UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA U
NOVOM SADU

Igor Mandić

Komponentni razvoj softvera, platforma i moduli

DIPLOMSKI RAD
- Osnovne akademske studije -

UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
NOVI SAD
Odsek / smer / usmerenje:
Računarstvo i automatika / Računarske nauke i informatika

DIPLOMSKI - BEČELOR RAD

Kandidat: Mandić, Igor
Broj indeksa: RA 30/2013

Tema rada: **Komponentni razvoj softvera, platforma i moduli**

Mentor rada: dr Igor Dejanović

Mesto i datum:
Novi Sad, 2018.

Sadržaj

1 Uvod

Tema ovog rada je opis načina funkcionisanja, te implementacije ***ExPreSsiVeNess*** (*Extensible Platform for Structure Visualization and Navigation*) platforme, njenih osnovnih modula, te, modula za preuzimanje sadržaja sa *phonoApi web*-stranice.

Projekat je urađen poštujući osobine komponentnog razvoja softvera koji će kasnije biti detaljnije objašnjen. Alat koji je korišćen za spajanje platfome, modula i *frameworka* (eng. radni okvir) je *SetupTools*.

Platforma je napisana u programskom jeziku *Python*, te koristi *Django framework* kao radni okvir.

Moduli su takođe pisani korištenjem *Python* programskog jezika, sa primjesama *JavaScripta* i tehnologija vezanih za *JavaScript*.

Osnovne funkcionalnosti ovog projekta su:

- Učitavanje različitih tipova podataka sa različitih izvora te njihovo oblikovanje u strukturu tipa grafa
- Smještanje i čuvanje podataka
- Prikaz podataka u vidu grafa i grafa tipa stabla

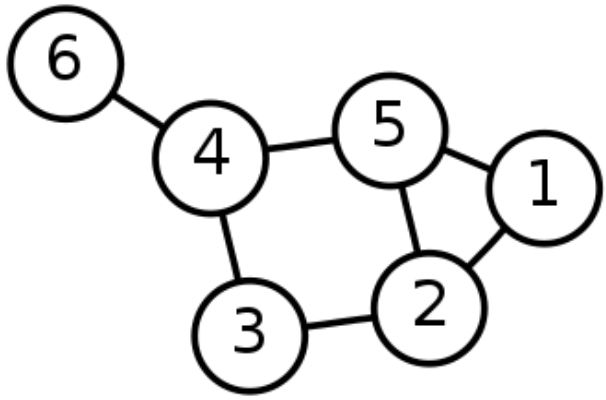
U radu će biti prezentovani:

- Teorija grafa
- Teorija grafa tipa stabla
- Komponentni razvoj softvera
- Kratak opis tehnologija korištenih u projektu
- Opis načina rada i implementacije platforme
- Kratak opis standardnih modula
- Opis načina rada i implementacije modula za prikupljanje podataka sa f*onoApi web*-stranice
- Primjer korišćenja

2 Teorija Grafova

2.1 Graf - definicija, osobine i tipovi

U matematici i teoriji grafova, graf je struktura sastavljena iz objekata u kome su parovi objekata povezani [1]. Objekti korespondiraju sa matematičkim apstrakcijama zvanim čvorovi i svi povezani parovi čvorova se nazivaju grane. Grane mogu biti usmjerene ili neusmjerene. Primjer grafa dat je na slici 2.1.1.



Slika 2.1.1. primjer grafa [2]

Definicija:

Graf je uređen par G = (V, E) koji obuhvata skup čvorova označenih sa V zajedno sa skupom grana označenih sa E. Skup grana E predstavlja neuređen par čvorova iz skupa V [1].

Neki od osnovnih tipova grafova [1]:

- Neusmjereni graf - graf kod koga grane nemaju usmjerenje (orijentaciju)
- Usmjereni graf - graf kod koga grane imaju usmjerenje (orijentaciju)
- Mješoviti graf - graf kod koga neke grane imaju usmjerenje a neke ne
- Multi-graf - graf kod koga je dozvoljeno kreiranje ciklusa ili više grana koje povezuju iste čvorove
- Prost graf - graf kod koga nije dozvoljeno kreiranje ciklusa i više grana koje povezuju iste čvorove

Neki od osnovnih klasa grafova [1] :

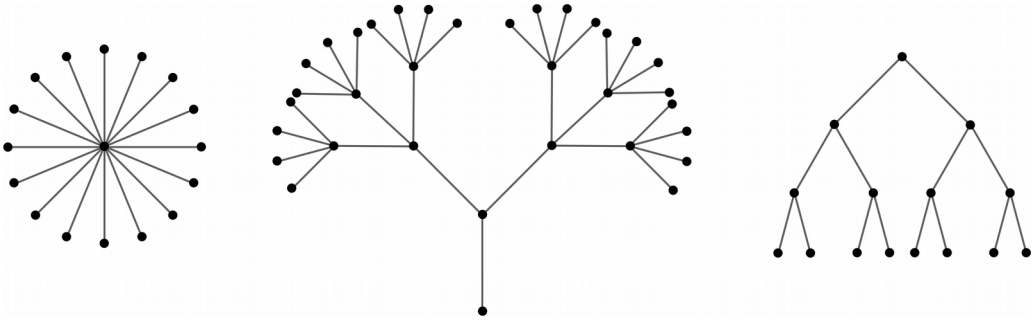
- Regularni graf - graf kod koga svaki čvor ima jednak broj susjednih čvorova
- Kompletan graf - graf kod koga je svaki par čvorova povezan
- Povezani graf - neusmjereni graf sa bilo kojim neuređenim parom čvorova {x,y} koji posjeduje putanju od x do y
- Bipartitni graf - graf kod koga je skup čvorova podjeljen u dva skupa W i X, takav da ne postoje dva čvora iz skupa W koji su povezani i ne postoje dva čvora iz skupa X koji su povezani.
- Planarni graf (graf jedne ravni) - graf koji se može nacrtati tako da se grane nigdje ne presjecaju
- Stablo - povezani graf koji nema cikluse

2.2 Stablo

2.2.1 Stablo u teoriji grafova

U matematici i teoriji grafova, stablo predstavlja neusmjereni graf u kome su bilo koja dva čvora povezana tačno jednom linijom. Unija više odvojenih stabala se naziva šuma [3].

Postoji više vrsta struktura podataka koji se u računarskim naukama nazivaju stablima iako su to najčešće stabla sa korijenom. Stablo sa korijenom može biti usmjereno (usmjereno korijenosko stablo) bilo da sve ivice usmjerava van korijena (izvan-stablo) ili da sve ivice usmjerava ka korijenu (u-stablo) [3]. Na slici 2.2.1.1 je prikazan izgled stabla.



Slika 2.2.1.1 Primjeri stabla [4]

Definicije:

Stablo je neusmjeren graf G takav da zadovoljava bilo koji od sledećih ekvivalentnih uslova [3]:

- G je povezan i nema prostih ciklusa
- G je acikličn; jednostavan ciklus je formiran ako se doda nova grana na G
- G je povezan ali neće biti povezan ako se ukloni bilo koja grana sa G
- G je povezan i kompletan graf sa tri čvora nije podskup grafa G
- Bilo koja dva čvora u G se mogu povezati unikatnom jednostavnom putanjom

Ukoliko G ima konačno mnogo čvorova n, tada vrijede i sledeći ekvivalentni uslovi:

- G je povezan i ima n-1 grana
- G nema prostih krugova i ima n-1 grana

Kao i u ostalim dijelovima teorije grafova, graf sa 0 čvorova je isključen iz razmatranja.

2.2.2 Stablo kao struktura podataka

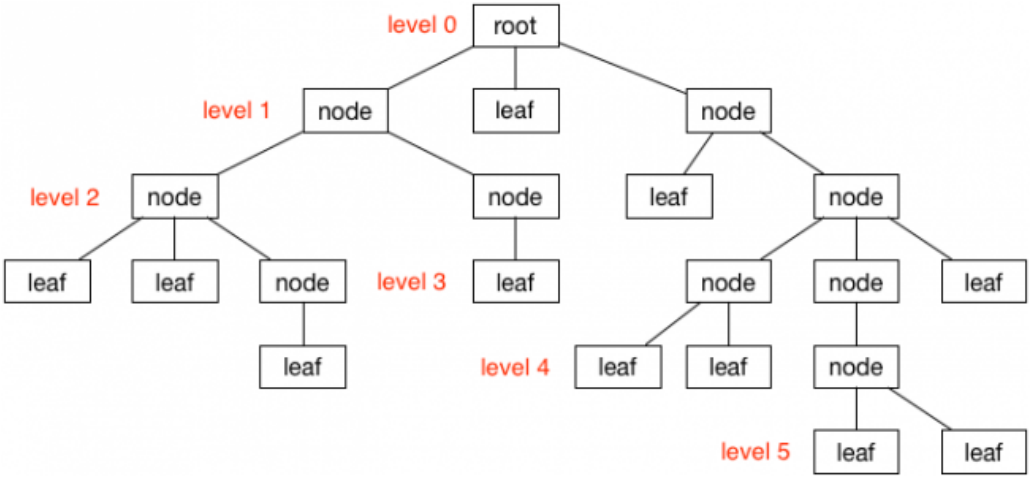
U računarskim naukama, stablo je često korišćen apstraktni tip podataka ili struktura podataka implementirana na osnovu apstraktnih tipova podataka, takva da simulira hijerarhijsku strukturu stabla sa korijenskom vrijednošću i pod-stablima djece sa roditeljskim čvorovima, predstavljenih kao par povezanih čvorova [5].

Stablo, kao struktura podataka, se može definisati rekurzivno (lokalno) kao kolekcija čvorova (počevši od korijenskog čvora). Svaki čvor je zasebna struktura koja se sastoji od vrijednosti i liste referenci ka "djeci" čvorovima. Pri tome važi ograničenje da niti jedna referenca ne smije biti duplicirana i da niti jedna vodi direktno do korijena [5].

Stablo se može definisati i apstraktno u cjelosti (globalno) kao uređeno stablo kod koga je svakom čvoru dodjeljena vrijednost.

Obe perspektive su korisne jer se stablo matematički može posmatrati kao cjelina a reprezentovati kao struktura podataka podjeljeno po čvorovima.

Na slici 2.2.2.2 prikazan je jedan primjer stabla kao strukture podataka.



Slika 2.2.2.2 Primjer stabla kao strukture podataka [6]

Definicija:

Stablo je struktura podataka sačinjena od čvorova i ivica bez kružnih ciklusa [5]. Prazno stablo je izraz za stablo koje nema čvorova. Stablo koje nije prazno, sačinjeno je od korijena i potencijalno mnogo nivoa dodatnih čvorova koji čine hijerarhiju.

