Sustav vođenja evidencije pohađanja predavanja

Anić, Vladimir

Master's thesis / Diplomski rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:666829

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-25**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Diplomski studij računarstva

SUSTAV VOĐENJA EVIDENCIJE POHAĐANJA PREDAVANJA

Diplomski rad

Vladimir Anić

Osijek, 2018

SADRŽAJ

1. UVOD 1

1.1. Zadatak diplomskog rada 1

2. VAŽNE TEORIJSKE OSNOVE 2

2.1. MVC 2

2.2. Android 3

2.3. Firebase 4

3. MODELIRANJE SUSTAVA 5

3.1. Arhitektura sustava i aplikacije 5

3.2. Arhitektura baze podataka 6

4. POSTUPAK IZRADE SUSTAVA VOĐENJA EVIDENCIJE POHAĐANJA PREDAVNJA 8

4.1. Izrada Android aplikacije 8

4.2. Izrada WEB aplikacije 18

5. ZAKLJUČAK 33

LITERATURA 34

SAŽETAK 35

ABSTRACT 35

ŽIVOTOPIS 36

PRILOZI 37

1. UVOD

Evidencija pohađanja predavanja predstavlja problem u današnje vrijeme. Trenutno postoje razni načini za evideciju studenta na predavanju (ručno na papiru, pomoću Google formi itd.), no mnogi nisu sigurni (student se može upisivati na predavanje iako nije prisutan, a prozivanje studenata je mukotrpan proces) niti moderni. Ovaj diplomski rad bavi se izradom sustava koji bi trebao biti siguran, pouzdan te moderan način za upisivanje i vođenje evidencije studenata te sastoji se od dva dijela:

1. mobilne aplikacije za pametne telefone s Android operacijskim sustavom (studentska strana) i
2. WEB aplikacije koja će omogućiti stvaranje i pregled kolegija i predavanja, prikaz podatka o upisanim studentima te omogućiti ručno dodavanje i brisanje studenata (profesorova strana).

U drugom poglavlju dana je teorijska osnova o problemu, korištenim alatima, programskim paradigmama i principima. Nadalje, u trećem poglavlju je razrađena i opisana arhitektura sustava te korištenje baze podataka. Također, za bazu podataka korišten je engl. *Code-first* pristup za izradu korištenjem *Entity Framework* programskog okvira. U četvrtom poglavlju opisane su specifičnosti i funkcionalnosti cijelog sustava (mobilne i WEB aplikacije), karakterističke funkcije i korištene biblioteke, programski okviri te primjer korištenja obje aplikacije (mobilne i WEB).

1.1. Zadatak diplomskog rada

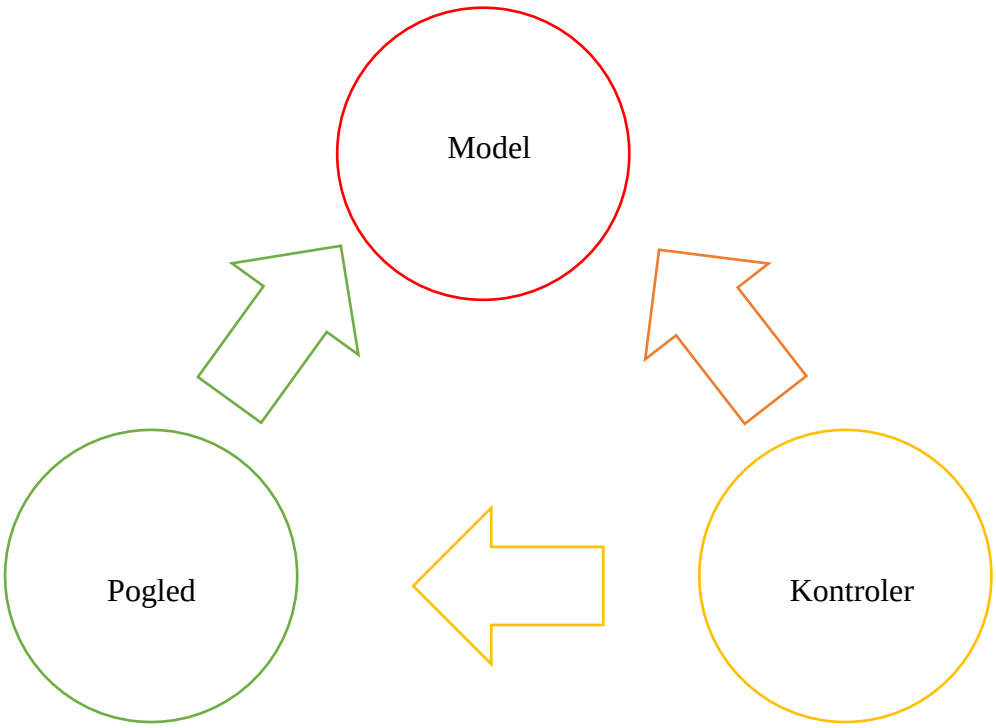
Kratko opisati problematiku vođenja evidencije pohađanja predavanja na fakultetu. Izraditi sustav koji bi olakšao vođenje evidencije dolazaka na predavanja i opisati proces izrade takvog sustava. Izraditi Android aplikaciju koja studentima omogućuje prijavu dolaska na predavanje. Nastavniku omogućiti aktivaciju prijave studenata i pregled prijavljenih studenata uz prikaz fotografije prijavljenog studenta. Tehnologija i alati za izradu zadatka: MS ASP.NET MVC, Android Studio, Firebase.

2. VAŽNE TEORIJSKE OSNOVE

U ovom poglavlju opisan je MVC (engl. *Model-View-Controller*) obrazac arhitekture programske podrške, engl. *Code-first* pristup s *Entity Framework* programskim okvirom, Android operacijskog sustava te Firebase.

2.1. MVC

MVC (engl. *Model-View-Controller*) je obrazac arhitekture programske podrške koji razdvaja aplikaciju u tri glavne komponente: modele, poglede i kontrolere. Ovaj obrazac pomaže u postizanju odvajanja zabrinutosti (engl. *Separation of Concerns*). Korištenjem navedenog obrasca, korisnički zahtjevi šalju se na kontroler, koji je odgovoran za rad s modelom i izvodi korisničke radnje i/ili dohvaća rezultate upita iz baze. Kontroler odabire pogled (engl. *View*) koji se treba prikazati korisniku i daje mu bilo koje podatke modela koji su mu potrebni.



Slika 2.1. Međusobno referenciranje modela, pogleda i kontrolera [1]

Ocrtavanje odgovornosti pomaže skaliranju aplikacija u smislu njihove kompleksnosti jer je lakše kodirati, debug-irati i testirati nešto što radi samo jedan posao i prati princip jednostruke odgovornosti (engl. *Single Responsibility Principle*).
Prednosti korištenja MVC uzorka su:

- lakše upravljanje aplikacijom jer je rastavljena na model, pogled i kontroler,
- ne koristi stanja pogleda ili forme osnovane na poslužiteljima,
- koristi uzorak kontrolera s prednje strane što znači da se svi zahtjevi WEB aplikacije rade kroz jedan kontroler,
- pruža bolju podršku za razvoj aplikacija pogonjen testiranjem (engl. *Test Driven Development*),
- odgovara najbolje za aplikacije na kojima radi veći broj razvojnih inženjera koji trebaju višu razinu kontrole nad ponašanjem aplikacije.

Za izradu baze podataka korišten je engl. *Code-first* pristup s *Entity Framework* programskim okvirom koji iz napisanih modela stvaraju relacijske tablice i veze između njih. Time se smanjuje razlika između objektno orijentiranog programiranja i relacijskih tablica jer daje rad s tablicama pomoću objekata *ASP.NET* klase (modela) i eliminira potrebu pisanja velikog koda za pristup podacima koje razvojni programeri uobičajeno moraju samostalno pisati. Koristeći *Entity Framework* moguće je napraviti i koristiti sve osnovne *CRUD* (engl. *Create, Read, Update, Delete*) upite te spremiti sve podatke i izmjene nad podacima u bazu podataka. *Entity Framework* prevodi i koristi *LINQ* upite za pristup i rad s bazom podataka. *LINQ* je Microsoftova metodologija koja omogućuje upite nad setovima podataka u bazi.

2.2. Android

Android je mobilni operacijski sustav razvijen od strane Google-a i osnovan je na modificiranoj verziji Linux jezgre te ostalih programa otvorenog koda dizajniranih prvenstveno za ekrane na dodir. Android operativni sustav je jedan od najraširenijih uz Apple-ovim *iOS* operacijskim sustavom. Android je povezan s nizom programa povezanih s Googleom i od kojih je najbitnija platforma za digitalnu distribuciju i prodaju aplikacija *Trgovina Play*. Android je najprodavanija platforma na pametnim telefonima od 2011. godine i na tabletima od 2013. godine. Ovaj operacijski sustav može iskoristiti mnogo različitih senzora (senzor blizine, žiroskop, akcelerometar, kamere itd.) i funkcija ekrana na dodir za manipulaciju aplikacijama. Aplikacije mogu biti instalirane koristeći *Trgovinu Play* ili razne aplikacijske pakete (APK) za aplikacije koje se ne nalaze na *Trgovini Play*. Zbog otvorenosti platforme u cijelosti, postoji i niz drugih platformi za preuzimanje aplikacija kao što su *Amazon Appstore, GetJar* i mnoga druga tržišta aplikacijama koja su namijenjena azijskom tržištu mobilnih aplikacija.

Android za razvoj mobilnih aplikacija koristi nekoliko različitih programskih jezika. Moguće je razvijati među-platformske aplikacije koristeći alate kao što su *Xamarin* i *ReactNative* ili koristiti Android Studio za razvoj nativnih Android aplikacija. U ovom radu bit će korišten Android Studio

i kod će biti pisan koristeći Java programski jezik za Android, no bitno je naglasiti da se u Android Studiju također može koristiti i Kotlin programski jezik. Java je odabrana zbog prijašnjih znanja i rada u istom te zbog bolje razvijene podrške i dokumentacije.

2.3. Firebase

Firebase je sustav za razvoj mobilnih i WEB aplikacija koji pruža različite servise za korištenje prilikom razvoja istih. U ovom radu Firebase je korišten za lakše upravljanje korisničkim računom u mobilnoj aplikaciji pri upisu studenata na predavanja. Kako bi se to postiglo, bit će korišten *Firebase Auth* servis koji omogućuje autentifikaciju korisnika koristeći samo kod na strani klijenta te je moguće koristiti prijavljivanje pomoću društvenim mreža kao što su: *Facebook*, *GitHub*, *Twitter* i *Google* (*Google Play Igre*). Ovaj servis također uključuje sustav upravljanja korisnicima gdje razvojni programeri mogu omogućiti autentifikaciju korisnika s e-poštom i lozinkama spremljenim u Firebase-u.

3. MODELIRANJE SUSTAVA

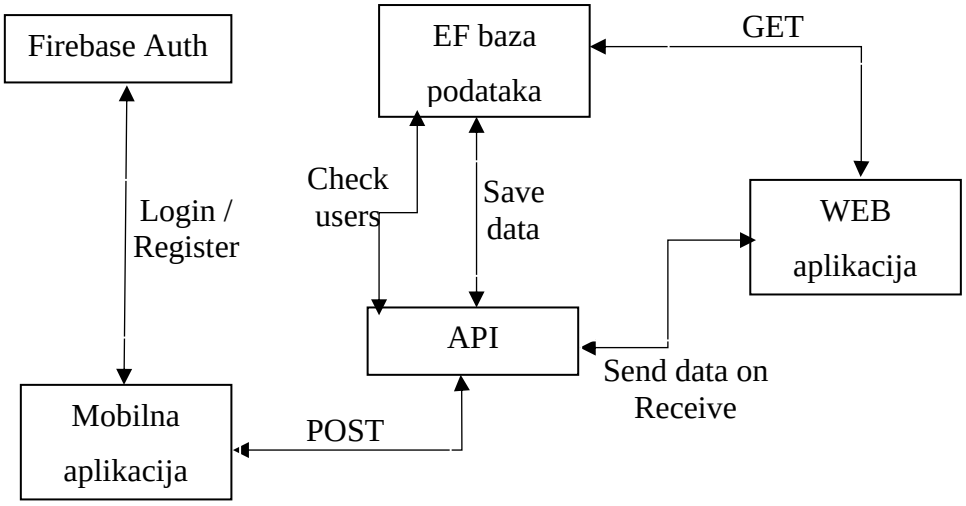
Kako bi bilo moguće napraviti sustav vođenja evidencije pohađanja predavanja, napravljena je arhitektura tog istog sustava s odgovarajućim elementima. Definiranjem arhitekture sustava rješava se povezanost mobilne i WEB aplikacije, odnosno profesorove i studentove strane. Osim arhitekture sustava potrebna je i arhitektura baze podataka s odgovarajućim tablicama i relacijama.

3.1. Arhitektura sustava i aplikacije

Sustav je moguće razdvojiti u nekoliko glavnih dijelova:

- mobilnu aplikaciju,
- *Firebase Auth* servis,
- *Entity Framework* bazu podataka,
- API koji upravlja svih zahtjevima i
- WEB aplikaciju,

kao što je moguće vidjeti na slici 3.1.



