Web aplikacija za upravljanje službenim putovanjima

Plazonić, Tomislav

Master's thesis / Diplomski rad

2016

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:200:098644

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

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

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

zir.nsk.hr

Image not found or type unknown

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

WEB APLIKACIJA ZA UPRAVLJANJE SLUŽBENIM PUTOVANJIMA

Diplomski rad

Tomislav Plazonić

Osijek, 2016.

Sveučilište Josipa Jurja Strossmayera u Osijeku

ELEKTROTEHNIČKI FAKULTET OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 04.05.2016.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Tomislav Plazonić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo, smjer Procesno računarstvo
Mat. br. studenta, godina upisa:	D-584R, 23.10.2013.
Mentor:	Doc.dr.sc. Josip Job
Sumentor:	
Predsjednik Povjerenstva:	Doc.dr.sc. Ratko Grbić
Član Povjerenstva:	Doc.dr.sc. Marijan Herceg
Naslov diplomskog rada:	Web aplikacija za upravljanje službenim putovanjima
Znanstvena grana rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak diplomskog rada:	Osmisliti i izraditi web aplikaciju koja će omogućiti upravljanje službenim putovanjima unutar poslovnih subjekata.
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 Postignuti rezultati u odnosu na složenost zadatka: 2 Jasnoća pismenog izražavanja: 3 Razina samostalnosti: 3
Datum prijedloga ocjene mentora:	04.05.2016.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

Sveučilište Josipa Jurja Strossmayera u Osijeku

ELEKTROTEHNIČKI FAKULTET OSIJEK

IZJAVA O ORIGINALNOSTIRADA

Osijek,17.05.2016.

Ime i prezime studenta:	Tomislav Plazonić
Studij:	Diplomski sveučilišni studij Računarstvo, smjer Procesno računarstvo
Mat. br. studenta, godina upisa:	D-584R, 23.10.2013.
Ephorus podudaranje [%]:	2

Ovom izjavom izjavljujem da je rad pod nazivom**: Web aplikacija za upravljanje službenim putovanjima**

izrađen pod vodstvom mentora Doc.dr.sc. Josip Job

i sumentora

mojvlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osimonih 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, ašto je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD **1**

1.1. ZADATAK DIPLOMSKOG RADA 1
1.2. KORIŠTENI JEZICI, OKVIRI, PROGRAMI, SUSTAVI I STANDARDI 2
1.2.1. JavaScript *2*
1.2.2. PHP *3*
1.2.3. AngularJS *3*
1.2.4. Laravel *3*
1.2.5. HTML *4*
1.2.6. CSS i Sass *4*
1.2.7. MySQL *5*
1.2.8. PhpStorm *5*
1.2.9. Git i GitHub *5*
1.2.10. Google Chrome *6*
1.2.11. Composer, npm i Bower *6*
1.2.12. JWT *7*

2. STRUKTURA APLIKACIJE **8**

2.1. POZADINSKI SUSTAV (*BACKEND*) 8
2.1.1. Model baze podataka *8*
2.1.2. REST API *10*
2.1.3. Postavke *12*
2.2. PRISTUPNI SUSTAV (*FRONTEND*) 13
2.2.1. Jednostranična aplikacija *14*
2.2.2. Proizvoljne usluge *15*
2.2.3. Moduli *18*
2.2.4. Dizajn *30*

3. KORIŠTENJE APLIKACIJE **32**

3.1. PRVA VRSTA KORISNIKA 32
3.2. DRUGA VRSTA KORISNIKA 39
3.3. TREĆA VRSTA KORISNIKA 40
3.4. ČETVRTA VRSTA KORISNIKA 40
3.5. ADMINISTRATOR 41

4. ZAKLJUČAK **43**

LITERATURA **44**

SAŽETAK **45**

KLJUČNE RIJEČI 45

APPLICATION FOR BUSINESS TRIP MANAGEMENT **45**

KEYWORDS 45

ŽIVOTOPIS **46**

1. UVOD

Računalne aplikacije (aplikacije za osobna računala) omogućuju rješavanje velikog broja problema iz različitih domena. Takve aplikacije možemo podijeliti na dvije osnovne kategorije: izvorne (engl. *native*) i mrežne (engl. *web*). Obje kategorije imaju svoje prednosti i mane.

Glavna prednost izvornih aplikacija je mogućnost izravne komunikacije sa sklopovljem i programskom podrškom računala na kojemu se aplikacija izvodi [1]. Zbog toga izvorne aplikacije omogućuju vrlo visoke performanse pri izvođenju, pogotovo ako se radi o zahtjevnijim rješenjima kao što su CAD (*computer-aided design*, hrv. dizajn potpomognut računalom) alati, video igre ili računalni sustavi koji upravljaju industrijskim postrojenjima. Najveći nedostatak izvornih aplikacija su njihova distribucija i održavanje − svako pojedino računalo zahtjeva vlastitu kopiju aplikacije [1].

S druge strane, mrežne aplikacije takav nedostatak nemaju. Svaka mrežna aplikacija pohranjena je na poslužitelju (engl. *server*). Dakle, u opticaj je stavljena jedna kopija aplikacije pa su distribucija i održavanje mrežnih aplikacija znatno jednostavniji [1]. Uz tu karakteristiku je vezana i glavna prednost mrežnih aplikacija, a to je mogućnost pristupa. Korisnici mrežnoj aplikaciji pristupaju koristeći mrežni preglednik (engl. *web browser*) tako što unesu jedinstvenu adresu koja pokazuje na poslužitelja na kojemu se mrežna aplikacija nalazi. To znači da je za pristup mrežnoj aplikaciji dovoljno računalo koje ima pristup internetu.

Jedan od nedostataka mrežnih aplikacija koji je donedavno bio prisutan je bila potreba za neprestanom komunikacijom s poslužiteljem u svrhu dohvaćanja različitih dijelova aplikacije tijekom njezina korištenja. Rješenje tog problema opisano je krajem 2002. godine [2], a rezultiralo je pojavom jednostraničnih aplikacija (engl. *single-page application* − SPA).

1.1. Zadatak diplomskog rada

Aplikacija za upravljanje službenim putovanjima, čija je izrada bio zadatak ovog diplomskog rada, primjer je jednostranične mrežne aplikacije. Aplikacija za upravljanje službenim putovanjima razvijana je u svrhu potvrde koncepta (engl. *proof of concept*) te ne zadovoljava sve potrebne zahtjeve za legalno korištenje i ne predstavlja alternativu procesu izdavanja putnih naloga.

Zahtjevi za aplikaciju bili su sljedeći:

- pristup aplikaciji trebaju imati četiri različite vrste korisnika i administrator
- prvoj vrsti korisnika treba omogućiti ispunjavanje i slanje zahtjeva za putnim nalogom i ispunjavanje i slanje putnog naloga

- drugoj vrsti korisnika treba omogućiti pregledavanje poslanih zahtjeva za putnim nalogom i poslanih putnih naloga te ocjenjivanje njihove ispravnosti
- trećoj vrsti korisnika treba omogućiti pregledavanje poslanih zahtjeva za putnim nalogom i poslanih putnih naloga te njihovo odobravanje ili odbijanje
- četvrtoj vrsti korisnika treba omogućiti pregledavanje poslanih putnih naloga, ocjenjivanje ispravnosti i uređivanje računa putnog naloga
- administratoru treba omogućiti unošenje novog korisnika u sustav

1.2. Korišteni jezici, okviri, programi, sustavi i standardi

Za razvoj ove aplikacije korišteni su JavaScript (unutar mrežnog programskog okvira AngularJS) i PHP (unutar mrežnog programskog okvira Laravel) programski jezici.

Mrežni programski okvir (engl. *web framework*) je nakupina paketa ili modula koja omogućuje razvoj mrežnih aplikacija bez potrebe upravljanja detaljima niske razine. Mrežni programski okviri dijele se na klijentske (engl. *client-side*), poslužiteljske (engl. *server-side*) i potpune (engl. *full-stack*) [3]. AngularJS primjer je klijentskog, a Laravel poslužiteljskog mrežnog programskog okvira. Korištenjem programskih okvira postiže se konzistentnija i stabilnija aplikacija te brži i tečniji razvoj.

Za definiranje strukture izgleda aplikacije korišten je HTML jezik, a za definiranje stilskog izgleda aplikacije korišteni su CSS i Sass jezici. Za spremanje podataka u bazu podataka korišten je sustav MySQL. Za pisanje i strukturiranje koda korišteno je razvojno okruženje PhpStorm, a za njegovo održavanje korišten je sustav Git u kombinaciji sa servisom GitHub. Za razvoj i testiranje aplikacije korišten je mrežni preglednik Google Chrome. Za dohvaćanje potrebnih paketa korišteni su rukovoditelji za pakete Composer, npm i Bower.

1.2.1. JavaScript

JavaScript je objektno-orijentirani skriptni programski jezik. Unutar odgovarajućeg okruženja (npr. mrežni preglednik) ima mogućnost vezanja na objekte tog okruženja pružajući programsko upravljanje nad njima. JavaScript je standardiziran unutar ECMAScript specifikacije. Sadrži standardnu biblioteku objekata (npr. *Array*, *Date*, *Math*) te niz jezičnih elemenata poput operatora, upravljačkih struktura i izjava (engl. *statement*). Temelji se na prototipima, sadrži prvoklasne funkcije te podržava objektno-orijentirani, imperativni i funkcionalni stil programiranja. JavaScript je moguće proširiti dodavanjem novih vrsta objekata. Proširenja JavaScripta dijele se na klijentski i poslužiteljski JavaScript [4]. JavaScript se pojavio 1995. godine, a razvoj je započeo Brendan Eich. Korištena je inačica 1.7 JavaScript jezika.

1.2.2. PHP

PHP (originalno *Personal Home Page*, a danas *PHP: Hypertext Preprocessor*) je skriptni jezik opće namjene (engl. *general-purpose*) koji se izvodi na poslužitelju. PHP je posebno pogodan za razvoj mrežnih aplikacija. Umjesto pisanja velikog broja naredbi koje za rezultat daju HTML, PHP kod je moguće ugraditi (engl. *embed*) u HTML kod korištenjem posebne početne (*<?php*) i završne (*?>*) procesorske naredbe. Klijent nema mogućnost uvida u PHP kod korišten za stvaranje rezultata, jer se skripta izvrši prije slanja odgovora [5]. PHP podržava imperativni, funkcionalni, objektno-orijentirani, proceduralni i reflektivni stil programiranja. PHP se pojavio 1994. godine, a razvoj je započeo Rasmus Lerdorf. Do 2004. godine jezik je bio razvijan bez službenih specifikacija ili standarda. Korištena je inačica 7.0.3.

1.2.3. AngularJS

AngularJS je strukturalni programski okvir za dinamične mrežne aplikacije. Pruža mogućnost korištenja HTML-a kao jezika za predloške te mogućnost proširenja HTML-ove sintakse u svrhu jasnog i sažetog izražavanja aplikacijskih stavki. AngularJS koristi vezanje podataka (engl. *data binding*) i ubacivanje objekata ovisnosti (engl. *dependency injection*) u svrhu smanjivanja količine koda kojega je potrebno napisati [6]. AngularJS funkcionira tako da čita HTML dokument koji sadrži proizvoljne označne atribute ili oznake. Te oznake zatim interpretira kako bi povezao ulazne i izlazne dijelove dokumenta s modelom kojeg predstavljaju standardne JavaScript varijable. AngularJS se pojavio 2010. godine, a razvoj je započeo Miško Hevery. U okviru ovog diplomskog rada AngularJS se koristio za definiranje upravljačke logike pristupnog (engl. *frontend*) dijela te za razvoj pristupnog dijela u obliku jednostranične aplikacije. Korištena je inačica 1.5.3.

1.2.4. Laravel

Laravel je mrežni programski okvir koji olakšava razvoj mrežnih aplikacija pojednostavljujući učestale zadatke potrebne za izradu većine mrežnih projekata. Neki od tih zadataka su ovjeravanje (engl. *authentication*), usmjeravanje (engl. *routing*), korištenje sesija (engl. *session*) i priručne memorije (engl. *cache*) [7]. Laravel karakterizira modularni sustav za pakete u kombinaciji s upraviteljem objekata ovisnosti, *Eloquent* ORM (*object-relational mapping*, hrv. objekt-relacijsko preslikavanje) koji tablicu baze podataka prikazuje kao klasu čija pojava (engl. *instance*) predstavlja jedan red tablice, graditelj upita prema bazi podataka, pokretač (engl. *engine*) za predloške *Blade*, migracije koje služe za održavanje strukture baze podataka, pokretači koji zadovoljavaju arhitekturu prijenosa predstavljajućeg stanja (engl. *representational state transfer* − REST) čime je omogućeno odvajanje logike za različite vrste primljenih zahtjeva

i drugo. Laravel se pojavio 2011. godine, a razvoj je započeo Taylor Otwell. U okviru ovog diplomskog rada Laravel se koristio za definiranje modela baze podataka te za razvoj REST API (*application programming interface*, hrv. primjensko programsko sučelje) dijela aplikacije. Korištena je inačica 5.2.22.

1.2.5. HTML

HTML (*hypertext markup language*, hrv. jezik za označavanje hiperteksta) je standardni prezentacijski jezik za izradu *web* stranica. Ono semantički opisuje strukturu *web* stranice i nije programski jezik. HTML elementi čine osnovne jedinice HTML jezika i služe za stvaranje strukture dokumenta i umetanje raznih objekata (npr. slika ili audio i video zapisa) ili skripti napisanih u nekim od programskim jezika (npr. u JavaScriptu). Mrežni preglednici pomoću HTML elemenata interpretiraju sadržaj na stranici. HTML se pojavio 1993. godine, a razvoj je započeo Tim Berners-Lee [8].

1.2.6. CSS i Sass

CSS (c*ascading style sheets*, hrv. kaskadni stilski listovi) je stilski jezik koji se koristi za opisivanje prezentacije dokumenta napisanog u nekom prezentacijskom jeziku (npr. u HTML-u). Većinom se koristi za postavljanje vizualnog stila *web* stranica i korisničkih sučelja napisanih u HTML-u, ali se može primijeniti na bilo koji XML (*extensible markup language*, hrv. proširljivi jezik za označavanje) dokument. CSS je dizajniran kako bi se omogućilo odvajanje sadržaja od prezentacije dokumenta, uključujući aspekte poput rasporeda elemenata, boje elemenata ili oblika teksta. U CSS-u se koriste strukture za odabir (engl. *selector*) kojima se definira dio prezentacijskog dokumenta na koji se želi primijeniti određeni stil. CSS se pojavio 1996. godine, a njegov začetnik je Hakon Wium Lie [9]. Za razvoj aplikacije koristila se CSS3 inačica jezika.

Sass (skraćeno od *syntactically awesome style sheets*, hrv. sintaktički zadivljujući stilski listovi) je proširenje CSS-a koje dodaje moć i elegantnost CSS jeziku. Sass omogućuje korištenje varijabli, ugniježđenih pravila, mješavina (*mixin*), uvoza s iste razine (hrv. *inline import*) i drugog. Sass pruža mogućnost korištenja sintakse koja je u potpunosti kompatibilna s CSS jezikom. Također, Sass olakšava organizaciju dugačkih stilskih listova i brzo učitavanje i pokretanje manjih stilskih listova uz pomoć stilske biblioteke Compass [10]. Službena implementacija Sassa napisana je u programskom jeziku Ruby, ali postoje i implementacije u drugim programskim jezicima (npr. PHP, Java). Sass se pojavio 2006. godine, a njegovi začetnici su Hampton Catlin i Natalie Weizenbaum. Korištena je inačica 3.4.21.

1.2.7. MySQL

MySQL je sustav za upravljanje relacijskim bazama podataka (engl. *relational database management system* − RDBMS). SQL dio imena odnosi se na *structured query language* (hrv. strukturni upitni jezik). SQL je najučestaliji standardizirani jezik korišten za pristupanje bazama podataka. Ovisno o programskom okruženju, SQL se unosi direktno, ugrađivanjem u kod drugog jezika ili korištenjem jezično-specifičnih naredbi koje sakrivaju sintaksu SQL-a [11]. MySQL je napisan u programskim jezicima C i C++. Pojavio se 1995. godine, a stvorila ga je tvrtka MySQL AB. Korištena je inačica 5.7.11.

1.2.8. PhpStorm

PhpStorm je integrirano razvojno okruženje (engl. *integrated development environment* − IDE). PhpStorm pruža uređivač koda za PHP, HTML i JavaScript jezike. Uključuje trenutnu analizu koda (engl. *on-the-fly code analysis*), prevenciju grešaka (engl. *error prevention*), automatizirano refaktoriranje (engl. *automated refactoring*) PHP i JavaScript koda, dovršavanje koda (engl. *code completion*), uređivač SQL-a (engl. *SQL editor*) i druge značajke. PhpStorm je izgrađen na IntelliJ IDEA platformi koja je napisana u programskom jeziku Java. Korisnici imaju mogućnost proširivanja IDE-a instaliranjem dodataka ili pisanjem vlastitih dodataka. Sve značajke sestrinskog IDE-a WebStorm su uključene u PhpStorm. PhpStorm se pojavio 2009. godine, a razvila ga je tvrtka JetBrains [12]. Korištena je inačica 9.0.

1.2.9. Git i GitHub

Git je raspodijeljeni sustav za upravljanje inačicama (engl. *version control system* − VCS) dizajniran da upravlja projektima različitih veličina na brz i efikasan način. Git omogućava i potiče korisnika na grananje (engl. *branch*) − stvaranje različitih inačica projekta u svrhu odvajanja provjerenog koda od koda koji tvori nove značajke ili služi eksperimentiranju. Git skoro sve operacije izvršava lokalno, zbog čega ga karakteriziraju brzina i visoke performanse. Također, Git koristi model podataka koji osigurava kriptografski integritet svakog pojedinog dijela projekta [13]. Git se pojavio 2005. godine, a njegov začetnik je Linus Torvalds. Korištena je inačica 2.7.0.

GitHub je mrežno-bazirana usluga za pohranjivanje Git skladišta na poslužitelju. Nudi sve funkcionalnosti Git sustava, ali i neke druge značajke. Jedna od njih je mrežno-bazirano grafičko sučelje koje omogućava i integraciju sa stolnim i prijenosnim uređajima. Također, ono nudi praćenje grešaka, zahtijevanje značajki, rukovođenje zadacima i *wiki* stranice (stranice strukturiranog sadržaja) za svaki projekt. GitHub se pojavio 2008. godine, a njegovi začetnici su Tom Preston-Werner, Chris Wanstrath i PJ Hyett [14].

