Izrada i testiranje web rješenja za izdavanje fiskalnih računa

Radonić, Stjepan

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Programsko inženjerstvo

IZRADA I TESTIRANJE WEB RJEŠENJA ZA IZDAVANJE FISKALNIH RAČUNA

Diplomski rad

Stjepan Radonić

Osijek, 2017.

SADRŽAJ

1. UVOD

Zakon o fiskalizaciji je nastao kao način provođenja kvalitetnije kontrole i nadzora Porezne uprave nad prometom poduzetnika, koji isporuke dobara i usluga naplaćuju u gotovini, kako bi se bolje pratila naplata poreza i smanjivale porezne utaje. Fiskalizacija je uglavnom provođenje nadzora prilikom izdavanja računa koji se plaćaju u gotovini, karticama, čekom ili nekim drugim sličnim načinima plaćanja. Računi se zbog obaveze fiskalizacije izdaju preko elektroničkih uređaja koji moraju biti izravno povezani s Poreznom upravom. Svaki račun se prilikom izdavanja šalje na CIS server Porezne uprave, koji može obraditi deset tisuća računa u sekundi, a od CIS-a se dobije JIR kod koji se ispisuje zajedno s ostalim elementima računa, među kojima je i ZKI. JIR stoji za jedinstveni identifikator računa i dostavlja ga Porezna uprava kao potvrdu o uredno zaprimljenim elementima računa, a ZKI je zaštitni kod izdavatelja računa i to je kod kojim se potvrđuje veza između obveznika fiskalizacije i izdanog računa te ga stvara obveznik fiskalizacije.

Svi računi u Republici Hrvatskoj se moraju fiskalizirati, bili to računi od maloprodajnih trgovaca, hotela ili obrtnika. Što znači da su svi izdavači računa prisiljeni nabaviti fiskalne alate i neki način pristupa internetu i s time se povećala potražnja za različitim rješenjima koja nude fiskaliziranje računa prema sistemu koji je propisala Porezna uprava. Sistem nije jednostavan te koristi moderne tehnologije u programiranju da bi mogao sigurno i uspješno raditi.

U nastavku rada će biti objašnjene korištenje tehnologije za izradu internet aplikacije koje uključuju Laravel programski okvir za PHP umjesto klasičnog pristupa PHP-u, također će biti naveden razlog korištenja Laravel-a. Od ostalih korištenih tehnologija za izradu još ima: JavaScript, jQuery, CSS i druge te razni paketi. Zatim će biti detaljno pojašnjena struktura aplikacije i sve njene funkcionalnosti u kojoj će biti dodatno razjašnjen sistem fiskalizacije. Nakraju, bit će još opisana teoretska osnova testiranja aplikacija i rezultati koji su dobiveni iz provođenja testiranja na ovoj aplikaciji za fiskalizaciju.

1.1. Zadatak rada

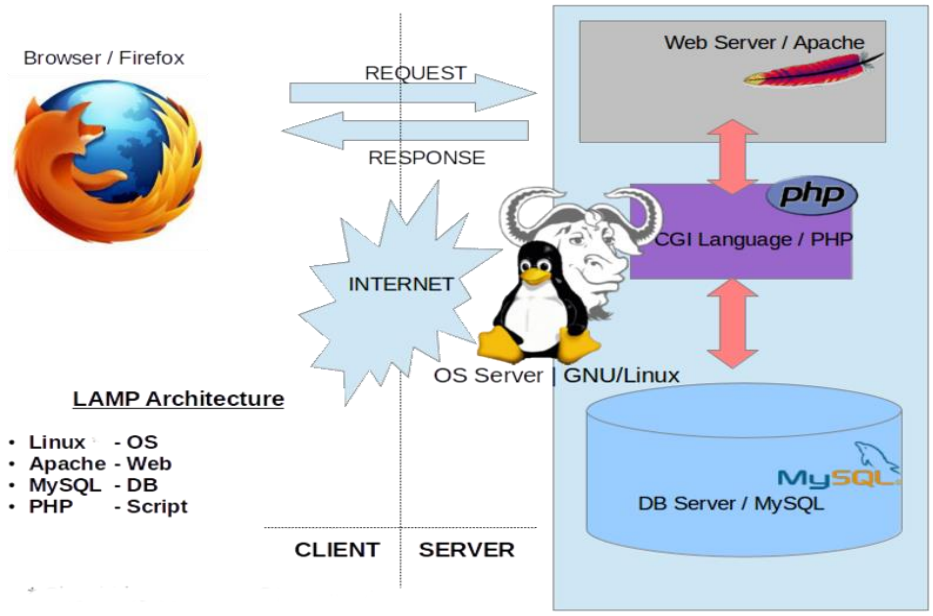
Opisati način izdavanja fiskalnih računa. Objasniti način komunikacije s poslužiteljem. Upotrebom dostupnih web tehnologija izraditi web aplikaciju za izdavanje fiskalnih računa. Opisati i dokumentirati postupak izrade aplikacije. Obaviti testiranje rada aplikacije, posebno komunikaciju s poslužiteljem za fiskalizaciju računa. U procesu izrade web aplikacije moguće je koristiti neki od programskih okvira.

2. TEHNOLOGIJE

Gotovo svaka web aplikacije se sastoji od programskog sučelja (engl. *frontend*) i programske podrške (engl. *backend*) te od servera na kojem se vrti, bio to lokalni server na osobnom računalu ili virtualni server na udaljenom računalu. Za izradu ove aplikacije koristio se Laravel programski okvir za PHP i MySql baza podataka za spremanje podataka koji čine programsku podršku. Struktura aplikacije se definirala korištenjem HTML-a, kao stilski jezici za oblikovanje izgleda aplikacije korišteni su CSS i Sass, a JavaScript i jQuery su skriptni jezici korišteni za potrebe funkcionalnosti i interaktivnosti na strani korisnika. Navedene tehnologije čine programsko sučelje. Sama aplikacija se vrti na lokalnom računalu koji pogoni Ubuntu distribucija Linux operacijskog sustava i stoga se koristi LAMP server. Za testiranje i razvoj aplikacije se koristio mrežni preglednik Google Chrome. Za održavanje, verzioniziranje i udaljeno spremanje aplikacije korišten je Git sustav u servisu Bitbucket.

2.1. LAMP

LAMP je internetska razvojna platforma otvorenog izvornog koda te je akronim gdje L stoji za Linux, A za Apache, M za MySQL i P za PHP. Iz razloga što se sastoji od više dijelova, LAMP je zapravo slojevit te je svaki sloj modularan, tj. može se zamijeniti s nekim drugim dijelom. Tako da se na primjer Apache može zamijeniti s Nginx te postaje LEMP. Slika 2.1. prikazuje pregled LAMP komponenti te slanje zahtjeva od klijenta i dobivanje odgovora od servera gdje se mrežni preglednik Firefox koristi za primjer [1].



Sl 2.1. LAMP arhitektura

2.2. HyperText Markup Language (HTML)

Opisni jezik *HyperText Markup Language*, skraćeno HTML, je najosnovniji građevinski element Interneta, on opisuje i definira sadržaj web stranice [2]. To bi značilo da je on opisni programerski jezik koji služi za izradu osnovnog kostura web stranica. Iako HTML nije zapravo programerski jezik jer ne može izvršiti obične funkcije ili operacije poput množenja dva broja, već je on samo opisni jezik.. Riječ *HyperText* koja stoji u njegovom imenu se odnosi na poveznice koje povezuju jednu web stranicu s drugom, one su temeljni dio Interneta.

Svaki HTML dokument se sastoji od osnovnih jedinica koji se zovu HTML elementi koje zatvaraju HTML oznake (engl. *tags*). Pomoću oznaka znamo točno gdje koji element počinje i završava. Unutar jednog elementa se mogu nalaziti i drugi elementi i zbog toga treba biti oprezan prilikom zatvaranja oznaka jer prilikom gniježđenja elemenata može doći do pogreške te nam se sadržaj neće dobro prikazati. Svaki element može imati atribute koji će ga pobliže opisati tipa klasa (engl. *class*) ili jedinstveni identifikator (engl. *id*), preko kojeg kasnije i možemo pristupiti točno tom elementu. Svaki HTML dokument treba prema standardu počinjati s inačicom standarda koja se koristi za izradu HTML dokumenta: <!DOCTYPE html>. Osnovni HTML elementi su zaglavlje (engl. *head*), naslov (engl. *title*), tijelo (engl. *body*), naslovi („h1-h6“, engl. *heading*) i odlomci (engl. *paragraph*).

```
<!DOCTYPE html>
<html>
<head>
        <title>Document</title>
</head>
<body>
        <h1>Heading</h1>
        <p>Paragraph</p>
</body>
</html>>
```

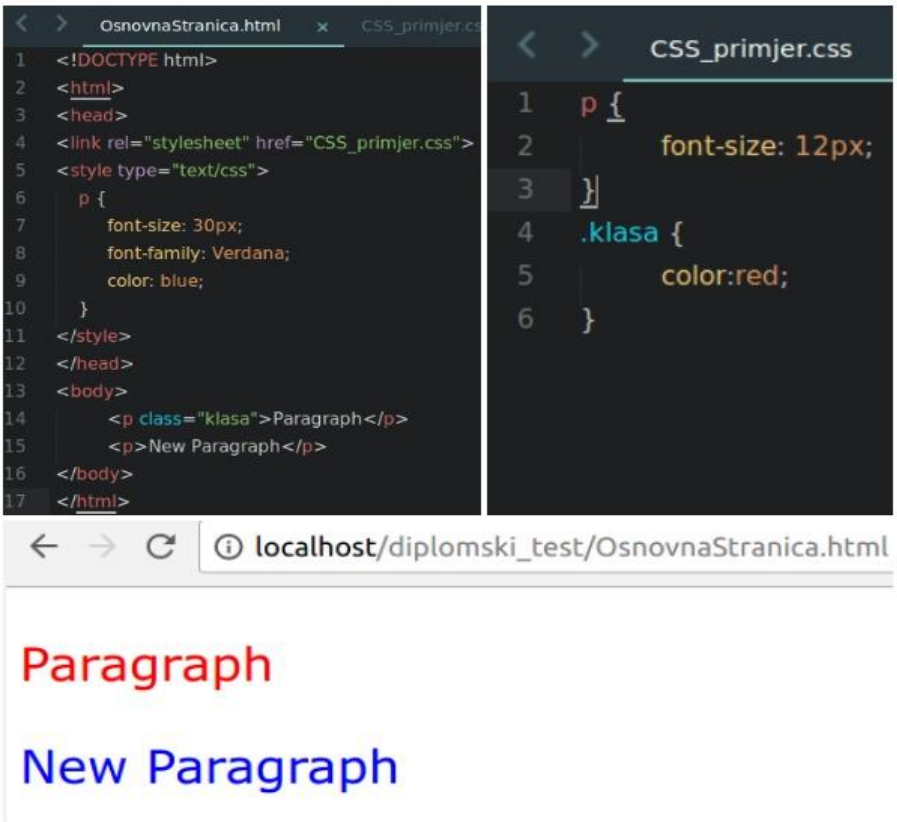
Sl 2.2. Primjer HTML koda

Jednostavni prikaz HTML koda nam daje uvid u opću, sad već objašnjenu, strukturu HTML-a. Kod nam prikazuje blokovski raspored HTML dokumenata i sadrži sve nabrojane osnovne HTML elemente nabrojane u prošlom odlomku (Slika 2.2.).

2.3. Cascading Style Sheets (CSS)

Stilski jezik *Cascading Style Sheets* ili skraćeno CSS je jezik kojim se opisuje prezentacija dokumenta napisana u HTML ili XML-u. CSS opisuje kako bi različiti elementi trebali biti prikazani na ekranu ili nekom drugom mediju [3]. Koristeći CSS možemo odrediti na koji način želimo prikazati sadržaj na web stranici, to radi jer je CSS povezan s DOM-om (engl. *Document Object Model).* Zbog te povezanosti s DOM-om može se brzo i jednostavno pristupiti i stilizirati bilo koji element. Na primjer, ako nam se ne sviđa predefinirani izgled paragrafa „<p>“ oznake ili bilo koje druge slične tekstualne oznake kao na primjer naslov „h3“, može toj oznaci pridodati novi stil kojim ćemo preoblikovati predefinirani stil kao što je veličina i vrsta fonta, „bold“ ili „italic“ način i mnoge druga svojstva.

Postoje dva načina dodavanja CSS-a u HTML datoteku, a tu se putem eksterne „.css“ datoteke koju je potrebno pozvati u HTML datoteci ili pisanjem unutar postojeće „.html“ datoteke pod „<style>“ oznakama. Zbog urednosti koda i standarda koji se prati treba CSS dodati kao eksternu datoteku, nego ju pisati pod stilskim oznakama što može dovesti do nepreglednosti koda.



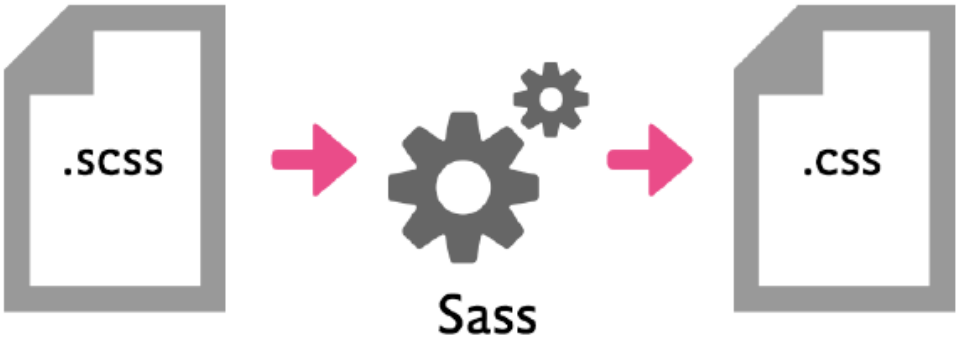
Sl 2.3. Primjer prvenstva kod različitih implementacija CSS koda

Elementi HTML-a mogu biti uređeni na više načina, moguće je utjecati na čitavu porodicu elemenata tipa paragraf, pa se uređuje svaki paragraf element „<p>“ s istim stilom ili se može pristupati elementu putem jedinstvenih klasifikatora kao što su jedinstveni identifikator (engl. *id*) i klasa (engl. *class*). Prilikom uređivanja nekog elementa korištenjem identifikatora tog elementa potrebno je dodati znak: „#“ (engl. *hashtag*) prije naziva identifikatora kako bismo pristupili elementu i uredili ga. Za uređivanje elementa korištenjem klase potrebno je dodati znak: „.“ (točka) prije imena klase kako bismo pristupili tom elementu.

Kod postavljanja CSS stilova HTML elementima postoji hijerarhija postavljanja istih, tako će eksterni stil uvijek biti većeg prioriteta i prepisati postojeći interni stil. Također, ista postoji i kod načina uređivanja elementa, najmanji prioritet tada ima globalni stil porodice elemenata koji će biti prepisan stilom klase tog elementa, a stil klase tog elementa će biti prepisan stilom identifikatora elementa.

2.3.1. Sass

Sass (engl. *Snytactically Awesome Stylesheets*) je CSS pretprocesor (engl. *preprocessor*) koji ga proširuje s modernim konceptima programskih jezika. Pretprocesori uzimaju jedan oblik podataka i pretvaraju ga u drugi oblik, tako se uzima Sass format i od njega se stvara CSS. S pretprocesorom CSS se može strukturirati slično kao i drugi programski jezici kao što su PHP ili JavaScript. Pomoću CSS pretprocesora je moguće koristiti varijable, funkcije, operacije pa čak i gniježđenje sve dok se programira u CSS. Koristeći CSS pretprocesor može se pratiti DRY (engl *Don’t Repeat Yourself*) koncept te praćenjem tog koncepta će se spriječiti ponavljanja u kodu, što je velika mana CSS-a [4].



Sl 2.4. Sass pretvara svoju sintaksu u sintaksu CSS-a

Sass je meta-jezik na vrhu CSS-a koji se koristi za čisti i strukturni opis stila dokumenta s više mogućnosti od tradicionalnog CSS-a. Sass nudi jednostavniju i elegantniju sintaksu za CSS i implementira različite korisne značajke koje su korisne za izradu lako održivih stilskih dokumenata [5].

Sass dokumenti imaju „.scss“ ekstenziju i moraju se kompilirati iz terminala odnosno pomoću naredbenih linija (engl. *command line*). Sass sintaksa je jednostavna i slična CSS-u, samo kao što je već napisano s više mogućnosti. Slika 2.5. prikazuje jednostavan primjer SASS koda, gdje se deklarira varijabla koja definira plavu boju i ta boja se koristi kasnije za promjene boje paragrafa i svih elemenata s klasom „klasa_elementa“.

```
$plava = #OOF

p{
        color: $plava;
}
.klasa_elementa {
        background-color: $plava;
        color: #FFF;
}
```

Sl 2.5. Primjer korištenja varijabli kod Sass-a

2.4. JavaScript

JavaScript, često skraćeno JS, je lagani, objektno orijentirani jezik s mogućnošću upotrebe funkcija, a najbolje je poznat kao skriptni jezik za web stranice, ali se upotrebljava i u mnogim okruženjima bez preglednika [6]. JavaScript se pokreće na strani klijenta i koristi se za programiranje ponašanja web stranice, odnosno odgovara na određene aktivnosti, drugim riječima on je zadužen za interaktivnost i dinamičnost web stranica.

JavaScript se najčešće koristi u HTML dokumentima kako bismo povećali njihovu funkcionalnost i dinamičnost. Za pozivanje JavaScript-a potrebno ga je postaviti između oznaka „<script>" ili ga postaviti u zasebnu „.js“ datoteku i pozvati ga u HTML dokumentu. Kao i kod CSS-a, zbog urednosti i globalnog standarda trebalo bi JavaScript pisati u zasebnoj „.js“ datoteci koju se samo pozove u HTML dokumentu.

Kod JavaScripta, za razliku od CSS-a i HTML-a, se prvi put dolazi do pojma ispravak grešaka (engl. *debugging*). Svaki mrežni preglednik ima mogućnost pristupa JavaScript porukama

o greškama (engl. *error messages*) gdje nam se daje pregled grešaka koje se su pojavile u mrežnom pregledniku.

JavaScript za rad koristi varijable koje se označavaju s ključnom riječi „var*"*. Varijable mogu biti stringovi, brojevi ili nizovi. JavaScript također može manipulirati varijablama putem operacija i funkcija. JavaScript ne mora raditi samo s varijablama već može i direktno raditi s DOM elementima, tako da se na primjer preko klase može pronaći neki element, pristupiti mu i sakriti ga ili se može definirati da se na pritisak dugmeta dobije jako poznati i često ozloglašeni skočni prozor (engl. *popup window*).

JavaScript je učinio web stranice dinamičnima, funkcionalnijima i stvorio bolje korisničko iskustvo, ali zahtjevi za dinamičnijim stranicama su sve veći te su se iz te potrebe razvijali razni JavaScript okviri (pr. React.js) i JavaScript biblioteke (pr. jQuery). Za izradu ovog rada je korišten jQuery te će sljedeće pod poglavlje biti posvećeno njemu.

2.4.1. jQuery i Ajax

jQuery je mala i brza JavaScript biblioteka koja olakšava i ubrzava proces dodavanja, pretraživanja i promjene postojećeg sadržaja web stranice te brisanje, pronalazak, postavljanje i promjenu postojećih atributa [7]. Također, olakšava rukovanje događajima, animacijama, reakcijama i podržava Ajax interakciju.

Potreba za bibliotekom kao jQuery je nastala zbog potrebe optimizacije u radu na različitim mrežnim preglednicima jer različiti mrežni preglednici drugačije implementiraju JavaScript, tako da program može raditi u Google Chrome pregledniku, ali davati grešku i ne raditi u Internet Explorer pregledniku. JQuery je taj problem riješio te radi na svim mrežnim preglednicima jednako. JQuery je nastao suradnjom programera te je objavljen otvorenog izvornog koda i dostupan svima besplatno.

Najveća prednost jQuery koda za razliku od JavaScript koda je to što se može brže pisati, odnosno JavaScript kod često zahtjeva više linija koda (engl. *LOC*) da se izvrši isti zadatak kao u jQuery-u [7]. Na primjer potrebno je provjeriti koji je radio element iz radio grupe elemenata trenutno označen i dobiti vrijednost tog označenog elementa. Prvo je potrebno locirati radio grupu elemenata i provjeriti svaku grupu jednu po jednu za pronalazak onih koji imaju vrijednost „*checked"* te se tada može dobiti njegova vrijednost (Slika 2.6.).

JavaScript:

```
var checkedValue;
var elements = document.getElementsByTagName('input');
for (var i = 0; i < elements.length; i++) {
  if (elements[i].type === 'radio' &&
      elements[i].name === 'some-radio-group' &&
      elements[i].checked) {
    checkedValue = elements[i].value;
    break;
  }
}
```

jQuery:

```
var checkedValue =
  jQuery('input:radio[name="some-radio-group"]:checked').val();
```

Sl 2.6. Usporedba JavaScript i jQuery koda

JavaScript je odličan alat, ali ako je potrebno prikazati ili obrisati podatke iz baze podataka, poslati e-poruku s rezultatima iz forme ili samo poslati formu potrebno je komunicirati s web serverom. Za sve te zadatke potrebno je učitati novu web stranicu, kao na primjer pretraživanje tablice s podacima iz baze podataka i za prikaz željenog rezultata je često potrebno ići na novu stranicu rezultata. Zbog potrebe čekanja na učitavanje nove stranice, što krajnjem korisniku može biti naporno i gubitak vremena, nastala je potreba za responzivnim i brzim web stranicama, odnosno za novom web tehnologijom koja bi nudila to. Iz tog razloga je nastao Ajax. Ajax omogućuje web stranici da zahtjeva i šalje odgovore od web servera i samo osvježi web stranicu bez potrebe učitavanja nove web stranice.

Ajax stoji za asinkroni JavaScript i XML (engl. *Asynchronous JavaScript and XML*) i nije službena tehnologija poput HTML-a, CSS-a ili JavaScript-a. To je pojam koji se odnosi na interakciju više tehnologija, poput JavaScript-a, mrežnog preglednika i web servera, s ciljem dohvaćanja i prikaza novog sadržaja bez učitavanja nove web stranice [8].

Ajax radi na način da JavaScript preko mrežnog preglednika šalje neki zahtjev web serveru, a web server mrežnom pregledniku onda vraća tražene podatke koji se zove odziv te JavaScript uzima te podatke iz odziva i s njima obavlja neki zadatak. Najbolji primjer Ajax tehnologije je Google Maps gdje korisnik može pretražiti mape za neku lokaciju, zoomirati na nju te uhvatiti mapu i pomicati ju lijevo-desno bez da je potrebno ikad učitati novu web stranicu [8].

2.5. PHP

PHP (engl. Hypertext Preprocessor) je opće namjenski (engl. *general-purpose*) skriptni jezik otvorenog izvornog koda koji je specifično namijenjen za razvoj web stranica koji može biti ugrađen u HTML [9]. PHP se prije smatrao skriptnim jezikom za početnike razvoja web stranice te mu je ta reputacije godinama ostala. Dio problema je bio u tome što nije odmah pronašao svoje mjesto kao programski jezik te mu je trebalo dosta vremena da implementira OOP ili objektno orijentiranu praksu (engl. *object oriented programming*) ili mogućnost da se izvodi bez potrebe web servera [10]. Iako je PHP sad evoluirao i postao moderniji jezik, pogotovo s najnovijim izvedbom PHP 7, još uvijek ima lošu reputaciju zbog prošlosti.

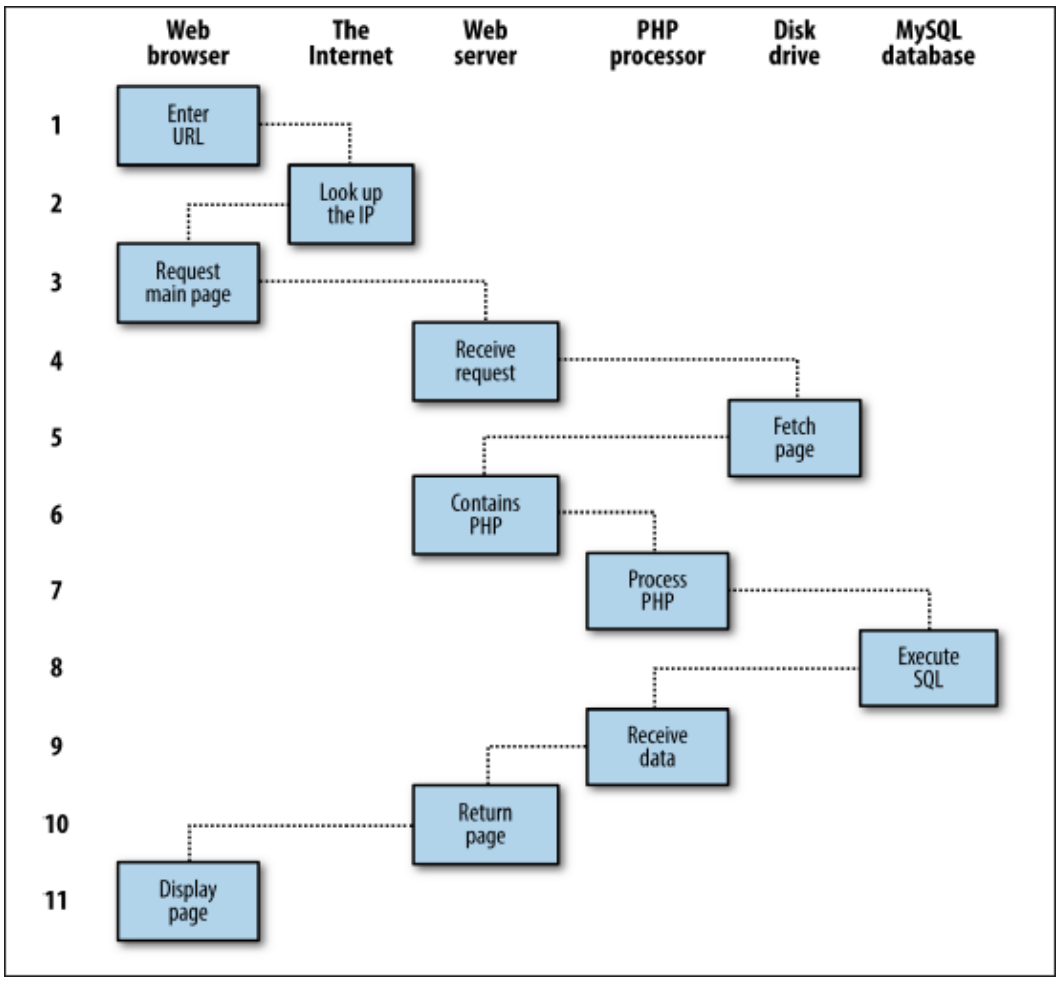
Prema standardu, PHP dokumenti završavaju s ekstenzijom „.php“. Kad web server susretne tu ekstenziju kao zahtjev, automatski će ju predati PHP prevoditelju, iako budući da su web serveri jako konfigurabilni neki web programeri će natjerati PHP prevoditelj da parsira datoteke s „.html“ ekstenzijom. Iako je taj proces nepotreban jer će PHP dokument svejedno samo davati HTML.

U PHP kodu je potrebno koristiti njegove oznake koje su : „<?php“ i „?>“. Oznaka „<?php“ označava početak PHP dijela koda, a „?>“ označava kraj PHP dijela koda i između te dvije oznake je potrebno pisati cijeli PHP kod koji želimo izvoditi. PHP skripta se može pisati u zasebnom dokumentu i samo pozvati u HTML dokumentu, što dovodi do preglednosti koda isto kao što je rečeno za CSS i JavaScript.

