Programsko rješenje korisničkog dijela web sustava za potporu upravljanja projektima agilnog razvoja programske podrške prilagođenog osobama s oštećenjima vida

Deskar, Samanta

Master's thesis / Diplomski rad

2022

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

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:251225

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

PROGRAMSKO RJEŠENJE KORISNIČKOG DIJELA WEB SUSTAVA ZA POTPORU UPRAVLJANJA PROJEKTIMA AGILNOG RAZVOJA PROGRAMSKE PODRŠKE PRILAGOĐENOG OSOBAMA S OŠTEĆENJIMA VIDA

Diplomski rad

Samanta Deskar

Osijek, 2022.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 13.05.2022.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Samanta Deskar
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1046R, 06.10.2019.
OIB studenta:	81761458937
Mentor:	Prof.dr.sc. Goran Martinović
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 1:	Prof.dr.sc. Goran Martinović
Član Povjerenstva 2:	Izv. prof. dr. sc. Alfonzo Baumgartner
Naslov diplomskog rada:	Programsko rješenje korisničkog dijela web sustava za potporu upravljanja projektima agilnog razvoja programske podrške prilagođenog osobama s oštećenjima vida
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U teorijskom dijelu diplomskog rada potrebno je opisati pristupe i izazove vođenja projekata agilnog razvoja programske podrške, te drugih aktualnih postupaka korištenih u agilnom i drugim pristupima razvoja programske podrške (kanban, Scrum, stand-up i drugi). Također, treba opisati probleme s kojima se u korištenju računalnih aplikacija susreću osobe s oštećenjem vida, kao što su slabovidne osobe, osobe s disleksijom, daltonizmom i drugim nedostacima, a te probleme treba analizirati i s gledišta upravljanja projektima agilnog razvoja, stvaranja korisničkog sučelja i korisničkog iskustva. Uzimajući u obzir poslužiteljski dio sustava, zahtjeve za uspješno vođenje projekata agilnog razvoja programske podrške i vremenske prostorni
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	13.05.2022.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 19.05.2022.

Ime i prezime studenta:	Samanta Deskar
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1046R, 06.10.2019.
Turnitin podudaranje [%]:	6

Ovom izjavom izjavljujem da je rad pod nazivom**: Programsko rješenje korisničkog dijela web sustava za potporu upravljanja projektima agilnog razvoja programske podrške prilagođenog osobama s oštećenjima vida**

izrađen pod vodstvom mentora Prof.dr.sc. Goran Martinović

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Projektni menadžer ima mnoge odgovornosti i zaduženja, a u današnje vrijeme pomoć pri organizaciji projekta pružaju mu različiti alati. Neki od alata koji pomažu projektnim menadžerima voditi projekte agilnim metodama su Asana, Jira, Monday i slični. Svaki od alata do određenog stupnja pruža mogućnosti prilagodbe osobama s oštećenjima vida, ali i uz određene prilagodbe, projektni menadžeri i ostali zaposlenici susreću se s raznim izazovima.

Iako su svi postojeći alati vrlo korisni prilikom vođenja projekata, rijetki od njih su u potpunosti prilagođeni osobama s oštećenjima vida. Iz tog razloga, tema i cilj ovog projekta, u sklopu diplomskog rada, je razviti programsko rješenje s korisničke strane za vođenje projekata agilnim načinom prilagođeno osobama s oštećenjima vida. Programsko rješenje bit će prilagođeno osobama koje boluju od daltonizma, disleksije, astigmatizma i sličnih oštećenja vida. Razvojem korisničke strane programskog rješenja, korisnici će na vrlo lak način moći prilagoditi alat svojim potrebama i u potpunosti iskoristiti sve mogućnosti ovog alata za vođenje projekata agilnim načinom. Kako bi programsko rješenje u potpunosti bilo ostvareno, razvijena je i poslužiteljska strana, koja je obrađena u diplomskom radu pod nazivom „Programsko rješenje poslužiteljskog dijela web sustava za potporu upravljanja projektima agilnog razvoja programske podrške zasnovano na višenitnom paralelizmu i arhitekturi mikrousluga“ [1].

U drugom poglavlju diplomskog rada opisane su osnove agilnog razvoja i postojeća rješenja za vođenje projekata agilnim metodama. Treće poglavlje opisuje oštećenja vida na primjerima daltonizma, disleksije i astigmatizma te mogućnosti prilagodbe pristupačnosti za navedena oštećenja vida. Analiziraju se i postojeća rješenja, njihove mogućnosti prilagodbe pristupačnosti i mogućnosti poboljšanja prilagodbe osobama s oštećenjima vida. U četvrtom poglavlju definiraju se zahtjevi korisničke strane programskog rješenja, model korisničkog sučelja te arhitektura. Peto poglavlje opisuje programske tehnologije korištene za razvoj aplikacija korisničke strane kao što su programski jezik *Javascript,* programska biblioteka *React* i testni okvir *Jest*. U šestom poglavlju prikazan je i objašnjen dizajn korisničkog sučelja, razvoj programskog rješenja korisničke strane sustava i testiranje na razini jedinica kôda. Sedmo poglavlje sadrži opis rada programskog rješenja te ispitivanje i analizu rada programskog rješenja.

2. VOĐENJE PROJEKATA AGILNOM METODOM RAZVOJA PROGRAMSKE PODRŠKE

2.1. Osnove agilnog razvoja i stanje u području

Pojam agilni razvoj podrazumijeva vođenje projekata razdvajanjem na manje faze. Takav način vođenja projekata odmiče se od linearnih modela i uvodi kružne i fleksibilne procese dizajniranja, razvoja i testiranja [2]. „Proglas o metodi agilnog razvoja softvera“ [3] navodi četiri glavne vrijednosti koje se cijene:

1. Ljudi i njihovi međusobni odnosi, a ne procesi i oruđa
2. Upotrebljivo programsko rješenje, a ne iscrpna dokumentacija
3. Suradnja s naručiteljem, a ne pregovaranje oko ugovora
4. Reagiranje na promjenu, a ne ustrajanje na planu.

Agilni razvoj prvi put se pojavljuje ranih 1990-ih godina kada je programerska industrija počela naglo rasti. U veljači 2001. godine, 17 individualnih sudionika stvorilo je jednostavan pristup razvoju programskih rješenja s osnovnim vrijednostima koje se cijene [4]. Praćenjem navedenih vrijednosti i principa, timovi koriste kratke, regularne i kontrolirajuće iteracije koje se nazivaju *sprint*. *Sprint* obično traje od 2 do 4 tjedna te se unutar tog vremena odrađuje puni razvojni ciklus [4]. Razvojni ciklus uključuje planiranje, dizajn, razvoj, testiranje isporuku te pregled proizvoda. Na slici 2.1 napravljenoj prema [5] vidljive su tri iteracije razvojnog ciklusa.

Sl 2.1. *Prikaz iteracija agilnog razvojnog ciklusa* [5]

Agilni pristup razvoju programskog rješenja najbolje je koristiti kada je potrebno često implementirati nove promjene i zahtjeve. Obzirom na česte nove inkremente razvoja, trošak dodavanja i razvoja novih zahtjeva sveden je na minimum. U slučaju kada je teže ostvariti komunikaciju između naručitelja ili ukoliko tim nije dovoljno iskusan, agilni pristup neće dati najbolje rezultate.

S vremenom, agilni pristup dobio je razne okvire kao što su *Scrum* i *Kanban* koji donose varijacije u implementaciji agilnog razvoja. *Scrum* je agilna metoda koja se koncentrira na upravljanje zadacima unutar timskog razvojnog okruženja. Sastoji se od tri uloge:

1. *Scrum master* – osoba koja je zadužena za tim, sastanke i uklanjanje prepreka kako bi se napredovalo
2. Vlasnik proizvoda – osoba koja kreira zadatke, prioritizira ih i osigurava uspješan završetak svake iteracije
3. *Scrum* tim – skupina ljudi koja upravlja i organizira vlastiti posao kako bi se uspješno završio *sprint* [6].

Svaka iteracija *Scruma* naziva se *sprint*. *Sprint* je događaj koji se ponavlja svakih 2-4 tjedna i uvijek je fiksnog trajanja. Novi *sprint* započinje čim prethodni završi, a sadrži sve događaje koji su potrebni kako bi se postigao cilj - planiranje *sprinta*, dnevne sastanke, pregled *sprinta* i retrospektivu. Ukoliko je *sprint* započeo, ne unose se nove promjene kako se ne bi ugrozio zadani cilj te kako kvaliteta ne bi bila narušena. Važan dio *Scruma* su i dnevni sastanci, odnosno tzv. dnevni *stand upovi*. *Stand up* je nastao po uzoru na američki nogomet u kojemu tim prije svake igre održava kratki sastanak kako bi se svi informirali, dogovorili i uskladili. Takvim sastancima prisustvuju vlasnik produkta, razvojni tim i *scrum master*. Svaki tim ima vlastiti tijek *stand upa*, ali najčešće se odgovara na tri pitanja - "što sam radio jučer?", "što ću raditi danas?" te "imam li kakvih blokada?" [7]. Odgovaranjem na postavljena pitanja cijeli tim dobiva uvid u stadije u kojima se nalazi pojedina osoba te cjelokupan tim.

Jedan od najpopularnijih okvira, uz *Scrum*, je i *Kanban*. *Kanban* se temelji na vizualnom sustavu vođenja projekata tako što vizualizira proces razvoja i samo izvođenje posla unutar procesa [8]. Cilj *Kanbana* je identifikacija potencijalnih problema unutar procesa i njihovo uklanjanje kako bi se posao izvršavao optimalnom brzinom uz minimalan trošak. *Kanban* se prvi puta pojavljuje u ranim 1940-ima unutar tvrtke Toyota, a David J. Anderson prvi puta *Kanban* primjenjuje na razvoj programskog rješenja 2004. godine. *Kanban* metoda slijedi četiri osnovna principa te šest temeljnih praksi [8]. Principi koje slijedi su:

1. Započni s onime što sada radiš
2. Slijedi postupne, evolucijske promjene
3. Poštuj trenutne uloge, odgovornosti i pozicije poslova
4. Poštuj i provodi sve odluke vodstva na svim razinama.

Temeljne prakse *Kanbana* su:

1. Vizualiziraj tijek posla
2. Ograniči trenutni posao
3. Upravljaj tijekom
4. Neka politike procesa budu eksplicitne
5. Primaj i pružaj povratne informacije
6. Poboljšavaj suradnju, razvijaj se eksperimentalno.

Na slici 2.2 nastaloj prema [8], vidljiv je uobičajen izgled ploče *Kanban*. Ploča je podložna izmjenama te konačan oblik ovisi o timu koji primjenjuje *Kanban* metodu. Svaka ploča bi trebala imati barem tri dijela - zadaci koji se trebaju napraviti, zadaci na kojima se trenutno radi te zadaci koji su završeni, ali u praksi, najčešće se dodaju dijelovi za testiranje, planiranje i slično.

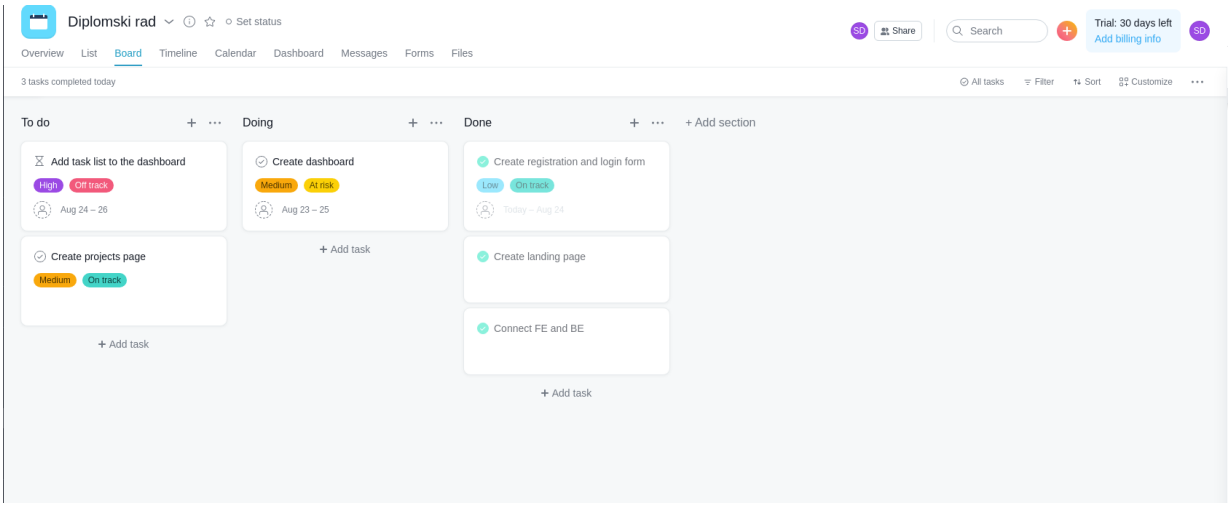
Sl. 2.2. *Prikaz ploče Kanban*

2.2. Postojeća rješenja za vođenje projekata agilnom metodom

2.2.1. Asana

Asana je programska podrška kao usluga (eng. *SaaS, Software as a Service*) kojoj se može pristupiti putem internetskog preglednika ili mobilne aplikacije. Napravljena je kako bi omogućila podršku te olakšala upravljanje i organizaciju projekata. Zamišljena je kao univerzalni alat za vođenje projekata bilo koje vrste, a ne nužno projekata vezanih za razvoj programske podrške [9]. S agilnog pogleda, Asana pruža mogućnost vođenja projekata pomoću okvira *Scrum* ili *Kanban* koji su detaljno objašnjeni u poglavlju 2.1.

Prilikom kreiranja projekta, korisnik odabire zadani način na koji su zadaci raspodijeljeni - kao lista, ploča, vremenska crta ili kalendar. Nakon kreiranja projekta, svaki korisnik uz zadani način može pregledavati zadatke i na ostale navedene načine. Na slici 2.3 vidljivi su zadaci koristeći način ploče, a pomoću navigacijske trake iznad ploče, može se mijenjati način prikaza.



Sl. 2.3. *Prikaz zadataka pomoću ploče*

Nakon postavljanja projekta, vlasnik projekta može dodavati nove članove te im dodjeljivati uloge. Vlasnik također može dodavati miljokaze i ciljeve projekata, mijenjati trenutni status, dodavati nove zadatke i slično. Uz glavne značajke vođenja projekata, Asana pruža mogućnost komunikacije među članovima projekta pomoću poruka i komentara, omogućuje automatiziranje ponavljajućih radnji poput mijenjanja statusa zadatka ili brisanja zadatka nakon što su zadovoljeni određeni uvjeti i slično. Također, omogućuje povezivanje s drugim aplikacijama i programskim podrškama kao što su Slack, GitHub, Google Drive te mnogi drugi [9].

2.2.2. Jira

Jira predstavlja programsko rješenje osmišljeno za korištenje prilikom vođenja projekata agilnom metodom. Dio je Atlassian grupacije uz Trello, Bitbucket, SourceTree i mnoge druge pa iz tog razloga predstavlja alat koji olakšava vođenje i automatiziranje agilnih projekata [10]. Poput Asane, Jira omogućava korištenje okvira *Scrum* ili *Kanban* te pruža mogućnost iscrtavanja putokaza, odnosno prikaz tijeka projekta pomoću vremenske crte.

Korištenjem okvira *Scrum*, Jira korisnicima omogućuje pisanje priča i problema, planiranje *sprintova*, dodjeljivanje zadataka te potiče iterativan i inkrementalan razvoj programske podrške [10]. Prilikom obavljanja dnevnih i *Scrum* sastanaka na kraju s*printa*, moguće je kreirati posebne

ploče kako bi se lakše pratio napredak projekta. Na kraju svakog *sprinta*, odnosno na početku novog, svi neizvršeni zadaci se automatizmom prebacuju u novi *sprint*. *Scrum* način veliku važnost pridonosi opterećenosti timova poslovima pa postoji mogućnost i praćenja količine posla kojeg članovi tima obavljaju na projektu pomoću tzv. *burndown* grafa.

Odabirom okvira *Kanban*, Jira korisnicima pomaže pri kontinuiranom poboljšanju vremena svakog ciklusa te povećanju učinkovitosti. Omogućuje korisnicima pregled stanja na jednom mjestu prikazujući najbitnije informacije za svaku priču, problem ili zadatak. Korištenjem ploče *Kanban*, moguće je postaviti ograničenje broja zadataka koji se trenutno izvršavaju te na taj način članovi tima mogu lako uočiti probleme i zastoje [10].

Jedna od važnijih značajki, koja uvelike pridonosi kvalitetnom vođenju projekata, su ugrađena izvješća i grafovi. Unutar okvira *Scrum*, projektni menadžeri mogu generirati izvješća za svaki *sprint*, a time dobivaju uvid o uspješnosti izvršavanja svih zadataka. Uz spomenuti *burndown* graf za praćenje količine posla članova tima, moguće je generirati i *burndown* graf za isporuku proizvoda pomoću kojeg se dobiva uvid u poštivanje rokova pa se mogu poduzeti potrebne akcije ako postoji mogućnost kašnjenja isporuke. Ukoliko tim koristi okvir *Kanban*, Jira pruža mogućnost generiranja kumulativnog dijagrama toka kako bi se lakše uočile blokade te mogućnost generiranja kontrolnog grafa pomoću kojeg se mogu odrediti buduće performanse tima.

2.2.3. Monday

Monday je programsko rješenje koje korisnicima omogućava vođenje projekata, bilo vezanih za razvoj programske podrške ili svih ostalih vrsta projekata. Predstavlja vizualan, jednostavan i intuitivan alat s više od 200 predložaka za vođenje projekata. Projekti mogu biti prikazani pomoću ploče ili tablice zadataka, omogućuje jednostavno upravljanje zadacima pomoću tzv. *drag and drop* metode s mogućnošću korištenja više od 30 stupaca. Tijek projekta moguće je prikazati pomoću ploče *Kanban*, kalendara, vremenske crte, gantograma i mnogih drugih prikaza kako bi članovi tima uspješno izvršavali zadatke unutar predviđenih rokova, a projektni menadžeri imali uvid u trenutno stanje projekta, koji član radi na kojem zadatku te lakše planiranje budućih *sprintova*.

Monday olakšava komunikaciju među članovima tima prilagodbom programskog rješenja za korištenje na mobilnim uređajima te uz značajke poput dijeljenja datoteka, slika i povratnih informacija u stvarnom vremenu. Kako bi se posao brže i jednostavnije izvršavao, Monday pruža mogućnost automatiziranja najčešćih radnji te obavještavanja članova tima o određenim

promjenama poput završetka zadatka, upozorenja o bitnim datumima i slično. Također, omogućava integraciju s raznim alatima i aplikacijama kao što su Slack, Google Drive, GitHub, GitLab, LinkedIn i mnogi drugi.

Prilikom kreiranja projekta, vlasnik projekta odabire jedan od načina prikaza, a Monday će postaviti projekt prema odabranom predlošku i automatski integrirati potrebne alate i aplikacije. Svaki dio projektne ploče je u potpunosti interaktivan i prilagodljiv kako bi svaki tim mogao raditi na način koji njemu najbolje odgovara. Budući da korisnici mogu dodavati vlastite datoteke i slike sve se sprema u oblak, a napredna tražilica omogućava pretraživanje podataka putem oznaka, imena ljudi, naziva i slično [11].

3. VRSTE OŠTEĆENJA VIDA I IZAZOVI ZA OSOBE S OŠTEĆENJEM VIDA PRI KORIŠTENJU POSTOJEĆIH RJEŠENJA ZA VOĐENJE PROJEKATA AGILNIM PRISTUPOM

Oštećenje vida smanjena je sposobnost ljudskog vida koja uzrokuje probleme koji se ne mogu riješiti uobičajenim sredstvima poput naočala. Oštećenje može biti uzrokovano gubitkom oštrine vida gdje oko ne vidi objekte čisto kao kod normalne oštrine. Također, može biti uzrokovano i smanjenjem vidnog polja. U tom slučaju, oko ne vidi uobičajeno široko područje [12].

Danas se, na svjetskoj razini, smatra da 253 milijuna ljudi ima neku vrstu oštećenja vida, od kojih je 36 milijuna definirano kao slijepe osobe [13]. Svjetska zdravstvena organizacija prihvatila je ICD 11 (eng. *International Classification of Diseases*) definiciju vizualnih oštećenja i sljepoće [13]. Prema navedenoj definiciji, osoba se smatra da ima oštećenje vida ako je oštrina vida zdravijeg oka lošija od 6/18, odnosno ako osoba s oštećenjem vidi na 6 metra ono što osoba bez oštećenja vidi na 18 metara. Ukoliko je rezultat 3/60, osoba se tada smatra slijepom. Tablica 3.1, nastala prema [14], prikazuje definicije oštećenja vida.

Tab. 3.1. *Definicija oštećenja vida*

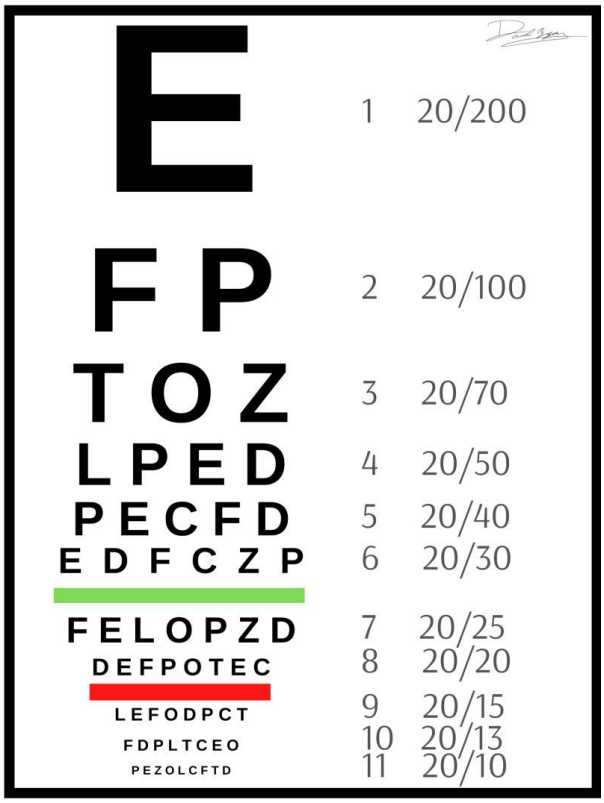
Kategorija	Lošije od:	Jednako ili bolje od:
Blago ili bez oštećenje vida		6/18
Umjereno oštećenje vida - kategorija 1	6/18	6/60
Teško oštećenje vida - kategorija: 2	6/60	3/60
Sljepoća - kategorija: 3	3/60	1/60 Moguće brojanje prstiju ruke na 1 metar udaljenosti
Sljepoća - kategorija: 4	1/60 Moguće brojanje prstiju ruke na 1 metar udaljenosti	Percepcija svjetla
Sljepoća - kategorija: 5	Bez percepcije svjetla, potpuna sljepoća	

Sljedeći rezultati pa i sam ICD 11 temeljeni su na testiranju pomoću grafa *Snellen*, na udaljenosti od 6 metara [15]:

- 6/6: osoba vidi na 6 metara isto što i prosječna osoba na 6 metara

- 6/18: osoba vidi na 6 metara isto što i prosječna osoba na 18 metara - umjereno oštećenje vida
- 6/60: osoba vidi na 6 metara isto što i prosječna osoba na 60 metara - teško oštećenje vida
- 3/60: osoba vidi na 3 metra isto što i prosječna osoba na 60 metara - sljepoća
- 1/60: osoba vidi na 1 metar isto što i prosječna osoba na 60 metara - sljepoća
- Ukoliko je vid toliko loš da se graf *Snellen* ne može koristiti, koristi se metoda brojanja prstiju ruke
- Ukoliko se ne može koristiti ni brojanje prstiju, tada se usmjerava jako svjetlo prema oku osobe i provjerava se ima li osoba percepciju svjetla. Ukoliko nema, tada se radi o potpunoj sljepoći

Slika 3.1 prikazuje primjer grafa *Snellen* preuzetog s [15].



Sl 3.1. *Graf Snellen*

3.1. Daltonizam

Daltonizam je oštećenje vida kod kojeg osobe vide boje drugačije od prosječnih osoba. Daltonizam je nasljedna bolest za koju ne postoji lijek, ali posebne naočale ili kontaktne leće mogu pomoći [16].

Postoje tri vrste daltonizma – monokromatski, dikromatski i trikromatski [17]. Monokromatski daltonizam najrjeđi je oblik sljepoće na boje. Osoba tada vidi samo nijanse crne, bijele i sive boje,

a često su prisutni i problemi s jasnoćom vida te osjetljivošću na svjetlost. Dikromatski daltonizam uključuje protanopiju, deuteranopiju i tritanopiju. Protanopija uzrokuje potpuni nedostatak crvenih receptora boje, deuteranopija uzrokuje nedostatak zelenih receptora, a tritanopija nedostatak plavih receptora [17]. Trikromatski daltonizam pojavljuje se u slučajevima kada je jednom od tri fotoreceptora promijenjena spektralna osjetljivost, a rezultira protanomalijom, deuteranomalijom ili tritanomalijom [17]. Protanomalija crvene boje prikazuje zelenijima i tamnijima, deuteranomalija zelene boje prikazuje crvenijima, a tritanomalija otežava razlikovanje plave i zelene te žute i crvene.

3.1.1. Daltonizam i pristupačnost

Kako bi internetska stranica bila pristupačna osobama s daltonizmom, treba osigurati čitljivost teksta temeljeno na kombinaciji boje teksta, boje pozadine i veličine slova [18]. U većini slučajeva nije potrebno u potpunosti ukloniti boje i koristiti samo crno-bijele kombinacije, ali je potrebno u slučajevima poput grafičkih prikaza (tortni grafikoni, stupčasti grafovi i slično) ili korištenja zelenog i crvenog teksta kako bi se naglasio prolaz ili pad te dobro ili loše. Kada se nešto naglašava bojama, ne treba se samo oslanjati na uočljive boje, nego treba i dodatno naglasiti pomoću ikona ili različitih uzoraka [18]. Na slici 3.2 vidljiv je primjer kako osoba bez daltonizma vidi poruku o grešci te kako je vidi osoba s potpunom sljepoćom na boje. Na primjeru je dodana ikona koja upozorava da se radi o grešci pa osoba s daltonizmom bez problema uočava da se radi o poruci koja upozorava na grešku.

Sl. 3.2. *Prikaz poruke o grešci i kako ga vidi osoba s potpunim daltonizmom*

Ukoliko se koristi tekst preko slike, potrebno je smanjiti prozirnost pozadine kako bi se tekst što više istaknuo iznad slike [18]. U slučaju odabira ili filtriranja boja, dobra praksa je uz boju navesti i naziv boje. Prilikom korištenja poveznica, ne smije se u potpunosti oslanjati na isticanje pomoću boje te svaku poveznicu treba dodatno naglasiti, na primjer, podcrtavanjem [18]. Prilikom

korištenja formi, potrebno je nazive polja staviti iznad samog polja za unos kako bi se što bolje istaknuli, a u slučaju obaveznih polja, ne treba se oslanjati samo na boju već je potrebno stavit oznaku poput zvjezdice ili uskličnika kako bi korisnik znao da je unos u polje obavezno [18].

3.2. Disleksija

Disleksija spada među najčešće oblike invalidnosti, podjednako zahvaća i muškarce i žene, odnosno od 10% do 15% svjetske populacije [19]. Predstavlja razvojni poremećaj koji uključuje poteškoće s čitanjem unatoč normalnoj inteligenciji, perifernom vidu, dobrom školovanju te odsutnosti psihijatrijskih poremećaja [20]. Utječe na dijelove mozga koji procesuiraju jezik, a osobe kojima je dijagnosticirana disleksija imaju normalnu inteligenciju te obično nemaju problema s vidom. Većina djece s disleksijom uspješno odrađuje školske obaveze uz specijalizirane edukacije [19].

Disleksija se najčešće detektira polaskom djeteta u školu, ali nekada se ne uoči sve do odrasle dobi. Prema [19] postoji nekoliko znakova koji govore da bi dijete moglo imati disleksiju, a neki od njih su:

- Dijete je počelo normalno razgovarati tek u kasnijoj dobi
- Sporo učenje novih riječi
- Problemi s točnim formiranjem riječi
- Problemi s pamćenjem slova, brojeva i boja
- Problemi s čitanjem
- Problemi s procesiranjem i shvaćanjem onoga što osoba čuje
- Poteškoće sa slovkanjem.

