Univerzitet u Beogradu
Matematički fakultet

Master rad

Implementacija Kanban-table u radnom okviru .NET Core

Autor:
Tijana Tošev

Mentor:
prof. dr Saša Malkov

Beograd,
2021.

Mentor:

prof. dr Saša Malkov
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

prof. dr Milena Vujošević-Janičić, redovni profesor
Univerzitet u Beogradu, Matematički fakultet

doc. dr Ivan Čukić, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Sadržaj

1 Uvod

Tokom poslednjih petnaest godina su razvoj tehnologija, interneta i povećanje potreba za aplikacijama podstakli dalji razvoj metodologija razvoja softvera. Klijenti imaju potrebu da svoje proizvode dobiju u što kraćem vremenskom roku i otuda se javlja potreba da vreme kojim razvijaoci raspolažu bude maksimalno iskorišćeno. Kanban-table predstavljaju alat za koordinisanje posla i koriste se u okviru više postojećih metodologija razvoja softvera. Vizualizacijom poslova u obliku kartica koje sadrže informacije šta je potrebno uraditi se omogućava uvid u količinu posla i uočavanje mogućih načina povećanja efikasnoti izvršavanja samih poslova. Kanban-table su popularne i veoma korisne ne samo u oblasti razvoja softvera. Tačnije, veliki broj ljudi danas koristi kanban-table u svakodnevnom životu kako bi svoje obaveze organizovali na najefikasniji mogući način.

U ovom radu su detaljno predstavljene kanban-table i na koji način se danas mogu koristiti u upravljanju projektima. U drugom poglavlju dat je kratak osvrt na metodologije razvoja softvera i način korišćenja kanban-tabli. U trećem poglavlju su predstavljeni radni okviri *.NET Core* i *Angular* koji su korišćeni prilikom razvoja same veb aplikacije. U četvrtom poglavlju je predstavljena veb aplikacija *TT-Kanban* koja je razvijena u okviru rada. Peto poglavlje je zaključak koji predstavlja kratak rezime urađenog i predloge unapređenja razvijene veb aplikacije.

2 Metodologije razvoja softvera

Metodologije razvoja softvera predstavljaju standardizovane procedure i tehnike koje se koriste u procesu razvoja softvera. Korišćenjem metodologije se čitav proces razvoja softvera deli na manje potprocese koji se mogu izvršavati relativno nezavisno jedni od drugih. Dodatna prednost je ako se potproseci mogu izvršavati paralelno čime se ubrzava čitav proces razvoja. Osnovni razlog korišćenja je lakši razvojni proces i postizanje boljih krajnjih rezulatata. Vremenom su nastale različite vrste metodologija koji prate određene principe i pravila, i neki od njih su: *agilni model*, *model vodopada*, *inkrementalni model*, *spiralni model*, *model prototipova*. Poslednjih godina se najviše koristi agilni razvoj softvera.

2.1 Agilni razvoj softvera

Agilni razvoj softvera predstavlja skup metodologija razvoja softvera koje prate pravila definisana 2001. godine od strane grupe programera koji su se nazivali *Agile Alliance*. U periodu koji je tome prethodio su mnoge kompanije imale probleme tokom projekata razvoja aplikacija i to je bio razlog definisanja manifesta agilnog razvoja softvera [1]:

> ”Otkrivamo bolje načine razvoja softvera razvijajući ga i pomažući drugima u tome. Kroz ovaj rad naučili smo više da vrednujemo:
>
> **Ljude i odnose među njima** - od procesa i alata
>
> **Funkcionalan softver** - od iscrpne dokumentacije
>
> **Saradnju sa klijentima** - od pregovaranja oko ugovora
>
> **Reagovanje na promene** - od praćenja plana
>
> Odnosno, iako cenimo vrednosti na desnoj strani, više vrednujemo one koje su na levoj strani.”

Prvi motiv manifesta govori da treba posvetiti pažnju ljudima jer oni predstavljaju najvažniji element do postizanja uspeha. Nije najvažnije imati najbolje veštine kodiranja već je potrebno znati kako komunicirati i raditi sa drugima u timu. Pored toga, treba obratiti pažnju na procese i alate u određenoj meri jer preterano korišćenje je jednako loše kao i nedostatak korišćenja [2].

Drugi motiv manifesta sugeriše da je funkcionalan softver uvek važniji od dokumentacije. Treba naći odgovarajući balans u količini potrebne dokumentacije i koja će sadržati najosnovnije opise funkcionalnosti softvera, a za detaljnije informacije o softveru je potrebno gledati kôd. Takođe, samo programski kôd nije dokumentacija. Suština je da ako je dokumentacija prevelika i ako nije u skladu sa kodom da to može dovesti do širenja netačnih informacija o softveru. Često

se desi da se više vremena posveti dokumentaciji umesto samom razvoju softvera pa zato postoji pravilo da se dokumentacija piše samo onda kad je njena potreba neposredna i značajna [2].

Treći motiv manifesta govori da softver nije nešto što se može naručiti i razviti na osnovu jednom definisanih zahteva. Nekada je teško zahteve klijenata koji nisu dobro definisani prevesti u zahteve koje će razvojni tim moći da ispuni. Razvijaoci uspešnih projekata su se zbog toga svakodnevno trudili da uzimaju u obzir povratne informacije klijenata radi definisanja što boljih zahteva. Unapred određivanje rokova, faza, zahteva i troškova pre samog početka projekta se nije pokazalo kao najbolja praksa [2].

Četvrti motiv manifesta nalaže da sposobnost reagovanja na promene određuje uspešnost projekta. Tok projekta ne može biti određen daleko u budućnosti i zato je potrebno da svi planovi koji se prave budu fleksibilni i podložni promenama. Poslovni prioriteti i zahtevi klijenata će se siguno promeniti u jednom trenutku i zato se ne može svaki korak unapred odrediti. Bolja strategija planiranja je da se detaljno planiraju naredne dve nedelje, zatim okvirno naredna tri meseca i da postoji zamisao kako će softver raditi nakon godinu dana [2].

Čitav manifesto i agilni razvoj softvera je proizašao iz već do tad postojećih metodologija kao što su *Ekstremno programiranje*, *Scrum*, *Objedinjen proces* i druge [3].

Na osnovu prehtodno opisanih motiva manifesta je definisano dvanaest pravila [1]:

1. Najveći prioritet je zadovoljiti klijenta kroz redovne isporuke vrednog softvera.

2. Otvoreno prihvatati promene, pa čak i u kasnim fazama razvoja. Agilni proces razvoja koristi promene za postizanje prednosti kupca u odnosu na njegovu konkurenciju.

3. Isporučivati funkcionalan softver što češće, poželjno u kratkim vremenskim okvirima, od par nedelja do par meseci.

4. Poslovni ljudi i razvijaoci moraju da sarađuju svakodnevno tokom trajanja projekta.

5. Zasnivati projekte na motivisanim pojedincima. Pružiti im potrebno okruženje i podršku i imajte poverenja da će obaviti posao.

6. Najbrži i najefikasniji način razmene informacija u razvojnom timu je razgovorom lice u lice.

7. Funkcionalan softver je osnovno merilo napretka.

8. Agilni proces razvoj promoviše održivi razvoj. Sponzori, razvijaoci i korisnici bi trebalo da neprestano održavaju ujednačen ritam.

9. Neprekidno posvećivanje pažnje tehničkoj doteranosti i dobrom dizajnu povećava agilnost.

10. Jednostavnost. Umetnost maksimizacije količine posla koji se ne obavlja je od suštinskog značaja.

11. Najbolje arhitekture, zahtevi i dizajni potiču od samoorganizovanih timova.

12. U regularnim intervalima, razvojni tim mora da sagledava svoj rad i uočava kako da bude efikasniji, i zatim da prilagođava svoje ponašanje u skladu sa tim.

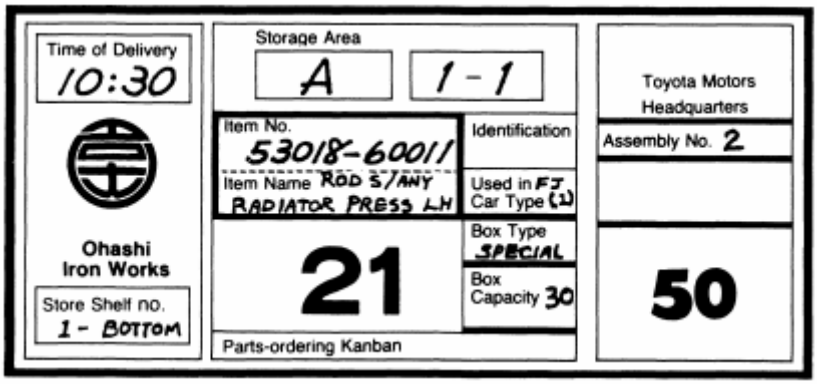
Jedna od tehnika koja se koristi u okviru nekih agilnih metodologija i korisna je za pridržavanje dvanaest pravila manifesta je *kanban*. Najvažniji element tehnike kanban je tzv. *kanban-tabla*.

2.2 Istorija Kanbana

Tehnika kanban, sa značenjem signalna kartica, se prvi put pojavila 50-ih godina 20. veka i to u okviru proizvodnog sistema automobila marke Tojota (eng. Toyota) [4]. Taiči Ono (eng. Taiichi Ohno) je ideju za oblikovanje tehnike kanban dobio zahvaljujući američkim supermarketima koji su postojali 40-ih godina 20. veka. Neki od važnih razloga primene kanban sistema u proizvodnom sistemu Tojota su postizanje isporuke delova u okviru proizvodnje *tačno na vreme* (eng. just-in-time), automatizacija i smanjenje gubitaka.

Kanban je bio papir koji sadrži potrebne informacije za rad u fabrici i prenosio se od jedne do druge stanice u okviru proizvodnog sistema. Suština je da naredni proces ne može da počne sa radom dok ne dobije signal od prethodnog procesa. Konkretan primer kanbana može biti papir koji sadrži informacije o preuzimanju, prenosu i proizvodnji delova. Kada kanban stigne do stanice za preuzimanje, tačno definisana količina delova će biti preuzeta. Nakon što se delovi prenesu i stignu do produkcije, tada će se proizvesti tačno definisana količina automobila ili konkretnih delova za automobile. Ovim je postignuto da ne postoji potrošnja i proizvodnja nepotrebne količine delova, a uz to je i smanjen gubitak prilikom čuvanja delova koji nisu potrebni. Na slici 1 je dat primer jednog kanbana iz proizvodnog sistema Tojota.

Korišćenjem kanbana postiže se automatizacija i prenos odgovornosti na same radnike. Naime, svaki radnik je imao uvid u vreme i time bi sam započinjao posao i donosio odluke, a pored toga postojali su menadžeri i nadzornici. Postojalo je više



Slika 1: Primer kanban kartice iz proizvodnog sistema Tojota

pravila koja su morala biti ispunjena kako bi se kanban koristio na dobar način u sistemu. Neka od tih pravila su:

1. roba se ne može preuzeti ili proizvesti ako ne postoji kanban;
2. kanban uvek mora biti *"zakačen"* na robu;
3. ne sme postojati neispravna roba, odnosno u naredni proces se ne sme poslati neka roba koja nije ispravna;
4. ukupan broj kanbana se mora smanjiti.

Ako se treće navedeno pravilo desi, tj. postoji neispravna roba, tada se mora zaustaviti sledeći proces. Ne može se nastaviti dalje ako prethodni proces nije bio urađen kako treba.

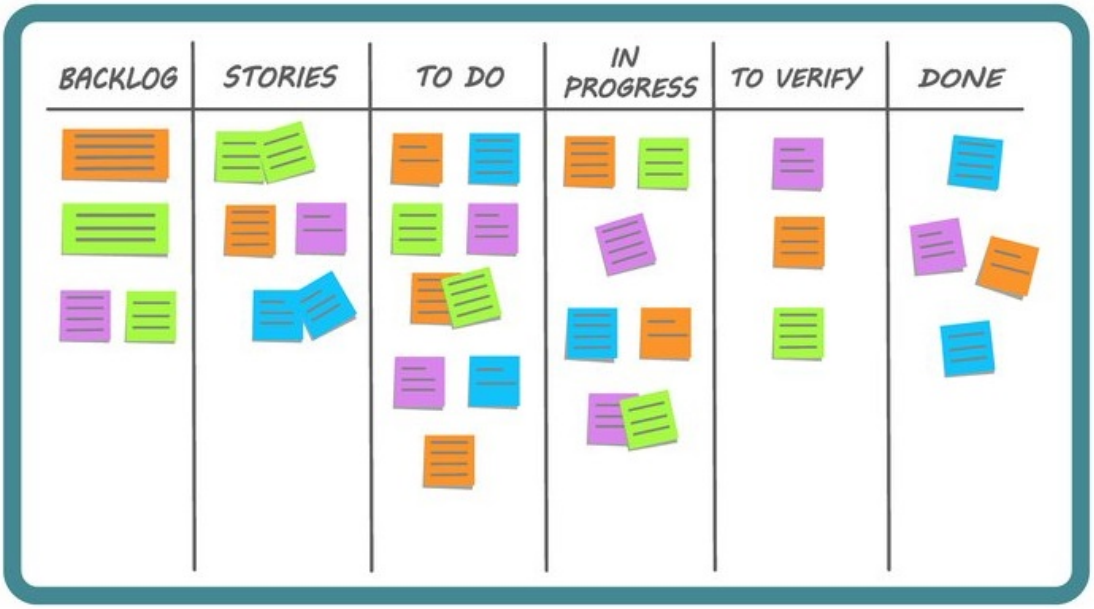
2.3 Kanban u razvoju softvera

U razvoju softvera, kanban uobičajeno predstavlja jedinicu posla u obliku virtuelne kartice, a ne kartica koja daje signal da se uzme još posla. Prvo konkretno usvajanje kanban-table i kanbana je bilo 2007. godine na projektu *Corbis* [5], dok se prvo pojavljivanje dogodilo 2004. godine na projektu za *Microsoft*. Prvi rezultati korišćenja kanban sistema na projektu *Corbis* su predstavljeni na konferenciji 2007. godine, a 2008. godine na *Agile 2008* je predstavljeno čak šest prezentacija o korišćenju kanban sistema na različitim projektima [5].

Kanban sistem predstavlja *pull* sistem što znači da jedan programer uvek radi na jednoj jedinici posla, što u ovom slučaju to predstavlja jednu kanban karticu [5]. Postoje *push* sistemi koji se razlikuju po tome što se u jednom trenutku radi

na više različitih jedinica posla. Osnovna prednost *pull* sistema je što ne može doći do preopterećenja zbog čega se i koriste u okviru razvoja softvera. Naime, u razvoju softvera klijenti imaju zahteve za koje žele da budu ispunjeni do nekog određenog vremena, a razvijaoci moraju da budu odgovorni i da znaju kada treba da odbiju određene zahteve. Ako razvijaoci prihvataju sve što im klijenti traže i ne uzimaju u obzir kapacitet tima i ljude koji ga čine onda je vrlo verovatno da će doći do preopterećenja tima. Svaki projekat mora biti isplaniran u skladu sa kapacitetima da timovi ne bi imali potrebu da rade prekovremeno kako bi ispunili vremenski nemoguće zahteve klijenata. U svetu je popularno mišljenje da programeri nemaju socijalni život i da jedino imaju vremena za posao i zato se korišćenjem kanban sistema omogućava da programeri ne budu prezaposleni tokom razvoja nekog projekta.

Kanban-table su virtuelna implementacija kanbana zato što kanban kartice fizički ne postoje kao u proizvodom sistemu Tojota. Kada se kanban sistem još uvek razvijao za vizuelizaciju su korišćene samolepljive nalepnice na zidovima ili tablama i primer se može videti na slici 2.



Slika 2: Primer kanban-table sa samolepljivim nalepnicama

Ispravno implementirana tehnika kanban mora da ima pet svojstava koja koristi kao osnovu za postavljanje pravilnih načina razvoja organizacija i projekata. Ova svojstva su korišćena i na prvom projektu koji je koristio kanban sistem i to su [5]:

1. vizuelizovati proces rada;

2. ograničiti trenutni posao koji je u toku;

3. kontrolisati i meriti protok;

4. učiniti procesne politike eksplicitnim;

5. korisiti u okviru postojećih modela metodologija kako bi se prepoznala moguća unapređenja.

Očekivalo se da će se osnovna lista svojstava promeniti jer se kanban i u proizvodnom sistemu Tojota razvijao skoro deset godina dok nije tačno utvrđen čitav proces. Tako su kroz različite projekte i iteracije primene otkriveni novi uticaji kanbana na razvoj softvera. Neka od novo-definisanih svojstva sa projekta *Corbis* su [5]:

- čitav proces kanban sistema zavisi od projekta;
- koristiti iterativni razvoj;
- posao planiranti u zavisnosti od cene kašnjenja;
- pojavu mogućih rizika rešiti unapred alociranim kapacitetom.

Korišćenjem kanbana u projektu *Corbis* su napredovali produktivnost, zadovoljstvo klijenata, kao i kvalitet isporučenog softvera.

Važno je napomenuti da kanban sistem nije metodologija razvoja softvera, već samo tehnika koja se koristi u okviru već razvijenih metodologija razvoja softvera. Kako bi korišćenje kanbana uspelo potrebno je da već postoji neki osnovni proces u koji bi se mogao postepeno primeniti kanban.

Najvažnije pitanje je kako u stvari na dobar način implementirati kanban i napraviti kanban-tablu? Na osnovu definisanih svojstava kanbana može se definisati osnovno uputstvo. Uputstvo je promenljivo u zavisnosti od projekta i tima, i obuhvata sledeće korake:

1. **Definisati proces rada** - Potrebno je definisati početni i krajnji proces u okviru vizualizacije. Važno je napraviti jasnu razliku koji svi procesi utiču na proces rada. Tim na projektu će se baviti analizom i dizajnom, a da bi to bilo moguće moraju biti dostavljeni svi poslovni zahtevi klijenata. Kako bi se testiralo sve što je implementirano potrebno je da se najnoviji kod podigne na sva testna okruženja. Ovo su neki osnovni procesi jednog projekta. Kasnije će ovi procesi odrediti kako će tačno izgledati kanban-tabla. Preporuka je ovaj korak odraditi na početku jer loše odluke mogu dovesti do neuspeha, a ako se na vreme odradi sve mogu se uočiti eventualne izmene.

