SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 754

SUSTAV ZA AUTOMATIZIRANO TESTIRANJE FISKALNIH BLAGAJNI

Tvrtko Ivasić

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 754

SUSTAV ZA AUTOMATIZIRANO TESTIRANJE FISKALNIH BLAGAJNI

Tvrtko Ivasić

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 11. ožujka 2022.

ZAVRŠNI ZADATAK br. 754

Pristupnik:	**Tvrtko Ivasić (0036525406)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	doc. dr. sc. Stjepan Groš

Zadatak: **Sustav za automatizirano testiranje fiskalnih blagajni**

Opis zadatka:

Kako bi se spriječilo crno tržište u RH uvedene su fiskalne blagajne čija zadaća je da svaki izdani račun prijave Poreznoj upravi. Međutim, fiskalne blagajne rade razni proizvođači programske podrške te se postavlja pitanje koliko su implementacije ispravne, odnosno, bez pogrešaka u kodu. Dodatno, zbog toga što je u proces prijave uključena kriptografija to znači da je vjerojatnost pogreške još veća budući da upotreba kriptografije, a posebno certifikata i digitalnog potpisa, nije toliko raširena. Zbog svega toga oportuno je razviti sustav koji bi omogućio ispitivanje ispravnosti implementacije fiskalnih blagajni. U sklopu ovog završnog rada potrebno je razviti proširiv sustav za ispitivanje ispravnost zahtjeva koje šalju fiskalne blagajne te manipulaciju odgovorima kako bi se ispitala ispravna reakcija fiskalnih blagajni na pogreške. Voditi računa o lakoći korištenja sustava, automatizaciji testiranja te mogućnosti istovremenog pristupa više korisnika. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 10. lipnja 2022.

Hvala svima od kojih sam nešto naučio tijekom preddiplomskog studija.

SADRŽAJ

1. Uvod

S 1. siječnjem 2013. godine u Republici Hrvatskoj uvedena je mjera fiskalizacije. Fiskalizaciju provode obveznici fiskalizacije, a glavni cilj je sprečavanje crnog tržišta. Svaka transakcija se prijavljuje poreznoj upravi tako da blagajna šalje zahtjev za izdavanje fiskalnog računa, a porezna uprava odgovara slanjem odgovora za fiskalni račun. Kako bi blagajna sudjelovala u procesu komunikacije s Poreznom upravom ona mora biti implementirana prema Fininoj specifikaciji. U toj specifikaciji propisano je korištenje certifikata, digitalnog potpisa i kriptografije, a pritom na tržištu ima mnoštvo blagajni raznih proizvođača. Zbog svega navedenog velike su mogućnosti da u njihovoj implementaciji ima grešaka. Stoga je cilj ovog rada opisati i objasniti princip rada sustava koji je razvijen kako bi se omogućilo testiranje implementacije fiskalne blagajne s ciljem smanjenja grešaka u njihovoj implementaciji.

U okviru ovog rada izrađen je sustav za automatizirano testiranje fiskalnih blagajni. Sustav se sastoji od web aplikacije koje omogućava registraciju korisnika, dodavanje fiskalnih blagajni za testiranje i pregledavanjerezultata testiranja. Drugi značajni dio sustava čini poslužitelj koji komunicira s blagajnom i testira njezinu ispravnost. Ta dva dijela sustav povezana su na istu bazu podataka. Na taj način omogučeno je da su podaci u sustavu sinkronizirani.

Rad je strukturiran na sljedeći način. U prvom poglavlju se obrađuju tehničke specifikacije fiskalizacije. Opisan je općeniti postupak fiskalizacije i način na koji su strukturirani zahtjevi i odgovori. U istom poglavlju također se opisuju načini na koje funkcioniraju certifikati i digitalni potpis. Drugo poglavlje rada "Arhitektura i implementacija sustava" razdvojeno je u potpoglavlja prema dijelovima sustava osim zadnjeg koje se bavi Dockerom. Potpoglavlje Web aplikacije opisuje tehnologije koje su korištene za njenu izgradnju kao i način na koji su implementirane. Opisuju se React i Redux JavaScript knjižnice i Django web okvir. Uz navedene tehnologije i njihove načine korištenja u sustavu opisani su postupci zaštite web aplikacije verifikacijom

elektroničke pošte i JSON web tokenom. Drugo potpoglavlje bavi se arhitekturom i implementacijom poslužitelja, Tornado Python web okvirom i načinom na koji se provode testovi na poslužitelju. Također je dan i opis skripte koja pomaže u upravljanju testiranjem. U potpoglavlju koje se bavi bazom podataka opisana je struktura baze podataka i sadržaj bitnih relacija koje se u njoj nalaze. Zadnje poglavlje "Korištenje aplikacije" opisuje načine korištenja aplikacije i postupak testiranja fiskalne blagajne.

2. Tehnička implementacija fiskalizacije

U ovom poglavlju opisani su tehnički zahtjevi koje blagajna mora ispunjavati kako bi mogla sudjelovati u procesu fiskalizacije. Zahtjeve propisuje Porezna uprava, a detaljan opis može se naći u [1]. Komunikacija je u sustavu fiskalizacije ostvarena SOAP (Simple Object Access Protocol) protokolom. SOAP je baziran na XML-u, a koristi za razmjenu informacija preko nekog komunikacijskog kanala (ovdje konkretno HTTP). Poruke su formatirane kao XML dokumenti i može ih provoditi bilo koji program namijenjen radu s XML dokumentima. Ovaj sustav svoje testiranje bazira na manipulaciji XML odgovorima. Fiskalnoj blagajni vračati će se odgovori koji su pogrešni za njen zahtjev nakon čega će sustav pratiti daljnje akcije fiskalne blagajne.

Unutar specifikacije je propisano: "U svim slučajevima kad obveznik iz nekog razloga nije dobio JIR za izdani račun (prekid Internet veze, potpuni prestanak rada naplatnog uređaja, greška u poruci odgovora, privremena nedostupnost CIS-a) obveznik je dužan naknadno ponoviti slanje poruke. Tek kad obveznik dobije ispravnu poruku odgovora od CIS-a koja sadrži JIR može smatrati da je račun prijavljen Poreznoj upravi" [1]. Taj zahtjev iskorištava i ovaj sustav tako da blagajnu testira greškama sve dok u jednom trenutku blagajni ne vrati JIR u odgovoru čime se test smatra završenim. Ako se kasnije ponovi zahtjev za račun koji je već prijavljen sustavu, ta greška će također biti zabilježena, a prethodno uspješno završeni test prebacit će se u skupinu neispravnih.

2.1. Struktura zahtjeva

Kao što je već navedeno za bilježenje računa koristi se XML označni jezik. Generalna podjela strukture zahtjeva je na polja "Zaglavlje" i "Račun". Unutar polja "Zaglavlje" nalaze se datum i vrijeme slanja zahtjeva i jedinstveni ID poruke (UUID). Unutar polja "Račun" nalaze se detaljni o računu poput broja računa, podataka o iznosu poreza,

načinu plaćanja i mnogi drugi. Na sljedećem primjeru vidljiv je dio jednog zahtjeva za fiskalizaciju:

Isječak 2.1: Primjer zahtjeva za račun

```
<tns:RacunPDZahtjev>
<tns:Zaglavlje>
 <tns:IdPoruke>3541f671-da0a-44ae-bf3b-800d63e454b9</tns:IdPoruke>
 <tns:DatumVrijeme>24.01.2020T09:59:43</tns:DatumVrijeme>
 </tns:Zaglavlje>
<tns:Racun>
 <tns:Oib>98765432198</tns:Oib>
 <tns:USustPdv>true</tns:USustPdv>
 <tns:DatVrijeme>24.01.2020T09:59:43</tns:DatVrijeme>
 <tns:OznSlijed>P</tns:OznSlijed>
 <tns:BrRac>
 <tns:BrOznRac>33</tns:BrOznRac>
 <tns:OznPosPr>POSL1</tns:OznPosPr>
 <tns:OznNapUr>12</tns:OznNapUr>
 </tns:BrRac>
 ...
```

2.2. Struktura odgovora

Odgovor se također kao i račun zapisuje XML označnim jezikom. Unutar odgovora nalazi se polje "Zaglavlje" koje također ima identifikator poruke i datum i vrijeme slanja odgovora. Kao identifikator poruke uzima se identifikator iz polja "Zaglavlje" zahtjeva. Dalje se unutar odgovora nalazi ili polje "Greška" koje sadrži podatke o greškama zabilježenim unutar zahtjeva ili polje "JIR" koje znači da je račun uspješno fiskaliziran. Primjer strukture odgovora:

Isječak 2.2: Primjer strukture odgovora

```
<tns:RacunOdgovor
   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:Jir>2cf55235-9470-4b5c-a539-463f52b109d2</tns:Jir>
</tns:RacunOdgovor>
```

Propisane pogreške

Unutar službene dokumentacije procesa fiskalizacije definirano je šest službenih pogrešaka koje mogu biti vraćene u odgovoru prema fiskalnoj blagajni. Unutar ovog sustava odgovor koji se vraća blagajni također može sadržavati bilo koju od tih šest pogrešaka. Popis pogrešaka nalazi se u sljedećoj tablici:

Šifra greške	**Naziv greške**
s001	"Poruka nije u skladu s XML shemom : #element ili lista elemenata koji nisu ispravni po shemi#."
s002	"Certifikat nije izdan od strane FINA RDC CA ili je istekao ili je ukinut."
s003	"Certifikat ne sadrži naziv 'Fiskal"
s004	"Neispravan digitalni potpis."
s005	"OIB iz poruke zahtjeva nije jednak OIB-u iz certifikata."
s006	"Sistemska pogreška prilikom obrade zahtjeva."

