Društvena mreža za suradnju i dijeljenje sadržaja

Klen, Ivan

Undergraduate thesis / Završni rad

2017

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:200:935406

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

zir.nsk.hr

Image not found or type unknown

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Stručni studij

DRUŠTVENA MREŽA ZA SURADNJU I DIJELJENJE SADRŽAJA

Završni rad

Ivan Klen

Osijek, 12. prosinac 2017.

SADRŽAJ

1. UVOD 1
1.1. Zadatak završnog rada 1
2. PREGLED UPOTREBLJENIH TEHNOLOGIJA 2
2.1. C# programski jezik i razvojno okruženje 2
2.2. Pogodnosti korištenja Entity Framework-a (EF) naspram direktnog SQL-a 3
2.3. Inverzija kontrole i Ninject modul 4
2.4. Sustav za kontrolu verzije 4
2.5. AutoMapper 5
3. RAZVOJ APLIKACIJE DRUŠTVENE MREŽE 6
3.1. Korisnički zahtjevi 6
3.2. Struktura projekta 7
3.3. Sloj pristupa bazi podataka (engl. Data Access Layer) 9
3.4. Modeli 11
3.5. Skladišta 17
3.6. Upravljači (engl. Controllers) 20
3.7. Klijentsko rješenje dijela aplikacije 27
4. ZAKLJUČAK 43
LITERATURA 44
SAŽETAK 45
SOCIAL NETWORK FOR COLLABORATION AND CONTENT SHARING 46
ŽIVOTOPIS 47

1. UVOD

Unatoč povećem broju društvenih mreža i kontinuiranom brojčanom rastu istih, potreba za usko specijaliziranim društvenim mrežama s mogućnošću prilagodbe je i dalje velika. Između ostalog, potrebno je zadovoljiti i korisnikove potrebe, odnosno dati im određeni dio kontrole i slobode pri upotrebljavanju takve društvene mreže, ali pritom zadovoljiti sigurnosne standarde i pokriti što je više moguće platformi. Upravo se na toj ideji temelji ovaj projekt koji će u konačnici olakšavati suradnju i komunikaciju korisnika na raznim manjim projektima.

Glavni cilj završnog rada je razviti samostalnu i funkcionalnu internet aplikaciju koja će zadovoljavati današnje standarde i korisničke zahtjeve. Prilikom izrade se koriste znanja stečena tokom studiranja ponajviše iz područja baza podataka te programiranja.

Sam dokument se sastoji od dva glavna dijela koja čine : teorijski i praktični dio. Teorijski dio se sastoji od navođenja i pojašnjenja pojedinih izbora tehnologija koje su korištene prilikom izrade same internet aplikacije, te zašto i na koji način je određena implementacija te tehnologije u samoj aplikaciji napravljena.

1.1. Zadatak završnog rada

Napraviti web aplikaciju za društvenu mrežu koja će se koristiti za suradnju na manjim projektima. Osobe mogu imati svoj profil i pripadati određenim grupama. Na projekt se mogu dodavati pojedinci i grupe, sadržaj projekta može biti javan ili podijeljen samo sa pojedincima i grupama. (Opis teme završnog rada)

2. PREGLED UPOTREBLJENIH TEHNOLOGIJA

2.1. C# programski jezik i razvojno okruženje

Unatoč popularnosti programskih jezika kao što su Javascript i PHP za izradu web aplikacija, C# programski jezik nudi opsežniji repertoar mogućnosti, od kojih su npr. strogo definirani objekti, zatim sigurnost tipa prilikom kreiranja objekta itd. .

C# je moderan, objektno-orijentirani jezik koji omogućava jednostavno, sigurno i brzo razvijanje aplikacije na virtualno bilo kojoj platformi. Nadalje, otkrivanje pogrešaka prilikom prevođenja (*engl. Compile)* same aplikacije pokazalo se kao velika prednost nad skriptnim programskim jezicima kao što je Javascript (iako i za Javascript postoje razni alati za provjeravanje grešaka (*engl. linter*), ali ih je podosta teže namjestiti i upotrebljavati).

Nužno je spomenuti i alat koji je korišten radi prevođenja C# kôda pri razvoju aplikacije, a radi se o Microsoft Visual Studio-u (MVS). Omogućava brzo, strukturirano pisanje kôda, pojednostavljeno traženje pogrešaka i dijagnosticiranje, testiranje i sadrži odlično grafičko sučelje koje je naravno, prilagodljivo vlastitim željama i potrebama (veličina slova, boja, itd.).

Dodaci koji su korišteni u sklopu razvojnog okruženja uvelike olakšavaju razvoj i čine ga relativno ugodnim procesom. Neki od dodataka čija će uloga kasnije biti detaljnije objašnjena su: *Entity Framework* (*EF*), *Ninject* (*Dependency Injection, Inversion of Control concept*) itd. Nadalje, veliku ulogu u održavanju i skalabilnosti same aplikacije imaju i uzorci (*engl. pattern*). Jedan od uzoraka koji je korišten je tzv. uzorak skladišta (*engl. Repository Pattern*). Takav uzorak omogućava unošenje određenog nivoa apstrakcije prilikom razvoja aplikacije, te ju čini vrlo prilagodljivom s obzirom na brzu promjenu modernih standarda razvoja aplikacija.

Prethodno navedene stavke služe za izradu serverskog rješenja (*engl. back-end)*, odnosno rukovanje s podacima baze podataka, briga o sigurnosti i brzini, pravovremena isporuka podataka, retransmisija kod neuspjelog pokušaja prijenosa itd. .

Za izradu klijentskog rješenja (*engl. front-end)*, korišten je Microsoftov licencirani okvir (*engl. framework)* C# MVC (*engl. Model-View-Controller*). Ukratko, radi se o arhitekturalnom uzorku koji dijeli klijentsko rješenje na tri dijela : model, pogled (*engl. view)*, upravljač (*engl. controller)*. U suštini ga je lako integrirati s bilo kojim postojećim rješenjem, a sadrži i više nego prijeko potrebna svojstva za održivo, lagano (*engl. lightweight),* neovisno o platformama, klijentsko rješenje.

2.2. Pogodnosti korištenja Entity Framework-a (EF) naspram direktnog SQL-a

Entity framework samo je jedan od objektno relacijskih mapera (*engl. O/RM – Object Relational Mapper)*, no korišten je zbog izuzetne podrške programerske zajednice i odlično koncipirane dokumentacije. Služi za učinkovito upravljanje entitetima baze podataka, odnosno upotpunjuje svojstvo strogo definiranih tipova podataka programskog jezika C#. Kasnije kod implementacije će se vidjeti razlozi zbog kojih je korišten ovaj objektno relacijski maper.

Direktni SQL se najčešće koristi kod manjih aplikacija, gdje se ne očekuje nagli porast korisnika ili promjena implementacije. U sljedećoj tablici se mogu vidjeti mane i vrline pojedinih tehnologija.

Entity Framework (EF)	Direktni SQL
Objektno-orijentirani principi rukovanjem s entitetima baze podataka	Pisanje dugačkih SQL-upita (*Insert, Delete..),* daleko veća mogućnost pogreške
Neznatno sporiji u odnosu na proceduralni SQL zbog složenosti (kreiranje objekta, briga o životnom vijeku objekta, itd.)	Dokumentacija je podosta opširnija, zajednica koja održava i unapređuje je veća
Nedvosmislena upotreba, koncizni i jasni upiti prema bazi podataka.	Jednostavnija implementacija
Jednostavna instalacija putem paket menadžera Visual Studio-a	Podložniji sigurnosnim propustima

Slika 2.2.1. *Tablica mana i vrlina tehnologija Entity Framework i direktnog SQL-a.*

Uz ovaj objektno relacijski maper dolaze i nekolicina pristupa za uspostavljanjem/manipulacijom tablica neke baze podataka. Primjerice, za kreiranje baze podataka ove aplikacije korišten je pristup kreiranja tablica pomoću klasa i svojstava (*engl. Code first approach*). Time se osigurava dinamičnost i ažurnost baze podataka kod svake promjene baze, no treba pripaziti da se dodavanjem raznih svojstava ne bi obrisala te nanovo kreirala baza

podataka. Nadalje, kod tog pristupa moguće je i popuniti bazu podataka s lažnim podacima radi testiranja funkcionalnosti (*engl. Initial seed*).

2.3. Inverzija kontrole i Ninject modul

U C# programskom jeziku se kod parametarskog konstruktora mora paziti kojeg je tipa argument. Tu je dakako i životni ciklus novonastalog objekta, na kojeg može ili ne mora prevodilac paziti.

Inverzijom kontrole (*engl. Inversion Of Control*) programera se vrlo lako može lišiti te odgovornosti, odnosno problem zavisnosti tipa novonastalog objekta se stavlja na jedno mjesto i njegov životni ciklus se vrlo lako prati. Odnosno predaje se kontrola programeru pomoću sučelja (*engl. interface)* i modula (u ovom slučaju Ninject) kojom on može vrlo lako izabrati više podtipova (kod nasljeđivanja (*engl. Inheritance*)) nekog tipa kojeg će predati kao argument bez suvišnih linija kôda.

Postoji nekoliko različitih okvira (*engl. framework)* inverzije kontrole od kojih su : Ninject, Unity, Castle Windsor i dr. . Okvir Ninject je korišten zbog jednostavnosti implementacije u trenutnom razvojnom okruženju, te dobre dokumentacije.

2.4. Sustav za kontrolu verzije

Uzmimo u obzir scenarij da se nakon implementacije nove funkcionalnosti većina korisnika požali zbog izuzetne sporosti odaziva nekih zahtjeva. Kako i na koji način brzo i sigurno vratiti aplikaciju u prethodno stanje (*engl. rollback)*? Odgovor na to pitanje je sustav za kontrolu verziju (*engl. Version Control System)*.

Sustav za kontrolu verzije je i više od toga, moguće je pojedinačne podatke vratiti u prethodno stanje. Nadalje, moguć je rad na više ogranaka (*engl. branches)* od kojih bi jedna grana bila proizvodna – gotova (*engl. production branch*), a druga razvojna – testna (*engl. test branch*).

Pri samom razvoju aplikacije korišten je *Git* sustav za kontrolu verzije, zato jer je besplatan (*engl. open source*) i jer ga integrirano razvojno okruženje kao što je Microsoft Visual Studio u potpunosti podržava (sadrži detaljno sučelje za kontrolu verzije).

2.5. AutoMapper

Automapper je alat koji izvrsno komplementira prethodno spomenutom O/RM-u, Entity Framework-u. Kako Entity Framework služi za lakšom manipulacijom entiteta baze podataka, tako AutoMapper služi za jednostavniju manipulaciju prepisivanja svojstva entiteta kao objekta, odnosno modela na drugi objekt, koji može biti složeniji ili jednostavniji, za svrhe apstrakcije.

Primjeri, entitet grupa u bazi podataka ne sadrži dovoljno informacija koje bi htjeli prezentirati korisnicima, stoga je potrebno kreirati model s dodatnim informacijama i zatim ga popuniti koristeći poveći broj linija koda. No, AutoMapperom se taj „nered“ i slučajne pogreške mogu izbjeći.

Dokumentacija i podrška ovom alatu je zaista enormna, potrebno ga je samo instalirati preko paket menadžera te zatim pozvati datoteku prilikom korištenja (praktički dolazi spreman *out of the box*), no za kompleksnija mapiranja potrebno je vladati API-em (*engl. Application Programming Interface*) ovog alata kako bi se upotrijebila sva njegova moć.

3. RAZVOJ APLIKACIJE DRUŠTVENE MREŽE

3.1. Korisnički zahtjevi

Vrlo je važno u početku jasno definirati korisničke zahtjeve, najprije kako bi korisnik dobio onakav proizvod kakav želi, a programeri jasniju predodžbu kakav bi taj proizvod trebao izgledati. Zahtjevi ujedno služe kao i smjernica ali i zaštita od zlouporabe reklamacije na gotov proizvod.

S obzirom na da se radi o web aplikaciji za društvenu mrežu, lako je zaključiti da je potreban sustav za prijavu i odjavu. Nadalje, treba implementirati i mogućnost povratka računa ukoliko korisnik zaboravi lozinku, ipak su korisnički računi privatna stvar, robusna i kvalitetna implementacija osigurava zaštitu korisnika.

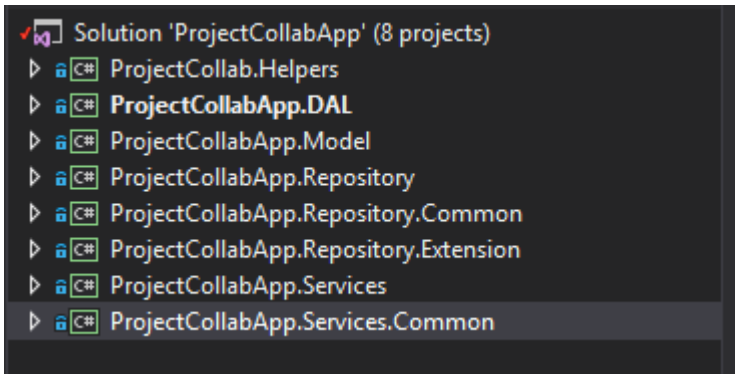
Mogućnost kreiranja grupa, dodavanja korisnika u vlastitu grupu ili pridruživanje tuđim grupama. Iz praktičnih razloga je napravljeno tako da jedan korisnik može biti član isključivo jedne grupe, jer to je stavka koja čini projekt potencijalno složenijim, odnosno skupljim. No bez obzira na kompleksnost te implementacije, modularnost komponenata projekta omogućava vrlo laganu integraciju nakon razvitka te funkcionalnosti.