2. **Definisati tipove zadataka** - Kada je definisan proces rada potrebno je odrediti koji su mogući tipovi kanban kartica, odnosno zadataka. Neki od uobičajenih tipova zadataka su:

   - *Funkcionalnost* (eng. feature);
   - *Zahtev* (eng. requirement);
   - *Poboljšanje* (eng. improvement);
   - *Slučaj upotrebe* (eng. use case);
   - *Korisnička celina* (eng. user story);
   - *Grupa korisničkih celina* (eng. epic);
   - *Greške* (eng. bug);
   - *Refaktorisanje* (eng. refactoring);
   - *Unakrsno testiranje* (eng. cross testing).

   Dobra ideja je tipove zadataka označavati različitim bojama na kanban-tabli.

3. **Kreirati kanban-tablu** - Korišćenjem kanbana je potrebno modelovati proces rada, a ne funkcije ili radnike. Na primer, testeri vrše testiranje pa bi jedna kolona mogla biti testiranje. Za početak je dobro skicirati kako bi trebalo da izgleda kanban-tabla na nekom zidu ili sličnoj površini. Prvih par nedelja postoji mogućnost izmene u procesu rada pa će se zajedno sa tim menjati i izgled kanban-table što je i očekivano. Danas je uobičajeno da se koriste sledeće kolone: *To Do*, *In Progress*, *Code Review*, *Ready for QA*, *In Progress QA*, *Done*, *Accepted*.

4. **Analizirati zadatake** - Kada su definisani tipovi zadataka potrebno je i popuniti svaku kanban karticu odgovarajućim tekstom ili uputstvom. U zavisnosti od tipa zadatka bi bilo poželjno imati i drugačiju strukturu teksta u okviru kanban kartice. Ovaj proces zahteva analizu kako bi se jasno definisale strukture zadataka. Potrebno je uočiti najefikasniji način dokumentovanja teksta jer ipak ne treba dodavati karticu za svaki sitan posao u okviru projekta. Takođe, jedno od pravila korišćenja kanban sistema je da broj kartica treba smanjiti.

5. **Odrediti kapacitet posla u odnosu na zahteve** - Svaki projekat ima poslovne zahteve klijenata i često se kanban može integrisati u već postojeći projekat. Sav softver i aplikacije su skloni greškama koje se moraju ispraviti. Ako je na primer izvorni kôd već postojećeg projekta star određen broj godina onda je vrlo verovatno da je taj kôd potrebno refaktorisati. Zbog navedenih i mnogih drugih prepreka je potrebno odrediti koliko vremena treba utrošiti na implementaciju poslovnih zahteva, ispravku grešaka, refaktorisanje i slično.

6. **Korisiti veb aplikaciju za kanban-tablu** - Nekada se tim koji radi na projektu ne nalazi na istoj lokaciji već je distribuiran po svetu. U tom slučaju je moguće koristi već postojeće veb aplikacije koje imaju kanban-table i mogućnost konfigurisanja prethodno opisanog procesa. Nekim timovima pored korišćenja veb aplikacije odgovara kada je kanban-tabla nacrtana na tabli ili nekom zidu u kancelariji radi lakšeg vizuelnog praćenja projekta.

3 Pregled tehnologija

U ovom poglavlju će biti prikazan pregled tehnologija za razvoj veb aplikacije koja će biti opisana u narednom poglavlju (4 Aplikacija TT-Kanban, str. 21).

3.1 Radni okvir .NET Core

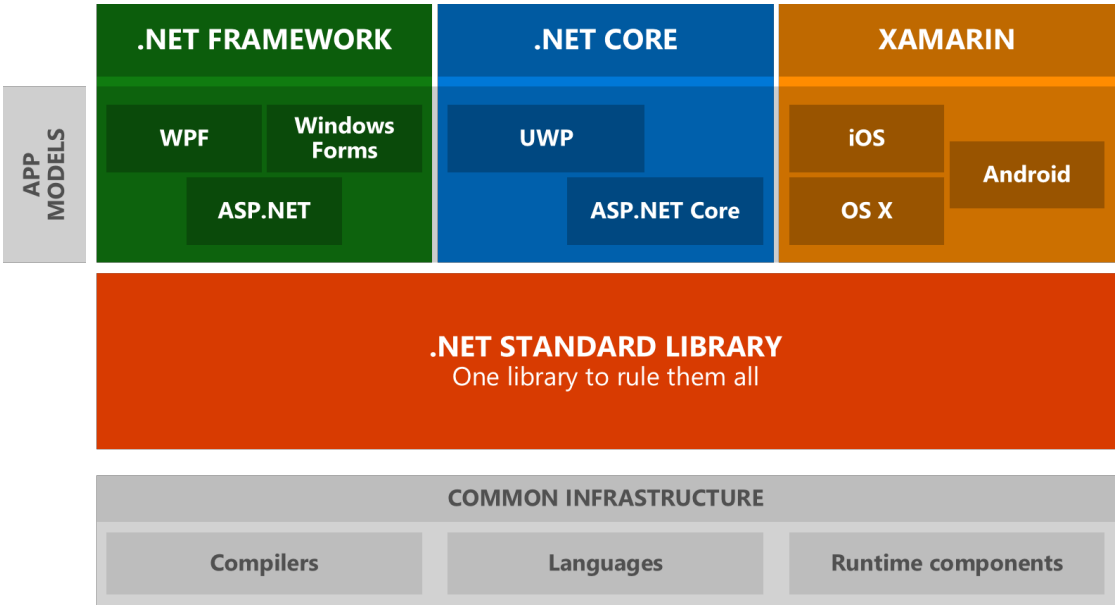
Radni okvir *.NET Core* je besplatna platforma otvorenog koda koja se može koristiti za razvoj aplikacija za različite operativne sisteme. Aplikacije se mogu pisati u različitim programskim jezicima:

- C# - podržava više paradigmi, ali je prvenstveno objektno-orijentisan programski jezik;
- F# - podržava više paradigmi, ali je u osnovi funkcionalan programski jezik i može se koristiti kao objektno-orijentisan;
- Visual Basic - objektno-orijentisan programski jezik koji je razvio *Microsoft*.

Postoji više verzija radnog okvira *.NET Core*, od 1.x pa do 5.x. Za implementaciju aplikacije TT-Kanban je korišćena verzija 3.1 i kao programski jezik je izabran C#. Sve aplikacije razvijene korišćenjem radnog okvira *.NET Core* se mogu pokretati na operativnim sistemima Windows, Linux i macOS [6].

Microsoft je 2002. godine prvi put objavio radni okvir *.NET Framework*, a tek 2016. godine prvu verziju radnog okvira *.NET Core*. Glavno pitanje je koja je njihova razlika? Suština je da je *.NET Framework* razvijen samo za operativni sistem Windows za razliku od *.NET Core-a*. Postoji i radni okvir *Mono* koji se pojavio još 2004. godine koji može da se pokreće na operativnim sistemima Windows, Linux i macOS. Pored toga se obično koristi za *Xamarin* aplikacije koje mogu da se pokreću na operativinim sistemima Android, iOS, tvOS, watchOS, XBoxOne, Sony PlayStation 4 i drugim. Svi navedeni radni okviri predstavljaju različite implementacije *.NET Standard API*-ja (eng. application programming interface) [7]. Ovo znači da ako je razvijena biblioteka u okviru jednog radnog okvira koji implementira određenu verziju *standarda .NET* da se ona može pokretati među svim aplikacijama u sklopu drugih radnih okvira ako oni implementiraju istu verziju *standarda .NET* [6].

Na slici 3 vidimo organizaciju implementacija *standarda .NET* u okviru *ekosistema .NET*. Svaka implementacija sadrži aplikacione modele koji se razlikuju u zavisnosti od potrebe aplikacije koja se razvija. *ASP.NET* i *ASP.NET Core* su vrsta aplikacionog modela namenjenih za veb i razlika je nad kojim radnim okvirom se izvršavaju. Implementirana aplikacija u okviru rada koristi aplikacioni model *ASP.NET Core* koji omogućava integraciju sa popularnim klijentskim radnim okvirima i bibliotekama kao što je radni okvir *Angular* [6].



Slika 3: Ekosistem .NET

Važno je napomenuti da je u novembru 2020. godine *Microsoft* objavio *.NET 5* što predstavlja novu verziju radnog okvira *.NET Core*. Ovo je od sada osnovna implementacija *.NET* jer je ranije to bio *.NET Framework*. Pored toga, 4.x verzije radnog okvira *.NET Framework* će biti i dalje podržane. Za aplikacije i biblioteke radnog okvira *.NET 5* cilnji radni okvir (eng. target framework moniker) je *.NET 5* čime su spojeni i ukombinovani *.NET Core* i *standard .NET*. Međutim, za aplikacije i biblioteke radnih okvira *.NET Framework*, *.NET Core* i *.NET 5* koje imaju deljeni kôd je kao ciljni radni okvir potrebno odabrati *standard .NET 2.0*. [6].

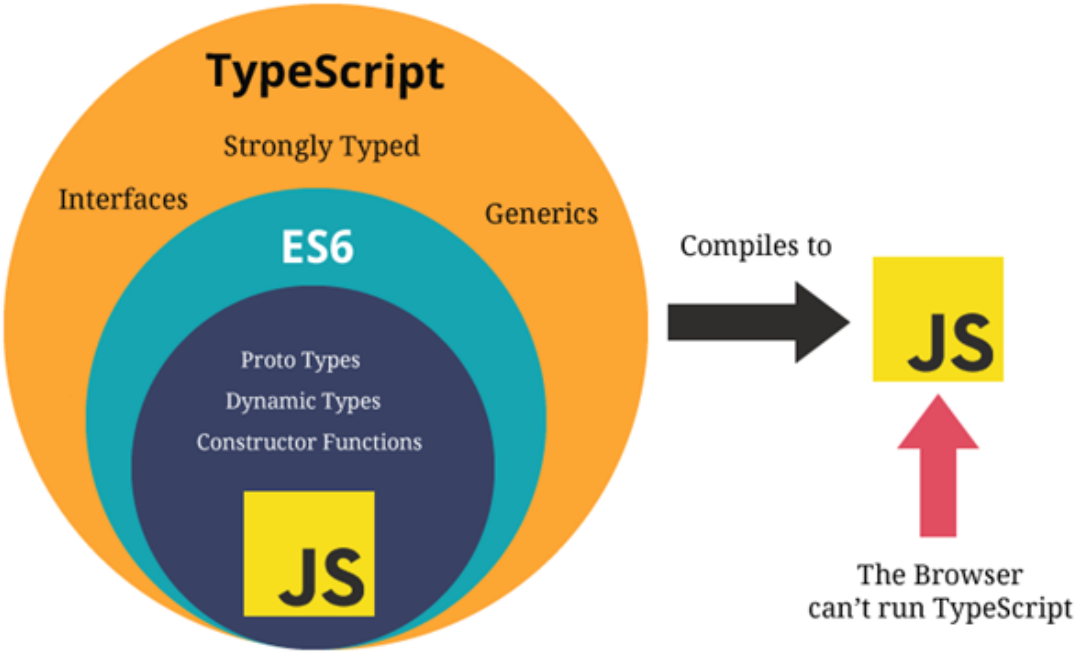
Jedna zanimljivost radnog okvira *.NET Core* je da se može pokretati u okviru *Docker*-kontejnera. *Docker* je platforma otvorenog koda koja omogućava pakovanje i pokretanje aplikacije u izolovanom okruženju koji se naziva kontejner (eng. container) [8]. Na ovaj način se aplikacije odvajaju od svoje infrastrukture i može se koristiti u procesu kontinuirane integracije i isporuke koda. Zbog svoje izolovanosti svaki kontejner ima svoj operativni sistem, memoriju i procese što omogućava kreiranje velikog broja kontejnera na jednoj mašini. Uz sve to mogu se pokretati na različitim operativnim sistema i na *cloud* servisima kao što je *Azure Kubernetes Service*. [1] Kada se koristi radni okvir *.NET Framework* onda se aplikacije mogu pokretati samo u okviru Windows-kontejnera, dok u slučaju radnog okvira *.NET Core* aplikacije se mogu pokretati i u okviru Linux-kontejnera [6].

[1] Više o Azure Kubernetes Service na https://azure.microsoft.com/en-us/services/kubernetes-service/#overview

3.2 Radni okvir Angular

Radni okvir *Angular* je besplatna platforma otvorenog koda, koju je razvio *Google* i koristi se za razvoj *SPA* (eng. single-page application) veb aplikacija. *SPA* se zasniva na tome da se ne učitava cela strana već se delovi stranice dinamički učitavaju u zavisnosti od korisnikove interakcije sa veb stranom [9].

Kada se koristi *Angular* važno je napomenuti koja se tačno verzija koristi. Prva verzija je *AngularJS* koji je nastao 2010. godine i on je pisan u programskom jeziku *JavaScript*. Druga verzija je *Angular2* koji je objavljen 2014. godine i danas je većina aplikacija razvijena koristeći ovu verziju. *Angular2*, odnosno *Angular*, je za razliku od *AngularJS* pisan u programskom jeziku *TypeScript*. Pretraživači nemaju mogućnost da samostalno prevedu *TypeScript* u *JavaScript* i zato se koriste posebni *tsc*-kompajleri koji to rade [9]. Programski jezik *TypeScript* obuhvata sve što ima *JavaScript* i uz sve to dodaje statičku tipiziranost. Može se reći da je *TypeScript* nadskup programskog jezika *JavaScript*. Na slici 4 možemo videti tačan odnos programskih jezika *TypeScript* i *JavaScript*, kao i standarda *ECMAScript6*[2].



Slika 4: Prikaz odnosa TypeScript, JavaScript i ECS6

Pored osnovne razlike u programskom jeziku, *AngularJS* i *Angular* imaju još

[2]ECMAScript je standard definisan od strane ECMA International organizacije radi standardizacije script programskih jezika.

jednu važnu razliku i to je arhitektura. *AngularJS* koristi obrazac *MVC*, tačnije obrazac *MV**, dok *Angular* koristi komponente, direktive i šablone. *Model-Pogled-Kontroler* (eng. model-view-controller) je arhitekturalni obrazac koji se realizuje na sledeći način:

- **Model** — predstavlja podatke i bazu podatka, i obično su prikazani u vidu klasa;
- **Pogled** — predstavlja korisnički interfejs i šta korisnik vidi;
- **Kontroler** — predstavlja vezu između modela i pogleda, i određuje na koji način će komunicirati.

Obrazac *MV** (eng. model-view-whatever) je skraćenica za *Model-Pogled-ŠtaGod* i označava da veza između modela i pogleda može biti bilo šta, odnosno da je potrebno samo da postoji model i pogled dok se o ostatku brine *AngularJS* [10].

U narednim podsekcijama će biti opisana arhitektura radnog okvira *Angular* koji je korišćen u implementaciji veb aplikacije *TT-Kanban* (4 Aplikacija TT-Kanban, str. 21).

3.2.1 Moduli

Osnovu *Angulara* čine moduli, tj. *NgModules*. Moduli obuhvataju komponente, servise, funkcije, direktive i ostale delove koda. Svaka aplikacija ima bar jedan koreni modul koji se naziva *AppModule* i nalazi se u datoteci *app.module.ts*. Aplikacija se učitava tako što se vrši "*bootstrapping*", odnosno inicijalizacija korenog modula.

Jedan *NgModule* se definiše tako što se klasa označi dekoratorom *@NgModule* koji je u stvari funkcija koja kreira objekat sa metapodacima koji opisuju modul. Najvažinija polja ovog dekoratora su:

- *declarations* — lista komponenti, direktiva i funkcija koje pripadaju modulu;
- *exports* — podskup deklaracija koje trebaju biti vidljive u ostalim modulima;
- *imports* — izvezene klase ostalih modula koje su potrebne komponentama ovog modula;
- *providers* — lista servisa koje koristi modul i koji kada su definisani postanu vidljivi u celoj aplikaciji;
- *bootstrap* — glavni pogled aplikacije, i jedino koreni modul treba da ima definisano ovo polje.

3.2.2 Komponenete

Komponente predstavljaju klase koje zajedno sa šablonima implementiraju pogled. Svaka aplikacija ima bar jednu komponentu i to je korena komponenta. U okviru korene komponente se pozivaju druge komponente koje su definisane i na taj način se formira hijerarhija komponenti.

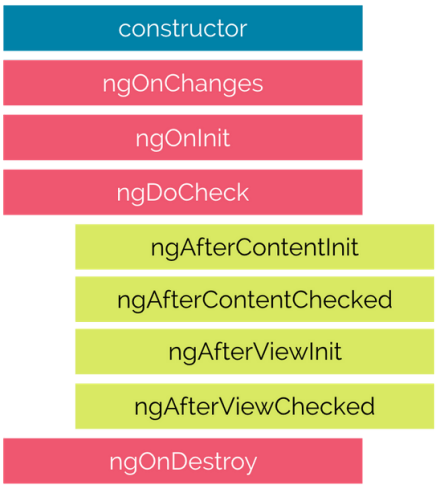
Komponenta se definiše tako što se klasa označi dekoratorom *@Component*. Najvažnija polja ovog dekoratora su:

- *selector* — CSS-selektor kojim se učitava i prikazuje komponenta ako je navedena u okviru nekog HTML-šablona;
- *templateUrl* — HTML-šablon komponente koji se može odmah definisati ili imati vrednost relativne adrese HTML datoteke;
- *providers* — niz servisa koji su potrebni komponenti.

Angular se stara o životu komponenti, tj. kreira, ažurira i uništava ih u zavisnosti od korisnikove interakcije sa aplikacijom. Komponente i direktive imaju svoje životne cikluse (eng. lifecycle hooks) koji se definišu implementiranjem određenih interfejsa. Ovi interfejsi imaju po jednu metodu čiji naziv se formira od prefiksa *ng* i samog naziva interfejsa, i oni se pozivaju u sledećim situacijama:

1. **ngOnChanges** — kada se vrednost ulazne/izlazne vezujuće vrednosti promeni;
2. **ngOnInit** — nakon prvog izvršavanja ngOnChanges;
3. **ngDoCheck** — detekcija *custom* promena;
4. **ngAfterContentInit** — nakon inicijializacije komponente;
5. **ngAfterContentChecked** — nakon svake provere sadržaja komponente;
6. **ngAfterViewInit** — nakon inicijalizacije šablona komponente;
7. **ngAfterViewChecked** — nakon svake provere šablona komponente;
8. **ngOnDestroy** — pre uništenja komponente.

