Sadržaj

1. Uvod 2

2. Opis strukture i načina rada sustava 4

2.1. Struktura XML zahtjeva fiskalne blagajne i odgovora Porezne uprave 6

2.2. Struktura podataka za upravljanje sustavom dodataka 9

2.3. Popis korištenih tehnologija 11

2.4. Oblikovanje i primjena sustava dodataka 13

3. Provođenje ispitivanja fiskalne blagajne korištenjem sustava dodataka. 20

3.1. Provjera ispravnosti XML zahtjeva 22

3.2. Stvaranje XML odgovora 23

3.3. Promjena XML odgovora 25

3.4. Potpisivanje XML odgovora 26

3.5. Dodavanje vlastitog dodatka 29

3.6. Dodavanje vlastitih pogrešaka 30

4. Zaključak 32

Sažetak 34

Summary 35

5. Literatura 36

1. Uvod

Fiskalne su blagajne sustav za reguliranje i upravljanje izdavanja računa. Koriste određeni protokol kojim komuniciraju sa poreznom upravom. Pri komunikaciji, fiskalna blagajna poreznoj upravi šalje zahtjev za izdavanje fiskalnog računa, a porezna uprava odgovara slanjem odgovora za fiskalni račun. [1] Svaka fiskalna blagajna mora obavezno slati ispravne zahtjeve poreznoj upravi, i ispravno upravljati sa odgovorima koje porezna uprava izda.
Ovim radom, izrađen je sustav koji testira ispravnost fiskalne blagajne, simuliranjem porezne uprave. Konkretno, sustav omogućuje prihvaćanje i obradu zahtjeva od fiskalne blagajne te izdavanje odgovora fiskalnoj blagajni. Testiranjem fiskalnih blagajni žele se otkriti pogreške u izradi sustava koji registrira odgovore koje šalje porezna uprava. Slanjem neispravnih odgovora fiskalnoj blagajni, može se utvrditi kako fiskalna blagajna reagira na greške.

Ovaj rad bavi se organizacijom, strukturiranjem, upravljanjem i izradom sustava dodataka (eng.plug-in system). Ukratko, dodatak je određena implementacija neke funkcionalnosti. Glavna karakteristika sustava koji koristi dodatak jest da se neki dodatak može zamijeniti nekim drugim.
Rad je podjeljen na dvije glavne cjeline:

Prva cjelina zove se "Opis strukture i načina rada sustava". U ovoj cjelini je opisana struktura, kao i konkretna implementacija programske podrške. Cjelina se dijeli na četiri potpoglavlja. Prvo potpoglavlje glasi "Struktura XML zahtjeva fiskalne blagajne i odgovora Porezne uprave". Unutar ovog potpoglavlja, popisuju se specifikacije koje je propisala porezna uprava. Također, opisane su strukture XML zahtjeva i XML odgovora, koje korištenih u ovome projektu. Naziv drugog poglavlja je “ Struktura podataka za upravljanje sustavom dodataka”. U njemu je detaljno opisana struktura i raspodjela podataka unutar projekta. U trećem potpoglavlju ,naziva “Popis korištenih tehnologija”,naveden je korišteni programski jezik te svi radni okviri koji su korišteni u projektu. Za svaki radni okvir, navedena je njihova primjena i njihov opis. Četvrto potpoglavlje u prvoj cjelini je „Oblikovanje i primjena sustava dodataka“ nabrojane su vrste dodataka, opisane su strukture pojedine vrste dodataka i definirana je njihova uloga unutar sustava.

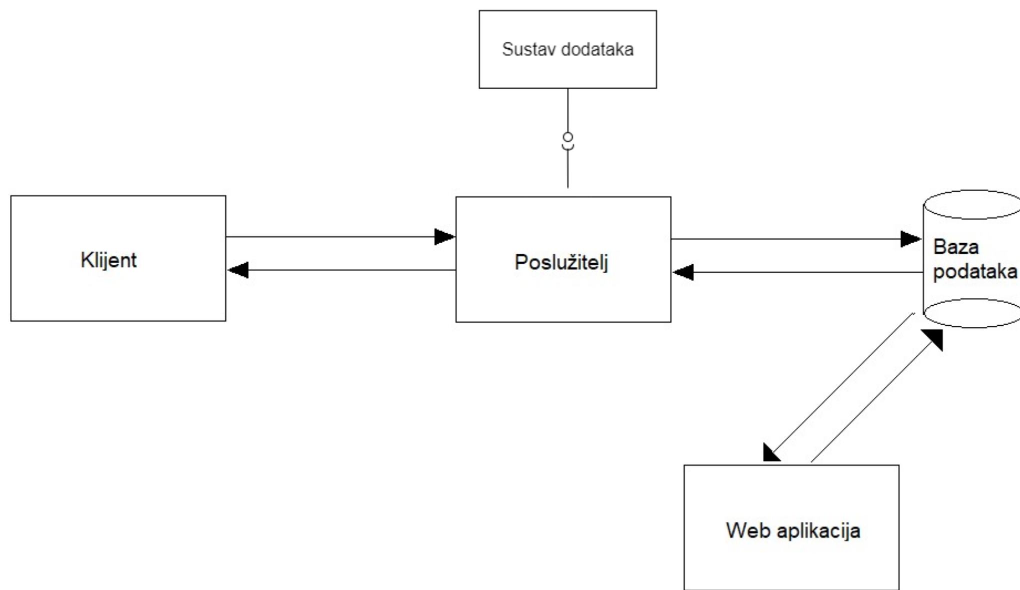
U drugoj cjelini, „Provođenje ispitivanja fiskalne blagajne korištenjem sustava dodataka“, predočeni su koraci za provođenje jednog ispitnog slučaja. Isto tako, navedene su upute za izradu

vlastitog dodatka i vlastite greške unutar sustava. Prva četiri potpoglavlja druge cjeline opisuju korake provođenja ispitivanja. Prvo potpoglavlje "Provjera ispravnosti XML zahtjeva", opisuje postupak provjere ispravnosti primljenog XML zahtjeva. Drugo potpoglavlje "Stvaranje XML odgovora", opisuje postupak izrade nepotpisanog XML odgovora temeljenog na ispravnosti XML zahtjeva. Treće potpoglavlje "Promjena XML odgovora", opisuje postupak izmjene novostvorenog XML odgovora. Četvrto potpoglavlje "Potpisivanje XML odgovora" opisuje postupak potpisivanja novostvorenog XML odgovora. Peto i šesto potpoglavlje, „Dodavanje vlastitog dodatka" i „Dodavanje vlastitih pogrešaka", navode upute za izradu vlastitog dodatka odnosno greške unutar sustava.

2. Opis strukture i načina rada sustava

Sustav implementira grafičku mrežnu stranicu za olakšani rad sa sustavom. Mrežna stranica pruža grafičko sučelje kojim se pružaju sljedeće funkcionalnosti:
registraciju i prijavu korisnika u sustav, unos nove fiskalne blagajne u sustav za testiranje i odabir novog ispitnog slučaja, čime se i biraju dodaci kojima se provodi ispitni slučaj, pregled rezultata svakog ispitnog slučaja.

