SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 273

Web aplikacija za upravljanje ispitivanjem fiskalnih blagajni

Vjekoslav Prpić

Zagreb, lipanj 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 12. ožujka 2021.

ZAVRŠNI ZADATAK br. 273

Pristupnik:	**Vjekoslav Prpić (0036520088)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	doc. dr. sc. Stjepan Groš

Zadatak: **Web aplikacija za upravljanje ispitivanjem 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 sustav na temelju okvira Django te Angular koji bi omogućio vlasnicima fiskalnih blagajni registraciju, prijavu blagajni za ispitivanje, provođenje i nadzor ispitivanja te pregled rezultata ispitivanja. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 11. lipnja 2021.

SADRŽAJ

1. Uvod

U Republiku Hrvatsku uvedena je fiskalizacija s ciljem nadzora prometa u pri gotovinskom plaćanju, te sprječavanje crnog tržišta. Fiskalne blagajne provode fiskalizaciju tako da se svaka provedena transakcija prijavi poreznoj upravi. S obzirom na to da fiskalne blagajne proizvode različiti proizvođači postoji mogućnost da se pojedinom proizvođaču dogodi pogreška u implementaciji. Kako se u komunikaciji fiskalne blagajne s poreznom upravom koristi kriptografija, certifikati i digitalni potpisi čija upotreba nije toliko raširena, povećava se mogućnost pogreške. Kako bi se olakšao pronalazak tih pogrešaka postoji potreba za izgradnjom sustava koji bi omogućio ispitivanje ispravnosti implementacije fiskalne blagajne.

U okviru ovog rada razvijena je web aplikacija za registriranje vlasnika fiskalnih blagajni, prijavu fiskalnih blagajni za testiranje, provođenje testiranja te analizu rezultata ispitivanja.

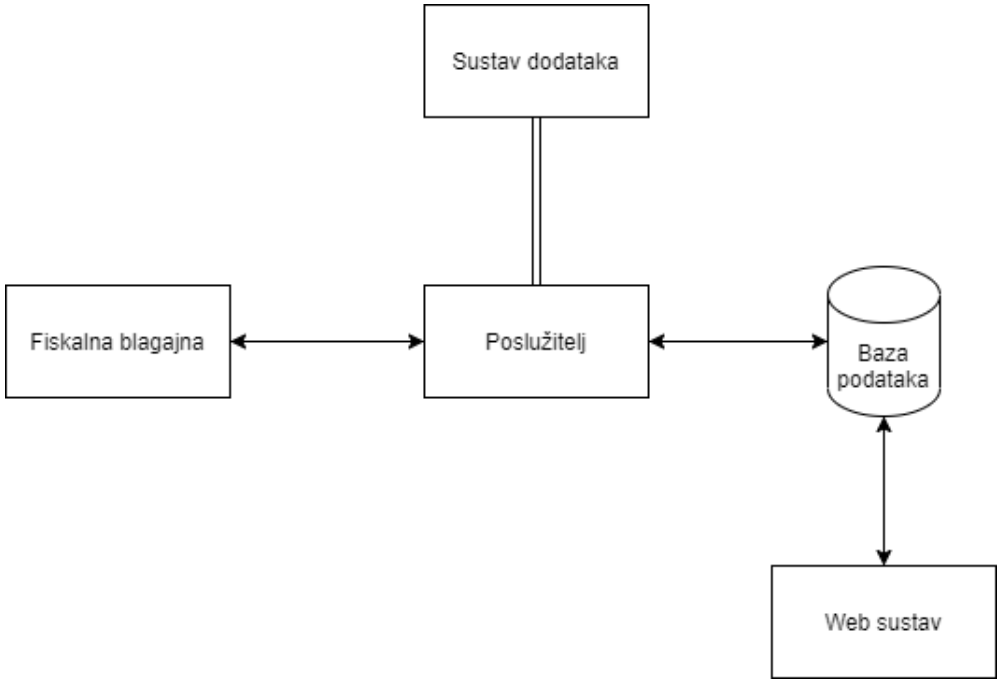
Ovaj rad je podijeljen na pet poglavlja. U prvom poglavlju *Kratki opis sustava* opisuje se način rada i struktura cijelog sustava. U drugoj cjelini *Arhitektura web sustava* opisana je arhitektura web sustava, odnosno opisuje se baza podataka, Django web okvir zajedno sa njegovim dodatkom Django REST okvir i klijentska web aplikacija koja je rađena u web okviru Angular. U trećoj cjelini *Autentifikacija korisnika* prikazan je način verificiranja i registriranja korisnika. Četvrta cjelina, *Generiranje certifikata*, pokazuje način generiranja potpisanih certifikata koji se koriste za potpisivanje XML zahtjeva blagajne. U petoj cjelini *Korištenje aplikacije* opisan je način na koji korisnik upravlja aplikacijom.

2. Kratki opis sustava

Koristeći grafičku web aplikaciju korisnik upravlja sustavom. Na web aplikaciji korisnik pregledava već izvršene, pripremljene testove te dodaje nove.

Sustav koristi poslužitelja kako bi kontrolirao provođenje testova tako da korisnik putem fiskalne blagajne šalje zahtjev koji se dalje obrađuje te šalje odgovor. Odgovor nije nužno ispravan, on se može mijenjati putem sustava dodataka. Sustav na taj način može slati neispravne odgovore te tako testirati ispravnost blagajne.