Na slici 5 se može videti redosled pozivanja prethodno opisanih metoda i pre svega će se uvek prvo pozvati konstruktor klase. Preporučena praksa je da se inicijalizacije promenljivih i servisa ne izvršavaju u konstruktoru klase, već je bolje to uraditi u okviru *ngOnInit* metode. Ovo važi za komponente i za direktive.



Slika 5: Redosled pozivanja metoda životnog ciklusa

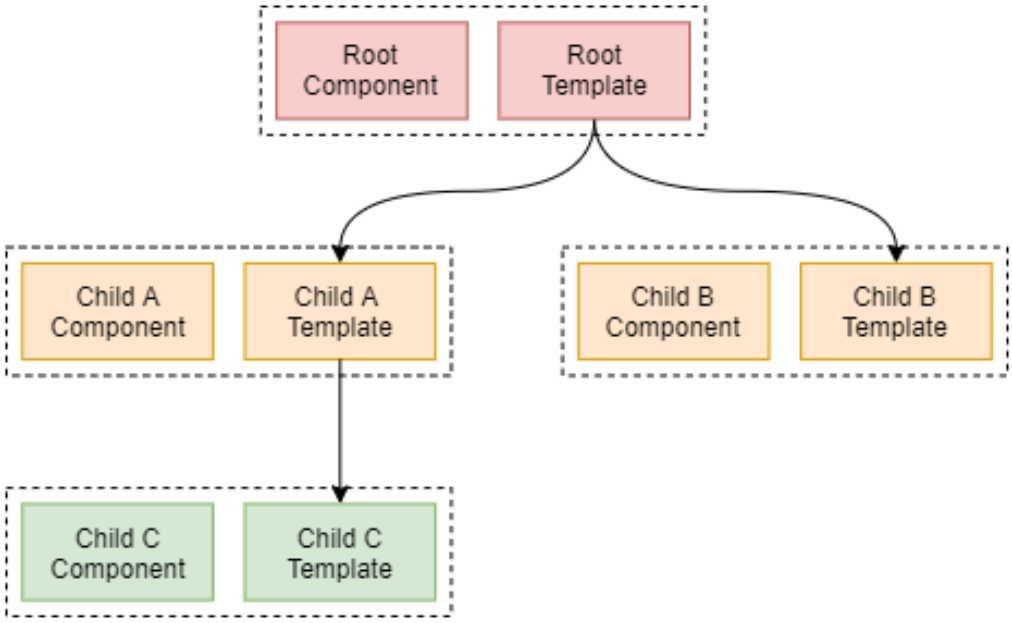
3.2.3 Šabloni

Šabloni su pogledi komponenti i oni su zaduženi kako će se određena komponenta prikazivati. Na ovaj način se delovi stranice ili čak i cela stranica mogu sakriti u zavisnosti od korisnikove interakcije sa aplikacijom. U šablonima se koriste HTML-elementi i direktive. Kao što postoji korena komponenta tako je uz nju vezan i koreni šablon. U okviru korenog šablona se umeću šabloni ostalih komponenti iz tog modula, kao i šabloni komponenti iz drugih modula. Ovakvim uvezivanjem više šablona se kreira hijerarhija pogleda što je ilustrovano na slici 6.

3.2.4 Direktive

Direktive su klase označene dekoratorom *@Directive* i one predstavljaju komponente bez šablona što znači da imaju životni ciklus, ulaze, izlaze, provajdere. Direktive služe da izmene izgled ili ponašanje već postojećih *DOM* (eng. document object model) elemenata. Postoje tri vrste direktiva:

1. komponente — direktive bez šablona;
2. direktive strukture — menjaju izgled dodavanjem i uklanjanjem elemenata;
3. direktive atributa — menjaju izgled ili ponašanje elementa, komponenata ili drugih direktiva.



Slika 6: Hijerarhija komponenti i šablona

Pored mogućnosti definisanja novih direktiva, *Angular* obuhvata ugrađene direktive koje se mogu koristiti.

NgIf je ugrađena direktiva ***strukture*** koja na osnovu vrednosti uslova dodaje ili uklanja element. Ako je vrednost uslova *null* tada se element neće prikazati. Primer korišćenja direktive možemo videti u listingu 1.

```
<app-board *ngIf="isActive" [board]="board"></app-board>
```

Listing 1: Direktiva NgIf

NgFor je ugrađena direktiva ***strukture*** koja se koristi za prikazivanje podataka neke liste. Konkretan primer je dat u listingu 2.

```
<div *ngFor="let board of boards"> {{ board.Name }} </div>
```

Listing 2: Direktiva NgFor

Delom koda *"let board of boards"* se prolazi kroz listu i svaki element se čuva u promenljivoj *"board"* kojoj se pristupa pomoću dela koda *{{ board.Name }}*. U okviru direktive je moguće koristiti *index*, *odd*, *even*, *first*, *last* i *trackBy* za dohvatanje određenih elemenata i njihovo kontrolisanje.

NgSwitch je ugrađena direktiva ***strukture*** koja predstavlja *switch* naredbu i obuhvata više direktiva za razliku od prethodne dve. To su sledeće direktive:

- *NgSwitch* - direktiva atributa koja menja ponašanje direktiva u sklopu nje;
- *NgSwitchCase* - direktiva strukture koja dodaje element kada je vrednost tog uslova ispunjena i uklanja u slučaju da nije;
- *NgSwitchDefault* - direktiva strukture koja dodaje element u slučaju da nijedan od prethodnih *NgSwitchCase* uslova nije ispunjen;

Primer korišćenja ove direktive je dat u okviru listinga 3.

```
<div *ngSwitch="priorities">
     <span *ngSwitchCase="1"> Lowest </span>
     <span *ngSwitchCase="2"> Low </span>
     <span *ngSwitchCase="3"> Medium </span>
     <span *ngSwitchCase="4"> High </span>
     <span *ngSwitchCase="5"> Highest </span>
     <span *ngSwitchDefault> Low </span>
</div>
```

Listing 3: Direktiva NgSwitch

NgClass je ugrađena direktiva ***atributa*** i koristi se za dodavanje i uklanjanje *CSS* (eng. cascading sheet style) klasa. Može se koristiti sa uslovnim izrazima ili sa metodama. Primer korišćenja ove direktive je prikazan u listingu 4.

```
<app-board [ngClass]="isActive ? 'visible' : 'hidden'" [board]="
board"></app-board>
```

Listing 4: Direktiva NgClass

NgStyle je ugrađena direktiva ***atributa*** koja se može koristiti za dodavanje više klasa jednom elementu na osnovu vrednosti komponente. Konkretan način korišćenja je prikazan u narednim listinzima 5 i 6.

```
<app-board [ngStyle]="styles" [board]="board"></app-board>
```

Listing 5: Direktiva NgStyle

```
styles: Record<string, string> = {};
....
styles() {
    this.currentStyles = {
      'font-weight':  this.isActive ? 'bold' : 'normal',
      'display': this.isVisible ? 'block' : 'none'
};
}
```

Listing 6: Postavljanje stilova za NgStyle direktivu u okviru komponente

NgModel je ugrađena direktiva ***atributa*** i služi za prikazivanje vrednosti polja i ažuriranje polja kada korisnik napravi neku izmenu u okviru formi. Ako želimo da koristimo ovu direktivu nad elementima koji nemaju *ControlValueAccessor*, onda je potrebno da se posebno implementira *value accessor*. Za korišćenje je potrebno uključiti modul *FormsModule* u koreni modul i dodati ga listi *imports*. U listingu 7 *[(ngModel)]* predstavlja način povezivanja podataka.

```
<label for="name">Name:</label>
<input [(ngModel)]="board.Name" />
```

Listing 7: Direktiva NgModel

3.2.5 Povezivanje podataka

Aplikacije napisane u okviru *Angulara* su zamišljene da rade po principu da korisnik ima mogućnost da napravi određene akcije u okviru šablona koje utiču na podatke i na koje moraju da odgovore komponente, ali te izmene treba da se odraze i na komponente. *Povezivanje podataka* predstavlja važan koncept jer je baš on odgovoran za komunikaciju određenih delova šablona sa delovima komponenti, a šabloni govore *Angularu* šta i na koji način se povezuje. Postoje dve vrste povezivanja podataka i to su *jednosmerno povezivanje* i *dvosmerno povezivanje*.

Prvi tip jednosmernog povezivanja je *interpolacija* i predstavlja način povezivanja dinamičkih niski iz komponenti u šablone. Interpolacija teksta se postiže korišćenjem duplih vitičastih zagrada *{{ }}* kao što se može videti u primeru koda 2. Pored vezivanja niski, mogu se uvezati funkcije i izrazi koji se prvo izračunavaju, a zatim prevode u niske. Primer interpolacije je dat u listingu 8.

```
<!-- "The value of expression 4 + 2 is 6." -->
<span>The value of expression 4 + 2 is {{ 4 + 2 }}.</span>
```

Listing 8: Primer interpolacije

Drugi tip jednosmernog povezivanja je *povezivanje po svojstvu* i obično se koristi za deljenje podataka između komponenti i postavljanje svojstava HTML-elemenata ili direktiva. Svojstva jednog elementa se navode u okviru uglastih zagrada *[ ]* i to svojstvo je ciljno svojstvo kom želimo da dodelimo vrednost. Uglaste zagrade označavaju da *Angular* desnu stranu izraza izračunava kao dinamički izraz, dok u suprotnom računa kao nisku i dodeljuje je kao statičku vrednost svojstvu. Za definisanje svojstava u okviru jedne komponente je potrebno dodati dekorator *@Input()* i pri pozivanju komponente u okviru drugih komponenti je neophodno dodeliti vrednost svojstvima. Primer korišćenja povezivanja po svojstvu u okviru HTML-elementa je dato u listingu 9, a primer korišćenja povezivanja po svojstvu u okviru komponente je prikazano u listingu 10.

```
<button [disabled]="isFormValid()">Save</button>
```

Listing 9: Primer povezivanja po svojstvu HTML-elementa

```
<app-board [board]="board"></app-board>
```

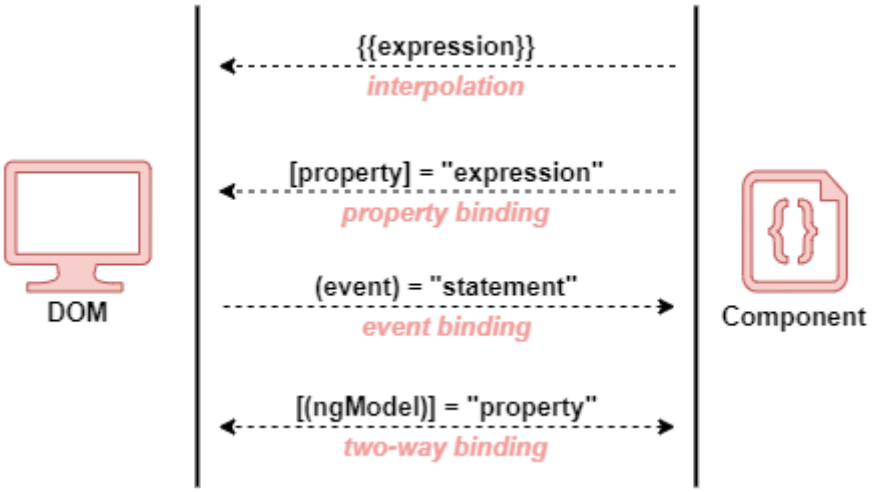
Listing 10: Primer povezivanja po svojstvu komponenti

Treći tip jednosmernog povezivanja je *povezivanje po događaju* i predstavlja način povezivanja odgovora na korisnikovu akciju, odnosno događaj, u okviru šablona. Na ovaj način se podaci šalju iz šablona ka komponenti. Neki od događaja su kada korisnik klikne na dugme, kada korisnik pritisne određeno dugme na tastaturi, kada korisnik pokazivačem miša pređe preko nekog teksta i slično. Za svaki događaj se vezuje objekat *$event* sa različitim poljima i vrednostima u zavisnosti od tipa događaja. Sintaksa događaja je da se navode u okviru *( )* zagrada, a u okviru desne strane izraza se navodi metoda koja će obraditi događaj. U listingu 11 je prikazano kako se povezuje na događaj *click*.

```
<button (click)="onClick($event)">Save</button>
```

Listing 11: Primer povezivanja po svojstvu komponenti

U okviru navedenih tipova jednosmernog povezivanja su se podaci slali samo u jednom smeru od komponente ka šablonu ili obrnuto. U slučaju dvosmernog povezivanja se podaci šalju u oba smera, od komponenete ka šablonu i od šablona ka komponenti što možemo videti na slici 7.



Slika 7: Smerovi povezivanja podataka

Dvosmerno povezivanje se obično koristi na formama kada je potrebno da se uneta vrednost u *input* elementu odmah pošalje komponenti, i može se koristiti za komunikaciju više komponenti. Kada se učita forma moguće je imati predefinisane vrednosti nekih polja i tada se koristi povezivanje po svojstvu, a kada korisnik izmeni tu vrednost tada se koristi povezivanje po događaju. Sintaksa dvosmernog povezivanja *[( )]* objedinjuje povezivanje po svojstvu i po događaju, respektivno. Ako se dvosmerno povezivanje koristi u okviru komponenti potrebno je definisati polja i označiti ih dekoratorima *@Input* i *@Output*. Tada će polje *@Input* biti zaduženo za dohvatanje podataka iz komponente, a polje *@Output* će biti zaduženo za izmenu podataka u zavisnosti od akcije korisnika u šablonu i ono mora biti tipa *EventEmitter*. Kako se dvosmerno povezivanje koristi u komponentama sa dekoratorim *@Input* i *@Output* je pokazano u listingu 12. U prikazanom primeru će se klikom na dugme uvećati vrednost polja *counter* i u konzoli pretraživača će biti ispisana poruka sa novom vrednošću polja *counter*.

```
// counter.component.ts
export class SizerComponent {

 @Input()  counter: number;
 @Output() counterChange = new EventEmitter<number>();

 public update() {
    this.counter++;
    this.counterChange.emit(this.counter);
 }

}

// counter.component.html
<label>{{ counter }}</label>
<button (click)="update()"> Update </button>

// app.component.html
<app-counter [counter]="counter"
    (counterChange)="onCounterChange($event)"></app-counter>

// app.component.ts
public counter = 0;

public onCounterChange(data) {
    console.log("Counter was changed: " + data);
}
```

Listing 12: Primer dvosmernog povezivanja u okviru komponenti

Ako se dvosmerno povezivanje koristi u okviru formi tada se koristi direktiva *ngModel*, a primer korišćenja je dat u listingu 13.

```
<input type="text" [(ngModel)]="user.username"/>
```

Listing 13: Primer dvosmernog povezivanja za elemente formi

3.2.6 Umetanje zavisnosti

Umetanje zavisnosti se u *Angularu* koristi kako bi sve komponente imale pristup potrebnim servisima. Zavisnost ne mora biti samo servis, već to može biti funkcija ili neka vrednost. Servis predstavlja klasu koja je označena dekoratorom *@Injectable*. Na ovaj način *Angular* će umetnuti servis određenoj komponenti kao zavisnost.

Kada se prvobitno vrši inicijalizacija korenog modula (bootstrapping) tada se kreira *"injector"* aplikacije. *Injector* kreira zavisnosti i održava kontejner sa njihovim instancama koje koristi po potrebi. *Provajderi* su objekti koji govore *injector*-u kako da dohvata i kreira zavisnosti. Za bilo koju zavisnost koju želimo u aplikaciji moramo da registrujemo provajder u okviru aplikacionog *injector*-a kako bi mogao da koristi provajder za kreiranje novih instanci. U slučaju servisa, provajder je obično sama klasa servisa. U toku kreiranja nove instance komponente se određuje koji servisi ili zavisnosti su joj potrebne, što se razrešava na osnovu navedenih parametara u konstruktoru te klase. Prvo se proverava da li *injector* ima neku instancu potrebnog servisa ili zavisnosti. U slučaju da ne postoji instanca *injector* koristeći registrovani provajder kreira novu instancu koju dodaje u kontejner pre prosleđivanja servisa ili zavisnosti u *Angularu*. Tek kada su sve zavisnosti razrešene i prosleđene, *Angular* pristupa konstruktoru sa svim instancama servisa.

3.2.7 Posmatrači i pretplatnici

Postmatrači (eng. observables) se u *Angularu* koriste za prosleđivanje više vrednosti, za rukovanje događajima i asinhrone pozive. Obrazac za projektovanje *Posmatrač* (eng. observer) obuhvata jedan objekat koji se naziva *Subjekat* i koji može imati više zavisnosti odnosno *Postmatrača*. Na ovaj način se uspostavlja veza **jedan prema više** i kada se subjekat promeni tada on automatski šalje obaveštenje svakom posmatraču [11]. U *Angularu* se *posmatrači* deklarišu što znači da se definiše funkcija za slanje vrednosti, odnosno obaveštenja, ali se ne izvršava sve dok se *potrošač* ne pretplati. *Pretplatnici* (eng. subscriptions) će primati vrednosti i obaveštenja sve dok ne prekinu pretplatu za tog posmatrača. Kreiranje posmatrača je omogućeno korišćenjem biblioteke *RxJS* [12]. Jednostavan primer upotrebe posmatrača i pretplatnika je dat u listingu 14.

```
let observable = new Observable<number>(observer => {
    observer.next(1);
});
```

```
observable.subscribe(value => {
    console.log(value);
});
```

Listing 14: Primer posmatrača i pretplatnika

Najčešće se posmatrači koriste za dohvatanje podataka pomoću modula HTTP. Ovaj modul obezbeđuje komunikaciju sa serverom kako bi se dohvatili ili poslali određeni podaci. Kada server odgovori, potrošač se može pretplatiti. Celokupan primer upotrebe i komunikacije posmatrača i pretplatnika je dat u listingu 15.

```
// board.service.ts
public getBoards() {
    return this.http.get("api/boards");
}

// board.component.ts
ngOnInit() {
    this.boardService.getBoards().subscribe(boards => {
      console.log(board)
    });
}

// board.component.html
<div *ngFor="let board of boards">
    {{ board.Name }}
</div>
```

Listing 15: Primer posmatrača i potrošača

4 Aplikacija TT-Kanban

U okviru rada razvijena je veb aplikacija *TT-Kanban* koja će razvijaocima projekata predstavljati pomoćni alat u upravljanju poslovima. Aplikacija omogućava vizualizaciju zadataka na kanban-tabli, korišćenje tajmera tokom izvršavanja jednog od zadataka i pregled nekih osnovnih statistika za svaki projekat.

