SVEUČILIŠTE U ZAGREBU

**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

Seminar iz kolegija:

**Objektno oblikovanje**

TUSK

*Tehnička dokumentacija*

## Bartol Bilić Luka Dumančić Luka Lenđel Karlo Ježić Jurica Kenda

Zagreb, siječanj, 2021.

[1. Design story](#_rjxifeqrqcza) 3

[2. Zahtjevi i use cases](#_36w1cny8wsc9) 4

[2.2. Use cases](#_95n7k69aldq5) 4

[3. Opis objektnog modela](#_33qzb4vckh3x) 10

[3.2. Dijagram klasa kontrolera](#_bl8ahj6bu3x9) 12

[3.3 Opis implementacije perzistencije](#_qr9spuqqe420) 12

[Aspect](#_bfo9vy30cw3s) 12

[3.4. Transakcije i unit of work](#_cygznfxzv36v) 19

[4. Backend i opis pozadinskog sučelja](#_xokcho9nlwsb) 21

4.1 [Uvjeti](#_rbojbykdkzts) 22

4.2 [Prikaz ključnih modela](#_dnpaqlmvu6dw) 23

[5. Desktop inačica](#_krtwqwk4le1n) 24

5.1 [MVVP obrazac u GUI sučelju aplikacije](#_lon7waz6jxpb) 24

5.2 [Tusk-desktop + MVVP](#_ahb0qlkm0pc0) 31

[6. Mobilna aplikacija iOS i Android](#_87qgw2dkdugj) 34

[7. Web aplikacija](#_aelmnwbft3o4) **43**

8. [NHibernate vs Entity Framework](#_k665gre614fm) **45**

[Distributeri](#_pkrk1xsvuzbw) 45

[Podržane baze podataka](#_96rt2c36xyrf) 45

[Upravljanje konkurentnim pristupom podacima](#_evferlylujlh) 45

[Korisne značajke](#_7164daeszt7e) 46

9. Testiranja web aplikacije 46

9[.1 White box testing](#_6n9hrise8lqa) 46

9[.2 Black box testing](#_mzp4vdvryg59) 47

## 

## 1. Design story

Sve više softverskih komanija prati i kontrolira efikasnost i efektivnost svojih radnika putem mnogih aplikacija. Neke od tih aplikacija su Jira, Asana te Trello. Aplikacije ove prirode namijenjene su prvenstveno konceptima povezanim uz organizaciju, usklađivanje i monitoriranje napretka projekta i njegovih manjih jedinica. nDakle, benefit ovakvog tipa aplikacija dolazi iz potrebe za optimiranim i efikasnijim načinima rada.

Obzirom na rapidni rast tehnologija, tehnika i kompanija u ovome području, aplikacije ovog tipa postale su sve složenije. Paradoksalno inicijalnoj želji za efikasnošću, same aplikacije za produktivnost su počele uzrokovati probleme vezane uz produktivnost.

Naime, aplikacije su počele patiti od problema poznatog kao "feature bloat". Ovaj problem se vrlo brzo manifestira otežanim korištenjem aplikacije. Aplikacija postaje švicarski nož, te u pokušaju da pruži sve, u stvari ne pruža ništa.

To je iz razloga što postaje otežano koristiti aplikaciju obzirom na njenu preveliku ponudu opcija. Korisnik postaje paraliziran opcijama, te čak može doći do ekstrema da ima potrebu ulagati vrijeme kako bi naučio koristiti aplikaciju koja mu je trebala služiti kao usputni alat za lakšu organizaciju.

Smatramo da su takve aplikacije nužne, ali mislimo da moraju postati jednostavnije. Potaknuti sržnim principima dizajna softvera (KISS, YAGNI) odlučili smo kreirati minimalno rješenje za firme, odnosno projekte kratke i srednje veličine.

Detektirali smo koje su to glavne značajke svih aplikacija za praćenje projekata i planiramo ih napraviti. Svi korisnici naše aplikacije su članovi timova i firmi, a sam projekt je podijeljen u sprintove (kratke blokove vremena - najčešće 7-14 dana).

Svaki sprint ima taskove koji prolaze kroz cikluse života (od trenutka kreiranja do trenutka kada su završeni), a taskovi imaju svoj opis, komentare, te tagove. Sprint je također podijeljen u aspekte projekta, a oni pokazuju tehničke aspekte projekta, ovise o skupu vještina osobe koja bi trebala riješiti zadatak.

## 2. Zahtjevi i use cases

2.1 Funkcionalni zahtjevi

1. Registracija korisnika
2. Prijava korisnika
3. Kreiranje i pregled kompanija
4. Kreiranje i pregled projekata
5. Kreiranje sprintova
6. Kreiranje zadataka
7. Kreiranje i pregled timova
8. Kreiranje i pregled aspekata
9. Komentiranje zadataka
10. Dodavanje reakcija na komentare
11. Pregled zadataka po “tagovima” i “skillovima”
12. Dodavanje novih korisnika u kompaniju i timove

### 2.2. Use cases

**2.2.1 Registracija korisnika**

**Primarni aktori:** Korisnik

**Stakeholders:**

* Baza

**Precondition:** Korisnik nema otvoren račun unutar aplikacije

**Postcondition:** Korisnik posjeduje račun s kojim može ući u aplikaciju

**Glavni uspješni scenarij:**

* Korisnik otvara aplikaciju i odlazi na tab za registraciju
* Korisnik upisuje potrebne informacije za registraciju
* Korisnik pritišće gumb za registraciju
* Korisnik registriran

**Alternativni scenarij:**

U svakom trenutku korisnik može odustati i otići na stranicu za prijavu

* Korisnik može mijenjati tab i vratiti se na prethodnu stranicu

Korisnik unosi postojeće podatke ili krivog formata

* Aplikacija upozorava korisnika i omogućuje mu prepravak

**2.2.2 Prijava korisnika**

**Primarni aktori:** Korisnik

**Stakeholders:**

* Baza

**Precondition:** Korisnik ima otvoren račun unutar aplikacije

**Postcondition:** Korisnik može pristupiti zaštićenom dijelu aplikacije

**Glavni uspješni scenarij:**

* Korisnik otvara aplikaciju i odlazi na tab za prijavu
* Korisnik upisuje potrebne informacije za prijavu
* Korisnik pritišće gumb za prijavu
* Korisnik prijavljen

**Alternativni scenarij:**

U svakom trenutku korisnik može odustati i otići na stranicu za registraciju

* Korisnik može mijenjati tab i vratiti se na stranicu za registraciju

Korisnik unosi pogrešnu lozinku ili korisničko ime

* Aplikacija upozorava korisnika i omogućuje mu prepravak

**2.2.3 Kreiranje i pregled kompanija**

**Primarni aktori:** Korisnik

**Stakeholders:**

* Baza

**Precondition:** Sustav je pokrenut, korisnik je prijavljen u sustav

**Postcondition:** Nova kompanija je kreirana

**Glavni uspješni scenarij:**

* Korisnik otvara aplikaciju i odlazi na prikaz svih kompanija
* Vidljive su mu postojeće kompanije
* Ako želi dodati novu, unosi podatke o novoj kompaniji
* Korisnik pritišće gum za dodavanje

**Alternativni scenarij:**

U svakom trenutku korisnik može odustati i ne dodati novu kompaniju

**2.2.4 Kreiranje i pregled projekata**

**Primarni aktori:** Korisnik

**Stakeholders:**

* Kompanija
* Baza

**Precondition:** Sustav je pokrenut, korisnik je prijavljen u sustav, korisnik je član neke kompanije

**Postcondition:** Novi projekt je kreiran

**Glavni uspješni scenarij:**

* Korisnik otvara ekran za kreiranje i pregled projekata
* Postojeći projekti su prikazani na ekranu
* Korisnik upisuje ime i opis novog projekta
* Korisnik pritišće gumb za spremanje novog projekta
* Novi projekt uspješno je spremljen

**Alternativni scenarij:**

U svakom trenutku korisnik može odustati od kreiranja novog projekta, te se bazni UC završava.

* Korisnik otvara ekran za kreiranje i pregled projekata
* Postojeći projekti su prikazani na ekranu
* Korisnik upisuje ime i opis novog projekta
* Ime novog projekta već postoji u popisu projekata
* Korisniku se javlja pogreška
* Novi projekt nije spremljen

**2.2.5 Kreiranje sprintova**

**Primarni aktori:** Korisnik

**Stakeholders:**

* Kompanija
* Projekt
* Baza

**Precondition:** Sustav je pokrenut, korisnik je prijavljen u sustav, korisnik je član neke kompanije

**Postcondition:** Novi sprint je kreiran

**Glavni uspješni scenarij:**

* Korisnik otvara ekran za kreiranje i pregled sprintova u projektu
* Postojeći sprintovi unutar projekta su prikazani na ekranu
* Korisnik upisuje ime i opis novog sprinta
* Korisnik pritišće gumb za spremanje novog sprinta u projektu
* Novi sprint uspješno je spremljen

**Alternativni scenarij 1:**

U svakom trenutku korisnik može odustati od kreiranja novog sprinta, te se bazni UC završava.

* Korisnik otvara ekran za kreiranje i pregled projekata
* Postojeći sprintovi u projektu su prikazani na ekranu
* Korisnik upisuje ime i opis novog sprinta
* Ime novog sprinta već postoji u popisu sprintova na projektu
* Korisniku se javlja pogreška
* Novi sprint nije spremljen

**Alternativni scenarij 2:**

* Korisnik otvara ekran za kreiranje i pregled sprintova u projektu
* Ne postoji niti jedan projekt unutar kompanije
* UC završava

**2.2.6 Kreiranje zadataka**

**Primarni aktori:** Korisnik

**Sudionici:**

* Kompanija
* Projekt
* Sprint
* Baza

**Precondition:** Sustav je pokrenut, korisnik je prijavljen u sustav, korisnik je član neke kompanije

**Postcondition:** Novi zadatak je kreiran

**Glavni uspješni scenarij:**

* Korisnik otvara ekran za kreiranje i pregled zadataka u sprintu
* Postojeći zadaci unutar sprinta su prikazani na ekranu
* Korisnik upisuje ime, opis i aspekt novog zadatka
* Korisnik pritišće gumb za spremanje novog zadatka u sprintu
* Novi zadatak uspješno je spremljen unutar sprinta

**Alternativni scenarij :**

U svakom trenutku korisnik može odustati od kreiranja novog zadatka, te se bazni UC završava.

* Korisnik otvara ekran za kreiranje i pregled zadataka
* Postojeći zadaci u projektu su prikazani na ekranu
* Korisnik upisuje ime i opis novog zadatka
* Ime novog zadatka već postoji u popisu zadataka na projektu
* Korisniku se javlja pogreška
* Novi zadatak nije spremljen

**2.2.7 Kreiranje i pregled timova**

**Primarni aktori:** Korisnik

**Sudionici:**

* Kompanija

***Preduvjet:*** Sustav je pokrenut, korisnik je prijavljen u sustav, korisnik je član neke kompanije

***Završni uvjet:*** Novi tim je kreiran

***Glavni uspješni scenarij:***

* Korisnik otvara ekran za kreiranje i pregled timova unutar njegove kompanije
* Postojeći timovi su prikazani na ekranu
* Korisnik unosi ime tima
* Korisnik pritišće gumb za stvaranje novog tima
* Novi tim je uspješno spremljen unutar kompanije

***Alternativni scenariji:***

U svakom trenutku korisnik može odustati od kreiranja novog zadatka

Korisnik može pokušati unijeti tim istog imena kao tim koji već postoji

* Korisnika se obavještava o pogrešci

**2.2.8 Kreiranje i pregled aspekata**

**Primarni aktori:** Korisnik, sprint

**Preduvjet:** Sustav je pokrenut, korisnik je prijavljen u sustav, napravljen je sprint nekog tima u kojem je korisnik član

**Završni uvjet:** Novi aspekt unutar sprinta je kreiran

**Glavni uspješni scenarij:**

* Korisnik otvara ekran za kreiranje i pregled aspekata unutar sprinta projekta
* Postojeći aspekti sprinta su prikazani na ekranu
* Korisnik unosi ime novog aspekta
* Korisnik pritišće gumb za spremanje novog aspekta sprinta
* Novi aspekt je uspješno unesen u listu aspekata sprinta

**Alternativni scenarij:**

Korisnik i bilo kojem trenutku u mogućnosti prekinuti unošenje novog aspekta

Korisnik može unijeti aspekt s imenom koje već postoji

* Korisniku se nakon pritiska gumba za unos prikazuje pogreška

**2.2.9 Komentiranje zadataka**

***Primarni aktori:*** Korisnik, comment, task

***Preduvjet:*** Sustav je pokrenut, korisnik je prijavljen u sustav, korisnik je član neke kompanije ili tima u kojem postoji zadatak