Tablica 2.1: Šifrarnik grešaka

Za strukturu odgovora i zahtjeva važno je napomenuti da se svaki zahtjev i odgovor potpisuju. U pravom procesu fiskalizacije za to služe certifikati koje izdaje FINA koja ima ulogu tijela za ovjeru, a unutar ovog sustava riječ je o samopotpisnim certifikatima. Primjeri zahtjeva s potpisom mogu se naći na kraju dokumenta specifikacije [1].

2.3. Digitalni certifikat

Digitalni certifikat je vjerodajnica (potvrda) u elektroničkom obliku. Predstavlja digitalni identitet te omogućava sigurnu i povjerljivu komunikaciju [3]. Certifikat sadrži ključ (informacije o ključu), informaciju o vlasniku (subjektu) i ovjeru o njegovoj valjanosti koja se izdaje od strane izdavatelja. Posjedovanjem digitalnog certifikata jamči se autentičnost informacije koju će elektronički servis ili osoba primiti.

Unutar sustava fiskalizacije koristi se X.509 v3 oblik certifikata, a tijelo za ovjeru koje

upravlja certifikatima je Fina. Tijelo za ovjeru je slično javnom bilježniku. Ono izdaje digitalne certifikate, potpisuje certifikate da bi se potvrdila njihova valjanost te bilježi opozvane certifikate i certifikate čija je valjanost istekla. Certifikat se šalje unutar zahtjeva za fiskalizaciju. Njime korisnik fiskalizacije potpisuje XML zahtjev te tako potvrđuje svoj identitet. Primjer certifikata unutar zahtjeva za fiskalizaciju:

Isječak 2.3: Primjer generiranog samopotpisnog certifikata

```
-----BEGIN CERTIFICATE-----
MIIDozCCAosCFFdI+6FjrVGbNSlD8ntuRIbR+l97MA0GCSqGSIb3DQEBCwUAMIGN
MQswCQYDVQQGEwJIUjERMA8GA1UECAwIS2FybG92YWMxCzAJBgNVBAcMAktBMRow
GAYDVQQKDBFGaXNrYWxUZXN0ZXIgZGVtbzEMMAoGA1UECwwDZGV2MQ8wDQYDVQQD
DAZUdnJ0a28xIzAhBgkqhkiG9w0BCQEWFHR2cnRrby5pdmFzaWNAZmVyLmhyMB4X
DTIyMDYxMDEwMjQzMFoXDTIzMDYxMDEwMjQzMFowgY0xCzAJBgNVBAYTAkhSMREw
DwYDVQQIDAhLYXJsb3ZhYzELMAkGA1UEBwwCS0ExGjAYBgNVBAoMEUZpc2thbFRl
c3RlciBkZW1vMQwwCgYDVQQLDANkZXYxDzANBgNVBAMMBlR2cnRrbzEjMCEGCSqG
SIb3DQEJARYUdHZydGtvLml2YXNpY0BmZXIuaHIwggEiMA0GCSqGSIb3DQEBAQUA
A4IBDwAwggEKAoIBAQC/e9HfPPvQX5OlePgPahnJqiU7UY8pLb/+rOyCICCdUnyj
vkwayuAvl8bzz6GyM139bREUkEmTYUh8uIYw+99wm7srvWMi/8uX2UyZgToEGNew
+ERZyU0UuDNSlJED4tug0PCJO2S02xaB6ols1g+3ab+V40Z8NZf4mHd3E1U3E4Ll
CtTNhmXO1AZJ7Ay4PTN68ONIxNBT7aCJQgBP95TYvwViRJWM98+GqRtZ/05i+YFs
fTy+XXKSLy+tRJhML62Ur5c4/1GBd55TQWKIFpU2vLntRkzJZ8snFR/xzXt+W1Vg
PY02sFY9PO0Ve4kTyNl2Uf5vNyMhWqKU47URbHm/AgMBAAEwDQYJKoZIhvcNAQEL
BQADggEBALcBlaYlFvs1uoWcIB1NE/2fs8ACA/I8kacoUCXEC3ayuymmFdZLCLG9
uNFcqwG2McD6r/QTFHEwwylEEAseGGcZktO2wazFF9sEMJ8wAyHLhqytibaEQ3Se
3AI7C1CNOxot7r5cWhHy9qC9M0uNn2ASBX8HSugQf3bWb2815RXs9uRB6flpOPIm
hRR/Z7LEBKD7dB1XlkNl3PU5lfFFb+dIJe3TlCM1la+BZUBFjUGNMVVmG93ScI0o
TkdZEqEK73c/tj53nZzs4VRRt/8Kq1/hXBpbWdyWjYWZlUUVOT6zRrIMAAMfiaP7
N5Upd4aqXXNwA9tSMcGbf4ZAFjsz3+Y=
-----END CERTIFICATE-----
```

Na gornjem isječku (Isječak 2.3) vidljiv je generirani samopotpisni certifikat. Certifikat je generiran pomoću OpenSSL knjižnice. Samopotpisni certifikati imaju svoje mane, ali za potrebe ovog sustava zbog brzine i jednostavnosti izrade [] korišten je upravo takav tip certifikata. Pomoću nekih od mnogo javno dostupnih dekodera moguće je dobiti sadržaj X509 certifikata. Primjer dijela dekodiranog certifikata iz Isječka 2.3 dostupan je na slici u nastavku (Slika 2.1).

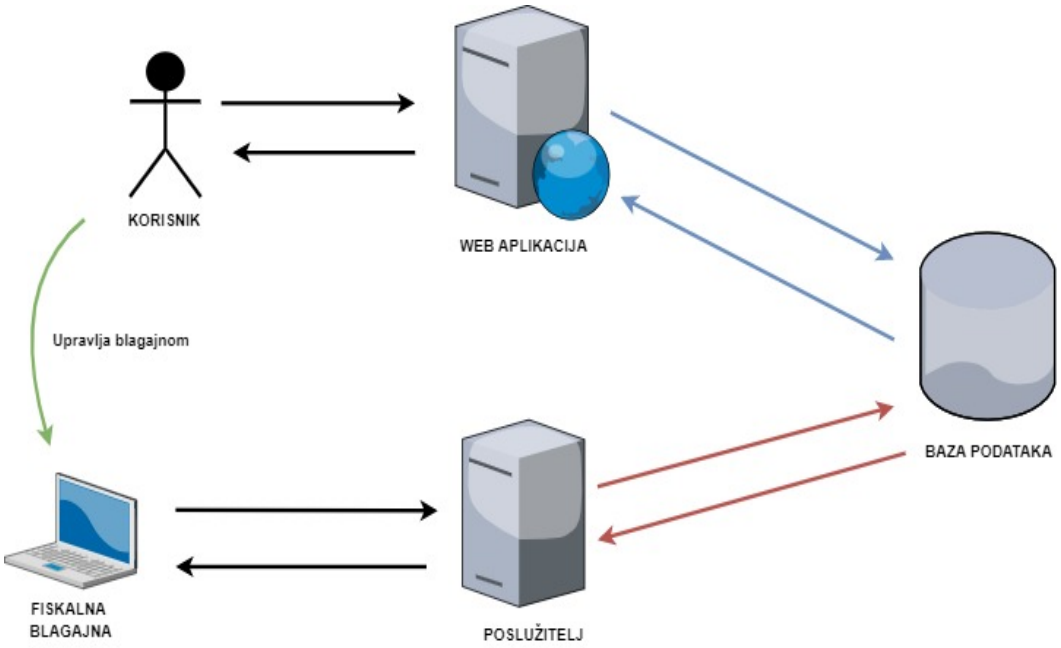
Slika 2.1: Primjer dijela dekodirano certifikata

2.4. Digitalni potpis

Digitalni potpis elektronička je i šifrirana oznaka valjanosti digitalnih podataka. On je digitalna zamjena za vlastoručni potpis na papiru te kao i tradicionalni potpis potvrđuje da podaci potječu od potpisnika i da nisu izmijenjeni. Digitalni potpisi bazirani su asimetričnoj kriptografiji, tj. kriptografiji javnog ključa. Privatni ključ predstavlja potpis, osigurava integritet i autentičnost, a njegovo je dijeljenje strogo zabranjeno kako se navedeni integritet i autentičnosti ne bi narušili. Javni ključ dijeli se s ostalim sudionicima komunikacije kako bi njime mogli dekriptirati podatke koji su vezani uz potpis. Ako primatelj ne može dekriptirati sadržaj znači da je ključ pogrešan ili se u komunikaciji pojavila neka pogreška.

3. Arhitektura i implementacija sustava

Sustav ja zamišljen i implementiran tako da se sastoji od tri dijela: Web aplikacije, poslužitelja za testiranje i zajedničke baze podataka. Unutar Web aplikacije korisnik se prvo mora registrirati. Nakon registracije korisniku su omogućene sve funkcionalnosti koje sustav pruža. Može uređivati i mijenjati podatke na svom profilu, preuzeti certifikat kojim se spaja na poslužitelj za testiranje, dodavati blagajne, stavljati ih na test i pregledavati rezultate testiranja. Nakon što je dodao svoju blagajnu u test korisnik pomoću nasumično generirane lozinke za spajanje istu može povezati s poslužiteljem za testiranje. Poslužitelj i web aplikacija koriste istu bazu podataka kako bi njihova međusobna interakcija bila lakša, a podaci unutar sustava bili sinkronizirani. Baza podataka sadrži sve podatke neophodne za rad sustava, podatke o testovima koji se pokreću na poslužitelju i rezultatima provedenih testova.