4.1 Osnovni pojmovi

Za dalji rad je potrebno definisati osnovne pojmove koji će biti korišćeni u opisu aplikacije i to su:

- **projekat** — sadrži kanban-tablu;
- **zadatak** — jedinica posla, odnosno kanban kartica;
- **status** — predstavlja kolonu kanban-table;
- **tim** — grupa korisnika koja ima pristup određenim projektima.

4.2 Korisničke celine

Funkcionalnosti koje su implementirane su sledeće:

1. Registracija i prijava korisnika.
2. Početna strana sa opcijom pravljenja projekata i timova.
3. Profilna strana korisnika.
4. Podešavanja.
5. Prikaz projekata i timova.
6. Zadaci.
7. Tajmer.
8. Statistike.
9. Vertikalni prikaz više projekata.

Sve navedene funkcionalnosti ujedno predstavljaju korisničke celine. Neke od funkcionalnosti su osmišljene po uzoru na već postojeće veb aplikacije kao što su *Atlassian Jira* [3], *Microsoft Planner* [4] i *Trello* [5]. Slike korisničkog interfejsa aplikacije *TT-Kanban* su prikazane u dodatku A.2.

4.2.1 Registracija i prijava korisnika

Registracija i prijava korisnika predstavlja administrativni rad sa korisnicima. Prijava korisnika je moguća samo ako je prethodno već registrovan. Tokom registracije neophodno je popuniti korisničko ime, prezime i ime, e-mail adresu i lozinku. Kako bi registracija bila uspešna uneto korisničko ime mora biti jedinstveno i lozinka se mora sastojati od velikih i malih slova, brojeva i biti dužine od barem osam karaktera. Na slici 8 je prikazan dijagram aktivnosti registracije korisnika.

Korisnik se može prijaviti samo ako je uneo validno korisničko ime i lozinku. Svaka lozinka je sačuvana korišćenjem heš funkcije *SHA-2* i dohvatanje vrednosti lozinke nije moguće na klijentskoj strani aplikacije. Provera podataka tokom prijave se obavlja na serverskoj strani aplikacije. Takođe, u aplikaciji uvek postoji jedan predefinisan korisnik koji je administrator i ima pristup svim projektima i timovima.

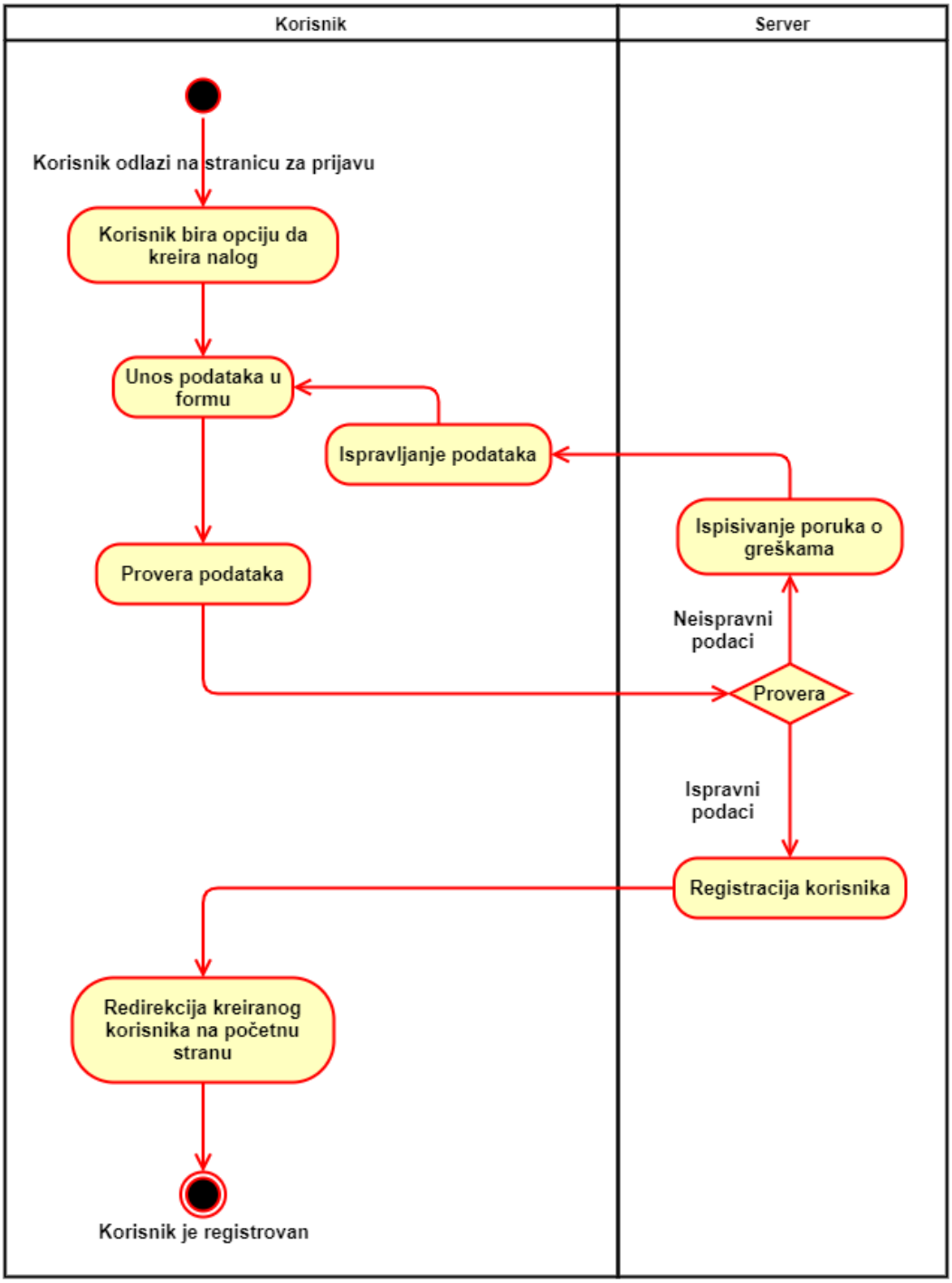
4.2.2 Početna strana sa opcijom pravljenja projekata i timova

Početna strana sa opcijom pravljenja projekata i timova je stranica koja se prikazuje nakon uspešne prijave korisnika. Ova stranica sadrži spisak svih projekata i timova koji su napravljeni i kojima korisnik ima pristup. Navigacija kroz aplikaciju se nalazi uz desnu ivicu stranice.

Svakom projektu je dodeljen jedan tim, a jedan tim može biti dodeljen većem broju projekata. Ako korisnik ne pripada nekom timu to znači da neće imati pristup nijednom projektu tog tima. Za pravljenje timova je potrebno definisati naziv, korisnika koji će biti administrator i izabrati one korisnike koji će biti u timu.

Prilikom pravljenja novog projekta se mora definisati naziv, administrator i tim koji će imati pristup projektu, odnosno tabli. Dodatna opcija je da se koristi *Pomodoro tehnika*. Ovu tehniku je krajem 1980-ih razvio Francesco Cirillo i glavna ideja je koristiti tajmer za podelu posla na intervale rada koji se nazivaju *pomodoro*, a nakon svakog intervala je kraća pauza [13]. Svakodnevno na poslu možemo imati veliki broj poziva ili e-mail pošte, a to sve nam odvlači pažnju i zbog toga lako gubimo fokus. Tehnika je veoma korisna za produktivnost, motivaciju i rad na

[3] Atlassian Jira https://www.atlassian.com/software/jira
[4] Microsoft Planner https://tasks.office.com/
[5] Trello https://trello.com/home



Slika 8: Dijagram aktivnosti registracije korisnika

zadatku bez ometanja. Ideja pomodoro tehnike u aplikaciji je da se korišćenjem tajmera korisnici fokusiraju samo na jedan zadatak. Na ovaj način će se ispuniti pravilo kanban sistema da se uvek radi samo na jednoj jedinici posla što je i osnova *pull* sistema. Optimalno vreme za intervale je 25 minuta i za pauze 5 minuta, a nakon 4 iteracije je veća pauza od 30 minuta. Parametri intervala rada, pauza i broja iteracija se mogu izmeniti.

4.2.3 Profilna strana korisnika

Profilna strana korisnika sadrži osnovne informacije koje je korisnik uneo prilikom registracije i do ove stranice se dolazi preko navigacije. Korisnik može dodati sliku na svoj profil. Na ovoj stranici korisnik ima uvid kojim timovima pripada, koje zadatke je označio kao omiljene i koji zadaci su mu dodeljeni da ih mora završiti.

4.2.4 Podešavanja

U okviru *Podešavanja* korisnik može prilagoditi određene delove aplikacije. Do ove sekcije se dolazi preko navigacije i ona obuhvata četiri podsekcije:

- Profil.
- Obaveštenja.
- Projekti.
- Oznake.

U podsekciji **Profil** se mogu izmeniti prethodno unete vrednosti za korisničko ime, prezime i ime, e-mail adresu i lozinku. Ako korisnik želi da izmeni lozinku tada je potrebno da unese trenutnu lozinku i novo-smišljenu lozinku. Ako trenutna lozinka nije dobra ili su trenutna i novo-smišljena lozinka iste korisniku će biti prikazane odgovarajuće poruke. Korisnik može obrisati svoj profil i tada će se vratiti na stranicu sa formom za prijavu. Ako tada pokuša da se prijavi sa prethodno izbrisanim kredencijalima to neće biti uspešno.

Podsekcija **Obaveštenja** omogućava svakom korisniku da izabere da li želi da prima određena obaveštenja u vidu e-mail poruka. Korisnik može dobiti obaveštenja u situacijama:

1. kada neko komentariše zadatak koji je korisnik napravio ili koji je dodeljen korisniku;
2. kada neko izmeni status zadatka koji je korisnik napravio ili koji je dodeljen korisniku;

3. kada neko izmeni zadatak koji je korisnik napravio ili koji je dodeljen korisniku.

Svaka e-mail poruka sadrži informaciju koji korisnik je napravio izmenu, šta je tačno izmena i u kom trenutku se desila.

U podsekciji **Projekti** korisnik može izmeniti podatke koje je uneo prilikom pravljenja projekta, a to su naziv, administrator, izabrani tim za projekat i da li će se koristi pomodoro tehnika. Ako se prethodno koristila tehnika i ako je tajmer za neki zadatak tog projekta pokrenut tada se podaci ne mogu izmeniti. Kada je projekat napravljen nije moguće napraviti zadatke zato što je neophodno definisati statuse. Za svaki status korisnik može odrediti tačan redosled prikazivanja kolona. Svaki korisnik ima mogućnost da promeni status zadatka na samoj stranici tog zadatka ili ako premešta karticu zadatka iz jedne kolone u drugu. Za računanje statistika je važno definisati koji status će značiti da je jedan zadatak konačno završen.

Podsekcija **Oznake** omogućava definisanje oznaka koje se mogu dodeliti nekom zadatku i one predstavljaju jedan vid tipa zadataka. Oznake su globalne, ne zavise od projekta i svaki zadatak može imati više od jedne oznake. Radi lakšeg razlikovanja, korisnik za svaku definisanu oznaku može izabrati određenu boju. Moguće je napraviti najviše osamnaest različitih oznaka.

4.2.5 Prikaz projekata i timova

Prikaz projekata i timova je početna stranica nakon prijave u aplikaciju. Stranica obuhvata spisak projekata i timova u obliku malih kartica.

Kartica projekta sadrži informaciju kom timu pripada, koliko procenata projekta je završeno i koliko ima ukupno zadataka. Pored toga, postoji dugme za preusmeravanje na stranicu sa statistikama i dugme za preusmeravanje na stranicu sa nedeljnim prikazom projekta.

Nedeljni prikaz projekta daje prikaz zadataka na osnovu početnog i krajnjeg datuma po nedeljama u godini. Svaki zadatak je boje dodeljenog prioriteta i trenutna nedelja prikaza se može menjati. Označavanjem jednog zadatka i pomeranjem levo ili desno se menjaju početni i krajnji datum, a u skladu sa tim će se nedeljni prikaz ažurirati. Klikom na dugme u obliku olovke se korisnik preusmerava na stranicu zadatka.

Na početnoj stranici, korisnik klikom na karticu jednog projekta će biti preusmeren na stranicu sa kanban-tablom izabranog projekta. Kolone table su u redosledu koji je izabran u podešavanjima projekta. Svaka kolona sadrži kartice sa osnovnim informacijama — kom korisniku je dodeljen zadatak, prioritet i oznake zadatka ako su dodeljene. Klikom na jedan zadatak se može pomerati između kolona, a klikom na dugme u okviru kartice se korisnik preusmerava na stranicu zadatka.

Iznad table se nalazi dugme za pravljenje zadataka i klikom se otvara forma sa poljima koje je potrebno popuniti. Moguće je filtrirati prikazane zadatke odabirom sledećih kriterijuma:

- dodeljeni korisnik zadataka;
- prioritet zadataka;
- oznaka zadataka.

Kartica tima na početnoj stranici sadrži informaciju o broju članova tima, koliko projekata pripada timu, ukupan broj zadataka u okviru svih projekata i koji korisnik je administrator tima. Klikom na karticu se korisnik preusmerava na stranicu tima gde su prikazane sve prethodno navedene informacije. Jedino administrator tima ima dozvolu da izbaci ili doda nekog člana tima.

4.2.6 Zadaci

Svaki projekat ima svoje *zadatke* i oni se mogu napraviti na stranici prikaza projekta. Prilikom pravljenja zadatka potrebno je popuniti sledeća polja:

1. *Naslov* — kratak opis šta je potrebno završiti;
2. *Opis* — detaljan opis šta sve treba da se uradi u okviru zadatka;
3. *Oznake* — tipovi zadatka;
4. *Dodeljeno* — korisnik koji treba da uradi zadatak;
5. *Status* — u kom statusu će se nalaziti zadatak nakon što je napravljen, i to ne sme biti predefinisan završni status projekta;
6. *Poeni* — vreme koje je potrebno da se završi zadatak izraženo u brojevima, i vrednosti su od 1 do 13;
7. *Prioritet* — koji prioritet ima zadatak, ukupno ih ima 5 i vrednosti se kreću od najvažnijeg do najnižeg prioriteta;
8. *Početni datum* — datum kada je potrebno započeti rad na zadatku;
9. *Krajnji datum* — datum do kada je potrebno završiti rad na zadatku.

Nakon što je zadatak napravljen biće prikazan u koloni table unetog statusa, i klikom na zadatak korisnik se preusmerava na stranicu sa svim podacima zadataka. Pored navedenih polja, postoji polje *zavisnost* koje se popunjava onim zadacima od kojih zavisi određeni zadatak. Vrednost ovog polja korisnik može popuniti, odnosno izmeniti, tek nakon što je zadatak napravljen. Kako bi se izbegla cirkularna zavisnost zadataka onemogućeno je da se zadatku A dodeli da zavisi od zadatka B ako zadatak B već zavisi od zadatka A. Spisak zavisnosti je prikazan iznad naslova zadatka i u okviru kog se pojedinačna zavinost može ukloniti.

Dodatna mogućnost je postavljanje komentara na pojedinačni zadatak. Na taj način korisnicima i članovima istog tima je omogućen neki vid komunikacije. Svaki komentar je moguće izmeniti ili obrisati. Ako je za projekat u okviru kog je napravljen zadatak uključena pomodoro tehnika postojaće dugme za pokretanje i zaustavljanje tajmera. Svaki korisnik može željene zadatke označiti kao omiljene i oni se prikazuju na profilnoj stranici korisnika.

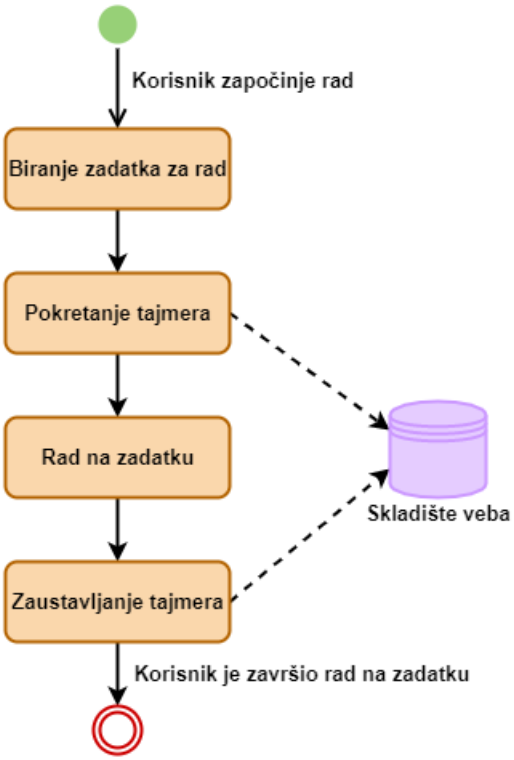
4.2.7 Tajmer

Tajmer je dodat zbog pomodoro tehnike. Dužine trajanja pomodora, pauza i broj iteracija zavise od projekta. Opcija da se pokrene tajmer je prikazana na stranici zadatka ako je omogućena pomodoro tehnika za taj projekat. Korisnik može da pokrene tajmer za samo jedan zadatak. Kada je tajmer pokrenut, vreme koje otkucava je prikazano u gornjem levom uglu aplikacije i klikom se preusmerava do stranice konkretnog zadatka. Nakon završetka poslednje velike pauze se prikazuje poruka i tada se ponovo ispočetka pokreće prvi pomodoro. U slučaju da korisnik ne želi da tajmer više otkucava potrebno je da ga zaustavi na stranici zadatka za koji je pokrenut. Na početku svakog intervala je dodato glasovno obavestenje u trajanju od jedne sekunde u slučaju da se korisnik nalazi na nekom drugom tabu pretraživača. Na slici 9 je prikazan dijagram procesa korišćenja tajmera.