Sustav je organiziran kao poslužitelj-klijent par aplikacija.
Klijent omogućuje slanje zahtjeva i služi za prikazivanje rada sustava. Klijenta je moguće zamijeniti sa fiskalnom blagajnom koju se želi testirati ili može poslužiti kao posrednik između poslužitelja i fiskalne blagajne koju se testira.
Poslužitelj simulira rad porezne uprave. Prima zahtjev fiskalne blagajne i stvara odgovor. Dani odgovor ne mora nužno biti ispravan niti odgovarajući za poslani zahtjev. Odgovor se može izmijeniti sustavom dodataka. Slanjem različitih odgovora, sustav može bilježiti ponašanje fiskalne blagajne i time ju testirati.
U našemu sustavu dodaci služe za upravljanje obrade zahtjeva i stvaranju odgovora, čime se povećava fleksibilnost cjelokupnog sustava. Time dodaci omogućuju sustavu prilagodbu različitim zahtjevima koje fiskalna blagajna šalje.
Pojedini dodatak omogućuje pojedinu funkcionalnost poslužitelja. To znači da se potencijalno mogu izmjeniti samo neke funkcionalnosti dok ostale ostaju iste.
Poslužitelj i Web aplikacija svoje podatke čuvaju unutar baze podataka. Web aplikacija čita potrebne podatke za prikaz web stranice i bilježi koje ispitne slučajeve je korisnik odabrao. Poslužitelj iz baze podataka čita ispitne slučajeve i sprema rezultate ispitivanja u bazu podataka.



Sl. 1 Prikaz strukture i međusobne komunikacije u sustavu

2.1. Struktura XML zahtjeva fiskalne blagajne i odgovora Porezne uprave

U ovome poglavlju opisana je XML struktura zahtjeva koje fiskalna blagajna šalje i odgovora koje fiskalna blagajna prima. Isto tako, nalaze se primjeri XML zahtjeva i odgovora. Detaljan opis nalazi se u [1].

Svi XML zahtjevi opisani su sa SOAP tehnologijom:

"SOAP protokol omogućuje komunikaciju između aplikacija koje rade na različitim operacijskim sustavima i različitim tehnologijama. Aplikacije razmjenjuju poruke dogovorenog formata." [1]

XML zahtjev je struktura koja se sastoji od dva glavna dijela. Prvo, „Zaglavlje" je dio dokumenta u kojem se nalaze ZKI (zaštitni kod izdavatelja) u polju „IdPoruke". Drugo, unutar „Racun" polja nalazi se „Oib". Ove dvije vrijednosti (zajedno sa digitalnim potpisom) koriste se za bilježenje i verifikaciju pojedinog XML zahtjeva.

Pregled primjera XML zahtjeva preuzetog iz [1]:


```xml
<?XML version="1.0" encoding="UTF-8"?>
<soapenv:Envelope XMLns:soapenv="http://schemas.XMLsoap.org/soap/envelope/">
<soapenv:Body>
<tns:RacunZahtjev XMLns:tns="http://www.apis-it.hr/fin/2012/types/f73"
XMLns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.apis-it.hr/fin/2012/types/f73
../schema/FiskalizacijaSchema.xsd">
<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:Oib>98765432198</tns:Oib>
<tns:USustPdv>true</tns:USustPdv>
<tns:DatVrijeme>01.09.2012T21:10:34</tns:DatVrijeme>
<tns:OznSlijed>P</tns:OznSlijed>
<tns:BrRac>
<tns:BrOznRac>123456789</tns:BrOznRac>
<tns:OznPosPr>POSL1</tns:OznPosPr>
<tns:OznNapUr>12</tns:OznNapUr>
</tns:BrRac>
<tns:Pdv>
<tns:Porez>
<tns:Stopa>25.00</tns:Stopa>
<tns:Osnovica>10.00</tns:Osnovica>
<tns:Iznos>2.50</tns:Iznos>
</tns:Porez>
<tns:Porez>
<tns:Stopa>10.00</tns:Stopa>
```

```
<tns:Osnovica>10.00</tns:Osnovica>
<tns:Iznos>1.00</tns:Iznos>
</tns:Porez>
<tns:Porez>
<tns:Stopa>0.00</tns:Stopa>
<tns:Osnovica>10.00</tns:Osnovica>
<tns:Iznos>0.00</tns:Iznos>
</tns:Porez>
</tns:Pdv>
<tns:Pnp>
<tns:Porez>
<tns:Stopa>3.00</tns:Stopa>
<tns:Osnovica>10.00</tns:Osnovica>
<tns:Iznos>0.30</tns:Iznos>
</tns:Porez>
</tns:Pnp>
<tns:OstaliPor>
<tns:Porez>
<tns:Naziv>Porez na luksuz</tns:Naziv>
<tns:Stopa>15.00</tns:Stopa>
<tns:Osnovica>10.00</tns:Osnovica>
<tns:Iznos>1.50</tns:Iznos>
</tns:Porez>
```

XML 1 Primjer zahtjeva

XML odgovor ima sličnu strukturu kao i zahtjev. Glavna je razlika u tome što XML odgovor ne sadrži detalje računa, a može sadržavati ili JIR (Jedinstveni identifikator računa) ili popis grešaka. XML odgovor će imati polje "JIR" ako sustav smatra da je primljeni zahtjev ispravan, u suprotnom , sustav će sadržavati polje "Greske" koje u sebi sadrži parove "SifraGreske" i "PorukaGreske" za svaku pronađenu grešku.

Primjer XML odgovora bez grešaka:

```
<?XML version="1.0" encoding="UTF-8"?>
<soap:Envelope XMLns:soap="http://schemas.XMLsoap.org/soap/envelope/"
XMLns:xsd="http://www.w3.org/2001/XMLSchema"
XMLns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soap:Body>
<tns:RacunOdgovor
xsi:schemaLocation="http://www.apis-it.hr/fin/2012/types/f73
../schema/FiskalizacijaSchema.xsd "
XMLns:tns="http://www.apis-it.hr/fin/2012/types/f73">
<tns:Zaglavlje>
<tns:IdPoruke>733362f1-063f-11e2-892e-0800200c9a66</tns:IdPoruke>
<tns:DatumVrijeme>01.09.2012T21:10:38</tns:DatumVrijeme>
</tns:Zaglavlje>
<tns:Jir>6b7749c6-56c1-4cf5-b7f7-9f29cebc9f7f</tns:Jir>
</tns:RacunOdgovor>
```

```
</soap:Body>
</soap:Envelope>
```

XML 2 Primjer odgovora bez grešaka

Primjer XML odgovora sa greškama:

```
<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:Greske>
<tns:Greska>
<tns:SifraGreske>s002</tns:SifraGreske>
<tns:PorukaGreske>Certifikat nije izdan od strane FINA RDC CA ili je istekao
ili je
ukinut.</tns:PorukaGreske>
</tns:Greska>
</tns:Greske>
</tns:RacunOdgovor>
```

XML 3 Primjer odgovora sa greškama

Skup svih pogrešaka

Unutar tehničke specifikacije definirane su moguće pogreške koje XML odgovor može sadržavati. Te pogreške registrira Porezna uprava. U ovome sustavu to obavlja skup dodataka iz skupa CheckPlugins (više u poglavlju „Oblikovanje i primjena sustava dodataka“).