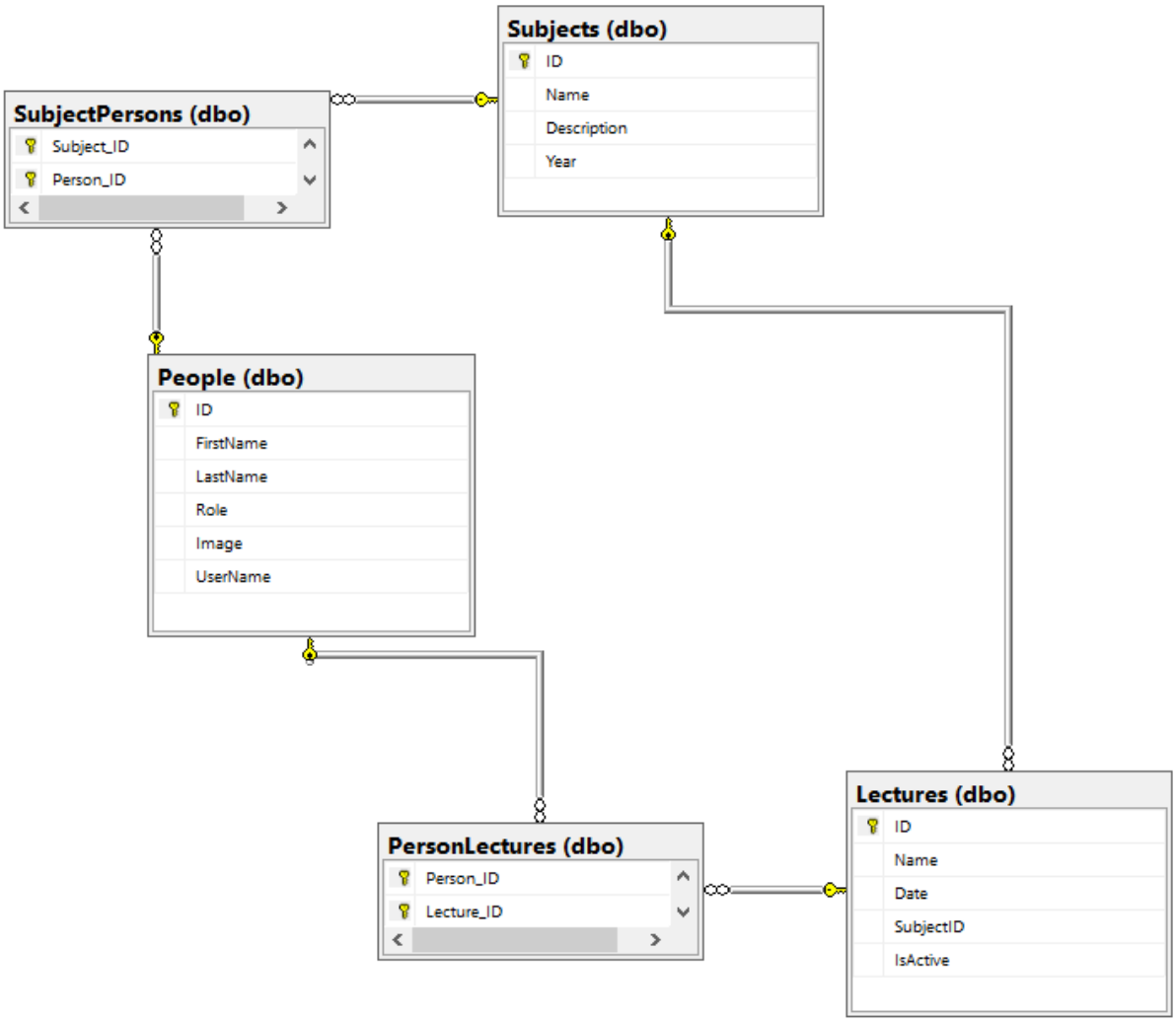
Slika 3.1. Arhitektura sustava s uključenim aplikacijama

Mobilna aplikacija će koristiti *Firebase Auth* servis za autentifikaciju korisnika kako bi se studenti mogli prijaviti u aplikaciju. Nakon što se student prijavi uspješno, aktivirati će se skener QR koda i student će moći skenirati kod koji će biti generiran u WEB aplikaciji i prikazan na projektorskom platnu te će automatski nakon skeniranja biti prijavljen na predavanje.

EF baza podataka je spremište svih podataka o studentima i njihovom prisustvu na predavanju te računima studenata koji postoje na određenom predavanju.
API provjerava je li prijavljeni korisnik u EF bazi podataka i ako je, daje dopuštenje za daljnji rad korisniku, a ako student nije u toj bazi, on se neće moći prijaviti na predavanje. API također ima zadaću spremati podatke o prijavi studenta na predavanju u bazu podataka.
WEB aplikacija je dio koji samo profesor vidi i omogućuje lakši prikaz upisanih studenata, njihovu prisutnost na predavanjima i omogućuje profesorima da ručno dodaju studente ukoliko je došlo do nekog problema i isto tako obrišu studente koji su upisani, a nisu prisustvovali na predavanju.

3.2. Arhitektura baze podataka

Na slici 3.2. moguće je vidjeti arhitekturu baze podataka koja je korištena za povezivanje entiteta koji su potrebni za ispravan rad WEB aplikacije.

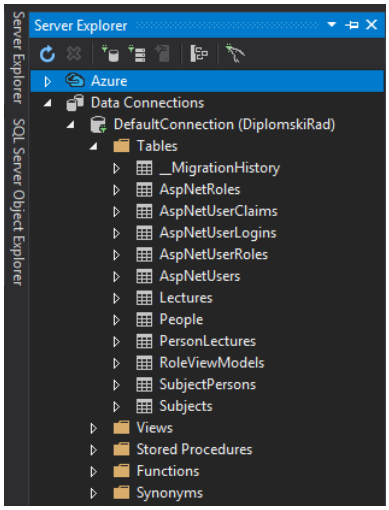


Slika 3.2. Dijagram baze podataka WEB aplikacije

Baza podataka se sastoji od tri glavna entiteta za upravljanjem i pregledom upisanih studenata, a to su tablice:

- engl. *Lecture* (predavanje),
- engl. *Person* (osoba) i
- engl. *Subject* (kolegij).

Tablica predavanja sadrži podatke o predavanju (ime predavanja, datum izvođenja predavanja, varijablu provjere je li predavanje aktivno te ID kolegija), tablica osoba sadrži sve glavne podatke o studentima i profesorima (ime, prezime, uloga, korisničko ime) te tablica kolegija koja sadrži podatke o kolegijima (naziv kolegija, opis i godina izvođenja). Tablica predavanje ima vezu više na više s tablicom osoba, što znači da više različitih osoba može prisustvovati jednome ili više predavanja. Tablica osobe također ima vezu više na više s tablicom kolegiji i na kraju tablica kolegiji ima vezu jedan na više s tablicom predavanja, što znači da jedan kolegij se sastoji od više predavanja, ali predavanje ne može biti u dva kolegija. Kod veza više na više stvorene su dodatne tablice (*SubjectPersons* i *PersonLectures*) koje vezu više na više pretvaraju u vezu jedan na više zbog lakšeg rada s tablicama.
Postoji još nekoliko pomoćnih tablica u bazi podataka koje služe za lakšu autentifikaciju korisnika i pristupu bazi podataka. Na slici 3.3. se nalazi popis svih tablica koje se koriste za ispravan rad WEB aplikacije.



Slika 3.3. Sve tablice korištene za WEB aplikaciju

Baza podataka se nalazi na Azure portalu i koristi se udaljeni pristup bazi podataka.

4. POSTUPAK IZRADE SUSTAVA VOĐENJA EVIDENCIJE POHAĐANJA PREDAVNJA

Pri izradi sustava za evidenciju studenata na predavanju, napravljena je mobilna aplikacija koju koriste studenti za skeniranje QR koda i upisivanja na predavanja te web aplikacija koja omogućava *CRUD* operacije za kolegije i predavanja, generiranje QR koda za aktivno predavanje te ručno dodavanje i brisanje studenata.

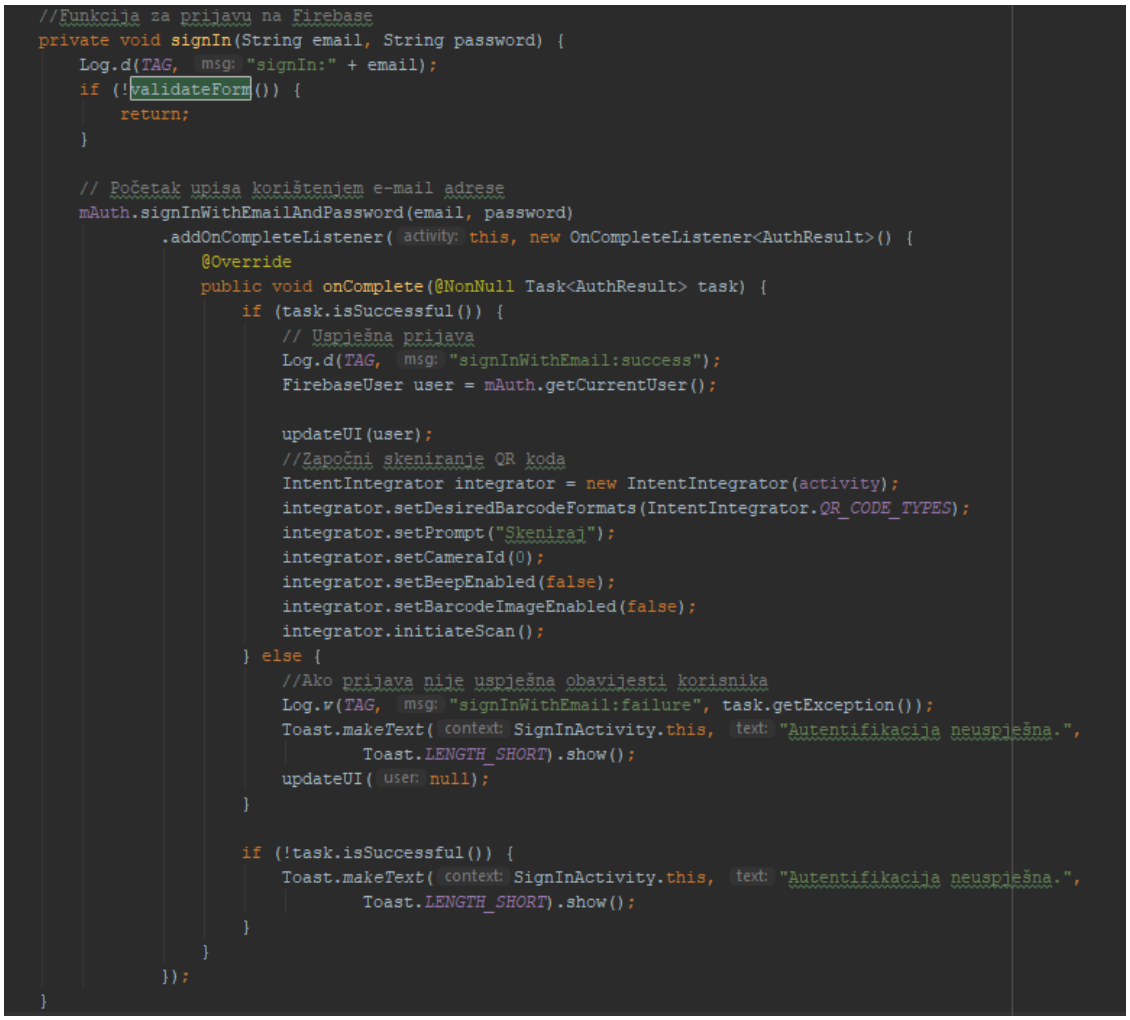
4.1. Izrada Android aplikacije

Kao što je već spomenuto, za izradu Android aplikacije korišteno je *Android Studio* razvojno okruženje i programski jezik Java. Za autentifikaciju korisnika korišten je *Google Firebase* i za skeniranje QR koda korištena je *ZXing* biblioteka i *Google Volley* biblioteka za rad s POST zahtjevom.

Za ispravan rad mobilne aplikacije prvo je potrebno napraviti aktivnost i pogled koji će ta aktivnost koristiti. U ovom slučaju pogled se sastoji od dva polja za unos teksta (engl. *EditText*) od kojih se u prvi upisuje dodijeljena e-mail adresa studenta, a u drugi dodijeljena lozinka. Osim polja za unos teksta postoje dva gumba (prijava i odjava) od kojih je gumb *prijava* nužan za funkcionalnost aplikacije, a gumb *odjava* nije prijeko potreban, ali je dobro imati ga u slučaju neispravnog rada aplikacije.

Kod stvaranja aktivnosti, prije nego program može početi koristiti, potrebno je deklarirati globalne varijable i postaviti pronalazak svih pogleda za unos polja i gumba te početnu inicijalizaciju Firebase autorizacije. Postavljanje svega navedenog osim globalnih varijabli se radi u *onCreate()* funkciji koja se izvodi prilikom stvaranje svake nove instance aplikacije. Kada je sve stvoreno potrebno je provjeriti je li korisnik prijavljen i ovisno o tome promijeniti izgled ekrana aplikacije. Za izgled ekrana aplikacije korištena je funkcija *updateUI(FirebaseUser user)* koja kao parametar prima trenutnog korisnika i ukoliko on ne postoji prikazuje pogled s formom za prijavu.

Funkcija za prijavu (slika 4.1.) *signIn(String email, String password)* kao parametre prima nizove znakova koji predstavljaju e-mail i lozinku korisnika. Ova funkcija provjerava je li zahtjev za autorizaciju uspješno obavljen. Ako je taj zahtjev uspješno obavljen otvara se skener QR koda (označen komentarom na slici 4.1.).