***Završni uvjet:*** Novi komentar je objavljen

***Glavni uspješni scenarij:***

* Korisnik otvara ekran na kojem je prikazan zadatak i ostali komentari
* Korisnik unosi novi komentar u za to predviđeno mjesto
* Korisnik pritišće gumb preko kojega se objavljuje komentar
* Komentar je objavljen i spremljen u komentarima tog zadatka

***Alternativni scenarij:***

Korisnik u bilo kojem trenutku prije pritiska gumba za objavu komentara može odustati od objave komentara

Korisnik koji nije dio kompanije unutar koje je zadatak ne može objavljivati komentare za taj zadatak

* Korisnika se obavještava o pogrešci

**2.2.10 Dodavanje reakcija na komentare**

**Primarni aktori:** Korisnik, komentar, reakcija **Preduvjet:** Sustav je pokrenut, korisnik je prijavljen u sustav, korisnik je član neke kompanije ili tima, u kompaniji postoji barem jedan zadatak, u zadatku postoji barem jedan komentar **Završni uvjet:** Reakcija na komentar je objavljena, reakcija može biti jedna od sljedećih: “HAHA”, “SAD”, “ANGRY”, “YAY”, “WOW”, “LIKE” **Glavni uspješni scenarij:**

* Korisnik otvara ekran na kojem je prikazan zadatak i ostali komentari
* Korisnik odabire jedan od komentara i odabire jednu od reakcija kojom će reagirati na komentar
* Korisnik pritišće traženi gumb i ostavlja reakciju na komentar
* Korisnik i ostali korisnici vide reakciju koju je korisnik ostavio

**Alternativni scenariji:**

* Korisnik pokušava reagirati na komentar koji nije unutar njegove tvrtke, dobiva obavijest o pogrešci
* Korisnik odustaje od reakcije na komentar

**2.2.11 Pregled zadataka po “tagovima” i “skillovima”**

**Primarni aktori:** Korisnik, tag, skill **Preduvjet:** Sustav je pokrenut, korisnik je prijavljen u sustav, korisnik je član neke kompanije ili tima **Završni uvjet:** Korisnik dobiva prikaz (listu) svih zadataka koji odgovaraju kriteriju traženja

**Glavni uspješni scenarij:**

* Korisnik otvara ekran na kojem može odabrati tag ili skill po kojem želi pretraživati
* Nakon odabira, korisnik dobiva filtriranu listu svih zadataka prema kriterijima koje je odabrao

**Alternativni scenariji:**

* Korisnik pokušava pregledati zadatke koji nisu unutar njegove tvrtke, dobiva obavijest o pogrešci
* Korisnik odustaje od pretrage zadataka
* Korisnik odabire kriterije koje niti jedan zadatak ne zadovoljava, dobiva praznu listu zadataka
* Korisnik odabire kriterije koje svaki zadatak zadovoljava, dobiva početnu listu koju je pokušao filtrirati

**2.2.12 Dodavanje novih korisnika u kompaniju i timove**

**Primarni aktori:** Korisnik koji je član kompanije, korisnik koji nije član kompanije, kompanija, timovi

**Preduvjet:** Sustav je pokrenut, korisnik je prijavljen u sustav, korisnik koji pokušava dodati novog korisnika je član kompanije ili tima u koji dodaje korisnika, korisnik kojeg želimo dodati u kompaniju nije član niti jedne kompanije **Završni uvjet:** Novi korisnik postaje član ciljane kompanije **Glavni uspješni scenarij:**

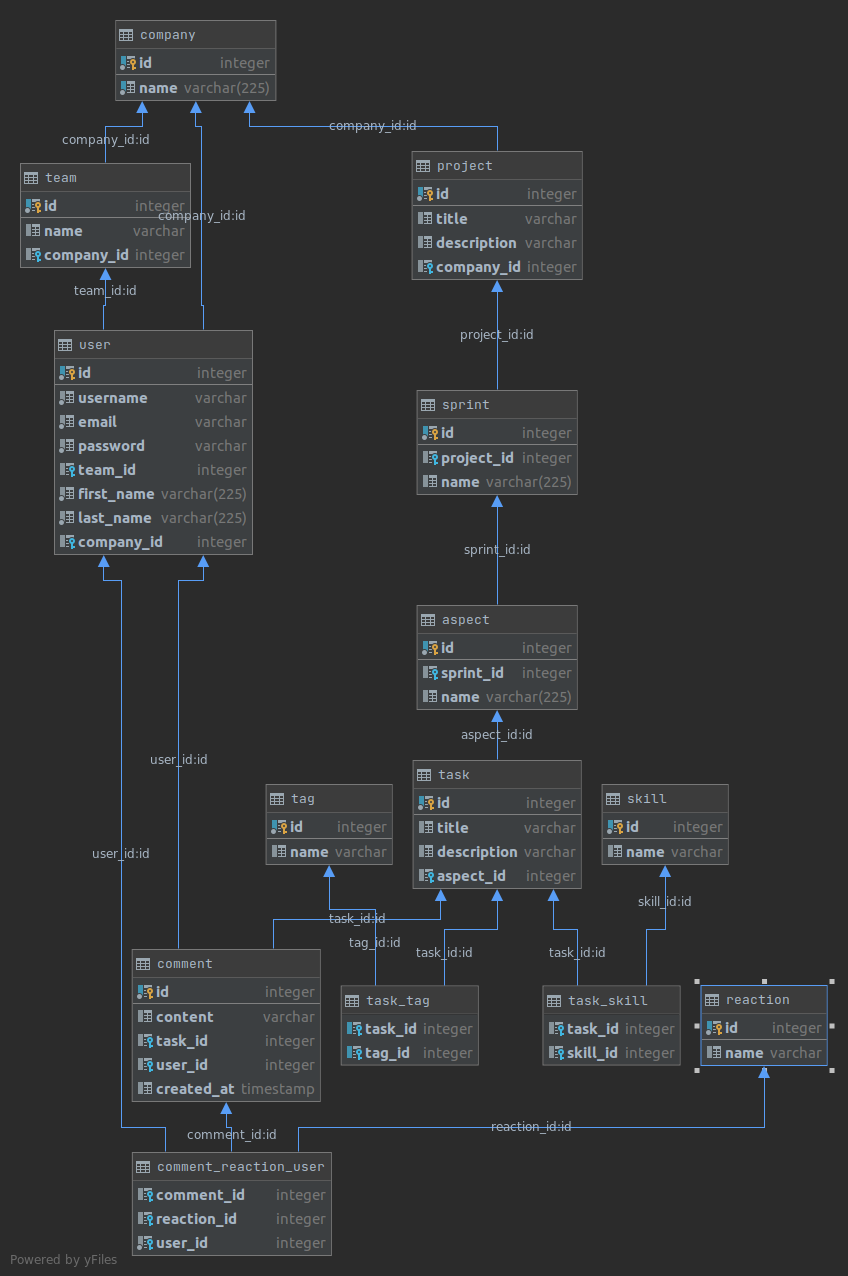
* Korisnik koji je član kompanije dolazi na ekran za dodavanje novih članova u kompaniju
* Korisnik pretražuje željenog korisnika po korisničkom imenu
* Korisnik odabire željenog korisnika
* Korisnik pritišće gumb za dodavanje novog korisnika u kompaniju
* Novi korisnik postaje član kompanije

**Alternativni scenarij:**

* Ne postoji niti jedan korisnik koji nema kompaniju, samim time nije moguće dodati korisnika dok jedan od korisnika ne ukloni svoju prijašnju kompaniju
* Korisnik odustaje od dodavanje novog korisnika

## 3. **Opis objektnog modela**

**3.1. Model baze podataka**



### 3.2. Dijagram klasa kontrolera

### 3.3 Opis implementacije perzistencije

Svaki entitet u bazi podataka ima 1-1 mapiranje s kontrolerom i repozitorijem. Ako imamo entitet *user* to znači da postoji *UsersController* i *UsersRepository* koji izlažu sučelje prema kontroleru odnosno klijentu.

Jednostavnim grafom tu vezu možemo prikazati na idući način:

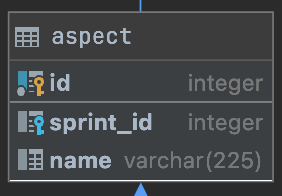


### Aspect

**Dijagram razreda**

****

**Tablica u bazi podataka**

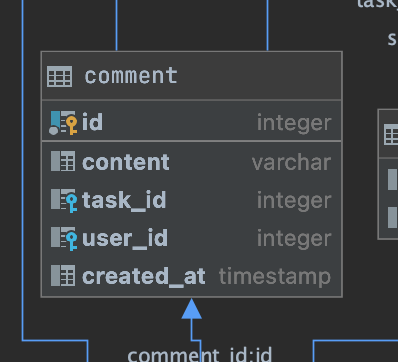
****

Comment

Dijagram razreda:

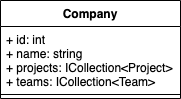


Tablica u bazi podataka:

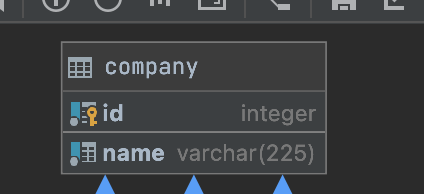


Company

Dijagram razreda:

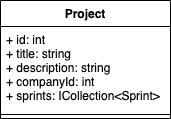


Tablica u bazi podataka:

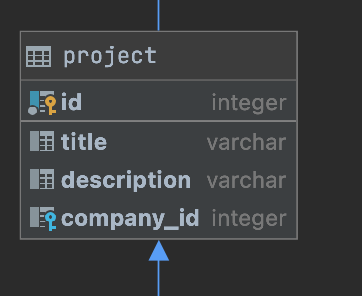


Project

Dijagram razreda:

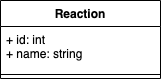


Tablica u bazi podataka:

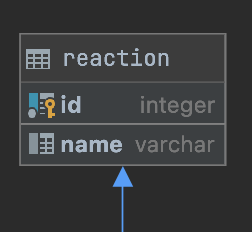


Reaction

Dijagram razreda:



Tablica u bazi podataka:

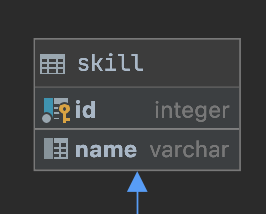


Skill

Dijagram razreda:

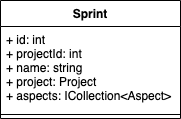


Tablica u bazi podataka:

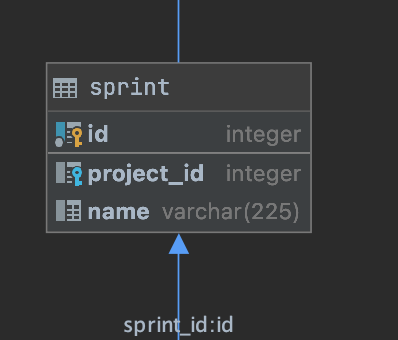


Sprint

Dijagram razreda:



Tablica u bazi podataka:

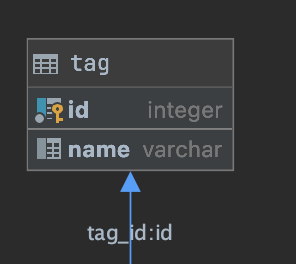


Tag

Dijagram razreda:



Tablica u bazi podataka:

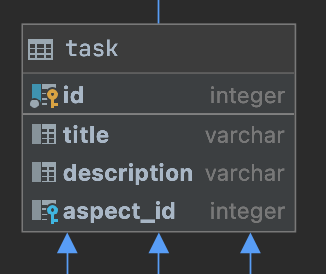


Task

Dijagram razreda:



Tablica u bazi podataka:

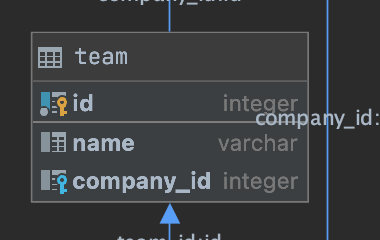


Team

Dijagram razreda:

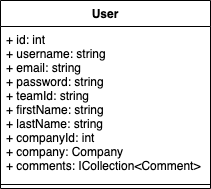


Tablica u bazi podataka:

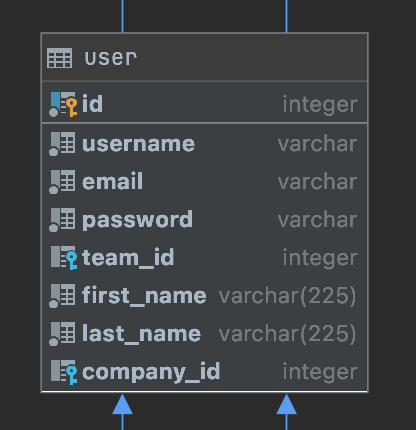


User

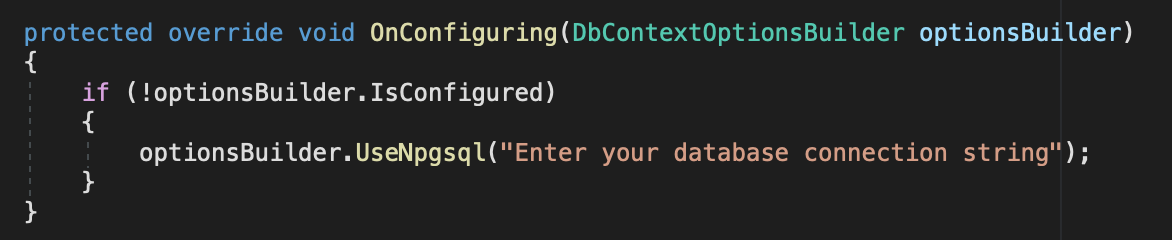
Dijagram razreda:



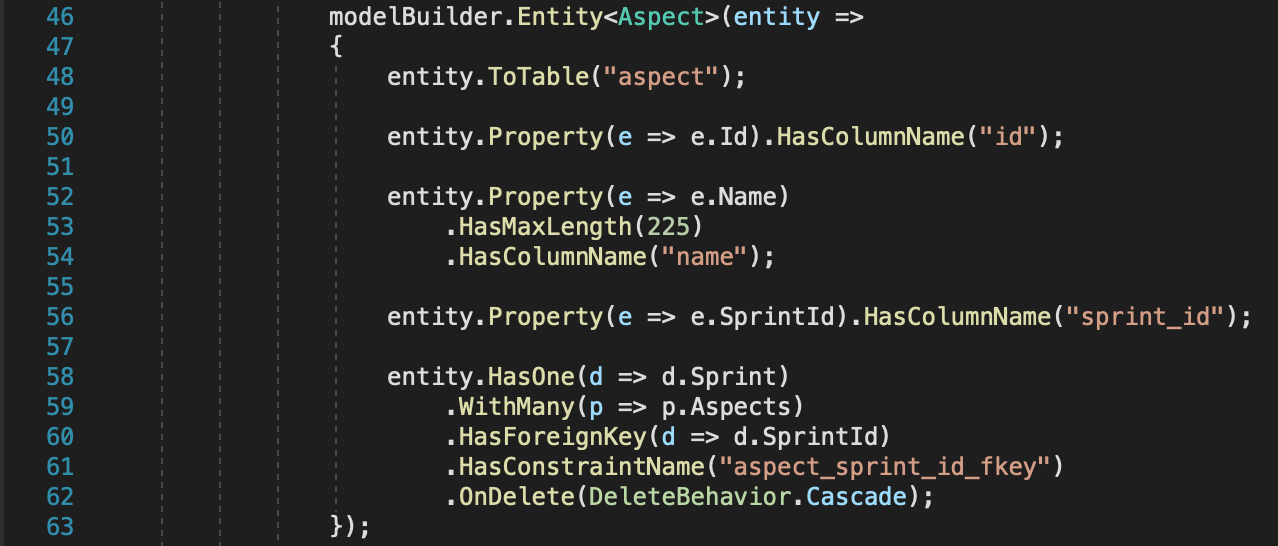
Tablica u bazi podataka:



U Entity Frameworku se O/R mapiranje radi na idući način: Definiramo poseban razred *DatabaseContext* koji nasljeđuje *Microsoft.EntityFrameworkCore.DbContext* i overrideamo metodu *OnConfiguring(DbContextOptionsBuilder optionsBuilder)* i *OnModelCreating(ModelBuilder modelBuilder).*U metodi *OnConfiguring()* se moramo spojiti na bazu podataka i odabrati koju bazu podataka koristimo. U našem slučaju koristimo PostgreSQL.



U metodi *OnModelCreating()* definiramo O/R mapiranja. U nastavku je dan primjer mapiranja entiteta *Aspect*.

Linija 48: entity.toTable(“aspect”);  
Mapiramo klasu *Aspect* na tablicu *“aspect”* u bazi podataka.

Linija 52:: entity.Property(e => e.Id).HasColumnName(“id”);  
Mapiramo field “name” na stupac *“*name”u tablici “aspect” u bazi podataka. Također možemo definirati i dodatna ograničenja kao što su duljina naziva.  
  
Linija 59:

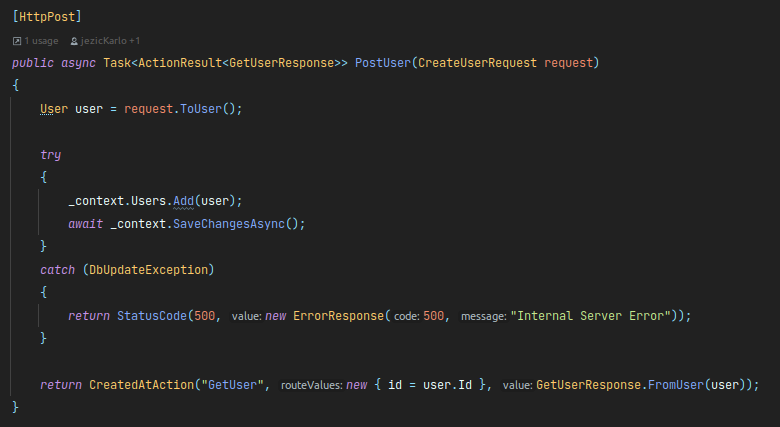
Definiramo 1-N mapiranje između tablice “sprint” i “aspect” što znači da 1 sprint sadrži više aspekata. Također dodajemo Foreign Key koji znači da se “sprintId” u tablici “aspect” odnosi na “id” u tablici “sprint”. Također definiramo ponašanje kada se obriše “roditeljski” entitet. U našem slučaju to je CASCADE i tada će se obrisati i svi entiteti djeca.

### 3.4. Transakcije i unit of work

Transakcije omogućuju da se više operacija nad bazom podataka odviju atomarno. To znači da su ili sve operacije unutar jedne transakcije uspješno provedene ili niti jedna nije provedena.

U našem slučaju jedna transakcija je sve ono što se odvije prilikom jednog korisničkog zahtjeva te završava naredbom SaveChanges(). SaveChanges() sama po sebi osigurava atomarnost, no mi, da bi bili još sigurniji pozivamo metodu SaveChanges() nakon svake promjene u bazi podataka jer smo zaključili da je takvih operacija malo i da nam taj tradeoff najbolje ide u korist.

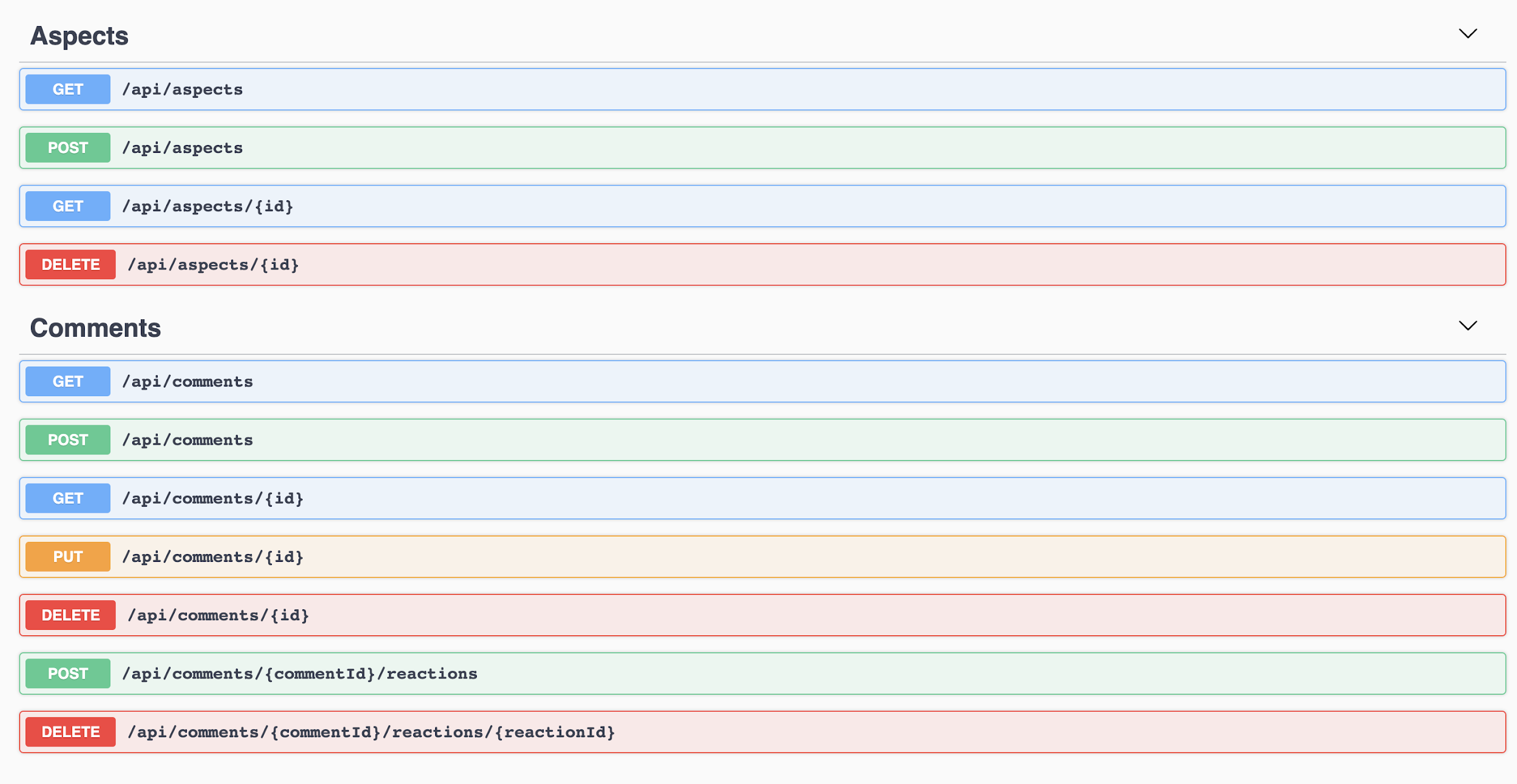
Unit of work je pattern koji nam omogućuje da se više operacija nad bazom podataka (insert/update/delete) odvija kao da je u pitanju jedna transakcija. U našem slučaju svaka je transakcija sama po sebi već napravljena prema Unit of work pattern-u jer je nakon svake operacije nad bazom podataka (koja nije dohvaćanje) pozvana metoda SaveChanges() kojom se završava transakcija.

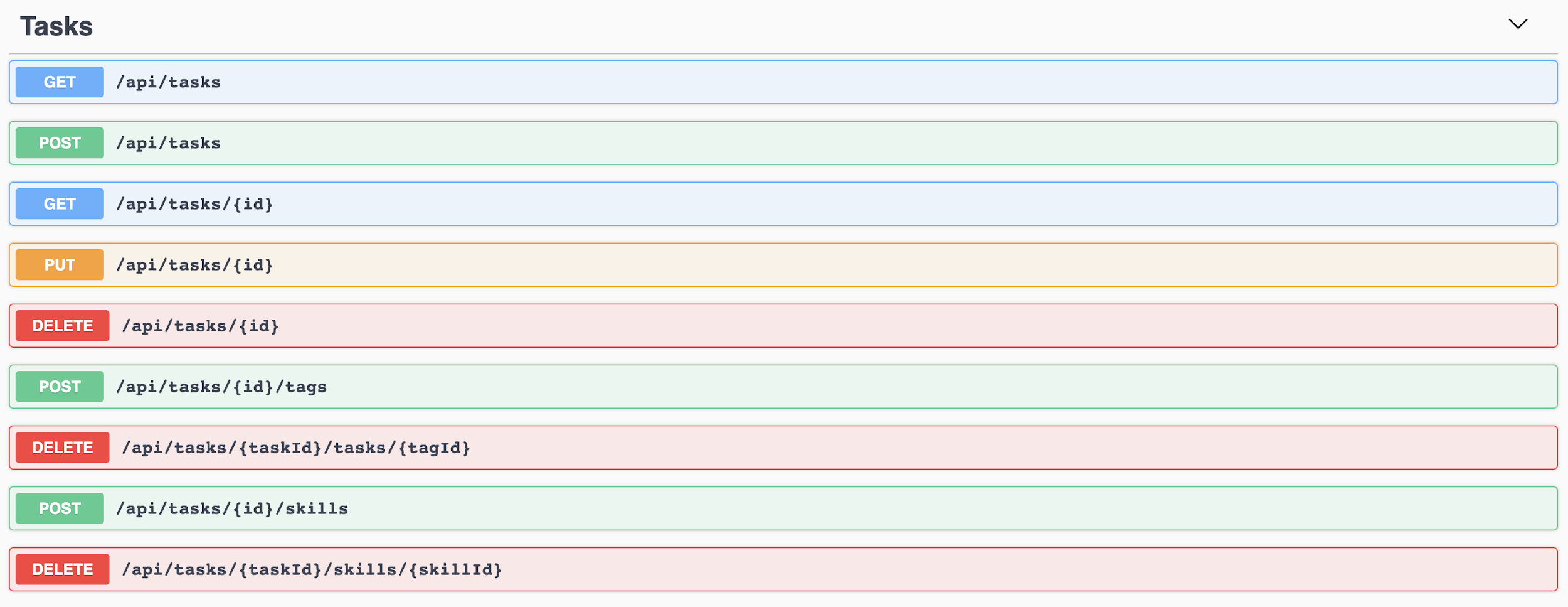


Kao primjer možemo vidjeti PostUser() methodu UsersController-a (slika iznad) koja služi za registraciju korisnika u sustav. Vidimo da je nakon metode Add(), koja unosi novog korisnika u bazu podataka, pozvana metoda SaveChangesAsync() koja zapravo provodi to što je napravljeno liniju prije.