Ukupno, definirana je sljedeća tablica pogrešaka [1]:

Šifra greške	**Poruka 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 1 Šifrarnik grešaka

2.2. Struktura podataka za upravljanje sustavom dodataka

U ovom poglavlju opisuju se svi direktoriji i poddirektoriji koji su stvoreni u svrhu upravljanja sustavom dodataka.

PluginSystem je najvažniji direktorij za upravljanje dodacima. Unutar PluginSystem direktorija nalaze se tri poddirektorija:

- CheckPlugins
- ModifyPlugins
- SignPlugins

Svaki od direktorija sadrži jednu vrstu dodatka. CheckPlugins sadrži dodatke za provjeru ispravnosti XML datoteke. ModifyPlugins sadrži dodatke za izmjenu XML datoteke, a SignPlugins za potpisivanje nepotpisane XML datoteke (više o implementaciji pojedinih dodataka unutar spomenutih direktorija može se pronaći u lekciji „Oblikovanje i primjena sustava dodataka").

PluginSystem još ima i Python datoteke koje se koriste za upravljanje dodataka:

- TestCase.py – razred za bilježenje uporabe svih dodataka
- testCaseHandler.py – upravljanje objektima instance TestCase razreda
- responseGenerator.py – stvaranje novog XML potpisa

Detaljniji opis i uporaba tih Python datoteka opisano je u poglavlju „Provođenje ispitivanja fiskalne blagajne korištenjem sustava dodataka".

Skripte unutar PluginSystem direktorija i njihovih poddirektorija koriste pomoćne strukture iz dva direktorija; Utils direktorij i Data direktorij.

Utils direktorij sadrži četiri skripte:

- utilityFunctions.py
- defaultResponse.py
- greskaDictionary.py
- plugin_controller.py

Prvo, utilityFunctions.py je skripta koja posjeduje jednu funkciju, printXML. Ovo je jednostavna funkcija koja se koristi za ispisivanje XML dokumenta na standardni izlaz:

```
def printXML(dom):

    print(ElementTree.tostring(dom, pretty_print="true"))
```

Drugo defaultResponse.py skripta sadrži popis elemenata XML odgovora. Ova skripta sadrži parametre kojima responseGenerator.py skripta stvara XML odgovor. Unutar defaultResponse.py datoteke, definirani su Label i Element razredi. Oni služe kako bi se stvorili objekti koji se onda kao liste prenose u responseGenerator.py.

Treće, greskaDictionary.py. Ovo je jednostavan rječnik. Ova skripta sadrži parove šifra:opis za svih šest vrsta pogrešaka definiranih u tehničkoj specifikaciji projekta.

Četvrto, plugin_controller.py je skripta koja sadrži nekoliko funkcija za povezivanje sustava dodataka sa serverom. Unutar plugin_controller.py postoje funkcije:

- `check_output`
- `updatedb`
- `getplugins`

`check_output` funkcija čita iz naše baze podataka. Preciznije, uzima poslani zahtjev iz kojega čita ZKI i JIR. Isto tako, provjerava koji su dodaci odabrani. Odabrane dodatke zki i jir predaje funkciji runTestCase (više u poglavlju Provođenje ispitivanja). RunTestCase.py funkcija tada vraća XML odgovor. Check_output funkcija tada čita primljeni XML i poziva sljedeću funkciju u plugin_controlleru, updatedb. Updatedb funkcija uzima dani XML dokument, parsira i sprema ga u bazu podataka. Za parsiranje koristi razred ConfigParser() iz radnog okvira configparser. Zadnja funkcija je getplugins.py. Ovo je jednostavna struktura kojom se čitaju svi dodaci iz baze podataka.

Data direktorij sadrži sve potrebne informacije za ispravan rad i testiranje rada sveukupnog sustava. Direktorij Data sadrži poddirektorij RequestExamples. RequestExamples sadrži primjere XML zahtjeva iz te correctData.XML i correctDataWithSig. [1] XML kao primjere odgovora koje sustav može stvoriti. Direktorij Data isto tako sadrži newcert.pem i newkey.pem. Ta dva dokumenta služe kao certifikat odnosno ključ za potpisivanje dokumenata.

2.3. Popis korištenih tehnologija

U ovom poglavlju se razmatraju svi alati koji su korišteni u izradi projekta.

Python programski jezik

Programski jezik Python služio je kao sveobuhvatan temelj izrade projekta. Preporuča se uporaba Python verzije 3.8 ili kasnijih inačica za ispravan rad sustava.

LXML knjižnica

LXML je knjižnica koja procesiranje XML i html dokumenata čini mogućim. [3] LXML je uveden za rad sa XML dokumentima. Korištena su dva razreda iz XML radnog okvira, a to su: XMLSigner i XMLVerifier.

XMLSigner razred ima implementirane funkcionalnosti za potpisivanje predanog XML dokumenata.[3]

XMLVerifier razred ima implementirane funkcionalnosti kojima provjerava potpis predanog XML dokumenta.[3]

Angular platforma

Angular je platforma koja služi za izradu web stranica i mobilnih aplikacija. [4] U ovome projektu Angular je korišten za izradu grafičkog slučelja kojim korisnik povezuje fiskalne blagajne i provodi ispitivanje.

Django radni okvir

Django je radni okvir koji služi za razvoj web stranica.[5] U sustavu se koristi za obradu podataka. Konkretno, služi za komunikaciju sa bazom podataka i dohvat relevantnih podataka kako bi se mogli prikazati na web stranici.

Tornado radni okvir

Tornado je radni okvir za asinkronu organizaciju parova klijenta i poslužitelja.[6] Tornado se koristi za povezivanje više klijenata na server. Nakon toga, na serveru se pokreće ispitni slučaj i rezultat ispita se šalje natrag klijentu.

Postgresql sustav

Postgresql je sustav baze podataka. [7] Baza podataka komunicira i sa klijentom i sa poslužiteljem te bilježi potrebne podatke i vraća tražene podatke.

Pluggy radni okvir

Pluggy je radni okvir izrađen za organiziranje svih dodataka u nekome projektu. Preuzeto je sa službene web-stranice za pluggy radni okvir u kojem piše opis pluggy sustava: [2]

„Pluggy is the crystallized core of plugin management and hook calling for pytest.“

Isto tako funkcionalnost je definirana:

„It gives users the ability to extend or modify the behaviour of a host program by installing a plugin for that program. The plugin code will run as part of normal program execution, changing or enhancing certain aspects of it.“

Pluggy u ovome projektu služi za specificiranje i implementiranje funkcionalnosti provjere zahtjeva, izmjene i potpisivanje odgovora. Pruža sučelje kojim se unutar definiranog projekta stvaraju razredi i njima dodijele funkcije.

Sastoji se od 3 glavna razreda:

- PluginSpec: služi za definiranje prototipa svih funkcija i određivanja kojem projektu pripadaju
- PluginImplementation: služi za definiranje implementacije funkcija koje su definirane u PluginSpec klasi
- PluginManager: služi za upravljanje Plugin sustavom.

Prije korištenja Pluggy sustava, prvo se moraju inicijalizirati pojedine vrijednosti.