PHP se najviše koristi za dinamičke web stranice, većinom gdje je potrebna komunikacija s bazom podataka tipa MySQL. Slika 2.7. predstavlja dijagram dinamičnog klijent/server zahtjev/odziv slijeda akcija. Koraci su sljedeći [11]:

1. Korisnik unosi URL http://primjer.com u mrežni preglednik
2. Mrežni preglednik potraži IP adresu od primjer.com
3. Mrežni preglednik šalje zahtjev prema toj adresi za početnu stranicu web servera
4. Zahtjev preko Interneta dolazi do primjer.com web servera
5. Web server, koji je zaprimio zahtjev, dohvaća početnu stranicu s tvrdog diska
6. S početnom stranicom u memoriji, web server primjećuje da datoteka sadržava PHP skriptu i prosljeđuje stranicu PHP prevoditelju
7. PHP prevoditelj izvršava PHP kod

8. Jedan dio PHP koda sadržava MySQL naredbe (engl. *statements*) koje PHP prevoditelj prosljeđuje MySQL mehanizmu baze podataka (engl. *database engine*)
9. MySQL baza podataka vraća rezultate naredbe PHP prevoditelju
10. PHP prevoditelj vraća rezultat izvršenog PHP koda, zajedno s rezultatima baze podataka MySQL web serveru
11. Web server vraća stranicu klijentu koji ju je zatražio te ju prikazuje



Sl 2.7. Klijent/server zahtjev/odziv slijed akcija

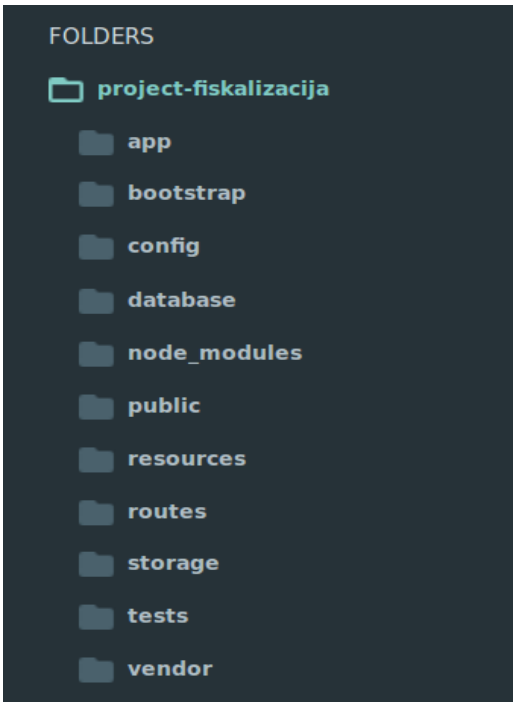
PHP 7 je danas postao standard, na primjer na operacijskom sustav Ubuntu 16.04, paket *php7.0-cli* je sad standardni repozitorij. PHP 7 je uveo nove operatore „<=>“ nazvani operatori svemirskog broda (engl. *spaceship operator*). Bazira se na PHPNH projektu (engl. *PHP Next-Gen*), koji je provodio Zend kako bi povećao brzinu PHP aplikacija, zbog toga su se performanse podigle između 25% do 75%, što znači da samo nadogradnjom verzije bez mijenjanja koda se dobiju veće performanse [8].

2.6. Laravel

PHP se tijekom godina, kao što je napisano u prethodnom poglavlju, dosta mijenjao i napredovao, zbog toga postoje razne mogućnosti pisanja PHP i dopušta svaki način pisanja. Programer jedan dio PHP koda može pisati proceduralno, drugi dio objektno orijentirano, što pisanje čistog PHP koda čini težim i programer mora dobro znati organizirati kod. Programer dok piše PHP kod će često pisati funkcije koje su već definirane u raznim programskim okvirima i stoga gubiti vrijeme i pisati puno više linije koda kako bi ponovo napravio istu stvar. Za vrijeme pisanja čistog PHP koda programeri moraju jako paziti na sigurnost, na primjer zaštitu od *SQL Injection*-a i *XSS*-a. Svi ti razlozi su privukli ljude programskih okvirima, koji nude stabilnost, dosljednost i štede puno vremena. Razvijeno je puno programskih okvira za PHP poput Symphony-a, Laravel-a, Yii-a i drugih, a u ovom radu će se pisati o Laravel-u koji se koristio za izradu aplikacije ovog diplomskog rada.

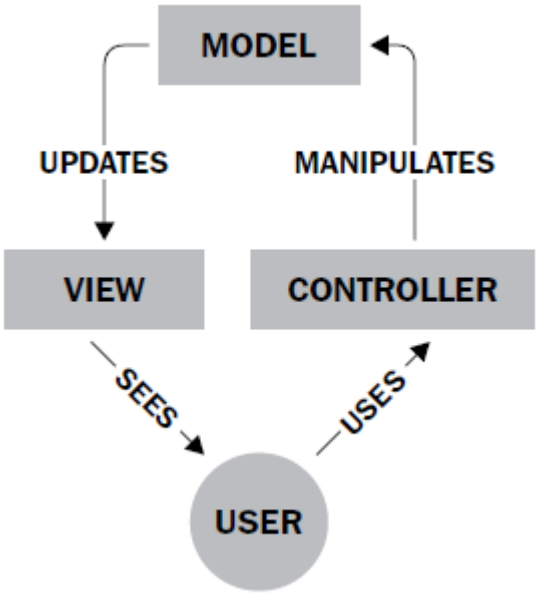
Laravel je programski okvir za razvoj web aplikacije koji pokušava cijeli proces razvoja web aplikacije olakšati, tako što pojednostavljuje rutinske zadatke poput autentikacije (engl. *authentication*), spremanja u privremenoj memoriji (engl. *caching*), rad sa sesijama (engl. *session*) i usmjeravanja (engl. *routing*) [13].

Laravel koristi jednu projektnu strukturu, odnosno strukturu mapa, koja se uvijek primjenjuje prilikom razvijanja web aplikacija (Slika 2.8.). U toj strukturi svaka datoteka ima točno svoje predodređeno mjesto i time Laravel tjera programere da koriste tu strukturu kako bi cijeli projekt bio organiziran na Laravel način. Tri najvažnije mape u toj strukturi su: „app“, „public“ i „vendor“. „App“ mapa sadržava modele, poglede (engl. *views*) i kontrolere (engl. *controllers*), tu većina koda piše. „Publi“c mapa je jedina mapa koju korisnici vide kakva je, u njoj se nalazi sav HTML, CSS, JavaScript, slike i druge datoteke koje korisnici trebaju vidjeti kad posjete web stranicu. „Vendor“ je mapa za izvorni kod Laravela i ovisnosti (engl. *dependencies*) i sve priključke (engl. *plugin*) koji sadržavaju dodatne unaprijed upakirane funkcionalnost (engl. p*repackaged funcitionality*).



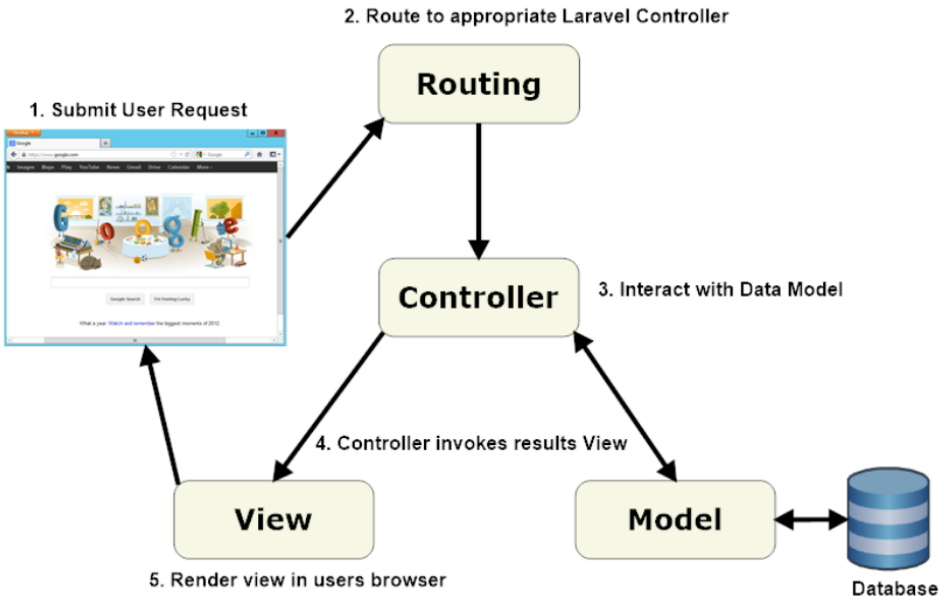
Sl 2.8. Projekta struktura Laravel projekta

Laravel prati model-pogled-kontroler ili skraćeno MVC (engl. *model-view controller*) arhitekturni obrazac. MVC se sastoji od tri komponente kao što se može vidjeti na slici 2.9. [14]. Prva komponenta je model koji sadrži glavne podatke kao što su informacije od objekata iz baze podataka i SQL upiti. Svi podaci se dobivaju iz modela iako se on ne može direktno pozvati, nego je kontroler taj koji od modela zatraži podatke, a model obrađuje taj zahtjev i vraća podatke kontroleru. Druga komponenta, to jest pogled sadrži korisničko okruženje aplikacije odnosno prikazuje podatke koje dobije od modela. U web aplikacijama pogled se sastoji od HTML-a, CSS-a, JavaScripta, XML-a, JSON-a i tako dalje. Pogled je jedina komponenta vidljiva od strane korisnika. Zadnja komponenta je kontroler koji sadrži sve što je bitno za kontrolu programa i odgovaran je za njegov tijek. U web aplikacijama kontroler predstavlja prvi sloj koji se poziva kad korisnik unese URL u mrežni preglednik. Kontroler još i upravlja korisničkim zahtjevima kao što su *GET* i *POST* zahtjevi. Kontroler najčešće poziva određeni model koji je potreban za trenutni zadatak i zatim odabire i prikazuje odgovarajući pogled.



Sl 2.9. MVC arhitekturni obrazac

Prilikom korisničke interakcije s web aplikacijom izrađenom u Laravel-u, mrežni preglednik šalje zahtjev web serveru koji taj zahtjev prosljeđuje Laravel usmjerivačkom mehanizmu. Laravel usmjeritelj prima taj zahtjev i šalje ga odgovarajućem kontroleru. Kontroler tada ili odmah prikazuje pogled koji sadržava HTML, CSS i tako dalje ili prvo komunicira s modelom kako bi dobio odgovarajuće podatke koje treba prikazati (Slika 2.10.).



Sl 2.10. Laravel komponente

2.7. MySQL

Baza podataka je strukturirana kolekcija zapisa podataka spremljenih na računalnom sustavu i organizirana na takav način da se može brzo pretražiti i da se informacije mogu brzo dobiti. MySQL je relacijska baza podataka otvorenog izvornog koda ili skraćeno RDBMS (engl. *relational database management system*). SQL u MySQL stoji za strukturirani upitni jezik (engl. *structured query language*). Taj jezik se temelji na izrazima iz engleskog jezika, tako da se zahtjevi na bazu podataka mogu jednostavno pisati: „SELECT username FROM users WHERE email ='example@mail.com';“

MySQL baza podataka se sastoji od jedne ili više tablice te svaka tablica ima jedan ili više zapisa ili redaka. Ti zapisi ili redci sadržavaju više stupaca ili polja koji sadržavaju same podataka. Prema slici 2.11. možemo vidjeti kako se tablica student sastoji od stupaca ID, broj_indeksa, ime, prezime, ispit, ocjena, opis i tri retka.

Sl 2.11. Primjer MySQL tablice

MySQL bazi podataka se može prilaziti preko PHP programskog jezika, sustava *phpMyAdmin* ili preko sučelja naredbenih linija, skraćeno CLI (engl. *command line interface*). *PhpMyAdmin* je u PHP napisano sučelje koje nudi najčešće moguće operacije kao što su upravljanje bazom podataka, tablicama, zapisima, relacija, indeksima, dozvolama te pisanje i izvođenje SQL naredbi kroz grafičko korisničko sučelje ili GUI (engl. *graphical user interface*).

2.8. Git i BitBucket

Git je sustav upravljanja inačicama ili skraćeno VCS (engl. *version control system*) koji bilježi izmjene u datoteci ili skupu datoteka tijekom vremena, tako da se kasnije mogu pregledati određene prijašnje verzije. Najviše se koristi za upravljanje izvornim kodom tijekom razvoja programske podrške, ali se može koristi i za praćenja promjena drugih datoteka. Gitom se upravlja preko terminala gdje korisnik upisuje naredbe poput „git init“ za izradu praznog Git repozitorija, „git push“ za zamjenu starih datoteka novima, ali se stare čuvaju pod prijašnjom verzijom, „git

pull“ za dohvaćanje podataka s repozitorija i mnogi drugi. Git je osim za osobne potrebe upravljanja inačicama stvore i za olakšavanje suradnje u timu ljudi koji rade na istom projektu, tako što ima grananje (engl. *branch*). Grananje omogućuje stvaranja različitih inačica istog projekta kako bi se kod mogao odvojiti na dio koda za stvaranje novih značajki ili eksperimentiranje prije puštanja u rad u glavnoj „master“ inačici.

Bitbucket je web usluga posluživanja u vlasništvu Atlassiana, koja se koristi za izvorni kod i razvojne projekte koji koriste Mercurial ili Git sustav kontrole nadzora. Bitbucket nudi sve usluge git-a, ali i mnoge druge kao što je mrežno bazirano grafičko sučelje koje omogućava integraciju s prijenosnim i stolnim računalima. BitBucket još nudi i desktop aplikacije koja u sebi sadržava sve mogućnosti Git sustava.

3. STRUKTURA APLIKACIJE

U ovom poglavlju je predstavljeno rješenje web aplikacije za fiskalizaciju kao i korištene tehnologije i alati koji su omogućili izradu web rješenja. Aplikaciju čine dva dijela: programsko sučelja (engl. *frontend*) odnosno što korisnik vidi i koristi i programska podrške (engl. *backend*) to jest pozadinskog dijela aplikacije koji omogućuje ono što korisnik vidi. Programsko sučelje je napisano u HTML, CSS i JavaScript tehnologiji, a programska podrška se sastoji od MySQL baze podataka i koda u PHP programskom jeziku napisanog unutar Laravel programskog okvira. Rad fiskalizacije i komunikacije s FINA serverom će biti detaljno opisana u poglavlju istoimenoga naziva.

3.1. Programska podrška

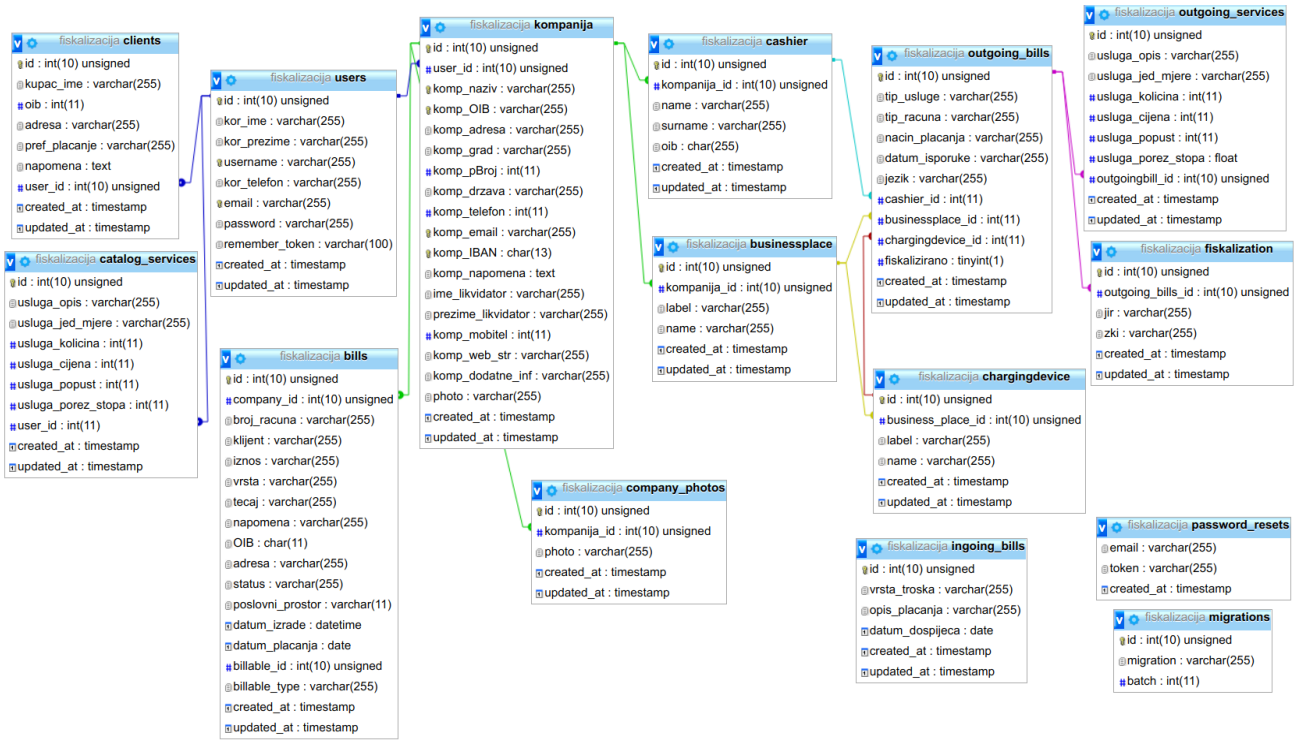
Programska podrška se sastoji od poglavlja za bazu podataka gdje je opisana struktura baze podataka i povezanost tablica. Zatim slijede poglavlja koja opisuju svaki od najbitniji koncepata Laravel-a za programsku podršku. Svaki je objašnjen kako radi i najvažnije kako se implementirao unutar ove aplikacije s prikazom odgovarajućeg dijela koda. Najbitniji koncepti su: migracije, modeli, rute, kontroleri, veze, međuslojevi i zapisi. Dodatno poglavlje je poglavlje autentikacije koje predstavlja registraciju i prijavu korisnika.

3.1.1. Baza podataka

Za povezivanje na bazu podataka u Laravel-u je potrebno izmijeniti dokument okruženja „.env“ koji zahtjeva unošenje tip povezivanja na bazu podataka, domaćina (engl. *host*), naziva baze podataka, korisničko ime i lozinku. Nakon unosa tih parametara nema potrebe za daljnjim unosom jer Laravel pamti taj unos i koristi ga kroz cijelu aplikaciju.

Baza podataka je naziva fiskalizacije i izrađena je u MySQL-u. Baza podataka se sastoji od tablica: users, bills, outgoing_bills, ingoing_bills, catalog_services, outgoing_services, kompanija, businessplaces, chargingdevices, clients, cashier, fiskalization, company_photos, migrations i password_resets. Od toga tablicu „migrations“ Laravel stvara prilikom prve migracije. Slika 3.1. prikazuje međusobne ovisnosti tablica u bazi podataka, ali nisu prikazane sve povezanosti tablica jer su neke napravljene programski preko Laravel-a. Kao što se može vidjeti iz slike 3.1. svaki korisnik ima svoju kompaniju te usluge i klijente koji pripadaju samo njemu. Preko kompanije su onda povezani računi koji se dijele na ulazne i izlazne račune. Izlazni računi su oni računi koji se izdaju kupcu. Izlazni računi su povezani s izlaznim uslugama. To su usluge koje se spremaju za svaki račun kako bi se znalo koja usluga pripada kojem računu. Također, osim

što su izlazni računi povezani s izlaznim uslugama, povezani su još i s operaterima, naplatnim uređajima i poslovnim prostorima. Poslovni prostor i operater su povezani s kompanijom jer moraju biti jedinstveni za jednu kompaniju. Naplatni uređaj je još povezan s poslovnim prostorom jer on mora biti jedinstven za poslovni prostor. Izlazni račun je još povezan s tablicom fiskalizacije. Ulazni računi nisu povezani niti s jednom drugom tablicom, osim s računima, ali to je programski riješeno, kao i za izlazne račune. Kompanija još ima dodatnu tablicu gdje se spremaju slike koje predstavljaju logo kompanije.



Sl 3.1. Baza podataka aplikacije sa svim relacijama

3.1.2. Migracije

Migracije su bazama podataka, slično što je git programskom rješenju, način verzioniziranja. Migracije omogućuju laku izmjenu i dijeljenje sheme baze podataka. Migracije koriste Laravel „Schema Builder“ fasadu kako bi stvorili, izmijenili ili obrisali staru shemu tablice. Kako bi se migracija napravila u terminalnu se koristi Artisan naredba „php artisan make migration create_table_name“. Norma je naziv tablice staviti u množini, ali ne i pravilo. Svaka nova migracija će biti smještena u mapu „database/migrations“. Sve migracije imaju vremensku oznaku (engl. *timestamp*) u imenu zbog određivanja redoslijeda migracija. Artisan naredbi za izradu migracija se mogu i dodati opcije. Na primjer ako korisnik želi može definirati naziv tablice koju želi napraviti treba samo dodati naziv tablice u Artisan naredbu: „php artisan make migration create_table_name --table=table_name“. Klasa migracije se sastoji od dvije metode: „up“ i

„down“. Metoda „up“ se koristi za dodavanje novih tablica, stupaca ili indeksa bazi podataka, dok „down“ metoda odradi suprotan posao od „up“ metode. Unutar obje navedene metode se koristi Laravel-ov „Schema Builder“ kako bismo stvorili tablicu ili je izmijenili. Primjer izrade tablice se može vidjeti na stvaranju tablice za blagajnika „cashiers“ (Slika 3.2.).

```
public function up() {
    Schema::create('cashiers', function (Blueprint $table) {
        $table->increments('id');
        $table->integer('company_id')->unsigned();
        $table->foreign('company_id')->references('id')->on('kompanija')->onDelete('cascade');
        $table->string('name');
        $table->string('surname');
        $table->integer('oib');
        $table->timestamps();
    });
}
```

Slika 3.2. Migracija za stvaranje tablice klijenata

Unutar metode „up“ se koristi metoda „Schema::create“ kojoj se predaje željeni naziv tablice i objekt „Blueprint“ koji se koristi za definiranje nove tablice „$table“. Kako bi se napravila tablica „cashiers“ potreban je identifikator „id“, strani ključ koji referencira na kompaniju, ime, prezime i OIB blagajnika, kao dodatan parametar se često dodaje „timestapams()“. Identifikator „id“ koji se automatski povećava i ne postavlja od strane korisnika se dodaje metodom „increments('id')“. Postoje razni tipovi podataka koji se mogu postaviti stupcu, sve od broja pa do teksta i mogu se naći u Laravel dokumentaciji. Prilikom izrade tablice „cashiers“ su se koristili samo „string“ za ime i prezime i „integer“ za OIB. Izrada stranog ključa korištenjem Laravel „Schema Buildera“ je jednostavna, samo je potrebno napraviti element na kojem će se referencirati željeni stupac tablice s kojim je povezan. U ovom slučaju se napravio „company_id“ koji je referencirao stupac „id“ u tablici kompanije. Metoda „timestamps()“ stvara dva stupca „created_at“ i „updated_at“ koje nije potrebno programski popunjavati već se popunjavaju automatski prilikom unosa ili izmjene. Unutar metode „down“ se samo definiralo da ako postoji „cashiers“ se obriše (engl. *drop*).

3.1.3. Model

Za svaku tablicu, samim time i migraciju, potreban je i model koji definira tablicu. Model dozvoljava na jednostavan način slanje upita odgovarajućoj tablici, kao i dodavanje novih zapisa u tablicu. Modeli su najpoznatiji po definiranju veza (engl. *relationship*) u njima, veze će bit kasnije detaljno objasnjene. Svi modeli su smješteni u mapi „app“, iako ih se može staviti bilo

gdje. Instanca modela se izrađuje Artisan naredbom: „php artisan make: Model ModelName“. U naredbu se upisuje ima modela koji želimo stvoriti, kao što je na primjer model „outgoing_bills“ (Slika 3.3.). Nakon stvaranja modela potrebno je definirati tablicu s kojom povezan. U slučaju da se po konvenciji koristio „snake case“ koji pretpostavlja da je tablica množina naziva modela, onda nije potrebno definirati naziv tablice. Ako se nije koristila konvencija, definira se svojstvo tablice koje određuje naziv tablice: „protected $table = 'table_name“. Nakon što se po potrebi definirao naziv tablice, potrebno je definirati koji su stupci popunjivi od strane korisnika kad šalje zahtjev. To se radi na način da se u svojstvo popunjivosti (engl. *fillable*) definira koje su svi stupci mogu popuniti iz zahtjeva. S time je gotova glavna forma modela. Svaki Eloquent model se može gledati kao moćan način izrade zahtjeva (engl. *query builder*) koji omogućuje lako i brzo slanje upita tablici s kojom je povezan model. Na modelu se mogu pozivati Eloquent metode kako bismo pravili zahtjeve. Tako da na primjer ako želimo dobiti sve zapise iz tablice, koristi se metoda „all()“ na modelu te tablice. Također, jer je Eloquent moćan alat, on omogućuje i dodavanje ograničenja na upit. Tako da ako se želi dobiti na primjer svaka usluga čiji je iznos manji od jedne stotine, koristi se „where()“ ograničenje te potom „get()“ koji zapravo dohvaća: „$services = \App\Catalog_services::where('iznos', '<', '100')- >get()“.

```
class Outgoing_bill extends Model {
   protected $fillable = [ 'tip_usluge', 'tip_racuna', 'nacin_placanja', 'datum_isporuke', 'jezik'];
   public function bills() {
         return $this->morphMany('App\Bill', 'billable');
   }
   public function services() {
      return $this->hasMany('App\Outgoing_service', 'outgoingbill_id');
   }
   public function cashier() {
      return $this->belongsTo('App\Cashier');
   }
}
```

Sl 3.3. Model izlaznih računa

3.1.4. Veze

Tablice baze podataka su često povezani na određeni način, kao što je na primjer korisnik povezan s kompanijom koju je napravio ili izlazni račun povezan s računom. Eloquent olakšava rad s vezama tako što podržava različite vrste veza: „OneToOne“, „OneToMany“, „ManyToMany“, „hasMany“ i „Polymorphic“. Eloquent veze se definiraju u modelima (Slika 3.3.). Poput samih Eloquent modela, veze služe kao moćni načini izrade zahtjeva. Definiranje veza se povećava mogućnost nizanja metoda i mogućnosti samih upita.

Veza „OneToOne“ definira takvu vrstu veze gdje je jedan model, na primjer korisnik, poveza samo s jednim drugim modelom kao što je kompanija. Ova veza se definira na način da se u model korisnika stavi metoda „hasOne()“ koja označava da korisnik ima samo jednu kompaniju: „public function kompanije() { return $this->hasOne('App\Kompanija'); }“. Metoda „hasOne ()“ kao argument prima naziv modela i automatski pretpostavlja da ima naziv stranog ključa zadan prema normi, što bi u ovom slučaju bio „user_id“. U slučaju da nije tako, metoda „hasOne ()“ može primiti osim naziva modela još jedan argument koji predstavlja naziv stranog ključa. Nadalje, Eloquent također pretpostavlja da se lokalni ključ u tablici naziva „id“, tako da ako to nije slučaj potrebno je metodi „hasOne ()“ kao treći argument predati naziv lokalnog ključa. Kako bi veza radila u model kompanije je potrebno definirati kome pripada: „public function user() { return $this->belongsTo('App\User'); }“. Metoda „belongsTo()“ ima iste argumente kao i metoda „hasOne“ te ih može primiti do tri, ovisno o potrebi. Sada se može na jednostavan način putem Eloquent dinamičkih svojstava (engl. *dynamic properties*) dobiti povezani zapis. Dinamička svojstva omogućuju pristup metodi veze koja je definira na samom modelu: „$kompanija= User::find(1)->kompanije“. Ovaj izraz vraća kompaniju koja je povezana s korisnikom kojeg smo odabrali.