3.2.1. Disleksija i pristupačnost

Postoji mnogo čimbenika koje treba uzeti u obzir kako bi web aplikacija bila pristupačna osobama s disleksijom, a prema [21] neki od njih su:

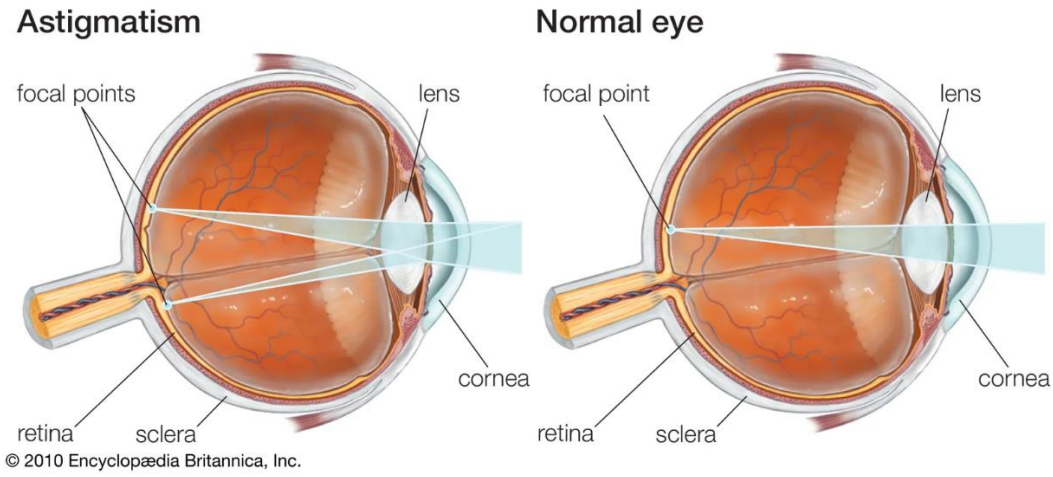
- Najbolji te najčešći fontovi koje osobe s disleksijom lakše čitaju pripadaju *Sans Serif* skupini, odnosno *Arial*, *Verdana* i *Tahoma* fontovi
- Veličina fonta bi se trebala kretati od, minimalno, 12 ili 14 pt.
- Prored bi trebao biti barem 1.5 ili 2 kako bi se lakše procesirali veći dijelovi teksta
- Treba izbjegavati nepotrebno podcrtavanje teksta, kurziv te pisanje teksta velikim slovima
- Gdje god je prigodno, treba razdvojiti velike dijelove teksta slikama ili grafovima
- Treba koristiti lijevo poravnanje umjesto obostranog
- Korisnicima treba omogućiti prilagodbu fonta, boje i veličine.

Uz navedene čimbenike, važno je paziti i na kontrast. Ukoliko je kontrast prenizak, moguće je teže raspoznavanje teksta, odnosno slova [22]. Kao što je navedeno, dobra praksa je ubacivanje slika i grafova kako bi se razdvojili veliki dijelovi teksta. Tada je slikama potrebno dodati tzv. *alt* atribut kako bi osobe koje koriste čitače zaslona imale uvid u sadržaj slike ili grafa. Korisnicima s disleksijom treba omogućiti i promjenu pozadine teksta kako bi se lakše fokusirali na sami tekst jer ponekad potpuno bijela pozadina može dezorijentirati i učiniti tekst težim za čitanje [22].

3.3. Astigmatizam

Astigmatizam je česta nesavršenost zakrivljenosti ljudskog oka koja uzrokuje zamagljen vid, bilo da se radi o vidu na daljinu ili blizinu. Nastaje kada prednji dio površine oka ili leća unutar oka imaju nepravilnu zakrivljenost [23]. Prosječno oko ima oblik lopte, dok u slučaju astigmatizma, oko najčešće ima oblik jajeta te iz tog razloga dolazi do zamućenog vida. Najčešći simptomi astigmatizma su, kao što je navedeno, zamućeni ili iskrivljeni vid, naprezanje oka ili nelagodnost, česte glavobolje, problemi s noćnim vidom te škiljenje [23].

Normalno oko ima jednako zakrivljenu rožnicu i leću u svim smjerovima te se zbog toga svjetlost pravilno fokusira i prelama [24] što je vidljivo na slici 3.3 preuzetoj s [25]. Ukoliko rožnica ili leća nisu glatke i pravilno zakrivljene dolazi do tzv. greške prijeloma.



Sl 3.3. *Prikaz pada svjetlosti na oko s astigmatizmom i na normalno oko* [25]

3.3.1. Astigmatizam i pristupačnost

U današnje vrijeme, najčešća opcija velikog kontrasta su bijela slova na tamnoj podlozi, ali u praksi ona ne predstavlja najbolje rješenje. Ukoliko se koriste potpuno bijela slova na potpuno crnoj pozadini, osobama s astigmatizmom se stvara efekt tzv. svjetlosne mrlje [26]. Navedeni efekt smanjuje čitljivost teksta, a osobama s astigmatizmom vrlo često može uzrokovati jake glavobolje.

Kako bi korisnici bez problema čitali tekst, potrebno je koristiti svjetlije nijanse crne boje za pozadinu ili u potpunosti izbjegavati takvu kombinaciju pozadine i teksta [26].

3.4. Postojeća rješenja za vođenje projekata agilnim pristupom i pregled prilagodbi za osobe s oštećenjima vida

Prema značajkama programskog rješenja za vođenje projekata Asana, svaki korisnik u postavkama korisničkog računa može uključiti način rada za osobe s deuteranopijom i protanopijom, odnosno sljepoćom za crveno-zelene boje. Uključivanjem navedenog načina rada, korisnici jednostavnije mogu raspoznavati projekte i statuse pojedinih zadataka. Uz način rada za osobe sa sljepoćom na boje, Asana pruža podršku za iOS *VoiceOver* čitač zaslona. *VoiceOver* je čitač zaslona temeljen na gestikulaciji, a Asana je time omogućila korištenje iOS aplikacije bez potrebe da korisnik vidi zaslon, odnosno omogućila je korištenje slijepim i slabovidnim osobama [27]. Detaljnijom inspekcijom pomoću Googleovih alata za programere, uočljivo je da Asana koristi prored 1.5, veličinu fonta od 16px te font iz *Sans Serif* obitelji što osobama s disleksijom olakšava čitanje teksta. Ukoliko govorimo o kontrastu, koji je vrlo važna stavka za osobe s daltonizmom, disleksijom i astigmatizmom, na nekim dijelovima programskog rješenja, kontrast je vrlo nizak, ali zadovoljava najnižu AA razinu prema WCAG (eng. *Web Content Accessibility Guidelines*), no ne i najvišu AAA.

Atlassian grupacija, kojoj pripada i programsko rješenje Jira, navodi da se u svakom trenutku brinu kako bi njihovi proizvodi zadovoljavali minimalne zahtjeve prema WCAG 2.1 inačici. Također, pružaju podršku za čitače zaslona te preporučuju korištenje NVDA čitača zaslona uz internetski pretraživač Firefox ukoliko se koristi operacijski sustav Windows. U slučaju korištenja operacijskog sustava OSX, preporučuju čitač zaslona *VoiceOver* uz internetski pretraživač Safari [28]. U slučaju programskog rješenja Jira, detaljnijom inspekcijom elemenata uočljivo je korištenje 1.5 proreda, veličine fonta od 14px te fonta iz *Sans Serif* obitelji, što odgovara čimbenicima koje treba uzeti u obzir ukoliko se programsko rješenje prilagođava osobama s disleksijom. Daljnjim korištenjem uočljivo je da se uz sami tekst i različite boje, koriste i različite ikone, ovisno o sadržaju, a takav način olakšava osobama s daltonizmom korištenje programskog rješenja Jira.

Programsko rješenje Monday svojim korisnicima omogućilo je korištenje tipkovničkih prečaca za najčešće radnje poput spremanja, zatvaranja, pretraživanja i slično. Monday također pruža podršku za čitače zaslona pri korištenju računalne, mobilne i internetske inačice čime omogućuje nesmetano korištenje slijepim i slabovidnim osobama. Za razliku od Asane i Jire, Monday nudi

mogućnost promjene veličine teksta i kontrasta čime osigurava jednostavnije korištenje osobama s daltonizmom, disleksijom i astigmatizmom [29]. Inspekcijom elemenata pomoću Googleovih alata za programere, vidljivo je korištenje proreda od 24px, veličine fonta od minimalno 14px te fonta iz *Sans Serif* obitelji, što, kao i prethodna programska rješenja, zadovoljava uvjete prilagodbe osobama s disleksijom.

3.5. Mogućnosti poboljšanja prilagodbe postupka vođenja projekata agilnim pristupom za osobe s oštećenjima vida

Kratkom analizom i nakon korištenja programskih rješenja Asana, Jira i Monday vidljivo je da svi zadovoljavaju minimalne uvjete prilagodbe i pristupačnosti slijepim i slabovidnim osobama. U tablici 3.2 navedene su neke od mogućnosti poboljšanja kako bi korisnicima s oštećenjima vida pružili jednostavnije korištenje programskih rješenja

Tab. 3.2. *Mogućnosti poboljšanja programskih rješenja*

Asana	Jira	Monday
Mogućnost prilagodbe veličine teksta	Mogućnost prilagodbe veličine teksta	Uz naglašavanje bojama, dodati različite ikone ovisno o sadržaju
Mogućnost promjene kontrasta	Mogućnost promjene kontrasta	
Uz naglašavanje bojama, dodati različite ikone ovisno o sadržaju	Mogućnost prilagodbe boja za osobe s daltonizmom	

4. MODELIRANJE PROGRAMSKOG RJEŠENJA KORISNIČKE STRANE

4.1. Zahtjevi korisničke strane programskog rješenja

Zahtjevi koji su dobro promišljeni i dobro definirani temelj su svakog, uspješno odrađenog, projekta. Postoje dva glavna tipa zahtjeva na sustav - funkcionalni i nefunkcionalni. Razumijevanje razlike između oba tipa zahtjeva doprinosi uspješnoj isporuci sustava koji radi točno ono što se od njega zahtjeva.

Kako bi zahtjevi korisničkog dijela bili uspješno ispunjeni, potrebno je definirati i ispuniti zahtjeve na poslužiteljskoj strani programskog rješenja. Poslužiteljska strana mora pružati podatke korisničkoj strani te treba moći u što kraćem vremenskom intervalu primiti korisničke zahtjeve, obraditi ih te vratiti rezultat [1].

4.1.1. Funkcionalni zahtjevi korisničke strane programskog rješenja za upravljanje projektima agilnim pristupom

Funkcionalni zahtjevi definiraju sustav ili njegove komponente te opisuju funkcije koje mora obavljati. Funkcije predstavljaju ulazne podatke, njihovo ponašanje te izlazne podatke. To mogu biti razni izračuni, manipulacije podacima, poslovni procesi, korisničke interakcije sa sučeljem i slično.

Tijekom faze modeliranja programskog rješenja korisničke strane, kao prvi korak utvrđeni su mogući korisnici te funkcionalni zahtjevi korisničkog sučelja. Zahtjevi koje je potrebno zadovoljiti su:

- registracija korisnika i organizacije
- prijava korisnika u sustav
- kreiranje zaposlenika
- promjena korisničkih informacija
- promjena informacija o zaposlenicima
- promjena informacija o organizaciji
- kreiranje projekta
- dodavanje zaposlenika na projekt
- pregled informacija o projektu
- pregled zadataka unutar projekta
- pregled gantograma unutar projekta
- zatvaranje i brisanje projekta

- kreiranje zadataka
- promjena detalja zadataka
- brisanje zadataka
- dodjela zadataka zaposlenicima
- pretraživanje zadataka
- filtriranje zadataka
- pregled zaposlenika
- pretraživanje zaposlenika
- filtriranje zaposlenika
- prilagodba pristupačnosti.

Registracija korisnika i organizacije prvi je korak prilikom korištenja sustava. Korisnik, koji je ujedno i administrator, registrira vlastiti račun i račun organizacije. Nakon prijave u sustav, administrator kreira nove zaposlenike koji se prijavljuju u sustav s vlastitim računima. Zaposlenici mogu biti administratori, projektni menadžeri ili obični zaposlenici poput programera, testera, dizajnera i slično, a svi imaju istu ulogu - zaposlenik (eng. *employee*). Administratori i projektni menadžeri također imaju svoje uloge – administrator (eng. *admin*) i projektni menadžer (eng. *project manager*).

Ukoliko je korisnik administrator ili projektni menadžer, može kreirati nove projekte unosom podataka u formu za kreiranje projekta. Tijekom kreiranja projekta dodaju se i zaposlenici koji će raditi na tom projektu. Nakon što je projekt kreiran, svi korisnici mogu pregledavati informacije o projektu poput zadataka, članova tima na projektu i gantograma projekta. Također, korisnici mogu kreirati zadatke, mijenjati detalje zadataka poput rokova, statusa te zaposlenika kojem je dodijeljen zadatak.

Svi korisnici mogu pregledavati, filtrirati i pretraživati ostale zaposlenike unutar organizacije. Administrator ima mogućnost i promjene uloge zaposleniku (administrator, projektni menadžer ili zaposlenik). Svaki zaposlenik može mijenjati detalje o sebi poput imena, prezimena, adrese e-pošte i slično.

Kako bi sustav u potpunosti bio prilagođen slabovidnim osobama i osobama s oštećenjima vida, svaki korisnik sustava može na početnom zaslonu sustava, prije prijave ili nakon prijave unutar vlastitih postavki, mijenjati postavke pristupačnosti. Korisnici imaju mogućnost prilagodbe sustava promjenom fonta, koji olakšava čitanje osobama s disleksijom. Također, mogu se promijeniti palete boja za osobe koje imaju daltonizam, odabirom vrste daltonizma - crveno-zeleno

ili plavo-žuto te se unutar cijelog sustava može primijeniti visoki kontrast ili promijeniti veličina slova.

4.1.2. Nefunkcionalni zahtjevi korisničkog dijela programskog sustava za upravljanje projektima agilnim pristupom

Nefunkcionalni zahtjevi na sustav definiraju atribute kvalitete programskog rješenja. Predstavljaju skup standarda koji se koriste za prosuđivanje specifičnog rada sustava. Na primjer, koliko brzo se sustav učitava ili koliko je sustav siguran od krađe podataka. Neispunjavanjem nefunkcionalnih zahtjeva može doći do sustava koji ne zadovoljava korisničke potrebe.

Drugi korak tijekom faze modeliranja je utvrđivanje nefunkcionalnih zahtjeva programskog sustava. Zahtjevi koje je potrebno zadovoljiti su:

- sigurnost podataka
- prilagodljivost
- proširivost
- brzina
- pouzdanost

Sigurnost podataka prvi je zahtjev koji treba ispuniti. Osjetljivi podaci poput podataka o zaposlenicima te njihovi pristupni podaci za sustav (adresa e-pošte i lozinka) trebaju biti zaštićeni i kriptirani. Lozinka se prilikom spremanja u bazu podataka automatski kriptira uz dodavanje *hasha* i *salta* [1]. Kriptiranjem pristupnih podataka, ukoliko dođe do napada i pristupa bazi podataka od treće strane, onemogućava se čitanje podataka i krađa podataka. Korisnicima se prilikom prijave u sustav dodjeljuje jedinstveni token koji omogućava pristup svim ili određenim funkcionalnim dijelovima sustava [1]. Korisnik bez valjanog tokena nema pristup podacima o organizaciji, projektima, zadacima i slično. Kako bi se onemogućio pristup podacima, prilikom svakog slanja zahtjeva na poslužitelj, obavezno je slanje autentifikacijskog tokena. Ukoliko token nije valjan, dobiva se poruka o pogrešci, a podaci se ne prikazuju. Kao dodatan oblik sigurnosti, uzimajući u obzir da je poslužiteljska strana zasnovana na arhitekturi mikrousluga, svaka usluga mora posjedovati vlastitu instancu *MongoDB* baze podataka te jedinstvenu domenu radu [1].

Prilagodljivost, u smislu nefunkcionalnih zahtjeva, podrazumijeva prilagodljivost sustava na različite preglednike i operacijske sustave. Prilagodljiv sustav je sustav koji se može pokretati na bilo kojem pregledniku ili bilo kojem operacijskom sustavu. Korisnici moraju imati pristup programskom rješenju u svakom trenutku bez obzira gdje i kada mu pristupaju.

Proširivost sustava postiže se pisanjem komponenti koje se mogu koristiti u što više slučajeva s minimalno ili bez izmjena. Na taj način sustav se može jednostavno nadograđivati i mijenjati bez opasnosti da će sustav prestati raditi ili zadovoljavati zahtjeve.

Programski sustav korisnicima mora omogućavati brz pristup podacima. Korisnici u svakom trenutku trebaju što brže dobiti potrebne podatke i informacije, bez obzira na količinu podataka ili broj korisnika koji trenutno koriste sustav. Brzina se postiže tako da se šalju samo nužni zahtjevi na poslužitelj te učitavaju i prikazuju samo nužni podaci. Pretrpavanjem sustava s podacima koji nisu potrebni u tom trenutku ili slanjem zahtjeva na poslužitelj koji nisu potrebni, može doći do opterećenja i korisničke i poslužiteljske strane programskog rješenja što dovodi do duže obrade i prikaza podataka.

Pouzdanost, kao nefunkcionalni zahtjev, znači da sustav u svakom trenutku mora biti dostupan korisnicima, odnosno da ne smije ispasti iz rada. Ukoliko sustav ispadne iz rada, korisnici nemaju pristup podacima koji u tom trenutku mogu biti prijeko potrebni. Pouzdanost je posebno važna ukoliko se sustav nalazi na udaljenom poslužitelju, jer u tom trenutku mogućnost za ispad iz rada postaje veća negoli na sustavu koji radi na lokalnom poslužitelju.

4.2. Model programskog rješenja korisničkog sučelja

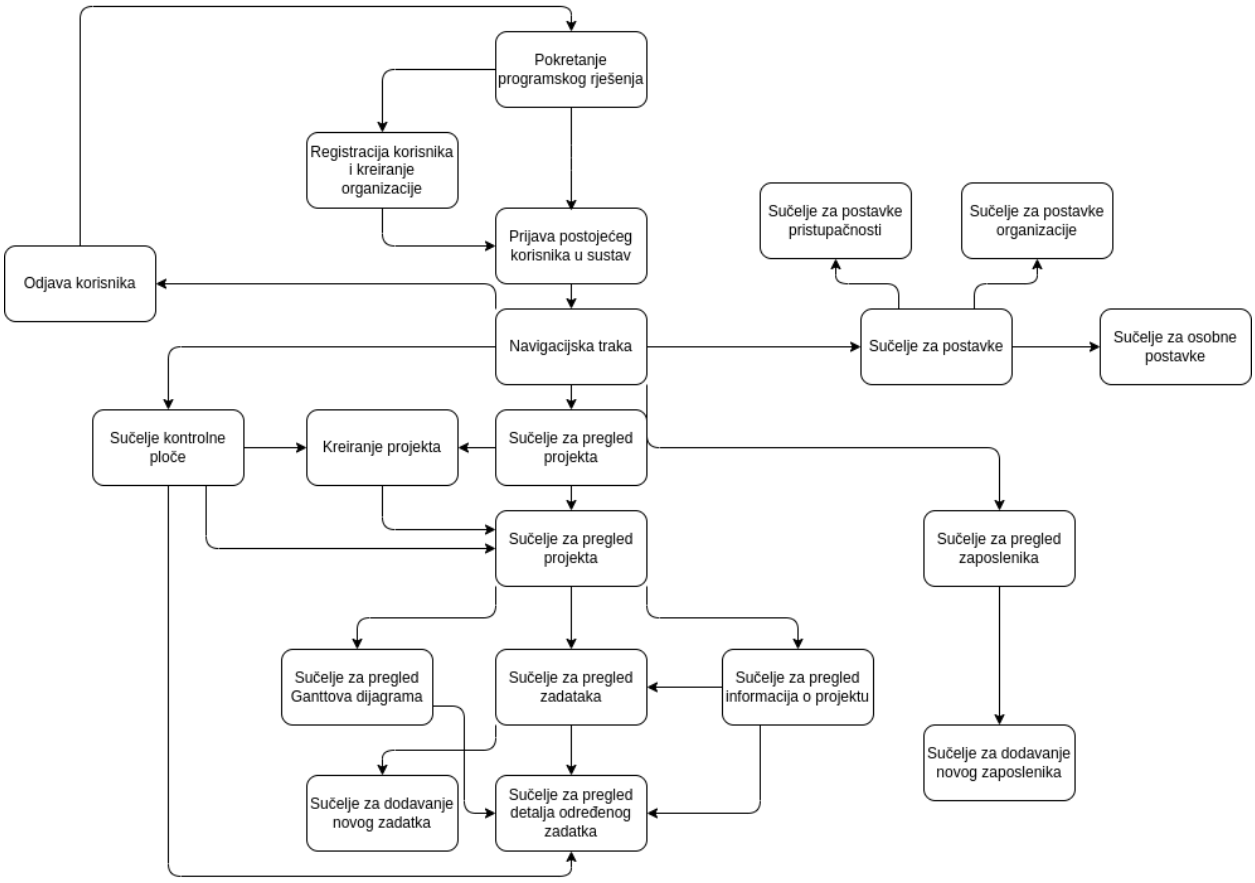
Nakon utvrđenih funkcionalnih i nefunkcionalnih zahtjeva, sljedeći korak je modeliranje korisničke strane programskog rješenja za upravljanje projektima. Na slici 4.1 vidljiv je model korisničke strane.

4.3. Arhitektura programskog rješenja

Arhitektura programskog rješenja sastoji se od samog izgleda programskog rješenja, komunikacije s poslužiteljskom stranom te prikaza podataka na korisničkom sučelju. Arhitektura programskog rješenja podijeljena je na:

- *assets* - direktorij s dodatnim sadržajima poput slika, ilustracija i fontova
- *components* - direktorij u kojem se nalaze sve komponente korisničkog sučelja
    - *pages* - direktorij u kojem se nalaze komponente kontrolne ploče, projekata, postavki, zaposlenika i slično
    - *menus* - direktorij u kojem se nalaze sve komponente različitih izbornika na korisničkom sučelju
    - *navbars* - direktorij u kojem se nalaze sve komponente različitih navigacijskih traki
    - *shared* - direktorij u kojem se nalaze sve višekratne komponente poput skočnih prozora, tablica, zaglavlja i slično

- *functions* - direktorij u kojem se nalaze zajedničke *Javascript* funkcije
- *redux* - direktorij u kojem se nalaze postavke za *Redux* skladište
    - *actions* - direktorij u kojem se nalaze sve funkcije koje predstavljaju akcije
    - *actionTypes* - direktorij u kojem se nalaze sve konstante vrsta akcija
    - *middlewares* - direktorij u kojem se nalaze sve funkcije međusloja
    - *reducers* - direktorij u kojima se nalaze svi reduktori
- *styles* - direktorij koji sadrži sve definirane boje i stilove elemenata korisničkog sučelja.



Sl. 4.1. *Model korisničke strane programskog rješenja*

Glavne značajke korisničke strane programskog rješenja za vođenje projekata agilnim pristupom su *Redux* skladište (eng. *store*) s popratnim funkcijama te *React* komponente. *Redux* arhitektura sastoji se od akcija, međusloja (eng. *middleware*), konstanti i reduktora (eng. *reducers*) koji su detaljnije opisani u poglavlju 5.3.

Druga glavna značajka korisničke strane su *React* komponente. Komponente se dijele na dva dijela - kontejnere i prezentere. Unutar kontejner dijela komponente piše se logika dohvaćanja i prikazivanja podataka na korisničko sučelje, a prilagođene podatke prosljeđuje prezenteru.

Prezenter tada primljene podatke prikazuje na korisničkom sučelju pomoću *HTML* elemenata ili drugih, manjih, *React* komponenti. Ukoliko više komponenti sadrži istu logiku dohvaćanja i prikazivanja podataka, tada se pišu višekratne komponente koje se mogu koristiti unutar ostalih komponenti te se time smanjuje pisanje istog kôda. *React* komponente i načini rada detaljnije su opisani u poglavlju 5.2.

5. PROGRAMSKE TEHNOLOGIJE KORIŠTENE ZA RAZVOJ APLIKACIJA KORISNIČKE STRANE S NAGLASKOM NA BIBLIOTEKU REACT

5.1. Javascript

Javascript je skriptni ili programski jezik koji omogućuje implementaciju kompleksnih značajki na web stranicama. Uz *HTML* (eng. *HyperText Markup Language*) i *CSS* (eng. *Cascading Style Sheets*), *Javascript* predstavlja treći dio standardnih web tehnologija, a omogućuje kreiranje dinamičkih web stranica, kontroliranje multimedije, animiranje slika i fotografija i slično.

Osnovni *Javascript* jezik na strani korisnika sastoji se od uobičajenih značajki poput pohrane korisnih vrijednosti unutar varijabli, operacija nad tekstom te pokretanja određenih dijelova kôda, ovisno o događajima na stranici, poput pritiska na gumb. Ne gledajući nužno samo mogućnosti za programiranje na korisničkoj strani, pomoću *API*-ja (eng. *Application Programming Interfaces*), *Javascript* jeziku dodaju se razne mogućnosti. Općenito, postoje dvije vrste *API*-ja, takozvani *Browser API*-ji i *API*-ji trećih strana. *Browser API*-ji su ugrađeni unutar internetskih pretraživača te omogućuju upravljanje podacima i izvođenje kompleksnih radnji vezanih za lokalno okruženje. Neki od njih su *DOM API* koji omogućuje manipulaciju nad *HTML* i *CSS* kôdom, zatim *Geolocation API* koji prikuplja geografske informacije, *Canvas* koji omogućuje kreiranje 2D i 3D grafičkih objekata te mnogi drugi. *API*-ji trećih strana nisu ugrađeni u internetske pretraživače, a njihov kôd dohvaća se s Interneta. Primjer takvih *API*-ja je *Twitter API* koji omogućuje prikaz najnovijih objava na web stranici, *Google Maps API* koji omogućuje implementaciju prilagođenih karti te mnogi drugi.

5.2. React

React je *Javascript* biblioteka nastala kako bi se omogućio razvoj brzih i interaktivnih korisničkih sučelja za web i mobilne aplikacije [30]. To je *open-source* biblioteka temeljena na komponentama, a zaslužna je za aplikacijski tzv. „pogled" sloj. *React* razdvaja korisničko sučelje na više komponenti što čini pronalazak i uklanjanje grešaka lakšim.

Jedna od prednosti korištenja biblioteke *React* je ta što *React* koristi virtualni *DOM* (eng. *Document Object Model*) koji uspoređuje prethodno stanje komponente te ažurira samo promijenjene dijelove na stvarnom *DOM*-u [30]. S obzirom da je *React* temeljen na komponentama, jedna ili više aplikacija može koristiti iste komponente što uvelike smanjuje vrijeme razvoja aplikacija.

5.2.1. Komponente, stanja i svojstva

Komponenta je osnovna jedinica svake *React* aplikacije, a svaka aplikacija sastoji se od više komponenti. Funkcijske komponente općenito ne sadržavaju vlastita stanja nego samo način prikazivanja, a podatke mogu primati iz drugih komponenti pomoću primljenih svojstava [30]. Za razliku od funkcijskih, klasne komponente mogu sadržavati i upravljati stanjima te koriste zasebnu metodu za prikazivanje koja vraća JSX (*Javascript XML*) kôd [31].

Stanje (eng. *state*) je ugrađeni *React* objekt koji se koristi za spremanje podataka i informacija o komponenti [30]. Stanje se može mijenjati, a pri svakoj promjeni, komponenta se ponovno prikazuje.

Poput stanja, svojstva (eng. *properties, props*) su također ugrađeni objekti unutar biblioteke *React*. Svojstva pružaju mogućnost prosljeđivanja podataka iz jedne komponente u drugu na isti način kao što se argumenti prosljeđuju funkcijama, a funkcioniraju slično kao *HTML* atributi [30].

5.2.2. React kuke

React kuke (eng. *React Hooks*) omogućavaju korištenje stanja i ostalih *React* značajki vezanih za životni ciklus komponente, unutar funkcionalnih komponenti [30]. Korištenjem *React* kuka značajno se doprinosi smanjenju vremena razvoja. *React* pruža nekoliko ugrađenih kuki, ali moguće je i dodavanje vlastitih, prilagođenih, kuki. Unutar biblioteke *React* postoje tri osnovne kuke – *useState, useEffect* i *useContext* te dodatne kuke poput *useCallback, useRef* i drugih [30]. Kreiranjem vlastitih kuka možemo izdvojiti logiku iz komponente u funkcije koje se mogu koristiti više puta te time smanjiti bespotrebno ponavljanje kôda. Pozivom kuke stanja se izoliraju i koriste samo za onu komponentu koja je pozvala kuku.

5.3. Redux

Dan Abramov i Andrew Clark 2015. godine stvorili su *Redux, Javascript* biblioteku za upravljanje stanjima. Jedan od ključnih načina funkcioniranja biblioteke *Redux* je korištenje *Redux* skladišta koji upravlja cijelom aplikacijom pomoću jednog objekta stanja [32]. Najčešće se koristi uz biblioteku *React*, ali funkcionira i s ostalim *Javascript* programskim okvirima i bibliotekama.