Sustavom dodataka se poslani zahtjev obrađuje, te se stvara odgovor tako da odabrani dodaci manipuliraju zahtjevom na različite načine.



Slika 2.1: Prikaz arhitekture sustava

3. Arhitektura web sustava

Struktura praktičnog dijela rada je podijeljena u tri sloja:

- Baza podataka
- Django REST okvir
- Klijentska Web aplikacija

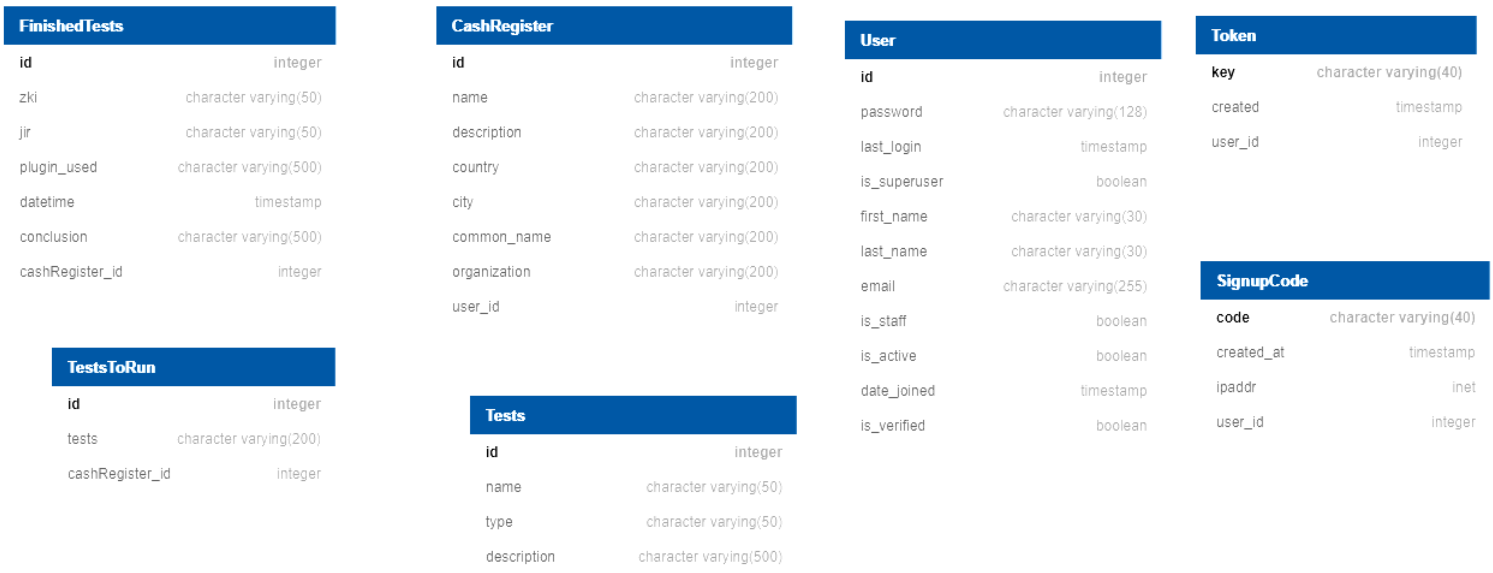
Takvom podjelom omogućeno je da svaki sloj obavlja svoje zadatke neovisno o drugim slojevima, odnosno ostvarena je REST (reprezentacijsko stanje prijenosa) arhitektura aplikacijskog programskog sučelja s Django i Angular web okvirima. Cilj REST-a je povećati skalabilnost, točnije omogučavanje podrške za veliki broj komponenti koje međusobno komuniciraju. Osim skalabilnosti, cilj je povećati preformanse, prilagodljivost, pouzdanost, jednostavnost i prenosivost. To se postiže slijedeći REST principe kao što su klijent-poslužitelj arhitektura, mogućnost predmemoriranja, uporaba slojevitog sustava, podrška za kod na zahtjev i korištenje jedinstvenog sučelja. Ova se načela moraju poštivati kako bi se ostvareni sustav mogao klasificirati kao REST. Isto tako opisana arhitektura omogućava dodatne nadogradnje u sustav, npr. postoji mogućnost ostvarivanja mobilne aplikacije koja koristi istu strukturu. Takva mobilna aplikacija u arhitekturi bi zamijenila samo klijentsku web aplikaciju.

Baza podataka ostvarena je koristeći objektno relacijski sustav za upravljanje bazom podataka PostgreSQL. PostgreSQL je u skladu s ACID pravilima te tako garantira ispravnost podataka u slučaju nepredviđenih situacija kao npr. pri nestanku struje i ostalim greškama.

Slika 3.1: Pojednostavljeniji prikaz arhitekture web sustava

3.1. Baza podataka

Kako bi ostvarili ovaj sustav koristi se baza podataka relacijskog tipa čiji su podaci organizirani u tablice. Svaka tablica sastoji se od jedinstvenog imena i skupa atributa koji čine stupce tablice. Cilj baze je strukturno organizirati podatke kako bi se što jednostavnije i efikasnije mogli dohvaćati, izmjenjivati, odnosno unositi podaci. Za pristup podacima iz baze podataka koriste se modeli Django web okvira o kojima je više napisano u nastavku ovog rada. Na slici (Slika 3.2.) prikazan je dijagram baze podataka, a u tablici (Tablica 3.2) opisana je svaka pojedina tablica.



