Web aplikacije za upravljanje korisničkim aktivnostima

Semjaniv, Nikola

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:997166

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-16**

Repository / Repozitorij:

Digital Repository Juraj Dobrila University of Pula

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

NIKOLA SEMJANIV

Web aplikacija za upravljanje korisničkim aktivnostima

Diplomski rad

Pula, ožujak 2023. godine

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

NIKOLA SEMJANIV

Web aplikacija za upravljanje korisničkim aktivnostima

Diplomski rad

JMBAG: Nikola Semjaniv, 0303069498

Studijski smjer: Informatika

Predmet: Izrada informatičkih projekata

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: Doc. dr. sc. Nikola Tanković

Pula, ožujak 2023. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani ***Nikola Semjaniv***, kandidat za magistra ***informatike*** ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da nikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, 5. ožujka 2023 godine

IZJAVA

o korištenju autorskog djela

Ja, ***Nikola Semjaniv*** dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom ***Web aplikacija za upravljanje korisničkim aktivnostima*** koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama. Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 5. ožujka 2023 godine

Potpis

_____

SADRŽAJ

1. UVOD **1**

2. BPMN - NAMJENA I KORIŠTENJE **2**
2.1. Notacija 2

3. PRIMJENA APLIKACIJE **9**
3.1. Definicija problema 9
3.2. Rješenje 9
3.3. Motivacija 11

4. ANALIZA KONKURENTSKIH RJEŠENJA **11**
4.1. Camunda tasklist 12
4.2. Jenkins pipeline 12
4.3. Service now platforma 12
4.4. Prednost aplikacije 13

5. OBLIKOVANJE **13**
5.1. Use case dijagram 14
5.2. Lista procesa 15
5.3. Primjena BPMN modela u aplikaciji 20
5.4. Dodatni primjer BPMN modela 25
5.5. Osnovne informacije o procesu 26
5.6. Dodatne informacije o procesu 27
5.7. Korisnička akcija: forma 28

6. KORIŠTENE TEHNOLOGIJE **35**
6.1. Vue.js 35
6.2. Korišteni paketi 35
6.2.1. NPM 35
6.2.2. Vuex 38
6.2.3. Vue router 39
6.2.4. ESLint 40
6.2.5. Husky 41

7. RAZVOJ APLIKACIJE **43**
7.1. Arhitektura 43
7.2. Struktura ruta 46
7.3. Upravljanje stanjem aplikacije 48

ZAKLJUČAK **56**

LITERATURA **57**

POPIS SLIKA **58**

SAŽETAK **60**

ABSTRACT **61**

1. UVOD

U današnje vrijeme informatizacije velik se naglasak stavlja na automatizaciju i transparentnost procesa koji se odvijaju. Kako bi proces ili neka djelatnost bila uspješna, potrebno je brzo i učinkovito proći određene korake. Kako se procesi razvijaju, potrebni su i ljudi koji njima znaju upravljati i koji ih znaju analizirati, no s druge strane javlja se problem prezentiranja procesa na jednostavan i transparentan način korisnicima tih procesa.

Ovdje počinje ideja o automatizaciji procesa na fakultetu. Za početak radi se o sustavu prijave prakse za studente informatike. Budući da ima dosta studenata koji žele obavljati praksu, nije mali posao koordinirati sve i biti ažuriran o situaciji svakog studenta na praksi. Svaka automatizacija ili čak i najmanja optimizacija procesa započinje s definiranjem samog modela. U ovom slučaju korišten je BPMN zbog svojih raznih prednosti koje će biti kasnije objašnjene. Dobra stvar je ta da već postoji razvijen BPMN *engine* koji čita model što uvelike doprinosi samom razvijanju cjelokupnog sustava koji bi unaprijediti procese na fakultetu. Osim toga, razvijena je i aplikacija za modeliranje procesa tako da je zapravo jedino što fali aplikacija koja će prezentirati taj model korisniku.

Aplikacija je osmišljena prvenstveno kao predložak koji se dalje može razvijati sukladno s potrebama. Fokus u razvoju trebalo je zadržati na sučelju, budući da je glavni problem prikaz BPMN modela korisniku tog procesa. U ovom slučaju studentu informatike koji rješava studentsku praksu. Ono što dodatno treba naglasiti je to da model treba sadržavati samo one korake koji su bitni za korisnika što uvelike doprinosi razumljivosti i jednostavnosti u čitanju procesa.

Izvorišni kod dostupan je na: https://github.com/SemjanivNikola/run-book.git

2. BPMN - NAMJENA I KORIŠTENJE

BPMN (eng. *Business Process Modeling Notation*) otvoreni je standard, odnosno norma za modeliranje poslovnog modela. Sastoji se od simbola i pravila po kojima se vežu simboli. BPMN model nije direktna implementacija tijeka procesa već predstavlja taj proces kroz dijagram na razumljiv način. Koristi se u svrhu prikaza tijeka nekog procesa, njegovih aktera i odluka koji određeni akter može poduzeti u pojedinoj situaciji opisanoj kroz sami model. Dijagrami, odnosno modeli su dizajnirani na način da su detaljni, ali isto tako i lagani za čitanje ukoliko je osoba imalo upoznata s notacijom. Nije potrebno ogromno znanje u čitanju modela, kao ni u modeliranju jednoga. Međutim, kao i sve drugo, učenje same notacije, odnosno kako što prikazati, zahtjeva vremena ako se netko odluči baviti time. Izvora za učenje ima poprilično, a i službena dokumentacija je odlično napravljena, dokumentirana i izrazito detaljna.

Originalna namjena BPMN modela je pomoć u svezi lakšeg razumijevanja procesa kojeg se modelira. Kada se govori o lakšem razumijevanju, prije svega se misli na osobe koje su direktno ili indirektno uključene u životni ciklus procesa, jer je važno održati jednako razumijevanje procesa između ljudi koji ga razvijaju, održavaju, prate i analiziraju kako ne bi došlo do neželjenih posljedica tijekom razvoja, implementiranja ili samog održavanja. Dobro je spomenuti i drugu stranu, osobe koje financiraju i odobravaju implementiranje i provođenje procesa, jer ukoliko oni ne razumiju što se događa vrlo vjerojatno taj proces neće ugledati svjetlo dana. Osim toga, pravilno implementiranje i korištenje BPMN-a može poboljšati stanje organizacije na način da poveća zadovoljstvo korisnika, poboljša organizacijsku responzivnost, kao i poslovne koordinacije i kontrole, prepoznavanje potencijalnih problema te uvid u potencijalna područja poboljšanja.

2.1. Notacija

Sadržaj modela izrađen je od raznih elemenata, odnosno simbola i oblika koji mogu predstavljati pojedini korak ili aktivnost u tijeku procesa. Sami simboli grupirani su u četiri osnovne skupine od kojih će svaka biti obrađena u radu, a to su:

1. Elementi tijeka procesa - sadrže simbole za događaje, aktivnosti i skretnice. Definiraju gdje će početi proces, kako će se izvesti, akcije koje će se izvršiti i gdje će završiti. Ne definiraju sam tijek, već elemente koji čine određeni tijek procesa.

2. Poveznice - povezuju elemente tijeka procesa i označavaju redoslijed prolaska kroz proces. Sami elementi tijeka ne znače ništa ako nisu povezani tako da tvore smislenu cjelinu. Poveznice određuju prolazak kroz tu cjelinu.

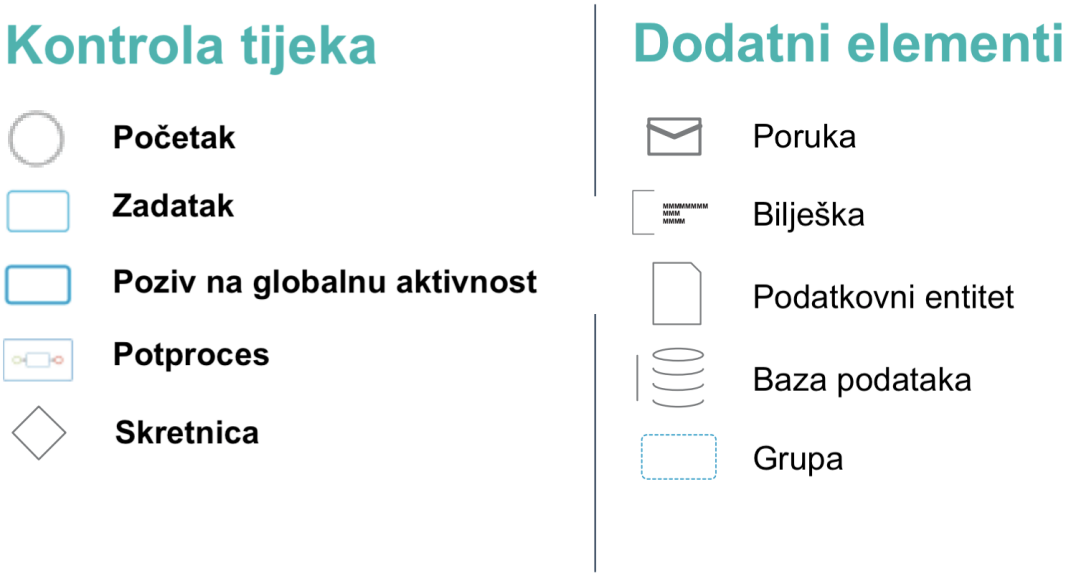
3. Polja i staze - čine značajne elemente u koje se slaže sam tijek procesa. Po hijerarhijskoj podjeli polje je najveći element koji sadrži sve druge, a odmah ispod polja su staze od kojih je samo polje sačinjeno. Jedno polje može imati više staza, a u modelu može postojati i više polja koja su međusobno povezana vezama. Na taj način se može predočiti komunikacija organizacije ili sustava s vanjskim akterima.

4. Artefakti (eng. *artefacts*) - dopune modelu. Predstavljaju važne informacije vezane uz pojedini element u modelu. Može se reći još da proširuju model u smislu dodatnih opisa i informacija. Ne utječu na sam proces, već služe u lakšem shvaćanju događaja ili uloga elemenata.

Određene simbole koristimo s obzirom na tijek koji se prikazuje ili željeni postupak. Tako za prikazivanje događaja koristimo simbole za **početak**, **kraj** i **među događaj** (slika 1). Svaki model procesa u BPMN dijagramu mora započeti sa simbolom za početak događaja i završiti sa simbolom za kraj događaja. Tijek modela odlučuje o tome hoće li biti samo jedan ili više mogućih završetaka.

Slika 1. Simboli za događaje

(Prilagođeno prema: https://www.softwareag.com)



Slika 2. Simboli za kontrolu tijeka i dodatni simboli

(Prilagođeno prema: https://www.softwareag.com)

Osim simbola koji označavaju početak, kraj ili neki važan među događaj u procesu, u velikoj većini se koriste simboli za **kontrolu tijeka** procesa (eng. *control flow elements*) (slika 2). Početak tijeka je već spomenut, a ostali su, redom, **zadatak** (eng. *task*), odnosno **aktivnost**; **poziv na globalnu aktivnost** (eng. *call activity*); **potproces** (eng. *sub-process*) i **prolaz** (eng. *gateway*). Ovi simboli će biti podrobnije

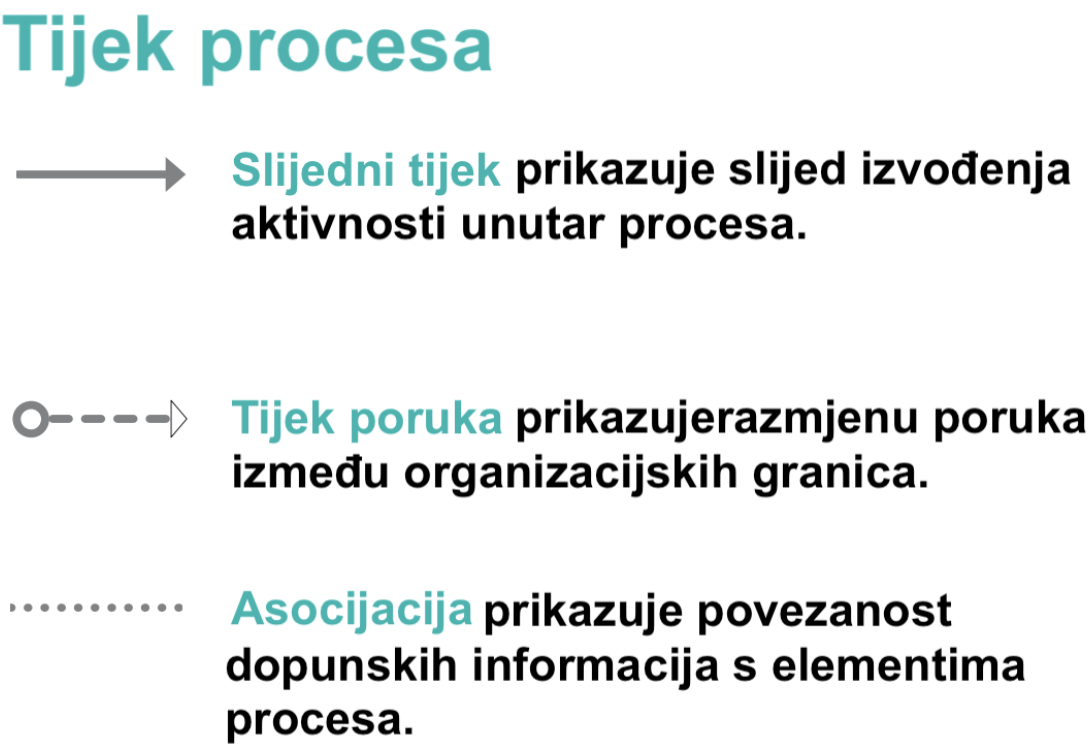
objašnjene u radu kasnije. Osim tih glavnih simbola za kontrolu procesa, koriste se i dodatni simboli u svrhu detaljnijeg opisa zbivanja (slika 2). To su simboli za **poruke** - prikazuju komunikaciju između aktera; **bilješke** - dodatni opis određenog entiteta ili aktivnosti koji je povezan s tom bilješkom; **podatkovne entitete** - definira koje informacije su potrebne ili koje informacije određena aktivnost generira; **baze podataka** - definira pohranjene informacije koje su dostupne izvan obujma procesa; **grupe entiteta** ili **aktivnosti** - grupacije elemenata s obzirom na pojam koji obuhvaćaju.

Nastavno na glavne simbole za kontrolu tijeka, zadatak i globalna aktivnost spadaju pod grupu simbola nazvanu **aktivnosti** (slika 3). Prikazane su kontejnerom i definiraju korake ili globalne procese koji se koriste unutar jednog modela. Pojedini zadatak predstavlja akciju koja se treba izvršiti, a globalna aktivnost predstavlja komponentu koja je definirana izvan samog procesa, ali se ovdje koristi. Isto tako ju je moguće koristiti u drugim modelima nevezanim uz trenutni, jer je sama komponenta definirana kao globalna. Ta globalna komponenta može predstavljati neki predefinirani zadatak ili neki drugi process koji će postati dio trenutnog procesa.

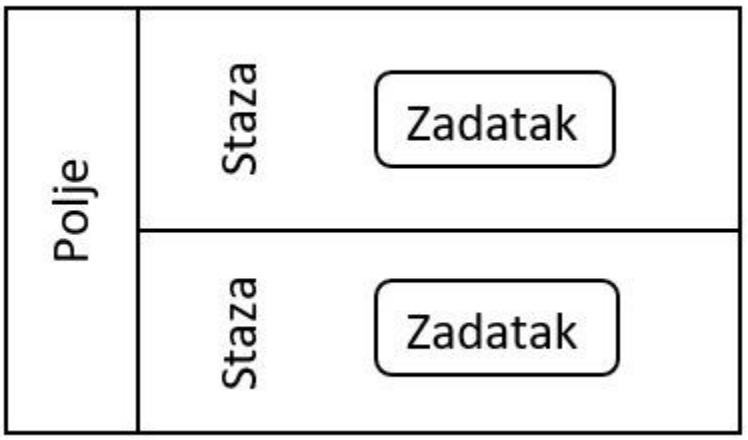
Kada je riječ o zadacima, oni specificiraju prirodu aktivnosti koja se izvodi tako da definiraju kako je ta aktivnost zadatak koju obavlja netko ili nešto. Ovaj detalj može još više pojasniti čemu služi, odnosno uz što se veže i za što je bitna pojedina aktivnost. Na taj način se još brže može razumjeti smisao aktivnosti, tj. tko ju obavlja. Tako su definirani zadaci koje obavlja **korisnik**, pojedini **servis** ili **skriptirani zadatak**, a s druge strane postoje i zadaci koji označavaju **slanje**, **primanje**, **poslovna pravila** te **ručno izvođenje** (slika 3). Osim toga, uz aktivnosti još vežemo i tijek procesa (slika 4). Tijek može biti prikazan kroz tri vrste: **slijedno**, kroz **asocijacije** i **tijek poruka**. Slijedni tijek prikazuje slijed izvođenja aktivnosti unutar modela, dok asocijacije povezuju informacije s elementom u modelu, a tijek poruka prikazuje razmjenu informacija ili poruka između organizacijskih granica. Tijek poruka može biti povezan s aktivnostima, **poljima**, odnosno **sudionicima** (eng. *pools* ili *participants*) i događajem koji označava razmjenu poruka. Kako su polja već spomenuta, valja napomenuti da je to još jedno od osnovnih obilježja BPMN-a (slika 5). Polja se koriste se za organiziranje pojedinih aspekata te prikazuju tijek procesa u kolaboraciji s drugim poljima, ukoliko postoje, i sačinjeni su od minimalno jedne

staze (eng. *lane*). Tako staze, s druge strane, predstavljaju određene organizacijske i tehničke odgovornost koju nosi sudionik u izvođenju procesa. Sudionik ili staza mogu predstavljati organizaciju, sustav ili neku ulogu u organizaciji. Ova dva elementa ne služe samo radi organizacije modela, odnosno procesa, već definiranjem sudionika definira se i proces kojeg on obavlja. Sudionik je zadužen za ono što se nalazi na njegovoj stazi te tako se, analizirajući proces, mogu uočiti potencijalne neučinkovitosti, kašnjenja, zastoji, neuravnoteženosti i slično.