Potrebno je stvoriti:

1. Inicijalnu strukuturu za hooks; ovime se zadaje ime projekta

   preferira se da se strukture stave u zasebnu datoteku:

   ```
   hookimpl = pluggy.HookimplMarker("pluggyGuide")

   hookspec = pluggy.HookspecMarker("pluggyGuide")
   ```

2. Razred za definiranje specifikacije; primjer konkretnog korištenja ovog sustava je naveden u poglavlju “Oblikovanje i primjena sustava dodataka” :

   ```
   class PluginSpec:
   ```

```
        @hookspec
        def verify(self, s: str):
            """Verify the given string"""
        @hookspec
        def validate(self, s: str):
            """Validate the given string"""
```

Kod 1 Oblikovanje specifikacije dodatka

3. Razred kojime se definira određena implementacija sustava. Sva funkcionalnost se ostvaruje unutar ovog razreda.

```
class PluginImplementation:
        @hookimpl
    def verify(self, s):
            print("Verify  " + s)
        @hookimpl
        def validate(self, s):
            print("Validate 2" + s)
```

4. Instanca PluginManager objekta:

```
pm = pluggy.PluginManager("pluggyGuide")
pm.add_hookspecs(PluginSpec())
pm.load_setuptools_entrypoints("pluggyGuide")
pm.register(PluginImplementation())
pm.hook.verify(s="Test")
pm.hook.validate(s="Test")
```

Kod 2 Instanciranje PluginManager objekta

2.4. Oblikovanje i primjena sustava dodataka

U ovome radu korišten je sustav dodataka kojim se omogućuje promjena funkcionalnosti poslužitelja. Sustav dodataka prati sve navedene korake navedene u poglavlju „Pluggy radni okvir“.

Inicijalizacija sustava dodataka

Prvi korak je inicijalizacija strukture za hooks, dok drugi korak definira razrede za specifikaciju. U ovome projektu to je napravljeno unutar hookInital.py datoteke:

```
hookimpl = pluggy.HookimplMarker("fiscal")

hookspec = pluggy.HookspecMarker("fiscal")
```

Opisivanje specifikacije pojedine grupe dodataka.

Drugi korak, primjer je dan za ModifySpec, ali analogno vrijedi za CheckSpec (verify umjesto modify) i SignSpec (sign umjesto modify):

```
class ModifySpec:

    @hookspec

    def name(self):

        """ Return the name of the the current plugin"""

    @hookspec

    def description(self):

        """ Return a short description of the functionality of the
plugin"""

    @hookspec

    def modify(self, XML):

        """Modify the XML file of the given root """
```

Kod 3 Specifikacija ModifySpec strukture

Implementacija funkcionalnosti za pojedini dodatak

Za treći korak, sustav implementira tri funkcionalnosti, stoga je sustav dodataka podijeljen u tri grupe. Svaka grupa omogućuje poslužitelju jednu funkcionalnost:

- Plugin za provjere: provjerava ispravnost XML dokumenta
- Plugin za izmjene: izmjenjuje dio XML dokumenta
- Plugin za potpisivanje: potpisuje dani nepotpisani dokument

Vrsta dodatka	Vrsta XML dokumenta	Direktorij	Naziv razreda
Plugin za provjere	XML zahtjev	CheckPlugins	CheckImplementation
Plugin za izmjene	XML odgovor	ModifyPlugins	ModifyImplementation
Plugin za potpisivanje	XML odgovor	SignPlugins	SignImplementation

Tablica 2 Vrste pluginova

1. Provjera zahtjeva koju je poslala fiskalna blagajna. (CheckPlugins direktorij, CheckImplementation razred)

2 Modificiranje odgovora koju je responseGenerator.py stvorio. (ModifyPlugins direktorij, ModifyImplementation razred)

3 Potpisivanje odgovora. (SignPlugins direktorij, SignImplementation razred). Za svaku vrstu dodatka postoji poseban direktorij i poseban naziv razreda koju svaki dodatak mora imati radi uspješne integracije u sustav.

Svaka datoteka unutar pojedinog direktorija je proizvoljna (pod pretpostavkom da je Python skripta, odnosno završava sa .py), ali je struktura strogo definirana.

1. Provjera zahtjeva:

Ova komponenta provjerava bilo koji primljeni XML dokument (objekt tipa lXML.Etree). Zahtjevi se provjeravaju dodacima koji se nalaze u direktoriju CheckPlugins.

Korištenjem više Check dodataka odjednom, sustav će bilježiti sve pogreške koje su pronašli svi dodaci zajedno u skup. To znači da će funkcija proslijediti popis svih mogućih pogrešaka, ali se ista pogreška neće ponavljati u popisu.

Svaka datoteka koja se sprema u CheckPlugins direktorij unutar sebe mora imati sljedeću strukuturu:

i) Mora uvesti (import) sljedeće parametre:

```
import PluginSystem.hookInitial as hookInitial

from signXML import XMLVerifier
```

```
from lXML import etree
```

(ostali importovi su dopušteni ako nemaju konflikta sa navedenim)

ii) Mora sadržavati jedan razred sa imenom CheckImplementation:

```
class CheckImplementation
```

iii) Razred u sebi mora imati sljedeće funkcije koje su povezane sa hookinitial sustavom prema načinu rada pluggy radnog okvira

Funkcija koja provjerava zahtjev:

Ulaz u funkciju: XML dokument koji se testira

Izlaz funkcije: skup svih pogrešaka u dokumentu

Prototip: `def verify(self, XML: etree):`

Funkcija koja vraća naziv dodatka:

Ulaz u funkciju: ništa

Izlaz iz funkcije: String imena

Prototip: `def name(self):`

Funkcija koja vraća opis dodatka

Ulaz u funkciju: ništa

Izlaz iz funkcije: String opisa

Prototip: `def description(self):`

2. Modificiranje zahtjeva:

Ova komponenta prima jedan XML odgovor i vraća modificirani XML odgovor.

Korištenjem više Modify dodataka odjednom, sustav će bilježiti sve promijene koje sve implementacije naprave. Pri tome, ako dvije implementacije mijenjaju isti dio odgovora, sustav će zabilježiti zadnju promjenu. Interno će se provesti sve promjene, no u slučaju preklapanja, sustav će prebrisati prijašnje izmjene.

Svaka datoteka koja se sprema u ModifyPlugins direktorij unutar sebe mora imati sljedeću strukturu:

i) Mora uvesti (import) sljedeće parametre:

```
import PluginSystem.hookInitial as hookInitial
from lXML import etree
```

(ostali importovi su dopušteni ako nemaju konflikta sa navedenim)

ii) Mora sadržavati jedan razred sa imenom ModifyImplementation:

```
class ModifyImplementation:
```

iii) Razred u sebi mora imati sljedeće funkcije koje su povezane sa hookinitial sustavom prema načinu rada pluggy radnog okvira:

Funkcija koja izmjenjuje primljeni XML:

Ulaz u funkciju: XML dokument koji se izmjenjuje

Izlaz funkcije: izmijenjeni XML dokument

Prototip: `def modify(self, XML: etree):`

Funkcija koja vraća naziv dodatka

Ulaz u funkciju: ništa

Izlaz iz funkcije: String imena

Prototip: `def name(self):`

Funkcija koja vraća opis dodatka

Ulaz u funkciju: ništa

Izlaz iz funkcije: String opisa

Prototip: `def description(self):`

3. Potpisivanje zahtjeva

Ova komponenta prima jedan XML odgovor i vraća potpisani XML odgovor. Korištenje više Sign dodataka nije dopušteno. Unutar sustava može se odjednom koristiti samo jedan dodatak za potpisivanje.