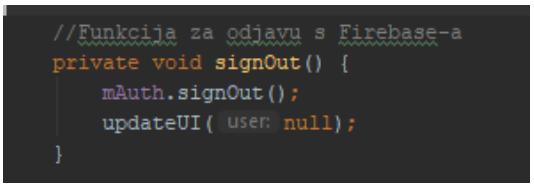


```
//Funkcija za prijavu na Firebase
private void signIn(String email, String password) {
    Log.d(TAG,  msg: "signIn:" + email);
    if (!validateForm()) {
        return;
    }

    // Početak upisa korištenjem e-mail adrese
    mAuth.signInWithEmailAndPassword(email, password)
            .addOnCompleteListener( activity: this, new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    if (task.isSuccessful()) {
                        // Uspješna prijava
                        Log.d(TAG,  msg: "signInWithEmail:success");
                        FirebaseUser user = mAuth.getCurrentUser();

                        updateUI(user);
                        //Započni skeniranje QR koda
                        IntentIntegrator integrator = new IntentIntegrator(activity);
                        integrator.setDesiredBarcodeFormats(IntentIntegrator.QR_CODE_TYPES);
                        integrator.setPrompt("Skeniraj");
                        integrator.setCameraId(0);
                        integrator.setBeepEnabled(false);
                        integrator.setBarcodeImageEnabled(false);
                        integrator.initiateScan();
                    } else {
                        //Ako prijava nije uspješna obavijesti korisnika
                        Log.w(TAG,  msg: "signInWithEmail:failure", task.getException());
                        Toast.makeText( context: SignInActivity.this,  text: "Autentifikacija neuspješna.",
                                Toast.LENGTH_SHORT).show();
                        updateUI( user: null);
                    }

                    if (!task.isSuccessful()) {
                        Toast.makeText( context: SignInActivity.this,  text: "Autentifikacija neuspješna.",
                                Toast.LENGTH_SHORT).show();
                    }
                }
            });
}
```

Slika 4.1. Funkcija za prijavu korisnika pomoću Firebase-a

Ako autorizacija nije uspješna aplikacija će izbaciti odgovarajuću poruku. Prije početka autorizacije korisnika, provjerava se validnost forme, odnosno jesu li svi zadaci uneseni ispravno. Funkcija za validaciju forme *validateForm()* provjerava jesu li sva polja ispunjena i ukoliko jesu vraća *bool* varijablu kao istinu, a u suprotnome prikazuje odgovarajuću poruku korisniku.

Za odjavu iz aplikacije koristi se funkcija *signOut()* (slika 4.2.) koja poziva ugrađenu Firebase funkciju za odjavu korisnika i poziva funkciju *updateUI(null)* s kojom javlja da se treba ažurirati pogled i ponovno prikazati formu za prijavu.



```
//Funkcija za odjavu s Firebase-a
private void signOut() {
    mAuth.signOut();
    updateUI( user: null);
}
```

Slika 4.2. Funkcija za odjavu korisnika pomoću Firebase-a

Kada se radi skeniranje koda provjerava se rezultat skeniranja i je li pritisnut gumb za povratak. Ako je pritisnut gumb za povratak javlja se poruka „Skeniranje prekinuto“ i korisnik je odjavljen, u suprotnome dohvaća se rezultat skeniranja *contents* i poziva se funkcija *CallAPI(contents)* koja izvodi POST zahtjev na WEB aplikaciju te se nakon toga korisnik odjavljuje.

Funkcija *CallAPI(final String contents)* (slika 4.3.) kao parametar prima rezultat skeniranja QR koda koji će kasnije biti upotrjebljen kao jedan od parametara koji će se slati POST zahtjevom.

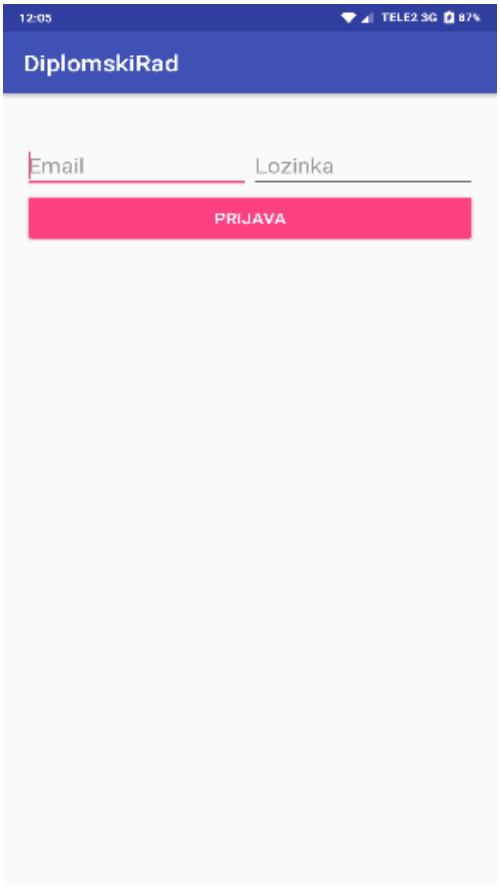
```
//Funkcija za slanje podataka na WEB aplikaciju s podacima studenta
public void CallAPI(final String contents)
{
    final FirebaseUser user = mAuth.getCurrentUser();
    // Započni RequestQueue
    RequestQueue queue = Volley.newRequestQueue( context: this);
    String url ="http://attendence.azurewebsites.net/test/AddStudentToLecture"; //URL za POST request
    StringRequest postRequest = new StringRequest(Request.Method.POST, url,
            new com.android.volley.Response.Listener<String>()
            {
                //Ako je student uspješno upisan na predavanje javi porukom
                @Override
                public void onResponse(String response) {
                    // response
                    Log.d( tag: "Response", response);
                    Toast.makeText(getApplicationContext(),  text: "Uspješno upisani na predavanje!", Toast.LENGTH_LONG).show();
                }
            },
            new com.android.volley.Response.ErrorListener()
            {
                //Ako je student neuspješno upisan na predavanje javi porukom
                @Override
                public void onErrorResponse(VolleyError error) {
                    // error
                    Log.d( tag: "Error.Response", error.toString());
                    Toast.makeText(getApplicationContext(),  text: "Ups, nešto je pošlo po krivu!", Toast.LENGTH_LONG).show();
                }
            }
    ) {
        //Stvaranje vrste POST zahtjeva i dodavanje parametara
        @Override
        public String getBodyContentType() {
            return "application/x-www-form-urlencoded; charset=UTF-8";
        }
            @Override
            protected Map<String, String> getParams() throws AuthFailureError
            {


                Map<String, String> params = new HashMap<~>();
            params.put( k: "UserName",user.getEmail().toString());
            params.put( k: "LectureID",contents);

            return params;
        }

    };
    queue.add(postRequest);
}
```

Slika 4.3. Kod funkcije *CallAPI*

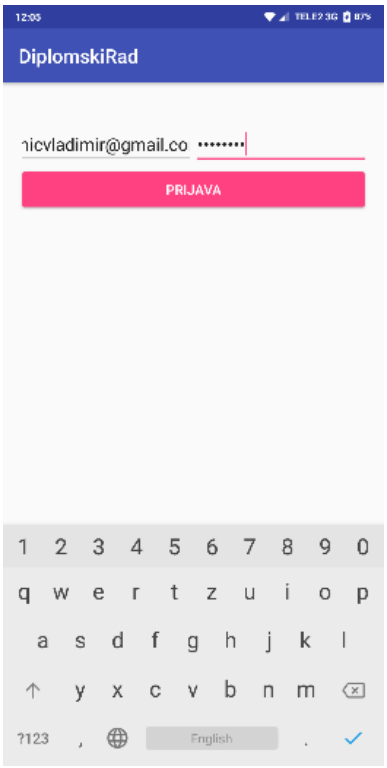
Funkcija *CallAPI* ovisi o biblioteci *Volley* koja je napravljena od strane Google-a i služi za olakšani rad s HTTP zahtjevima [7]. Prvo je potrebno napraviti novi red zahtjeva i postaviti adresu na koju će se slati zahtjev i vrsta zahtjeva. Nakon toga se postavlja osluškivanje odgovora na zahtjev od strane WEB aplikacije koji ovisno o odgovoru daje odgovarajuću poruku. Također, potrebno je koja vrsta objekta se šalje POST zahtjevom (u ovome slučaju je to JSON objekt s dva parametra)

i ubaciti te parametre u tijelo zahtjeva. Ova funkcija u tijelo zahtjeva postavlja korisničko ime prijavljenog korisnika i ID predavanja na koje se upisuje (ovaj ID je dohvaćen čitanjem QR koda). Nakon što je tijelo zahtjeva gotovo, dodaje se u red i funkcija se ponavlja dokle god red nije pun ili nema više zahtjeva.

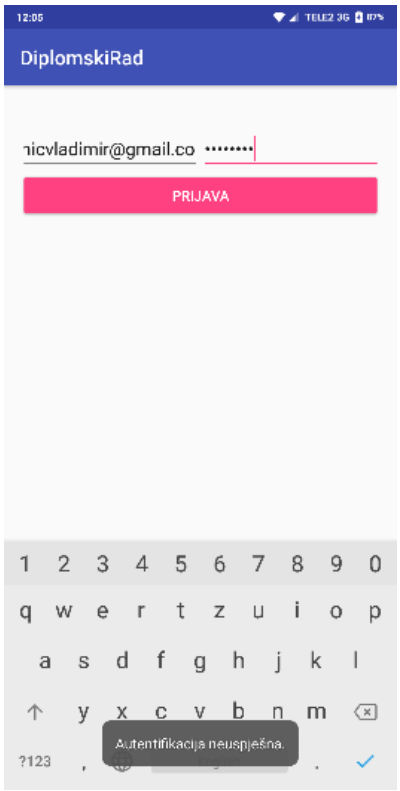
Slika 4.4. prikazuje zaslone gotove aplikacije s opisanim funkcionalnostima.



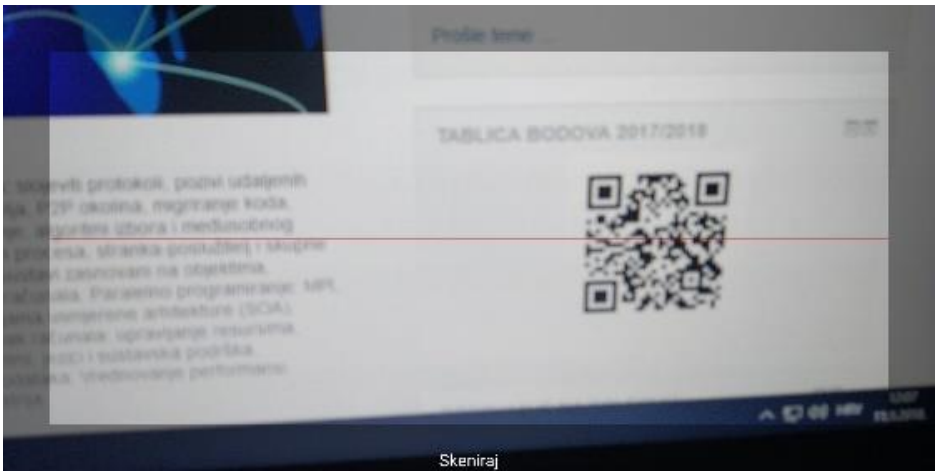
a) Početni zaslon aplikacije



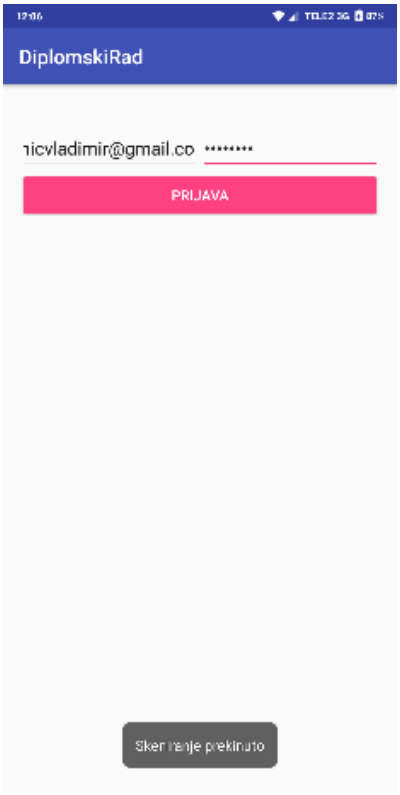
b) Početni zaslon s upisanim podacima



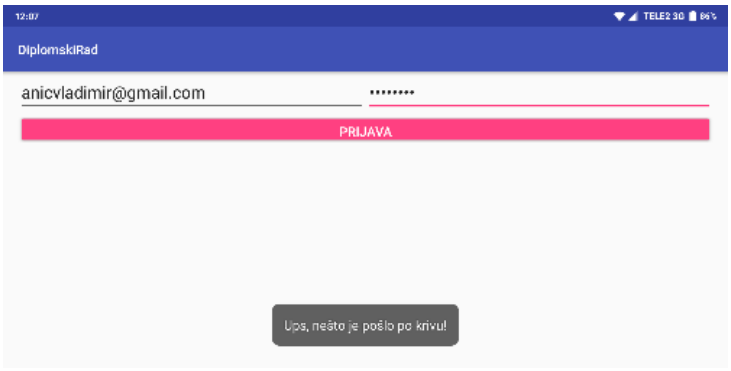
c) Poruka kada su uneseni podaci neispravni



d) Ekran skeniranja

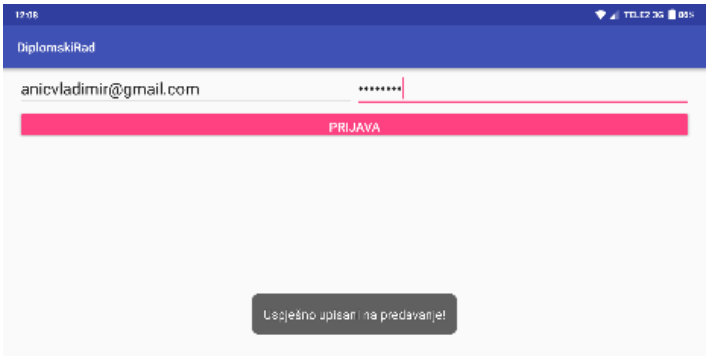


e) Prekid skeniranja



f) Loš zahtjev (engl. *Bad request*)