Proučavanjem ovog primjera vidljivo je da je u našem slučaju 1 request = 1 transaction = 1 unit of work.

## 4. Backend i opis pozadinskog sučelja





## Uvjeti

* Nakon registracije korisnik se pridružuje kompaniji
* Nakon pridruživanja kompaniji, korisnik se pridružuje timu
* Svaki tim se sastoji od jednog ili više korisnika, a jedan korisnik u bilo kojem trenutku može pripadati isključivo jednom timu.
* Svaka tvrtka se sastoji od više timova, a jedan tim pripada isključivo jednoj tvrtki.
* Svaka tvrtka može imati jedan ili više projekata.
* Svaki projekt može imati jedan ili više sprintova.
* Svaki sprint se sastoji od jednog ili više aspekata.
* Svaki aspekt se sastoji od jednog ili više zadataka.
* Svaki zadatak na sebi može imati jedan ili vise komentara, a svaki komentar moze imati nula ili vise reakcija (“LIKE”, “HAHA”, “WOW”...).
* Svaki zadatak ima jedan ili više skillova koji mu se mogu pridijeliti, npr. zadatak za koji je potrebno poznavanje programskog jezika Ruby će imati skill ‘Ruby’.

## Prikaz ključnih modela

Kompanija



Projekt



Korisnik



Timovi



## 5. Desktop inačica

## MVVP obrazac u GUI sučelju aplikacije

U prošlosti, bila je česta praksa imati *fat client* desktop aplikacije. Ovakav tip aplikacije sadržavao je velike količine poslovne logike. Također, bila je česta praksa da se ovakve aplikacije spajaju izravno na bazu podataka te upravljaju spremištem. Ovo je nepoželjno iz mnogo očitih razloga. Jedini benefit diskutabilno može biti brzina, obzirom da je velika količina posla obavljena lokalno. Usprkos tome, zaključilo se da ovakav tradeoff nije vrijedan. Gubi se skalabilnost, povezanost, otpornost na kvarove, apstrakcija, ponovna iskoristivost…

Sukladno novim standardima i učestalim praksama, odlučili smo implementirati desktop putem MVVP (MVVM) obrasca. Ovaj obrazac temelji se na tri ključne jedinice upravljanja podacima i logikom.

Relevantne jedinice su:

* Model
* View
* View Presenter (View Model)

Svaka od ovih jedinica ima vlastite odgovornosti i zadaće. U sljedećem ulomku te odgovornosti razložene su po jedinicama.

## Model

Implementacija domenske logike. Ni na koji način ne ovisi o korisničkom sučelju. Ovaj uvjet mora biti ispunjen, u suprotnom gubimo mnoga korisna svojstva ovog obrasca.

## View

Implementacija korisničkog sučelja. Sastoji se od deklarativnih opisa korisničkog sučelja, često napisanih u nekom od markup jezika. Iz ovih opisa se u konačnici generira prikaz s kojim korisnik interagir. Uz ove opise nalazi se i naredbeni kod koji poziva aplikacijski kod pri kojem se izvodi aplikacijska logika. View često sadrži kuke (hooks) na koje se moguće spojiti raznim action listenerima i sličnim mehanizmima.

## View-model

Međuklase koje orkestriraju interakcije između prikaza i modela. Mogu se promatrati kao svojevrsni adapteri (prilagodnici) za preostale dvije komponente.

## Tusk-desktop + MVVP

## 

Način na koji smo mi implementirali MVVP obrazac/arhitekturu jes kroz *.resx scheme* (View),

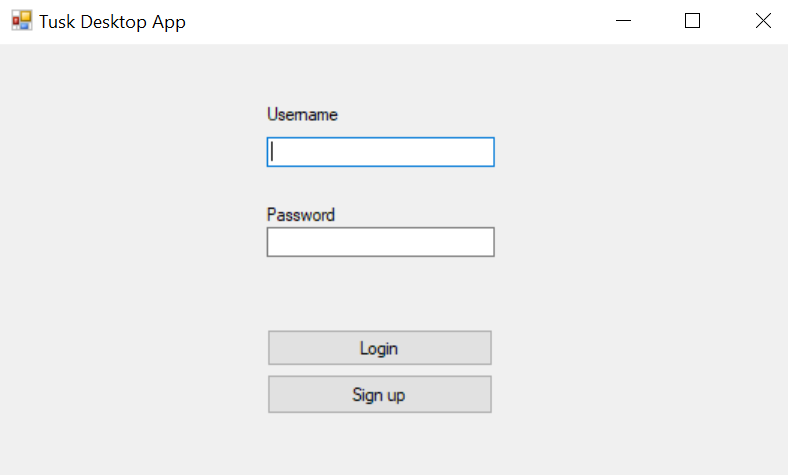
*.cs* hookove (View-Model), te pozive na *REST API* deployan na vanjskom resursu (Model).

Kao reprezentativni primjer analizirajmo *Sign Up use-case*.

1. Korisnik upisuje podatke preko grafičkog sučelja i prosljeđuje podatke aplikaciji putem submit gumba, razina apstrakcije - view
2. Pritiskom na gumb, radni okvir okida pozivanje funkcije za obradu pristignutih podatka, razina apstrakcije - middleware
3. Podatci se pakiraju i šalje se zahtjev na REST API, razina apstrakcije - view-model
4. Zahtjev se obrađuje na vanjskom poslužitelju, potencijalno se dodatno šalje zahtjev za spremanje na vanjski posluživanu bazu podataka (implementacijski ovisno), razina apstrakcije - model
5. Odgovor o uspješnosti izvršenog zahtjeva se vraća s REST-a na aplikacijsku logiku
6. Aplikacijska logika interpretira odgovor, te postupa sukladno potrebama aplikacije
7. Rezultat ukupne akcije prikazan je korisniku na korisničkom sučelju

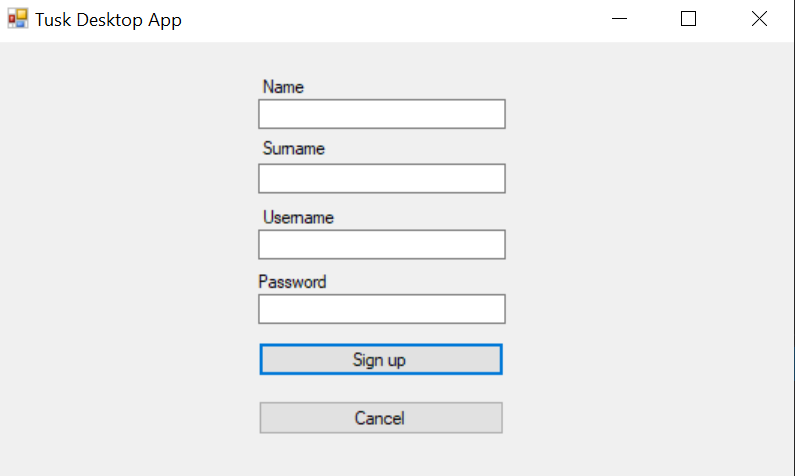
## 

## Glavne forme



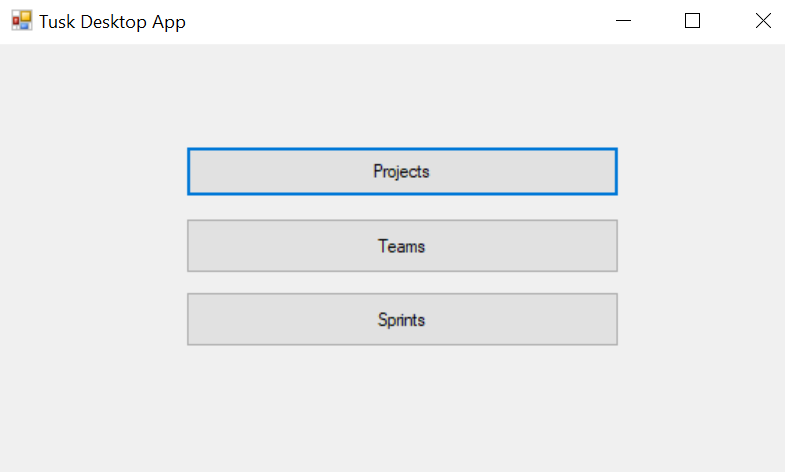
Forma za prijavu

Forma za prijavu omogućuje korisniku da se prijavi u aplikaciju putem korisničkog imena i lozinke. U slučaju da korisnik nema otvoreni korisnički račun, moguće ga je izraditi u formi za registraciju.



Forma za registraciju

Forma za registraciju omogućuje korisniku da izradi korisnički račun.

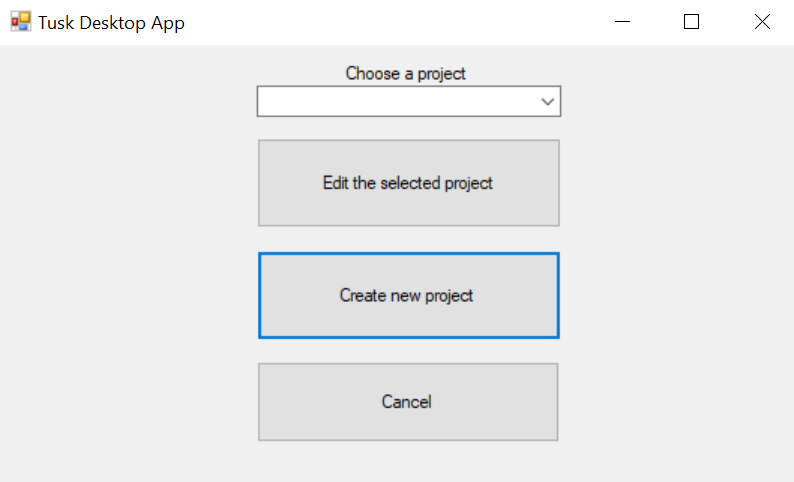


Forma glavnog izbornika

Forma glavnog izbornika predstavlja središnju formu aplikacije koja služi za navigiranje prema ostatku aplikacije. Dakle, iz te središje forme korisnik pristupa formama za upravljanje projektima, timovima i sprintevima.

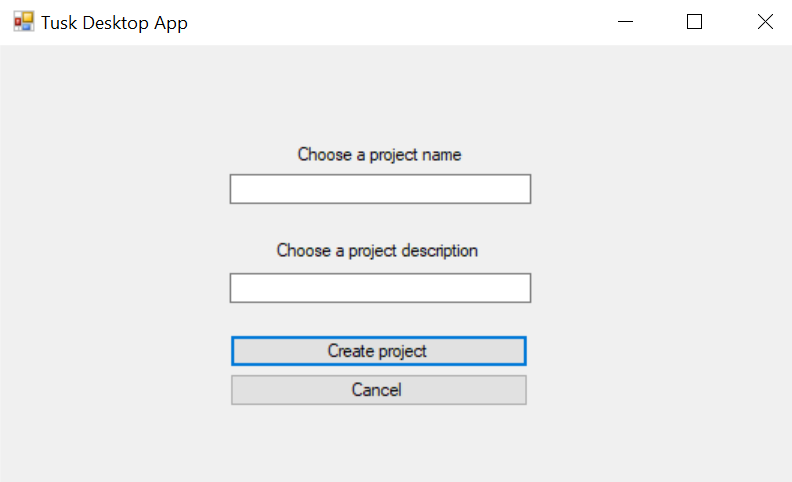
## 

## Forme za upravljanje projektima



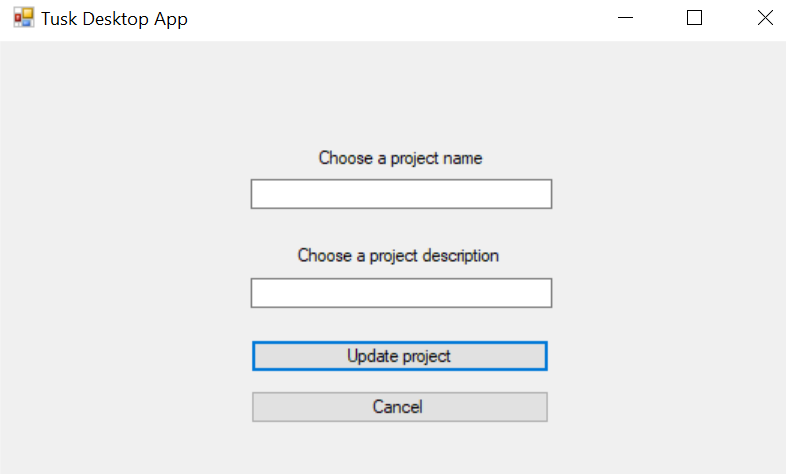
Središnja forma za upravljanje projektima

Iz padajućeg izbornika korisnik odabire projekt kojim želi upravljati. Nakon odabira projekta postoji mogućnost uređivanja odabranog projekta ili izrada novog.



