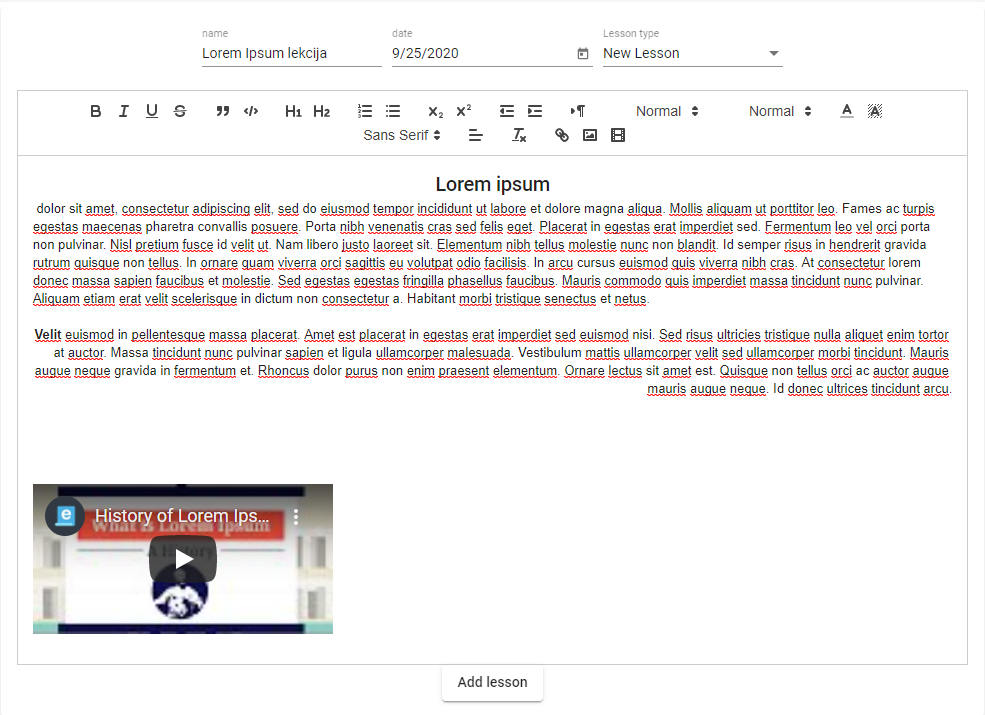
@Entity  
class Lesson(@Id  
 @GeneratedValue  
 var id: Long?,  
 @ManyToOne  
 var course: Course,  
 @OneToMany  
 var absences: Set<Absence>,  
 @OneToMany  
 var comments: Set<Comment>,  
 @Column  
 var name: String,  
 @Column  
 var date: Date,  
 @Column  
 var lessonContentId: String,  
 @Column  
 var linked: Boolean?  
) {  
}

Lesson klasa

### Nova lekcija

Ukoliko nastavnik odluči da kreira novu lekciju, dostupan mu je editor za unos teksta i toolbar za njegovo stilizovanje. Neke od funkcija editora su:

* Podebljan (bold) tekst,
* Italic
* Underline
* Citat,
* Podešavanje veličine teksta,
* Izbor fonta,
* Alignment teksta,
* Unos slike i
* Unos videa

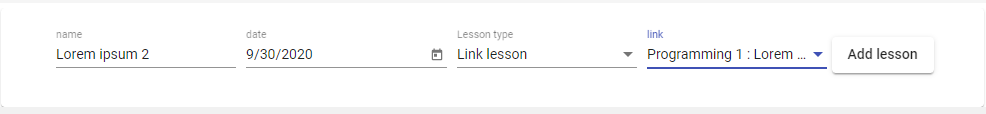


Kreiranje nove lekcije.

### Linkovana lekcija

Ukoliko nastavnik želi studentima da podeli lekciju iz nekog ranijeg kursa, to može da uradi putem linkovane lekcije.

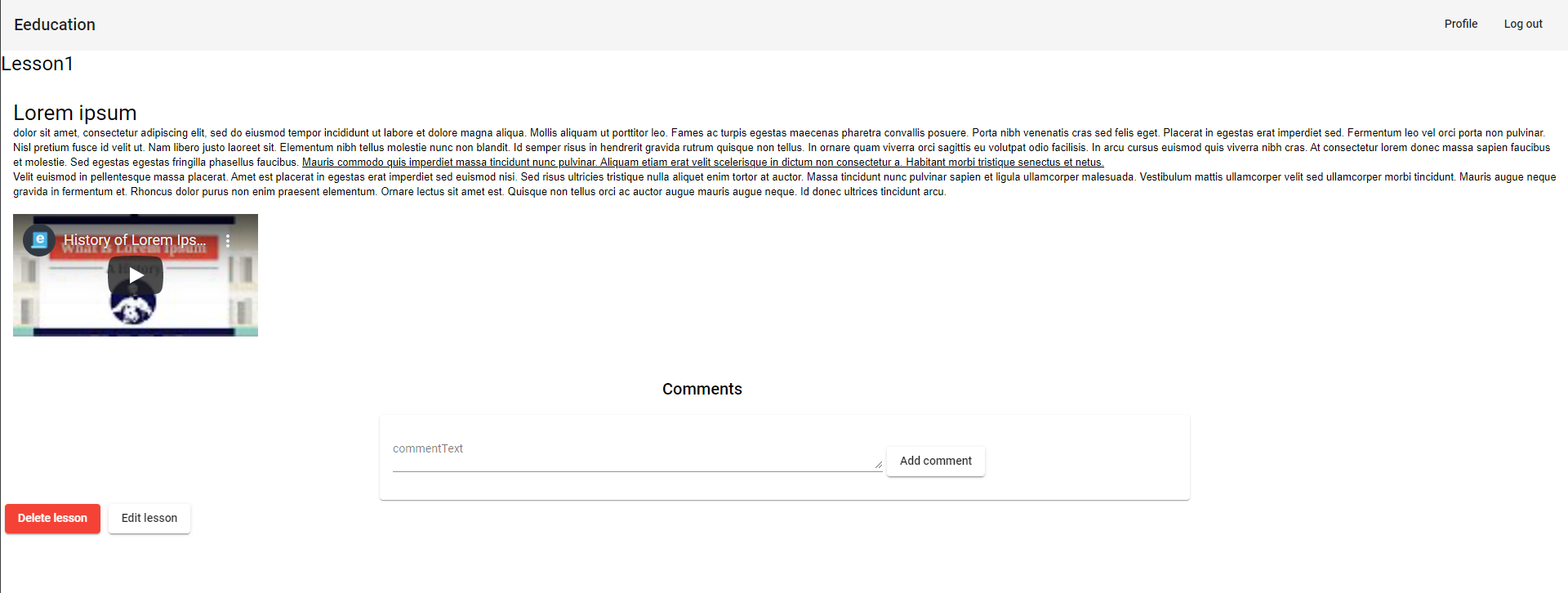
Nastavnik može da podeli samo lekcije koje se nalaze u nekom od kurseva u koje je nastavnik uključen.



Unos linkovane lekcije

## Prikaz lekcije

Nakon klika na *view lesson* prelazi se na stranicu za pregled lekcije.



Prikaz lekcije

Stranica za prikaz lekcije uključuje

* Naslov lekcije,
* Sadržaj lekcije,
* Prikaz komentara,
* Polje za unos komentara,
* Dugme za brisanje lekcije i
* Dugme za izmenu lekcije koje je dostupno samo u koliko lekcija nije liknovana.