Veza „OneToMany“ se koristi kada se definira veza gdje jedan model posjeduje veliki broj drugih modela. Na primjer, model izlaznog računa može imati skoro pa beskonačan broj izlaznih usluga, ali izlazna usluga može imati samo jedan izlazni račun. U modelu se ta veza definira na način da se u modelu izlaznog računa definira metoda „hasMany()“ koja prima kao argument naziv modela te po potrebi naziv stranog i lokalnog ključa. Primjer „hasMany()“ metode iz modela izlaznih računa „Outgoing_bill“: „public function services() {return $this->hasMany('App\Outgoing_service', 'outgoingbill_id'); }“. Ovim se definirala veza koja znači da izlazni račun može imati puno izlaznih usluga. Kao što se vidi na primjeru, ovdje je kao drugi argument postavljen naziv stranog ključa jer nije postavljen po konvenciji. Kako bi se upotpunosti definirala veza potrebno je u modelu izlaznih usluga dodati vezu na izlazne račune. Veza na izlazne račune u modelu izlaznih uslugama se radi korištenjem iste metode kao i u „OneToOne“ vezi, „belongsTo()“ metodom. Primjer „belongsTo()“ metode u modelu izlaznih usluga „Outgoing_services“: „public function outgoing_bill() {return $this->belongsTo ('App\Outgoing_bill'); }“. Ovom vezom je definirano da svaka izlazna usluga ima samo jedan izlazni račun. U slučaju kad bismo htjeli definirati „ManyToMany“ vezu, umjesto „belongsTo()“ bi se stavila „belongsToMany()“ metoda i to bi onda značilo da svaka izlazna usluga može imati više izlaznih računa, kao što izlazni račun može imati usluga. Korištenjem Eloquent-a se može

doći do svih izlaznih usluga odabranog računa. Tako da ako na primjer želimo pronaći koje sve usluge ima prvi račun: „$outgoing_services = \App\Ougoing_bills::find(1)- >services;“. Ova naredba će vratiti kolekciju svih usluga kojima pripada te kako bismo ih sve ispisali, potrebno je koristiti „foreach“ petlju.

Veza „Polymorphic“ omogućuje modelu, to jest tablici, da pripada u više od samo jednog drugog modela. Primjer toga su računi. Svaki izlazni i ulazni račun imaju parametre koji su im jednaki, kao što su: broj računa, klijent, iznos, napomena, adresa, datum izrade, i slično. Također ulazni i izlazni računi imaju parametre koji se razlikuju te ih zato nije moguće staviti u samo jednu tablicu. Iz razloga kako se ne bi u dvije tablice ponavljali isti parametri, odnosno stupci, postoje polimorfske veze. Polimorfske veze omogućuju da jedna tablica sadrži sve stupce koji su im zajednički, ali zato ima dva dodatna stupca koja označuju pripadnost određenom tipu. Primjer u aplikaciji je tablica računa koje sadrži sve zajedničke stupce tablica: izlazni i ulazni računi. Tablica računa stoga u sebi sadržava stupce „billable_id“ i „billable_type“. Ta dva stupca su najvažnija za polimorfsku vezu jer „billable_id“ sadržava „id“ izlaznog ili ulaznog računa, a „billable_type“ stupac određuje vlasnički model zapisa kojem treba vratiti upit kad se koristi „billable“ veza. U modelu računa „bills“ se definira metoda „morphTo()“ kako bi se znalo da taj model sadrži „billable_id“ i „billable_type“ i ta metoda ne prima argumente: „public function billable() { return $this->morphTo(); }“. U modelu izlaznih ili ulaznih računa se definira metoda „morphMany()“ koja prima kao prvi argument naziv tablice koja sadrži „morphTo()“ i kao drugi argument naziv te veze, što je u ovom slučaju „billable“. Primjer metode „morphMany()“ se može vidjeti u modelu „Outgoing_bill“ ili u modelu „Ingoing_bill“, sintaksa je ista: „public function bills() { return $this->morphMany('App\Bill', 'billable'); }“. Pozivanje te veze kako bismo dobili na primjer parametre iz tablice računa za određeni izlazni račun se radi na način da se prvo pronađe izlazni račun: „$outgoing_bill = \App\Outgoing_bill::find(1);“. Nakon što se pronađe izlazni račun poziva se polimorfska veza kako bi se dobili parametri (stupci) računa: „$bill=$outgoing_bill->bills;“. U varijabli „$bill“ se sada nalaze svi parametri računa koji su povezani s izlazni računom kojeg smo pronašli navedenom „find()“ metodom.. Obrnuti postupak vrijedi, gdje se na temelju računa mogu dobiti parametri izlaznog računa. Prvo se na isti način pronađe račun „$bill = \App\Bill::find(1);“ i onda se na tom računu poziva poliformska veza „$outgoing_bill=$bill->billable“ kako bi se u varijabli „outgoing_bill“ dobili svi parametri izlaznog računa povezanog s računom koji smo dobili prethodnim upitom.

3.1.5. Rute

Laravel rute (engl. *routes*) se nalaze u mapi „routes“ i automatski se dodaju u programski okvir. Mapa „routes“ se sastoji od tri datoteke: „web.php“, „api.php“ i „console.php“. U datoteci „web.php“ su definirane rute za web sučelje te rute su dodijeljene grupi međusloj web i pružaju mogućnosti kao što su sesije i CSRF zaštite. Rute unutar „api.php“ datoteke pripadaju api međusloj grupi. Za većinu aplikacija, pa tako i za ovu koristi se „web.php“ datoteka. Unutar te datoteke se registriraju rute za HTTP metoda kao što su: POST, PUT, GET, DELETE. Također Laravel ima i svoje metode koje se mogu postaviti umjesto onih HTTP-a: „match“, „any“ i „resource“. Kako bismo dobili popis svih ruta korištenih u aplikaciji u terminal se može upisati Artisan naredba: „php artisan route:list“ koja vraća popis ruta.

Svaka ruta se sastoji od HTTP metode, linka, odnosno stranice za koju je ta ruta i funkcije. Funkcija može biti napisana direktno u mapi ruta (datoteci „web.php“) što nije optimalno ili u kontroleru i samo se referencira ta funkcija u rutama. Rute mogu biti statičke ili dinamičke, ako su statičke onda ne primaju nikakvu vrijednost poput identifikatora, a ako su dinamičke onda primaju vrijednost. Slika 3.4. prikazuje primjer rute koja je statička te ima GET metodu i dinamičke rute s POST metodom.

```
Route::resource('client', 'ClientController');

Route::get('racuni', 'BillsControler@index');

Route::post('racuni/store', 'OutgoingBillController@store');
```

Sl 3.4. Ruta za klijenta tipa „resource“ i POST i GET rute za računa

Na rutama se može direktno definirati međusloj, tako da se ruti doda metoda „middleware()“. Metoda „middleware()“ kao argument prima naziv međusloja. Također, ruta se može omotati međuslojem ili s više međuslojeva, na način da se ispiše „Route::middleware([jedan, dva])…“ i unutar njega se onda upiše na primjer GET ruta koja će koristiti međusloj jedan i dva.

3.1.6. Kontroler

Budući da definiranje načina rukovanja logikom u rutama nije dobro za organizaciju i čitljivost koda, logika je napravljena u kontroleru. Kontroler upravlja korisničkim zahtjevima kao što su POST, GET, PUT, DELETE i drugi. Kontroleri su spremljeni u mapi „app/Http/Controllers“. Stvaranje kontrolera se radi Artisan naredbom: „php artisan make:controller NameController“. Na tu naredbu je moguće dodati i argument „--resource“ i s time će Laravel pretpostaviti da radimo tipičan CRUD i automatski napraviti „resource“ rutu za

taj kontroler i kontroler popunjen standardnim metodama. Kad se napravi „resource“ kontroler nije potrebno na rutu dodavati metodu kontrolera kako bismo znali koju da pokrene već se koristi predefinirani način. Tako da ako želimo na primjer spremiti novog klijenta, to bi se stavilo u „store()“ metodu kontrolera, jer to je predefinirana ruta za stvaranje novog (POST). Inače, ako se ne koristi „resource“ kontroler, što često zna i biti slučaj, svakoj se ruti dodaje metoda kontrolera kako bi ruta znala što treba pokrenuti jer se kontroler prvi pokreće.

Najčešće korištene metode kontrolera, ako se naziva prema konvenciji, su:

- „index()“
- „create()“
- „store()“
- „edit()“
- „destroy()“.

Unutar metode „index()“ se definira početna stranica koju korisnik vidi kada pristupi određenoj ruti, odnosno ono što se prvo pokreće. Unutar nje se uvijek vraća pogled (engl. *view*) i često se s njome i šalju varijable kako bismo prikazali podatke iz baze podataka na pogledu. Za prosljeđivanje varijabli se koristi „compact()“ metoda u koju se prosljeđuje varijabla napunjena podacima iz baze podataka i kasnije parsira i prikazuje u pogledu.

```
public function index() {
    $user = Auth::user();
    $catalog_services = $user->catalog_services()->get();
    return view('pages.service', compact('catalog_services'));
}
```

Sl 3.5. Metoda „index()“ za učitavanje i slanje podataka pogledu usluga

Metoda „create()“ se koristi za dodavanje novih modela, odnosno zapisa u tablicu i vraća pogled za stvaranje novog modela, ali nije zadužena za logiku koja služi za spremanja novog. Za to je zadužena metoda „store()“.

Metoda „store()“ uvijek ima argument zahtjeva (engl. *request*) koji u sebi sadržava sve podatke poslane forme kontroleru. Preko varijable zahtjeva se može pristupiti cijeloj formi i spremiti ju korištenjem metoda „create()“ ili „save()“. Prije spremanja je potrebno definirati instancu modela u koji želimo spremiti, na primjer: „$client = new Client;“. Zatim se preko varijable zahtjeva dohvate svi ili neki parametri poslani iz forme i spreme u bazu podataka s „create()“ ili „save()“ metodom.

```
public function store(ServicesRequest $request) {
   $user = Auth::user();
   $catalog_service = $user->catalog_services()->create($request->all());
   return Redirect::to('service');
}
```

Slika 3.6. Metoda „store()“ za spremanje usluge

Metoda „edit()“ služi za prikaz pogleda za izmjenu postojećeg zapisa. Za argument najčešće prima identifikator zapisa i pretražuje odgovarajuću tablicu s tim identifikator metodom „find()“ i sprema sve argumente tog zapisa u varijablu koju onda vraća pogledu. Pogled tu varijablu parsira i prikazuje korisniku sve podatke tog zapisa kako bi znao koji želi izmijeniti. Izmjenu zapisa odrađuje metoda „update()“ koja kao argument isto najčešće prima identifikator i zatim preuzima sve poslane podatke iz forme preko „Input::get()“ i zatim se metodom „save()“ izmjeni zapis. Metodu „create()“ nije moguće koristit jer ona samo dodaje novi zapis, dok „save()“ može dodati novi i izmijeniti postojeći zapis.

```
public function update($id) {
      $catalog_service = Catalog_service::find($id);
      $catalog_service->usluga_opis       = Input::get('usluga_opis');
      $catalog_service->usluga_jed_mjere      = Input::get('usluga_jed_mjere');
      $catalog_service->usluga_kolicina = Input::get('usluga_kolicina');
      $catalog_service->usluga_cijena = Input::get('usluga_cijena');
      $catalog_service->usluga_popust = Input::get('usluga_popust');
      $catalog_service->usluga_porez_stopa = Input::get('usluga_porez_stopa');
      $catalog_service->save();
      return Redirect::to('service');
   }
```

Slika 3.7. Metoda „update()“ za izmjenu usluga

Za brisanje zahtjeva se koristi metoda „destroy()“. Ta metoda kao argument prima identifikator kao i „update()“ i „edit()“. Zatim se pronađe instanca modela s tim identifikatorom i na njoj pozove metoda „delete()“ koja briše postojeći zapis.

Međusloj se može definirati izravno u kontroleru u metodi „_construct()“. Ta metoda se poziva prije svih ostalih metoda, pa i prije „index()“ metode. Upravo iz tog razloga je najpogodnija za pozivanje međusloja. Međusloj se poziva na varijabli „$this“: „$this->middleware()“ koji prima naziv međusloja. Osim za pristup kontroleru, međusloj se može pozvati samo za određenu metodu kontrolera korištenjem „only()“ metode koja prima naziv metode kontrolera ili se može pozvati na sve metode kontrolera osim jedne metodom „except()“ koja prima isti argument kao i „only()“.

3.1.7. Međusloj i zahtjevi

Međusloj pruža prikladan mehanizam filtriranja HTTP zahtjeva. Međusloj se većinom koristi kao zaštitni sloj aplikacije prije obrade zahtjeva. Na primjer, Laravel automatski uključuje međusloj za provjeru je li korisnik aplikacije registriran. Ako nije registriran, međusloj će ga preusmjeriti stranici za prijavu, ali ako je registriran dozvolit će mu zahtjev. Svaki međusloj se nalazi u mapi „app/Http/Middleware“. Za stvaranje međusloja koristi se Artisan naredba: „php artisan make:middleware NameOfMiddleware“. Laravel napravi klasu međusloja s imenom koje smo upisali i napravi funkciju u koju se upisuje što želimo da međusloj radi. Primjer međusloja u aplikaciji je „HasCompany“ međusloj koji provjerava ima li korisnik dodanu kompaniju, jer ako nema kompaniju ne može pristupiti niti jednoj drugoj stranici osim ono koja je određena za dodavanje kompanije (Slika 3.8.).

```
class HasCompany
{
   public function handle($request, Closure $next) {
       $user = Auth::user();
       if(!is_null($user->kompanije)) {
           return $next($request);
       }
       return redirect('/company');
   }
}
```

Sl 3.8. „hasCompany“ međusloj

Zahtjevi za forme (engl. *form requests*) postoje ako se žele zadati kompliciraniji slučaji validacije. Zahtjevi za forme su vlastoručno napravljene klase koje sadrže logiku validacije. Kako bi se napravila takva klasa, koristi se Artisan naredba: „php artisan make:request NameOfRequest“. Novostvorena klasa će biti spremljena u mapu „app/Http/Requests“. Unutar klase se nalaze dvije funkcije „authorize()“ i „rules()“. Funkcija „authorize()“ određuje ima li korisnik odobrenje da napravi ovaj zahtjev. Unutar funkcije „rules()“ se pišu pravila validacije koji se primjenjuju na zahtjev. Slika 3.9. prikazuju pravila za zahtjev „KompanijaRequest“. Kao što se može vidjeti, u funkciji „rules()“ se definira niz pravila koji se primjenjuje na zahtjev. Tu se definira koji su podaci obavezni za slanje zahtjeva i kakvog tipa moraju biti. Nakon što se klasa zahtjeva popunila svim potrebnim pravilima, primjenjuje se u kontroleru umjesto generalne „Request“ klase. Tako da sad više neće pisati u metodi za spremanje: „public function store (Request $request)“ već „public function store(KompanijaRequest $request). Sad će se prilikom zahtjeva za spremanje koristiti set pravila postavljen u „KompanijaRequest“.

```
public function rules(){
    return [
        'komp_naziv' => 'required',
        'komp_email' => 'required',
        'komp_OIB' => 'required|integer',
        'komp_IBAN' => 'required',
        'komp_adresa' => 'required',
        'komp_grad' => 'required',
        'komp_drzava' => 'required',
        'komp_napomena' => 'required'
    ];
}
```

Slika 3.9. Zahtjev „KompanijaRequest“

3.1.8. Autentikacija korisnika

Skoro svaka web aplikacija koja ima bazu podataka ima i autentikaciju korisnika. Upravo iz tog razlog je Laravel napravio autentikaciju korisnika jako jednostavnom jer je sve već konfigurirano. Prema zadanim postavaka Laravel uključuje Eloquent model „User“ i migraciju za korisnika. Nadalje, pokretanjem „php artisan make:auth“ naredbe u terminalnu se dobiju kontroleri i pogledi za prijavu i registraciju korisnika. Dobiju se: kontroler za registraciju „RegisterController“, kontroler za prijavu „LoginController“ i kontroler koji rukuje slanjem e-poruke za resetiranje lozinke „ResetPasswordController“. Svi pogledi za autentikaciju korisnika se nalaze u mapi „resources/views/auth“ i to su pogledi „login.blade.php“ i „register.blade.php“. Svaki kontroler i pogled koji je Laravel napravio se može izmijeniti potrebama aplikacije. Laravel-ova autenikacija dolazi s metodama koje se često koriste u aplikaciji: „id()“, „user()“ i „check()“. Svaka od tih metoda se poziva preko „Auth“ fasade: „$user = Auth::user();“. Metodom „user()“ se dobije trenutno prijavljeni korisnik, točnije svi podaci iz tablice korisnika za tog korisnika. Metoda „id()“ vraća samo identifikator trenutnog korisnika, a „check()“ provjera je li korisnik prijavljen u aplikaciji. Osim kontrolera i pogleda, Laravel automatski napravi i međusloj naziva „auth“. Taj međusloj se poziva na svakoj ruti za koju korisnik treba biti prijavljen te ako nije prijavljen međusloj ga vraća natrag na početnu stranicu, a ako je dopušta mu pristup ruti ili slanju zahtjeva.

3.2. Programsko sučelje

Programsko sučelje se sastoji od općenitog opisa pogleda u Laravel-u i svake stranice koju aplikacije koristi. Sve stranice su detaljno objašnjenje, kako rade, što prikazuju i koji im je naziv. Unutar poglavlja pogleda će se osvrnuti na glavne koncepte pogleda u Laravel-u kao što je forma pogleda, Blade alat za stvaranje predložaka te Blade sintaksa.

3.2.1. Pogledi

Pogledi (engl. *views*) sadrže HTML koji pruža web aplikacija i odvajaju kontroler, odnosno logiku aplikacije, od logike prezentiranja podataka aplikacije. Pogledi su spremljeni u „resources/views“ mapi. Svaki pogled ima ekstenziju tipa „.blade.php“, što znači da pogledi koriste Blade sintaksu. Blade je jednostavan, ali moćan alat za stvaranje predložaka. Za razliku od drugih alata, Blade ne ograničava pisanje PHP koda direktno u pogledu. Štoviše, svi Blade pogledi su zapravo prevedeni u PHP kod i spremljeni u memoriju dok nisu izmijenjeni.

Svaki pogled može imati svoj predložak (engl. *layout*) ili više pogleda mogu dijeliti jedan. Unutar predloška se najčešće nalazi zaglavlje HTML-a, znači „html“, „head“, „title“ oznake i slično. Najvažniji dio predloška je „@yield()“ direktiva koja se koristi za prikaz sadržaja na tom dijelu stranice. Obično će taj dio popunjavati pogled koji će koristiti direktivu „@section()“ za popunjavanje predloška. Iako, prije toga je potrebno u pogledu definirati „@extends()“ direktivu koja će označavati koji će predložak zapravo popunjavati pogled sa „@section()“.

Kontroler može predati podatke pogledu, najčešće su to podaci iz baze podataka. Kako bismo ih prikazali u pogledu, potrebno je koristiti Blade sintaksu. Unutar vitičastih zagrada se upiše naziv varijable koje smo primili od kontrolera: „{{ $variable }} i taj se podatak onda prikaže u pogledu. U slučaju da kontroler vrati niz podataka, potrebno je koristiti petlje. Blade pruža jednostavne direktive za rad s PHP petljama. Samo dodavanjem znaka „@“ ispred PHP funkcija, kao što su „for“, „foreach“, „while“ se može proći kroz petlju. Petlje se završavaju samo dodavanjem „end“ na već spomenute PHP funkcije, na primjer za „foreach“ kraj označava „endforeach“. Unutar petlji se koristi Blade sintaksa jednaka onoj za prikaz jednog podatka, samo što je potrebno označiti koju točno vrijednost iz niza želimo. Tako da ako na primjer iz niza korisnika želimo dobiti korisničko ime, upisalo bi se unutar petlje: „{{ $users->username }}. Postoje i druge funkcije koje Blade sintaksa dozvoljava: „if“, „unless“, „continue“, „break“, i slično. Sve se te funkcije pišu se na isti način kao što su se pisale petlje.

Blade ima svoju sintaksu za forme. Forma se otvara korištenjem naredbe „open()“. Unutar „open()“ se piše niz parametara forme, kao što je URL na koji će voditi ili metoda kontrolera kojoj

će slati, koja se HTTP metoda koristi i slično. Forma se zatvara korištenjem „close()“ naredbe na formi: „Form::close()“. Za dodavanje polja za unos teksta, lozinke, datoteke, brojeva i za izbornike kao što je „checkbox“ ili „radio“ koriste se Blade metode koje odgovaraju nazivima polja na engleskom jeziku. Primjer se može vidjeti u polju za unos teksta: „Form::text()“ ili za unos lozinke: „Form::password()“.

Laravel olakšava zaštitu aplikacije od CSRF (engl. *cross-site request forgery*) napada. Laravel automatski generira CSRF tokene za svaku korisničku sesiju aplikacije. Taj token se koristi za potvrdu da prijavljeni korisnik je zapravo taj koji pravi zahtjev aplikaciji. Svaki put kad se definira HTML forma u aplikaciji potrebno je dodati skriveni CSRF token polja formi kako bi CSRF zaštita međusloja mogla potvrditi zahtjev. Skriveni CSRF token polje se generira u formi tako se da se poziva „csrf_field()“.

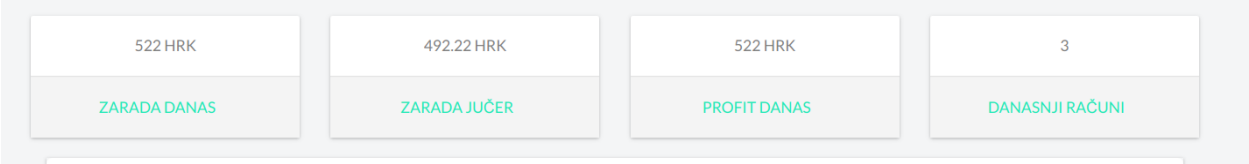
3.2.2. Registracija

Prije nego korisnik može započeti korištenje aplikacije mora se registrirati. Prilikom registracije korisnik mora unijeti standardne podatke kao što su ime, prezime, korisničko ime, adresu e-pošte i lozinku. Lozinka se štiti sa SHA enkripcijom koju Laravel automatski implementira. Nakon što se korisnik registrira mora dodati kompaniju. Bez dodavanja kompanije on ne može pristupiti niti jednoj drugoj stranici jer međusloj štiti stranice od pristupa korisnika koji nema dodanu kompaniju. Prilikom dodavanja kompanije korisnik može i ne mora dodati logo kompanije. U slučaju da odluči dodati logo, on se sprema u „img/logos“ javnu putanju te se ime loga i putanja spremaju u tablicu „company_photos“ i povezuju s kompanijom za koju je dodan račun. Nakon što doda kompaniju korisnik može početi koristit aplikaciju koja ga i automatski preusmjeri na početnu stranicu. Korisnik u bilo kojem trenutku može prestati s trenutnim korištenjem aplikacije tako da se odjavi. Aplikacija ga također, nakon što neko vrijeme nije ništa radio automatski odjavi te se mora ponovo prijaviti. Za prijavu korisnik mora unijeti adresu e pošte i lozinku koju je koristio prilikom registracije.

3.2.3. Početna stranica

Početni pogled se sastoji od četiri panela koji služe za evidenciju poslovanja kompanije, od grafičkog prikaze zarade te tablice svih računa bilo izlaznih ili ulaznih. Paneli za evidenciju poslovanja kompanije su: zarada danas, zarada jučer, profit danas i današnji računi. Zarada danas se dobiva tako da se u kontroleru „HomeController“ uzimaju svi iznosi izlaznih računa koji su izdani danas te korištenjem metode „sum()“ na modelu se svi oni zbrajaju: „$todayProfit = Bill::where('datum_izrade', '=', Carbon::today())->where('billable_type', 'App\Outgoing_bill')-

>where('company_id', $company_id)->sum('iznos');“. Zarada jučer se dobije na isti način, samo se umjesto „Carbon::today()“ koristi „Carbon::yesterday()“. Profit danas se dobije oduzimanjem zarade danas od zarade jučer korištenjem Blade sintakse: „{{$todayProfit - $todayDeficit}}“. Broj današnjih računa se dobije na slični način kao i profit danas, samo što se umjesto metode „sum()“ koristi metoda „count()“.



Sl 3.10. Paneli zarada na početnoj stranici

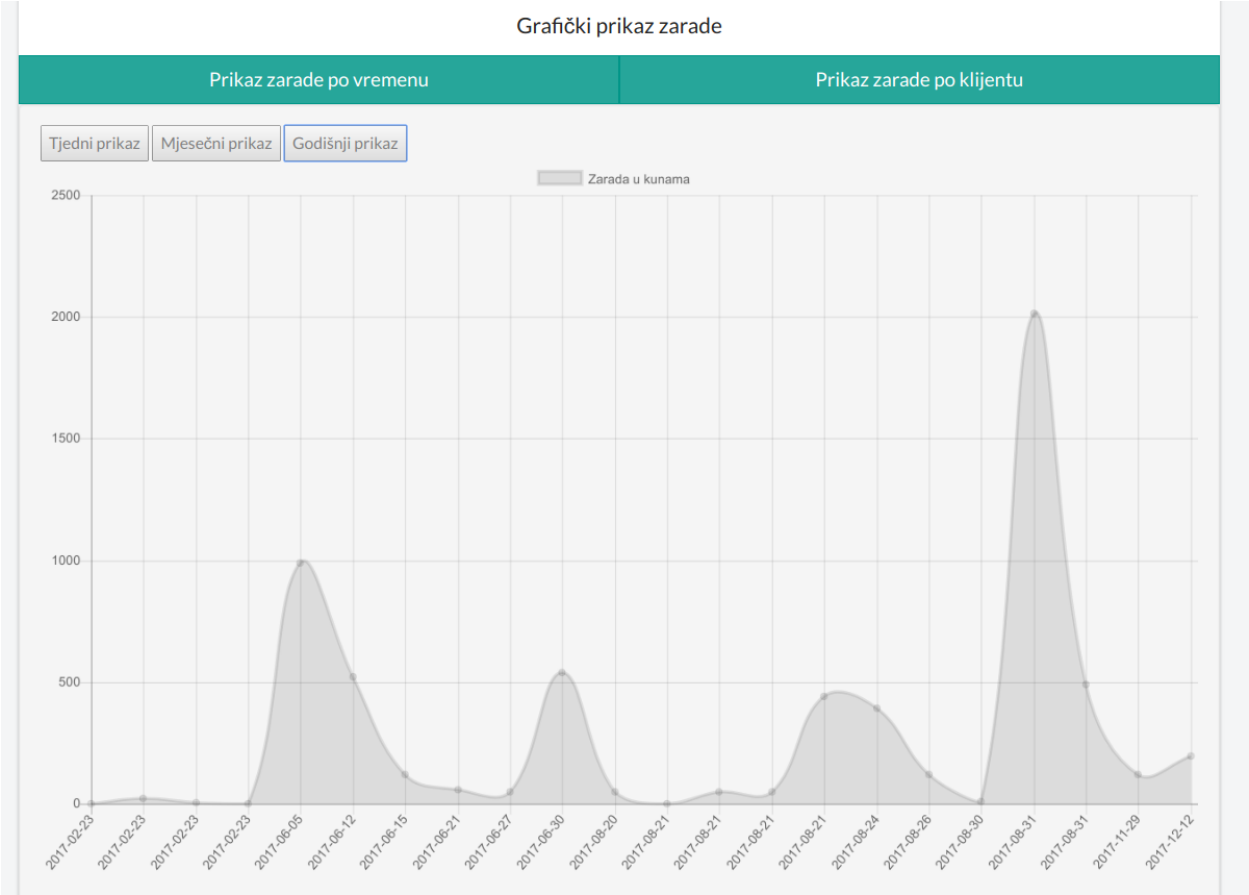
Grafički prikaz zarade se može vidjeti po vremenu ili po klijentu. Svaki od tih grafičkih prikaza nudi tjedni prikaz, mjesečni prikaz i godišnji prikaz zarade. Na slici 3.12. se može vidjeti grafički godišnji prikaz zarade po vremenu. Za dobivanje podataka za crtanje grafa se koristio Eloquent u modelu „HomeController“ i prikaz koda za dobivanje godišnje zarade za klijenta i godišnje zarade po vremenu se može vidjeti na slici 3.11.

```
$thisYearProfits = Bill::select('klijent', DB::raw('SUM(iznos) as iznos'))->where('datum_izrade', '>=', $start_date_year)->where('datum_izrade' ,'<=', $end_date_year)->where('billable_type', '=', 'App\Outgoing_bill')->where('company_id', '=', $company_id)->groupBy('klijent')->get();

$thisWeekProfits = Bill::select('klijent', DB::raw('SUM(iznos) as iznos'))->where('datum_izrade', '>=', $start_date_week)->where('datum_izrade' ,'<=', $end_date_week)->where('billable_type', '=', 'App\Outgoing_bill')->where('company_id', '=', $company_id)->groupBy('klijent')->get();
```

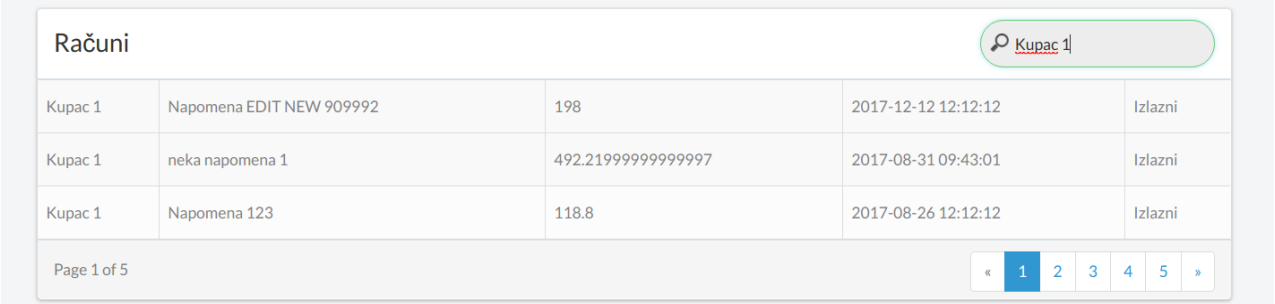
Sl 3.11. Dobivanje godišnje zarada za klijente i po vremenu

Za izradu grafova se koristio paket Chartjs. Chartjs je moćan paket za stvaranje grafova koji nudi jednostavnu izradu svih glavnih tipova grafova kao što su „pie chart“, „line chart“, „bar chart“ i drugi. Za prikaz zarade po vremenu se koristio linijski graf, a za prikaze zarade po klijentu su se koristili tortni graf i stupičasti graf. Svaki graf se poziva funkcijom na odabir dugmeta. Tako da se za prikaz godišnje zarade poziva „yearly_graph()“ funkcija. Unutar funkcije se nalazi definirana dva prazna niza koja se popunjavaju podacima iz baze podatka koje je poslao „HomeController“. Popunjavane nizova se odvija u „foreach“ petlja. Kad se nizovi popune stvara se graf „new Chart()“ funkcijom. Unutar te funkcije se postavlja mjesto gdje se postavlja graf na stranici (klasa ili identifikator HTML elementa), tip grafa i podaci s kojima da graf napravi.