1.2.10. Google Chrome

Google Chrome je višeplatformski mrežni preglednik kojeg je razvio Google. Pojavio se 2008. godine. Većina Chromeovog izvornog koda je otvorena (engl. *open source*), a javnosti je dostupna u obliku projekta pod nazivom Chromium. Uz mnoštvo značajki, ono pruža i mogućnost pregledavanja elemenata (korištenjem značajke *Inspect Element*) namijenjenu programerima mrežnih aplikacija (engl. *web developer*). Značajka *Inspect Element* omogućuje pregledavanje objektnog modela dokumenta (engl. *document object model*) te time otkrivanje dijelova od kojih je *web* stranica načinjena. Za pokretanje JavaScript koda Google Chrome koristi *V8 JavaScript Engine* koji nudi bolje performanse od ostalih pokretača [15]. Korištena je inačica 49 Google Chrome mrežnog preglednika.

1.2.11. Composer, npm i Bower

Composer, npm i Bower su alati za upravljanje paketima (engl. *package manager*). Alati za upravljanje paketima su skupine programskih alata koji automatiziraju proces instaliranja, nadograđivanja, izmjenjivanja i uklanjanja računalnih programa za operativne sustave na konzistentan način [16].

Composer je alat za upravljanje paketima u PHP programskom jeziku. Pojavio se 2012. godine, a razvili su ga Nils Adermann i Jordi Boggiano. Integralni je dio nekoliko otvorenih PHP projekata, uključujući i Laravel. Datoteka koja sadrži svojstva aplikacije vezana uz Composer jest *composer.json*, a nalazi se u korijenskom direktoriju. U okviru ovog diplomskog rada Composer je korišten za dohvaćanje *laravel* i *jwt-auth* paketa. Korištena je inačica 1.0.

Node Package Manager (npm) je alat za upravljanje paketima za Node.js JavaScript okruženje izvršavanja (engl. *runtime environment*). Pojavio se 2010. godine, a razvio ga je Isaac Z. Schlueter. Datoteka koja sadrži svojstva aplikacije vezana uz npm jest *package.json*, a nalazi se u *public* direktoriju. U okviru ovog diplomskog rada npm je korišten za dohvaćanje nekoliko paketa, a to su: *angular*, *angular-animate*, *angular-aria*, *angular-material*, *angular-messages*, *angular-ui-router*, *font-awesome*, *ng-file-upload*, *restangular*, *satellizer*, *sc-date-time*, *gulp* i *gulp-concat*. Korištena je inačica 2.4.12.

Bower je alat za upravljanje paketima u JavaScript programskom jeziku. Ovisi o Node.js-u i npm-u. Pojavio se 2012. godine, a razvio ga je Jacob Thornton. Datoteka koja sadrži svojstva aplikacije vezana uz Bower jest *bower.json*, a nalazi se u *public* direktoriju. U okviru ovog diplomskog rada Bower je korišten za dohvaćanje *pdfmake* i *signature_pad* paketa. Korištena je inačica 1.7.7.

1.2.12. JWT

JWT (*JSON web token*, hrv. JSON mrežni token) je otvoreni standard koji definira kompaktni i samodovoljni način za sigurno prenošenje informacija između stranaka u JSON (*JavaScript object notation*, hrv. notacija JavaScript objekta) zapisu. Prenesena informacija je pouzdana, jer je digitalno potpisana. Oslanja se na dva standarda koja su također bazirana na JSON zapisu: JWS (*JSON web signature*, hrv. JSON mrežni potpis) i JWE (*JSON web encryption*, hrv. JSON mrežna enkripcija). Sastoji se od tri dijela koja su odvojena točkom. To su zaglavlje (engl. *header*), podaci (engl. *payload*) i potpis (engl. *signature*). Ispod je prikazan primjer JWT-a.


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzY290Y2guaW8iLCJleHAiOj
EzMDA4MTkzODAsIm5hbWUiOiJDaHJpcyBTZXZpbGxlamEiLCJhZG1pbiI6dHJ1ZX0.03f32
9983b86f7d9a9f5fef85305880101d5e302afafa20154d094b229f75773
```


Zaglavlje se tipično sastoji od dva dijela: vrste tokena (to je JWT) i algoritma za prikrivanje (engl. *hashing algorithm*) kao npr. HMAC SHA256 ili RSA. Podaci se sastoji od tvrdnji (engl. *claim*) koje predstavljaju izjave o entitetu (npr. o korisniku) i ostale metapodatke. Zaglavlje i podaci se kodiraju prema *Base64Url* načinu kodiranja. Potpis se stvara primjenom algoritma za prikrivanje navedenog u zaglavlju na skup podataka koji čine kodirano zaglavlje, kodirani podaci i tajna. Potpis se koristi za provjeru identiteta pošiljatelja i osiguranje da je poruka ostala nepromijenjena.

JWT funkcionira tako da se token vraća korisniku nakon uspješnog prijavljivanja u određeni sustav (koji koristi jednu od JWT implementacija). Korisniku se lokalno sprema primljeni token koji se koristi pri slanju zahtjeva upućenim prema određenim rutama ili resursima na poslužitelju. Poslužitelj provjerava ispravnost tokena (koji se nalazi u *Authorization* zaglavlju zahtjeva) te ako je token prisutan i ispravan, korisniku se omogućuje pristup traženoj ruti ili resursu [17].

Za razvoj aplikacije korištena su dva paketa za implementiranje JWT funkcionalnosti. Za stvaranje tokena na pozadinskoj strani korišten je paket *jwt-auth*, a za rukovođenje na pristupnoj strani korišten je paket *satellizer*.

2. STRUKTURA APLIKACIJE

Aplikaciju za upravljanje službenim putovanjima čine dva osnovna dijela: pristupni i pozadinski (engl. *backend*) sustav. Pristupni sustav napisan je u programskom jeziku JavaScript unutar AngularJS programskog okvira, a realiziran je u obliku jednostranične aplikacije. Pozadinski sustav napisan je u programskom jeziku PHP unutar Laravel programskog okvira, a realiziran je u obliku REST API-ja. Strukturu projekta čini pripremljeno okruženje Laravela s pripadajućim objektima ovisnosti (engl. *dependency*) koje se nalaze u direktoriju *vendor*. Nekoliko Laravelovih datoteka je podešeno kako bi se omogućile željene funkcionalnosti potrebne za rad aplikacije. Izvorni kod pristupnog sustava nalazi se u direktoriju *public/app*.

2.1. Pozadinski sustav (*backend*)

Pozadinski sustav se može podijeliti na dvije stavke. Prvu stavku čine klase (engl. *class*) i migracije (engl. *migration*) koje predstavljaju model baze podataka, a drugu stavku čine pozadinske rute (engl. *route*) i upravljači (engl. *controller*) koji predstavljaju pristup aplikaciji i bazi podataka. Druga stavka predstavlja već spomenuti REST API.

2.1.1. Model baze podataka

Model baze podataka kreiran je pomoću migracija (engl. *migration*). U ovom slučaju taj se pojam odnosi na shematske migracije (upravljanje shematskim promjenama baze podataka), a ne na podatkovne migracije (proces prijenosa podataka između različitih spremnika, formata ili računalnih sustava). Svaka pojedina migracija je klasa koja sadrži metode *up* i *down*. Metoda *up* služi za dodavanje novih tablica, kolumni ili indeksa bazi podataka, a metoda *down* za ukidanje operacija ostvarenih metodom *up* [18].

Migracije se nalaze u direktoriju *database/migrations*. Pokretanjem procesa migracije baze podataka kronološkim redom se izvršavaju one migracije koje predstavljaju novo stanje (ovisno o trenutnom stanju) baze podataka. S obzirom da aplikacija tijekom razvoja nije bila ustupljena korisnicima na korištenje (engl. *deployment*), migracije kojima se stvaraju tablice po potrebi su bile uređene (nasuprot stvaranju novih migracija u slučaju da je tablice potrebno izmijeniti). Nakon uređivanja željenih migracija (npr. dodavanje kolumne), proces migracije se izvršavao s opcijom osvježavanja što sve prijašnje promjene ukida (poziva se metoda *down* svake migracije) nakon čega se svaka migracija ponovno izvršava (poziva se metoda *up* svake migracije).

Za razvoj aplikacije stvorene su tri različite migracije. Svaka od tih migracija odnosi se na jednu tablicu baze podataka. Stvorena je tablica za korisnike, zahtjeve za putnim nalogom te za putne naloge. Svakoj od tablica definirane su kolumne i pripadajuća svojstva (npr. smije li kolumna pri

unosu biti bez vrijednosti). Kao primjer prikazan je dio koda za stvaranje tablice u koju se spremaju korisnici.

```
// Iz datoteke 2016_02_29_163353_create_users_table.php
Schema::create('users', function (Blueprint $table) {
  $table->increments('id');
  $table->timestamps();
  $table->unsignedTinyInteger('type');
  $table->string('name', 30);
  $table->string('surname', 30);
  $table->string('email', 50)->unique();
  $table->char('password', 60);
});
```

Za ispunjavanje baze podataka testnim podacima stvorene su *seeder* (hrv. sijačica) klase za modele korisnika i zahtjeva za putnim nalogom. Nalaze se u direktoriju *database/seeds*. Svaka *seeder* klasa sadrži metodu *run* koja se pokreće pozivanjem klase. Klasa *DatabaseSeeder* je također *seeder* klasa u čijoj se *run* metodi pozivaju ostale *seeder* klase, a poziva se pokretanjem procesa *seed*, naredbom u konzoli.

Uz migracije, koje predstavljaju pojedine tablice baze podataka, stvorene su i klase koje definiraju model onoga što se u pojedinu tablicu unosilo, odnosno koje predstavljaju red tablice. Klase modela nalaze se u direktoriju *app*. Za svaku klasu modela definirano je svojstvo *$fillable* koje predstavlja niz kolumni za koje je dozvoljeno dodjeljivanje vrijednosti od strane korisnika. Na taj način je aplikacija zaštićena od neželjenih unosa ili izmjena usmjerenima prema bazi podataka. Kao primjer prikazan je dio koda kojim se definira svojstvo *$fillable* za model korisnika.

```
// Iz datoteke User.php
protected $fillable = [
  'type', 'name', 'surname', 'email', 'password'
];
```

Uz navedeno svojstvo po potrebi se definiraju i druga svojstva ili metode. U slučaju modela korisnika definirane su tri metode. Jedna metoda enkriptira korisnikovu lozinku prije unosa u bazu podataka, a prikazana je ispod. Za enkriptiranje lozinke koristila se metoda *bcrypt* koja je za argument primala nektriptiranu lozinku.

```
// Iz datoteke User.php
public function setPasswordAttribute($password) {
  $this->attributes['password'] = bcrypt($password);
}
```

Druge dvije metode definiraju odnos modela s preostala dva modela (npr. da je uz jednog korisnika vezano više putnih naloga), što olakšava dohvaćanje potrebnih podataka iz baze. Jedna od tih metoda je prikazana kao primjer.

```
// Iz datoteke User.php
public function warrants() {
  return $this->hasMany('App\Warrant');
}
```

Sukladno modelu korisnika definirana su i preostala dva modela. Model koji predstavlja zahtjev za putni nalog (*Request*) i model koji predstavlja putni nalog (*Warrant*) imaju definirano svojstvo *$guarded* koje je obrnuto svojstvu *$fillable* − predstavlja niz kolumni za koje nije dozvoljeno dodjeljivanje vrijednosti od strane korisnika. Uz to, oba modela imaju definiranu metodu koja opisuje njihov odnos s modelom korisnika (npr. pojedini putni nalog vezan je uz isključivo jednog korisnika).

2.1.2. REST API

REST (*representational state transfer*, hrv. prijenos predstavljajućeg stanja) je programski arhitekturni stil *World Wide Weba*. Preciznije, REST je arhitekturni stil koji se sastoji od koordiniranog niza arhitekturnih ograničenja primijenjenih na sastavnice, priključke i podatkovne elemente unutar raspodijeljenog hipermedijskog sustava. Sustavi koji zadovoljavaju REST arhitekturne smjernice u velikoj većini slučajeva za komunikaciju koriste HTTP (*hypertext transfer protocol*, hrv. protokol za prijenos hiperteksta) s pripadajućim vrstama HTTP zahtjeva (npr. GET, POST, PUT, DELETE) koje mrežni preglednici koriste za dohvaćanje *web* stranica s poslužitelja i za slanje podataka poslužitelju. REST sustav sadrži sučelje prema vanjskom sustavu u obliku mrežnih resursa identificiranih pomoću URI-ja (*uniform resource identifier*, hrv. uniformni identifikator resursa) [19], kao npr. *warrants/7*. U tom slučaju GET zahtjev upućen navedenom URI-ju rezultirao bi odgovorom koji bi sadržavao nalog (*warrant*) identificiran brojem 7. Imenom arhitekturnog stila (prijenos predstavljajućeg stanja) želi se pobuditi slika o dobro dizajniranom načinu na koji se mrežna aplikacija ponaša − postojanje skupa povezanih *web* stranica od kojih svaka predstavlja određeno stanje identificirano na već opisani način te postojanje mogućnosti da korisnik s jednog stanja pređe na drugo [19].

API (*application programming interface*, hrv. sučelje za programiranje aplikacija) je skup potprograma (engl. *subroutine*), protokola i alata za izgradnju programske podrške. Najčešće predstavlja biblioteku koja sadrži specifikacije za potprograme, podatkovne strukture, klase i varijable. API može biti namijenjen mrežno-baziranom sustavu, operativnom sustavu ili sustavu baze podataka te pruža olakšanja pri razvoju aplikacija za taj sustav. U kontekstu razvoja

mrežnih aplikacija, API se definira kao skup poruka u obliku HTTP zahtjeva te strukture odgovora za svaki pojedini zahtjev. Odgovor najčešće poprima oblik XML ili JSON zapisa [20].

API koji zadovoljava smjernice REST arhitekturnog stila je REST API (često se koristi i izraz *RESTful* API). Aplikacija za upravljanje službenim putovanjima sadrži REST API, a to je vidljivo iz definiranih pozadinskih ruta i upravljača. Rute su definirane unutar *routes.php* datoteke koja se nalazi u direktoriju *app/Http*, a upravljači se nalaze u direktoriju *app/Http/Controllers*.

Ruta koja je prva definirana je korijenska ruta (*/*). Odgovor na GET zahtjev prema korijenskoj ruti jest datoteka *index.php* koja se nalazi u direktoriju *resources/views*. Ta datoteka sadrži HTML kod kojim se aplikacija inicijalizira. Njome se dohvaćaju sve CSS i JavaScript datoteke potrebne za rad aplikacije. Većina tih datoteka sadrži kod paketa koji se koriste u aplikaciji, a datoteke *all.css* i *all.js* sadrže proizvoljno napisani kod koji tvori pristupni sustav aplikacije. Direktivom *ng-app* u oznaci *body* inicijalizira se AngularJS dio aplikacije (pristupni sustav), točnije AngularJS korijenski modul koji je nazvan *app*. Pristupni sustav opisan je u sljedećem potpoglavlju.

Ostale rute vezane su za metode upravljača. Primanjem određene vrste HTTP zahtjeva prema određenoj ruti poziva se odgovarajuća metoda upravljača koja odrađuje definirane akcije kojima se postiže željena funkcionalnost. Primjerice, za brisanje priloga vezanih uz određeni putni nalog, potrebno je poslati DELETE zahtjev prema ruti *api/v1/warrants/{warrantId}/attachments* (gdje *{warrantId}* predstavlja identifikacijski broj putnog naloga), što rezultira pozivanjem metode *deleteAttachments* koju sadrži *WarrantsController*. To je prikazano sljedećim dijelom koda.