g) Traženje točaka za skeniranje

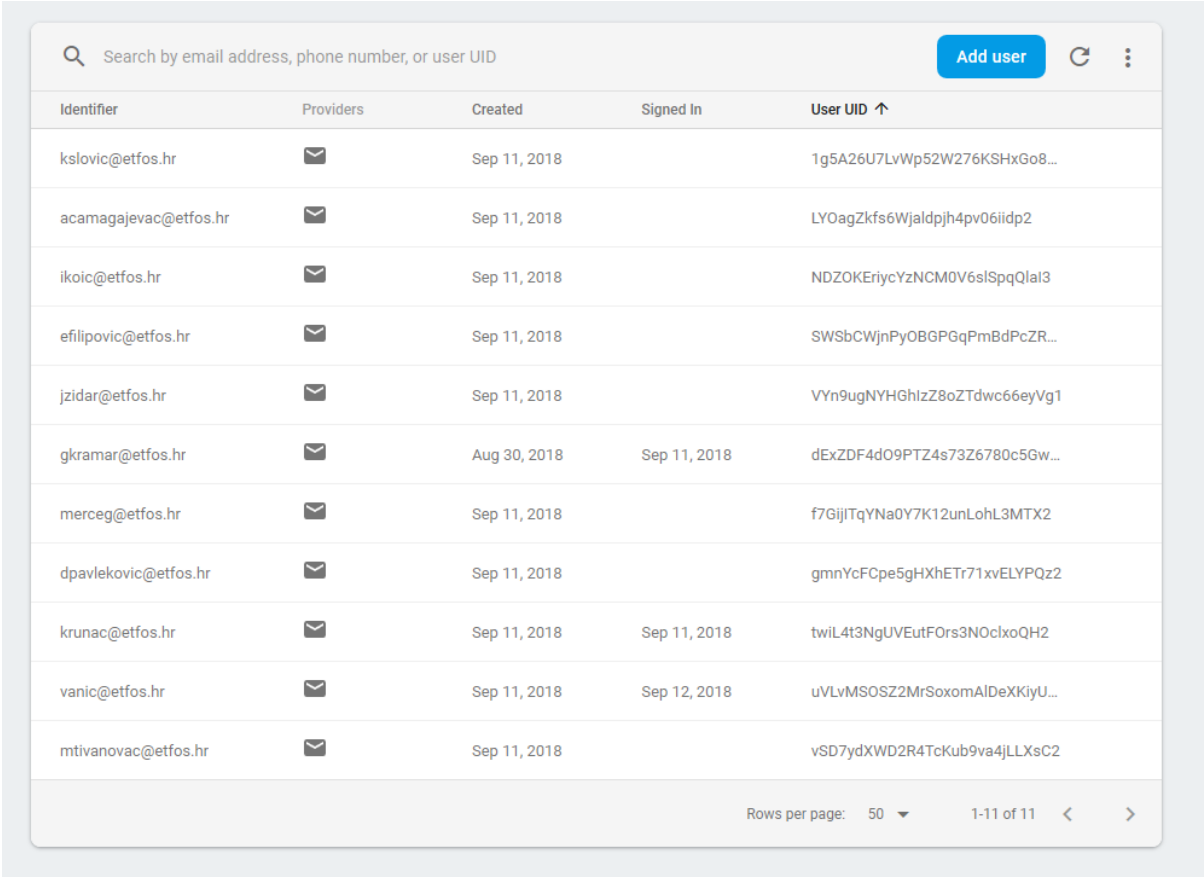


h) Uspješan upis na predavanje

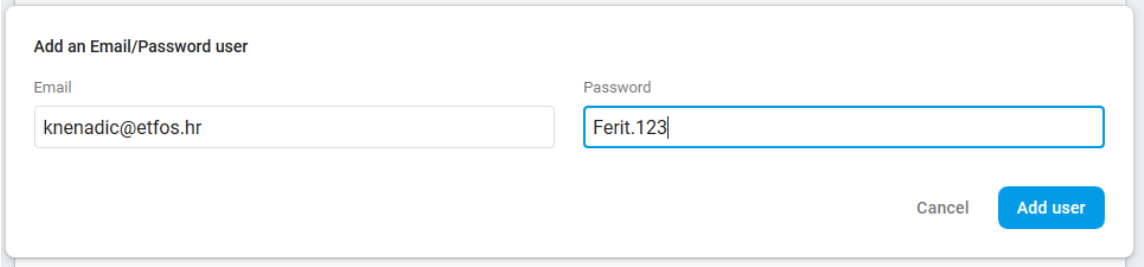
i) Forma nije validna

Slika 4.4. Funkcionalnost mobilne aplikacije

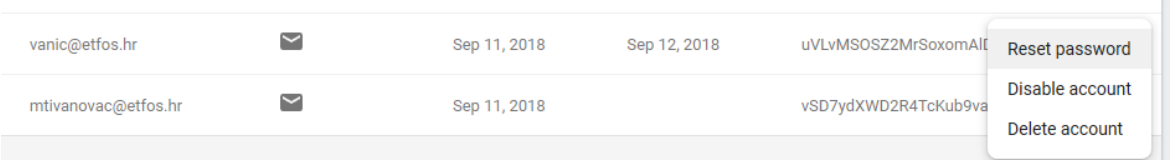
Korisnike je potrebno ručno dodati u Firebase bazu podataka i posebno paziti da im se korisničko ime (e-mail) slaže s korisničkim imenom (e-mailom) na WEB aplikaciji. Korisnicima je moguće poslati e-mail pomoći kojega mogu promijeniti svoju lozinku. Slika 4.5. prikazuje postupak dodavanja novih korisnika i postupak promjene lozinke.



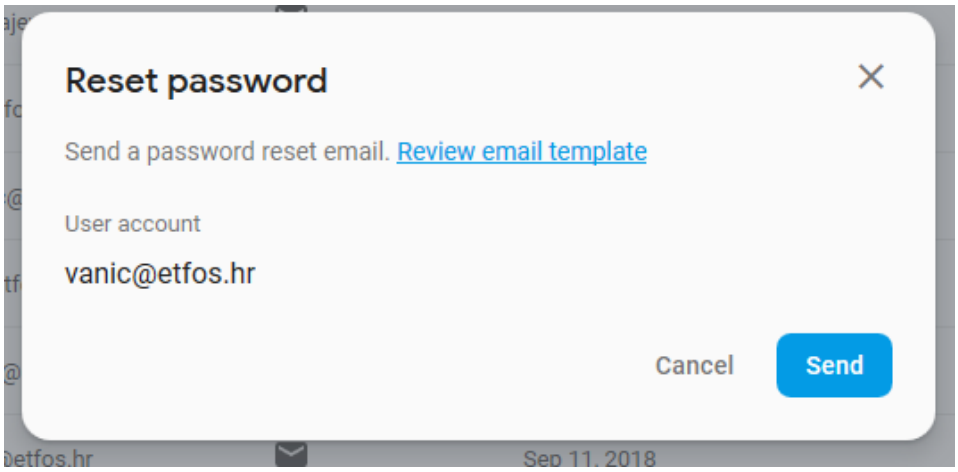
a) Lista korisnika s mogućnosti dodavanja u bazu podataka



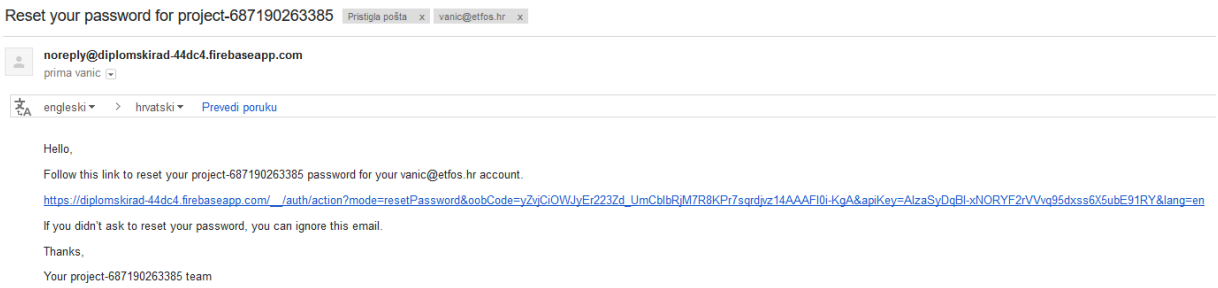
b) Forma za upis novog korisnika



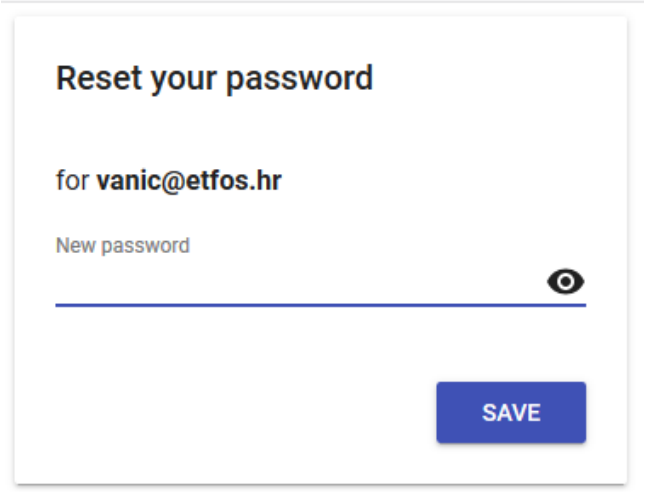
c) Mogućnosti upravljanja računom



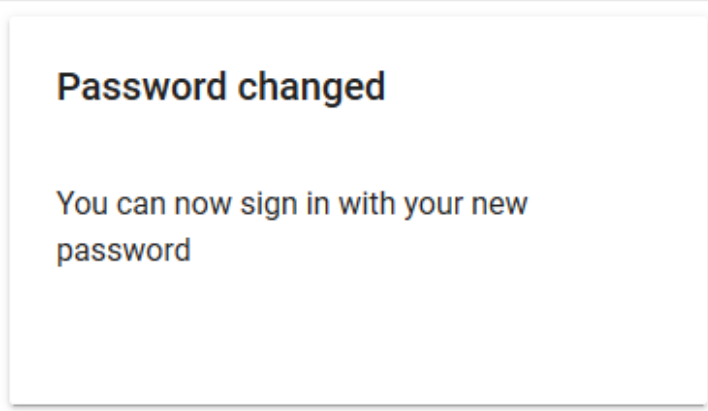
d) Potvrda zahtjeva za slanje promjene lozinke



e) E-mail promjene lozinke



f) Forma za promjenu lozinke



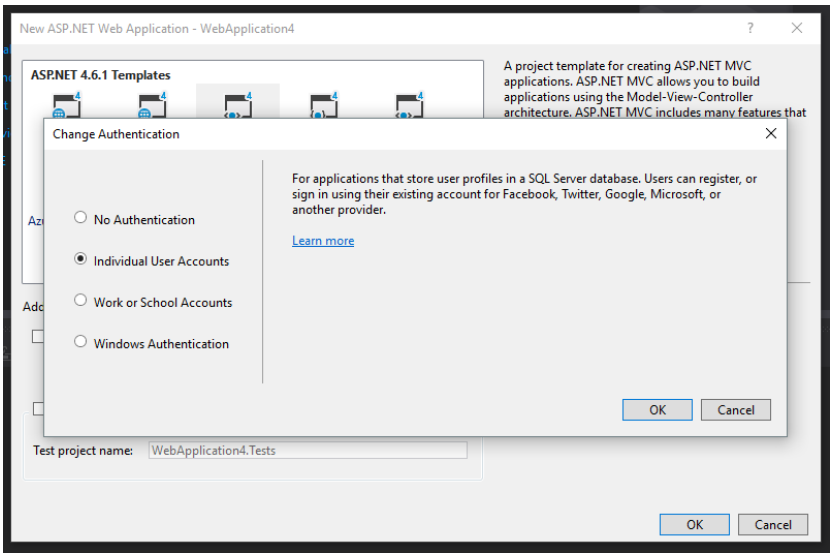
g) Poruka o uspješnoj promjeni lozinke

Slika 4.5. Postupak dodavanja korisnika i promjene lozinke za Firebase

4.2. Izrada WEB aplikacije

Za izradu WEB aplikacije korišteno je *Visual Studio 2017* razvojno okruženje i programski jezik C# za izradu aplikacije na poslužiteljskoj strani. Za objavu aplikacije i udaljenu bazu podataka korišten je *Microsoft Azure* portal kako bi se mogla testirati funkcionalnost web stranice u potpunosti te je za generiranje QR koda korištena *ZXing* biblioteka.

Prvo je potrebno napraviti projekt s ugrađenom autentifikacijom (engl. *Individual User Accounts*) kao što je prikazano na slici 4.6.