Svaka datoteka koja se sprema u SignPlugins direktorij unutar sebe mora imati sljedeću strukuturu:

i) Mora uvesti (import) sljedeće parametre:

```
import PluginSystem.hookInitial as hookInitial
from signXML import XMLSigner
```

(ostali importovi su dopušteni ako nemaju konflikta sa navedenim)

ii) Mora sadržavati jedan razred sa imenom SignImplementation:

```
class SignImplementation:
```

iii) Razred u sebi mora imati sljedeće funkcije koje su povezane sa hookinitial sustavom prema načinu rada pluggy radnog okvira

Funkcija koja potpisuje primljeni XML:

Ulaz u funkciju: XML dokument koji se potpisuje

Izlaz funkcije: potpisani XML dokument

Prototip: `def sign(self, XML: etree):`

Funkcija koja vraća naziv dodatka

Ulaz u funkciju: ništa

Izlaz iz funkcije: String imena

Prototip: `def name(self):`

Funkcija koja vraća opis dodatka:

Ulaz u funkciju: ništa

Izlaz iz funkcije: String opisa

Prototip: `def description(self):`

4. Pokretanje testiranja

Četvrti i zadnji korak obavlja se u TestCase razredu unutar testCase.py.

Inicijalizacija PluginManager objekta:

Za vrijeme inicijalizacije TestCase objekta, stvaraju se tri PluginManager objekta. checkerManager, modifierManager i signerManager. Oni su zaduženi za upravljanje dodacima koji su implementirani sa CheckImplementation, ModifyImplementation odnosno SignImplementation razredima (korak 3).

Primjer za modifierManager, analogno za checkerManager (CheckSpec umjesto ModifySpec) i signerManager (SignSpec umjesto ModifySpec):

```
self.modifierManager = pluggy.PluginManager(project)

self.modifierManager.load_setuptools_entrypoints(project)

self.modifierManager.add_hookspecs(hookInitial.ModifySpec)
```

Nakon inicijalizacije PluginManager objekta, uporaba pojedine funkcije iz dodataka, vrši se pozivom funkcije:

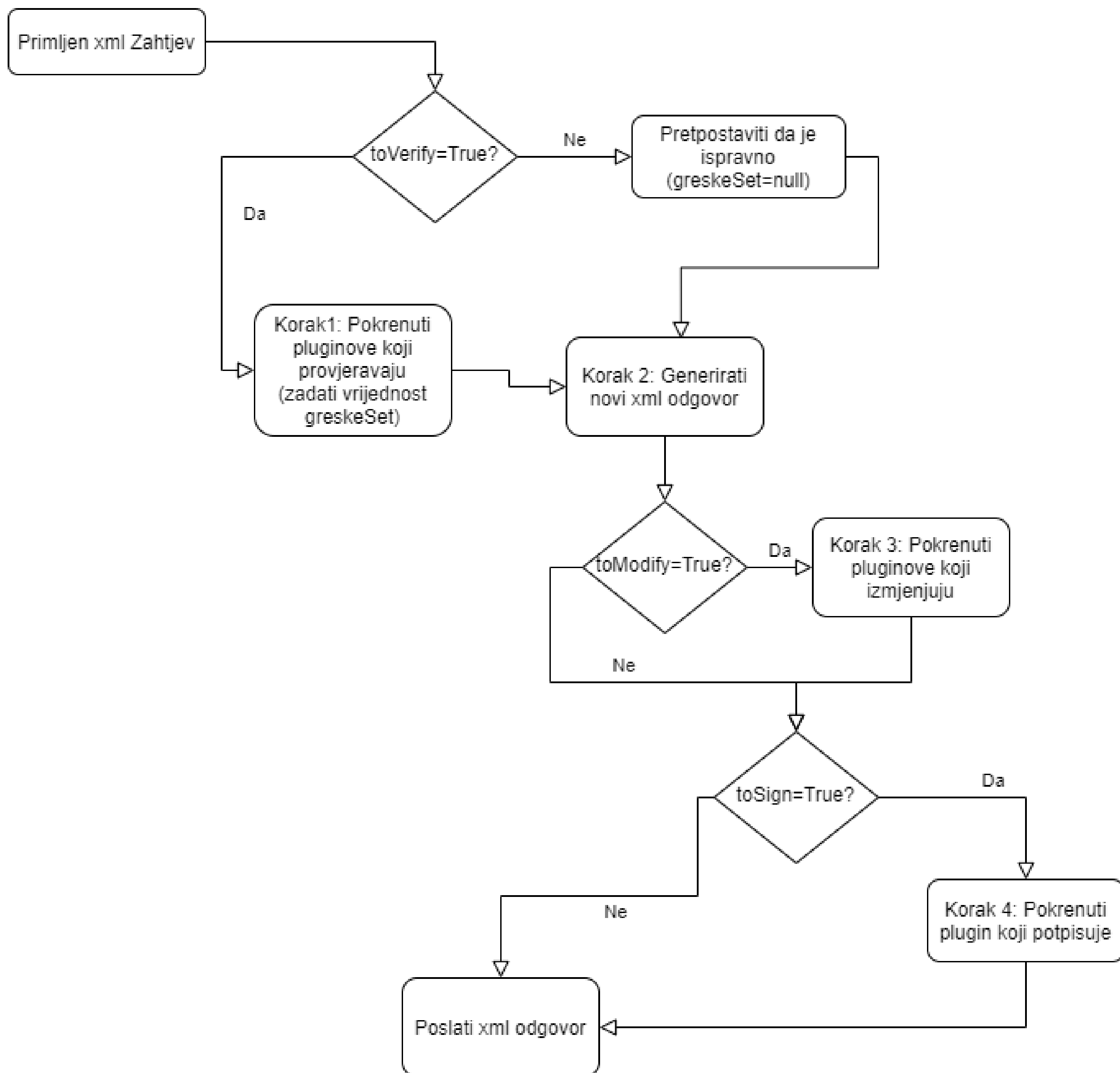
```
self.checkerManager.hook.verify(XML=self.dom)
```

Analogno za ostale objekte tipa PluginManager i funkcije unutar pojedine implementacije.

3. Provođenje ispitivanja fiskalne blagajne korištenjem sustava dodataka.

Ispitivanje se provodi tako da poslužitelj provodi jedan ispitni slučaj. Svaki ispitni slučaj provodi točno 4 koraka od kojih 3 koriste dodatke:

1. Provjera ispravnosti primljenog XML zahtjeva
2. Stvaranje XML odgovora (nepotpisani)
3. Promjena XML odgovora (modificiranje odgovora)
4. Potpisivanje XML odgovora (modificiranog)



Sl. 2 Dijagram toka provedbe ispitivanja

Korak 2 ne koristi sustav dodataka, razlog tome je što sustav mora uvijek generirati odgovor, a sve eventualne promjene mogu se ostvariti koristeći dodatke za izmjenu.

Koraci 1. 3. i 4. koriste dodatke i mogu biti preskočeni. Preskakanje se provodi pozivom `runTestCase` funkcije sa specifičnim parametrima (više o tome dalje u poglavlju).