Slika 3.1: Apstraktni prikaz sustava

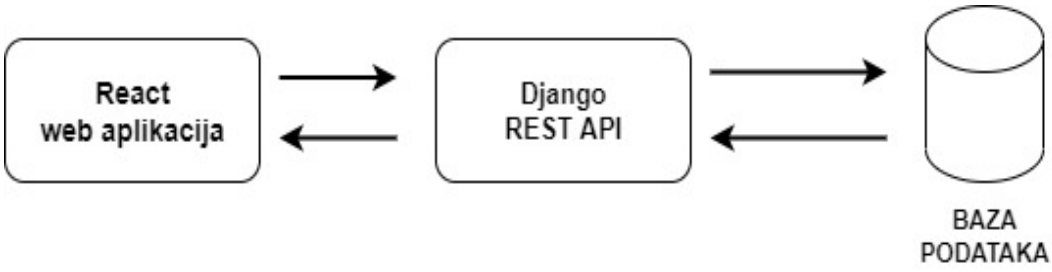
U daljnjim potpoglavljima dani su detaljni opisi pojedinačnih dijelova sustava kao i opisi tehnologija koje su korištene kako bi oni bili razvijeni. Tako poglavlje 3.1 govori o Web aplikaciji i načinu na koji je ista implementirana kao i specifičnostima tehnologija potrebnih za izgradnju iste. U poglavlju 3.2 opisan je poslužitelj za testiranje fiskalnih blagajni i njegove posebnosti. Nadalje poglavlje 3.3 govori o karakteristikama baze podataka, a dan je i opis njenih značajnih tablica. U zadnjem poglavlju 3.4 govori se o tehnologiji Docker koja je korištena kako bi se olakšalo i standardiziralo pokretanje sustava.

3.1. Web aplikacija

Web aplikacija je programsko rješenje kojem se pristupa putem interneta (internet mreže). Ovaj tip aplikacija podrazumijeva da se određeni resursi nalaze na udaljenom poslužitelju, kojem onda klijent pristupa kako bi dohvatio te podatke. Web aplikacija ulazna je točka sustava. Kao takva razvijena je prema REST arhitekturi. REST kao arhitektura nameće određena pravila u komunikaciji koja se međusobno nadograđuju jedno na drugo kako bi bila postignuta što brža i pouzdanija komunikacija, ali i kako bi se omogućila što veća skalabilnost []. Jedan ob bitnih REST principa je slojevitost. U njemu svaki sloj obavlja svoje zadaće neovisno o drugim slojevima. Sustav ove Web aplikacije može se podijeliti u tri sloja:

- React Web aplikacija
- Django REST API
- Baza podataka

Na slici (Slika 3.2) vidljiva je apstraktna prezentacija slojeva unutar web aplikacije. Kako je ista baza podataka korištena i od strane poslužitelja njen detaljniji opis bit će dan izvan ovog poglavlja u poglavlju 3.3.



Slika 3.2: Arhitektura sustava web aplikacije

3.1.1. Django

Pristupna točka za dohvaćanje i pohranu podataka web aplikacije razvijena je u Django okviru otvorenog koda. Django je baziran na programskom jeziku Python. Django slijedi i definira MTV (eng. *Model-Template-View*) arhitekturni obrazac [2]. Glavna karakteristika Djanga je da želi olakšati razvoj Web aplikacija tako da programer mora napisati što manje koda, a da pritom ne narušava sigurnost i skalabilnost aplikacija koje razvija [6]. Uz sami Django korišteni su i neke dodatne knjižnice.

Korištenje knjižnice Django REST framework (DRF) omogućilo je izradu mrežnog API-a koji je u skladu s REST načelima [10]. Dalje u ovom poglavlju na isječcima koda iz sustava biti će opisani neki bitni dijelovi Django projekta.

Modeli: modeli su klase koje unutar Django okvira predstavljaju tablice iz baze podataka. Prilikom kreiranje Django projekta generira se datoteka models.py unutar koje se dodaju modeli koji će se koristi u aplikaciji. Primjer jednog Django modela:

Isječak 3.1: Primjer modela testa

```
class Test(models.Model):
    name = models.CharField(max_length=100, null=False,
        blank=False)
    description = models.TextField(max_length=400, null=True,
        blank=True)

    def __str__(self) :
        return '%s: %s' % (self.name, self.description)
```

U isječku koda (Isječak 3.1) vidljiv je model "Testa". Taj model kao i svaki drugi nasljeđuje Python Django klasu models.Model kako bi Django znao njime ispravno baratati. Unutar klase postavljaju se atributi. Na ovom primjeru modelu su postavljena dva atributa jedan tipa *CharField* i drugi tip *TextField*. Svakom od tih atributa mogu se dalje postavljati razna ograničenja. U ovom primjeru postavljeno je ograničenje da objekti nisu *null* i ograničenje duljine znakove. Moguće je dodavati i razna druga ograničenja ili postavke poput postavljanja nekog atributa kao primarnog ključa. Na kraju vidljivo je nadjačavanje *toString* metode klase. U poglavlju 3.3 koji se odnosi na bazu podataka bit će više detalja o modelima u sustavu i opisa za što se koriste.

Pogledi: pogledi služe kako bi se u Djangu mogle implementirati REST API metode. Unutar sustava pogledi su raspisani u nekoliko datoteka radi preglednosti koda. Primjer jednog pogleda:

Isječak 3.2: Primjer pogleda za brisanje testa blagajne

```
@api_view(['DELETE'])
@permission_classes([IsAuthenticated])
def deleteTest(request, pk):
    try:
        user = request.user
        test = TestResult.objects.get(id=pk, user=user)
        test.delete()
        return Response('Test deleted!')
    except TestResult.DoesNotExist:
        message = {'detail': 'Test with this id does not
            exsists!'}
        return Response(message,
            status=status.HTTP_404_NOT_FOUND)
```

U primjeru (Isječak 3.2) vidi se pogled za brisanje testa. Pogled koristi anotacije knjižnice *Django REST framework*. One omogućavaju ograničavanje dostupnosti ovog pogleda pa je ovaj pogled dostupan samo autoriziranim korisnicima, a označavaju i da je ovo REST DELETE metoda koja omogućava brisanje nekog modela. Kao rezultat vraća se Response u JSON obliku s porukom o grešci ili uspješno obavljenoj operaciji, a uz poruku dolazi i njoj odgovarajući HTPP statusni kod.

URL putanje: URL putanjama ostvarena je izloženost pogleda prema klijentima. Unutar putanje spaja se lokacija pristupa nekoj metodi i pogled koji ju sadrži. Django projekt sadrži jednu glavnu urls.py datoteku. Unutar te datoteke nalazi se *urlpatterns* varijabla koja definira rute dostupne u Web aplikaciji. Unutar te varijable moguće je dodavati *uralpatterns* varijable iz drugih datoteka. Primjer jedne takve varijable:

Isječak 3.3: Primjer urlpatterns varijable

```
urlpatterns = [
    path('login/', views.MyTokenObtainPairView.as_view(),
        name='token_obtain_pair'),
    path('register/', views.registerUser, name='register'),
    path('profile/', views.getUserProfile,
```

```
        name="user-profile"),
    path('profile/update/', views.updateUserProfile,
        name="user-profile-update"),
    path('verify/', views.validateEmailToken,
        name="user-verify"),
]
```

U primjeru (Isječak 3.3) je vidljivo pet ruta vezanih za poglede koji obavljaju operacije na korisnicima. Ova varijabla izdvojena je u svoju datoteku i kasnije uključena u glavnu *urlpatterns* varijablu aplikacije.

Serijalizatori: unutar web aplikacije serijalizatori omogućavaju slanje objekata preko HTTP protokola pretvorbom objekata u JSON oblik. Oni su bitan element u ostvarenju REST API-a. Na primjer koda serijalizatora (Isječak 3.4) vidljiv je kod koji uzima model rezultata testa pretvara sve njegove atribute u JSON oblik i šalje ih kao odgovor. On također na ulazu raspakirava JSON oblik u tip modela kojim barata Django REST API.

Isječak 3.4: Serijalizator rezultata testa

```
class TestResultSerializer(serializers.ModelSerializer):
    register = RegisterSerializer(many=False, read_only=True)
    class Meta:
        model = TestResult
        fields ='__all__'
```

Signali: Signali osluškuju radnje u aplikaciji i reagiraju ako su zadovoljeni njima dani uvjeti. Nakon toga obavljaju neki koristan posao i ponovno osluškuju ostvarenje uvjeta. Primjer signala:

Isječak 3.5: Signal za sinkronizaciju korisničkog imena

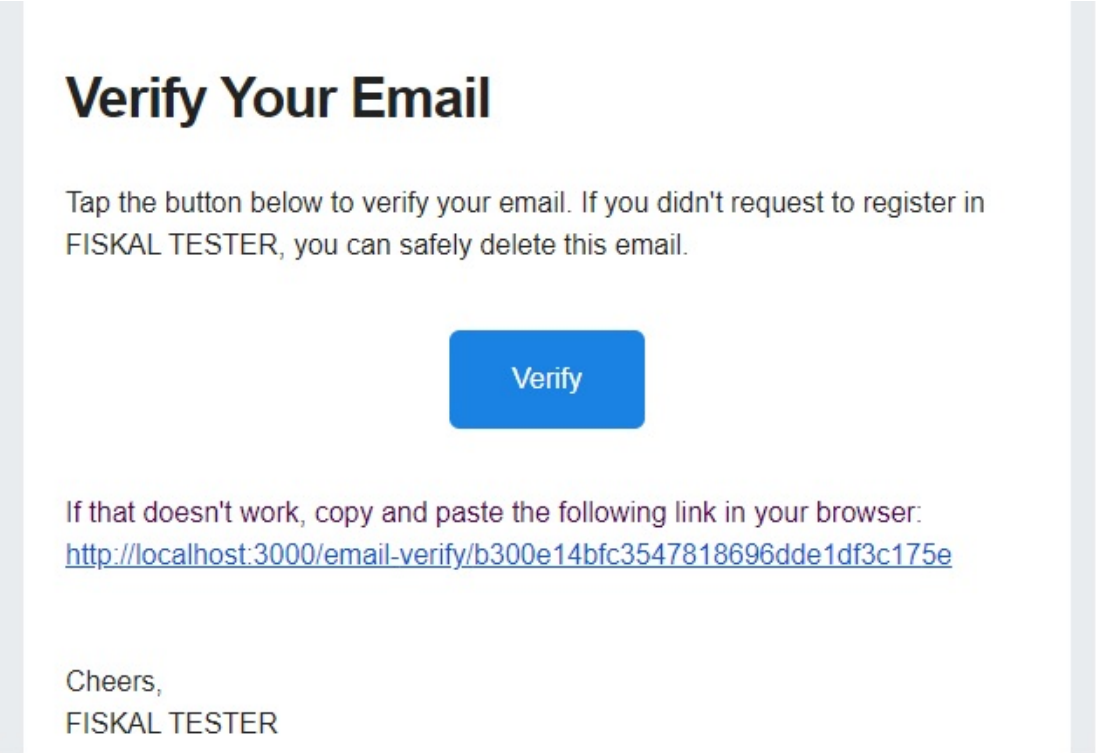
```
def updateUser(sender, instance, **kwargs):
    user = instance
    if user.email != '':
        user.username = user.email

pre_save.connect(updateUser, sender=User)
```

U primjeru (Isječak 3.5) je kod signala koji prije svakog spremanja korisnika u bazu

podataka vrijednost atributa *username* postavi na vrijednost atributa elektroničke pošte korisnika. Ova radnja se odvija jer je unutar aplikacije omogućena prijava samo preko elektroničke pošte korisnika.

3.1.2. Verifikacija elektroničke pošte

Nakon što korisnik izvrši registraciju u aplikaciju njegov račun i dalje nije aktivan (ne može se prijaviti i koristiti sve funkcionalnosti). Prije toga mora verificirati svoju elektroničku poštu. To se ostvaruje da tako da se prilikom registracije generira nasumičan kod. Taj kod se formira u URL koji se šalje korisniku na njegovu elektroničku poštu. Ako se korisnik uspješno verificira njegov računa postaje aktivan i omogućeno mu je korištenje aplikacije, a kod se uklanja iz baze podataka. Primjer verifikacijske poruke:

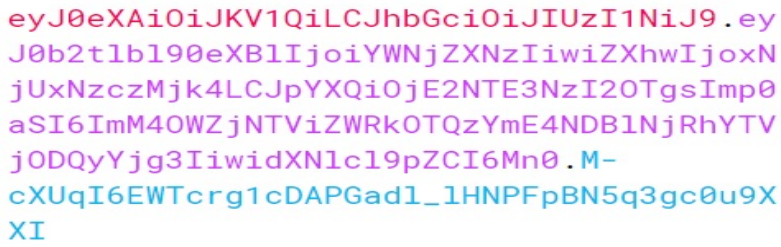


Slika 3.3: Verifikacijska poruka koja se šalje korisniku

3.1.3. JSON Web Token (JWT)

Kako bi se ostvarila autorizacija unutar web aplikacije korišten je JSON Web Token ili skraćeno JWT. Pojam autorizacije predstavlja ima li neki korisnik prava pristupa resursu kojem pokušava pristupiti. Jednom kad se korisnik autentificira tj. uspješno

prijavi u aplikaciju, aplikacija mu dodjeljuje JWT token. Taj token korisnik koristi za pristup zaštićenim dijelovima web aplikacije. Korisnik će token morati postaviti u svaki svoj sljedećim zahtjev. Za postavljanje tokena u zahtjeve brinuti će se React knjižnica. Detaljnije JSON Web Token (JWT) je otvoreni standard (RFC 7519) koji definira kompaktan i samostalan način za siguran prijenos informacija između strana kao JSON objekta [12]. JWT-ovi se mogu potpisati korištenjem tajne (s HMAC algoritmom) ili para javnih/privatnih ključeva pomoću RSA ili ECDSA. Unutar aplikacije JWT se potpisuje HMAC algoritmom.



Slika 3.4: Primjer JSON Web tokena

Glavne karakteristike JWT-a su jednostavnost korištenja i sigurnost. Također JWT je manje opsežan od SAML tokena što ga čini lakšim za prijenos. Također JSON parseri uobičajeni su u većini programskih jezika jer se mapiraju izravno na objekte što olakšava rad s JWT tokenima na raznim domenama. Prednost JWT tokena naspram kolačića sjednice je što za JWT pristup nije potrebna nikakva dodatna tablica unutar baze podataka što čini sustav lakšim za skaliranje [15].

3.1.4. React Web aplikacija

React je JavaScript biblioteka otvorenog koda nastala za razvoj pogleda korisničkih sučelja. Koristi se za izradu kompliciranijih web aplikacija i poboljšanja interaktivnosti web aplikacije [8]. React je nastao interno pod okriljem Facebooka koji i danas najviše doprinosi razvoju ove biblioteke. Za izradu aplikacije potrebno je koristiti dvije biblioteke React i React DOM. React aplikacija u ovom projektu služi za dohvat podataka preko Django REST API-a i za slanje podataka prema istom API-u.

Isječak 3.6: Primjer React pogleda za izmjenu slika početne stranice

```
function HomeCarousel() {
   return (
```

```
        <Carousel pause='hover' className='bg-dark'>
            {welcomes.map(welcome => (
                <Carousel.Item key={welcome._id}>
                        <Image src={welcome.image}
                            alt={welcome.name} fluid/>
                        <Carousel.Caption
                            className='carousel.caption'>
                            <h4>{welcome.name}</h4>
                        </Carousel.Caption>
                </Carousel.Item>
            ))}
        </Carousel>
    )
}
export default HomeCarousel
```

Primjer (Isječak 3.6) koristi predefinirani React element *Carousel* iz biblioteke React Bootstrap. On se koristi na početnoj stranici aplikacije, a funkcija mu je izmjena slika i opisa slika koji su mu predani. Na ovaj način s vrlo malo koda ostvarena je funkcionalnost za koju bi prilikom korištenja "običnog" JavaScripta bilo potrebno napisati puno više koda. Također postavkom atributa *pause* na vrijednost *hover* izmjena se stopira ako korisnik pokazivačem dođe iznad slike.

Isječak 3.7: Primjer React pogleda za prijavu korisnika

```
    ...
    const [password, setPassword] = useState('')

    const dispatch = useDispatch()
    const location = useLocation()
    const navigate = useNavigate()

    const redirect = location.search ?
        location.search.split('=')[1] : '/'
    const userLogin = useSelector(state => state.userLogin)
    const { error, loading, userInfo } = userLogin

    useEffect(() => {
        if (userInfo) {
```

```
            navigate(redirect)
        }
    }, [navigate, userInfo, redirect])

    const submitHandler = (e) => {
        e.preventDefault()
        dispatch(login(email, password))
    }

    return (
        <FormContainer>
            // kod za kreiranje forme za prijavu korisnika
        </FormContainer>
    )
    ...
```

Na gornjem isječku koda (Isječak 3.7) vidljiva je upotreba Reacta s bibliotekom Redux. Za njihovu jednostavniju integraciju korištena je biblioteka React Redux. Na isječku je prikazan kod korišten na stranici za prijavu korisnika u aplikaciju. On korisniku vraća formu za prijavu (detalji nisu vidljivi na isječku), a nakon što korisnik pošalje podatke u suradnji s Reduxom šalje podatke prema Django REST API-u. Više o funkcionalnosti Reduxa u sljedećem poglavlju.

3.1.5. Redux

Redux je JavaScript knjižnica otvorenog koda namijenjena upravljanju i centraliziranju stanja aplikacije. Knjižnica je inspirirana sličnom knjižnicom Flux, a nastala je kako bi se olakšalo upravljanje stanjem aplikacije. Najčešće se koristi uz web aplikacije izrađene u Reactu, Angularu, Vue.js-u. Redux biblioteka je izrađena na paradigmama funkcijskog programiranja čije prednosti iskorištava. Situacije kada u web aplikacijama dolazi do čestih promjena stanja gdje se u različitim dijelovima aplikacije mijenja isti podatak značajno otežavaju razvoj istih te može doći do gubitka stanja, točnije kontrole nad stanjem. Redux upravo rješava takve probleme tako da se svi podaci spremaju u jedno skladište koji služi kao jedini izvor istine u aplikaciji [9]. Promjene nad tim stanjem moguće su jedino preko akcija. Primjer koda akcije:

Isječak 3.8: Primjer akcije za prijavu korisnika

```
export const login = (email, password) => async (dispatch) =>
    {
    try {
        dispatch({
            type: USER_LOGIN_REQUEST
        })

        const config = {
            headers: {
                'Content-type': 'application/json'
            }
        }

        const { data } = await axios.post(
            '/api/users/login/',
            { 'username': email, 'password': password },
            config
        )

        dispatch({
            type: USER_LOGIN_SUCCESS,
            payload: data
        })

        localStorage.setItem('userInfo', JSON.stringify(data))

    } catch (error) {
        dispatch({
            type: USER_LOGIN_FAIL,
            payload: error.response && error.response.data.detail
                ? error.response.data.detail
                : error.message,
        })
    }
}
```

Akcija koda na isječku (Isječak 3.6) koristi se prilikom prijave korisnika u aplikaciju. Ona uzima elektroničku poštu i lozinku korisnika i u JSON formatu ih prosljeđuje

prema Django REST API komponenti aplikacije. Tip ove akcije je *USER_LOGIN _REQUEST* i on se mora razlikovati za svaku akciju. Akcija čeka na odgovor koji zatim metodom *dispatch* pohranjuje u skladište. To metoda jedini je način za slanje stanja prema skladištu. U ovom primjeru akcija će ovisno hoće li API javiti grešku ili vratiti neki sadržaj odabrati koje stanje će pohraniti u skladište.