5.3.1. Osnovne komponente biblioteke Redux

Redux biblioteka sastoji se od nekoliko osnovnih dijelova – akcija, kreatora akcija, reduktora, skladišta i *dispatch* funkcije.

Akcija (eng. a*ction*) je običan *Javascript* objekt koji uvijek sadrži element naziva *type*. Akcija se može opisati i kao događaj koji opisuje što se dogodilo unutar aplikacije [33]. Uz element *type*, akcije mogu sadržavati dodatne informacije o događajima, koje se nalaze unutar elementa *payload*.

Kreator akcije (eng. *action creator*) je funkcija koja kreira i vraća objekt akcije. Obično se koristi kako se ne bi svaki put morao zasebno pisati objekt akcije [34].

Reduktor (eng. *reducer*) je funkcija koja prima trenutno stanje i objekt akcije, zatim odlučuje kako se ažurira stanje, ukoliko je potrebno te vraća novo stanje [34]. Može se opisati i kao osluškivač događaja koji upravlja događajima ovisno o primljenoj akciji, odnosno događaju. Prema [34] reduktori moraju pratiti određena pravila:

- Određuju novo stanje ovisno o stanju i akciji koju su primili.
- Ne smiju mijenjati postojeće stanje. Prvo moraju kopirati postojeće stanje, napraviti promjene i tada ažurirati originalno stanje.
- Ne smiju koristiti asinkronu logiku, računati nasumične vrijednosti ili izazivati bilo kakve nuspojave.

Trenutno stanje aplikacije pohranjeno je u skladište (eng. *store*). Skladište se kreira prosljeđivanjem reduktora te sadrži metodu *getState* pomoću koje se dohvaćaju vrijednosti trenutnog stanja [34]. Skladište također sadrži i metodu *dispatch* koja pruža jedini način ažuriranja i promjene stanja. Metodi *dispatch* prosljeđuje se objekt akcije, a skladište nakon toga poziva izvodi funkciju reduktora kako bi se trenutno stanje ažuriralo ovisno o događaju koji je nastupio [34].

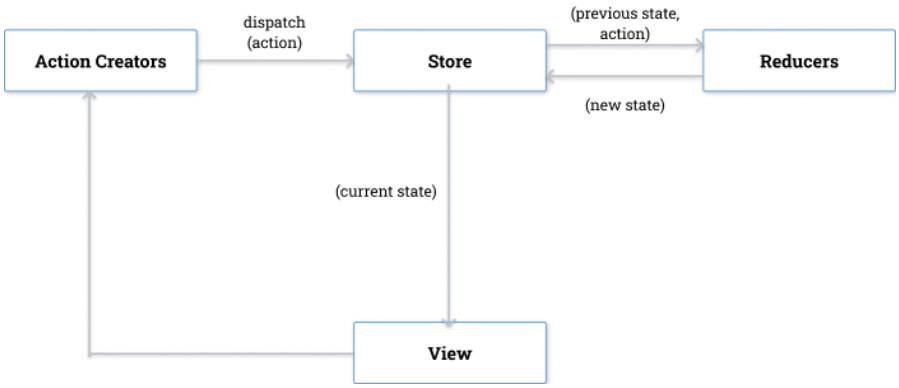
5.3.2. Protok podataka unutar Redux aplikacija

Ažuriranje i prikaz stanja pomoću *Redux* biblioteke sastoji se od nekoliko koraka, a pojednostavljeni prikaz protoka podataka vidljiv je na slici 5.1 nastaloj prema [33]. Koraci od kojih se sastoji *Redux* aplikacija mogu se podijeliti na dva dijela – inicijalne postavke te ažuriranja. Prema [34] inicijalne postavke su:

- Kreiranje *Redux* skladišta korištenjem korijenske reduktor funkcije.
- Skladište poziva korijensku reduktor funkciju jednom te sprema njenu povratnu vrijednost kao početno stanje.
- Nakon što je korisničko sučelje prvi put učitano, komponente pristupaju trenutnom stanju skladišta i koriste podatke unutar skladišta kako bi odlučile što prikazati. Također, pretplaćuju se na sva buduća ažuriranja skladišta kako bi znale kada se stanje promijenilo.

Ažuriranja se sastoje od sljedećih koraka [34]:

1. Dogodi se događaj unutar aplikacije, na primjer, korisnik pritisne gumb.
2. Poziva se *dispatch* funkcija kojoj se predaje objekt akcije
3. Skladište poziva funkciju reduktora te joj predaje trenutno stanje i trenutnu akciju kao argumente. Nakon što funkcija reduktora vrati vrijednosti, skladište te vrijednosti sprema kao novo stanje.
4. Skladište obavještava sve dijelove korisničkog sučelja koji su pretplaćeni da se dogodila promjena stanja
5. Svaka komponenta koja koristi podatke iz skladišta provjerava je li došlo do promjene podataka koje koriste
6. Svaka komponenta koja vidi da su se dogodile promjene nad podacima se ponovno prikazuje s novim podacima.



Sl. 5.1. *Prikaz protoka podataka u Redux aplikaciji* [33]

5.4. Biblioteke Jest i Enzyme i načini prikaza komponenti

Testiranje jedinica kôda (eng. *unit testing*) važan je, sastavni dio, razvoja programskog rješenja koji pomaže osigurati stabilnost proizvoda. Testiranjem jedinica kôda provjerava se ispravnost i točnost svake komponente sustava. Na taj način poboljšava se kvaliteta samog kôda te se omogućava rano otkrivanje pogrešaka prilikom razvoja ili izmjene zahtjeva [35].

Jest je *Javascript* testni okvir temeljen na jednostavnošću. Najbolje funkcionira s *React* aplikacijama, ali se može koristiti i uz druge biblioteke i okvire poput razvojnog okvira *Angular* . Uz *Jest*, za testiranje *React* aplikacija, koristi se programska biblioteka *Enzyme* koja je dizajnirana za testiranje *React* komponenti. *Enzyme* omogućava pisanje tvrdnji za simuliranje akcija kako bi se potvrdila točnost izvođenja korisničkog sučelja [36].

Uz testiranje komponenti, njihovih ulaza, izlaza i funkcija, važna stavka je i testiranje snimki zaslona. Prilikom prvog pokretanja testova, ukoliko je kreiran slučaj testiranja snimke, kreirat će se *.snap* datoteka. Svakom izmjenom i pokretanjem testova kreira se nova snimka i uspoređuje s prethodnom. Ukoliko se snimke ne podudaraju, test neće proći i tada se treba provjeriti je li došlo

do neočekivanog ponašanja komponente ili je potrebno ažurirati zastarjelu snimku [35]. Na taj način osigurava se da korisničko sučelje u svakom trenutku prikazuje točne i relevantne komponente i podatke.

Kako bi se određena komponenta mogla koristiti i testirati unutar testnih slučajeva, potrebno je prikazati komponentu pomoću jedne od funkcija za prikaz. Svaka funkcija koristi se u određenim trenutcima ovisno o željenim funkcionalnostima. Funkcije koje se koriste su *mount, shallow* i *render*.

Mount funkcija koristi se u slučajevima kada je potrebno prikazati i testirati cijeli DOM, odnosno komponentu i sve njene komponente djecu. Idealna je u situacijama kada komponenta mora komunicirati s DOM *API*-jem ili ukoliko se koriste *React* metode životnog ciklusa. *Mount* također omogućuje pristup svojstvima koji su direktno proslijeđeni u korijensku komponentu kao i svojstvima koji su proslijeđeni u komponente djecu [37].

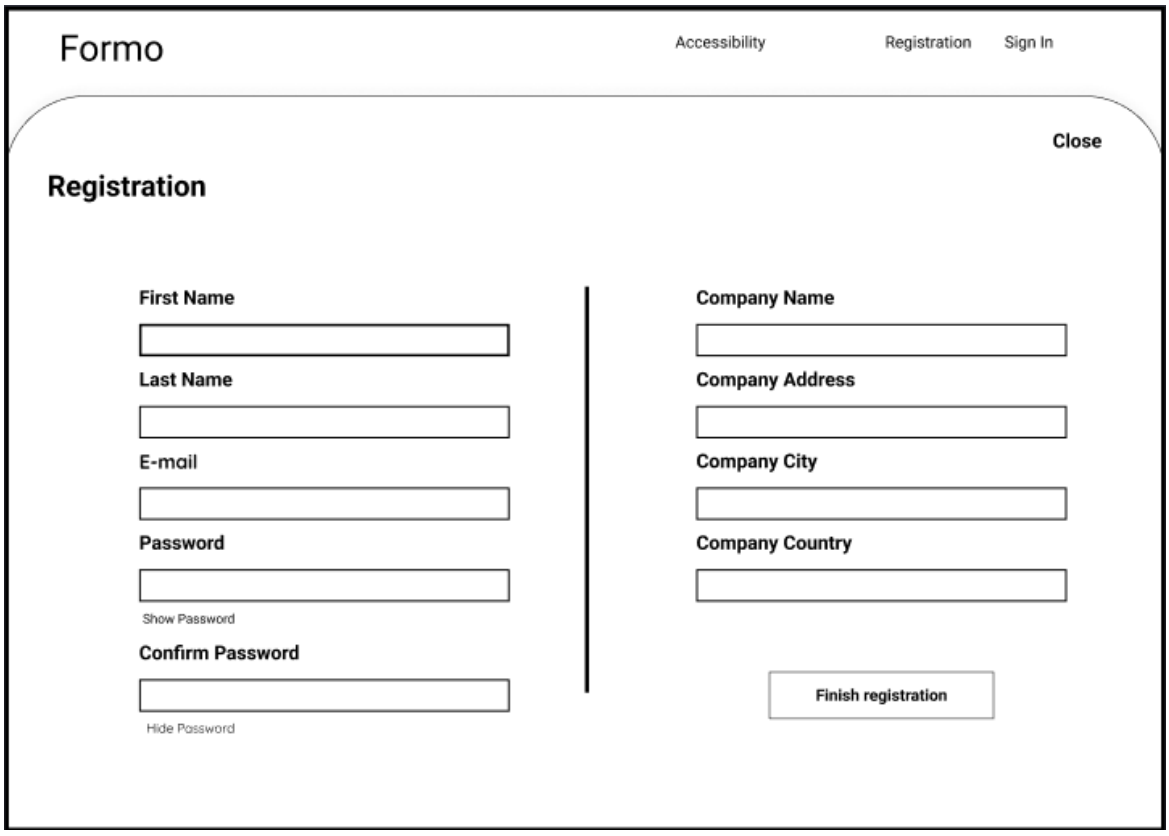
Shallow funkcija prikazuje samo trenutnu komponentu bez komponenti djece. Koristi se kada je potrebno izolirati komponentu za čisto jedinično testiranje koda.

Render funkcija kreira statičke *HTML* elemente trenutne komponente i njenih komponenti djece. Najčešće se koristi ukoliko se žele testirati snimke zaslona.

6. PROGRAMSKO RJEŠENJE KORISNIČKOG DIJELA WEB SUSTAVA

6.1. Dizajn korisničkog sučelja

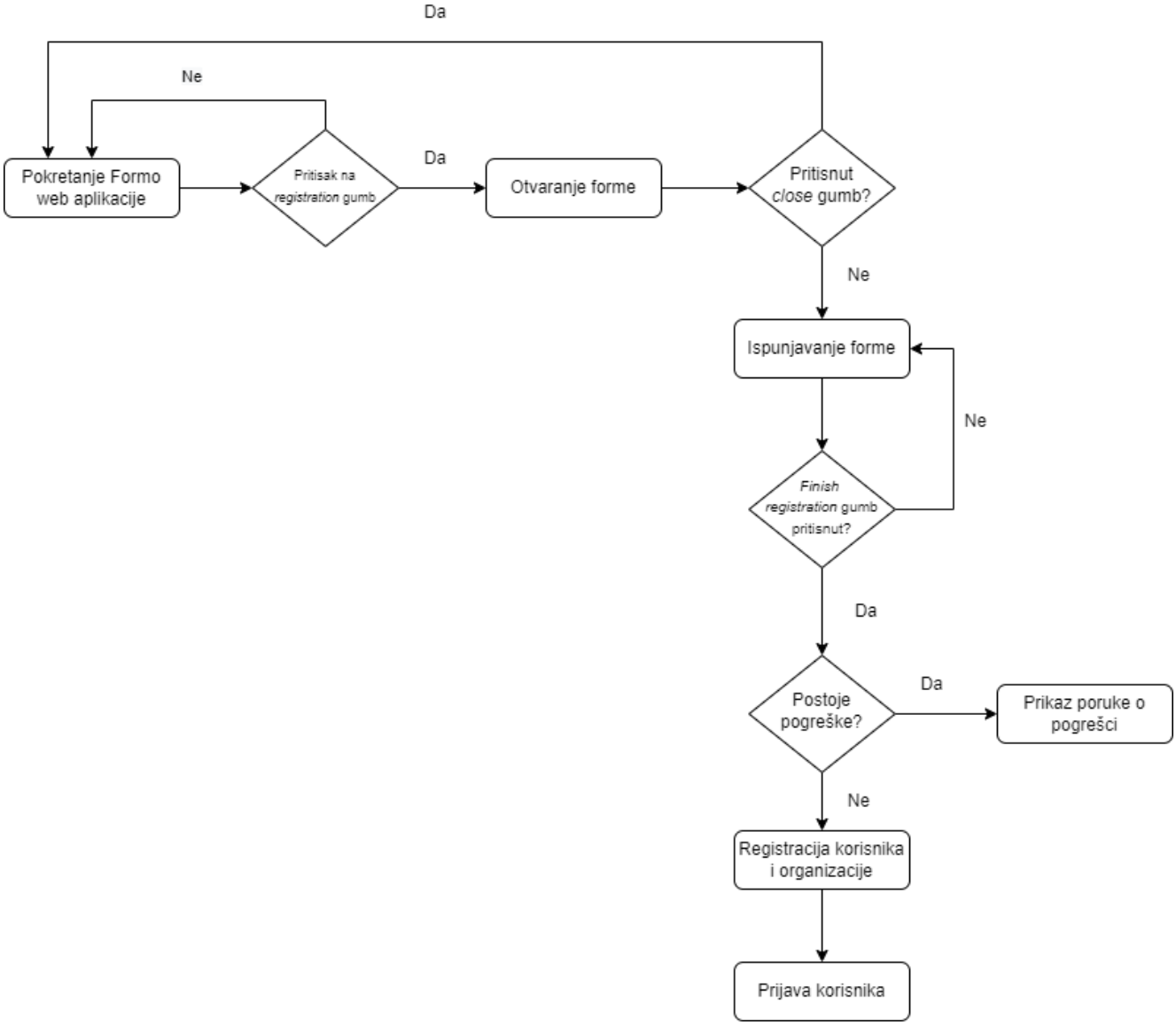
Nakon uspješno definiranih korisničkih zahtjeva na programsko rješenje, sljedeći važan korak je dizajn korisničkog sučelja. Prilikom dizajniranja korisničkog sučelja važno je obratiti pozornost na dvije stavke – sam izgled korisničkog sučelja te korisničko iskustvo prilikom korištenja sustava. Kao prvi korak dizajniranja korisničkog sučelja potrebno je napraviti skice potrebnih sučelja. Izradom skice može se uštedjeti vrijeme tako što se u početku ne troši vrijeme na odabir boja, fontova, ilustracija i slično, nego se raspoređuju elementi uzimajući u obzir korisničko iskustvo. Slika 6.1 prikazuje skicu sučelja za registraciju korisnika i organizacije.



Sl. 6.1. *Skica sučelja za registraciju korisnika i organizacije*

Prilikom kreiranja skica važno je uzeti u obzir načine interakcije korisnika sa sustavom kako bi korištenje sustava bilo što jednostavnije. Taj korak se naziva dizajniranje korisničkog iskustva te se pokušava postići što jednostavniji i brži korisnički protok do željenog cilja, poput registracije korisnika. U navedenom primjeru, ukoliko korisnik želi kreirati korisnički račun i organizaciju, na početnoj stranici potrebno je pritisnuti gumb *Registration* koji potom otvara formu za registraciju.

Nakon ispunjavanja forme, pritiskom na gumb *Finish registration* korisnik kreira osobni račun zajedno s organizacijom. Slika 6.2 prikazuje korisnički protok prilikom registracije



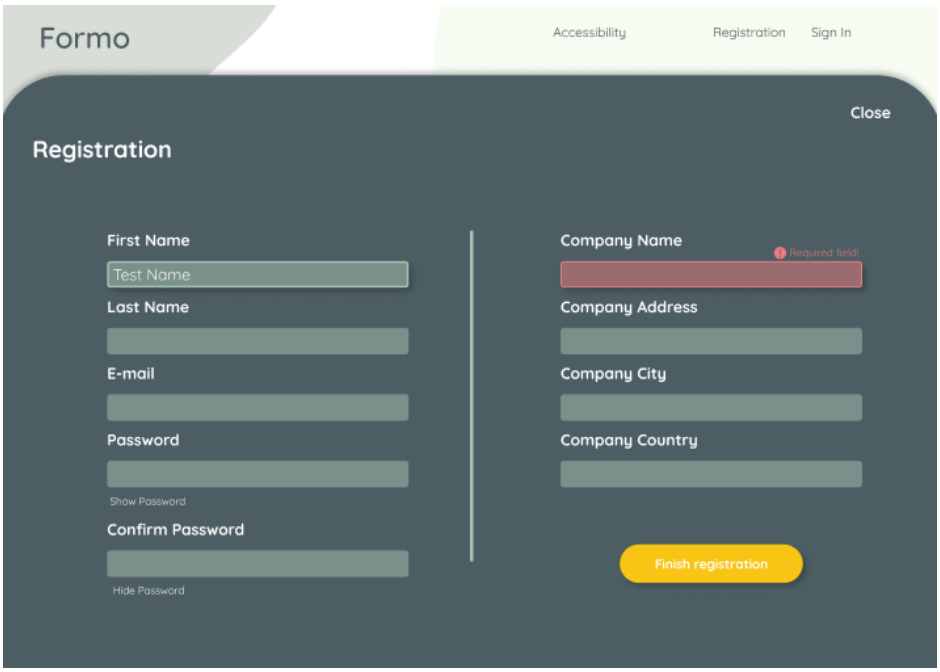
Sl. 6.2. *Korisnički protok prilikom registracije korisnika i organizacije*

Nakon kreiranja skica svih sučelja te odabranih optimalnih korisničkih protoka za što bolje korisničko iskustvo, slijedi dizajn korisničkog sučelja. Prije početka dizajniranja važno je odabrati palete boja, fontove koji će se primjenjivati, ilustracije, slike i ikone. Prema [38], prilikom dizajniranja korisničkog sučelja važno je voditi se osnovnim načelima dizajniranja kao što su:

- jednostavnost – korisničko sučelje mora biti što jednostavnije, bez nepotrebnih elemenata koji odvraćaju korisnike od važnih podataka i funkcionalnosti

- konzistentnost – potrebno je koristiti slične uzorke dizajna sa što manje različitih fontova ili boja
- jasnoća – korisničko sučelje mora biti intuitivno za korištenje s jasnim opisima i rasporedom elemenata
- pružanje relevantnih povratnih informacija – korisnike je potrebno obavještavati o trenutnom progresu ili odrađenim radnjama poput prikaza obavijesti o uspješnom ili neuspješnom kreiranju računa
- pristupačnost – korisničko sučelje treba biti prilagođeno slabovidnim osobama i osobama s oštećenjima vida.

Slika 6.3 prikazuje gotov dizajn forme za registraciju korisnika i organizacije uzimajući u obzir navedena načela.



Sl. 6.3. *Prikaz dizajna forme za registraciju korisnika i organizacije*

Prema slici 6.3 vidljivo je da je većina načela pokrivena. Korisničko sučelje je jednostavno, što znači da nema nepotrebnih elemenata koji korisnicima odvraćaju pažnju. Također, sučelje je konzistentno na način da se koristi određena paleta boja od tri glavne boje – zelene koja predstavlja tematsku boju, žute koja predstavlja boju naglašavanja i crvene koja predstavlja boju naglašavanja poruka o pogreškama. Jednostavnost sučelja vidi se i jasno naznačenim naslovima polja za unos kako bi korisnici znali što treba unijeti u formu, a pružanje relevantnih povratnih informacija vidljivo je na primjeru poruke o obaveznom polju ukoliko korisnik poželi dovršiti registraciju bez da unese obavezne podatke poput naziva organizacije.

6.2. Programska implementacija korisničkih zahtjeva

Programska implementacija korisničkog dijela programskog rješenja ostvarena je pomoću programskog jezika *Javascript* te biblioteka *React* i *Redux*. Za razvoj programskog rješenja korišteni su Visual Studio Code uređivač kôda i GitHub platforma koja omogućava jednostavno verzioniranje i pohranu kôda.

Korisnički dio programskog rješenja podijeljen je na više manjih ili većih komponenti koje se mogu koristiti više puta. Primjer jedne takve komponente je komponenta *Header* koja se nalazi na određenim sučeljima poput sučelja projekata ili zaposlenika. Komponenta *Header* uključuje se unutar neke druge komponente te joj se prosljeđuju određena svojstva kako bi se prikazale relevantne informacije. Svojstva koja je potrebno proslijediti komponenti su naslov, slika ili ilustracija, sadrži li dodatnu navigacijsku traku, vrstu navigacijske trake te svojstva trake. Slika 6.4 prikazuje programski kôd komponente *Header*.

```jsx
export const Header = ({
  title,
  image,
  withNavbar,
  navbarType,
  navbarProps,
}) => {
  const getNavbar = () => {
    switch (navbarType) {
      case 'settings':
        return (
          <SettingsNavbar
            activeItem={navbarProps.activeItem}
            setActiveItem={navbarProps.setActiveItem}
            userRole={navbarProps.userRole}
          />
        );
      case 'project':
        return (
          <ProjectNavbar
            activeItem={navbarProps.activeItem}
            setActiveItem={navbarProps.setActiveItem}
          />
        );
      default:
        return null;
    }
  };
  return (
    <div className='header'>
      {withNavbar ? (
        <>
          <div className='header__left'>
            <div className='header__left__title'>{title}</div>
            {getNavbar()}
          </div>
          <img className='header__img' src={image} alt='illustration' />
        </>
      ) : (
        <>
          <div className='header__title'>{title}</div>
          <img className='header__img' src={image} alt='illustration' />
        </>
      )}
    </div>
  );
};
```

Sl. 6.4. *Programski kôd komponente Header*

6.2.1. Početna stranica

Pokretanjem programskog rješenja korisničkog dijela dolazi se na početnu stranicu. Komponenta početne stranice sadrži manju komponentu navigacijske trake s nazivom programskog rješenja te gumbove za otvaranje izbornika za prilagodbu pristupačnosti, otvaranje registracijske forme ili forme za prijavu u sustav. Komponenta početne stranice sadrži i animiranu ilustraciju, kratku poruku korisnicima i još jedan, lako uočljiv, gumb putem kojeg se otvara forma za registraciju. Forme za registraciju i prijavu te izbornik za prilagodbu pristupačnosti zasebne su komponente koje su uključene u komponentu navigacijske trake i glavnu komponentu. Slika 6.5 prikazuje programski kôd početne stranice i način uključivanja ostalih komponenti te prosljeđivanja svojstava.

```
<div className='home-page'>
      <HomePageBlobs
        accessibilityActive={accessibilityActive}
        setAccessibilityActive={setAccessibilityActive}
      />
      <HomepageNavbar
        registrationActive={openRegistration}
        loginActive={openLogin}
        accessibilityActive={accessibilityActive}
        onRegistrationClick={onRegistrationClick}
        onLoginClick={onLoginClick}
        onAccessibilityClick={onAccessibilityClick}
        closeForm={closeForm}
      />
      <div
        className='home-page__main'
        onClick={() => {
          if (accessibilityActive) setAccessibilityActive(false);
        }}
      >
        <HomePageBody setOpenRegistration={setOpenRegistration} />
        <HomepageIllustration
          className='home-page__main__illustration'
          alt='homepage illustration'
        />
        <LoginAndRegistration
          isRegistrationActive={openRegistration}
          isLoginActive={openLogin}
          closeForm={closeForm}
          onRegistrationClick={onRegistrationClick}
          onLoginClick={onLoginClick}
        />
      </div>
    </div>
```

Sl. 6.5. *Prikaz programskog kôda početne stranice i uključenih komponenti*

Ukoliko korisnik posjeduje korisnički račun i želi se prijaviti u sustav, pritiskom na gumb *Sign in* unutar navigacijske trake otvara se forma za prijavu korisnika. Slika 6.6 prikazuje programski kôd koji sadrži navigacijska traka, a koji se odnosi na gumbove za registraciju i prijavu.

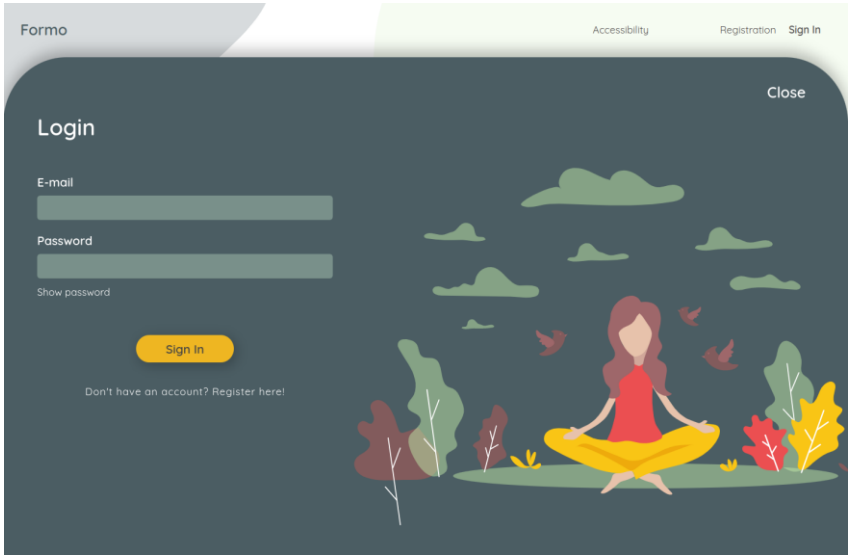
```jsx
<div className='home-page__navbar__login-register-btns'>
  <button
    type='button'
    className={classNames({
      'home-page__navbar__registration-btn': true,
      active: registrationActive,
    })}
    aria-label='registration'
    onClick={() => onRegistrationClick()}
  >
    Registration
  </button>
  <button
    type='button'
    className={classNames({
      'home-page__navbar__login-btn': true,
      active: loginActive,
    })}
    aria-label='login'
    onClick={() => onLoginClick()}
  >
    Sign In
  </button>
</div>
```

Sl. 6.6. *Prikaz programskog kôda gumbova za registraciju i prijavu*

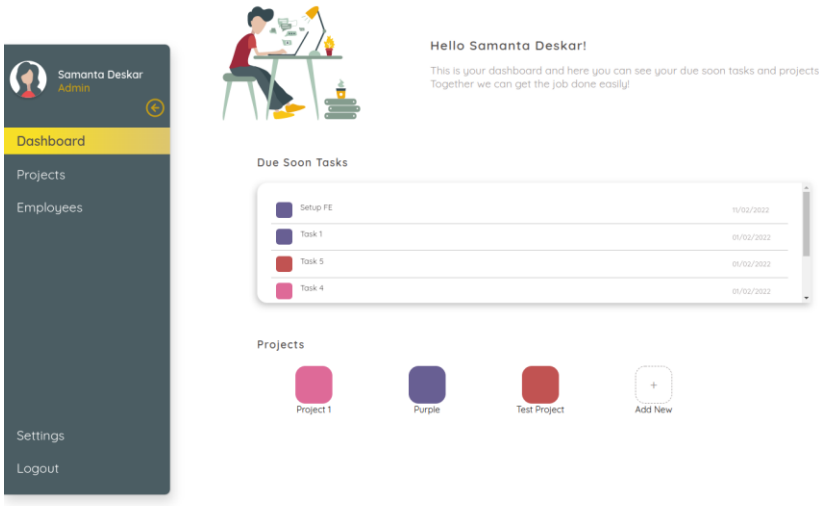
Prema slici 6.6 vidljivo je da oba gumba primaju funkciju *onClick* koja se izvršava kada je gumb pritisnut. Nakon pritiska gumba *Sign in* poziva se funkcija *onLoginClick* koja postavlja stanje *openLogin* unutar komponente početne stranice u istinito kako bi se mogla prikazati forma za prijavu. Forma za prijavu, prikazana na slici 6.7, sadrži dva polja za unos, jedno za unos adrese e-pošte i drugo za unos lozinke te gumb za prijavu. Nakon unosa podataka, podaci se spremaju u stanje *loginData* i čuvaju dok se ne pritisne gumb *Sign in* ili dok se ne zatvori forma. Pritiskom na gumb *Sign in,* uneseni podaci se šalju poslužitelju sa zahtjevom za prijavu. Nakon obrade podataka na poslužitelju, ukoliko korisnik ima valjani račun, poslužitelj vraća odgovor s tokenom pomoću kojeg može pristupati ostalim funkcionalnostima sustava te podacima o korisniku.