Jedna od bitnijih stavaka ovog pod poglavlja su uloge (*engl. roles*). Prilikom kreiranja grupe korisnik koji ju je kreirao automatski dobiva ulogu administratora, a to znači da može mijenjati ime grupe, opis grupe, izbacivati i pozivati članove.

Vezano uz privatnost korisnika odnosno korisničkog profila i računa, implementirana je i mogućnost skrivanja sadržaja grupe odnosno projekta od korisnika koji nisu članovi grupe ili nisu korisnici web aplikacije, samim time je jasno definirano da svaki član grupe ima mogućnost dijeljenja podataka s ostalim sudionicima grupe, a tu mogućnost administrator grupe može onesposobiti. Iz logističkih i financijskih razloga, opseg podataka (*engl. bandwidth*) je ograničen. Idealno bi bilo sprovesti marketinšku strategiju pretplatničkih paketa različitih razina koje bi omogućavale grupama koje su pretplaćene korištenje većeg *bandwidth-a*.

3.2. Struktura projekta

Struktura projekta sadrži nekoliko klasnih datoteka (*engl. Class library*), iz razloga što je se puno lakše snaći u modulariziranoj projektnoj strukturi negoli da je sve stavljeno u jednu klasnu datoteku. Svaka od tih datoteka ima svoju namjenu i pojačava nivo apstrakcije, a u sljedećoj slici se jasno može vidjeti ta struktura te nazivi pojedinih datoteka.



Sl. 3.1. *Struktura projekta s nazivima klasnih datoteka*

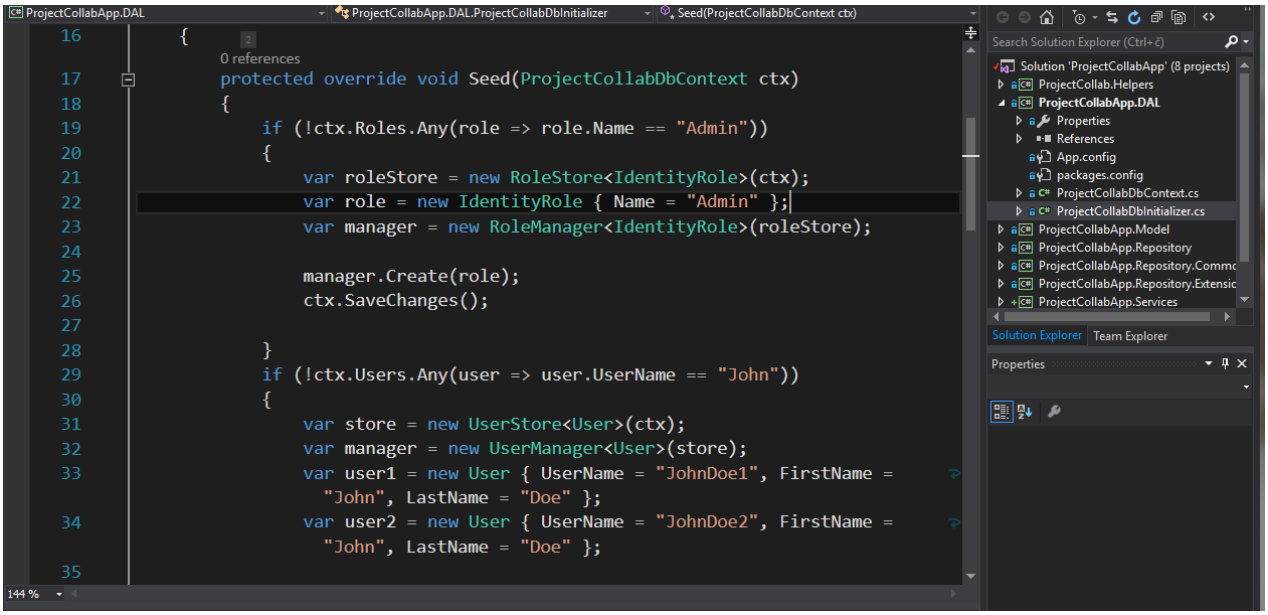
Naziv klasne datoteke	Opis klasne datoteke
ProjectCollab.Helpers	Sadrži klase s pomoćnim metodama, odnosno klasne metode koje omogućavaju proširenje ostalih metoda (pr. pretvorba podatka u bitove)
ProjectCollabApp.DAL (*DAL – engl. Data Access Layer*)	Ova klasna datoteka sadrži klase koje imaju ulogu u pristupu bazi podataka i manipulaciji tablica
ProjectCollabApp.Model	Klasna datoteka koja sadrži klase sa svojstvima (*engl. properties*) prema kojima se generira kompletna baza podataka
ProjectCollabApp.Repository	Glavna uloga je razdvajanje logike dohvaćanja podatka te mapiranja tog podatka na model.
ProjectCollabApp.Services	
ProjectCollabApp.Repository.Extensions	Slično kao i kod *Helpers* klasne datoteke, ali služi isključivo za za proširivanje klasa klasne datoteke *Repository*
ProjectCollabApp.*.Common	Klasne datoteke koje sadrže sučelja s ulogom nacrta prema kojima se klase ostalih klasnih datoteka moraju definirati (imena metoda, povratni tipovi metoda, broj parametara, itd..)

Sl. 3.2. *Opis klasnih datoteka*

3.3. Sloj pristupa bazi podataka (*engl. Data Access Layer*)

Ova klasna datoteka je sloj aplikacije koji je odgovoran za pristup bazi podataka. U klasi se jasno može vidjeti metoda *Seed* (*hrv. posijati*), koja se prilikom generiranja baze podataka poziva te popunjava kreirane tablice podacima, u ovom slučaju korisnicima s korisničkim imenima *JohnDoe1* i *JohnDoe2*. Svrha ove klase je ponajviše radi početnog testiranja raznih planiranih funkcionalnosti same aplikacije.

Isto tako je prilikom generiranja baze definirana početna uloga, a ta je uloga *Admin* (*engl. administrator*), koji ima jasno definirane ovlasti u prethodnom poglavlju korisničkih zahtjeva.

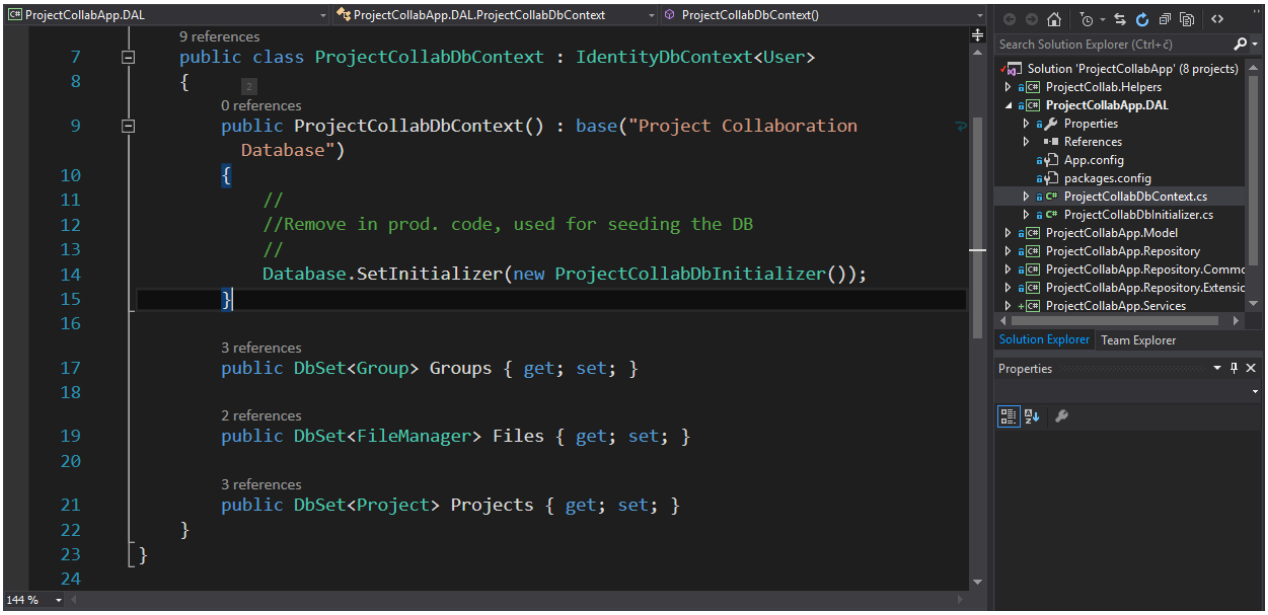


Sl. 3.3. *Kod klase ProjectCollabDbInitializer.cs*

Nadalje, klasa koja je primarna za interakciju s podacima kao objektima je *ProjectCollabDbContext.cs* klasa. Razlog je taj što prethodno spomenuti *Entity Framework* koristi ovu klasu kao most za interakciju između baze podataka i entiteta kao objekta.

Primjerice, ona sadrži takozvani set entiteta (*engl. Entity set*), a oni su sljedeći : *Groups, Files, Projects*, te se oni kao entiteti mapiraju na tablice baze podataka prilikom inicijalizacije baze podataka. Doduše tipovi unutar *DbSet<type>* kao što su *Group, FileManager* i *Project* su zapravo klase koje služe kao model sa svojstvima prema kojima se definiraju polja pripadajućih tablica.

Korisnički *DbSet* nije naveden u ovoj klasi jer model klase *User* nasljeđuje klasu *IdentityUser* (predstavlja zadanu implementaciju korisnika sa svim pripadnim svojstvima kao što su : korisničko ime, lozinka, e-mail, itd..) koja dolazi u paketu s *Entity Framework*-om.

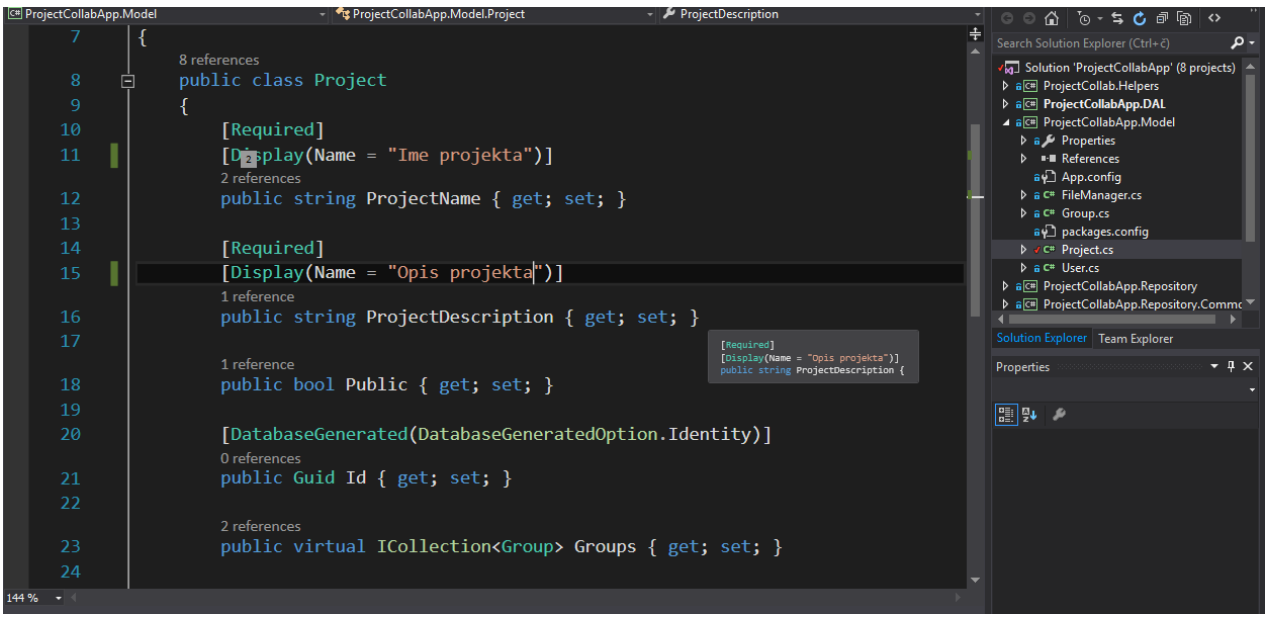


Sl. 3.4. *Klasa ProjectCollabDbContext.cs – most između baze podataka i entiteta kao objekta*