Sl 3.12. Graf prikaza zarade po vremenu – godišnji prikaz

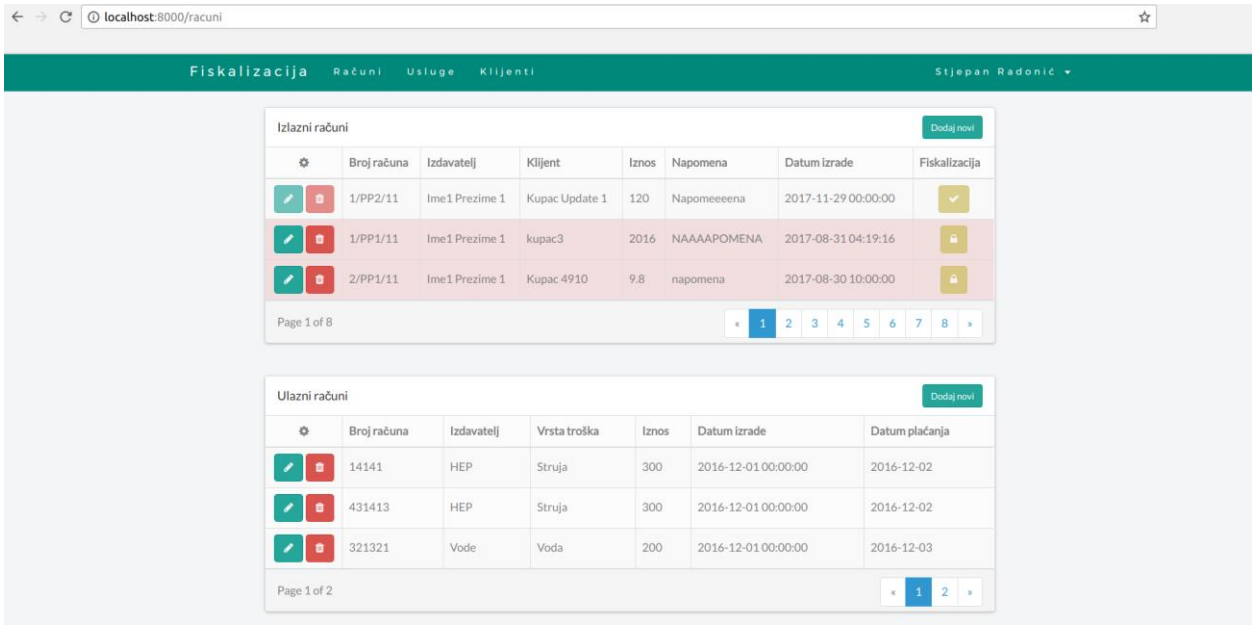
Tablica svih računa se sastoji i od ulaznih i od izlaznih računa poredanih prema vremenu nastanka. Korištenjem metode "orderBy()" se poredaju podaci dobiveni uz baze podataka u kontroleru "HomeController" ovisno prema argumentu koji se u tu metodu upiše. Uz svaki račun u tablici se upisuje i tip računa kako bi korisnik znao je li račun izlazni ili ulazni. Tablica se može pretraživati po bilo kojem elementu. Kad se započne upis u polje za pretragu on počinje pretraživati svaki element koji sadrži isti zapis. Pretraživanje radi JavaScript i na „keyup“ provjera što je upisano i filtrira tablicu tako da prikazuje samo retke koji sadrže zahtjev pretrage.



Sl 3.13. Tablica svih računa s pretragom

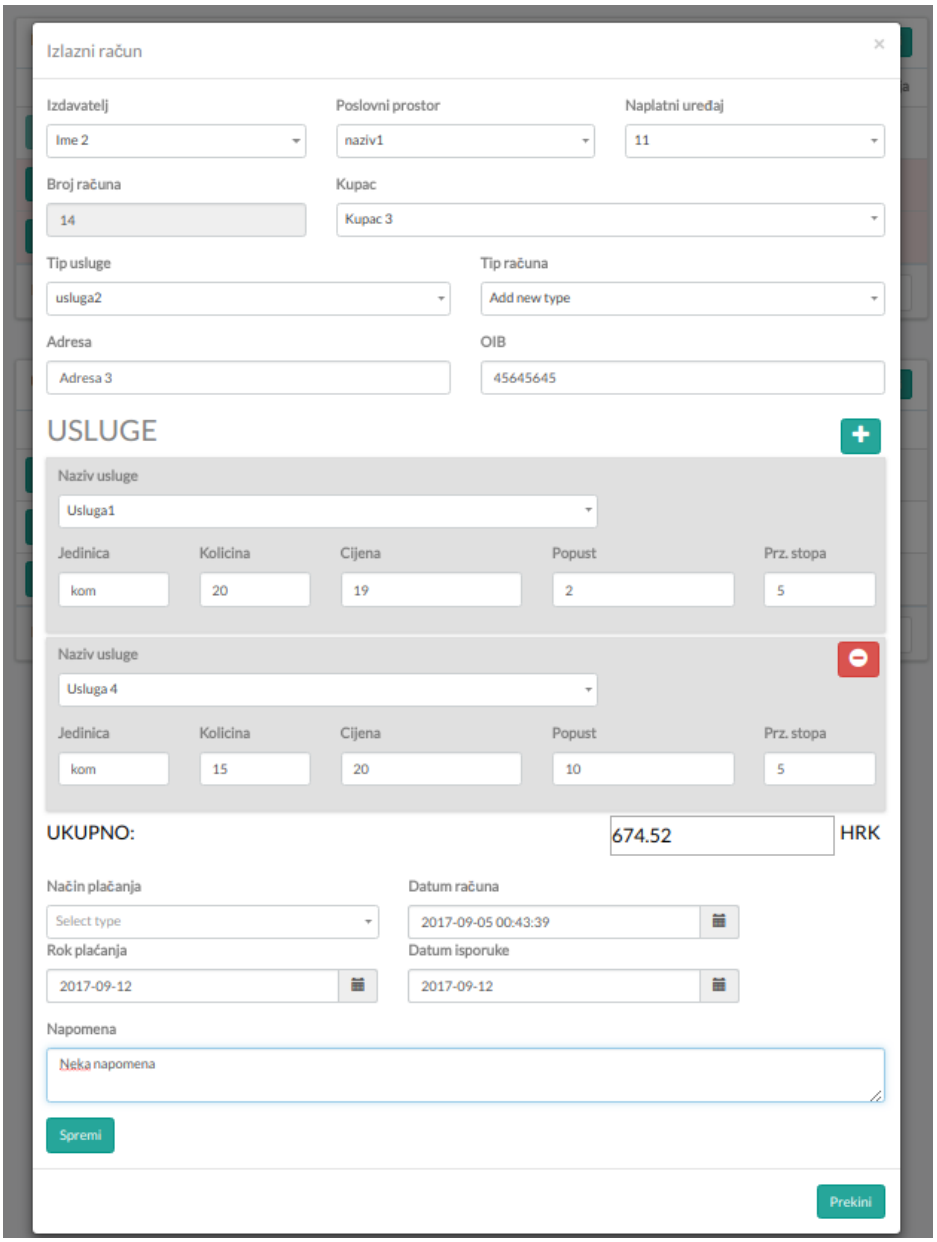
3.2.4. Stranica računa

Pogled računa se sastoji od dva panela, od kojih svatko ima svoje stranice. Prvi panel je za izlazne račune, to jest račune koje korisnik izdaje, a drugi panel je za ulazne račune koje korisnik dobije od drugih kompanija. Ulazni računi služe samo za evidenciju poslovanja kompanije. Oba panela se sastoje od identičnih tablica i dugmadi za dodavanje novih. Svaka tablica ima dugmad za izmjenu i brisanje postojećih zapisa. Tablica izlaznih računa još dodatno ima dugme za fiskalizaciju.



Sl 3.14. Stranica računa

Dodavanje novih računa se obavlja klikom na dugme za „Dodaj novi“. Odabirom tog dugmeta se upali odgovarajući modal. Modal za dodavanje novih računa se sastoji od polja: izdavatelj, poslovni prostor, naplatni uređaj, broj računa, kupac, tip usluge, tip računa, adresa, OIB, načina plaćanja, datuma računa, roka plaćanja, datuma isporuke, napomene i usluga. Usluge se još sastoje od polja: naziv usluge, jedinica, količina, cijena, popust i porezna stopa.



Sl 3.15. Dodavanje novog računa

Može se dodati više od jedne usluge na računu i iz tog razloga je korišten Handlebars.js alat za stvaranje predložaka. Handlebars.js dozvoljava definiranje predloška u HTML-u koji će se kasnije izvršiti. Predložak za Handlebars.js koji je napisan u pogledu se pokreće pozivanjem funkcije „buildTemplate“. Budući da postoji dosta komponenti koje Handlebars.js stvara, svaka ima svoju funkciju. Funkcije za stvaranje Handlebars.js predloška se poziva klikom na dugme, tako da ako se u izlaznom računu klikne na dugme „Dodaj novi“ pozivat će se funkcija za stvaranje tog predloška ili ako se klikne na „+“ za dodavanje usluge pozivat će se taj predložak.

```
function buildEditTemplate() {
  var titleEditTemplateScript = $("#modalEdit-outgoingBill-naslov").html();
  var titleEditTemplate = Handlebars.compile(titleEditTemplateScript);
  var titleEditContext = {"title" : "Izlazni račun - uređivanje"};
  var titleEditTemplateCompiled=titleEditTemplate(titleEditContext);
  $('.outgoing_title_placeholder').html(titleEditTemplateCompiled);
}
```

Sl 3.16. Handlebars.js funkcija za stvaranje predloška

Svaka padajuća lista elementa za odabir (engl. *select*) ja napravljena pomoću Select2 paketa. Select2 paket dodaje nove mogućnosti standardom elementu za odabir, od kojih je bila potrebna mogućnost za dodavanje novog ako niti jedan izbor ne odgovara. Zato što se element za odabir popunjava podacima iz baze podataka ili predefiniram statičnim podacima te se može dogoditi da niti jedan ne odgovara trenutnim potrebama korisnika. Select2 se aktivira funkcijom „activateSelect2()“. Unutar te funkcije inicijalizira se Select2 na običnom elementu za odabir i implementira logika za dodavanje novog. Select2 provjerava je li iz padajućeg izbornika odabrano dodavanje novog elementa. U slučaju da je odabrano dodavanje novog elementa pokreće se prozor za unos novog. Za prozor unosa novog se koristio paket Alertify kako bi se mogao prepoznati uneseni tekst u njegovoj funkciji. U funkciji Alertify-a se provjera postoji li unesenog teksta u njegovom prozoru za dodaj novi element na popis elemenata padajućeg izbornika. Ako postoji tekst, onda se dodaje novi element na popis elemenata padajućeg izbornika i šalje se poruka o dodanome element. Inače, ako se u prozor za dodavanje novog elementa u padajući izbornik ništa ne upiše, dobije se i odgovarajuća poruka koja glasi: „Niste ništa unijeli“. Kako bi Select2 radio potrebno ga je pozvati nakon stvaranja elementa za odabir na kojem ga želimo primijeniti, a budući da se takav element za odabir stvara dinamički, funkcija za Select2 se poziva nakon stvaranje predloška Handlebars.js-a.

```
function activateSelect2() {
  $(".usluga_opis")
    .select2({
      placeholder: 'Select type'
    })
    .on('select2:close', function() {
      var el = $(this);
      if(el.val()==="NEW") {
        var newval = alertify.prompt('Unos', 'unesi ovo', '',
        function(evt, newval) {
          if(evt){
            var input = newval;
            if(newval!==''){
              el.append('<option>'+input+'</option>')
                .val(input);
              alertify.success('Unijeli ste: ' + newval);
            }
          else {
            alertify.error('Niste ništa unijeli');
          }
        }
      },
      function() { alertify.error('Zatvorili ste') });
  }
    })
...
```

Sl 3.17. Pokretanja i implementacija Select2 alata i alertify skočni prozor

Elementi za odabir: izdavatelj, poslovni prostor, naplatni uređaj, kupac i naziv usluge se pune iz baze podataka. Za svaki od tih elemenata je potrebno napraviti niz iz baze podataka korištenjem Eloquent veza. Tako da se na primjer za izdavatelja koristila veza: „$cashiers = $company->cashiers()->get()“ jer je svaki izdavatelj računa povezan s kompanijom, kao što je i objašnjeno u poglavlju baza podataka. Parsiranje podataka za padajuće liste navedenih elemenata za odabir se obavio korištenje Blade petlje „@foreach“ (Slika 3.18).

```
@foreach($businessplaces as $businessplace)
   <option value="{{ $businessplace->id}}"> {{ $businessplace->name}}</option>
@endforeach
```

Sl 3.18. Parisanje podataka „foreach“ petljom (Blade sintaksa)

Odabirom jednog od opcija iz elementa za odabir kupca ili naziva usluge se poziva Ajax za dinamično popunjavanje polja. Svaki korisnik aplikacije može napraviti predložak kupca i usluge i zatim te predloške koristit u izradi računa. Tako da kad korisnik aplikacije izabere jednu

od opcija na kupcu ili usluzi, odgovarajuća polja u formi se popune podacima iz baze za odabranu opciju. Na primjer kad korisnik odabere uslugu, automatski se popune polja: jedinica, količina, cijena, popust i porezna stopa. To se odvija korištenjem Ajax-a. Ajax funkcija za dinamičko popunjavanje se poziva na odabir elementa iz padajuće liste usluga. Nakon toga, Ajax koristi HTTP metodu GET i šalje identifikator računa metodi koja preko tog računa pretražuje model, odnosno tablicu, kataloga usluga i zatim ta metoda vrati Ajax-u u Json formatu podatke tog zapisa. Ajax zatim popunjava polja s tim podacima (Slika 3.19.).

```
var url ="/racuni/service/";
   $('#usluga_opis').change(function(){
      $.ajaxSetup({
           headers: {
            'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
            }
      });
      var valueID=$(this).find('option:selected').val();
      $.get(url + valueID, function(data){
         var usluga_jed_mjere = document.getElementById('usluga_jed_mjere');
         var usluga_kolicina = document.getElementById('usluga_kolicina');
         var usluga_cijena = document.getElementById('usluga_cijena');
         var usluga_popust = document.getElementById('usluga_popust');
         var usluga_porez_stopa = document.getElementById('usluga_porez_stopa');
         usluga_jed_mjere.value = data.usluga_jed_mjere;
         usluga_kolicina.value = data.usluga_kolicina;
         usluga_cijena.value = data.usluga_cijena;
         usluga_popust.value = data.usluga_popust;
         usluga_porez_stopa.value = data.usluga_porez_stopa;
…
```

Sl 3.19. Ajax dinamičko popunjavanje polja usluga

Računanje iznosa usluge, ukoliko se ona ne popunjavaju dinamički iz baze, nego se ručno unosi nova usluga, odrađuje JavaScript. Svaki od elemenata usluge, znači jedinica, količina cijena, popust i porezna stopa imaju istu HTML klasu naziva „.usluge_input“. Funkcija za računanje ukupnog iznosa usluge se poziva na otpuštanje tipke na tipkovnici (engl. *keyup*) kad se upisuje u element s klasom „.usluge_input“. Ta funkcija uzima vrijednost svakog od polja i zatim računa ukupni iznos, gdje provjerava koji od elemenata je jednak nuli te ga zaobilazi. Funkcija izračunava za svaki blok usluga posebno ukupni iznos, jer može postojati više blokova i kad bi za sve odjednom računalo ne bi se dobio ispravan ukupan iznos. Nakraju kad se svi ukupni iznosi za svaki blok zbroje, oni su upisuju u formi pod element ukupnog iznosa.

Nakon što se popuni forma, ona se šalje metodi za spremanje unutar kontrolera za računa naziva „OutgoingBillController“. Prilikom slanja forme u metodi za spremanje se poziva zahtjev „OutgoingBillsRequest“ koji provjera jesu li poslana sva potrebna polja za spremanju u bazu podataka te ako nisu vraća poruku o elementu koji treba popuniti prije slanja. Metoda unutar „OutgoingBillControllera“ za spremanje se zove „store()“. Ta metoda prvo uzima iz zahtjeva sve potrebne podatke za spremanje u tablicu izlaznih računa, tako što poziva „create()“ metodu na modelu „Outgoing_bill“. Zatim uzima ostale parametre računa iz zahtjeva koji odgovaraju modelu „Bill“ i sprema ih u tu tablicu. Nakraju, se iz zahtjeva uzimaju svi elementi potrebi za spremanje usluge i spremaju se u varijablu „$services“. Zatim se provjerava koliko usluga ima, tako što se izbroji koliko elemenata naziva „usluga_opis“ je poslano zahtjevu, jer po usluzi postoji samo jedan takav element i onda se zna ako ima više od jedne poslane usluge. Za svaku uslugu se stvara nova instanca modela „Outgoing_services“ i zatim se za svaku uslugu iz varijable „$services“ parsiraju usluge da bi se svaki element usluge koja ima brojke spremio kao tip podataka „integer“ i svakom elementu koji se ne sastoji samo od brojki maknulo sve što nije broj ili slovo. Za provjeru sastoji li se element od samo brojeva se koristi PHP funkcija „ctype_digit()“.

```
$services = Input::only(['usluga_opis', 'usluga_jed_mjere', 'usluga_kolicina', 'usluga_cijena',
'usluga_popust', 'usluga_porez_stopa']);
    $c=count(Input::get('usluga_opis'));
    for($j=0; $j<$c; $j++) {
        $newService = new Outgoing_service;
        foreach($services as $service => $value) {
          $newService->outgoingbill_id = $request->editBillBtnId;
          $newService->$service = ctype_digit($value[$j]) ? (int)$value[$j] : trim($value[$j], "");
        }
        $newService->save();
     }
```

Sl 3.20. Spremanje jedne ili više usluga prilikom stvaranja računa

Svaki račun koji nije fiskaliziran se može obrisati ili izmijeniti. Račun se izmjenjuje odabirom dugmeta za izmjenu. Klikom na dugme za izmjenu se poziva Ajax metoda koja uzima identifikator računa i šalje ga metodi „edit()“ u kontroleru „OutgoingBillController“. Metoda „edit()“ preko preuzetog identifikatora pretražuje tablice „bills“, „outgoing_bills“, „outgoing_services“, „businessplaces“, „chargingdevices“ i „cashiers“ kako bi pronašao sve potrebne podatke za račun. Ti se podaci primaju u Ajax funkciju koja ih onda samo upisuje u polja u tablici. Prilikom stvaranja modala za izmjenu se pozivaju funkcije za Select2 i Handlebars.js, a Handlebars.js se poziva i u Ajax funkciji jer se prilikom stvaranja usluga poziva Handlebars.js koji stvori sve usluge. Prilikom izmjene računa korisnik može izmijeniti bilo koje polje, dodati i

obrisati usluge i zatim to potvrditi odabirom dugmeta za spremanje izmjene. Klikom na to dugme se forma šalje metodi „update()“ u kontroleru „OutgoingBillsController“ koji radi na sličan način kao i metoda „store()“, samo što sad izmjenjuje postojeći.

Brisanje računa se odvija na način da se odabere dugme za brisanje. Nakon klika na to dugme se pojavi poruka koja obavještava korisnika da je krenuo brisati zapis. Poruka ga pita je li siguran, jer više neće moći vratit ovaj račun te ako odabere prekidanje, račun će mu ostati, ali ako odabere brisanje, poziva se Ajax metoda za brisanje. Ajax metoda prima identifikatora računa i šalje ga metodi za brisanje. Metoda za brisanje pronalazi taj račun i briše zapise u tablicama „bills“, „outgoing_bills“ i „outgoing_services“ za poslani račun korištenjem metode „destroy()“ i vraća Ajax-u poruku o uspjehu. Ajax dinamički miče račun „remove()“ iz pogleda nakon dobivanja poruke o uspjehu. Za skočni prozor se koristio SweetAlerts.js paket.

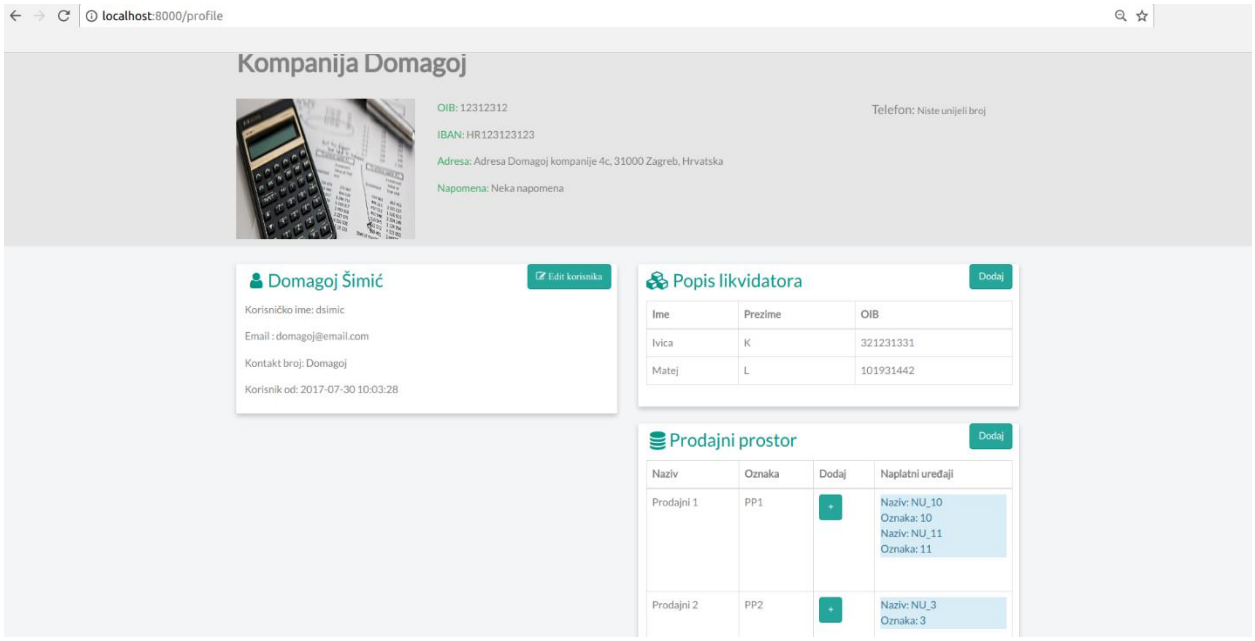
```
$.ajax({
            type: "DELETE",
            url: urls+ ingoing_id,
            success: function (data) {
              console.log(data.id);
                swal("Deleted!", "Račun je obrisan.", "success");
                $("#inGoing" + data.id).remove();
            },
            error: function (data) {
                console.log('Error', data);
            }
        });
```

Sl 3.21. Ajax kod za brisanje odabranog računa

Klikom bilo gdje na račun, ako to nije jedno od dugmadi, će se pojaviti modal u kojem možemo pogledati naš račun bez izmjena. Kad pregledamo je li sve uredu s računom možemo odabrati skidanje računa u PDF formatu klikom na dugme za skidanje (engl. *download*). Klikom na dugme za skidanje se putem Ajax-a poziva metoda koja šalje identifikator računa metodi za skidanje naziva „download()“ u kontroleru „OutgoingBillController“. Prvo se pronađu preko identifikatora svi potrebni podaci o računi i spreme se u odgovarajuće varijable. Zatim se svi ti podaci spremaju u niz naziva „$data“ i šalju pogledu naziva „bill_download“. Taj pogled se učitava pozivom metode „loadView('bill_download)“ iz biblioteke za skidanje PDF-ova. I zatim se skida pdf pozivam metode „download()“. Naziv PDF-a će sadržavati broj računa i datum skidanja, a sami PDF će sadržavati sve podatke o računu pa tako i ZKI i JIR.

3.2.5. Stranica profila

Kako bi korisnik mogao pregledati podatke o sebi, o svojoj kompaniji, izdavačima računa i prodajnim prostorima s odgovarajućim naplatnim uređajima postoji stranica profila. Na toj stranici se nalaze svi navedeni podaci koji su dobiveni iz kontrolera „ProfileController“. Unutar stranice profila korisnik može dodati novog izdavatelja računa, prodajni prostor i za svaki prodajni prostor dodati naplatni uređaj. Izmjena navedenih nije moguća jer bi to ugrozilo dosad izdane račune, ali ih je zato moguće obrisati jer se može dogoditi da je osoba koja izdaje račune prestala raditi, ali nije moguće da je promijenila OIB. Dodavanje izdavatelja, prodajnih prostora i naplatnih uređaja se odbija u modalu koji se otvara klikom na odgovarajuće dugme. Prvo što korisnik vidi na stranici su podaci o kompaniji kao i logo koji je dodao. Zatim vidi tri panela, jedan koji mu daje podatke o njemu i dva koji su tablice s izdavateljima računa i prodajnih prostora. Tablica prodajnih prostora još u sebi sadrži listu svih naplatnih uređaja.