6.2.2. Kontrolna ploča

Nakon prijave u sustav, korisniku se prikazuje kontrolna ploča, prikazana na slici 6.8, koja sadrži pozdravnu poruku za korisnika, popis zadataka korisnika te projekte na kojima korisnik sudjeluje. Ukoliko korisnik ima ulogu administratora ili projektnog menadžera, korisniku se tada prikazuje i gumb za kreiranje novog projekta.



Sl. 6.7. *Prikaz forme za prijavu korisnika*



Sl. 6.8. *Prikaz kontrolne ploče*

Slika 6.9 prikazuje programski kôd komponente kontrolne ploče, popratnih komponenti koje su uključene unutar nje i predanih svojstva tim komponentama. Pritiskom na pojedini zadatak otvara se skočni prozor koji sadrži detalje zadatka. Da bi otvaranje detalja bilo moguće, na svaki zadatak postavljen je osluškivač pritiska koji poziva funkciju *handleOnTaskClick*. Ta funkcija tada postavlja stanje *taskDetailsOpened* u istinito, postavlja podatke trenutnog zadatka u stanje *currentTask* i na kraju prikazuje skočni prozor. Projekti također imaju postavljene osluškivače pritiska koji pozivaju funkciju *handleOnProjectClick* prikazanu na slici 6.10. Funkciji se predaje vrsta gumba i odabrani projekt. Unutar funkcije se tada provjerava vrsta pritisnutog gumba.

Ukoliko je pritisnut gumb za dodavanje novog projekta, stanje *isProjectFormActive* se postavlja u istinito te se prikazuje forma za dodavanje novog projekta. U suprotnom, sustav preusmjerava korisnika na novu *URL* adresu koja prikazuje detalje o odabranom projektu.

```
<div className='dashboard'>
      <DashboardHeader userName={userName} />
      <div className='dashboard__body'>
        <DashboardTask tasks={tasks} handleOnTaskClick={handleOnTaskClick} />
        <DashboardProjects
          user={user}
          projects={projects}
          handleOnProjectClick={handleOnProjectClick}
        />
      </div>
      {isProjectFormActive && (
        <CreateProjectForm
          isActive={isProjectFormActive}
          setIsActive={setIsProjectFormActive}
        />
      )}
      {taskDetailsOpened && (
        <TaskDetails
          task={currentTask}
          isActive={taskDetailsOpened}
          setIsActive={setTaskDetailsOpened}
        />
      )}
    </div>
```

Sl. 6.9. *Prikaz programskog kôda kontrolne ploče*

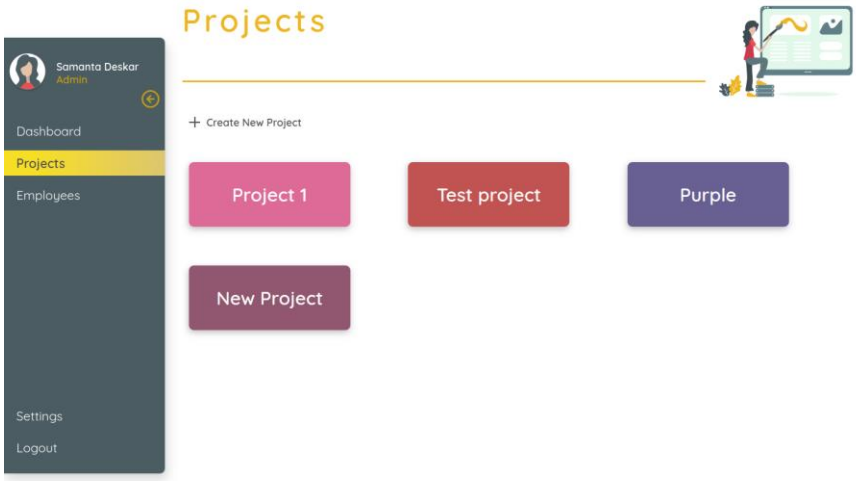
```
const handleOnProjectClick = (type, project) => {
  if (type === 'add') {
    setIsProjectFormActive(true);
  } else {
    window.location.replace(`/projects/${project._id}`);
  }
};
```

Sl. 6.10. *Prikaz programskog kôda funkcije handleOnProjectClick*

6.2.3. Projekti

Pritiskom na gumb *Projects*, na navigacijskoj traci korisničkog sučelja prijavljenog korisnika, otvara se korisničko sučelje svih projekata, prikazano na slici 6.11. Ukoliko je korisnik administrator ili projektni menadžer, prikazuje se i gumb za kreiranje novog projekta. Pritiskom na gumb otvara se forma kreirana pomoću komponente *Modal*. Komponenta *Modal* jedna je od višekratnih komponenti pomoću koje se kreiraju svi skočni prozori, dijalozi, poruke i slično. Na slici 6.12 prikazan je programski kôd unutar komponente *Modal*. Komponenta koristi portale, jedan od prvorazrednih načina prikaza komponenti izvan *DOM* hijerarhije roditeljske komponente. Pomoću portala, komponenta *Modal* ne prikazuje se unutar roditeljske komponente, što je u ovom

slučaju komponenta *Projects*, nego se prikazuje unutar korijenske komponente aplikacije što je prikazano na slici 6.13.



Sl. 6.11. *Korisničko sučelje projekata*

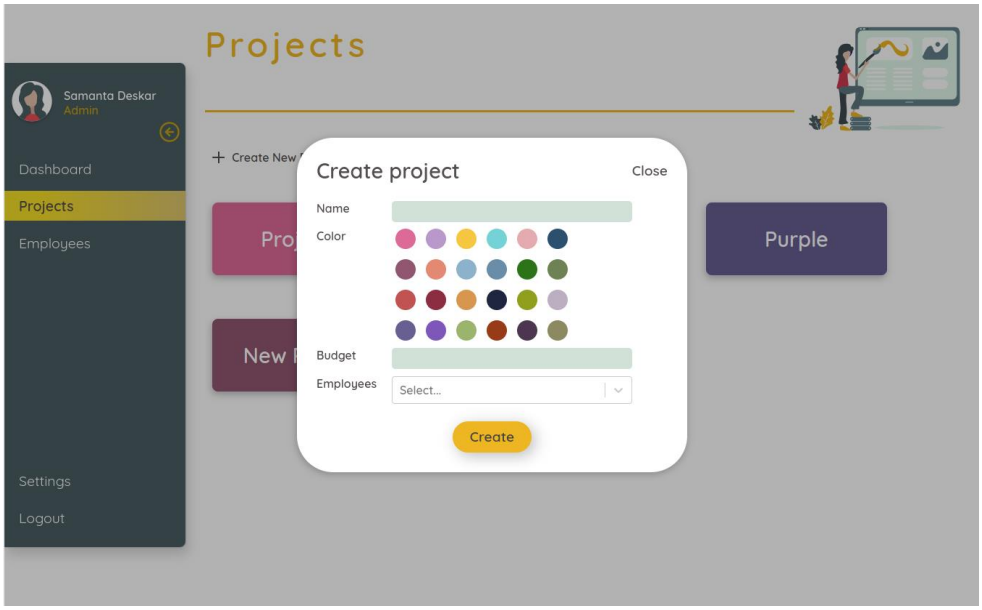
```
export const Modal = (props) => {
  const ref = useRef(null);

  useEffect(() => {
    ref.current.scrollIntoView();
  }, []);

  return reactDom.createPortal(
    <div
      className={classNames(
        {
          modal: true,
          active: props.isActive,
        },
        props.className
      )}
    >
      <div
        ref={ref}
        className={classNames({
          overlay: true,
          active: props.isActive,
        })}
        onMouseDown={props.closeModal}
      >
        <div
          className='overlay__children'
          onMouseDown={(e) => e.stopPropagation()}
        >
          {props.children}
        </div>
      </div>
    </div>,
    document.getElementById('root')
  );
};
```

Sl. 6.12. *Prikaz komponente Modal*

Na slici 6.13 vidljivo je da se unutar korijenskog elementa nalaze dva glavna elementa – *app* s elementima navigacijske trake i projekata te element *modal* koji prikazuje formu za kreiranje novog projekta. U slučaju da nisu korišteni portali, komponenta *Modal* bi bila prikazana unutar elementa *projects* s obzirom da ju ta komponenta poziva nakon pritiska gumba. Kako bi se prikazala željena forma, komponenti *Modal* se prosljeđuje forma sa svim potrebnim elementima i funkcijama putem svojstva *children*. Forma sadrži polje za unos naziva projekta, izbornik boja pomoću kojeg se projekt može lako razlikovati između ostalih projekata, polje za unos budžeta te izbornik za odabir zaposlenika koji će raditi na projektu. Slika 6.14 prikazuje korisničko sučelje forme za kreiranje projekata.

```
<!DOCTYPE html>
<html lang="en">
 ▸<head>…</head>
 ▾<body>
  ▾<div id="root">
   ▾<div class="app  positive-0"> flex
    ▸<div class="sidebar">…</div>
    ▸<div class="projects">…</div>
    </div>
  ▸<div class="modal active create-project">…</div> flex == $0
   </div>
   <script src="/static/js/bundle.js"></script>
   <script src="/static/js/vendors~main.chunk.js"></script>
   <script src="/static/js/main.chunk.js"></script>
  </body>
</html>
```

Sl. 6.13. *Prikaz strukture HTML elemenata prilikom prikaza Modala*



Sl. 6.14. *Prikaz forme za kreiranje projekata*

Pritiskom na jedan od postojećih projekata prikazuje se novo korisničko sučelje s detaljima odabranog projekta. Početna stranica odabranog projekta prikazuje naziv projekta, budžet, zadatke i zaposlenike tog projekta. Pritiskom na jedan od zadataka otvaraju se detalji zadatka. Ukoliko se zadaci žele pregledavati pomoću ploče *Kanban*, na navigacijskoj traci unutar projekta, treba pritisnuti gumb *Tasks* koji tada otvara sučelje zadataka. Ploča *Kanban* odabrana je za jedan od načina prikaza zadataka jer vizualizira proces razvoja te omogućava identifikaciju potencijalnih problema unutar procesa. Zadaci se mogu filtrirati prema zaposleniku kojem je dodijeljen zadatak ili se mogu pretraživati prema nazivu zadatka. Slika 6.15 prikazuje kôd funkcija za filtriranje i pretraživanje zadataka. Na primjeru funkcija za filtriranje i pretraživanje vidljivo je korištenje tzv. *guard* clauses, odnosno zaštitnih klauzula. Zaštitne klauzule predstavljaju jednostavnu provjeru uvjeta kako bi se smanjila kompleksnost kôda. Ukoliko je zaštitna klauzula zadovoljena, izvođenje funkcije se završava s povratnom vrijednošću ili iznimkom [39]. Na taj način smanjuje se vrijeme izvođenja ukoliko uvjeti za zadatak funkcije nisu zadovoljeni. U suprotnom, funkcija nastavlja s radom i izvodi potrebne naredbe. Na primjer, funkciji *filterTasksByAssignee* potrebno je predati zaposlenika prema kojem se žele filtrirati zadaci. Ukoliko zaposlenik nije predan funkciji, zaštitna klauzula prekida izvođenje funkcije. U suprotnome, filtrira zadatke tako da pregleda sve zadatke i zaposlenike koji su dodijeljeni i sprema u novu varijablu sve zadatke predanog zaposlenika. Nakon što su svi zadaci pregledani, filtrirani zadaci se postavljaju u stanje *filteredTasks*, a prikaz komponente se osvježava s novim zadacima.

```
1  const filterTasksByAssignee = (assignee) => {
2    if (!assignee) {
3      setFilteredTasks(tasks);
4      return;
5    }
6    const filtered = tasks.filter(
7      (task) => task.assignee && task.assignee._id === assignee.value
8    );
9    setFilteredTasks(filtered);
10 };
11
12 const filterTasksBySearch = (value) => {
13   if (!value) {
14     setFilteredTasks(tasks);
15     return;
16   }
17   const filtered = tasks.filter((task) => task._id === value.value);
18   setFilteredTasks(filtered);
19 };
```

Sl. 6.15. *Prikaz funkcija za filtriranje i pretraživanje zadataka*

Ukoliko se zadaci žele pregledavati prema datumima ili ovisnostima o drugim zadacima, pritiskom na gumb *Calendar* na navigacijskoj traci trenutnog projekta otvara se prikaz zadataka pomoću

gantograma. Prikaz zadataka se može prilagođavati tako da prikazuje zadatke prema mjesecima, tjednima, danima, svakih 12 sati ili svakih 6 sati. Ako se prilikom vođenja projekta projektni menadžeri služe okvirom *Scrum*, gantogram omogućuje učinkovit prikaz zadataka unutar *sprinta*, odnosno prikaz početnih i završnih datuma zadataka. Za prikaz gantograma korištena je biblioteka *Frappe Gantt React*. Biblioteka je nastala prema *Frappe*-ovoj biblioteci *Gantt* te je prilagođena za korištenje unutar *React* aplikacija. Slika 6.16 prikazuje kôd za prikaz gantograma ili poruke ukoliko ne postoje zadaci za trenutni projekt. Kako bi se prikazao gantogram, potrebno je provjeriti postoje li zadaci za trenutni projekt te ukoliko postoje, tada se komponenti *FrappeGantt* predaju zadaci, način prikaza, osluškivač promjene datuma te osluškivač pritiska na zadatak. U suprotnome, prikazuje se poruka korisniku da ne postoje zadaci za trenutni projekt.

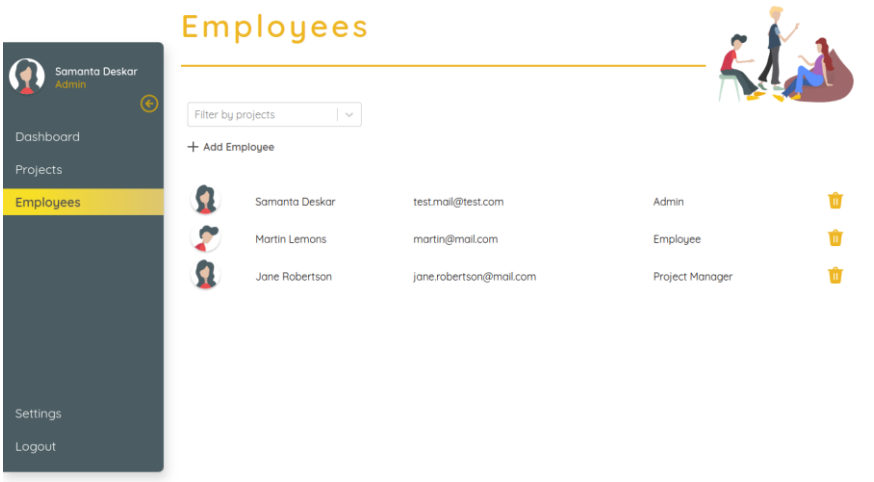
```
<div className='calendar__gantt'>
  {mappedTasks && mappedTasks.length > 0 ? (
    <FrappeGantt
      tasks={mappedTasks}
      viewMode={mode}
      onDateChange={(task, start, end) =>
        updateDates(task.id, start, end)
      }
      onClick={(task) => {
        findCurrentTask(task);
      }}
    />
  ) : (
    <div className='calendar__no-tasks'>
      <img
        src={alertIcon}
        className='calendar__no-tasks__icon'
        alt='alert'
      />
      <div className='calendar__no-tasks__msg'>
        There are no tasks for a current project
      </div>
    </div>
  )}
</div>
```

Sl. 6.16. *Prikaz elemenata komponente Calendar*

6.2.4. Zaposlenici

Nakon prijave korisnika, pritiskom na gumb *Employees* na navigacijskoj traci otvara se korisničko sučelje za prikaz zaposlenika unutar organizacije, koje je vidljivo na slici 6.17. Zaposlenici se mogu filtrirati prema projektima na koji su uključeni, a ukoliko je korisnik administrator, tada može dodavati nove zaposlenike ili brisati postojeće. Popis zaposlenika prikazan je pomoću višekratne komponente *Table* unutar koje je korištena biblioteka *React Table*. Komponenti *Table* potrebno je predati željene stupce i podatke kojima se tablica želi popuniti. Svaki stupac predstavlja objekt koji sadrži element *accessor* ili *Cell* pomoću kojeg zna koji podaci pripadaju

određenom stupcu. Prema slici 6.18 vidljivo je da ukoliko se koristi element *Cell* tada se određena ćelija može oblikovati na željeni način. Na primjer, ukoliko je potrebno ime i prezime zaposlenika prikazati u jednoj ćeliji, tada se unutar elementa *Cell* dodaje *HTML* kôd sa željenim načinom prikaza. Ukoliko nije potrebna prilagodba prikaza podataka, tada se koristi element *accessor* koji prikazuje podatke onako kako su dobiveni od strane poslužitelja.



Sl. 6.17. *Prikaz korisničkog sučelja zaposlenika*

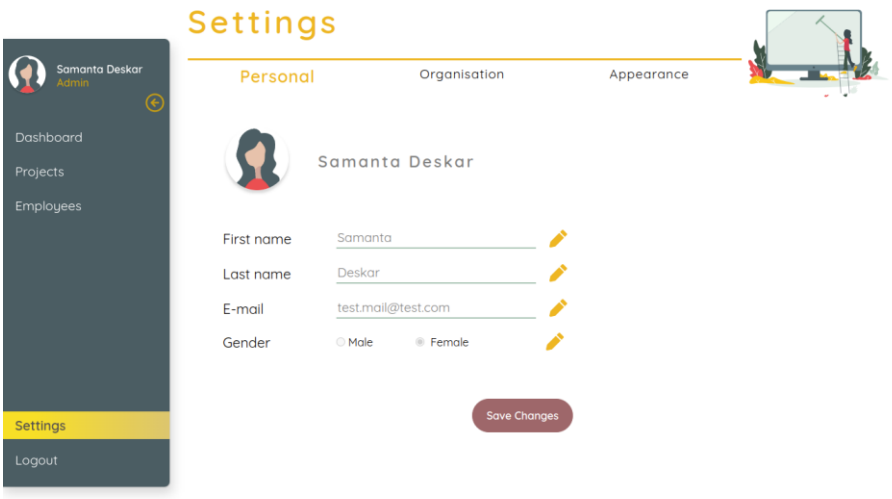
6.2.5. Postavke

Svaki korisnik nakon prijave u sustav ima mogućnost prilagodbe vlastitih postavki. Pritiskom na gumb *Settings* na navigacijskoj traci prikazuje se korisničko sučelje postavki. Komponenta *Settings* sadrži višekratnu komponentu *Header* unutar koje se nalazi i navigacijska traka za odabir željenih postavki.

Odabirom osobnih postavki, korisniku se prikazuje forma, prikazana na slici 6.19, s ispunjenim osobnim podacima. Kao dodatan oblik sigurnosti od neželjenih promjena, korisnik mora pritisnuti ikonu olovke kako bi omogućio unos novog podatka te pritisnuti ikonu kvačice kako bi potvrdio unos i omogućio gumb za spremanje podataka. Slika 6.20 prikazuje programski kôd za izmjenu imena korisnika te način izmjene ikone ovisno je li izmjena omogućena ili nije.

```jsx
const columns = useMemo(() => [
  {
    id: 'image',
    Cell: ({ row }) => (
      <img
        src={row.original.sex === 'male' ? manProfile : womanProfile}
        alt='profile illustration'
        className='employees-table__img'
      />
    ),
  },
  {
    id: 'name',
    Cell: ({ row }) => (
      <div className='employee-table__name'>
        {row.original.firstName} {row.original.lastName}
      </div>
    ),
  },
  { accessor: 'email' },
  {
    id: 'role',
    Cell: ({ row }) => (
      <div className='employee-table__role'>{getRole(row.original.role)}</div>
    ),
  },
  { accessor: 'projects' },
  {
    id: 'actionButtons',
    Cell: ({ row }) => (
      <div className='employees-table__action-btns'>
        <button
          className='employees-table__action-btns__delete'
          type='button'
          onClick={() => removeUser(row.original._id)}
        >
          <img
            src={highContrast ? deleteIconWhite : deleteIcon}
            alt='remove employee'
          />{' '}
        </button>
      </div>
    ),
  },
]);
```

Sl. 6.18. *Prikaz polja objekata stupaca tablice zaposlenika*



Sl. 6.19. *Prikaz korisničkog sučelja osobnih postavki*

```
<div className='settings__personal__form__first-name'>
  <div className='settings__personal__form__first-name__label'>
    First name
  </div>
  <input
    type='text'
    defaultValue={firstName}
    onChange={(e) =>
      setUpdatedInfo({ ...updatedInfo, firstName: e.target.value })
    }
    disabled={!firstNameEditable}
  />
  <img
    src={firstNameEditable ? getCheckMark() : getEditIcon()}
    className='settings__personal__form__first-name__icon'
    onClick={() => setFirstNameEditable(!firstNameEditable)}
    alt={firstNameEditable ? 'check mark icon' : 'edit icon'}
    onKeyDown={(e) => {
      if (e.key === 'Enter') setFirstNameEditable(!firstNameEditable);
    }}
  />
</div>
```

Sl. 6.20. *Prikaz programskog kôda za izmjenu korisničkog imena*

Pritiskom na gumb *Organisation* na navigacijskoj traci postavki, prikazuje se korisničko sučelje za izmjenu podataka organizacije. Korisnik tada može mijenjati naziv organizacije, adresu, grad i državu. Promjena podataka organizacije izvedena je na isti način kao i promjena osobnih podataka.

Odabirom pristupačnosti na navigacijskoj traci postavki, korisniku se prikazuje korisničko sučelje za prilagodbu pristupačnosti kako bi se slabovidnim korisnicima i korisnicima s oštećenjima vida omogućilo jednostavnije korištenje programskog rješenja. Postavke pristupačnosti sastoje se od gumbova u obliku sklopki, potvrdnih okvira te gumbova za povećavanje i smanjivanje veličine slova. Svaki od gumbova ima vlastiti osluškivač pritiska koji poziva funkciju *onSwitchChange* ukoliko je pritisnut gumb u obliku sklopke, funkciju *handleDaltonismTypeChange* ukoliko je pritisnut potvrdni okvir za odabir vrste daltonizma te funkciju *handleFontSizeChange* ukoliko je pritisnut gumb za povećanje ili smanjenje veličine slova. Slika 6.21 prikazuje primjer kôda funkcije *onSwitchChange*.

Na primjeru kôda funkcije *onSwitchChange* vidljivo je da prilikom svake promjene stanja gumba u obliku sklopke, poziva se određeni kreator akcije koji postavlja novo stanje postavki pristupačnosti. Na taj način promjene ostaju pohranjene u *Redux* skladištu te se mogu dohvaćati u svakoj komponenti kako bi pristupačnost bila primjenjiva unutar svih korisničkih sučelja. Postavke pristupačnosti, kao i osobne postavke te postavke organizacije spremaju se i u bazu podataka slanjem zahtjeva poslužitelju. Detaljniji opis o načinu komunikacije s poslužiteljskom stranom biti će naveden u poglavlju 6.3.

```
const onSwitchChange = (type) => {
  switch (type) {
    case 'dyslexia': {
      dispatch(setDyslexia(!dyslexia));
      break;
    }
    case 'contrast': {
      dispatch(setHighContrast(!highContrast));
      break;
    }
    case 'daltonism': {
      dispatch(
        setDaltonism({
          isActive: !daltonism.isActive,
          redGreen: daltonism.redGreen,
          blueYellow: daltonism.blueYellow,
        })
      );
      break;
    }
    default:
      break;
  }
};
```

Slika 6.21. *Prikaz kôda funkcije onSwitchChange*

6.3. Komunikacija korisničke s poslužiteljskom stranom

Kako bi programsko rješenje u potpunosti ispunjavalo sve zahtjeve potrebno je razviti i poslužiteljsku stranu. Poslužiteljska strana programskog rješenja razvijena je za potrebe diplomskog rada pod nazivom "Programsko rješenje poslužiteljskog dijela web sustava za potporu upravljanja projektima agilnog razvoja programske podrške zasnovano na višenitnom paralelizmu i arhitekturi mikrousluga" [1].

Prema [1], poslužiteljska strana bazirana je na arhitekturi mikrousluga što znači da se sastoji od više neovisnih dijelova koji sadrže vlastite modele podataka i logiku. Iako su mikrousluge međusobno neovisne, potrebno je ostvariti komunikaciju između svake mikrousluge i korisničke strane. Kako bi se ostvarila komunikacija korišten je programski uzorak *API Gateway* koji se nalazi između korisničke i poslužiteljske strane [1]. Korisnička strana prilikom slanja zahtjeva na krajnju točku poslužitelja šalje zahtjev na *API Gateway* koji prosljeđuje zahtjev određenoj mikrousluzi na poslužiteljskoj strani.

6.3.1. Slanje zahtjeva s korisničke strane programskog rješenja

Gotovo svaki dio programskog rješenja zahtjeva podatke i odgovore od strane poslužitelja. Na korisničkoj strani slanje zahtjeva je izvedeno pomoću *Redux* akcija i *Redux* međusloja te programske biblioteke *Axios*.

Prilikom prvog korištenja programskog rješenja, korisnik mora dovršiti registraciju vlastitog korisničkog računa i organizacije. Nakon ispunjavanja forme za registraciju, pritiskom na gumb *Finish registration* pokreće se *Redux* akcija, prikazana na slici 6.22, s podacima unesenim u formu. Primjer unesenih podataka unutar forme prikazan je na slici 6.23.

```
export const registerAdmin = (data) => ({
  type: REGISTER_ADMIN,
  payload: data,
});
```

Sl. 6.22. *Primjer kôda Redux kreatora akcije za registraciju*

```
const registrationData = {
  firstName: 'Samanta',
  lastName: 'Deskar',
  password: 'Testing123',
  confirmPassword: 'Testing123',
  organisationName: 'My First Company',
  organisationAddress: 'Comapany Adress 23',
  organisationCity: 'Company City',
  organisationCountry: 'Company Country',
  settings: [
    {
      accessibility: {
        dyslexia: false,
        fontSize: 0,
        highContrast: false,
        daltonism: {
          isActive: false,
          redGreen: false,
          blueYellow: false,
        },
      },
    },
  ],
};
```

Sl. 6.23. *Primjer objekta podataka prilikom registracije korisnika*

Nakon što je kreator akcije izvršen, presreće ga *Redux* međusloj za korisničke akcije koji provjerava vrstu akcije te poziva funkciju za registraciju korisnika, prikazanu na slici 6.24. Unutar funkcije za registraciju korisnika poziva se *Axios* funkcija kojoj se predaje vrsta zahtjeva, *URL* krajnje točke na poslužitelju i podaci koje je potrebno spremiti u bazu podataka. S obzirom da je *Axios* temeljen na *Promise* objektima koji govore da će izvođenje funkcije eventualno biti uspješno ili neuspješno izvršeno, ulančavaju su pozivi funkcija ovisno je li zahtjev na poslužitelja uspješno ili neuspješno izvršen. Ukoliko je zahtjev uspješno izvršen te je uspješno zaprimljen odgovor

poslužitelja, izvodi se funkcija *then* unutar koje se poziva novi *Redux* kreator akcije za prijavu korisnika u sustav. U suprotnome, ukoliko dođe do neuspješnog slanja zahtjeva i primanja odgovora, izvršit će se funkcija *catch* koja poziva *Redux* kreator akcije za slanje obavijesti o neuspješnom kreiranju računa.

```
const registerAdmin = (data) => {
  axios({
    method: 'POST',
    url: `${process.env.REACT_APP_API_URL}/user/first`,
    data: {
      user: {
        firstName: data.firstName,
        lastName: data.lastName,
        email: data.email,
        password: data.password,
        settings: data.settings,
        role: 'admin',
      },
      org: {
        name: data.organisationName,
        address: data.organisationAddress,
        city: data.organisationCity,
        country: data.organisationCountry,
      },
    },
  })
    .then(() => {
      store.dispatch(
        loginUser({
          email: data.email,
          password: data.password,
        })
      );
    })
    .catch(() => {
      store.dispatch(errorUserCreated());
    });
};
```

Sl. 6.24 *Primjer kôda funkcije za slanje zahtjeva poslužitelju prilikom registracije korisnika*

Nakon registracije i prijave korisnika u sustav, korisnik određenim radnjama šalje razne zahtjeve poslužitelju. Prijavom u sustav svaki korisnik dobiva jedinstveni identifikacijski token (eng. *JWT, JSON Web Tokens*) koji je kriptiran, a sadrži algoritam kriptiranja, vrstu tokena te osnovne podatke korisnika poput ID-a korisnika unutar baze podataka te trajanja tokena. Slanjem svakog zahtjeva poslužitelju, obavezno je uključiti autorizacijsko zaglavlje unutar kojeg se šalje korisnički token. Ukoliko je token neispravan ili je istekao, poslužitelj vraća odgovor s porukom o neispravnoj autentifikaciji i zabranjenom pristupu podacima. U suprotnome, ukoliko je token valjan i korisnik ima pravo pristupa podacima, poslužitelj uz ispravan status šalje i zatražene podatke. Na slici 6.25 prikazana je funkcija koja šalje zahtjev poslužitelju za podatke o svim zaposlenicima unutar organizacije.

```
const fetchEmployees = () => {
  const { organisation, token } = state.user.currentUser;
  axios({
    method: 'GET',
    url: `${process.env.REACT_APP_API_URL}/user/org/${organisation}?`,
    headers: {
      Authorization: `Bearer ${token}`,
    },
  })
    .then((response) => store.dispatch(allEmployeesFetched(response.data)))
    .catch((error) => store.dispatch(errorFetch(error.message)));
};
```

Sl. 6.25. *Primjer funkcije za slanje zahtjeva poslužitelju prijavljenog korisnika*

Ako su podaci uspješno zaprimljeni, poziva se *Redux* kreator akcije *allEmployeesFetched*. Prema slici 6.25, nakon izvršavanja kreatora akcije, *Redux* reduktor presreće funkciju i provjerava vrstu akcije. Ukoliko reduktor, prikazan na slici 6.26, sadrži uvjet koji se podudara s vrstom akcije, tada mijenja stanje unutar *Redux* skladišta podataka. Nakon ažuriranja podataka unutar skladišta, sve komponente koje su pretplaćene na podatke koji su se ažurirali, osvježavaju svoje korisničko sučelje te prikazuju nove podatke.

```
export const user = (state = initialState, action) => {
  switch (action.type) {
    case USER_LOGGED_IN: {
      return {
        ...state,
        currentUser: action.payload,
        isAuthenticated: true,
      };
    }
    case LOGOUT_USER: {
      return { ...state, currentUser: null, isAuthenticated: false };
    }
    case ALL_EMPLOYEES_FETCHED: {
      return { ...state, employeesInOrganisation: action.payload };
    }
    default:
      return state;
  }
};
```

Sl. 6.26. *Primjer kôda Redux reduktora korisnika*

Gotovo svakom korisničkom akcijom, poput pritiska na gumb, u pozadini se izvršavaju slanja zahtjeva na poslužitelj. Slanja zahtjeva i dobivanje odgovora događa se vrlo brzo te, u gotovo milisekundama, korisnik dobiva relevantne informacije na korisničkom sučelju. Svako slanje zahtjeva izvršava se na isti način kao što se šalje zahtjev u navedenim primjerima registracije korisnika i dohvaćanja svih zaposlenika unutar organizacije. Iako je način slanja svakog zahtjeva jednak, postoje razlike u podacima koji se šalju prilikom slanja zahtjeva. Ti podaci mogu biti različiti ID-jevi projekata, zadataka, korisnika i slično, različite krajnje točke na poslužitelju ili

različite metode slanja poput *POST*, *GET*, *UPDATE* i *DELETE*. Slanjem različitih metoda, podataka i krajnjih točaka, poslužitelj nakon zaprimanja zahtjeva zna točno koji zahtjev je poslan te što mora vratiti kao odgovor [1].