Slika 3. Prikaz aktivnosti i specifikacija zadataka
(Prilagođeno prema: https://www.softwareag.com)



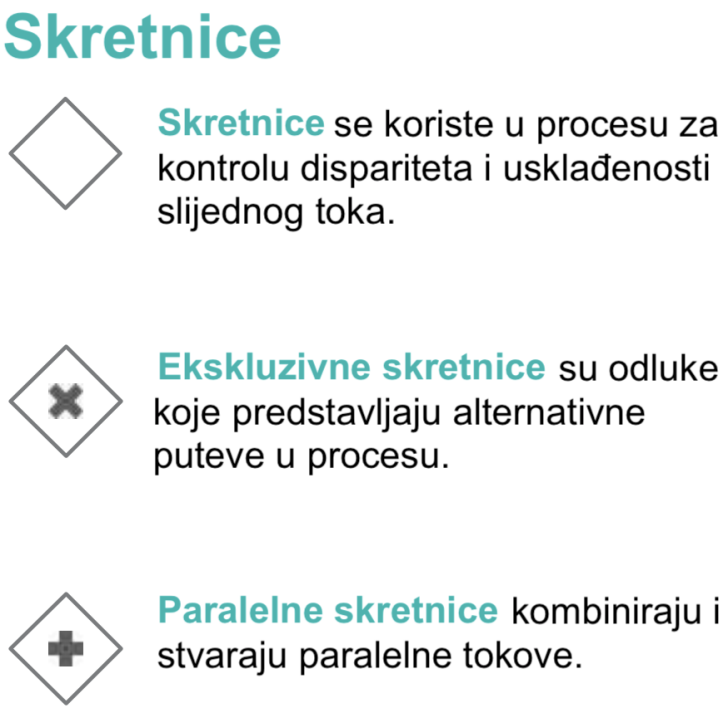
Slika 4. Simboli za određivanje tijeka procesa
(Prilagođeno prema: https://www.softwareag.com)



Slika 5. Simbol za polja i staze (Prilagođeno prema: http://www.bpmb.de)

Osim aktivnosti koje obuhvaćaju zadatke i globalne procese postoje i potprocesi. Oni su u suštini slični kao i globalne aktivnosti, međutim nalaze se samo u jednom konkretnom procesu gdje su definirani i nisu dostupni izvan njegovih granica. Mogu sadržavati druge aktivnosti, prolaze i događaje koji su međusobno slijedno povezani.

Zadnja stvar bitna za spomenuti je i skretnica, odnosno prolaz (slika 6). Prolaz je u modelu prikazan dijamantom koji spaja ili razdvaja tijek procesa na više grana. S obzirom na vrstu prolaza tijek procesa je drukčiji. Prolazi se dijele na ekskluzivne uvjetovane podacima, ekskluzivne uvjetovane događajem, ekskluzivne uvjetovane početnim događajem, inkluzivni, paralelne, paralelne uvjetovane događajem i složene. Međutim, potrebno je naglasiti samo dva najbitnija, a to su ekskluzivni uvjetovani podacima i paralelni. Kod ekskluzivnih prolaza tijek procesa se dijeli na više tokova na temelju određenog uvjeta, dok se paralelni prolaz koristi za predstavljanje dva istodobna koraka u tijeku procesa. Nadalje, paralelni prolaz razlikuje se od drugih prema tome što nije ovisan o uvjetima ili događajima. Ako se aktivnosti odvijaju paralelno, ne znači nužno da se moraju obaviti u isto vrijeme. Kada se koristi paralelni prolaz, prilikom ponovnog spajanja paralelnih tokova u jedan potrebno je opet uključiti paralelni prolaz u model. Na taj način, ukoliko jedan tijek završi prije drugoga pričekat će ga u prolazu prije nastavka procesa.



Slika 6. Simboli za prolaze, odnosno skretnice (Prilagođeno prema: https://www.softwareag.com)

3. PRIMJENA APLIKACIJE

3.1. Definicija problema

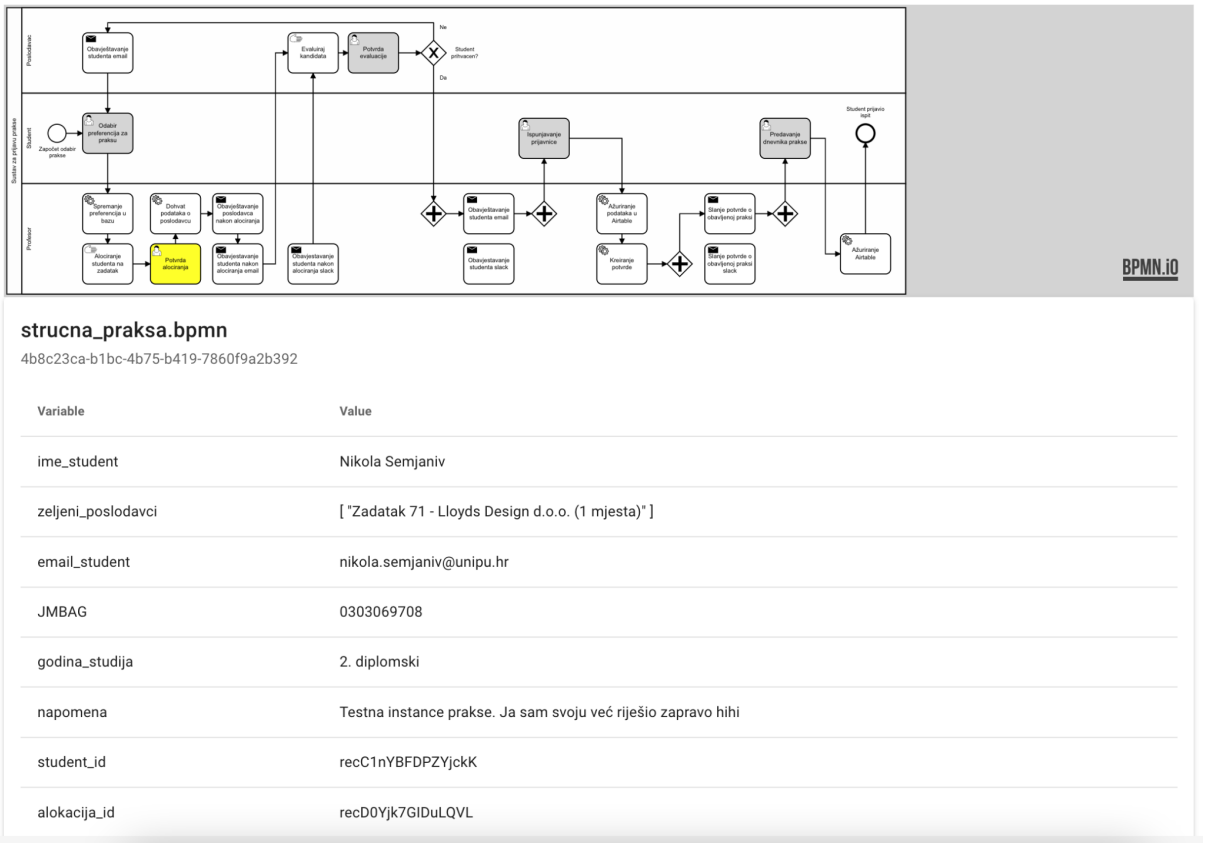
U trenutnom sustavu, koji se razvija radi potrebe rješavanja jednog većeg problema koji obuhvaća zastarjelost obavljanja određenih procesa, došao je na red i problem prezentiranja procesa korisnicima tog procesa. Problem koji je potaknuo razmišljanje na razvoj ove aplikacije je taj da korisnici procesa trebaju vidljivost i preglednost nad procesom u kojem se nalaze. Ponekad je teško snaći se u situaciji u kojoj se čovjek nađe po prvi put i pomoć uvijek dobro dođe. U ovom slučaju, to je razvijanje BPMN modela, a koji treba biti reprezentativan za obje strane i gdje se ne može očekivati da svi korisnici znaju čitati BPMN notaciju i razumiju što se događa u procesu, kao niti koji je sljedeći korak, odnosno trenutni i, konačno, što se od njih očekuje. Zbog toga je potrebno prevesti model, odnosno prikazati ga na jednostavan, pregledan, logičan i shvatljiv način za obje strane. Za ljude koji ga razvijaju, prate, analiziraju i optimiziraju, i za ljude koji ga čitaju kako bi ga uspješno završili.

3.2. Rješenje

Unutar trenutnog sustava koji obuhvaća već nekoliko servisa, odnosno aplikacija potrebno je dodati aplikaciju koja će prikazati BPMN model korisniku na jednostavan i razumljiv način uz sve potrebne informacije koje se veću uz taj proces. Trenutni sustav sadržava sljedeće aplikacije i servise:

- BPMN *engine* - servis koji čita i izvodi BPMN modele. Trenutno podržava samo neke od osnovnih elemenata (elemente za početak i kraj procesa, korisnički zadatak, zadatak servisa, zadatak za poslati, globalnu aktivnost, paralelne i ekskluzivne skretnice, slijedni tijeka procesa i kolaboraciju između polja). Uz *engine* napravljena je i web aplikacija pomoću koje se može složiti BPMN model.
- Sučelje za modeliranje BPMN modela - web aplikacija koja implementira sučelje koje omogućava slaganje BPMN modela, unos (eng. *import*) već gotovog i izvoz modela iz aplikacije (eng. *download*). Osim toga, pruža mogućnost pregleda svih izrađenih modela.

- Adminsko sučelje - osnovno sučelje, odnosno aplikacija koja pruža mogućnost pretrage otvorenih instanci, između ostalog i pretraživanje prema korisniku. Ovo sučelje je napravljeno kao baza za daljnji razvoj sustava. Trenutno prikazuje započete instance s detaljnim pregledom procesa (slika 7) i pregledom izrađenih BPMN modela koji se koriste. Rješenje koje će biti prezentirano dalje u radu je rađeno prema uzoru na sučelje sa slike 7.



strucna_praksa.bpmn

4b8c23ca-b1bc-4b75-b419-7860f9a2b392

Variable	Value
ime_student	Nikola Semjaniv
zeljeni_poslodavci	["Zadatak 71 - Lloyds Design d.o.o. (1 mjesta)"]
email_student	nikola.semjaniv@unipu.hr
JMBAG	0303069708
godina_studija	2. diplomski
napomena	Testna instance prakse. Ja sam svoju već riješio zapravo hihi
student_id	recC1nYBFDPZYjckK
alokacija_id	recD0Yjk7GIDuLQVL

Slika 7.Detalji procesa admminskog sučelja

- Aplikacija za generiranje i slaganje pdf dokumenta - još jedna aplikacija u sklopu cjelokupnog sustava. Nije toliko povezana s trenutnim rješenjem, ali ju valja spomenuti zbog njene izrazite korisnosti.

Konačno rješenje prema prioritetu treba uključivati sljedeće funkcionalne zahtjeve: pregled aktivnih, dostupnih i arhiviranih procesa te prikaz detalja i svih potrebnih informacija koje su vezane uz instancu procesa (slika 7). Proces ne mora biti pokrenuti da prikaz detalja bude omogućen. Korisnik može pregledavati svaki

proces, međutim sučelje će imati određene razlike, odnosno ograničenja s obzirom na status procesa koji je u pregledu. Osim toga, sljedeća bitna stvar je prikazivanje BPMN modela na način koji je jednostavan i razumljiv korisniku, a nakon toga dolaze dodatne funkcionalnosti kao što su pretraživanje i sortiranje stavki popisa procesa. Nakon funkcionalnih zahtjeva potrebno je razmotriti kakvo mora biti ponašanje sustava. Zbog same prirode organizacije u kojoj bi sustav bio primjenjiv, održivost i pouzdanost jedne su od važnijih obilježja koje aplikacija, ali i sam sustav treba imati. Prvenstveno zbog toga što je školovanje dugotrajan proces pa bi tako i ovaj sustav trebao biti održiv bez poteškoća i zastoja kroz duži period vremena, a što je ujedno i pouzdanost da se sustav neće srušiti. Osim toga, pretpostavka je da će se sustav nastaviti razvijati kroz vrijeme što znači da je skalabilnost još jedno od važnijih obilježja koje bi sustav morao imati.

3.3. Motivacija

Tijekom školovanja i sam sam se mnogo puta našao u situaciji gdje nisam bio siguran što se od mene očekuje i što bih trebao napraviti. Uvijek je opcija bila potražiti pomoć kolega, asistenata, profesora, čak i osoblja obrazovne ustanove. Međutim, nisu svi uvijek bili otvoreni za pomoći. Tako da razvoj ove aplikacije ima veliki potencijal u primjenjivosti, jer jednostavnost i razumljivost trenutne situacije igraju veliku ulogu u efektivnom rješavanju zadatka. Primjenjivost cjelokupnog sustava seže čak i van opsega fakulteta. Vidljiv je potencijal da se primjeni u raznim državnim i privatnim sektorima. Uz priliku koja mi je pružena da razvijam ovaj produkt i sa znanjem kojeg imam o razvoju aplikacija, osjećam potrebu da doprinesem društvu ili barem da pokušam doprinijeti društvu na ovaj način. Samim time što je prvobitna publika velika i što bi svima koristilo ovo rješenje, velika je i motivacija za odraditi ovaj projektni zadatak na adekvatan način.

4. ANALIZA KONKURENTSKIH RJEŠENJA

Budući da je ovo specifična stvar vezana uz modeliranje procesa, tržište još ne buja pretjeranim brojem alata koji pristupaju ovom problemu na traženi način. Prije samog razvoja potrebno je istražiti druga rješenja koja se baziraju na istom ili sličnom problemu. Uvidom u mogućnosti drugih aplikacija lakše je osmisliti vlastito

rješenje i nadograditi ga na mjestima na kojima ostale aplikacije manjkaju. Postoji ih nekoliko koji će biti spomenuti dolje.

4.1. Camunda tasklist

Camunda tasklisti je aplikacija integrirana s *Camunda modeler* aplikacijom tako da prikazuje korisniku zadatak koji je na njemu da odradi. S instrukcijama i detaljima vezanim uz zadatak. Kada je model izrađen i pušten u produkciju korisnik procesa u svojoj tasklist aplikaciji vidi zadatke za koje je zadužen. Model kao takav nije prikazan, već samo lista zadataka gdje svaki zadatak ima detaljan pregled podataka vezanih uz korisnikovo zaduženje. Pitanje sučelja, na koje je potrebno osvrnuti se, upozorava na to da prije samog korištenja aplikacije korisnik treba uložiti vremena kako bi se kasnije mogao snaći u aplikaciji i odraditi ono što se od njega očekuje. Što znači da razina UX-a nije na visokoj razini. Ovime se dosta ovisi o trećoj strani i načinu na koji ona pristupa svome rješenju i razvoj istoga.

4.2. Jenkins pipeline

Jenkins je sustav temeljen na poslužitelju. Uz to je i otvorenog koda, a sama namjena je ta da pomaže u automatizaciji određenih dijelova razvoja softvera. Prvenstveno testiranje, proces izgradnje aplikacije te olakšava kontinuirane integracije i kontinuirane isporuke. Njegova veza s BPMN-om leži u tome da se preko modela može definirati i isporučiti proces koji automatski obavlja posao određen modelom. Stvar koja komplicira integraciju je sama konfiguracija tijeka izvođenja procesa i čitanja modela.

4.3. Service now platforma

Platforma koja pruža velik broj mogućnosti od koji su važnije optimizacija procesa u hodu, pametni virtualni agenti, enkripcija osjetljivih podataka, analiza izvođenja procesa te pronalaženje mogućnosti za automatizaciju procesa u izvođenju. Aplikacija korisnicima daje mnogo opcija za upravljanje poslovanjem i poslovnim procesima što je u jednu ruku i više nego previše od ciljeva obrađenih u ovom radu. Što se tiče samog sučelja vezanog uz prikaz procesnog modela moderniji je od prijašnje spomenutih platformi, međutim prikazani model je cjelokupni

BPMN model. Što opet ne zadovoljava uvjet da je potrebno prikazati zadatke relevantne za korisnika procesa.

4.4. Prednost aplikacije

Prednost ove aplikaciju u odnosu na ostale je to što rješava točno određeni problem koje ostala rješenja ne sadrže u potrebnom obliku. Uz to pruža jednostavno intuitivno sučelje uz minimalno mjesta za korisnika da se izgubi između mnogo odabira mogućnosti tj. opcija koje nude ostala rješenja na tržištu. Jednostavnost u sučelju je određen kao obavezan fokus tako da je to veliki faktor kojeg nije lako zadovoljiti pogotovo u već postojećim platformama i aplikacijama. Nije lako zadržati jednostavnost uz veliki broj mogućnosti koje aplikacija pruža. Imajući to na umu aplikaciju je potrebno razvijati tako da dodavanjem novih funkcionalnosti se ne gubi na drugim obilježjima. Uz to lakse je povezati akcije s korakom i prikaz stanja tog koraka

5. OBLIKOVANJE

Kroz ove podnaslove cu vam poblize prikazati razvoj aplikacije i na cemu sm ose temeljili

Budući da je fokus bio na izradi web aplikacije prvenstveno zbog jednostavnosti korištenja, potrebno je orijentirati se u tom smjeru kod dizajniranja sučelja. Dodatni razlozi zašto je mobilna aplikacija isključena iz razvoja, barem za sada, je to zato što web aplikacija ne zahtjeva instalaciju na uređaj kako bi se mogla koristi. Nadalje, korisnik ne mora brinuti o ažuriranju aplikacije. Prema tim stavovima kreirana je i osnovna, admin aplikacija. Međutim, isključivanjem razvoja mobilne aplikacije, nije isključena opcija da aplikacija bude prilagođena za rad na mobilnim uređajima. Stoga je responzivnost važna karakteristika koju aplikacija mora imati. To se isto treba uzeti u obzir prilikom dizajniranja sučelja. Korisnici neće provoditi dane u ovoj aplikaciji, kao npr. na Instagramu ili Facebooku, ali je potrebno učiniti ju praktičnom i pružiti im maksimalan ugođaj tijekom vremena provedenog s radom u aplikaciji. Ono što je još bitno je stvoriti intuitivno sučelje kako bi se mogli snaći bez previše razmišljanja. Značajno poboljšana tehnologija u kasnom dvadesetom stoljeću eliminirala je mnoštvo prepreka dobrom dizajnu sučelja i oslobodila niz novih tehnika prikaza i interakcije umotanih u paket koji se naziva grafičko korisničko sučelje ili, kako se obično naziva, GUI (Galitz, 2007, p. 3). Rečeno je da količina programskog koda posvećenog korisničkom sučelju sada prelazi 50% (ibid.).