Slika 4.6. Stvaranje projekta s autentifikacijom

Ovo je bitno kako bi mogli kasnije povezati račun i autentifikaciju s tablicom osoba u bazi podataka koje će biti provjeravane za upis na predavanje (studenti) ili administrirati WEB aplikacijom (profesori). Kada je projekt napravljen, prvo je potrebno napraviti modele koji će predstavljati tablice u bazi podataka. Korišten je engl. *Code-first* pristup bazama podataka što znači da se korištenjem migracija generiraju tablice u bazi podataka i uključivanjem određenih kolekcija dobiju se relacije između tablica. Nakon što su napravljeni modeli potrebno je napraviti migracije koje spajaju sve modele relacijama i generiraju tablice koje će biti potrebne kasnije za korištenje i dohvaćati podaci pomoću *LINQ* upita. Kada su migracije napravljene, potrebno je ažurirati bazu podataka kako bi se primijenile sve promjene nad njom. Također, u modelima se postavljaju određena ograničenja nad upitima i koji su atributi potrebni za upis podataka, kakav će format biti određenim podacima, itd. Nakon što su modeli završeni te sva ograničenja i formati postavljeni, potrebno je napraviti kontrolere koji sadržavaju reference na modele i rade upite, te izvlače podatke iz baze podataka i šalju ih dalje na poglede. Najjednostavniji način izrade kontrolera je da se napravi novi objekt pomoću skela (engl. s*caffolded item*) pomoću kojega *Visual Studio* generira osnovne metode u kontroleru i stvara poglede koje kontroler koristi za unos podataka u bazu i prikaz podataka iz baze podataka. Bitno je za naglasiti da nakon što su napravljeni osnovni kontroleri i pogledi pomoću skela oni neće biti savršeni i bit će ih potrebno izmijeniti da mogu raditi željene funkcionalnosti. Osim klasičnih MVC kontrolera u ovome radu napravljen je i jedan API kontroler u kojemu je omogućen rad s HTTP zahtjevima. Pomoću tog kontrolera obrađuju se zahtjevi kojima se upisuju studenti na predavanje. Za upis studenata na predavanje napravljena je funkcija *AddStudent(APIAddStudentBM addStudentBM)* koja prima objekt klase *APIAddStudentBM* kao parametar. Ako je poslani model od strane korisnika validan, zahtjev se dalje izvršava i radi se provjera postoji li taj student u bazi, postoji li predavanje u bazi te je li to predavanje aktivno. Ako je sve prošlo provjeru, student se upisuje u bazu i spremaju se promjene, dok ako nije, funkcija vraća poruku pošiljatelju zahtjeva da je došlo do problema (ili ne postoji student, predavanje ili nije dobar model). Slika 4.7. predstavlja funkciju dodavanja studenta u bazu podataka.

```
[Route("AddStudentToLecture")]
[HttpPost]
public IHttpActionResult AddStudent(APIAddStudentBM addStudentBM)
{
    if (ModelState.IsValid)
    {
        var student = db.Person.FirstOrDefault(p => p.UserName == addStudentBM.UserName);
        var lecture = db.Lecture.FirstOrDefault(l => l.ID == addStudentBM.LectureID);
        if (student != null && lecture!=null && lecture.IsActive==true)
        {

            lecture.People.Add(student);
            db.SaveChanges();
            return Ok("uspjesno upisano");
        }
        else
            return BadRequest("student ne postoji u bazi, predavanje zavrsilo ili ne postoji");
    }
    return BadRequest("Nesto je poslo po krivu");
}
```

Slika 4.7. Funkcija koja obrađuje POST zahtjev za upis studenta na predavanje

Klasa *APIAddStudentBM* služi ako pomoćna klasa za stvaranje objekata koji će se koristiti u POST zahtjevu. Ona se sastoji od niza znakova za korisničko ime – *string UserName* i cjelobrojne vrijednosti identifikatora predavanja – *int LectureID*. Oba atributa klase su obvezni i bez njih nije moguće napraviti POST zahtjev s vanjske aplikacije.

Na početnom pogledu WEB aplikacije nalaze se osnovne informacije o ovoj aplikaciji (za koga je namijenjena i čemu služi), u navigacijskoj traci WEB aplikacije nalazi se kontakt s razvojnim inženjerom te poveznica na formu za prijavu korisnika. Nakon što je korisnik prijavljen, ovisno o njegovoj ulozi mu se prikazuje forma za registraciju novog korisnika (administrator) ili popis kolegija tog korisnika (profesor i adminstrator).

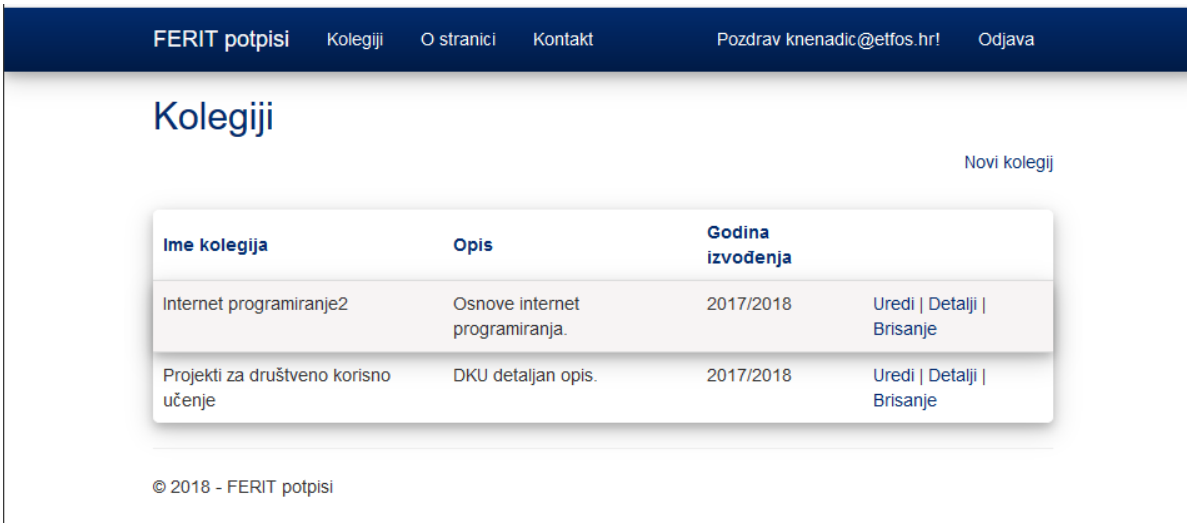
Kolegiji se sastoje od nekoliko različitih pogleda:

- početne stranice (engl. *Index*),
- detalji (engl. *Details*),
- uređivanje (engl. *Edit*),
- stvaranje novog (engl. *Create*),
- brisanje (engl. *Delete*),

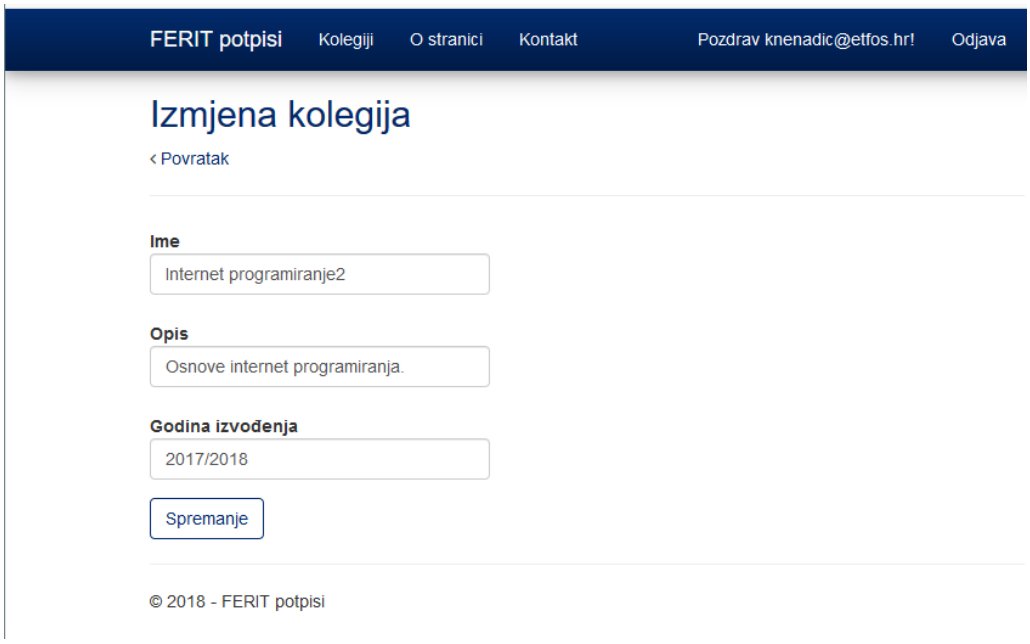
jednog modela (*Subjects*) i jednog kontrolera (*SubjectsController*). Model kolegija u sebi sadrži svoj identifikator (*ID*), naziv kolegija (engl. *Name*), opis kolegija (engl. *Decription*), godinu izvođenja (engl. *Year*), kolekciju predavanja (engl. *Lectures*) koja služi kao strani ključ na tablicu predavanja i kolekciju osoba (engl. *People*) koja služi kao strani ključ na tablicu osoba, odnosno

definira vezu između tih tablica. Kontroler za upravljanje kolegijima (engl. *SubjectsController*) sadrži sve metode i akcije za dohvaćanje podataka iz baze i prikaz istih u pogledima.

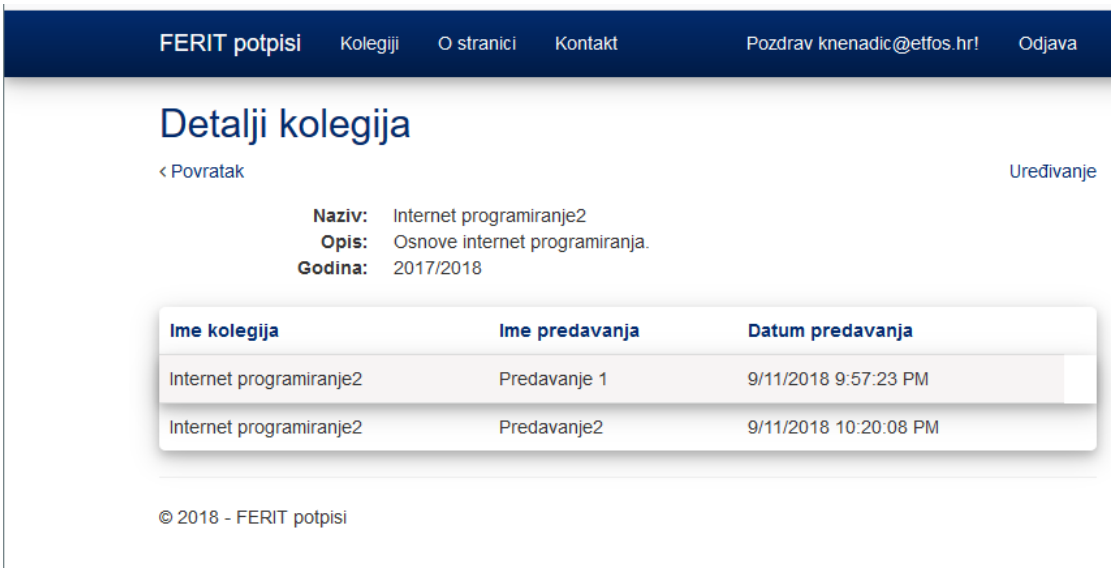
Metoda *Index()* dohvaća sve kolegije po prijavljenom korisniku i prikazuje ih u obliku tablice na početnoj stranici tog pogleda. Metoda *Details(int? id)* kao parametar prima jedinstveni identifikator na osnovu kojega dohvaća informacije o kolegiju i sva predavanja napravljena za taj kolegij. Metoda *Create([Bind(Include = "ID,Name,Description,Year")] Subject subject)* kao parametar prima objekt klase *Subject* i vezuje sve parametre te klase s samim objektom. Ako je model koji prima validan (ima sve potrebne atribute), radi se provjera uloge korisnika koji radi zahtjev i ako je ulogiran profesor, on se dodaje u listu i povezuje s tim predavanjem te se dodaju sve informacije za kolegij i kolegij se sprema u bazu. Kada je kolegij spremljen u bazu, korisnik je vraćen na početnu stranicu s listom svih kolegija. Metoda *Edit([Bind(Include = "ID,Name,Description,Year")] Subject subject)* prima iste parametre kao i metoda *Create*, ali ona provjerava samo promjenu podataka i ako postoje, upisuje ih u bazu. Metoda *Delete(int? id)* samo pronalazi o kojem se kolegiju radi, dok metoda *DeleteConfirmed(int id)* potvrđuje brisanje kolegija i briše isti iz baze podataka. Na slici 4.8. mogu se vidjeti svi pogledi s funkcionalnostima vezani za kolegij.