```
// Iz datoteke routes.php
Route::group(['prefix' => 'api/v1'], function() {
  Route::group(['middleware' => 'jwt.auth'], function() {
    Route::group(['prefix' => 'warrants'], function() {
      Route::delete(
        '{warrantId}/attachments',
        'WarrantsController@deleteAttachments'
      );
    });
  )};
});
```

Prefiks *api/v1* koristi se zbog toga što je uobičajena praksa da rute koje pristupaju API-ju započinju na takav način. Drugi dio prefiksa odnosi se na inačicu API-ja. Slično su definirane i

rute za rad s drugim modelima. Na primjer, rute s prefiksom *requests* odnose se na rad s modelom zahtjeva za putnim nalogom (*Request*) i pozivaju metode upravljača *RequestsController*.

Iz prethodno prikazanog dijela koda vidljivo je i korištenje *jwt-auth* paketa kao posredničke programske podrške (engl. *middleware*). Posrednička programska podrška predstavlja mehanizam za filtriranje pristiglih zahtjeva [21]. Pristup rutama koje se nalaze unutar grupe s definiranom *jwt.auth* posredničkom programskom podrškom bit će onemogućen ako pristigli zahtjev ne sadrži ispravan token (JWT). Stvaranje i slanje tokena korisniku definirano je u upravljaču *AuthController*. Slanjem POST zahtjeva prema ruti *api/v1/auth* poziva se metoda *authenticate* upravljača *AuthController*. Metoda *authenticate* provjerava podatke koje je korisnik unio za prijavljivanje u sustav (email adresu i lozinku). U slučaju da su uneseni podaci neispravni ili u slučaju da tijekom operacije dođe do greške, odgovor na zahtjev će biti greška (engl. *error*). Suprotno, ako su uneseni podaci ispravni, stvorit će se token (JWT) koji će odgovorom biti poslan korisniku.

2.1.3. Postavke

Datoteka *.env* koja se nalazi u korijenskom direktoriju sadrži popis definiranih konstanti koje postanu globalno dostupne unutar PHP-ove varijable *$_ENV* nakon što aplikacija primi zahtjev. Neke od važnijih konstanti su: *APP_ENV* koja sadrži vrijednost okruženja u kojemu se aplikacija izvodi (postavljena u 'local'), *APP_DEBUG* koja sadrži vrijednost stanja rada za uklanjanje grešaka (postavljena u *true*), *APP_KEY* koja sadrži vrijednost ključa aplikacije koji mora biti nasumičan niz 32 znaka, *DB_HOST* koja sadrži vrijednost adrese lokacije baze podataka, *DB_DATABASE* koja sadrži vrijednost imena baze podataka, *DB_USERNAME* koja sadrži vrijednost korisničkog imena kojim se pristupa bazi podataka, *DB_PASSWORD* koja sadrži vrijednost lozinke za pristup bazi podataka, i *JWT_SECRET* koja sadrži vrijednost nasumičnog niza znakova koja se koristi za potpisivanje JWT tokena.

U direktoriju *config* nalaze se datoteke koje služe za uređivanje postavki (engl. *configuration*) bitnih za funkcioniranje pozadinskog dijela aplikacije. Bit će navedene samo one datoteke čije su postavke bile uređivane.

Najvažnija od tih datoteka je *app.php* koja sadrži postavke za neka općenita svojstva za rad aplikacije. Neka od njih su: *env* koja definira okruženje (engl. *environment*) u kojem se aplikacija izvodi što može utjecati na postavljanje raznih usluga koje aplikacija koristi (postavljena vrijednost je 'local'), *debug* (hrv. uklanjanje grešaka) koja definira stanje rada za uklanjanje grešaka te, ako je takav rad omogućen, opis grešaka je opsežniji (postavljena vrijednost je *true*),

url koja definira aplikacijski URL (*uniform resource locator*, hrv. uniformni lokator resursa) te joj vrijednost mora biti korijen aplikacije (postavljena vrijednost je 'http://localhost'), *timezone* (hrv. vremenska zona) koja definira zadanu vremensku zonu aplikacije (postavljena vrijednost je 'Europe/Zagreb'), *providers* koja definira pružatelje usluga koji će automatski biti učitani pri početku rada aplikacije (uz postavljene pružatelje usluga, dodan je i pružatelj usluge JWT Auth), *aliases* koja definira pseudonime za klase koje se koriste u aplikaciji (uz postavljene pseudonime, dodana su dva pseudonima za klase iz *jwt-auth* paketa: 'JWTAuth' i 'JWTFactory'), i druge.

Datoteka *database.php* služi za podešavanje rada s bazom podataka. U ovoj datoteci definirano je nekoliko vrsta pristupa bazi podataka za različite sustave baze podataka. Aplikacija za upravljanje službenim putovanjima koristi MySQL sustav za bazu podataka pa je stoga i vrijednost varijable *default* ove datoteke postavljena u 'mysql'. Definicija 'mysql' vrste pristupa bazi podataka sadrži nekoliko svojstava s definiranim vrijednostima. Bitno je izdvojiti sljedeća svojstva: *driver* koje definira upravljački program (sustav) baze podataka, *host* koje definira adresu lokacije baze podataka, *database* koje definira ime baze podataka, *username* koje definira korisničko ime kojim se pristupa bazi podataka, *password* koje definira lozinku za pristup bazi podataka, *charset* koje definira vrstu niza znakova (engl. *character set*) koja se koristi za spremanje podataka u bazu (postavljeno u 'utf8'), i *collation* koje definira pravila za uspoređivanje znakova u određenom nizu znakova (postavljeno u 'utf8_unicode_ci').

Datoteka *jwt.php* služi za podešavanje svojstava implementacije JWT-a na pozadinskoj strani (paket *jwt-auth*). Neka od tih svojstava su: *ttl* (*time to live*, hrv. vrijeme života) koje definira vrijeme trajanja ispravnosti tokena (u minutama, zadana vrijednost je 60), *algo* koje definira algoritam prikrivanja koji se koristiti za potpisivanje tokena (zadana vrijednost je 'HS256') te *user* koje definira model korisnika koji je potreban za ispravno funkcioniranje *jwt-auth* paketa.

2.2. Pristupni sustav (*frontend*)

Pristupni sustav strukturiran je prema funkcionalnosti što znači da se dijelovi aplikacije koji skupa čine neku funkcionalnu cjelinu nalaze unutar zasebnog direktorija. Korijenski direktorij pristupnog sustava jest direktorij *public/app* te se u nastavku potpoglavlja sve putanje navode relativno prema korijenskom direktoriju (osim onda kada je očito da to nije slučaj). Dva direktorija koja predstavljaju funkcionalne cjeline pristupnog sustava su *login* i *main*, a unutar *main* direktorija se nalaze ostale funkcionalne cjeline. Direktorij *icons* sadrži ikone korištene za pristupni sustav aplikacije, a direktorij *services* definirane usluge pristupnog sustava. Unutar

direktorija *build* su datoteke *app.js* i *app.css*. Datoteka *app.js* sadrži spoj svih JavaScript datoteka pristupnog sustava, a datoteka *app.css* CSS kôd pristupnog sustava.

Za spajanje svih proizvoljno napisanih JavaScript datoteka u jednu korišten je sustav *gulp* zajedno s priključkom *gulp-concat*. Sustav *gulp* razvijen je u svrhu automatiziranja određenih zadataka potrebnih za razvoj aplikacije. Primjeri takvih zadataka su umanjivanje datoteka izvornog koda (engl. *minification*) i spajanje više datoteka u jednu (engl. *concatenation*). Datoteka koja definira svojstva korištenja sustava *gulp* jest *gulpfile.js*, a nalazi se u direktoriju *public/node_modules*. Definiran je niz *jsSrc* koji predstavlja skup putanja proizvoljno napisanih JavaScript datoteka. Uz to, definirana su i dva zadatka (engl. *task*) za sustav *gulp*. Jedan od njih je zadatak *concat* koji spaja definirane datoteke u jednu i čiji je kod prikazan ispod.

```
// Iz datoteke gulpfile.js
gulp.task('concat', function() {
  gulp
    .src(jsSrc)
    .pipe(concat('app.js'))
    .pipe(gulp.dest('app/build'));
});
```

To se odrađuje pozivanjem nekoliko metoda: *src* (definiranje niza putanja datoteka izvornog koda), *pipe* (korištenje izlaza prethodne metode za ulaz nove metode), *concat* (metoda spajanja koja je definirana priključkom *gulp-concat*, a za parametar zahtjeva naziv odredišne datoteke) i *dest* (definira željeni odredišni direktorij datoteke korištenjem parametra kojeg zahtjeva). Drugi zadatak je zadatak *default* koji predstavlja osnovni zadatak koji se poziva pokretanjem sustava *gulp*, a definiran je da nadgleda datoteke niza *jsSrc* i u slučaju promjene izvrši zadatak *concat*.

2.2.1. Jednostranična aplikacija

Pristupni dio aplikacije za upravljanje službenim putovanjima razvijen je u obliku jednostranične aplikacije. Jednostranične aplikacije su mrežne aplikacije koje prikazuju jednu stranicu koja se dinamički ažurira ovisno o akcijama koje poduzima korisnik [22]. Rezultat takvog pristupa izrađivanju mrežnih aplikacija je rasterećenost poslužitelja i tečnije izvođenje aplikacije.

Olakšani razvoj jednostranične aplikacije pruža paket *angular-ui-router* koji omogućuva definiranje različitih stanja aplikacije, njihovih svojstava i izvođenje prijelaza između tih stanja. Stanje se definira pozivanjem metode *state* usluge *$stateProvider* (dio modula *ui.router*) koja za prvi argument prima ime stanja, a za drugi argument objekt s definiranim svojstvima stanja, a to su: *url* (putanja, odnosno adresa stanja), *templateUrl* (putanja do predloška koje stanje koristi) i *controller* (upravljač kojeg stanje koristi). Predložak koji se koristi za pojedino stanje prikazuje

se unutar roditeljskog predloška unutar *div* oznake s definiranim atributom *ui-view*. Neka stanja sadrže i svojstvo razrješavanja (engl. *resolve*) kojemu je pridružen objekt sa svojstvima koja sadrže obećanja (engl. *promise*). Ta obećanja se u ovoj aplikaciji odnose na dohvaćanje vrijednosti iz baze podataka. Stanje koje sadrži svojstvo razrješavanja omogućeno je tek nakon što se sva obećanja svojstva razrješavanja razriješe. Ispod je kao primjer prikazan dio koda kojim se definira stanje.

```
// Iz datoteke main.config.js
$stateProvider
  .state('main', {
    abstract: true,
    url: '/main',
    templateUrl: 'app/main/main.html',
    controller: 'MainCtrl as main',
    resolve: {
      user: function(apiService) {
        return apiService.getUser();
      }
    }
  })
```

Stanje je moguće definirati i kao apstraktno (pridruživanjem vrijednosti *true* svojstvu *abstract*). U apstraktno stanje prijelaz nije moguć. Ono služi kao roditeljsko stanje drugim stanjima i tipično se koristi u slučaju kada se više stanja samo djelomično razlikuju. Na taj način je kod "suši" (engl. *don't repeat yourself* − DRY), jer je izbjegnuto ponavljanje identičnog koda kojeg bi inače svako pojedino stanje sadržavalo. Apstraktno stanje korišteno je na nekoliko mjesta u aplikaciji: za definiranje osnovnog stanja (*main*) koje sadrži zaglavlje (engl. *header*) aplikacije, za definiranje stanja koje prikazuje zahtjeve za putni nalog (*requests*) i za definiranje stanja koje prikazuje putne naloge (*warrants*).

2.2.2. Proizvoljne usluge

Usluge unutar AngularJS okvira su zamjenjivi objekti koji se međusobno povezuju pomoću ubacivanja objekata ovisnosti. Koriste se za organiziranje i dijeljenje koda unutar aplikacije. Usluge su "pojedinci" (engl. *singleton*) − svaka stavka koja ovisi o usluzi dobiva referencu na jednu te istu pojavu usluge. Usluge karakterizira "lijeno stvaranje" (engl. *lazy instantiation*) − stvaraju se samo ako neka aplikacijska stavka ovisi o njima [23]. Stvaraju su pozivanjem metode *factory* koju sadrži objekt koji predstavlja stvoreni modul. Metoda *factory* prima dva argumenta. Prvi argument je ime usluge, a drugi argument je funkcija koja vraća vrijednost usluge, odnosno

ono što čini uslugu nakon što je ubačena kao objekt ovisnosti. U aplikaciji za upravljanje službenim putovanjima proizvoljno je definirano pet usluga od kojih je svaka predstavljena objektom kojim je izolirana određena funkcionalnost. Nalaze se u direktoriju *services*.

Usluga *apiService* predstavlja objekt koji sadrži metode za pristupanje API-ju. Ono je sučelje prema pozadinskoj strani i čini produžetak API-ja na pristupnog strani. Za olakšano i čitljivije pristupanje API-ju korištena je usluga *Restangular* (dio modula *restangular*). Usluga *Restangular* olakšava stvaranje potrebnih putanja prema API-ju i slanje potrebnih zahtjeva prema tim putanjama. Metode usluge *Restangular* kojima se šalju zahtjevi (npr. metode *post*, *getList*, *doPUT*) vraćaju obećanje koje u nekim slučajevima, nakon što je uspješno razriješeno, pruža podatke dohvaćene od pozadinske strane. Ispod je kao primjer prikazan kod metode *createRequest* koja šalje POST zahtjev s predanim podacima (*newRequest*) prema ruti *api/v1/requests*. Uspješnim razrješavanjem obećanja prikazuje obavijest te se aktivno stanje mijenja u *main.requests.sent*.

```
// Iz datoteke apiService.js
function createRequest(newRequest) {
  Restangular.all('requests').post(newRequest).then(function() {
    toastService.show("Zahtjev stvoren!");
    $state.go('main.requests.sent');
  }, function() {
    toastService.show("Greška tijekom stvaranja zahtjeva!", 3000);
  });
}
```

Svaka metoda objekta kojeg predstavlja usluga *apiService* ima odgovarajuću metodu upravljača na pozadinskoj strani.

Usluga *dialogService* predstavlja objekt koji sadrži metode za dobivanje objekata potrebnih za prikazivanje različitih dijaloga. Dijalozi se prikazuju pozivanjem metode *show* usluge *$mdDialog* koja zahtjeva objekt s definiranim postavkama dijaloga. Svaka metoda usluge *dialogService* vraća jedan od takvih objekata koji se međusobno razlikuju s obzirom na vrstu dijaloga koji se želi prikazati. Svojstva koja svaki od objekata sadrži su: *controller* (upravljač kojeg dijalog koristi), *templateUrl* (putanja do predloška kojeg dijalog koristi), *parent* (roditeljski element dijalog), *targetEvent* (događaj koji definira akciju koja uzrokuje stvaranje dijaloga), *clickOutsideToClose* (definira mogućnost zatvaranja dijaloga klikom izvan predloška dijaloga) i *locals* (objekt koji sadrži varijable dijaloga koje mu se predaju kao objekti ovisnosti, po funkcionalnosti jednak svojstvu razrješavanja koje sadrži objekt za stvaranje stanja). Nekim

objektima se predaje i skup postojećih varijabli aplikacije (*scope*) definiranjem svojstava *scope* i *preserveScope*.

Usluga *documentService* predstavlja objekt koji sadrži metodu *getDocument* koja na temelju predanih podataka vraća objekt potreban za stvaranje dokumenta u formatu PDF. Takav objekt, koji sadrži definirane postavke i sadržaj dokumenta, predaje se metodi *createPdf* objekta *pdfMake* (dio paketa *pdfmake*) kojom se stvara PDF dokument. Metoda *getDocument* ovisno o predanim podacima vraća objekt koji predstavlja zahtjev za putnim nalogom ili putni nalog.

Usluga *helperService* predstavlja objekt koji sadrži nekoliko metoda čije su se funkcionalnosti izolirane u svrhu preglednosti koda ili su učestalo bile potrebne tijekom razvoja aplikacije. Metoda *formatDate* korištena je za proizvoljno formatiranje datuma. Zahtjeva parametre *timestamp* (vremenska oznaka) i *format* (način formatiranja) te koristi uslugu *$filter*. Metoda *getDuration* korištena je za dobivanje trajanja putovanja u tekstualnom obliku, a zahtjeva parametre *start* i *end* koji predstavljaju vremenske oznake početka i kraja putovanja. Metoda *formatTransportation* korištena je za formatiranje ispisa korištenih vrsta prijevoznih sredstava, a za parametar zahtjeva niz korištenih vrsta prijevoznih sredstava. Metoda *getDurationDays* korištena je za izračunavanje trajanja putovanja, a zahtjeva jednake parametre kao i metoda *getDuration*. Metode *getNumberOfRoutes* i *getNumberOfOther* korištene su za dobivanje broja relacija i broja ostalih putnih troškova unesenih u putni nalog, a za parametar zahtjevaju objekt koji sadrži svojstva putnog naloga. Metode *getFileExtension* i *isFileExtensionValid* korištene su za dobivanje ekstenzije datoteke i njihove prihvatljivosti, a za parametar zahtjevaju ime datoteke. Metode *areFileExtensionsValid* i *isFilesArrayUnderMaxSize* korištene su za provjeravanje prihvatljivosti niza datoteka (njihovih ekstenzija i ukupne veličine), a za parametar zahtjevaju niz objekata sa svojstvima imena i veličine datoteke.

Usluga *toastService* predstavlja objekt koji sadrži metodu *show* čijim se pozivanjem korisniku prikazuje poruka obavijesti (*toast*), a zahtjeva parametre *text* (tekst poruke) i *duration* (trajanje prikazivanja poruke). Kod metode *show* prikazan je ispod.

```
// Iz datoteke toastService.js
function show(text, duration) {
  $mdToast.show(
    $mdToast
      .simple()
      .textContent(text)
      .position('bottom right')
      .capsule(true)
      .hideDelay(duration == undefined ? 2000 : duration)
```

```
    );
  }
```

Metoda *show* koristi istoimenu metodu usluge *$mdToast* za prikazivanje poruke obavijesti, a za parametar zahtjeva objekt sa svojstvima za stvaranje poruke. Taj objekt stvara se pomoću metode *simple* koja vraća objekt koji predstavlja jednostavni oblik poruke obavijesti te koji sadrži metode za definiranje ostalih svojstava. Te metode se pozivaju u nastavku i definiraju sadržaj teksta poruke (metoda *textContent*), poziciju poruke (metoda *position*), oblikovanje poruke (metoda *capsule*) i trajanje prikazivanja poruke (metoda *hideDelay*).

2.2.3. Moduli

Glavna stavka pristupnog sustava su AngularJS moduli. Moduli se stvaraju pozivanjem metode *module* globalno-dostupnog objekta *angular* koja zahtjeva dva parametra: ime i niz modula o kojima ovisi. Unutar pojedinog modula moguće je definirati usluge (pozivanjem metode *factory*), upravljače (pozivanjem metode *controller*), direktive (pozivanjem metode *directive*), filtere (pozivanjem metode *filter*) i postavke (pozivanjem metode *config*). Svaka pojedina stavka modula definirana je u zasebnoj datoteci. Takav pristup poboljšava strukturu aplikacije, a kôd unutar pojedinih datoteka čini čitljivijim. Osnova modula definirana je u datotekama s nastavkom *.module.js*, postavke modula u datotekama s nastavkom *.config.js*, upravljači modula u datotekama s nastavkom *.ctrl.js*, a usluge u datotekama s nastavkom *Service.js*.

Modul *app* koji se nalazi u korijenskom direktoriju pristupnog sustava jest korijenski modul, odnosno modul na koji se nadovezuju ostali moduli. Učitavanjem modula *app* učitavaju se i svi ostali moduli koji su modulu *app* predani ubacivanjem objekata ovisnosti, a to su moduli dohvaćenih paketa i moduli *login* i *main* koji predstavljaju zasebne funkcionalne cjeline aplikacije. Ispod je prikazana definicija modula *app*, a u nastavku su ukratko opisani moduli dohvaćenih paketa.