Prikazane su akcije koje korisnik moze poduzeti u okviru ove aplikacije

5.1. Use case dijagram

Use case dijagram (slika 8) prikazuje definirane funkcionalnosti aplikacije. Tako se može uočiti kako korisnik (za početak student) ima dvije glavne aktivnosti koje može obaviti. To su pregled liste procesa i pregled detalja procesa odabranog s liste. Lista procesa obuhvaća tri vrste procesa: aktivne, dostupne i arhivirane. Aktivni i arhivirani procesi su instance BPMN modela vezane za određenog korisnika, dok su dostupni procesi samo virtualni prikaz instance. Drugim riječima, instanca još ne postoji, ali prilikom pokretanja se kreira i veže za korisnika.

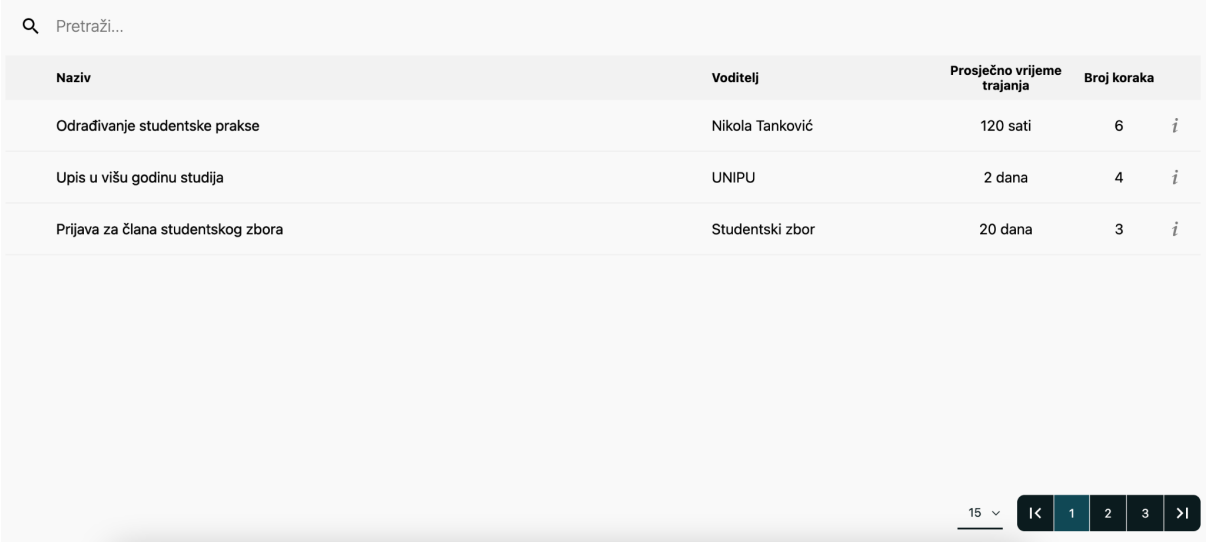
Druga mogućnost je pregled jednog procesa, odnosno njegovih detalja, informacija konkretno vezanih uz taj proces i tog korisnika. Uz informaciju tu je i prikaz koraka koje je potrebno proći kako bi se proces uspješno završio. Na korake se vežu i akcije. Akcije nisu uvijek orijentirane direktno prema korisniku, već ponekad je potrebno pričekati drugog sudionika procesa (npr. profesora) ili sustav da odradi svoj dio zadatka kako bi se tijek procesa mogao nastaviti. Ukoliko je potrebna akcija od strane korisnika, na sučelju će biti prikazana i jasno uočljiva uz koji korak se veže. Trenutno mogu biti dva tipa akcije. Jedno je da korisnik dobije poveznicu na vanjski servis, a drugo da ispuni formu unutar aplikacije. Forma se dinamički generira na sučelju prema određenim, unaprijed definiranim pravilima koja se dobiju s poslužitelja. Ako je to jedna od akcija, forma ima tri moguća stanja: ispunjavanje, uređivanje i pregled ispunjenih varijabli.

Slika 8. Prikaz use case dijagrama za RunBook aplikaciju

5.2. Lista procesa

Format liste najčešći je način predstavljanja skupnih podataka na ekranu i s programske točke gledišta lako ga je napraviti, a mnogi alati za programiranje također omogućuju da se ugradi popis ili tablica u obrazac (Lauesen, 2005, p. 85). U aplikaciji se nalaze tri stranice koje su namijenjene za prikaz skupnih podataka. Lista je dobar, jednostavan i učinkovit način za prezentiranje velike količine podataka

korisniku. Kod liste je potrebno dobro promisliti koji podaci će se prikazati u stavci. To su većinom najbitniji podaci, odnosno nekakva obilježja koja karakteriziraju stavku. U ovom slučaju na pregledu dostupnih procesa, to su osnovne informacije koje će korisnik tražiti - koji je to proces, tko ga vodi i koliko traje. Broj koraka je dodatna, korisna informacija. Posebice je dobra stvar uključiti i tražilicu prilagođenu za pretraživanje podataka na stranici. To uvelike olakšava korisniku snalaženje među podacima, pogotovo ako ima jako puno stavki u listi. Velika količina stavki dovodi do još jednog problema na koji treba pripaziti prilikom dizajniranja sučelja, a to je paginacija. Paginacija rješava problem sporog učitavanja podataka ukoliko prikazujemo sve stavke, a ima ih jako puno. Na maloj količini podataka se to ni ne osjeti, međutim ako se zna da će podaci rasti to je nešto na što se posebno treba obratiti pažnja budući da igra veliku ulogu u tome kako će korisnik doživjeti aplikaciju. Primjer liste s tražilicom i paginacijom je prikazan na slici 9.

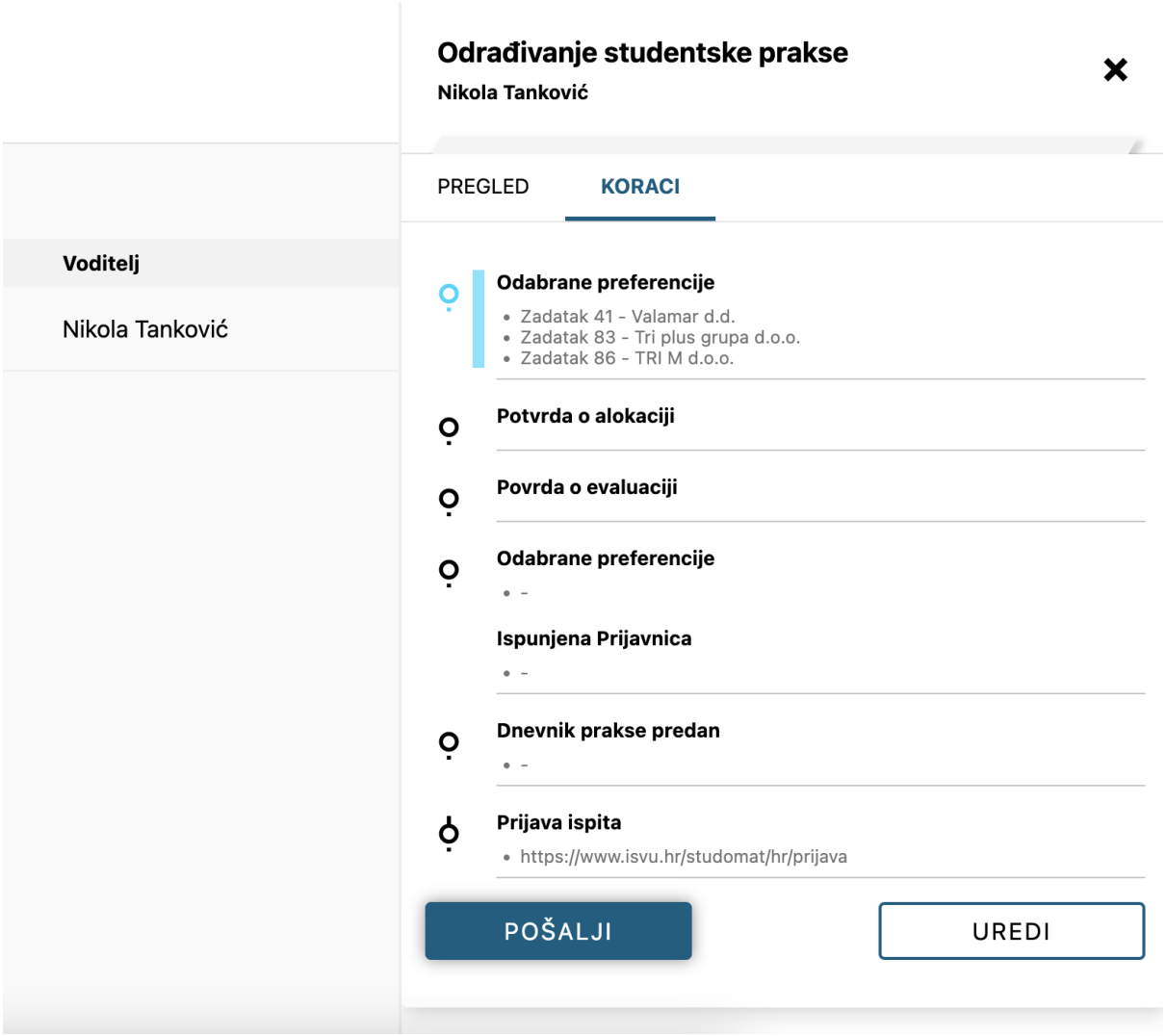


Slika 9. Primjer sučelja gdje se koristi lista kao prikaz skupa podataka

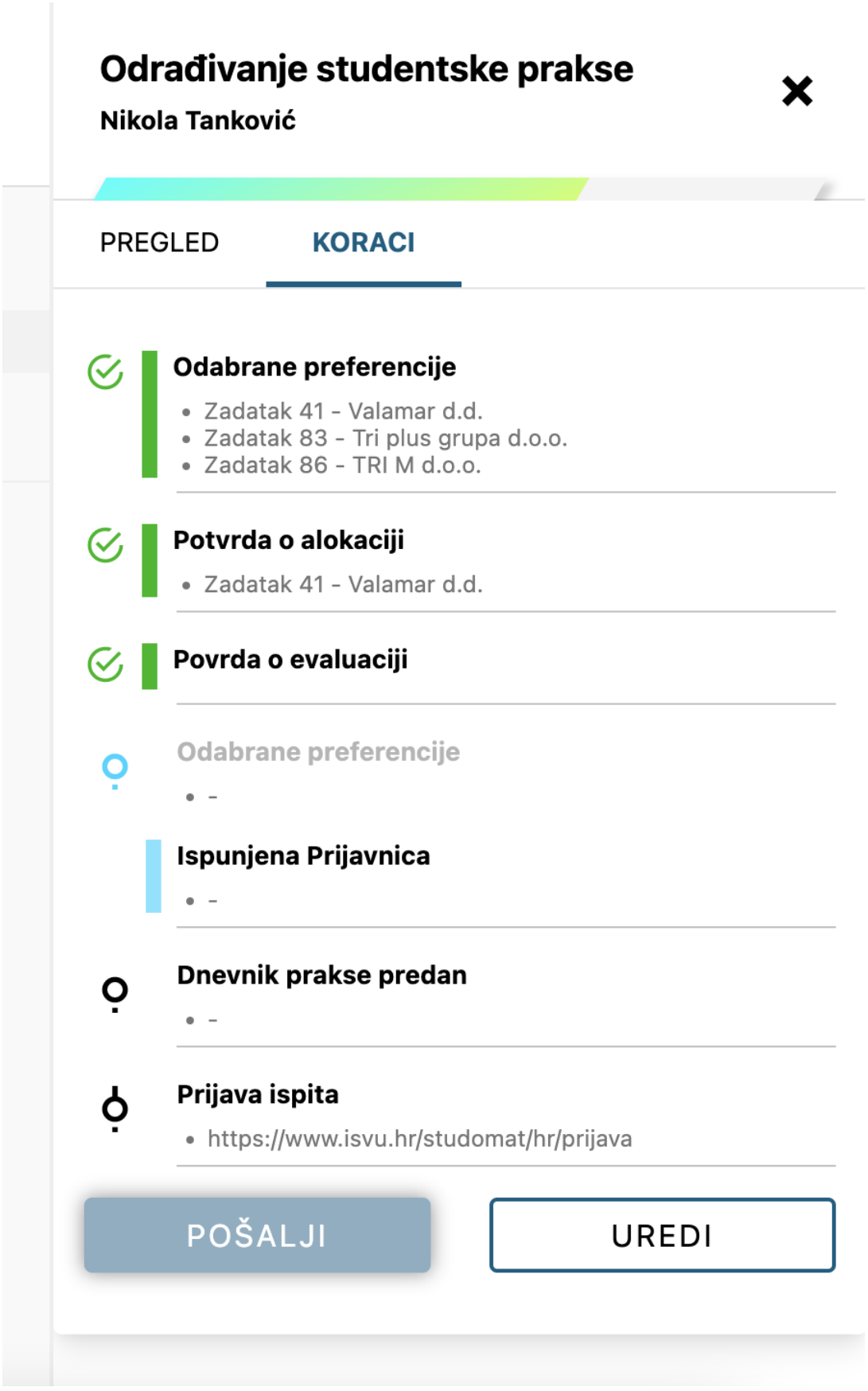
Postoji mnogo načina za prikazivanje podataka i dobra je ideja eksperimentirati s različitim oblicima, a u korisničkim sučeljima često se kombinira format liste s detaljnim prikazom stavke koji prikazuje više detalja o odabranom elementu na popisu (ibid.). Detalji su sastavljeni od pregleda samog procesa (slika 10) gdje se mogu pročitati upute vezane uz sam proces i od pregleda koraka (slika 11) gdje se daje korisniku na uvid što se sve od njega očekuje ukratko. Pojedina grupa koraka počinje prikazanom ikonicom (kružić s točkicom dolje, osim zadnjeg

koraka koji ima još i okomitu crticu gore) i završava horizontalnom crtom kako bi bila izražena i odvojena od ostalih. Također, s obzirom na status, grupa koraka prikazana je određenom bojom. Tako trenutni korak je prikazan svijetlo plavom bojom, a riješeni svijetlo zelenom (slika 12). Ujedno je gumb za slanje podataka onemogućen, budući da nema potrebnih podataka.

Slika 10. Detalji odabranog elementa s popisa - pregled



Slika 11. Prikaz detalja odabranog elementa s popisa - koraci

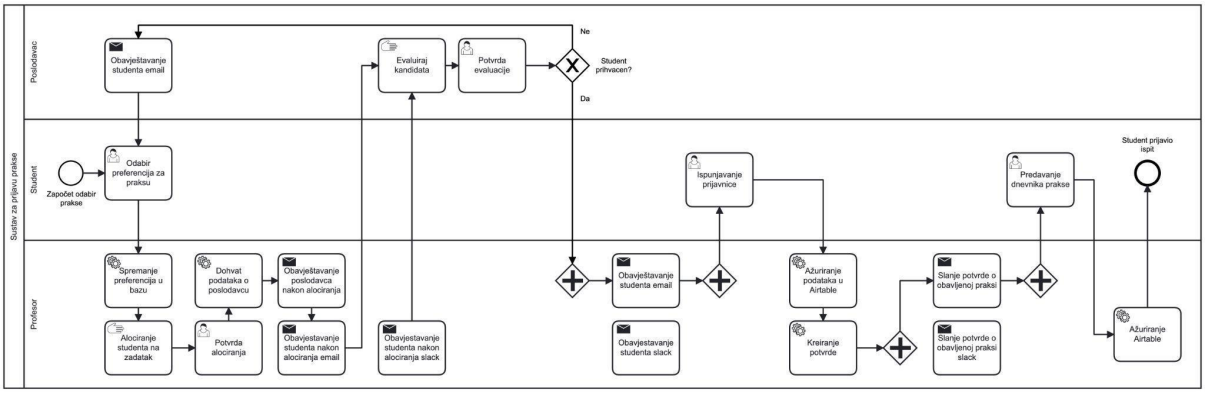


Slika 12. Prikaz detalja odabranog elementa s popisa - koraci

5.3. Primjena BPMN modela u aplikaciji

Na početku rada spomenut je BPMN model i objašnjena je osnovna notacija, međutim vrijeme je dotaknuti se konkretne primjene modela u aplikaciji. One osnovne namjene aplikacije. Na slici sedam prikazano je sučelje s podacima o procesu, tzv. stranica s detaljima o stavci. Tu je već prikazan BPMN model, međutim problem je taj što je to izvorni model i ne bi bilo učinkovito prikazati ga kao takvog korisnicima koji se ne razumiju u BPMN notaciju, još manje koji nisu povezani s informatičkim sektorom ili su slabo upoznati. Zbog toga, najbolji pristup je osmisliti na koji način će se prikazivati model na korisničkom sučelju. Lista nije toliko učinkovita, jer u modelu postoje elementi koji sadrže određenu kompleksnost u prikazu. To su skretnice. Što znači da iz jednog koraka možemo ići na više strana. Pored toga, još jedan izazov nalazi se u tome da se trebaju prikazati samo koraci, odnosno zadaci relevantni za korisnika. To znači da aktivnosti obilježene kao *business rule task, script task* i *service task* se neće prikazivati na sučelju. Matrica ima sposobnost pokriti slučajeve koji su van opsega liste. Budući da su ograničenja definirana potrebno je odrediti i pravila po kojima će se slagati podaci u matricu:

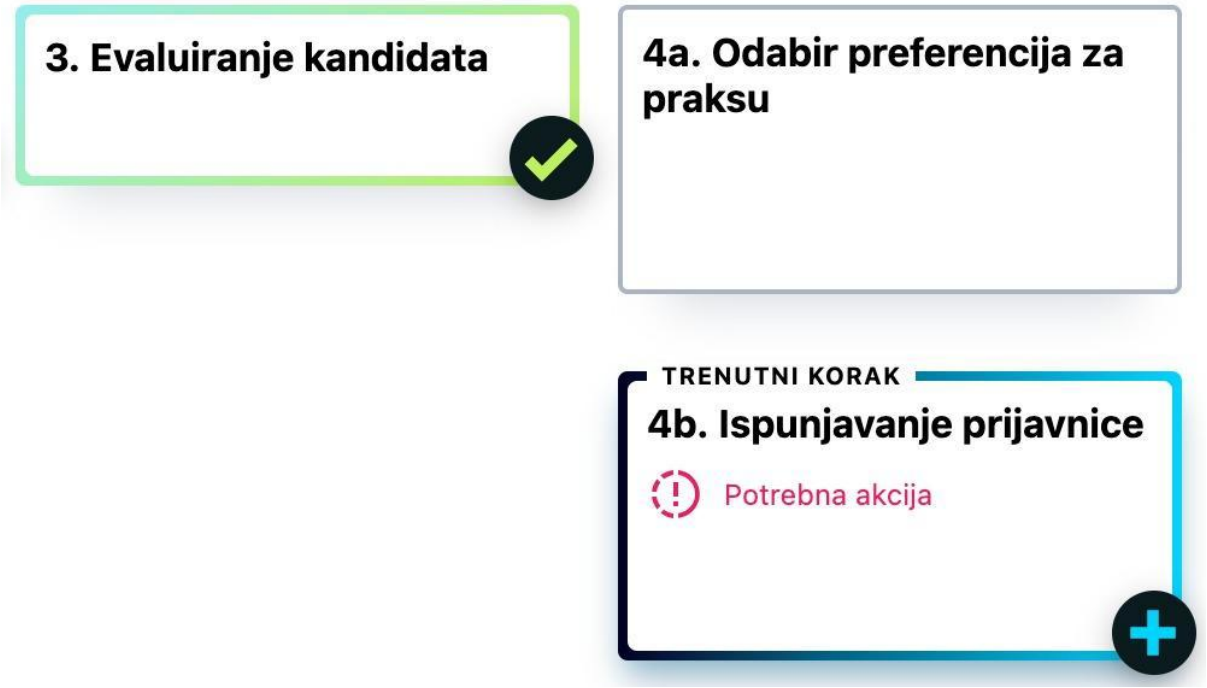
1. Prikazati samo aktivnosti, odnosno zadatke.
2. Od aktivnosti uvijek prikazati samo one označene kao *user task* i *manual task*.
3. Ako se *send task* nalazi nakon *user task-a*, pripoji se njemu, inače se zanemaruje. Osim ako se *receive task* nalazi nakon *send task-a*, onda je zaseban korak.
4. Ako je *receive task* prije *manual* ili *user task-a*, onda se spaja s njime.
5. Ekskluzivna skretnica uvjetovana događajem prikazuje se tako da se svi mogući ishodi nalaze u stupcu koji označava sljedeći korak. Ukoliko jedan od sljedećih koraka predstavlja povratak na neki od prethodnih, korisnik se vraća na njega. Ne nastavlja se dalje.
6. Paralelna skretnica znači da se dva koraka odrađuju paralelno, a to se prikazuje na isti način kao i da se jedan korak odrađuje.