6.4. Prilagodba programskog rješenja osobama s oštećenjima vida

Jedan od funkcionalnih zahtjeva programskog rješenja je i prilagodba sustava osobama s oštećenjima vida. U poglavlju 3 opisane su najčešće vrste oštećenja vida i načini prilagodbe korisničkog sučelja. Vođeni time, programsko rješenje prilagođeno je i testirano alatima prema WCAG smjernicama.

Kako bi se omogućilo mijenjanje postavki prilagodbe osobama s oštećenjima vida, kreirana je zasebna forma, prikazana na slici 6.27, pomoću koje se jednostavno uključuju i isključuju određene postavke.

Sl. 6.27. *Forma za promjenu postavki prilagodbe osobama s oštećenjima vida*

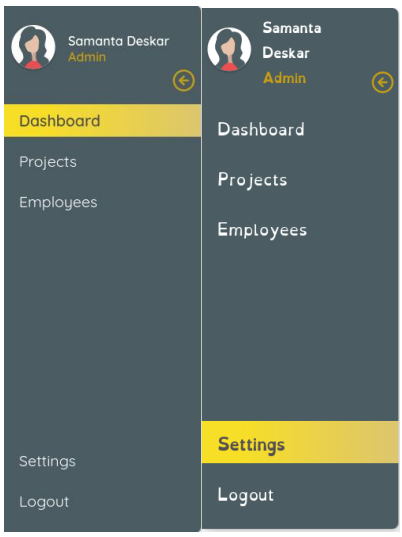
Unutar forme mogu se uključivati i isključivati prilagodbe za osobe s disleksijom ili daltonizmom te se može uključiti način visokog kontrasta ili promijeniti veličina fonta unutar cijelog sustava. Uključivanjem jedne od opcija, poziva se *Redux* kreator akcija koji pomoću *Redux* reduktora ažurira podatke unutar *Redux* skladišta, a pomoću *Redux* međusloja šalje zahtjev poslužitelju kako bi se ažurirale postavke korisnika unutar baze podataka. Slika 6.28 prikazuje kako se i u kojim slučajevima mijenjaju klasni nazivi korijenske komponente *App*. Promjenom klasnog naziva komponente, unutar datoteka za primjenu stilova, ovisno o klasnom nazivu, primjenjuju se određena pravila poput promjene fonta, promijene pozadinskih boja, boja teksta i slično.

```
<div
  className={classNames(
    {
      'app ': true,
      dyslexia: dyslexia ? 'dyslexia' : null,
      'high-contrast': highContrast ? 'high-contrast' : null,
      'red-green': daltonism.isActive && daltonism.redGreen,
      'blue-yellow': daltonism.isActive && daltonism.blueYellow,
    },
    getFontSizeClassName()
  )}
>
  <Router>
    {isAuthenticated && <Sidebar />}
    <Switch>
      <Routes />
    </Switch>
    <NotificationModal />
  </Router>
</div>
```

Sl. 6.28. *Primjer kôda komponente App komponente i klasnih naziva*

6.4.1. Prilagodba programskog rješenja osobama s disleksijom

Kako bi sustav bio prilagođen osobama s disleksijom, na početnoj stranici sustava ili unutar postavki korisničkog računa moguće je uključiti postavke za osobe s disleksijom. Prateći smjernice, sustav izbjegava korištenje kurziva, podcrtavanje riječi te pisanje velikim slovima. Ne koriste se veliki odlomci tekstova, a samim uključivanjem postavke mijenja se font kroz cijeli sustav. Prema preporukama, font se mijenja u font *OpenDyslexic*. Font *OpenDyslexic* je besplatan font koji je dizajniran kako bi se izbjegli najčešći simptomi i poteškoće kod osoba s disleksijom. Slova imaju zadebljane donje dijelove kako bi se lakše prepoznao smjer slova, odnosno kako bi osobe s disleksijom znale koji dio slova je donji dio. Slika 6.29 prikazuje izgled navigacijske trake prijavljenog korisnika bez uključene prilagodbe disleksiji i kontrolnu ploču s uključenom prilagodbom.

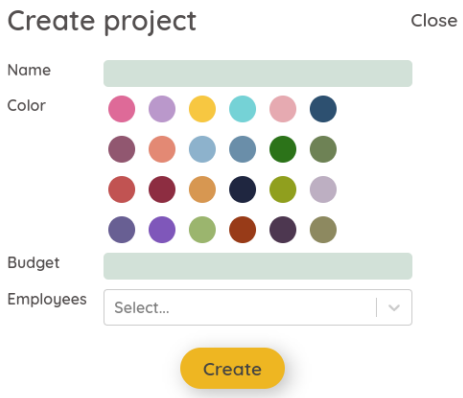


Sl. 6.29. *Usporedba korisničkog sučelja bez prilagodbe osobama s disleksijom i s prilagodbom*

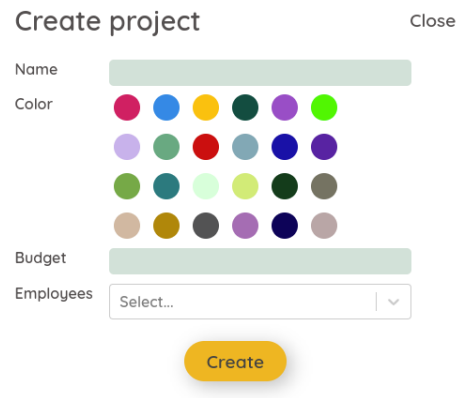
Na slici 6.29 vidljivo je da ukoliko nije uključena prilagodba, slova su manja, prored i razmak je manji te je korišten zadani font *Quicksand*. Nakon uključivanja prilagodbe disleksiji, mijenja se zadani font u font *OpenDyslexic*, slova postaju veća te se povećavaju razmak i prored između slova.

6.4.2. Prilagodba programskog rješenja osobama s daltonizmom

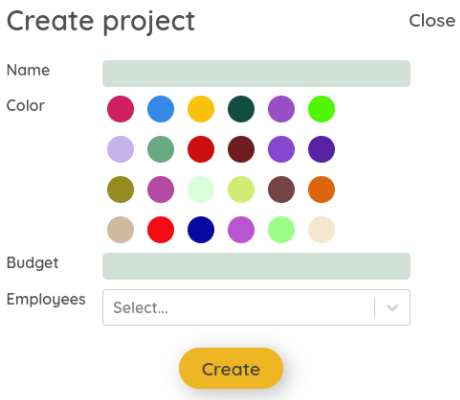
S obzirom da osobe s daltonizmom razlikuju nijanse, ali ne razlikuju određene boje, ovisno o vrsti daltonizma, uključivanjem postavki za prilagodbu daltonizmu ne mijenja se izgleda cijelog programskog rješenja, nego određenih paleta boja. Uključivanjem postavki za daltonizam potrebno je odabrati vrstu daltonizma, odnosno crveno-zelenu vrstu ili plavo-žutu. Odabirom određene vrste, unutar korijenske komponente *App* mijenja se klasni naziv, a ovisno o klasnom nazivu primjenjuje se određena paleta boja unutar programskog rješenja. Na slikama 6.30, 6.31 i 6.32 vidljiva je promjena paleta boja za odabir boje projekta, ovisno o uključenoj vrsti daltonizma.



Sl. 6.30. *Prikaz forme za kreiranje projekta bez uključene prilagodbe daltonizmu*



Slika 6.31. *Prikaz forme za kreiranje projekta s uključenom prilagodbom daltonizmu i crveno-zelenom vrstom*



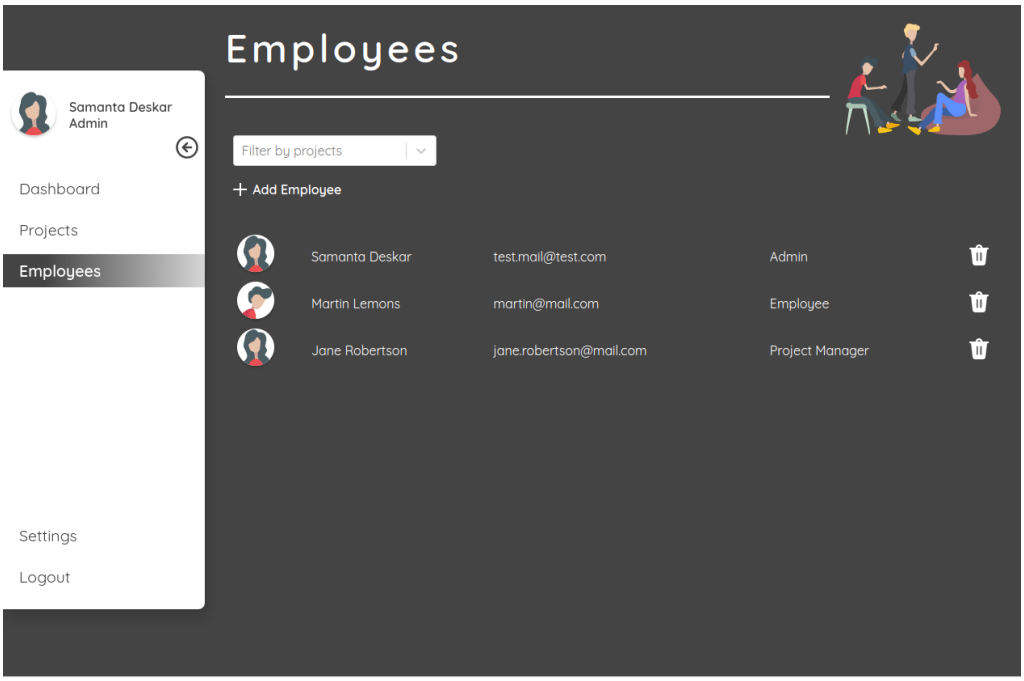
Slika 6.32. *Prikaz forme za kreiranje projekta s uključenom prilagodbom daltonizmu i plavo-žutom vrstom*

Kod odabira palete boja za crveno-zelenu vrstu daltonizma pazi se da crvene i zelene nijanse boje ne budu sličnih svjetlina i sličnih vrijednosti kontrasta. Prilikom testiranja paleta, korišten je alat za simulacije različitih vrsta daltonizma *Coblis*. Korištenjem navedenog alata može se vidjeti kako osobe s različitim vrstama daltonizma percipiraju određene boje i nijanse.

Iako su tematske boje programskog rješenja većinom nijanse crvene i zelene, u ovom slučaju nije bilo potrebe prilagođavanja boja jer su svjetline i vrijednosti kontrasta dovoljno različiti da se mogu razlikovati. Također, crvene i zelene nijanse koje se protežu sustavom, ni u jednom trenutku ne nalaze se kao kombinacije boja za pozadinu i tekst pa u tom slučaju ne dolazi do otežanog korištenja sustava.

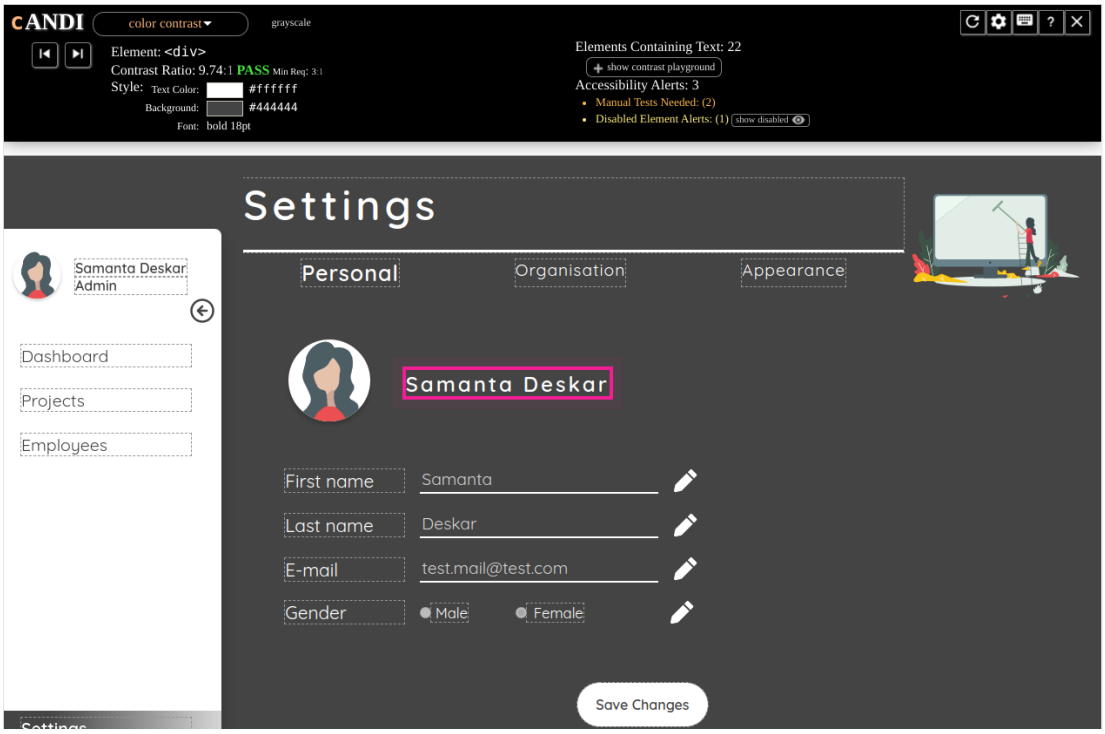
6.4.3. Prilagodba sustava visokim kontrastom

Kako bi se olakšalo korištenje osobama osjetljivim na jaku svjetlinu te osobama s astigmatizmom, korisnici unutar postavki mogu uključiti prilagodbu visokim kontrastom. Uključivanjem prilagodbe visokog kontrasta, korisničko sučelje mijenja boju pozadine u tamnu, a boju slova u svijetlu boju. Uzimajući u obzir osobe s astigmatizmom, nijanse tamnih i svijetlih boja nisu u potpunosti crne i bijele, nego kombinacije sivih i bijelih nijansi. Takvim kombinacijama nijansi ne dolazi do efekta svjetlosne mrlje te se korisnicima ne umaraju oči od prevelike svjetline. Na slici 6.33 prikazano je korisničko sučelje zaposlenika nakon uključene prilagodbe visokog kontrasta.



Sl. 6.33. *Prikaz korisničkog sučelja s uključenim visokim kontrastom*

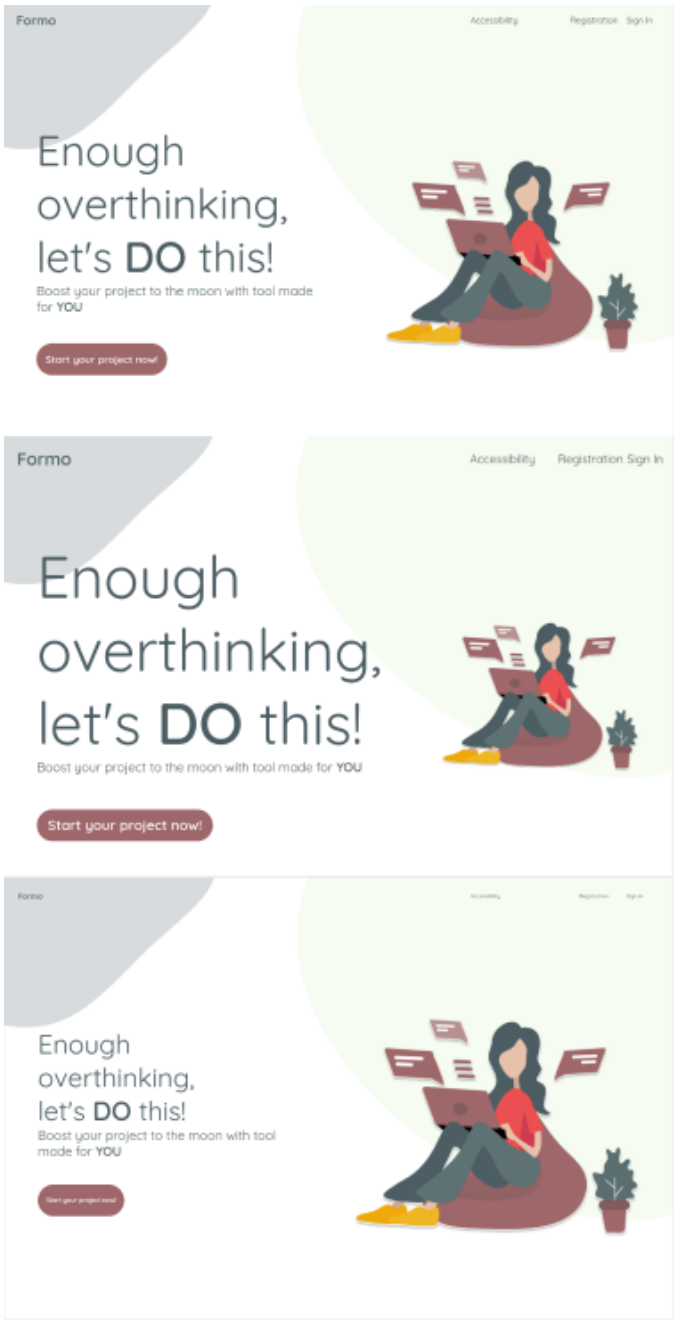
Kako bi se postigao visoki kontrast, boja pozadina je mogla ostati bijela, ali uzimajući u obzir osobe osjetljive na svjetlinu, s tamnom pozadinom se oči manje umaraju jer svjetlina nije jaka, a samim time je olakšano korištenje programskog rješenja. Na primjeru je vidljiva promjena navigacijske trake koja ima bijelu pozadinu sa sivim slovima te gradijentom iz sive u bijelu za označene stavke. Također, popis zaposlenika se nalazi na sivoj pozadini s bijelim slovima te bijelim ikonama, gumbovima i izbornicima. Kako bi bili sigurni da visoki kontrast zadovoljava uvjete WCAG smjernica, alatom za testiranje pristupačnosti *ANDI*, testiran je visoki kontrast. Na slici 6.34 vidljivi su rezultati testiranja korisničkog sučelja osobnih postavki korisnika. Prikazom rezultata označene stavke na slici, vidljivo je da je kontrast između boje slova i boje pozadina izuzetno visok s omjerom 9.74:1 što zadovoljava najvišu WCAG razinu pristupačnosti. S obzirom da programsko rješenje na svim korisničkim sučeljima koristi iste nijanse za pozadinu i tekst, može se zaključiti da programsko rješenje u cjelini zadovoljava najviši WCAG nivo pristupačnosti u smislu visokog kontrasta.



Sl. 6.34. *Prikaz rezultata testiranja visokog kontrasta ANDI alatom*

6.4.4. Prilagodba veličine fonta korisničkog sučelja

Ukoliko korisnici nemaju oštećenja vida poput disleksije, daltonizma i slično, nego imaju poteškoće s čitanjem prevelikih ili premalih slovima, u postavkama pristupačnosti moguće je prilagoditi veličinu fonta. Pritiskom na gumb za smanjenje ili povećanje slova, kroz cijelo programsko rješenje mijenja se zadana veličina slova. Zadana veličina slova, bez promjene, je 16px. Svakim pritiskom na gumb za povećanje ili smanjenje zadana veličina se poveća ili smanji za 2px. Budući da korisničko sučelje sadrži tekstove različitih veličina, prilikom implementiranja dizajna pojedinog korisničkog sučelja i teksta, kao mjerne jedinice korištene su em jedinice. Em jedinice predstavljaju responzivne jedinice jednake zadanoj px jedinici. Tako će, na primjer, 1em biti jednak 16px, ukoliko je 16px zadana jedinica programskog rješenja. Na taj način, prilikom promjene veličine fonta, potrebno je na jednom mjestu promijeniti zadanu jedinicu, umjesto na svim mjestima gdje se postavlja određena veličina pojedinog teksta. Slika 6.35 prikazuje početnu stranicu sa zadanom veličinom fonta od 16px, zatim početnu stranicu s povećanim fontom i zadanom veličinom od 22px te početnu stranicu sa smanjenim fontom i zadanom veličinom od 10px.



Sl. 6.35. *Prikaz početne stranice s različitim veličinama fonta*

6.5. Testiranje jedinica koda

Prilikom testiranja jedinica kôda korišten je okvir za testiranje *Jest* i programska biblioteka *Enzyme*. Nakon instalacije potrebnih paketa, kreirane su konfiguracijske datoteke za testiranje – *jest.config.json*, koji je prikazan na slici 6.36 i datoteka *setupTests.js*

Sl. 6.36. *Primjer konfiguracijske datoteke okvira Jest*

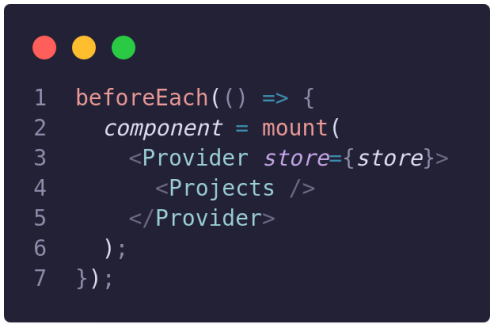
Prilikom postavljanja konfiguracijske datoteke okvira *Jest* potrebno je navesti putanju do datoteke s postavkama testiranja, navesti oblik putanje testova, je li potrebno prikupljati pokrivenost testovima i slično.

Nakon završenih konfiguracija i postavki moguće je započeti s pisanjem testova jedinica kôda. Prilikom testiranja jedinica kôda testira se prikaz pojedine komponente, jesu li dobiveni i prikazani ispravni podaci te pojedine funkcije unutar komponenti.

Kako bi se raspoznavali testni slučajevi pojedinih komponenti i *Redux* datoteka, potrebno ih je podijeliti u testne pakete (eng. *test suite*). Svaka komponenta i *Redux* datoteka posjeduje vlastitu testnu datoteku. Kako bi se definirao testni paket, koristi se funkcija *describe* okvira *Jest* kojoj se predaje naziv testnog paketa poput *Homepage tests* te povratna funkcija koja može sadržavati dodatne testne pakete i testne slučajeve. Na slici 6.37 prikazan je primjer testnog paketa. Prilikom testiranja moguće je kreirati funkcije i naredbe koje je potrebno izvršiti prije testnih slučajeva pomoću funkcija *beforeEach* i *beforeAll*. Najčešći slučaj korištenja funkcije *beforeEach* je definiranje prikaza komponente, koji je vidljiv na slici 6.38.

```
describe('NotificationModal', () => {
  let component;

  afterEach(() => {
    useSelector.mockClear();
  });

  beforeEach(() => {
    useSelector.mockImplementation((callback) =>
      callback({
        ...initialState,
      })
    );
    component = mount(
      <Provider store={store}>
        <NotificationModal />
      </Provider>
    );
  });

  it('renders correctly', () => {
    expect(component.html()).toMatchSnapshot();
  });
});
```

Sl. 6.37. *Primjer testnog paketa*



```
beforeEach(() => {
  component = mount(
    <Provider store={store}>
      <Projects />
    </Provider>
  );
});
```

Sl. 6.38. *Primjer funkcije beforeEach*

Prilikom testiranja komponenti korisničkog sučelja, potrebno je kreirati testni slučaj prikaza komponente snimkom. Slika 6.39 prikazuje testni slučaj prikaza početne stranice. Unutar funkcije *beforeEach* određen je način prikaza korištenjem funkcije *mount* biblioteke *Enzyme*. Početna stranica se prikazuje pomoću funkcije *mount* kako bi se u isto vrijeme prikazale i komponente djeca te na taj način dobio što točniji test. Kreiranjem testnog slučaja *renders correctly*, prikazana komponenta pretvara se u *HTML* kôd unutar funkcije *expect* i pomoću podudarača *toMatchSnapshot* provjerava podudara li se sa snimkom zaslona.

```
it('renders correctly', () => {
  expect(component.html()).toMatchSnapshot();
});
```

Sl. 6.39. *Primjer testnog slučaja snimke*

Pomoću funkcije *expect* unutar testnih slučajeva provjeravaju se uvjeti prolaska testova pomoću raznih podudarača (eng. *matchers*) koje funkcija pruža. Jedan od najčešćih podudarača, uz *toMatchSnapshot* je *toEqual*. Korištenjem podudarača *toEqual* moguće je provjeravati jesu li dobiveni ispravni izlazi funkcija poput primjera na slici 6.40.

```
it('should set font size', () => {
  const expectedState = {
    ...initState,
    fontSize: 2,
  };
  const action = {
    type: actionTypes.SET_FONT_SIZE,
    payload: 2,
  };
  const actualState = accessibility(initState, action);

  expect(actualState).toEqual(expectedState);
});
```

Sl. 6.40. *Primjer podudarača toEqual*

Ukoliko je potrebno testiranje je li neka od funkcija pozvana nakon određenog uvjeta, potrebno je koristiti podudarače *toBeCalled* ili *toHaveBeenCalled*. Također, moguće je testirati i koliko puta je funkcija pozvana pomoću podudarača *toBeCalledTimes* i *toHaveBeenCalledTimes*. Slika 6.41 prikazuje testni slučaj za provjeru je li pozvana određena funkcija nakon pritiska gumba

```
it('should call closeForm function on button click', () => {
  component.find('.forms-container__close-btn').simulate('click');
  expect(instance.props.closeForm).toHaveBeenCalled();
});
```

Sl. 6.41. *Primjer korištenja podudarača toHaveBeenCalled*

Kako bi se testirao poziv funkcije nakon pritiska gumba, potrebno je unutar prikaza komponente pronaći željeni gumb putem klasnog naziva te simulirati pritisak pozivom funkcije *simulate*. Nakon simulacije pritiska gumba, provjerava se je li pozvana funkcija *closeForm* koja je komponenti predana putem svojstava te se ulančava podudarač *toHaveBeenCalled*, koji kao rezultat vraća *true* ili *false*, odnosno potvrdu ili negaciju poziva funkcije.