```
// Iz datoteke app.module.js
angular
  .module('app', [
    'ui.router', 'restangular', 'ngMaterial', 'ngMessages',
    'satellizer', 'scDateTime', 'ngFileUpload', 'login', 'main'
  ]);
```

Modul *ui.router* je modul paketa *angular-ui-router*, a sadrži programski okvir za usmjeravanje jednostraničnih aplikacija razvijenih u AngularJS-u [24]. Modul *restangular* je modul istoimenog paketa, a sadrži uslugu koja pojednostavljuje učestale GET, POST, PUT i DELETE zahtjeve te koja je pogodna za aplikacije koje manipuliraju podacima korištenjem REST API-ja

[25]. Modul *restangular* ovisi o modulu *lodash* koji se također koristio za razvijanje aplikacije. Modul *lodash* je modul istoimenog paketa koji predstavlja knjižnicu za JavaScript koja olakšava rad s nizovima, brojevima, objektima i drugime [26]. Modul *ngMaterial* je modul paketa *angular-material*, a sadrži niz elemenata za korisničko sučelje (direktive, klase, usluge i drugo) koji implementiraju Material Design (Googleova specifikacija za ujedinjeni vizualni, pokretni i interaktivni dizajn) specifikaciju i koji su razvijeni za korištenje unutar AngularJS jednostranične aplikacije [27]. Svaka direktiva, usluga ili klasa čije ime počinje prefiksom "md" dio je modula *ngMaterial*. Modul *ngMessages* je modul paketa *angular-messages*, a sadrži poboljšanu podršku za prikazivanje poruka unutar predložaka (tipično unutar formi) [28] koja je potrebna za korištenje modula *ngMaterial*. Modul *satellizer* je modul istoimenog paketa koji služi za ovjeravanje bazirano na tokenima (engl. *token-based authentication*) i sadrži podršku za Google, Facebook, LinkedIn i druge pružatelje, ali i za ovjeravanje pomoću email adrese i lozinke [29]. Modul *scDateTime* je modul paketa *sc-date-time* koji sadrži elemente koji služe za odabiranje datuma i vremena i koji su realizirani poštivanjem Material Design specifikacije. Modul *ngFileUpload* je modul paketa *ng-file-upload*, a sadrži direktivu za slanje datoteka poslužitelju (engl. *upload*).

U datoteci s postavkama modula *app* definirana su svojstva nekih modula o kojima modul *app* ovisi. Definirane su objektne varijable *scDateTimeI18n* i *scDateTimeConfig* kojima se definiraju svojstva za rad *scDateTime* modula. Varijabla *scDateTimeI18n* sadrži svojstva koja definiraju lokalizirane nazive koji se koriste unutar *scDateTime* modula. U istu svrhu u aplikaciju je uključena (u kodu datoteke *index.php*) i datoteka *angular-locale_hr-hr.js* koja se nalazi u direktoriju *app/public/node_modules/angular/i18n*. Internacionalizacija (i18n) je proces razvijanja proizvoda na takav način da ga je moguće prilagoditi različitim jezicima i kulturama. U smislu razvoja aplikacije, internacionalizacija predstavlja apstrahiranje svih nizova znakova (engl. *string*) i ostalih lokalno-specifičnih dijelova aplikacije (npr. formata datuma i valute) [30]. Unatoč omogućenim i18n postavkama, *scDateTime* modul je neznatno izmjenjen na nekoliko mjesta kako bi se ispravilo formatiranje datuma. Uz navedene varijable, definirana je i funkcija *configure* u kojoj se postavljaju osnovna tema, boje i ikone (za modul *ngMaterial*), putanja za ovlaštenje tijekom prijave korisnika u aplikaciju (za modul *satellizer*), osnovna putanja prema API-ju (za modul *restangular*) te putanja na koju se aplikacija preusmjerava u slučaju onemogućenog pristupa željenoj putanji (za modul *ui.router*).

Unutar modula *app* definirane su i sve proizvoljne usluge koje su korištene za razvoj aplikacije. Svaka usluga definirana je u zasebnoj datoteci, a datoteke se nalaze u direktoriju *services*. Usluge su opisane u zasebnom odjeljku.

Modul *login* i njegovi funkcionalni dijelovi nalaze se u istoimenom direktoriju, a predstavljaju dio aplikacije za prijavljivanje korisnika (engl. *sign in*) u sustav. Modul *login* nema modula o kojima ovisi, što je vidljivo u datoteci gdje je modul definiran (*login.module.js*). U datoteci s postavkama za modul *login* definirano je istoimeno stanje aplikacije. Unutar modula *login* definiran je i upravljač naziva *LoginCtrl* koji sadrži logiku pristupne strane (preciznije, metodu *login*) potrebnu za prijavljivanje korisnika u sustav. Prijavljivanje u sustav odvija se pozivanjem metode *login* usluge *$auth* (dio modula *satellizer*) koja za parametar zahtjeva objekt koji sadrži svojstva *email* i *password* kojima su pridružene vrijednosti koje je korisnik unio u formu. Ista metoda vraća obećanje kojim se, u slučaju uspješnog razrješavanja, trenutno stanje mijenja u *main.home* korištenjem metode *go* usluge *$state* (dio modula *ui.router*), a u slučaju neuspješnog razrješavanja prikazuje se obavijest korisniku korištenjem metode *show* usluge *toastService*. Za pridruživanje vrijednosti elemenata forme svojstvima upravljača korištena je direktiva *ng-model* koja za parametar zahtjeva ime varijable u koju se vrijednost želi spremiti. Za obavještavanje korisnika o neispravnom unosu vrijednosti korištene su direktive *ng-messages* i *ng-message*. Direktiva *ng-messages* za parametar zahtjeva vrijednost kojom se sugerira da je unos neispravan, a direktiva *ng-message* vrijednost koja definira atribut elementa unosa koji predstavlja potencijalni razlog neispravnosti. Primjer koda koji predstavlja element korišten za unos prikazan je ispod.

```
<!-- Iz datoteke login.html -->
<md-input-container>
  <label>Email</label>
  <input type="text" name="email" ng-model="login.email" required>
  <div ng-messages="loginForm.email.$error">
    <div ng-message="required">Potrebno!</div>
  </div>
</md-input-container>
```

Vrijednost *loginForm.email.$error* predstavlja ispravnost elementa unosa imena *email* forme *loginForm*. U slučaju kršenja definiranih ograničenja predstavljenih atributima elementa unosa (u ovome slučaju jedini takav atribut je atribut *required* (hrv. potrebno)) svojstvo *$error* (hrv. greška) poprima vrijednost *true*. Tada se pregledavaju elementi koji sadrže direktivu *ng-message*, a prikazuje se samo onaj koji za direktivu *ng-message* ima definiranu vrijednost prekršenog ograničenja. Ako je barem jednom elementu unosa svojstvo *$error* istinito, također će biti istinito i svojstvo *$invalid* (hrv. neispravno) forme unutar koje se nalazi taj element unosa. Svaki element unosa (npr. element definiran oznakom *input* ili *textarea*) zajedno s ostalim pripadajućim elementima stavljen je unutar direktive *md-input-container* koja definira

konzistentan izgled i ponašanje elemenata unosa . Za potvrđivanje unosa definirano je dugme stvoreno direktivom *md-button* koje, među ostalim, ima definirane atribute *ng-click* i *ng-disabled*.

```
<!-- Iz datoteke login.html -->
<md-button class="md-fab md-mini md-raised"
  ng-click="login.login()" ng-disabled="loginForm.$invalid"
  aria-label="Prijavi se">
  <md-tooltip>Prijavi se</md-tooltip>
  <md-icon md-svg-src="check"></md-icon>
</md-button>
```

Atribut *ng-click* predstavlja direktivu kojom se definira akcija koja se izvršava klikom na element (u ovome slučaju to je pozivanje metode *login* upravljača *LoginCtrl* čije je skraćeno ime *login*), a atribut *ng-disabled* predstavlja direktivu kojom se element onesposobljuje (engl. *disabled*) kada je definirani uvjet istinit (u ovome slučaju kada je vrijednost *loginForm.$invalid* istinita). Unutar elementa definiranog direktivom *md-button* definirana su još dva elementa. Direktiva *md-tooltip* stvara tekst koji se pojavljuje kada korisnik mišem lebdi nad (engl. *hover on*) elementom te može sadržavati atribut *md-direction* koji definira stranu pojavljivanja teksta relativno prema roditeljskom elementu. Direktiva *md-icon* prikazuje ikonu čije se definirano skraćeno ime ili apsolutna putanja postavlja atributom *md-svg-src*.

Modul *main* je modul u kojemu su definirani središnji funkcionalni dijelovi aplikacije. Ono kao objekte ovisnosti prima module *requests* i *warrants*. U datoteci s postavkama za modul *main* definirana su stanja ovog modula. Datoteke upravljača i predložaka pojedinih stanja nalaze se unutar zasebnih direktorija, a iznimka je direktorij *main/dialogs* unutar kojeg su definirani upravljači i predlošci korišteni za pojedine dijaloge.

Stanje *main* je roditeljsko apstraktno stanje na koje se nadovezuju ostala stanja modula, ali i stanja modula *requests* i *warrants*. Pri aktiviranju stanja *main* dohvaćaju se podaci trenutno prijavljenog korisnika te se pridružuju varijabli *user*. Stanje *main* definira zaglavlje aplikacije koje sadrži elemente za prijelaz u druga stanja koja su dostupna korisniku i za odjavljivanje iz aplikacije. Upravljač stanja *main* sadrži niz objekata koji predstavljaju sva stanja glavnog dijela aplikacije (*allStates*). Svaki od tih objekata ima definirana sljedeća svojstva: *name* (ime stanja), *label* (tekst koji opisuje stanje), *icon* (ikona stanja), *type* (niz vrsta korisnika koji imaju pristup stanju). Većina tih objekata ima definirano i svojstvo *character* (slovo koje opisuje stanje). Niz *allStates* korišten je za definiranje niza *userStates* koji sadrži samo ona stanja kojima trenutno prijavljeni korisnik ima pristup. Definiranje niza *userStates* prikazano je u kodu ispod.

```
// Iz datoteke main.ctrl.js
vm.userStates = _.filter(vm.allStates, function(state) {
  return _.includes(state.type, user.type);
});
```

Niz *userStates* definiran je korištenjem metode *filter* objekta _ kojoj su proslijeđena dva argumenta: niz *allStates* i funkcija koja za svaki objekt niza *allStates* vraća istinitu ili lažnu vrijednost (*true* ili *false*). Vraćena vrijednost ovisi o tome je li vrijednost vrste trenutno prijavljenog korisnika (svojstvo *type* objekta *user*) sadržana u nizu objekta koji se trenutno ispituje (svojstvo *type* objekta *state*). Za tu provjeru korištena je funkcija *includes* objekta _ kojoj su za argumente prosljeđeni spomenuta vrijednost i spomenuti niz. Niz *userStates* koristi se u predlošku stanja *main* za uvrštavanje korisnikovih stanja u predložak (prikazano u kodu ispod).

```
<!-- Iz datoteke main.html -->
<div ng-repeat="state in main.userStates">
  <md-button class="md-fab md-mini md-raised"
    ng-click="main.goToState(state.name)"
    ng-disabled="state.name | isState" aria-label="{{ state.label }}">
    <md-tooltip>{{ state.label }}</md-tooltip>
    <span ng-if="state.character">{{ state.character }}</span>
    <md-icon md-svg-src="{{ state.icon }}"></md-icon>
  </md-button>
</div>
```

Korisnikova stanja uvrštavaju se u predložak pomoću direktive *ng-repeat* koja prolazi kroz niz *userStates* te za svako objekt toga niza stvara gore prikazani kod. Jedan od elemenata koji se stvara je i dugme koje, među ostalim, ima definirane atribute *ng-click* (pozivanje metode *goToState* upravljača *MainCtrl*) i *ng-disabled* (onesposobljavanje elementa u slučaju kad je stanje koje element predstavlja aktivno, za što je korišten filter *isState* modula *ui.router*). Uz spomenuto, upravljač stanja *main* sadrži metodu *logout* (koja odjavljuje korisnika iz aplikacije) i metodu *goToState* (koja obavlja prijelaz na stanje koje joj je predano kao argument). Jedan od elemenata definiranih unutar elementa *md-button* je element definiran oznakom *span* koji sadrži atribut *ng-if*. Atribut *ng-if* sadrži logički izraz koji, ako je točan, uklanja pripadajući element iz DOM-a. Predložak stanja *main*, kao i neki drugi predlošci, sadrži element s atributom *ng-cloak*. Atribut *ng-cloak* sprječava prikazivanje pripadajućeg elementa sve dok AngularJS aplikacija nije u potpunosti inicijalizirana, a koristi se zbog toga što postoji mogućnost da se za za vrijeme učitavanja aplikacije umjesto željene vrijednosti prikaže izvorni kod.

Stanja modula *main* koja se nadovezuju na nj su: *main.home*, *main.new-request*, *main.new-user* i *main.pending-warrants*.

Stanje *main.home* je početno stanje koje se aktivira nakon prijavljivanja korisnika u aplikaciju. Pristup ovom stanju ima svaka vrsta korisnika. Ono korisniku prikazuje broj neobrađenih, odobrenih, odbijenih te ukupan broj zahtjeva i putnih naloga. Prikazane brojke se razlikuju ovisno o vrsti korisnika, odnosno o njihovoj nadležnosti unutar aplikacije. Zahtjevi i putni nalozi dohvaćaju se pozivanjem metode *getRequests*, odnosno *getWarrants* usluge *apiService*. Nakon uspješnog dohvaćanja zahtjeva i putnih naloga vrši se njihovo filtriranje (korištenjem metode *filter* objekta _) i pridruživanje svojstvima upravljača koja se koriste za prikaz brojki unutar predloška.

Stanje *main.new-requests* je stanje kojemu pristup ima samo prva vrsta korisnika, a služi za stvaranje novog zahtjeva za putni nalog. Predložak ovog stanja sadrži formu s elementima za unos potrebnih podataka koji čine zahtjev. Elementi za odabiranje ponuđenih opcija (engl. *select*) definirani su direktivom *md-select* koja uključivanjem atributa *multiple* omogućuje odabir više od jedne opcije. Opcija za odabir definira se direktivom *md-option*. Neki elementi za unos sadrže i atribute definirane direktivama *ng-maxlength* i *ng-minlength* koji predstavljaju ograničenja za maksimalnu, odnosno minimalnu duljinu unosa. Kod nekih elemenata prisutan je i atribut *readonly* koji onemogućuje uređivanje unosa pripadajućeg elementa. Elementi za unos brojeva imaju definirane sljedeće atribute: *min* (minimalna moguća vrijednost unosa), *max* (maksimalna moguća vrijednost unosa) i *step* (korak za smanjivanje ili povećavanje vrijednosti). Uz to, predložak stanja *main.new-requests* sadrži i dugmeta za brisanje vrijednosti svih elemenata za unos te za prikaz dokumenta u formatu PDF. Upravljač stanja *main.new-requests* (*NewRequestCtrl*) ima definirane četiri metode: *showDateTimeDialog*, *showDocumentDialog*, *sign* i *clear*.

Metoda *showDateTimeDialog* služi za pozivanje dijaloga koji pruža mogućnost odabira datuma i vremena. Prije pozivanja dijaloga u metodi se definiraju svojstva korištena za ispravno postavljanje spomenutog dijaloga. To su: *label* (oznaka dijaloga), *property* (svojstvo upravljača kojemu se pridružuje odabrana vrijednost datuma i vremena), *mindate* (najraniji datum kojega je moguće odabrati) i *maxdate* (najkasniji datum kojega je moguće odabrati). Upravljač i predložak dijaloga za unos datuma i vremena nalaze se u direktoriju *main/dialogs/date-time-dialog*. Predložak svakog dijaloga sadrži glavni element definiran direktivom *md-dialog* unutar kojega su definirani ostali elementi, među kojima su obvezni oni definirani direktivama *md-toolbar* i *md-dialog-content*. Direktivom *md-toolbar* definira se zaglavlje dijaloga, a direktivom *md-dialog-content* sadržaj dijaloga. Za omogućavanje odabira datuma i vremena korištena je direktiva *time-date-picker* modula *scDateTime*.

```
<!-- Iz datoteke date-time-dialog.html -->
<time-date-picker
  ng-model="dateTimeDialog.value"
  on-save="dateTimeDialog.save($value)"
  on-cancel="dateTimeDialog.cancel()"
  mindate="{{ dateTimeDialog.mindate }}"
  maxdate="{{ dateTimeDialog.maxdate }}">
</time-date-picker>
```

Direktiva *time-date-picker*, među ostalim, pruža mogućnost definiranja atributa *on-save* i *on-cancel*. Atribut *on-save* služi za definiranje akcije koja se izvršava klikom na dugme za spremanje odabranog datuma i vremena, a atribut *on-cancel* za definiranje akcije koja se izvršava klikom na dugme za poništavanje. U ovom slučaju, to su akcije pozivanja metoda *save* i *cancel* upravljača *DateTimeDialogCtrl* koje postavljaju, odnosno poništavaju svojstvo upravljača *NewRequestCtrl* definirano u lokalnoj varijabli *property*. Metoda *cancel* spomenuto svojstvo postavlja u *null*, a metoda *save* u odabranu vrijednost datuma i vremena koja se prethodno formatira na potreban način. Metoda *save* uz spomenuto svojstvo postavlja i svojstvo koje u imenu sadrži sufiks 'Raw' (hrv. sirovo) i to u odabranu neformatiranu vrijednost datuma i vremena. Obje metode na kraju izvršavanja pozivaju metodu *hide* istog upravljača koja pozivanjem istoimene metode usluge *$mdDialog* sakriva trenutno prikazani dijalog.

Metoda *showDocumentDialog* upravljača *NewRequestCtrl* služi za pozivanje dijaloga koji pruža mogućnost pregledavanja dokumenta u formatu PDF na temelju vrijednosti koje su trenutno unesene u elementima forme. Prije pozivanja dijaloga u metodi se definiraju svojstva potrebna za stvaranje dokumenta unutar dijaloga. Upravljač i predložak dijaloga za prikaz dokumenta nalaze se u direktoriju *main/dialogs/document-dialog*. Predložak sadrži oznaku *iframe* unutar koje se dokument prikazuje i dugme za slanje prikazanog dokumenta. Upravljač *DocumentDialogCtrl* sadrži metodu *send* koja definira svojstva potrebna za spremanje dokumenta u bazu podataka te ta svojstva predaje metodama usluge *apiService* kojima se spremanje izvršava. Uz to, upravljač postavlja dokument unutar oznake *iframe*, što je prikazano sljedećim dijelom koda.