Forma za stvaranje projekta

Unosom željenog imena i opisa projekta, svaki korisnik ima mogućosti stvoriti novi projekt.

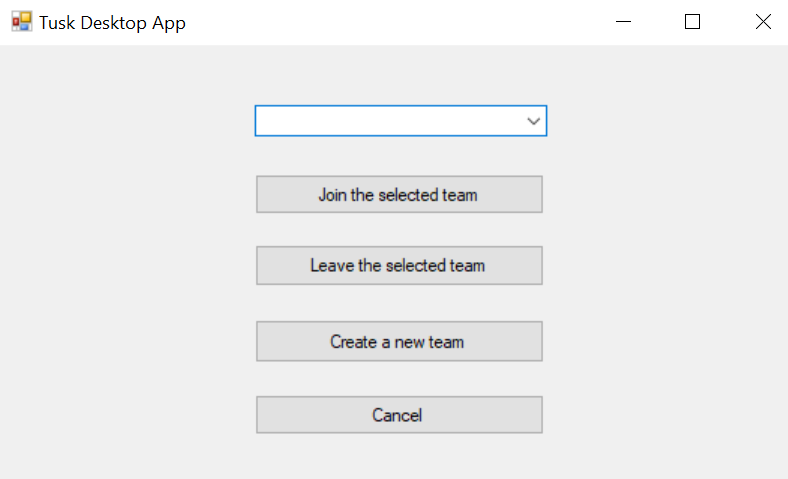


Forma za uređivanje projekta

Forma za uređivanje projekta se izgledom značajno ne razlikuje od forme za izradu projekta no funkcionalnost je zato jasno različita.

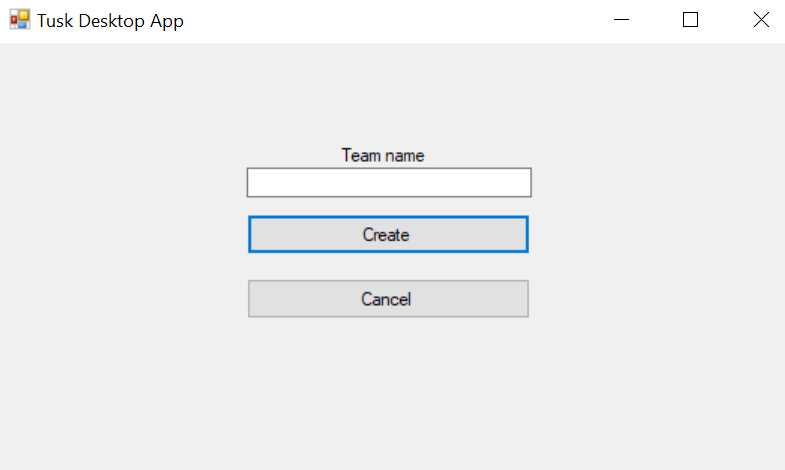
## 

## Forme za upravljanje timovima



Središnja forma za upravljanje timovima

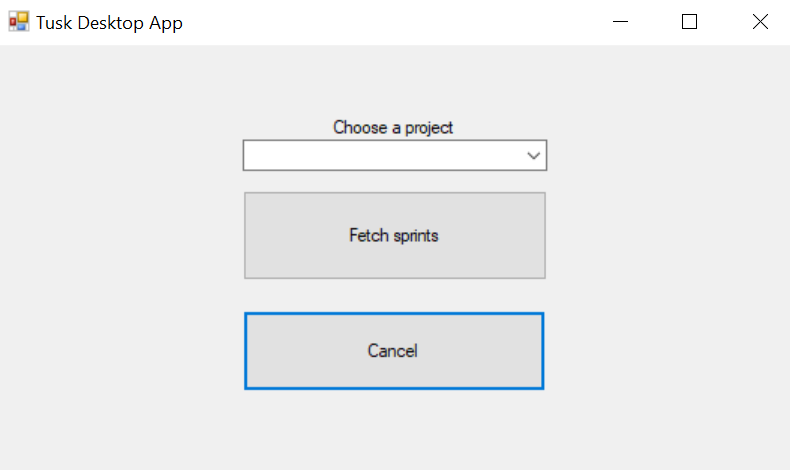
Iz padajućeg izbornika je moguće izabrati neki od postojećih timova. Po odabiru tima je moguće učlaniti se i izaći iz tima ovisno o trenutnom statusu članstva vezanom za dotični tim. Osim toga, moguće je i stvoriti novi tim.



Forma za stvaranje novog tima

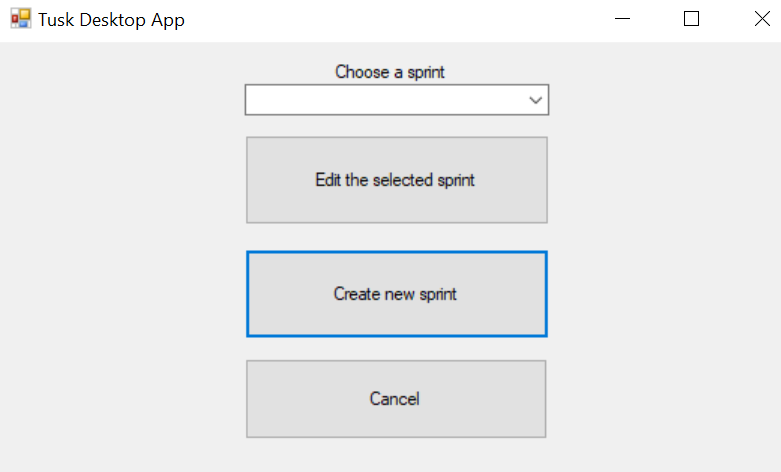
Unosom imena tima, svakom korisniku je omogućeno stvaranje vlastitog tima.

## Forme za upravljanje sprintevima



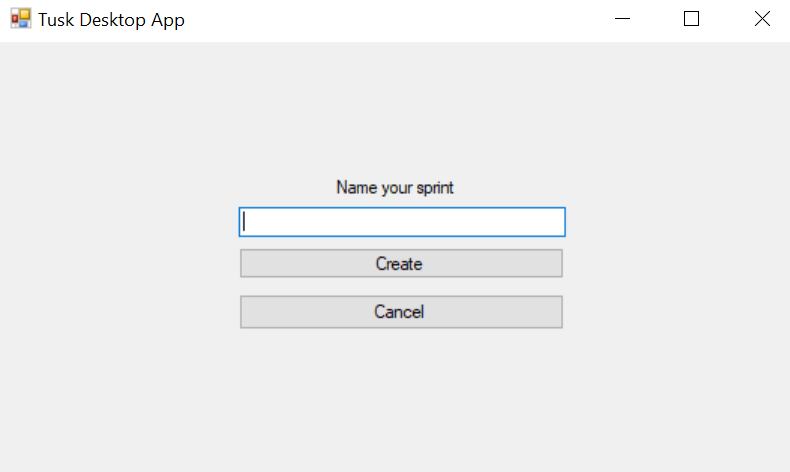
Forma za dohvat sprinteva u projektu

Iz padajućeg izbornika, korisnik bira projekt za koji želi dohvatiti postojeće sprinteve kako bi ih mogao urediti ili dodati novi sprint u odabrani projekt.



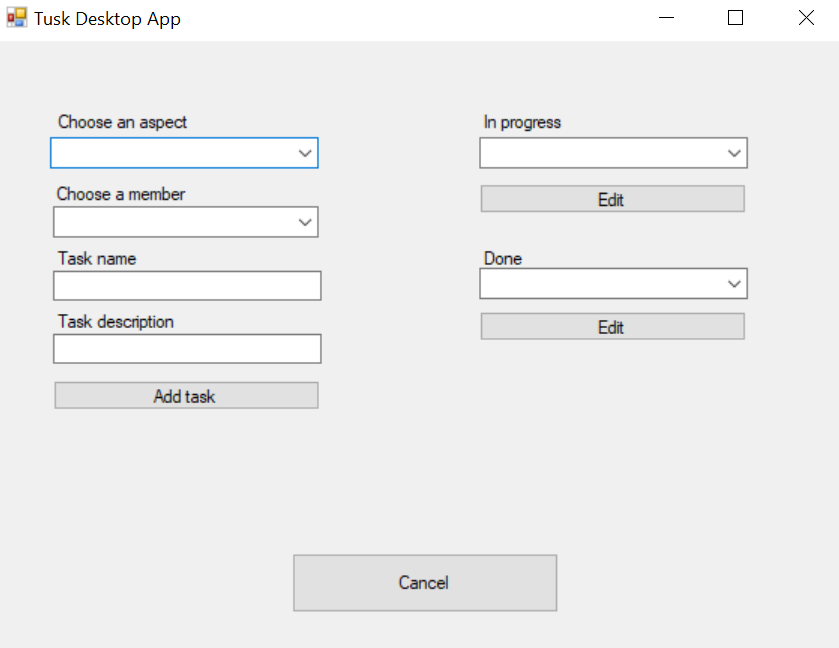
Forma za upravljanje sprintevima

Iz forme za upravljanje sprintevima je moguće urediti sprint odabran iz padajućeg izbornika i dodati novi sprint u prethodno odabrani projekt.



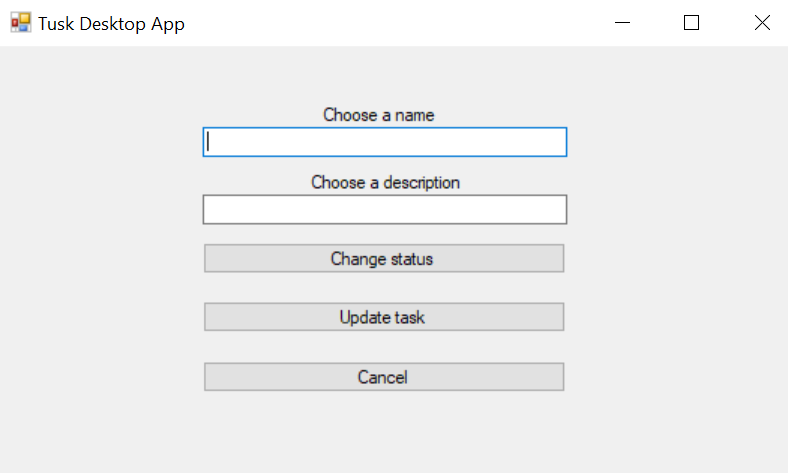
Forma za stvaranje sprinta

Odabirom imena, korisnik ima mogućnost stvoriti novi sprint unutar odabranog projekta.



Foma za upravljanje pojedinostima sprinta

Putem forme za upravljanje pojedinostima sprinta korisnik može preko aspekta dodati nove taskove u sprint, također može i urediti postojeće taskove.



Forma za uređivanje task-a

Putem forme za uređivanje task-a korisnik može mijenjati naziv, opis i status odabranog task-a.

## 6. Mobilna aplikacija iOS i Android

Mobilna aplikacija je izrađena koristeći radni okvir React Native. Zbog toga što je aplikacija izrađena koristeći React Native, u stanju smo napraviti iOS i Android aplikaciju koristeći jedan izvorni kod. React Native u svojoj pozadini koristi Javascript koji se zatim pakira zajedno s Javascript enginom i izvršava na iOS ili Android sustavu.

**Redux** je paket koji se često kombinira s radnim okvirom React i React Native kako bi se lakše pratilo globalno stanje aplikacije. Tako je slučaj i u Tusk aplikacji gdje React koristimo kako bismo spremali sve informacije o korisniku, podacima koji stižu od poslužitelja i o unesenim ili odabranim stavkama unutar aplikacije.