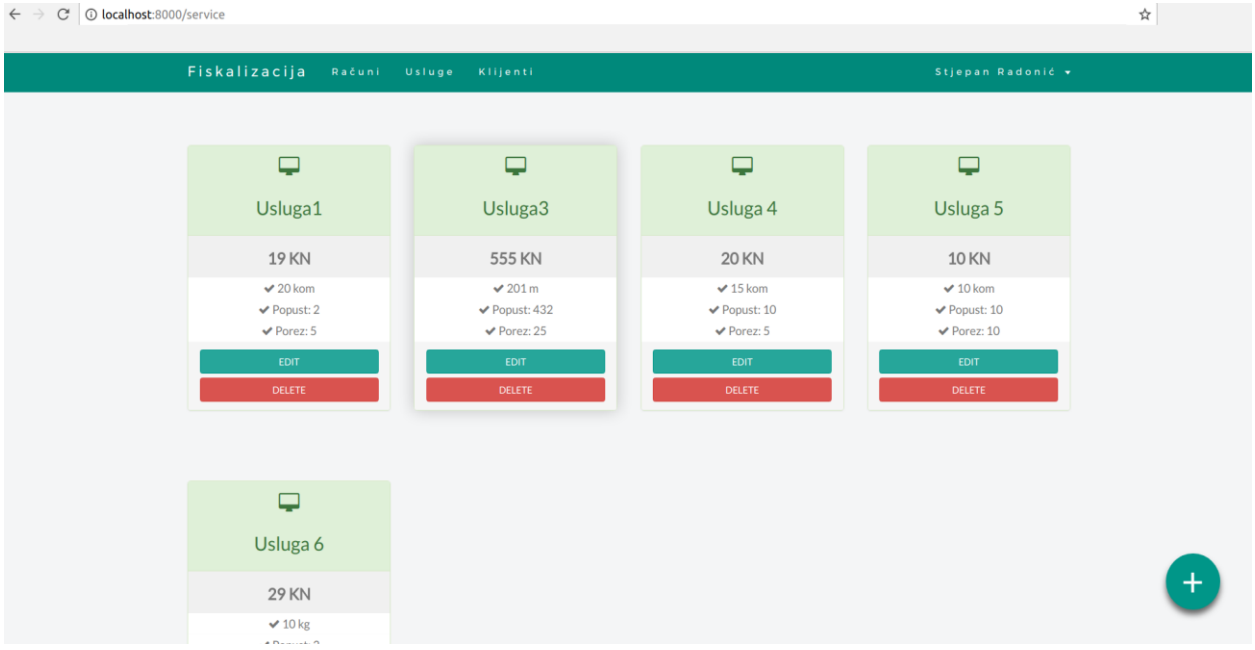


Slika 3.22. Stranica profila

3.2.6. Stranica usluga i klijenata

U pogledu usluge korisnik može pregledavati postojeće usluge, dodavati nove, izmijeniti i brisati usluge. Svaka usluga ima svoj blok koji se dinamički stvara ovisno o tome koliko usluga u bazi podataka imamo preko Handlebars.js-a. Svaki blok usluga ima napisan naziv usluge, iznos usluge, jedinicu mjere, popust i porez te dugmad za izmjenu i brisanje. Nova usluga se dodaje odabirom dugmeta s oznakom „+“ koja stoji na donjem desnom dijelu ekrana. Klikom na to dugme se preusmjerava na stranicu da dodavanje novih usluga gdje se upisuju svi podaci o usluzi kao što

su: naziv usluge, cijena, količina, jedinica mjere, porezna stopa i popust. Klikom na dugme kreiraj uslugu se šalje zahtjev kontroleru „ServiceController“ koji provjerava ispravnost zahtjeva preko „ServicesRequesta“ i ako je zahtjev uredu on se i odrađuje. Nove usluge se spremaju u tablicu „catalog_services“. Nakon spremanja se odmah preusmjerava korisnika na početnu stranicu usluga gdje može i vidjeti novostvorenu uslugu. Svaku uslugu može i izmijeniti odabirom dugmeta za izmjenu koji ga vodi na novu stranicu koja izgleda isto kao i stranica za stvaranje nove usluge samo što je popunjena podacima iz baze podataka za tu uslugu. Ovdje se ne koristi Ajax metoda za prikaz novih već se preko rute šalje identifikator metodi za stvaranje pogleda za izmjenu: „Route::get('service/{id}/edit', 'ServiceController@edit_page');“. Nakon izmjene se odabere dugme za slanje izmjenjene forme koja šalje metodi „update()“ zahtjev za izmjenom i ta metoda izmjenu odradi i spremi u tablici „catalog_services“. Korisnika se nakon izmjene preusmjerava na početni pogled usluga te je izmjena usluge odmah vidljiva. Ako korisnik želi obrisati uslugu odabere dugme za brisanje i pojavi mu se skočni prozor je li siguran da želi obrisati uslugu, ako odabere da nije siguran da želi obrisati uslugu, usluga će ostati, a ako odabere brisanje poziva se Ajax metoda koja preuzima identifikator usluge. Taj identifikator se šalje metodi „destroy()“ koja pronađe instancu modela s tim identifikatorom i na toj instanci modela poziva „delete()“ metodu koja briše zapis iz tablice. Metoda „destroy()“ nakon brisanja šalje Ajax-u poruku o uspješnosti brisanja i Ajax daje poruku o uspjehu i zatim dinamički makne obrisanu uslugu.



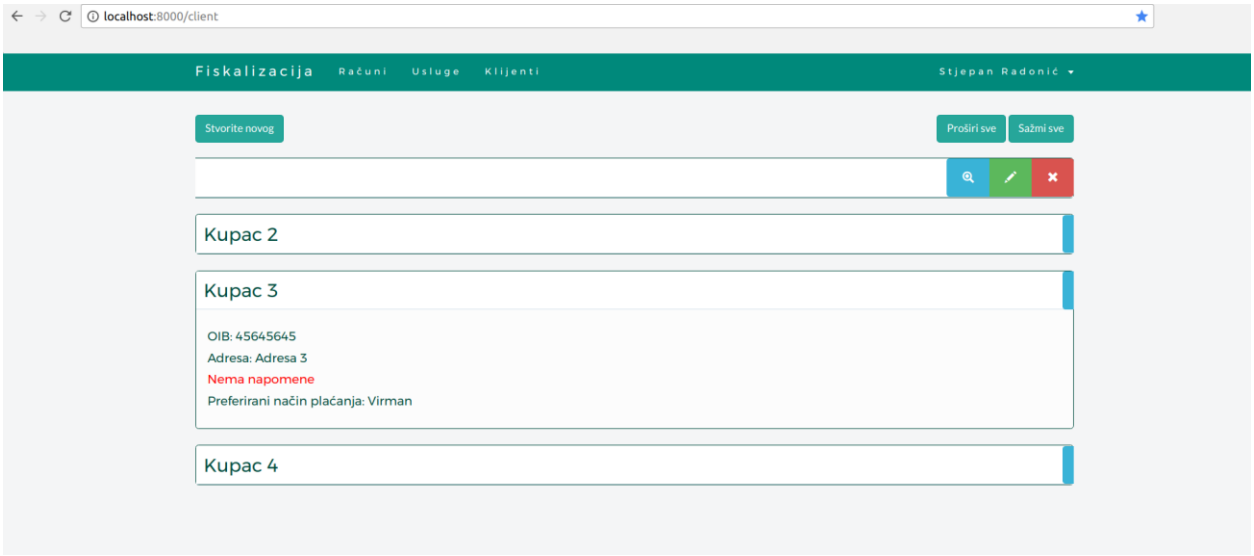
Sl 3.23. Stranica usluga

U pogledu klijenata korisnik može vidjeti sve postojeće klijente, odnosno kupce te dodavati nove, izmijeniti i brisati stare. Svi klijenti se nalaze u listi u kojoj se mogu vidjeti samo imena klijenata. Držanjem miša iznad određenog klijenta se s desne strane pojavi panel koji sadrži dugmad za pogled, izmjenu i brisanje klijenata tim redoslijedom. Odabirom dugmeta za pogled se kartica klijenta proširi i mogu se vidjeti detalji o njemu, kao što je njegov OIB, adresa, napomena i preferirani način plaćanja, zadnja dva su opcionalna. Prilikom dodavanja klijenta nije potrebno popuniti podatke napomena i preferirani način plaćanja jer su to dodatni elementi koji mogu služiti kao pomoć pri dodavanju računa jer automatski popunjavaju ta polja u formi. Ako kojim slučajem nema dodane napomene ili preferiranog načina plaćanja to se može vidjeti u detaljima o klijentu jer su označeni crvenom bojom. Prikaz nepostojanja tih elemenata crvenom bojom se napravio Blade sintaksom kao što se može vidjeti na slici 3.24. i imaju predefiniranu poruku s ključnom riječi „nema“.

```
<h5>OIB: @{{oib}}</h5>
<h5>Adresa: @{{adresa}}</h5>
@{{#checklength napomena 1}}
    <h5>Napomena: @{{napomena}}<h5>
@{{else}}
     <h5 style="color:red;">Nema napomene</h5>
@{{/checklength}}
@{{#checklength placanje 1}}
      <h5>Preferirani način plaćanja: @{{placanje}}<h5>
@{{else}}
       <h5 style="color:red;">Nema preferiranog načina plaćanja</h5>
@{{/checklength}}
```

Sl 3.24. Blade sintaksa za „if“ petlju za klijente

Osim dugmeta za dodavanje novog klijenta postoji i dugmad naziva „Proširi sve“ i „Sažmi sve“. Klikom na dugme „Proširi sve“ svaki od klijenata u listi se proširi tako da se mogu vidjeti svi detalji svih klijenata, dok dugme „Sažmi sve“ radi obrnuto i sakrije sve detalje svih klijenata. Dodavanje novog klijenta se obavlja klikom na dugme „Stvorite novog“ koji vodi na stranicu za dodavanje novog klijenta. Izmjena klijenta se odrađuje klikom na dugme za izmjenu koje vodi na stranicu za izmjenu s popunjenim podacima o izabranom klijentu. Klijenta je moguće obrisati klikom na dugme za brisanje i prikazuje se poruka o brisanju i zatim Ajax odrađuje brisanje i automatski miče klijenta iz pogleda i daje prozor s porukom o uspjehu. Dodavanje klijenta, izmjena i brisanje radi na isti način kao i dodavanje usluga.



Slika 3.25. Stranica klijenata

4. FISKALIZACIJA

Svaka aplikacija koja izdaje račune, mora ih i fiskalizirati, pa tako i ova web aplikacija. Fiskalizirati račun zapravo znači dostaviti račun FINI. Račun se dostavlja FINI u XML obliku, digitalno se potpisuje i dostavlja CIS-u, odnosno Centralnom informacijom sustavu Ministarstva financije, Porezne uprave. Unutar ove aplikacije postoji jedan kontroler naziva „FiskalizacijaController“ koji sadrži cijelu logiku i odrađuje sav posao fiskaliziranja računa.

4.1. Priprema fiskalizacije

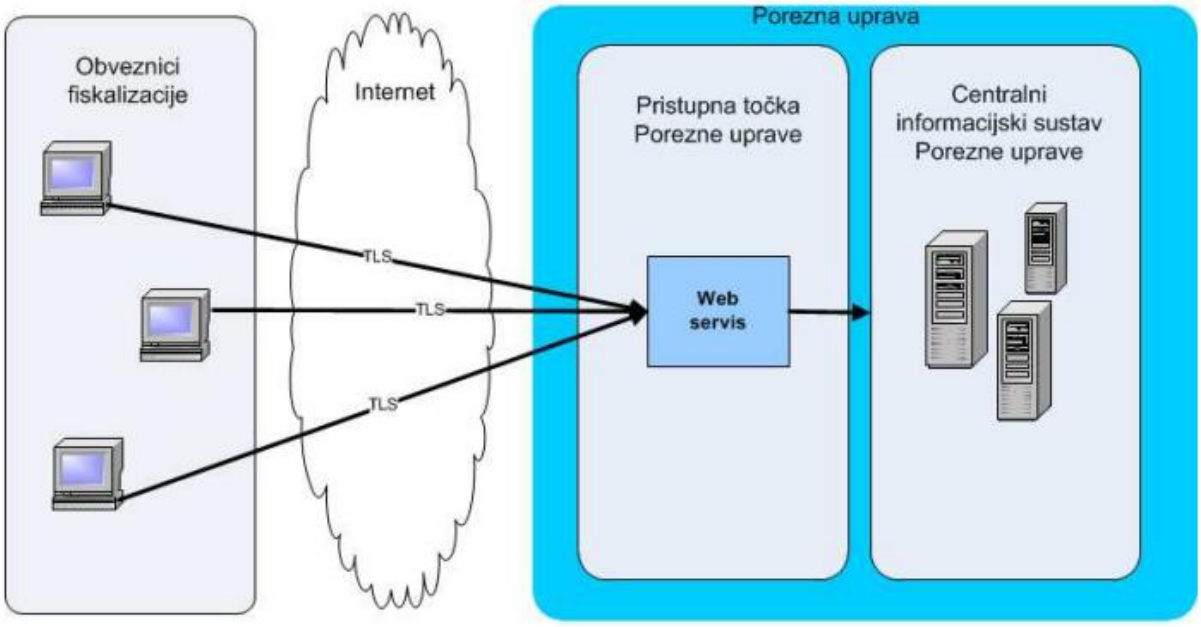
Prvo što je potrebno napraviti je preuzeti najnoviji dokument koji se zove Tehnička specifikacije za korisnike. Verzija koja se koristila prilikom izrade ovog rada i aplikacije je „Fiskalizacije – Tehnička specifikacija za korisnike Verzija 1.5“. Unutar tog dokumenta je opisana poslovna interakcija korisnika sustava, topologija sustava, preduvjeti za spajanje na CIS Porezne Uprave, postupak spajanja, pregled tehnologija korištenih u komunikaciji s pristupnom točkom CIS-a i tako dalje. Odnosno, dokument sadrži sve informacije potrebne za razumijevanje fiskalizacije i kako je napraviti.

Za fiskalizaciju je potreban digitalni certifikat jer pripremljenu XML poruku zahtjeva je potrebno elektronički potpisati privatnim ključem aplikativnog certifikata koji je izdan obvezniku u svrhu fiskalizacije (u testnoj okolini koristi se DEMO aplikativni certifikat). Digitalni certifikat se dobije od strane FINA-e. Prilikom izrade ovog web rješenja koristio se DEMO aplikativni certifikat. Certifikat je isporučen od strane FINA-e putem e-poruke, ako druga metoda nije unaprijed dogovorena. Na pretinac e-pošte se ne dobije certifikat već referentni broj „Reference number“ i autorizacijski kod „Authorization code“. Uz ta dva podatka se skida certifikat uz uputu naziva „Procedura izdavanja DEMO aplikativnog certifikata za fiskalizaciju“. Bitno je naglasiti da je certifikat moguće skinuti samo iz Microsoft Windows operativnog sustava i Internet Explorer mrežnog preglednika. Osim aplikativnog certifikata, potreban je i Verifikacijski/root (samopotpisani) certifikat za Demo CA koji se skida s FINA-ine stranice. Obavljanjem cijele navedene procedure korisnik ima dva certifikata na računalu: Demo aplikativni certifikat i Verifikacijski/root (samopotpisani) certifikat za Demo CA. Potrebno je da su certifikati odgovarajućih formata za potrebe ove web aplikacije. Zato je Demo aplikativni certifikat potrebno skinut u „.pfx“ formatu, a Verifikacijski/root (samopotpisani) certifikat za Demo CA u „.cer“ formatu.

4.2. Topologija i sigurnosni preduvjeti

FINA i CIS Porezna uprava imaju preduvjete koji se moraju poštivati prilikom spajanja i preporuke koje bi se trebalo pratiti. Preduvjeti i preporuke se dijele u tri kategorije: mrežni, sigurnosni i aplikacijski. Za spajanje na CIS Porezne uprave klijentski sustav mora zadovoljiti sljedeće mrežne preduvjete: vrsta mreže mora biti Internet i preporučeni otvoreni TCP portovi prema CIS sustavu trebaju biti 8449. Mrežna preporuka za spajanje je da karakteristika linka treba biti stalni simetrični link i da propusnost treba biti najmanje 2MB/s, iako se potrebna propusnost procjenjuje na temelju broja poruka po sekundi za vrijeme najvećeg opterećenja. Sigurnosni preduvjeti su ti da sva komunikacija sa CIS-om Porezne uprave treba biti zaštićena“ 1-way TLS“ enkripcijom na transportnom sloju. Također, potreban je odgovarajući aplikativni digitalni certifikata za elektronično potpisivanje izdan od strane FINA-e. Aplikacijski preduvjet je taj da aplikacijski protokol treba biti SOAP/HTTPS (SOAP 1.1) i kodna stranica XML poruke zahtjeva treba biti UTF-8 [15].

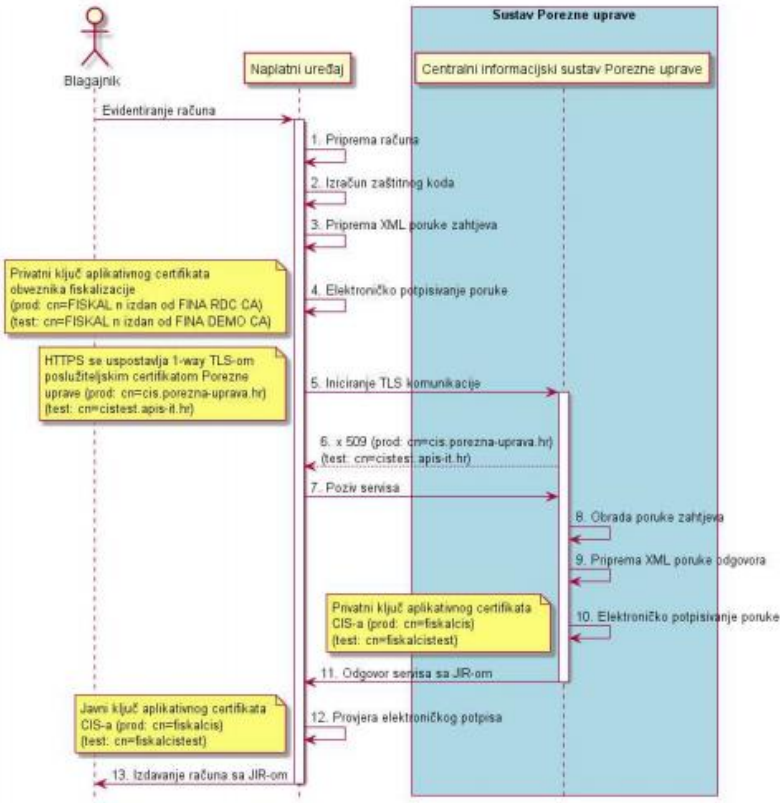
Klijenti su zaduženi osigurati hardversku i softversku podršku za razmjenu poruka s pristupnom točkom. Implementacija i održavanju pristupne točke je zadaća APIS IT-a koji mora omogućiti klijentu spajanje na CIS pristupnu točku [15]. Slika 4.1. prikazuje topologiju pristupa CIS-u Porezne uprave te se prema njoj može vidjeti da klijent sam mora odabrati platformu i implementirati softversko rješenje te osim toga mora i osigurati Internet vezu prema CIS pristupnoj točki.



Sl 4.1. Topologija pristupa CIS-u Porezne uprave

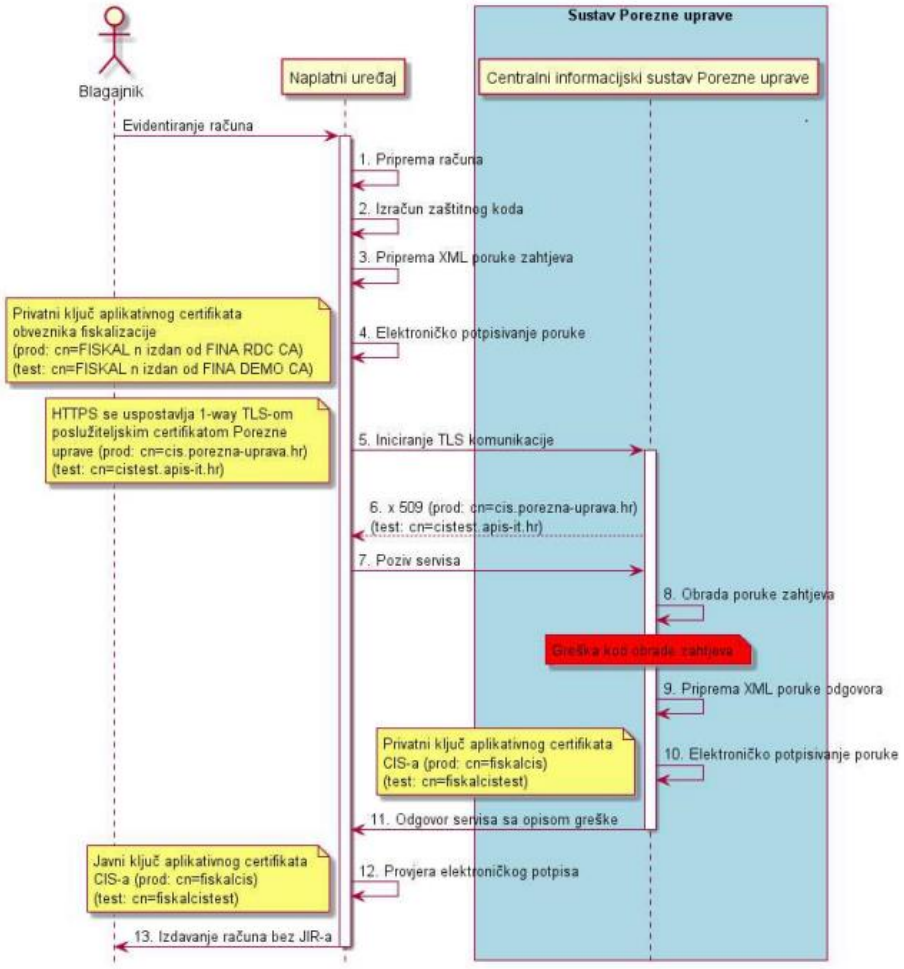
4.3. Interakcija korisnika sustava

Svaki obveznik fiskalizacije mora dostaviti podatke o računu i to pojedinačno za svaki račun u trenutku izdavanja računa. Proces izmjene podataka počinje izdavanjem računa od strane operatera na naplatnom uređaju, odnosno blagajni, koju u ovom slučaju predstavlja web aplikacija. Naplatni uređaj priprema podatke za račun na temelju stavki koje je operater unio. Na temelju tih podataka se izračunava zaštitni kod izdavatelja ili skraćeno ZKI prema algoritmu koji je propisan od strane FINA-e. Nakon toga se priprema XML poruka i elektronički se potpisuje privatnim ključem aplikativnog certifikata. U testnoj okolini, koja se i koristila u ovoj aplikaciji, to je Demo aplikativni certifikat. Zatim se pokreće „1-way TLS“ komunikacija pri čemu se poslužitelj Porezne uprave predstavlja s poslužiteljskim certifikatom. Za testnu okolinu to je „cistest.apis-it.hr“ certifikat. Nakon uspješne TLS komunikacije vrši se poziv poslužitelja. CIS zaprima i obrađuje poruku zahtjeva i ako je zahtjev uspješno obrađen CIS priprema XML poruku odgovora. XML poruka odgovora se sastoji od JIR-a i elektronički ju potpisuje privatnim ključem aplikativnog certifikata. U testnoj okolini koristi se DEMO aplikativni certifikat „fiskalcistest“. Nakon elektroničkog potpisivanja poruke, ona se šalje natrag naplatnom uređaju obveznika. Naplatni uređaj obveznika prima poruku odgovara i provjera elektronički potpis te ako je on ispravan operater na naplatnom uređaju izdaje kupcu račun s ispisanim JIR-om [15].



Sl 4.2. Slijedni dijagram procesa za slanje računa

U slučaju da se dogodi greška za vrijeme obrade poruke zahtjeva zbog neispravnog XML-a, neispravnog elektroničkog potpisa i slično, CIS vraća XML poruku odgovara koja sadrži opis greške. Svaka greška se može naći u šifrarniku grešaka u tehničkoj dokumentaciji FINA-e. Detaljno o greškama se nalazi u poglavlju testiranje. U slučaju pogreške odgovor ne sadrži JIR, ali se račun svejedno mora izdati kupcu iako nema JIR-a.



Sl 4.3. Slijedni dijagram procesa za slanje računa u slučaju greške

Oblik svakog računa je propisan od strane FINA-e. Svaka stavka računa ima točno određen podatak, opis ili napomenu, obaveznost, tip i duljinu. Podatak je vrsta stavke računa kao što je broj računa. Opis ili napomena dodatno pojašnjava stavke računa te daje informacije o tome što se očekuje da je ta stavka te postoje li neka ograničenja. Obaveznost označava mora li postojati određena stavka na računu ili se može izdati bez nje. Tip i duljina označavaju format podataka i ako postoji njegovu dozvoljenu duljinu kao što je na primjer „char(36)“.

Zaglavlje XML-a se mora sastojati od identifikatora poruke i datuma i vremena slanja. Identifikator poruke je ID poruke „UUID“ te svaka poruka koja se šalje CIS-u mora imati različiti

identifikator. Datum i vrijeme slanja je „datetime“ parametar i on se odnosi na vrijeme slanja poruke CIS-u i mora biti propisanog formata „dd.mm.ggggThh:mm:ss“.