Slika 3.2: Dijagram baze podataka

Naziv	Opis
User	Korisnik aplikacije, odnosno vlasnik fiskalne blagajne koju želi testirati.
SignupCode	Kod koji se šalje korisniku na adresu elektroničke pošte kako bi je mogao verificirati.
Token	Nasumično generirani skup brojeva i slova pomoću kojih se u http zahtjevu provjerava koji korisnik koristi sustav
CashRegister	Popis svih blagajni registriranih u sustav.
TestsToRun	Popis testova za izvršavanje koje je korisnik odabrao.
FinishedTests	Izvršeni testovi u kojima je zapisan rezultat testiranja.
Tests	Popis i opis svih postojećih testova.

Tablica 3.1: Opis tablica baze podataka

3.2. Django

Django je web okvir otvorenog koda baziran na programskom jeziku Python. Omogućava brzu i jednostavnu izradu sigurnih aplikacija bez zamaranja programera detaljima sigurnosnih implementacija.[7]
U ovom radu se koristi kao pristupna točka putem koje se pohranjuju i dohvaćaju podaci korištenjem REST API-ja. Korištenjem dodatnog Django paketa Django REST framework (okvir) izrađen je mrežni API u skladu s REST načelima.

3.2.1. Modeli

Modeli su klase unutar Django okvira koje predstavljaju tablice iz baze podataka. Svi modeli se nalaze unutar datoteke models.py koja je automatski generirana pri stvaranju novog Django projekta. Svi modeli nasljeđuju Djangovu klasu Model iz *django.db.models*. Na primjeru modela Fiskalne Blagajne možemo vidjeti strukturu Django modela:

```
from django.db import models

class CashRegister(models.Model):
        name = models.CharField(max_length=200)
        description = models.CharField(max_length=200, blank=True,
                null=True)
```

```
        country = models.CharField(max_length=200,
                validators=[validateCountry])
        city = models.CharField(max_length=200)
        common_name = models.CharField(max_length=200)
        organization = models.CharField(max_length=200)
        user = models.ForeignKey(MyUser, on_delete=models.CASCADE,
                unique=False)

def validateCountry(value):
        if pycountry.countries.get(alpha_2=value) == None
                                or len(value) != 2:
                raise ValidationError(
                        _('%(value)s is not a country (e.g. HR)'),
                        params={'value': value},)
```

Ispis 3.1: Model fiskalne blagajne

Unutar modula *django.db.models* postoje određena polja koja služe za različitu primjenu, tako npr. *CharField* s opcijom *max_length=200* označuje da će u taj atribut biti zapisan skup znakova do duljine 200. Opcija validators služi kako bi provjerili što se želi zapisati u taj atribut. U ovom slučaju poziva funkciju *def validateCountry(value)* koja će provjeriti odgovaraju li zapisani znakovi imenu države. Isto tako možemo primijetiti *models.ForeignKey*, koji služi kako bi ostvarili relaciju između Fiskalne blagajne i korisnika. Opcija *on_delete=models.CASCADE* postoji kako bi u slučaju brisanja korisničkog računa koji je povezan s fiskalnom blagajnom i ona bila izbrisana.

3.2.2. Django REST okvir

REST je arhitekturalni koncept za aplikacije čije komponente komuniciraju putem mreže. Koristi HTTP protokol kako bi klijentska aplikacija i udaljeni server koji služi za obradu i upravljanje podataka mogli komunicirati. U ovom radu dodatak Django REST okvir služi kako bi se olakšalo korištenje Django web okvira u obliku REST API-a.[4]

Serijalizatori služe kako bi preveli zapis iz baze podataka, odnosno instance modela u jednostavniji oblik zapisa kao npr. JSON i obratno. Potrebni su kako bi se instance modela mogle slati putem HTTP protokola. Serijalizatori se deklariraju u datoteci serializers.py. Sve klase serijalizatora koriste već ugrađeni modul *rest_framework.serializers*. Kao primjer može nam poslužiti serijalizator već spome-

nutog modela Fiskalne blagajne:

```
from rest_framework import serializers
class CashRegisterSerializer(serializers.ModelSerializer):
    class Meta:
        model = CashRegister
        fields = ('id', 'name', 'description', 'country', 'city',
                  'common_name', 'organization', 'user')
```

Ispis 3.2: Serijalizator modela fiskalne blagajne

Moguće je stvoriti novu instancu ovakvog serijalizatora, a kao argument potrebno je proslijediti instancu modela (Ispis 3.3 (1)), odnosno podatke koje želimo serijalizirati u jednostavnom obliku (Ispis 3.3 (2)) .

```
(1) cashRegister = CashRegister.objects.filter(user=request.user)
    serializer = CashRegisterSerializer(cashRegister, many=True)

(2) serializer = self.CashRegisterSerializer(data=request.data)
```

Ispis 3.3: Pozivanje serijalizatora

Pri pozivu serijalizatora u primjeru (Ispis 3.3 (1)) možemo primijetiti da je uključena opcija *many=True* koja označava da se pri jednom pozivu serijalizatora može u argument proslijediti više instanci modela što s podrazumijevanim vrijednostima nije moguće. Nakon ovako spremljenog formata moguće je izvršiti različite pozive na serijalizator:

- *serializer.data* - neobrađeni serijalizirani podatci
- *serializer.is_valid()* - funkcija koja provjerava ispunjavaju li uneseni podaci pravila zadana u modelu
- *serializer.validated_data* - obrađeni serijalizirani podatci
- *serializer.save()* - kreira novu instancu modela sa *serializer.validated_data*