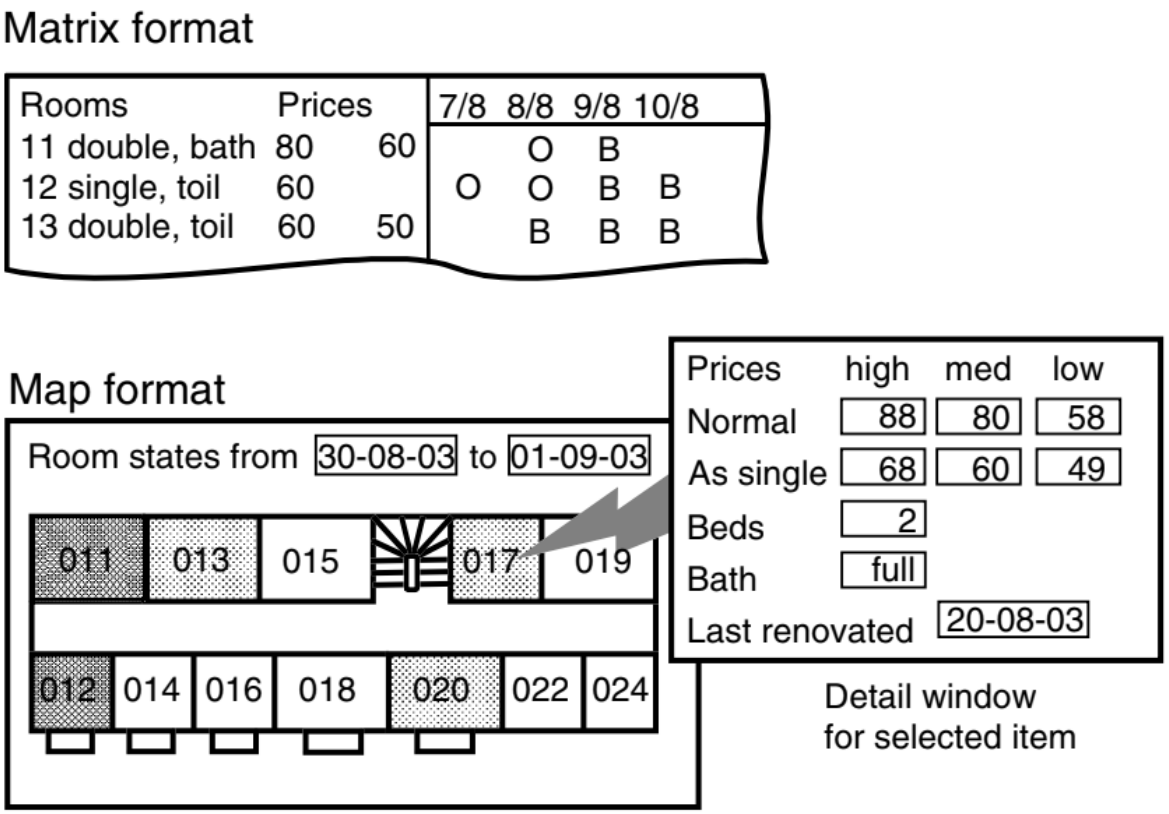
Slika 13. Sustav za prijavu prakse

Budući da su ograničenja i pravila definirana, može se prijeći na konkretnu primjenu modela sustava za prijave prakse u aplikaciji (slika 13). U modelu se nalazi nekoliko ključnih koraka koji su bitni za glavnog sudionika procesa, a to su: odabir preferencija za praksu, potvrda alociranja, potvrda evaluacija, ispunjavanje prijavnice, predaja dnevnika prakse i prijava ispita, što je ujedno i kraj procesa. Sve te aktivnosti označene su kao korisnički zadaci i nose određenu važnost na sebi. Sve navedene aktivnosti, osim potvrde alociranja i potvrde evaluacije, zahtijevaju od korisnika, tj. glavnog sudionika procesa da ih odradi na način da ispuni određenu formu kako bi skupio potrebne informacije.

U modelu se nalazi i jedna bitna skretnica koja se nalazi nakon potvrde evaluacije i s obzirom na ishod evaluacije određuje daljnji tijek procesa. Ako je evaluacija pozitivna, sudio nastavlja dalje s ispunjavanje prijavnice, međutim ako je evaluacija negativna, korisnik se ponovno nalazi na početku procesa gdje se od njega traži da ažurira odabir poduzeća za obavljanje prakse. Kako bi korisnik znao koji korak slijedi nakon potvrde evaluacije, potrebno je oba ishoda staviti pod jednu grupu (slika 14). To je zapravo i jedini smisao grupacije koraka. Ako ima više koraka u istoj grupi to znači da ih je moguće pokrenuti nakon trenutnog. Grupacija uvijek dolazi nakon skretnice, odnosno koraka koji je prije skretnice. S obzirom na skretnicu, koraci u grupi se mogu odrađivati paralelno ili pojedinačno.

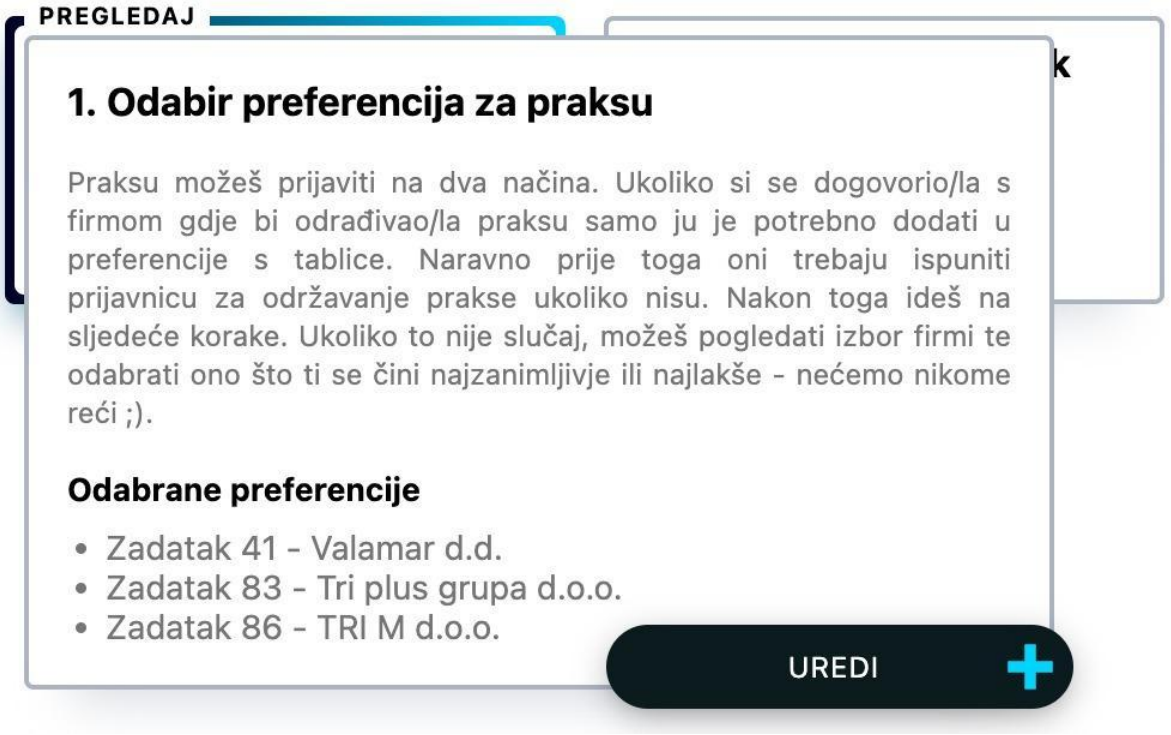


Slika 14. Grupacija koraka nakon skretnice - 4a i 4b



Slika 15. Format matrice s prozorom za detalje stavke (Izvor: Lauesen, 2005, p. 87)

Oblik matrice može dati izvrstan pregled, ali je teško naći mjesta za sve detalje pa se može ponovno kombinirati s detaljnim prikazom stavke (Lauesen, 2005, p. 87) kako je prikazano na slici 15, dok je konkretna primjena u aplikaciji prikazan na slici 16. Detalji su vrlo jednostavno prikazani, bez suvišnih informacija.

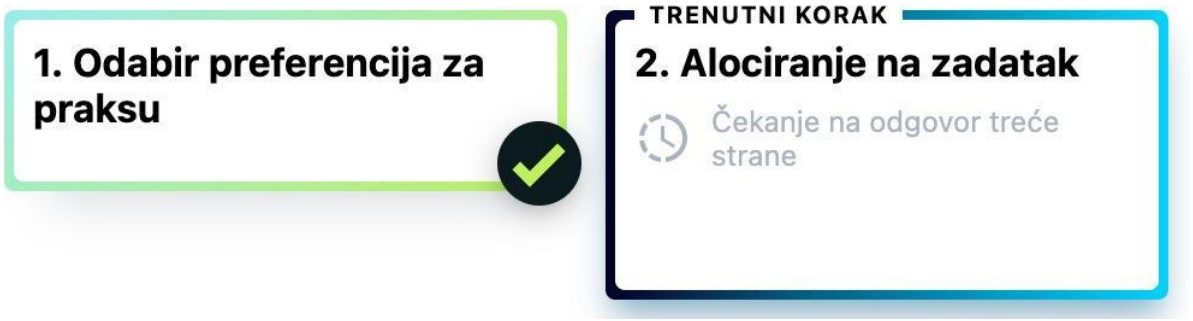


Slika 16. Prikaz detalja matrice

Ono što se vidi kada se otvore detalji su naziv koraka, kratke upute ili opis koraka te, ukoliko postoje, neki podaci koje je korisnik ispunio u ovom koraku. Osim mogućnosti pregleda podataka, moguće je i uređivanje tih podataka.

Kada je riječ o prikazu akcije koju korisnik treba poduzeti, riješena je na način prikazan na slici 17. Korisnik se obavještava na način da je korak označen s „potrebna akcije“ i to crvenom bojom s ikonicom koja predstavlja obavijest. Također, potrebno je prikazati i kada nije na korisniku da obavi određeni zadatak, a što je vidljivo na slici 18.

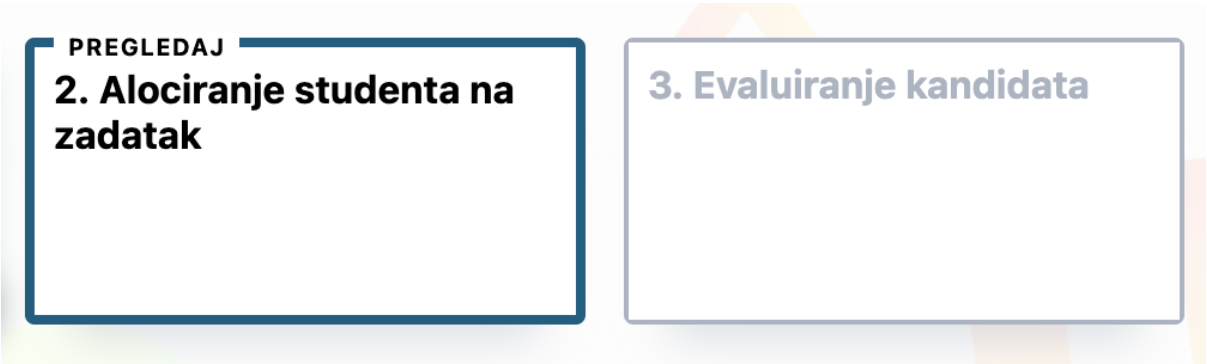
Slika 17. Prikaz situacije kada je na korisniku da izvrši određenu radnju



Slika 18. Prikaz situacije kada nije na korisniku da izvrši zadatak

Korak koji je završen (slika 19) prikazan je u kontrastnim bojama kako bi se razlikovao od neizvršenih koraka (slika 20), a ujedno i od trenutnog koraka. Da bi kontrast funkcionirao, mora postojati pozadina za kontrast - ako se naglašavaju previše stvari, učinak nestaje, tj. ako je sve u živim bojama, ništa nije naglašeno, a ako su mnoge stvari već naglašene, potreban je još veći učinak da bi se jednu od njih naglasilo pa je opće pravilo umjereno korištenje kontrasta i naglašavanja, inače učinak nestaje (Lauesen, 2005, p. 80).

Slika 19. Prikaz završenog koraka

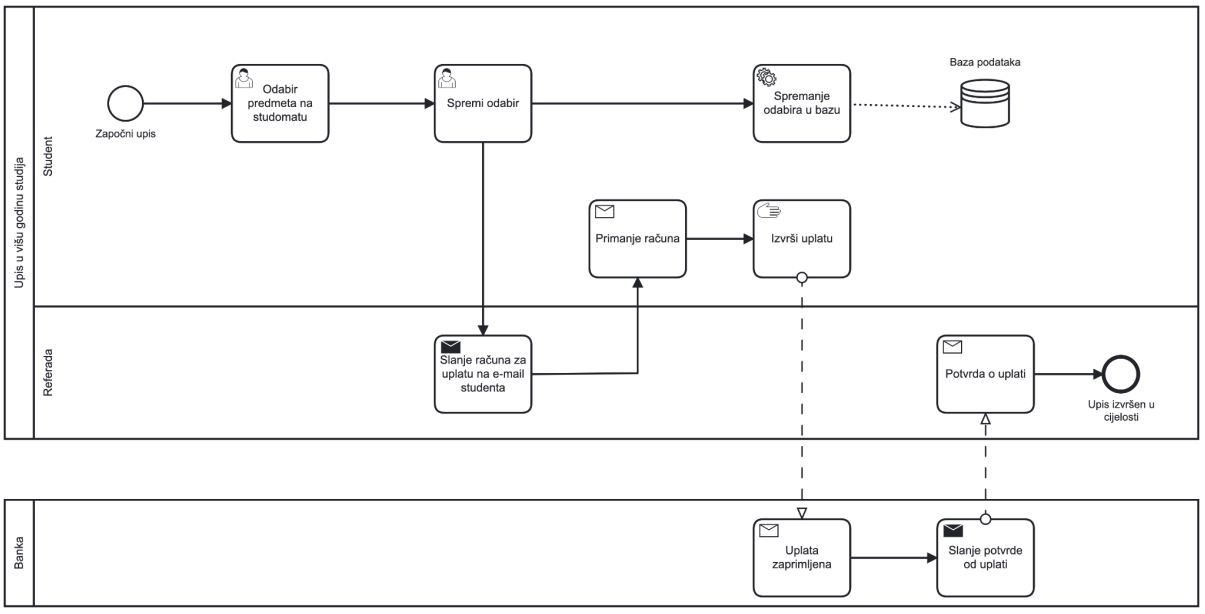


Slika 20. Korak koji je neaktivan

Jedan tip BPMN modela je prikazan kroz matricu u detaljima procesa, dok u aplikaciji postoji i drugi tip prikaza, a to je kroz listu kako je kratko prikazano na slici 12. Lista je umanjeni i pojednostavljeni prikaz matrice s varijablama što je zapravo i osnovna namjena, prikazati koje su varijable prikupljene u kojem koraku i što je sve skupljeno kroz proces. Koraci su kao i u matrici podijeljeni u grupe koje odvaja ikonica s lijeve strane koja označava početak i horizontalna linija između grupa koja označava kraj grupe.

5.4. Dodatni primjer BPMN modela

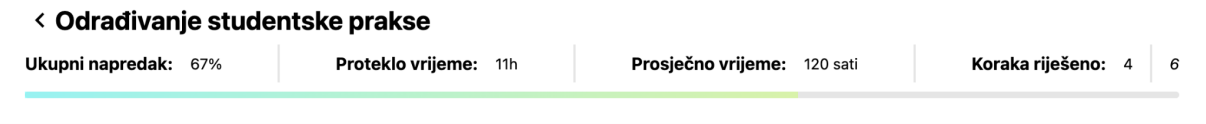
Gore je naveden osnovni primjer na kojem je započeo razvoj aplikacije, dok se na slici 21 može pogledati novi primjer BPMN modela koji opisuje proces upisa u višu godinu studija.