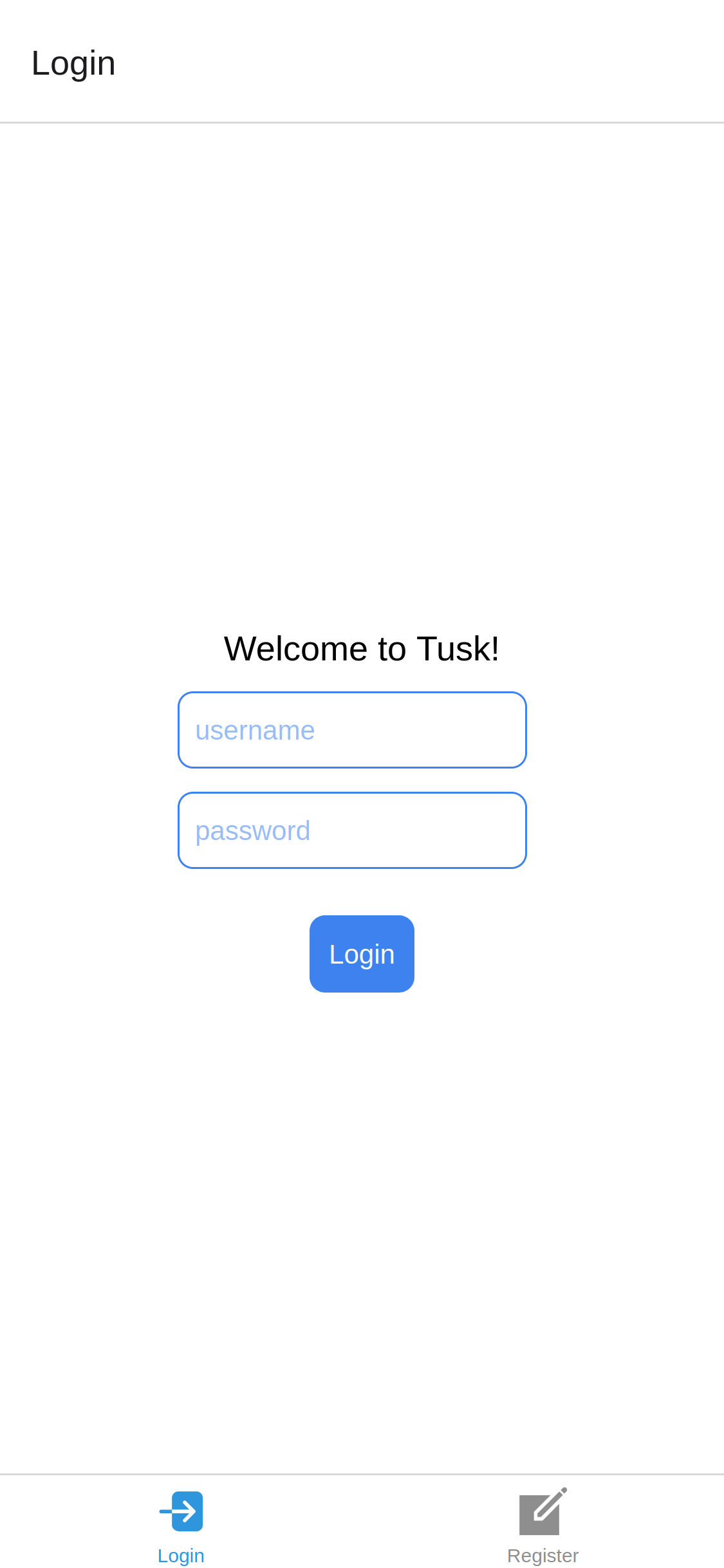
Aplikacija ima nekoliko logičkih cjelina koje rade zajedno.

1. **View** koji prikazuje podatke koje korisnik želi vidjeti, odnosno status nekakvih poziva koji se događaju u pozadini
2. **ViewController** koji zna kako napraviti izmjene u ViewModelu i odgovoran je za korisničke akcije
3. **ViewModel** ima pristup podacima i ovisno o njima donosi odluke u skladu s aplikacijskom biznis logikom
4. **Model** je sloj podataka koji se spremaju i mijenjaju u skladu s pozivima na poslužiteljski server

Aplikacija razlikuje i stanje aplikacije kada je korisnik ušao u sustav nakon uspješne prijave, te mijenja navigaciju i mogućosti koje korisnik može. Nakon uspješne prijave u sustavi, pozivi prema poslužitelju dobivaju dodatni “Authorization” header koji dozvoljava klijentu da radi radnje specifične za njega ili kompaniju kojoj pripada.

## 6.1 Prijava u sustav

Od svakog kornisnika koji prvi puta dolazi u aplikaciju se očekuje da se ili prijavi korištenjem korisničkog imena i lozinke ili da kreira nove pristupne podatke kroz registraciju



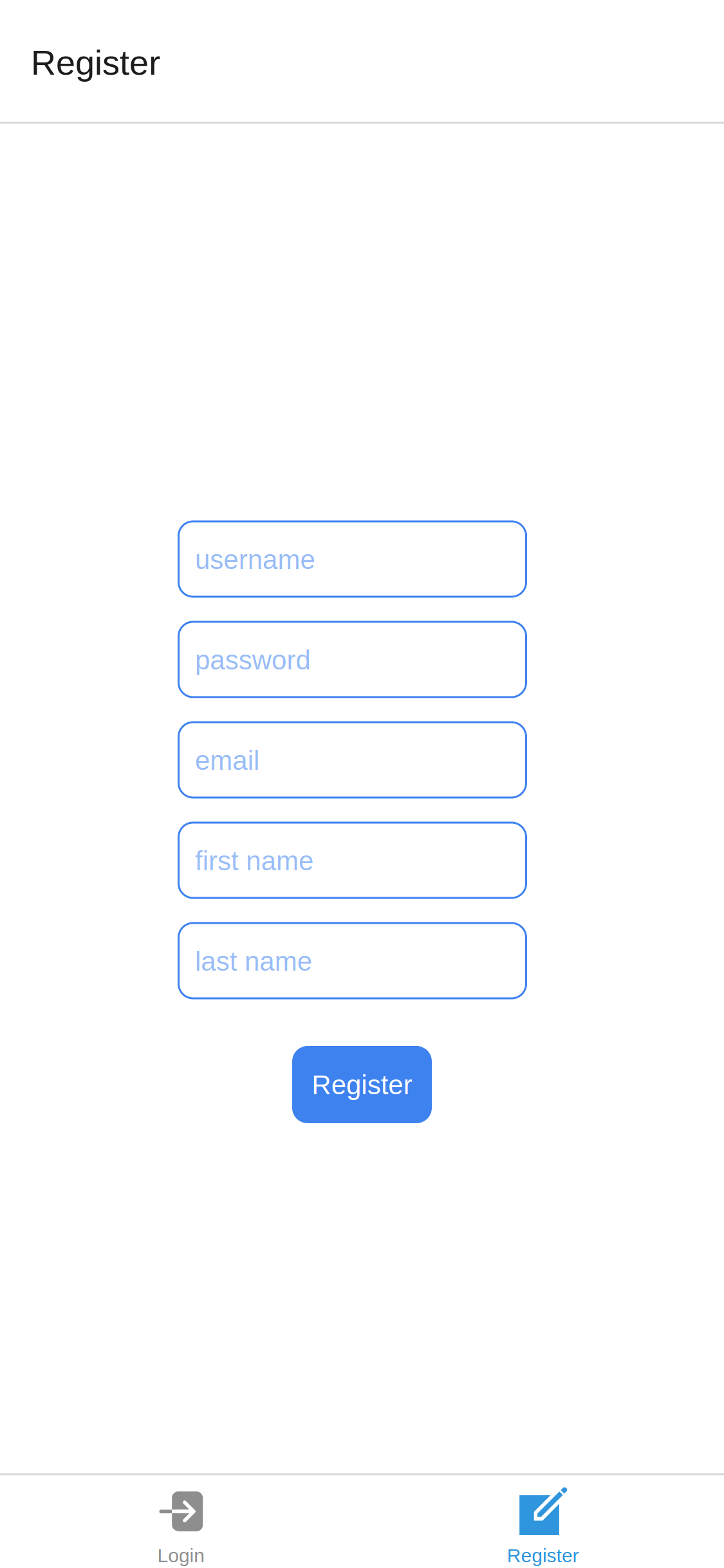
6.1 slika - Prijava u sustav

## 

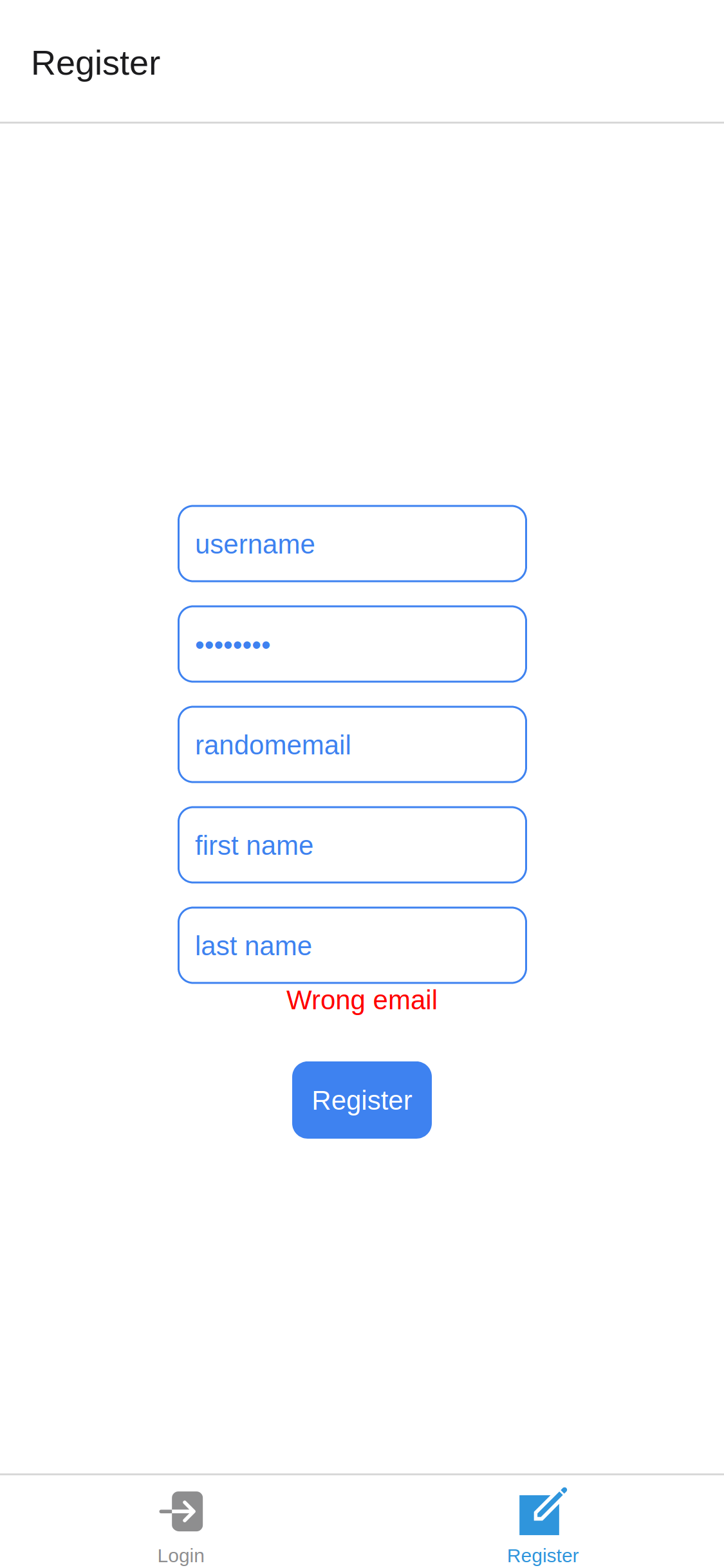
6.2 slika - Pogrešni podaci za prijavu

## 6.2 Registracija u sustav

Ukoliko korisnik nema korisničko ime i lozinku, mora kreirati novi račun odlaskom na Registraciju.