3.1. Provjera ispravnosti XML zahtjeva

U ovome koraku, sustav koristi vrstu dodataka koji se nalaze u CheckPlugins direktoriju, i koji imaju implementiran razred CheckImplementation.

Ovaj korak provjerava je li primljeni zahtjev točno oblikovan, i sadrži li sve potrebne informacije. Svaki CheckImplementation razred koristi `verify(self, XML)` funkciju za verificiranje. Ova funkcija mora vratiti skup (set) grešaka. Svaka greška ima jednu šifru (s001, s002, s003, s004, s005, s006). Ako funkcija primijeti da primljeni zahtjev sadrži jednu ili više od spomenutih grešaka, onda mora poslati skup šifri kao izlaz. Ako verify sustav smatra da je sustav ispravan, onda može proslijediti prazan skup.

Primjer implementacije verify funkcije CheckImplementation razreda iz CheckPlugin datoteke:

```
def verify(self, XML: etree):

        greskeSet = set()

        try:

            parent = XML.find('X509IssuerSerial')

            child = parent.find('X509IssuerName')

            # if the document isn't signed or if it has the wrong 
certificate

            if not parent or (not child) or not child.find("O=FINA, 
C=HR"):

                greskeSet.add("s002")

        except AttributeError:

            greskeSet.add("s002")

    try:

           verified_data = XMLVerifier().verify(XML).signed_XML

        except Exception:

            greskeSet.add("s004")  # Neispravan digitalni potpis
```

```
        return greskeSet
```

Kod 4 Implementacija verify naredbe

Primjer implementacije verify funkcije CheckImplementation razreda iz AlwaysValidPlugin datoteke koja uvijek vraća prazan skup (time javlja sustavu da je zahtjev ispravan):

```
    def verify(self, XML: etree):
        greskeSet = set()
        return greskeSet
```

Primjer implementacije verify funkcije CheckImplementation razreda iz AlwaysInvalidPlugin datoteke koja uvijek vraća prazan skup (time javlja sustavu da je zahtjev neispravan, i da su se dogodile sve moguće greške):

```
    @hookInitial.hookimpl
    def verify(self, XML: etree):
        greskeSet = set(["s001", "s002", "s003", "s004", "s005", "s006"])
        return greskeSet
```

3.2. Stvaranje XML odgovora

Nakon ispitivanja zahtjeva, u sustavu se prosljeđuje skup grešaka. Taj skup grešaka koristi se za izradu novog XML odgovora. XML odgovor se izrađuje sa responseGenerator.py.

Za izradu XML odgovora, koristi se lXML radni okvir. Iz radnoga okvira, koriste se dva razreda:

- lXML.ElementTree – razred za izradu strukture XML odgovora
- lXML.ElementTree.Qname – razred za organizaciju prostora imena

Isto tako, unutar responseGenerator.py datoteke nalazi se razred zvan XMLNamespaces. Ovaj razred sadrži sva prostorna imena: njihov naziv i web link.

Za izradu strukture XML dokumenta poziva se funkcija generateOutputFile koja koristi sve spomenute razrede.

GenerateOutputFile funkcija prima tri parametara:

- greskeSet – skup grešaka koje stvaraju verify funkcije od dodatka za provjeru
- JIR – jedinstveni identifikator fiskalnog računa (izdaje Porezna uprava)
- IdPoruke – jedinstveni identifikator između zahtjeva i odgovora

Ovim parametrima funkcija izrađuje fiskalni odgovor prema specifikacijama Porezne uprave.

Poredak elemenata i njihovo nazivlje je definirano u.[1] Svi su odgovori strukturno slični i dijele se u dvije skupine. Oni kojima nije prenijet popis pogrešaka zahtjeva, i onima kojima je to napravljeno. Točnije, ovisno o ispravnosti zahtjeva koje je sustav primio, odgovori će sadržavati ili element „JIR“ ili „Greške“.

Isječak XML odgovora:

Za ispravni zahtjev (za prazni greskeSet):

```
...
            <tns:Jir>6b7749c6-56c1-4cf5-b7f7-9f29cebc9f7f</tns:Jir>
...
```

Za neispravan zahtjev (za greskeSet koji sadrži „s002“ i „s004“):

```
...
      <tns:Greske>
            <tns:Greska> "s002": "Certifikat nije izdan od strane FINA RDC CA
ili je istekao ili je ukinut.",</tns:Greska>
            <tns:Greska>"s004": "Neispravan digitalni potpis." </tns:Greska>
      <tns:Greske>
...
```

Za izradu početnog (root) elementa XML datoteke poziva se konstruktor ElementTree.Element().

Konstruktor razreda ElementTree.Element kao parametre prima:

- jedan QName objekt

- jedan rječnik svih prostora imena koje će SubElement razredi koristiti.

Konkretno naš se element inicijalizira sa:

```
Data = ElementTree.Element(QName(XMLNamespaces.soap, labelData.key),
nsmap=namespaces)
```

Svaki element osim `root` izrađuje se kao instanca ElementTree.SubElement razreda. Konstruktor ovog razreda kao parametre prihvaća:

- jedan ElementTree.Element ili ElementTree.SubElement objekt kao roditelj.
- jedan QName objekt
- jedan rječnik svih prostora imena koje će nasljednici ovog objekta koristiti

primjer inicijalizacije jednog ElementTree.SubElement objekta:

```
racunOdgovor = ElementTree.SubElement(body, QName(XMLNamespaces.tns,
labelRacunOdgovor.key), nsmap=tnsNameSpace)
```

3.3. Promjena XML odgovora

Svaki se XML odgovor može u potpunosti izmijeniti. Dodaci koji izmjenjuju imaju potpunu kontrolu nad XML odgovorom. To znači da bilo koja `modify()` funkcija može izmijeniti bilo koju vrijednost nekog elementa te čak dodati i izbaciti bilo koji broj elemenata.

Sustav će ispravno raditi dokle god `modify()` funkcija vrati bilo kakav XML odgovor (objekt tipa ElementTree).

U nastavku slijedi primjer u kojem se mijenja datum unutar XML odgovora:

Ova implementacija koristi ElementTree.Element find funkciju kako bi pronašla točan element unutar XML odgovora. Za ispravan rad funkcije, potrebno je zadati ispravno ime prostora. Najčešće je to tnsNameSpace.

Zadavanje prostora imena za pretragu:

```
tnsNameSpace  = {"tns": "http://www.apis-it.hr/fin/2012/types/f73"}
```

Traženje elementa za izmjenu i njegovog roditelja:

```
parent = XML.find('.//tns:Zaglavlje', tnsNameSpace)

currentData = parent.find('.//tns:DatumVrijeme', tnsNameSpace)
```

Kako bi sustav ispravno zamijenio trenutni element sa imenom, sustav mora isto tako pronaći roditelja traženog elementa. U ovome primjeru, traži se tns:Zaglavlje element koji je roditelj tns:Datum elementu. Potrebno je pronaći roditelja kako bi se mogla pozvati funkcija `ElementTree.Element.replace()` kojom se zamjenjuje trenutni element sa novostvorenim elementom.