3.4. Modeli

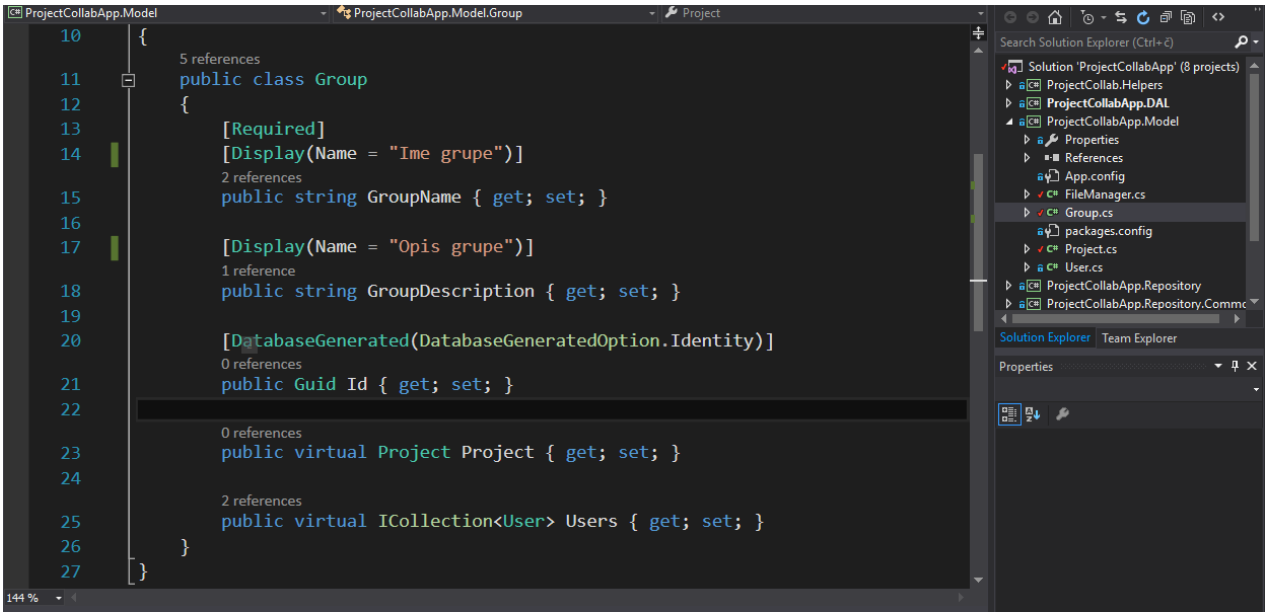
Unutar klasne datoteke *Model* nalaze se klase sa svojstvima prema kojima se definiraju polja tablica. U sljedećoj slici možemo jasno vidjeti dio svojstava klase *Project*. Oznake poput *[Required]* i *[Display]* su dio klase *DataAnnotations*, a služe najčešće za validaciju (ukoliko korisnik prilikom unošenja ostavi to polje prazno, dobiva grešku kao povratnu informaciju prilikom pokušaja spremanja tog projekta).

Jedna od bitnijih stavaka klase *DataAnnotations* je *[DatabaseGenerated (DatabaseGeneratedOption.Identity)]*, a služi za generiranje nasumičnog jedinstvenog globalnog identifikatora (*engl. Globally Unique Identifier*). Time se osigurava da se u bazi, posebice u tablici *Project* ne mogu naći dva projekta s istim globalnim identifikatorom. Isto tako, s obzirom da taj identifikator sadrži nasumični broj malih, velikih slova i brojeva za razliku od običnog broja (*engl. integer*), je sigurnost veća i mogućnost pogreške manja. No, takav identifikator ima svoju cijenu, a ona je da ga je skuplje implementirati u smislu zauzimanja prostora u samoj bazi podataka, ipak, prednosti korištenja takvog identifikatora su daleko veće.



Sl. 3.5. *Klasa Project.cs – model sa pripadajućim svojstvima*

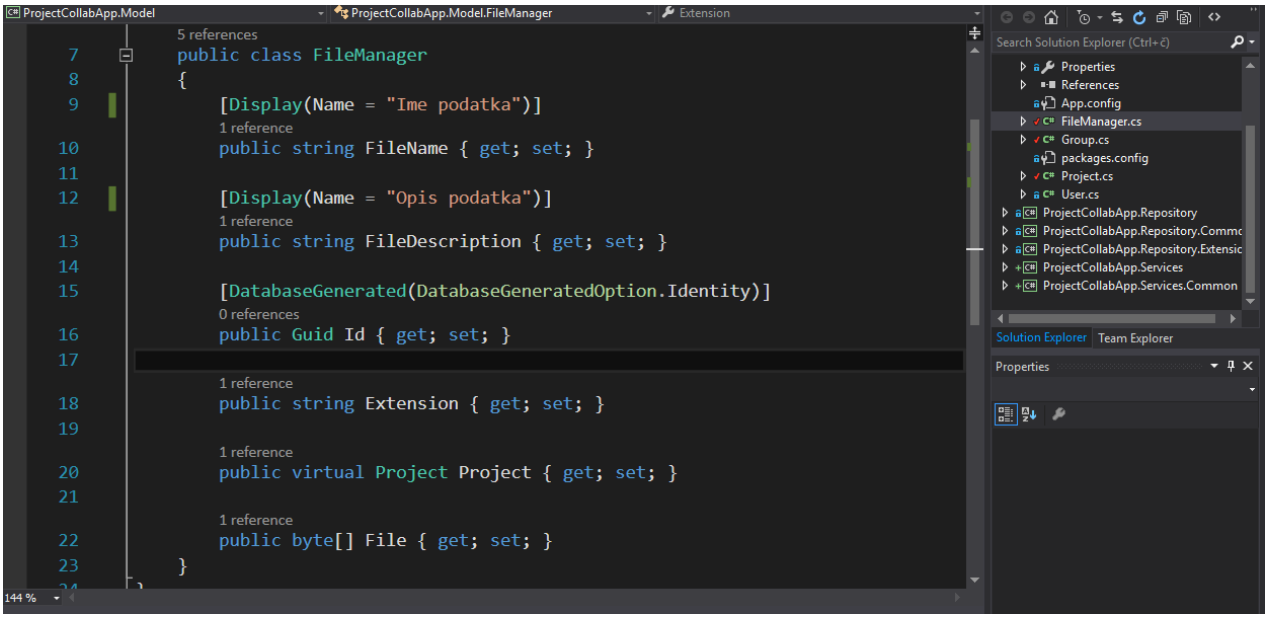
Između prethodne (Sl. 3.5.) i sljedeće slike (Sl. 3.6.) modela klase *Group* može se vidjeti sličnost svojstava, posebice je zanimljivo svojstvo *ICollection<User> Users*. To svojstvo označava da takav model u tablici može i ne mora sadržavati neodređeni broj korisnika, isto tako svaki korisnik u tablici ima definiran strani ključ (*engl. Foreign key*) koji sadrži jedinstveni identifikator grupe u kojoj se nalazi (ili je polje jednostavno prazno ukoliko se korisnik ne nalazi ni u jednoj grupi).



Sl. 3.6.. *Klasa Group.cs – model sa pripadajućim svojstvima*

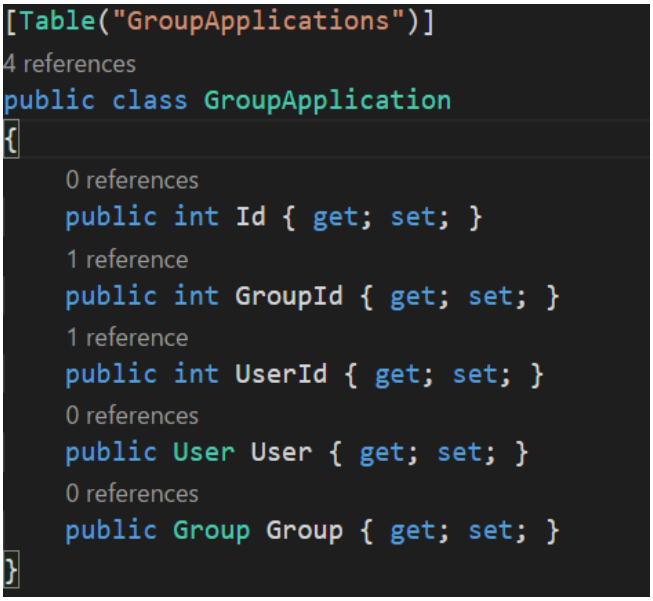
Prethodno opisano svojstvo je zapravo najveća prednost korištenja *Entity Framework*-a, jer taj dodatak automatski dodaje strane ključeve na osnovu tipa podatka svojstva unutar neke klase. Isto tako ima mogućnost prepoznavanja glavnih ključeva (ime svojstva mora sadržavati *Id* u svom nazivu).

Na sljedećoj slici se nalazi model klase *FileManager* sadrži svojstvo *File* tipa *byte[ ]*, u prijevodu, to polje je zapravo polje bajtova, jer je to jedan od najboljih načina spremanja podatka u bazu podataka. No to zahtjeva pomoćne metode koje će podatak prilikom postavljanja konvertirati u polje bajtova te spremiti u bazu. Isto to vrijedi prilikom preuzimanja nekog podatka, te je iz tog razloga dodano svojstvo *Extension* koji će sadržavati ekstenziju spremljenog podatka u bazi, te će ga pravilno moći konvertirati prilikom preuzimanja.



Sl. 3.7. *Klasa FileManager.cs – klasa koja služi, osim kao model, kao upravitelj spremanja i preuzimanja datoteka iz baze podataka*

Slika 3.8. predstavlja model grupnih aplikacija, odnosno model koji sadrži korisnika i grupu za koju se određeni korisnik prijavljuje. Važno je napomenuti da je model spremljen kao tablica u bazu podataka sa kompozitnim ključem, tako su šanse za kreiranjem nepravilne prijave za grupu ravne nuli. Vlasnik grupe je definiran u klasi *Group* te on može pomoću ključa *GroupId* vidjeti svaku prijavu za grupu koja se odnosi na njegovu grupu.

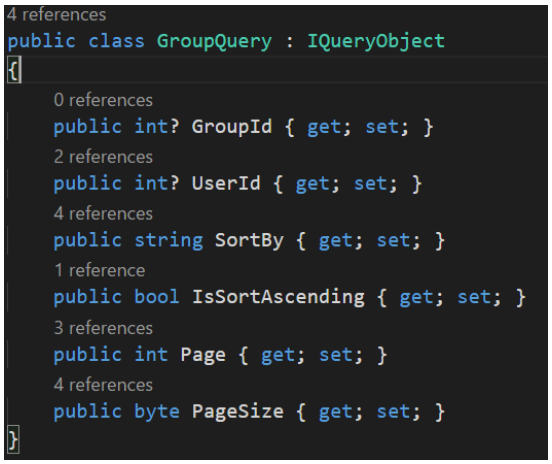


Sl. 3.8. *Klasa GroupApplication.cs – model koji definira prijavu korisnika u određenu grupu, sadrži takozvani kompozitni ključ (engl. composite key)*

Jedan od glavnih problema aplikacija koje imaju veći broj korisnika i mogućnosti je taj što je dohvaćanje svih podataka iz baze podataka vremenski skup proces, ne toliko za server i bazu podataka koliko za korisnika. Primjerice, korisnik ne bi ni osjetio dohvaćanje tridesetak grupa ili podataka iz baze podataka, no što da ih ima preko nekoliko tisuća?

Jedno, ako ne i jedino rješenje je straničenje (*engl. paging*) podataka na strani servera. Tim rješenjem se osigurava bolji korisnički doživljaj, jer se dohvaća i prikazuje određeni broj podataka, dok korisnik ima percepciju da su svi podaci učitani, u ovom slučaju grupe.

Klasa implementira sučelje *IQueryObject*, koji služi kao kalup za definiranje klasa odnosno podataka koje želimo straničiti i zatim servirati korisniku.



Sl. 3.9. *Klasa GroupQuery.cs – model korišten za sortiranje i straničenje grupa na strani servera*

Sljedeća klasa prikazuje vrhunac objektno orijentiranog jezika, što je vidljivo u samoj definiciji klase, naime <*T*> dodatak u nazivu klase označava da se radi o općoj klasi (*engl. generic*). Što znači da prilikom kreiranja ove klase umjesto parametra <*T*> moramo koristiti konkretnu klasu, a omogućava nam da u svojstvo *Items* spremamo tip podatka koji smo koristili prilikom kreiranja ove klase.

Ova klasa služi, ukratko, da ne dolazi do nepotrebnog ponavljanja koda. Time se poštuje jedan od glavnih koncepata programiranja, a taj je da, ukoliko se isti kod pojavljuje na više mjesta, potrebno ga je prenamijeniti za višekratnu upotrebu.

Sljedeća slika (sl. 3.11.) je slika statične funkcije koju svi tipovi *IQueryable*<*T*> mogu koristiti, a upravo ona se odnosi na straničenje podataka, stoga je vrlo praktično implementirati ju kao proširenje.

```
5 references
public class QueryResult<T>
{
    1 reference
    public int TotalItems { get; set; }
    1 reference
    public IEnumerable<T> Items { get; set; }
}
```

Sl. 3.10. *Klasa QueryResult.cs – opća klasa koja koristi svojstva za spremanje izlistanih podataka*

```
public static IQueryable<T> ApplyPaging<T>(this IQueryable<T> query, IQueryObject
{
    if (queryObject.Page <= 0)
        queryObject.Page = 1;

    if (queryObject.PageSize <= 0)
        queryObject.PageSize = 10;

    return query.Skip((queryObject.Page - 1) * queryObject.PageSize).Take(queryOb
}
```

Sl. 3.11. *Metoda ApplyPaging– metoda koja proširuje osnovne mogućnosti tipa IQueryable s općom namjenom*

3.5. Skladišta

U dijelu s objašnjenjima pojedinih tehnologija koje su korištene objašnjeno je što su i za što služe skladišta, stoga ovaj dio služi za podrobniji opis pojedinog skladišta.