Reduktori su funkcije koje vraćaju novo stanje za primljeno staro stanje i akciju. Unutar reduktora implementira se na koji način će akcija promijeniti stanje aplikacije. Reduktori unutar ove web aplikacije mogu se podijeliti unutar tri grupe: reduktori za upravljanje korisnicima, reduktori za upravljanje fiskalnim blagajnama i reduktori za upravljanje testovima. Na kraju se svi reduktori kombiniraju u jedan glavni reduktor. Primjer kod reduktora prijave korisnika:

Isječak 3.9: Reduktor akcije za prijavu korisnika

```
export const userLoginReducer = (state = {}, action) => {
   switch (action.type) {
      case USER_LOGIN_REQUEST:
         return { loading: true }

      case USER_LOGIN_SUCCESS:
         return { loading: false, userInfo: action.payload }

      case USER_LOGIN_FAIL:
         return { loading: false, error: action.payload }

      case USER_LOGOUT:
         return {}

      default:
         return state
   }
}
```

Ovaj reduktor na temelju stanja rezultata akcije dane u prethodnom isječku 3.8 obavlja operaciju nad skladištem ovisno o vrijednosti koju dobije na ulaz od navedene akcije. Važno je napomenuti da ako se niti jedan rezultat akcije ne poklopi s onim što reduktor očekuje, reduktor mora vratiti prijašnje stanje jer će se u protivnom ono izgubiti.

3.2. Poslužitelj

Poslužitelj je računalo koje prima i šalje podatke prema mnogostrukim klijentima. Računala koje traže neku uslugu od poslužitelja nazivaju se klijenti. Za sustav koji je oblikovan na ovakav način kaže se da ima klijentsko-poslužiteljsku arhitekturu. Za implementaciju poslužitelja u ovom sustavu izabran je okvir Tornado (detaljnije u potpoglavlju 3.2.1). Poslužitelj se unutar sustava mora ponašati prema specifikaciji Porezne uprave. Zbog toga mora primati zahtjeve u XML formatu, manipulirati njima i vračati ih u XML formatu. Poslužitelj za svaki zahtjev gleda njegovu lozinku (prethodno generirana unutar Web aplikacije) poslanu u zaglavlju poruke. Ako takva lozinka ne postoji ili nije valjana zahtjev za testiranje se odbija. Ako je sve u redu poslužitelj prema broju računa i blagajni sortira zahtjev, validira ga, manipulira odgovorom i vrača ga klijentu.

3.2.1. Tornado

Tornado je web aplikacijski okvir napisan u programskom jeziku Python i asinkrona mrežna knjižnica [14]. Tornado je odabran jer nudi veliki broj istovremenih konekcija, a i vrlo je jednostavan za korištenje. Na isječku u nastavku (Isječak 3.10) vidljivo je kako se u nekoliko linija koda korištenjem Tornado okvira izradi poslužitelj. U konkretnom primjeru poslužitelj će čekati zahtjeve na vratima 8888.

Isječak 3.10: Primjer kreiranja tornado servera

```
...
def make_app():
    return tornado.web.Application([
        (r"/", MainHandler),
    ])

if __name__ == "__main__":
    app = make_app()
    app.listen(8888)
    tornado.ioloop.IOLoop.current().start()
```

Kod s prethodnog isječka 3.10 dio je klase MainHandler. Također na isječku je vidljivo da je *MainHandler* klasa predana web aplikaciji. Ona će na lokaciji "/" osluškivati klijentske zahtjeve. Svaki zahtjev se zatim obrađuje unutar datoteke test_controller.py. Unutar te datoteke provjerava se ima li korisnik prava pristupa poslužitelju, a zatim dalje datoteka test_handler upravlja postupkom testiranja. Rezultat test ponovno se

vrača u datoteku test_controller.py u kojoj se nalaze metode za sinkronizaciju s bazom podataka.

U postupku testiranja prva na redu je validacija zahtjeva. Unutar Python klase *CheckImplementation* odvija se provjera je li zahtjev u skladu sa specifikacijom Porezne uprave tj. postoji li neka od šest propisanih pogrešaka. Kao primjer provjere izdvojen je isječak koda za provjeru potpisa zahtjeva. Na isječku (Isječak) je vidljivo da se prvo iz XML zahtjeva uzima certifikat. To je odrađeno unutar metode *get_signature_cert()*. Nakon toga se taj certifikat sa zahtjevom predaje klasi *XMLVerifier* koja obavlja provjeru potpisa. Klasa *XMLVerifier* uvezena je iz Python signxml knjižnice. Ta knjižnica osim za provjere potpisa također je kasnije korištena i za potpisivanje odgovora.

Isječak 3.11: Primjer provjere potpisa

```
try:
        cert = CheckImplementation.get_signature_cert()
        XMLVerifier().verify(xml, x509_cert=cert).signed_xml
    except (InvalidCertificate, InvalidSignature) as e:
        errorSet.add("s004") # Signature not valid
```

Nakon što je obavljena validacija zahtjeva unutar sustava se kreira odgovor. Odluka kakav će odgovor biti vraćen blagajni donosi se slučajno, točnije rečeno generiranjem slučajnih brojeva i usporedbom s unaprijed određenim vrijednostima. Na isječku 3.12 vidljiv je dio koda koji odlučuje hoće li se odgovor potpisati. Svaki odgovor ima pedeset posto šanse biti potpisan. Na identičan način prethodno se određuje hoće li se vratiti blagajni vratiti nepostojeće pogreške ili ispravan odgovor za njen zahtjev.

Isječak 3.12: Primjer dijela koda za modifikaciju odgovora

```
signed = random.randint(0,1)
   if(signed == 0):
      testCase.dom = sign_response.sign(testCase.dom)
```

U procesu testiranja na poslužitelju dalje se komunicira s bazom pdoataka, bilježe se rezultati validacije i testovi koji su pokrenuti. Taj postupak razdvojen je u više datoteka i metoda. Za pomoć pri učitavanju, kreiranje i manipulaciju XML zahtjeva i odgovora korištena je Python knjižnica lxml. Na isječku 3.13 je primjer kako knjižnica

lxml olakšava posao pri pretvorbama u formatima jer je u samo jednoj liniji odgovor pretvoren *String* format.

Isječak 3.13: Primjer kreiranja tornado servera

```
    response = etree.tostring(response,
        pretty_print="true").decode(encoding="utf-8")
```

3.2.2. Skripta za upravljanje blagajnama u testu

Uz pomoč APScheduler Python knjižnice unutar sustava je implementirana skripta za čišćenje starih lozinki za pokretanje testova iz baze. Skripta se pokreče u intervalu od svakih sat vremena i šalje bazi podataka upit koji će obrisati svaku lozinku za test koja je generirana prije više od sat vremena. Ova funkcionalnost dodana je iz sigurnosnih značajki. Sat vremena je dovoljno da korisnik obavi test koji je započeo, a ako mu je potreban novi vrlo jednostavno će izgenerirati novu lozinku. Važno je napomenuti da skripta neće izbrisati rezultate i detalje testa već samo njegovu lozinku. Isječak dijela skripte:

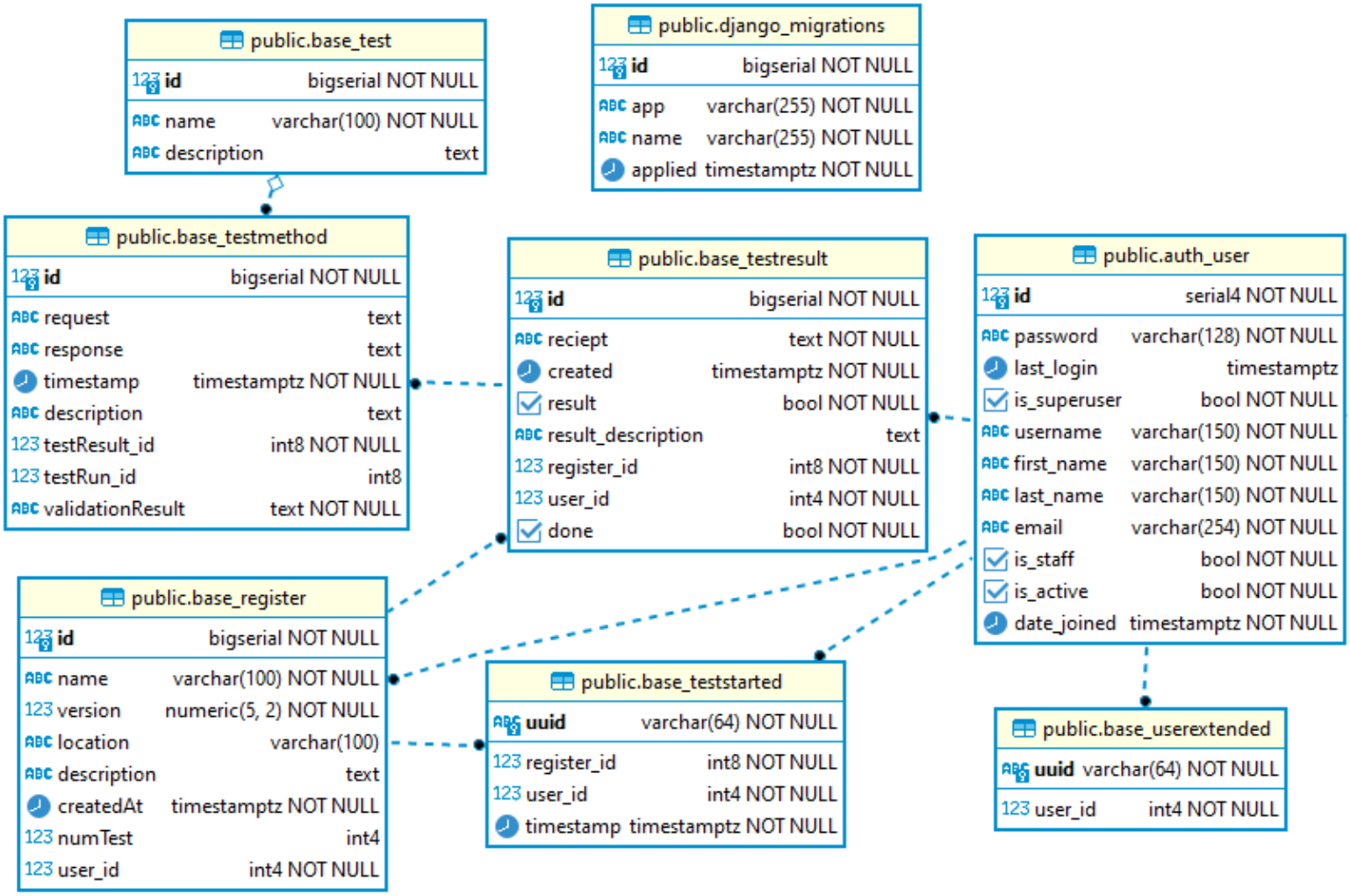
Isječak 3.14: Primjer skripte za sinkronizaciju s bazom

```
if __name__ == '__main__':
    scheduler = TornadoScheduler()
    scheduler.add_job(clean_old_uuids, 'interval', hour=1)
    scheduler.start()

    try:
        IOLoop.instance().start()
    except (KeyboardInterrupt, SystemExit):
        pass
```