Zamjena objekta (pretpostavlja se inicijalizacija replaceData objekta):

```
parent.replace(currentData, replaceData)
```

3.4. Potpisivanje XML odgovora

U zadnjem koraku potpisuje se generirani XML odgovor. Odgovor se potpisuje funkcijom sign(). Ova funkcija prihvaća nepotpisani XML odgovor i potpisuje ga. Odgovor može biti izmjenjen, ali i ne mora (prethodni korak je proizvoljan).

Potpisivanje se provodi putem XMLSigner razreda iz signXML knjižnice. Ovaj razred ne treba parametre za inicijalizaciju. XMLSigner sadrži funkciju `sign` koji prima tri parametra.

- XML = XML odgovor koji se potpisuje
- key = ključ koji se koristi za potpisivanje dokumenta
- case = certifikat kojim se potvrđuje dokument

```
signed_root = XMLSigner().sign(XML, key=key, cert=cert)
```

Poslužitelj poziva skriptu nazvanu testCaseHandler.py. Ova skirpta sadrži funkciju runTestCase koja stvara instancu razreda TestCase.

runTestCase preuzima sve parametre potrebne za provođenje jednog ispitnog slučaja:

- XML : XML zahtjev nad kojim se provodi ispitivanje
- modifyPlugins : lista dodataka za modificiranje odgovora
- checkPlugins : lista dodataka za provjeru odgovora

- signPlugin : dodatak kojim se potpisuje
- JIR: niz znakova za identifikaciju fiskalnog odgovora
- IDPoruke: niz znakova za identifikaciju fiskalnog zahtjeva
- toVerify: True/False vrijednost koja govori treba li provjeravati zahtjev
- toModify: True/False vrijednost koja govori treba li mijenjati odgovor
- toSign: True/False vrijednost koja govori treba li potpisati odgovor

runTestCase prati sljedeći princip rada:

U prvome koraku provjerava treba li verificirati:

```
greskeSet = set()

if toVerify:

    # add all errors found by all verify plugins

    for element in testCase.verify():

        print(element)

        greskeSet.add(element)

    if not greskeSet:

        print("File is correct")
```

Kod 5 Dio runTestCase koji provjerava verifikaciju

Ovime `runTestCase` poziva sve moguće dodatke koji verificiraju i od svih njih bilježi sve pogreške koje dodaci pošalju.

Ako dodaci ne pošalju ni jednu pogrešku, skup pogrešaka je prazan, i sustav zna da je primljeni zahtjev ispravan.

U suprotnom, ako dodaci pošalju barem jednu pogrešku, sustav zna da je primljeni zahtjev neispravan.

U drugome koraku runTestCase poziva funkciju generateOutputFile iz skripte responseGenerator.py

Rezultat se sprema u varijablu testCase objekta stvorenog unutar runTestCase funkcije:

```
testCase.dom = generateOutputFile(greskeSet, JIR, IdPoruke)
```

Ovime sustav priprema XML dokument za sljedeće korake : modificiranje i potpisivanje

U trećem koraku runTestCase funkcija provjerava treba li modificirati, te ako treba, dokument se modificira.

```
if toModify:
    print(testCase.modify())
```

Sustav ispisuje uspješnost modificiranja dokumenta na standardni izlaz.

Četvrti je korak sličan trećem; funkcija provjerava treba li potpisati dokument

```
if toSign:
    print(testCase.sign())
```

Sustav ispisuje uspješnost potpisivanja dokumenta na standardni izlaz.

U zadnjem koraku, funkcija vraća konačnu vrijednost XML dokumenta:

```
return testCase.dom
```

Ovu funkciju poziva poslužitelj. Nakon uspješnog poziva funkcije `runTestCase`, poslužitelj dobiva XML dokument kojeg treba proslijediti, a sustav na standardni izlaz dobiva povratne informacije o uspješnosti izvršavanja modify() i sign() funkcija te jesu li zadani dodaci pronašli pogreške u primljenom XML dokumentu.

Struktura TestCase razreda:

TestCase razred služi za sveobuhvatno upravljanje sustava dodataka.

Test case razred inicijalizira se sljedećim parametrima (svi parametri dobiveni su iz runTestCase funkcije):

- project: ime projekta, u našem slučaju „fiscal“
- dom : XML zahtjev
- modifyPlugins : lista dodataka za modificiranje odgovora
- checkPlugins : lista dodataka za provjeru odgovora
- signPlugin : dodatak kojim se potpisuje

Test Case razred tada čita imena svih dodataka koji se koriste i raspoređuje ih po njihovim ulogama.

Test Case razred ima četiri funkcije (izvedbe pojedine funkcije u nastavku):

- `listPlugins()`: ispisuje imena trenutnih dodataka na standardni izlaz
- `verify(XML)`: provjerava primljeni XML
- `modify(XML)`: mijenja strukturu/sadržaj primljenog XML-a
- `sign(XML)`: potpisuje primljeni XML
- `readDirectory(folderName)`: čita dodatak za dano ime datoteke (namijenjeno za interne pozive)

3.5. Dodavanje vlastitog dodatka

Dodavanje vlastitog dodatka radi se u tri koraka.

1. Odabir vrste dodataka koji se želi dodati
2. Izrada dodataka specifične strukture
3. Spremanje dodataka u ispravni direktorij

Prvo, postoje tri vrste dodatka. Svaki ima svoju preddefiniranu strukturu. (Oblikovanje i primjena sustava dodataka).

Drugo, sustav mora obavezno imati `name()` `description()` i jednu od sljedećih funkcija:

`verify()` za CheckImplementation, `modify()` za ModifyImplementation i `sign()` za SignImplementation.

Na kraju, funkcije se moraju spremiti u direktorij kojem pripada (CheckPlugins, ModifyPlugins i SignPlugins). Ako nije spremljen u ispravan direktorij, dodatak se neće koristiti.

Ako su praćeni svi koraci, sustav može koristiti dodatak. Dodatak se koristi tako da se unutar skripte testCaseHandler u funkciju `runTestCase()` predaju liste koje sadrže prikladni novostvoreni dodatak:

Na primjer, ako je stvoren novi check dodatak nazvan „CustomCheckPlugin“ i novi `sign` dodatak nazvan „CustomSignPlugin“:

String „CustomCheckPlugin“ mora se nalaziti u listi `checkPlugins` koja se predaje kao parametar `runTestCase()` funkcije.

String „CustomSignPlugin“ mora se predati kao parametar `signPlugin` za `runTestCase()` funkcije.

Treba pripaziti pri korištenju `runTestCase()` funkcije. Prima proizvoljnu veličinu liste za `checkPlugins` i „modifyPlugins“ listi, no samo JEDAN string za signPlugin (dokument se može samo jednom potpisati).

3.6. Dodavanje vlastitih pogrešaka

Popis šifri svih pogrešaka i pripadajućih opisa nalazi se u datoteci greskaDictionary.py unutar utils direktorija. Šifre i opis su spremljeni kao „dictionary“ struktura iz Python-a. Za dodavanje nove pogreške, potrebno je unijeti šifru (sifra) i opis (tekstualni opis) na sljedeći način:

„sifra“ : „tekstualni opis“

S obzirom da sustav koristi „dictionary“, potrebno je staviti zarez (,) nakon svih osim zadnjeg člana.