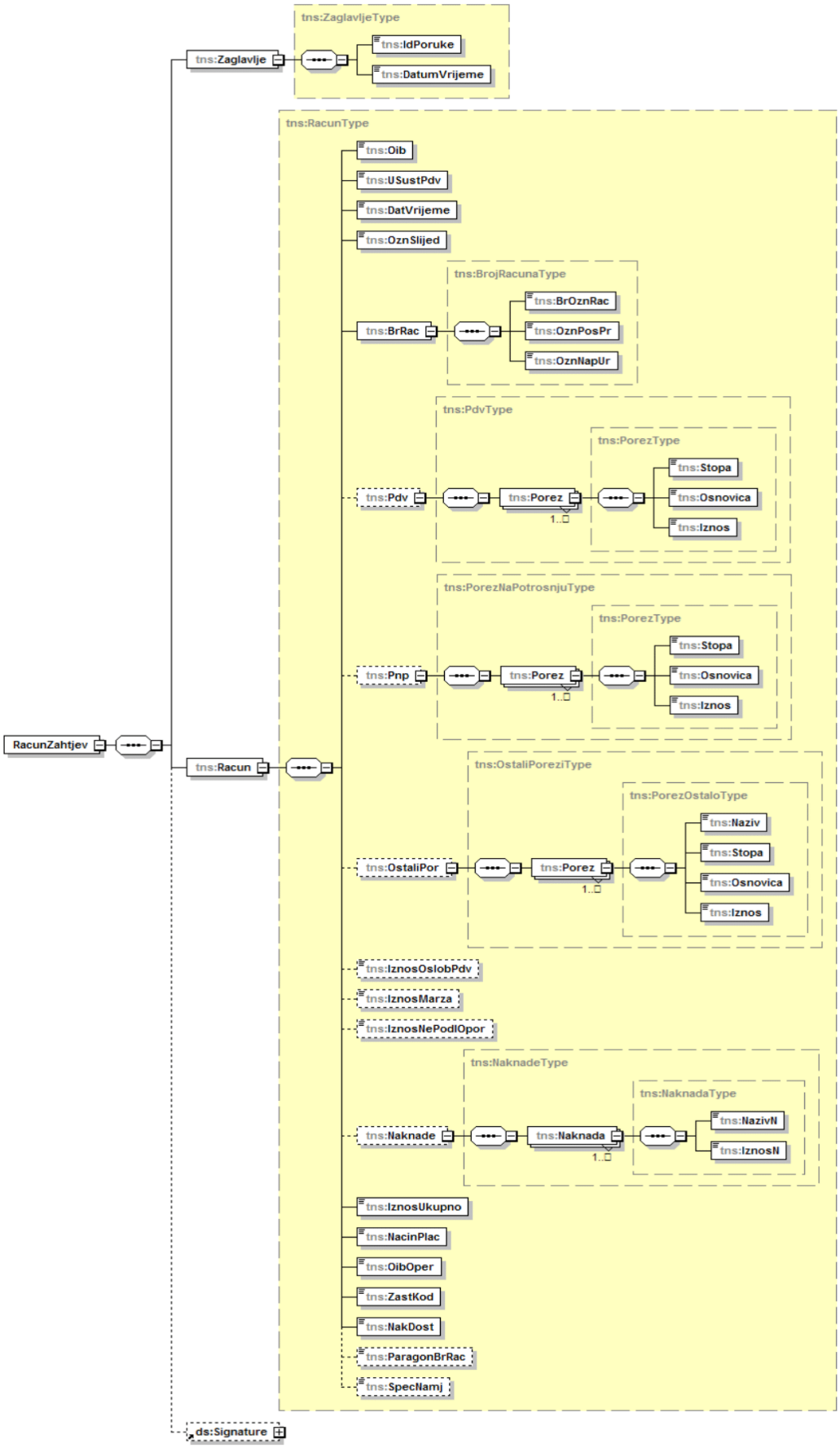
Tijelo XML-a predstavlja račun koji se mora sastojati od sljedećih parametara: OIB, u sustavu PDV, datum i vrijeme izdavanja, oznaka slijednosti, broj računa, ukupan iznos, način plaćanja, OIB operatera na naplatnom uređaju i ZKI. Također postoje dodatni parametri računa koji se upisuju samo ako postoje, kao što je PDV, razne naknade i drugi koji se mogu vidjeti u tehničkoj dokumentaciji. OIB predstavlja OIB obveznika fiskalizacija, što znači OIB osobe ili kompanije koja izdaje račune. Parametar u sustavu PDV je tipa „boolean“ i oznaka koja ako je istinit (engl. *true*) pokazuje da je obveznik u sustavu PDV-a ili ako je lažan (engl. *false*) da nije u sustavu PDV-a. Oznaka slijednosti je bitan parametar prilikom izdavanja računa. Svaki račun se može dodjeljivati ili s naplatnog uređaja (N) ili s poslovnog prostora (P), što znači da je jedan od tih parametara zapravo centralni uređaj. Ukoliko se broj račun dodjeljuje s naplatnog uređaja, svaki zasebni naplatni uređaj započinje dodjeljivanje broja računa od početka. Što znači da ako kompanija ima četiri naplatna uređaja, neovisno u kojem se poslovnom prostoru nalaze, svaki naplatni uređaj će prvi račun izdati s brojem „1“. U drugom slučaju, gdje se broj računa dodjeljuje s poslovnog prostora, početni broj računa „1“ će se dodjeljivati samo na poslovnom prostoru neovisno o tome koliko naplatnih uređaja ima. Što znači da ako postoje dva poslovna prostora i svaki ima dva naplatna uređaja, prvi naplatni uređaj u prvom poslovnom prostoru koji izdaje račun će imati broj „1“, a kad drugi naplatni uređaj na istom poslovnom prostoru koji još nije izdao krene u izdavanje računa njegov prvi račun će biti broj „2“ jer je broj računa „1“ već izdan. Shema dodjeljivanja broja računa se može vidjeti i na tablici 4.4.. Web aplikacija za fiskalizaciju opisana u ovom radu dodjeljuje broj računa s poslovnog prostora.

Oznaka slijednosti: poslovni prostor		**Oznaka slijednosti: Naplatni uređaj**		
PP1	PP2	NU - 1	NU - 2	NU – 3
1/PP1/1	1/PP2/2	1/PP1/1	1/PP1/2	1/PP2/3
3/PP1/2	2/PP2/1	2/PP2/1	2/PP2/2	2/PP1/3
3/PP1/3	3/PP2/2	3/PP1/1	3/PP2/2	3/PP2/3
4/PP1/1	4/PP2/3			

Tab 4.4. Shema dodjeljivanja broja računa

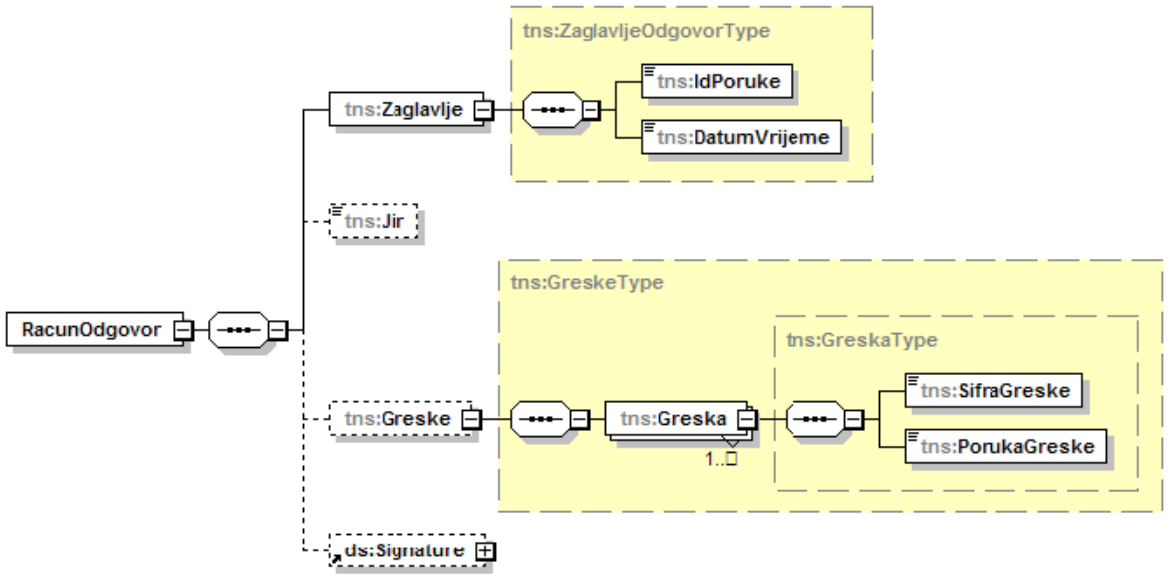
Broj računa se sastoji od tri dijela: brojčana oznaka računa, oznaka poslovnog prostora i oznaka naplatnog uređaja. Brojčana oznaka računa može sadržavati samo znamenke i nisu dopuštene vodeće nule, isto tako i oznaka naplatnog uređaja, ali on treba biti jedinstven na razini jednog poslovnog prostora obveznika. Oznaka poslovnog prostora može sadržavati i znamenke i slova i mora biti jedinstva na razini OIB-a obveznika. Broj računa izgleda ovako: 1234567/POSL1/10, gdje „1234567“ predstavlja brojčanu oznaku račune, „POSL1“ oznaku poslovnog prostora, a „10“ oznaku naplatnog uređaja. Parametar ukupan iznos je sam po sebi jasan i označava iznos iskazan na računu. Način plaćanja se sastoji od pet mogućnosti : K – kartice, G – gotovina, C- Ček, T – transakcijski račun i O – ostalo. OIB operatera na naplatnom uređaju je zapravo izdavatelj računa kupcu. Zaštitni kod izdavatelja obveznika fiskalizacije je alfanumerički zapis koji potvrđuje vezu između obveznika fiskalizacije i izdanog računa. To je zaštitni kod obveznika fiskalizacije koji se generira na temelju algoritma koji je opisan u tehničkoj dokumentaciji. Mora biti 32-znamenkasti broj zapisan u heksadecimalnom formatu i može sadržavati znamenke od nula do devet i mala slova: a-f [15]. Primjer ZKI-a je sljedeći: „e4d909a11bf4d4441aaddfe998cbd0d1“.

Od dodatnih parametara najvažniji je vjerojatno parametar PDV, odnosno porez na dodatnu vrijednost. PDV se sastoji od tri dijela koja su obavezna ako postoji PDV u računu: porezna stopa, osnovica i iznos poreza. Parametri PNP (porez na potrošnju) i ostali porezi se sastoje od istih dijelova kao i PDV, samo što parametar ostali porezi ima još jedan dodatan: naziv poreza.



Sl 4.5. Shema XML poruke

Slanjem poruke zahtjeva s računom na CIS korisnik će od njih dobiti poruku odgovara na račun. Poruka odgovora se sastoji od tri dijela: zaglavlje, jedinstveni identifikator računa i greška. Zaglavlje se sastoji od identifikatora poruke (UUID) i kao identifikator poruke odgovora se uzima ID iz poruke zahtjeva i od datuma i vremena obrade. JIR se šalje samo u slučaju da nema greške. Greške se sastoje od šifre greške i poruke. Šifra greške je u formatu sXXX, gdje je XXX prirodni broj, svaka greška se može naći u šifrarniku grešaka. Poruka je tekstualni opis greške koji je sukladan šifrarniku grešaka. Slika 4.6. prikazuje shemu poruke odgovora.



Sl 4.6. Shema poruke odgovora na račun

4.4. Postupak spajanja i fiskalizacija

Korisnik web aplikacije za fisklizaciju prvo mora napraviti račun sa svim potrebnim parametrima, kao što je već i objašnjeno. Nakon što se račun napravi on se može fiskalizirati. Svaki račun koji je izdan danas, što bi značilo da mu datum računa nije niti prije niti poslije današnjeg dana, se može fiskalizirati. Račun se fiskalizira odabirom dugmeta za fiskalizaciju koji se može naći u tablici izlaznih računa na stranici računa pod kolonom fiskalizacija. Dugme će biti zaključano ako je račun istekao ili će biti zatamnjeno i označeno kvačicom ako je već fiskaliziran račun. Kada se klikne na dugme za fiskalizaciju poziva se Ajax metoda koja uzima identifikator računa. Ajax metoda se zatim povezuje na rutu „/racuni/xml/“ i to je POST ruta koja je povezana s kontrolerom za fiskalizaciju: „Route::post('racuni/xml/{id}', 'FiskalizacijaController@fiskal');“. Ajax šalje identifikator računa kontroleru za fiskalizaciju putem rute. Kontroler preuzima identifikator račune kao varijablu „$id“. S tom varijablom on pronalazi račun, izlazni račun,

poslovni prostor, naplatni uređaj i sve usluge vezane uz taj identifikator. Svi ti podaci se spremaju u svoje varijable kako bi ih se moglo kasnije koristiti.

```
var urls ="/racuni/xml/";
   $.ajaxSetup({
         headers: {
          'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
          }
        });
      var id_xml = $(this).attr("value");
        $.ajax({
          type: "POST",
          url: urls+ id_xml,
          success: function (data) {
            swal("Uspjeh!", "Račun je fiskaliziran.", "success");
            $('#fiskal_btn').attr('class', 'fa fa-lock')
          },
          error: function (data) {
            swal("Pogreška!", "Račun nije fiskaliziran. Probajte ponovno kasnije!", "error");
          }
    });
```

```
Route::post('racuni/xml/{id}', 'FiskalizacijaController@fiskal');
```

```
class FiskalizacijaController extends Controller
{
   public function fiskal($id) {

     $bill = Bill::find($id);
     $outgoing_bill = $bill->billable;
     $businessplace = $outgoing_bill->businessplace;
     $chargingdevice = $outgoing_bill->chargingdevice;
     $cashier = $outgoing_bill->cashier;
     $outgoingServices = Outgoing_service::where('outgoingbill_id', $outgoing_bill->id)->get();
     $bill_date = $bill->datum_izrade;
.....

}
```

Sl 4.7. Proces slanja identifikatora kontroleru za fiskalizaciju

Prvi korak fiskalizacije koju kontroler odrađuje je rukovanje s FINA aplikacijskim certifikatom u „.pfx“ formatu. U prvom koraku se postavlja lozinka certifikata, to je lozinka koja se dobije od FINA-e i određuje se putanja do certifikata. Zatim je potrebno parsirati i spremiti certifikat u niz. To odrađuje funkcija „openssl_pkcs12_read()“ koja prima kao prvi parametar sami certifikat, kao drugi parametar se postavlja željeni naziv niza u koji će spremiti certifikat i treći parametar je lozinka certifikata. Također, potrebno je prije postavljanja certifikata kao parametra „openssl_pkcs12_read()“ funkcije preuzeti njegov sadržaj. Sadržaj certifikata se preuzima pomoću

funkcije „file_get_contents()“. Iz parsiranog certifikata koji je spremljen u varijablu „$certificate“ potrebno je izvaditi privatni ključ i informacije o certifikatu. Kako bi se došlo do privatnog ključa i moglo ga se koristit dalje kroz aplikaciju potrebna je funkcija „openssl_pkey_get_private()“. Isto je potrebno napraviti za informacije o certifikatu kako bi se sve informacije dobile kao niz te se za to koristi funkcija „openssl_x509_parse()“.

```
$certificate = null;
$certPswd = '**********'; //postaviti lozinku
pfxCert = public_path("cer/fiskal_demo.pfx");
openssl_pkcs12_read(file_get_contents($pfxCert), $certificate, $certPswd);
$cert = $certificate['cert'];
$pKey = $certificate['pkey'];
$pKeyRes = openssl_pkey_get_private($pKey, $certPswd);
$certData = openssl_x509_parse($cert);
```

Sl 4.8. Rukovanje s FINA aplikacijskim certifikatom

Nakon rukovanja s FINA aplikacijskim certifikatom potrebno je rukovati s FINA CA certifikatom u „.cer“ formatu. Potrebno je certifikat koji je u „.cer“ formatu pretvoriti u certifikat koji je „.pem“ formata. Prvo se učitava certifikat koji se nalazi u datoteci naziva „cer“. Zatim se certifikat iz datoteke učitava u varijablu koristeći već spomenutu funkciju „file_get_contents()“. Nakon toga se certifikat kako je već propisano pretvori u „.pem“ tako što se na početak doda fraza „begin certificate“ koja označava početak certifikata i na kraj se dodaje „end certifikate“ koja označava kraj certifikata. Također, sami je certifikat potrebno šifrirati s „base64“ kodiranjem što odrađuje funkcija „base64_encode()“. Kako bi šifriranje radilo, prvo je potrebno certifikat podjeli u manje komade funkcijom „chunk_split()“ te ih se onda šifrira „base64“ kodiranjem. Nakon toga se samo koristi funkcija „file_put_contens()“ kako bi se certifikat formata „.pem“ spremio u varijablu za sljedeće korištenje. S ova dva koraka je riješeno rukovanje sa certifikatima i sljedeći korak je generiranje zaštitnog koda ZKI.

Zaštitni kod izdavatelja je alfanumerički zapis kojim se potvrđuje veza između obveznika fiskalizacije i izdanog računa. Zaštitni kod se ispisuje na računu i dostavlja se poreznoj upravi kao obavezni element računa. Osnovna namjena ZKI-a je zaštita obveznika fiskalizacije od pokušaja nanošenja štete, jer samo obveznik fiskalizacije može ponovno izraditi isto zaštitni kod na temelju istih ulaznih parametara. Zaštitni kod mora biti određen s parametrima koji osiguravaju jedinstvenost računa i autentičnost korisnika. Jedinstvenost računa osiguravaju: OIB obveznika, datum i vrijeme izdavanja računa, brojčana oznaka računa, oznaka poslovnog prostora, oznaka naplatnog uređaja i ukupni iznos računa. Autentičnost korisnika se osigurava putem elektroničkog

potpisa s FINA certifikatom za fiskalizaciju koji je dodijeljen obvezniku. Pri računanju zaštitnog koda se koristi UTF-8 (engl. *Unicode Transformation Format 8*) kodiranje. Prilikom korištenja decimalnih mjesta mora se koristiti točka, a ne zarez. Elektroničko potpisivanje se izvodi korištenjem RSA-SHA1 elektroničkog potpisa. Zatim se korištenjem md5 kriptografske funkcije dobije rezultat koji je 32-znamenkasti broj koji se ispisuje na računu i predstavlja ZKI [15]. Kao što se može i vidjeti iz koda na slici 4.9., prvo se učitava OIB obveznika u istoimenu varijablu, zatim se uzima trenutno vrijeme kao vremenski atribut. Kontroler sada uzima datum izdavanja računa, brojčanu oznaku računa, oznaku poslovnog prostora, oznaku naplatnog uređaja i iznos računa iz već spomenutih varijabli u koje su spremljene potrebne vrijednosti računa preuzete iz baze podataka za odabrani račun. Zatim se svaki od tih podataka dodaje u varijablu koja se potpisuje funkcijom „openssl_sign()“. Ta funkcija prima nepotpisanu varijablu sa svim potrebnim parametrima, naziv varijable u koju želimo spremiti potpisanu varijablu, privatni ključ i algoritam za potpisivanje. Nakraju se uzima potpisana varijabla i funkcijom „md5()“ kriptira u 32-znamenkasti broj koji se sastoji od brojeva od nula do devet i slova od a do f. Ta kriptirana varijabla predstavlja važeći ZKI.

```
$oib = '18952151960';
$dt = new \DateTime('now');
$datum = $datum_izrade_racuna; //iz varijable
$broj_racuna = $bill->broj_racuna; //pozivanje relationshipa za broj racuna
$poslovni_prostor = $businessplace->label; //pozivanje relationshipa za prostor
$naplatni_uredaj = $chargingdevice->label; //pozivanje relationshipa za uredaj
$ukupni_iznos = number_format($bill->iznos, 2);//pozivanje relationshipa za iznos
$ZKI_un = '' . $oib . $dt . $broj_racuna . $poslovni_prostor . $naplatni_uredaj . $ukupni_iznos;
$ZKI_sign = null;
openssl_sign($ZKI_un, $ZKI_sign, $pKeyRes, OPENSSL_ALGO_SHA1);
$ZKI = md5($ZKI_sign);
```

Sl 4.9. Generiranje zaštitnog koda izdavatelja

Kad su svi potrebni parametri za XML dokument napravljeni potrebni je i generirati XML dokument. Za generiranje XML dokumenta se koristi ekstenzija „XMLWriter“. Prvo se stvara novi XML koristeći „openMemory()“. Svaki novi element XML-a se dodaje koristeći „startElementNS()“ kojem se kao argumenti predaju prefiks, naziv elementa i imenski prostor URI-a. Kad se započne element u njega se mogu upisivati elementi i atributi. Atributi se dodaju korištenjem „writeAttribute()“ i kao argumente prima naziv atributa i vrijednost atributa. Element se dodaje pomoću „startElementNS()“ koji prima prefiks, naziv elementa i imenski prostor URI-a. Važno je napomenuti da se za imenski prostor URI-a može postaviti „null“ ako ga nema. Prefiks je isti kroz cijeli XML te je „tns“ i dodaje se kao varijabla koja je unaprijed deklarirana i

postavljena na „tns“. Kad se završi upisivanje svih potrebnih atributa i elemenata u element, zatvara se korištenjem „endElement()“. Kad se cijeli XML dokument završi vraća se trenutni međuspremnik u varijablu koja sadrži cijeli napisani XML dokument.

Nakon što je XML napravljen potrebno ga je potpisati certifikatom. Za potpisivanje XML-a koristi „DOMDocument“ spremnik koji predstavlja cijeli HTML ili XML dokument. Za učitavanje XML-a u „DOMDocument“ se koristi „loadXML()“ koji prima već spomenutu varijablu koja sadrži cijeli napisani XML dokument. Prije dodavanja certifikata potrebno je učitani XML dokument pretvoriti u kanonski oblik (engl. *canonical form*). Taj oblik omogućuje stvaranje fizičke reprezentacija XML dokumenta koji dopušta razne promjene u XML sintaksi bez mijenjanja njenog značenja. Na primjer, redoslijed atributa u XML dokumentu je nevažan i stoga ako jedan dokument ima atribute složene abecedno, a drugi dokument složene na neki drugi način, oba dokumenta što se XML tiče su identična, iako je različita fizička reprezentacija podataka što može dovesti do problema. Tako da ako imamo digitalno potpisani dokument koji dokazuje da nije mijenjan, promjena redoslijeda atributa bi onda poništio potpis, iako je što se XML-a tiče, dokument nije stvarno mijenjan [16]. Tako da je onda rješenje pretvoriti XML dokument u kanonski oblik prije potpisivanja i sličnih operacija, što se i odrađuje uporabom „C14N()“ funkcije. Nakon toga se pravi „hash“ kanonskog oblika XML dokumenta koristeći SHA1 algoritam te se cijeli „hash“ šifrira „base64“ kodiranjem, koji je potreban kao jedan od atributa potpisa naziva „DigestValue“. Iz XML-a je potrebno dobiti korijenski čvor elementa i sprema se u varijablu naziva „rootElem“. Zatim se prema opisanom primjeru elektronički potpisane poruke zahtjeva DEMO certifikata iz tehnički dokumentacije dodaju elementi korištenjem funkcije „appendChild()“ i atributi funkcijom „setAttribute()“.

```
$XMLReqDOM = new \DOMDocument();
$XMLReqDOM->loadXML($XMLRequest);
$canonical = $XMLReqDOM->C14N();
$DigestValue = base64_encode(hash('sha1', $canonical, true));
$rootElement = $XMLReqDOM->documentElement;
$SignNode = $rootElement->appendChild(new \DOMElement('Signature'));
$SignNode->setAttribute('xmlns','http://www.w3.org/2000/09/xmldsig#');
```

Sl 4.10. Potpisivanje XML-a sa certifikatom

Potpis (engl. *signature*) element ima strukturu koja se sastoji od: „<SignedInfo>“, „<SignatureValue>“ i „<KeyInfo>“ elemenata. Unutar „SignedInfo“ elementa su definirani podaci o kanonikalizaciji i metodi potpisivanja. Kanonikalizacijska metoda koja se koristi prilikom potpisa poruka zahtjeva obavezno mora biti „Exclusive XML Canonicalization“, kao što se može

i vidjeti u elementu „<CanonicalizationMethod>“. Metoda potpisivanja je uvijek „RSA-SHA1“ te je to vidljivo u elementu za potpisivanje „<SignatureMethod>“. Element „SignedInfo“ se još sastoji od elementa „<Reference>“ i taj element ima atribut URI koji se odnosi na podatke koji se potpisuju, u slučaju ove web aplikacije to je uvijek „RacunZahtjev“. Element „Reference“ unutar sebe još ima elemente : „<Transforms>“, „<DigestMethod>“, „<DigestValue>“. Elementi „Transforms“ i „DigestMethod“ se popunjavaju vrijednostima koji se mogu naći u tehničkoj dokumentaciji, dok element „DigestValue“ ima vrijednost izračunatu algoritmom koji je već prije opisan. „SignatureValue“ element sadrži kodiranu vrijednost koja predstavlja elektronički popis unutar „SignedInfo“ elementa. Vrijednost za „SignatureValue“ element se dobije tako da se iz XML-a uzme atribut od oznake „SignedInfo“ te već napravljena varijabla za privatni ključ i onda se funkcijom „openssl_sign()“ kodira varijabla koja sadrži potrebnu i ispravnu vrijednost za „SignatureValue“ element. Element „KeyInfo“ sadržava podatke o certifikatu pomoću kojeg je potpisan zahtjev i sastoji se od elementa „<X509Data>“. Unutar elementa „X509Data“ se nalaze elementi „<X509Certificate>“ i „<X509IssuerSerial>“ koji se još sastoji od elemenata „<X509IssuerName>“ i „<X509IssuerNumber>“. Element „X509Certificate“ sadrži dio aplikacijskog certifikata sa „cert“ koji zapravo sadrži informacije u certifikatu i već je od prije bio spremljen u varijablu. Element „X509IssuerName“ se sastoji od elemenata „OU=DEMO, O=FINA i C=HR“ koji su uzeti iz istog certifikata samo ovaj put dio koji sadrži „issuer“, a element „X509IssuerNumber“ sadrži serijski broj tog certifikata. Primjer gotovog izgleda potpisa se može vidjeti na slici 4.11.

```
<tns:RacunZahtjev Id="RacunZahtjev" xmlns:tns="http://www.apis-it.hr/fin/2012/types/f73"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <tns:Zaglavlje>
    <tns:IdPoruke>f81d4fae-7dec-11d0-a765-00a0c91e6bf6</tns:IdPoruke>
    <tns:DatumVrijeme>01.09.2012T21:10:34</tns:DatumVrijeme>
  </tns:Zaglavlje>
  <tns:Racun>
  ...
  </tns:Racun>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <Reference URI="#RacunZahtjev">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
          <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>VItfxY/A1BITZ/BuWpsGd9gKix4=</DigestValue>
      </Reference>
    </SignedInfo>
        <SignatureValue>0+5UDLzJuGy56HojH510+d.....</SignatureValue>
    <KeyInfo>
      <X509Data>
        <X509Certificate>MIIEyDCCA7CgAwIBAgIEPssQ2TANBgkqh...</X509Certificate>
        <X509IssuerSerial>
          <X509IssuerName>OU=DEMO,O=FINA,C=HR</X509IssuerName>
          <X509SerialNumber>1053495513</X509SerialNumber>
        </X509IssuerSerial>
      </X509Data>
    </KeyInfo>
  </Signature>
</tns:RacunZahtjev>
```

Sl 4.11. Primjer gotovog potpisa XML-a

Zadnji korak s XML-dokumentom je dodavanja SOAP-a na već potpisani XML. SOAP omotava čitav potpisani XML i sastoji se od elementa omotnice „<soapenv:Envelope>“ i od tijela „<soapenv:Body>“. Također je potrebno dodati i način kodiranja koji je UTF-8 i verziju XML-a 1.0. Element „envelope“ sadrži atribut koji je definiran u tehničkoj dokumentaciji. SOAP se učitava u varijablu funkcijom „loadXML()“. Kad se učita sintaksa SOAP XML-a, učitavaju se verzija i način kodiranja te se njima omotava već napravljeni potpisani XML. Nakraju se samo novi XML koji se sad sastoji i od SOAP-a spremi u varijablu funkcijom „saveXML()“.