Slika 3.12. predstavlja metodu unutar skladišta grupa koja se poziva s jednim argumentom kad se dohvaćaju grupe. Parametar tipa *GroupQuery* sadrži svojstva koja ova metoda koristi kako bi se izvršilo straničenje na serveru, te se rezultati spremaju u varijablu pozivatelja ove metode. Važno je napomenuti da je korišten *await* operator u više navrata, ukratko, on služi za asinkroni način dohvaćanja podataka u ovom slučaju. Primjerice, korisnik ne mora zuriti u zaslon čekajući kada će mu se pojaviti podaci na zaslonu, već može raditi nešto drugo dok se podaci ne učitaju.

```
public async Task<QueryResult<Group>> GetAllGroupsAsync(GroupQuery queryObject)
{
    var result = new QueryResult<Group>();
    var query = context.Groups.Include(v => v.GroupUsers).AsQueryable();

    if (queryObject.UserId.HasValue)
        query = query.Where(v => v.Id == queryObject.UserId.Value);

    var columnsMap = new Dictionary<string, Expression<Func<Group, object>>>
    {
        ["id"] = g => g.Id,
        ["name"] = g => g.Name,
        ["memberCount"] = v => v.GroupUsers.Count()
    };

    query = query.ApplyOrdering(queryObject, columnsMap);

    result.TotalItems = await query.CountAsync();

    query = query.ApplyPaging(queryObject);

    result.Items = await query.ToListAsync();

    return result;
```

Sl. 3.12. *Metoda GetAllGroupsAsync– metoda koja se koristi za asinkrono dohvaćanje svih grupa na temelju argumenta s kojom se metoda poziva*

Nadalje, sljedeća slika pokazuje metodu koja se koristi u slične svrhe kao i prethodna. Dohvaća se isključivo jedna grupa pomoću njenog jedinstvenog ključa, što bi bio parametar *id*, dok parametar *includeRelated*, koji je neobavezan (u tom slučaju je uvijek istinit), služi za učitavanje grupe korisnika i podataka koje spadaju pod tu grupu. Iako metoda može funkcionirati i bez tog parametra, koristi se u svrhe kada nije potrebno učitavanje grupe korisnika ili podataka koji su vezani za tu grupu. Na taj način se može znatnije uštediti na prethodno spomenutoj skupoći dohvaćanja podataka.

```
6 references
public async Task<Group> GetGroupAsync(int id, bool includeRelated = true)
{
    if (includeRelated)
    {
        return await context.Groups.Include(g => g.GroupUsers)
        .Include(f => f.Files)
        .SingleOrDefaultAsync(g => g.Id == id);
    }

    return await context.Groups.FindAsync(id);
}
```

Sl. 3.13. *Metoda GetGroupAsync - metoda koja se koristi za asinkrono dohvaćanje određene grupe na temelju ključa*

```
public void AddGroup(Group group)
{
    this.context.Add(group);
}


public void RemoveGroup(Group group)
{
    this.context.Remove(group);
}
```

Sl. 3.14. *Metoda AddGroup i RemoveGroup – metode koje se koriste za brisanje i spremanje grupe*

Usporedno s skladištem grupa, također postoji i skladište podataka, te su neke od glavnih metoda prikazane u sljedećoj slici.

Metoda *GetFileAsync* s parametrom jedinstvenog ključa služi za dohvaćanje jednog jedinog podatka za daljnju manipulaciju, dok metoda *GetFilesAsync* s jedinstvenim parametrom grupe služi za dohvaćanje svih podataka grupe čiji se ključ upotrijebi kao argument.

```
5 references
private readonly ProjectCollabDbContext context;
0 references
public FileRepository(ProjectCollabDbContext context)
{
    this.context = context;
}


public void AddFile(File file)
{
    this.context.Add(file);
}

3 references
public async Task<File> GetFileAsync(int fileId)
{
    return await this.context.Files.FindAsync(fileId);
}

1 reference
public async Task<IEnumerable<File>> GetFilesAsync(int projectId)
{
    return await context.Files.Where(f => f.ProjectId == projectId).ToListAsync();
}

1 reference
public void RemoveFile(File file)
{
    this.context.Files.Remove(file);
}
```

Sl. 3.15. *Metode skladišta podataka– metode koje se koriste za brisanje, spremanje i dohvaćanje podataka*

3.6. Upravljači (engl. *Controllers*)

S obzirom da je nužno imati i klijentsko rješenje kako bi korisnik lakše koristio aplikaciju, korišteni su upravljači koji izlažu serversko rješenje (bazu podataka, validacije, spremanje podataka itd..) ostatku svijeta.

Način na koji oni funkcioniraju je sljedeći: recimo da korisnik klikne na formu za spremanje novog podatka u postojeći projekt. Šalje se zahtjev serveru gdje se provjerava postoji li uopće taj projekt i odgovara li veličina podatka koju serverska strana može zaprimiti. Ukoliko sve validacije prođu, podatak se sprema u bazu podataka kao veza na fizičku lokaciju na serveru (u ovom slučaju lokalno računalo).

Sve to omogućava API (*engl. Application Programming Interface*) što su zapravo krajnje točke na koje korisnik šalje zahtjeve pomoću razno raznih formi ili tipaka koje su upravo definirane na upravljačima atributom *Route*. No što ako ti putevi nisu zadani? Onda se koristi ime upravljača kao zadani (engl. *default*) put.

Na sljedećoj slici se vidi upravljač podataka pod nazivom *FileController*, s točno definiranim putevima za rad s krajnjim točkama.

```
[Route("/api/projects/{projectId}/files")]
0 references
public class FileController : Controller
{
    5 references
    private readonly IMapper mapper;
    3 references
    private readonly FileSettings fileSettings;
    4 references
    private readonly IUnitOfWork unitOfWork;
    6 references
    private readonly IFileRepository fileRepository;
    1 reference
    private readonly IGroupRepository groupRepository;
    2 references
    private readonly IProjectRepository projectRepository;
    2 references
    private readonly IHostingEnvironment host;
    0 references
    public FileController(IMapper mapper, IOptionsSnapshot<FileSettings> options, IUnit
                          IFileRepository fileRepository, IGroupRepository groupReposit
    {
        this.projectRepository = projectRepository;
        this.groupRepository = groupRepository;
        this.fileRepository = fileRepository;
        this.fileSettings = options.Value;
        this.unitOfWork = unitOfWork;
        this.mapper = mapper;
        this.host = host;
    }
```

Sl. 3.16. *Upravljač podataka koji koristi krajnje točke ili put za pristup podacima*

Na prethodnoj slici se jasno vidi atribut *Route* kao pristupna točka, no važno je napomenuti da se u konstruktoru ovog upravljača koriste razna sučelja (u praksi se sučelja definiraju s prefiksom 'I' uz naziv tipa podatka) umjesto konkretnih klasa. Razlog tome je što korištenjem ovakvog pristupa vrlo lako izmijeniti bilo koju od navedenih klasa s novom, bez poništavanja ugovora koja sučelja snažno primjenjuju na klase koje ih implementiraju.

U nastavku se vidi detaljnije implementacija samih pristupnih točaka odnosno funkcionalnost. Primjerice metoda pod nazivom *UploadFileAsync* koja mora primiti 2 argumenta, od kojih je jedan jedinstveni ključ projekta, a drugi podatak koju korisnik kroz formu pošalje.

```
[HttpGet]
0 references
public async Task<IEnumerable<FileResource>> GetFiles(int projectId)
{
    var files = await fileRepository.GetFilesAsync(projectId);

    return mapper.Map<IEnumerable<Core.Models.File>, IEnumerable<FileResource>>(fil
}

[HttpPost]
0 references
public async Task<IActionResult> UploadFileAsync(int projectId, IFormFile file)
{
    var project = await projectRepository.GetProjectAsync(projectId, includeRelated

    if (project == null)
        return NotFound();

    var projectName = project.Name;

    if (file == null) return BadRequest("Nepostojeći podatak.");
    if (file.Length == 0) return BadRequest("Prazan podatak.");
    if (file.Length > fileSettings.MaxBytes) return BadRequest("Maksimalna veličina
    if (!fileSettings.IsSupported(file.FileName))
        return BadRequest("Nevaljali tip podatka.");

    var uploadsFolderPath = Path.Combine(host.WebRootPath, projectName);
    if (!Directory.Exists(uploadsFolderPath))
        Directory.CreateDirectory(uploadsFolderPath);

    var encryptedFileName = Guid.NewGuid().ToString() + Path.GetExtension(file.File
```

Sl. 3.17. *Metode upravljača podatka s pripadnim atributima za slanje/dohvaćanje podataka*

Prvo se radi validacija, odnosno provjera projekta, zatim validacija samog podatka uz pomoćnu klasu na slici 3.18. . Princip rada ovog upravljača zapravo je jednostavan, uz standardnu validaciju i spremanje ili dohvaćanje, brine se i o strukturnom prostoru na fizičkoj lokaciji servera za pojedini projekt, na način da kod prvog spremanja podatka za neki projekt kreira datoteku u koju se pohranjuju podaci jedinstveni za taj projekt.

Kako potencijalni napadači ne bi mogli koristiti ovaj mehanizam spremanja podatka da bi dobili pristup serveru, ime podatka se štiti na način da mu se dodaje poseban ključ kao prefiks već odabranom imenu.

```
3 references
public class FileSettings
{
    1 reference
    public int MaxBytes { get; set; }
    1 reference
    public string[] AcceptedFileTypes { get; set; }
    1 reference
    public bool IsSupported(string fileName)
    {
        return AcceptedFileTypes.Any(
            s => s == Path.GetExtension(fileName).ToLower());
    }
}
```

Sl. 3.18. *Klasa koja sadrži svojstva koja dodatno validiraju podatak prilikom spremanja*

Na prethodno već spomenutoj slici je dobro reći da su vrijednosti ovih svojstava definirane u postavkama servera (sl. 3.6.4.), koje je moguće vrlo lako izmijeniti, a nije potrebno ponovno pokretati server kako bi one stupile na snagu.

Postavke su spremljene u *JSON* formatu (engl. *Javascript Object Notation*), što je zapravo standardizirani format.

Objekt pod nazivom *jwt* (skraćeno od engl. *JSON Web Token*) ima svojstva *audience* i *authority*, koja označavaju vrijednosti za pristup vanjskoj autorizaciji korisnika (engl. *external authorization*). Naravno, moguće je definirati i implementirati vlastito rješenje odnosno generiranje ključa za pristup, no nema potrebe za izmišljanjem tople vode jer sigurna i besplatna (s ograničenim opcijama) rješenja i alati već postoje. Nisu podržani podaci s proširenjem *.exe* (skraćeno od *executable*) jer oni u najgorem slučaju mogu ozbiljno naštetiti korisnikovom računalu.



Sl. 3.19. *Jedne od postavki servera, odnosno vrijednosti s kojima se olakšava rad sa serverom*

Kod upravljanja korisnika, odnosno registracije i prijave, a i prihvaćanja prijava na projekt i grupe odgovorni su upravljači *AccountController* i *UserController*.

Na sljedećoj slici se vidi implementacija tih upravljača. Kao parametar prve metode, metode odgovorne za registraciju odnosno spremanje korisnika u bazu podataka, koristi se poseban model pod nazivom *RegistrationResource*. U njemu je jasno definirano što server prihvaća i traži od korisnika (iz sigurnosnih razloga) kako bi se korisnik uspješno registrirao. Između ostalog radi se validacija zahtjeva, upit postoji li već korisnik itd..

```
[HttpPost]
0 references
public async Task<IActionResult> Register([FromBody] RegistrationResource registrationRes
{
    var user = userRepository.GetUser(registrationResource.Id);

    if (user != null)
        return BadRequest("Korisnik već postoji");

    if (!ModelState.IsValid)
        return BadRequest(ModelState);

    var newUser = mapper.Map<RegistrationResource, User>(registrationResource);

    await context.Users.AddAsync(newUser);

    return Ok(registrationResource.Id);
}

[HttpGet("{tokenId}")]
0 references
public async Task<IActionResult> Login(string tokenId)
{
    if (!ModelState.IsValid)
        return BadRequest(ModelState);

    var user = await context.Users.FindAsync(tokenId);

    if (user == null)
        return BadRequest(ModelState);
```