<app-navbar></app-navbar>  
<div>  
 <h1>{{lesson.name}}</h1>  
 <quill-view [content]="content" theme="snow"></quill-view>  
 <div>  
 <h2 class="center">Comments</h2>  
 <div class="comment" *\*ngFor*="let ***comment*** of lesson.comments"><app-comment *\*ngIf*="!***comment***.reply" [lesson]="lesson"  
 [comment]="***comment***"></app-comment></div>  
 </div>  
 <div>  
 <mat-card class="create-lesson-card">  
 <form class="form" [formGroup]="addCommentForm" (ngSubmit)="addCommentForm.valid && onCommentSubmit()">  
 <mat-form-field class="field-full-width">  
 <textarea id="commentText"  
 matInput  
 formControlName="commentText"  
 placeholder="commentText"></textarea>  
 <mat-error>name can not be blank.</mat-error>  
 </mat-form-field>  
 <button mat-raised-button>Add comment</button>  
 </form>  
 </mat-card>  
 </div>  
 <div class="bottom">  
 <button *\*ngIf*="teacherPartOfCourse()" mat-raised-button color="warn" (click)="delete()">Delete lesson</button>  
 <button *\*ngIf*="teacherPartOfCourse() && (!lesson.linked)" mat-raised-button (click)="edit()">Edit lesson</button>  
 </div>  
</div>

Izgled html stranice za prikaz lekcije.

Nakon učitavanja stranice sa lekcijom, poziva se metoda ngOnInit() na klijentskoj strani. Iz putanje se dobija id lekcije, nakon toga se šalje zahtev serveru za dobavljanje podataka o lekciji.

export class LessonComponent implements OnInit {  
 lesson: Lesson;  
 lessonId: number;  
 addCommentForm: FormGroup;  
 course: Course;  
 content: string;  
  
 constructor(private lessonService: LessonService,  
 private router: Router,  
 private route: ActivatedRoute,  
 private formBuilder: FormBuilder,  
 private commentService: CommentService,  
 private courseService: CourseService,  
 private visibilityService: VisibilityService) {  
 }  
  
 get commentText() {  
 return this.addCommentForm.controls.commentText.value as string;  
 }  
  
 ngOnInit(): void {  
 this.addCommentForm = this.formBuilder.group({  
 commentText: ['', [  
 Validators.*required* ]]  
 });  
 this.lessonId = ***Number***(this.route.snapshot.paramMap.get('id'));  
 this.lessonService.getLesson(this.lessonId).subscribe(  
 response => {  
 this.lesson = response;  
 this.content = this.lesson.lessonContent;  
 this.courseService.getCourse(this.lesson.courseId).subscribe(  
 courseResponse => {  
 this.course = courseResponse;  
 }  
 );  
  
 }  
 );  
 }  
  
 delete() {  
 this.lessonService.deleteLesson(this.lessonId).subscribe(  
 response => {  
 this.router.navigateByUrl('course/' + this.lesson.courseId);  
 }  
 );  
 }  
  
 onCommentSubmit() {  
 ***console***.log(this.commentText);  
 const addComment = new AddComment(this.commentText, this.lesson.id, null);  
 this.commentService.addComment(addComment).subscribe(  
 response => {  
 this.lesson.comments.push(response);  
 }  
 );  
 }  
  
 edit() {  
 this.router.navigateByUrl('edit-lesson/' + this.lessonId);  
 }  
  
 teacherPartOfCourse(): boolean {  
 return this.visibilityService.teacherPartOfCourse(this.course);  
 }  
}

Lesson.component.ts

Na serverskoj strani se poziva metoda getLesson u LessonController-u. Iz MySql baze se dobavlju podaci o lekciji, osim njenog sadržaja. Nakon što su dobaljeni ti podaci, sadržaj lekcije se dobavlja iz Mongo baze podataka. Nakon što su svi podaci dobavljeni, kreira se LessonDto i šalje se klijentu.

//Get lesson  
@GetMapping(value = ["{lessonId}"])  
fun getLesson(@PathVariable("lessonId") lessonId: Long): ResponseEntity<LessonDto> {  
 val lesson = lessonService.findById(lessonId)  
 val lessonContent = try{  
 lessonService.getLessonContent(lesson)  
 }catch (e:Exception){  
 LessonContent("","")  
 }  
 val dto = LessonDto(lesson, lessonContent.lessonContent, lessonContent.id!!)  
 return ResponseEntity(dto, HttpStatus.OK)  
}

LessonController.kt

class LessonDto {  
  
 var id: Long?  
 var absences: List<Absence>  
 var comments: List<CommentDTO>  
 var lessonContent: String  
 var name: String  
 var date: Date  
 var courseId: Long  
 var lessonLink: String  
 var linked: Boolean?  
  
  
  
 constructor(lesson: Lesson, content: String, link: String) {  
 this.id = lesson.id  
 this.lessonContent = content  
 this.absences = ArrayList()  
 this.comments = lesson.comments.*map* **{** comment **->** CommentDTO(comment) **}** this.name = lesson.name  
 this.date = lesson.date  
 this.courseId = lesson.course.id!!  
 this.lessonLink = link  
 this.linked = lesson.linked  
 }  
}

LessonDto

### Komentari

Korisnici imaju mogućnost da dodaju komentare na lekcije. Komentar je na serverskoj strani predstavljen sa atributima:

* id,
* user,
* replies,
* text,
* datePosted i
* reply

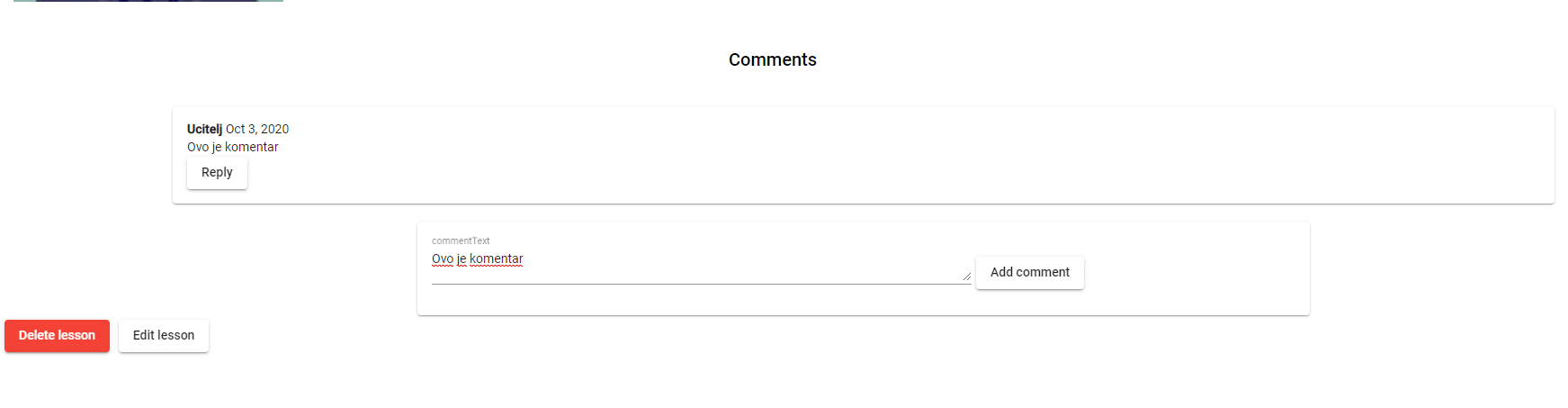
@Entity  
class Comment(@Id  
 @GeneratedValue  
 var id: Long?,  
 @ManyToOne  
 var user: User,  
 @OneToMany  
 var replies: Set<Comment>,  
 @Column  
 var text: String,  
 @Column  
 var datePosted: Date,  
 @Column  
 var reply: Boolean) {  
}

Klasa Comment.ts

Na klijentskoj strani za predstavljanje pojedinačnog komentara odgovorna je komponenta Comment.