Pogledi se definiraju u views.py datoteci, a služe kako bi obrađivali GET i POST zahtjeve te na njih dali odgovor. U dodatku Django REST okvir postoji i APIView klasa koja služi kako bi se HTTP zahtjevi obrađivali u Django REST *Request* obliku umjesto Django *HttpResponse* obliku. Isto tako svaki zahtjev koji se šalje kao odgovor bit će u *Response* obliku umjesto *HttpResponse*. Tako će svaka *APIException* greška biti uhvaćena i obrađena. Primjer *TestsListView(APIView)* klase:

```
class TestsListView(APIView):

    def get(self, request, format=None):
        tests = Tests.objects.all()
        serializer = TestsSerializer(tests, many=True)
        return Response(serializer.data, status=status.HTTP_200_OK)
```

Ispis 3.4: Primjer APIView pogleda

Iz prethodnog primjera vidimo pogled za dohvaćanje popisa svih postojećih testova. Može se primijetiti i korištenje ranije spomenutih serijalizera.

3.2.3. URL putanje

URL putanje se povezuju s odgovarajućim pogledom u urls.py datoteci. Unutar te datoteke nalazi se lista *urlpatterns* u kojoj se vrši takvo povezivanje. U listi zapisana je putanja u regularnom izrazu i klasa pogleda koja će obrađivati zahtjev koji je ta adresa primila. Primjer *urlpatterns*:

```
urlpatterns = [
        path('', views.CashRegisterView.as_view(),
                name="cashregister"),
        path('download/', views.DownloadCert.as_view(),
                name="download"),
        path('finishedTests/', views.FinishedTestsView.as_view(),
                name="finishedTests"),
        path('testsList/', views.TestsListView.as_view(),
                name="testsList"),
        path('testsToRun/', views.TestsToRunView.as_view(),
                name="testsToRun"),
]
```

Ispis 3.5: Primjer varijable *urlpatterns*

3.3. Klijentska web aplikacija

Klijentska web aplikacija ostvarena je u Angular web okviru. Zadužena je za komunikaciju HTTP zahtjevima s Django REST API serverom kako bi dohvaćala, odnosno slala podatke. Dohvaćeni podaci se onda prezentiraju korisniku, a poslani podaci se dalje obrađuju u Django serveru.

Angular je web okvir otvorenog koda baziran na programskom jeziku TypeScript koji je razvijen kako bi se kompenzirali nedostaci opaženi prilikom uporabe programskog

jezika JavaScript. Iz tog razloga se TypeScript opisuje kao *superset* JavaScript-a. Kako web preglednici ne mogu interpretirati TypeScript on se prevodi u JavaScript. TypeScript omogućava opcionalno statičko tipiziranje podataka u kodu (varijable moraju imati striktno definiran tip podataka te moraju biti statične prilikom prevođenja koda). Statičko tipiziranje uvedeno je kako bi se prilikom prevođenja koda mogla vršiti provjera tipova podataka te na taj način spriječiti pojava greške tijekom izvođenja. Također, uvođenje objektno orijentiranog načina programiranja jedna je od značajnijih nadopuna JavaScript-u.
Glavna svrha Angular web okvira je razvoj klijentske strane web aplikacije. Angular aplikacija se sastoji od više komponenti koje kao skup čine modul. Svaka Angular aplikacija sadrži minimalno jedan modul, odnosno korijenski modul uobičajeno naziva *AppModule*, u kojem se nalazi korijenska komponenta. Ostale komponente tog modula su ugnježđene unutar korjenske komponente te tako grade stablo. Takve komponente, pisane u TypeScriptu, upravljaju HTML predlošcima dajući im logiku.[1]

Komponente su osnovna gradivna jedinica aplikacije. Svaka komponenta je u obliku TypeScript klase sa dekoratorom *@Component()*, HTML predloškom i CSS stilovima. Unutar klase komponente nalazi se aplikacijska logika potrebna za ostvarivanje pojedinog zadatka, odnosno funkcije koje su potrebne kako bi komponenta ispunila željenu funkcionalnost. S obzirom da se komponenta ostvaruje klasom, moguće ju je instancirati i u drugim komponentama koje onda tako tvore stablo. U primjeru (Ispis 3.6) [2] možemo vidjeti osnovnu građu Angular komponente:

```
import { Component } from '@angular/core';

@Component({
  selector: 'hello-world',
  template: `
    <h2>Hello World</h2>
    <p>This is my first component!</p>
    `,
})
export class HelloWorldComponent {
  // The code in this class drives the component's behavior.
}
```

Ispis 3.6: Primjer komponente

Atribut *selector* označava kojim se imenom trenutna komponenta poziva unutar ostalih komponenti. *Template* pokazuje koji će se HTML prikazati. Kao vrijednost *template*

atributa može se zapisati HTML kod ili se koristi atribut *templateUrl* čija je vrijednost putanja do HTML predloška.