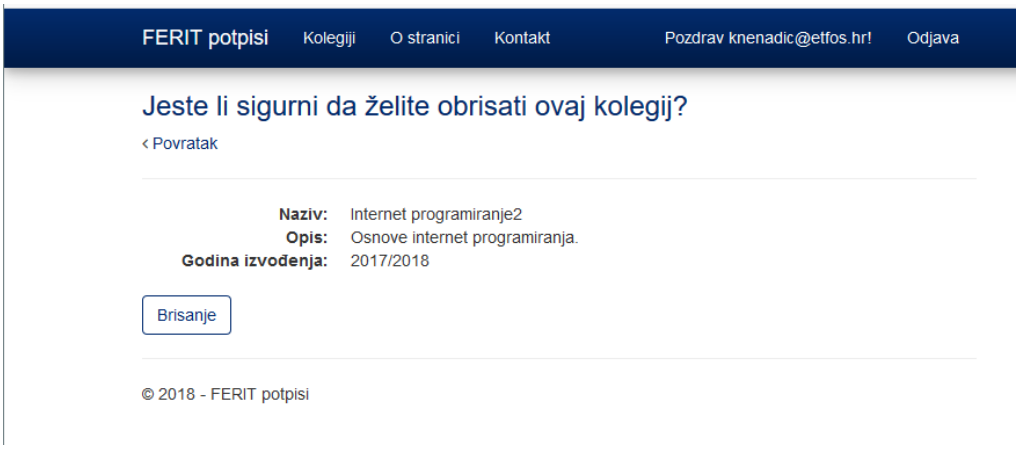
a) Pregled svih kolegija



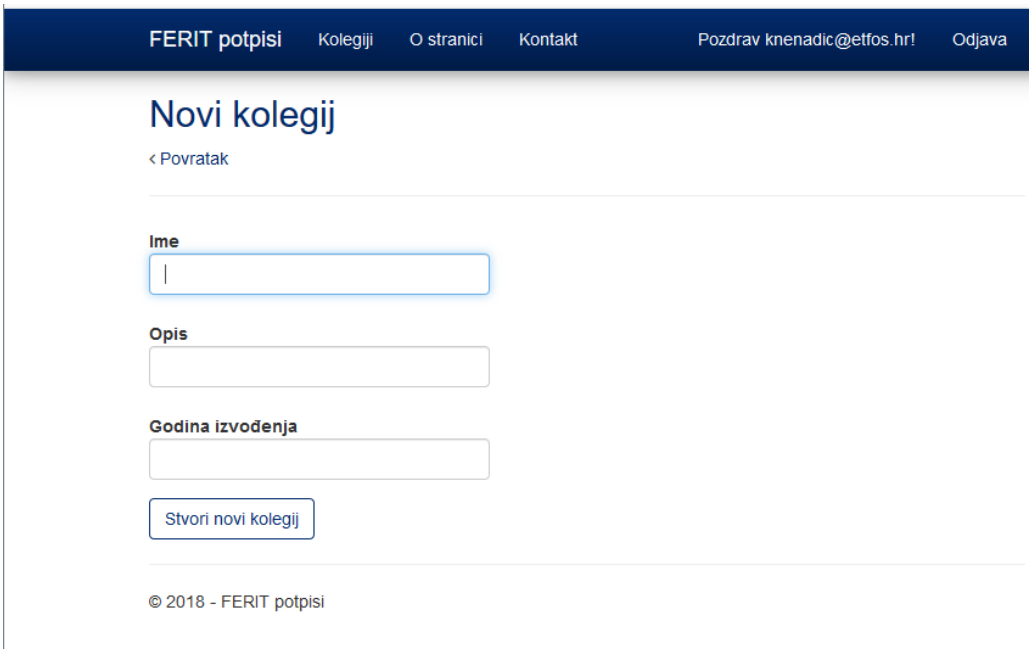
b) Forma za uređivanje postojećeg kolegija



c) Pregled detalja kolegija s već upisanim predavanjima



d) Pregled brisanja kolegija



e) Forma za stvaranje novog kolegija

Slika 4.8. Pogledi vezani za kolegij

Predavanja se kao i kolegiji sastoje od nekoliko različitih pogleda:

- početne stranice (engl. *Index*),
- detalji (engl. *Details*),
- uređivanje (engl. *Edit*),
- stvaranje novog (engl. *Create*),
- brisanje (engl. *Delete*),
- prijava studenta (engl. *SignInStudent*),

jednog modela (engl. *Lectures*) i jednog kontrolera (engl. *LecturesController*) za upravljanje predavanjima. Model predavanja u sebi sadrži svoj identifikator (*ID*), naziv predavanja (engl. *Name*), datum predavanja (engl. *Date*), identifikator kolegija (engl. *SubjectID*), provjeru je li kolegij aktivan (engl. *IsActive*), kolekciju osoba (engl. *People*) koja služi kao svojevrsni strani ključ na tablicu osoba i objekt predavanja (engl. *Subject*) koji zajedno s identifikatorom kolegija povezuje te dvije tablice izravno stvarajući vezu jedan na više između kolegija i predavanja. Kontroler u sebi sadrži sve metode i akcije potrebne za rad s podacima iz baze podataka i prikazom istih u pogledima.

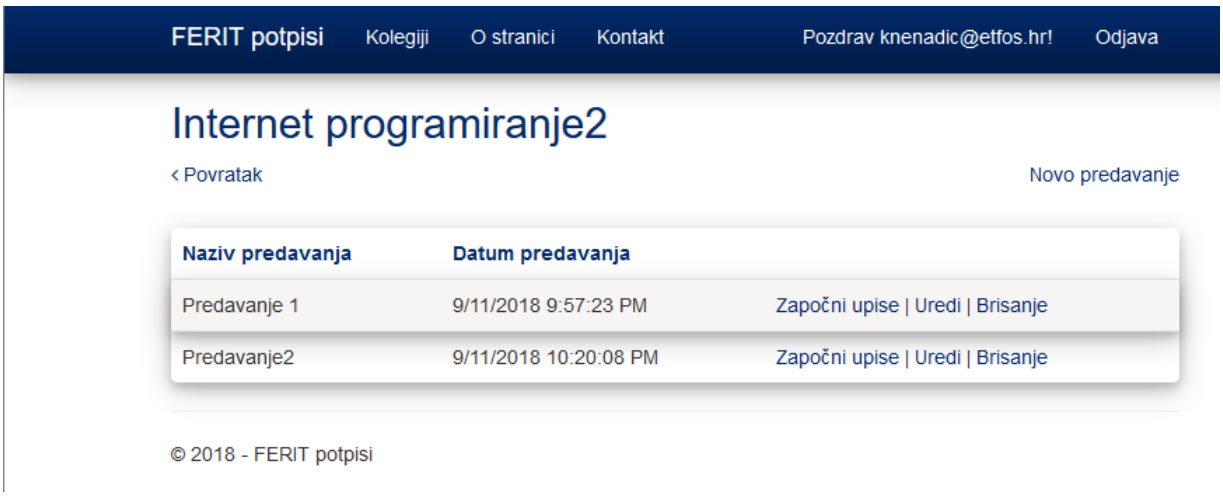
Metoda *Index(int? id)* kao parametar prima identifikator kolegija i na osnovu istog radi provjeru i prikazuje sva predavanja na osnovu tog kolegija (klikom na kolegij izbacuju se sva njegova predavanja). Metoda *Details(int? id)* prima identifikator osoba kao parametar i na osnovu njega

vraća odgovarajući rezultat. Odnosno, ako je identifikator nepostojeći vraća odgovarajuću poruku, a ako on postoji uključuju se sve osobe koje imaju taj identifikator i provjeravaju se sve osobe koje su upisane na predavanje i ispisuju se iste u tablicu. U pogledu detalja predavanja postoje 2 gumba kojima se rade različite akcije. Gumb „Započni upise“ aktivira funkciju *ActivateLectureDetailsPage(int id)* pomoću koje se dolazi do metode za generiranje QR koda koji sadrži identifikator predavanja i ako je predavanje aktivno drugi gumb „Upiši studenta“ nestaje dok se ne zatvori upis i omogući ručno upisivanje studenta. Generiranje QR koda se izvodi u klasi *QRHelper* pomoću metode *GenerateQrCode()* koja pravi QR kod na osnovu znaka koji je zadan i pomoću nje se mogu mijenjati razni atributi kao što su veličina slike i čak vrsta izlazne slike. Gumb „Upiši studenta“ otvara modalnu formu u kojoj su prikazani svi studenti i moguće je odabrati studenta i upisati ga. Modalna forma i pogled za istu se nalaze u pogledu za prijavu studenta i on se sastoji padajućeg izbornika u kojemu se mogu tražiti studenti po korisničkom imenu te nakon odabira ih upisati u predavanje ili zatvoriti modalnu formu i prekinuti ručni unos studenta. Postoje dvije različite metode koje upravljaju stvaranjem novog predavanja. Prva metoda *Create(int id)* koja prima identifikator posljednjeg predavanja kao parametar i pomoću njega dohvaća pogled koji će prikazati odgovarajuće podatke forme za stvaranje novog predavanja. Druga metoda *Create([Bind(Include = "ID, Name, Date, SubjectID, IsActive")] Lecture lecture)* kao parametar prima objekt predavanja koji vezuje sa svim parametrima tog objekta. Kada je model validan datum se postavlja na trenutni datum kada se pravi predavanje, dodaju se svi podaci upisani u formu, podaci se spremaju u bazu podataka i vraća se na početnu stranicu s listom svih predavanja. Metode *Edit* imaju vrlo sličnu funkcionalnost metodama *Create* uz iznimku da su njima forme popunjene s prijašnjim podacima. Metoda *Delete(int? id)* vraća pogled za brisanje predavanja, a metoda *DeleteConfirmed(int id)* briše samo predavanje iz baze podataka ovisno o njegovome identifikatoru i sprema promjene koje su se dogodile brisanjem istih. Metoda *ActivateLecture(int id)* kao parametar prima identifikator predavanja i negira *bool* varijablu provjere je li predavanje aktivno, odnosno ako je predavanje aktivno deaktivira ga i vraća na pregled svih predavanja, a ako je predavanje neaktivno aktivira ga te generira QR kod i prikazuje ga na stranici detalja. Metoda *ActivateLectureDetailsPage(int id)* ima vrlo sličnu funkcionalnost prethodno opisanoj metodi uz iznimku da ona uvijek prikazuje pogled detalja traženog predavanja. Metoda *AddStudentToLecture(int lectureID, int studentID)* kao parametre prima identifikatore predavanja i studenta i ovisno o njima ispisuje studente koji se prijavljuju na predavanje. Metoda *DeleteStudent(int lectureID, int studentID)* radi slično kao i prethodna metoda uz razliku toga da ona uklanja studente s predavanja, umjesto da ih upisuje na isto. Kod predavanja ubacivanje slika

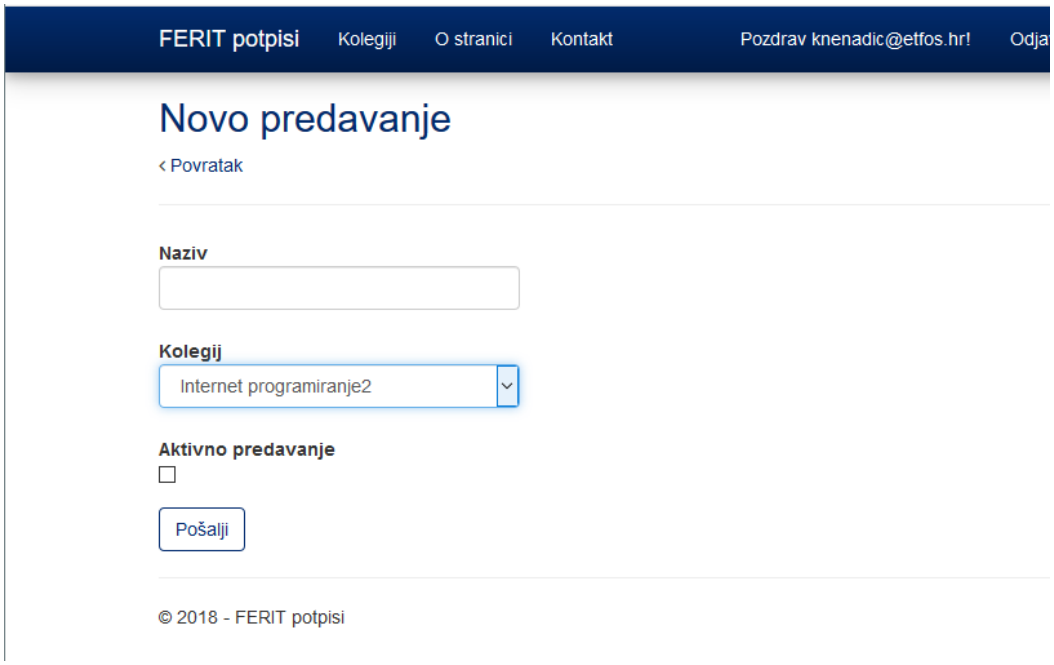
studenata riješeno je koristeći vanjski API [3] koji uzima inicijale imena i prezimena te ih prikazuje kao sliku u tablici kao što se može vidjeti na slici 4.9..

Slika 4.9. Prikaz studenata upisanih na predavanje s pripadajućim slikama

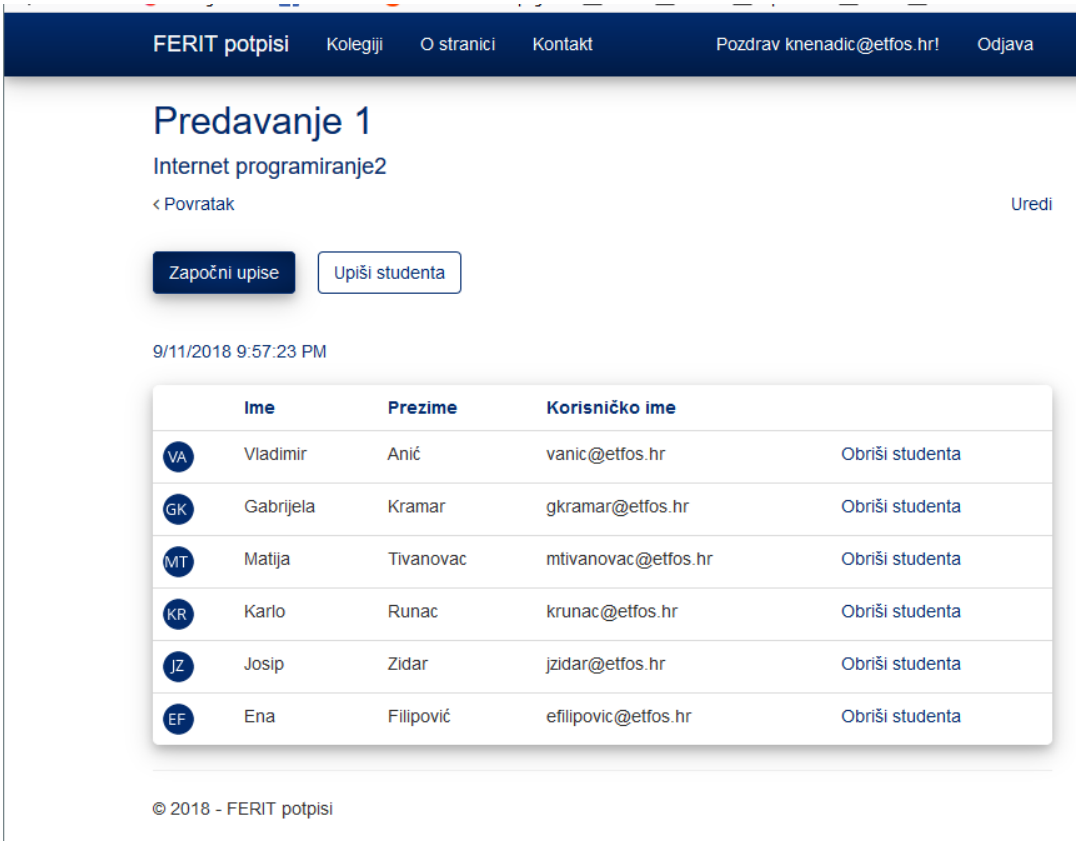
Slika 4.10. prikazuje poglede s funkcionalnostima vezane za predavanja.