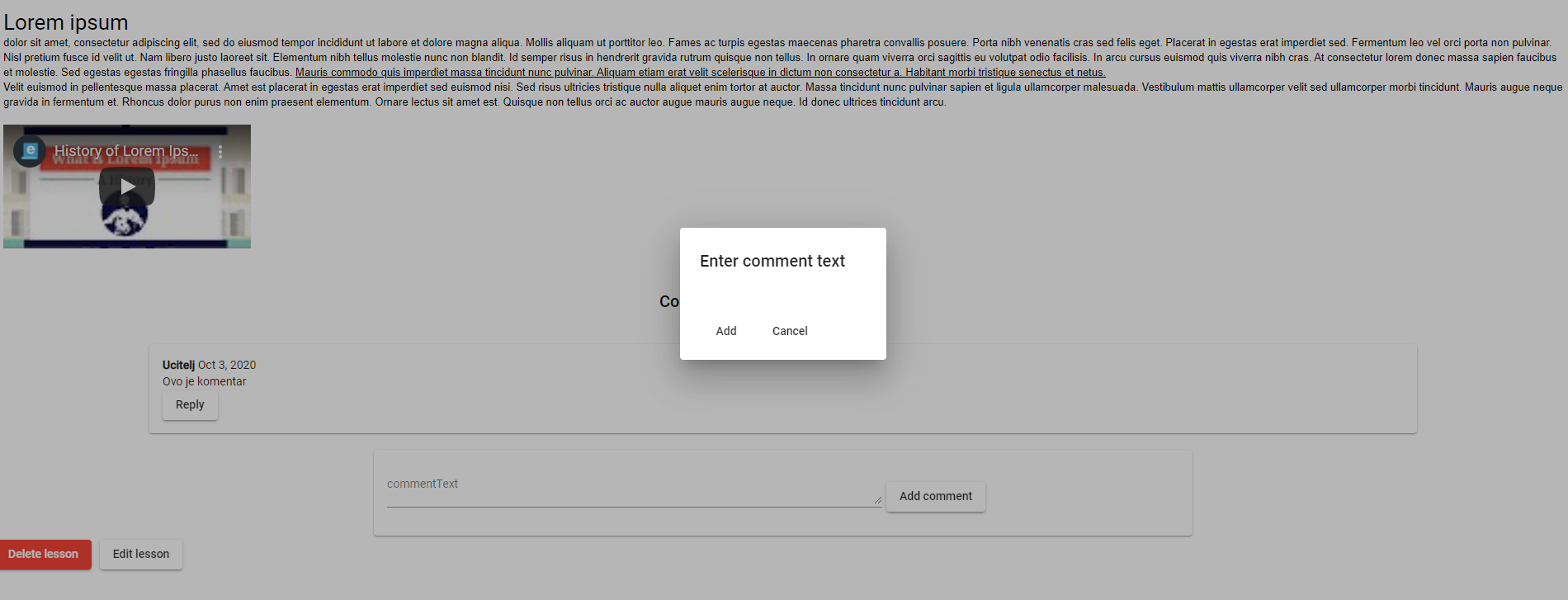
<mat-card>  
 <b>{{comment.user}}</b>  
 {{comment.datePosted|date}}  
 <div>{{comment.text}}</div>  
 <span *\*ngIf*="comment.replies.length>0">  
 Replies  
 </span>  
 <div class="comment-margin">  
 <app-comment *\*ngFor*="let ***c*** of comment.replies" [lesson]="lesson" [comment]="***c***"></app-comment>  
 </div>  
 <button mat-raised-button (click)="reply()">Reply</button>  
</mat-card>

Html deo komponente Comment.



Prikaz komentara

Korisnici takođe imaju opciju da dodaju odgovor na komentar. To se postiže klikom na dugme „Reply“.



Dodavanje odgovora na komentar

Nakon klika za dodavanje odgovora na komentar ili unosa novog komentara zahtev stiže do metode addComment u CommentControlleru

@RestController  
@RequestMapping(value = ["/api/comments"])  
@CrossOrigin  
class CommentController(private val commentService: CommentService) {  
  
 //Add comment  
 @PostMapping(consumes = [MediaType.*APPLICATION\_JSON\_VALUE*])  
 fun addComment(@RequestBody addCommentDTO: AddCommentDTO): ResponseEntity<CommentDTO> {  
 val comment = commentService.addComment(addCommentDTO)  
 return ResponseEntity(CommentDTO(comment), HttpStatus.OK)  
 }

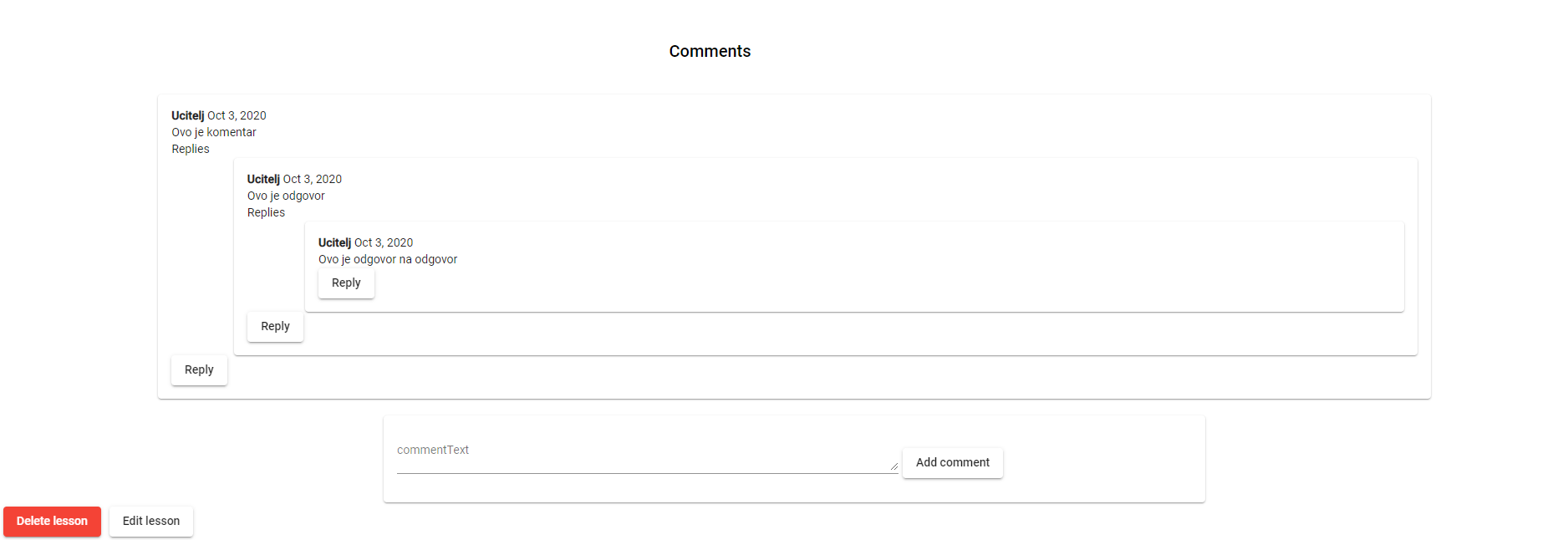
}

Deo CommentController-a

Metoda iz kontrolera poziva metodu addComment u CommentService-u koja kreira novi objekat i čuva ga u bazi.

fun addComment(addCommentDTO: AddCommentDTO): Comment {  
 val user = userService.getLoggedInUser()  
 val reply = addCommentDTO.replyTo != null  
 var comment = Comment(null, user!!, HashSet(), addCommentDTO.text, Date(), reply)  
 comment = commentRepository.save(comment)  
  
 if (reply) {  
 val parentComment = findById(addCommentDTO.replyTo!!)  
 (parentComment.replies as MutableSet).add(comment)  
 save(parentComment)  
 }  
 val lesson = lessonService.findById(addCommentDTO.lessonId);  
 (lesson.comments as MutableSet).add(comment);  
 lessonService.save(lesson)  
 return comment  
}

Metoda addComment u CommentService

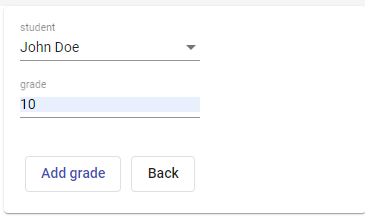


Prikaz odgovora na komentare

## Rad sa ocenama

Nastavnik ima opciju da unese ocenu studentu. Tip ocene je string kako bi aplikacija bila kompatibilna sa različitim sistemima ocenjivanja.