Skripta iz primjera (Isječak 3.14) koristi klasu *Tornado Scheduler* definiranu u knjižnici APScheduler. Ta klasa specijalizirana je radu s Tornado serverom [13]. Iz isječka se vidi da se ona pokreče svakih sat vremena i zove metodu *clean_old_uuids* koja odrađuje prethodno opisani posao.

3.3. Baza podataka

Baza podataka središnji je dio informacijskog sustava. Baza podataka skup je međusobno povezanih podataka, pohranjenih zajedno, uz isključenje bespotrebne redundantnosti, koji mogu zadovoljiti različite primjene. Baza korištena u ovom sustavu je relacijska baza podataka (uz relacijske baze podataka popularne su i semantičke baze podataka). Podaci su u bazi strukturirani tako da mogu zadovoljiti zahtjeve korisnika, tj. da je što jednostavnije njihovo dohvaćanje, pohranjivanje, brisanje ili modificiranje. Na slici (Slika 3.5) vidljiv je dijagram baze podataka sustava sa značajnijim tablicama koje baza sadrži, a dalje u tekstu je unutar tablice (Tablica 3.1) dan kratki opis svake od tih tablica.



Slika 3.5: Dijagram baze podataka

Naziv tablice	Opis
AuthUser	Vlasnik fiskalne blagajne, tj. korisnik sustava.
Userextended	Tablica koja sadrži UUID za verifikaciju korisničkog računa.
DjangoMigrations	U ovu tablicu pohranjene su sve migracije baze podataka. Ti zapisi sadrže detalje promjena rađenih na bazi.
TestStarted	Tablica u koju se dodaje blagajna za koju je pokrenut test. Uz UUID (koji služio kao lozinka za test pohranjuje se i vremenska oznaka valjanosti lozinke.
Register	Tablica sadrži sve podatke o fiskalnoj blagajni.
TestResult	Podaci o rezultatima testiranja za određeni test koji je blagajna pokrenula.
TestMethod	Detaljni podaci za pojedinu test metodu unutar jednog testa blagajne.
Test	Naziv i opis testa koji se izvršava na fiskalnoj blagajni.

Tablica 3.1: Opis tablica baze podataka

Baza podataka unutar sustava izrađena je na PostgreSQL (poznat i kao pgSQL) besplatnom sustavu za upravljanje bazama podataka otvorenog koda. Kao što je već u odlomku navedeno PostgreSQL je relacijska baza podataka, vrlo je popularan odabir za sustav upravljanja bazom podataka i koriste ju mnoge renomirane tehnološke tvrtke. Također PostgreSQL implementira ACID pravila te tako garantira ispravnost podataka u nepredviđenim situacijama [11].

3.4. Docker

Da bi se moglo govoriti o Dockeru potrebno je prvo objasniti pojam kontejnera. Kontejneri su oblik virtualizacije (kao "lagana" virtualna računala) [5]. Kontejneri su dakle izolirano računalno okruženje imaju vlastiti datotečni sustav, izolirano stablo proces, mrežno sučelje. Docker je tehnologija koja je omogućila široku upotrebu i lagano is-

korištavanje Linux kontejnera [7]. Docker omogućuje kreiranje, implementaciju i pokretanje aplikacija koristeći kontejnere. Docker omogućava izvođenje svake aplikacije sigurno i izolirano u kontejneru tako da će se kontejner na svakom računalu pokretati s istim postavkama.

Isječak 3.15: Primjer Dockerfile datoteke

```
FROM node:14.17.0

WORKDIR /app
# add `/app/node_modules/.bin` to $PATH
ENV PATH /app/node_modules/.bin:$PATH
# install app dependencies
COPY package.json ./
COPY package-lock.json ./
RUN npm install --silent
RUN npm install react-scripts@3.4.1 -g --silent

# add app
COPY . ./
EXPOSE 3000
# start app
CMD ["npm", "start"]
```

Na isječku 3.15 prikazan je Dockerfile za izgradnju Docker slike. Iz izrađene slike Docker komandama se dalje izrađuje Docker kontejner. Dockerfile je zapravo tekstna datoteka koja sadrži komande koje korisnik može zvati iz komandnog retka kako bi izgradio Docker sliku.

Isječak 3.16: Primjer docker-compose datoteke

```
version: "3.9"

services:
  server:
    build: tornado/
    ports:
      - "8888:8888"
```

```
react:
   build: frontend/
   ports:
     - "3000:3000"

django:
   build: backend/
   command: ["python", "manage.py", "runserver",
      "0.0.0.0:8000"]
   ports:
     - "8000:8000"
```

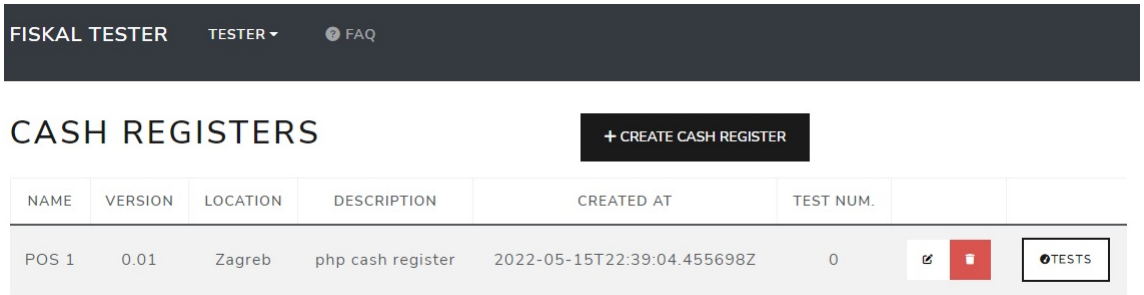
Na isječku 3.16 prikazana je struktura docker-compose.yml datoteke. Compose je alat za definiranje i izgradnju multi-kontejnerskih Docker aplikacija [4]. Unutar YAML datoteka konfiguriraju se postavke za izgradnju aplikacije. zatim se samo jednom narednom pokreću svi servisi koji su prethodno konfigurirani. Docker-compose datoteka iz isječka 3.16 tako pokreče tri kontejnera potrebna za rad sustava. Kontejner poslužitelja, kontejner s React i kontejner s Django aplikacijom. Baza podataka pokrenuta je na internet servisu Heroku.

4. Korištenje aplikacije

Kako bi korisnik koristio sve funkcionalnosti sustava i uopće mogao testirati svoje blagajne prvo se mora registrirati. Korisnik nakon registracije mora potvrditi svoju adresu elektroničke pošte i prijaviti se u sustav. Nakon toga korisnik je uspješno prijavljen. U zaglavlju će se prikazati korisnikovo ime, pritiskom na njega korisnik može odabrati jednu od dostupnih opcija u aplikaciji poput: pregleda blagajni, pregleda blagajni u testu, uređivanja profila ili odjave. Unutar prikaza profila korisnik također može preuzeti certifikate potrebne za testiranje.