```
// Iz datoteke document-dialog.ctrl.js
$document.ready(function() {
  var doc = documentService.getDocument(data);
  pdfMake
    .createPdf(doc)
    .getDataUrl(function(url) {
      var iframe = angular.element(
        document.querySelector('.document-dialog iframe'
```

```
        ));
        iframe.attr('src', url);
      });
  });
```

Metodi *ready* usluge *$document* predana je funkcija koja izvršava postavljanje dokumenta. Metoda *ready* poziva se u trenutku kada DOM postane u potpunosti spreman te kada je sigurno da je oznaka *iframe* omogućena za dohvaćanje iz DOM-a. Za stvaranje PDF dokumenta i dobivanje njegove putanje korištene su metode *createPdf* i *getDataUrl*. Metoda *createPdf* (metoda objekta *pdfMake* koji je dio paketa *pdfmake*) za parametar zahtjeva objekt koji sadrži svojstva koja dokument opisuju (sadržaj i formatiranje), a za dohvaćanje tog objekta korištena je metoda *getDocument* usluge *documentService* kojoj su unutar objekta *data* predane vrijednosti unesene u formu. Metoda *createPdf* vraća objekt stvorenog PDF dokumenta čija metoda *getDataUrl* ustupljuje njegovu putanju. Putanja dokumenta sadržana je u varijabli *url* funkcije predane kao argument metodi *getDataUrl*. Unutar te funkcije dohvaća se oznaka *iframe* iz DOM-a korištenjem metode *querySelector* globalno dostupnog objekta *document* kojoj je kao argument predan niz znakova koji predstavljaju *CSS selector* koji jednoznačno određuje element unutar DOM-a. Dohvaćeni element (oznaka *iframe*) kao argument se predaje metodi *element* globalno dostupnog objekta *angular* čija se vraćena vrijednost pridružuje varijabli *iframe*. Ta metoda predstavlja funkciju *jQuery* (ako je biblioteka *jQuery* dostupna), odnosno *jqLite* (ako biblioteka *jQuery* nije dostupna, što je ovdje slučaj). Biblioteka *jqLite* je uključena u AngularJS i predstavlja osiromašeni primjerak biblioteke *jQuery* koja, međuostalim, služi za dohvaćanje i manipuliranje elementima koji čine DOM. Metodom *attr* je elementu (oznaci) *iframe* postavljen atribut *src* (skraćeno za *source*, hrv. izvor) na vrijednost varijable *url*, čime je dokument čija je putanja sadržana u varijabli *url* postavljen za sadržaj oznake *iframe* u DOM-u.

Metoda *sign* upravljača *NewRequestCtrl* služi za pozivanje dijaloga koji pruža mogućnost stvaranja potpisa korištenjem miša. Upravljač i predložak tog dijaloga nalaze se u direktoriju *main/dialogs/signature-dialog*. Predložak sadrži oznaku *canvas* unutar koje korisnik mišem iscrtava vlastiti potpis. Uz to, u predlošku su prisutna dugmeta za brisanje i za potvrđivanje unesenog. Upravljač *SignatureDialogCtrl* tijekom inicijalizacije postavlja potrebno okruženje, što je prikazano kodom ispod.

```
// Iz datoteke signature-dialog.ctrl.js
$document.ready(function() {
  var canvas = document.querySelector('canvas');
  signaturePad = new SignaturePad(canvas, {
    minWidth: 0.4,
```

```
      maxWidth: 1.0,
      onEnd: onEnd
    });
    confirmButton = angular.element(document.querySelector(
      'button[ng-click="signatureDialog.confirm()"]'
    ));
    confirmButton.attr('disabled', 'true');
  });
```

Kao i kod prethodno opisanog dijaloga koristi se metoda *ready* usluge *$document*, jer je potrebno da DOM bude spreman prije dohvaćanja elemenata. Dohvaća se element oznake *canvas* te se pridružuje istoimenoj varijabli. Zatim se varijabli *signaturePad* pridružuje novi objekt klase *SignaturePad* (dio paketa *signature_pad*), čijem se konstruktoru kao prvi argument predaje varijabla *canvas*, a kao drugi argument objekt sa svojstvima koja definiraju način funkcioniranja *SignaturePad* objekta. Svojstvo *minWidth* definira minimalnu širinu, a svojstvo *maxWidth* maksimalnu širinu linije potpisivanja. Svojstvo *onEnd* definira funkciju koja se poziva nakon završetka ispisivanja linije. Funkcija *onEnd* osposobljuje dugme za potvrđivanje (uklanjanjem atributa *disabled*) ako podloga za unos potpisa nije prazna. Sljedeće se dohvaća element dugma za potvrđivanje i pridružuje ga se varijabli *confirmButton*, a zatim se to dugme onesposobljuje postavljanjem njegovog atributa *disabled* u *true*. Tijekom izvršavanja funkcionalnosti ovog dijaloga dugme za potvrđivanje će biti onesposobljeno ako potpis nije unesen, čime se sprječava potvrđivanje prazne bijele pozadine bez potpisa. Upravljač *SignatureDialogCtrl* ima definirane metode *clear* i *confirm*. Pozivanjem metode *clear* briše se uneseni potpis i dugme za potvrđivanje se onesposobljuje. Metoda *confirm* služi za potvrđivanje unesenog potpisa te ovisno o vrsti korisnika i vrsti dokumenta sprema potpis na željenu lokaciju pozivanjem metode *toDataUrl* objekta *signaturePad*. Ako je potpis unesen od strane prve vrste korisnika, ono će biti spremljeno u svojstvo *applicantSignature* relevantnog upravljača (*NewRequestCtrl* ili *PendingWarrantsCtrl*). Ako je potpis unesen od strane neke druge vrste korisnika, ono će (zajedno s ostalim potrebnim podacima) kao argument biti prosljeđeno metodama *updateRequest* ili *updateWarrant* usluge *apiService* kojima se ažurira zahtjev, odnosno putni nalog. Metoda *clear* upravljača *NewRequestCtrl* sve vrijednosti koje se tiču korisnikova unosa postavlja u *null*, odnosno briše korisnikov unos u formi i potpis.

Stanje *main.new-user* je stanje kojem pristup ima samo administrator, a služi za unos novih korisnika u sustav. Predložak ovog stanja sadrži formu za unos potrebnih podataka o korisniku i dugmeta za brisanje trenutno unesenih podataka i za potvrđivanje, odnosno stvaranje novog korisnika. Metoda *createUser* upravljača *NewUserCtrl* poziva se klikom na dugme za

potvrđivanje, a sadrži poziv istoimene metode usluge *apiService* kojoj se kao argument predaje objekt sa svojstvima unesenima u formu.

Stanje *main.pending-warrants* je stanje kojem pristup ima samo prvi tip korisnika, a predstavlja tekuće putne naloge, odnosno nudi korisniku pregled i uređivanje putnih naloga koje je potrebno ispuniti i poslati. Predložak stanja sadrži listu tekućih putnih naloga i formu za unos vrijednosti koje se odnose na trenutno odabrani tekući putni nalog. Lista je definirana direktivom *md-list*, a pojedini elementi liste direktivom *md-list-item*. Za dohvaćanje liste tekućih putnih naloga trenutno prijavljenog korisnika korištena je metoda *getWarrants* usluge *apiService* kojoj je kao argument predan niz znakova 'user/pending'. Uz navedeno, predložak ovog stanja sadrži i dugmeta za brisanje i spremanje trenutno unesenih vrijednosti te za pregled dokumenta u formatu PDF. U predlošku su također prisutna i dugmeta za dodavanje i uklanjanje stavki unosa koje se tiču relacija putovanja i ostalih troškova putovanja. Neki od elemenata unosa predloška sadrže atribut definiran direktivom *ng-change* kojom se definira akcija koja se izvršava na promjenu vrijednosti pripadajućeg elementa unosa. U ovom slučaju ta direktiva je korištena za pozivanje metoda kojima se ažuriraju brojčane vrijednost prikazane u predlošku. Za odabiranje lokalnih datoteka u svrhu priloga korištene su direktive *ngf-select* i *ngf-multiple* modula *ngFileUpload*. Direktiva *ngf-select* poziva prozor za odabiranje lokalnih datoteka i sprema ih u niz koji se pridružuje varijabli određenoj direktivom *ng-model*, a direktiva *ngf-multiple* omogućuje odabiranje više datoteka odjednom. Upravljač *PendingWarrantsCtrl* tijekom inicijalizacije poziva funkciju *init* koja odabire prvi tekući putni nalog ako lista tekućih putnih naloga nije prazna. Nalog se odabire pozivanjem metode *selectWarrant* kojoj se kao argument predaje indeks putnog naloga kojeg se želi učitati. Metoda *selectWarrant* vrijednosti koje se tiču uređivanja putnog naloga (elementi forme, popis priloga i potpis) postavlja u vrijednosti koje sadrži odabrani putni nalog. Za dohvaćanje priloga putnog naloga korištena je metoda *getAttachments* usluge *apiService* kojoj se kao argument predaje identifikacijska oznaka putnog naloga. Upravljač također ima definirane metode za brisanje trenutno unesenih vrijednosti (*clear*), za pozivanje dijaloga koji omogućuje unos potpisa (*sign*) i za pozivanje dijaloga koji korisniku omogućuje pregled dokumenta i njegovo slanje. Uz navedene metode, upravljač sadrži i metodu *save* koja sprema trenutno unesene vrijednosti. Za spremanje vrijednosti putnog naloga pozivaju se metode *updateWarrant* i *postAttachments*, odnosno *deleteAttachments* usluge *apiService*. Metoda *save* definirana je kako bi se korisniku omogućilo postepeno ispunjavanje putnog naloga. Ostale metode upravljača *PendingWarrantsCtrl* korištene su za dinamično ažuriranje brojčanih vrijednosti putnih troškova i za dodavanje, odnosno uklanjanje pojedinih stavki unosa.

Moduli *requests* i *warrants* imaju vrlo sličnu funkcionalnost, no razlikuju se u vrsti dokumenata nad kojima je stvarana njihova funkcionalnost. Oba modula u datoteci za postavljanje imaju definirana sljedeća stanja: roditeljsko apstraktno stanje, stanje *validation*, stanje *approval*, stanje *sent* i stanje *processed*. Modul *warrants* uz ta stanja ima definirano i stanje *accounting*. Stanja *sent* i *processed* koriste isti predložak i upravljač koji se nalazi u direktoriju *main/requests/view*, odnosno *main/warrants/view*, a predlošci i upravljači ostalih stanja nalaze se u zasebnim direktorijima.

Roditeljsko apstraktno stanje oba modula omogućuje pregled, filtriranje i sortiranje liste dokumenata te prikaz odabranog dokumenta u PDF formatu. Svako dječje stanje (engl. *child state*) listu dokumenata dohvaća tijekom aktiviranja korištenjem svojstva *locals*. Ispod je prikazan kod za stvaranje elemenata liste i njihovih svojstava.

```
<!-- Iz datoteke warrants.html -->
<md-list-item ng-repeat=
  "warrant in warrants.warrants |
  filter: { surname: warrants.searchSurname } |
  orderBy: warrants.orderBy"
  ng-click="warrants.select($index)"
  ng-class="{ 'selected': $index == warrants.current }"
  class="md-2-line"
  ng-mouseenter="warrants.verbose[$index] = true"
  ng-mouseleave="warrants.verbose[$index] = false">
```

Elementi liste stvaraju se direktivom *ng-repeat* za čiju su funkcionalnost dodani filteri *filter* i *orderBy* koji predstavljaju dio usluge *$filter*. Filter *filter* definira objekt sa svojstvima koja posjeduju objekti iz liste, a kojima su pridružene vrijednosti filtriranja. U ovom slučaju omogućeno je filtriranje prezimena (engl. *surname*) − vrijednost filtriranja unosi korisnik te se ono sprema u svojstvo *searchSurname* upravljača. Filter *orderBy* definira svojstvo prema kojem su objekti liste poredani. Odabrana vrijednost sprema se u svojstvo *orderBy* upravljača. Uz to su korištene i direktive *ng-class*, *ng-mouseenter* i *ng-mouseleave*. Direktiva *ng-class* pripadajućem elementu pridružuje klasu ako je logički izraz istinit. U ovom slučaju elementu se pridružuje klasa *selected* (hrv. odabrano) ako taj element predstavlja trenutno odabrani element liste. Direktive *ng-mouseenter* i *ng-mouseleave* definiraju akciju koja se izvršava kada pokaznik miša uđe u, odnosno napusti granice pripadajućeg elementa. U ovom slučaju tim akcijama se omogućuje detaljniji prikaz informacija elemenata liste. Predložak još sadrži elemente unosa vrijednosti filtriranja, dugmeta za odabir prijašnjeg ili sljedećeg elementa, ispis imena i veličine priloga putnih naloga te dugmeta za dohvaćanje pojedinih priloga. Element oznak *iframe*

sakriven je u predlošku (korištenjem direktive *ng-hide*) i služi kao pomoćni element za dohvaćanje priloga. Upravljači roditeljskog stanja oba modula imaju definiranu inicijalizacijsku metodu *init* koju pozivaju upravljači dječjih stanja nakon što im lista dokumenata postane dostupna. Uz to, imaju definirane i metode za odabiranje dokumenta te metodu *getClass* koja vraća određenu klasu CSS-a ovisno o razini obrađenosti dokumenta. Upravljač *WarrantsCtrl* ima definiranu i metodu *downloadAttachment* koja se poziva u svrhu dohvaćanja priloga. Prilog se dohvaća postavljanjem atributa *src* elementa oznake *iframe* u vrijednost putanje prema pozadinskom API-ju što pokreće preglednikovo spremanje datoteke.

Dječja stanja *validation* spomenuta dva modula su stanja kojima mogućnost pristupa ima samo druga vrsta korisnika, a služe provjeravanju ispravnosti dokumenata. U predlošku sadrže dugmeta za ocjenu ispravnosti dokumenata. Upravljač stanja za provjeru ispravnosti zahtjeva (*RequestsValidationCtrl*) ima definirane metode *invalid* (hrv. neispravno) i *valid* (hrv. ispravno). Pozivanjem metode *invalid* poziva se dijalog koji služi za neprihvaćanje dokumenta. Upravljač i predložak dijaloga nalaze se u direktoriju *main/dialogs/reject-dialog*. Za ovaj dijalog postoje dva predloška: za korisnike druge i treće vrste (*reject-dialog-reason.html*) te za korisnike četvrte vrste (*reject-dialog-bill.html*). Predložak koji se dohvaća za korisnika četvrte vrste sadrži formu za izmjenu vrijednosti računa putnog naloga koji se ne prihvaća. Oba predloška sadrže element za unos razloga neprihvaćanja dokumenta i dugme za potvrđivanje unesenog. Upravljač dijaloga (*RejectDialogCtrl*) tijekom aktiviranja dohvaća prosljeđene vrijednosti: *docType* (vrsta dokumenta), *docId* (identifikacijski broj dokumenta), *userType* (vrsta korisnika) i objekt *warrant* (po potrebi) koji sadrži svojstva računa putnog naloga. Tijekom inicijalizacije upravljača dijaloga postavljaju se tekstualne vrijednosti dijaloga, a za četvrtu vrstu korisnika i vrijednosti elemenata unosa koje odgovaraju vrijednostima računa putnog naloga. Upravljač, međuostalim, ima definiranu metodu *confirm* koja se poziva u trenutku potvrđivanja unosa. Metoda *confirm* poziva metode *getRejectionObject* i *getNewWarrantObject* (samo ako se radi o putnom nalogu) te ažurira dokument pozivanjem metoda usluge *apiService*. Metoda *getRejectionObject* vraća objekt s ažuriranim svojstvima koja se tiču neprihvaćanja dokumenta, a metoda *getNewWarrantObject* objekt sa svojstvima koja predstavljaju kopiju odbijenog putnog naloga. Za korisnika četvrte vrste definiraju se i metode upravljača koje služe za izmjenjivanje i ažuriranje vrijednosti računa putnog naloga.

Pozivanjem metode *valid* upravljača *RequestsValidationCtrl* poziva se dijalog za unošenje oznake odobrenog zahtjeva. Predložak i upravljač tog dijaloga nalaze se u direktoriju *main/dialogs/mark-request-dialog*. Predložak sadrži element za unos oznake i dugme za potvrdu. Potvrđivanjem unosa poziva se metoda *confirm* upravljača *MarkRequestDialogCtrl* koja ažurira

svojstva ispravnosti i oznaku zahtjeva. S druge strane, pozivanjem metode *valid* upravljača *WarrantsValidationCtrl* dijalog se neće pozivati, jer nije potreban nikakav dodatni unos.

Dječje stanje *accounting* roditeljskog stanja *warrants* je stanje kojemu mogućnost pristupa ima samo četvrta vrsta korisnika, a služi provjeravanju ispravnosti i izmjeni računa putnog naloga u slučaju njegove neispravnosti. Predložak stanja *accounting* sadrži dugmeta za odbijanje i odobravanje putnog naloga, a upravljač ima definirane metode *disapprove* (hrv. nemoj odobriti) i *approve* (hrv. odobrij). Metoda *disapprove* poziva dijalog za odbijanje dokumenta, a metoda *approve* dijalog za potpisivanje dokumenta.

Dječja stanja *approval* su stanja kojima mogućnost pristupa ima samo treća vrsta korisnika, a služe odobravanju, odnosno odbijanju dokumenata. Predlošci i upravljači ovih stanja također imaju dugmeta i metode za odbijanje i odobravanje. Metode *disapprove* i *approve* pozivaju iste dijaloge kao i upravljač stanja *accounting*, samo s drugačijim argumentima.

Dječja stanja *sent* i *processed* koriste jednaki predložak i upravljač pojedinog modula (*requests* ili *warrants*). Predložak ovih stanja sadrži element (u obliku dugmeta) koji prikazuje razinu obrađenosti dokumenta i dugme za detaljniji uvid u tijek obrađivanja dokumenta. Klikom na dugme poziva se metoda *showDetails* odgovornog upravljača kojom se poziva dijalog za prikaz detalja obrađivanja dokumenta. Predložak i upravljač tog dijaloga nalaze se u direktoriju *main/dialogs/details-dialog*. Predložak dijaloga čine ispisi vrijednosti pojedinih svojstava upravljača koja predstavljaju podatke o obradi dokumenta. Vrijednosti svojstava upravljača definiraju se pozivanjem pojedinih metoda upravljača, a to su: *getCreationInfo* (vraća tekst o vremenu stvaranja dokumenta), *getMainInfo* (vraća tekst o obrađenosti dokumenta) i *getReason* (vraća razlog neprihvaćanja dokumenta).

2.2.4. Dizajn

Za dizajn aplikacije korištene su stilske datoteke modula kojih ih zahtjevaju (moduli *ngMaterial* i *scDateTime*), stilska datoteka s fontovima (*font-awesome.css*) korištenima za stvaranje PDF dokumenata te Sass jezik za predobradu i stvaranje proizvoljne stilske datoteke (*app.css*).

Za definiranje poravnanja elemenata korištene su direktive *layout* i *layout-align* modula *ngMaterial*. Njima se definira način slaganja elemenata unutar pripadajućeg elementa. U primjeru koda ispod definirano je slaganje dječjih elemenata u red i njihovo poravnanje uz lijevi rub po osi slaganja te u sredinu po okomitoj osi.