Slika 21. BPMN model upisa u višu godinu studija

5.5. Osnovne informacije o procesu

Osnovne informacije o procesu prikazane su u zaglavlju stranice. Informacije su sastavljene od podataka vezanih uz sam tijek procesa. To su: napredak korisnika kroz proces prikazan u postotku te kroz vizualni prikaz, tzv. *progress bar*, koji se puni s kontrastnom bojom ovisno o postotku riješenosti; podatak o vremenu iz kojeg se iščitava korisnikovo ukupno utrošeno vrijeme i prosječno ukupno vrijeme potrebno za rješavanje procesa; i kao zadnja stavka postavljena je riješenost koraka. Slika 22 prikazuje izgled sučelja za web, a slika 23 izgled sučelja na mobilnim uređajima.



Slika 22. Osnovne informacije prikazane na webu

Slika 23. Osnovne informacije prikazane na mobilnom uređaju

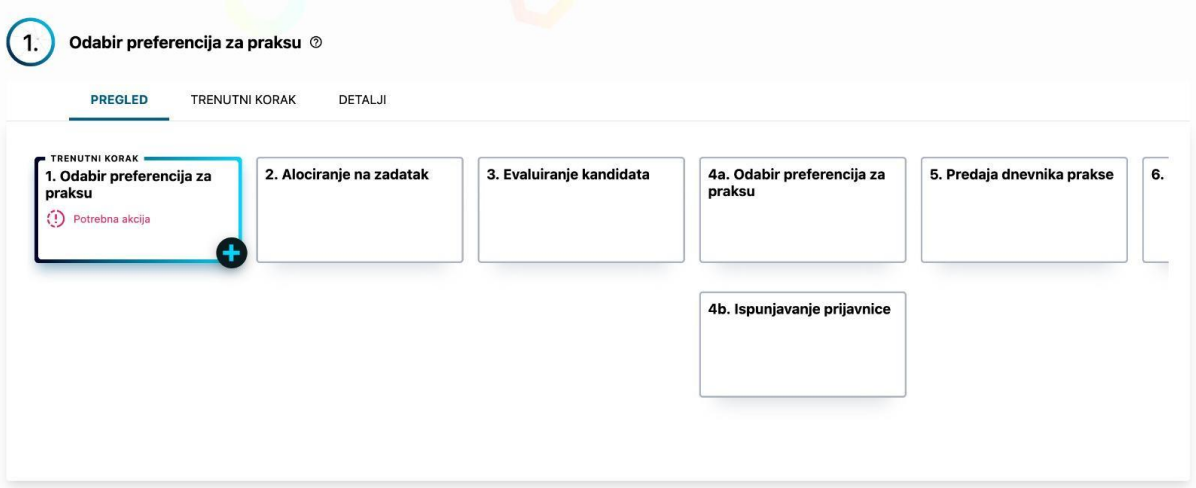
5.6. Dodatne informacije o procesu

Dodatne informacije o procesu prikazane su kroz tri taba:

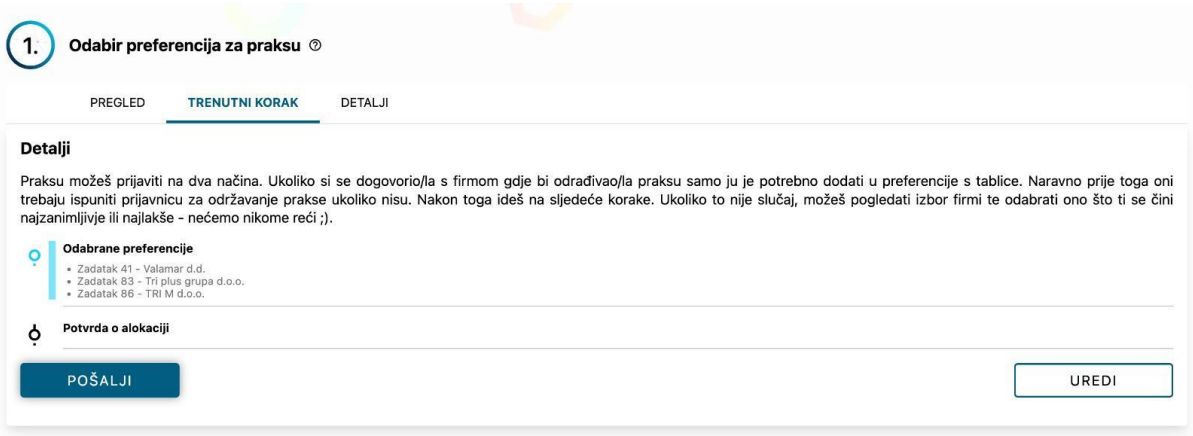
1. Pregled - prikazuje dijagram procesa na razumljiv, jednostavan i intuitivan način (slika 24.).

2. Trenutni korak - prikazuje detalje o trenutnom koraku (slika 25). Pod detalje o koraku spada kratak opis koraka, ukoliko postoji, u slučaju da je potrebno podrobnije objašnjenje. Osim toga, tu se nalazi i kratka lista koraka u procesu koja se sastoji samo od trenutnog i sljedećeg koraka. Također, bitno je za naglasiti da je tu određene varijable, koje je korisnik skupio na trenutnom koraku, moguće urediti prije spremanja i prelaska na sljedeći korak.

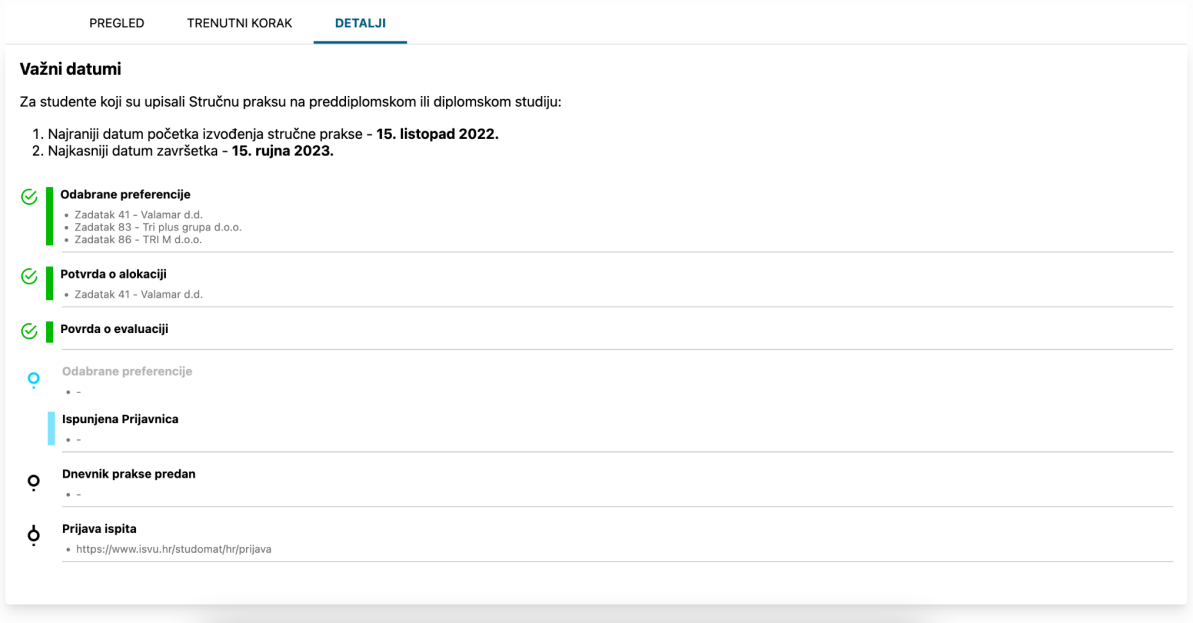
3. Detalji procesa - prikazuju opis, odnosno tekst uz proces ako ga ima. Ispod teksta se nalazi lista svih koraka i varijabli skupljenih tijekom rješavanja zadataka tj. prolaska kroz proces. U ovom slučaju, akcije koje pružaju mogućnost uređivanja i slanja podataka nisu dostupne, jer je ovo prvenstveno pregled cjelokupnog stanja.



Slika 24. Pregled procesa



Slika 25. Trenutni korak



Slika 26. Detalji - prikaz koraka kroz listu

5.7. Korisnička akcija: forma

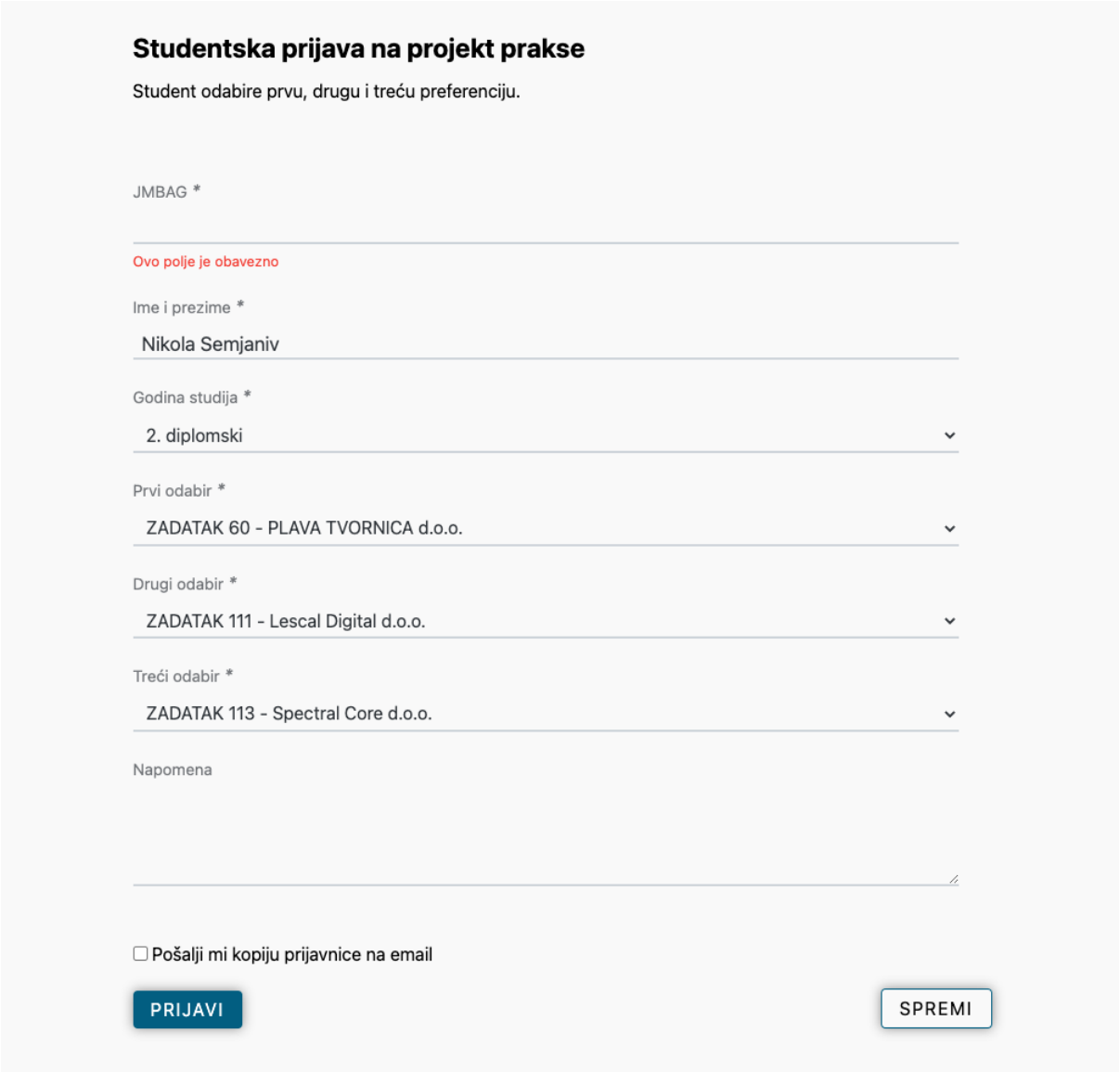
Forma je jedna od dvije akcije ukoliko se očekuje od korisnika da treba nešto poduzeti. Forma se dinamički slaže, tj. generira na stranici preko paketa koji će biti kasnije u radu spomenut. Bitno je samo naglasiti kako se prije svega mora posložiti struktura forme. Struktura može sadržavati razna stanja. Od običnih tekstualnih inputa preko *select*-a pa sve do tabova i gumba s nekom akcijom. Za aplikaciju su

korištene već definirane strukture forme, jer nije bitno prolaziti kako se fore slažu, već samo kako se prikazuju u aplikaciji. Promatrajući sustav za prijavu prakse prikazan na slici 14 mogu se izvući tri koraka kod kojih se od korisnika očekuje ispunjavanje neke vrste forme.

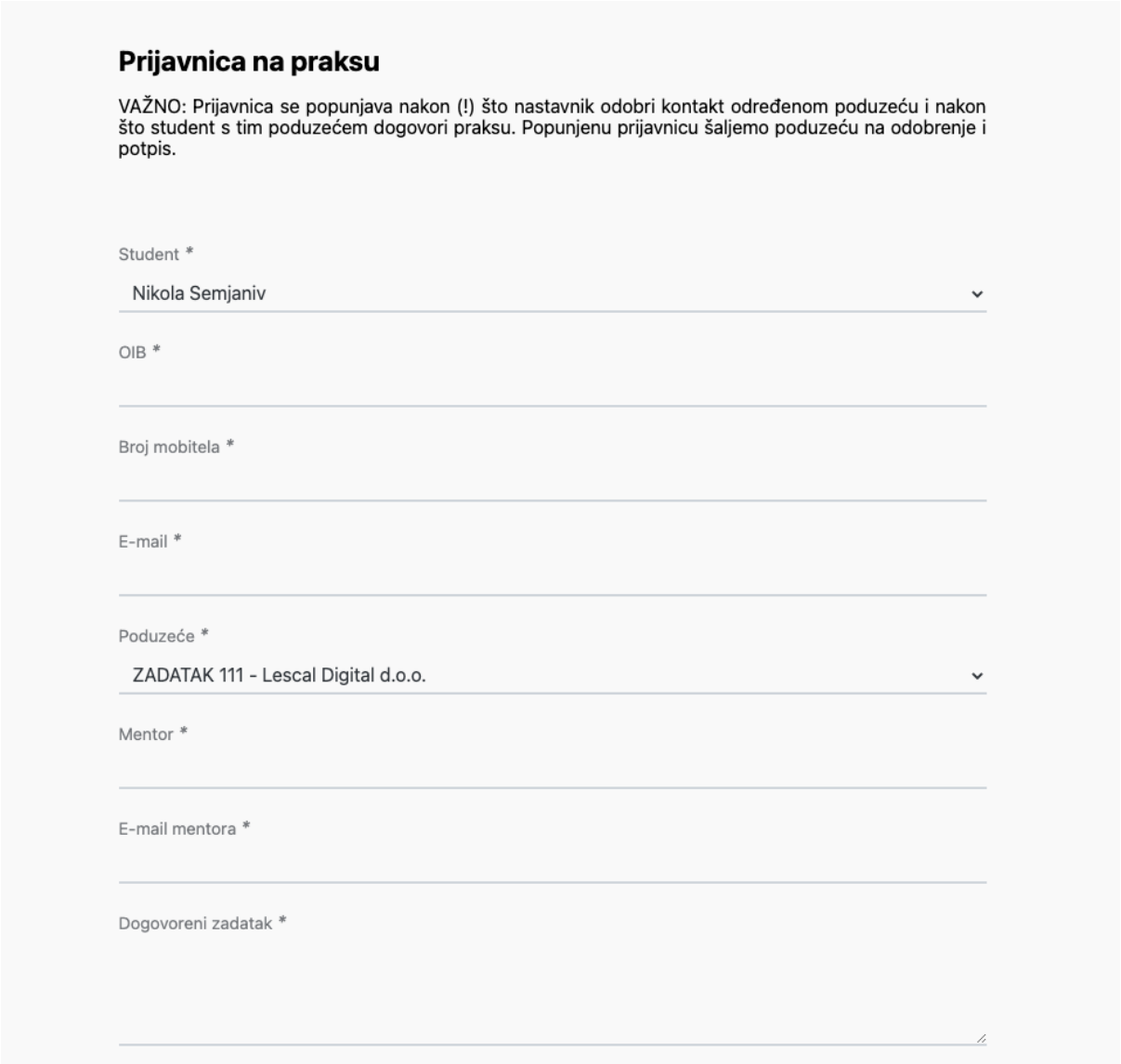
1. Prva forma nalazi se na prvom koraku kod odabira željenih preferencija za obavljanje prakse (slika 27) i sadrži sljedeća polja kao obavezna: JMBAG, ime i prezime, godinu studija, prvi, drugi i treći odabir firme. Uz to na formi se nalazi još mogućnost unosa napomene i odabira da se na e-mail korisnika pošalje kopija unesenih odgovora.

2. Sljedeća forma pojavljuje se na koraku ispunjavanja prijavnice i izgleda kao na slici 28 i 29. Tu se od korisnika traži da ispuni polja kao što su student (ime i prezime), OIB, broj mobitela, e-mail, odabir poduzeća, mentor, e-mail mentora, dogovoreni zadatak, dogovoreni broj sati, datum početka, datum završetka, te potvrde alokacije i dogovorenih detalja.