4.1. Funkcionalnosti vezane uz fiskalne blagajne

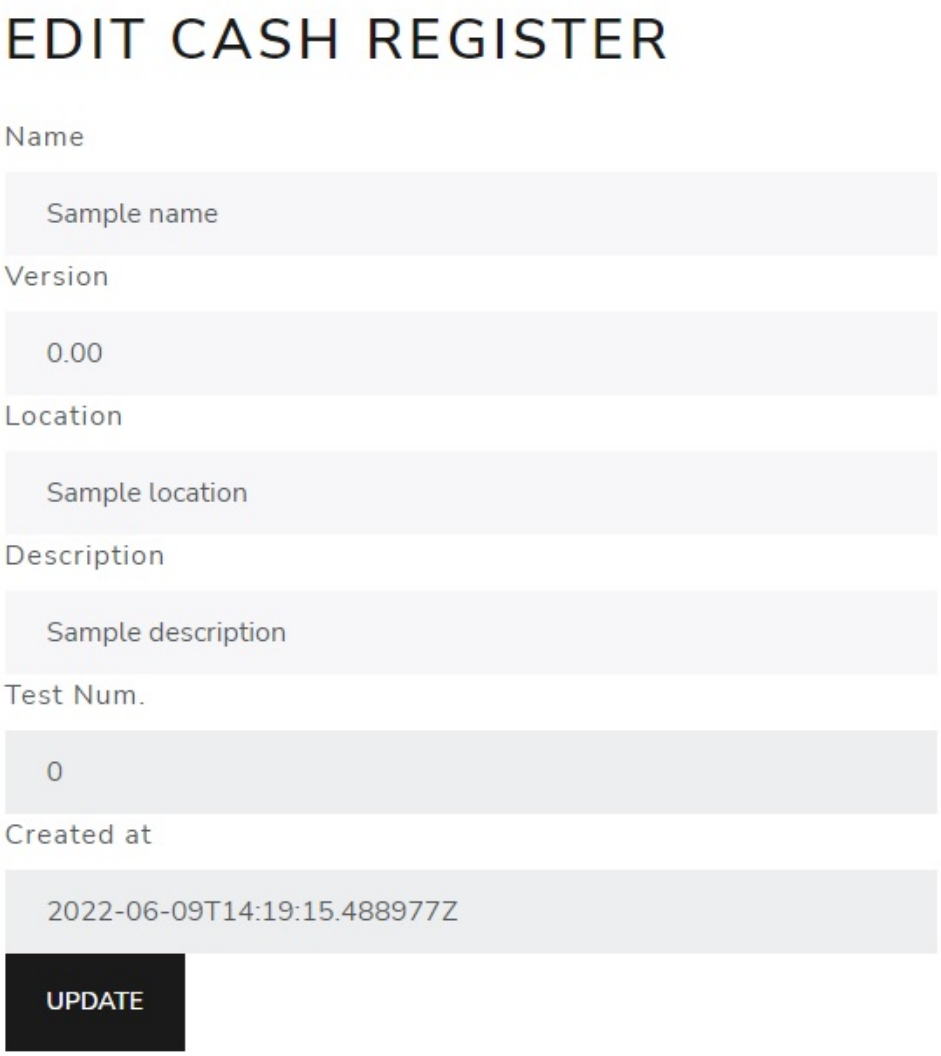
Na komponenti web aplikacije korisnik može pregledavati svoje fiskalne blagajne koje je dodao u sustav (vidljivo na Slika 4.1). Blagajne su prikazane u listi i vidljive su osnovne informacije za svaku blagajnu poput imena, verzije, lokacije i opisa. Također za svaku fiskalnu blagajnu vidljiv je vremenski zapis kada je dodana u sustav i broj testova koji je za nju pokrenut.



Slika 4.1: Primjer prikaz zaslona blagajni

Uz to korisnik na stranici može vidjeti i četiri gumba. Pritiskom na gumb *CREATE CASH REGISTER* korisnik može dodati novu blagajnu u sustav. Uz to korisniku su dostupni i gumbi za brisanje blagajne koji će automatski i izbrisati sve povezane testove. Dostupan je i gumb koji otvara formu za uređivanje podataka o blagajni. Zadnji

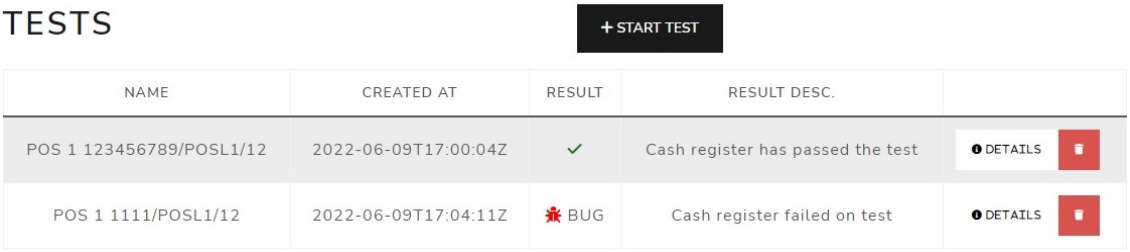
preostali gumb je onaj koji korisnika vodi na stranicu sa svim testovima za određenu blagajnu



Slika 4.2: Primjer zaslona za uređivanje blagajne

4.2. Funckionalnosti vezane uz testiranje

Na komponenti testova za blagajnu korisnik može pregledavati rezultate postojećih testova (Slika 4.3) ili dodati blagajnu u test (Slika 4.4). Za svaki test vidljivo je ime. Ime se konstruira iz imena blagajne i broja računa. Svaki broj računa mora biti jedinstven i on označava zaseban test unutar sustava. Korisnik može vidjeti i vremenski zapis kada je test pokrenut te rezultat tog testa uz kratku poruku. Na prikazu su vidljiva i dva gumba kojima se mogu pregledavati detalji testova ili izbrisati odabrani test.



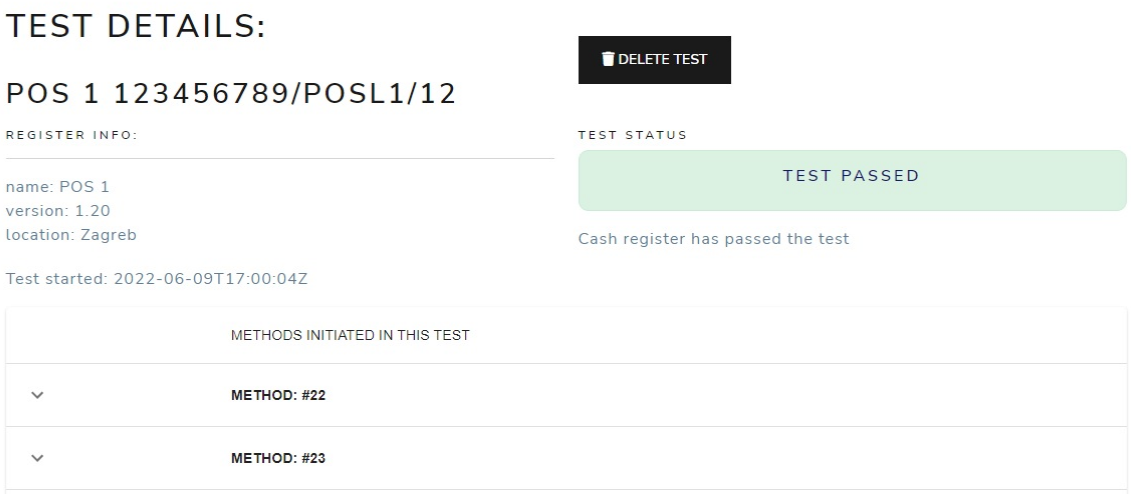
Slika 4.3: Primjer zaslona s rezultatima testova

Dodavanje blagajne u test dodaje se pritiskom na gumb *START TEST*. Nakon pritiska korisnik će biti odveden na stranicu gdje može vidjeti sve svoje blagajne u testu (Slika 4.4). Uz ime blagajne vidljiva je i jedinstveni nasumično generirani tekst (UUID) koji služi kao lozinka za testiranje. Njega je potrebno slati uz svaki zahtjev inače oni neće biti prihvaćeni na poslužitelju.

Slika 4.4: Primjer zaslona s blagajnama dodanim na test

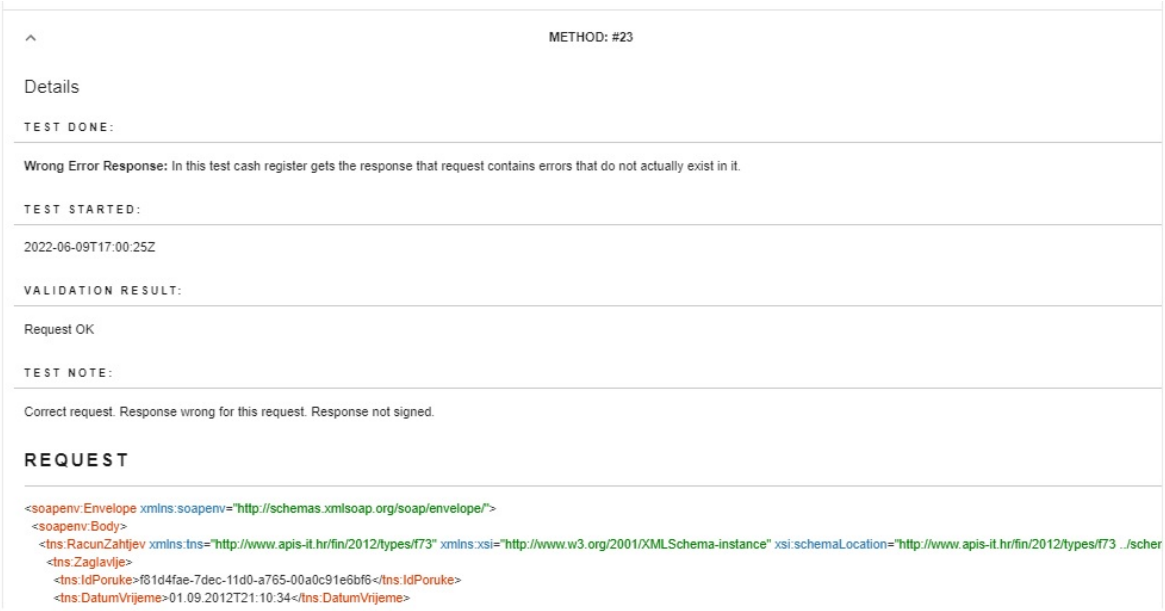
4.2.1. Detaljni prikaz testova

Nakon pritiska na gumb *DETAILS* na stranici s rezultatima testa korisniku će se prikazati detaljni rezultati i svaka izvedena metoda unutar nekog testa. Unutar stranice za prikaz testa vidljiva je poruka o rezultatu testa i osnovne informacije o blagajni na koju se test odnosi. Uz to na stranici se nalazi i gumb kojim korisnik može izbrisati test. Prikaz stranice vidljiv je na slici u nastavku (Slika 4.5).