Nakon što je XML potpisan i na njega dodan SOAP, potrebno je pokrenuti POST zahtjev s tim XML-om prema CIS-u. Za pokretanja POST zahtjeva prema CIS-u se koristi biblioteka cURL. CURL je biblioteka koja dozvoljava spajanja i komuniciranje sa serverom putem velikog broja različitih protokola koje podržava. Najčešće se koristi za slanje podataka između servera. Prvo je potrebno stvoriti cURL resurs naredbom „curl_init()“. Prilikom inicijalizacije je potrebno postaviti opcije koje su potrebne za konekciju. Postoji veliki broj opcija koja cURL podržava i sve se mogu naći na službenim stranicama biblioteke. Za potrebe ove konekcije su se koristile opcije za namještanja isteka konekcije, vraćanja podataka nakon uspješne transakcije kako bi se dobio odgovor servera, postavio se protokol kao istinit (engl. *true*) i kao datoteka koja se šalje dodan potpisani XML. Također, zbog potreba certifikacije, FINA CA certifikat koji je već konvertiran u „.pem“ format je dodan u opciju „CURLOPT_CAINFO“. Zatim se funkcijom „curl_setopt_array“ postavljaju sve navedene opcije za cURL sesiju. Naredbom „curl_exec()“ se pokreće cURL sesija i provjerava se je li sesija uspješna. Za provjeru uspješnosti se koristi funkcija „curl_getinfo()“ u koju su kao argumenti postavljeni konekcija i opcija „CURLINFO_HTTP_CODE“ i zbog te opcije vraća zadnji primljeni HTTP kod. Upravo zbog toga se uspješnost provjerava tako da se rezultat te funkcije uspoređuje s vrijednosti dvije stotine, jer dvije stotine je HTTP kod da je sve uredu. U slučaju da se pojavi kod greške, to jest različit od dvije stotine, pokreće se dio koda koji iz poruke odgovora uzima grešku iz oznake „SifraGreske“ i oznake „PorukaGreske“ te ih ispisuje kako bi se znalo zašto je do greške došlo. Ukoliko nije došlo do greške, iz poruke odgovora se preuzima JIR. Zatim se pokreće se skripta koja upisuje JIR i ZKI u bazu podatka za račun na kojem se to izvodilo i postavlja se zastavica fiskalizacije tog računa na istinu, kako bi se znalo u aplikaciji da je fiskaliziran. Nakraju se zatvori cURL konekcije funkcijom „curl_close()“.


```
$response = curl_exec($ch);
if ($response) {
        $code = curl_getinfo($ch, CURLINFO_HTTP_CODE);
        $DOMResponse = new \DOMDocument();
        $DOMResponse->loadXML($response);
         if ($code === 200) {
            $Jir = $DOMResponse->getElementsByTagName('Jir')->item(0);
            if ($Jir) {
               echo $Jir->nodeValue;
               $fiskal = new Fiskalization;
               $fiskal->jir = $Jir->nodeValue;
               $fiskal->zki = $ZastKod;
               $fiskal->outgoing_bills_id = $outgoing_bill->id;
               $fiskal->save();
               $outgoingBILL = Outgoing_bill::find($outgoing_bill->id);
               $outgoingBILL->fiskalizirano = 1;
               $outgoingBILL->save();
             }
             echo $response;
....
```

Slika 4.12. Prikaz ispravnog odgovora FINA-e i koda koji to izaziva

5. TESTIRANJE

Testiranje je pokusna provjera rada softvera ili njegovih dijelova korištenjem umjetno pripremljenih podataka i slučajeva koje se izvode uz analiziranje podataka s namjerom pronalaženja kvara. Svrha testiranja je verifikacija ili validacija. Verifikacija se odnosi na provjere radi li softvera prema svojoj specifikaciji, a validacija odgovara li stvarnim potrebama korisnika. Proces testiranja softvera ili životni ciklus testiranja (engl. *STLC*) se obično sastoji od niza koraka: analiza zahtjeva, planiranje testa, razvoj testnih slučajeva, postavljanje testnog okruženja, izvođenje testa i završetak testnog ciklusa [17]. Testiranje web aplikacija ima šest tehnika koja se najčešće koriste [18]:

- Funkcionalno testiranje
- Testiranje upotrebljivosti
- Testiranje sučelja
- Testiranje kompatibilnosti
- Testiranje performansi
- Sigurnosno testiranje

Funkcionalno testiranje se sastoji od niza provjera kao što su provjere dostupnosti linkova, validacija formi, validacija HTML-a i CSS-a i slično. Testiranje upotrebljivosti je testiranje korisničke navigacije web aplikacijom, to jest provjera rada dugmadi koje vode na linkove, otvaranja prozora i tako dalje. Testiranjem sučelja se provjerava jesu li zahtjevi ispravno poslani na bazu podataka, prikazuje li aplikacija ispravno podatke, hoće li server vraćati pogrešku prilikom zahtjeva, vraćaju li upiti baze podataka očekivane rezultate. Testiranje kompatibilnosti je zapravo testiranje hoće li aplikacije ispravno raditi u različitim mrežnim preglednicima i operacijskim sustavima. Testiranje performansi je testiranje kako se aplikacije nosi s velikim brojem korisnika, zahtjeva, i slično. Sigurnosno testiranje provjerava koliko je aplikacija sigurna od stranih napada, neovlaštenog korištenja, dodavanja parametara na zahtjeve i tako dalje.

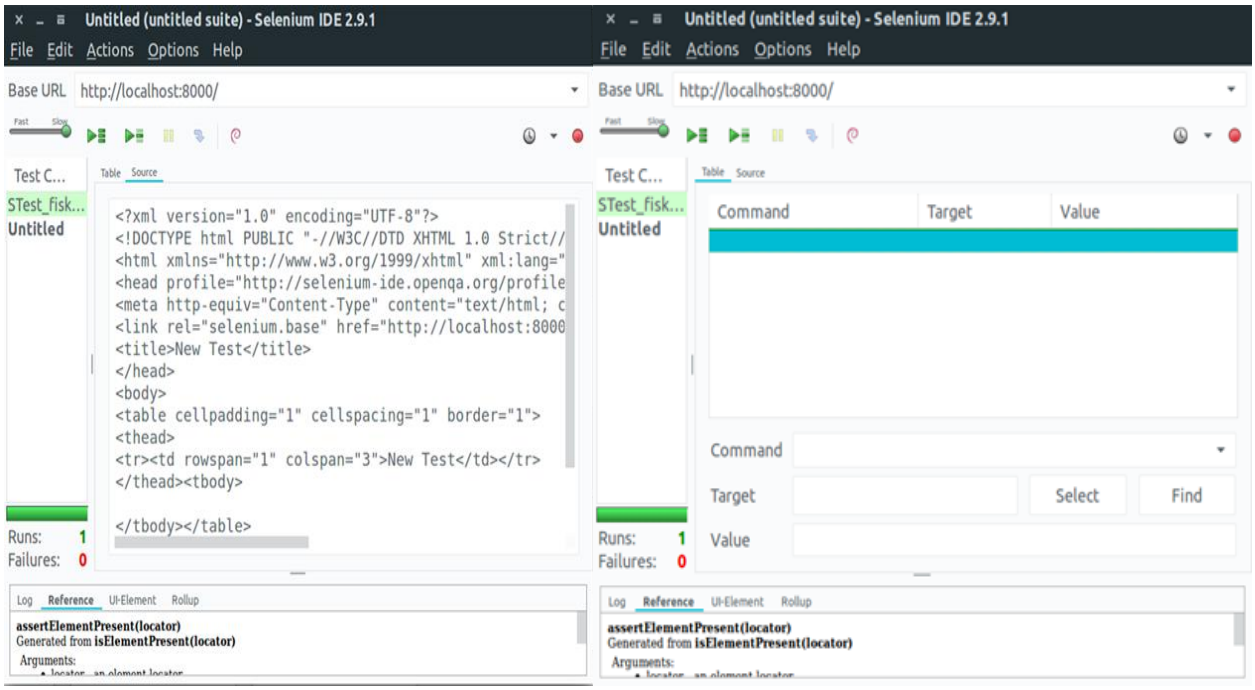
Testiranje web aplikacije za fiskalizaciju će se izvodit na dva načina, testiranjem komunikacije s fina poslužiteljem i testiranjem same web aplikacije. Testiranje komunikacije s FINA poslužiteljem se preveo slanjem niza različitih odgovora bilo ispravnih ili neispravnih kako bi se provjerilo kako će se poslužitelj ponašati. Testiranje rada web aplikacije se provelo korištenjem Selenium IDE alata i PHPunit testiranja koje je integrirano u programskom okruženju Laravel

5.1. Testiranje web aplikacije

Prilikom testiranja web aplikacije za fiskalizaciju se izvodilo funkcionalno testiranje, jedinično testiranje, testiranje kompatibilnosti, testiranje upotrebljivosti, testiranje sučelja i testiranje baze podataka.

5.1.1. Selenium testiranje

Za funkcionalno testiranje koristio se Selenium IDE alat za pravljenje testnih slučajeva koji se automatski izvršavaju. Selenium IDE je zapravo Mozilla Firefox dodatak (engl. *plugin*) koji omogućuje snimanje, izmjenu, izvođenje i otklanjanje kvarova (engl. *debugging*). Testni slučajevi ili skripte su niz komandi koje se snimaju automatski, ali se mogu i ručno unositi. Selenium omogućuje testiranje web aplikacija na različite načine, od testiranja postojanja nekog elementa na osnovu HTML oznaka i odabira nekog od tih elemenata, do popunjavanje polja za unos teksta i slanja formi.



Slika 5.1. Selenium IDE sučelje

Slika 5.1. prikazuje Selenium IDE sučelje u kojem se snimaju i izrađuju testovi te provode isti. Za korištenje Selenium IDE alata potrebno je upisati pod „base url“ link aplikacije koja se želi testirati i odabrati snimanje klikom na crveno dugme. Nakon toga započinje korištenje aplikacije i upisuju se podaci, odabiru linkovi, popunjavaju i šalju forme, itd., to jest sve što je potrebno za testiranje aplikacije. Korištenjem aplikacije glavni Selenium IDE prozor, odnosno test skripta se popunjava naredbama (engl. *command*), ciljevima (engl. *target*) i vrijednostima (engl. *value*). Ti

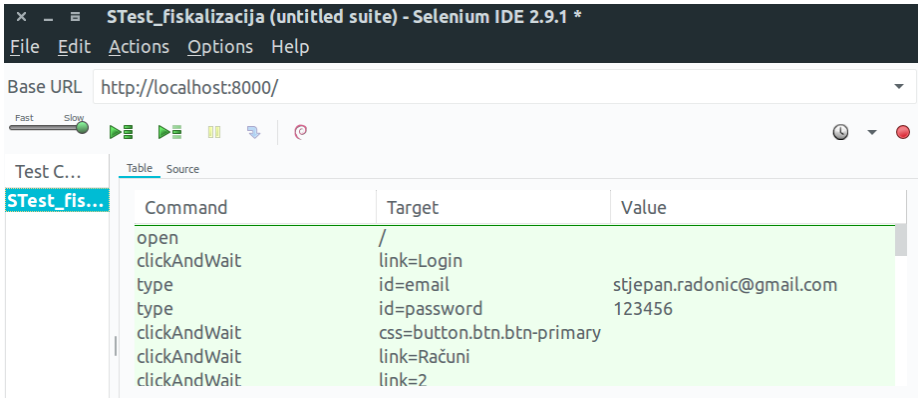
podaci se mogu i izmijeniti dodavanjem novih komandi, novih komentara, brisanjem nekih naredbi i tako dalje. Izmjena se odvija desnim klikom u prozor za naredbe ili odabirom dugmeta „Edit“ u traci izbornika. Izvršavanje testove se kontrolira odabirom dugmeta „Actions“ te izborom jednom od ponuđenih opcija. Test skripta se može pokrenuti pojedinačno ili u grupi. Može se upravljati i brzinom izvođenja testova, može se proći kroz test liniju po liniju, proći samo određenu liniju, zaustavi test usred izvođenja i tako dalje. Ovaj izbornik omogućuje i opcije za otklanjanje kvarova. Sve akcije se mogu izvoditi odabirom jedne od navedenih opcija na alatnoj traci ili desnim klikom unutar prozora snimljenih naredbi. Prozor snimljenih naredbi se dijeli na još dva prozora, tablica (engl. *table*) i izvor (engl. *source*). Prozor tablica (slika 5.1.) kao što je već spomenuto, se sastoji od tri kolone. Kolona naredbi koje sadrže naredbe koju korisnik može ili unijeti ili izabrati iz liste dostupnih naredbi, kolona ciljeva koji opisuju položaj elementa na koji se odnosi naredbe i kolona vrijednosti koji se sastoje od ulaznih vrijednosti korisnika za neke naredbe, kao što je to naredba za unos teksta. Prozor izvor prikazuje testnu skriptu u formatu u kojoj je spremljena, predodređeni i standardni format je HTML format (slika 5.1.), ali korisnik može odabrati spremanju i u nekom drugom formatu kao što je to C#, Java, Python ili Ruby. Nakon što se testiranje završi, osim što se vizualno može primijetiti na prozoru tablica ima li greške, Selenium IDE ima prozor sa četiri kartice koji prikazuju detaljan opis provedenih akcija. Kartica „Log“ prikazuje status svake naredbe koja se izvršava i poruku o grešci, ukoliko do greške dođe, što je važno za njihovo otklanjanje. Kartica „Reference“ sadrži opis svake naredbe, broj parametara, njihov redoslijed i tip, prilikom pisanja ili izmjene testova potrebno je voditi računa da vrijednosti polja ciljeva i vrijednosti za izabranu naredbu odgovaraju upisu komande na „Reference“ kartici.

Selenium IDE alatom je napravljen test koji obuhvaća najvažnije aktivnosti aplikacije te ih sve provjerava jednim testom, kao što je i uobičajena praksa. Funkcionalnosti web aplikacije za fiskalizaciju koje se provjeravaju ovim testom su:

1. Provjera pristupa aplikaciji
2. Provjera dostupnosti stranice „racuni“
3. Provjera postojanja najvažnijih elemenata na stranici „racuni“
4. Provjera funkcionalnosti osnovnih elemenata
5. Provjera odjave iz aplikacije

Testiranje se izvodilo na način da se pod „base url“ unio URL lokalnog mjesta web stranice koji glasi: localhost:8000. Nakon toga se pritiskom na crveno dugme u Selenium IDE alatu započelo snimanje akcije korisnika dok se, u isto vrijeme, odgovarajuće komande upisivale u

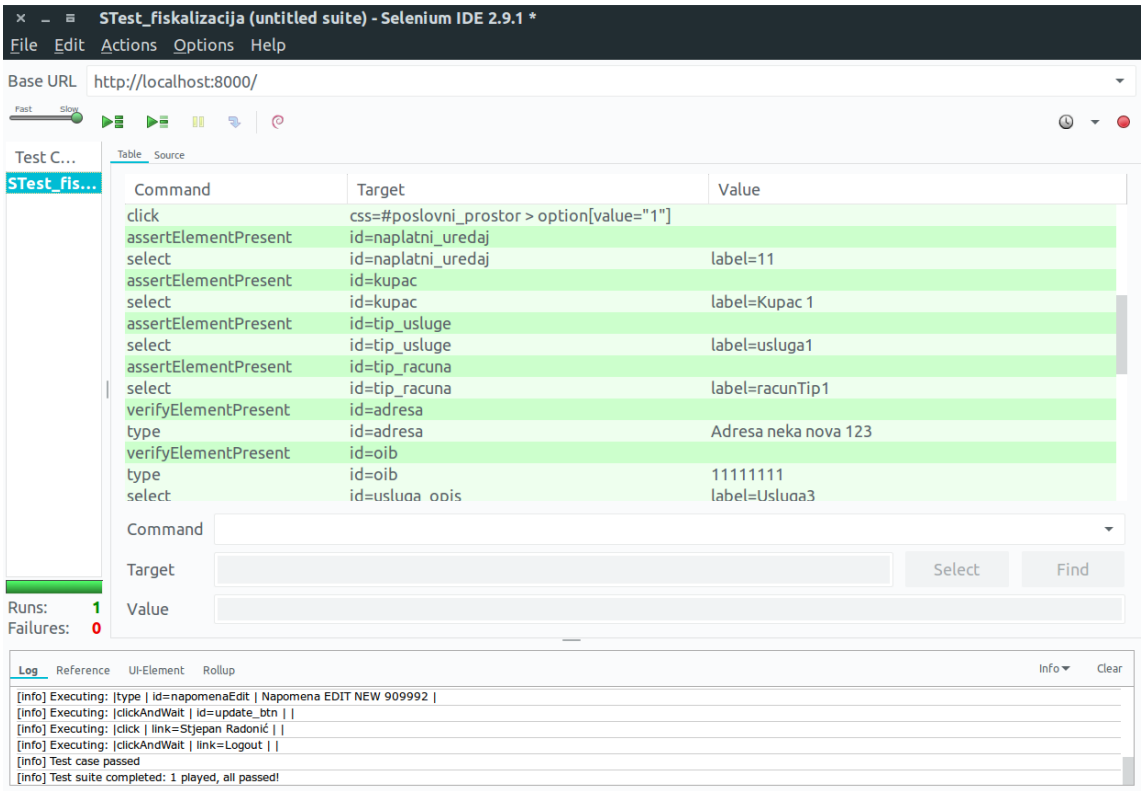
prozor za pisanje i izvršavanje test skripta. Upisivale su se akcije, odabir linka, upis vrijednosti u odgovarajuća polja, odabir željene opcije iz liste, klik na dugmad i druge.



Sl 5.2. Test skripta (prijava korisnika)

Slika 5.2. prikazuje dio test skripte za prijavu dobivenu snimanjem. Naredba „open“ se odnosi na otvaranje stranice čiji URL je naveden u „Base url“ polju. Zatim kreće prijava korisnika u aplikaciju, „clickAndWait“ označava odabir linka za prijavu u stranicu te čekanje na učitavanje stranice. Sljedeće dvije naredbe su „type“ i označavaju unos adrese e-pošte i lozinku u polja za prijavu aplikacije. Nakon toga slijede naredbe koje su kombinacija već navedenih naredbi, s time da je naredba „click“ slična kao i „clickAndWait“ samo što ne čeka da se stranica učita prije nego prijeđe na sljedeću naredbu.

Ako se pokrene izvršavanje testa cijelog testa on će se uspješno izvršiti, kao što se može vidjeti na slici 5.3. Međutim, kako bi se ovaj test mogao nazvati pravim testom potrebno je dodati naredbe koje provjerava i osiguravaju postojanje elemenata na stranica koje su neizostavne za rad i testiranje. Za provjeru postojanja elemenata korištene su naredbe: „assertElementPresent“ i „verifyElementPresent“. Razlika između te dvije naredbe je ta što „assert“ naredba prekida izvođenje testa ako element nije pronađen, a „verify“ naredba omogućava nastavak testiranja ako element nije pronađen. Naredba „assertElementPresent“ se koristi prilikom testiranja elemenata bez kojih aplikacije ne može raditi ispravno, kao što se može vidjeti na slici 5.3. gdje se forma za dodavanje računa ne može izvršiti ako nije dodan na primjer „tip_racuna“ ili bilo koji drugi element koji ima prije sebe „assert“. Za postojanje ostalih elemenata koji nisu neizostavni ili na kojima se neće izvršavati dodatne akcije se koristi naredbe „verifyElementPresent“ te kako se vidi na slici 5.3., bez elementa „adresa“ se može izvršiti dodavanje računa te ako ga nema, nije problem za funkcionalnost aplikacije.



Slika 5.3. Uspješno izvedena test skripta

Testiranje Selenium IDE alatom je bilo uspješno te su sve provjere koje su zadana zadovoljene. To se može provjeriti u izvještaju (engl. *log*) Selenium IDE alata, na kojem je napisano je li testiranje odnosno test skripta uspješno provedena ili ne (slika 5.3.).

5.1.2. PHPUnit

Za jedinično testiranje, testiranje baze podataka, integracijsko testiranje i testiranje upotrebljivosti se koristio PHPunit alat. Budući da je web aplikacija za fiskalizaciju napravljena u Laravel programskom okviru, korišten je PHPunit koji je integriran u Laravelu. Laravel omogućuje korisnicima izradu testova unutar svoga sučelja naredbom „php artisan make:test TestName“. Svaki napravljeni test se nalazi u „tests“ folderu te se tamo i izmjenjuje kako bi odgovarao korisničkoj namjeni. Pokretanje testova se obavlja upisivanjem naredbe „phpunit“ u terminalu te se testovi izvode i dobiju se rezultati testova u istom prozoru terminala. Rezultat može biti „OK“ što označuje da su svi testovi ispravno provedeni ili „Error“ što označuje grešku prilikom izvođenja i daje korisniku izvještaj o grešci. Detalji o izvođenju testa i o mogućim greškama se mogu naći u Laravel dokumentu „laravel.log“ koji se nalazi u datoteci „logs“. Laravel omogućuje dobru interakciju s aplikacijom kroz niz opcija za testiranje koje se mogu naći u dokumentaciji kao što su: rad s formama, rad s bazom podataka, provjera linkova, validacija tipa podataka, i tako dalje.

PHPunit testom u aplikaciji za fiskalizaciju će se provjeravati sljedeće:

1. Provjera naslova i tablica
2. Provjera likova
3. Provjera pristupa stranicama
4. Provjera rada formi
5. Provjera modela
6. Rad s bazom podataka

Za provjeru naslova i tablica je napravljen test „NameTableTest“. Unutar tog testa je definirano pet funkcija. Od tih pet funkcija, četiri funkcije provjeravaju postojanje najvažnijih elemenata na svim stranicama aplikacije, dok zadnja funkcija služi za definiranje korisnika. Korisnika je potrebno definirati jer se bez njega ne može pristupiti stranicama zato što postoje zaštite neovlaštenog pristupa. Svaka funkcija za testiranje elemenata ima intuitivno ime kako bi se znalo ako dođe do pogreške koja stranica ne radi. To se može i vidjeti na slici 5.4., gdje na primjer funkcija „testRacuniPageNameTable()“ koja posjećuje stranicu računi i provjera postojanje glavnih elemenata, kao što je dugme za dodavanje novih računa, tablice računa i jedno od polja na obje tablice računa. Posjećivanje stranice se obavlja pozivanjem metode „visit()“ koja prima naziv stranice, dok se postojanje provjera obavlja pozivanjem „see()“ metode s argumentom naziva elementa. Obje metode su unaprijed definirane u Laravel programskom okviru. Funkcija „realUser()“ pronalazi jednog od postojećih korisnika u bazi podataka. Tog korisnika svaka funkcija može koristiti samo pozivanjem funkcije „realUser()“, kao što se može vidjeti u funkciji „testRacuniPageNameTable()“ gdje se prvo poziva funkcija za pronalazak korisnika te se zatim posjećuje stranica i provjera postojanje elemenata.

```
public function testRacuniPageNameTable() {
      $this->realUser();
      $this->visit('racuni')
           ->see('Izlazni')->see('Dodaj')->see('ulazni')->see('Izdavatelj');
   }

public function realUser() {
      $user = \App\User::first();
      $this->be($user);
   }
```

Sl 5.4. Funkcija za testiranje stranice računa i pronalaska korisnika

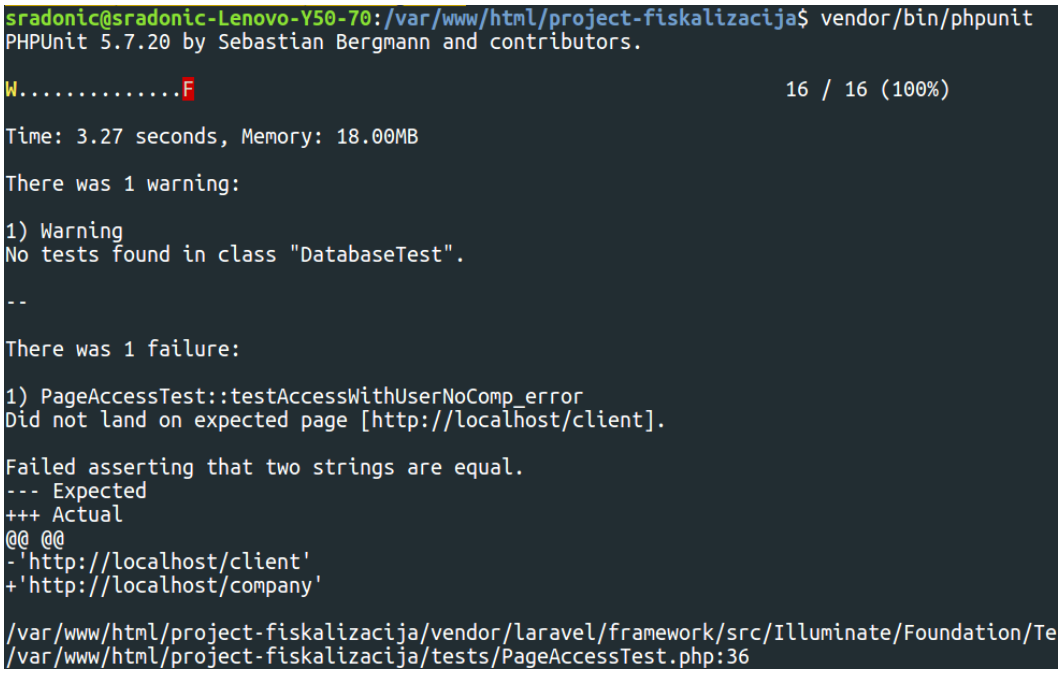
Provjeravanje linkova se izvršava u testu „LinkTest“ i u njemu se provjerava ispravnost linkova na sve najbitnije stranica web aplikacije za fiskalizaciju, a to su stranice za račune, klijente

i usluge. Prilikom testiranja ispravnosti i dostupnosti tih stranica se u isto vrijeme i testirala dostupnost stranica za stvaranje novih klijenata ili usluga. Za odabir, odnosno praćenje linka se koristi Laravel metoda „click()“ u koju se upisuje naziv elementa na kojem želimo simulirati klik miša te je najčešće element s „href“ oznakom. Za provjeru dolaska na željenu stranicu se koristi metoda „seePageIs()“ u koje se upisuje željena odredišna stranica koja se želi dobiti nakon praćenja linka koji je definiran iznad te metode. Primjer funkcije za provjeru ispravnosti linkova se može vidjeti na slici 5.5., naziva „testClientPageLink()“ u kojoj se provjera dolazak na stranicu klijenti s početne „home“ stranice i linka za stvaranje novog klijenta.

```
public function testClientPageLink() {
      $this->realUser();
      $this->visit('home')
           ->click('Klijenti')
           ->seePageIs('/client')
           ->click('Stvorite novog')
           ->see('Stvorite novog klijenta')
           ->seePageIs('/client/create');
  }
```

Sl 5.5. Funkcija za testiranje linkova

Nakon testiranja osnova kao što su postojanje stranice i ispravnost najvažnijih linkova za rad aplikacije, slijedi testiranje pristupa i sigurnosti aplikacije. Za testiranje pristupa i samim time sigurnosti se koristi test „PageAccessTest“. Ovaj test je jedan od najvažnijih testova u aplikaciji i sastoji se od četiri funkcije koje provjeravaju sve moguće načine pristupa. Prilikom ovog testiranja se željelo doći do pogrešaka kako bi se provjerila sigurnost aplikacije te zato dvije funkcije daju pogreške prilikom pokretanja (Slika 5.6.). To su funkcije „testAccessWithNoUser_error()“ i „testAccessWithUserNoComp _error()“. Prva od prethodno navedenih funkcija provjerava pristup stranici računa ako korisnik nije prijavljen. Ta funkcija vraća grešku (Slika 5.6.) jer korisnik koji pristupa toj stranici neće vidjeti stranicu računa, već će biti prebačen na stranicu za prijavu. Funkcija „testAccessWithUserNoComp_error“ provjera pristup korisnika koji ima stvoren profil, ali nije dodao kompaniju. Ta funkcija prvo provjerava pristup stranici za stvaranje kompanije, jer toj stranici jedino on ima pristup te joj može pristupiti. Zatim pristupa stranici klijenata kojoj nema pristup i funkcija vraća grešku (Slika 5.6.) jer navedeni korisnik koji pristupa stranici klijenata neće vidjeti stranicu klijenti, nego će biti prebačen na stranicu za stvaranje kompanije. Tim se dvjema funkcijama dokazala sigurnost web aplikacije, jer kao što rezultati prikazuju, ako korisnik nema profil i dodanu kompaniju, ne može pristupiti glavnom dijelu aplikacije.