Terminologija [5]:

- Korijen - Najviši čvor na vrhu stabla
- Dijete - Čvor direktno povezan za drugi čvor, krećući se od korijena.
- Roditelj - Čvor direktno povezan za neki drugi čvor, krećući se ka korijenu
- List - Čvor bez djece
- Susjedi - Grupa čvorova sa istim roditeljom
- Grana - Poveznica između dva čvora
- Putanja - Put koji se mora preći da bi se od jednog čvora stiglo do nekog drugog
- Nivo - Broj konekcija između nekog čvora i korijena stabla
- Visina čvora - Broj grana najduže putanje od čvora ka listu.
- Visina stabla - Broj grana najduže putanje od korijena do najudaljenijeg lista
- Šuma - Više odvojenih stabala

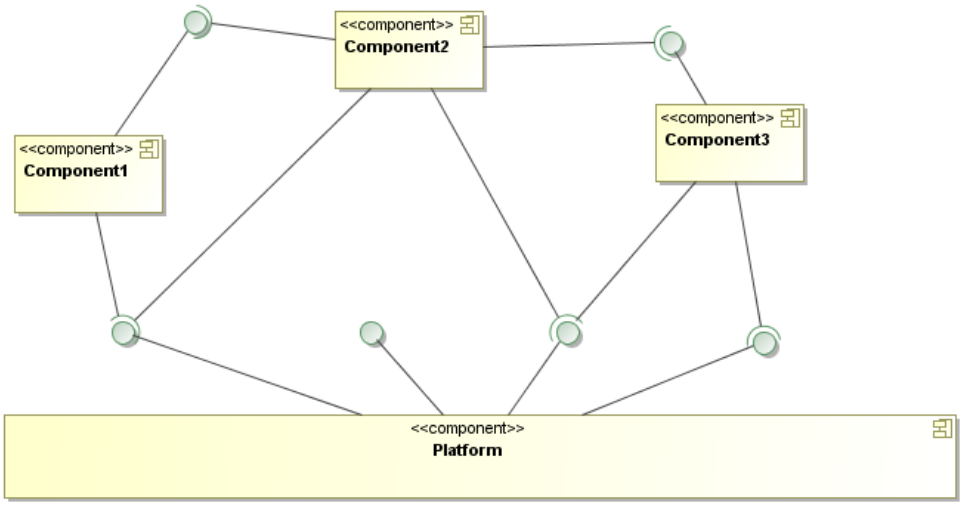
3 Tehnologije

3.1 Komponentni razvoj softvera

Softverske komponente su izvršive jedinice koje se nezavisno proizvode, dobavljaju i instaliraju i koje se mogu komponovati u cilju formiranja funkcionalnog sistema [7]. U cilju obezbjeđenja kompozicije, komponente moraju da poštuju određeni komponentni model i da budu kreirane za određenu ciljnu platformu [7]. Sistem sastavljen od softverskih komponenti se naziva komponentni softver [7]. Na slici 3.1.1 su prikazani osnovni dijelovi komponentno razvijenog softvera.

Prednosti komponentnog razvoja [7]:

- Jasna podjela nadležnosti
- Visok stepen kohezije i nizak stepen sprege - komponente se bave jasno određenim poslom i komuniciraju sa svojom okolinom preko jasno definisanih interfejsa i protokola
- Ponovna iskoristivost - ista komponenta se može koristiti u više aplikacija
- Komponente se mogu kupovati od nezavisnih proizvođača koji se specijalizuju za izradu određenog tipa komponenti - brža izgradnja kvalitetnog softvera
- Održavanje i unapređenje komponente se prebacuje na proizvođača koji može biti eksterni (*third-party*)
- Proizvođači komponenti mogu istu komponentu prodavati velikom broju kupaca i time stvoriti preduslove za dalji razvoj i usavršavanje komponente



Slika 3.1.1 Osnovni dijelovi komponentno razvijenog softvera [7]

Platforma [7]:

- Obezbjeđuje infrastrukturu u koju se ugrađuju komponente
- Pruža skup bazičnih sevisa koje komponente mogu da koriste

Servisi [7]:

- Predstavljaju skup funkcionalnosti koje određena komponenta pruža drugim komponentama i čije korišćenje je definisano ugovorom, koji propisuje način upotrebe (npr. interfejsi i protokoli), i ograničeno politikom upotrebe servisa
- Platforma mora imati propisane mehanizme za obavljanje i pronalaženje servisa

Konzumenti i pružaoci servisa [7]:

- U komponentnom softveru svaka komponenta može biti u ulozi pružaoca i/ili konzumenta servisa
- Postoje različite tehnike prolanaženja i povezivanja servisa

3.2 *Python*, programski jezik

Python je interpreterski, interaktivni, objektno-orjentisani programski jezik nastao 1991. godine [8]. Podržava module, izuzetke, dinamičko pisanje, klase, te visok nivo dinamičkih tipova podataka [8]. Kombinuje visoku moć i čistu sintaksu. Posjeduje interfejse ka velikom broju sistemskih poziva i biblioteka. Moguće ga je proširiti čak i sa *C* ili *C++* programskim jezicima. Značajna osobina *pythona* je i njegova portabilnost koja mu omogućava da se pokrene na većini *Unix*-baziranih operativnih sistema, te na *Mac* i *Windows* operativnim sistemima [8].

3.3 *Django Framework*

Django je *open-source web framework* (eng. okvir), napisan u *Pythonu* koji poštuje *model-view-template* (eng. model-pogled-šablon) arhitektualni obrazac [9]. *Web framework* u svojoj osnovi je grupisan skup biblioteka nekog programskog jezika. Osnovni cilj *Django Framework*-a je da olakša kreaciju *web*-sajtova koji su kompleksni i oslanjaju se na baze podataka. Promoviše ponovnu iskoristivost, modularnost i brz razvoj [9]. Osnovni princip Djanga je "Ne ponavljaj! ".

Komponente [9] :
U jezgru *Django framework*-a se nalaze sledeće komponente:

- *Web*-server za razvoj i testiranje
- Sistem za serijalizaciju i validaciju koji može prevoditi *HTML* (*Hypertext markup language*) forme u podatke pogodne za smještanje u bazu podataka.
- Sistem šablona koji koriste koncepte nasljeđivanja
- Sistem za internacionalizaciju
- Sistem za serijalizaciju koji može čitati i kreirati *XML* (Extensible markup language) i/ili *JSON* (*JavaScript object notation*) reprezentacije Djangovih modela
- Dinamički administrativni interfejs
- Alate za generisanje *RSS* (*Rich Site Summary*) i *Atom* poruka

Djangov konfiguracioni sistem omogućava da kod stranih lica bude uključen u projekat, pod pretpostavkom da poštuje konvencije ponovno korištenih aplikacija (*reusable apps*). Postoji više od 2500 paketa koji omogućuju proširenje originalnog Djangovog ponašanja. Ovi paketi nude rješenja za probleme koje Django osnova ne pokriva. Neki od problema su: registracija, pretraga, *API* (*Application programming interface*) ponuda i konzumacija, *CMS* (eng. *Content management system* - Sistem za upravljanje sadržajem), itd [9].

3.4 *JavaScript*

Javascript je prototipski-baziran, multi-paradigmicni, interpreterski programski jezik [10].

Zajedno sa *HTML*-om i *CSS*-om, Javascript čini jednu od tri osnovne tehnologije *World Wide Web* (eng. Svijetom rasprostranjena mreža) produkcije sadržaja.

Koristi se kako bi napravio *web*-stranice interaktivnim te ponudio online programe i video igre.

Svi popularni *web* preglednici imaju podršku za *JavaScript* bez potrebe za korištenjem dodatnih modula.

Kao multi-paradigmični jezik, *JavaScript* podržava događajem-vođene, funkcionalne i imperativne stilove programiranja [10]. Posjeduje *API* za rad sa tekstom, nizovima, datumima, regularnim izrazima, te osnovnim manipulacijama *DOM*-om (*Document Object Model*) [10].

3.5 *D3.js*

D3.js je *JavaScript* biblioteka za manipulaciju dokumenata baziranih na podacima [11]. *D3.js* pomaže vizualizaciji podataka korištenjem *HTML*-a (*Hypertext markup language*), *SVG*-a (*Scalable vector graphics*) i *CSS*-a.(*Cascading style sheets*) [11].

Osnovne osobine [12]:

- *Web*-Standardi: *D3.js* Koristi moderne *web*-standarde kao što su *svg*, *html* i css kako bi omogućio vizualizaciju
- Vođen podacaima: *D3.js* može koristiti statične ili dovući podatke sa stranih servera u različitim formatima kao što su nizovi, objekti, *csv*, *json*, *xml* i ostali, kako bi kreirao različite tipove grafikona.
- *DOM* manipulacija
- Elementi vođeni podacima: Omogućava podacima da dinamički generišu elemente i stilove elemenata.
- Dinamičke osobine: *D3* nudi fleksibilnost ponude dinamičkih osobina većini svojih funkcija. Osobine mogu biti specificirane kao funkcije podataka. To bi značilo da podaci mogu da utiču na stilove i atribute.
- Prilagođena vizualizacija: *D3* omogućava korisniku potpunu kontrolu nad vizualizacijom podataka
- Tranzicije: *D3* nudi funkciju *transition()* koja je veoma moćna jer omogućava *D3* da logički interpolira vrijednosti.
- Interakcija i animacija: *D3* korisnicima nudi veliku pomoć pri radu sa animacijama.

3.6 *SetupTools*

Setuptools je kolekcija proširenja i dodataka *Python*-ovoj *distutils* biblioteci koji omogućavaju programerima da lakše kreiraju i distribuiraju *Python* pakete, pogotovo onima koji imaju zavisnosti u svojim paketima [15].

Paketi kreirani i distribuirani korišćenjem *SetupTools*-a izgledaju korisniku kao i obični ***Python*** paketi kreirani običnom bibliotekom *distutils [15]*.

Biblioteka *SetupTools* u sebi posjeduje [14]:

- Definicije *python* paketa i podula
- Meta-podatke o distribuciji paketa
- Testove
- Instalacije projekata
- Detalje vezane za specifične platforme
- Podršku za *python 3*

Mogućnosti koje *SetupTools* biblioteka pruža [15]:

- Automatsko pronalaženje/čuvanje/instaliranje zavisnosti u trenutku kreiranja korišćenjem *EasyInstall* alata.
- Kreiranje *Python Eggs* fajla namjenjenog za učitavanje distribucija
- Dodatno poboljšana podrška za pristup podacima uskladištenim u *.zip* pakete
- Automatsko uključivanje paketa u razvojno stablo.
- Automatsko generisanje okvirnih skripti za *Windows .exe* fajlove.

Minimalan primjer *setup.py* fajla u kome se navode bitni meta - podaci, dat je je u listingu 3.6.1 .

```
1. from setuptools import setup, find_packages
2.       setup(
3.       name="HelloWorld",
4.       version="0.1",
5.       packages=find_packages(),
6.       )
```

Listing 3.6.1 Minimalan primjer *setup.py* fajla

4 Platforma

Kao što je već ranije pomenuto, platforma služi kao jezgro čitavog projekta. Platforma posjeduje bazu podataka, servise na osnovu kojih se moduli mogu povezati sa njom, prikaze, sistem za registrovanje modula i mnoge druge stvari.

Platforma koristi *Django framework* kako bi funkcionisala kao *web*-stranica na koji se moduli mogu zakačiti. Na ovaj način, moguće je instalirati platformu na neki server i njene mogućnosti na korišćenje ponuditi velikom broju ljudi putem interneta.

4.1 Povezivanje platforme sa *Django Framework-om*

Bilo koja aplikacija pisana u pythonu, pa tako i ova platforma, se može veoma lako povezati sa *Django framework*-om.