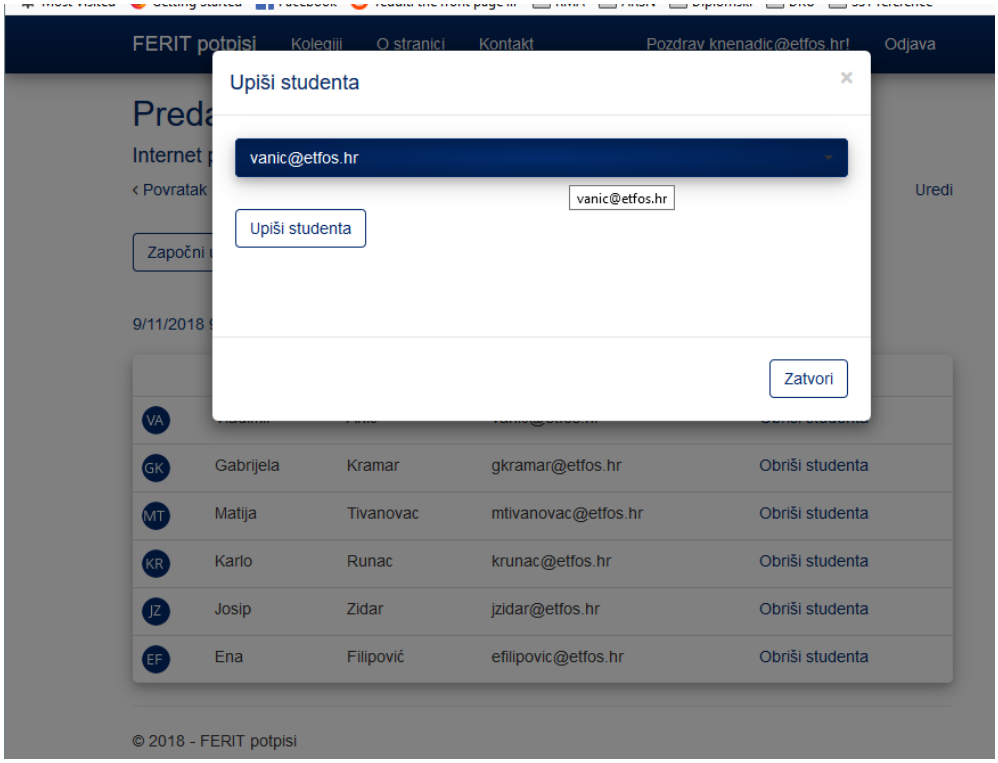
a) Pregled početne stranice s listom kolegija



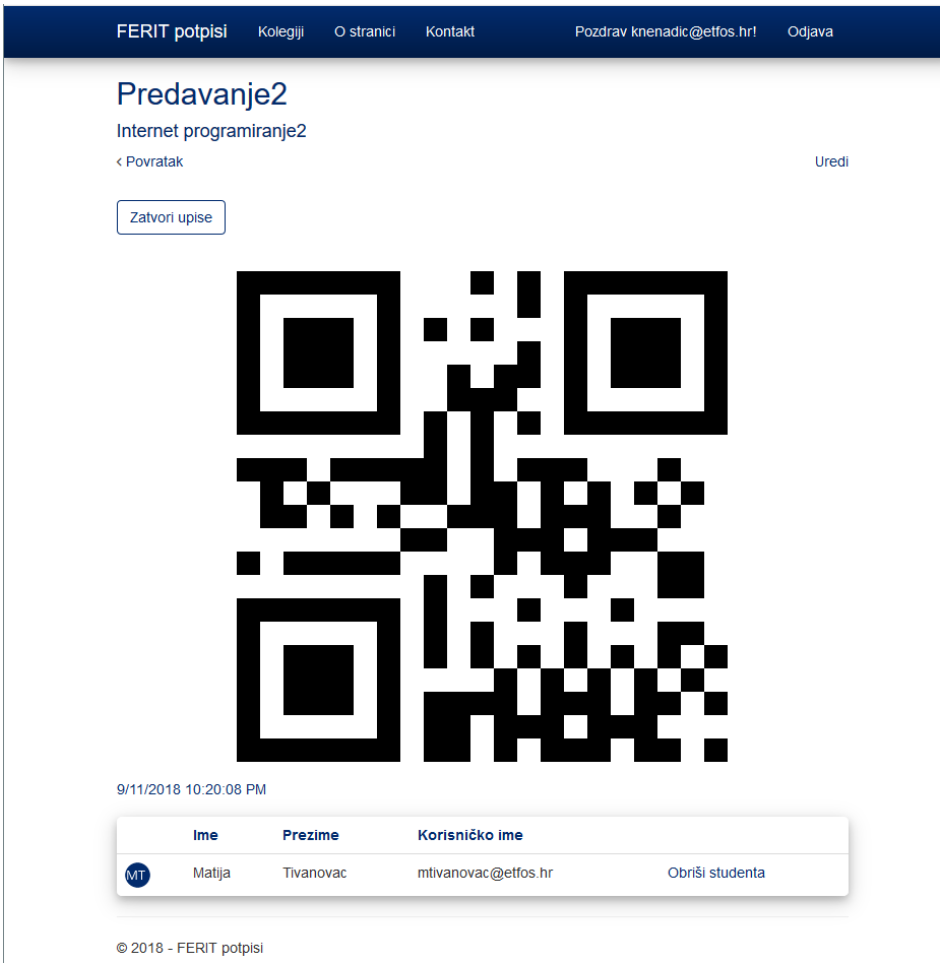
b) Pogled za stvaranje novog predavanja



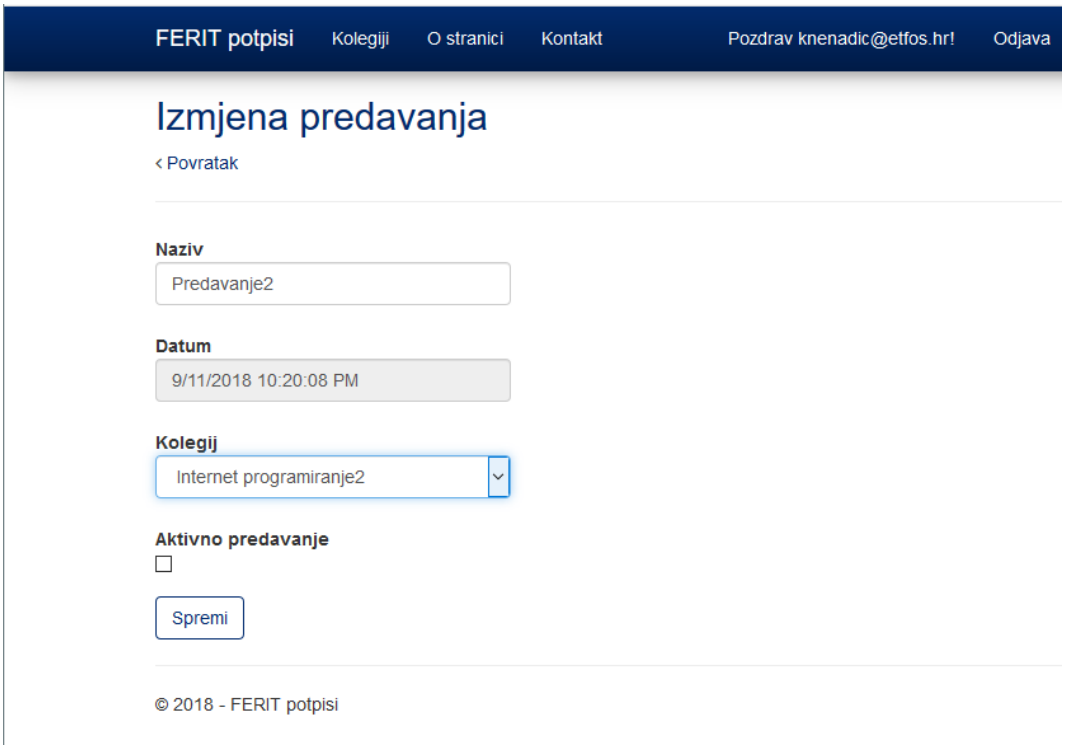
c) Detalji predavanja s završenim upisivanjem



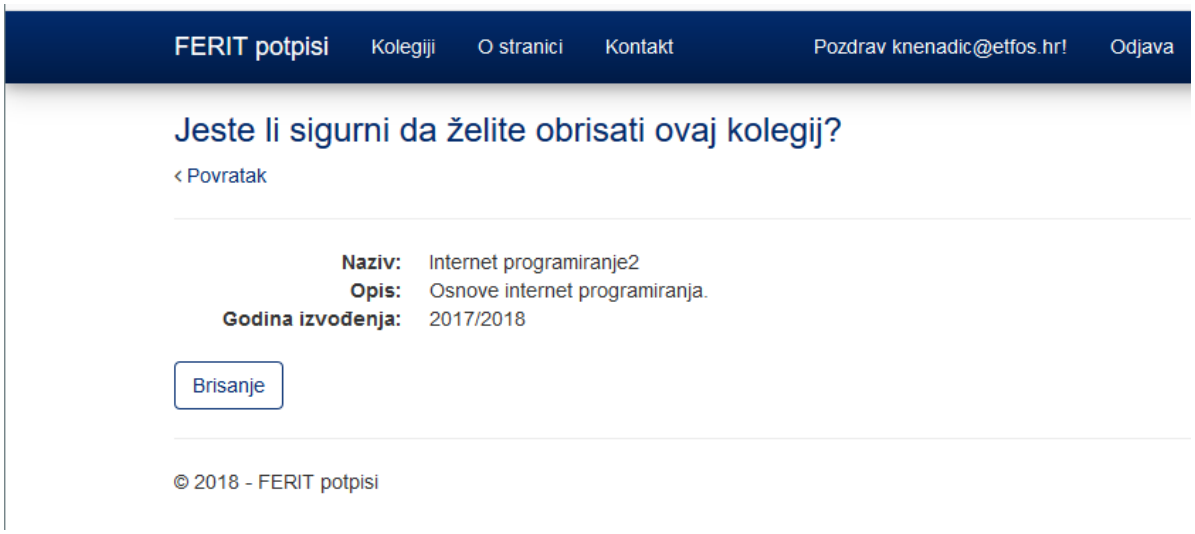
d) Prikaz modalne forme za ručni upis studenata



e) Detalji predavanja s započetim upisivanjem



f) Forma za uređivanje predavanja



g) Pregled brisanja predavanja

Slika 4.10. Pogledi vezani za predavanja

Upravljanje osobama i računima se dijeli na nekoliko različitih modela, pogleda i kontrolera. Prvi model koji je bitan prilikom upravljanja osobama i računima je *AccountViewModels* jer se u njemu definiraju atributi koji će se koristiti za upravljanje s računima, odnosno za promjenu lozinke korisnika (*ChangePasswordViewModel*), registracijom (*RegisterViewModel*) i prijavom (*LoginViewModel*) korisnika. Taj model se koristi u različitim pogledima potrebnim za prijavu, registraciju, ponovno postavljanje lozinke itd. U kontroleru za račune (*AccountController*) nalaze

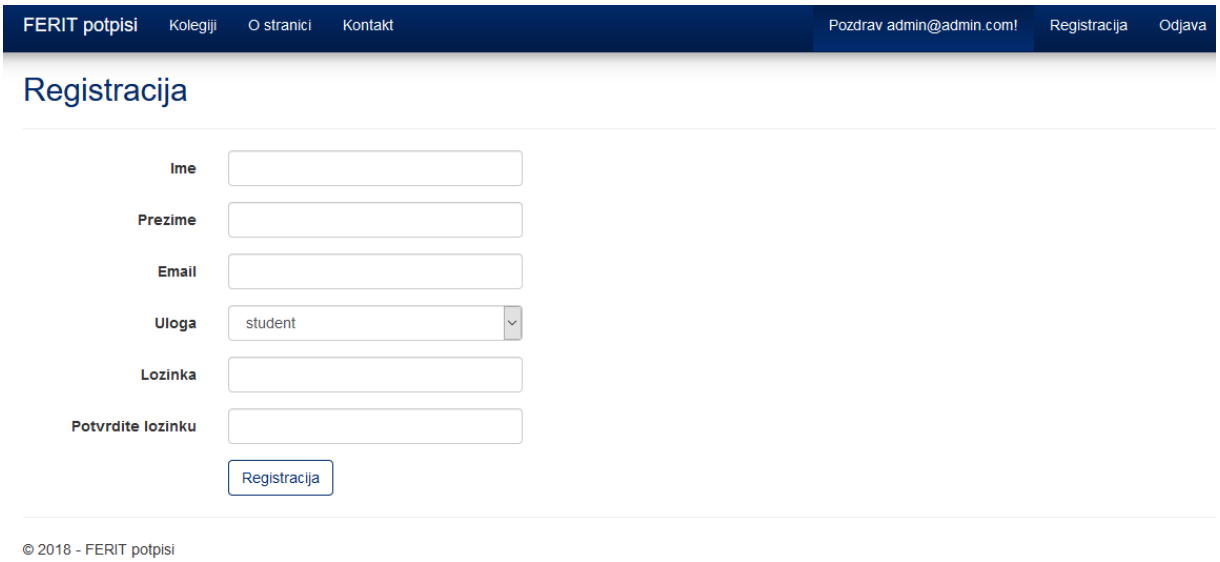
se sve metode potrebne za izradu korisničkih računa, dodavanja uloga korisnicima i upis novih korisnika u bazu podataka. Bitno je naglasiti da ASP.NET MVC nema ugrađeno dodavanje uloga korisnicima te je to bilo potrebno napraviti mijenjanjem već ugrađenih modela za identitete korisnika i primjenom tih metoda u kontroleru za korisničke račune. U klasi *IdentityConfig* dodane su metode kojima se omogućuje izrada metoda stvaranje i upravljanjem uloga (slika 4.11.).

```
public class ApplicationRoleManager: RoleManager<IdentityRole>
{
    public ApplicationRoleManager(IRoleStore<IdentityRole, string> store)
        :base(store)
    {

    }
    public static ApplicationRoleManager Create(IdentityFactoryOptions<ApplicationRoleManager> options, IOwinContext context)
    {
        var manager = new ApplicationRoleManager(new RoleStore<IdentityRole>(context.Get<ApplicationDbContext>()));
        return manager;
    }
}
```