Predlošci su HTML datoteke koje renderiraju komponente. Angular proširuje HTML s dodatnom sintaksom kojom se dodaju dinamične vrijednosti iz komponenti. Varijable iz komponenti se zapisuju s dvostrukim vitičastim zagradama unutar HTML-a, npr. *{{ message }}*. Postoji mogućnost pokretanja funkcija komponente iz HTML-a s opcijom *(click)*. Npr. *<button (click)="sayMessage()"* ..., gdje je *sayMessage()* funkcija odgovarajuće komponente. U HTML-u se također mogu koristiti direktive. Direktive služe kako bi mijenjali izgled, ponašanje ili strukturu HTML elementa. Već postojeće strukturalne direktive su:

- *NgFor* - služi za iteriranje kroz polje unutar komponente kako bi kreirali više HTML elemenata s vrijednostima zapisanih u polju komponente

```
<p *ngFor="let message of messages">
                    <td>{{message}}</td>
</p>
```

Ispis 3.7: Primjer direktive NgFor

- *NgIf* - koristi se kako bi u ovisnosti zadanog uvjeta HTML element bio prikazan, odnosno uklonjen

```
<div *ngIf="condition">
        Content to render when condition is true.
</div>
```

Ispis 3.8: Primjer direktive NgIf

- *NgSwitch* - prikazuje određeni HTML element ovisno o rezultatu zadanog uvjeta na način da se iz predodređenih HTML elementa prikaže onaj koji ispunjava uvjet

```
<div [ngSwitch]="switch_expression">
  <p *ngSwitchCase="match_expression_1">...</p>
  <p *ngSwitchCase="match_expression_2">...</p>
  <span *ngSwitchCase="match_expression_3">...</span>

  <p *ngSwitchDefault>...</p>
</div>
```

Ispis 3.9: Primjer direktive NgSwitch

HTTP zahtjevi se kontroliraju pomoću Angular klase *HttpClient*. Primjer iz datoteke *auth.service.ts*:

```
        private _registerUrl =
                "http://localhost:8000/api/accounts/signup/"

  constructor(private http: HttpClient,
              private _router: Router) { }

  registerUser(user: any) {
    return this.http.post<any>(this._registerUrl, user)
  }
```

Ispis 3.10: Izvadak iz datoteke *auth.service.ts*

Na ovom primjeru vidimo kako se može oblikovati bilo koji HTTP zahtjev, koji onda pojedine komponente izvršavaju s metodom *.subscribe()*

4. Autentifikacija korisnika

Kako bi se ograničio pristup web aplikaciji neregistriranim, odnosno neverificiranim korisnicima korišten je dodatak Djangu *django-rest-authemail*[5]. Autentifikacija se temelji na provjeravanju poslanog tokena unutar zahtjeva u odnosu na tokene postojećih korisnika.

4.1. Kodovi za registraciju

Prilikom registracije korisnika generira se nasumični kod koji služi kako bi se verificirala unesena adresa e-pošte. Taj kod je spremljen u bazu podataka zajedno sa detaljima odgovarajućeg korisnika, kako bi se prilikom pristupa kreiranom URL-u, koji u sebi sadrži navedeni kod, korisnik mogao verificirati. Ako je verifikacija uspješna korisniku se generira token za pristup stranici i kod za verifikaciju se briše iz baze. Primjer poruke za registraciju:

```
Verify your email address by clicking on this link:
http://127.0.0.1:4200/signup/verify/3988cfae24c221cf198da084cb4b47d5617bd091
```

Ispis 4.1: Primjer poruke poslane na adresu e-pošte

4.2. Token

Nakon uspješne registracije korisnika, generira se token koji služi kako bi korisnik mogao pristupiti zaštićenim dijelovima web aplikacije. Autentifikacija se provjerava tako da se prilikom ulogiravanja, unutar spremnika web preglednika pohrani token, te se prilikom svakog zahtjeva unutar zaglavlja pošalje taj isti token na Django REST API, koji onda provjerava ispravnost i šalje povratnu informaciju o korisniku.

Kako bi ubacili token u svaki HTTP zahtjev potrebno je iskoristiti Angular klasu *Injector* i sučelje *HttpInterceptor*. Pomoću njih presretnemo svaki HTTP zahtjev, modificiramo ga, te ga takvog proslijedimo dalje. Prikaz funkcije *intercept(req, next)*:

```
intercept(req, next) {
    let authService = this.injector.get(AuthService);
    if (authService.getToken() != null) {
      let tokenizedReq = req.clone({
        setHeaders: {
          Authorization: `Token ${authService.getToken()}`
        }
      })
      return next.handle(tokenizedReq)
    }
    return next.handle(req.clone())
  }
```

Ispis 4.2: Injektiranje tokena u HTTP zahtjev

Primjer zaglavlja HTTP zahtjeva u koji je dodan token:

```
GET /api/accounts/users/me/ HTTP/1.1
Host: localhost:8000
Connection: keep-alive
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="90",
          "Microsoft Edge";v="90"
Accept: application/json, text/plain, */*
Authorization: Token 38146c237a7116e3d13dc805014f9fe1857b59db
```

Ispis 4.3: Zaglavlje injektiranog HTTP zahtjeva

4.3. Limitiranje sadržaja

Ukoliko se prilikom provjere statusa korisnika ustanovi da on nije ulogiran limitira mu se sadržaj web aplikacije. Korisnik koji nije ulogiran može pristupiti isključivo stranicama za registraciju i ulogiravanje. Kako bi se to ostvarilo koriste se čuvari putanje *(eng. Route Guard)*, točnije opcija *CanActivate* koja sprječava pristup određenim komponentama sustava. Dio koda iz *app-routing.module.ts* koji je zadužen za povezivanje URL putanje i komponente:

```
const routes: Routes = [
  {
    path: '',
    component: DashboardComponent,
    canActivate: [AuthGuard]
  },
  ...
  ]
```

Ispis 4.4: Izvadak koda iz *app-routing.module.ts*

Možemo primijetiti kako se *canActivate* koristi kao funkcija iz *[AuthGuard]*, odnosno *auth.guard.ts* datoteke u kojoj se provjerava je li korisnik ulogiran, te ako nije šalje se na stranicu za ulogiravanje.