4.2.8 Statistike

Statistike se izračunavaju za svaki projekat i biće prikazane samo ako je definisan završni status zadataka. Prikazani su sledeći dijagrami:

- broj zadataka po statusu;
- broj zadataka po oznaci;
- BurnUp dijagram;
- odnos broja zadataka koji su završeni na vreme, ranije ili kasnije.



Slika 9: Dijagram procesa korišćenja tajmera

BurnUp dijagram predstavlja odnos datuma završetka i broja poena nekog zadatka. Tek kada je neki zadatak prebačen u završni status biće ucrtan na dijagramu sa vrednostima poena i datumom završetka. Dijagram odnosa broja zadataka koji su završeni na vreme, ranije ili kasnije se računa u odnosu na krajni datum i datum završetka zadatka.

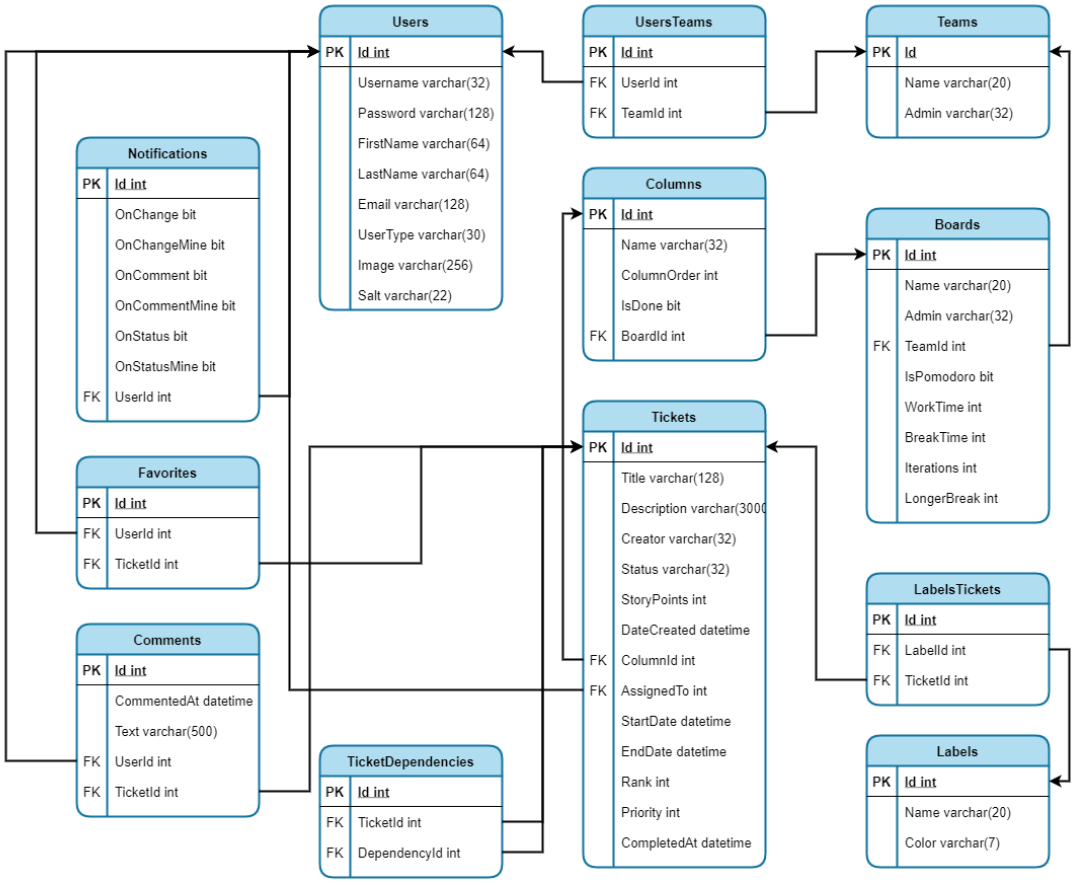
4.2.9 Vertikalni prikaz više projekata

Vertikalni prikaz više projekata se koristi za vizualizaciju projekata po nedeljama u godini i moguće je prikazati najviše tri projekta. Korisnik ne može izabrati da se prikazu dva ili tri ista projekta već je potrebno da se svi razlikuju. Zadaci svakog izabranog projekta su različitih boja radi lakše vizualizacije. Pomeranjem zadataka po danima u nedelji menjaju se vrednosti početnog datuma i krajnjeg datuma označenog zadatka. Ovaj prikaz nam omogućava da ako dva projekta zavise jedan od drugog možemo jasno videti kada će određeni zadaci biti započeti ili

završeni, da li je potrebno izmeniti datume nekog zadatka, da li je potrebno čekati da se neki zadatak završi i slično.

4.3 Baza podataka

Relaciona baza podataka *KanbanBoard* sadrži dvanaest tabela koje su potrebne za rad sa prethodno opisanom veb aplikacijom. Pored tabela koje čuvaju informacije potrebne za rad, kreirane su i vezne tabele radi ostvarivanja relacije **N:N**. Konkretna shema baze podataka se može videti na slici 10.



Slika 10: Dijagram baze podataka

Spisak sa kratkim opisom tabela koje čuvaju podatke definisanih entiteta u aplikaciji je:

- *Users* — čuva podatke o korisnicima koji su kreirali nalog, odnosno koji se mogu prijaviti u aplikaciji;

- *Teams* — čuva podatke o definisanim timovima;
- *Boards* — čuva podatke o definisanim projektima;
- *Columns* — čuva podatke o definisanim statusima svih projekata;
- *Tickets* — čuva podatke o napravljenim zadacima svih projekata;
- *Labels* — čuva podatke o definisanim oznakama zadataka;
- *Comments* — čuva podatke o postavljenim komentarima na nekom zadatku;
- *Favorites* — čuva podatke koji sve korisnici su označili zadatke kao omiljene;
- *Notifications* — čuva podatke o definisanim pravilima slanja obaveštenja svakog korisnika.

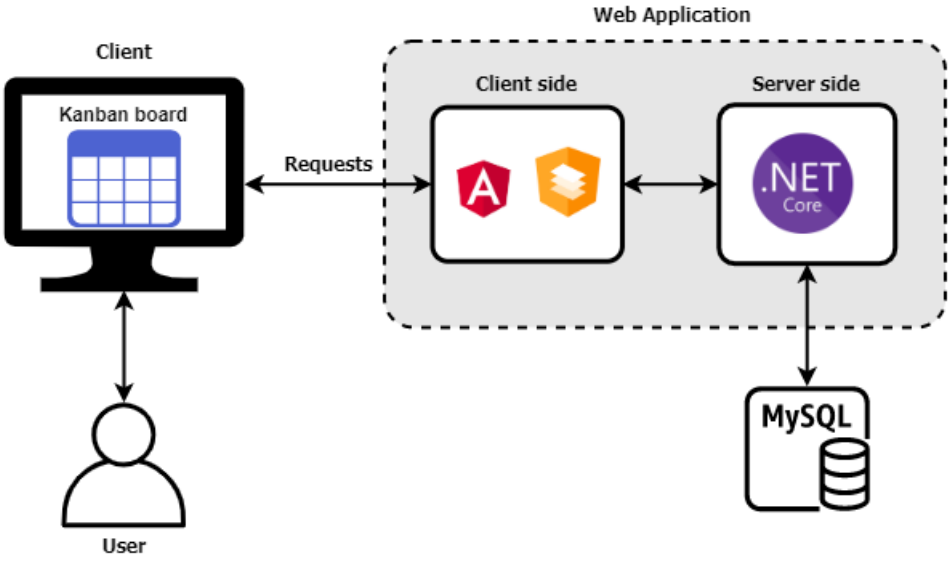
Relacija **1:N**, odnosno **jedan prema više**, je ostvarena dodavanjem stranih ključeva određenim tabelama. Spisak i kratak opis veznih tabela koje oslikavaju relaciju **N:N**, odnosno **više prema više**, je sledeći:

- *UsersTeams* — čuva podatke koji korisnik pripada kom timu i koristi se zato što korisnik može pripadati većem broju timova;
- *LabelsTickets* — čuva podatke koje oznake ima koji zadatak i koristi se zato što svaki zadatak može imati više oznaka;
- *TicketsDependencies* — čuva podatke koji zadatak zavisi od ostalih zadataka i koristi se zato što svaki zadatak može zavisiti od više zadataka.

4.4 Arhitektura aplikacije

Klijentski deo aplikacije je napisan u radnom okviru *Angular*, a serverski deo aplikacije u radnom okviru *.NET Core*. Komunikacija između klijenta i servera se odvija putem *API*-poziva, i korisnik ni na jedan način nema uvid u to kako se komunikacija tačno odvija. *API* predstavlja interfejs programske aplikacije koji definiše klijentu na koji način može komunicirati sa serverom. Za upravljanje relacionom bazom podataka u okviru aplikacije se koristi *MySQL Server*.

Kada korisnik izvrši neku izmenu u okviru klijentskog dela aplikacije *Angular* formira zahtev koji se šalje serverskom delu aplikacije *.NET Core*. Nakon obrađenog zahteva se podaci čitaju, upisuju ili brišu iz baze podataka. Komunikacija između servera i baze podataka se realizuje korišćenjem *ADO.NET*-klasa. Zatim se formira odgovor koji se šalje nazad klijentskom delu koji se u zavisnosti od odgovora ažurira. Takođe, u okviru radnog okvira *Angular* je korišćena biblioteka komponenata *Angular Material Design*. Na slici 11 je vizuelno prikazana prethodno opisana arhitektura aplikacije.



Slika 11: Arhitektura veb aplikacije

4.5 Detalji implementacije aplikacije

U ovom poglavlju će biti navedeni neki zanimljivi detalji implementacije aplikacije u radnim okvirima *.NET Core* i *Angular*.

4.5.1 Korišćenje lokalnog skladišta prgledača

Lokalno skladište pregledača (eng. local storage) je svojstvo *Window*-interfejsa kojim se pristupa objektu *Storage*, a podaci se čuvaju u okviru sesija pregledača. Razlika između lokalong skladišta i sesionog skladišta je što se sesiono skladište briše na kraju svake sesije, odnosno kada se zatvori pregledač, dok se lokalno skladište mora eksplicitno obrisati [14].

U implementiranoj aplikaciji se koristi lokalno skladište prilikom prijave korisnika. Kada se korisnik jednom uloguje i nakon zatvaranja pretraživača, po povratku na stranicu neće morati opet da se prijavljuje. Iz skladišta će se tek nakon odjavljivanja u okviru aplikacije obrisati podaci prethodno prijavljenog korisnika.

Kada korisnik koristi aplikaciju često se menjaju posećene stranice i nekada je potrebno lakše vratiti se unazad. Ovo je omogućeno dodavanjem novog polja u lokalnom skladištu koje čuva putanju prethodno posećene stranice. Nakon zatvaranja pretraživača i ponovnog pristupa aplikaciji, ako je korisnik ulogovan tada će biti preusmeren na stranicu sačuvane putanje u lokalnom skladištu.

Prilikom korišćenja tajmera se koristi lokalno skladište za čuvanje vrednosti početnog vremena i informacija o podešavanju tajmera koje su potrebne za računanje vremena. Kada korisnik zatvori pretraživač i ponovo pristupi aplikaciji, ako je bio pokrenut tajmer za neki zadatak vreme će nastaviti da se odbrojava tamo gde je stalo pre prekidanja sesije. Ako je tajmer prekinut tek tada će se vrednosti obrisati iz lokalnog skladišta.

4.5.2 Enkriptovanje lozinki

Za enkripciju podataka je korišćena funkcija heširanja *SHA-2*, tačnije iz radnog okvira *.NET Core* je korišćena klasa *SHA512CryptoServiceProvider*. Pre samog heširanja je korišćena tehnika *dodavanja soli* gde se svakoj lozinki doda nasumično izgenerisana niska, odnosno *so*. Ova tehnika omogućava da iste lozinke nemaju isti izračunati heš. Prvo se generiše *so* koja se dodaje na kraj prosleđene lozinke i novodobijena niska se pretvara u niz bajtova, a zatim se izračunava heš. Heš je izračunat kao niz bajtova koji se zatim pretvaraju u nisku, a novodobijena niska lozinke i niska *soli* se odvojeno čuvaju u bazi podataka. Konkretan algoritam heširanja lozinke se može videti u listingu 16.

```
public class HashingManager : IHashingManager
{
    private readonly SHA512CryptoServiceProvider
        sha512CryptoServiceProvider;
    private readonly IUserPersistenceManager
        userPersistenceManager;

    public HashingManager(IUserPersistenceManager
        userPersistenceManager)
    {
        this.userPersistenceManager = userPersistenceManager;
        sha512CryptoServiceProvider =
            new SHA512CryptoServiceProvider();
    }

    public (string, string) HashPassword(string password, int id)
    {
        string salt = GetSalt(id);
        byte[] hashedPassword = sha512CryptoServiceProvider.
            ComputeHash(StringToByteArray(password + salt));
        return (ByteArrayToString(hashedPassword), salt);
    }

    public byte[] StringToByteArray(string text)
    {
        return Encoding.UTF8.GetBytes(text);
    }
```

```
        public string ByteArrayToString(byte[] text)
        {
            StringBuilder stringBuilder = new StringBuilder();

            for (int i = 0; i < text.Length; i++)
            {
                stringBuilder.Append(text[i].ToString("x2"));
            }

            return stringBuilder.ToString();
        }

        private string GetSalt(int id)
        {
            if (id > 0)
            {
                return userPersistenceManager.LoadSalt(id);
            }

            Guid guid = Guid.NewGuid();
            string salt = Convert.ToBase64String(guid.ToByteArray());
            salt = salt.Replace("=","");

            return salt;
        }
    }
```

Listing 16: Heširanje šifre korisnika

4.5.3 Korišćenje protokola SMTP

Prethodno je opisano da je moguće podesiti da se određena obaveštenja šalju na e-mail adresu korisnika. Za slanje e-mail poruka se koristi protokol *SMTP* (eng. simple mail transfer protocol) i potreban je nalog čiji se kredencijali nalaze u datoteci *appsettings.json*. Podacima se pristupa pomoću obrasca za projektovanje *Singleton* koji se koristi kada želimo da imamo samo jednu instancu klase i da je ta instanca svima dostupna [11]. Kreirana je klasa *EmailConfiguration* koja se popunjava sa ključ-vrednost podacima iz *appsettings.json* datoteke, a zatim se registruje kao *singleton* u okviru konfiguracije servisa. Svi servisi dodati u okviru konfiguracije će biti dodati u kontejner zavisnosti. U listingu 17 je prikazan proces registrovanja *EmailConfiguration* kao *singleton*.

```
    // EmailConfiguration.cs file
    public class EmailConfiguration
    {
        public string From { get; set; }
```

```
        public string SmtpServer { get; set; }
        public int Port { get; set; }
        public string Username { get; set; }
        public string Password { get; set; }
    }

    // appsettings.json file
    "EmailConfiguration": {
        "From": "kanbanboard42@gmail.com",
        "SmtpServer": "smtp.gmail.com",
        "Port": 587,
        "Username": "kanbanboard42@gmail.com",
        "Password": "********"
    }

    // Startup.cs
    EmailConfiguration emailConfiguration = Configuration
        .GetSection("EmailConfiguration")
        .Get<EmailConfiguration>();

    services.AddSingleton(emailConfiguration);
```

Listing 17: Primer email konfiguracije

Kada se ispuni neki događaj za slanje obaveštenja, tada server prima POST zahtev sa sadržajem e-mail poruke. Koristi se klasa *SmtpClient*, a poruke se šalju asinhrono i u okviru poruka je omogućeno formatiranje pomoću HTML-elemenata.

4.5.4 Spisak implementiranih veb API-ja

Klijentski i serverski delovi aplikacije komuniciraju korišćenjem *API*-poziva, odnosno slanjem HTTP-zahteva. *HTTP* (eng. hypertext transfer protocol) predstavlja mrežni protokol za prenos hiperteksta u informacionim sistemima [15]. Ovaj protokol radi po principu *zahtev-odgovor*. To znači da klijentski deo aplikacije šalje zahtev, a serverski deo aplikacije obrađuje zahtev i u skladu sa tim šalje odgovor. Definisan je određen broj HTTP-metoda, a metode koje se trenutno u informacionim sistemima najčešće koriste su:

- ***GET*** — server vraća entitet/e u skladu sa zahtevom;
- ***POST*** — server kreira entitet koji je prosleđen u zahtevu;
- ***PUT*** — server ažurira entitet u skladu sa prosleđenim entitetom u zahtevu, ako entitet ne postoji tada će biti kreiran;
- ***PATCH*** — server parcijalno ažurira entitet u skladu sa prosleđenim entitetom u zahtevu;

- ***DELETE*** — server briše entitet koji je prosleđen u zahtevu.

U okviru aplikacije je razvijeno deset različitih kontrolera. Spisak svih putanja i kratak opis metoda koje obuhvata zaseban kontroler je dat u tabeli 1. Implementirane su osnovne *CRUD* [6] metode za čitanje, čuvanje, ažuriranje i brisanje podataka u zavisnosti od kontrolera.