Nastavnik bira studenta iz liste studenata, potom unosi ocenu i pritiska dugme „Add grade“.

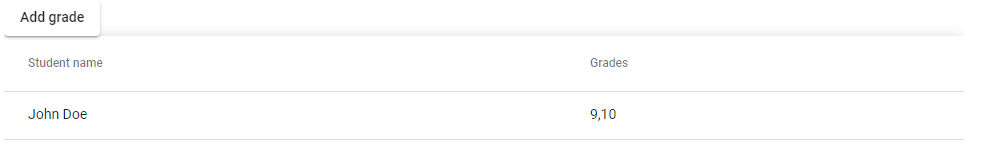


Forma za unos ocene.

<div>  
 <app-navbar></app-navbar>  
 <mat-card class="box" id="login-box">  
 <form class="form" [formGroup]="gradeForm" (ngSubmit)="gradeForm.valid && addGrade()">  
 <mat-card-content>  
 <mat-form-field class="field-full-width">  
 <mat-select id="student"  
 formControlName="student"  
 placeholder="student">  
 <mat-option value="-1">  
 None  
 </mat-option>  
 <mat-option *\*ngFor*="let ***student*** of course.students" [value]=***student***.id>  
 {{***student***.name}}  
 </mat-option>  
 </mat-select>  
  
 <mat-error>student can not be blank.</mat-error>  
 </mat-form-field>  
  
 <mat-form-field class="field-full-width">  
 <input id="grade"  
 matInput  
 formControlName="grade"  
 placeholder="grade"/>  
 <mat-error>grade can not be blank.</mat-error>  
 </mat-form-field>  
  
 </mat-card-content>  
 <button id="createBt" mat-stroked-button color="primary" class="btn-block">Add grade</button>  
 <button type="button" (click)="back()" mat-stroked-button class="btn-block">Back</button>  
 </form>  
 </mat-card>  
</div>

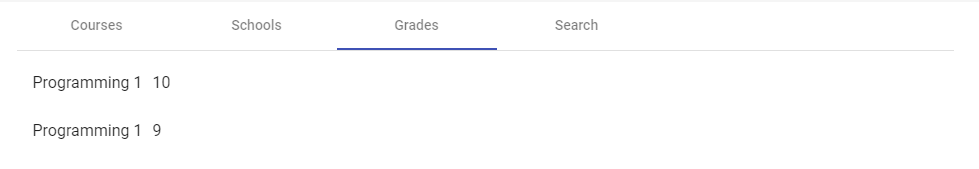
Html komponenta za unos ocene.

Nakon unete ocene, nastavnik može da vidi ocene svih studenata na kursu grupisane po studentu.



Nastavnikov pregled ocena studenata.

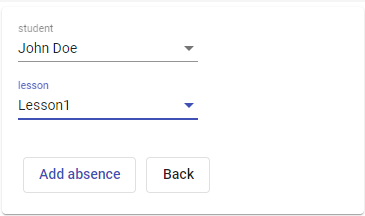
Takođe i student može da vidi svoje ocene. Nakon klika na tab ocene, pojavljuje se lista ocena iz kurseva.



Studentov prikaz ocena

## Izostanci

Ukoliko student izostane sa online ili normalne lekcije nastavnik ima mogućnost da unese izostanak za studenta.



Unos izostanka.

Za unos izostanka potrebno je iz liste izabrati studenta i lekciju na kojoj student nije bio prisutan, potom pritisnuti dugme Add absence.

class AddAbsenceDTO(var studentId: Long, var lessonId: Long) {  
}

Dto klasa za dodavanje izostanaka.

@RestController  
@RequestMapping(value = ["/api/absences"], produces = [MediaType.*APPLICATION\_JSON\_VALUE*])  
@CrossOrigin  
class AbsenceController(private val absenceService: AbsenceService) {  
  
 //Add absence  
 @PostMapping(consumes = [MediaType.*APPLICATION\_JSON\_VALUE*])  
 @PreAuthorize("hasAnyAuthority('TEACHER')")  
 fun addAbsence(@RequestBody absenceDTO: AddAbsenceDTO): ResponseEntity<AbsenceDTO> {  
 val absence = absenceService.createAbsence(absenceDTO)  
 return ResponseEntity(AbsenceDTO(absence), HttpStatus.OK)  
 }

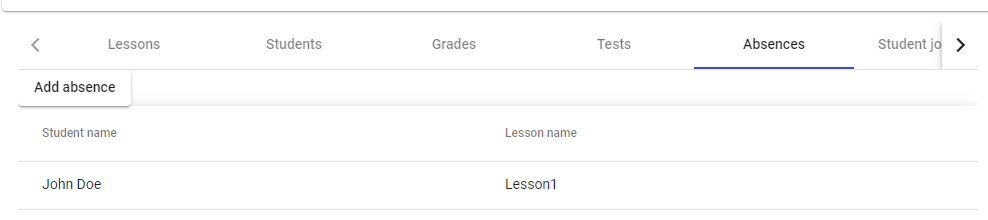
}

Deo AbsenceController-a za dodavanje izostanka.

fun createAbsence(absenceDTO: AddAbsenceDTO): Absence {  
 //Get student, course and lesson  
 val student = studentService.findById(absenceDTO.studentId)  
 val lesson = lessonService.findById(absenceDTO.lessonId)  
 val course = lesson.course  
  
 //Check if student is taking this course  
 val existing = course.students.*find* **{** studentCourse **->** studentCourse.id == student.id **}** ?: throw UnAuthorizedException("Student is not taking course")  
  
 //Check if logged in person is teaching this course  
 val loggedInUser = userService.getLoggedInUser()  
 when (loggedInUser) {  
 is Teacher -> {  
 loggedInUser.courses.*find* **{** courseTeacher **->** courseTeacher.id == course.id **}** ?: throw UnAuthorizedException("Teacher is not teaching course")  
 }  
 is School -> {  
 if (course.school?.id != loggedInUser.id) {  
 throw UnAuthorizedException("School is not in charge of that course")  
 }  
 }  
 }  
  
 var absence = Absence(null, course, lesson, student)  
 absence = absenceRepository.save(absence)  
 //Add to remaining lists  
 (course.absences as MutableSet).add(absence)  
 (lesson.absences as MutableSet).add(absence)  
 (student.absences as MutableSet).add(absence)  
 courseService.save(course)  
 lessonService.save(lesson)  
 studentService.save(student)  
 return absence  
}

Metoda za dodavanje izostanka u AbsenceService klasi.

Nastavnik može da vidi izostanke studenta iz kursa.