```
<-- Iz datoteke pending-warrants.html -->
<div layout="row" layout-align="start center">
```

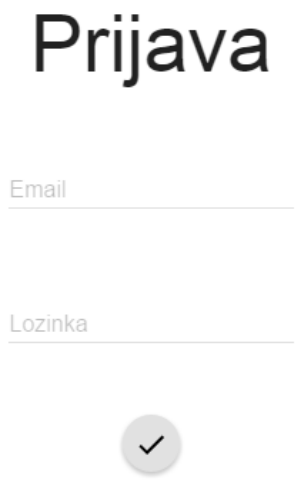
Korištena je i direktiva *flex* modula *ngMaterial* koja elementu daje mogućnost dinamičkog mijenjanja veličine u skladu s mijenjanjem veličine prozora. Za definiranje stila elemenata korištene su klase stilske datoteke modula *ngMaterial*. Neke od njih su: *md-fab*, *md-mini*, *md-raised* (za dugmeta), *md-2-line*, *md-3-line*, *md-list-item-text* (za tekst elemenata liste). Proizvoljno definirane klase i stilska uređenja nalaze se u datoteci *app.scss* roditeljskog direktorija koju Sass interpretira te naposljetku stvara odgovarajuću CSS datoteku. Sass jezik, međuostalim, omogućuje definiranje varijabli, što je olakšalo postavljanje korištenih tonova boja. Definirane boje su konzistentne s bojama definiranim za modul *ngMaterial*, a vrijednosti su prikazane ispod.