Tabela 1: Spisak svih API kontrolera

Putanja	**Kratak opis metoda**
api/boards	obuhvata metode *CRUD* operacija za rad sa projektima, kao i dohvatanja osnovnih statistika
api/columns	obuhvata metode za dohvatanje, kreiranje i brisanje kolona, i ažuriranja redosleda kolona
api/comments	obuhvata metode *CRUD* operacija za rad sa komentarima
api/favorites	obuhvata metode *CRUD* operacija za rad sa omiljenim zadacima, kao i proveru da li je jedan zadatak označen kao omiljeni
api/helpers	metode za slanje e-mail poruka i otpremanje slika korisnika
api/labels	obuhvata metode *CRUD* operacija za rad sa oznakama zadataka
api/notifications	metode za čuvanje i čitanje podešavanja obaveštenja
api/teams	obuhvata metode *CRUD* operacija za rad sa timovima
api/tickets	obuhvata metode *CRUD* operacija za rad sa zadacima, s tim što se svako polje zadatka ažurira u zasebnom PUT zahtevu
api/users	obuhvata metode *CRUD* operacija, kao i zasebne metode za autentikaciju korisnika i proveru lozinke korisnika

4.5.5 Korišćenje biblioteke Angular Material

Prilikom instalacije radnog okvira *Angular* je instalirana i biblioteka *Angular Material*. Ova biblioteka se sastoji od komponenata i shema čijim korišćenjem je omogućeno brzo kreiranje korisničkog interfejsa koji je konzistentan kroz celu aplikaciju. Moguće je izabrati jednu od četiri predefinisane teme tokom instalacije

[6] *Create, Read, Update, Delete* (CRUD) predstavlja osnovne operacije za rad sa podacima u bazi podataka.

ili kasnije po želji dodati svoju temu. Na ovaj način se omogućava da se čitava aplikacija uklopi po bojama i to na veoma lak način.

Sve predefinsane komponente imaju prefiks *mat* i tačan spisak se može naći na zvaničnom sajtu dokumentacije bibilioteke [16]. Ako želimo da koristimo komponenete ove biblioteke potrebno je dodati odgovarajuće module u listu uvezenih modula u datoteci *app.module.ts*. Zatim je potrebno samo dodati željenu komponentu u okviru nekog šablona. Primer korišćenja komponente *<mat-progress-bar>* je dat u listingu 18.

```
<mat-progress-bar [mode]="determinate" [value]="42">
</mat-progress-bar>
```

Listing 18: Primer korišćenja komponente <mat-progress-bar>

Osnovna funkcionalnost kanban-table je da se kartice prevlače iz jedne u drugu kolonu i za ovo je iskorišćena shema *prevuci i otpusti* (eng. drag and drop). U listingu 19 je prikazan primer upotrebe ove sheme.

```
// statuses.component.html
<div cdkDropList class="status-list"
    (cdkDropListDropped)="drop($event)">
    <div class="status-item" *ngFor="let status of statueses">
        {{ status }}
    </div>
</div>

// statuses.component.ts
public statuses = ['To-Do', 'In Progress', 'QA', 'Done']
```

Listing 19: Primer upotrebe sheme prevuci i otpusti

4.5.6 Programski kôd kreiranja novih zadataka

U narednim listinzima će biti prikazan programski kôd koji opisuje funkcionalnost kreiranja novih zadataka. U listingu 20 je priložena sama komponenta, odnosno klasa, *AddTicketComponent*. Klasa sadrži metode za inicijalizaciju i validaciju forme i metodu za čuvanje koja zatim poziva serverski deo aplikacije. Listinzi 21 i 22 obuhvataju HTML-šablon i stil šablona komponente. U okviru HTML-šablona su korišćene komponente biblioteke *Angular Material* kao što su *mat-form-field*, *mat-select*, *mat-datepicker* i druge. Komponenta *AddTicketComponent* se poziva klikom na dugme *Create* koje se nalazi na stranici projekta i biće prikazana u iskačućem prozoru na ekranu. Kada je zadatak kreiran ili ako je korisnik odustao od kreiranja novog zadatka prozor će se zatvoriti.

```
@Component({
  selector: 'app-add-ticket',
```

```
  templateUrl: './add-ticket.component.html',
  styleUrls: ['./add-ticket.component.css']
})
export class AddTicketComponent implements OnInit {
  @Input() boardId: number;
  @Input() teamId: number;
  public ticketForm;
  public members: User[];
  public statuses: Column[];
  public points = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13];
  public priorities = Priorities;
  public prioritiesValues = Priorities.values();
  public labels: Label[];
  public minDate = new Date("1/1/1970");
  public maxDate = new Date("1/1/2050");

  constructor(private ticketService: TicketService,
              private formBuilder: FormBuilder,
              private userService: UserService,
              private columnService: ColumnService,
              private labelService: LabelService,
              private authService: AuthService,
              private snackBarService: SnackBarService) { }

  ngOnInit() {
    this.ticketForm = this.formBuilder.group({
      title: ['', { validators: [Validators.required,
      Validators.maxLength(128), Validators.nullValidator] }],
      description: ['', {validators: [Validators.required,
      Validators.maxLength(3000), Validators.nullValidator] }],
      assignedTo: ['', { validators: [Validators.required] }],
      status: ['', { validators: [Validators.required] }],
      storyPoints: ['', { validators: [Validators.required] }],
      priority: ['', { validators: [Validators.required] }],
      startDate: [''],
      endDate: [''],
      labels: ['']
    }, {
      validators: [this.validateStartDateEndDate()]
    });
    this.columnService.getColumnsByBoardId(this.boardId).subscribe(
    statuses => this.statuses = statuses.filter(x => !x.IsDone));
    this.userService.getUsersByTeamId(this.teamId).subscribe(users =>
    {
      this.members = users;
    });
    this.labelService.getLabels().subscribe(labels => this.labels =
    labels);
  }
```

```
  public save(data) {
    let ticket = new Ticket();
    ticket.Title = data.value.title;
    ticket.Description = data.value.description;
    ticket.AssignedTo = data.value.assignedTo;
    ticket.Creator = this.authService.getUsernameFromToken();
    ticket.Status = data.value.status[0];
    ticket.StoryPoints = data.value.storyPoints;
    ticket.DateCreated = new Date();
    ticket.ColumnId = data.value.status[1];
    ticket.BoardId = this.boardId;
    ticket.StartDate = data.value.startDate == "" ? null : data.value
    .startDate;
    ticket.EndDate = data.value.endDate == "" ? null : data.value.
    endDate;
    ticket.Priority = data.value.priority;
    ticket.CompletedAt = null;
    this.ticketService.getRankForColumn(ticket.ColumnId, ticket.
    BoardId).subscribe(rank => {
      if (rank > -1) {
        ticket.Rank = rank + 1;
        this.ticketService.addTicket(ticket).subscribe(ticketId => {
          if (ticketId > 0) {
            let labels = data.value.labels;
            for (let i = 0; i < labels.length; i++) {
              this.labelService.addLabelByTicketId(labels[i],
    ticketId).subscribe();
            }
            this.snackBarService.successful();
          } else {
            this.snackBarService.unsuccessful();
          }
        });
      } else {
        this.snackBarService.unsuccessful();
      }
    });
  }

  private validateStartDateEndDate() {
    return (control: AbstractControl) => {
      return control.value.startDate != "" && control.value.endDate
    != "" && control.value.startDate > control.value.endDate ?
        this.ticketForm.controls.startDate.setErrors({ '
    startDateGreaterThanEndDate': true }) : null };
  }
}
```

Listing 20: Komponenta AddTicketComponent

```
<h5 style="color: #3F51B5;">Create ticket</h5>
<mat-dialog-content>
  <form [formGroup]="ticketForm">
    <mat-form-field appearance="outline">
      <mat-label>Title</mat-label>
      <input matInput placeholder="Insert title" formControlName="
    title">
    </mat-form-field>

    <mat-form-field appearance="outline">
      <mat-label>Labels</mat-label>
      <mat-select formControlName="labels" multiple>
        <mat-option *ngFor="let label of labels" [value]="label" [
    style.color]="label.Color">
          {{ label.Name }}
        </mat-option>
      </mat-select>
    </mat-form-field>

    <mat-form-field appearance="outline">
      <mat-label>Description</mat-label>
      <textarea matInput formControlName="description"
    cdkTextareaAutosize="true" cdkAutosizeMinRows="10"
    cdkAutosizeMaxRows="20">
      </textarea>
    </mat-form-field>

    <mat-form-field appearance="outline">
      <mat-label>Assign To</mat-label>
      <mat-select formControlName="assignedTo">
        <mat-option *ngFor="let member of members" [value]="member.Id
    ">
           {{ member.FirstName }} {{ member.LastName }}
        </mat-option>
      </mat-select>
    </mat-form-field>

    <mat-form-field appearance="outline">
      <mat-label>Status</mat-label>
      <mat-select formControlName="status">
        <mat-option *ngFor="let status of statuses" [value]="[status.
    Name, status.Id]">
           {{ status.Name }} 
        </mat-option>
      </mat-select>
    </mat-form-field>
```

```
<div class="row">
  <div class="col">
    <mat-form-field appearance="outline">
      <mat-label>Story Points</mat-label>
      <mat-select formControlName="storyPoints">
        <mat-option *ngFor="let point of points" [value]="point">
          {{ point }}
        </mat-option>
      </mat-select>
    </mat-form-field>
  </div>
  <div class="col">
    <mat-form-field appearance="outline">
      <mat-label>Priority</mat-label>
      <mat-select formControlName="priority">
        <mat-option *ngFor="let priority of prioritiesValues" [
value]="priorities[priority]">
          {{ priority }}
        </mat-option>
      </mat-select>
    </mat-form-field>
  </div>
</div>

<div class="row">
  <div class="col">
    <mat-form-field appearance="outline">
      <mat-label>Start Date</mat-label>
      <input matInput [matDatepicker]="pickerStartDate" [min]="
minDate" [max]="maxDate" formControlName="startDate">
      <mat-datepicker-toggle matSuffix [for]="pickerStartDate"></
mat-datepicker-toggle>
      <mat-datepicker #pickerStartDate></mat-datepicker>

      <mat-error *ngIf="ticketForm.get('startDate').errors &&
ticketForm.get('startDate').errors.startDateGreaterThanEndDate">
        Start Date can't be bigger than End Date.
      </mat-error>
    </mat-form-field>
  </div>
  <div class="col">
    <mat-form-field appearance="outline">
      <mat-label>End Date</mat-label>
      <input matInput [matDatepicker]="pickerEndDate" [min]="
minDate" [max]="maxDate" formControlName="endDate">
      <mat-datepicker-toggle matSuffix [for]="pickerEndDate"></
mat-datepicker-toggle>
      <mat-datepicker #pickerEndDate></mat-datepicker>
```

```
        </mat-form-field>
      </div>
    </div>
  </form>
</mat-dialog-content>
<mat-dialog-actions align="center">
  <button mat-raised-button type="submit" color="primary" [mat-dialog
   -close]="true" [disabled]="!ticketForm.valid" (click)="save(
   ticketForm)">Save</button>
  <button mat-button mat-dialog-close>Cancel</button>
</mat-dialog-actions>
```

Listing 21: Šablon komponente AddTicketComponent

```
.mat-form-field {
  display: block;
}

.mat-error {
  text-align: center;
}
```

Listing 22: Stil šablona komponente AddTicketComponent

4.6 Poređenje aplikacije sa već postojećim aplikacijama

Ideje za funkcionalnosti koje su implementirane u aplikaciji TT-Kanban su nastale prilikom korišćenjea već postojećih aplikacija kao što su *Trello* i *Atlassian Jira*. U tabeli 2 redovi predstavljaju funkcionalnosti dok kolone predstavljaju postojeće aplikacije. U svakoj ćeliji je upisano *da* ili *ne* u zavisnosti da li određena aplikacija podržava navedenu funkcionalnost. Važno je napomenuti da aplikacije *Trello* i *Redmine* podržavaju *umetanje dodataka* (eng. plugins) koje omogućavaju korišćenje dodatnih funkcionalnosti koje ne postoje u okviru osnove aplikacije.

Tabela 2: Poređenje funkcionalnosti aplikacije TT-Kanban i već postojećih aplikacija

	Trello	**Redmine**	**Atlassian Jira**	**TT-Kanban**
prevlačenje kartica	da	da	da	da
statusi koje definiše korisnik	da	da	da	da
postojanje timova i projekata	da	da	da	da
komentari na zadacima	da	da	da	da
prikaz zadataka u kalendaru	da	da	ne	da
prikaz statistika	da	da	da	da
korišćenje tajmera tokom rada na zadatku	da	ne	ne	da
Gantov dijagram	da	da	ne	ne
integracija sa sistemima za kontrolu verzija	da	da	da	ne
oznake zadataka	da	da	da	da
postojanje sprintova-/iteracija	da	da	da	ne

5 Zaključak

Kanban-table su veoma popularan alat za organizaciju poslova, kako u svetu razvijanja aplikacija, tako i u privatnom životu za organizaciju obaveza. Neverovatna brzina napretka tehnologija je uslovila napredak u procesu razvoja softvera i zato metodologije razvoja softvera danas predstavljaju najvažniji korak kada se započinje projekat. Jedna zanimljivost je da je tokom razvoja aplikacije *TT-Kanban* takođe bila korišćena kanban-tabla za definisanje zadataka i upravljanje projektom ovog rada.

5.1 Rezime urađenog posla

Cilj ovog rada je bio razviti veb aplikaciju *TT-Kanban* koja se može koristiti u okviru procesa razvoja softvera. Opisane su metodologije razvoja softvera i prikazan je proces integracije kanban sistema i kanban-tabli u svet programiranja. Korišćenjem radnih okvira *.NET Core* i *Angular* je omogućeno da se arhitektura veb aplikacije postavi brzo, a zatim zavhaljujući opisanim karakteristikama radnih okvira da sam razvoj teče lako. Detaljnim opisom aplikacije i korisničkih celina je prikazano na koji način je moguće koristiti razvijenu aplikaciju.

5.2 Moguća dalja unapređenja aplikacije TT-Kanban

Aplikacije kao što su *Redmine*, *Atlassian Jira* i *Trello* imaju u osnovi slične funkcionalnosti, ali pored toga imaju mogućnost prilagođavanja delova aplikacije prema potrebama korisnika. Vođeni idejom prilagođavanja aplikacije potrebama su predložena sledeća unapređenja:

- **Lični projekti** — Ideja je da aplikaciju korisnik može da koristi za svoje lične svakodnevne obaveze i da na taj način projekat ne mora da zavisi od tima.
- **Integracija sa sistemom *GitHub*** - U razvoju softvera se koriste sistemi kontrole verzija i jedan od popularnijih je *Git*. *GitHub* je veb platforma za kontrolu verzija koda. U okviru *GitHub*-a postoje repozitorijumi koji na neki način predstavljaju projekte. Ideja je da se jedan projekat u aplikaciji može povezati sa jednim repozitorijumom na *GitHub*-u. Bilo bi potrebno da se za svaki repozitorijum podese *Webhooks*. *Webhooks* osluškuju događaje u okviru repozitorijuma i šalju POST zahteve aplikaciji putem definisane putanje. Na ovaj način bi bilo moguće da kreiranjem grane na *GitHub*-u određeni zadatak promeni svoj status.

- **Sprintovi** — Zamisao je da kada se definiše projekat postoje početni i krajnji datum na osnovu kojih bi se računali sprintovi. Takođe, bilo bi moguće definisati dužinu svakog sprinta.
- **Dodavanje datoteka i slika** — Omogućilo bi se dodavanje datoteka i slika u opisima zadataka i komentarima.
- **Testovi** — Ideja je da se serverski deo aplikacije testira automatskim *unit* testovima. *Unit* testovi proveravaju jednu *"jedinicu"* u okviru aplikacije. Obično jedan *unit* test predstavlja testiranje jedne metode. Prednost testiranja je da na neki način imamo validaciju da svi delovi aplikacije rade kako je i predviđeno.
- **Mikroservis sa radnim okvirom Kafka** — Ako bi se dodala integracija sa sistemom *GitHub*, postoji mogućnost da se doda zaseban mikroservis koji bi koristio radni okvir *Kafka*. Ovaj mikroservis bi obrađivao zahteve poslate od strane *GitHub*-a i u skladu sa tim ažurirao aplikaciju.
- **PATCH zahtev za ažuriranje zadataka** — Trenutno se svako polje zadatka ažurira slanjem zasebnih *PUT* zahteva što se može unaprediti korišćenjem *PATCH* zahteva.
- **Docker-kontejner** — Upakovati aplikaciju u Linux/Windows kontejner koji se može pokrenuti na servisu Azure Kubernetes.
- **Statistika odnosa broja zadataka po statusu i vremenu** — Ideja je da se čuvaju informacije kada neki zadatak promeni status i u kom tačno trenutku. Na osnovu ovih podataka može da se kreira statistika koja predstavlja broj zadataka po statusima u odnosu na vreme. Na ovaj način bi postojao uvid da li se tokom projekta troši više vremena na neke određene faze.

Literatura

[1] Manifesto for Agile Software Development. on-line at: https://agilemanifesto.org/.

[2] Martin C. Robert. *Agile Software Developemn - Principles, Patterns and Practices*. Pearson Education, 2003.

[3] Larman Craig. *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley, 2004.

[4] Ohno Taiichi. *Toyota Production System: Beyond Large-Scale Production*. Productivity Press, 1988.

[5] Anderson David J. *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, 2010.

[6] *.NET Documentation*. on-line at: https://docs.microsoft.com/en-us/dotnet/.

[7] Landwerth Immo. Introducing .NET Standard. on-line at: https://devblogs.microsoft.com/dotnet/introducing-net-standard/.

[8] *Docker*. on-line at: https://docs.docker.com/.

[9] *Angular Documentation*. on-line at: https://angular.io/docs.

[10] AngularJS Github Repository. on-line at: https://github.com/angular/angular.js.

[11] Gamma Erich, Vlissides John, Helm Richard and Johnson Ralph. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[12] *RxJs library*. on-line at: https://rxjs.dev/.

[13] Cirillo Francesco. Pomodoro technique. on-line at: https://francescocirillo.com/pages/pomodoro-technique.

[14] *MDN Web Docs - Mozilla*. on-line at: https://developer.mozilla.org/en-US/.

[15] Fielding Roy, Gettys Jim, Mogul Jeffrey, Frystyk Henrik, Masinter Larry, Leach Paul. *Hypertext Transfer Protocol – HTTP/1.1*, 1999. on-line at: https://www.ietf.org/rfc/rfc2616.txt.

[16] *Angular Material*. on-line at: https://material.angular.io/.