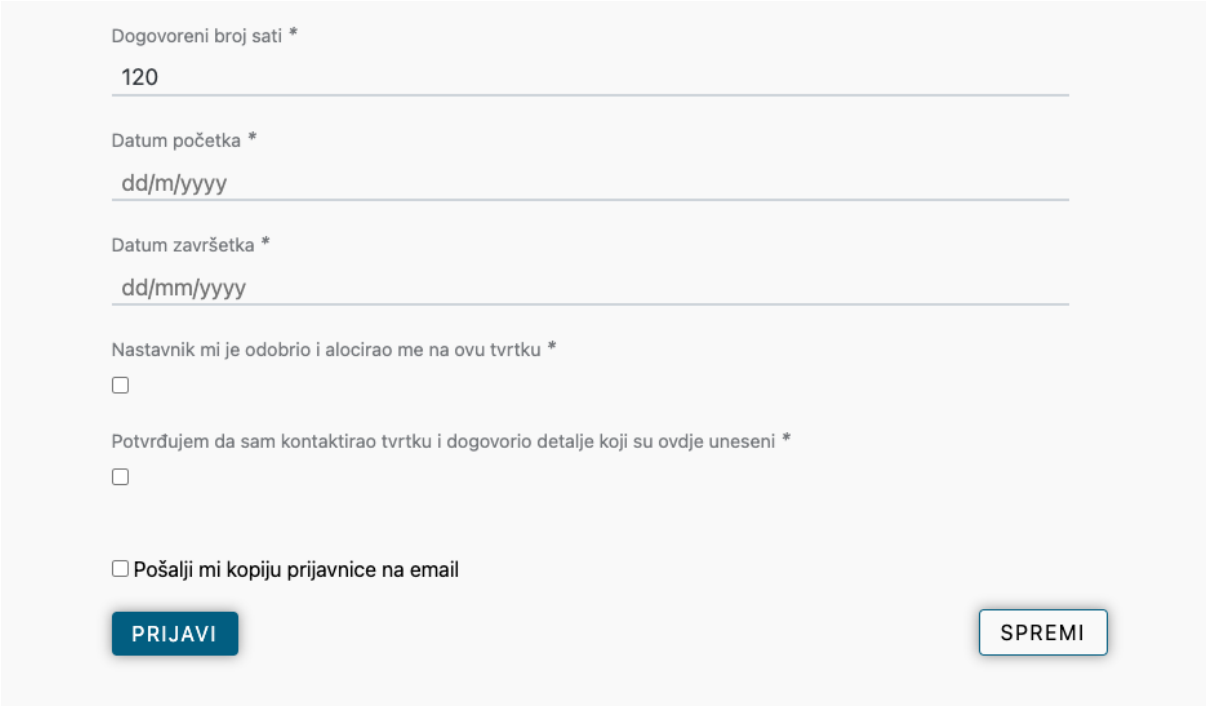
3. Zadnja forma koju korisnik mora ispuniti je predaja dnevnika prakse. Tu se nalazi samo jedno input polje koje traži korisnika da preda dnevnik prakse (slika 30). Kada korisnik obavi uspješnu predaju forme, aplikacija ga vodi na sučelje gdje je prikazano da je uspješno odradio zadatak (slika 31). Kako bi korisniku dali do znanja da je obavio dobar zadatak, potrebno je to dočarati. To se može ostvariti na način da se pruži pozitivna povratna informacija.



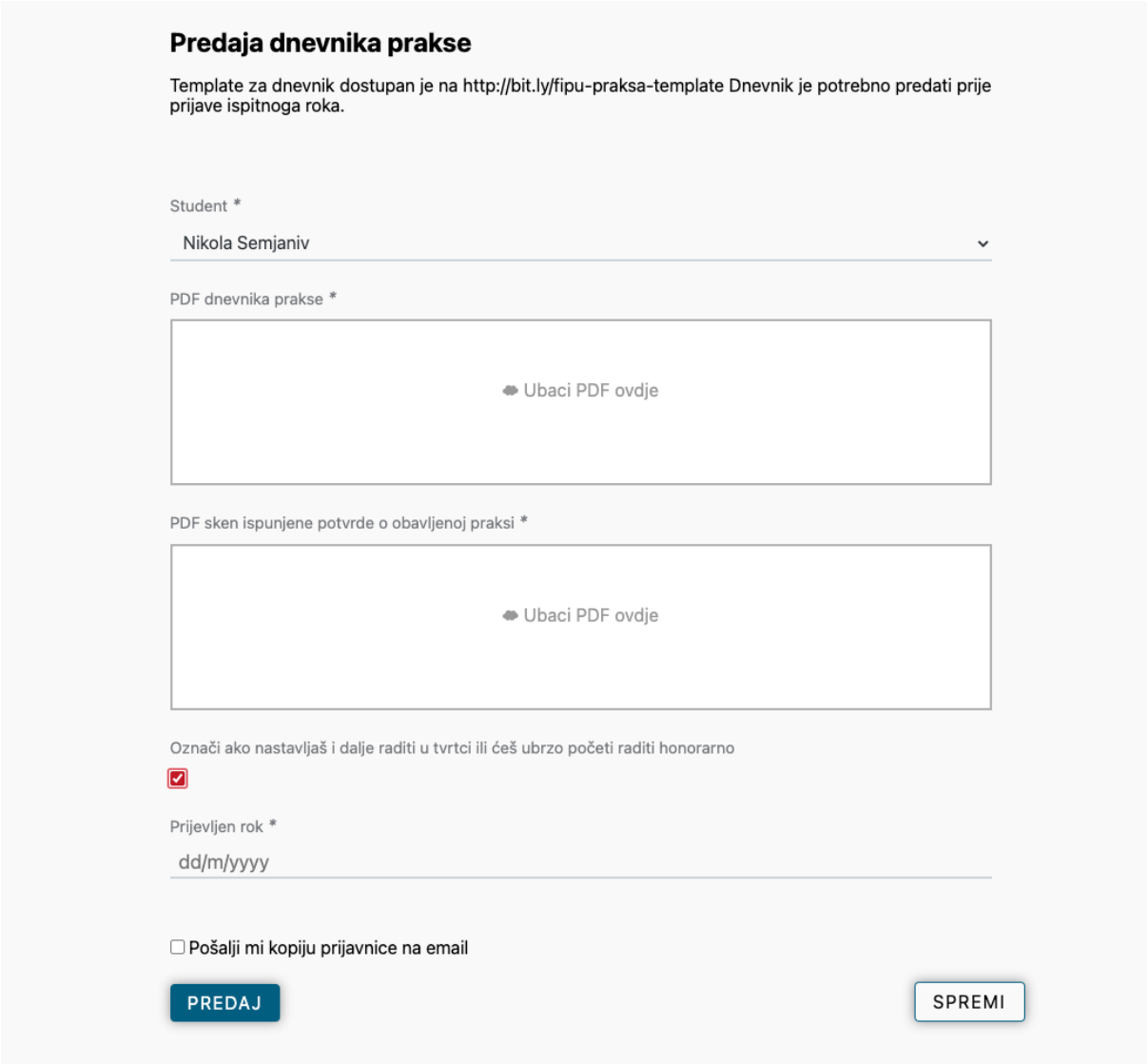
Slika 27. Forma za prijavu prakse



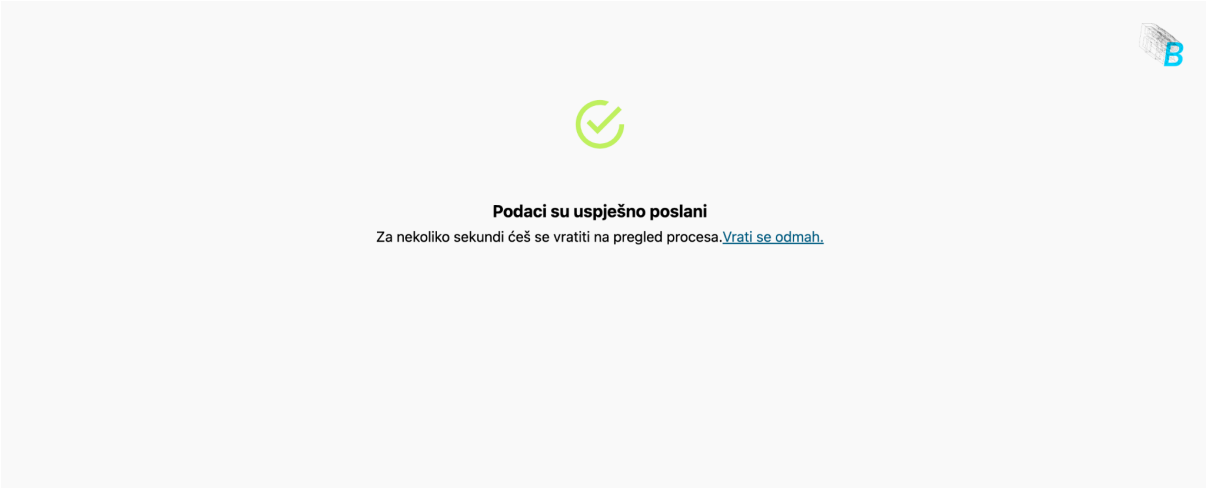
Slika 28. Forma za ispunjavanje prijavnice - 1. dio



Slika 29. Forma za ispunjavanje prijavnice - 2.dio



Slika 30. Forma za predaju dnevnika prakse



Slika 31. Prikaz uspješne predaje forme

6. KORIŠTENE TEHNOLOGIJE

6.1. Vue.js

Vue je vrlo moćna JavaScript biblioteka stvorena za izgradnju interaktivnih korisničkih sučelja, a unatoč tome što ima sposobnost rukovanja velikim aplikacijama od jedne stranice, Vue je također odličan za pružanje okvira za male, pojedinačne slučajeve uporabe te njegova mala veličina datoteke znači da se može integrirati u postojeće ekosustave bez dodavanja previše napuhanosti (Street et al., 2018, p. 7). Bez okvira, završili bismo s neredom neodrživog koda, čija bi se velika većina bavila stvarima koje okvir apstrahira od nas, ali uz Vue, kod koji pruža istu funkcionalnost kao primjerice jQuery, puno je jednostavniji za čitanje i razumijevanje (Macrae, 2018, pp. 1-2). Vue se sve češće koristi u razvoju jednostavnih, ali i složenih web aplikacija. Za Vue se kaže da povezuje najbolje iz Angulara i Reacta. Za razliku od Reacta ili Angulara, jedna od prednosti Vuea je čisti HTML izlaz koji proizvodi, budući da druge Javascript biblioteke obično ostavljaju HTML razbacan s dodatnim atributima i klasama u kodu, dok ih Vue uklanja kako bi proizveo čist, semantički izlaz (Street et al., 2018, p. 7).

Jednostavan je za naučiti za nekoga tko već ima iskustva s *Javascript-om*, ali čak i ako nema, može se brzo uhodati. Posjeduje točno određenu strukturu pisanja tako da nema puno mjesta za greške, no ako se neka jednostavnija i provuče, Vue je dovoljno pametan da ju uhvati. Neke od većih prednosti su njegova mala veličina kao programskog okvira, što mu ujedno daje i na brzini kod učitavanja web stranica, pružajući korisnicima ugodno iskustvo. Osim toga, reaktivnost i dvostrano povezivanje podataka daju mu mnogo na popularnosti. Isto tako, jednostavnost u integriranju u postojeći projekt, kao i kreiranje novog, ne odudaraju od prethodno navedenih pluseva. Sve ove stvari drže Vue pri vrhu ljestvice *frameworka Javascript*.

6.2. Korišteni paketi

6.2.1. NPM

Npm (eng. *Node package manager*) je upravitelj paketima koje koristimo u aplikaciji. Općenito govoreći, dvije primarne odgovornosti upravitelja paketa su instaliranje paketa i upravljanje ovisnostima, a kao brz, sposoban i bezbolan upravitelj paketa, Brown smatra kako je npm velikim dijelom odgovoran za brzi rast i

raznolikost ekosustava Node (Brown, 2019, p. 12). Pomoću njega instaliramo i upravljamo paketima. To je vidljivo kroz package.json datoteku koja je sažeti prikaz informacija aplikacije s korištenim paketima. Na vrhu datoteke imamo uvid u osnovne informacije o aplikaciji (ime, opis, verzija, licenca i autor) (slika 32).

```json
{
  "name": "runbook",
  "description": "University master degree project",
  "version": "1.0.0",
  "license": "MIT",
  "author": "Nikola Semjaniv <nikola.sem450@gmail.com>"
}
```

Slika 32. Osnovne informacije o projektu

Sljedeći objekt sadrži listu skripti s nazivom akcije i načinom izvođenja iste. Tu smo slobodni dodati što god nam je potrebno i važno osigurati za razvoj projekta. Osnovne skripte koje su dodane tijekom kreiranja Vue projekta su *build* - skripta koja transformira aplikacijski kod iz razvojnog procesa u produkcijski kod koji je kompresiran i optimiziran kako bi ga poslužitelj mogao brzo izvesti; i *serve* - koji pokreće aplikaciju u razvojnom modu koji je dosta sporiji tijekom prvog izvođenja (slika 33).

```json
{
  "scripts": {
    "start": "vue-cli-service serve",
    "build": "vue-cli-service build",
    "prepare": "husky install",
    "lint": "npx eslint . --ext .vue --ext .js",
    "lint-fix": "npx eslint . --fix",
    "check-for-errors": "node ./pre-commit.js",
    "remove-nm": "rm -rf node_modules"
  }
}
```

Slika 33. Definirane skripte unutar package.json datoteke

Sljedeće što slijedi je vezano uz pakete koji se koriste u projektu. Svaki paket ima naziv i verziju koja je instalirana. Podijeljeni su u dvije strukture. Jedna se zove *dependencies* te sadrži pakete koji se direktno pozivaju i koriste u kodu same aplikacije. Oni su prijeko potrebni za nesmetan rad aplikacije, dok *devDependencies*, druga struktura, sadrži pakete koji se ne koriste direktno u kodu, već s obzirom na njihovu namjenu su većinom okrenuti prema tome da developerima olakšaju sam razvoj aplikacije (slika 34). Važno je napomenuti da je veliku pozornost potrebno obratiti na verzije paketa, budući da je moguće da se neke verzije paketa ne slažu s određenima ili se trenutno slažu, ali u jednom trenutku u budućnosti neće. Kako se ne bi doveli do situacije gdje se projekt ne može pokrenuti zbog konflikta između paketa, potrebno je izričito označiti verziju koja se trenutno koristi. Primjer je prikazan na slici 34. Tako osiguravamo nesmetano pokretanje projekta na bilo kojem operacijskom sustavu u bilo koje vrijeme.

```
{
  "dependencies": {
    "axios": "0.27.2",
    "core-js": "3.24.1",
    "v-form-builder": "git+https://github.com/markocunj/vue-form-builder.git",
    "vue": "2.7.2",
    "vue-router": "3.2.0",
    "vuex": "3.4.0"
  },
  "devDependencies": {
    "@vue/cli-plugin-babel": "4.5.0",
    "@vue/cli-plugin-router": "4.5.0",
    "@vue/cli-plugin-vuex": "4.5.0",
    "@vue/cli-service": "4.5.0",
    "eslint": "7.32.0",
    "eslint-config-standard": "16.0.3",
    "eslint-plugin-import": "2.25.2",
    "eslint-plugin-node": "11.1.0",
    "eslint-plugin-promise": "5.1.1",
    "eslint-plugin-vue": "9.3.0",
    "husky": "8.0.0",
    "vue-template-compiler": "2.6.11"
  }
}
```

Slika 34. Korišteni paketi

6.2.2. Vuex

Svaka kompleksnija aplikacija koristi *state*, što je zapravo stanje podataka direktno vezani za aplikaciju. Biblioteka Vuex pomaže programerima upravljati stanjem svojih aplikacija, tj. omogućuje jednu centraliziranu pohranu koja se može koristiti u cijeloj svojoj aplikaciji za pohranjivanje i rad s globalnim stanjem te daje mogućnost provjere podataka koji ulaze kako bi se osiguralo da su podaci koji ponovno izlaze predvidljivi i točni (Macrae, 2018, p. 103).

Vuex nam upravo to omogućava - upravljanje stanjem na jednostavan i precizan način dostupan kroz čitavu aplikaciju. Vuex se nadovezuje na jednostavan obrazac pohrane uvođenjem: eksplicitno definiranih getera, mutacija i akcija te integracije s Vue razvojnim alatima za otklanjanje pogrešaka putovanja kroz vrijeme i uvoz/izvoz snimke stanja (Djirdeh et al., 2018, p. 126). Početni izgled glavne Vuex datoteke je prikazan na slici 35.

```
export default new Vuex.Store({
    strict: process.env.NODE_ENV !== "production",
    state: {},
    mutations: {},
    getters: {},
    actions: {},
    modules: {},
});
```

Slika 35. Početno stanje Vuex-a

Vuex možemo zamisliti kao *state management pattern*, drugim riječima predstavlja samostalnu aplikaciju izrađenu po sljedećem uzorku:

- ***State*** označava kako su podaci pohranjeni u *store-u* Vuex: to je kao jedan veliki objekt kojemu možemo pristupiti s bilo kojeg mjesta u našoj aplikaciji - to je jedini izvor istine (Macrae, 2018, p. 106).

- ***Actions*** sadrži metode koje mogu uzrokovati promjene na *state-u* što je reakcija na korisničku interakciju s aplikacijom.
- ***Mutations*** sadrži metode preko kojih se direktno mijenja state. Budući da koristimo striktni mode za Vuex, mutacija na podacima u *store-u* dopuštene su isključivo kroz *mutations* metode. Striktni mode vidljiv je na slici 35. Budući da je uključen svaki put kada se promjena podataka dogodi izvan *store-a*, vue šalje upozorenje u konzolu.
- ***Getters*** je svojstvo preko kojeg bi se trebalo dohvaćati trenutno stanje podataka. To su zapravo metode čija je uloga vratiti točno određeni podatak.