Prilikom testiranja *Redux* akcija potrebno je koristiti podudarač *toEqual*. Testiranjem *Redux* akcija provjerava se podudara li se očekivana akcija s pozvanim kreatorom akcije. Primjer testiranja *Redux* akcije prikazan je na slici 6.42. Unutar očekivane akcije postavlja se vrsta akcije koju je potrebno testirati te eventualni dodatni podaci koji se trebaju predati kreatoru akcije. Nakon toga unutar funkcije *expect* poziva se željeni kreator akcije te se ulančava podudarač *toEqual* s očekivanom akcijom.

```
it('when action type is SET_DYSLEXIA it should dispatch setDyslexia action creator', () => {
  const payload = {
    ...accessibility,
    dyslexia: true,
  };
  const expectedAction = {
    type: actionTypes.SET_DYSLEXIA,
    payload,
  };
  expect(actions.setDyslexia(payload)).toEqual(expectedAction);
});
```

Sl. 6.41. *Primjer testiranja Redux akcije*

Na sličan način kao *Redux* akcije, testiraju se i *Redux* reduktori. Na slici 6.42 vidljiv je testni slučaj *Redux* reduktora za vrstu akcije *ALL_TASKS_FETCHED*. Unutar testnog slučaja definira se očekivano stanje reduktora zadataka te se isti poziva s inicijalnim stanjem i željenim kreatorom akcije. Funkciji e*xpect* predaje se stvarno stanje, a ulančavanjem podudarača *toEqual* provjerava jesu li očekivano i stvarno stanje jednaki.

Nakon napisanih testnih paketa i testnih slučajeva, pomoću okvira *Jest* potrebno je pokrenuti testove putem terminala. Kako bi se olakšalo pokretanje testova, unutar datoteke *package.json*, moguće je kreirati skriptu s naredbom za pokretanje. Nakon pokretanja, testovi su rezultirali padom dva testna paketa, odnosno tri testna slučaja. Prvi testni paket koji nije prošao testove je paket pod nazivom *organisation reducer*, odnosno reduktor organizacije. Unutar dobivenog izvješća, prikazanog na slici 6.43, vidljivo je da se očekivani ID organizacije ne podudara sa stvarnim stanjem. Detaljnim pregledom kôda i rezultata, pronađena je greška u načinu spremanja

ID-a organizacije. Način spremanja rezultirao je duplim spremanjem ID-a pod različitim nazivima – „id“ i „_id“. Nakon promjene kôda, oba testna slučaja organizacijskog reduktora su zadovoljila uvjete prolaska testiranja.

```
it('should fetch all tasks', () => {
  const expectedState = {
    ...initState,
    currentProjectTasks: tasksMock,
  };
  const action = {
    type: actionTypes.ALL_TASKS_FETCHED,
    payload: tasksMock,
  };
  const actualState = tasks(initState, action);

  expect(actualState).toEqual(expectedState);
});
```

Sl. 6.42. *Primjer testiranja Redux reduktora*

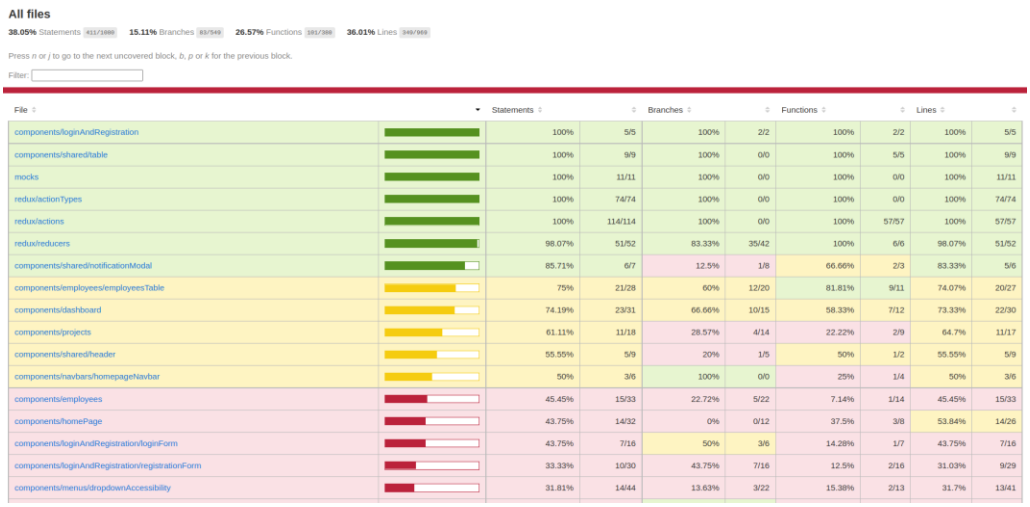
```
FAIL src/redux/reducers/tests/organisationReducer.test.js
● organisation reducer › should set updated organisation info

  expect(received).toEqual(expected) // deep equality

  - Expected  - 0
  + Received  + 1

  @@ -2,10 +2,11 @@
      "_id": "123",
      "address": "address",
      "city": "city",
      "country": "country",
      "createdAt": "2022-04-01",
  +   "id": "123",
      "members": Array [],
      "name": "name",
      "projects": Array [],
      "updatedAt": Array [],
    }

    45 |               const actualState = organisation(initState, action);
    46 |
  > 47 |               expect(actualState).toEqual(expectedState);
       |                                   ^
    48 |       });
    49 |
    50 |       it('should fetch organisation info', () => {

    at Object.<anonymous> (src/redux/reducers/tests/organisationReducer.test.js:47:23)

● organisation reducer › should fetch organisation info

  expect(received).toEqual(expected) // deep equality

  - Expected  - 0
  + Received  + 1

  @@ -2,10 +2,11 @@
      "_id": "123",
      "address": "address",
      "city": "city",
      "country": "country",
      "createdAt": "2022-04-01",
  +   "id": "123",
      "members": Array [],
      "name": "name",
      "projects": Array [],
      "updatedAt": Array [],
    }

    76 |               const actualState = organisation(initState, action);
    77 |
  > 78 |               expect(actualState).toEqual(expectedState);
       |                                   ^
    79 |       });
    80 |
    81 |       it('should set default state', () => {

    at Object.<anonymous> (src/redux/reducers/tests/organisationReducer.test.js:78:23)
```

Sl. 6.43. *Izvješće testnog paketa organisation reducer*

Drugi testni paket koji nije uspješno prošao testiranje je testni paket *Projects*, odnosno testni paket za testiranje komponente projekata. Pregledom izvješća, prikazanog na slici 6.44, vidljivo je da za predano nepostojeće polje projekata, komponenta vraća grešku i ne može se prikazati. Dodavanjem provjere postoji li polje projekata prije mapiranja svakog projekta, uvjet testnog slučaja je zadovoljen i test uspješno prolazi testiranje.

Sl. 6.44. *Izvješće testnog paketa Projects*

Nakon što su svi testni slučajevi uspješno prošli testiranje, potrebno je pokrenuti provjeru pokrivenosti kôda testovima putem terminala. Završetkom provjere pokrivenosti kôda generira se izvješće *Istanbul* koje je moguće detaljno pregledati unutar internetskog preglednika. Na slici 6.45 prikazan je primjer izvješća *Istanbul*. Pokretanjem izvješća vidljiv je ukupan postotak pokrivenosti testovima te pokrivenost svake datoteke. Izvješće pruža uvid i u pokrivenost različitim vrstama testova poput pokrivenosti linija koda, grananja koda, funkcija i slično.



Sl. 6.45. *Primjer Istanbul izvješća pokrivenosti koda testovima*

Uvidom u izvješće moguće je provjeriti dijelove koji nisu pokriveni testovima te na taj način dodati nove testne pakete i testne slučajeve kako bismo poboljšali pokrivenost. Slika 6.46 prikazuje

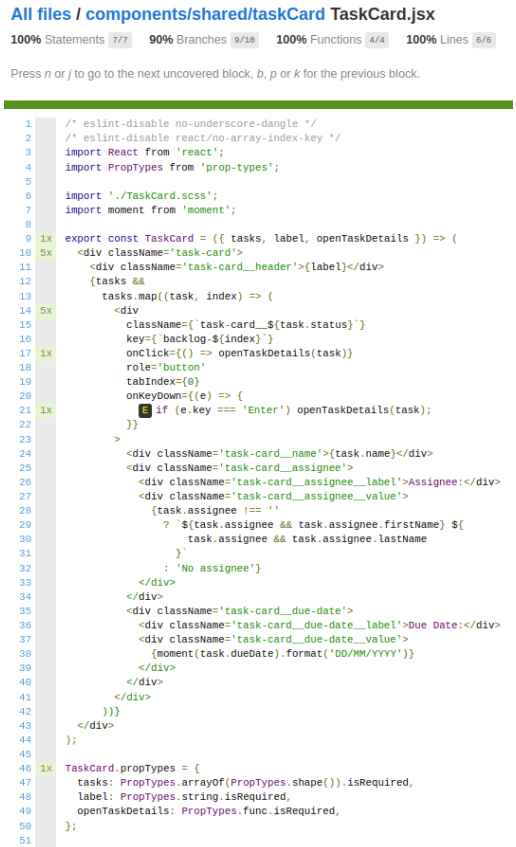
primjer izvješća pokrivenosti testovima komponente *TaskCard*. Što je postotak pokrivenosti testovima veći, veća je mogućnost otkrivanja ranih grešaka sustava te grešaka koje je nemoguće otkriti ručnim testiranjem. Detaljnim pregledom izvješća komponente *TaskCard* kreiran je novi testni paket naziva *TaskCard* te testni slučajevi za ispravan prikaz komponente i testiranje poziva funkcija pritiskom na određene gumbove.

Sl. 6.46. *Primjer izvješća pokrivenosti testovima za komponentu TaskCard*

Nakon napisanog testnog paketa naziva *TaskCard*, prikazanog na slici 6.47 i testnih slučajeva, ispravljenih pronađenih pogrešaka te ponovno generiranog izvješća pokrivenosti kôda testovima, novo stanje pokrivenosti za komponentu *TaskCard* vidljivo je na slici 6.48.

```
describe('TaskCard', () => {
  let component;
  let instance;

  beforeEach(() => {
    component = mount(
      <Provider store={store}>
        <TaskCard {...props} />
      </Provider>
    );
    instance = component.find(TaskCard).children().instance();
  });

  it('renders correctly', () => {
    expect(component.html()).toMatchSnapshot();
  });

  it('should call openTaskDetails on card click', () => {
    component.find('.task-card__backlog').simulate('click');
    expect(instance.props.openTaskDetails).toHaveBeenCalled();
  });

  it('should call openTaskDetails on enter pres', () => {
    component.find('.task-card__backlog').simulate('keyDown', { key: 'Enter' });
    expect(instance.props.openTaskDetails).toHaveBeenCalled();
  });

  it('should render correctly when there is no assignee', () => {
    const newProps = {
      ...props,
      tasks: tasksMockNoAssignee.currentProjectTasks,
    };

    component = mount(
      <Provider store={store}>
        <TaskCard {...newProps} />
      </Provider>
    );

    expect(component.html()).toMatchSnapshot();
  });
});
```

Sl. 6.47. *Prikaz testnog paketa TaskCard*



All files / components/shared/taskCard TaskCard.jsx

100% Statements 7/7 90% Branches 9/10 100% Functions 4/4 100% Lines 6/6

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

```
/* eslint-disable no-underscore-dangle */
/* eslint-disable react/no-array-index-key */
import React from 'react';
import PropTypes from 'prop-types';

import './TaskCard.scss';
import moment from 'moment';

export const TaskCard = ({ tasks, label, openTaskDetails }) => (
  <div className='task-card'>
    <div className='task-card__header'>{label}</div>
    {tasks &&
      tasks.map((task, index) => (
        <div
          className={`task-card__${task.status}`}
          key={`backlog-${index}`}
          onClick={() => openTaskDetails(task)}
          role='button'
          tabIndex={0}
          onKeyDown={(e) => {
            if (e.key === 'Enter') openTaskDetails(task);
          }}
        >
          <div className='task-card__name'>{task.name}</div>
          <div className='task-card__assignee'>
            <div className='task-card__assignee__label'>Assignee:</div>
            <div className='task-card__assignee__value'>
              {task.assignee !== ''
                ? `${task.assignee && task.assignee.firstName} ${
                    task.assignee && task.assignee.lastName
                  }`
                : 'No assignee'}
            </div>
          </div>
          <div className='task-card__due-date'>
            <div className='task-card__due-date__label'>Due Date:</div>
            <div className='task-card__due-date__value'>
              {moment(task.dueDate).format('DD/MM/YYYY')}
            </div>
          </div>
        </div>
      ))}
  </div>
);

TaskCard.propTypes = {
  tasks: PropTypes.arrayOf(PropTypes.shape()).isRequired,
  label: PropTypes.string.isRequired,
  openTaskDetails: PropTypes.func.isRequired,
};
```

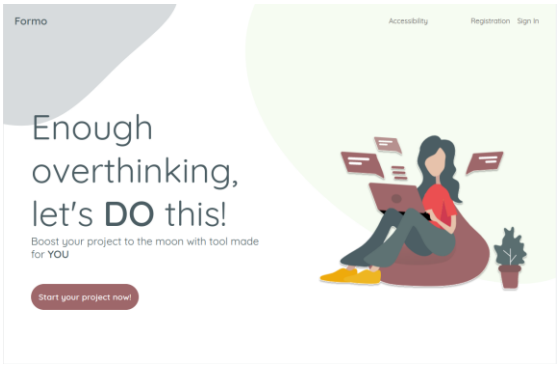
Sl. 6.48. *Prikaz novog izvješća TaskCard.jsx komponente*

7. OPIS RADA PROGRAMSKOG RJEŠENJA S ISPITIVANJEM I ANALIZOM

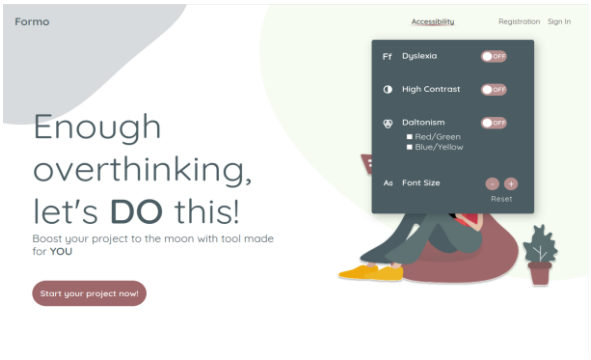
Nakon završetka razvoja i testiranja programskog rješenja potrebno je korisnike upoznati s gotovim proizvodom. Kako bi korisnici uspješno koristili sustav, u ovom poglavlju bit će opisano kako na ispravan način koristiti isti te sve njegove funkcionalnosti.

7.1. Korištenje programskog rješenja

Pokretanjem web aplikacije korisniku se prikazuje početna stranica prikazana na slici 7.1. Na vrhu početne stranice nalazi se navigacijska traka putem koje korisnik može odabrati postavke pristupačnosti, registraciju novog računa ili prijavu u postojeći korisnički račun. Odabirom postavki pristupačnosti otvara se izbornik, vidljiv na slici 7.2, s opcijama uključivanja pristupačnosti za disleksiju, daltonizam, visoki kontrast te povećanje ili smanjenje veličine fonta.



Sl. 7.1. *Početna stranica*

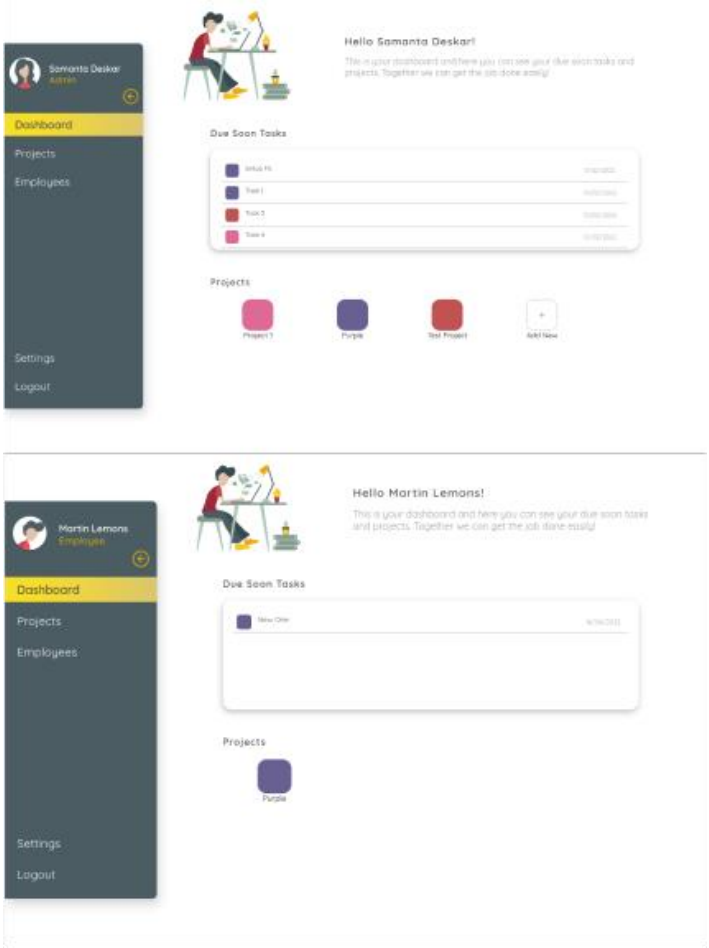


Sl 7.2. *Prikaz postavki pristupačnosti početne stranice*

Odabirom registracije korisničkog računa putem navigacijske trake ili pritiskom na gumb *Start your project now* prikazuje se forma za unos podataka kako bi se omogućilo kreiranje novog korisničkog računa i organizacije. Forma za registraciju vidljiva je na slici 6.3. Kreiranjem

korisničkog računa, korisnik postaje administrator unutar kreirane organizacije. Ukoliko korisnik posjeduje korisnički račun, pritiskom na tekst *Already registered? Sign in here!*, unutar forme za registraciju, otvara se nova forma za prijavu postojećeg korisnika. Formu za prijavu korisnika, vidljivu na slici 6.7, moguće je otvoriti i pritiskom na gumb *Sign In* na navigacijskoj traci.

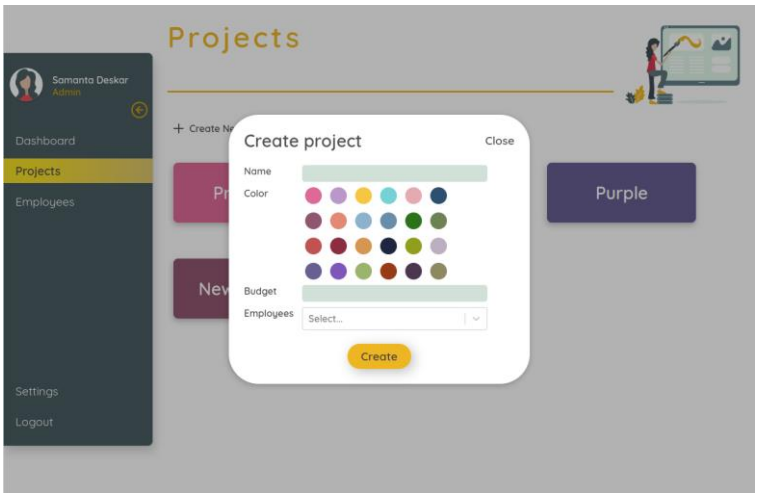
Nakon prijave korisnika, ovisno o ulozi koju korisnik posjeduje, prikazivati će se različita korisnička sučelja. Prijavljenom korisniku prikazuje se nova navigacijska traka s lijeve strane te kontrolna ploča, prikazana na slici 7.3. Korisnik na kontrolnoj ploči vidi zadatke koji su mu dodijeljeni te projekte na kojima sudjeluje. Ukoliko je korisnik administrator ili projektni menadžer, tada se uz projekte prikazuje i gumb za kreiranje novog projekta. Pritiskom na pojedini zadatak, prikazuju se detalji zadatka. Pritiskom na pojedini projekt korisnik se preusmjerava na sučelje odabranog projekta.



Sl. 7.3. *Prikaz kontrolne ploče administratora i zaposlenika*

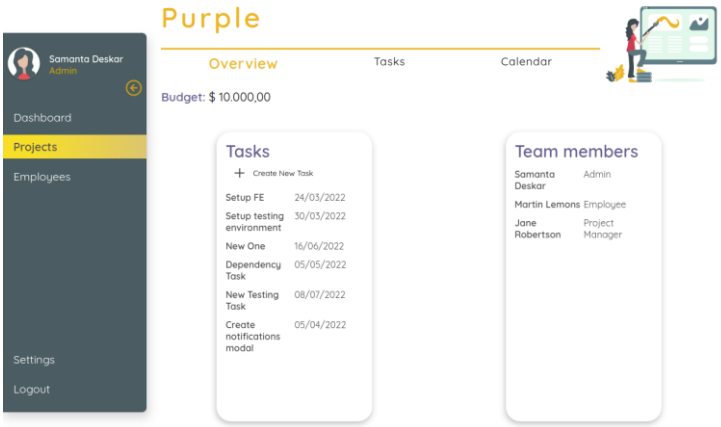
Pritiskom na gumb *Projects*, unutar navigacijske trake, korisniku se prikazuje sučelje s popisom projekata te gumbom za kreiranje novog projekta ukoliko je korisnik administrator ili projektni

menadžer. Pritiskom na gumb *Create new project* prikazuje se forma za kreiranje novog projekta, vidljiva na slici 7.4. Prilikom kreiranja novog projekta potrebno je unijeti naziv projekta, odabrati boju koja će predstavljati projekt, zadani budžet te odabrati zaposlenike koji će sudjelovati na projektu. Pritiskom na gumb *Create* kreira se novi projekt s unesenim podacima.



Sl. 7.4. *Prikaz forme za kreiranje projekta*

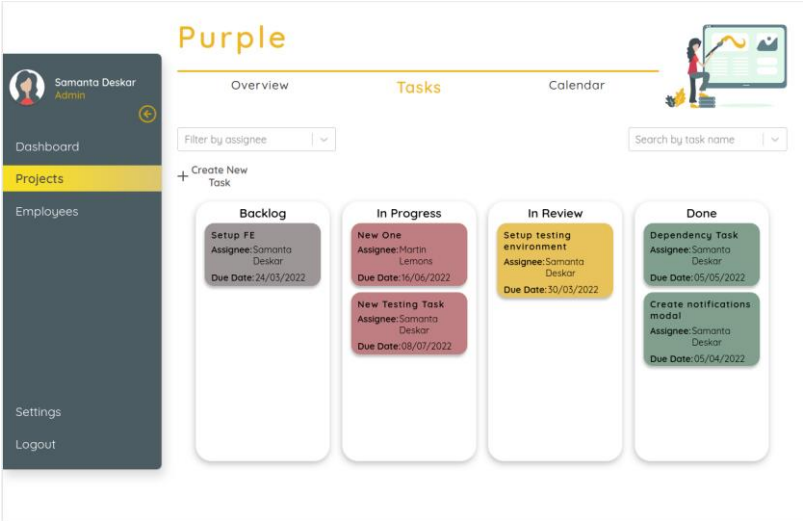
Ukoliko korisnik pritisne na postojeći projekt, prikazuje se sučelje odabranog projekta, prikazano na slici 7.5. Korisnik prvo vidi općeniti pregled projekta poput budžeta, popisa zadataka te popisa zaposlenika. Pomoću nove navigacijske trake, koja se nalazi ispod naslova *Projects*, korisnik može pregledavati zadatke pomoću ploče *Kanban* ili pomoću gantograma.



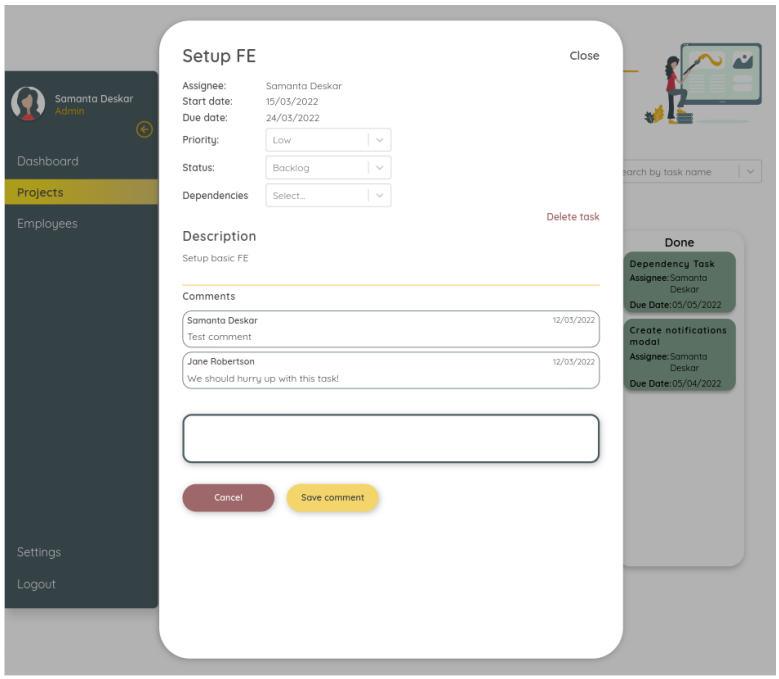
Sl. 7.5. *Prikaz sučelja odabranog projekta*

Pritiskom na gumb *Tasks* na gornjoj navigacijskoj traci, korisniku se prikazuju zadaci na ploči *Kanban*, što je vidljivo na slici 7.6. Pritiskom na određeni zadatak, prikazuju se detalji odabranog

zadatka, prikazani na slici 7.7. Unutar prikaza detalja, korisnik može mijenjati prioritet, status i ovisnosti o drugim zadacima. Također, korisnik može čitati komentare drugih korisnika na zadatku te može kreirati vlastiti komentar pritiskom na gumb *Add comment*. Nakon pritiska na gumb *Add comment* korisnik može unijeti proizvoljan tekst unutar tekstualnog okvira te pritiskom na gumb *Save comment* spremiti komentar ili obrisati pritiskom na gumb *Cancel*.

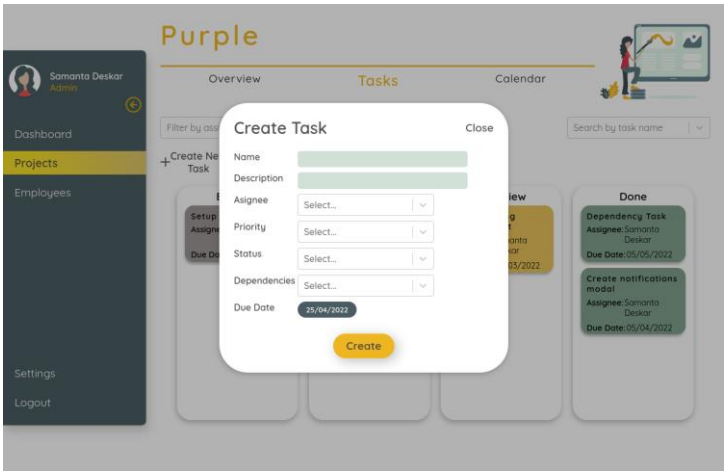


Sl. 7.6. *Prikaz zadataka pomoću ploče Kanban*



Sl. 7.7. *Prikaz detalja zadatka*

Pritiskom na gumb *Create new task*, unutar općenitog pregleda projekta ili pregleda zadataka na ploči *Kanban*, korisniku se prikazuje forma za kreiranje novog zadatka, prikazana na slici 7.8. Prilikom kreiranja zadatka korisnik treba unijeti naziv zadatka, opis, odabrati osobu koja će raditi na zadatku, odabrati prioritet, status, ovisnosti o drugim zadacima te krajnji rok za završetak zadatka. Pritiskom na gumb *Create* kreira se novi zadatak s unesenim podacima.