```
// Iz datoteke app.scss
$md-primary: rgb(76, 175, 80);
$md-accent: rgb(68, 138, 255);
$md-warn: rgb(255, 171, 0);
$selected: rgb(236, 236, 236);
```

Korištene ikone dohvaćene su s adrese *https://design.google.com/icons/* koja nudi službene Googleove ikone izrađene u skladu sa *Material Design* specifikacijama.

3. KORIŠTENJE APLIKACIJE

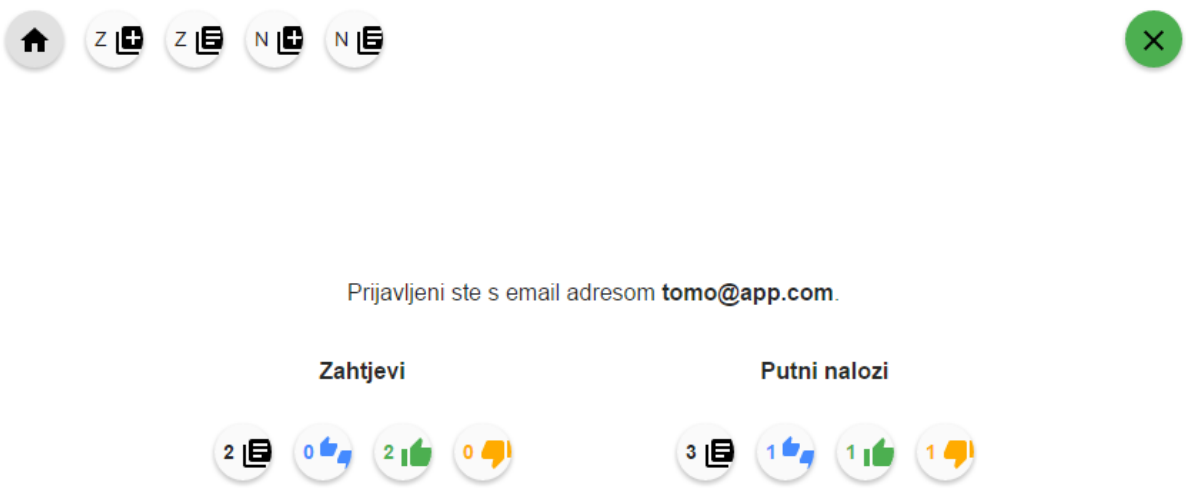
Prvi korak tijekom korištenja aplikacije je upisivanje u sustav. Korisnik za to koristi poznatu email adresu i lozinku. Upisivanje u sustav korisnika vodi na početnu stranicu aplikacije (Sl. 3.2.). Svaki postojeći korisnik spada u jednu od četiri vrste korisnika od kojih svaka ima pristup samo određenom dijelu aplikacije. U gornjem desnom kutu nalazi se dugme za odjavu iz sustava. U sustav se također može upisati i administrator.



Sl. 3.1. Stranica za prijavu u sustav.

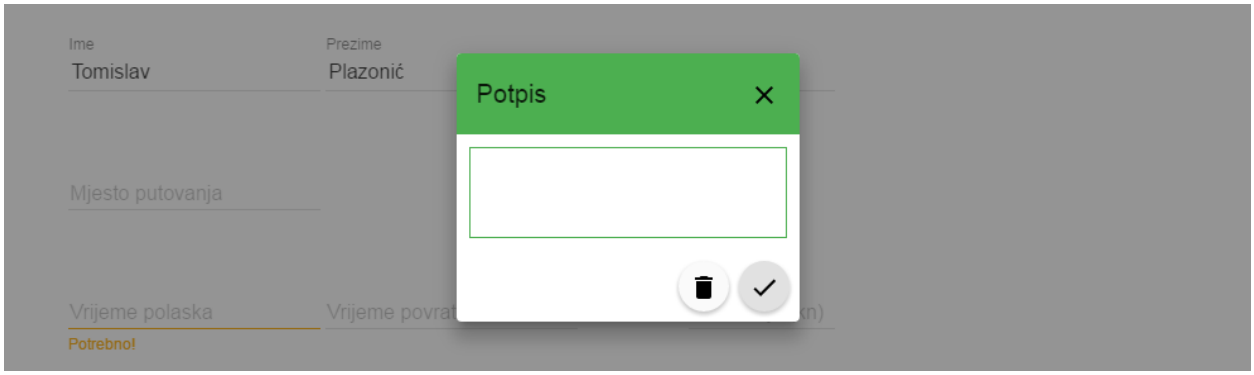
3.1. Prva vrsta korisnika

Prva vrsta korisnika predstavlja korisnike koji podnose zahtjeve za putnim nalogom te ispunjavaju i šalju putne naloge dobivene odobravanjem podnesenih zahtjeva. Korisnik prve vrste ima pristup sljedećim stranicama: *Početna*, *Novi zahtjev*, *Poslani zahtjevi*, *Tekući putni nalozi*, *Poslani putni nalozi*. Početna stranica za korisnike prve vrste prikazuje broj zahtjeva i putnih naloga koje su poslali, koji im nisu odobreni, koji su im odobreni i čija je obrada u tijeku. Početne stranice ostalih vrsta korisnika izgledaju jednako, ali prikazuju drugačije brojke.



Sl. 3.2. Stranica prikazana nakon prijave u sustav (*Početna*).

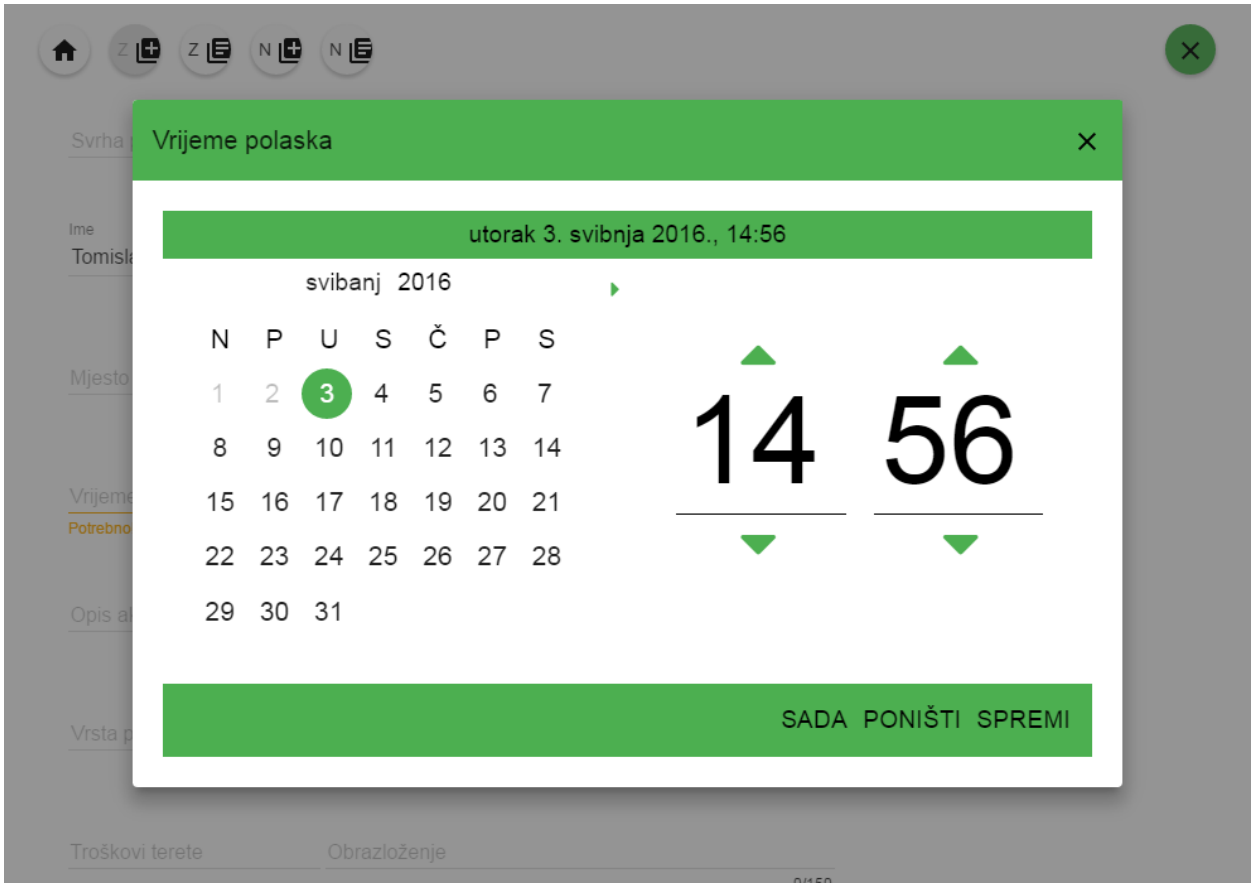
Za stvaranje novog zahtjeva za putni nalog potrebno je odabrati stanje *Novi zahtjev*. Stranica za stvaranje novog zahtjeva (Sl. 3.5.) sadrži formu s elementima unosa pojedinih dijelova zahtjeva. Ime i prezime podnositelja su automatski postavljeni, no moguće ih je izmijeniti. Pritiskom na elemente za unos vremena polaska i vremena povratka prikazuje se dijalog za unos datuma i vremena (Sl. 3.4.). Pritiskom na dugme za unos podnositelja zahtjeva prikazuje se dijalog za unos potpisa (Sl. 3.3.). Unos potpisa vrši se zadržavanjem pritiska miša na ponuđenoj bijeloj podlozi dijaloga. Uneseni potpis moguće je obrisati pritiskom na dugme za brisanje, a potvrditi pritiskom na dugme za potvrđivanje.



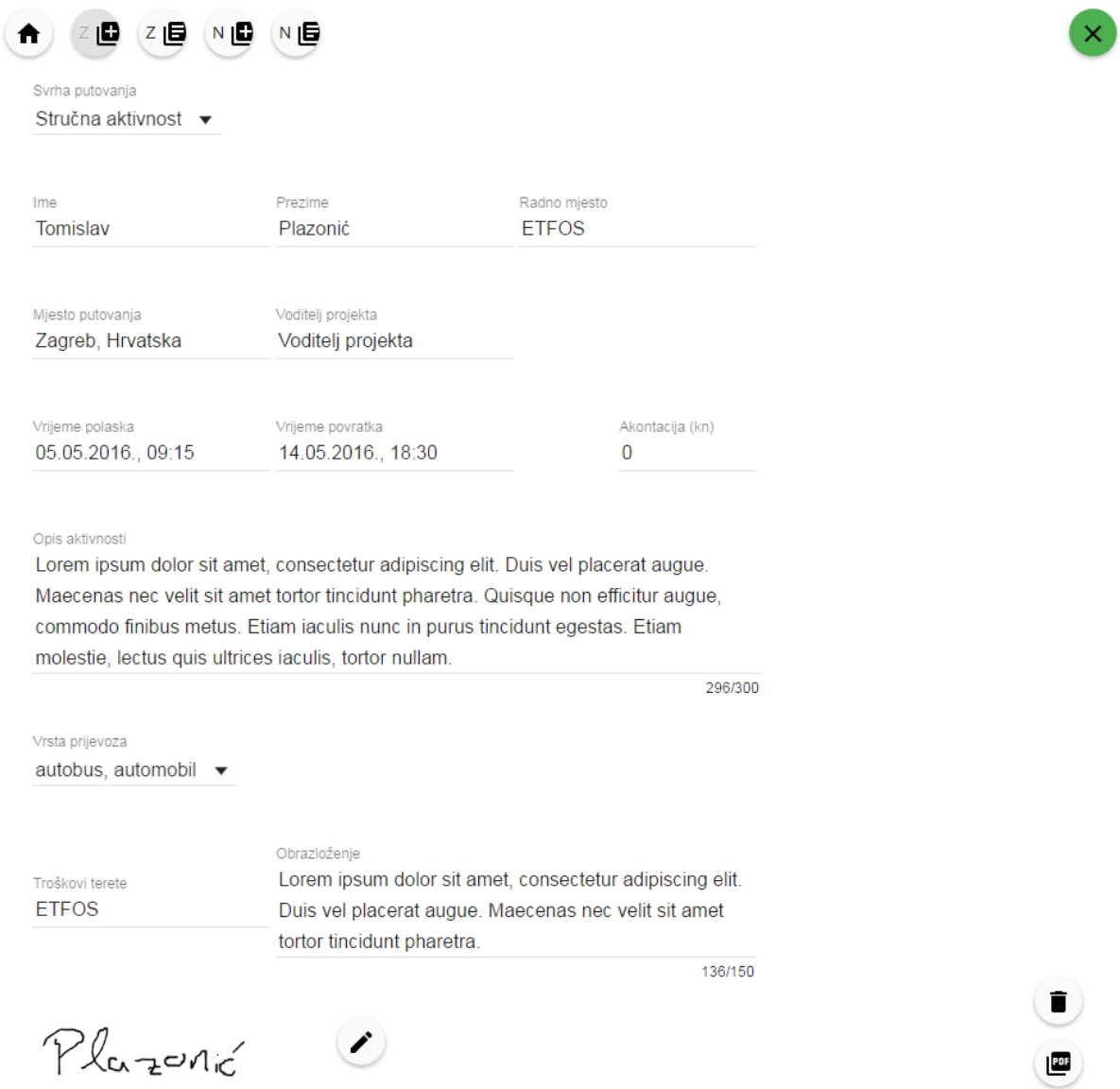
Sl. 3.3. Dijalog za unos potpisa.

Odabir datuma u dijalogu vrši se pritiskom na pojedini dan u prikazanom kalendaru. Moguće je mijenjati prikazani mjesec i godinu. Za vrijeme polaska moguće je odabrati samo one dane koji su tjedan dana nakon tekućeg dana i koji nisu nakon odabranog vremena povratka. Isto tako, za vrijeme povratka moguće je odabrati samo one dane koji su nakon odabranog vremena polaska. Odabir vremena u dijalogu vrši se pritiskom na dugmeta ili unosom brojčanih vrijednosti. Odabrani datum i vrijeme potrebno je spremiti pritiskom na *Spremi*. Za poništavanje odabrane

spremljene vrijednosti potrebno je pritisnuti *Poništi*. Pritiskom na *Sada* bit će odabran tekući dan.

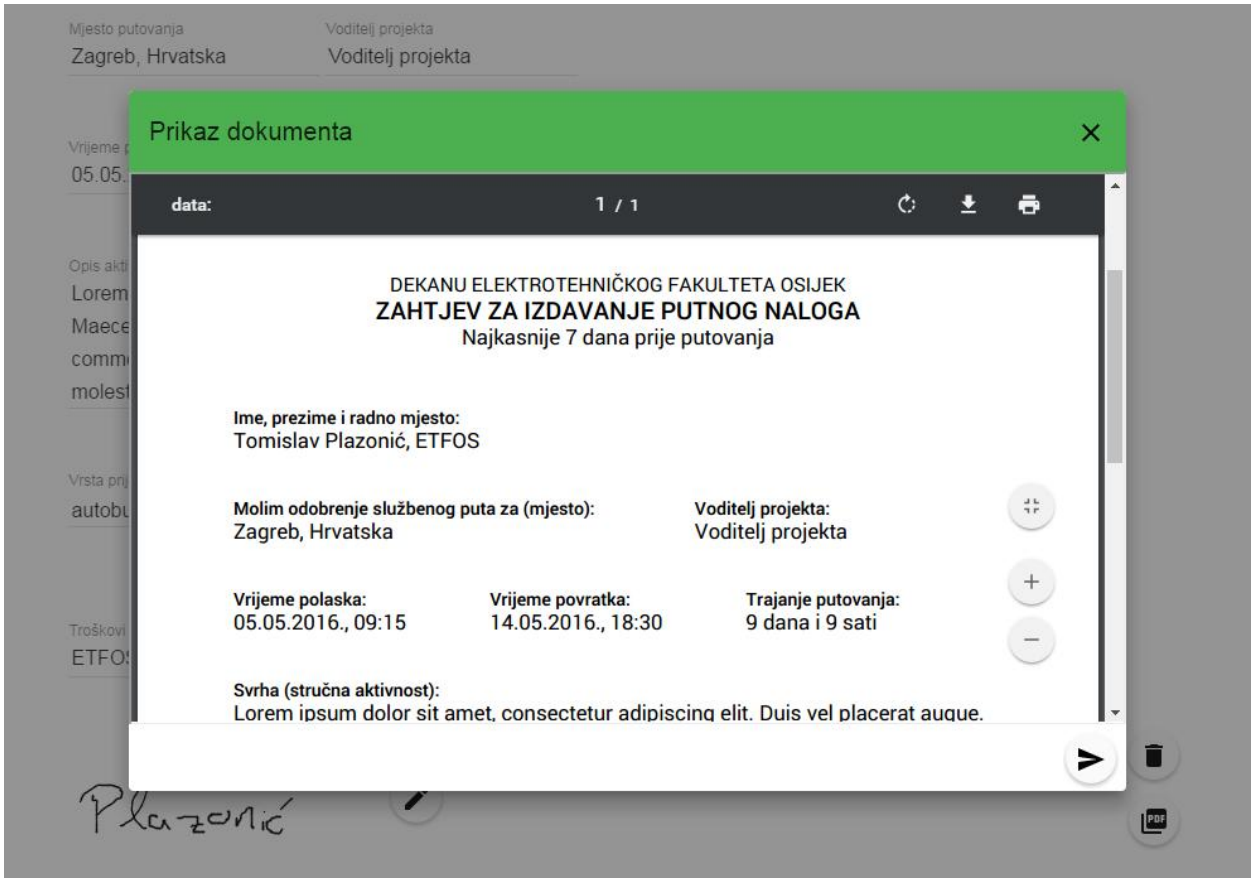


Sl. 3.4. Dijalog za odabir datuma i vremena.



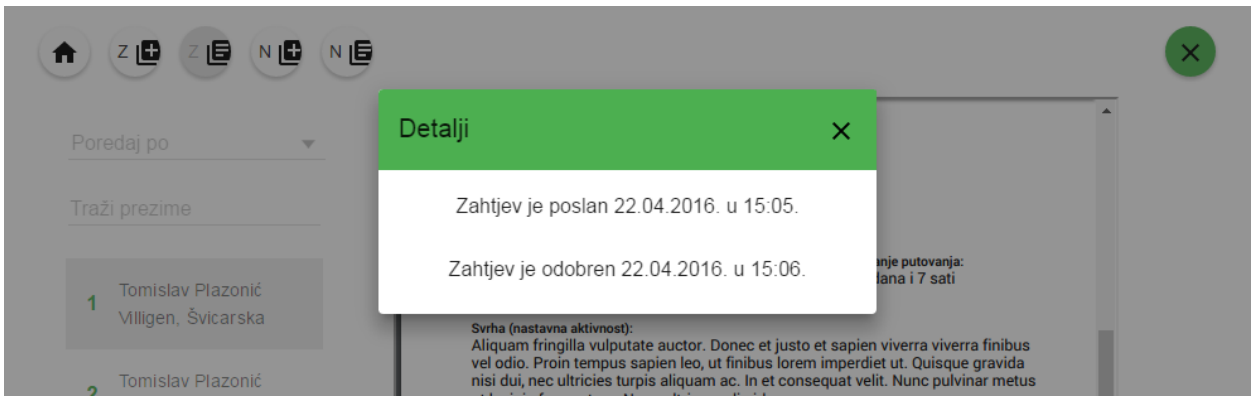
Sl. 3.5. Stranica za stvaranje novog zahtjeva za putni nalog (*Novi zahtjev*).

Stranica *Novi zahtjev* uz formu sadrži i dugme za brisanje trenutno unesenih vrijednosti te dugme za prikaz dokumenta u formatu PDF. Potrebno je ispuniti sva polja za unos i unijeti potpis kako bi dugme za prikaz dokumenta bilo omogućeno. Pritiskom na dugme za prikaz dokumenta otvara se dijalog unutar kojega se zahtjev prikazuje u formatu PDF (Sl. 3.6.). Taj dijalog sadrži dugme na čiji se pritisak potvrđuje i šalje prikazani zahtjev.

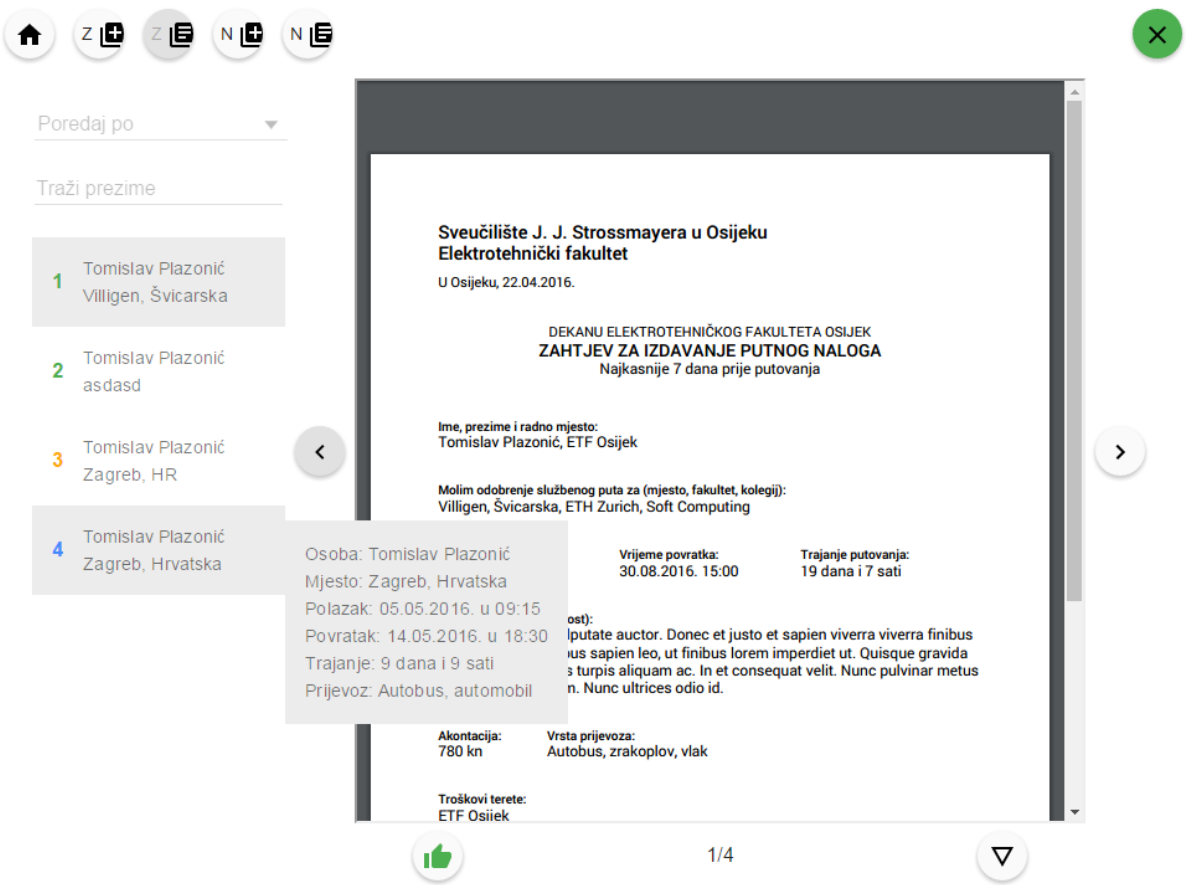


Sl. 3.6. Dijalog za prikaz dokumenta u PDF formatu.

Potvrđivanjem i slanjem zahtjeva za putni nalog korisnika se prebacuje na stranicu *Poslani zahtjevi* (Sl. 3.8.) koja pruža pregled svih zahtjeva poslanih od strane prijavljenog korisnika. Zahtjeve je moguće sortirati i filtrirati unošenjem vrijednosti u ponuđene elemente. Na stranici *Poslani zahtjevi* nalazi se i dugme na čiji se pritisak prikazuje dijalog s podacima o tijeku obrade zahtjeva (Sl. 3.7.).

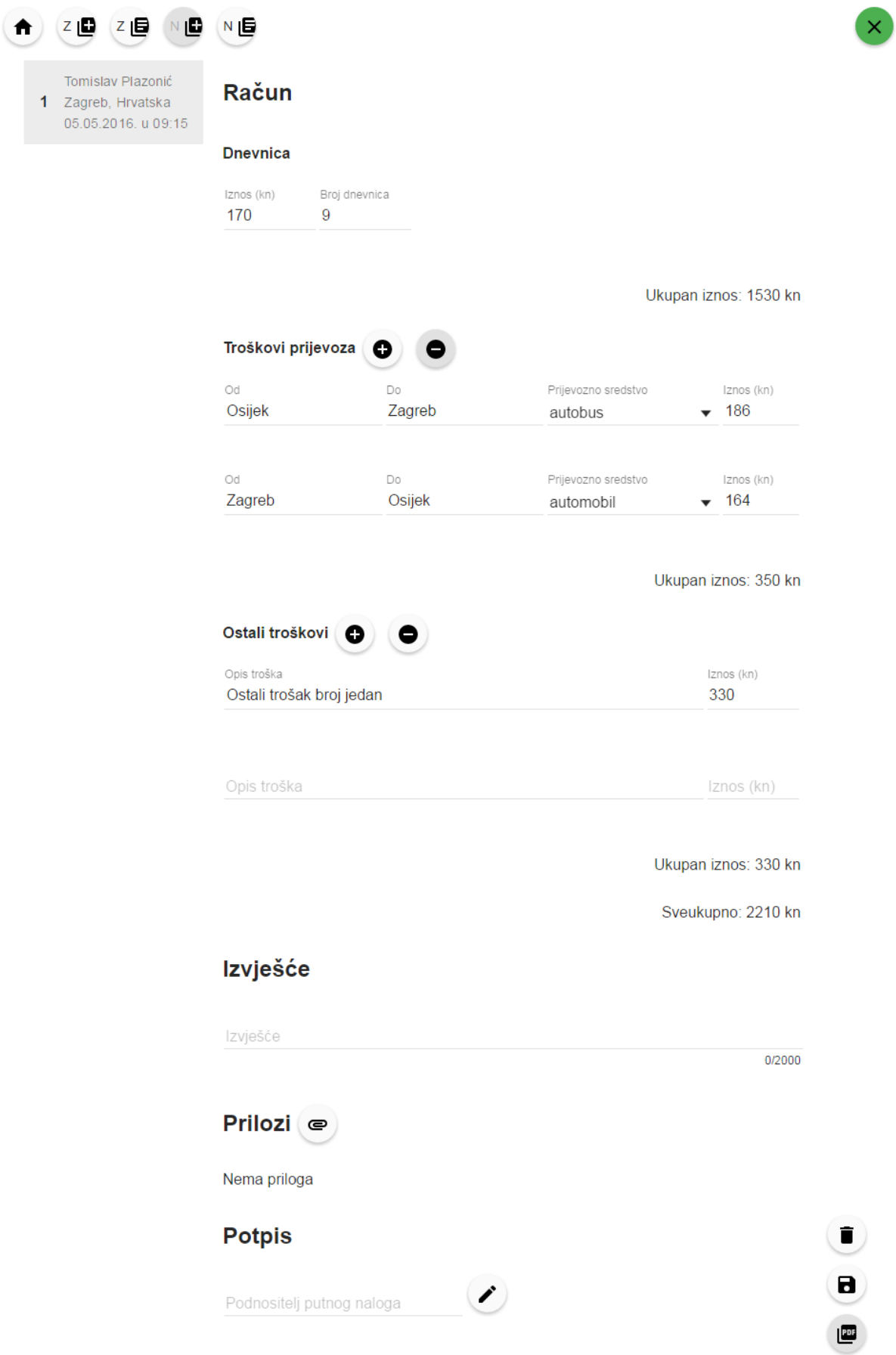


Sl. 3.7. Dijalog za prikaz podataka o tijeku obrade dokumenta.



Sl. 3.8. Stranica za pregledavanje poslanih zahtjeva (*Poslani zahtjevi*). Primjer je stranice za pregledavanje obrađenih dokumenata.

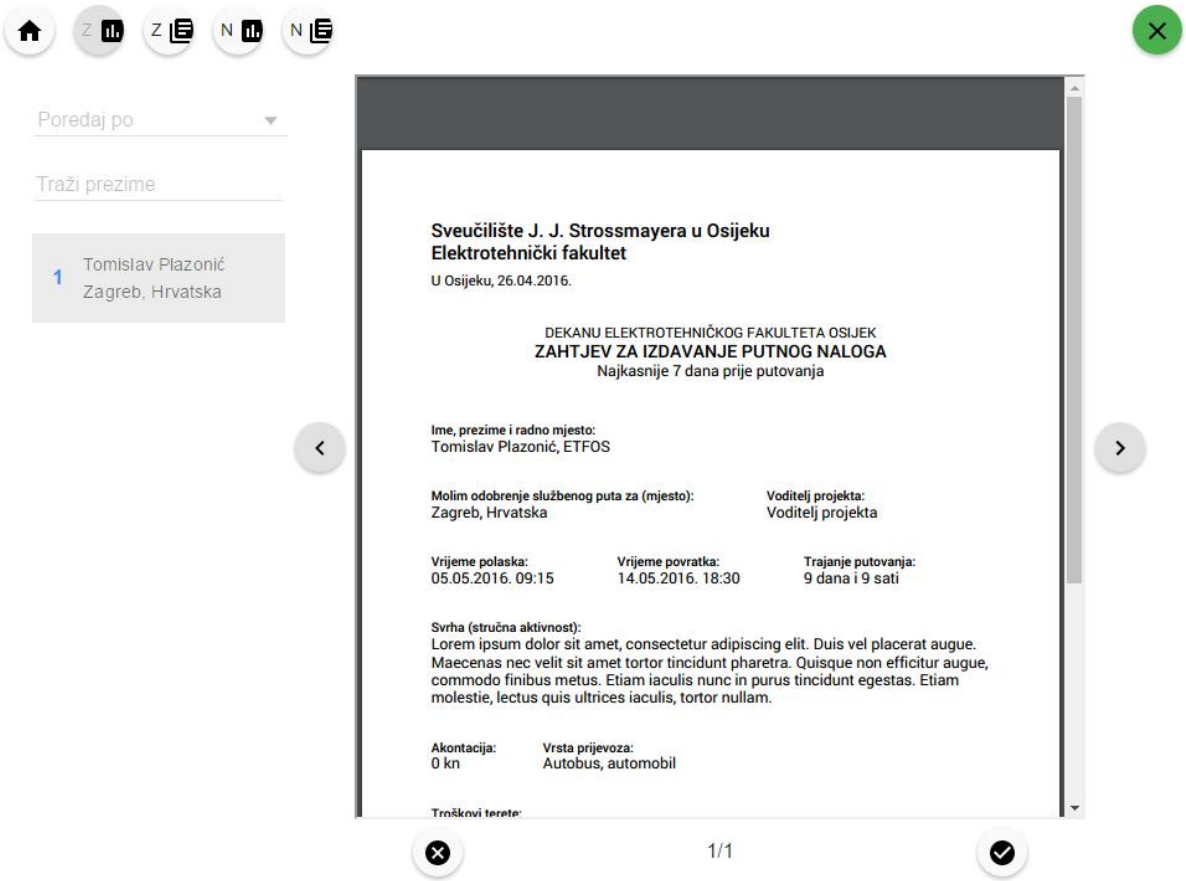
Odobravanjem zahtjeva za putni nalog korisniku se otvara putni nalog kojeg je moguće ispuniti i poslati na stranici *Tekući putni nalozi* (Sl. 3.9). Putni nalog sastoji se od putnog računa, izvješća i priloga. Korisnik obavezno mora unijeti najmanje dvije relacije (maksimalno 7) vezane za putne troškove, a unosi za ostale troškove putovanja ne moraju biti prisutni (maksimalno ih može biti 4). Polja za unose dodaju se ili uklanjaju odgovarajućim tipkama. Prilozi se odabiru pritiskom na dugme za odabir datoteka, čime se otvara prozor u kojem korisnik vrši odabir. Stranica *Tekući putni nalozi* omogućuje brisanje i spremanje trenutno unesenih vrijednosti te prikaz dokumenta u PDF formatu gdje ga se potvrđuje, odnosno šalje. Slanjem putnog naloga osvježava se trenutna stranica (*Tekući putni nalozi*). Poslane putne naloge moguće je pregledati na stranici *Poslani putni nalozi* koja je funkcionalnošću identična stranici *Poslani zahtjevi*.



Sl. 3.9. Stranica za ispunjavanje tekućih putnih naloga (*Tekući putni nalozi*).

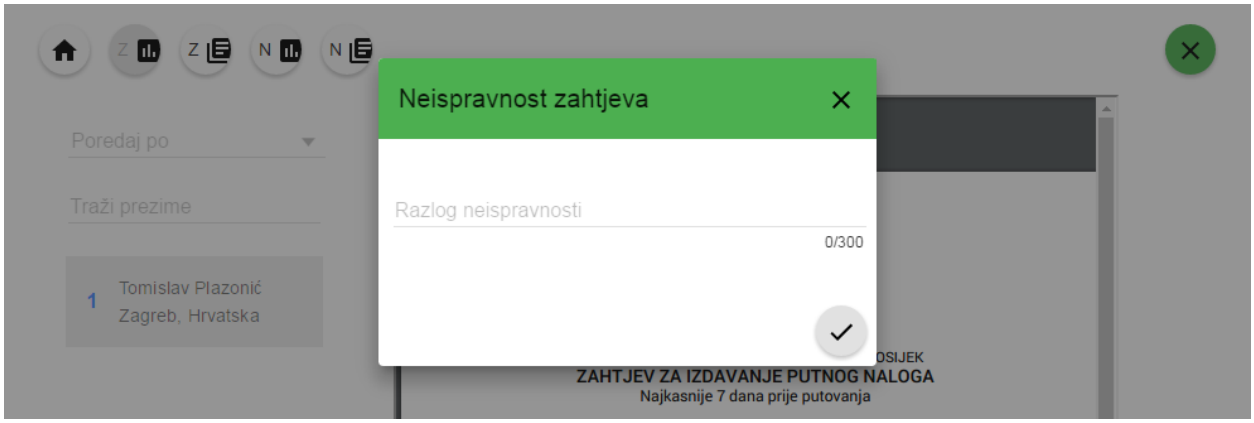
3.2. Druga vrsta korisnika

Druga vrsta korisnika predstavlja zaposlenike koji provjeravaju ispravnost zahtjeva i putnih naloga poslanih od strane prve vrste korisnika. Time se izbjegava dolazak nezadovoljavajuće ispunjenih dokumenata korisnicima koji vrše njihovo odobravanje (treća vrsta korisnika). Korisnik druge vrste ima pristup sljedećim stranicama: *Početna*, *Validacija zahtjeva*, *Obrađeni zahtjevi*, *Validacija putnih naloga*, *Obrađeni putni nalozi*. Početna stranica druge vrste korisnika prikazuje ukupan broj zahtjeva i putnih naloga te broj istih koje su korisnici druge vrste ocijenili ispravnim, neispravnim ili koji čekaju provjeru ispravnosti.

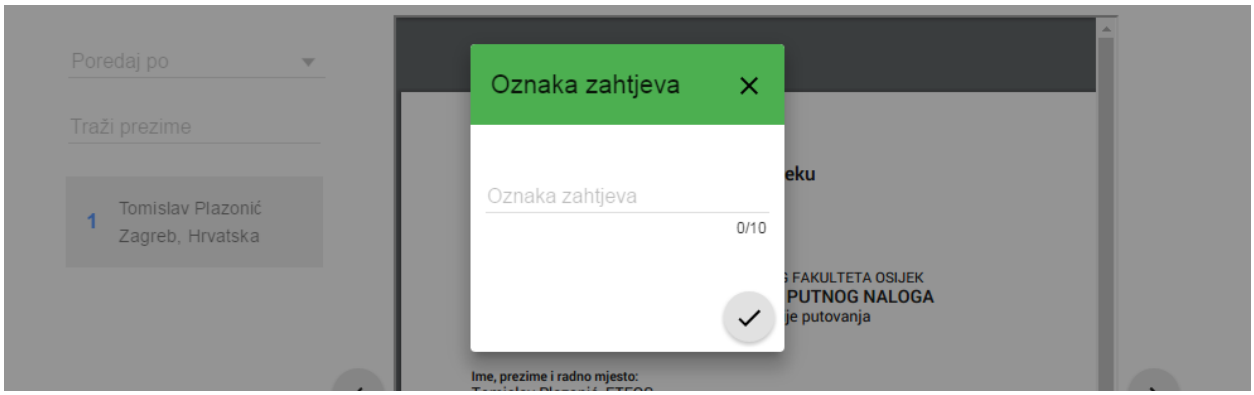


Sl. 3.10. Stranica za pregled i ocjenu ispravnosti zahtjeva (*Validacija zahtjeva*). Primjer je stranice na kojoj se dokumenti odobravaju ili odbijaju.

Stranice *Validacija zahtjeva* (Sl. 3.10.) i *Validacija putnih naloga* omogućavaju pregled primljenih dokumenata nad kojima nije izvršena provjera ispravnosti. Dokument se ocjenjuje ispravnim ili neispravnim pritiskom na odgovarajuće dugme. U slučaju ocjenjivanja dokumenta neispravnim prikazuje se dijalog za unos razloga neispravnosti (Sl. 3.11.). U slučaju ocjenjivanja zahtjeva ispravnim prikazuje se dijalog za unos oznake zahtjeva (Sl. 3.12.). Nakon obrade dokumenta osvježava se trenutna stranica.



Sl. 3.11. Dijalog za unos razloga neispravnosti dokumenta.



Sl. 3.12. Dijalog za unos oznake zahtjeva.

Obrađene dokumente moguće je pregledati na stranicama *Obrađeni zahtjevi* i *Obrađeni putni nalozi* koje su funkcionalnošću jednake ostalim stranicama za pregledavanje obrađenih dokumenata.

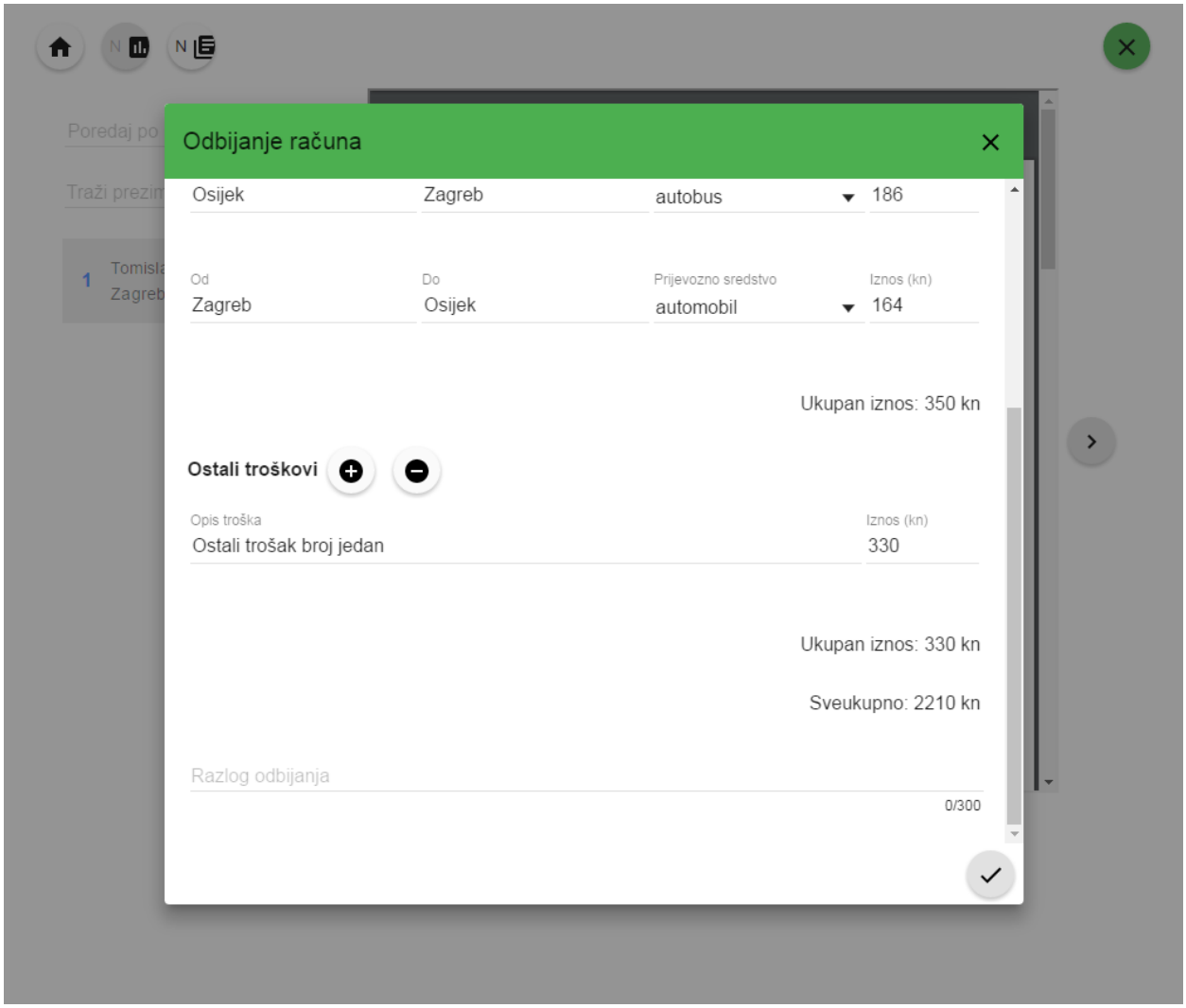
3.3. Treća vrsta korisnika

Treća vrsta korisnika predstavlja osobu koja donosi konačnu riječ u vezi odobravanja dokumenta. Korisnik treće vrste ima pristup sljedećim stranicama: *Početna*, *Odobravanje zahtjeva*, *Obrađeni zahtjevi*, *Odobravanje putnih naloga*, *Obrađeni putni nalozi*. Početna stranica treće vrste korisnika prikazuje ukupan broj zahtjeva i putnih naloga te broj istih koje su korisnici treće vrste odbili, odobrili ili koji čekaju odobravanje. Stranice *Odobravanje zahtjeva* i *Odobravanje putnih naloga* funkcionalnošću su jednake ostalim stranicama za odobravanje ili odbijanje dokumenta. Odbijanje dokumenta zahtjeva unos razloga neispravnosti, a odobravanje dokumenta zahtjeva unos potpisa.