Sl. 3.20. *Upravljači koji se koriste za rukovanje korisnicima*

Sljedeća slika prikazuje upravljač projekata *ProjectController*, koji ima ulogu, slično kao i prethodni upravljači, rukovanja s projektima (između ostalog *CRUD* (engl. *Create, Read, Update, Delete*)). Primjer je metoda *CreateProjectAsync* s točno jednim parametrom, a ima funkciju validacije i spremanja novog projekta u bazu podataka.

```
[HttpPost]
0 references
public async Task<IActionResult> CreateProjectAsync([FromBody] SaveProjectResource project
{
    if (!ModelState.IsValid)
        return BadRequest(ModelState);

    var project = mapper.Map<SaveProjectResource, Project>(projectResource);
    project.DateCreated = DateTime.Now;

    var projects = await projectRepository.GetAllProjectsAsync();

    if (projects.Any(p => p.Name == project.Name))
        return BadRequest("Ime projekta već postoji");

    projectRepository.AddProject(project);
    await unitOfWork.CompleteAsync();

    project = await projectRepository.GetProjectAsync(project.Id);

    if (project == null)
        return NotFound();

    var result = mapper.Map<Project, ProjectResource>(project);

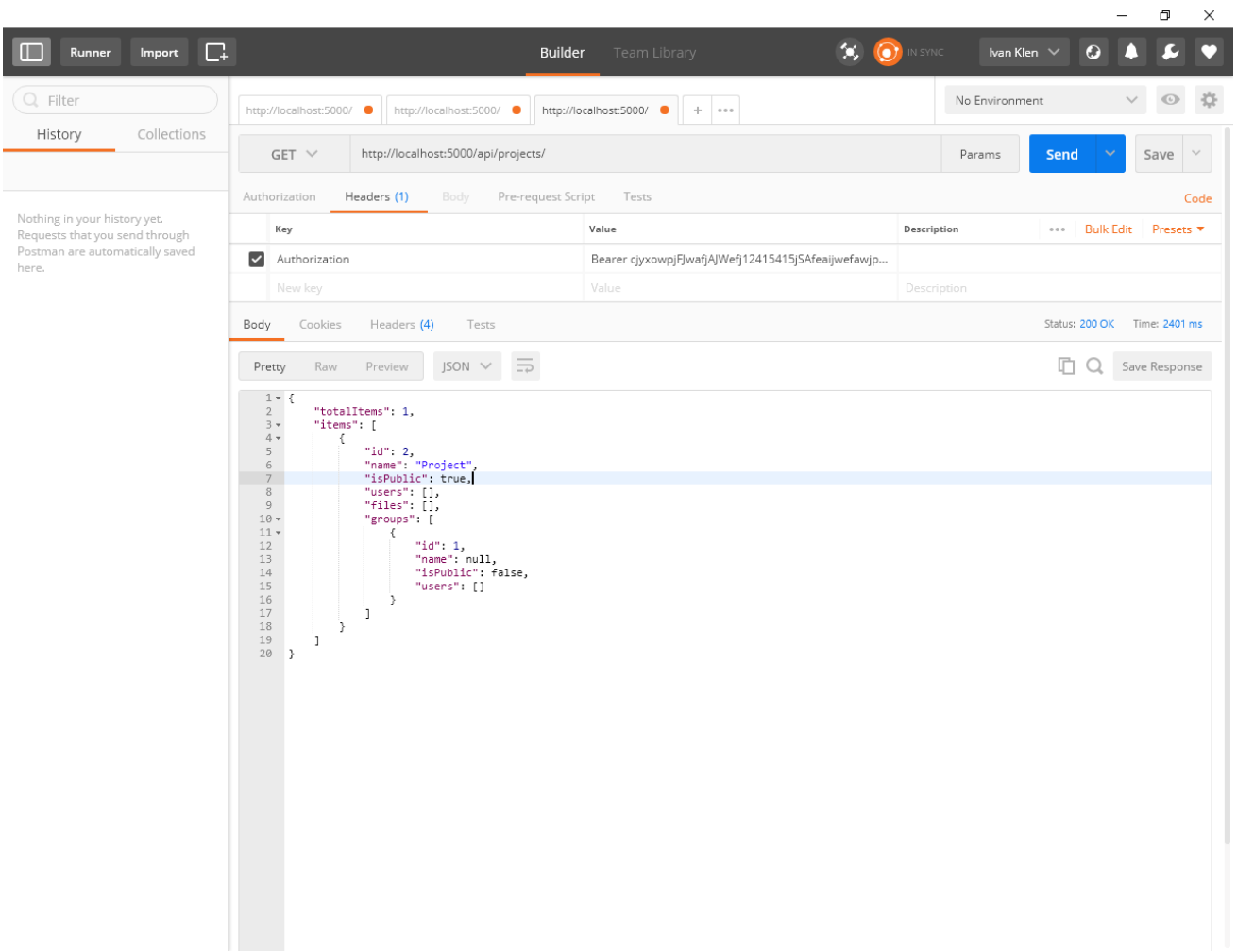
    return Ok(result);
}
```

Sl. 3.21. *Upravljač projekta koji koristi krajnje točke ili put za pristup i rukovanje projektima*

Prije nego što se krenulo u razvoj klijentskog rješenja dijela aplikacije, postavilo se pitanje rade li pristupne krajnje točke prethodno navedenih upravljača kako treba.

Za provjeru pristupnih točaka korišten je alat pod nazivom *Postman* (besplatan alat), koji jednostavno radi zahtjeve prema serveru s neobaveznim parametrima i podesivosti. Na sljedećoj slici se vidi rad tog alata i neki od zahtjeva i odgovora na zahtjeve.

Zahtjev koji je poslan na upravljač *ProjectController* je *GET* zahtjev, odnosno zahtjev za dohvaćanje svih projekata. No, važno je napomenuti da se u zaglavlju (engl. *Header*) s neobaveznim parametrima nalazi i autorizacijski ključ (kojeg korisnik mora imati) bez kojeg nije moguće dobiti uspješan odgovor od servera. Ključ je zapravo jedinstven niz znakova (engl. *token*) kojeg stvara vanjski sustav za autorizaciju.



Sl. 3.22. *Alat 'Postman', korišten za provjeru krajnjih točaka pristupa podacima*

3.7. Klijentsko rješenje dijela aplikacije

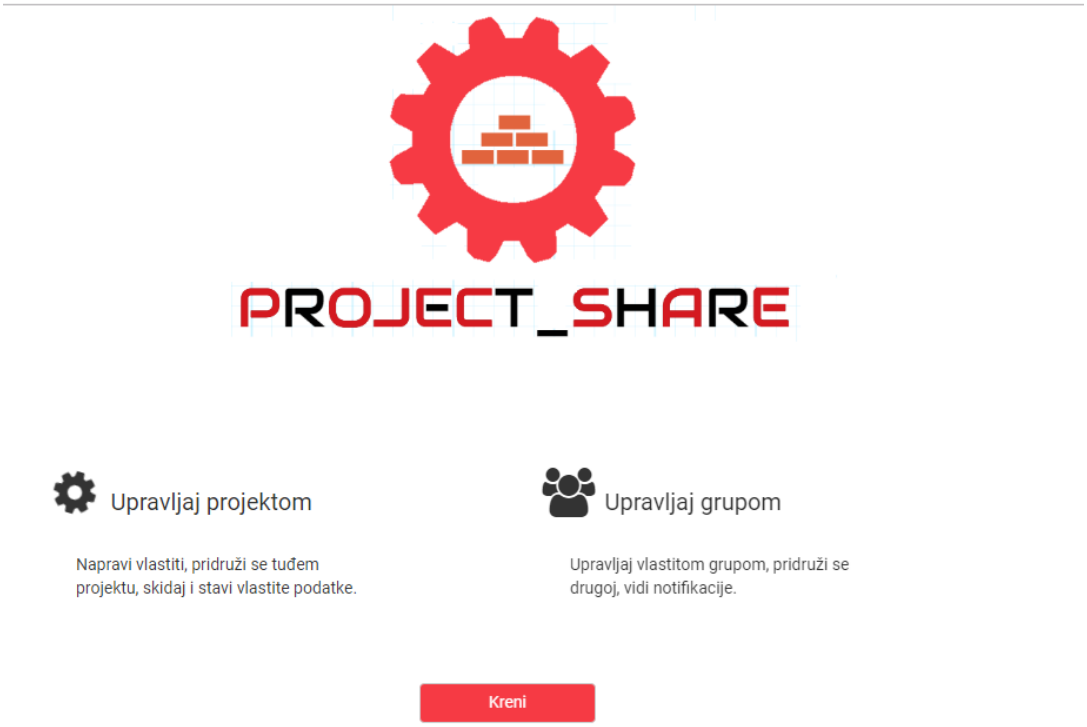
S obzirom na to da je funkcionalnost serverskog rješenja dijela aplikacije provjerena, važno je dati običnom korisniku način kojim može jednostavno koristiti aplikaciju.

U prijevodu, serversko rješenje je ono što se nalazi 'ispod haube', a klijentsko je ono što korisnik vidi, čuje i s čim može uspostaviti razumnu interakciju (pod pojmom razumne interakcije se podrazumijeva da korisnik može nešto kliknuti na zaslonu i vidjeti rezultate tog klika).

Na sljedećoj slici se vidi početna stranica kada neregistrirani i neprijavljeni korisnik pristupi stranici.

Logo koji je korišten na početnoj stranici je rađen jednim od besplatnih alata na internetu. Poanta je da je vrlo lako izraditi vlastiti logo. Iako je za projekte ove veličine (društvena mreža s punom funkcionalnošću) potreban veći broj ljudi sa specifičnim vještinama (dizajn, iskustvo korisnika UX (engl. *User Experience*)) nije nikakav problem organizirati kod i prirediti korisniku ugodno iskustvo prilikom korištenja aplikacije.

Sl. 3.23.



Početna stranica aplikacije

Dio koda koji je korišten prilikom izrade početne stranice je vidljiv na sljedećoj slici. Kod je pisan u HTML-u (engl. *HyperText Markup Language*) koji je standard za izradu web stranica. Između ostalog, korišten je i *Bootstrap* (*open-source* alat koji ima set definiranih klasa spremne za upotrebu) za pozicioniranje elemenata na ekranu.

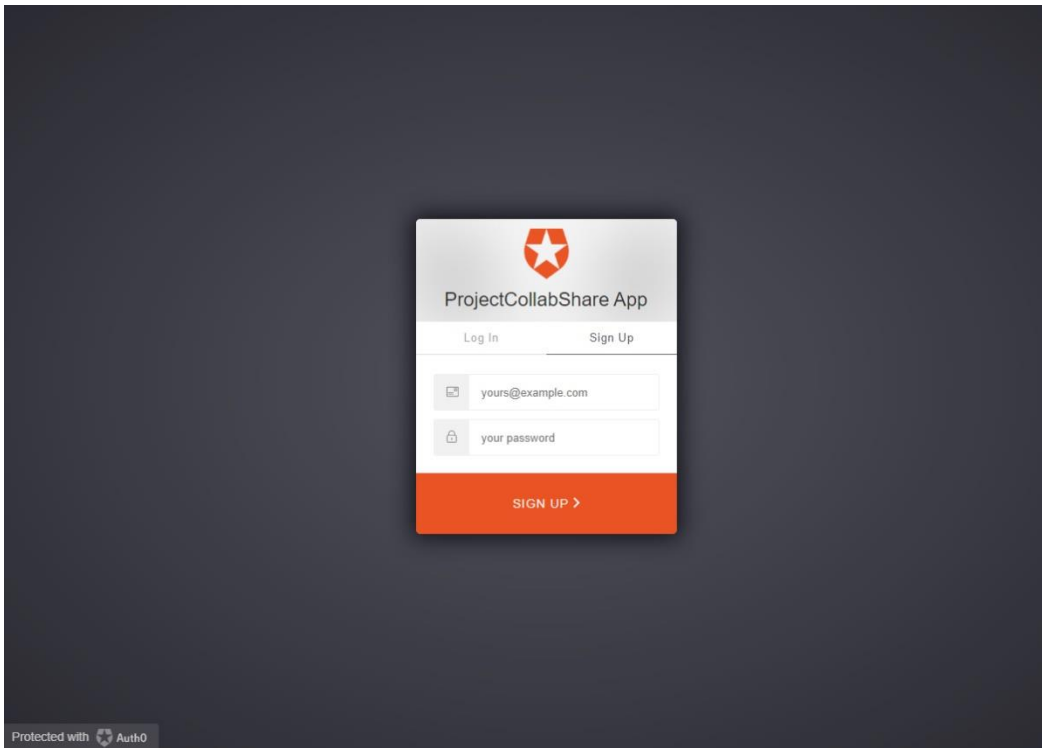
Moguće je koristiti i druge alate prilikom izrade korisničkog rješenja, no dokumentacija i resursi za ovu tehnologiju su najdostupniji.

```
<div>
    <div class="row">
        <div class="col-sm-2">
        </div>
        <div class="col-sm-7">
            <img class="img-responsive" src="project logo.png" alt="">
        </div>
    </div>
    <div class="row" style="margin-top: 60px;">
        <!-- feature box -->
        <div class="col-md-4 col-sm-4 col-xs-12 xs-margin-fifteen-bottom" style="margin-right: 100px;">
            <div class=" display-table margin-five-bottom width-100">
                <div class="text-greenish-blue icon-style margin-seven-right display-table-cell-vertical-middle float-none">
                    <i class="fa fa-cog fa-spin fa-3x fa-fw"></i>
                    <span class="text-medium text-medium-gray tz-text" style="font-size: 20px;">Upravljaj projektom</span>
                </div>
            </div>
            <div class="text-medium width-95 xs-width-100 xs-margin-five-top tz-text">
                <p class="no-margin-bottom" style="margin-top: 30px; margin-left: 28px;">
                    Napravi vlastiti, pridruži se tuđem projektu, skidaj i stavi vlastite podatke.
                </p>
            </div>
        </div>
        <!-- end feature box -->
        <!-- feature box -->
        <div class="col-md-4 col-sm-4 col-xs-12 xs-margin-fifteen-bottom">
            <div class=" display-table margin-five-bottom width-100">
                <div class="text-greenish-blue icon-style margin-seven-right display-table-cell-vertical-middle float-none">
                    <i class="fa fa-users fa-3x fa-fw"></i>
                    <span class="text-medium text-medium-gray tz-text" style="font-size: 20px;">Upravljaj grupom</span>
                </div>
            </div>
```

Sl. 3.24. *Kod korišten prilikom izrade početne stranice aplikacije*

Na slici 3.23. klikom na tipku *Kreni,* korisniku se otvara nova stranica za registraciju i za autorizaciju, što je vidljivo na sljedećoj slici. Korišten je vanjski resurs za autorizaciju, registraciju i prijavu, što zbog sigurnosti, jednostavnosti implementacije, a što zbog uštede vremena. Naziv kompanije koja se brine o tom je *Auth0*. Kao što je već rečeno, moguće je implementirati vlastito rješenje, no ono iziskuje veliko ulaganje, a samim time vlasnik aplikacije nosi veliki rizik od curenja podataka. Ovako je većina tih stvari izbjegnuta i autorizacija korisnika je predana u ruke 'velikih igrača'.