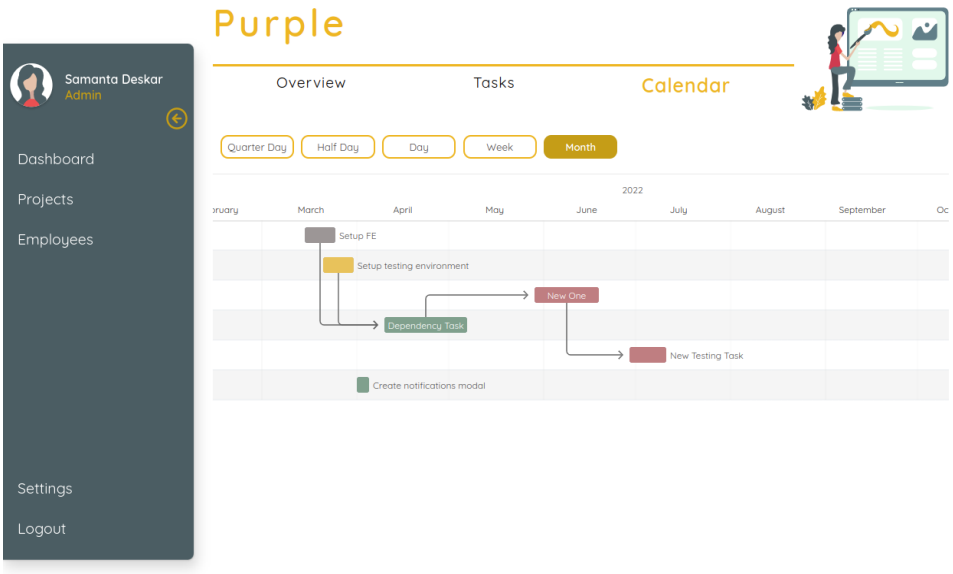


Sl. 7.8. *Prikaz forme za kreiranje zadatka*

Odabirom stavke *Calendar* unutar gornje navigacijske trake, korisniku se prikazuje sučelje zadataka raspoređenih na gantogramu, vidljivo na slici 7.9. Unutar prikazanog sučelja, korisnik može odabrati različite načine prikaza - mjesečni, tjedni, dnevni, poludnevni te po četvrtini dana, pritiskom na određene gumbove. Pritiskom na pojedini zadatak, korisnik ponovno može vidjeti detalje zadatka. Povlačenjem lijevog ili desnog kraja zadatka te pritiskom, držanjem i pomicanjem zadatka, moguće je promijeniti početni i krajnji datum završetka zadatka.

Odabirom stavke *Employees* unutar lijeve navigacijske trake, korisniku se prikazuje sučelje s popisom zaposlenika unutar organizacije. Korisnici se mogu filtrirati prema projektima na kojima sudjeluju, a ukoliko je trenutni korisnik administrator, tada ima mogućnost kreiranja novih zaposlenika ili brisanja postojećih. Na slici 7.10 vidljiva je razlika u prikazu korisničkog sučelja zaposlenika ovisno je li trenutni korisnik zaposlenik ili administrator. Pritiskom na gumb *Add employee*, korisniku se prikazuje forma za kreiranje novog zaposlenika, vidljiva na slici 7.11. Prilikom kreiranja zaposlenika potrebno je unijeti ime i prezime zaposlenika, adresu e-pošte, spol i ulogu. Pritiskom na gumb *Create* kreira se novi zaposlenik s unesenim podacima. Nakon uspješnog kreiranja, administrator dobiva poruku s pristupnim podacima novog zaposlenika koje

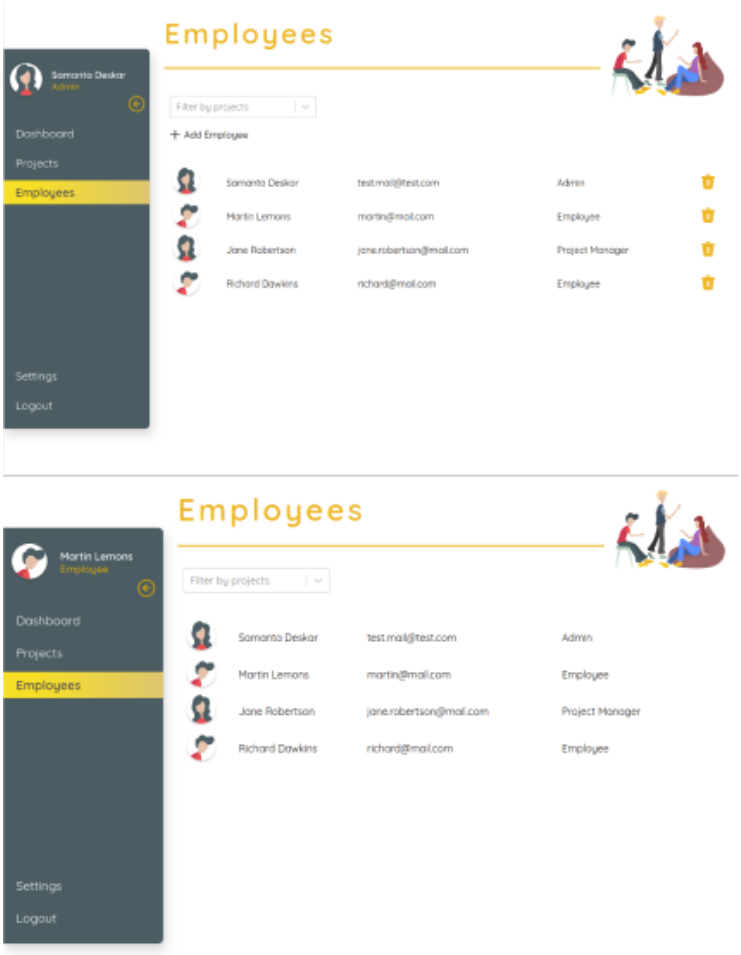
mu treba proslijediti uz napomenu da je preporučljivo promijeniti automatski generiranu lozinku novog zaposlenika.



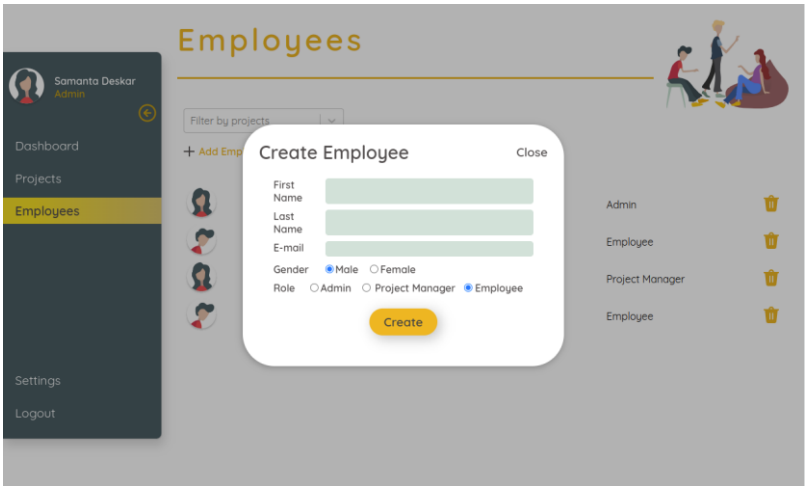
Sl. 7.9. *Prikaz zadataka pomoću gantograma*

Odabirom stavke *Settings* unutar lijeve navigacijske trake korisniku se prikazuju postavke sustava. Postavke sustava sadrže gornju navigacijsku traku s različitim vrstama postavki. Ukoliko je korisnik administrator, prikazuje mu se navigacijska traka s osobnim postavkama, postavkama organizacije te postavkama pristupačnosti. U suprotnome, korisniku se prikazuje navigacijska traka s osobnim postavkama i postavkama pristupačnosti. Na slikama 7.12 i 7.13 vidljivi su različiti prikazi postavki, ovisno je li trenutni korisnik administrator ili zaposlenik, odnosno projektni menadžer.

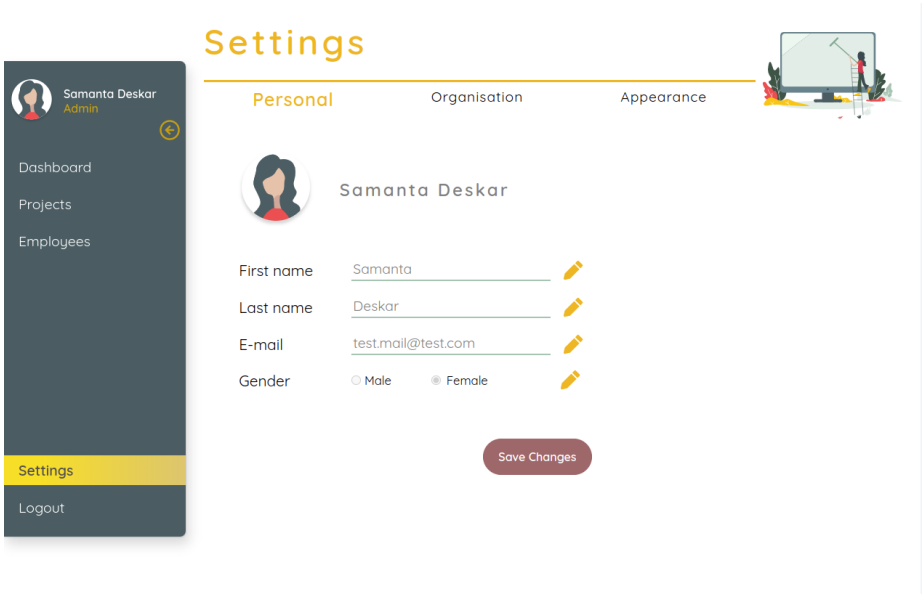
Odabirom osobnih postavki, korisniku se prikazuju njegovi osobni podaci koje može mijenjati. Kako bi se osobni podaci promijenili potrebno je pritisnuti ikonu olovke. Pritiskom na ikonu olovke, omogućuje se izmjena odabrane stavke. Nakon što korisnik izmijeni odabrani podatak, pritiskom na ikonu kvačice omogućuje se gumb za spremanje podataka *Save changes*. Odabirom stavke *Organisation* unutar gornje navigacijske trake, korisniku se prikazuju podaci organizacije, vidljivi na slici 7.14. Promjena podataka organizacije odvija se na isti način kao i promjena osobnih podataka.



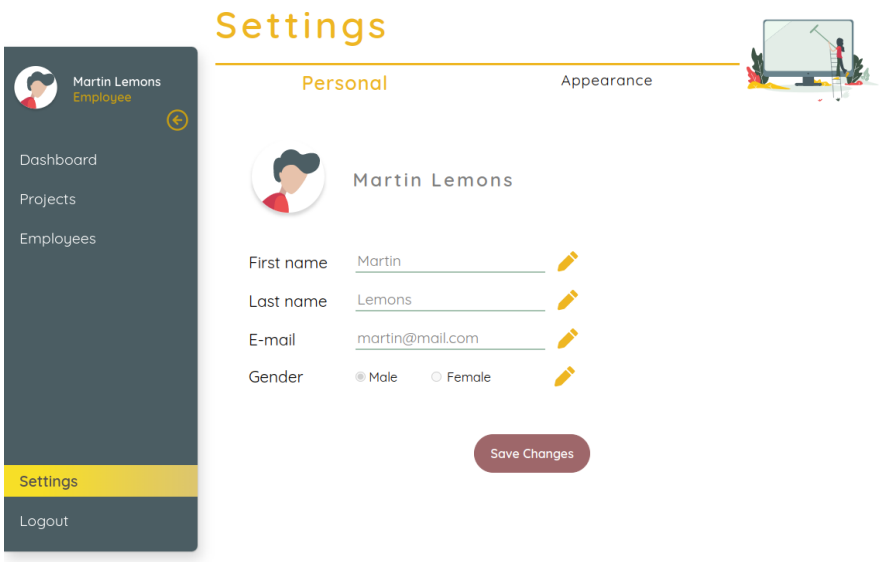
Sl. 7.10. *Prikaz sučelja zaposlenika kada je korisnik administrator i kada je korisnik zaposlenik*



Sl. 7.11. *Prikaz forme za kreiranje zaposlenika*



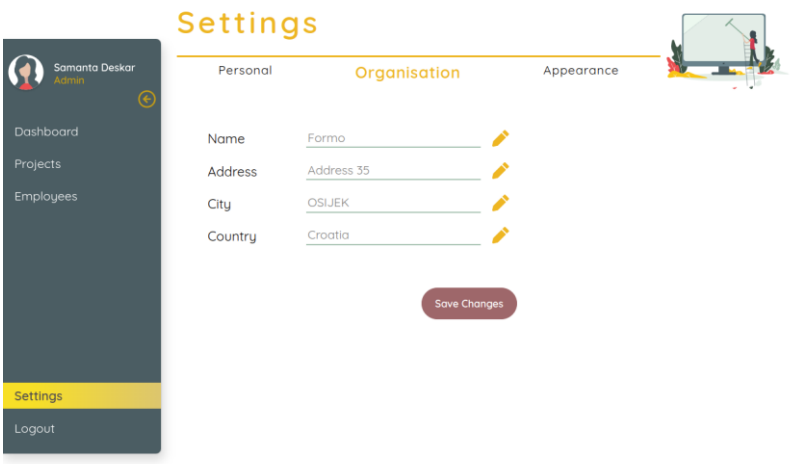
Sl. 7.12. *Prikaz postavki administratora*



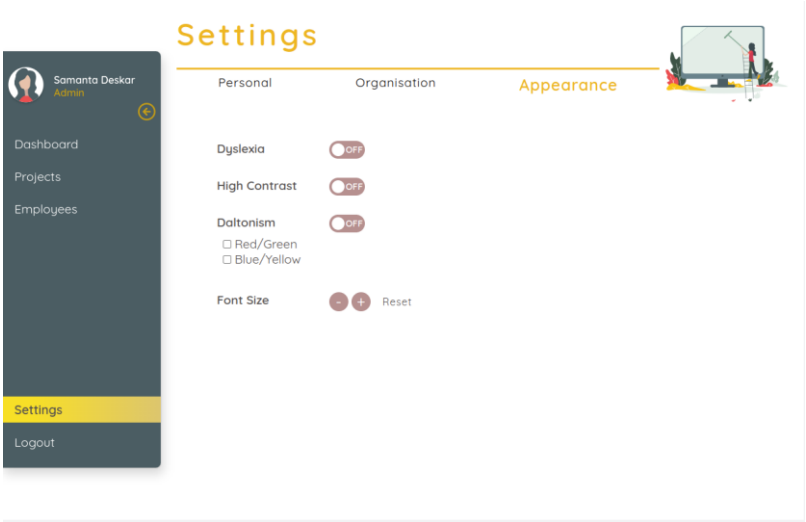
Sl. 7.13. *Prikaz postavki zaposlenika ili projektnog menadžera*

Odabirom stavke *Appearance* unutar gornje navigacijske trake, korisniku se prikazuje korisničko sučelje postavki pristupačnosti, prikazano na slici 7.15. Pritiskom na gumbove pored opisa postavki pristupačnosti, određena pristupačnost se uključuje ili isključuje. Ukoliko se želi uključiti pristupačnost za disleksiju, potrebno je pritisnuti gumb pored naziva *Dyslexia*. Kako bi se uključio način visokog kontrasta, potrebno je pritisnuti gumb pored naziva *High Contrast*. Ako korisnik boluje od daltonizma, potrebno je uključiti pristupačnost za daltonizam, pritiskom na gumb pored naziva *Daltonism* te je potrebno odabrati vrstu daltonizma - *Red/Green* za crveno-zeleno vrstu

daltonizma ili *Blue/Yellow* za plavo-žutu vrstu. Korisnik također ima mogućnost promjene veličine slova unutar sustava. Pored naziva *Font size* nalaze se tri gumba – gumb za smanjenje veličine slova s oznakom „-“, gumb za povećanje veličine slova s oznakom „+“ te gumb za ponovno postavljanje zadane veličine s nazivom *Reset*.



Sl. 7.14. *Prikaz postavki organizacije*



Sl. 7.15. *Prikaz postavki pristupačnosti*

Kako bi se korisnik odjavio iz sustava, potrebno je pritisnuti stavku *Logout* unutar lijeve navigacijske trake. Nakon odjave korisnika, ponovno se prikazuje početna stranica web aplikacije.

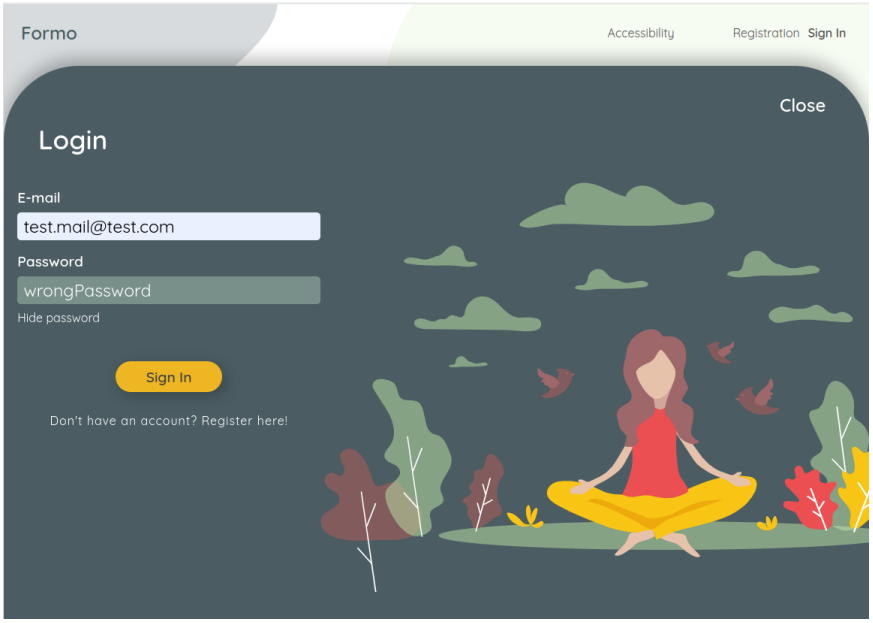
7.2. Ispitivanje rada programskog rješenja

Nakon završetka razvoja i testiranja programskog rješenja potrebno je provjeriti ispravnost rada programskog rješenja. Ispravnost rada provjerava se unosom realnih podataka i usporedbe izlaznih podataka s očekivanim ponašanjem. Kako bi sustav bio ispravan sustav potrebno je provjeriti neke od slučajeva korištenja od strane korisnika. Slučajevi koji će se provjeravati su:

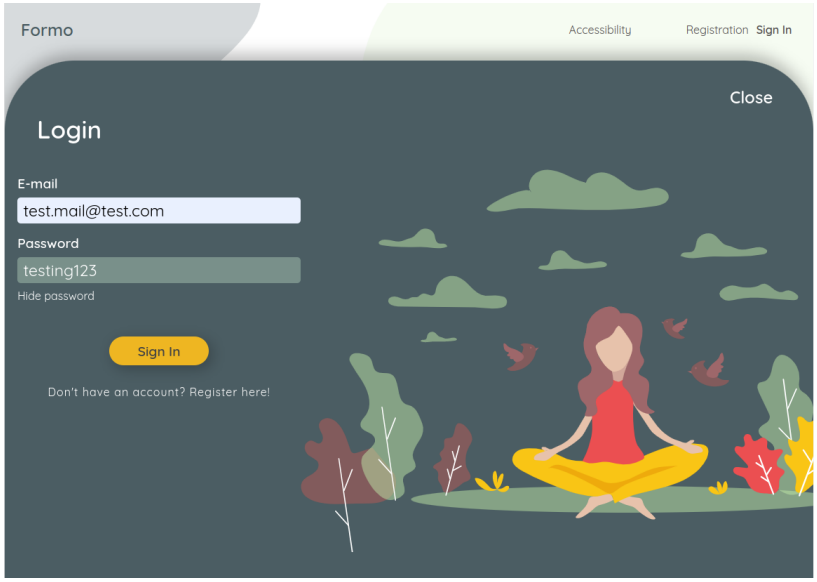
- prijava korisnika u sustav
- kreiranje i upravljanje novim projektom
- prilagodba sustava osobama s oštećenjima vida

7.2.1. Prvi slučaj ispitivanja rada programskog rješenja

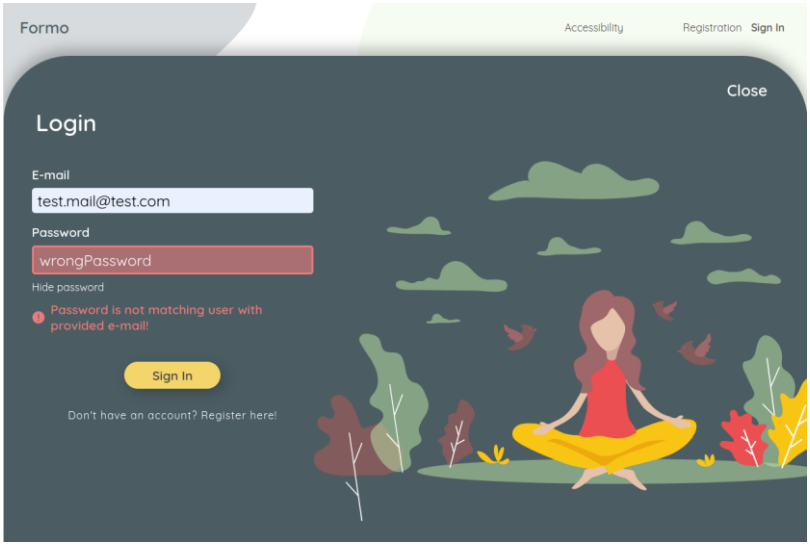
Da bi korisnik bio u mogućnosti koristiti sustav potrebna je prijava s pristupnim podacima. Podaci koji se unose prilikom prijave su adresa e-pošte i lozinka. Na slikama 7.16 i 7.17 prikazani su primjeri za unos točnih i netočnih pristupnih podataka. Ukoliko su uneseni neispravni podaci sustav bi trebao korisniku prikazati poruku o pogrešci, vidljivu na slici 7.18, poslanu od strane poslužitelja [1], u suprotnome, nakon uspješne prijave, korisnika se preusmjerava na korisničko sučelje kontrolne ploče.



Sl. 7.16. *Prvi primjer pristupnih podataka prilikom prijave u sustav*



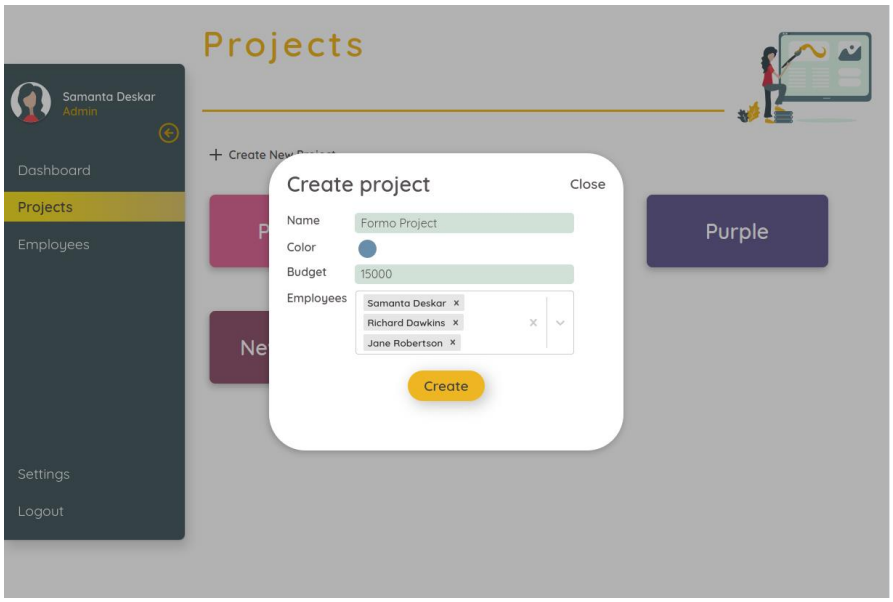
Sl. 7.17. *Drugi primjer pristupnih podataka prilikom prijave u sustav*



Sl. 7.18. *Primjer poruke o pogrešci prilikom prijave u sustav*

7.2.2. Drugi slučaj ispitivanja rada programskog rješenja

Ukoliko je korisnik prijavljen u sustav i ima ulogu administratora ili projektnog menadžera, tada može kreirati projekt pritiskom na gumb *Add new* na korisničkom sučelju kontrolne ploče ili pritiskom na gumb *Create new project* na korisničkom sučelju projekata. Nakon unosa ispravnih podataka, vidljivih na slici 7.19 i spremanja projekta, projekt se prikazuje na kontrolnoj ploči i unutar korisničkog sučelja projekata, što je vidljivo na slici 7.20. Pritiskom na gumb s nazivom novog projekta, prikazuje se korisničko sučelje s općenitim podacima novog projekta.



Sl. 7.19. *Primjer unosa podataka prilikom kreiranja projekta*

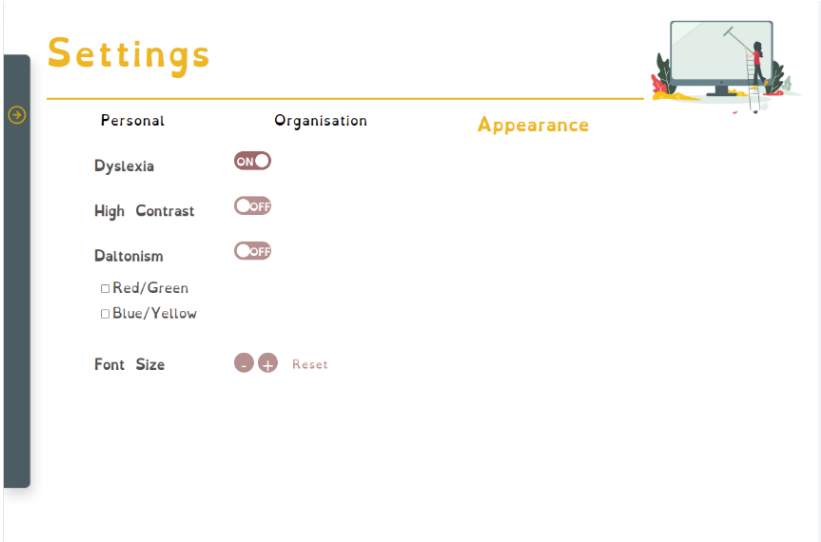
Odabirom stavke *Tasks* omogućen je pregled zadataka novog projekta pomoću ploče *Kanban*, stvaranje novih zadataka te prikaz i izmjena detalja zadataka. Odabirom stavke *Calendar* omogućen je prikaz zadataka pomoću gantograma te pregled i izmjena detalja zadataka.

Sl. 7.20. *Prikaz korisničkog sučelja nakon kreiranja novog projekta*

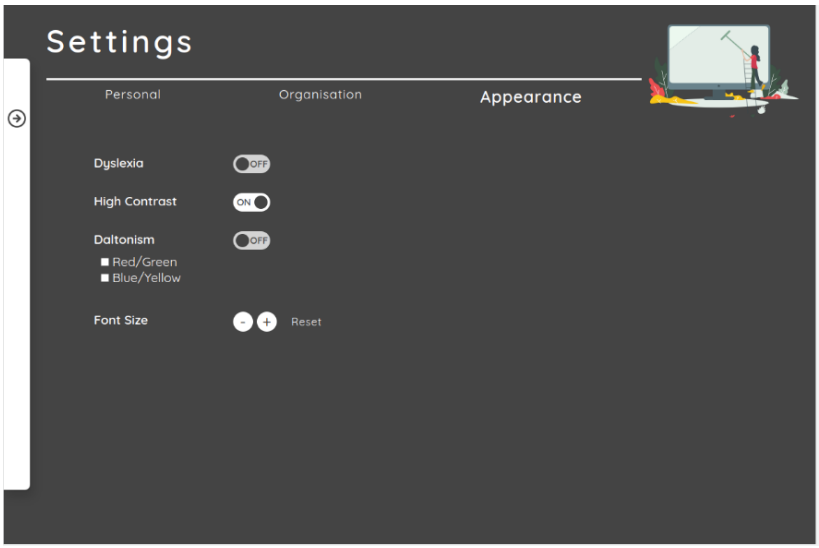
7.2.3. Treći slučaj ispitivanja rada programskog rješenja

Ukoliko je trenutni korisnik osoba s oštećenjima vida, potrebno je prilagoditi programsko rješenje kako bi ono bilo pristupačno korisniku. Nakon prijave korisnika u sustav, odabirom stavke *Settings* na navigacijskoj traci, prikazuje se korisničko sučelje postavki. Unutar korisničkog sučelja postavki, pritiskom na stavku *Appearance* prikazuje se korisničko sučelje s postavkama

pristupačnosti. Pritiskom na gumb u obliku sklopke pored naziva *Dyslexia,* mijenja se zadani font te se primjenjuje *OpenDyslexic* font unutar cijelog sustava, što je vidljivo na slici 7.21. Ponovnim pritiskom na gumb u obliku sklopke pored naziva *Dyslexia*, zadani font se vraća na početni *Quicksand* font te pritiskom na gumb u obliku sklopke pored naziva *High Contrast* primjenjuje se visoki kontrast, što je vidljivo na slici 7.22, unutar cijelog sustava.