Sl 5.6. Primjer pogreške prilikom testiranja pristupa

Nadalje, funkcijom „testAccessWithUserComp()“ obavljeno je testiranje u kojem registrirani korisnik koji ima dodanu kompaniju pristupa stranici računa. Testiranje s već registriranim korisnikom obavlja metodom „actingAs()“ u koju se upisuje korisnik kojeg smo dobili iz baze podataka. Takav korisnik može pristupiti bilo kojem dijelu aplikacije i to je dokazano ovim testom. Zadnji test ovdje je testiranje kada se ukloni međusloj (engl. *middleware*). Taj test je obavila funkcija „testAccessWithUserNoComp_NOerror()“. Navedena funkcija uklanja međusloj metodom „withoutMiddleware()“. Uklanjanjem međusloja se miču sva pravila za pristup koja su se prije testirala i bilo tko može pristupiti bilo kojem dijelu aplikacije jer ju više ništa ne štitit. Uklanjanjem međusloja korisnik koji nema dodanu kompaniju može sada bez problema pristupi stranici klijenti, što je prije davalo grešku u testu funkcijom „testAccessWithUserNoComp_error“ i vraćalo korisnika na stranicu za stvaranje kompanije. Takvo testiranje bez međusloja je i odrađeno u već spomenutoj funkciji „testAccessWithUserNoComp_NOerror()“ gdje korisnik koji nema dodanu kompaniju može pristupiti stranici klijenti jer nema međusloja (Slika 5.7.). Ta funkcija ne vraća grešku i time dokazuje potrebu međusloja, jer bez zaštite međusloja aplikacije nije sigurna i bilo tko može raditi što želi u aplikaciji.

```
public function testAccessWithUserNoComp_error() {
   $user = \App\User::where('id', '3')->first();
   $this->actingAs($user)->visit('company')->seePageIs('/company')->visit('client')
          ->seePageIs('/client');
}
public function testAccessWithUserNoComp_NOerror() {
    $this->withoutMiddleware();
    $user = \App\User::where('id', '3')->first();
    $this->actingAs($user)->visit('company')->seePageIs('/company')->visit('client')
          ->seePageIs('/client');
}
```

Sl 5.7. Razlika funkcija za testiranje pristupa stranici klijenata s greškom i bez greške

Forme su jako važan dio svih web stranice koje imaju bazu podataka i korisničke podatke, već i samo registriranje korisnika je forma. Iz tog razloga je jako važno da forme ispravno rade, inače nema korisničkih podataka u bazi podataka. Za provjeru ispravnog rada formi koristio se test naziva „FormTest“. Unutar tog testa se isprobavao rad formi korištenjem lažnih (engl. *mock*) korisnika i stvarnih korisnika gdje se rezultat zapravo i mogao vidjeti u bazi podataka ili na web stranici. Za testiranje sa stvarnim korisnicima se koristila već unaprijed spomenuta funkcija „realUser()“ koja vraća korisnika iz baze podataka, dok se za testiranje s lažnim korisnicima koristila funkcija „mockUser()“. Funkcija „mockUser()“ stvara lažnog korisnika koji oponaša stvarnog korisnika, ali zato ne puni bazu podataka nego samo testira rad. To je jako korisno kad se provodi više testova kako ne bismo napunili bazu podataka nepotrebnim podacima ili kada se testovi moraju iz nekog razloga provoditi na „live“ verziji stranice. Stvaranje lažnog korisnika se obavlja tako da se poziva instanca modela korisnika „User“ i pridodaju se potrebni atributi poput adrese e-pošte, korisničkog imena i tako dalje. Kako bi se došlo do lažnog korisnika, samo se poziva funkcija „mockUser()“, isto kao što se pozivala funkcija „realUser()“. Primjer testiranja formi se može vidjeti na slici 5.8., funkcija „testMockUserCreateService()“ korisni lažnog korisnika za testiranje dodavanja usluge. Za popunjavanje formi korisni se metoda „type('podatak', 'naziv_polja')“ u koju se upisuje podatak koji želimo spremiti i naziv polja u formi, a za pokretanje forme se koristi metoda „press()“ u koju se samo predaje naziv dugmeta za predaju. Na ovaj način je lako testirati unos tipa podatka koji ne odgovara. Na primjer za cijenu usluge se napiše tekst te se tako može testirati sigurnost i ispravnost formi. Primjer na slici 5.8. ima sve ispravne podatke te neće davati grešku, već će test proći. Odradilo se i testiranje s pogrešnim podacima, na način koji je već opisan to jest samo promjenom podatka u metodi „type()“. Međutim, testiranje rada formi se nije odradilo samo na lažnim korisnicima nego i na stvarnim korisnicima. Primjer toga je funkcija „testRealUserCreateClient()“ gdje se koristila funkcija

„realUser()“ koja označuje korisnika učitanog iz baze podataka i na njemu se odradilo spremanje klijenta na isti način na koji se odradilo spremanje za lažnog klijenta. Taj test osim što se može vidjeti da je prošao bez greške u terminalu, se može provjeriti i u bazi podataka gdje bi taj isti zapis trebao stajati.

```
public function testMockUserCreateClient() {
    $this->mockUser();
    $_SERVER['HTTP_HOST'] = 'localhost:8000';
    $this->visit('/client/create')->see('Stvorite novog')
         ->type('Klijent Test', 'kupac_ime')
         ->type('12312312', 'oib')
         ->type('Testna adresa 1x', 'adresa')
         ->type('Gotovina', 'pref_placanje')
         ->type('Neka testna napomena.', 'napomena')
         ->press('Stvorite Klijenta!')
         ->seePageIs('/client') ->see('Klijent');
}
public function mockUser() {
    $user = new \App\User([
          'email'     => 'test@mail.com',
          'kor_ime'  => 'tester'
         ]);
   $user->setAttribute('id', 2);
  //jer je user s id-om 2 povezan s komp. pa se ne mora praviti nova
   $this->be($user);
   }
```

Sl 5.8. Funkcija stvaranja lažnog korisnika i testiranja dodavanja klijenta pomoću istog

Testiranje HTTP zahtjeva je slično kao i testiranje formi, učita se stvarni ili lažni korisnik te se poziva metoda „call()“. Unutar metode „call()“ se upisuje način komuniciranja, odnosno HTTP zahtjev, zatim ruta na kojoj pozivamo taj zahtjev i nakraju ako je potrebno za zahtjev, niz podataka koji želimo proslijediti. Testiranje se odradilo na dvije najvažnije metode, a to su: POST i GET. Za testiranje GET zahtjeva u metodi „call()“ se samo prosljeđuje GET i ruta koju želimo posjetiti, a za provjeru ispravnosti zahtjeva koristimo metodu „assertResponseOk()“. Testiranje POST zahtjeva ide slično kao i testiranje GET zahtjeva, ali u metodu „call()“ se sad prosljeđuje POST, ruta i još dodatno niz podataka. Kako POST zahtjev ne bi vraćao grešku, iako je bio uspješan, ne smije se koristit metoda „assertResponseOk ()“ jer će ona provjeravati je li odziv dobar za rutu navedenu u „call()“ metodi, a zapravo je potrebna ruta na koju će POST zahtjev preusmjeriti nakon uspješno obavljenog zahtjeva. Iz tog razloga se umjesto „assertResponseOk()“ metode, koristi „assertRedirectTo()“ metoda koja će provjeravati preusmjeravanje POST metode nakon uspješnog obavljenog zahtjeva. Ako POST zahtjev vrati grešku, najbolje je grešku u

cijelosti provjeriti u Laravel izvještaju „laravel.log“ koji pokazuje detalje pogreške za lakše ispravljanje, nego u terminalu koji će samo pokazati da greška postoji, ali ne i kako je došlo do nje.

```
public function testGET() {
    $this->realUser();
    $response = $this->call('GET', '/home');
    $this->assertResponseOk();
}
public function testPOST() {
   $this->realUser();
   $response = $this->call('POST', '/service', ['usluga_opis' => 'Usluga 98', 'usluga_cijena' => '10',
'usluga_kolicina' => '10', 'usluga_jed_mjere' => '3', 'usluga_porez_stopa' => '10', 'usluga_popust'
=> '2']);
    $this->assertRedirectedTo('/service');
}
```

Sl. 5.9. Testiranje POST i GET zahtjeva

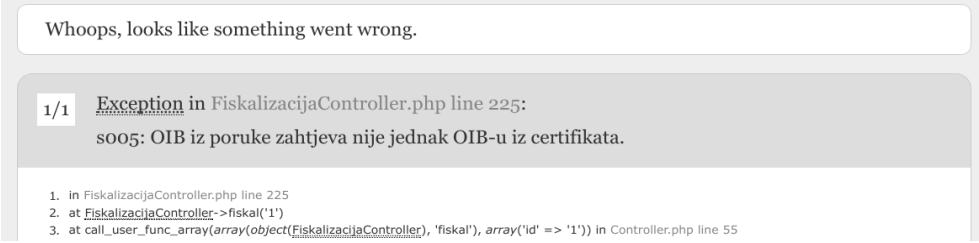
U testu za testiranju modela nije obavljeno samo testiranje modela već i testiranje baze podataka. Testiranje baze podataka je odrađeno na način da se provjeri postoji li željena tablica u bazi podataka te postoji li odgovarajući zapis. S time se zapravo provjerila veza s bazom podataka, provjerilo postojanje tablica i njihov rad. Metoda koja odrađuje i testira sve to navedeno je „seeInDatabase()“. Ta metoda prima naziv tablice i element za koji želimo provjeriti postoji li kao zapis u toj tablici. Testiranje baze podataka je određeno u istom testu kao i testiranje modela jer se prilikom testiranja svakog od modela određivao unos podataka u tablice koje model predstavlja i zatim se metodom „seeInDatabase()“ odmah i dodatno provjeravalo je li taj podatak postoji kao zapis u tablici. Testiranje modela se obavljalo na način da se u funkciju učita određeni željeni model kao što je na primjer model za korisnike „Users“ i primjer te funkcije je onda „testModelUser()“. Unutar te funkcije se osim deklariranja modela na kojem želimo raditi, napravi i unos svih potrebnih polja za model i zatim se pokrene metoda za spremanje „save()“ i provjera se metodom za testiranje naziva „assertTrue()“. Metoda „assertTrue()“ pretpostavlja da će metoda koja se poziva unutar nje vraćati istinu „true“, kao što se može vidjeti na slici 5.10. Testirao se svaki model koji nije samo „many-to-many“ i s time se svaki put testirala veza s bazom podataka i željenom tablicom. Ovaj test je mogao vraćati dvije greške, jednu ili niti jednu, s time da dvije greške znače da nije uspio spremiti podatke niti uspostaviti konekciju.

```php
public function testModalUser() {
  $user = new \App\User;
  $user->kor_ime = "ImeT";
  $user->kor_prezime = "PrezimeT";
  $user->username = "testington";
  $user->kor_telefon = "098111909922";
  $user->email  = "test@email.com";
  $user->password  = "123456";
  $this->assertTrue($user->save());
  $this->seeInDatabase('users', ['username' => 'testington']);
}
```

Sl 5.10. Testiranje modela korisnika i baze podatka

5.2. Testiranje fiskalizacije

Komunikacijom sa CIS-om Porezne uprave dobije se poruka odgovora koja može sadržavati JIR ili može sadržavati kod greške. To ovisi o tome je li CIS-u poslan ispravan potpisani XML ili nije. FINA ja razradila šifrarnik grešaka koji je vodilja za testiranje komunikacije sa CIS-om jer točno označava zbog čega je greška nastala. Cijeli popis svih grešaka se može naći u tehničkoj dokumentaciji za fiskalizaciju FINA-e. Testiranje komunikacije se izvodilo tako da se slao veliki broj različitih XML-ova CIS serveru od kojih je veliki broj namjerno bio neispravan kako bismo natjerali njihov server da nam pošalje poruku greške i možda ga naveli na pogrešku. Slika 5.11. prikazuje izgled odgovora CIS servera kad se pojavi greška. Prilikom testiranja se jednom naveo njihov server na pogrešku, što znači da je je skoro uvijek javljao točnu grešku.

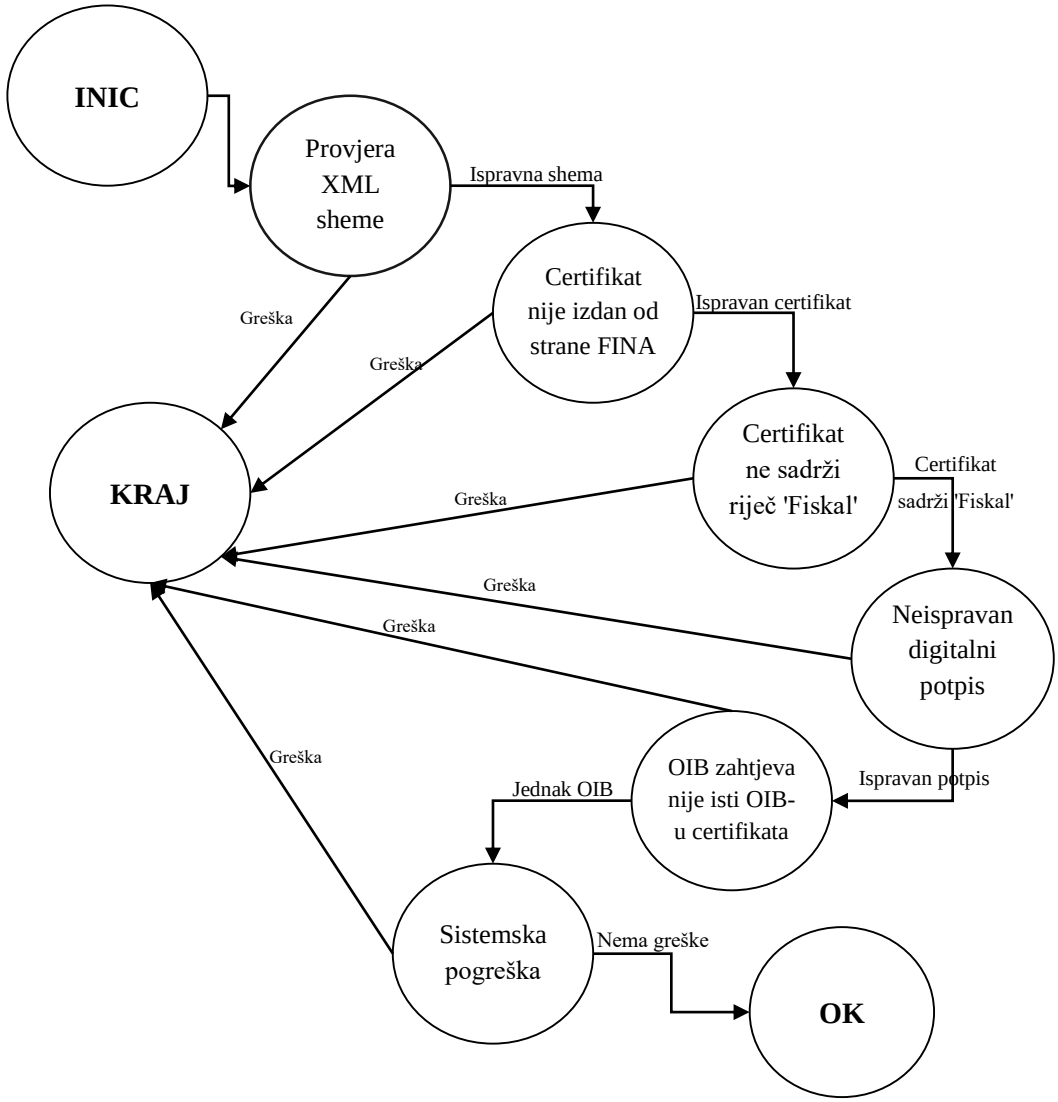


Slika 5.11. Primjer CIS odgovora u slučaju greške

Prvo se testiralo šest glavnih tipova grešaka koje su sljedeće:

- Poruka nije u skladu s XML shemom
- Certifikat nije izdan od strane FINA-e
- Certifikat ne sadrži naziv 'Fiskal'
- Neispravan digitalni potpis
- OIB iz poruke zahtjeva nije jednak OIB-u iz certifikata
- Sistemska pogreška

Prva greška naziva „poruka nije u skladu s XML shemom“ se sastoji od više slučajeva od kojih je svaki detaljno objašnjen i razrađen u dodatku tehničke dokumentacije FINA-e. Tih dodanih slučajeva ima preko četrdeset i nisu se svi testirali već su se odabrali najvažniji i najrazličitiji testni slučajevi. Na temelju izvođenja testnih slučajeva koji izazivaju grešku se napravio automat naziva FSM (engl. *finate state machine*). FSM je zapravo model koji se pravi na osnovi zadanih zahtjeva i specifikacija, u ovom slučaju su to testni slučajevi. Čvorovi predstavljaju stanje programa, a strelice predstavljaju operacije koje vrše prijelaz iz jednog u drugo stanje. Na slici 5.12. se može vidjeti FSM model koji prolazi kroz sve testne slučajeve te dolazi odmah na kraj ako je slučajno došlo do greške.



Sl 5.12. FSM Model

Prilikom testiranja CIS sustava se provjeravalo i više grešaka odjednom da se vidi prvenstvo među njima. Na primjer, ako je poruka zahtjeva imala neispravan digitalni potpis i OIB koji nije jednak onom iz certifikata, CIS bi vratio grešku neispravnog digitalnog potpisa zbog prvenstva neispravnog digitalnog potpisa. Popis grešaka koji je napisan predstavlja prvenstvo grešaka, odnosno ako se prva greška pojavi, sljedeću neće provjeravati i tako do kraja.

Tablica 5.13. prikazuje specifikacije testnih slučajeva. Specifikacija testnih slučajeva se sastoji od svih mogućih grešaka koje se mogu pojaviti i koji je razlog njihovog pojavljivanja. Do razloga pojavljivanja je došlo testiranjem različitih neispravnih situacija i analizom kako se CIS sustav ponaša, točnije koju će grešku javiti. Tako da na primjer za grešku sistemska pogreška se došlo samo ako se upiše datum koji ne postoji kao što je „28.14.2017“ te je u tablici pod slučajem sistemska pogreška kao razlog pojavljivanja postavljen samo nepostojeći datum i primjer koji se koristio da bi to toga došlo.

1. Provjera XML Sheme
a) Broj račun nije dozvoljene vrijednosti
b) Broj računa jednak 0
c) Porezna stopa nije iz dozvoljenog skupa vrijednosti
d) Osnovica PDV-a je veća od ukupnog iznosa
2. Certifikat nije izdan od strane FINA-e
3. Certifikat ne sadrži riječ 'fiskal'
4. Neispravan digitalni potpis
5. OIB iz poruke zahtjeva nije jednak OIB-u iz certifikata
6. Sistemska pogreška

Tab 5.13. Specifikacija testnih slučajeva

Dokumentiranje testiranja se obavilo na način da se za svaki razlog pojavljivanja greške svakog testnog slučaja napravio test u aplikaciji i pratio se odgovor CIS servera. U praksi, dokumentiranje testiranja se koristi kako bi se provjerilo postoji li slučaj kad je odgovor potvrdan, ali ne bi trebao biti jer je zahtjev bio netočan. Budući da se ovdje radi o već unaprijed testiranom udaljenom serveru na kojeg se ne može utjecati, rezultati dokumentacije će samo dokazivati ispravan rad servera. U slučaju da postoji koja greška dokumentirat će se samo kao primjer na koji se ne može direktno utjecati. Tablica 5.14. prikazuje dokumentiranje testiranja prema već utvrđenoj specifikaciji testnih slučajeva.

Testiranje 1			
Korišteni testni slučajevi			
R.br.	Testni slučaj	Specifikacija testnog slučaja	Kod greške
1.	broj_racuna = '322231321a“	Broj račun nije dozvoljene vrijednosti	V106
2.	broj_racuna = '0'	Broj računa ima vrijednost 0	V100 (sve ok)
3.	pdv = '322'	Porezna stopa nije iz dozvoljenog skupa vrijednosti	V110
4.	osnovica_pdv = '100' ukupan_iznos = '0'	Osnovica PDV-a je veća od ukupnog iznosa	V113
5.	Pogrešni certifikat učitan	Certifikat nije izdan od strane FINA-e	S002
6.	Certifikat s maknutim 'fiskal'	Certifikat ne sadrži riječ 'fiskal'	S003
7.	Pogreška XML potpisa	Neispravan digitalni potpis	S004
8.	Pogrešni OIB unesen	OIB iz poruke zahtjeva nije jednak OIB-u iz certifikata	S005
9.	datum_izrade = ’12.13.2019'	Sistemka pogreška	S006

Tab 5.14. Dokumentiranje testiranja

Kao što se vidi, testiranje je skoro uvijek bilo uspješno te se pojavila samo jedna greška koja se pojavila zbog odrađivanja testiranja u testnoj okolini, za koju FINA tvrdi da nije potpuno ispravna te da treba paziti jer to neće proći u produkcijskoj okolini.

6. ZAKLJUČAK

Internet aplikacija za fiskalizaciju je napravljena za korisnike koje imaju bilo kakvu vrstu poduzeća ili su privatni obrtnici koji moraju izdavati fiskalizirane račune. Aplikacije korisnicima nudi evidenciju poslovanja kompanije, dodavanje predloška usluga i stalnih klijenata i najvažnije stvaranja računa. Svaki račun koji se napravi unutar aplikacije se može fiskalizirati i izdati u PDF obliku. Također, aplikacija nudi svim korisnicima pregled izdanih računa radi evidencije. Uporaba aplikacije u službene svrhe zahtjeva od korisnika da ima ispravne certifikate izdane od strane FINA-e. Ti certifikati se trebaju samo postaviti u aplikaciji s ispravnim nazivom i aplikacija će ih moći koristiti i fiskalizirati račune.

Za izradu ove aplikacije je korišten Laravel programski okvir. Značaj programskih okvira proizlazi iz toga da potiču poštivanja pravila razvoja i olakšavaju pisanje koda, ako ga se zna koristiti. To znači da se korištenjem programskog okvira ne trebaju pisati dijelovi koda, to jest funkcije, koje su standardne, one su već implementirane u Laravel-u. Da zaključim, Laravel programski okvir pojednostavljuje rutinske zadatke pisanja koda i iz tog razlog je preporučen ljudima koji su iskusni s PHP jezikom i objektno orijentiranim programiranje.

Ovaj rad daje najvažnije informacije o konceptima Laravel programskog okvira i kako ih koristiti. Ti koncepti su: stvaranje nove aplikacije, autentikacija, rad s kontrolerima, rutama, modelima, pogledima, Blade-om, Eloquent-om i Artisan-om. Također, unutar rada je opisana web aplikacije za fiskalizaciju, od kojih se stranica sastoji te na koji je način svaka implementirana i sve njene funkcionalnosti. Najvažnije, opisano je kako napraviti sustav za fiskalizaciju unutar PHP programskog jezika.

Još jedan ključan aspekt rada je bilo testiranje aplikacije. Odrađeno je testiranje rada i ponašanja aplikacije i testiranje komunikacije s CIS sustavom za fiskalizaciju. Svako testiranje koja se odradilo unutar aplikacije je detaljno opisano u radu, tako da ih svaka osoba može pokrenuti. Osim toga, za svako testiranje su dani i pojašnjeni rezultati. S testiranjem je potvrđeno da aplikacije radi ispravno i da je komunikacija s CIS sustavom za fiskalizaciju u redu.

LITERATURA

[1] The Advantages of LAMP as a Web Development Platform, http://blog.koenig-solutions.com/2015/06/23/advantages-of-lamp-as-a-web-development-platform/, pristupljeno 02.07.2017

[2] HTML, https://developer.mozilla.org/en-US/docs/Web/HTML, pristupljeno 02.07.2017.

[3] CSS, https://developer.mozilla.org/en-US/docs/Web/CSS, pristupljeno 02.07.2017.

[4] Anirudh Prabhu, Beginning CSS Preprocessors With Sass, Compass, and Less, 2015

[5] SaSS, http://bkaprt.com/sass/4/, pristupljeno 02.07.2017.

[6] About JavaScript, https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript, pristupljeno 02.07.2017.

[7] Bear Bibeault, Yehuda Katz, Aurelio De Rosa, jQuery in Action, 3rd Edition, 2015

[8] David Sawyer McFarland, JavaScript & jQuery the Missing Manual, 3rd Edition, 2014

[9] Intro What is Php, http://php.net/manual/en/intro-whatis.php, pristupljeno 02.07.2017.

[10] Rob Aley, PHP Beyond the Web – 1st Edition, 2016

[11] Robin Nixon Learning PHP, MySQL, JavaScript, CSS & HTML5, 2014

[12] PHP 7 5 Things to know infographic, http://www.zend.com/en/resources/php7-5-things-to-know-infographic, pristupljeno 05.07.2017.

[13] Gregory Blade, Laravel Basic, creating web apps its simple, 2016

[14] Arda Kılıçdağı, H. İbrahim Yilmaz, Laravel Design Patterns and Best Practices, 2014

[15] Fiskalizacije – Tehnička specifikacija za korisnike Verzija 1.5

[16] Canonical XML, https://www.ibm.com/developerworks/xml/standards/x-c14nspec.html, pristupljeno 14.08.2017

[17] Software Testing Life Cycle STLC, http://www.softwaretestingclass.com/software-testing-life-cycle-stlc/, pristupljeno 14.08.2017.

[18] Web Application Testing, https://www.tutorialspoint.com/software_testing_dictionary/web_application_testing.htm, pristupljeno 14.08.2017.

[19] Laravel, https://github.com/laravel/laravel, pristupljeno 15.08.2017.

[20] Database: Migrations, https://laravel.com/docs/5.3/migrations, pristupljeno 15.08.2017.

[22] Eloquent: Getting Started, https://laravel.com/docs/5.3/eloquent, pristupljeno 16.08.2017.

[22] Relationships, https://laravel.com/docs/5.4/eloquent-relationships, pristupljeno 16.08.2017.

[23] Routing, https://laravel.com/docs/5.3/routing, pristupljeno 16.08.2017.

[24] Controllers, https://laravel.com/docs/5.3/controllers, pristupljeno 17.08.2017.

[25] Middleware, https://laravel.com/docs/5.3/middleware, pristupljeno 17.08.2017.

[26] Requests, https://laravel.com/docs/5.4/requests, pristupljeno 17.08.2017.

[27] Authentication, https://laravel.com/docs/5.3/authentication, pristupljeno 17.08.2017.

[28] Views, https://laravel.com/docs/5.3/views, pristupljeno 17.08.2017.

[29] Blade Templates, https://laravel.com/docs/5.3/blade, pristupljeno 17.08.2017.

[30] Testing, https://laravel.com/docs/5.3/testing, pristupljeno 14.08.2017.

[31] cURL, https://curl.haxx.se/, pristupljeno 15.08.2017.

[32] PHP, http://php.net/, pristupljeno 17.08.2017.

SAŽETAK

Internet aplikacija za fiskalizaciju je aplikacija koja korisnicima nudi evidenciju poslovanja kompanije, stvaranja računa i dodavanja predloška usluga i stalnih klijenata. Svaki račun koji se napravi unutar aplikacije se može fiskalizirati i izdati u PDF obliku. Aplikacije je razvijena u programski jezicima Javascript i PHP. Za razvoj je još korišten programski okvir Laravel. Aplikacija je testirana korištenjem Selenium IDE alata i PHPUnit testova unutar Laravel programskog okvira. U radu su opisani osnovni koncepti Laravel programskog okvira i ostalih korištenih alata. Uz to je još opisana funkcionalnost, mogućnosti i izgled aplikacije. Kraj rada opisuje proces fiskalizacije i provedene testove na aplikaciji.

Ključne riječi: Laravel, fiskalizacija, testiranje, aplikacija, programski okvir, račun

ABSTRACT

Title: Design and testing of a web solution for fiscalized bills issuing

Web application for fiscalization is an application that offers users full track of company's business, bill creation and adding templates for services and clients. Each bill made within this application can be fiscalized and downloaded in PDF form. The application is developed in Javascript and PHP programming languages. Laravel web framework is also used for this application's development. The application is tested using Selenium IDE tool and PHPUnit tests which are a part of Laravel web framework. Inside the thesis the basic concepts of Laravel web framework are described and also other tools which were used for development. In addition, funcionality, appearance are described in the thesis. At the end of the thesis the process of fiscalization and conducted tests inside the application are also described.

Keywords: Laravel, fiscalization, testing, application, web framework, bill

ŽIVOTOPIS

Stjepan Radonić rođen je 9. prosinca 1993 u Osijeku. Od rođenja živi u Osijeku, gdje stječe osnovnoškolsko obrazovanje u OŠ Franje Krežma od 2000. do 2008 godine. 2008.godine upisuje II. Jezičnu gimnazija u Osijeku te većinu razreda prolazi s vrlo dobrim uspjehom. Nakon završetka srednjoškolskog obrazovanja 2012. godine upisuje „Elektrotehnički fakultet“, preddiplomski studij računarstva u Osijeku kojeg završava 2015. godine. 2016. godine upisuje diplomski studij smjer računarstvo, modul programsko inženjerstvo na istom fakultetu, promijenjenog naziva u „Fakultet elektrotehnike, računarstva i informacijskih tehnologija“, kojeg trenutno pohađa.

Vlastoručni potpis: _____

PRILOZI

Na CD-u:

a) Diplomski rad „Izrada i testiranje web rješenja za izdavanje fiskalnih računa“ u „.docx“ formatu.
b) Diplomski rad „Izrada i testiranje web rješenja za izdavanje fiskalnih računa“ u „.pdf“ formatu.
c) Izvorni kod web aplikacije za fiskalizaciju.

Internet:

a) Laravel službena dokumentacija: https://laravel.com/docs/5.3/
b) Fina službena dokumentacija: https://www.porezna-uprava.hr/HR_Fiskalizacija/Documents/Fiskalizacija%20-%20Tehnicka%20specifikacija%20za%20korisnike_v1%205.pdf