Slika 4.5: Primjer zaslona s detaljima testa

Na stranici je dostupna i svaka metoda koja je izvedena u testu. Pritiskom na redak s imenom metode on se proširuje i korisniku su dostupne sve informacije o toj metodi (Slika 4.6).



Slika 4.6: Primjer detalja metode unutar nekog testa

Detalji koji se mogu vidjeti su test koji je izveden unutar polja *TEST DONE*. U metodi vidljivoj na navedenoj slici riječ je o testu koji blagajni javlja da je prisutna pogreška iako ona zapravo ne postoji. Svaki zahtjev se na ulazu u sustav validira, a u

polju *VALIDATION RESULT* moguće je vidjeti ima li grešaka u njegovoj implementaciji. U polju *TEST NOTE* dostupna je sažeta poruka korisniku o testnoj metodi. U konkretnom slučaju vidljivo je da je zahtjev bio ispravan, a da je korisniku vraćen odgovor koji nije potpisan i koji sadrži pogrešku koja ja zapravo nepostojeća. Također uz svaku metodu korisnik može vidjeti cijeli zahtjev koji je stigao u sustav kao i odgovor koji je proslijeđen blagajni.

Ovdje je vidljiva i mana ovakvog načina testiranja jer isti nije skalabilan. Testovi se kako je već objašnjeno generiraju slučajno i moguće je da blagajna tijekom jednog testa više puta redundantno testira blagajnu istim testom (npr. vračanjem nepotpisanog odgovora). Također isto je moguće da poslužitelj završi test i označi ga ispravnim, a da pritom propusti neki test koji bi našao pogrešku u implementaciji fiskalne blagajne.

5. Zaključak

Ovim radom opisan je razvijeni sustav za automatizirano testiranje fiskalnih blagajni. Sustav se sastoji od klijentske web aplikacije razvijene u Django REST okviru i React JavaScript knjižnici, poslužitelja za provođenje testiranja razvijenog u Tornado okviru i PostgreSQL baze podataka. Za što bolji rad sustava korištene su i razne druge Python i JavaScript knjižnice poput Reduxa, signxml-a, lxml-a i mnogih drugih. Nadalje opisani su osnovni zahtjevi koje fiskalna blagajna mora zadovoljiti kako bi sudjelovala u procesu fiskalizacije, objašnjeni su pojmovi digitalnog potpisa i digitalnih certifikata. Opisano je testiranje i dane su upute za korištenje sustava.

Sustav je kreiran s ciljem sprečavanja uporabe pogrešno implementiranih fiskalnih blagajni. Cilj sustava je pružiti proizvođačima fiskalnih blagajni mjesto gdje će moći dobiti pouzdane i detaljne informacije o svim greškama koje su prisutne u njihovim blagajnama. Tako sam proces fiskalizacije postaje sigurniji i proizvođači dobivaju jasne povratne informacije jesu li njihovi proizvodi spremni za tržište.

Rezultati ovog rada su generalno zadovoljavajući. Za vrijeme izrade sustava naučio sam dosta o samom procesu fiskalizacije. Generalno rad smatram zadovoljavajućim. Sustav je generalno dobar i daje dobar temelj za izgradnju još kompleksnijeg, skalabilnijeg sustava.

Najveću mogućnost napretka vidim u procesu testiranja koji trenutno nije previše skalabilan. Kako bi se on poboljšao trebalo bi propisati način na koji će blagajna bilježiti odgovore koje joj poslužitelj proslijedi. Njih bi kasnije korisnik predao u sustav unutar testa za koji je taj odgovor zabilježen. Tako jasno bi se znalo kako je blagajna reagirala na koji test, lakše bi se pratile i ispravljale greške i spriječilo bi se redundantno ponavljanje ili preskakanje nekih testova. U trenutnom načinu testiranja moguće je da blagajna više puta prođe jedan test, a preskoči neki koji bi otkrio moguću pogrešku u njenoj implementaciji. Drugu mogućnost poboljšanja vidim u organizaciji koda. Toč-

nije trebalo bi naći način u kojem će se u budućnosti bez pisanja puno koda lagano moći dodati novi test. Trenutno zbog načina na koji je sustav automatiziran potrebno je dodati novi kod na više mjesta kako bi sustav i dalje bio automatiziran, a svakim novim dodavanjem koda povećava se složenost koda u sustava. Najveći problemi u tome predstavljaju promjene nad dijelom koda koji javlja povratne informacije korisniku i bilježi provođenje testova. U nekoj daljoj budućnosti u sustav bi se mogla integrirati i umjetna inteligencija koja bi učila koja su greške najčešće i time možda smanjila vrijeme testiranja i povećala zadovoljstvo korisnika. Unaprjeđenja su moguća i u poboljšanju izgleda web aplikacije kako bi korisnicima korištenje bilo što jednostavnije i ugodnije.

LITERATURA

[1] Republika hrvatska, Porezna uprava, APIS IT. Fiskalizacija - Tehnička specifikacija za korisnike, Verzija 2.1.

[2] Django MTV arhitekturni obrazac. [Mrežno]. Dostupno: https://docs.djangoproject.com/en/4.0/faq/general/. [Pristupano: 9. lipnja 2022.].

[3] Digitalni certifikat. [Mrežno]. Dostupno: https://sectigo.com/resource-library/what-is-x509-certificate. [Pristupano: 4. lipnja 2022.].

[4] Docker Compose. [Mrežno]. Dostupno: https://docs.docker.com/compose/. [Pristupano: 9. lipnja 2022.].

[5] Docker detalji. [Mrežno]. Dostupno: https://www.srce.unizg.hr/files/srce/docs/edu/edu4it/edu4it_uvod_u_docker_20161025.pdf. [Pristupano: 9. lipnja 2022.].

[6] Django web okvir. [Mrežno]. Dostupno: https://www.djangoproject.com/. [Pristupano: 6. lipnja 2022.].

[7] Docker. [Mrežno]. Dostupno: https://docs.docker.com/get-started/overview/. [Pristupano: 9. lipnja 2022.].

[8] React knjižnica. [Mrežno]. Dostupno: https://reactjs.org/,. [Pristupano: 6. lipnja 2022.].

[9] Redux knjižnica. [Mrežno]. Dostupno: https://redux.js.org/,. [Pristupano: 7. lipnja 2022.].

[10] Django REST okvir. [Mrežno]. Dostupno: https://www.django-rest-framework.org/. [Pristupano: 7. lipnja 2022.].

[11] Postgres. [Mrežno]. Dostupno: https://www.postgresql.org/files/developer/transactions.pdf. [Pristupano: 5. lipnja 2022.].

[12] JSON Web Token. [Mrežno]. Dostupno: https://jwt.io/introduction. [Pristupano: 5. lipnja 2022.].

[13] APScheduler Tornado. [Mrežno]. Dostupno: https://apscheduler.readthedocs.io/en/3.x/modules/schedulers/tornado.html. [Pristupano: 7. lipnja 2022.].

[14] Tornado web okvir. [Mrežno]. Dostupno: https://www.tornadoweb.org/en/stable/. [Pristupano: 7. lipnja 2022.].

[15] JWT usporedba. [Mrežno]. Dostupno: https://developer.okta.com/blog/2022/02/08/cookies-vs-tokens. [Pristupano: 5. lipnja 2022.].

Sustav za automatizirano testiranje fiskalnih blagajni

Sažetak

U procesu fiskalizacije svaki obveznik fiskalizacije dužan je izdane račune prijavljivati Poreznoj upravi. Kako bi obveznici mogli ostvariti te uvjete moraju koristi programska rješenja fiskalnih blagajni. Fiskalne blagajne izrađuju razni proizvođači što uz korištenje tehnologija poput kriptografije postavlja pitanje koliko su njihove implementacije ispravne. U ovom radu opisuje se automatizirani sustav razvijen za potrebe testiranja njihove implementacije. U procesu testiranja sustav validira svaki zahtjev, za njega generira pogrešan odgovor i prati daljnje reakcije blagajne. U radu su opisani tehnički zahtjevi koje blagajna mora ispuniti kako bi sudjelovala u procesu fiskalizacije. Opisana je arhitektura izrađenog sustava i tehnologije koje su se koristile u njegovoj izgradnji. Na kraju su dane upute za ispravno korištenje sustava.

Ključne riječi: Fiskalna blagajna, Porezna uprava, automatizirani sustav, Web aplikacija.

System for automated testing of fiscal cash registers

Abstract

In the process of fiscalization, each obligor of fiscalization is obliged to report the issued invoices to the Tax Administration. For obligors of fiscalization to be able to meet these conditions, they must use fiscal cash register software solutions. Fiscal cash registers are developed by different manufacturers, which with the use of technology such as cryptography raises the question of how correct their implementations are. This paper describes an automated system developed for testing their implementation. In the testing process, the system validates each request, generates the wrong answer for it, and monitors further cash register reactions. It describes the technical requirements that the fiscal cash register must meet to participate in the fiscalization process. It also describes the architecture of the developed system and the technologies used in its construction. In the end, instructions for the correct use of the system are given.

Keywords: Fiscal cash registers, Tax administration, automated system, Web application.