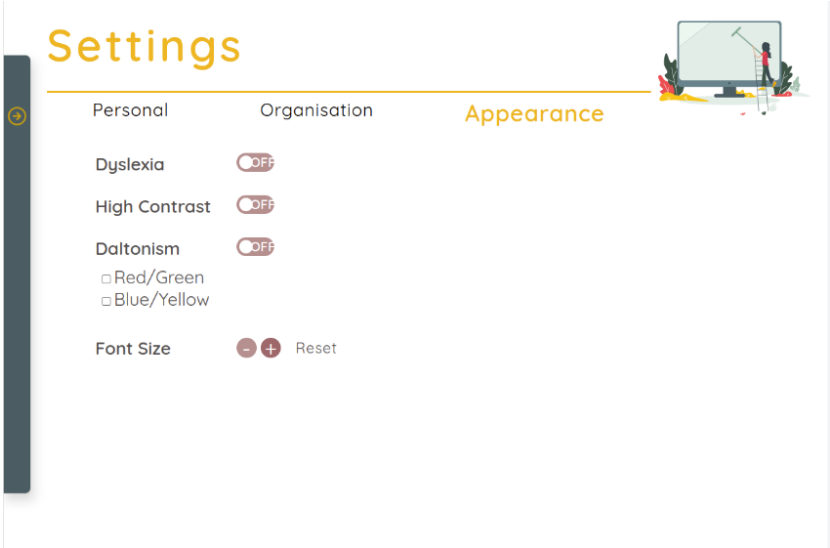
Sl. 7.21. *Prikaz korisničkog sučelja s uključenom postavkom pristupačnosti za disleksiju*



Sl. 7.22. *Prikaz korisničkog sučelja s uključenom postavkom pristupačnosti visokog kontrasta*

Gašenjem postavke visokog kontrasta, ponovnim pritiskom na isti gumb u obliku sklopke, korisničko sučelje se vraća na prvobitne postavke. Pritiskom tri puta na gumb s oznakom „+“ pored naziva *Font Size*, veličina slova znatno je povećana, što je vidljivo na slici 7.23, a pritiskom na

gumb *Reset*, veličina slova vraćena je na početnu zadanu veličinu. Pritiskom tri puta na gumb s oznakom „-“ pored naziva *Font Size* veličina slova znatno je umanjena, što je vidljivo na slici 7.24.



Sl. 7.23. *Prikaz korisničkog sučelja s povećanom veličinom fonta*

Sl. 7.24. *Prikaz korisničkog sučelja sa smanjenom veličinom fonta*

7.3. Rezultati ispitivanja s analizom

Prilikom ispitivanja rada web aplikacije, za određene unose očekivani su određeni izlazi. Izlazi mogu predstavljati poruke o pogreškama, poruke o uspješnosti izvršenja akcije ili sam prikaz podataka. Očekivani izlazi za određene ulaze navedeni su u tablici 7.1.

Tab. 7.1. *Prikaz očekivanih i stvarnih izlaznih podataka u ovisnosti o ulaznim podacima*

Naziv slučaja ispitivanja rada	Ulazni podaci	Očekivani izlazni podaci	Stvarni izlazni podaci
Prijava korisnika u sustav	1. Pritisak na gumb *Sign In* na početnoj stranici 2. Unos teksta „*test.mail@test.com*“ u polje za unos adrese e-pošte 3. Unos teksta „*wrongPassword“* u polje za unos lozinke 4. Pritisak na gumb *Sign In*	Korisniku se prikazuje poruka o pogrešci za krivo unesenu lozinku. Polje za unos lozinke mijenja boju u crvenu kako bi se naznačilo koje polje ima pogrešan unos	Korisniku se prikazuje poruka o pogrešci za krivo unesenu lozinku. Polje za unos lozinke mijenja boju u crvenu kako bi se naznačilo koje polje ima pogrešan unos
Prijava korisnika u sustav	1. Pritisak na gumb *Sign In* na početnoj stranici 2. Unos teksta „*test.mail@test.com*“ u polje za unos adrese e-pošte 3. Unos teksta „*testing123*“ u polje za unos lozinke 4. Pritisak na gumb *Sign In*	Korisnik se uspješno prijavljuje i preusmjerava na korisničko sučelje kontrolne ploče	Korisnik se uspješno prijavljuje i preusmjerava na korisničko sučelje kontrolne ploče
Kreiranje i upravljanje novim projektom	1. Pritisak na gumb *Create new project* na korisničkom sučelju projekata 2. Unos teksta „*Formo Project*“ u polje za unos naziva projekta 3. Odabir boje projekta 4. Unos teksta „15000“ u polje za unos budžeta projekta 5. Odabir zaposlenika „Samanta Deskar“, „Richard Dawkins“ i „Jane Robertson“ iz padajućeg izbornika za odabir zaposlenika 6. Pritisak na gumb *Create*	Forma za kreiranje novog projekta uklanja se s korisničkog sučelja, prikazuje se obavijest o uspješno kreiranom projektu, a projekt se prikazuje na popisu svih projekata	Forma za kreiranje novog projekta uklanja se s korisničkog sučelja, prikazuje se obavijest o uspješno kreiranom projektu, a projekt se prikazuje na popisu svih projekata

Kreiranje i upravljanje novim projektom	1. Pritisak na gumb *Formo Project* na korisničkom sučelju projekata	Prikaz korisničkog sučelja s podacima novog projekta. Boja naziva projekta i natpisa za budžet jednake su odabranoj boji projekta. Prikazuje se prazna kartica s popisom zadataka i gumb za kreiranje novog zadatka te se prikazuje kartica sa zaposlenicima "Samanta Deskar", "Richard Dawkins" i "Jane Robertson"	Prikaz korisničkog sučelja s podacima novog projekta. Boja naziva projekta i natpisa za budžet jednake su odabranoj boji projekta. Prikazuje se prazna kartica s popisom zadataka i gumb za kreiranje novog zadatka te se prikazuje kartica sa zaposlenicima "Samanta Deskar", "Richard Dawkins" i "Jane Robertson"
Kreiranje i upravljanje novim projektom	1. Pritisak na gumb *Formo Project* na korisničkom sučelju projekata 2. Pritisak na gumb *Tasks* na gornjoj navigacijskoj traci odabranog projekta	Prikaz zadataka pomoću ploče *Kanban*. Na ploči se ne nalazi niti jedan zadatak jer je prikazan novi projekt.	Prikaz zadataka pomoću ploče *Kanban*. Na ploči se ne nalazi niti jedan zadatak jer je prikazan novi projekt.
Kreiranje i upravljanje novim projektom	1. Pritisak na gumb *Formo Project* na korisničkom sučelju projekata 2. Pritisak na gumb *Calendar* na gornjoj navigacijskoj traci odabranog projekta	Prikaz zadataka pomoću gantograma. Na korisničkom sučelju prikazuje se poruka „*There are no tasks for a current project*“.	Prikaz zadataka pomoću gantograma. Na korisničkom sučelju prikazuje se poruka „*There are no tasks for a current project*“.
Prilagodba sustava osobama s oštećenjima vida	1. Odabir stavke *Settings* na lijevoj navigacijskoj traci. 2. Odabir stavke *Appearance* na gornjoj navigacijskoj traci korisničkog sučelja postavki 3. Pritisak na gumb u obliku sklopke pored naziva *Dyslexia*	Zadani font unutar cijelog sustava mijenja se iz *Quicksand* u font *OpenDyslexic.*	Zadani font unutar cijelog sustava mijenja se iz *Quicksand* u font *OpenDyslexic.*
Prilagodba sustava osobama s oštećenjima vida	1. Odabir stavke *Settings* na lijevoj navigacijskoj traci. 2. Odabir stavke *Appearance* na gornjoj navigacijskoj traci korisničkog sučelja postavki 3. Pritisak na gumb u obliku sklopke pored naziva *High Contrast*	Korisničko sučelje prikazano je u načinu visokog kontrasta. Korisničko sučelje prikazano je s tamnosivom pozadinom i bijelim slovima, a lijeva navigacijska traka ima bijelu pozadinu i tamnosiva slova.	Korisničko sučelje prikazano je u načinu visokog kontrasta. Korisničko sučelje prikazano je s tamnosivom pozadinom i bijelim slovima, a lijeva navigacijska traka ima bijelu pozadinu i tamnosiva slova.

Prilagodba sustava osobama s oštećenjima vida	1. Odabir stavke *Settings* na lijevoj navigacijskoj traci. 2. Odabir stavke *Appearance* na gornjoj navigacijskoj traci korisničkog sučelja postavki 3. Pritisak na gumb s oznakom „+“ pored naziva *Font Size* 4. Pritisak na gumb s oznakom „+“ pored naziva *Font Size* 5. Pritisak na gumb s oznakom „+“ pored naziva *Font Size*	Veličina slova na korisničkom sučelju povećana je tri puta.	Veličina slova na korisničkom sučelju povećana je tri puta.
Prilagodba sustava osobama s oštećenjima vida	1. Odabir stavke *Settings* na lijevoj navigacijskoj traci. 2. Odabir stavke *Appearance* na gornjoj navigacijskoj traci korisničkog sučelja postavki 3. Pritisak na gumb s oznakom *Reset* pored naziva *Font Size*	Veličina slova na korisničkom sučelja vraćena je na početnu zadanu veličinu slova	Veličina slova na korisničkom sučelja vraćena je na početnu zadanu veličinu slova
Prilagodba sustava osobama s oštećenjima vida	1. Odabir stavke *Settings* na lijevoj navigacijskoj traci. 2. Odabir stavke *Appearance* na gornjoj navigacijskoj traci korisničkog sučelja postavki 3. Pritisak na gumb s oznakom „-“ pored naziva *Font Size* 4. Pritisak na gumb s oznakom „-“ pored naziva *Font Size* 5. Pritisak na gumb s oznakom „-“ pored naziva *Font Size*	Veličina slova na korisničkom sučelju smanjena je tri puta.	Veličina slova na korisničkom sučelju smanjena je tri puta.

Prema dobivenim rezultatima, sustav postiže ispravan rad za navedene slučajeve. Ispravnost rada sustava ovisi o uspješno detektiranim unosima korisnika na korisničkom sučelju, obradi unosa na korisničkoj strani programskog rješenja, ispravnoj komunikaciji korisničke strane s poslužiteljskom stranom te ispravnoj obradi podataka na poslužiteljskoj strani.

Kako bi analiza pružila kvantitativna mjerila, programsko rješenje ustupljeno je ispitanicima na korištenje. Zbog vremenske ograničenosti, ispitivanje je provedeno na 3 osobe s različitim stanjima oštećenja vida. Osoba 1 je slabovidna, osoba 2 boluje od astigmatizma, a osoba 3 nema oštećenja vida. Ispitanici su prošli tri slučaja ispitivanja rada te odgovorili na pitanja prikazana u

tablici 7.2. Tijekom ispitivanja pratilo se je i uspješno izvršavanje slučajeva te učestalost pogrešaka.

Tab. 7.2. *Rezultati ispitivanja slučajeva nad ispitanicima*

Naziv slučaja	Zadovoljstvo korisničkim iskustvom prilikom izvršavanja slučaja	Je li sustav zadovoljio prilagodbu oštećenjima vida tijekom izvršavanja slučaja?	Komentar
Prijava korisnika u sustav	100%	87%	Osoba 1 prilikom prijave smatra da slova trebaju biti veća. Povećanjem slova unutar izbornika za pristupačnost problem je otklonjen. Osoba 2 je prilikom prijave morala uključiti prilagodbu visokim kontrastom kako bi ispravno vidjela tekst.
Kreiranje i upravljanje novim projektom	100%	80%	Osoba 1 smatra da slova prilikom kreiranja projekta trebaju biti veća. Povećanjem slova problem nije otklonjen. Osoba 2 smatra da je forma za kreiranje projekta presvijetla. Uključivanjem prilagodbe visokim kontrastom problem nije otklonjen.
Prilagodba sustava osobama s oštećenjima vida	73%	80%	Svi ispitanici smatraju da bi postavke pristupačnosti trebale biti izdvojene izvan postavki kako bi se jednostavnije i brže uključile željene prilagodbe. Prilagodbe visokim kontrastom te promjene veličine slova ne primjenjuju se na svim komponentama sustava.

Tijekom ispitivanja, ispitanici su odgovorili na dva pitanja, ocjenama od 1 do 5. Za pitanje „Zadovoljstvo korisničkim iskustvom prilikom izvršavanja slučaja“ odgovarali su s ocjenama:

- 1 – u potpunosti nezadovoljan
- 2 – nezadovoljan
- 3 – niti nezadovoljan, niti zadovoljan
- 4 – zadovoljan
- 5 – u potpunosti zadovoljan

Pitanje „Je li sustav zadovoljio prilagodbu oštećenjima vida tijekom izvršavanja slučaja?“ ispitanici su odgovarali sa sljedećim ocjenama:

- 1 – u potpunosti nije zadovoljio
- 2 – nije zadovoljio
- 3 – niti nije zadovoljio, niti je zadovoljio
- 4 – zadovoljio je
- 5 – u potpunosti je zadovoljio

Analizom provedenog ispitivanja, vidljivo je kako je zadovoljstvo korisničkog iskustva na vrlo visokoj razini, odnosno 91%, uzimajući u obzir sva tri slučaja, dok je prilagodba sustava nešto lošija, no i dalje na visokoj razini od 82%. Uz odgovore, ispitanici su ostavili dodatne komentare koji pokazuju da bi određene komponente trebalo bolje prilagoditi oštećenjima vida te olakšati pristup postavkama prilagodbe.

Tijekom ispitivanja, ispitanike se nadziralo kako bi se dobio uvid u broj uspješno izvršenih slučajeva te broj pogrešaka. Nadzorom je utvrđeno da su svi slučajevi uspješno izvršeni, dok je kod slučaja kreiranja projekta utvrđena pogreška prilikom unosa budžeta. Ispitanik je u polje za unos budžeta unio broj i valutu što je rezultiralo porukom o pogrešci te je u drugom pokušaju, unosom samo broja, slučaj uspješno izvršen. Analizom uspješnosti i pogrešaka izvršavanja slučajeva, zaključeno je da bi trebalo dodatno naglasiti korisniku ako određeno polje sadrži ograničenja prilikom unosa.

8. ZAKLJUČAK

Izradom diplomskog rada istražene su različite metode i okviri agilnog razvoja te su uspoređena postojeća programska rješenja za vođenje projekata agilnim metodama. Nakon istraživanja vrsta oštećenja vida i mogućnosti prilagodbe pristupačnosti, definirani su zahtjevi korisničkog dijela programskog rješenja te su istražene potrebne tehnologije za ispunjavanje korisničkih zahtjeva. Razvijeno programsko rješenje korisnicima omogućuje kreiranje i upravljanje projektima korištenjem agilnih metoda uz prilagodbu pristupačnosti korisničkog sučelja oštećenjima vida. Kako bi sustav ispravno radio, bilo je potrebno ostvariti komunikaciju s poslužiteljskom stranom, koja je razvijena za potrebe diplomskog rada pod nazivom „Programsko rješenje poslužiteljskog dijela web sustava za potporu upravljanja projektima agilnog razvoja programske podrške zasnova na višnitnom paralelizmu i arhitekturi mikrousluga“. Tijekom razvoja pisani su testovi jedinica kôda kako bi se u što ranijoj fazi otkrile moguće pogreške. Na kraju su analizirani rezultati testova, ispravljene postojeće pogreške te analizirana i povećana pokrivenost kôda testovima.

Analiza ispravnosti rada programskog rješenja pokazuje da su moguća unaprjeđenja poput izdvajanja postavki prilagodbe pristupačnosti na lako uočljivo mjesto te poboljšanje pokrivenosti komponenti prilagodbama oštećenjima vida. Programsko rješenje može biti unaprijeđeno dodatnim opisima polja za unos kako bi korisnici što jednostavnije i točnije koristili značajke korisničkog sučelja. Daljnja unaprjeđenja uključivala bi dodatne načine prilagodbe pristupačnosti poput dodavanja podrške za čitače zaslona, omogućavanje korisnicima izbor okvira za vođenje projekata i načina pregleda zadataka te mogućnost povezivanja s drugim aplikacijama kao što su Slack, GitHub ili Google Drive. Dodatna unaprjeđenja podrazumijevala bi i poboljšanje programskog kôda dodavanjem vlastitih *React* kuka kako bi se smanjila količina kôda, ažuriranje inačica programskih biblioteka čime bi se povećala sigurnost i brzina te dodatne mjere zaštite poput kriptiranja podataka i na korisničkoj strani.

LITERATURA

[1] M. Arlović, *Programsko rješenje poslužiteljskog dijela web sustava za potporu upravljanja projektima agilnog razvoja programske podrške zasnovano na višenitnom paralelizmu i arhitekturi mikrousluga*, Osijek: Fakultet elektrotehnike, računarstva i informacijskih tehnologija, 2021.

[2] J. Mero, M. Leinonen, H. Makkonen and H. Karjaluoto, "Agile logic for SaaS implementation: Capitalizing on marketing automation software in a start-up," *Journal of Business Research*, vol. 145, pp. 583-594, 2022.

[3] Agile Manifesto, "Proglas o metodi agilnog razvoja softvera," Agile Manifesto, 2001. [Online]. Available: http://agilemanifesto.org/iso/hr/manifesto.html. [Accessed 4 Srpanj 2021].

[4] K. Brush and V. Silverthorne, "Agile Software Development," Tech Target, Studeni 2019. [Online]. Available: https://www.techtarget.com/searchsoftwarequality/definition/agile-software-development. [Accessed 7 Svibanj 2022].

[5] L. Hazevytch, "Agile Advantages for Software Development and Your Business," Devcom, 26 Veljača 2020. [Online]. Available: https://devcom.com/tech-blog/agile-advantages-for-business/. [Accessed 4 Srpanj 2021].

[6] Guru99, "Agile Methodology: What is Agile Software Development Model & Process in Testing?," Guru99, [Online]. Available: https://www.guru99.com/agile-scrum-extreme-testing.html. [Accessed 4 Srpanj 2021].

[7] D. Radigan, "Stand-ups for agile teams," Atlassian, 2021. [Online]. Available: https://www.atlassian.com/agile/scrum/standups. [Accessed 4 Srpanj 2021].

[8] Digite, "What is Kanban?," Digite, 2021. [Online]. Available: https://www.digite.com/kanban/what-is-kanban/. [Accessed 4 Srpanj 2021].

[9] Asana, "Asana," 2021. [Online]. Available: https://asana.com/. [Accessed 8 Kolovoz 2021].

[10] Atlassian, "Atlassian," Atlassian, 2021. [Online]. Available: https://www.atlassian.com. [Accessed 20 Kolovoz 2021].

[11] Monday, "Monday Product," Monday, 2021. [Online]. Available: https://monday.com/product/. [Accessed 20 Kolovoz 2021].

[12] Centers for Disease Control and Prevention, "What is Vision Imapirment?," 2021. [Online]. Available: http://ophthalmology.pitt.edu/vision-impairment/what-vision-impairment. [Accessed 8 Kolovoz 2021].

[13] P. V. KV Vaishali, "Understanding definitions of visual impairment and functional vision," Common Eye Health, 21 Travanj 2021. [Online]. Available: https://www.cehjournal.org/article/understanding-definitions-of-visual-impairment-and-functional-vision/. [Accessed 8 Kolovoz 2021].

[14] P. Vashist, S. S. Senjam, V. Gupta, N. Gupta and A. Kumar, "Definition of blindness under National Programme for Control of Blindness," *Indian Journal of Ophthalmology*, vol. 65, no. 4, pp. 92-96, Veljača 2017.

[15] D. Azzam and Y. Ronquillo, *Snellen Chart*, Treasure Island, Florida: StatPearls Publishing,

2021.

[16] National Eye Institute, "Color Blindness," National Eye Institute, 3 Srpanj 2019. [Online]. Available: https://www.nei.nih.gov/learn-about-eye-health/eye-conditions-and-diseases/color-blindness. [Accessed 8 Kolovoz 2021].

[17] T. Tola Geletu, M. Muthuswamy and T. Oljira Raga, "Identification of colorblindness among selected primary school children in Hararghe Region, Eastern Ethiopia," *Alexandria Journal of Medicine*, vol. 54, pp. 327-330, 2018.

[18] A. Silver, "Improving The Color Accessibility For Color-Blind Users," Smashing Magazine, 21 Lipanj 2016. [Online]. Available: https://www.smashingmagazine.com/2016/06/improving-color-accessibility-for-color-blind-users/. [Accessed 21 Kolovoz 2021].

[19] Mayo Clinic Staff, "Dyslexia," Mayo Clinic, 22 Srpanj 2017. [Online]. Available: https://www.mayoclinic.org/diseases-conditions/dyslexia/symptoms-causes/syc-20353552. [Accessed 21 Kolovoz 2021].

[20] F. Destoky, J. Bertels, M. Niesen, V. Wens, M. Vander Ghinst, A. Rovai, N. Trotta, M. Lallier, X. De Tiège and M. Bourguignon, "The role of reading experience in atypical cortical tracking of speech and speech-in-noise in dyslexia," *NeuroImage*, vol. 253, 2022.

[21] Bureau of Internet Accessibility, "How to Create Accessible Content and Designs for People with Dyslexia," Bureau of Internet Accessibility, 12 Veljača 2019. [Online]. Available: https://www.boia.org/blog/how-to-create-accessible-content-and-designs-for-people-with-dyslexia. [Accessed 21 Kolovoz 2021].

[22] Dyslexic, "Quick Guide to Making Your Content Accessible," Dyslexic, 2021. [Online]. Available: https://www.dyslexic.com/quick-guide-making-content-accessible/. [Accessed 21 Kolovoz 2021].

[23] Mayo Clinic Staff, "Astigmatism," Mayo Clinic, 5 Listopad 2021. [Online]. Available: https://www.mayoclinic.org/diseases-conditions/astigmatism/symptoms-causes/syc-20353835. [Accessed 10 Kolovoz 2021].

[24] D. M. Albert and D. M. Gamm, "Astigmatism," Encyclopedia Britannica, 23 Svibanj 2021. [Online]. Available: https://www.britannica.com/science/astigmatism-eye-disorder. [Accessed 7 Svibanj 2022].

[25] Encyclopædia Britannica, *Astigmatism*, Encyclopædia Britannica, 2010.

[26] Essential Accessibility, "Accessibility for People with Astigmatism," Essential Accessibility, 30 Lipanj 2020. [Online]. Available: https://www.essentialaccessibility.com/blog/accessibility-for-people-with-astigmatism. [Accessed 7 Svibanj 2022].

[27] Asana, "Asana Colorblind Friendly Mode," Asana, 2021. [Online]. Available: https://asana.com/guide/help/faq/accessibility. [Accessed 21 Kolovoz 2021].

[28] Atlassian, "Atlassian," Atlassian Accessibility, 2021. [Online]. Available: https://www.atlassian.com/accessibility. [Accessed 21 Kolovoz 2021].

[29] K. Kaplan, "Accessibility," Monday, 2021. [Online]. Available: https://support.monday.com/hc/en-us/articles/360000571925-Accessibility. [Accessed 21

Kolovoz 2021].

[30] ReactJS, "React Docs," ReactJS, 2021. [Online]. Available: https://reactjs.org/docs. [Accessed 11 Srpanj 2021].

[31] T. Sufiyan, "What is ReactJS: Introduction To React and Its Features," SimpliLearn, 9 Travanj 2021. [Online]. Available: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs. [Accessed 9 Srpanj 2021].

[32] A. Xie, "Understanding Redux: Beginner's guide to modern state management," Educative, 12 Svibanj 2020. [Online]. Available: https://www.educative.io/blog/understanding-redux. [Accessed 11 Srpanj 2021].

[33] A. Cronj, "React Redux: A Complete Guide to Beginners," Dev.to, 28 Svibanj 2021. [Online]. Available: https://dev.to/ashikacronj/react-redux-a-complete-guide-to-beginners-2a45. [Accessed 11 Srpanj 2021].

[34] Redux, "Redux Essentials, Part 1: Redux Overview and Concepts," Redux, 2021. [Online]. Available: https://redux.js.org/tutorials/essentials/part-1-overview-concepts. [Accessed 11 Srpanj 2021].

[35] S. Periyaiah, "Unit Testing in React with Jest and Enzyme Frameworks," Syncfusion, 26 Studeni 2021. [Online]. Available: https://www.syncfusion.com/blogs/post/unit-testing-in-react-with-jest-and-enzyme-frameworks.aspx. [Accessed 18 Ožujak 2021].

[36] A. D. Abiodun, "A Practical Guide To Testing React Applications With Jest," Smashing Magazin, 24 Lipanj 2020. [Online]. Available: https://www.smashingmagazine.com/2020/06/practical-guide-testing-react-applications-jest/. [Accessed 18 Ožujak 2022].

[37] B. Skoutaris, "Introduction to Jest and Enzyme," Medium, 25 Veljača 2020. [Online]. Available: https://medium.com/@byron.skoutaris/introduction-to-jest-and-enzyme-a2f52c50fb31. [Accessed 18 Ožujak 2022].

[38] R. Fleck, "10 Fundamental UI Design Principles You Need to Know," Dribbble, 6 Prosinac 2021. [Online]. Available: https://dribbble.com/resources/ui-design-principles. [Accessed 22 Ožujak 2022].

[39] DevIQ, "Guard Clause," DevIQ, 2022. [Online]. Available: https://deviq.com/design-patterns/guard-clause. [Accessed 3 Travanj 2022].

[40] MDN Web Docs, "An overview of HTTP," MDN Web Docs, 2022. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview. [Accessed 5 Travanj 2022].

[41] MDN Web Docs, "XMLHttpRequest," MDN Web Docs, 2022. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest. [Accessed 5 Travanj 2022].

SAŽETAK

U ovom diplomskom radu istražene su i analizirane metode agilnog razvoja projekata te su uspoređena postojeća rješenja. Kako bi web sustav bio prilagođen osobama s oštećenjima vida, istražene su različita oštećenja vida i načini kako sustav učiniti pristupačnim osobama s oštećenjima vida. Za razvoj programskog rješenja korištene su tehnologije poput programskog jezika *Javascript* te programskih biblioteka *React* i *Redux*. Nakon istraživanja agilnog razvoja, oštećenja vida i tehnologija, razvijeno je i testirano, na razini jedinica kôda, programsko rješenje korisničkog dijela web sustava za upravljanje projektima agilnim metodama prilagođeno osobama s oštećenjima vida. Programsko rješenje korisnicima pruža kreiranje korisničkih računa i organizacija, kreiranje zaposlenika, kreiranje i upravljanje projektima te kreiranje, izmjenu, pregled i komentiranje zadataka unutar pojedinih projekata s mogućnostima prilagodbe pristupačnosti za oštećenja vida poput disleksije, daltonizma i astigmatizma. Analiza rezultata pokazuje da je sustav moguće unaprijediti poboljšanjem pokrivenosti komponenti prilagodbama oštećenjima vida, izdvajanjem postavki prilagodbi na lako uočljivo mjesto te pružanjem dodatnih opisa i poruka na korisničkom sučelju.

Ključne riječi: agilni razvoj, korisničko iskustvo, oštećenja vida, prilagodba korisničkog sučelja, testiranje.

ABSTRACT

Title: Frontend solution of web system for agile project management of software development adapted for people with visual impairments

In this master thesis, methods of agile project development are researched and analyzed with comparison of existing solutions. In order to make the web system accessible to people with visual impairments, various visual impairments and ways to make the system accessible to people with visual impairments, have been researched. Technologies such as Javascript programming language and React and Redux programming libraries were used to develop the software. After research of agile development, visual impairments and programming technologies, the frontend part of the software solution for agile project management adapted to people with visual impairments have been developed and unit tested. The software solution provides users the possibility to create user accounts and organizations, possibility to create and manage projects and possibility to create, modify, review and comment tasks within individual projects with the ability to adjust accessibility for visual impairments such as dyslexia, color blindness and astigmatism. The results of analysis shows that the software can be improved by improving the coverage of components for visual impairment adjustments, moving the accessibility settings to an easily visible place and by providing additional descriptions and messages on the user interface.

Keywords: agile development, user experience, visual impairments, accessibility, testing.

ŽIVOTOPIS

Samanta Deskar rođena je 17. kolovoza 1996. godine u Virovitici. U Virovitici završava Osnovnu školu Vladimira Nazora nakon koje upisuje Gimnaziju Petra Preradovića smjer prirodoslovno-matematička gimnazija. Godine 2015., nakon završetka srednje škole, upisuje preddiplomski stručni studij Elektrotehnike, smjer Informatika kao redoviti student na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Na trećoj godini studija pohađa praksu u tvrtki Siemens Convergence Creators u Osijeku kao Java programer. Iste godine pohađa Android Dev Akademiju u organizaciji udruge Osijek Software City. Godine 2018. stječe akademski naziv stručni prvostupnik inženjer elektrotehnike, smjer Informatika. Iste godine upisuje program Razlikovnih obveza na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija. Završetkom razlikovnih obveza, 2019. upisuje sveučilišni diplomski studij Računarstva, izborni blok Programsko inženjerstvo. Na prvoj godini diplomskog studija zapošljava se u informatičkoj tvrtki Kod Savjetovanje d.o.o na poziciji *junior frontend developer*. Godine 2022. zapošljava se u tvrtki Glooko d.o.o na poziciji *frontend developer* gdje radi do dan danas.

Samanta Deskar

PRILOZI

Prilog 1: Diplomski rad u .docx formatu

Prilog 2: Diplomski rad u .pdf formatu

Prilog 3: Programski kod korisničke strane programskog rješenja sustava