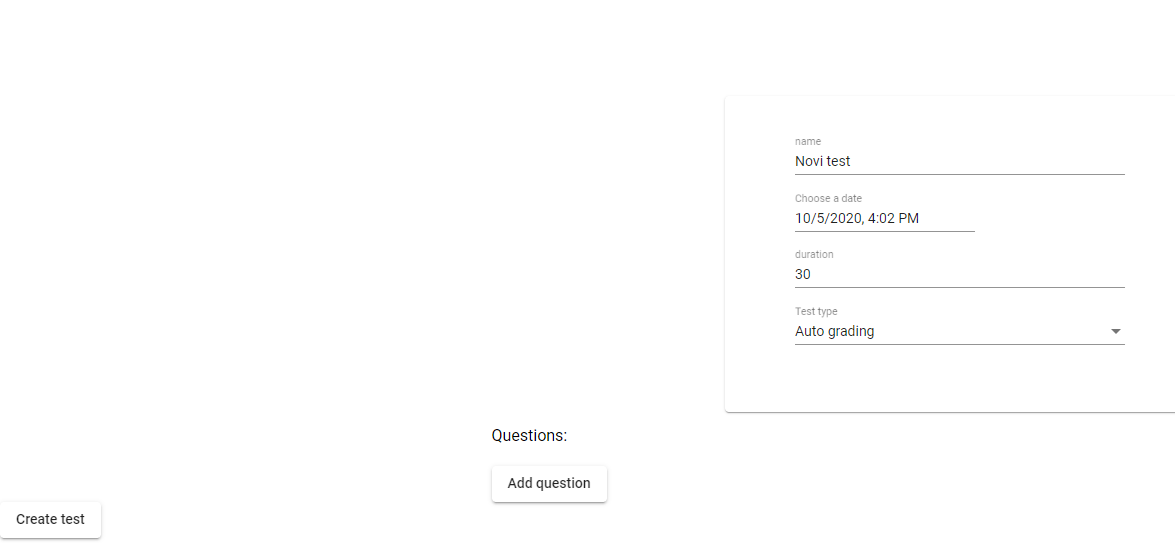
Pregled izostanaka studenata.

## Testovi

### Kreiranje testa

Nastavnik može da kreira testove. Klik na dugme „Create Test“ ga vodi na stranicu za kreiranje testa gde je potrebno da unese:

* Naziv testa,
* Datum testa,
* Vreme trajanja testa u minutima i
* Da li želi da računar da preliminaran broj bodova ili sam želi da ocenjuje.
  + - Auto grading – Nakon završetka testa, studentu će biti ispisan broj osvojenih bodova. Nastavnik mora da pregleda test i da unese ocenu.
    - Manual grading – Nakon završetka testa, student ne dobija poruku o bodovima. Nastavnik mora da pregleda test i unese ocenu.

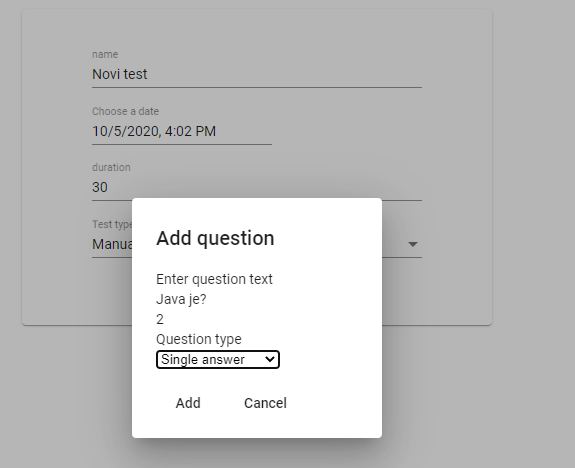


Kreiranje testa

Potom unosi pitanja i odgovore na pitanja. Potrebno je uneti naziv pitanja, broj bodova i tip pitanja.

Tipovi pitanja su:

* Single answer – samo jedan odgovor od ponuđenih je tačan,
* Multiple answers – više odgovora je tačno,
* Short answer – Unosi se kratak tekst i
* Long answer – Unosi se tekst proizvoljne dužine. Long answer je dostupan samo ako je izabrana opcija Manual grading.



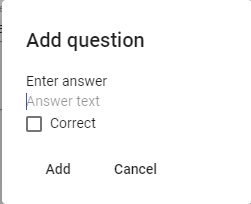
Dijalog za unos pitanja.

Nakon klika na Add dugme, pitanje je dodato u listu i moguće je da se doda odgovor.



Pitanje u listi.

Klikom na dugme Add answer, otvara se dijalog za unos odgovara.

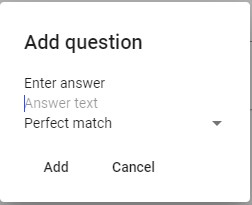


Dijalog za unos odgovora na pitanje

Dijalog za unos odgovora se razlikuje u zavisnosti od pitanja.

Prilikom unosa odgovora za za pitanje sa jednim tačnim odgovorom. Checkbox za tačan odgovor će biti dostupan dok se ne doda tačan odgovor. Dok će isti checkboc uvek biti dostupan.

Dijalog za dodavanje odgovora na pitanje gde se unosi kratak tekst je drugačiji.



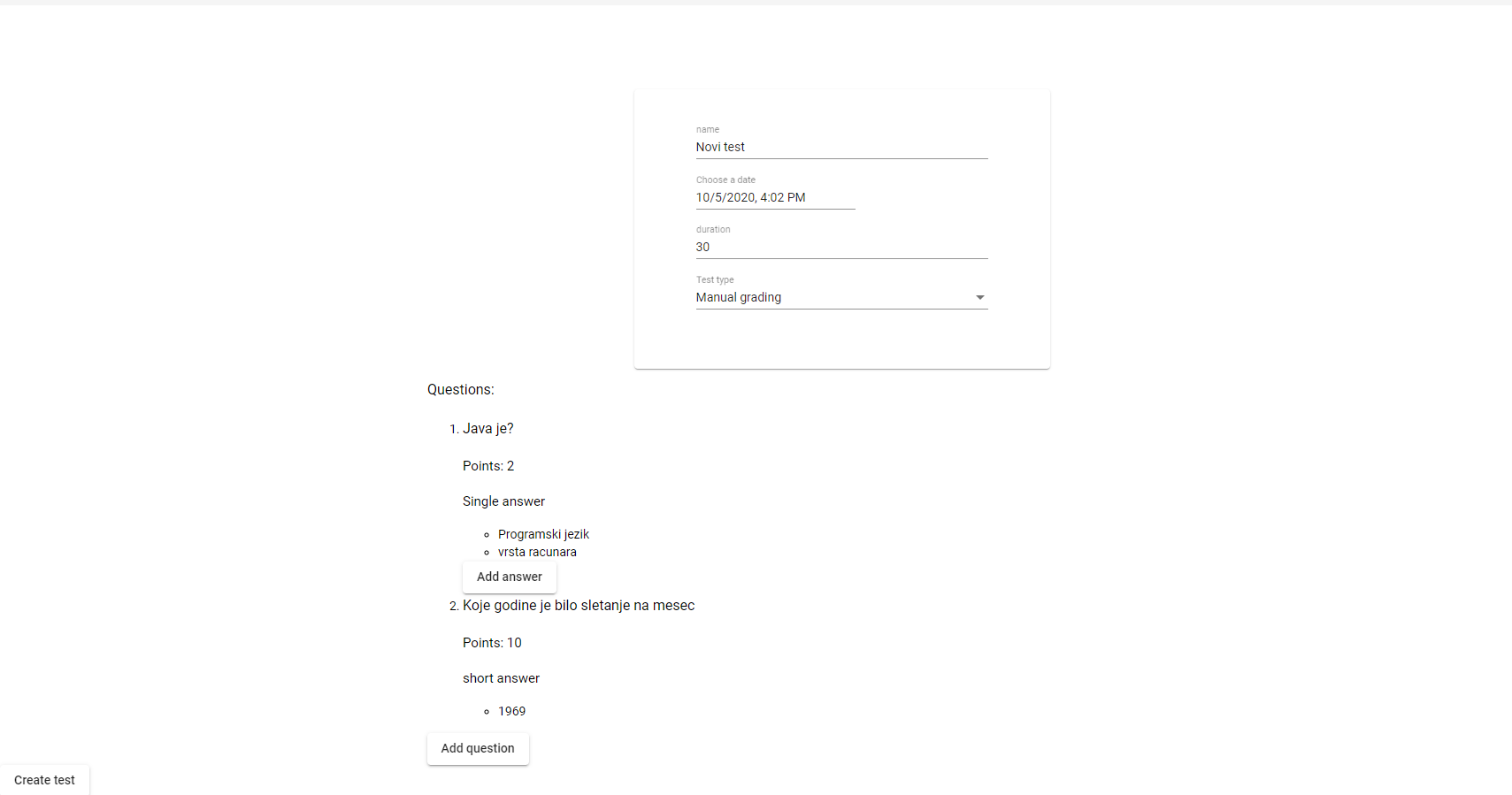
Dijalog za unos odgovora na pitanje.