Nažalost, koliko je to dobro, toliko je i loše, jer ovaj vanjski resurs zasad nije moguće puno mijenjati, no barem je aplikacija sigurna od napadača.



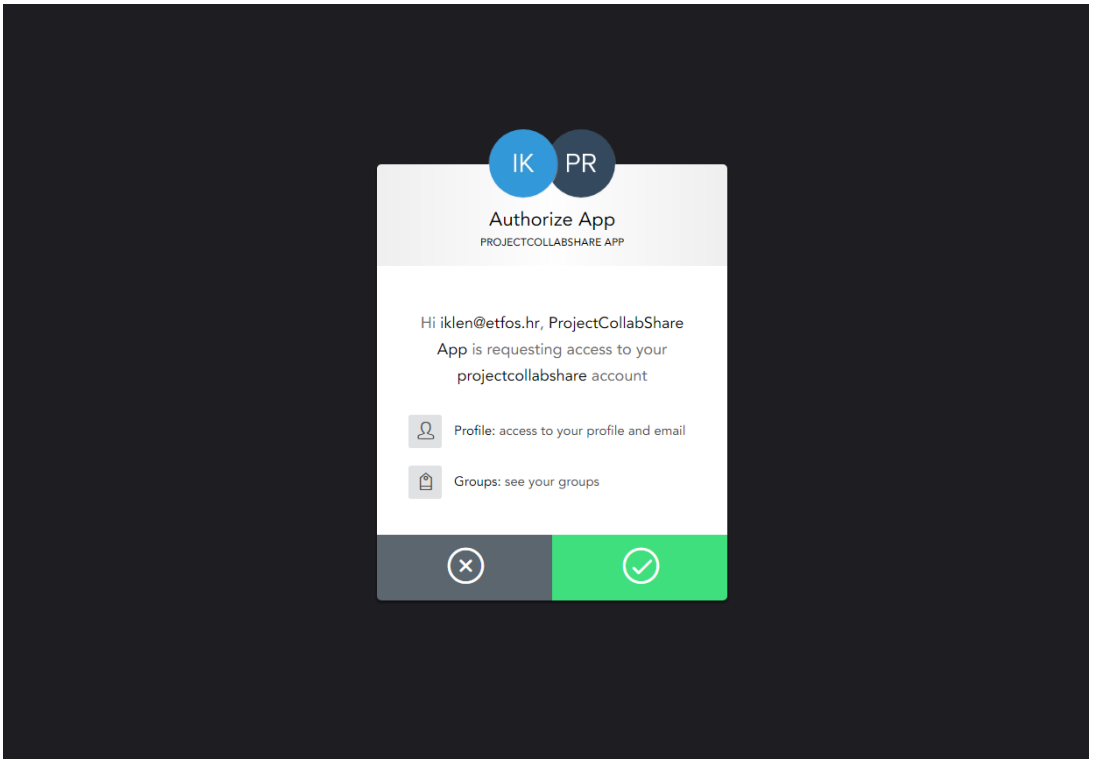
Sl. 3.25. *Vanjski resurs korišten za autorizaciju, prijavu i registraciju korisnika*

Nakon što se korisnik registrira ili prijavi, biva preusmjeren na našu aplikaciju, a aplikacija tu prijavu ili registraciju registrira i sprema korisnika u bazu, odnosno dohvaća korisnika iz baze radi dodatne autorizacije odnosno provjere.

Nadalje, nakon što se novi korisnik registrira, pojavljuje mu se upit slaže li se s tim da aplikacija ima pravo u uvid njegovih grupa i njegov profil.

Korisnik ima opcije odbiti ili dozvoliti aplikaciji pristup u uvid u njegove podatke. Ukoliko odbije dozvoliti uvid, i dalje će se uspješno registrirati, ali se neće moći prijaviti i početi s korištenjem aplikacije sve dok ne da dozvolu aplikaciji u uvid podataka. (Konstantno će se pojavljivati upit dok korisnik ne pritisne kvačicu).

Sl.



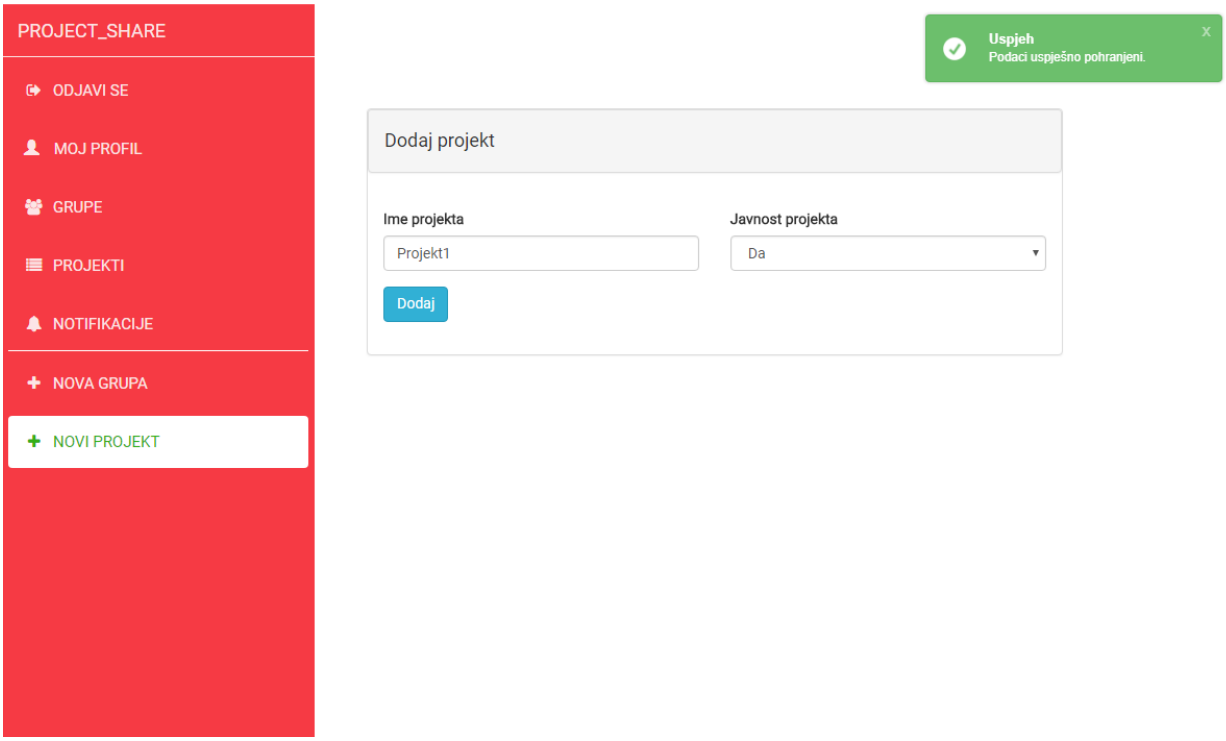
3.26. *Vanjski resurs daje upit korisniku slaže li se s policom aplikacije, odnosno daje li dozvolu aplikaciji da ima uvid u njegove podatke*

Nakon što korisnik potvrdi dozvolu aplikaciji u uvid podataka, biva preusmjeren na vlastiti profil te može početi s normalnim korištenjem aplikacije. Nužno je da korisnik da dozvolu jer aplikacija tako funkcionira. Vanjski resurs *Auth0* zatim korisniku šalje elektronsku poštu koju korisnik mora potvrditi.

S lijeve strane mu se pojavljuje izbornik s kojim se može kretati do ostalih dijelova aplikacije, kao što su grupe ili projekti, također ima uvid u notifikacije, koje će biti dostupne tek kada bude prihvaćen u projekt ili grupu.

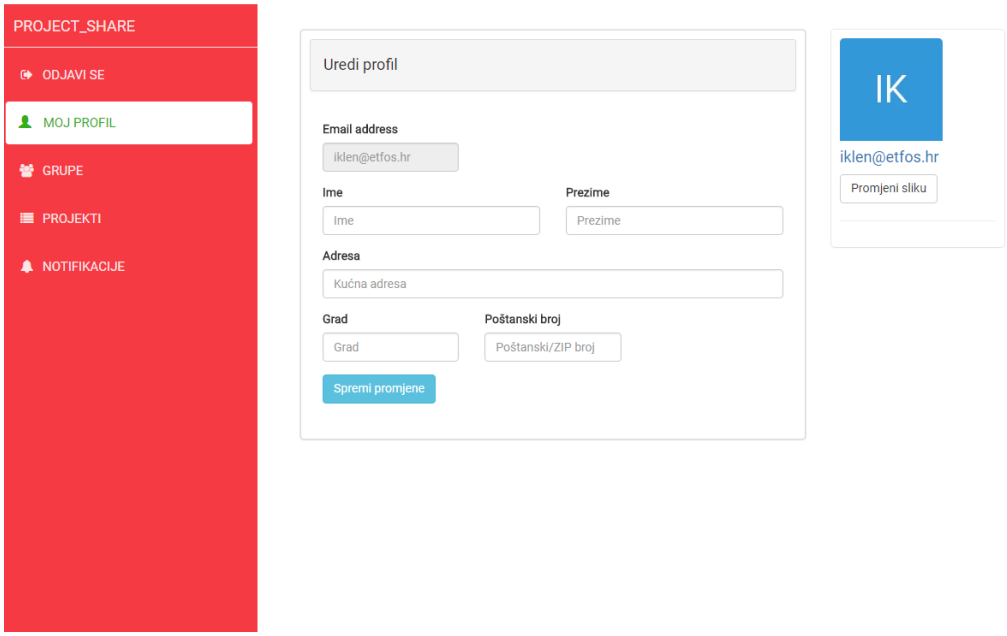
Slika 3.27. prikazuje proces dodavanja projekta, ukoliko je projekt uspješno spremljen, pokazuje se poruka u gornjem desnom uglu da su podaci uspješno pohranjeni.

Iz tehničkih razloga, moguće je isključivo da jedan korisnik bude vlasnik jedne grupe ili projekta, te mu se onemogućuje dodavanje novog projekta nakon što napravi prvi projekt.



Sl. 3.27. *Stranica na kojoj korisnik može dodati vlastiti projekt, a do stranice dođe putem izbornika s lijeve strane*

Na stranici vlastitog profila može uređivati svoje podatke, te klikom na tipku *Spremi promjene* se ti podaci spremaju u bazu podataka. Dakle, poanta prethodno spomenutog vanjskog resursa identifikacije korisnika je ta da može biti siguran da se njegova lozinka nigdje na ovoj aplikaciji neće spremati.



Sl. 3.28. *Stranica korisnikovog profila, gdje je moguće uređivati vlastiti profil*

Sljedeća slika prikazuje kod koji je korišten prilikom izrade stranice za uređivanje vlastitog profila pojedinog korisnika.



Sl. 3.29. *HTML kod korišten za izradu stranice za uređivanje profila*

Slika 3.30. prikazuje sloj niže od prethodne slike, odnosno poveznicu između sloja za prikaz podataka i sloja pristupa podacima (serverskog sloja).

TypeScript je nadskup *JavaScript* programskog jezika, sa svim blagodatima objektno orijentiranog jezika. Razlog zašto je korišten *TypeScript* umjesto *JavaScript*-a je taj što kao i kod *C#* programskog jezika, ima provjeru tipova podataka, tako da su šanse za ranijim otkrivanjem pogreške u kodu daleko veće.

```typescript
import { AuthService } from './../../services/auth.service';
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-profile',
  templateUrl: './profile.component.html',
  styleUrls: ['./profile.component.css']
})
export class ProfileComponent implements OnInit {

  profile: any;

  constructor(public auth: AuthService) { }

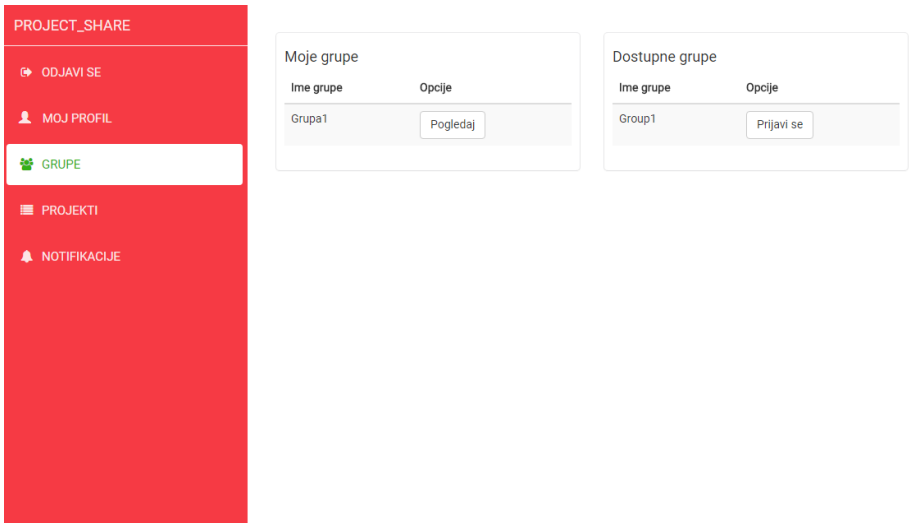
  ngOnInit() {
    if (this.auth.userProfile) {
      this.profile = this.auth.userProfile;
    } else {
      this.auth.getProfile((err, profile) => {
        this.profile = profile;
      });
    }
  }

}
```

Sl 3.30. *Typescript kod koji je odgovoran za prijenos korisnikovog profila iz baze na stranicu i obratno*

Slika 3.31. prikazuje stranicu s dostupnim grupama i trenutnim grupama u koje je korisnik učlanjen. U tablici dostupnih grupa korisnik može poslati zahtjev za učlanjenjem u pojedinu grupu klikom na tipku *Prijavi se*. Zatim se prijava za učlanjenje u grupu šalje vlasniku te grupe, koji ima mogućnost odbiti prijavu ili prihvatiti.