Sveukupno, nakon dodavanja, unutar datoteke bi trebalo pisati:

```
greskaDictionary = {
    "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.",
    "sifra" : "Tekstualni opis"
```

```
}
```

Kod 6 Knjižnica za greške (greskaDictionary)

Šifra je ovime dodana, no niti jedna verify funkcija neće ju još koristiti. Da bi se greška pravilno koristila, mora se unijeti ispravan uvjet unutar verify funkcije svakog dodataka za provjeru (svi oni unutar CheckPlugin direktorija). Za svaku grešku koja se želi koristiti, njezina šifra mora biti uključena u greskeSet skup koju `verify` funkcija vraća.

Na primjer, želi li se naša dodana greška **uvijek** ispisati, unutar verify funkcije može se upisati:

```
...

greskeSet.add(„sifra“)

...
```

4. Zaključak

Općenito, sustav koji koristi dodatke ima povećanu fleksibilnost u odnosu na projekt koji ih ne koristi. Pluggy radni okvir pruža zadovoljavajuću strukturu za izradu sustava dodataka. Izrada dodataka je jednostavna, no problem se javlja kod dodavanja pojedine implementacije dodataka. Naime, sustav treba nekako znati sav popis dodataka koji će se koristiti, ali mi želimo izraditi sustav kojem možemo dodati proizvoljan broj dodataka, svaki sa proizvoljnim imenom. U ovome projektu, to je riješeno kompromisom, svaka se skripta dodatka može zvati proizvoljno, no mora biti spremljena u ispravni direktorij i mora sadržavati ispravno oblikovani razred.

Sustav ne radi za zahtjeve koji se odnose na popratni dokument, niti ne odgovara na „echo" XML odgovore koje fiskalna blagajna može poslati. Sustav je dizajniran isključivo za zahtjev koji se ne odnosi na popratni dokument. Sustav isto tako šalje samo odgovor koji nije povezan ni sa kojim drugim dokumentom.

Postoje dva problema sa potpisivanjem u sustavu dodataka. Prvo, XML dokument ne može se više puta potpisati različitim potpisima. U buduće, sustav bi trebao omogućiti više različitih potpisa, čime bi se omogućilo još jedna vrsta testiranja fiskalne blagajne. Drugo, dani ključ i certifikat unutar „Data" direktorija nije ispravan. Za ispravan potpis, potreban je ispravan certifikat i njegov ključ, inače niti jedan XML odgovor koji se generira ne može biti ispravan.

Provjera pojedinog XML zahtjeva je vrlo rudimentarna. Sustav ne određuje nikakvu razliku između pojedinih vrsta grešaka. Točnije, dane implementacije prvo provjeravaju zahtjev za svaku moguću grešku, i onda šalju popis svih pronađenih grešaka. U kasnijim implementacijama, u sustav bi se mogla dodati jedna ili više implementacija koje, ako pronađu neku važniju grešku, odmah ju vraćaju, bez provjeravanja ostalih.

Unutar „Data" direktorija, unutar „RequestExamples" poddirektorija, moglo bi se dodati sve generirane primjere, kako bi se generirani odgovori mogli bilježiti i međusobno uspoređivati.

Sustav dodataka ne omogućava korisniku da izravno oblikuje XML odgovor. Ovime se održava struktura svakog ispitnog slučaja, no onemogućava korisniku da testira fiskalne blagajne izrađene za različite protokole. Međutim, ako korisnik želi, unutar sustava može izraditi dodatak koji bi

izmijenio cijeli XML odgovor. Time bi sustav generirao proizvoljan XML dokument, čime bi korisnik, neizravno, omogućio testiranje bilo kojeg sustava koji prima XML dokument kao parametar (ne samo fiskalne blagajne).

Struktura datotečnog sustava stvara i uzrokuje probleme u programskom jeziku Python. Princip unošenja funkcionalnosti iz drugih dijelova koda je otežan. Mora se koristit Python-ova sys knjižnica za koordiniranje pokazivača za ispravno pronalaženje datoteka. U budućnosti bi trebalo napraviti drugačiju strukturu datoteka, osmisliti bolji način međusobne komunikacije sučelja i skripti ili koristiti neki drugi programski jezik.

Sažetak

Ključne riječi: dodatak,pluggy radni okvir, xml, fiskalna blagajna, Porezna uprava, Python

Fiskalne su blagajne sustav za izdavanje fiskalnog računa. Da bi fiskalna blagajna izdala pojedini račun, ona mora komunicirati sa Poreznom upravom. Izrađen je sustav kojim se ispituje rad fiskalne blagajne. Sustav emulira Poreznu upravu, primajući zahtjeve i šaljući odgovore. Sustav može poslati ispravne, ali i neispravne odgovore na zahtjev fiskalne blagajne. Ispitivanje se provodi promatranjem kako fiskalna blagajna reagira na dane ispravne i neispravne odgovore.

Sustav je implementiran u tri dijela. Prvo, web stranica kojom se korisnik može registrirati u sustav i povezati fiskalnu blagajnu. Drugi dio sustava bavi se povezivanjem web stranice sa "backend" dijelom sustava. Sustav je organiziran kao par klijent, poslužitelj.

Ovaj rad bavi se promatranjem zadnjega dijela sveukupnog sustava. Opisana je tehnologija kojom je sustav izrađen, koraci provedbe samog ispitivanja (provjera, izrada odgovora, izmjena odgovora i digitalno potpisivanje dokumenta) i način izrade pojedinog dodatka. Objašnjen je način provedbe pojedinog ispitnog slučaja. Isto tako, opisan je način izrade i korištenja pojedinog plugina.

Summary

Keywords: plugin,pluggy framework, xml, fiscal register, tax administration, Python

Fiscal registers are a system for issuing fiscal bills. For a fiscal register to properly issue a bill, it must communicate with the tax administration. A system was made to test the workings of a fiscal register. The system emulates the tax administration, accepting requests and sending responses. The system can send correct, but also incorrect responses for any request. The testing is conducted by observing how the fical register reacts to the given correct and incorrect responses.

The system is implemented in three parts. First, a web site with which a user can register into the system and connect a fiscal register. The second part of the system handles the connection of the web site with the "backend" part of the system. The system is implemented as a client-server pair.

This paper deals with the examination of the last part of the complete system. It describes the tehnology with which the system was made, the steps of conducting the testing (verification, request generation, modification and digital signing of documents) and and how to make a particular plug-in. The method of conducting an individual test case is explained. In addition, the method of making and using an individual plug-in is described.

5. Literatura

[1] Republika Hrvatska, Porezna uprava,APIS IT: Fiskalizacija - Tehnička specifikacija zakorisnike Verzija 2.0

[2] Pluggy radni okvir: https://pluggy.readthedocs.io/en/latest/ preuzeto 19.5.2021 u (12:54)

[3] LXML knjižnica: https://lXML.de/ preuzeto 26.5.2021 u (20:50)

[4] Angluar platforma: https://angular.io/docs/ preuzeto 26.5.2021 u (20:38)

[5] Django radni okvir : https://www.djangoproject.com/ preuzeto 26.5.2021 u (20:49)

[6] Tornado radni okvir: https://www.tornadoweb.org/en/stable/ preuzeto 26.5.2021. u (20:50)

[7] Postrgresql sučelje: https://www.postgresql.org/about/ preuzeto 26.5.2021 u (21:01)