Prije svega, naravno, potrebno je na računar instalirati *Django Framework*. Ovo se najlakše postiže korišćenjem pythonovog *Pip* alata. *Pip* je alat za upravljanje paketima koji može da dodaje, upravlja i briše pakete napisane u *Python* programskom jeziku [16]. Komanda za instalaciju Django frameworka, na *Linux* operativnim sistemima, uz pomoć *pip*-a je sledeća:

```
pip install django
```

Nakon kraćeg podešavanja konfiguracionih fajlova, Djangov ugrađeni server se može pokrenuti sledećom komandom:

```
python manage.py runserver
```

Da bi se određena aplikacija spojila sa *Django framework-om*, potrebno je editovati par Djangovih fajlova i u njih popuniti informacije o platformi.

Prvi korak jeste napisati fajl *apps.py.* U ovom fajlu bi trebalo kreirati novu klasu, takvu da nasljedjuje *Djangovu* klasu *AppConfig* iz paketa *django.apps.config* i u toj klasi napisati sve potrebne podatke koji se očekuju. Detaljnije o *apps.py* će biti riječi u nastavku.

Drugi korak bi bio, u fajl *settings.py* napisati pristupnu lokaciju do konfiguracione klase koja nasljedjuje klasu *AppConfig* u *apps.py* fajlu, kao što je prikazano u listinga 4.1.1 :

```
1. INSTALLED_APPS = [
2.     'ProjekatTim6.apps.Projekattim6Config',
3.     'django.contrib.admin',
4.     'django.contrib.auth',
5.     'django.contrib.contenttypes',
6.     'django.contrib.sessions',
7.     'django.contrib.messages',
8.     'django.contrib.staticfiles',
9. ]
```

Listing 4.1.1 Povezivanje platforme sa *django framework*-om koristeći *settings.py* fajl

Nakon ovoga, ostaje samo da se u *Djangov* fajl *urls.py* napiše pristupna putanja do *urls.py* fajla aplikacije kao što je prikazano u sledećem kratkom listingu 4.1.2 .

```
1. from django.conf.urls import url, include
2. from django.contrib import admin
3. urlpatterns = [
4.     url(r'^admin/', admin.site.urls),
5.     url(r'^',include("ProjekatTim6.urls"))
6. ]
```

Listing 4.1.2 *urls.py* fajl *Django Framework*-a

Urlpatterns je niz tipova *url* (*Uniform Resource Locator*) koji služe kao poveznice između adresa i lokacija *urls.py* učitanih aplikacija.

Na ovom primjeru se može vidjeti kako je putanja "*/admin/*" smjernica za pristup daljnjem adresnom račvanju Djangove aplikacije za upravljanje administratorima, te, kako je prazna putanja "/" smjernica za daljnje račvanje adresa platforme. Svaka aplikaciaj povezana sa Djangom mora posjedovati i svoj sopstveni fajl *urls.py*. Primjer *urls.py* fajla platforme dat je listingom 4.1.3 .

```
1. from django.conf.urls import url
2. from ProjekatTim6 import views
3. urlpatterns = [
4.     url(r'^$', views.index, name="index"),
5.     url(r'^ucitavanje/plugin/(.*?)',
   views.ucitavanje_plugin, name="ucitavanje_plugin"),
```

```
6.      url(r'^ucitavanje/plugin_param/(?P<id>([a-z]+|[_])+)
   $', views.ucitavanje_plugin_broj_param,
   name="ucitavanje_plugin_broj_param"),
7.      url(r'^prikazati/plugin/(?P<id>([a-z]+|[_])+)$',
   views.prikazi_plugin, name="prikazi_plugin"),
8.      url(r'^primer1$', views.primer1, name="primer1"),
9.      url(r'^primer2$', views.primer2, name="primer2"),
10.       ]
```

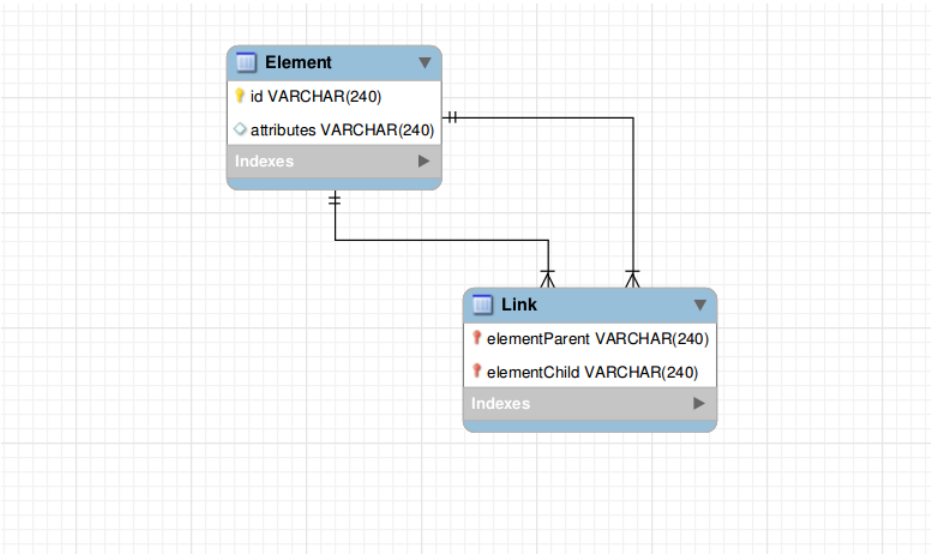
Listing 4.1.3 Fajl *urls.py* platforme

Niz ***urlpatterns***, kao i ranije, predstavlja niz url šablona koji upućuju ka određenom cilju.
Npr. regularni izraz "`r'^prikazati/plugin/(?P<id>([a-z]+|[_])+)$`" predstavlja putanju ka pozivu funkcije *prikazi_plugin* sa određenim parametrima, u fajlu *views.py*. Fajl *views.py* će biti naknadno detaljnije objašnjen.
Regularni izraz "`(?P<id>([a-z]+|[_])+)`" označava da se radi o parametru "`id`" koji se može sastojati iz jednog ili više slova te jedne ili više donjih crtica.

4.2 Modeli

Platforma posjeduje samo dva modela objekata koji se koriste za smještanje podataka i njihovo prikazivanje. Svaki model se direktno preslikava u tabelu baze podataka. Modeli su prikazani i slikom 4.2.1 .

- **Element** - predstavlja određen podatak (čvor) i njegove atribute
- **Link** - predstavlja vezu između dva elementa (čvora)



Slika 4.2.1 Modeli

Python klase koje predstavljaju modele objekata date su listingom 4.2.2 .

```
1. class Element(models.Model):

2.     id = models.CharField(max_length=240,
   primary_key=True)
3.     attributes = models.CharField(max_length=240)
4.     def __str__(self):
5.         return self.id
6. class Link(models.Model):
7.     elementParent = models.ForeignKey(Element,
   related_name='parentElementi')
8.     elementChild = models.ForeignKey(Element,
   related_name='childElementi')
9.     def __str__(self):
10.             return self.elementParent
```

Listing 4.2.2 Klase modela Element i Link

Klasa ***Element*** nasljeđuje Djangovu klasu *models.Model*. Element predstavlja čvor grafa, tj. čvor stabla. *Element* posjeduje svoju identifikaciju koja ce ujedino služiti i kao primarni ključ za smještanje u bazu podataka, te, atribute koje čvor posjeduje.

Klasa ***Link*** također nasljedjuje Djangovu klasu *models.Model*. Link predstavlja granu, tj. vezu između dva Elemeneta, tj. čvora na grafu i stablu. *ElementParent* predstavlja čvor od koga počinje veza, dok je *elementChild* čvor u koji je veza usmjerena.

Da bi programeri mogli lagodno komunicirati sa bazama podataka, *Django* nudi *API* za komunikaciju sa bazama podataka koji omogućava kreiranje (čuvanje), povlačenje, ažuriranje i brisanje podataka.

Primjeri osnovnih operacija sa Djangovim *API*-jem za rad sa bazama:

- Kreiranje (čuvanje) podataka:
  - `element1 = Element(id="element1", attributes = "some atts")`
  - `element1.save()`
- Ažuriranje podataka:
  - `element1.attributes = "name=element1; colour=black; fill=solid"`
  - `element1.save()`
- Povlačenje (dobavljanje) podatka:
  - `elementx = Element.objects.get( id="element1" )`
  - `all_elements = Element.objects.all()`
- Filtriranje:
  - `filtered_elements = Element.objects.all().filter( attributes = "some atts")`

- Limitiranje broja povučenih (dobavljenih) podataka:
  - `limited_elements = Element.objects.all()[:10]`
- Brisanje elemenata:
  - `element1.delete()`
  - `Element.objects.all().delete()`

4.3 Servisi

Servisi su usluge koje platforma nudi modulima. Drugim riječima, to su mjesta za pristupanje platformi. Oni predstavljaju uputstvo za programere; na koji način bi moduli trebali biti implementirani i kako ih vezati za platformu.

Ova platforma nudi dva tipa servisa:

- **Učitavanje podataka** - Servis za učitavanje podataka ne ograničava programera koje podatke može da učita niti koje algoritme koristi.

- **Prikaz podataka** - Servis za prikaz podataka mora biti implementiran od strane programera modula namjenjenog za prikaz podataka. Ovaj modul ograničava programera da implementacija pojedinih metoda mora biti napisana u *JavaScript* programsom jeziku korištenjem *d3.js* biblioteke.

Listingom 4.3.1 prezentovan je kod koji predstavlja interfejs servisa za prikaz podataka:

```
1. import abc
2. class PrikazatiService(metaclass=abc.ABCMeta):
3.     @abc.abstractmethod
4.     def naziv(self):
5.         pass
6.     @abc.abstractmethod
7.     def identifier(self):
8.         pass
9.     @abc.abstractmethod
10.          def srediPrikaz(self):
11.              pass
12.          @abc.abstractmethod
13.          def staticPrikaz(self):
14.              pass
```

Listing 4.3.1 Interfejs servisa za prikaz podataka

Klasa ***PrikazatiService*** služi kao servis za prikaz u ovoj platformi. Kako bi se modul namjenjen za prikaz sadržaja mogao prikazati neophodno je da naslijedi ovu klasu te implementira njene metode.

Metoda ***naziv*** bi trebalo da vraća prikladno i ljudski čitljivo ime modula. Metoda ***identifier*** bi trebalo da vraća identifikaciju modula koja se koristi za njegovo učitavanje u platformu.

Metoda ***staticPrikaz*** bi trebala da vrati tekst *JavaScript* formata koji bi u sebi sadržao programske instrukcije o iscrtavanju elemenata, linkova, fizici, pomjeranju i sličnim stvarima.

Metoda ***srediPrikaz*** bi trebalo da vrati tekst *JavaScript* formata koji bi u sebi sadržao sve napredne funkcije koje se mogu često pozivati iz aplikacije. Npr: *Search*, Filter, *Zoom*, itd..

Listingom 4.3.2 prezentovan je kod koji predstavlja servis za prikupljanje i učitavanje podataka:

```
1. import abc
2. class UcitatiService(metaclass=abc.ABCMeta):
3.
4.     @abc.abstractmethod
5.     def naziv(self):
6.         pass
7.
8.     @abc.abstractmethod
9.     def identifier(self):
10.            pass
11.
12.        @abc.abstractmethod
13.        def ucitatiPodatke(self,*args,**kwargs):
14.            pass
15.
16.        @abc.abstractmethod
17.        def brojPotrebnihParametara(self):
18.            pass
19.
20.        @abc.abstractmethod
21.        def naziviParametara(self):
22.            pass
```

Listing 4.3.2 Interfejs servisa za prikupljanje i učitavanje podataka

Klasa ***UcitatiService*** služi kao servis za učitavanje podataka ove platforme. Moduli namjenjeni za učitavanje podataka bi morali naslijediti ovu klasu te implementirati njene metode.