Kao i gore, unosi se tekst odgovora. Zatim se unosi tip ocenjivanja za to pitanje.

Tip ocenjivanja može biti:

* Perfect match – Studentov odgovor se mora u potpunosti podudarati sa gore napisanim odgovorom,
* Ignore casing – Program prilikom pregledanja ne pravi razliku između velikih i malih slova i
* Accept close answers – Program prihvata odgovore koji su približni tačnom.

Nakon što nastavnik unese pitanja i odgovore potrebno je da pritisne dugme „*Create test“* da bi završio kreiranje testa.



Stranica nakon unesenih pitanja

Klik na Create test dugme poziva funkciju publishTest u createTest.component.ts

publishTest() {  
 const createTest = new CreateTest(this.name, this.date, this.duration, this.testType, this.questions, this.courseId);  
 let displayAlert = false;  
 if (this.testType === 'SELF\_GRADING') {  
 for (const question of this.questions) {  
 if (question.questionType === 'PARAGRAPH') {  
 displayAlert = true;  
 break;  
 }  
 }  
 if (displayAlert) {  
 alert('Auto grading is not possible. Please select Manual grading');  
 } else {  
 this.testService.createTest(createTest).subscribe(  
 response => {  
 this.router.navigateByUrl('course/' + this.courseId);  
 }  
 );  
 }  
 }  
}

Metoda publish test.

Na serverskoj strani se poziva createTest metoda u TestController-u.

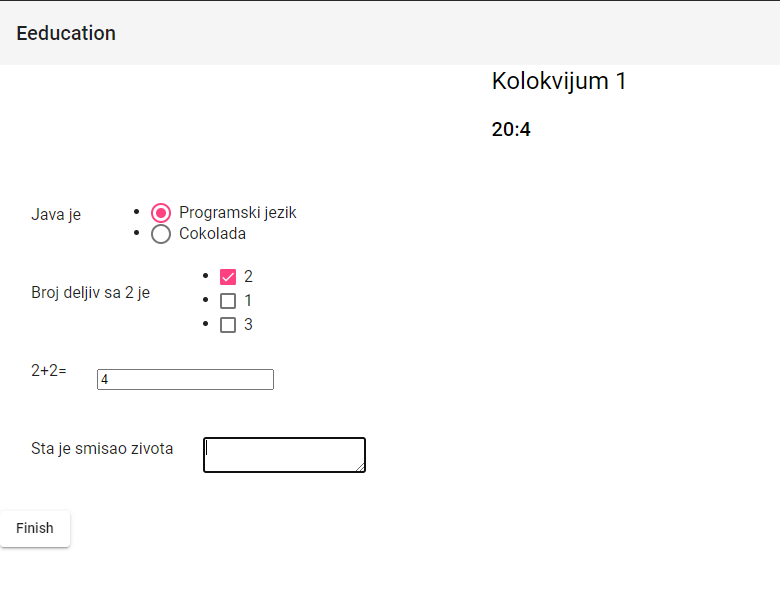
@RestController  
@RequestMapping(value = ["api/test"], produces = [MediaType.*APPLICATION\_JSON\_VALUE*])  
@CrossOrigin  
class TestController(private val testService: TestService) {  
  
 //Create Test  
 @PostMapping(consumes = [MediaType.*APPLICATION\_JSON\_VALUE*])  
 @PreAuthorize("hasAuthority('TEACHER')")  
 fun createTest(@RequestBody createTestDto: CreateTestDto): ResponseEntity<TestDTO> {  
 val test = testService.createTest(createTestDto)  
 val dto = TestConverter.toDto(test)  
 return ResponseEntity(dto, HttpStatus.OK)  
 }

}

Deo klase TestController sa metodom createTest

### Rađenje testa

Student može da pristupi testu od trenutka kada test počne pa do trenutka kad se test završi. Studentu je prikazan naziv testa kao i vreme koje mu je preostalo da završi test. Nakon što uradi sva pitanja i pritisne dugme Finish, rezultati testa se šalju na server gde se čuvaju.



Rađenje testa.

Logika je na klijentskoj strani implementirana u klasi TestComponent. U njoj postoje metode:

* ngOnInit – U njoj se sa server dobavlja test, kao i podešava tajmer,
* sortQuestions – Metoda koja sortira pitanja,
* questionAnsweredRadio i questionAnsweredChecked su metode koje se pozivaju nakon što se promeni vrednost u radio button-ima i check box-ovima i
* metoda finish koja šalje podatke na server.

export class TestComponent implements OnInit {  
  
 test: Test;  
 testId: number;  
 timer = '30:30';  
 timeRemaining: number;  
 testResults: TestResult;  
 score: string;  
  
 constructor(private testService: TestService, private router: Router, private route: ActivatedRoute,  
 private snackBar: MatSnackBar) {  
 }  
  
 ngOnInit(): void {  
 this.testId = ***Number***(this.route.snapshot.paramMap.get('id'));  
 this.testService.getTest(this.testId).subscribe(  
 response => {  
 this.test = response;  
 this.sortQuestions();  
 const dateTimeSplitted = this.test.date.split(' ');  
 const dateDate = dateTimeSplitted[0];  
 const time = dateTimeSplitted[1];  
  
 const dateDateSplitted = dateDate.split('/');  
 const timeSplitted = time.split(':');  
 const startDate = new ***Date***(***Number***(dateDateSplitted[2]), ***Number***(dateDateSplitted[1]) - 1,  
 ***Number***(dateDateSplitted[0]), ***Number***(timeSplitted[0]), ***Number***(timeSplitted[1]));  
 const endDate = moment(startDate).add(this.test.duration, 'm').toDate();  
 this.timeRemaining = ***Math***.abs(endDate.getTime() - (new ***Date***()).getTime()) / 1000;  
 const x = setInterval(() => {  
 this.timeRemaining = this.timeRemaining -= 1;  
 const minutes = this.timeRemaining / 60;  
 const seconds = this.timeRemaining % 60;  
 this.timer = ***Math***.floor(minutes) + ':' + ***Math***.floor(seconds);  
 }, 1000);  
 },  
 error => {  
 this.snackBar.open('Error getting test');  
 }  
 );  
 this.testResults = new TestResult(this.testId);  
 }  
  