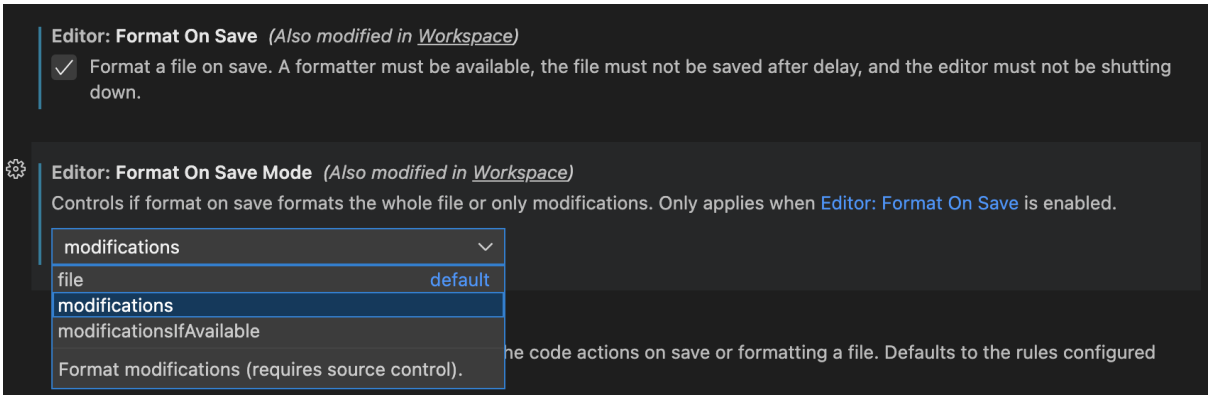
6.2.3. Vue router

Za kreiranje SPA (eng. *Single page application*) potrebno je koristiti biblioteku koja će znati kada i gdje prikazati određenu komponentu koja je kreirana i koja se koristi u aplikaciji. Vue u kombinaciju s vue-router-om nam upravo to i pruža. Vue.js implementira SPA obrazac kroz svoj temeljni dodatak, vue-router te za vue-router, svaka URL ruta odgovara komponenti, a to znači da će se reći vue-routeru kako da se ponaša kada korisnik ode na određeni URL u smislu njegove komponente, tj. svaka komponenta u ovom novom sustavu je stranica u starom sustavu (Street et al., 2018, p. 457). Vue-router shvaća da se ista komponenta koristi između stranica i stoga, umjesto da uništi i stvori novu instancu, ponovno koristi komponentu, no loša strana ovoga je da se podaci ne ažuriraju; mora se pokrenuti funkcija za uključivanje novih podataka o proizvodu, ali dobra strana je što je kod učinkovitiji (ibid. 276). Vrlo jednostavno i uz minimalan kod, može se postaviti upravljanje sustavom navigiranja u aplikaciji. Pisanje ruta se odvija prema uzorku koji je vrlo jednostavan, a opet ima otvorena vrata prema složenim, ali i korisnim opcijama koje se mogu iskoristiti kod kreiranja kompleksnijeg sustava navigacije u aplikaciji koju stvaramo. Bitno je obratiti pažnju na to gdje želimo prikazati komponente iz routera. Kako bi to riješili koristimo *router-view* komponentu (slika 36). Ta komponenta je predložak u koji Vue router ubacuje komponentu s obzirom na trenutni URL.

Slika 36. App.vue - koristi <router-view /> za prikazivanje pojedinih komponenti preko vue-routera

6.2.4. ESLint

Linter ESLint ima ulogu osigurati da pisanje programskog koda za svakog razvojnog inženjera bude isto. Linter pregledava kako je kod napisan kroz određena pravila definirana u konfiguracijskoj datoteci. Ako se napisani kod ne slaže s definiranim pravilima, linter stavlja obavijest na taj dio koda zajedno s opisom pravila koje je prekršeno. ESLint je dizajniran da bude iznimno fleksibilan linter koji se lako može prilagoditi i priključiti te pruža 236 osnovnih pravila, grupiranih u sedam kategorija osmišljenih da pomognu programerima da razumiju njihovu svrhu: *Possible Errors*, *Best Practices*, *Strict Mode*, *Variables*, *Node.js & CommonJS*, *Stylistic Issues* i *ECMAScript 6* (Tómasdóttir et al., 2020, p. 864). Također, koristeći uređivač teksta, npr. Visual studio code, moguće je namjestiti da linter pregleda cijelu datoteku ili samo novododani blok koda i/ili popravi odnosno formatira tekst prema pravilima (slika 37).



Slika 37. Postavke formatiranja izvornog koda u Visual Studio Code-u

6.2.5. Husky

Paket Husky radi u kombinaciji s gitom te omogućava pokretanje skripte s obzirom na git značajke. U projektu je korišten *pre-commit hook* (slika 38) koji se pokreće prije nego se izvrši *commit hook* za spremanje izvornog koda na repozitorij. Skripta pokreće pretprocesor ESLint koji pregledava usklađenost izvornog koda postavljenim pravilima. Zanemaruje jedino sadržaj datoteka navedenih u .eslintignore datoteci (slika 39). Tijekom izvođenja linter radi istu stvar koju radi kada ga pokrećemo na spremanje dokumenta, samo što onda provjera brže traje budući da se provjerava samo jedan dokument i to u nekim slučajevima ne cijeli, već novododani dio izvornog koda. Razlog zašto je husky korišten u projektu jest da unese još jedan sloj sigurnosti kada se kod sprema na repozitorij. U ovom slučaju husky pokreće samo ESLint, međutim omogućava puno više. Pogotovo ga je korisno imati ako na projektu sudjeluje više ljudi koji rade kao stalni razvojni tim ili se mijenjaju. U svakom slučaju, husky je dobar pomoćnik u održavanju čistog i jednoličnog koda između developera.

```
const exec = require("child_process").execSync;

try {
    exec(
        "npx eslint . --ext .vue --ext .js --max-warnings 0",
        { stdio: "inherit" },
    );
    console.warn("ESLint check completed with no errors.");
} catch (error) {
    process.exit(-1);
}
```

Slika 38. Skripta za pokretanje lintera

```
**/*.md
package.json
public/*
src/assets/*
docs/js/*
**/*.ts
```

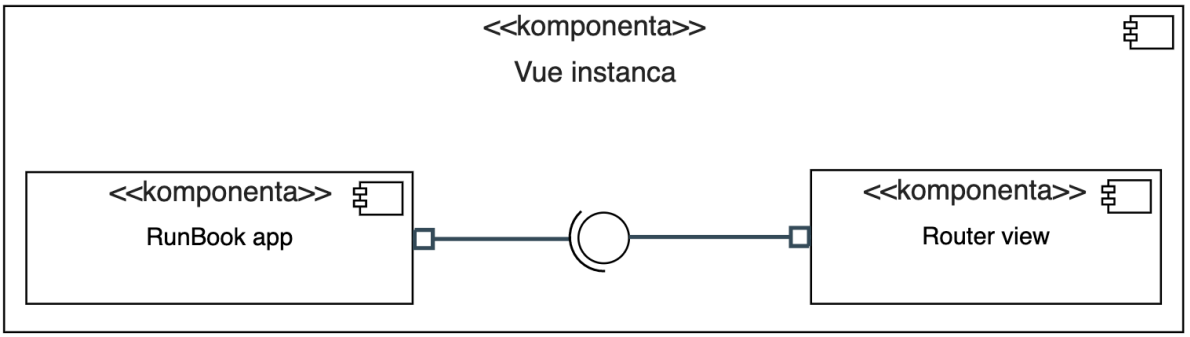
Slika 39. Sadržaj .eslintignore datoteke

7. RAZVOJ APLIKACIJE

7.1. Arhitektura

Aplikacija je strukturirana na način da ne odstupa od standardnog pristupa razvoja aplikacija u okruženju Vue. Aplikacija se sastoji od jedne instance te aplikacije čije se stranice dinamički mijenjaju s obzirom na aktivni URL. Instance u sebi sadrži *app root*, odnosno ulaznu točku aplikacije koja obuhvaća sva ostala obilježja, funkcionalnosti i sad kod koji se nalazi unutar. Vizualno je prikazano na slici 40. Router view radi kao omotač oko glavnih komponenti. U ovom slučaju roditeljskih komponenti kako je prikazano na slici, a same roditeljske komponente sačinjene su od svoje djece što su zapravo nove, manje komponente. Te komponente možemo svrstati u dvije kategorije:

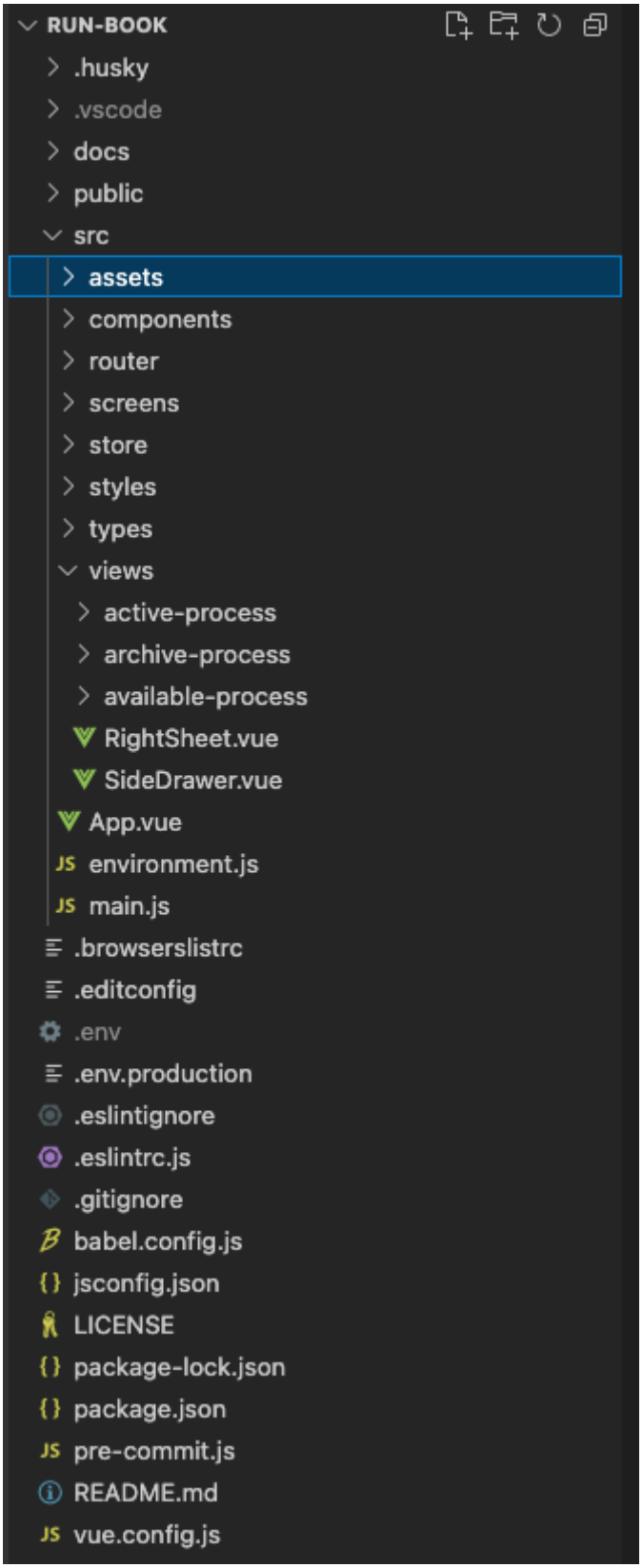
1. Obične komponente - komponente za višekratnu upotrebu (eng. *reusable components*). Na primjer to su gumbi, input polja, nekakvi elementi koji određuju raspored (eng. *layout*) na stranici i slično.
2. *View* komponente - karakteristične vue komponente koje služe kao pregledi, odnosno stranice koje se prikazuju s obzirom na URL. Međutim, važno je za naglasiti da je bitno strukturirati aplikaciju na način kako odgovara pojedincu. Tako da ova aplikacija u ovom slučaju varira od standard.



Slika 40. Standardna arhitektura razvoja vue aplikacije

Što se tiče strukture *foldera* aplikacije (slika 41) bazirana je na standardnom načinu gdje su sve potrebne stvari smještene u *src* mapu koja onda sadrži podmape:

- *assets* - sadrži stvari koje se koriste kod prikazivanja na sučelju ili uređivanje samih elemenata sučelja što su većinom slike, fontovi, videa i slično.
- *components* - sadrži komponente za višekratnu upotrebu što su recimo u ovom slučaju *Icon*, *Loader*, *ContentLoader*, *ModalProvider*, *SearchBar*, *TabBar*, *TabBarItem*, *PaginationWrapper* i ostale komponente vezane za prikaz procesa.
- *views* - slično kao u mapi *components*, međutim ovdje su smještene veće komponente koje se koriste samo jednom u aplikaciji ili su posebno prilagođene za određenu stranicu što znači da može biti više komponenti s istim nazivom, ali drugačijim svojstvima. Inače se savjetuje da se u *views* mapu stavljaju komponente koje predstavljaju stranice u aplikaciji, međutim ovdje je odvojeno radi jednostavnosti.
- *screens* - sadrži vue komponente koje predstavljaju stranice u aplikaciji. Ove komponente koriste i komponente iz *views* i *components* mape, a uključuju se u aplikaciju kroz router.
- *router* - sadrži samo index datoteku u kojoj se nalazi router (slika 43).
- *store* - isto kao i za router, sadrži samo index datoteku u kojoj se kreira sam store.
- *styles* - sadrži *.css* datoteke koje se koriste u više stranica ili komponenti pa su zato odvojene u posebnu datoteku, a uz to su i velike u smislu boja linija tako da ovaj način pruža preglednost u kodu.



Slika 41. Struktura aplikacije

7.2. Struktura ruta

Strukturiranje ruta, odnosno slaganje komponenti na način kako ih želimo prikazati je uvjetovano s dvije strane. Jednu stranu čini UI, odnosno dizajn sučelja na koji se treba paziti, dok drugu stranu čini struktura izvornog koda.

Kod kreiranja takozvanog rutera, popis ruta se može uključiti direktno u objekt koji se kreira (slika 43). Međutim, izvađen je van radi same preglednosti. Ako aplikacija ima tendenciju rasti, poželjno je odmah razdvojiti neke elemente izvornog koda. Sam objekt se sastoji od sljedećih svojstava: *path* - koji definira URL na kojem će se pokazati komponenta i to je obavezno svojstvo zajedno s *component* - definira komponentu te *name*, svojstvo koje nije obavezno, ali ima prednosti kod usmjeravanja. *Props* definirano pod *true*, još je jedno od obaveznih svojstava ukoliko se na tu rutu, odnosno komponentu prosljeđuju podaci. Struktura ruta aplikacije prikazana je na slici 42.

```
const routes = [
  {
        path: "/",
        name: "ActiveListScreen",
        component: ActiveListScreen,
    },
    {
        path: "/available-processes",
        name: "AvailableListScreen",
        component: AvailableListScreen,
    },
    {
        path: "/archive",
        name: "ArchiveListScreen",
        component: ArchiveListScreen,
    },
    {
        path: "/process-detail/:id",
        name: "ProcessDetailScreen",
        component: ProcessDetailScreen,
        props: true,
    },
    {
        path: "/process/action-form/:id",
        name: "FormScreen",
        component: FormScreen,
        props: true,
    },
    {
        path: "/process/action-form/:id/successful",
        name: "SuccessfulFormScreen",
        component: SuccessfulFormScreen,
    }
]
```

Slika 42. Struktura ruta

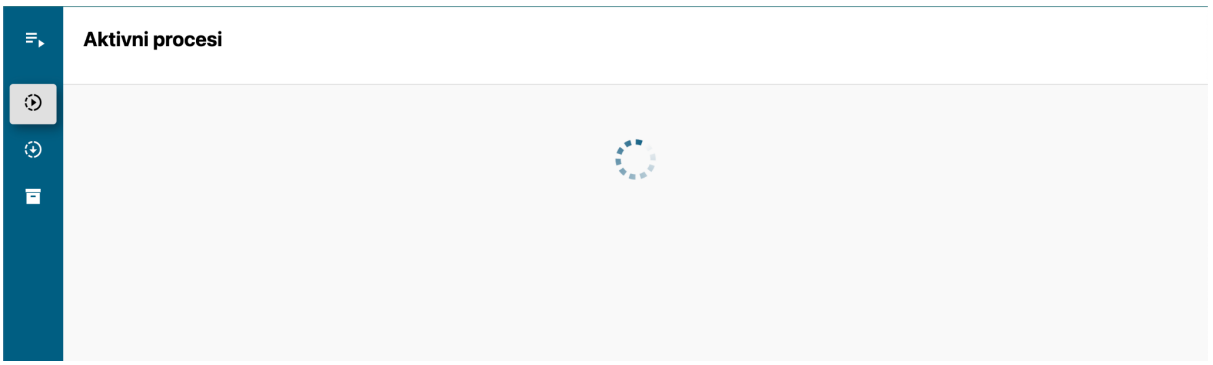
```
1 Vue.use(VueRouter);
2
3 const router = new VueRouter({
4     mode: "history",
5     base: process.env.BASE_URL,
6     routes,
7 });
8
9 export default router;
```

Slika 43. Kreiranje vue-router-a

7.3. Upravljanje stanjem aplikacije

Kada se govori o upravljanju stanjem aplikacije, to se najčešće veže za podatke koji su u nekom procesu obrade pa je potrebno dočarati korisniku kako se nešto unutar aplikacije događa. Bilo to učitavanje, nekakva pretvorba, čekanje na odgovor s poslužitelja, pretraživanje ili filtriranje i slično, bitno je vizualno prikazati da je aplikacija živa i da radi kako je zamišljena. Čak i ako ne radi, bitno je predočiti grešku koja se dogodila na sučelju.

Prikaz učitavanja važan je za pokazati korisniku kako bi znao da se nešto događa u aplikaciji. Slika 44 pokazuje sučelje u trenutku učitavanja, odnosno dohvaćanja podataka s poslužitelja. Sve dok se podaci ne obrade na željeni način, animacija učitavanja, u ovome slučaju *spinner*, bit će prikazana.