Metoda ***naziv*** bi trebala vratiti ljudski čitljiv tekst koji jasno označava i predstavlja ime modula za učitavanje podataka.

Metoda ***identifier*** bi trebala vratiti jedinstvenu identifikaciju modula kako bi platforma mogla da ga registruje i koristi.

Metoda ***brojpotrebnihParametara*** bi trebala da vrati broj neophodnih parametara koje bi korisnik morao da unese kako bi koristio ovaj modul.

Metoda ***naziviParametara*** bi trebala vratiti nazive svih neophodnih parametara kako bi korisnik znao šta sve mora da unese da bi mogao koristiti modul.

Metoda ***ucitatiPodatke*** bi trebala implementirati mehanizam učitavanja podataka. Programer nije ograničen kakve podatke će učitati niti koje algoritme će implementirati. Jedina obaveza programera po pitanju ove metode jeste da podatke snimi u bazu u skladu sa modelima koje platforma propisuje.

4.4 Registrovanje platforme i modula na Django Framework

Kada su moduli napisani, potrebno ih je instalirati i učitati u platformu, te sve zajedno povezati sa Djangom. Instalacija modula će biti naknadno objašnjena. Način učitavanja, tj. registrovanja i prepoznavanja modula u samoj platformi najlakše je objasniti listingom 4.4.1.

```
1. import pkg_resources
2. from django.apps.config import AppConfig
3.
4. class Projekattim6Config(AppConfig):
5.     name = 'ProjekatTim6'
6.     plugini_prikaz = []
7.     plugini_ucitavanje = []
8.
9.     def ready(self):
10.
   self.plugini_prikaz=load_plugins("graph.prikazati")
11.             self.plugini_ucitavanje=
   load_plugins("data_import.ucitati");
12.
13.      def load_plugins(oznaka):
14.          plugins = []
15.
16.          for ep in
   pkg_resources.iter_entry_points(group=oznaka):
17.              p = ep.load()
18.              print("{} {}".format(ep.name, p))
19.              plugin = p()
20.              plugins.append(plugin)
21.          return plugins
```

Listing 4.4.1 *apps.py* fajl platforme

Metoda ***load_plugins*** služi za registrovanje, tj. učitavanje modula u platformu. Njen način rada opisan je u sledećem paragrafu:

Dok god za dodjeljeni grupni identifikator postoje paketi (moduli) sa tom oznakom, metoda učitava module, inicijalizuje ih i smješta u listu modula, te, tu listu vraća kao rezultat svoje obrade.

Klasa *Projekattim6Config* nasljedjuje Djangovu klasu *AppConfig* koja služi za prepoznavanje i učitavanje komponenti.

Atribut *name* služi kao identifikatorski naziv platfotforme sa adresom, koji će *Django* koristiti za sve što mu bude bilo potrebno kada bude imao potrebu da komunicira sa platformom.

U klasi *AppConfig* je takođe definisana i metoda *ready* koja služi kao okidač svaki put kada se *Django* server i platforma pokrenu. Ona je u ovom slučaju podešena tako da poziva metodu *load_plugins* tako da pronalazi sve instalirane module namjenjene za učitavanje i prikaz podataka.

4.5 Pogledi i vizuelizacija

Fajl *views.py* aplikacije bi trebao služiti kao osnovno mjesto u kome se pišu *view* (eng. pogled) funkcije. *View* funkcije su obične *Python* funkcije koje primaju *web* zahtjev i vraćaju *web* odgovor. Odgovor koji vraćaju može biti sadržaj *web*-stranice, preusmjerenje, greška "404", *xml* fajl, slika ili bilo šta drugo. *View* funkcije u sebi sadrže proizvoljnu logiku potrebnu za vraćanje odgovora [17]. Primjer jedne *view* funkcije je funkcija *index*, data u listingu 4.5.1 .

```
1. from django.shortcuts import render, redirect
2. from django.apps.registry import apps
3. from ProjekatTim6.models import Element, Link, TreeElement
4.
5. def index(request):
6.     elementi = Element.objects.all()
7.     linkovi = Link.objects.all()
8.     broj_param = 0
9.     plugini =
   apps.get_app_config('ProjekatTim6').plugini_prikaz
10.          ucitavanje_plugini =
   apps.get_app_config('ProjekatTim6').plugini_ucitavanje
11.          return render(request,"index.html",
   {"title":"Index","linkovi":linkovi,"elementi":elementi,"pl
   ugini":plugini,"ucitavanje_plugini":ucitavanje_plugini,
   "broj_param":broj_param})
```

Listing 4.5.1 Funkcija *index* iz fajla *views.py*

Funkcija *index* je funkcija koja se pokreće kada korisnik pokuša da direktno pristupi *web*-stranici aplikacije. Ova funkcija kao odgovor korisniku vraća izgenerisanu *index.html* stranicu koja se korisniku može jasno prikazati na *web*-pretraživaču.

U posljednjoj liniji koda iz listinga se kao odgovor ove funkcije vraća objekat *render*. *Render* kao parametre, u ovom slučaju, prima zahtjev, lokaciju i naziv šablona za generisanje *web*-sadržaja, te, mapu (*context*) u kojoj se kao parovi ključ-vrijednost nalaze nazivi i elementi potrebni šablonu i generatoru sadržaja. Dio šablona za izradu *web*-sadržaja *index.html* prikazan je listingom 4.5.2 .

```
1. <body style="background-color: #ccffff;">

2. <nav class="navbar navbar-default navbar-default"

3.       style="border-radius: 0px !important; margin-bottom:
   0px;">
4.    <div class="container-fluid">
5.       <!-- Brand and toggle get grouped for better mobile
   display -->
6.       <div class="navbar-header">
7.          <button type="button" class="navbar-toggle
   collapsed" data-toggle="collapse" data-target="#bs-
   example-navbar-collapse-1"
   aria-expanded="false">
8.             <span class="sr-only">Toggle navigation</span>
9.              <span class="icon-bar"></span>
10.            <span class="icon-bar"></span>
11.            <span class="icon-bar"></span>
12.               </button>
13.            </div>
14.      <h1>  SOK: </h1>
15.         </div>
16.      </nav>
17.      <hr>
18.      <div>
19.         <div>
20.      <table border="3" >
21.         <h2> Plugins:</h2>
22.         <tr border="3">
23.             {% if ucitavanje_plugini %}
24.                 {% for p in ucitavanje_plugini %}
25.                     <td border="3">
```

```
26.                     <a class="form-control btn btn-
   success" href="{% url 'ucitavanje_plugin_broj_param'
   id=p.identifier %}">{{p.naziv}}</a>
27.                     </td>
28.                 {% endfor %}
29.             {% else %}
30.                     <td border="3">
31.                     <h3>There are 0 importing
   plugins</h3>
32.                     </td>
33.             {% endif %}
34.             {% if plugini %}
35.                 {% for p in plugini %}
36.                     <td border="3">
37.                      <a class="form-control btn btn-
   success" href="{% url 'prikazi_plugin' id=p.identifier
   %}">{{p.naziv}}</a>
38.                     {% endfor %}
39.                 {% else %}
40.                         <td border="3">
41.                      <h3>There are 0 visualization
   plugins</h3>
42.                         </td>
43.             {% endif %}
44.         </tr>
45.         </table>
46.         </div>
47.     </div>
48.     <hr>
49.     <div>
50.         {% if neophodni_parametri %}
51.             {%if naziviParametara %}
52.         <h5> Required parameters (in order) :
   {{ naziviParametara | safe }} </h5>
53.             {% endif %}
54.             {{ neophodni_parametri|safe }}
55.         {% endif %}
56.     </div>
57.     <hr>
58.     {% block content %}
59.     <table border="1px">
60.         <tr border="3px">
61.             <td border="3px">
62.                 <input type="text" id="filterx"
   name="filterx" placeholder="Filter">
63.             </td>
64.             <td>
```

```
65.                 <input type="text" id="searchx"
   name="searchx" placeholder="Search">
66.             </td>
67.         </tr>
68.     </table>
69.     <div position:relative; width = "100%"; height =
   "600px">
70.         <div width="35%" height = "500px"
   style="float:right">
71.             <h2>Graphview:</h2>
72.             <svg id="svggraph" width="500" height="500">
73.             </svg>
74.         </div>
75.         <div width="30%" height = "500px"
   style="float:right" >
76.             <h2> Tree View:</h2>
77.             <svg id="treesvg" width="350" height="500">
78.         </svg>
79.         </div>
80.         <div width="35%" height = "600px"
   style="float:left">
81.             <h2>Birdview:</h2>
82.             <svg id="svgbird" width="500" height="500">
83.             </svg>
84.         </div>
85.     </div>
86.     <div>
87.         <h2>Tree Data:</h2>
88.         <div class="scroll" id="treeData">
89.         </div>
90.     </div>
91.     <div >
92.         <p  class="centerofh1">Softverski obrasci i
   komponente</p>
93.     </div>
```

Listing 4.5.2 Isječak koda *index.html* fajla

Iz listinga 4.5.2 se može vidjeti kako je urađena osnovna stranica platforme, *index.html*. Namjena *index.html* stranice je da korisniku prikaže sve mogućnosti za rad sa platformom. Na ovoj stranici ponudjeni su svi moduli. Na njoj se moduli mogu konfigurisati, izvršiti te prikazati.

Poslednjih dvadesetak linija koda prikazuju postavljanje prostora za iscrtavanje podataka. Mogu se uočiti *svg* (*Scalable vector graphics*) "platna" za iscrtavanje sa sledećim identifikacijama: ***svggraph***; ***treesvg*** i ***svgbird***. Na ovim "platnima" uz pomoć ***d3.js*** biblioteke biće iscrtani grafovi i stabla. Iscrtavanje podataka će kasnije biti detaljno objasnjeno.

Index.html stranica je načinjena korišćenjem *HTML*-a, *CSS*-a, *JavaScripta*, ali i **Djangovog šablonskog jezika**. Djangov šablonski jezik sluzi kao poveznica između *backenda* i *frontenda*. Omogućava da se *python* kod iskoristi i za generisanje *HTML* sadržaja. Djangov šablonski jezik i generator sadržaja rade sa podacima koji su proslijeđeni kao **mapa** (***context***) u ***render*** objektu.

Osnovna sintaksa Djangovog šablonskog jezika:

- **Varijable** - Predstavljaju sadržaj proslijeđen u mapi (*context*-u) *render* objekta. Upisuju se između dva para vitičastih zagrada. Primjer: `{{ p.naziv }}`.

- **Tagovi** - Omogućuju upotrebu logike. Standardna *python* logika se moze upisati između parova vitičastih zagrada i znakova postotka. Primjeri: `{% for p in plugini %}, {% endfor %}`.

- **Filteri** - Transformišu vrijednosti varijabli i tagova. Predstavljeni su uspravnom linijom. Primjer: `{{nazivParametra | safe }}`

- **Komentari** - Karakteri koji neće biti generisani u *html* kod. Zapisuju se između parova vitičastih zagrada i taraba. Primjer: `{# komentar #}`

4.6 Prikaz stabla

Za potrebe prikaza stabla na platformi, bez modula, korišćeni su Djangov šablonski jezik, *JavaScript* i *d3.js* biblioteka.