5. Generiranje certifikata

5.1. Digitalni certifikat

U kriptografiji certifikat je elektronički dokument koji se koristi kako bi se dokazao identitet, odnosno vlasništvo javnog ključa. Certifikat sadrži informacije o ključu, informaciju o identitetu vlasnika (subjekt) i digitalni potpis entiteta koji je potvrdio sadržaj certifikata (izdavač). U tipičnoj infrastrukturi javnog ključa, izdavač je nadležno tijelo za izdavanje certifikata (*Certificate Authority (CA)*). Standardni oblik zapisa digitalnih certifikata je X.509 v3. Primjer digitalnog certifikata u X.509 v3 obliku:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            98:33:c9:a8:00:00:00:00:56:54:bc:6e
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=HR, O=Financijska agencija, CN=Fina Root CA
        Validity
            Not Before: Nov 24 19:07:30 2015 GMT
            Not After : Nov 24 19:37:30 2035 GMT
        Subject: C=HR, O=Financijska agencija, CN=Fina Root CA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
                Modulus:
                    00:b4:73:d2:d6:65:d3:02:ef:58:3a:f4:82:82:d3:
                                                            ...
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Authority Key Identifier:
                keyid:FE:11:A2:6C:10:EE:DE:E2:03:B8:55:82:4E:22:3C:86:E4:3E:6B:54

            X509v3 Subject Key Identifier:
                FE:11:A2:6C:10:EE:DE:E2:03:B8:55:82:4E:22:3C:86:E4:3E:6B:54
    Signature Algorithm: sha256WithRSAEncryption
         a4:80:86:2e:e7:a6:9d:8d:ff:82:57:63:72:e7:1a:60:df:ab:
                                        ...
```

Ispis 5.1: Primjer digitalnog certifikata

Korištenjem digitalnih certifikata prilikom elektroničke komunikacije zadovoljavamo sljedeće osnovne zahtjeve:

- *Autentikacija* - proces kojim korisnik dokazuje da on uistinu je onaj za kojeg se predstavlja
- *Integritet* - sigurnost da su podaci u prijenosu ostali nepromijenjeni
- *Tajnost* - podaci u prijenosu su kriptirani i zaštićeni od čitanja neovlaštene osobe
- *Neporecivost* - onemogućava poricanje akcije koju je osoba poduzela ili autorizirala.

U ovom projektu digitalni certifikati služe kako bi korisnik mogao potpisati XML zahtjeve koji šalje blagajna.

5.2. Biblioteka cryptography

U praktičnom dijelu ovog rada generiranje certifikata ostvaruje se uporabom Python biblioteke *cryptography*. Kako bi napravili potpisani certifikat prvo moramo generirati zahtjev za potpisivanje certifikata (*Certificate Signing Request (CSR)*). To ćemo napraviti tako da generiramo RSA ključ koji će nam služiti kao privatni ključ, zatim koristeći *x509.CertificateSigningRequestBuilder()* "sagradimo" *CSR* dodajući potrebne atribute. Primjeri atributa:

```
x509.NameAttribute(NameOID.COUNTRY_NAME, "US"),
x509.NameAttribute(NameOID.STATE_OR_PROVINCE_NAME,
    "California"),
x509.NameAttribute(NameOID.LOCALITY_NAME, "San␣Francisco"),
x509.NameAttribute(NameOID.ORGANIZATION_NAME, "My␣Company"),
x509.NameAttribute(NameOID.COMMON_NAME, "mysite.com"),
```

Ispis 5.2: Primjer zadavanja atributa x509 certifikatu

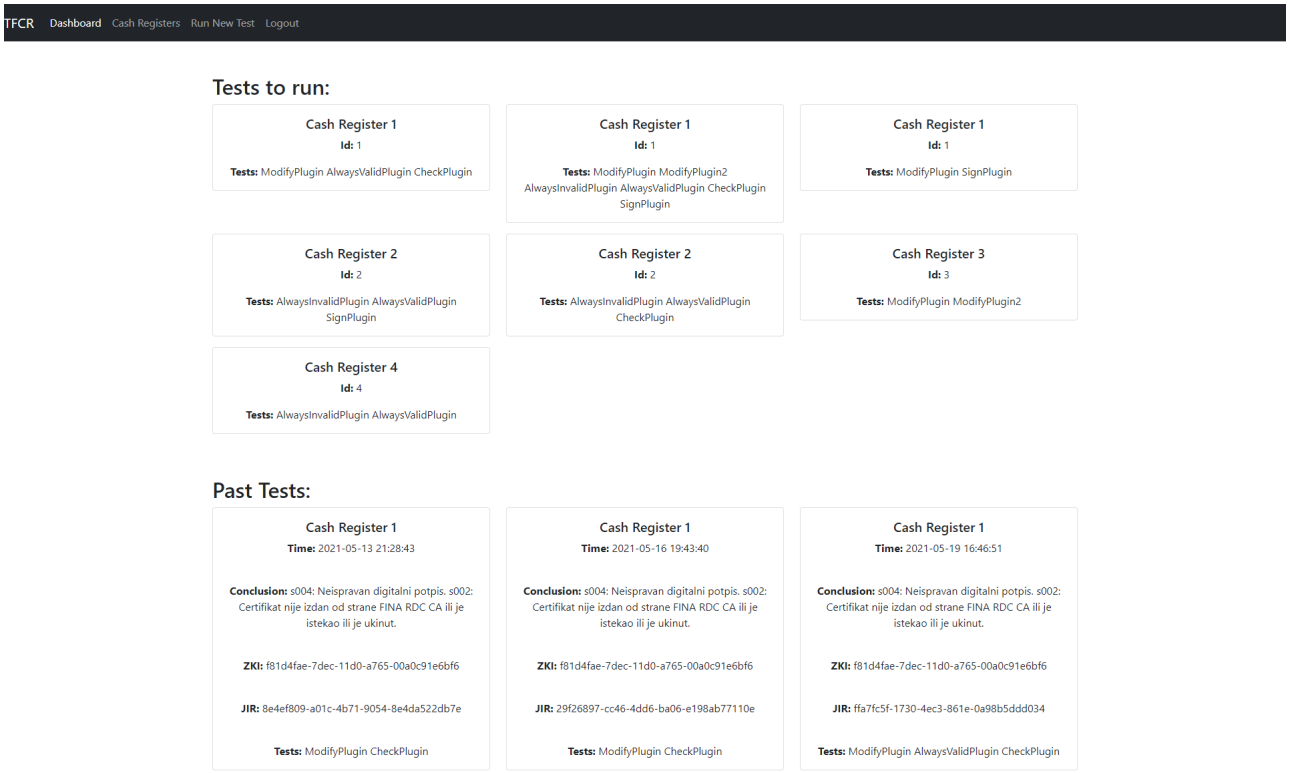
Takav *x509.CertificateSigningRequestBuilder()* potpišemo s funkcijom *sign* s generiranim RSA ključem kao argumentom.
Nakon što smo generirali *CSR* potpisujemo ga s već postojećim *CA* certifikatom.
Zip arhiva s potpisanim certifikatom, ključem i šifrom ključa se sprema u Django *STATIC_ROOT* folder, iz kojeg je korisnik preuzima.

6. Korištenje aplikacije

Nakon što se korisnik registrira u sustav, verificira adresu e-pošte i ulogira se u sustav na način opisan u **3. Autentifikacija korisnika**. Kao početna stranica otvorit će mu se nadzorna ploča.

6.1. Nadzorna ploča

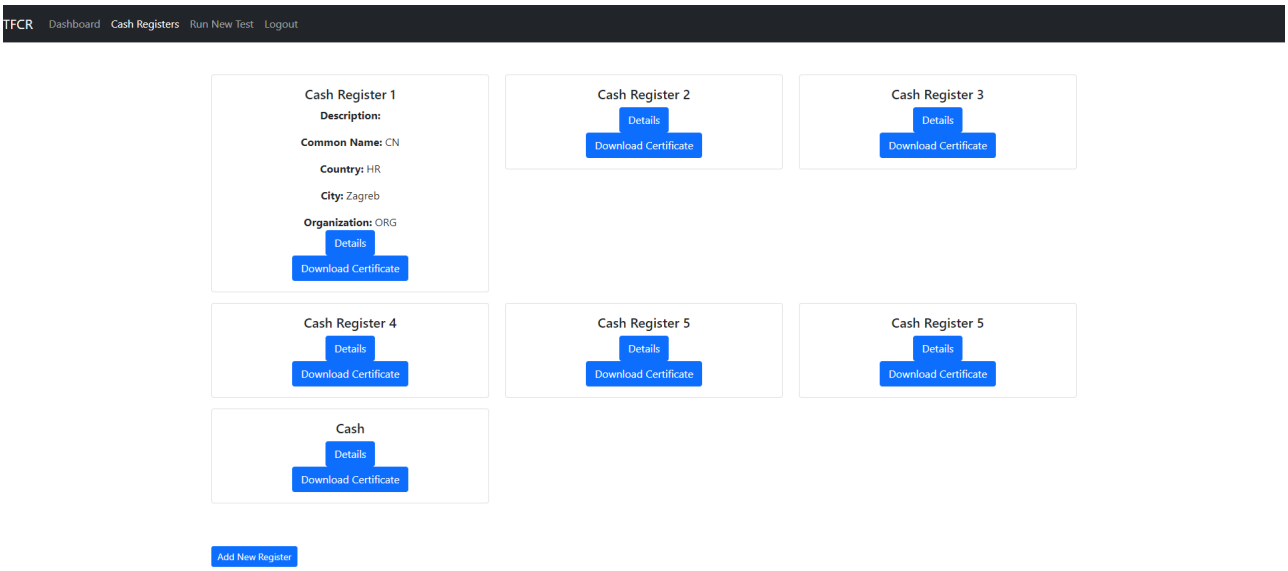
Na nadzornoj ploči korisnik može vidjeti popis novih testova koji još nisu pokrenuti, te već pokrenute testove sa njihovim zaključkom kao što je vidljivo na *Slici 6.1*



Slika 6.1: Nadzorna ploča

6.2. Fiskalne blagajne

U komponenti fiskalne blagajne korisnik može vidjeti sve postojeće fiskalne blagajne zajedno s njihovim detaljima čije certifikate može preuzeti pritiskom na gumb *Download Certificate*. Pritiskom na gumb *Add New Register* korisniku se omogućava registriranje nove fiskalne blagajne u sustav, tako da se pokaže forma s atributima fiskalne blagajne koje korisnik ispunjava. (Vidljivo na *Slika 6.3*)

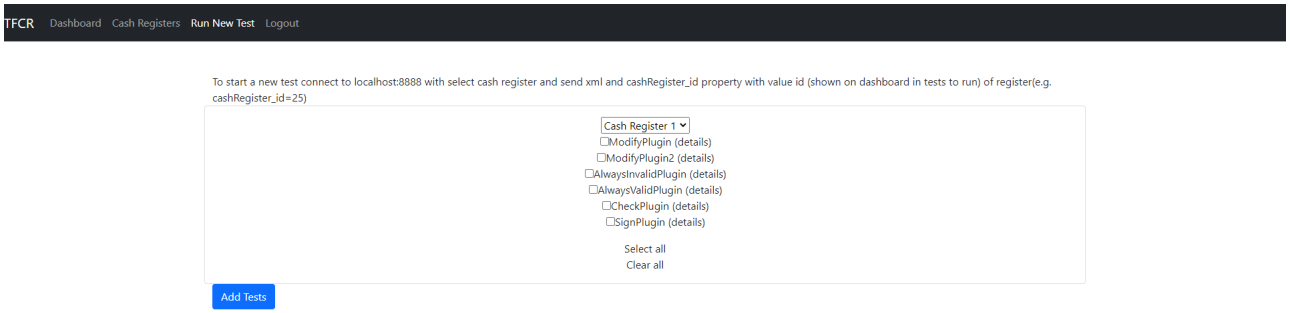


Slika 6.2: Popis fiskalnih blagajni

Slika 6.3: Dodavanje nove fiskalne blagajne

6.3. Pokretanje novog testa

Novo testiranje se pokreće tako da se u komponenti *Run New Test* odabere blagajna na kojoj se žele odraditi testiranje te se označe odabrani testovi. Pritiskom na gumb *Add Tests* se ti testovi spremaju u bazu podataka i spremni su za pokretanje.



Slika 6.4: Pokretanje novih testova

7. Zaključak

U ovom radu je opisana i ostvarena web aplikacija za testiranje fiskalnih blagajni koja se sastoji od klijentske web aplikacije pisane u Angular web okviru, servisnog sloja napravljenog u Django REST okviru te baze podataka.

S obzirom na to da nije poznato postoji li već ovakav sustav, moglo bi se zaključiti da bi mogao naći svoje mjesto na tržištu. Omogućava analiziranje i testiranje fiskalnih blagajni u čijoj je implementaciji vrlo lagano doći do pogreške.