Slika 44. Stanje aplikacije tijekom učitavanja podataka

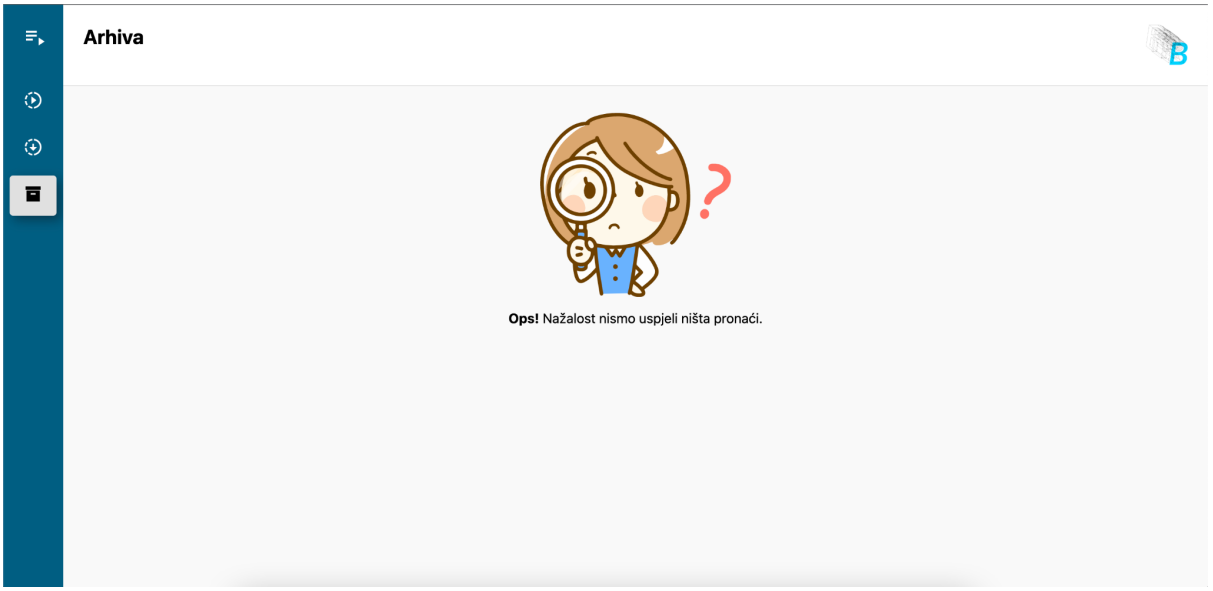
Učitavanje je riješeno kroz komponentu *<component-loader />* prikazanu na slici 45. Komponenta je napravljena za višekratnu upotrebu na način da izvršava zadanu funkciju *readData* prema dobivenim parametrima uz dodatna obilježja ako ih ima. Funkcija se pokreće odmah pri kreiranju komponente i postavlja stanje na učitavanje što prikazuje *spinner*. Kada dobije odgovor s poslužitelja, prikazuje sljedeće stanje s obzirom na dva moguća odgovora: grešku i pozitivan odgovor gdje dobiva podatke. Ako je odgovor pozitivan, podatke šalje roditelju koji rukuje s njima, odnosno prikazuje ih na stranci.

```vue
<template>
    <div v-if="isLoading">
        <div v-if="error" class="error-wrapper">
            <icon name="warning" color="#e81e63" />
            <div class="inner-wrapper">
                <h4>Dogodila se greška</h4>
                <p>{{ error }}</p>
            </div>
        </div>
        <div v-else class="loader-wrapper">
            <div class="loader"></div>
        </div>
    </div>
</template>
...
<script>
  {
    methods: {
        async readData () {
            this.isLoading = true;
            try {
                const response = await this.$store.dispatch(
                    this.path,
                    this.param,
                );
                this.$emit("is-fetched", response);
                this.isLoading = false;
            } catch (error) {
                this.error = error;
            }
        },
    }
  }
</script>
```

Slika 45. ContentLoader.vue komponenta

Kada podaci dođu u roditeljsku komponentu prilikom završetka učitavanja, dva su moguća ishoda:

1. Kada ima podataka - prikazuje se lista s podacima.
2. Kada nema podataka - prikazuje se sučelje sukladno tome. Slika 46 prikazuje kako izgleda sučelje kada je lista prazna. Bitno je računati na ovu situaciju, budući da je puno ljepše i razumljivije kada je prikazan tekst koji kaže da je lista prazna i da nema podataka, a još je jasniji doživljaj kada se može povezati mala, zanimljiva ilustracija s time.



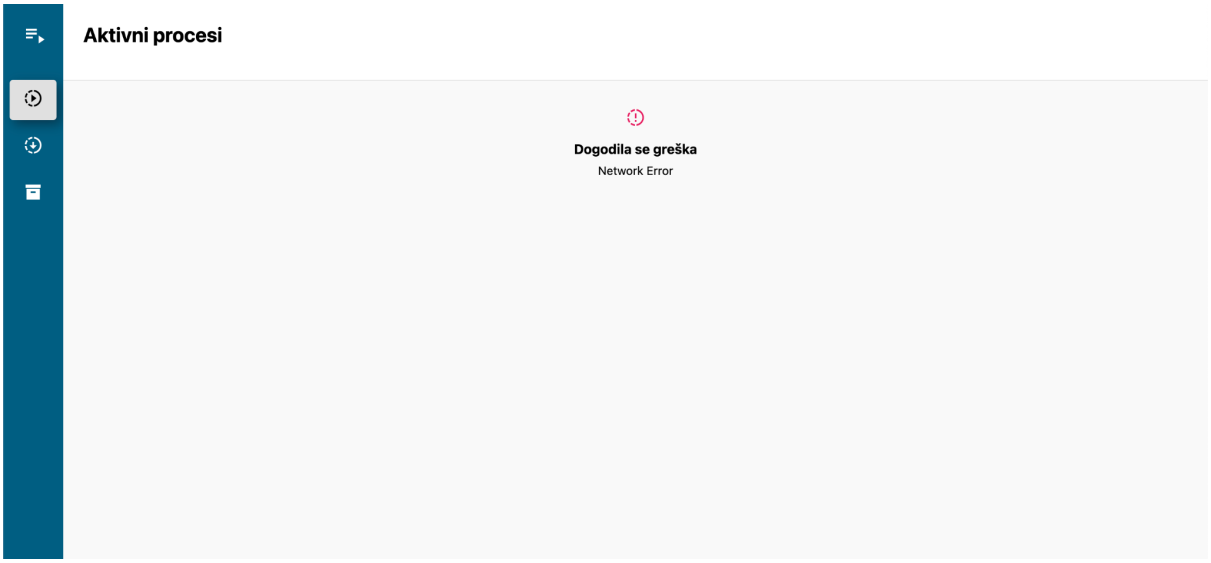
Slika 46. Prikaz sučelja kada je lista stavki prazna

Slika 47 prikazuje kako se koristi komponenta *<content-loader>* u roditeljskoj komponenti. Ima jedno obavezno svojstvo koje joj se treba proslijediti, a to je *path* koji kaže koju funkciju treba pozvati za dohvaćanje podataka, dok je *param* svojstvo koje je opcionalno i ako ga funkcija ne zahtjeva, nije ga potrebno slati. Još jedna stvar koju je potrebno dodati kako bi sve funkcioniralo je *event listener is-fetched* koji sluša određeni događaj i s obzirom na njega, poziva funkciju *fetched* koja postavlja podatke dobivene s poslužitelja u varijablu na sučelju koja je zadužena za čuvanje podataka iz koje se prikazuje lista.

```vue
<template>
    <content-loader
            @is-fetched="fetched"
            path="processStore/readProcessList"
            param="active"
        />
</template>
...
<script>
  {
    methods: {
        fetched (content) {
            if (content.length > 0) {
                this.queryList = content;
            } else {
                this.isListEmpty = true;
            }
        }
    }
  }
</script>
```

Slika 47. Korištenje ContentLoader komponente

Kada se dogodi greška prilikom učitavanja podataka, potrebno ju je prikazati na sukladan način. Slika 48 prikazuje grešku koja se dogodila na sučelju s porukom “Network Error” što asocira korisnika da provjeru internetsku vezu.



Slika 48. Greška kod dohvaćanja podataka

Store je postavljen tako da sadrži jedan glavni (slika 49) i jedan dodatni *store* u kojem se nalaze reaktivni podaci i funkcije vezane uz te podatke. Kada se kaže reaktivni, znači da njihova promjena, ukoliko se ispravno koriste, uvjetuje ponovno prikazivanje, odnosno kreiranje sučelja (eng. *re-rendering*). Konkretno, dodatni *store* je vezan samo uz procese, što znači da je zadužen za dohvaćanje liste aktivnih, dostupnih i arhiviranih procesa te dohvaćanje formi koje je potrebno prikazati kako bi ju korisnik mogao ispuniti. Nadalje, zadužen je za dohvaćanje detalja procesa koji je u pregledu i spremanje istoga, kao i spremanje liste procesa. Slike od 50 do 53 prikazuju komponentu *processStore* koja sadrži akcije i varijable vezane uz procesne podatke.

```
export default new Vuex.Store({
    strict: process.env.NODE_ENV !== "production",
    state: {
        drawerOpen: false,
    },
    mutations: {
        toggleDrawer (state) {
            state.drawerOpen = !state.drawerOpen;
            localStorage.drawerState = state.drawerOpen;
        },
        setDrawerInitState (state, stateValue) {
            state.drawerOpen = stateValue;
        },
    },
    getters: {
        isDrawerOpen (state) {
            return state.drawerOpen;
        },
    },
    actions: {},
    modules: {
        processStore: {
            namespaced: true,
            ...processStore,
        },
    },
});
```

Slika 49. Glavni store

```
export default {
    state: {
        list: [],
        detail: null,
    },
  ...
};

```

Slika 50. Varijable u koje se pohranjuje lista podataka i detalji

```
export default {
    mutations: {
        setList (state, payload) {
            state.list = payload;
        },
        setDetail (state, payload) {
            state.detail = payload;
        },
        setDetailAction (state, payload) {
            const activeStep = state.detail.currentStep;
            state.detail.stepList[activeStep[0]][activeStep[1]].action = payload;
        },
    },
  ...
};

```

Slika 51. Mutacije zadužene za promjenu podataka

```
export default {
    getters: {
        getPreview: (state) => (id) => {
            return state.list.find((item) => item.id === id);
        },
        getDetail (state) {
            return state.detail;
        },
        getStepListForActiveStep (state) {
            const stepList = [[], []];
            const activeStep = state.detail.currentStep;

            stepList[0].push(state.detail.stepList[activeStep[0]][activeStep[1]]);
            stepList[1] = state.detail.stepList[state.detail.currentStep[0] + 1];
            return stepList;
        },
    },
  ...
};

```

Slika 52. Geteri koji vraćaju podatak iz store-a

```
export default {
    actions: {
        readProcessList ({ commit }, payload) {
            return axios.get(`/process?status=${payload}`).then((res) => {

                commit("setList", res.data);
                return res.data;
            }).
                catch((err) => {
                    throw err.message;
                });
        },
        readProcessById ({ commit }, payload) {
            return axios.get(`/process/${payload}`).then((res) => {

                commit("setDetail", res.data);
                return res.data;
            }).
                catch((err) => {
                    throw err.message;
                });
        },
        readActionById (payload) {
            return axios.get(`/process/action/${payload}`).
                catch((err) => {
                    throw err.message;
                });
        },
    },
  ...
};

```

Slika 53. Akcije vezane uz čitanje procesa

ZAKLJUČAK

Kako se organizacije sve više razvijaju, tako raste i potreba za tehnologijama koje će pomoći u tom razvoju. Paralelno s razvojem organizacije raste i ljudski rad koji veže na sebe nove poslove i obveze. Automatizacija procesa je vrlo dobar odgovor na takvo stanje. Nije lako dostižna, ali je vrlo učinkovita i korisna.

To je ono na čemu se radi i čemu se teži na fakultetu. Ova aplikacija jedna je u nizu aplikacija koje će činiti sustav za kreiranje, praćenje, unaprjeđivanje i prezentiranje procesa.

Na tržištu postoji nekoliko rješenja koje imaju u svome okruženju uključene aplikacije za modeliranje, upravljanje i prezentiranje procesima, ali ono što je u ovoj aplikaciji bio glavni cilj je prikazati proces toliko jednostavnim i razumljivim za svakog korisnika. Glavna razlika od aplikacija na tržištu koje imaju riješeno prezentiranje procesa i ove aplikacije je u tome što ova aplikacija prikazuje korake relevantne za korisnika procesa, odnosno ne prikazuje cjelokupan BPMN model kroz lijepi UI, već filtrira korake važne za korisnika.

BPMN, model koji definira proces i pomoću kojeg se točno zna kako se neki proces odvija, u ovoj aplikaciji nema direktnu primjenu, već prolazi BPMN engine koji procesira model u podatke koji će biti prezentirani na sučelju aplikacije. Osim što je aplikacija zadužena za prezentiranje modela, velika korist se izvlači iz toga što ima mogućnost reći korisniku da se od njega očekuje obavljanja zadatak koji može biti u smislu ispunjavanje određene forme kroz aplikaciju ili odrađivanja zadatka preko poveznice na vanjsku aplikaciju. Pored toga, važno je pokazati da se od njega nekad ne očekuje ništa, već da je red na trećoj strani da odradi svoje. To su neke od možda manjih, ali značajnih stvari koje se u drugim tržišnim rješenjima ne mogu naći.

LITERATURA

1. Brown, E. (2019). *Web development with node and express: leveraging the JavaScript stack*. O'Reilly Media.
2. Djirdeh H. Murray N. & Lerner A. (2018). *Fullstack vue : the complete guide to vue.js and friends*. Fullstack.io.
3. Galitz W. O. (2007). *The essential guide to user interface design : an introduction to GUI design principles and techniques*. Wiley Pub.
4. Lauesen, S. (2005). *User interface design: a software engineering perspective*. Pearson Education.
5. Macrae C. (2018). *Vue.js: up and running : building accessible and performant web apps*. O'Reilly Media : O'Reilly Media.
6. npm. *package.json*. [Internet] Dostupno na: https://docs.npmjs.com/cli/v8/configuring-npm/package-json (Pristupljeno 01. rujna.2022.)
7. Street M. Passaglia A. & Halliday P. (2018). *Complete vue.js 2 web development : practical guide to building end-to-end web development solutions with vue.js 2*. Packt Publishing.
8. Tómasdóttir, K., Aniche, M., & van Deursen, A. (2020). The Adoption of JavaScript Linters in Practice: A Case Study on ESLint. *IEEE Transactions on Software Engineering*, *46*(8), 863-891.

POPIS SLIKA

Slika 1. Simboli za događaje 3
Slika 2. Simboli za kontrolu tijeka i dodatni simboli 4
Slika 3. Prikaz aktivnosti i specifikacija zadataka 6
Slika 4. Simboli za određivanje tijeka procesa (Prilagođeno prema: https://www.softwareag.com) 6
Slika 5. Simbol za polja i staze (Prilagođeno prema: http://www.bpmb.de) 7
Slika 6. Simboli za prolaze, odnosno skretnice (Prilagođeno prema: https://www.softwareag.com) 8
Slika 7.Detalji procesa admminskog sučelja 10
Slika 8. Prikaz use case dijagrama za RunBook aplikaciju 13
Slika 9. Primjer sučelja gdje se koristi lista kao prikaz skupa podataka 14
Slika 10. Detalji odabranog elementa s popisa - pregled 15
Slika 11. Prikaz detalja odabranog elementa s popisa - koraci 16
Slika 12. Prikaz detalja odabranog elementa s popisa - koraci 17
Slika 13. Sustav za prijavu prakse 19
Slika 14. Grupacija koraka nakon skretnice - 4a i 4b 20
Slika 15. Format matrice s prozorom za detalje stavke (Izvor: Lauesen, 2005, p. 87) 20
Slika 16. Prikaz detalja matrice 21
Slika 17. Prikaz situacije kada je na korisniku da izvrši određenu radnju 21
Slika 18. Prikaz situacije kada nije na korisniku da izvrši zadatak 22
Slika 19. Prikaz završenog koraka 22
Slika 20. Korak koji je neaktivan 22
Slika 21. BPMN model upisa u višu godinu studija 23
Slika 22. Osnovne informacije prikazane na webu 24
Slika 23. Osnovne informacije prikazane na mobilnom uređaju 24
Slika 24. Pregled procesa 25
Slika 25. Trenutni korak 25
Slika 26. Detalji - prikaz koraka kroz listu 26
Slika 27. Forma za prijavu prakse 27
Slika 28. Forma za ispunjavanje prijavnice - 1. dio 28

Slika 29. Forma za ispunjavanje prijavnice - 2.dio 29
Slika 30. Forma za predaju dnevnika prakse 30
Slika 31. Prikaz uspješne predaje forme 31
Slika 32. Osnovne informacije o projektu 33
Slika 33. Definirane skripte unutar package.json datoteke 33
Slika 34. Korišteni paketi 34
Slika 35. Početno stanje Vuex-a 35
Slika 36. App.vue - koristi <router-view /> za prikazivanje pojedinih komponenti preko vue-routera 37
Slika 37. Postavke formatiranja izvornog koda u Visual Studio Code-u 38
Slika 38. Skripta za pokretanje lintera 39
Slika 39. Sadržaj .eslintignore datoteke 39
Slika 40. Standardna arhitektura razvoja vue aplikacije 40
Slika 41. Struktura aplikacije 42
Slika 42. Struktura ruta 44
Slika 43. Kreiranje vue-router-a 45
Slika 44. Stanje aplikacije tijekom učitavanja podataka 45
Slika 45. ContentLoader.vue komponenta 46
Slika 46. Prikaz sučelja kada je lista stavki prazna 47
Slika 47. Korištenje ContentLoader komponente 50
Slika 48. Greška kod dohvaćanja podataka 50
Slika 49. Glavni store 51
Slika 50. Varijable u koje se pohranjuje lista podataka i detalji 52
Slika 51. Mutacije zadužene za promjenu podataka 52
Slika 52. Geteri koji vraćaju podatak iz store-a 53
Slika 53. Akcije vezane uz čitanje procesa 54

SAŽETAK

Cilj ovog rada je izrada web aplikacije zadužene za jednostavno prikazivanje BPMN modela, odnosno procesa u kojem korisnik sudjeluje ili kojeg želi pokrenuti uz svoje arhivirane procese. Ideja za razvoj aplikacije došla je iz potrebe za prikazom BPMN modela, koji se koristi za definiranje procesa, za korisnika tog procesa. Zasniva se na jednom većem sustavu čija je namjena automatizacija i poboljšanje procesa, a sve je započelo sa sustavom prijave prakse kod studenata informatike.

Aplikacija je izrađena kroz Vue.js 2 te je velika pozornost posvećena samome sučelju kako bi bilo što jednostavnije i intuitivnije prema korisniku, a naročito dio vezan uz prezentiranje BPMN modela. Razvoj je baziran na svojstvu aplikacije da bude skalabilna kako bi se u bližoj budućnosti mogla jednostavno i brzo nastaviti razvijati na postojeći kod.

Ključne riječi: korisničke aktivnosti, Vue.js, BPMN vizualizacija, web aplikacija, dizajniranje sučelja

ABSTRACT

The goal of this thesis is the creation of a web application in charge of simply displaying the BPMN model, i.e. the process in which the actor participates or wants to start with his archived processes. The idea for developing the application came from the need to display the BPMN model, which is used to define a process, for the actor of that process. It is based on one larger system whose purpose is automation and process improvement, which started with the internship application system for computer science students.

The application was created using Vue.js 2, and great attention was paid to the interface itself to make it as simple and intuitive as possible for the user, especially the part related to presenting the BPMN model. The development is based on the property of the application to be scalable so that in the near future it can easily and quickly continue to develop on the existing code.

Keywords: User activities, Vue.js, BPMN visualization, web application, interface design