Listingom 4.6.1 dat je isječak koda *index.html* stranice koji je zaduzen za prevođenje podataka iz *Python* programskog jezika u *JavaScript*. Podaci su proslijeđeni iz fajla *views.py html*-generatoru sadržaja korišćenjem objekta *render*. Kod je napisan korišćenjem Django šablonskog jezika i *JavaScripta*.

```
1. var elementi_za_obradu = [

2.     {% for e in elementi %}
3.         {id:"{{e.id}}",attributesx:"{{e.attributes}}"},
4.     {% endfor %}
5. ];

6. var linkovi_za_obradu =[

7.     {% for l in linkovi %}
8.         { elementParent: "{{l.elementParent.id}}",
   elementChild: "{{l.elementChild.id}}"},
9.     {% endfor %}
10.      ];
```

Listing 4.6.1 Prevođenje podataka iz *Python*-a u *Javascript*

Nakon prevođenja podataka iz *Pythona* u *JavaScript*, program, korišćenjem veza, kreira strukturu grafa tipa stabla. Ovaj posao se obavlja tako što se pretražuju čvorovi roditelji, i u njih, uz pomoć veza, kače čvorovi djeca. Kod je dat listingom 4.6.2 .

```
1. var first_parents = [];

2. for (i=0; i< elementi_za_obradu.length; i++)
3. {
4.     var naslo = false;
5.     for (j=0; j<linkovi_za_obradu.length; j++)
6.     {
7.         if(elementi_za_obradu[i]
   ['id']==linkovi_za_obradu[j].elementChild)
8.         {
9.             naslo=true;
10.                 break;
11.             }
12.          }
13.          if(naslo == false)
14.          {
15.              first_parents.push(elementi_za_obradu[i]
   ['id'])
16.          }
17.      }


18.      var dd= function (parent)

19.      {
20.          var lista_djece = []
21.          var lista_te_djece = []
22.          for(var i =0; i< linkovi_za_obradu.length; i++)
23.          {
24.              if(parent ==
   linkovi_za_obradu[i].elementParent)
25.              {
26.
   lista_djece.push(linkovi_za_obradu[i].elementChild)
27.              }
28.          }
29.          if(lista_djece.length!=0)
30.          {
31.          for(var z =0; z< lista_djece.length; z++)
32.          {
33.              var treeElementx = {}
34.              treeElementx['name']=lista_djece[z];
35.              treeElementx['naziv']=lista_djece[z];
36.              treeElementx['attributesx']="";
37.              for(var qw =0; qw<elementi_za_obradu.length;
   qw++)
38.              {
```

```
39.                   if(elementi_za_obradu[qw]
   ['id']==lista_djece[z])
40.                   {
41.
         treeElementx['attributesx']=elementi_za_obradu[qw]
   ['attributesx'];
42.                       break;
43.                   }
44.               }
45.               treeElementx['children']=dd(lista_djece[z]);
46.               lista_te_djece.push(treeElementx)
47.           }
48.           }


49.           if(lista_te_djece.length!=0)


50.           {
51.           return lista_te_djece;
52.           }
53.           else
54.           {
55.               return null
56.           }
57.       }
58.       var lista_pp_te =[];
59.       for (var xm =0 ; xm<first_parents.length; xm++)
60.       {
61.           var treeElement ={}
62.           treeElement['name']=first_parents[xm];
63.           treeElement['naziv']=first_parents[xm];
64.           for(var qw =0; qw<elementi_za_obradu.length; qw+
   +)
65.               {
66.                   if(elementi_za_obradu[qw]
   ['id']==first_parents[xm])
67.                   {
68.
   treeElement['attributesx']=elementi_za_obradu[qw]
   ['attributesx'];
69.                       break;
70.                   }
71.               }
72.           treeElement['children']=dd(first_parents[xm]);
73.           lista_pp_te.push(treeElement);
74.       }
75.       // Add tooltip div
```

```
76.              var div = d3.select("body").append("div")
77.              .attr("class", "tooltip")
78.              .style("opacity", 1e-6);
79.      var data ={}
80.      data['name']="Root";
81.      data['naziv']="Root";
82.      data['attributes']="Root element.."
83.      data['children']=lista_pp_te
```

Listing 4.6.2 Kreiranje strukture grafa tipa stabla za prikaz

Kada se kreiranje strukture grafa tipa stabla završi, preostaje samo da se rezultat prikaže. Prikaz podataka urađen je korištenjem *JavaScripta* i njegove *D3.js* biblioteke, što je moguće i vidjeti na listingu 4.6.3 .

```
1. svgtree=d3.select('#treesvg').call(d3.behavior.zoom().scal
   eExtent([-5, 20]).on("zoom", zoomTree)).append("g");

2.     numOfLinks = MyData.links.length;
3.     if(numOfLinks<100)
4.     {
5.         tree=d3.layout.tree().size([1000,1000]);
6.     }
7.     else
8.     {
9.
   tree=d3.layout.tree().size([numOfLinks*15,numOfLinks*10]);
10.          }
11.          nodestree=tree.nodes(data);
12.          linkstree=tree.links(nodestree);
13.          var diagonal = d3.svg.diagonal()
14.          .projection(function(d) { return [d.y, d.x]; });
15.          //Dodavanje linkova
16.           linktreex = svgtree.selectAll('.linktree')
17.              .data(linkstree)
18.              .enter().append('line')
19.              .attr('class', 'link')
20.              .attr('x1', function(d) { return d.source.y;
   })
21.              .attr('y1', function(d) { return d.source.x;
   })
22.              .attr('x2', function(d) { return d.target.y;
   })
23.              .attr('y2', function(d) { return d.target.x;
   });
```

```
24.          // Dodavanje cvorova
25.          nodetreex = svgtree.selectAll('.nodetree')
26.              .data(nodestree) //add
27.              .enter().append('g')
28.              .attr('class', 'nodetree')
29.              .attr('transform', function(d){return
   "translate("+d.y+","+d.x+")";});
30.          nodetreex.append('circle')
31.              .on("mouseover", mouseover)
32.              .on("mousemove", function(d){mousemove(d);})
33.              .on("mouseout", mouseout)
34.              .attr('r', 10) //radius of circle
35.              .attr('fill','yellow');
36.          nodetreex.append('text').attr('fill','black')
37.              .text(function(d){return d.naziv;})
38.              function zoomTree() {
39.              svgtree.attr("transform", "translate(" +
   d3.event.translate + ")scale(" + d3.event.scale + ")");
40.              }
41.          function mouseover() {
42.                      div.transition()
43.                      .duration(300)
44.                      .style("opacity", 1);
45.                  }
46.                  function mousemove(d) {
47.                      div
48.                      .text("Info about " + d.naziv + ":"
   + d.attributesx)
49.                      .style("left", (d3.event.pageX ) +
   "px")
50.                      .style("top", (d3.event.pageY) +
   "px");
51.                  }
52.                  function mouseout() {
53.                      div.transition()
54.                      .duration(300)
55.                      .style("opacity", 1e-6);
56.                  }
57.      }
```

Listing 4.6.3 Prikaz podataka korištenjem *d3.js* biblioteke

Prikaz podataka radi po istom principu kao i crtanje po platnu. U ovom slučaju, platno predstavlja ***svgtree***, *html* ***svg*** element. Sva iscrtavanja će biti prikazana u okviru njegovih tačno definisanih granica.

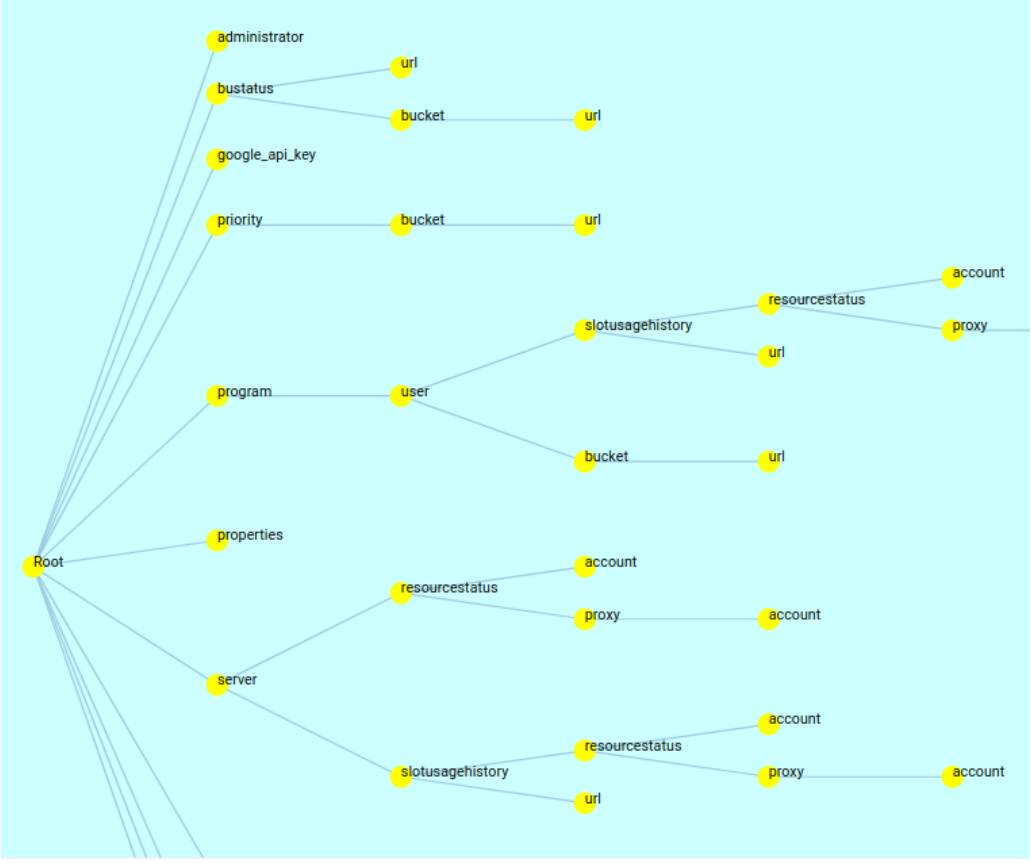
Algoritam funkcionisanja ovog dijela koda:

1. Na samom početku koda iz listinga, zauzima se *svg* platno. Platno se skalira i vidljiv spektar približava korisniku radi jasnosti;
2. Na osnovu količine veza, pomjera se vidljiv spektar namjenjen za iscrtavanje stabla na platnu;
3. Iscrtavanje i "ušminkavanje" veza;
4. Iscrtavanje i "ušminkavanje" čvorova, te dodavanje njihovih posebnih efekata

Primjeri upotrebe *d3.js* biblioteke:

- Selekcija jednog objekta nad kojima će se vršiti manipulacija - Primjer: `d3.select("body");`
- Selekcija više srodnih objekata nad kojima će se vršiti manipulacija - Primjer: `d3.selectAll("p");`
- Dodavanje ili izmjena izgleda (stila) objektu/ima - Primjer: `d3.select("p").style("color","blue");`
- Dodavanje ili izmjena atributa objekta/ata - Primjeri:
  `d3.select("p").attr("class","squares");`
  `d3.select("h2").attr("width",120);`
- Iscrtavanje elemenata i osvježavanje (Update) - Primjer [18]: `var p = d3.select("body").selectAll("p").data([4, 8, 15, 16, 23, 42]).text(function(d) { return d; });`
- Iscrtavanje elemenata (Enter) - Primjer [18]: `p.enter().append("p").text(function(d) { return d; });`
- Uklanjanje elemenata (Exit) - Primjer [18]: `p.exit().remove();`
- Animacije i tranzicije- Primjer [18] : `d3.selectAll("circle").transition().duration(750).delay(function(d, i) { return i * 10; }).attr("r", function(d) { return Math.sqrt(d * scale); });`

Izgled iscrtanog stabla prikazan je slikom 4.6.4 . Na slici se može uočiti da su čvorovi prikazani žutim krugovima pored kojih piše naziv čvora, te, da su veze prikazane plavim linijama.