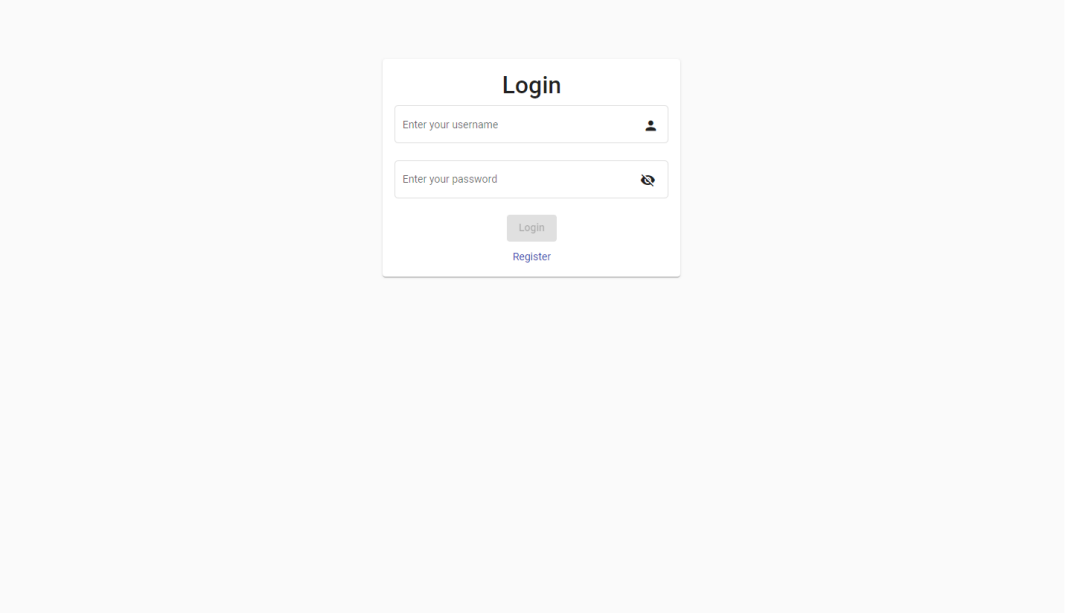
A Dodatak

A.1 Pokretanje aplikacije

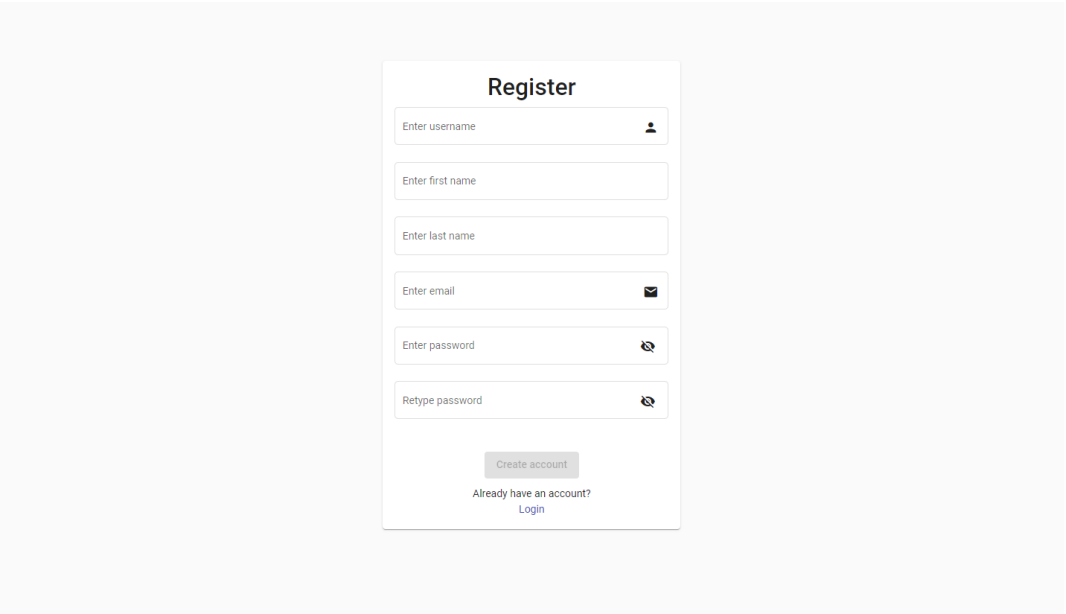
Za pokretanje aplikacije na računaru je neophodno instalirati *.NET*, *Node.js*, *Angular* i *Angular Material* biblioteku. Na zvaničnim sajtovima navedenih radnih okvira postoje detaljna uputstva za instalaciju. Kod je moguće preuzeti iz direktorijuma *TT-Kanban* koji je priložen kao dodatak elektronskoj verziji rada ili sa adrese `https://github.com/tijanatosev/master-project`. U okviru direktorijuma *TT-Kanban/KanbanBoard/* postoji datoteka *database.sql* koju treba iskoristiti za kreiranje baze. Pored navedenih radnih okvira je potrebno instalirati *MySql Server* i zatim izmeniti sekciju *ConnectionStrings* u datoteci *appsettings.json* u skladu sa instaliranim *MySQL* serverom i kreiranom bazom. Za komunikaciju sa *MySQL* bazom je potrebno instalirati paket u okviru *.NET*-a pomoću komande **dotnet add package MySqlConnector**.

Nakon pozicioniranja u direktorijum *TT-Kanban/KanbanBoard/KanbanBoard* je potrebno izvršiti komande **dotnet build** i **dotnet run** iz *Command Prompt* terminala i zatim se pristupa adresi `https://localhost:5001/` iz pretraživača.

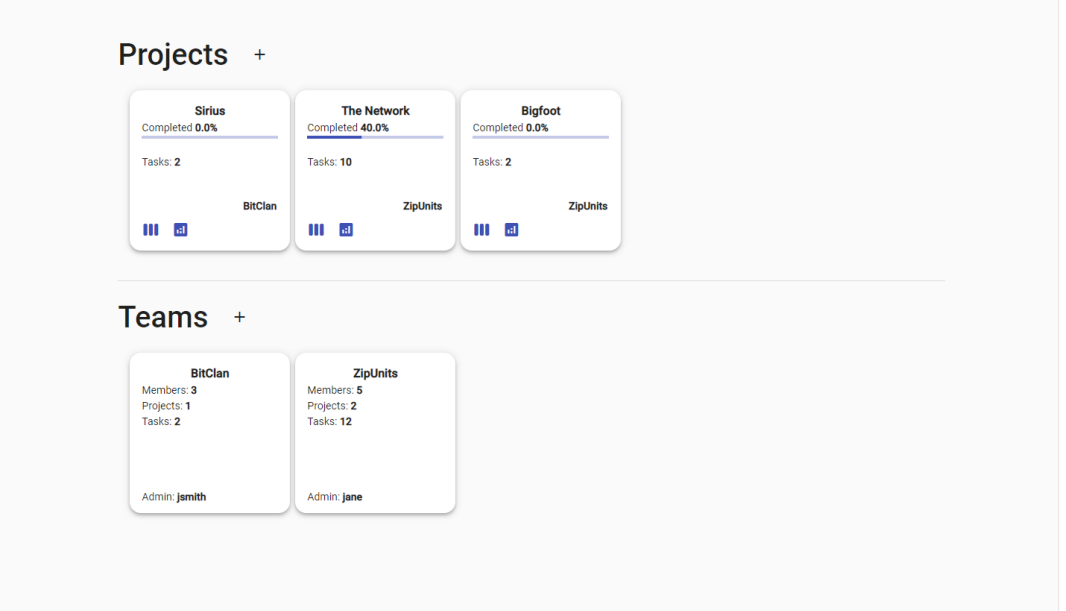
A.2 Slike korisničkog interfejsa aplikacije TT-Kanban



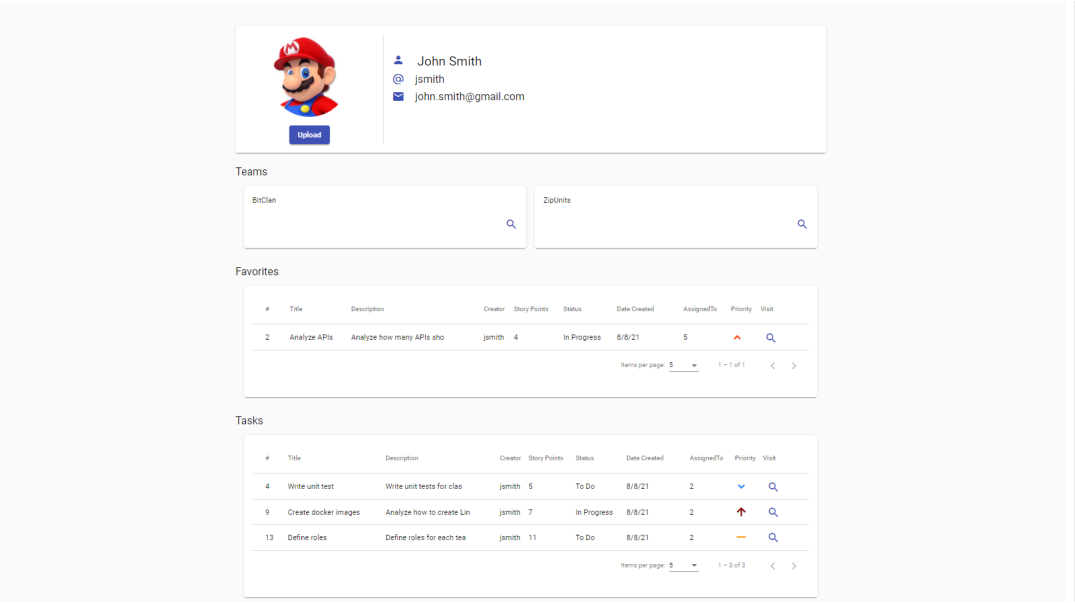
Slika 12: Stranica za prijavu korisnika



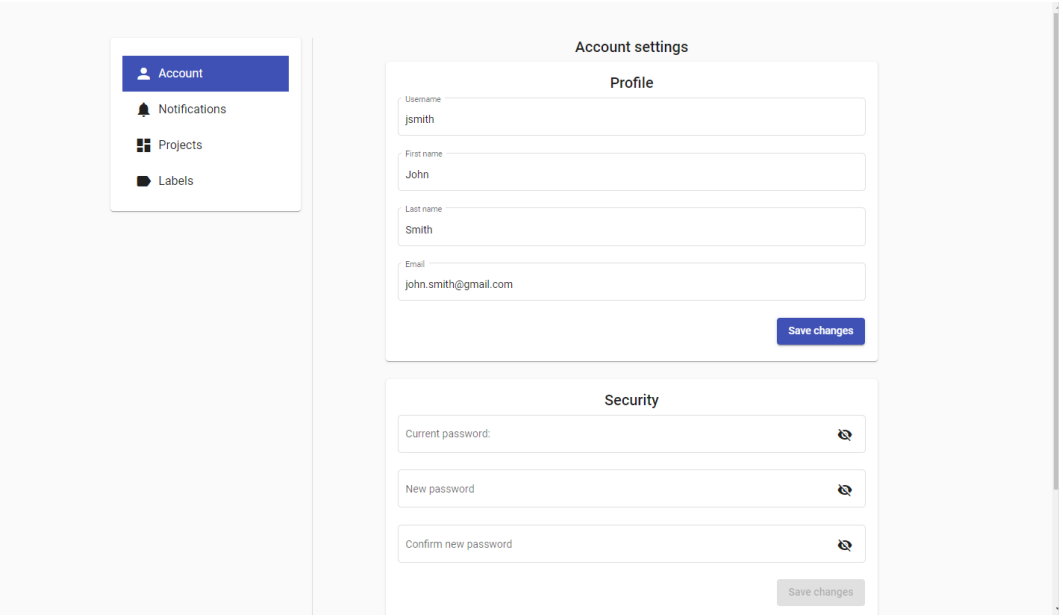
Slika 13: Stranica za registraciju korisnika



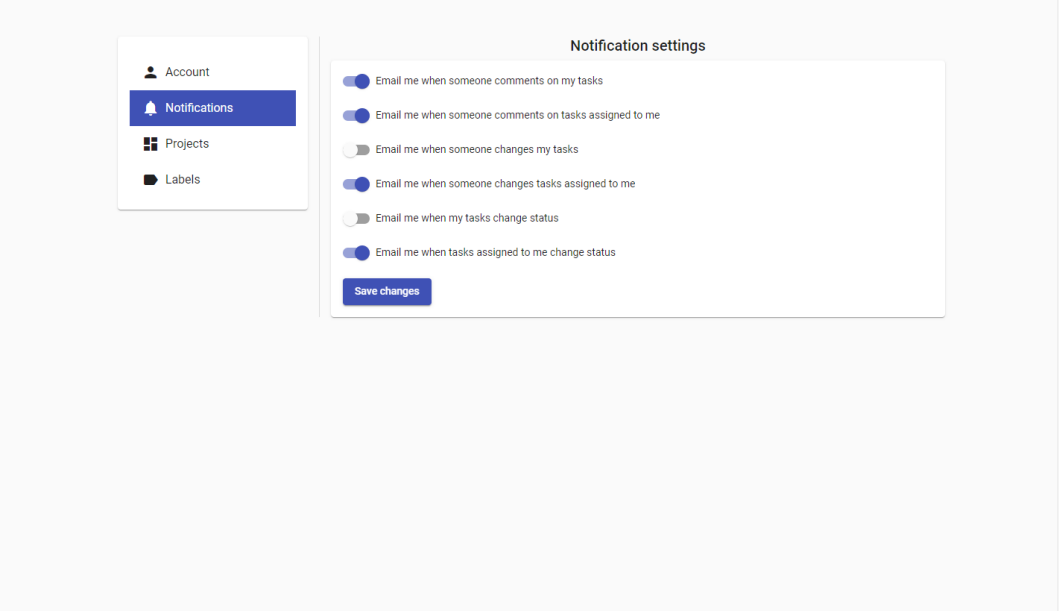
Slika 14: Početna stranica



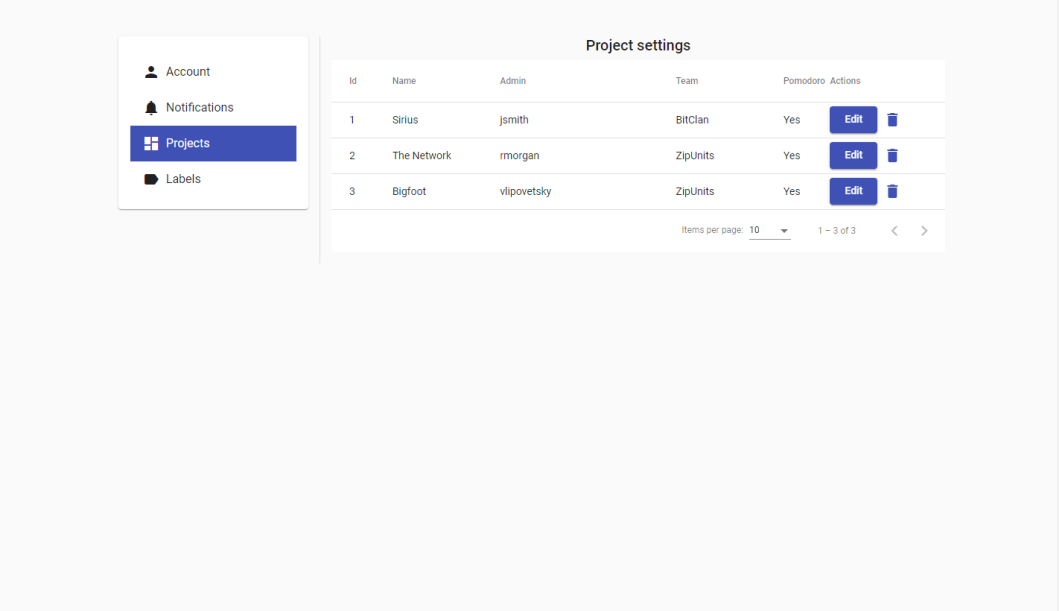
Slika 15: Profilna stranica korisnika



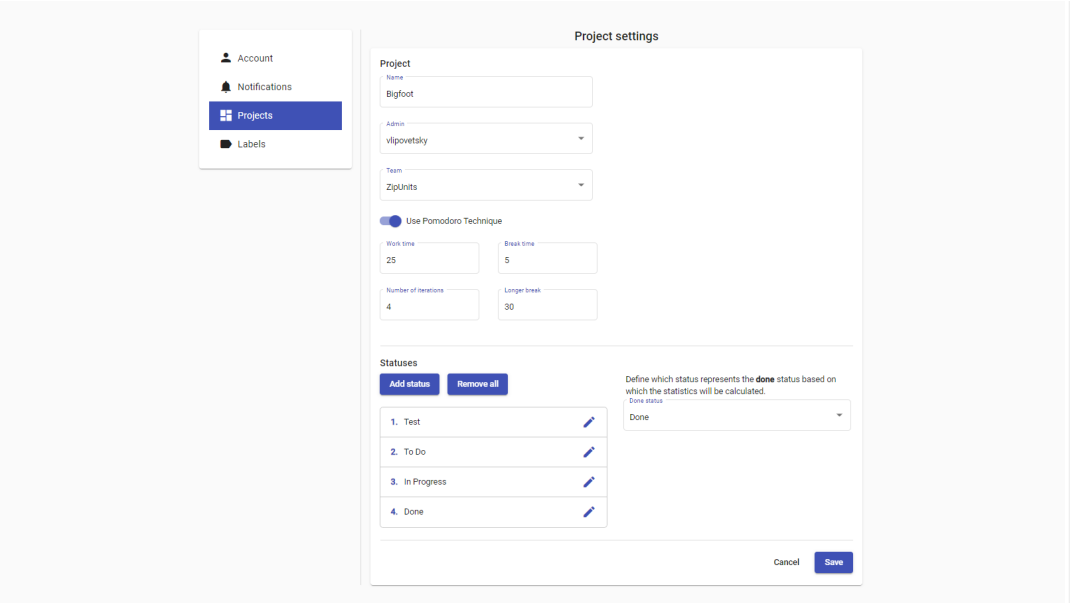
Slika 16: Stranica za podešavanje korisničkog naloga