6.3 slika - Podaci koji se očekuju da ih korisnik popuni

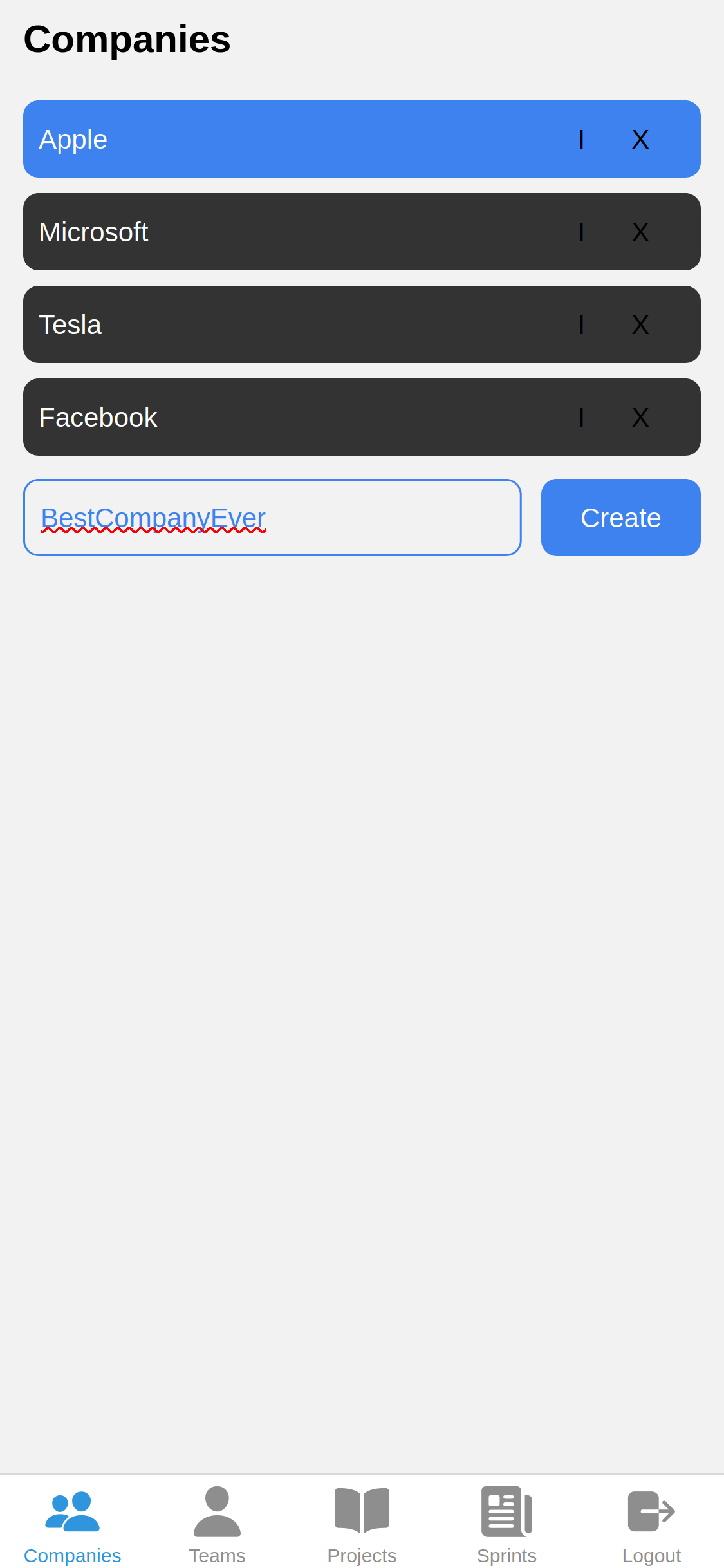


## 

6.4 slika -Pogrešni podaci kod registracije

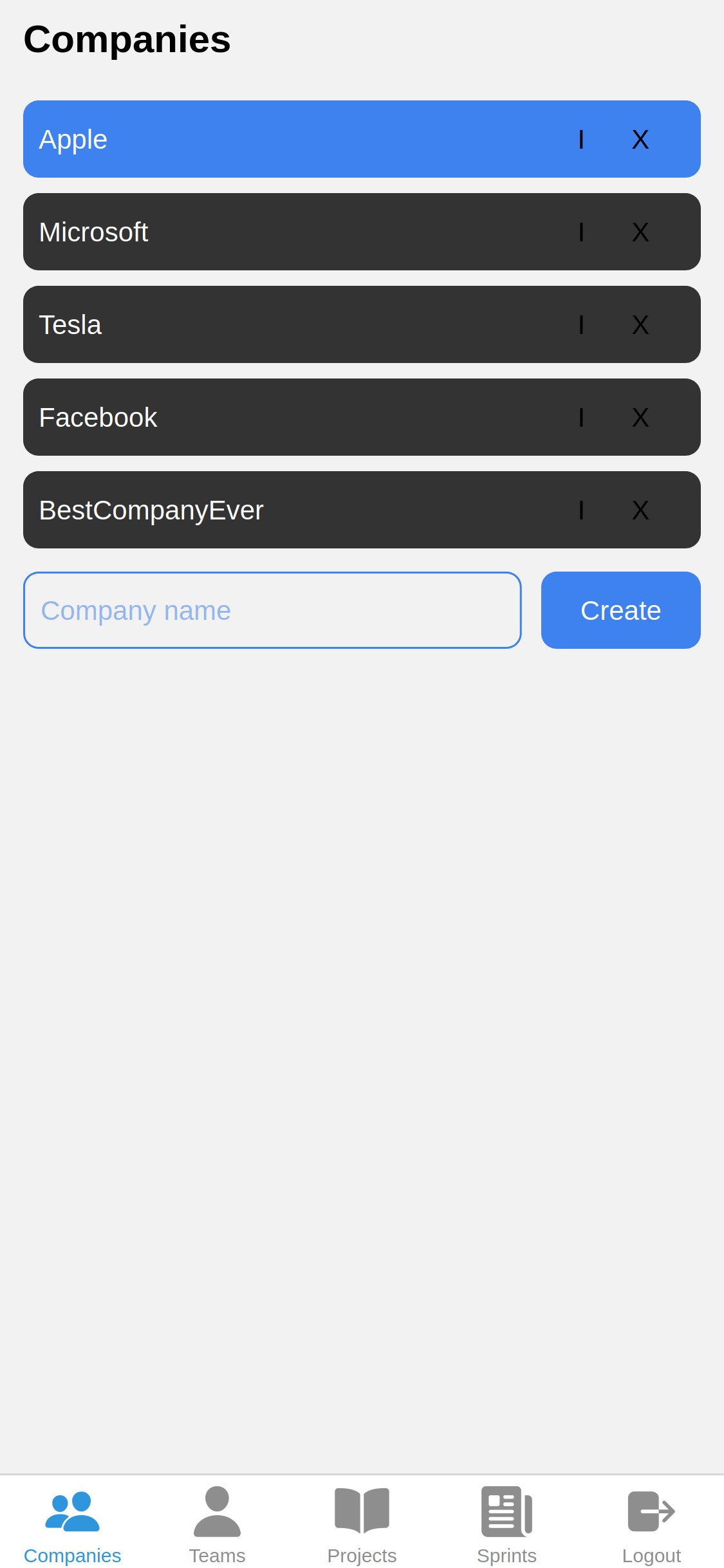
## 6.3 Izrada nove kompanije

Nakon uspješne prijave u sustav, svaki korisnik može odabrati koju kompaniju želi pregledati, odnosno može izraditi novu kompaniju, te mijenjati podatke o postojećima.



6.5 slika - Unos podataka za novu kompaniju

Gumbi “X” i “I” služe za brisanje kompanije i izmjenu imena kompanije.



6.6 slika - Izgled nakon unosa nove kompanije

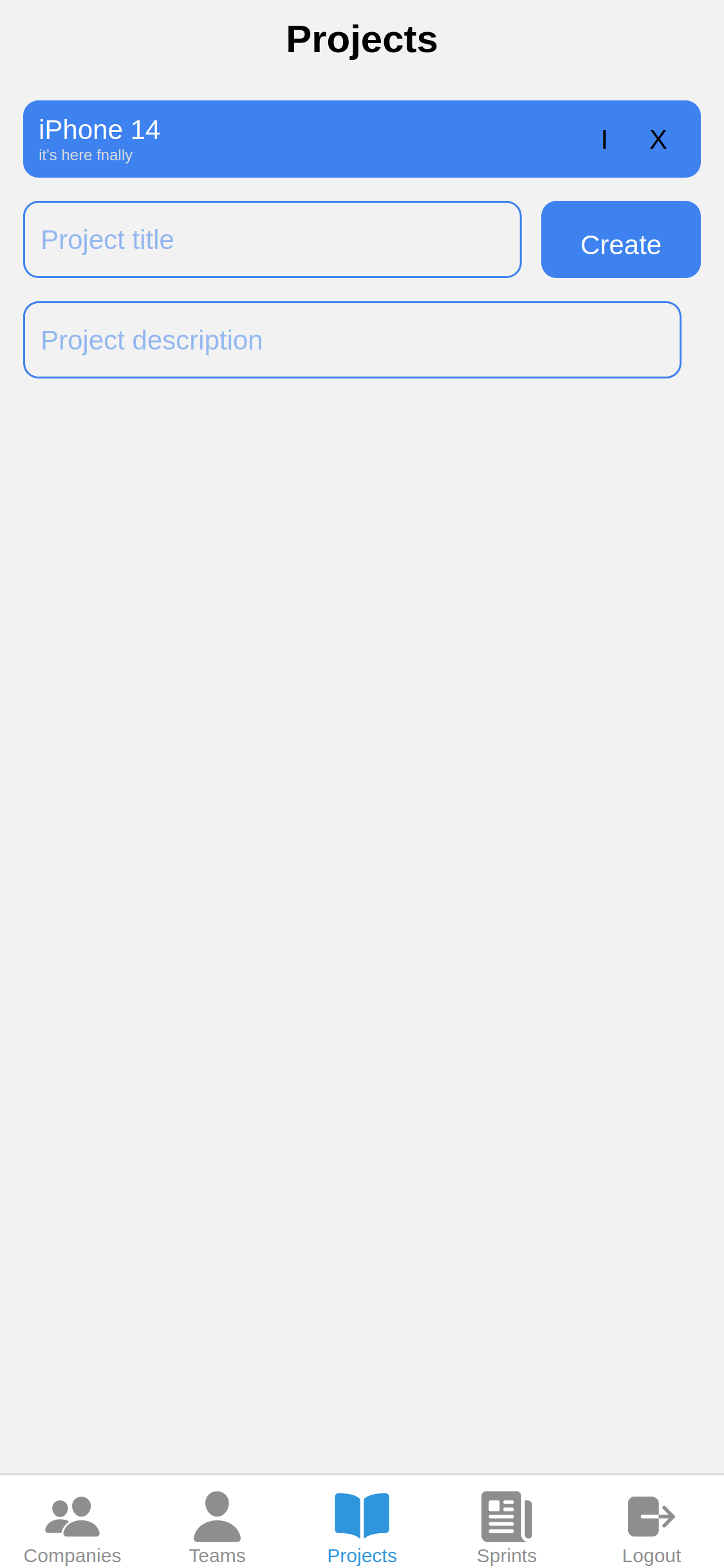
## 6.4 Prikaz i dodavanje novih članova u tim

Odabirom jednog od prikazanih timova, otvara se popis svih članova tima. Kako bi korisnik dodao novog člana tima, može klikom na gumb “Add user” dodati novog člana tima.  
Osim dodavanja novih članova, moguće je brisati, mijenjati ime i kreirati nove timove.

## 6.7 Prikaz “Core tima” 6.7 slika - Dodavanje novog člana u “Core tim”

## 6.5 Prikaz i izrada novih projekata

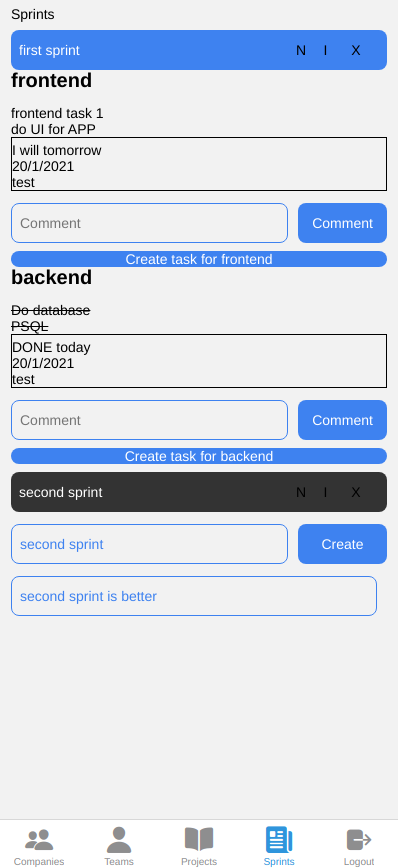
Svaka kompanija ima svoje projekte. Na stranici projekata moguće je raditi nove projekte i izmjene na postojećima.