 sortQuestions() {  
 for (let i = 0; i < this.test.questions.length - 1; i++) {  
 for (let j = i + 1; j < this.test.questions.length; j++) {  
 if (this.test.questions[j].id < this.test.questions[i].id) {  
 const temp = this.test.questions[i];  
 this.test.questions[i] = this.test.questions[j];  
 this.test.questions[j] = temp;  
 }  
 }  
 }  
 }  
  
 questionAnsweredRadio(questionId, answerText, answerId, $event) {  
 let exist = false;   
 for (const question of this.testResults.userQuestionResults) {  
 if (question.questionId === questionId) {  
 exist = true;  
 question.answers = [];  
 const answer = new Answer(answerId, answerText, false, null);  
 question.answers.push(answer);  
 return;  
 }  
 }  
 if (!exist) {  
 const userQuestionResult = new UserQuestionResult();  
 userQuestionResult.questionId = questionId;  
 userQuestionResult.answers = [new Answer(answerId, answerText, false, null)];  
 this.testResults.userQuestionResults.push(userQuestionResult);  
 }  
 }  
  
 questionAnsweredChecked(questionId, answerText, answerId, $event) {  
 let exist = false;  
 for (const question of this.testResults.userQuestionResults) {  
 if (question.questionId === questionId) {  
 exist = true;  
 if ($event.checked) {  
 question.answers.push(new Answer(answerId, answerText, false, null));  
 } else {  
 for (let i = 0; i < question.answers.length; i++) {  
 if (question.answers[i].id === answerId) {  
 question.answers.splice(i, 1);  
 }  
 }  
 }  
 }  
 }  
 if (!exist) {  
 const userQuestionResult = new UserQuestionResult();  
 userQuestionResult.questionId = questionId;  
 userQuestionResult.answers = [new Answer(answerId, answerText, false, null)];  
 this.testResults.userQuestionResults.push(userQuestionResult);  
 }  
 }  
  
 questionTextAnswer(questionId, $event) {  
 const input = (***document***.getElementById(questionId) as HTMLInputElement).value;  
 let exist = false;  
 for (const question of this.testResults.userQuestionResults) {  
 if (question.questionId === questionId) {  
 exist = true;  
 const answer = new Answer(null, input, false, null);  
 question.answers = [answer];  
 }  
 }  
 if (!exist) {  
 const userQuestionResult = new UserQuestionResult();  
 userQuestionResult.questionId = questionId;  
 userQuestionResult.answers = [new Answer(null, input, false, null)];  
 this.testResults.userQuestionResults.push(userQuestionResult);  
 }  
 }  
  
 finish() {  
 this.testResults.date = new ***Date***();  
 this.testService.studentSubmitsTest(this.testResults).subscribe(  
 response => {  
 this.testResults = response;  
 this.score = this.testResults.score;  
 }  
 );  
 }  
  
 back() {  
 this.router.navigateByUrl('course/' + this.test.courseId);  
 }  
  
}

Prikaz klase test.component.ts

### Ocenjivanje testa

U slučaju da je izabrano automatsko ocenjivanje testa. Student nakon urađenog testa dobija ispis o osvojenim bodovima.



Ispis o osvojenim bodovima.

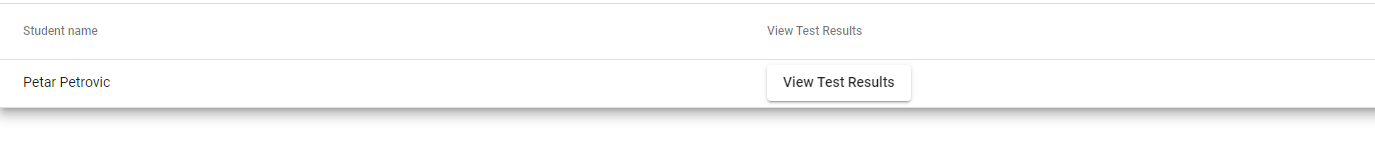
Za ocenjivanje je zaslužna metoda gradeTest. Metoda prolazi kroz svako pitanje i proverava da li je odgovor tačan. Kod pitanja sa kratkim odgovorom, gde se kao tačni priznaju i slični odgovori koristi se JaroWinkler distance. Iz biblioteke java-string-similaritz koju je razvio rrice [10]. Ispod je kod koji se koristi za ocenjivanje testa.

fun gradeTest(testResults: TestResults): String {  
 val test = testResults.test  
 val testQuestions = test.questions  
 val userQuestionResult = testResults.userQuestionResults  
 val totalPoints = testQuestions.*sumBy* **{** question **->** question.questionPoints **}** var userPoints = 0  
 for (questionResult in userQuestionResult) {  
 val question = questionResult.question  
 val questionPoints = gradeQuestion(question, questionResult.userAnswer.*toList*())  
 questionResult.points = questionPoints  
 userPoints += questionPoints  
 }  
 return "$userPoints/$totalPoints"  
  
}  
  
private fun gradeQuestion(question: Question, userAnswers: List<UserAnswer>): Int {  
 if (question.questionType == QuestionType.CHECKBOXES) {  
 userAnswers.*forEach* **{** userAnswer **->** var failed = false  
 question.answer.*forEach* **{** answer **->** if (answer.id == userAnswer.answer?.id) {  
 if (!answer.correct!!) {  
 failed = true;  
 }  
 }  
 **}** val numOfCorrectAnswers = question.answer.*filter* **{** answer **->** answer.correct!! **}** if (userAnswers.size != numOfCorrectAnswers.size) {  
 failed = true  
 }  
 if (!failed) {  
 return question.questionPoints  
 }  
 **}** } else if (question.questionType == QuestionType.MULTIPLE\_CHOICE) {  
 if (userAnswers.*isNotEmpty*()) {  
 val userAnswer = userAnswers[0]  
 val answer = question.answer.*filter* **{** answer **->** answer.id!! == userAnswer.answer!!.id **}** if (answer[0].correct!!) {  
 return question.questionPoints  
 }  
 }  
 } else if (question.questionType == QuestionType.SHORT\_ANSWER) {  
 if (userAnswers.*isNotEmpty*()) {  
 val answer = question.answer.*toList*()[0]  
 if (answer.answerType == ANSWER\_TYPE.PERFECT\_MATCH) {  
 val text = answer.answerText  
 val userAnswerText = userAnswers[0].text  
 if (text == userAnswerText) {  
 return question.questionPoints  
 }  
 } else if (answer.answerType == ANSWER\_TYPE.IGNORE\_CASING) {  
 val text = answer.answerText.*toLowerCase*()  
 val userAnswerText = userAnswers[0].text.*toLowerCase*()  
 if (text == userAnswerText) {  
 return question.questionPoints  
 }  
 } else {  
 val strategy: SimilarityStrategy = JaroWinklerStrategy()  
 val target = question.answer.*toList*()[0].answerText  
 val source = userAnswers[0].text  
 val service: StringSimilarityService = StringSimilarityServiceImpl(strategy)  
 val score = service.score(source, target)  
 if (score >= 0.95) {  
 return question.questionPoints  
 }  
 }  
 }  
  