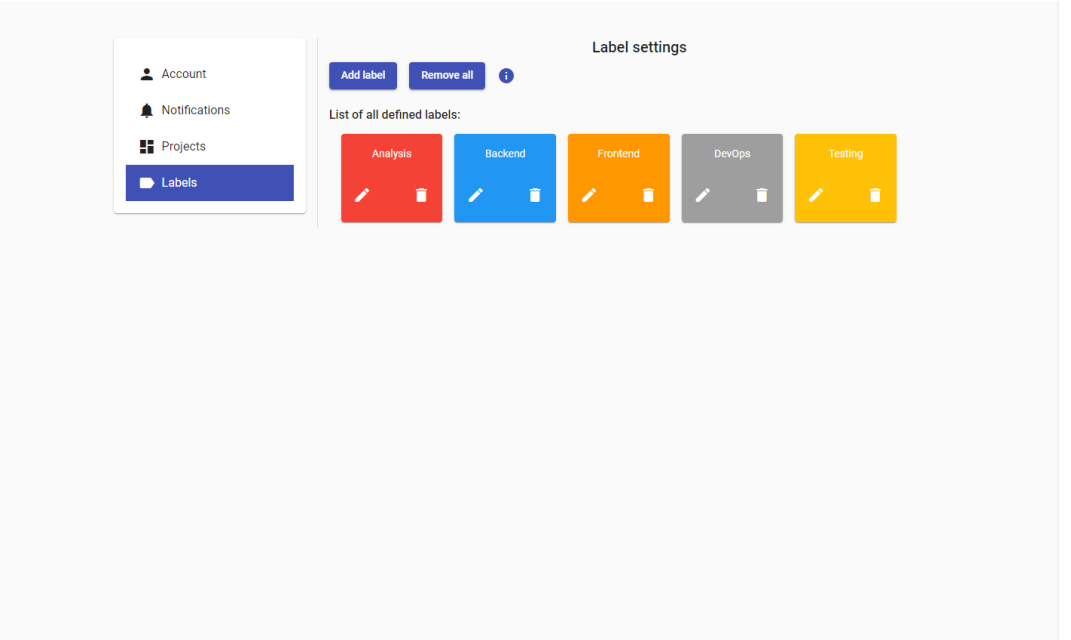
Slika 17: Stranica za definisanje obaveštenja



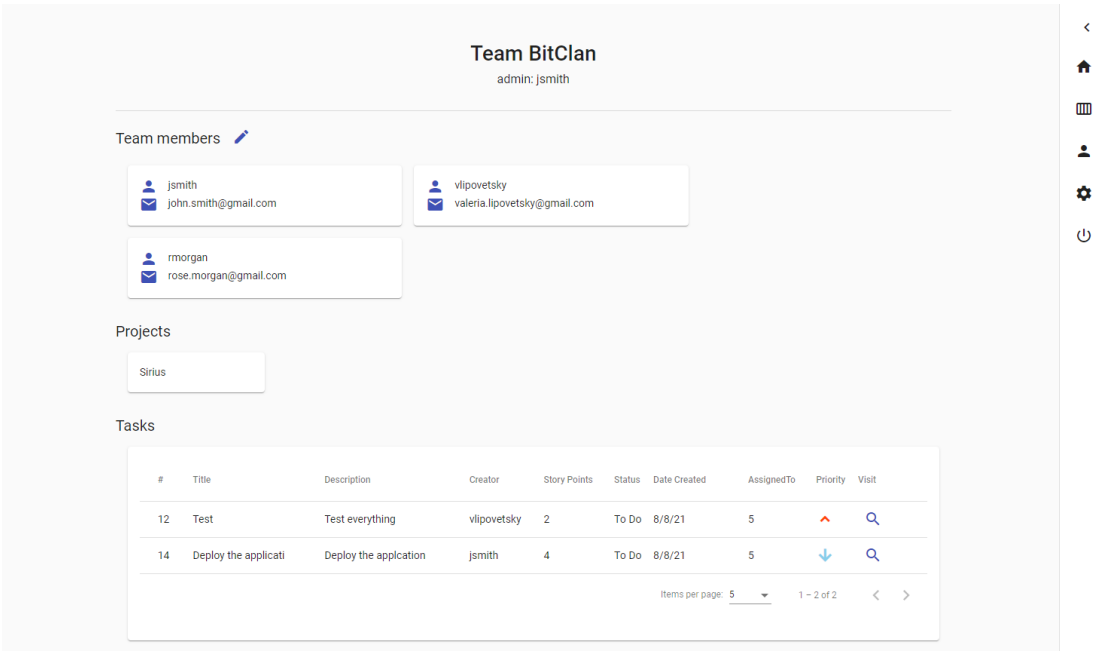
Slika 18: Stranica za podešavanje projekata



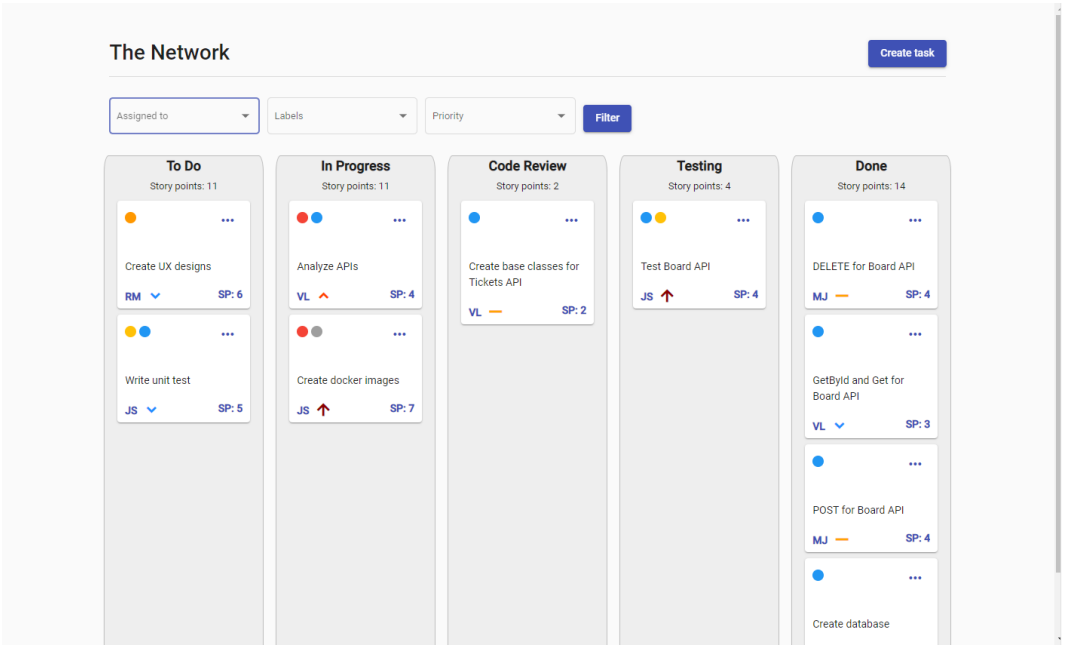
Slika 19: Stranica za podešavanje jednog projekta



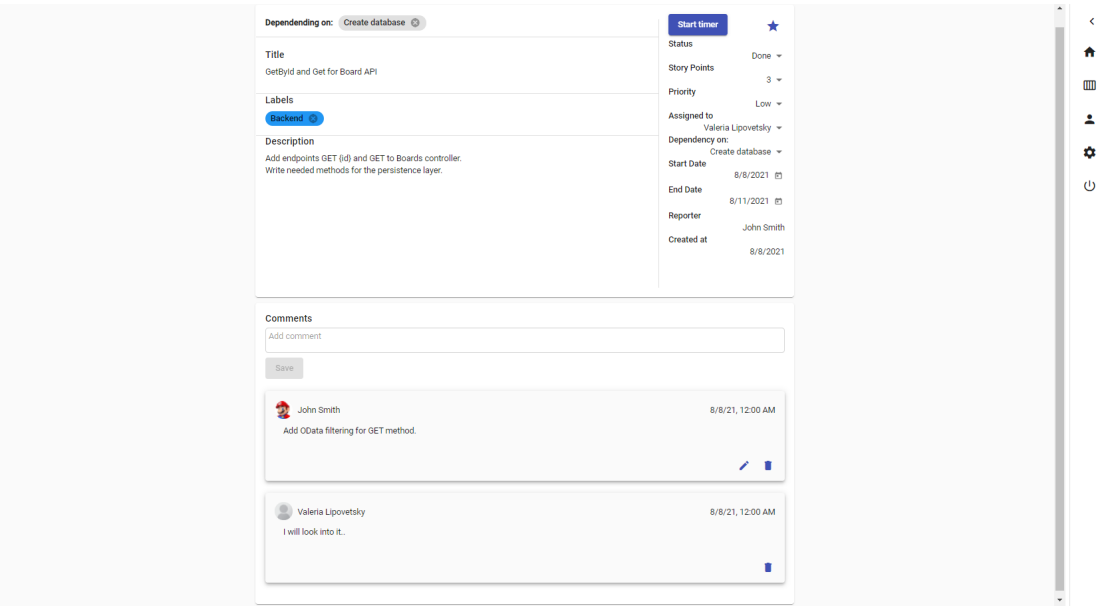
Slika 20: Stranica za definisanje oznaka



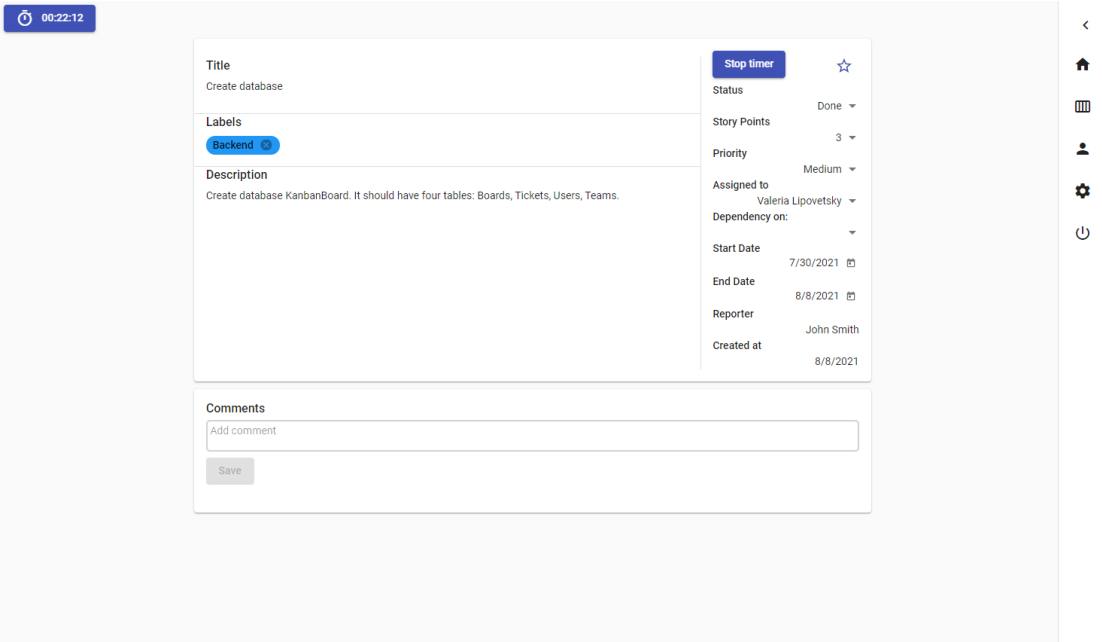
Slika 21: Stranica tima



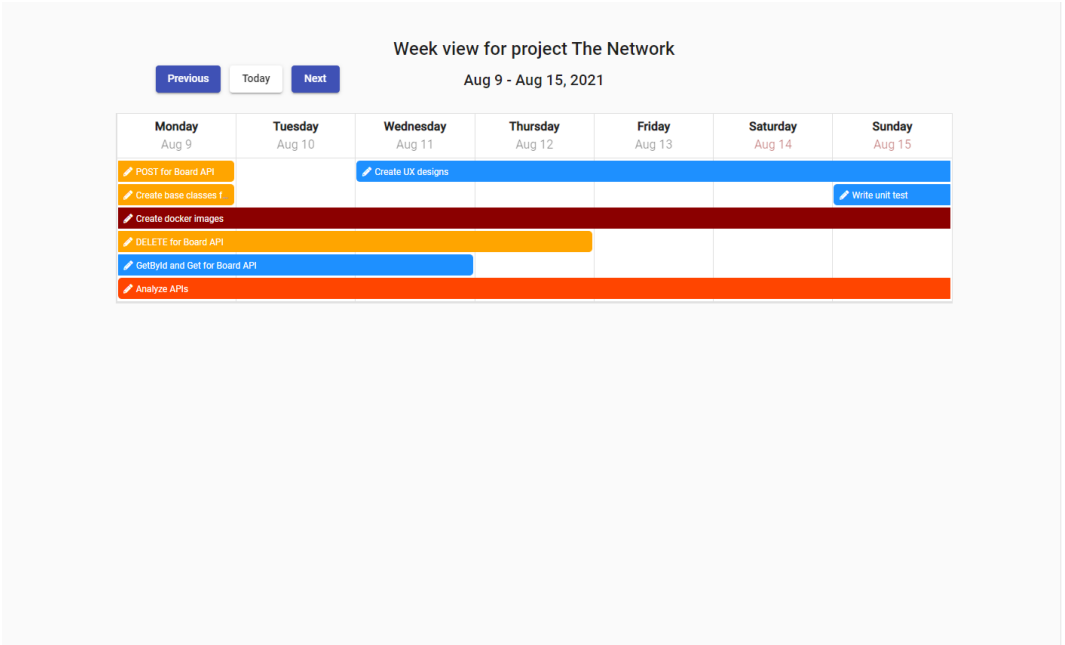
Slika 22: Stranica projekta



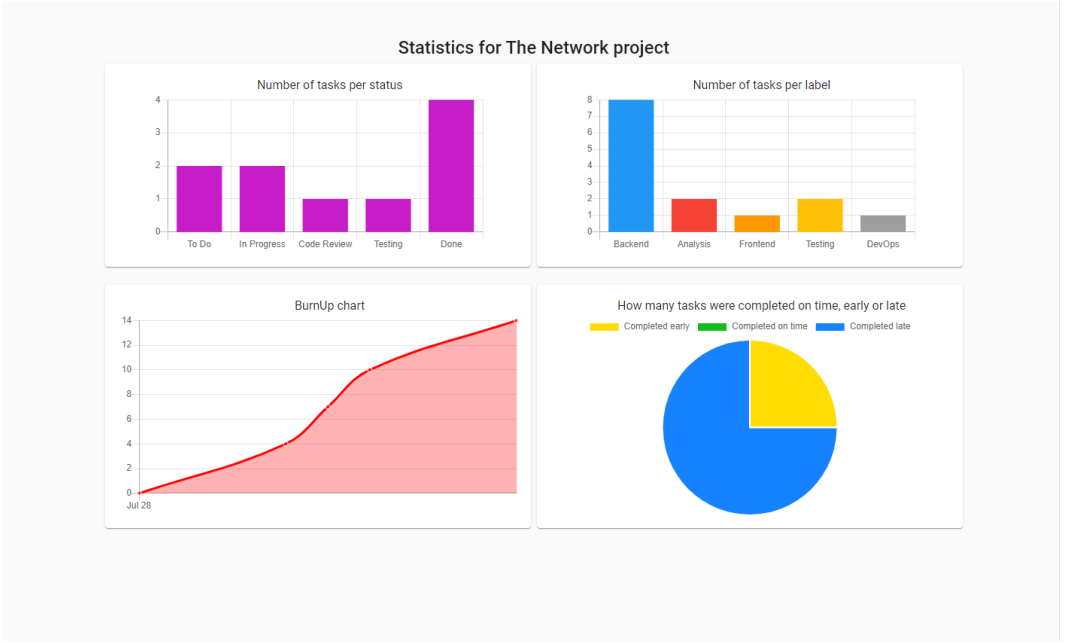
Slika 23: Stranica zadatka



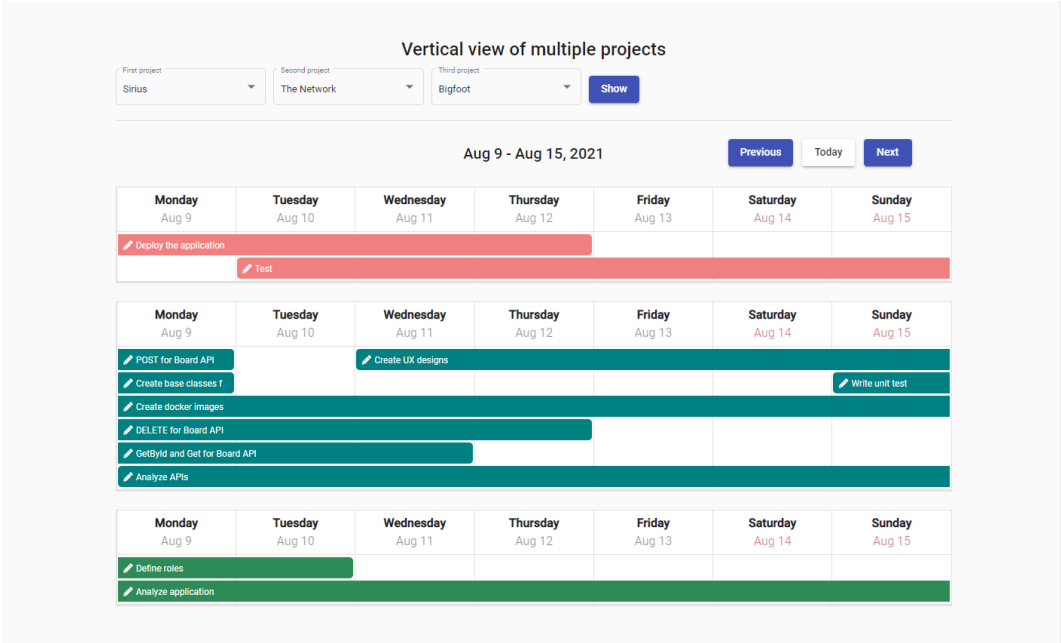
Slika 24: Stranica zadatka sa pokrenutim tajmerom



Slika 25: Stranica nedeljnog prikaza projekta



Slika 26: Stranica izračunatih statistika jednog projekta



Slika 27: Stranica vertikalnog prikaza više projekata