Kao nadogradnja na već postojeći sustav mogla bi se dodatno ostvariti bolja automatizacija i lakše povezivanje fiskalne blagajne sa sustavom testiranja. Isto tako postoji mogućnost napretka u dizajnu korisničkog sučelja koje je trenutno vrlo jednostavno.

8. Literatura

[1] Angular web okvir: https://angular.io/docs pristupano 18.03.2021.

[2] Angular uvod: https://angular.io/guide/what-is-angular pristupano 18.03.2021

[3] Django web okvir: https://www.djangoproject.com/ pristupano 18.03.2021.

[4] Django REST okvir: https://www.django-rest-framework.org/ pristupano 02.04.2021.

[5] Django dodatak django-rest-authemail: https://github.com/celiao/django-rest-authemail pristupano 09.04.2021.

[6] Python Cryptograpy biblioteka: https://cryptography.io/en/latest/x509/tutorial/ pristupano 08.05.2021

[7] Sigurnost Django aplikacije: https://docs.djangoproject.com/en/3.2/topics/security/ pristupano 26.05.2021

Web aplikacija za upravljanje ispitivanjem fiskalnih blagajni

Sažetak

Fiskalne blagajne služe za provođenje sustava fiskalizacije, tako da se svaki izdani račun prijavi poreznoj upravi. U ovom sustavu se testira ispravnost fiskalnih blagajni. Ovaj sustav prihvaća zahtjeve fiskalne blagajne te ih manipulira i šalje fiskalnoj blagajni ispravne i neispravne odgovore. Promatranjem kako fiskalna blagajna reagira na takvo ponašanje mogu se utvrditi greške u implementaciji. Sustav se sastoji od poslužitelja, dodataka i web sustava.
U ovom radu detaljno je opisan način rada web sustava. Opisane su korištene tehnologije, arhitektura sustava, način autentificiranja korisnika, kako se generiraju certifikati potrebni za daljnje korištenje u sustavu, te sama uporaba web aplikacije.

Ključne riječi: Fiskalne blagajne, Porezna uprava, Angular, Django, Web aplikacija.

Web application for managing the testing of fiscal cash registers

Abstract

Fiscal cash registers are used to implement the fiscalization system, in a way that every issued invoice is reported to the tax administration. This system accepts requests made by the fiscal cash register and manipulates them and returns correct and incorrect responses. Observing how the cash register reacts to such behavior we can identify implementation errors. The system consists of a server, plugins and of web system.
This paper describes in detail how the web system works. It describes the technologies used, the system architecture, the method of user authentication, how the certificates required for further use in the systems are generated, and the use of the web application are described.

Keywords: Fiscal cash registers, Tax administration, Angular, Django, Web application.