Slika 4.6.4 Izgled stabla iscrtanog *d3.js* bibliotekom

5 Moduli

Moduli predstavljaju softverske komponente koje dodaju specifične funkcionalnosti postojećim programima [19]. Samim tim, moduli proširuju spektar mogućnosti koje program može ponuditi korisnicima.

Moduli su se pojavili sredinom sedamdesetih godina prošlog vijeka, dok je *EDT* tekstualni editor radio na *Unisys VS/9* operativnom sistemu koristeći *UNIVAC Series 90* računare koji su nudili mogućnost pokretanja programa iz editora i omogućavali programima da koriste editorov bafer. Na ovaj način je bilo omogućeno eksternim programima da koriste memorijsku sesiju editora. Modularni programi su mogli tražiti od editora da izvrši zadatke tekstualnog editovanja nad baferom koji je editor dijelio sa modulom. *Waterloo Fortran* kompajler je koristio ovu mogućnost kako bi dozvolio kompajliranje *fortran* programa napisanih u *EDT* tekstualnom editoru [19].

Danas se moduli uveliko koriste. Većina ozbiljno napisanih programa ih podržava i podstiče njihov razvoj. Najčešće se koriste kod internet preglednika, muzičkih plejera, video plejera, tekstualnih editora, muzičko-razvojnih programa, video-obrađivackih programa, video igrica, programa za rad sa bazama podataka i mnogim drugim.

Najosnovnije prednosti i namjene modula [19]:

- Omogućavaju stranim programerima da kreiraju mogućnosti koje proširuju program
- Omogućavaju jednostavno dodavanje novih funkcionalnosti
- Smanjuju veličinu programa
- Odvajanje programskog koda od aplikacije zbog softverskih licenci koje se ne poklapaju

5.1 Povezivanje modula i platforme

Da bi se moduli i platforma povezali, neophodno je instalirati modul. Za potrebe programskih uputsava za instalaciju korisi se *setupTools* alat. Programska uputstva za instalaciju se pišu u *setup.py* fajlu modula. Primjer *setup.py* fajla moguće je vidjeti u listingu 5.1.1 . *Setup.py* bi trebao da sadrži sledeće podatke:

- *name* - naziv modula
- *version* - razvojna verzija modula
- *install_requires* - moduli/programi koji su zahtjevani da prethodno budu instalirani kako bi modul neometano radio
- *entry_points* - smjernica/putanja ka interfejsu programa na koji će modul biti povezan
- *zip_safe* - sigurnosna provjera da li je sigurno modul zapakovati u *.zip* arhivu
- *package_data* - podaci i fajlovi koji bi trebali biti arhivirani zajedno sa programskim kodom

```
1. from setuptools import setup, find_packages

2. setup(
3.     name="ProjekatTim6",
4.     version="0.1",
5.     packages=find_packages(),
6.     install_requires=['Django>=1.10'],
7.     package_data={'ProjekatTim6':
   ['static/*.css','static/*.js','static/*.html','templates/*
   .html']},
8.     zip_safe=False
9. )
```

Listing 5.1.1 Primjer *setup.py* fajla platforme

5.2 Osnovni moduli

Za platformu ExpreSsiVeness su do sada kreirana četiri osnovna modula koja će biti ukratko objašnjena u narednom izlaganju.

5.2.1 *Database Schema Import* modul

Namjena ovog modula je učitavanje šeme baze podataka. Modul učitava nazive tabela i njihove atribute. Ovaj modul takođe vrši i vezivanje između uzajamno povezanih tabela te čuvanje istih u bazu podataka.

Parametri koje korisnik mora popuniti:

- ***hostname*** - adresa udaljenog servera ili *localhost* za lokalne baze podataka
- ***port*** - port na kome je pokrenuta baza podataka
- ***schema*** - naziv šeme iz koje će se vuci podaci
- ***username*** - korisničko ime za pristup bazi podataka
- ***password*** - korisnička sifra za pristup bazi podataka

Osnovni radni algoritam modula:

1. Na osnovu zadatih parametara ostvariti vezu sa bazom podataka
2. Prikupiti i sačuvati sve nazive tabela
3. Prikupiti i sačuvati sve atribute tabela
4. Na osnovu prethodno prikupljenih podataka napraviti i sačuvati veze između tabela

5.2.2 *HTML Import* modul

Namjena ovog modula je da povuče (dobavi) sav sadržaj sa odredjene *web*-stranice te prikaže korisniku sve njene *HTML* tagove, njihove medjusobne veze i bitne atribute.

Parametri koje bi korisnik trebao popuniti:

- ***site address*** - internet adresa na kojoj se *web*-stranica nalazi
- *http proxy* - *http proxy* adresa korisnika (ukoliko je potrebno)
- *https proxy* - *https proxy* adresa korisnika (ukoliko je potrebno)

Osnovni radni algoritam modula:

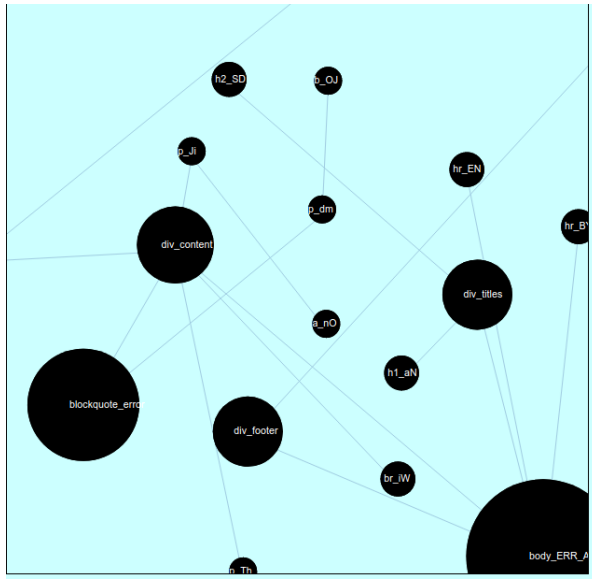
1. Na osnovu zadatih parametara dobaviti sadrzaj *web*-stranice
2. Pretragom u dubinu proći kroz sve *html* elemente (tagove) te sačuvati njihove nazive i atribute
3. Na osnovu sačuvanih elemenata, kreirati njihove međusobne veze te ih sačuvati

5.2.3 *Simple Graph* Modul

Simple graph modul je namjenjen za minimalistično prikazivanje podataka u vidu strukture tipa grafa. Čvorovi su predstavljeni crnim krugovima unutar kojih se nalazi naziv čvora, dok su veze predstavljene plavim linijama izmedju čvorova. Primjer izgleda modula moguce je vidjeti na slici 5.2.3.1 .

Funkcionalnosti koje modul nudi:

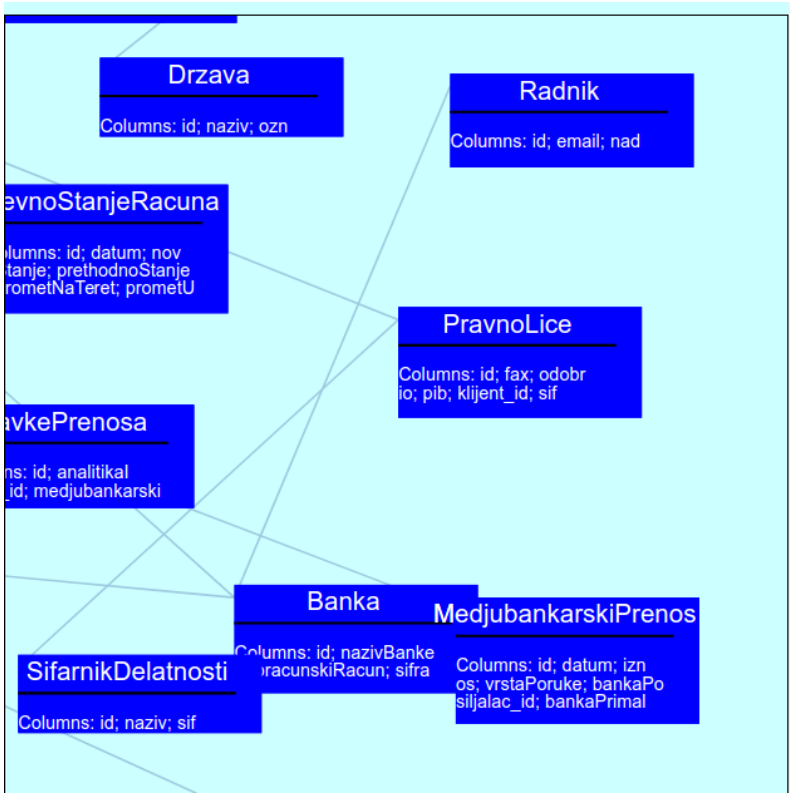
- Pomjeranje čitave strukture
- Pomjeranje pojedinacnih čvorova
- Mehanizam priblizavanja (zoom)
- Filtriranje
- Pretraživanje



Slika 5.2.3.1 Izgled *Simple Graph* modula

5.2.4 *Complex Graph* modul

Namjena *Complex Graph* modula je da korisniku ponudi detaljan i kompleksan pregled podataka prikazanih u vidu strukture grafa. Čvorovi su prikazani plavim pravougaonicima u kojima se pored naziva nalaze i svi atributi tog čvora. Veze su predstavljene plavim linijama koje spajaju čvorove. Ovaj modul nudi iste funkcionalnosti koje nudi i modul za jednostavan prikaz podataka. Primjer *Complex Graph* vizuelizacije moguće je vidjeti na slici 5.2.4.1 .



Slika 5.2.4.1 Izgled *Complex Graph* modula

5.4 *Phone Release Api* modul

Phone release api modul je namjenjen za prikupljanje podataka o mobilnim telefonima odredjenog brenda. Modul prikuplja podatke koristeći otvoreni strani interfejs *web*-stranice *fonoAPi* te od njih pravi strukturu tipa grafa na osnovu datuma njihovog pojavljivanja na tržištu, tj. datuma kada su mobilni telefoni prvi put stupili u komercijalnu fazu.

5.4.1 *FonoApi*

Web-stranica https://fonoapi.freshpixl.com/ nudi *API* (Application programming interface) za detaljne opise mobilnih telefona kao što su model, brend, datum pojavljivanja na tržištu, centralni procesor, graficki procesor, ram memorija i sl. *Api* posjeduje sopstvenu bazu podataka u kojoj su detaljno zabilježeni podaci o većini popularnih *android*, *apple*, *iOS* ali i starijih mobilnih telefona [20] . Sve što je potrebno korisniku da bi mogao da koristi fono*API* jeste da kreira svoj lični token na njihovoj *web*-stranici.