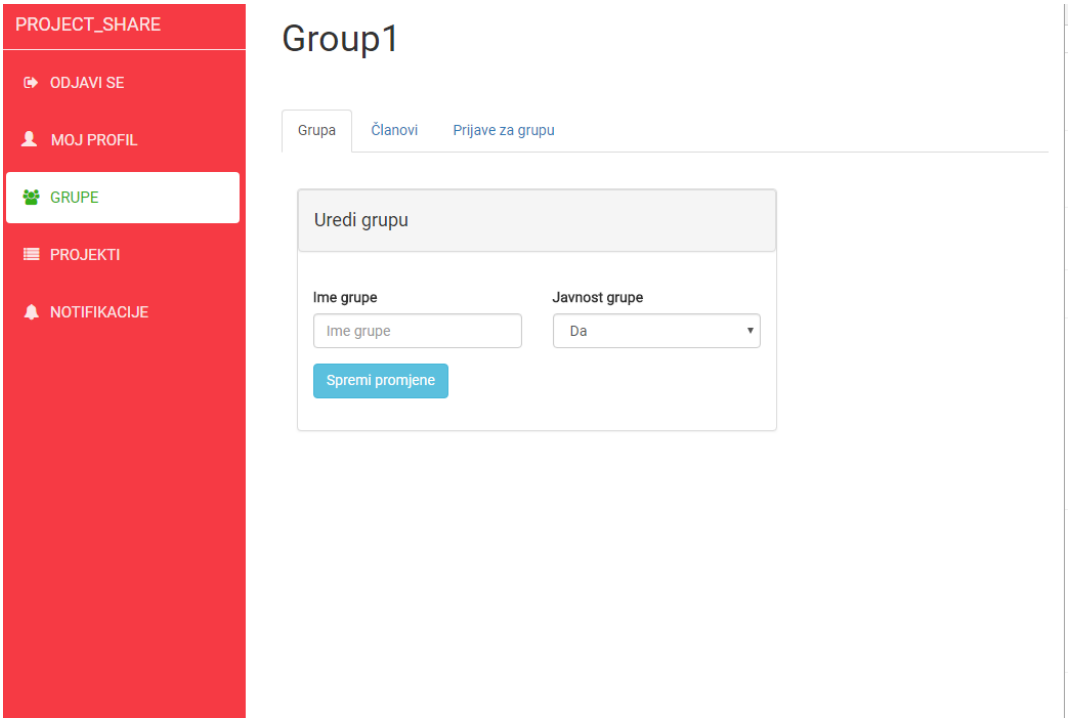
Isto tako, klikom na tipku *Pogledaj* u tablici *Moje grupe* korisniku se otvara nova stranica koja sadrži nekoliko stvari, od kojih su članovi te grupe, i opcije koje su mu dostupne ukoliko je taj korisnik vlasnik te grupe, kao što su opcije za brisanje, mijenjanje sadržaja grupe.



Sl. 3.31. *Stranica sa grupama na do koje korisnik može doći klikom na izborniku na poveznicu 'Grupe'*

Slika 3.32. sadrži prikaz stranice na kojoj korisnik (koji je ujedno i vlasnik te grupe) ima ovlasti promijeniti postavke grupe (ime i vidljivost ostalim korisnicima aplikacije).

Manji izbornik ispod naziva korisnikove grupe sadrži tablice slične kao i na prethodnoj slici koje prikazuju članove i opcije za brisanje ili prihvaćanje članova u grupu.



Sl. 3.32. *Stranica sa grupom u kojoj je trenutni korisnik ujedno i vlasnik te ima ovlasti uređivanja grupe*

Na slici 3.33. vidljiv je sličan kod kao i na slici 3.30. , no funkcija ovog koda je znatno drugačija. Naime, njegova funkcija je sortiranje i dohvaćanje manjeg broja redaka podataka o grupi iz baze podataka. Važno je napomenuti da se na ovaj način poboljšava korisničko iskustvo, jer korisnik steče dojam da se dostupne grupe brzo učitavaju, dok se zapravo na jedan način radi segmentiranje odnosno dijeljenje podataka u stranice, odnosno straničenje (engl. *server-side paging*).

```
    .subscribe(res => {
      this.queryResult = res
    });
}

onFilterChange() {
  this.query.page = 1;
  this.populateGroups();
}

resetFilter() {
  this.query = {
    page: 1,
    pageSize: this.DEFAULT_PAGE_SIZE
  };
  this.queryResult = {};
  this.populateGroups();
}

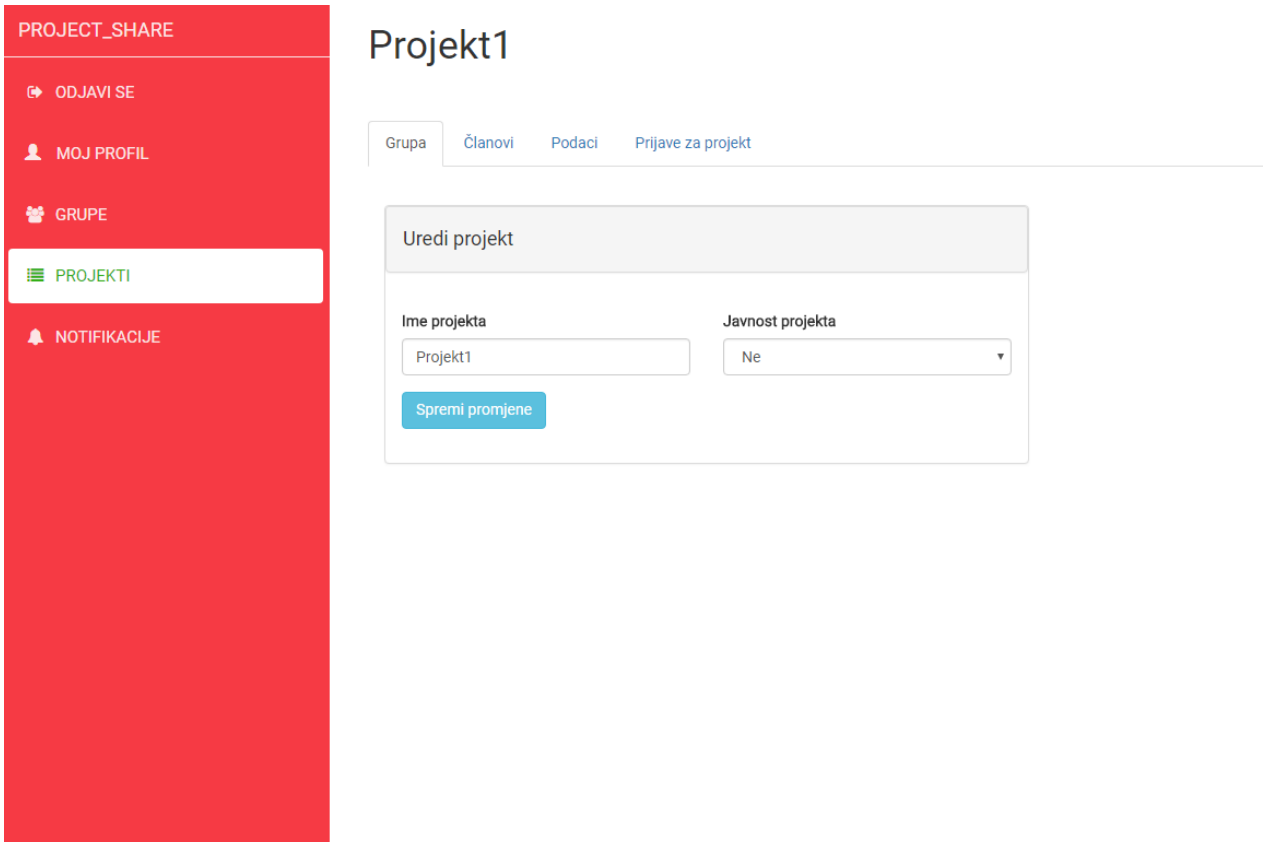
sortBy(columnName) {
  if (this.query.sortBy == columnName) {
    this.query.isSortAscending = !this.query.isSortAscending;
  } else {
    this.query.sortBy = columnName;
    this.query.isSortAscending = true;
  }
  this.populateGroups();
}

onPageChange(page) {
  this.query.page = page;
```

Sl. 3.33. *Primjer koda odgovoran za 'server-side paging' na klijentskoj strani*

Sljedeća slika prikazuje pregled projekta korisnika koji je također vlasnik tog projekta. Stranica sadrži nekoliko manjih izbornika, no od prethodne slike sa stranicom grupe, razlika je ta što ovdje korisnik ima izbornik *Podaci*.

Klikom na taj izbornik korisniku se otvara pod-stranica koja sadrži pregled podataka koje su trenutno u datoteci na fizičkoj lokaciji (u ovom slučaju – lokalno računalo).



Slika 3.34. *Stranica sa projektom iz perspektive vlasnika projekta s malim izbornicima za navigaciju do raznih opcija*

Naravno, kod je većim dijelom podijeljen u komponente stoga je vrlo lako izmijeniti redoslijed navigacijskog izbornika ili slično. No, modularnost, kao i apstrakcija slojeva, ima svoju cijenu, a ta je da je potrebno puno više vremena novom programeru da se 'uhoda' u projekt i započne s radom.

Sljedeća slika prikazuje kod korišten prilikom izrade stranice za upravljanje projektom. U gornjem dijelu slike se jasno vide navigacijski izbornici sa *id* atributima pomoću kojih se korisniku klikom miša otvara novi sadržaj, a stari zatvara.

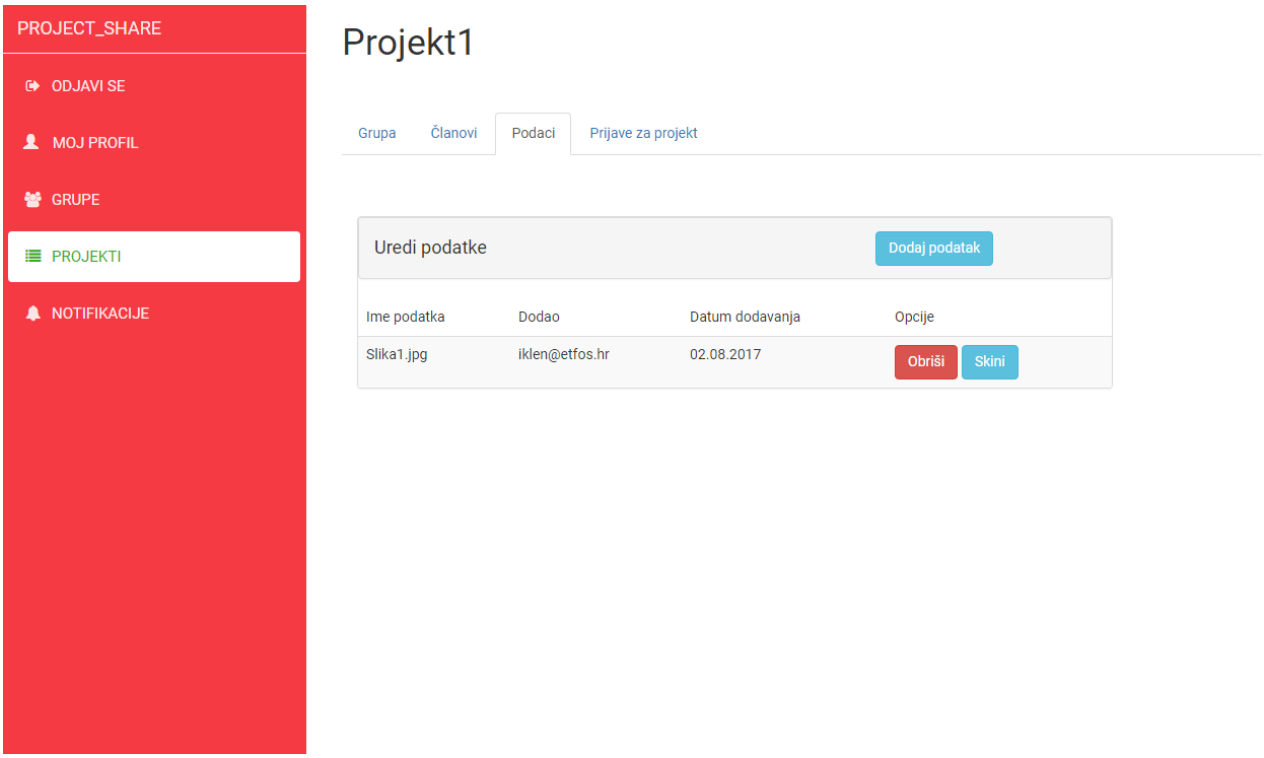
```
<!-- Nav tabs -->
<ul class="nav nav-tabs" role="tablist" style="margin-top: 50px;">
    <li role="presentation" class="active"><a href="#project" aria-controls="project" role="tab" data-toggle="tab">
    <li role="presentation"><a href="#members" aria-controls="members" role="tab" data-toggle="tab">Članovi</a></li
    <li role="presentation"><a href="#files" aria-controls="files" role="tab" data-toggle="tab">Podaci</a></li>
    <li role="presentation"><a href="#applications" aria-controls="applications" role="tab" data-toggle="tab">Prija
</ul>

<!-- Tab panes -->
<div class="tab-content">
    <div role="tabpanel" class="tab-pane" id="project" style="margin-top: 35px;">
        <div class="main-content">
            <div class="container-fluid">
                <div class="row">
                    <div class="col-md-8">
                        <div class="card panel panel-default">
                            <div class="header panel panel-heading">
                                <h4 class="title">Uredi projekt</h4>
                            </div>
                            <div class="content panel-body">
                                <form novalidate="" class="ng-untouched ng-pristine ng-valid">
                                    <div class="row">
                                        <div class="col-md-6">
                                            <div class="form-group">
                                                <label>Ime projekta</label>
                                                <input class="form-control" placeholder="Ime projekta" type="text"
                                            </div>
                                        </div>
```