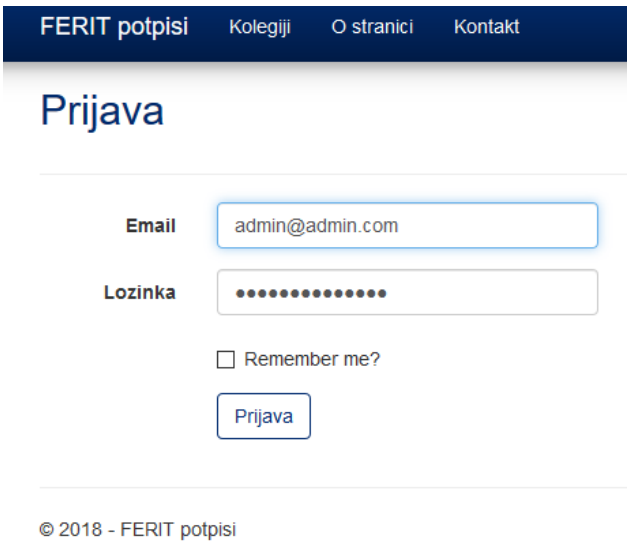
Slika 4.11. Upravljanje i stvaranje uloga

Sada kako bi se uloge mogle stvoriti i upravljati s istima, potrebno je u kontroleru napraviti metode *CreateRole* za izradu uloga i *AddRoleToUser* pomoću kojih će se te uloge dodijeliti korisnicima. Metoda *CreateRole(RoleViewModel model)* uzima podatke iz pogleda *CreateRole* i stvara ulogu. Ovaj pogled je stvoren radi jednostavnosti korištenja i upravljanja ulogama. Metoda *AddRoleToUser* služi za dodavanje uloge korisniku na osnovu njegovog korisničkog imena i koristi se kada administrator registrira korisnika. Metoda *Login(string returnUrl)* služi samo za vraćanje pogleda za prijavu korisnika, ali asinkrona metoda *Login(LoginViewModel model, string returnUrl)* prijavljuje korisnika ovisno o upisanim podacima u formu za prijavu. Ona prima model za prijavu iz kojeg izvlači podatke i provjerava ih s podacima u baze te povratni URL koji koji je vraćen kao rezultat ako je korisnik uspješno prijavljen. Metoda *Register()* vraća formu za registraciju i u njoj se koristi *CommonViewModel* koji sadrži podatke za model osoba i ASP.NET-ov model za izradu korisnika, te daje formi za registraciju uloge koje će administrator dodijeliti određenim korisnicima. Asinkrona metoda *Register(CommonViewModel model, string role)* kao parametre prima objekt modela *CommonViewModel* i ulogu, a služi za spremanje korisnika i osobe u bazu podataka. U ovoj metodi se dodjeljuju korisničko ime, e-mail, lozinka i uloga korisniku te ime, prezime, uloga i korisničko ime (e-mail) osobi te ih upisuje u bazu nakon što je prikupio podatke, a ako dođe do nekih problema će ih prikazati. Osim kontrolera za račune (engl. *AccountController*) za upravljanje korisnikom je bitan i *ManageController* pomoću kojeg se radi promjena lozinke i s kojim se u budućnosti mogu dodati određeni sigurnosni mehanizmi. Asinkrona metoda *ChangePassword(ChangePasswordViewModel model)* kao parametar prima

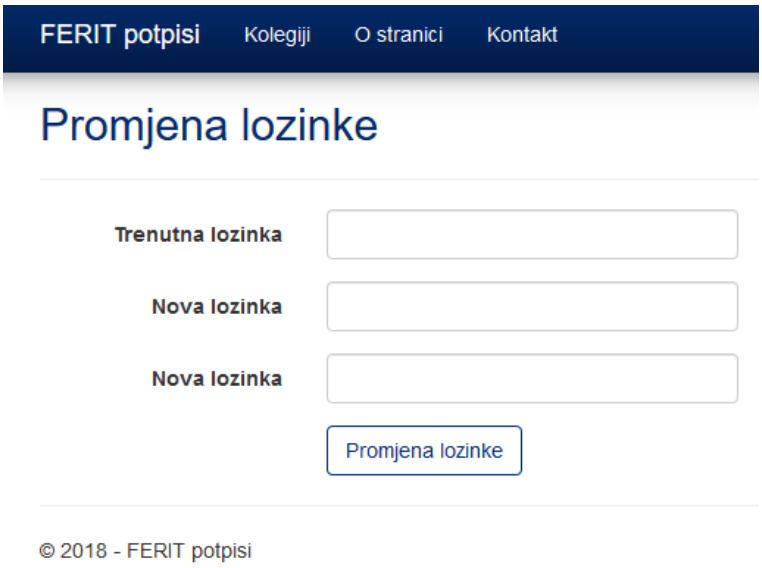
objekt klase *ChangePasswordViewModel* koji je dohvaćen iz forme i na osnovu stare lozinke se radi promjena na novu lozinku koju je potrebno potvrditi. Na slici 4.12. mogu se vidjeti pogledi s funkcionalnostima za upravljanje računima i korisnicima.



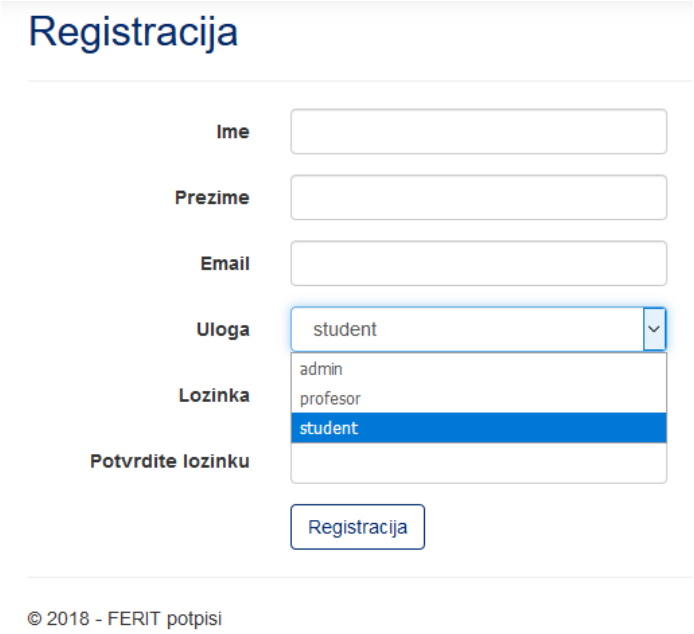
a) Registracija novog korisnika



b) Prijava postojećeg korisnika



c) Promjena postojeće lozinke



d) Prikaz uloga za korisnike

Slika 4.12. Pogledi vezani za upravljanje korisnicima

WEB aplikacija je objavljena na *Microsoft Azure* platformi koristeći studentski račun. Za objavu stranice na Internet potrebno je u *Azure* portalu napraviti novu WEB aplikaciju (engl. *App Service*) i bazu podataka na koju će se spremati (*SQL Server* i *SQL database*). Objava stranice obavljena je koristeći *Microsoft Visual Studio 2017*. pomoću profila za objavu koji je preuzet s *Microsoft Azure* portala. WEB aplikaciju je bilo potrebno objaviti kako bi se mogla provjeriti ispravnost POST zahtjeva kojeg šalje mobilna aplikacija. WEB aplikaciju je moguće isprobati na:

http://attendence.azurewebsites.net, gdje se moguće prijaviti kao administrator s slijedećim podacima:

- korisničko ime: admin@admin.com,
- lozinka: Superadmin.123,

ili kao profesor s slijedećim podacima:

- korisničko ime: knenadic@etfos.hr,
- lozinka: Ferit.123 .

5. ZAKLJUČAK

Upisivanje pohađanja studenata na predavanjima predstavlja problem zbog raznih faktora, a ovaj sustav rješava dio tih problema uvođenjem sigurnog i modernog načina za upis pomoću QR koda i ograničenja upisivanja. Opisani sustav sastoji se od dva dijela, mobilne i WEB aplikacije, koji rade zajedno kako bi riješili problematiku s upisom studenata na predavanja.

Prije same izrade aplikacije bilo je potrebno steći neka osnovna znanja o korištenju tehnologija kako bi se ta znanja mogla steći. Nakon toga je potrebno osmisliti arhitekturu sustava i baze podataka radi lakše izrade projekta.

Mobilna aplikacija daje mogućnost prijave studentima s danim podacima kako bi omogućili prijavu na predavanje. Kada je student prijavljen otvara se skener QR koda koji prilikom skeniranja šalje zahtjev WEB API-u koji obrađuje taj zahtjev. Kada je kod skeniran korisnik se automatski odjavljuje iz aplikacije i dobiva odgovarajuću poruku ako je upisan ili ako je nešto pošlo po krivu. WEB aplikacija prikazuje kolegije specifične za korisnika i taj korisnik ima mogućnost dodavanja, izmjene i brisanja kolegija te pregleda svih predavanja za svaki pojedini kolegij. Predavanja se također mogu dodavati, mijenjati i brisati te ovisno o statusu predavanja (je li aktivan ili ne) daju mogućnost ručnog upisa studenta (ako upis nije aktivan) ili upisa pomoću generiranog QR koda (ako je upis aktivan). WEB aplikaciji mogu pristupiti samo profesori i administratori, od kojih administratori imaju mogućnost dodavanja novih korisnika (studenata, profesora, administratora).

LITERATURA

[1] S. Smith, „What is the MVC pattern?“, Microsoft, 08.01.2018. https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-2,15.06.2018

[2] „Firebase Auth Quickstart“, Google, https://github.com/firebase/quickstart-android/tree/master/auth, 15.08.2018.

[3] L. Rafn, „UI Avatars“, https://ui-avatars.com/, 01.09.2018.

[4] „Scan QR Code by Camera“, EDMT Dev, 26.12.2016., https://www.youtube.com/watch?v=o69UqAKi47I, 1.09.2018.

[5] „ZXing ("Zebra Crossing") barcode scanning library for Java, Android“, https://github.com/zxing/zxing, 01.09.2018.

[6] M. Wasson, „Get Started with ASP.NET Web API 2 (C#)“, Microsoft, 28.11.2018, https://docs.microsoft.com/en-us/aspnet/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api, 02.09.2018.

[7] „Volley overview“, Google, 15.07.2018., https://developer.android.com/training/volley/, 03.09.2018

[8] R. Menoth, „QR Code Generator In ASP.NET Core Using ZXING.NET“, C# Corner, 12.05.2017., https://www.c-sharpcorner.com/article/qr-code-generator-in-asp-net-core-using-zxing-net/, 06.09.2018.

[9] Github, https://github.com/, 08.09.2018.

[10] StackOverfolw, https://stackoverflow.com/, 08.09.2018.

SAŽETAK

Sustav vođenja evidencije pohađanja predavanja

Glavni zadatak diplomskog rada je riješiti problematiku upisivanja studenata na predavanja. U radu je dana teorijska osnova korištenih tehnologija. Izrađena je mobilna aplikacija koju će koristiti studenti te WEB aplikacija za profesore koja obrađuje zahtjeve studenata i daje prikaz svih studenata po predavanjima.

Ključne riječi: pohađanje nastave, Android, ASP.NET MVC, QR kod, Firebase, Microsoft Azure

ABSTRACT

Title: Lecture attendance records system

The main task of this graduate thesis is to solve the issue of enrolling students in lectures. This paper consists of theoretical background for used technologies. Mobile application which helps students to be kept on record in lectures is created and WEB application which processes requests made by students and gives overview of all student sin said lecture is created for professors.

Keywords: lecture attendance, Android**,** ASP.NET MVC, QR code, Firebase, Microsoft Azure

ŽIVOTOPIS

Vladimir Anić rođen je 09.03.1995. godine u Osijeku. Od rođenja živi u Belišću gdje stječe osnovnoškolsko obrazovanje u Osnovnoj školi Ivana Kukuljevića Belišće. Godine 2009. upisao se u Srednju školu Valpovo smjer Elektrotehnika. Godine 2013. završava srednju školu i državnu maturu, te upisuje Elektrotehnički fakultet, preddiplomski studij računarstva, u Osijeku. Godine 2016 sudjeluje na Elektrijadi u Riminiju iz kolegija Engleski jezik gdje ekipno postiže drugo mjesto, no zbog tehničke prirode poništen je taj dio natjecanja. Preddiplomski studij računarstva završava 2016. godine i time postiže titulu sveučilišni prvostupnik računarstva. Iste godine upisuje diplomski studij računarstva – smjer programsko inženjerstvo. Godine 2017. pohađa i završava praksu na mjestu softver developera u tvrtki Mono.

PRILOZI

Projekt ASP.NET MVC WEB aplikacije, Android studio projekt mobilne aplikacije