FonoApi nudi dvije metode koje korisnici mogu iskoristiti, a to su:

- ***getdevice*** - metoda koja vraća detaljan opis za jedan određen uredjaj. Pristupna adresa ove metode je: `https://fonoapi.freshpixl.com/v1/getdevice` . Parametri koje korisnik može proslijediti metodi su sledeći [20]:
    - *brand* - naziv brenda (proizvođača) mobilnih telefona
    - ***device*** - naziv traženog mobilnog telefona. (može se desiti da ima više rezultata)
    - *position* - ukoliko je prethodni vraćeni rezultat bio skup od više rezultata, moguće je proslijediti poziciju željenog telefona kako bi se dobio samo njegov detaljan opis.
    - ***token*** - korisnički *token* (ključ) za korištenje *fonoApi* usluga

- ***getlatest*** - metoda koja vraća detaljne opise za više uređaja kao rezultat. Pristupna adresa ove metode je: https://fonoapi.freshpixl.com/v1/getlatest . Parametri koje korisnik moze proslijediti metodi su sledeći [20]:

    - ***brand*** - naziv brenda (proizvođača) mobilnih telefona. Vraćeni rezultat će biti skup mobilnih telefona opadajuće poredanih po datumu pojave na tržištu.
    - *limit* - ograničenje broja vraćenih mobilnih telefona (maksimalno 100)
    - ***token*** - korisnički token (kljuc) za korištenje fonoApi usluga

Obe metode vraćaju rezultat u *JSON* formatu. Vraćeni rezultati će biti kasnije detaljnije objašnjeni i prezentovani.

5.4.2 Opis funkcija i implementacija modula

Parametri koje bi korisnik trebao popuniti kako bi koristio *Phone Release Api* modul:

- ***brand name*** - naziv brenda (proizvođača) mobilnih telefona
- *http proxy* - http proxy adresa i port korisnika (ukoliko je potrebno)
- *https proxy* - https proxy adresa i port korisnika (ukoliko je potrebno)

Prikupljanje podataka:

Kada korisnik popuni parametre neophodne za rad modula, te pokrene modul, njegovi podaci se skladište u promjenjive koje se dalje proslijeđuju funkciji za slanje *POST* zahtjeva. Prethodno zabilježeni podaci u bazi podataka se brišu kako ne bi došlo do preplitanja podataka. Pythonova metoda *post()* iz *requests* biblioteke je zadužena za slanje *POST* zahtjeva ka ***getlatest*** metodi *API*-ja. Nakon uspješnog *POST* zahtjeva, *API* modulu vraća odgovor u *JSON* formatu koji se takođe smješta u promjenljivu za dalju obradu. Naredni listing (5.4.2.1) prikazuje implementaciju ove operacije.

```
1. Element.objects.all().delete()

2. Link.objects.all().delete()
3. naziv_brenda = args[0]['p0']
4. url = "https://fonoapi.freshpixl.com/v1/getlatest"
5. post_data = {"brand": naziv_brenda, "limit":70, "token":
   "e2bd47e976e0118124f6254702efee5c62aca08cb9707734"}
6. httpprox = args[0]['p1']
7. httpsprox = args[0]['p2']
8. proxyDict = {"http":httpprox, "https":httpsprox}
9. request = requests.post(url, json=post_data, proxies =
   proxyDict)
```

Listing 5.4.2.1 Slanje *POST* zahtjeva *fonoApi*-ju

Kao što je već napomenuto, vraćeni rezultat metode *getlatest* je u ***JSON*** formatu. Primjer jednog takvog neobrađenog rezultata dat je u sledećem listingu (5.4.2.2) .


```
{
        "DeviceName": "Xiaomi Redmi 5A",
        "Brand": "Xiaomi",
        "technology": "GSM / CDMA / HSPA / LTE",
        "gprs": "Yes",
        "edge": "Yes",
        "announced": "2017, November",
        "status": "Coming soon. Exp. release 2017, December
7th",
        "dimensions": "140.4 x 70.1 x 8.4 mm (5.53 x 2.76 x 0.33
in)",
        "weight": "137 g (4.83 oz)",
        "sim": "Dual SIM (Nano-SIM, dual stand-by)",
        "type": "IPS LCD capacitive touchscreen, 16M colors",
        "size": "5.0 inches, 68.0 cm2 (~69.1% screen-to-body
ratio)",
        "resolution": "720 x 1280 pixels, 16:9 ratio (~296 ppi
density)",
        "display_c": "- MIUI 9.0",
        "card_slot": "microSD, up to 256 GB (dedicated slot)",
        "alert_types": "Vibration; MP3, WAV ringtones",
        "loudspeaker_": "Yes",
        "sound_c": "- Active noise cancellation with dedicated
mic",
        "wlan": "Wi-Fi 802.11 b/g/n, Wi-Fi Direct, hotspot",
        "bluetooth": "4.1, A2DP, LE",
        "gps": "Yes, with A-GPS, GLONASS, BDS",
```

```
        "infrared_port": "Yes",
        "radio": "FM radio",
        "usb": "microUSB 2.0",
        "messaging": "SMS(threaded view), MMS, Email, Push Mail,
IM",
        "browser": "HTML5",
        "java": "No",
        "features_c": "- Xvid/MP4/H.265 player\r\n  -
MP3/WAV/eAAC+/FLAC player\r\n  - Photo/video editor\r\n  -
Document viewer",
        "battery_c": "Non-removable Li-Ion 3000 mAh battery",
        "colors": "Gold, Dark Gray, Rose Gold, Blue",
        "sensors": "Accelerometer, proximity",
        "cpu": "Quad-core 1.4 GHz Cortex-A53",
        "internal": "32 GB, 3 GB RAM or 16 GB, 2 GB RAM",
        "os": "Android 7.1.2 (Nougat)",
        "primary_": "13 MP, f/2.2, phase detection autofocus,
LED flash",
        "video": "1080p@30fps",
        "secondary": "5 MP, f/2.0",
        "speed": "HSPA, LTE",
        "network_c": "CDMA 800 & TD-SCDMA",
        "chipset": "Qualcomm MSM8917 Snapdragon 425",
        "features": "Geo-tagging, touch focus, face/smile
detection, HDR, panorama",
        "gpu": "Adreno 308",
        "multitouch": "Yes",
        "build": "Front glass, aluminum body",
        "price": "About 110 EUR",
        "_2g_bands": "GSM 850 / 900 / 1800 / 1900 - SIM 1 & SIM
2",
        "_3_5mm_jack_": "Yes",
        "_3g_bands": "HSDPA 850 / 900 / 1900 / 2100 ",
        "_4g_bands": "LTE band 1(2100), 3(1800), 5(850),
7(2600), 8(900), 34(2000), 38(2600), 39(1900), 40(2300),
41(2500)"
    }
```


Listing 5.4.2.2 Primjer *JSON* rezultata dobijenog sa *fonoApi*-ja

Nakon što je modulu vraćen odgvor, provjerava se da li je odgovor prazan (u slučaju da je izabran nepostojeći brend telefona ili je došlo do neke greške). Ukoliko je odgovor zaista prazan, modul kreira adekvatan odgovor, tj. element koji će naznačiti da za izabrani brend *fonoApi* nema podataka. Ako je odgovor *API*-ja bio neprazan, tj. validan, tada se prvo radi prevođenje iz *JSON* formata na strukturu koja je mnogo pogodnija *Python* programskom jeziku. Kada je prevođenje podataka završeno, kreira se osnovni (rootni) čvor u koji se smješta naziv brenda. Nakon ovog koraka, modul iterira kroz sve podatke te bilježi godine i mjesece pojavljivanja mobilnih telefona na tržištu, vezuje mjesece za godine, kreira elemente od podataka vezanih za mobilne telefone, te ih vezuje za mjesece. Naredni listing (5.4.2.3) pokazuje implementaciju ovog dijela posla.

```
1. if request.text == "[[]]":

2.      el1 =
   Element(id="_"+naziv_brenda+"_",attributes="___")
3.      el2 = Element(id="NO_DATA", attributes = "no data")
4.      el1.save()
5.      el2.save()
6.      link = Link()
7.      link.elementParent =
   Element.objects.get(id="_"+naziv_brenda+"_")
8.      link.elementChild = Element.objects.get(id="NO_DATA")
9.      link.save()
10.       else:
11.           rec_data = []
12.           rec_data = jsx.loads(request.text)
13.           for i in range(0, len(rec_data)):
14.               print(rec_data[i]['DeviceName'], "->",
   rec_data[i]['status'])
15.           elemBrand =
   Element(id="_"+naziv_brenda,attributes="")
16.           elemBrand.save()
17.           godine = dict()
18.           for i in range(0, len(rec_data)-1):
19.               mjesto = rec_data[i]['status'].find("20")
20.               if mjesto != -1:
21.                   godina = rec_data[i]['status']
   [mjesto:mjesto + 4]
22.                   elemGod = None
23.                   try:
24.                       elemGod =
   Element.objects.get(id="_"+godina)
25.                   except Element.DoesNotExist:
26.                       elemGod=None
```

```
27.                     if elemGod==None:
28.                         elemGod =
    Element(id="_"+godina,attributes="")
29.                         elemGod.save()
30.                         linkG = Link()
31.                         linkG.elementParent = elemBrand
32.                         linkG.elementChild = elemGod
33.                         linkG.save()
34.                     mjesec = rec_data[i]['status'][mjesto +
    6:]
35.                     mjx = replacement(mjesec)
36.                     elemMjesec = None
37.                     try:
38.                         elemMjesec =
    Element.objects.get(id=mjx+"_"+elemGod.id)
39.                     except Element.DoesNotExist:
40.                         elemMjesec = None
41.                     if elemMjesec == None:
42.                         elemMjesec =
    Element(id=mjx+"_"+elemGod.id)
43.                         elemMjesec.save()
44.                         link = Link()
45.                         link.elementParent = elemGod
46.                         link.elementChild = elemMjesec
47.                         link.save()
48.                     nazivUredjajaw = rec_data[i]
    ['DeviceName']
49.                     nazivUredjaja =
    replacement(nazivUredjajaw)
50.                     nazivUredjaja = "_"+nazivUredjaja
51.                     wantedData =
    ['sensors','cpu','internal','os','primary_']
52.                     atts=""
53.                     for key in rec_data[i]:
54.                         if key in wantedData :
55.                             ak1 = replacement(rec_data[i]
    [key])
56.                             atts=atts+key+": "+ak1+" | "
57.                     elemDevice = Element(id=nazivUredjaja,
    attributes=atts)
58.                     elemDevice.save()
59.                     link2 = Link()
60.                     link2.elementParent=elemMjesec
61.                     link2.elementChild=elemDevice
62.                     link2.save()
```