 }  
 return 0  
}

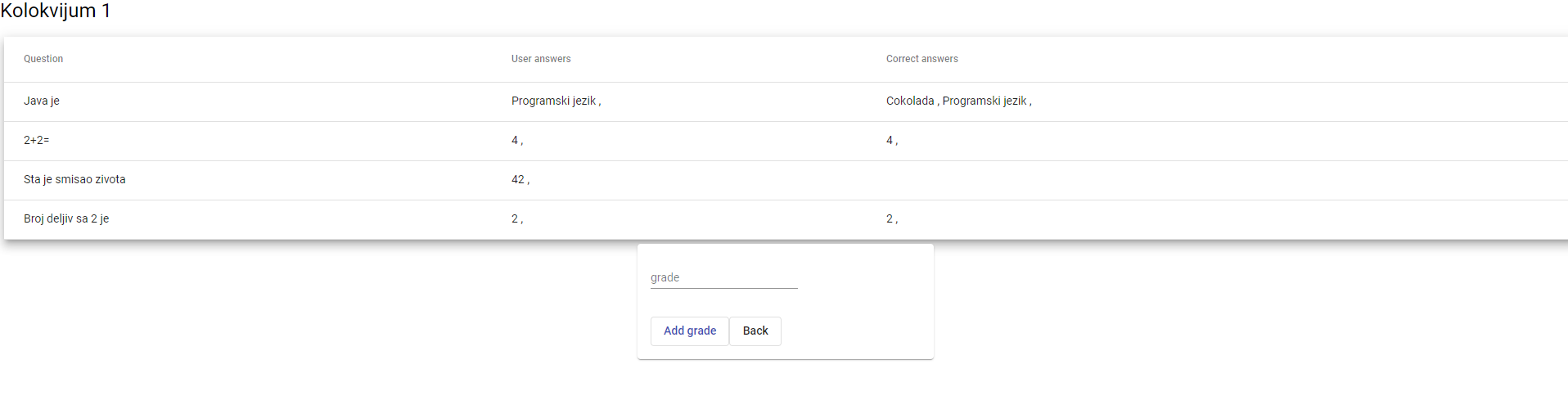
Metode gradeTes i gradeQuestion u TestService klasi.

### Ručno ocenjivanje testa

Nakon što studenti završe test nastavnik može da pregleda i unese ocenu. Klikom na dugme View Test Results, nastavnik je odveden na stranicu gde može da vidi listu studenata koji su uradili test. Pored studentovog imena se nalazi I dume View Test Results koje, kada ga nastavnik pritisne, ga vodi na novu stranicu gde može da vidi kako je student odgovorio na pitanja I koji su tačni odgovori.



Lista studenata koji su uradili test.



Pregled studentovih odgovora.

# Zaključak

Projekat predstavlja jednostavnu aplikaciju za internet obrazovanje. Dosadašnji rad obuhvata osnovne funkcionalnosti koje sistem treba da podrži poput kreiranja kursa, dodavanja lekcije, dodavanja ocena, kao i kreiranje i unos testova.

* Neke od budućih funkcionalnosti uključuju:
* Veća uloga škole – da se određenim kursevima može pristupiti samo ako je student učlanjen u školu,
* Poboljšati mehanizam za ocenjivanje testova. Da aplikacija može sama da unese ocenu na osnovu dobijenih bodova i
* Omogućiti izmenu ocena i izostanaka

# Literatura

1. Kotlin   
   <https://kotlinlang.org/>
2. Spring boot  
   <https://spring.io/projects/spring-boot>
3. Spring data  
   <https://spring.io/projects/spring-data>
4. Spring security  
   <https://spring.io/projects/spring-security>
5. Angular  
   <https://angular.io/>
6. Angular CLI  
   <https://cli.angular.io/>
7. Angular Material  
   <https://material.angular.io/>
8. Ngx-quill  
   <https://github.com/KillerCodeMonkey/ngx-quill>
9. Comparison to Java Programming Language  
   <https://kotlinlang.org/docs/reference/comparison-to-java.html>
10. String similarity  
    <https://github.com/rrice/java-string-similarity>

# KLJUČNA DOKUMENTACIJSKA INFORMACIJA

|  |  |  |  |
| --- | --- | --- | --- |
| Редни број, **РБР**: | |  | |
| Идентификациони број, **ИБР**: | |  | |
| Тип документације, **ТД**: | | монографска публикација | |
| Тип записа, **ТЗ**: | | текстуални штампани документ | |
| Врста рада, **ВР**: | | дипломски-бечелор рад | |
| Аутор, **АУ**: | | Миле Праштало | |
| Ментор, **МН**: | | Проф Др Милан Видаковић | |
| Наслов рада, **НР**: | | Веб апликација за интернет образовање | |
| Језик публикације, **ЈП**: | | српски | |
| Језик извода, **ЈИ**: | | српски / енглески | |
| Земља публикововања, **ЗП**: | | Србија | |
| Уже географско подручје, **УГП**: | | Војводина | |
| Година, **ГО**: | | 2020 | |
| Издавач, **ИЗ**: | | ауторски репринт | |
| Место и адреса, **МА**: | | Нови Сад, Факултет техничких наука, | |
| Физички опис рада, **ФО**:  (поглавља/страна/цитата/табела/слика/графика/прилога) | | 5 / 0 / 0 / 0 / 0/0 / 0 | |
| Научна област, **НО**: | | Информатика | |
| Научна дисциплина, **НД**: | | Рачунарске науке | |
| Предметна одредница/Кључне речи, **ПО**: | | Kotlin, ВЕБ, REST, Angular 9 | |
| **УДК** | |  | |
| Чува се, **ЧУ**: | | Библиотека Факултета техничких наука, Трг Доситеја Обрадовића 6, Нови Сад | |
| Важна напомена, **ВН**: | |  | |
| Извод, **ИЗ**: | | Задатак рада представља развој веб апликације за интернет образовање. Серверски део апликације ће бити реализован у програмском језику Котлин, коришћењем Spring Boot окружења. Клијентски део задатка ће бити реализовати коришћењем Angular 9 окружења. | |
| Датум прихватања теме, **ДП**: | |  | |
| Датум одбране, **ДО**: | |  | |
| Чланови комисије, **КО**: | Председник: |  |  |
|  | Члан: |  | Potpis mentora |
|  | Члан,ментор: | др Милан Видаковић, ред. проф., ФТН Нови Сад |  |

# KEY WORDS DOCUMENTATION

|  |  |  |  |
| --- | --- | --- | --- |
| Accession number, **ANO**: | |  | |
| Identification number, **INO**: | |  | |
| Document type, **DT**: | | monographic publication | |
| Type of record, **TR**: | | textual material | |
| Contents code, **CC**: | | BSc thesis | |
| Author, **AU**: | | Mile Praštalo | |
| Mentor, **MN**: | | Prof Dr Milan Vidaković | |
| Title, **TI**: | | WEB APPLICATION FOR NOTES MANAGEMENT | |
| Language of text, **LT**: | | Serbian | |
| Language of abstract, **LA**: | | Serbian / English | |
| Country of publication, **CP**: | | Serbia | |
| Locality of publication, **LP**: | | Vojvodina | |
| Publication year, **PY**: | | 2013 | |
| Publisher, **PB**: | | author’s reprint | |
| Publication place, **PP**: | | Novi Sad, Faculty of Technical Sciences, Trg Dositeja Obradovića 6 | |
| Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes) | | 5 / 0 / 0 / 0 / 0/ 0 / 0 | |
| Scientific field, **SF**: | | Electrical Engineering | |
| Scientific discipline, **SD**: | | Computer Science | |
| Subject/Key words, **S**/**KW**: | | Јava, ВЕБ, REST, Angular 6 | |
| **UC** | |  | |
| Holding data, **HD**: | | Library of the Faculty of Technical Sciences, Trg Dositeja Obradovića 6, Novi Sad | |
| Note, **N**: | |  | |
| Abstract, **AB**: | | The thesis deals with the implementation of a web application for online education. The back-end part is done in Kotlin using Spring boot framework, and the frontend is implemented using Angular 9 framework. | |
| Accepted by the Scientific Board on, **ASB**: | |  | |
| Defended on, **DE**: | |  | |
| Defended Board, **DB**: | President: |  |  |
|  | Member: |  | Menthor's sign |
|  | Member, Mentor: | Milan Vidaković, PhD, full prof.,  FTN Novi Sad |  |