3.4. Četvrta vrsta korisnika

Četvrta vrsta korisnika predstavlja zaposlenika računovodstva koji provjerava ispravnost računa putnog naloga. Korisnik četvrte vrste ima pristup sljedećim stranicama: *Početna, Računovođenje*

putnih naloga, *Obrađeni putni nalozi*. Početna stranica četvrte vrste korisnika prikazuje ukupan broj putnih naloga te broj istih koje su korisnici četvrte vrste odbili, odobrili ili koji čekaju pregled ispravnosti računa. Stranica *Računovođenje putnih naloga* funkcionalnošću je jednaka ostalim stranicama za odobravanje ili odbijanje dokumenta. Razlikuje se po tome što u slučaju ocjenjivanja računa neispravnim korisnik, uz unos razloga neispravnosti ima i mogućnost izmjene pripadajućih vrijednosti računa (Sl. 3.13.). Odobravanje računa putnog naloga zahtjeva unos potpisa.

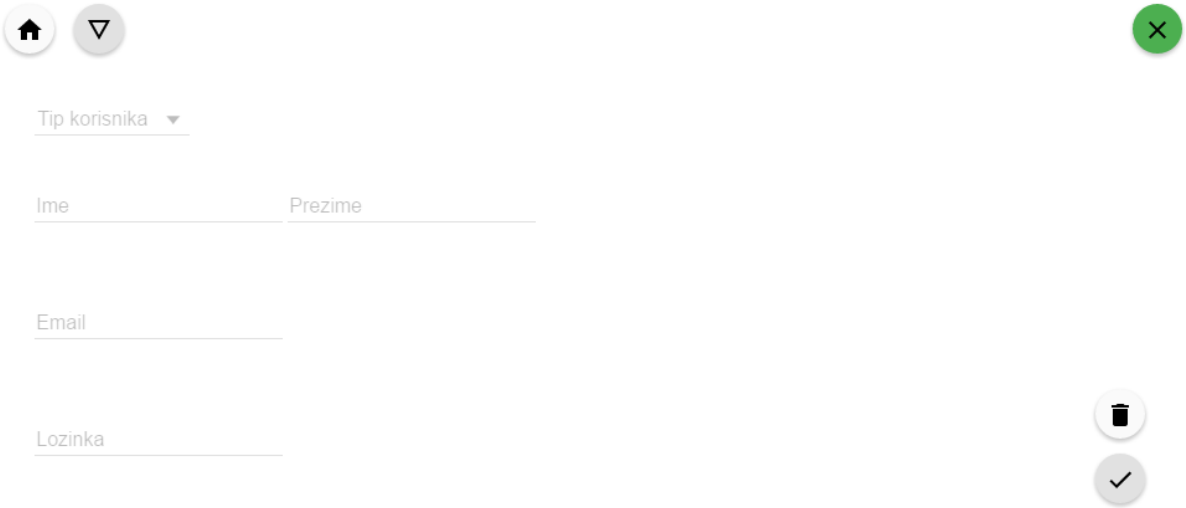


Sl. 3.13. Dijalog za odbijanje računa putnog naloga.

3.5. Administrator

Administratoru je omogućeno unošenje novih korisnika u sustav. Administrator ima pristup sljedećim stranicama: *Početna*, *Novi korisnik*. Početna stranica administratora prikazuje ukupan broj, broj odbijenih, odobrenih i neobrađenih zahtjeva i putnih naloga. Stranica *Novi korisnik* (Sl.

3.14.) sadrži formu za unos podataka novog korisnika, dugme za brisanje trenutno unesenih vrijednosti i dugme za potvrđivanje stvaranja korisnika.



Sl. 3.14. Stranica za stvaranje novog korisnika (*Novi korisnik*).

4. ZAKLJUČAK

Mrežna aplikacija za upravljanje službenim putovanjima potvrđuje mogućnost odrađivanja procesa izdavanja putnih naloga korištenjem mrežnog preglednika uz preduvjet posjedovanja internetske veze. Aplikacija poslove podnošenja, pregledavanja, ispravljanja i odobravanja relevantnih dokumenata čini znatno lakšima, a cijeli proces efikasnijim i bržim. Također, uklanja se potreba za papirnatim dokumentima, a integritet podataka ostaje očuvan zbog načina pristupanja aplikaciji i korištenja vlastoručnog potpisa.

Upotreba aplikacije u službene svrhe zahtjeva nadogradnju različitih dijelova. Jedan od njih je izmjena i proširivanje baze podataka kako bi se zadovoljio širi opseg podataka koje je potrebno spremati. Drugi dio nadogradnje nadovezuje se na prvi, a čini ga nadogradnja pristupnog sustava koja omogućuje unos i prikaz spomenutog šireg opsega podataka. Također, korištenje aplikacije u službene svrhe zahtjeva i izmijenjeni način upisivanja u aplikaciju. Takav izmijenjeni način koristi postojeće korisničke podatke dohvaćene sa službenog poslužitelja koji se već upotrebljavaju u neke druge svrhe, a koje korisnici dobro poznaju. S obzirom da je aplikacija razvijena za hrvatsko govorno područje, uključivanje prijevoda aplikaciju bi prilagodilo ostalim govornim područjima, a po potrebi bi se izmijenila i njena funkcionalnost. Unatoč činjenici da aplikacija ispravno radi i zadovoljava sve definirane zahtjeve, procesi refaktorizacije koda, normalizacije baze podataka i uvođenja jediničnih (engl. *unit*) i cjelovitih (engl. *end-to-end*) testova učinili bi ju kvalitetnijom i potpunijom. Spomenute nadogradnje i izmjene omogućile bi da takav sustav bude legalna alternativa klasičnom obliku izdavanja i obrađivanja putnih naloga, a stvorio bi se i komercijalni potencijal.

LITERATURA

[1] *https://msdn.microsoft.com/en-us/library/ms973831.aspx* (12.04.2016.)

[2] *http://www.google.com/patents/US8136109* (29.04.2016.)

[3] *https://wiki.python.org/moin/WebFrameworks* (13.04.2016.)

[4] *https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction* (29.04.2016.)

[5] *http://php.net/manual/en/intro-whatis.php* (29.04.2016.)

[6] *https://docs.angularjs.org/guide/introduction* (29.04.2016.)

[7] *https://laravel.com/docs/4.2/introduction* (29.04.2016.)

[8] *https://en.wikipedia.org/wiki/HTML* (13.04.2016.)

[9] *https://en.wikipedia.org/wiki/Cascading_Style_Sheets* (13.04.2016.)

[10] *http://sass-lang.com/documentation/file.SASS_REFERENCE.html* (29.04.2016.)

[11] *http://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html* (29.04.2016.)

[12] *https://en.wikipedia.org/wiki/PhpStorm* (14.04.2016.)

[13] *https://git-scm.com* (29.04.2016.)

[14] *https://en.wikipedia.org/wiki/GitHub* (14.04.2016.)

[15] *https://en.wikipedia.org/wiki/Google_Chrome* (14.04.2016.)

[16] *https://en.wikipedia.org/wiki/Package_manager* (14.04.2016.)

[17] *https://jwt.io/introduction* (15.04.2016.)

[18] *https://laravel.com/docs/5.1/migrations* (15.04.2016.)

[19] R. T., Fielding, Architectural Styles and the Design of Network-based Software Architectures, 2000.

[20] *https://en.wikipedia.org/wiki/Application_programming_interface* (15.04.2016.)

[21] *https://laravel.com/docs/master/middleware* (17.04.2016.)

[22] *https://msdn.microsoft.com/en-us/magazine/dn463786.aspx* (29.04.2016.)

[23] *https://docs.angularjs.org/guide/services* (18.04.2016.)

[24] *https://github.com/angular-ui/ui-router* (18.04.2016.)

[25] *https://github.com/mgonto/restangular* (18.04.2016.)

[26] *https://github.com/lodash/lodash* (24.04.2016.)

[27] *https://github.com/angular/material* (18.04.2016.)

[28] *https://docs.angularjs.org/api/ngMessages* (18.04.2016.)

[29] *https://github.com/sahat/satellizer* (18.04.2016.)

[30] *https://docs.angularjs.org/guide/i18n* (18.04.2016.)

SAŽETAK

Aplikacija za upravljanje službenim putovanjima je mrežna aplikacija koja omogućuje ispunjavanje, podnošenje, pregledavanje i obradu dokumenata koji se upotrebljavaju za službena putovanja. Postoje četiri vrste korisnika od kojih svaka ima pristup samo određenim dijelovima aplikacije, ovisno o njihovoj ulozi i nadležnosti. Dokumenti se prikazuju u formatu PDF, a njihova vjerodostojnost zajamčena je korištenjem potpisa korisnika aplikacije. Aplikacija je razvijena u programskim jezicima JavaScript i PHP. Za razvoj su korišteni programski okviri AngularJS i Laravel.

Ključne riječi

Mrežna aplikacija, baza podataka, programski okvir.

APPLICATION FOR BUSINESS TRIP MANAGEMENT

Application for business trip management is a web application that enables business-trip-related documents to be filled out, submitted, reviewed and processed. There are four groups of users with access only to certain parts of application, depending on their role and jurisdiction. Documents are shown in PDF format and their authenticity is guaranteed by users' signatures. The application has been developed in JavaScript and PHP programming languages. AngularJS and Laravel frameworks have been used for development.

Keywords

Web application, database, framework.

ŽIVOTOPIS

Tomislav Plazonić rođen je 26. kolovoza 1991. godine u Zagrebu. 1998. godine upisuje se u Osnovnu školu Mladost u Osijeku gdje se ističe kao izvrstan učenik. Zatim se 2006. godine upisuje u III. gimnaziju u Osijeku gdje također postiže izvrsne rezultate te sudjeluje na natjecanjima (2009. godine osvaja treće mjesto na Županijskom natjecanju iz informatike). Državnu maturu polaže s odličnim uspjehom, nakon čega 2010. godine upisuje sveučilišni preddiplomski studij računarstva na Elektrotehničkom fakultetu u Osijeku. 2013. godine sudjeluje na Software StartUp Academy programu u Osijeku i na StartUp Camp programu u Rijeci. Nakon završetka preddiplomskog studija, 2013. godine upisuje sveučilišni diplomski studij računarsva, smjer Procesno računarstvo, također na Elektrotehničkom fakultetu u Osijeku. Iste godine pridružuje se međunarodnoj studentskoj udruzi IAESTE preko koje 2013. odrađuje dvomjesečnu stručnu praksu na Tehničkom sveučilištu u Košicama u Slovačkoj, a 2014. tromjesečnu stručnu praksu na institutu Paul Scherrer u Švicarskoj. 2015. zapošljava se kao web programer u američkoj tvrtci Aquicore koja se bavi analizom i optimizacijom energetske potrošnje u komercijalnim nekretninama te tamo provodi sedam mjeseci.