Listing 5.4.2.3 Obrada podataka dobijenih sa *fonoApi*-ja

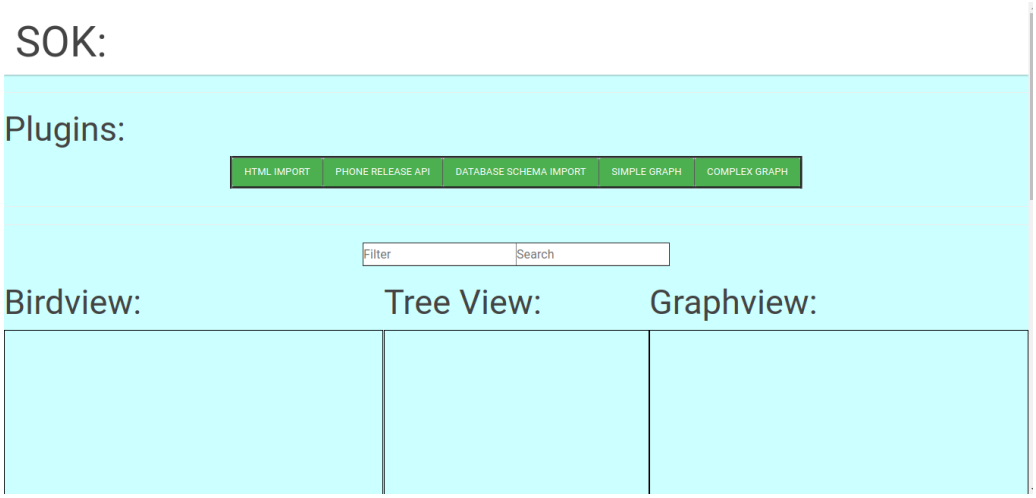
Formiranje čvorova (elemenata) **i veza** (linkova) se vrši na sledeći način:

- modul iterira kroz sve prikupljene podatke
- modul pokušava da nađe gdje je zapisana godina izlaska mobilnog telefona na tržište, ukoliko je pronađe, te ukoliko već nije sačuvana, čuva godinu u bazi podataka i kreira vezu između godine i brenda
- modul pokušava da nađe gdje je zapisan mjesec izlaska mobilnog telefona na tržiste; ukoliko ga pronađe, te ukoliko već nije sačuvan, čuva mjesec u bazi podataka i kreira vezu između mjeseca i godine
- modul pronalazi naziv mobilnog telefona i njegove atribute, kreira čvor (element) i vezuje ga za mjesec njegovog pojavljivanja na tržištu

6 Primjer korištenja aplikacije

U ovom odjeljku će biti predstavljen primjer korištenja aplikacije. Korisnik će uz pomoć ***phone release api*** modula učitati podatke i na kraju ih prikazati preko *complex graph* modula.

Kada se aplikacija učita u internet pregledniku, korisnisniku se prikazuje osnovna stranica koja je prikazana na slici 6.1 .

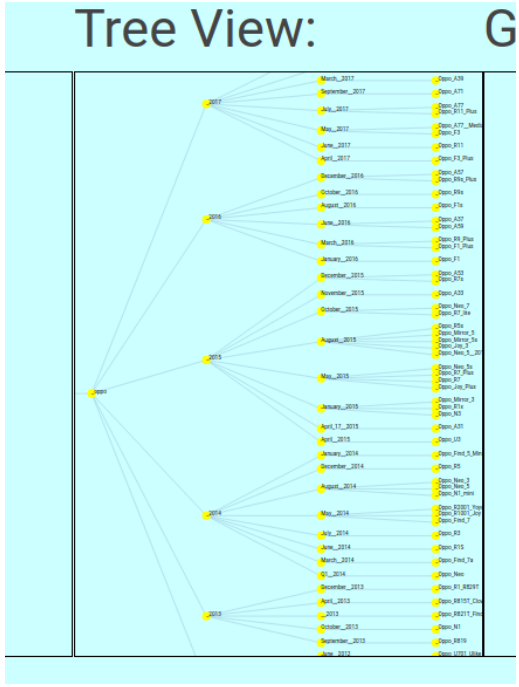


Slika 6.1 Osnovna stranica aplikacije

Zelenim dugmićima su predstavljeni svi učitani moduli platforme. Korisnik bira jedan od modula za učitavanje podataka. U ovom slučaju, to bi bio *Phone release api* modul. Nakon izbora modula, od korisnika se traži da unese potrebne parametre koje modul zahtjeva kako bi funkcionisao, što je demonstrirano na narednoj slici (6.2) .

Slika 6.2 Unos potrebnih parametara za korištenje *Phone release api* modula

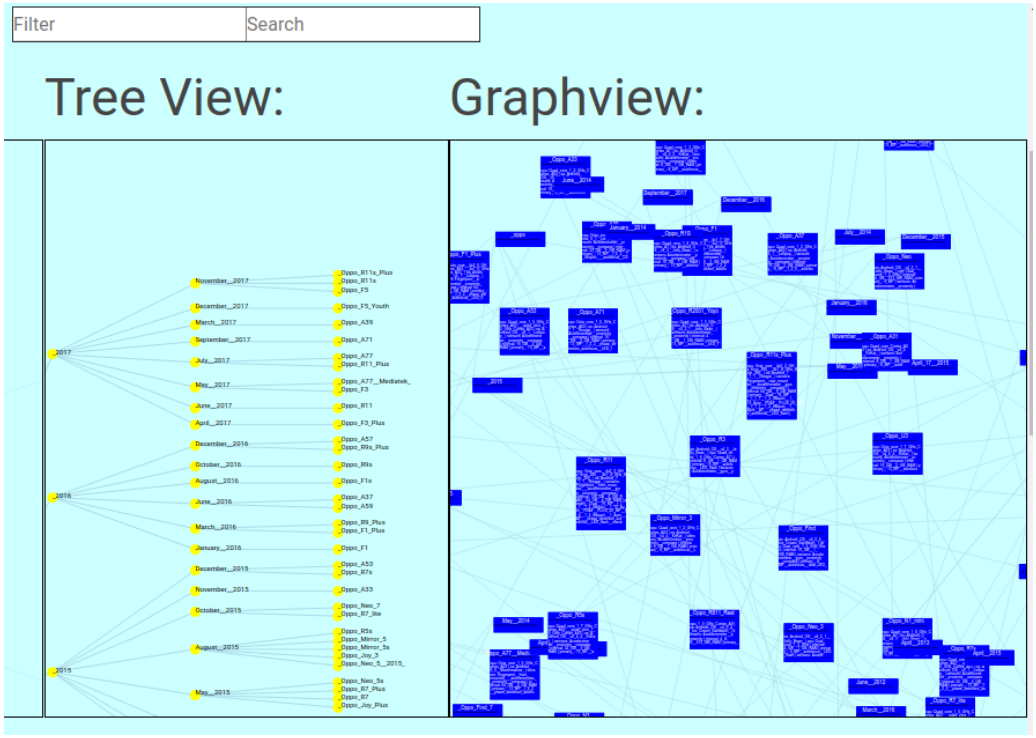
Kada korisnik unese sve potrebne parametre i klikne na dugme *submit*, modul kreće da se izvršava. Nakon što je preuzimanje podataka i njihova obrada završena, korisniku se prikazuju podaci u vidu grafa tipa stabla na platnu nazvanom *Tree View*, što je i prikazano na sledećoj slici (6.3).



Slika 6.3 *Tree View* prikaz

Tree View prikaz je moguće zumirati (približiti) korisniku i pomjerati. Takođe je moguće postaviti kursor miša iznad nekog od čvorova i dobiti detaljne informacije o čvoru.

Ukoliko korisnik želi, on može odabrati neki od modula za prikaz i tako na neki drugi način vidjeti podatke. Na narednoj slici (6.4) prikazan je primjer korisnika koji je odabrao modul *Simple Graph,* kako bi prikazao podatke u vidu grafa.



Slika 6.4 Detaljan graf prikaz podataka koristeći modul *Simple Graph*

Korišćenjem ovog modula korisnik ima mogućnost da pomjera pojedinačne čvorove ili čitavu grupu čvorova, približi ili udalji prikaz, filtrira i/ili pretraži podatke.

7 Zaključak

U radu je opisan način funkcionisanja i implementacija platforme ***ExpreSsiVeness*** koja nudi mogućnost uvezivanja modula za učitavanje i prikazivanje podataka. Pored platforme, predstavljeni su i njeni do sada implementirani moduli sa naglaskom na modul za preuzimanje podataka sa ***fonoApi*** *web*-stranice. Čitav projekat je realizovan poštujući principe komponentnog razvoja svoftvera što nudi veliku mogućnost proširivosti. Labava, ali jasno specificirana sprega između komponenti omogućava da se komponente razvijaju samostalno ali isto tako doprinosi smanjenju broja potencijalnih grešaka i nekompatibilnosti između komponenti.

Platforma je implementirana korištenjem više tehnologija kao što su: *Python*, *Django*, *HTML*, *CSS*, *JavaScript* i *SetupTools*. Moduli su takođe implementirani programskim jezikom *Python*, uz primjese: *HTML*-a, *CSS*-a i *JavaScript*-a.

Ostavljeno je veoma mnogo prostora za dalji razvoj na ovom projektu. Sama platforma bi se mogla proširiti da pruza još neke određene usluge, tj. da posjeduje još dodatnih servisa. Neki od potencijalnih servisa ili mogućnosti koji bi se mogli dodati u platformu su: Mogućnost konzolnog rada sa aplikacijom, servis za komunikaciju sa *benchmark* programima, servis za uvezivanje sa tekstualnim editorom, mogućnost izbora drugih struktura podataka za čuvanje, servis za podršku *VR* (v*irtual reality*) ili *AR* (*augmented reality*) vizualizacije sadržaja, i mnogi drugi. Kada su moduli koji se vezuju na platformu u pitanju, mogućnosti su velike. Potencijal *d3.js* biblioteke bi se mogao mnogo više iskoristiti za prikazivanje podataka ne samo u vidu grafa nego i mnogih drugih struktura. Za ovakve promjene, blaga rekonstrukcija i ekspanzija platforme bi mogla biti poželjna.

Literatura

1. https://en.wikipedia.org/wiki/Graph_(discrete_mathematics) (datum pristupa: 17.12.2017.)
2. https://upload.wikimedia.org/wikipedia/commons/thumb/5/5b/6n-graf.svg/333px-6n-graf.svg.png (datum pristupa: 17.12.2017.)
3. https://en.wikipedia.org/wiki/Tree_(graph_theory) (datum pristupa: 17.12.2017.)
4. https://jeremykun.files.wordpress.com/2012/09/different_trees.png (datum pristupa 5.1.2018.)
5. https://en.wikipedia.org/wiki/Tree_(data_structure) (datum pristupa 17.12.2017.)
6. https://koenig-media.raywenderlich.com/uploads/2016/06/Tree-2-650x300.png (datum pristupa 17.12.2017.)
7. http://www.igordejanovic.net/courses/sok/razvoj-baziran-na-komponentama.html
8. https://docs.python.org/3/faq/general.html
9. https://en.wikipedia.org/wiki/Django_(web_framework) (datum pristupa 18.12.2017.)
10. https://en.wikipedia.org/wiki/JavaScript (datum pristupa 18.12.2017.)
11. https://d3js.org/#introduction 1X (datum pristupa 18.12.2017.)
12. http://www.tutorialsteacher.com/d3js/what-is-d3js (datum pristupa 18.12.2017.)
13. http://www.igordejanovic.net/courses/tech/setuptools.html
14. https://en.wikipedia.org/wiki/Setuptools (datum pristupa 19.12.2017.)
15. http://setuptools.readthedocs.io/en/latest/setuptools.html (datum pristupa 21.12.2017.)
16. https://en.wikipedia.org/wiki/Pip_(package_manager) (datum pristupa 20.12.2017.)
17. https://docs.djangoproject.com/en/2.0/topics/http/views/ (datum pristupa: 23.12.2017.)
18. https://d3js.org/ (Datum pristupa 5.1.2018.)
19. https://en.wikipedia.org/wiki/Plug-in_(computing) (Datum pristupa 4.1.2018.)
20. https://github.com/shakee93/fonoapi (Datum pristupa 4.1.2018.)