Sl. 3.35. *Stranica sa projektom iz perspektive vlasnika projekta s malim izbornicima za navigaciju do raznih opcija*

Slika 3.36. ima jasan prikaz kontrole vlasnika projekta nad podacima koje korisnici njegovog projekta stavljaju, primjerice, klikom na tipku *Obriši*, briše se podatak iz podatkovne strukture namijenjene za rad tog projekta. Samim time taj podatak postaje nedostupan i nije ga moguće povratiti.

Pritiskom na tipku *Skini*, korisnik automatski počinje skidati podatak. S obzirom da se radi o lokalnom računalu i lokalnoj adresi (*Localhost*), ne pojavljuje se nikakav sigurnosni niti zaštitna mjera, no web agent i dalje nakon skidanja podatka, skenira podatak radi virusa i sl. .

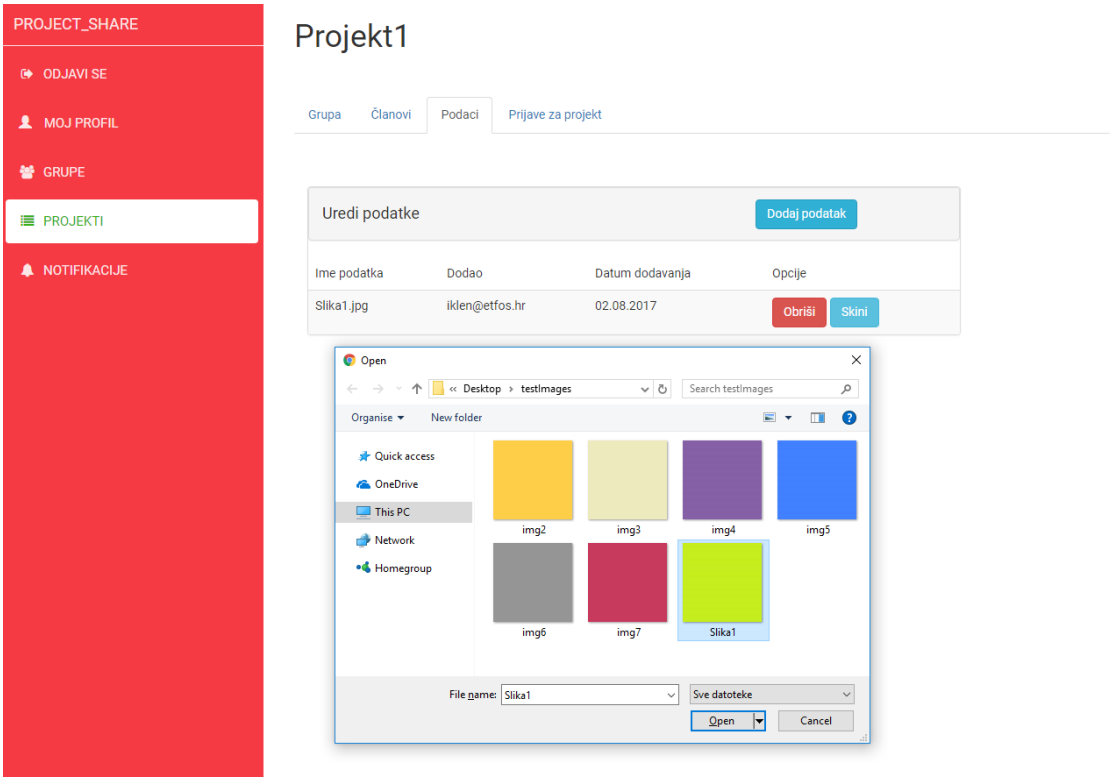


Sl. 3.36. *Stranica sa projektom – izbornik 'Podaci', prikaz kontrole nad podacima, tko je, i kada dodao koji podatak*

Koncept interaktivnosti je u sljedećoj slici očit, naime, korisnik klikom miša na tipku *Dodaj podatak* dobiva mogućnost stavljanja podatka na fizičku lokaciju na serveru (u ovom slučaju lokalno računalo). Taj podatak je automatski vidljiv ostalim sudionicima projekta.

Korisnicima aplikacije i sudionicima projekta se ostavlja na izbor žele li skidati te podatke ili ne. Potrebno su znatno veći resursi i znanje za zaštitu protiv stavljanja malicioznih podataka.

Sl.



3.37. *Stranica sa projektom – izbornik 'Podaci', prikaz dijaloškog okvira koji se pojavljuje ukoliko korisnik želi staviti neki podatak*

Sljedeća slika prikazuje poveznicu između klijentskog i serverskog rješenja za prijenos podataka. Jasno se vide prethodno spomenute krajnje pristupne točke kojima pojedina funkcija na slici ispod pristupa sa određenim parametrima. No, tko sprječava da korisnik sam ne zatraži podatke drugih projekata?

Odgovor na to pitanje je dvostruka validacija. U praksi ju je teško potpuno implementirati, no uzmimo za primjer način na koja ona funkcionira kroz analogiju. Recimo da korisnik dođe na kontrolnu točku koja zahtijeva samo ime i prezime, on se lažno predstavi i čuvar ga pusti dalje. No da bi došao do pravih podataka mora proći još jednu kontrolnu točku, a na toj kontrolnoj točki radi čuvar koji zahtijeva razno razne dodatne dokumentacije bez kojih korisnik ne može dalje.



```
import { Http } from '@angular/http';
import { Injectable, Inject } from '@angular/core';

@Injectable()
export class FileService {
  private readonly groupEndPoint = this.originUrl + '/api/groups';
  constructor(private http: Http, @Inject("ORIGIN_URL") private originUrl: string) { }

  upload(projectId, file) {
    var formData = new FormData();
    formData.append('file', file);
    return this.http.post(this.originUrl + `/api/projects/${projectId}/files`, formData)
      .map(res => res.json());
  }

  getFiles(projectId) {
    return this.http.get(this.originUrl + `/api/projects/${projectId}/files`)
      .map(res => res.json());
  }

  download(fileId, projectId) {
    return this.http.get(this.originUrl + `/api/projects/${projectId}/files/${fileId}`)
      .map(res => res.json());
  }
}
```

Sl. 3.38. *Klasa FileService – funkcija joj je prijenos podatka u oba smjera*

4. ZAKLJUČAK

S obzirom na kompleksnost ovog projektnog zadatka, veliki dio funkcija modernih društvenih stranica je uspješno implementiran. Naglasak je bio na dobro razvijenom i održivom serverskom rješenju za koji praktički i ne treba puno dokumentacije. Način implementacije i izloženost pruža mogućnost programerima da koriste svoja klijentska rješenja za pristup serveru (upravo zbog *API*-ja, odnosno krajnjih točaka s kojih server zaprima zahtjeve).

Korisnicima je omogućen siguran pristup i upravljanje vlastitim podacima, što je jedna od osnovnih stavki koje svaka iole ozbiljna aplikacija mora imati. Broj dostupnih besplatnih alata je puno pomogao u razvoju ove aplikacije, no s obzirom na količinu alata razne kvalitete, te upravo zbog tog što su besplatni, su podložni vrlo dinamičnim promjenama u kratkom vremenu, najčešće s nepotpunom dokumentacijom promjena.

Dinamičnost promjene današnje tehnologije je velika, a naprijed ju tjera upravo korisnik. Stoga je potrebno neprestano raditi na proizvodu kako bi se osiguralo najbolje korisničko iskustvo.

Također ne treba zanemariti marketinške mogućnosti koje se pružaju izradom ove aplikacije, naravno, uz veći broj ljudi koji radi na razvoju i održavanju te uz veće resurse, a te su primjerice povećavanje slobodnog prostora korisnicima koji se pretplate na korištenje aplikacije.

Sve u svemu, prostora za poboljšanje i napredak aplikacije uvijek ima, no ograničenost resursa, marketinške strategije i nedostatak ispitivanja tržišta predstavljaju veliku zapreku. Razvoj aplikacije je izuzeo podosta vremena, no, taj proces je predstavio dobru priliku za daljnjom nadogradnjom znanja.

LITERATURA

[1] Nagel, C.: **Professional C# and .NET Core 1.0**, John Wiley & Sons, Inc., Indianapolis, Indiana, 2016. .

[2] Albahari, J., Albahari, B.: **C# in a Nutshell**, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2016.

[3] Freeman, A.: **Pro ASP.NET MVC 5**, Springer Science + Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013, 2013.

[4] Haverbeke M.: **Eloquent JavaScript, 2nd Edition**, No Starch Press, Inc., 245 8th Street, San Francisco, CA 94103, 2015.

[5] MacDonald M.: **HTML5: The Missing Manual, 2nd Edition**, O'Reilly Media, 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2013.

[6] Price M. J.: **C# 6 and .NET Core 1.0: Modern Cross-Platform Development**, Packt Publishing Ltd., Livery Place, 35 Livery Street, Birmingham B3 2PB, UK. 2016.

SAŽETAK

Glavni cilj završnog rada bio je izrada samostalne web aplikacije za dijeljenje podataka i suradnju. Korisnicima je kroz korištenje aplikacije osiguran pristup i privatnost, te je samo korištenje aplikacije dosta intuitivno. U svijetu gdje sve vrvi od malicioznih programa i nesigurnosti tokom surfanja internetom, sigurnost je prilikom izrade bila na prvom mjestu. Teorijski dio rada ukratko opisuje glavne alate i tehnologije korištene pri izradi, te arhitekturalni koncept i smjer u kojem se razvoj aplikacije kretao.

Praktični dio je posvećen prikazu načina implementacije prilikom razvoja, uz objašnjenja zašto i na koji način je izabran taj način implementacije u odnosu na druge. Unatoč tome što je aplikacija funkcionalna, za današnje standarde i korisnikove potrebe, koje rastu nekontroliranom brzinom, moglo bi se reći da je aplikacija tek u povojima.

Ključne riječi: podataka, korisnik, aplikacije, projekta, metoda, klase, baze, stranica, framework, podatak, broj, implementacije, objekta, pristupa, tehnologija

ABSTRACT

SOCIAL NETWORK FOR COLLABORATION AND CONTENT SHARING

The main purpose of this paper was to develop self sustaining Internet application for sharing files between users, and to provide additional collaboration. Through using this application, users are guaranteed security and privacy, also, navigating and using this application is intuitive. In the world, where Internet dangers are lurking everywhere, from malicious programs to insecurity of suring through the Internet, implementing security was of the utmost importance. Theoretical part of this paper shortly describes main tools and technologies which were used while developing the application, it also describes architectural concept and general direction in which development of the application was going.

Practical part is dedicated to displaying different ways of implementations through development, with clarifications why and in what way was that exact implementation used compared to others. In spite of application being fully functional, nowadays, standards and user needs are growing uncontrollably, it could be said that this application is in very early stages of its lifecycle.

Keywords: data, user, application, project, method, class, database, page, framework, file, number, implementation, object, access, technology

ŽIVOTOPIS

Ivan Klen rođen je 09.siječnja 1994. godine u Osijeku. 2000. godine započinje školovanje u Osnovnoj školi Tin Ujević, te se nakon četiri godine školovanja prebacuje u Osnovnu školu Grigor Vitez. Povodom završetka osnovne škole, 2008. godine upisuje Jezičnu gimnaziju u Osijeku koju završava 2012. godine s polaganjem obavezne državne mature. 2014. godine upisuje Elektrotehnički fakultet u Osijeku, stručni studij elektrotehnike, smjer informatika. U slobodno vrijeme čita knjige, bavi se programiranjem te razno raznim društvenim aktivnostima.

Ivan Klen