## 6.8 slika - Prikaz svih projekata jedne kompanije

## 6.6 Sprintevi i zadaci

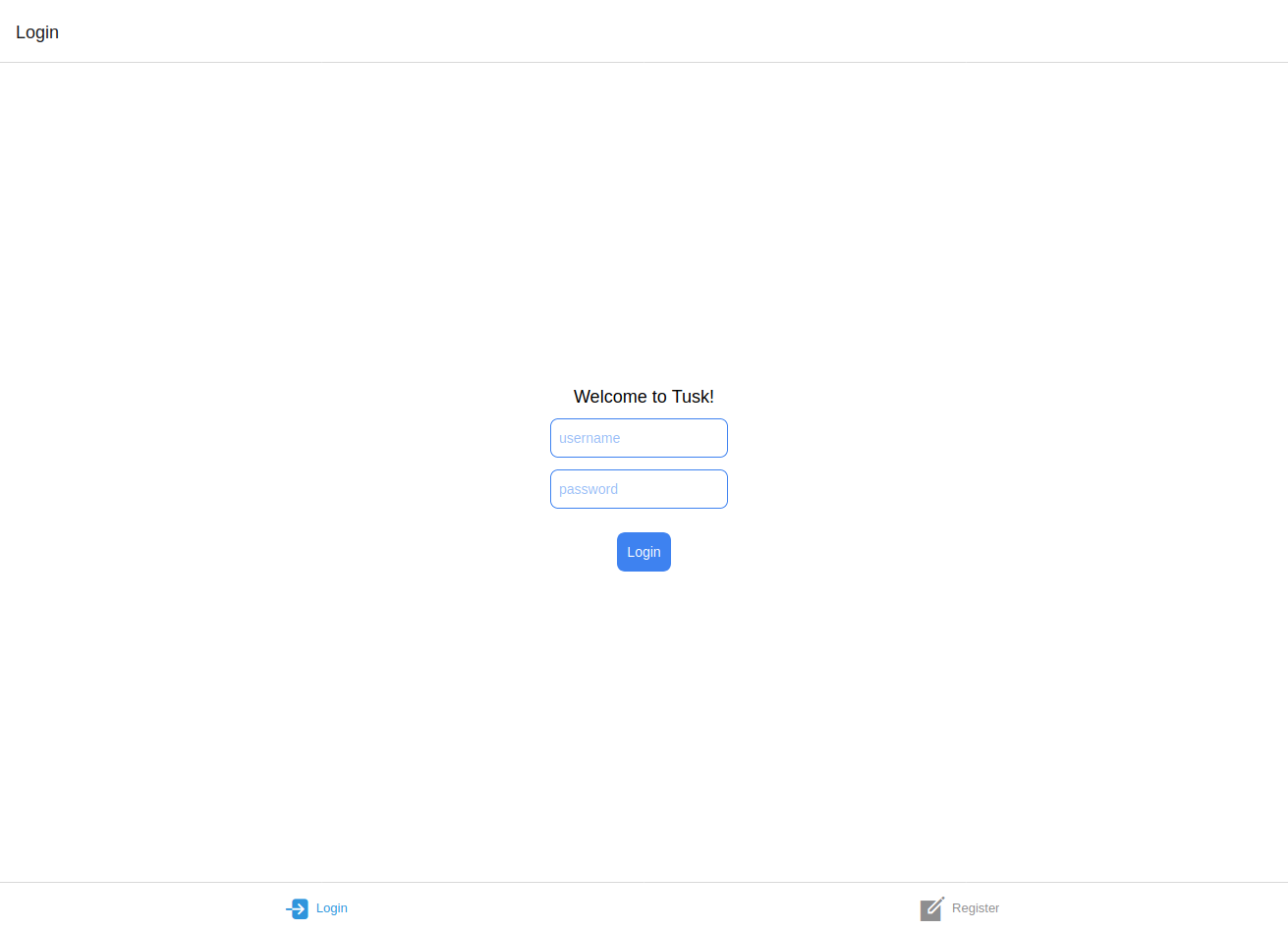
Svaki član kompanije može dodavati nove zadatke unutar aspekata svakoga zadatka. Također može dodavati i komentare na postojeće zadatke, te mijenjati njihov status u “gotovo”.



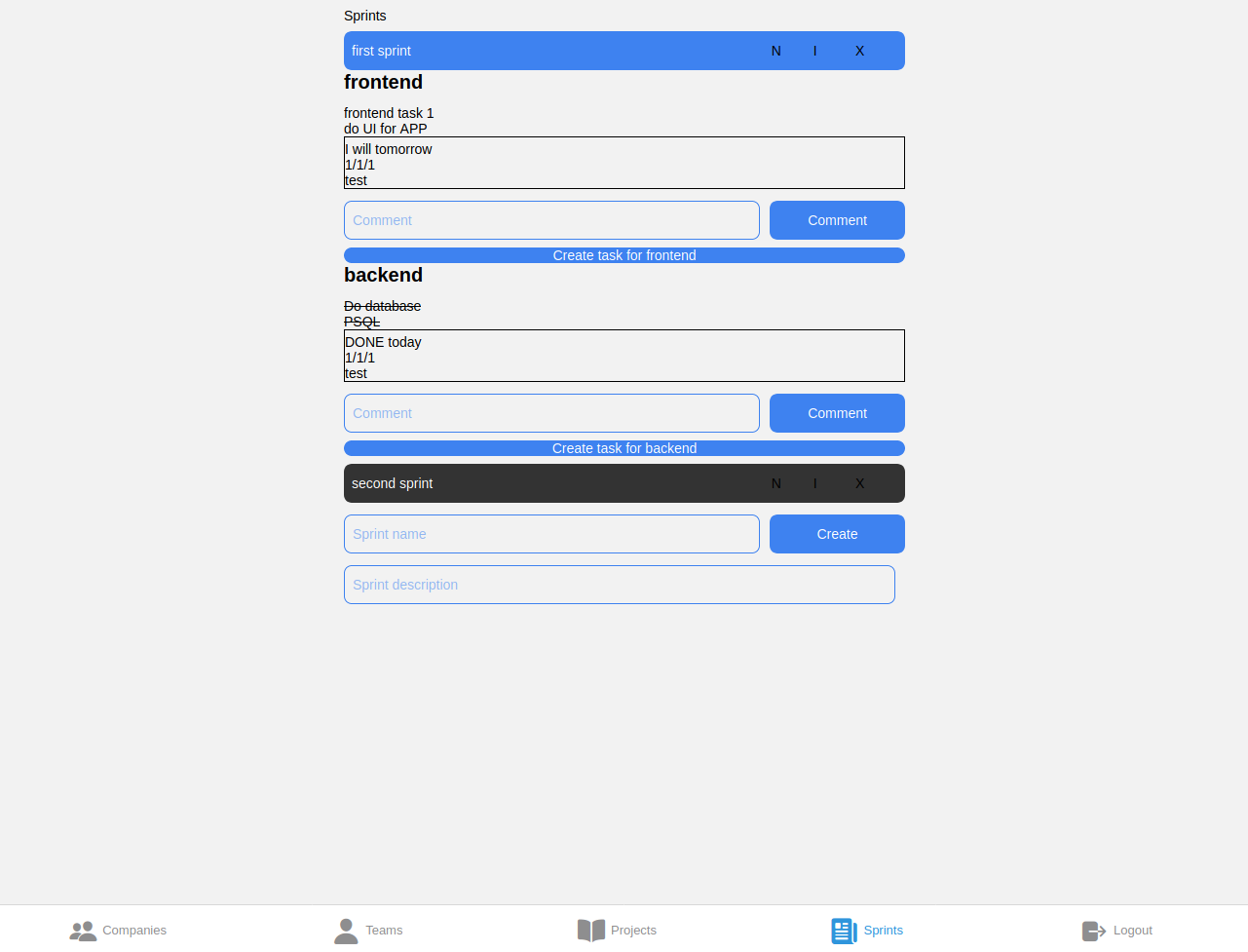
## 6.9 slika - Prikaz svih sprinteva i zadataka

# 7. Web aplikacija

Web aplikacija nije izgrađena uz pomoć ASP.NET radnog okvira, već radnog okvira React. Najviše našom pogreškom i prekasnim saznanjem da se tako traži od nas u zadatku.  
  
Po izgledu i ponašanju, aplikacija se ne razlikuje previše od mobilne aplikacije jer smo uspjeli ponovno iskoristiti kod iz Mobilne aplikacije i pretvoriti ga u web aplikaciju uz pomoće React radnog okvira.



7.1 slika - stranica za prijavu na web poslužitelju



7.2 slika - prikaz svih sprintova i zadataka po aspektima

# 

## 8. NHibernate vs Entity Framework

Ovaj dio dokumenta bavi se proučvanjem sličnosti i razlika između ova dva radna okriva. Cilj je dobiti što veći uvid u trade-offe koji se rade odabirom jedne od ovih tehnologija nad drugom. Kao stupove usporedbe odlučili smo uzeti: distributere, podržane baze podataka, korisne značajke, upravljanje konkurentnim pristupom podacima..

## Distributeri

NHibernate je radni okvir distribuiran i održavan od strane zajednice, bez ikakvih sponzora ili sponzorskih kompanija. Entity Framework je radni okvir usko vezan uz tvrtku Microsoft. To generalno nije najbolji indikator, obzirom da ta tvrtka esencijalno drži monopol nad tehnologijom i donosi relevantne odluke koje utječu na naš kod. S druge strane, ako je tvrtka pouzdana i pokazuje da su politike odluka u skladu s dobrim principima poput “backwards compatibilitya”, nije loše imati ju kao distributera koji će kontinuirano održavati taj kod.

## Podržane baze podataka

Obzirom da koristimo bazu podataka za perzistenciju, svakako želimo analizirati koja tehnologija podržava koji stack. NHibernate je u ovom pogledu svakako daleko ispred Entity Frameworka. Podržava PostgreSQL, SQL Server, SQL Lite, My SQL , DB2, Ingres … dok EF podržava mnogo manje. Neke koje EF podržava su SQL Server, PostgreSQL, te SQL Lite.

## Upravljanje konkurentnim pristupom podacima

Po oba aspekta, obje tehnologije pružaju zadovoljavajuću podršku. Iako, naizgled NHibernate pruža više potencijalno korisnih značajki u ovom aspektu poput eksplicitnog zaključavanja n-torki. Nama to ne donosi nikakvu vrijednost, tako da ovo nije komponenta koja nas je nagnala na odluku.

## Korisne značajke

### Kolekcije

NHibernate nudi mnogo veću podršku za kolekcije nego Entity Framework. Za razliku od EF koji nudi podršku samo za liste, NHibernate nudi neporedane kolekcije, setove, mape, primitivne kolekcije..

Iz tog aspekta NHibernate pruža značajno više nego EF, te ako imamo neke aplikacijski specifične potrebe, ovo može biti komponenta koja nam utemelji konačnu odluku.

### Lazy loading

NHibernate podržava lazy loading, za razliku od Entity Frameworka. Iako je to veoma koristan feature, nama ne utječe na odluku obzirom da ne očekujemo mnogo podataka u aplikaciji. Kada bi aplikaciju skalirali na produkcijsku razinu, svakako je ovo točka koja bi nam mnogo značila u procjeni.

### Infrastrukturalna podrška

Jedna od najvećih podrški u Entity Frameworku jest autogeneriranje entiteta iz baze podataka. Ovo je vrlo važna stavka jer može inženjerima uštediti velike količine vremena koje bi inače uložili u pisanje boilerplate koda. Također, najviše grešaka nalazi se upravo u ovakvom kodu. On je repetitivan, te developeri često posežu za CPP (copy-paste-programming) metodologijom :) .

### Iskoristivost

Obzirom na mnoge feature koje pruža NHibernate, learning curve se može činiti poprilično strm. Za razliku od njega, Entity Framework je lak za uporabu i temeljito dokumentiran.

9. Testiranje web aplikacije

### 9.1 White box testing

Unit testovi

Unit testovima smo testirali najbitnije dijelove web aplikacije kako bi se uvjerili u ispravan rad. Testovi uključuju provjeru odgovora s web poslužitelja, upravljanje pogreškama i domensku logiku. Testovi su uključivali mocking HTTP klijenta i baze podataka kako bi mogli testirati bez pokretanja HTTP poslužitelja na neprodukcijskoj bazi. Za mockanje baze podataka nam je poslužila SQLite in-memory baza podataka u koju smo prilikom pokretanja testova unijeli testne podatke i provjeravali dobivene rezultate. Bilo je potrebno mockati HTTP klijenta kako bi mogli dodati *Authorization* zaglavlje zato što neautorizirani klijenti nemaju pravo pristupa resursima na poslužitelju. In-memory baza podataka nam je jako dobro koristila kako ne bi zagađivali produkcijsku bazu podataka s testnim primjerima, a i kako bi znali što očekivati kao odgovor na SQL upit jer znamo da se produkcijska baza podataka mijenja ovisno o tome što klijenti upisuju i kako koriste web-aplikaciju. Testovi su nam značajno ubrzali razvoj web-aplikacije zato što smo prije dodavanja nekog featurea mogli bezbolno provjeriti jesmo li potrgali neku od starih funkcionalnosti. Na taj način smo uspjeli ubrzati i dovršiti projekt do kraja.

U nastavku je prikazan test koji testira ispravno dohvaćanje korisnika s identitetom “1”.



### 9.2 Black box testing

Naravno, nakon dodavanja nekog novog featurea potrebno je provjeriti kako to sve radi u aplikaciji iz perspektive krajnjeg korisnika. Na taj način možemo uočiti greške koje nismo mogli predvidjeti jer nismo mogli predvidjeti način na koji će korisnik koristiti našu aplikaciju. Također, ovo je bitna stavka i iz razloga što neke stvari koje su naizgled točne mogu krajnjem korisniku biti veoma neintuitivne i nepristupačne i tada je to potrebno promijeniti. Na ovaj način kombiniranjem white box i biack box testiranja smo uspjeli aplikaciju dovesti do zadovoljavajućeg rezultata i za developere i za krajnje korisnike aplikacije.